DIGITAL
multimedia

Nigel Chapman and
Jenny Chapman

WILEY

DIGITAL
multimedia

WORLDWIDE
SERIES IN
COMPUTER
SCIENCE

Series Editors **Professor David Barron, Southampton University, UK**
Professor Peter Wegner, Brown University, USA

The Worldwide series in Computer Science has been created to publish textbooks which both address and anticipate the needs of an ever evolving curriculum thereby shaping its future. It is designed for undergraduates majoring in Computer Science and practitioners who need to reskill. Its philosophy derives from the conviction that the discipline of computing needs to produce technically skilled engineers who will inevitably face, and possibly invent, radically new technologies throughout their future careers. New media will be used innovatively to support high quality texts written by leaders in the field.

Books in Series
Ammeraal, *Computer Graphics for Java Programmers*
Ammeraal, *C++ for Programmers 3rd Edition*
Barron, *The World of Scripting Languages*
Ben-Ari, *Ada for Software Engineers*
Gollmann, *Computer Security*
Goodrich & Tamassia, *Data Structures and Algorithms in Java*
Kotonya & Sommerville, *Requirements Engineering: Processes and Techniques*
Lowe & Hall, *Hypermedia & the Web: An Engineering Approach*
Magee & Kramer, *Concurrency: State Models and Java Programs*
Peters, *Software Engineering: An Engineering Approach*
Preiss, *Data Structures and Algorithms with Object-Oriented Design Patterns in C++*
Preiss, *Data Structures and Algorithms with Object-Oriented Design Patterns in Java*
Reiss, *A Practical Introduction to Software Design with C++*
Schneider, *Concurrent and Real-time Systems*
Winder & Roberts, *Developing Java Software 2nd Edition*

DIGITAL multimedia

Nigel Chapman and
Jenny Chapman

JOHN WILEY & SONS, LTD
Chichester • New York • Weinheim • Brisbane • Singapore • Toronto

Baffins Lane, Chichester,
West Sussex PO19 1UD, England
National 01243 779777
International (+44) 1243 779777
E-mail (for orders and customer service enquiries):
cs-books@wiley.co.uk
Visit our Home Page on http://www.wiley.co.uk
or http://www.wiley.com

Reprinted November 2000, July 2001, January 2002

Cover illustration: Pablo Picasso's "La Guitare" 1913 © Succession Picasso/DACS 2001. Reproduced with permission from The Museum of Modern Art, New York.

PICASSO, Pablo
Guitar. Céret (after March 31, 1913)
Pasted paper, charcoal, ink, and chalk on blue paper, mounted on ragboard, 26½ x 19½" (66.4 x 49.6 cm).
The Museum of Modern Art, New York, Nelson A. Rockefeller Bequest.
Photograph © 1999 The Museum of Modern Art, New York.

Neither the author nor John Wiley & Sons, Ltd accept any responsibility or liability for loss or damage occasioned to any person or property through using the material, instructions, methods or ideas contained herein, or acting or refraining from acting as a result of such use. The author and publisher expressly disclaim all implied warranties, including merchantability or fitness for any particular purpose.

Designations used by companies to distinguish their products are often claimed as trademarks. In all instances where John Wiley & Sons, Ltd is aware of a claim, the product names appear in initial capital or all capital letters. Readers, however, should contact the appropriate companies for more complete information regarding trademarks and registration.

John Wiley and Sons Ltd., Baffins Lane, Chichester, West Sussex,
UK PO19 1UD.

Other Wiley Editorial Offices
John Wiley & Sons, Inc., 605 Third Avenue,
New York, NY 10158-0012, USA
Weinheim • Brisbane • Singapore • Toronto

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

ISBN 0-471-98386-1

Produced from authors' own PostScript files
Printed and bound in Great Britain by Biddles Ltd, Guildford and King's Lynn.
This book is printed on acid-free paper responsibly manufactured from sustainable forestry, in which at least two trees are planted for each one used for paper production.

Contents

This page intentionally left blank

Preface

This book is a text for introductory courses on digital multimedia. There is at present a diversity of approaches to such courses, ranging from the purely technical, with an emphasis on underlying system technologies or multimedia programming, to those aimed primarily at multimedia content production. Increasingly, courses with a strong practical orientation are being offered within computing and IT departments, to students coming from very mixed backgrounds who are aiming at a career in multimedia. The book will be of use to students on all of these types of course. For multimedia producers, we describe the way media are represented and processed digitally and the consequences that follow. For software engineers, we describe what multimedia producers do with the data and why they want to do it. Ultimately, these are two sides of the same coin, and an understanding of both sides will be of value to anyone seeking to work professionally in multimedia.

In order to focus tightly on these concerns, and to keep the size of the book within reasonable bounds, we have only dealt lightly with some topics. We have not described multimedia design methodologies or multimedia production management: one of the things we do when we're not writing books is explore — in our own work and with students — the potential of digital multimedia for new forms of expression that are enabled by digital computers and networks. It is clear that this potential has barely begun to be properly understood or exploited. To claim to offer design methods for multimedia at such an early stage in its evolution would be arrogant. Instead, in our final chapter, we invite readers to explore the problems and possibilities through a wide variety of multimedia projects, which are intended to provide some insight into the forms and processes of multimedia through practical experience.

Concerns about human computer interaction and ethical issues can be found throughout the book, but we have not presented a formal or thorough treatment of either; nor, since this is an introductory text, have we gone into the utmost detail about system issues, although our description of technology is rigorous and accurate. All of these topics can be pursued in depth in more specialized books, to which we include references.

In order to fully appreciate the problems and potential of multimedia, it is essential to have a good understanding of the characteristics of the media that are being combined. Armed with such an understanding, much of what happens when media are brought together becomes clear. In our descriptions of graphics, text and time-based media, we do not lose sight of our ultimate subject of multimedia, and have emphasized the relevant aspects, while eliding some topics that would normally be central to the study of these individual media as subjects in their own right.

To write about multimedia production without making reference to the specific software tools that are used in the industry would be artificial and lend an air of unreality to our descriptions. Accordingly, we describe particular applications when we need to explain how images, video, and so on, are produced and manipulated. We have chosen to use only packages that are widely used in the real world, and that students might be expected to encounter after graduation, if not before. Access to the programs described in the text, or their equivalent, will be a considerable advantage to students. A complementary volume, entitled *Digital Media Tools*, which deals with a selection of the relevant programs in more detail, is in preparation as this book is going to press.

Our treatment of multimedia is somewhat biased towards the World Wide Web. We have featured the Web, rather than more advanced forms of distributed multimedia, because students are likely to have ready access to the Web itself and to tools for producing Web content; because the emphasis on textual representations and markup languages makes it easy to see and explain what is going on; and because of the ready availability of standards and other documentation. Additionally, although we are reluctant to make predictions about the future, it is clear that the Internet is going to become a major distribution medium for multimedia content, as shown, for example, by the speed with which leading software manufacturers are adding Web features to their page layout, graphics and multimedia applications, and developing new tools specifically for Web page development. Many graduates of multimedia courses will be hoping to pursue careers in this area.

Towards the end of the book, readers will find an afterword by Professor Brent MacGregor, head of the School of Visual Communication at Edinburgh College of Art. This short essay considers what may, or may not, be the future of multimedia, and following on from the final chapter, effectively throws down the gauntlet to all those who will be working in multimedia in the future.

We make some assumptions about our readers' prior knowledge. They should have experience of using a modern graphical user interface to an operating system such as the MacOS or Windows, and should have used the World Wide Web and seen other forms of multimedia. We do not assume any mathematical or computer science knowledge, except in specifically flagged passages (see below), but some programming experience will be valuable in understanding Chapter 14.

Presentation

Some aspects of the presentation require comment.

⇨ Some passages are typeset like this one, in a smaller font, preceded by a ⇨ symbol. These passages are 'asides' — they contain material that is somewhat tangential to the main presentation and may be omitted without great loss.

⊃ Other passages are typeset like asides but in preceded by this ⊃ symbol. These contain material at a higher technical level than the surrounding passages. They might be found difficult by readers with little technical and mathematical background, but should be illuminating to those with such knowledge.

The Web Site

Teaching about multimedia by writing is rather like teaching swimming on dry land. An extensive Web site, with examples of those media that cannot be appreciated by reading alone, expands the limited illustration of material which is possible in a printed book. Additional supporting material, including links to other Web sites demonstrating multimedia, are available from the site, together with product reviews, and hints on solutions for exercises. The URL for the site is http://www.wiley.com/digital_multimedia.

Acknowledgements

We are grateful to Adobe Systems, Inc., Electrifier, Inc., Macromedia, Inc., MetaCreations Corp. and Terran Interactive, Inc. for their generous support of this project. We would also like to thank David Barron, University of Southampton; George Beekman, Oregon State University; Rhonda Berger, Miami Dade Community College; Andrew Calway, University of Bristol; Russell Campion, University of Staffordshire; Maxine Cohen, Nova Southeastern University; Greg Cranitch, Griffith University; Stephan Fischer, Darmstadt University of Technology; Rachelle Heller, George Washington University; David Hutchison, University of Lancaster; Peter Kenny, University of Kent and Jimmy L Reeves, Texas Tech University, all of whom read and commented on a draft of the book. We are also grateful to John Taylor for help with Bryce.

We have received wonderful support — better than we deserved — from our publisher John Wiley & Sons, and we would particularly like to express our thanks to Dawn Booth, for supervising the production, Katrina Arens, for expediting things and making a great many telephone calls, and above all, to Gaynor Redvers-Mutton, for her encouragement, enthusiasm, patience, and for one or two small miracles.

Introduction 1

It was a dark and stormy night.

A storyteller might use these portentous words to capture your attention and draw you into a tale. A novelist might use the same words to begin the first page of a book. A radio play based on the same story could open with a crash of thunder and the howling of the wind. A film of the book could open on a scene of darkness, suddenly illuminated by a flash of lightning that reveals trees bent over in the gale.

The dark and stormy night can be represented in different *media*, each of which tells the story through different means, appealing to different senses. One of the key insights in computing is that all these media can be represented *digitally*, as a structured collection of bits, and can then be manipulated by programs on a computer, stored on disks and other storage devices, and transmitted over networks. Their shared digital representation means that different media can be combined into what is loosely called *multimedia*.

The combination of different media is not something new. On the contrary, established forms mix media routinely. A TV news bulletin, for example, might include sound and moving pictures — both live and recorded — still images, such as photographs of politicians, graphic illustrations, such as a histogram showing the trend in unemployment, and text, in the form of subtitles, captions, quotations, and credits, which will usually be accompanied by suitable annunciatory music. A contemporary live theatrical performance may incorporate two- or three-dimensional artwork and design (sets, backdrops, costumes), sound, which may be a mixture of speech and live music, perhaps with tape recordings and synthesized sounds, projected still and moving images, and sometimes text in the form of surtitles (written text displayed above the stage, which are often used for translation, especially in opera).

The integration of media is natural — we perceive the world through all the senses we have at once; the separation of media is artificial, and may often seem unsatisfactory.

Consider film, for example. From the earliest days of cinema, in the absence of any method of recording and playing sound in synchronization with picture, separate live sounds were produced at the same time as the film was shown: not just the live piano accompaniment that you may find in contemporary art-house cinema showings of silent classics, but dialogue, singing, and sound effects, produced by actors, singers, and noisemakers in the theatre, often hidden behind the screen. So great was the film-going public's desire to hear sound naturally integrated with moving pictures that in 1927 when Al Jolson spoke his phrase, 'Wait a minute, wait a minute. You ain't heard nothing yet,' in *The Jazz Singer*, audiences stood up and cheered. It is hardly surprising, then, that once the technical obstacles to representing media digitally (which we will consider in Chapter 2) were overcome, the next step should be to combine separate media. In fact, the separation of digital media into distinct data types — different 'sorts of things' to which different operations can be applied — can be seen as a temporary exigency demanded by an immature technology.

Applications of Multimedia

What is it, then, if not the combination of media, that distinguishes digital multimedia from previous forms of combined media? It is the fact that the bits that represent text, sound, pictures, and so on can be treated as data by computer programs. The full potential of this fact has not yet been fully explored, but one facet of it immediately distinguishes digital multimedia from its predecessors. A program can control the order in which various components are presented and combined, and can do so in response to input from a computer user. In other words, digital multimedia can be *interactive*, in a way that, for example, a TV news bulletin is not, and that goes far beyond the simple control afforded by a VCR.

The digital multimedia version of the story we began this book with could open with a picture depicting a dark and stormy night. By clicking on an icon on the screen, the user could cause the scene to play as a video clip, or add sound effects, according to their choice. Perhaps by clicking on different areas of the scene, details could be revealed, or a new scene could be introduced — a different one for each part of the first scene, allowing the tangled tale of betrayal and redemption that began on that dark stormy night to unfold in more than one way, told through the eyes of different characters. Users whose hearing was poor could choose to have a transcription of any dialogue displayed for them, while users with impaired vision could have a description of what was happening 'read' to them by their computer. Different interface options might be offered for users with different tastes and needs. If the story had attracted critical attention, it might be possible to bring up learned commentaries on particular aspects, or a history of its telling in different media. There might be an opportunity for the user to take an active rôle, change the story, add new elements or recombine the existing ones.

Suppose, though, that this is not just a story — that some crime was committed on a real dark and stormy night, and that somebody had subsequently been arrested and charged with an offence. By collating evidence from witnesses, and facts about the location, the events of the night could be reconstructed and presented, like our story, as a mixture of animation, graphics, sound and video, with textual annotations. Members of the jury could then examine the event from different angles, verify that it was possible to see something at a certain distance, perhaps factor out the parts of the reconstruction based on a certain witness's evidence, or try out various hypotheses suggested by the lawyers in the case. *Forensic multimedia* of this sort is already being used in courts in the United States.

Digital multimedia may be used for many things besides stories and forensic reconstructions. The most prominent and economically significant uses of multimedia at present are in the fields of entertainment and education. At one extreme, the combination of sound, animation, video and text with interactivity is commonly found both in adventure games, such as *Starship Titanic* and action games, such as *Doom* and its bloody offspring. At the opposite extreme are the multimedia reference works, such as the *Cinemania* cinema encyclopedia, or the digital versions of the *Encyclopedia Britannica.*

Multimedia is also being used educationally in schools, and for extra-curricular and pre-school learning, where multimedia teaching materials can be used to present information in more direct ways than traditional books and lessons. For example, film clips and original recordings of radio broadcasts can be integrated with a textual account of events leading up to the Second World War in a modern history lesson. Virtual science experiments can be constructed using multimedia techniques, to allow pupils to work with simulations of equipment that is not available to them. In both these cases, interactive features allow pupils to work at their own pace, or to pursue specialized aspects of a project according to their particular interests. Multimedia teaching materials can be bought in as complete CD-ROM titles, or they can be made by teachers in their own schools. Children themselves can also make multimedia, in the form of project reports, or as an end in itself, as an electronic class book or newsletter.

The distinction between education and entertainment is not a clear one; several succesful multimedia CD-ROMs use elements of games for an educational purpose. For example, *The Logical Journey of the Zoombinis* develops children's problem-solving skills through a series of puzzles, organized in the form of a game. This approach to learning seems to be popular for very young children, with a range of titles available for teaching literacy and basic mathematics. These are aimed both at schools and at parents, for use in the home. (As a side-effect, the use of such materials also teaches computer skills from an early age.)

In contrast to the entertainment and education sectors, where most multimedia is professionally produced and marketed, much of the material used in business is produced for a specific company, often in-house. In a continuation of the use of multimedia in formal education, multimedia materials are used for training. As with the virtual science experiments in schools, multimedia can be used to produce interactive simulations of equipment and procedures, in cases where it would be expensive or hazardous to use the real thing for training. Multimedia is also used in sales presentations and promotions — for example, computers on sale in showrooms often run multimedia presentations describing their capabilities, and Web pages often function as sales brochures.

Some professions and business areas lend themselves especially well to multimedia. Graphic designers produce interactive presentations of their work as a supplement to, or even as a substitute for, a conventional portfolio. The electronic portfolio can be inexpensively duplicated on CD-ROM and sent to potential clients, or posted on a Web site as a virtual exhibition. In the shrink-wrapped software business, paper manuals have been gradually giving way to electronic documentation for some years. The move from hard copy to digital formats has provided an opportunity to augment the text and diagrams of conventional manuals with animations, video clips, and sound. Many programs now come with a multimedia product tour, demonstrating their best features, and interactive tutorials, which guide novice users through simple tasks.

One area in which computation plays a more active part is *visualization*, in which graphics and animation are used as a means of presenting complex data. Banal examples of visualizations include pie-charts and bar-charts being generated from spreadsheet data: their graphical form summarizes the data in a way that makes recognition of trends and comparisons between magnitudes possible at a glance. More complex data than the sort normally kept in spreadsheets demands more complex visualizations to provide the same degree of comprehensibility. Three-dimensional and time-varying presentations, making use of false colour to carry extra information, are often favoured. Such elaborate visualizations are frequently generated from simulations of complex dynamic systems: for example, a program that simulates atmospheric dynamics may generate a time-lapse movie of the gestation of a tropical storm; a program simulating an organic chemical reaction may generate a three-dimensional model of the structure of the molecules involved, showing the changes they undergo. As these examples show, visualization is widely used by scientists; it is also used in business, where the systems being tracked or simulated are financial, and the output is correspondingly more abstract.

The established entertainment industry, while still delivering an essentially linear and non-interactive product, now makes extensive use of digital multimedia technology in the production of everything from blockbuster movies to gardening programmes on TV. The use of connections with digitally delivered material, such as links between programming and supporting Web sites, or accompanying CD-ROMs for mainstream movies, is rapidly expanding. At the same time, a rapidly increasing number of festivals, conferences and other international events are providing showcases for "new media" at all levels from the artistic avant garde to the purely commercial.

Delivery of Multimedia

The applications we have mentioned so far are all basically concerned with presenting material, using a combination of media, in an interactive way. Some means of delivery is required, to get this material from its producer to its consumers. It is useful to distinguish between *online* and *offline* delivery.

Online delivery uses a network to send information from one computer, often a server machine providing centralized storage of bulky data, to another, usually a personal computer on somebody's desk. The network in question may be a local area network serving a single organization, but more often it will be the Internet. In particular, the World Wide Web has established itself as a popular means of delivering multimedia online.

> ⇨ The World Wide Web has achieved such supremacy as a means of accessing data over the Internet that in popular perception the Web and the net are sometimes seen as synonymous. This is not the case, as we will explain in Chapter 2, and in more detail in Chapter 15.

Where multimedia is delivered offline, some removable storage medium must be used. The widespread deployment of CD-ROM drives in personal computers in the mid-1990s was, to some extent, responsible for the surge in interest in multimedia at that time. This is because, for the first time, it made available a form of storage with adequate capacity (around 650 Mbytes, compared to the 1.44 Mbytes of floppy disks) to accommodate the large files characteristic of multimedia data, while still being relatively inexpensive to duplicate. The data transfer rate of early CD-ROM drives was too slow to support playback of video and sound, but more recent models have increased the speed, to the point where multimedia can be played directly from a CD-ROM, although smooth playback of full-screen video from CD-ROM is still impossible without additional hardware support.

While online delivery is widely seen as the future for multimedia, the low bandwidth available for most domestic connections to the Internet means that, especially for large and elaborate work, offline delivery on CD-ROM remains popular. There is an increasing trend towards a combination of offline and online delivery, with CD-ROMs being used to hold a fixed collection of data, augmented with links to Web sites, where updates and supporting information are available (sometimes for a supplementary fee).

In 1995, an industry consortium announced the specification for a successor to CD-ROM, using the same size and shape platters, called DVD. Originally, this stood for Digital Video Disk, since the format was intended as a replacement for VHS cassettes as a distribution medium for video, as analogue methods of storing video began to be replaced by digital methods. It was immediately understood that DVD could be used equally well for any digital data, just as the audio Compact Disc had been adapted as the CD-ROM. Accordingly, the name was changed, somewhat desperately, to Digital Versatile Disk, thus keeping the abbreviation DVD. DVD offers a much higher storage capacity (up to 17 Gbytes on a double-sided disk), with similar transfer rates to modern (12× and higher) CD-ROM drives. Despite considerable media enthusiasm for DVD, it has not been taken up as rapidly as originally predicted, but during 1999, an increasing number of personal computers were being shipped with DVD drives fitted. Since these drives can also read CD-ROMs, it seems likely that they will become standard peripherals, and that DVD will be used to deliver multimedia, especially high quality video.

Online delivery, however, offers possibilities which are not available offline. In particular, it enables the delivery of (almost) live multimedia content, which in turn makes possible novel applications such as video conferencing and broadcast multimedia. Generally, when multimedia is delivered online, the delivery need not be passive, in the way that it is from CD-ROM: things can happen at the delivering end; for example, a database query can be executed.

Historical Context

Compared with established forms such as film or the novel, multimedia is still in its earliest days. The CD-ROM specification was published in 1985, with drives appearing on desktop machines from about 1989; the World Wide Web became publicly available outside CERN at the start of 1992, in the form of a line-based browser giving access to a handful of servers; by January 1997, when the HTML 3.2 specification was adopted as a World Wide Web Consortium Recommendation, audio and video were only supported in Web pages through the use of proprietary extensions. We have become so accustomed to the high speed of technological change in computing that multimedia is already seen as an established feature of the computing landscape. But, on the time-scale of cultural change, it has barely arrived.

The history of film and animation, for example, demonstrates that it takes time — much longer than digital multimedia has existed — for conventions about content and consumption to become established. The very first films exhibited to the public, by the Lumière brothers at the Grand Café on the Boulevard des Capucines, in Paris in 1895, showed such subjects as workers going home from a factory, and a train arriving at a station. To be able to reproduce movement was enough, without the farrago of plot, character, or message we normally associate with films. The early trick films of Georges Mélies were shown as part of his magic show, in the same way as the magic lantern projections he used to create illusions. In a similar vein, Winsor McCay's *Gertie the Dinosaur*, one of the first short animations made in America (in 1909), was used as an integral part of his vaudeville act. While Gertie did tricks, McCay stood in front of the screen on stage and talked to her, telling her what to do, and scolding her when she got it wrong[1], in response to which Gertie started to cry; finally McCay appeared to walk into the frame and ride off on the dinosaur's back.

At the same time, films, including narrative and animation, were already being shown, particularly in Europe, as entertainments in their own right, purely to be watched on a screen. The diversity of early ways of using film shows that there was originally no consensus about a 'right' way to present a film. Later on, anything other than a cinema screening would be seen as eccentric. Even films that attempted to use the medium in original ways would mainly do so within the cinema context, and a film shown in other ways, for example projected onto the floor of a gallery, would be re-defined — for example, as an 'installation'.

Another notable feature of early cinema is the way in which established forms were translated into the new medium. In particular, the newsreels shown around the time of the First World War were based on the same principles as a newspaper, even to the extent of including an animated cartoon corresponding to a comic

[1] In some versions of the film currently in circulation, this monologue has been added as a soundtrack, but the disembodied voice does not convey the presence or interactive nature of the vaudeville act.

strip. Characters were transported from the comic strips to film, and the animators who brought them to life were often (but not always) the artists who had drawn the strips. Perhaps more to the point, one of the most succesful studios producing newsreels and animations was the International Film Service, owned by the newspaper proprietor William Randolph Hearst — who also owned the syndication rights to many popular comic strips of the period.

Remember that the time we are now describing was twenty years after the invention of cinema, yet we still find film looking to earlier media for its forms and content. In a similar way, multimedia still adopts the format of earlier media; the most obvious example is the multimedia encyclopedia, which has the same form — a series of short articles, accessible through one or more indexes — as its paper equivalent, and is often published by the same organization. Other reference works also follow the form of reference books, sometimes, as in the case of some of Dorling-Kindersley's CD-ROMs, following the distinctive house style of their originals. Electronic software manuals rarely depart far from the organization of hard-copy manuals, merely adding better searching facilities and sometimes some animation.

One of the things that was needed before film could acquire its distinct character as a medium was an appreciation of the way the movie camera and film could be used to create new sorts of images, by framing, movement, editing and special effects. In multimedia, part of what is presently missing is a real understanding of how we can take advantage of the fact that digital multimedia is data, to integrate the presentation of multimedia with computation. For the moment, we are largely confined to controlling the presentation interactively, and to moving the data across networks.

Multimedia Production

The making of multimedia requires software not only for the preparation of individual media elements, but for their integration into a finished production. Programs that allow a designer to assemble different media elements in space and time, and add interactive behaviour to them, are usually called *authoring systems.* These must interpret user actions and commands in non-trivial ways in order to translate the mode of working that is natural to a designer, who will usually feel most comfortable with direct manipulation of the elements on a virtual pasteboard or its equivalent, into the data and control structures of the finished piece of work.

Multimedia authoring systems take many forms, though: if we are currently unsure about the form of multimedia, we are equally uncertain about the best means of producing it. Again taking film for comparison, we find a well-defined set of jobs, from director to clapper operator, and established procedures for coordinating the stages of production that are involved in the making of a finished film. No equivalent division of labour or procedures have yet been established for multimedia. Some authoring systems, such as Director, attempt to provide tools for the production of all types of media element as well as for their integration, almost as if a single person could do the whole job, working on different parts at the same time. Other tools, such as Photoshop, only work with one type, suggesting a division of labour along the lines of the different media, but requiring separate mechanisms and a separate production stage for integration.

Just as the forms of multimedia are largely derived from the forms of other media, so attempts to impose order on the process of making multimedia are largely based on procedures from other media. *Storyboards*, for example, are sometimes advocated as a means of making and communicating multimedia designs. Storyboards are used in large animation studios, and by music video and advertisement producers, to plan the construction of a piece of work, and to communicate its structure among the team producing it. (In the industries where storyboards are widely used, they are also a means of selling ideas to financial backers.) A storyboard is simply a sequence of still pictures, showing the composition of shots at key points in the production. Rather like a blueprint, it records a plan of the work, which can be used to organize its subsequent development. One of the things that makes storyboards effective is the way they can be read linearly like a comic strip. As we will see in later chapters, and as you probably know from your own experience of Web pages and CD-ROMs, multimedia is often arranged non-linearly, with the possibility of taking branches, so that the elements may be displayed in different orders. When non-linearity is added to a storyboard, by annotating it with arrows, for example, the clarity with which it communicates structure, and thus its *raison d'être*, is lost. Conversely, starting out with the fundamentally linear format of a storyboard when planning multimedia will inevitably prejudice the designer in favour of linear structures.

> A parallel can be drawn with the use of flowcharts for program design. The conventions of 'classical' flowcharts can only represent conditional and unconditional branches. A flowchart design therefore suggests an implementation in terms of jumps, not in terms of the higher level control structures supported by modern programming languages. The 'structured programming' movement of the 1970s discarded flowcharts, in favour of working directly with programming language constructs that express control abstractions. It might be hoped that multimedia authoring tools might themselves take on a design rôle, in a similar fashion.

Terminology

A telling symptom of the immaturity of digital multimedia is the absence of satisfactory terminology. What do you call a mixture of media under software control? Where the display and presentation of the media elements is the sole purpose, as in a Web page or an encyclopedia on CD-ROM, we will usually refer to a *multimedia production*. Where the display of the multimedia is more intimately bound up with computation, we will refer to a *multimedia application*. A simple example of a multimedia production might be a Web page containing a chronological listing of forthcoming film festivals, with some attached images and video clips. A multimedia application using the same data could provide an interface (which might still be a Web page) to a database containing the festivals' details and related material, so that a user could search for festivals in a particular geographical area, or those that accepted material on video tape, and so on. The distinction between the two is not a sharp one. A certain amount of computation might be done to control the presentation of a multimedia production, so there is a continuum from productions that simply present material in a fixed way, through increasingly complex interactivity, to applications that generate multimedia dynamically.

There remains another question. We know that we *read* text, *look at* images, *watch* video and *listen to* sound, but what do we do to a multimedia production? All of these, perhaps, as well as interacting with it. There is no satisfactory term either for the act or for the person who performs it. For the latter, we reluctantly fall back on the over-worked and vague term *user*. The usage at least has the virtue that it draws attention to the active rôle that, potentially anyway, a user adopts with relation to most multimedia productions. Although much influential critical work has focused on the active rôle of the reader in creating readings of texts, this activity is of a cerebral and largely unconscious type. In contrast, the user of multimedia is active in a literal sense, using input devices such as the mouse and keyboard to control what goes on — within the limits allowed by the multimedia producer.

It can be useful to distinguish between *multimedia* and *multiple media*. The distinction is best understood from the point of view of the user. It is commonplace that we perceive different media in different ways: we just referred to reading text, looking at pictures, listening to sound, and so on. Cognitive scientists call these different sorts of perception *modalities*. A multiple media production requires us to switch between modalities, for example, by reading some text, then looking at a picture, and so on. Many early 'multimedia' CD-ROM presentations were like this. True multimedia requires us to combine modalities (as we do in real life — think of the experience of walking through a shopping mall, for example), or even develop new ones, to cope with the combination of media. A familiar example is the pop video, where the music and moving images are (usually) presented as a composite — arguably often to the detriment of one or the other considered in isolation — with, for example, visual action illustrating lyrics, and scenes being cut in time to the beat of the music.

While modalities provide a useful way of thinking about multimedia from a cognitive point of view, we prefer to adopt a definition more in keeping with the technical emphasis of this book, and consider digital multimedia to be

> any combination of two or more media, represented in a digital form, sufficiently well integrated to be presented via a single interface, or manipulated by a single computer program.

This definition is broader than some that have been proposed. It is quite common to require at least one of the media to be a time-based one, such as sound or video. It is certainly true that, at present, the addition of time introduces significant new technical difficulties, which may be considered sufficiently challenging to make a qualitative difference between such dynamic multimedia and purely static multimedia, involving only text and images. However, we consider these difficulties to be transient effects of the current state of the underlying digital technologies, and prefer to use the more general definition just given.

Interactivity

"You can mistake endless choice for freedom"
Bruce Springsteen, speaking on BBC2, December 1998

He was talking about something more momentous than multimedia productions, of course, but his words are curiously appropriate to the way in which interactivity has been applied to multimedia.

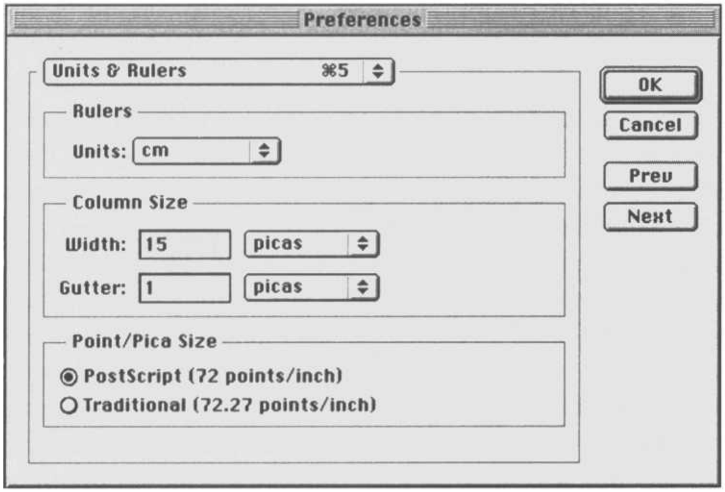
Interactivity is frequently portrayed as the feature that distinguishes digital multimedia from other forms of combined media, such as television. It is often claimed that it puts the user in control — by implication, in a way that has not previously been possible: "Interactivity empowers the end users of your project by letting them control the content and flow of information."[2] While there is some truth in this, it should be recognized that the amount of control offered is strictly limited within parameters established by the multimedia producer. This must be the case in any situation where interaction takes place with a finite system responding according to a program. Only choices that are coded into the program are allowed. However, providing choices in a computer program is much easier than providing choices via a hardware device — such as through the controls of a VCR — so it is possible in a multimedia production to allow the user to control events at many points. Where a sequence of choices can be made in succession, the possibilities expand combinatorially: if, for example, there are four choices at each of five stages of a production, although only twenty branches must be programmed, there is a total of $4^5 = 1024$ possible sequences which can be followed; that is, 1024 possible ways of experiencing the production. Such a range of possible experience might well be mistaken for freedom, but ultimate control over "the content and flow of information" remains with the producer not the user, and even a significantly increased range of choices will not necessarily enhance a production. In certain cases — e.g. the provision of access to information in

[2] Tay Vaughan [Vau98].

facilities such as Web site railway timetables — a single, fast, optimized route to the required information is all that is needed. And no amount of interactivity can compensate for poor or dull content, or bad organization of the work — one of the worst cases of this is when, through a series of selections on buttons, etc., one navigates through a site to find that the desired page is an empty space labelled "under construction".

The character of interaction can be appreciated by considering how it operates in the popular game *Myst*. The player is presented with a series of images, depicting scenes in an imaginary world. Sometimes, if you click on an object, something happens — doors open, machines operate, and so on. When the cursor is placed near the edge of the screen, it changes shape and a mouse click causes the scene to change, as if you had walked out of the frame into the adjoining space; in this way you can, for example, walk round a building. Apart from that, there are no controls, such as buttons, so the game demands a process of exploration before you can find out how to make anything happen. Advertising copy and reviews of the game placed great emphasis on the idea that you have to explore a fantastic world. In fact, though, all that your choices and explorations enable you to do is to complete a few little puzzles, obtaining fragments of a story along the way, until you reach the end. You don't have any choice over this — if you can't work out the combination to open a door, you can't attack it with a crowbar instead, for example. All you can do is go off and try a different puzzle, or give up. However, the extraordinary popularity of *Myst* demonstrates how interactivity, when embedded in a rich environment with high-quality graphics and evocative sound, can dramatically increase the attractiveness of a production. It is hard to believe that the puzzles presented on their own, or the flimsy piece of fantasy fiction upon which the game is built, would draw the same audience.

⇨ 'Interactivity' is really a misnomer — although we will continue to use the term in deference to its wide currency. When the computer's rôle is to present choices and respond to them, it cannot be said to be keeping up its end of an interaction, while at the same time it reduces the user's options for contributing to the intercourse to a few mouse gestures. The application of artificial intelligence can improve the level of discourse, but (with all due respect to Alan Turing) true interaction is only possible where another person is involved. Even then, if the interaction is mediated by a program, the form it can take may be severely limited. In many contemporary



Preferences

Units & Rulers ⌘5

Rulers

Units: cm

Column Size

Width: 15 picas

Gutter: 1 picas

Point/Pica Size

● PostScript (72 points/inch)

○ Traditional (72.27 points/inch)

Figure 1.1
A typical dialogue box

> networked games, for example, the only way you can interact with your fellow players is apparently by trying to kill them.

Since interactivity is a novel contribution that computers make to multimedia, there is a strong tendency for all multimedia productions to be interactive. It is worth remembering, though, that 'endless choice' may not only be inappropriate or redundant in the ways described above — it is not necessarily what everybody always wants. In some cases an audience may be quite content to let a story unfold just the way the writer or director intended it to, without the distraction of commentaries, out-takes, or supplementary material. In many of these cases, it would probably be better to read a book or go to the pictures — digital multimedia is not appropriate for everything. There will also be times when producers do not wish to offer interactivity to the user. An increasing number of artists are creating work in the form of digital multimedia, but in many cases they choose to retain control of what is shown, and do not allow their audience to interact with the work. Such productions can be innovative, and even seek to push the boundaries of what can be achieved in the digital domain, without being interactive or departing from an essentially linear form of presentation.[3]

[3] Perhaps in the future some digital multimedia productions may provide a linear experience that is nevertheless unpredictable, because the development at each viewing could be controlled by computation.

User Interfaces

The means by which choices can be presented to users can vary enormously. At one extreme there is the stylized set of user interface elements — menus, dialogue boxes, outlined buttons, and so on — and conventions used by most operating systems and mainstream applications; at the other extreme, interaction with some sorts of game is essentially free-form, with any part of the screen liable to cause a response at some time. Adoption of the conventions has the advantage of predictability: users will generally know what to do when confronted with a dialogue box, such as the one shown in Figure 1.1, containing text entry areas, pop-up menus and buttons — fill in the text, make a selection from the menus, click on the buttons. By following the interface guidelines for the platform your multimedia production is intended for, and designing your dialogues and other interface elements carefully, you can often make it possible for an experienced user of that platform to work with your production without further instruction. A disadvantage is that users who are not already familiar with the interface conventions might find it harder to use a fully-fledged conventional interface than a specially designed simpler one. Another disadvantage is that everything looks the same; you might have legitimate reasons for wanting to differentiate the appearance of your interface from the operating system. Doing this while at the same time making it clear how your various controls work is a difficult task. The innovative interfaces designed by Kai Krause for products such as KPT and Bryce (see Plate 1[4] and Figure 4.49 in Chapter 4), for example, have sharply divided users, with some hailing them as a breakthrough while others criticize them for being confusing and affected.

The interface elements that have become most familiar to desktop computer users were devised in the context of a static graphical environment. With the incorporation of time-based media — video, animation, sound — new types of operation, such as starting, pausing, rewinding, and adjusting the volume, become possible, and so new types of control are required. Menu commands with keyboard shortcuts could be used, but the practice of supplying playback controls in the form of buttons that mimic the operation of those on cassette players and VCRs, and are labelled with the *de facto* standard icons used on such appliances, is more common. Volume controls sometimes have the appearance of sliders or

[4] By default, Bryce's interface takes over the whole screen, obscuring the desktop entirely and hiding the menu bar the way some games do. This is in direct contravention to at least the MacOS user interface guidelines — it's actually quite difficult to do on a Mac — but it doesn't prevent the program from being usable, at least in many users' opinion.

thumb wheels. Since digital video players make it easy to go to a specific point or scrub through a movie, using mouse movements, an additional control is often supplied in the form of a strip, representing the movie, with a marker that can be pulled through it; this marker moves during normal playback to show how much of the movie has been played (see Figure 1.2). The requirements of incorporating time-based media have thus given rise to new interface conventions. As always, these conventions can be ignored, and a free-form or individualistic interface substituted where that is felt to be justified by the nature of a particular production. It should be borne in mind, however, that accepted interface designs are the tried and tested products of expert designers, and the attempt to create a radical or superior interface should not be undertaken too lightly.

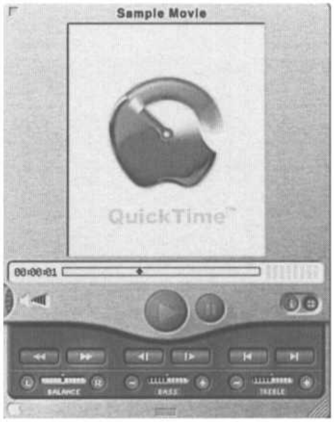


Figure 1.2
Controls for playing video

Social and Ethical Considerations

It is a commonplace among technologists that technology itself is neutral and without moral value, either good or bad. It is only the uses to which people choose to put technology that need to be subjected to moral scrutiny. While this assertion is literally true — because ethics by definition applies only to the thoughts and actions of sentient beings — it is often the case that, in a specific social, political and economic context, the introduction of new technology presents opportunities for behaviour that were not there before. In such cases, certain ethical problems caused by that behaviour can conveniently be described as arising from particular technological innovations.

Access to Multimedia: Consumption

In the case of multimedia, the issue springing most directly from the nature of the technology itself is that of access. To begin with, access to multimedia depends on access to appropriate hardware and on possession of the necessary skills to use a modern computer system. In developed countries, access to hardware is largely a function of wealth. Although personal computers have become a consumer commodity, they remain among the more expensive of such items: a personal computer equipped to handle multimedia costs substantially more than a standard VCR or a games console, for example. In less developed countries, computers remain rare, and the infrastructure to support modern computer systems — reliable power supplies, ubiquitous telecommunications — is absent. In some countries large numbers of people are still denied the opportunity to acquire basic literacy — this, too, is often a side-effect of poverty. It is sobering for the multimedia developer to remember that substantial numbers of people around the world live without even those basic domestic facilities and appliances which members of the more privileged societies take for granted — in conditions where issues such as access to clean water are of vital importance, and access to computing not even in question. Any talk of the global nature of the Internet, or of multimedia distribution, should be tempered by this knowledge.

Even among the wealthiest countries, local conditions can affect the ease with which certain kinds of multimedia can be accessed. In most parts of the United States, for example, a single charge covers the cost of all local telephone calls, effectively making individual calls free. It is thus quite feasible to leave a personal computer permanently connected to the Internet over a telephone line. In the United Kingdom, local telephone calls are charged for by the second, so most people with dial-up Internet connections try to minimize the time they are online. Somebody in Britain is much less likely to download huge video files, or tune in to a live Web audio broadcast, than their counterpart in the U.S. The Internet is a global phenomenon in principle, but in practice it wears different faces in different places, and in many places it has no face at all.

There are other factors which can prevent access to multimedia. As we stated earlier, the ability to use a computer system is a prerequisite — and this in itself presupposes basic literacy. These skills are by no means universal, even in developed countries. There is conflicting evidence about how computer skills are distributed among age groups. Certainly, young people in most developed countries are now routinely exposed to computers in schools, and often at home, if only through games. But the skills they acquire may be superficial, and the quality of computer education varies as much between schools as any other subject does. The highest levels of computer skills are usually acquired through work, so a failure to master computer technology may be an increasingly significant effect of long-term unemployment. And, of course, not all forms of employment expose people to computers, so disparities of skill are found between occupations, which may, in time, reinforce other social distinctions — e.g. between office and manual workers. People who are too old to have been exposed to computers in schools and who have not used computers as part of their work may find even a user-friendly graphical interface perplexing, so, even if they can afford the hardware to access multimedia, they may not feel able to take advantage of it.

Finally, physical disabilities or learning difficulties may interfere with a person's ability to use computers or experience multimedia content. This is particularly poignant as computers can be of great value in ameliorating some of these difficulties. A range of problems, from arthritis to motor neurone disease, may make it difficult or impossible for someone to operate a conventional keyboard, mouse and other input devices. Blindness and visual impairment raise specific concerns. Voice synthesizers and Braille keyboards and output devices can make it possible for blind people to use computers effectively (again assuming sufficient wealth or social provision makes such devices available), but most multimedia has a marked graphic bias, relying on images not just to convey information, but for navigation and interaction. This is evident on the World Wide Web, where many Web pages use image maps for navigation; the habit of using small images as navigation icons (or merely to provide text labels in a more interesting font than the defaults) means that some Web pages are unusable without images.[5] People who are dyslexic, on the other hand, typically find graphic images and icons more readily intelligible than text, and will be assisted by good design of these elements. Although it is well known that many people have difficulty in distinguishing between certain colours, and most books on design offer advice on how to choose colour combinations to minimize the problems arising from this, few designers appear to apply this advice when it conflicts with their aesthetic intent. The range of difficulties which may be experienced by potential users is very wide, and embraces all media elements, with correspondingly far-reaching implications for the conscientious multimedia designer.

The World Wide Web Consortium has done an exemplary job in addressing the problem of access for the disabled (we will describe the details in Chapter 13), but there is little evidence at present that many Web site designers are taking advantage of the facilities to make their Web pages more accessible to everyone. It is not within the scope of this book, or our own fields of expertise, to do justice

[5] If you don't believe this, try turning off image loading in your Web browser.

to this aspect of access and to provide solutions, but we stress that this is an area which requires the most careful attention.

Provision of computers in public libraries, schools, community centres and Internet cafés, inexpensive set-top boxes and network computers, popular education programmes, and accessibility initiatives can all broaden access to the Internet and other forms of multimedia. But it should be understood that access is not, and probably never will be, universal, especially outside the industrialized nations. As long as the bulk of multimedia is directed to entertainment, this denial of access cannot be considered too serious — one can, after all, live without *Riven* or *Cinemania.* Nevertheless, if access to multimedia is portrayed as the norm in a society, to be without that access becomes a marginalizing influence. When advertisements routinely feature URLs, and you have no Internet connection, what message are the advertisements conveying to you? If multimedia does indeed become the primary channel for information dissemination, as some pundits envisage, being denied access to multimedia will mean being denied access to information. On a mundane level, when information about entitlement to social security benefits, refurbishment grants or medical aid is readily available from the home to those with an Internet connection, in the form of a slickly packaged interactive multimedia presentation, but requires a trip to a crowded and depressing government office to pick up a leaflet for those without, it is easy to see how the shift of information provision to new media can deepen existing economic divisions. In extreme cases, simple access to information saves lives, as with health education programmes. Access to information is empowering, and a deliberate denial of access to information has always been one of the most powerful weapons used by governments and the rich and powerful to exercise control and protect their interests. These issues have a social and political reality, and are not simply a matter for peripheral academic debate by multimedia students and producers. The work you make, if it is released to the world at large or to a part of it, will affect other people's lives.

Access to Multimedia: Production

Anybody with access to the Internet can have their own Web site — and anybody without can't. People who cannot use the Internet for any of the reasons given in the preceding section are not only denied access to the information it contains, they are also denied a platform for expression or advertisement. However, in the parts of the world and segments of society where Internet access is readily available, what computing people would call write access is almost as widespread as read access. An Internet Service Provider that did not offer several megabytes of free Web space to its customers is unlikely to survive for long. The availability of inexpensive tools for constructing Web sites (and the possibility of constructing sites by hand in a simple text editor), plus the relatively small effort needed to acquire enough knowledge to use these tools, means that anybody who feels they have something to say to the world can say it on the World Wide Web. Relatively affordable computers, together with very low cost access to the Internet and cheap CD-ROM technology, is bringing about a widespread revolution in access to the means of production and distribution of digital material.

In contrast, access to the means of production and distribution of traditional media is tightly restricted. Probably the most accessible medium is the printed book. It is possible for an author to write, print, and distribute their own books, but the costs involved are considerable, and the likely returns, in the absence of any marketing machinery, are slim. Such personal production is reserved for the independently wealthy. Most books are published through publishers, who are able to take advantage of their size to spread production costs, and to market and distribute their books more effectively. But publishers can only stay in business by making money, so they can only publish books that they think are going to sell. This means that book proposals are submitted to editorial scrutiny and refereeing — with the result that established authors have a better chance of being published than newcomers, and unconventional subject matter or styles of writing that only appeal to a minority of readers are unlikely to appear in print at all. There are small independent publishers — especially in the United States, where there is a tradition of independent and underground presses — who may pick up marginal work, but, like independent authors, their limited resources make it hard for them to achieve wide distribution. Contrast this with the Web, which offers a potentially enormous audience to anybody.

The situation in other traditional media is similar to that for books. Mass distribution of printed images takes place largely through magazines, newspapers and books, and is subject to the same editorial control as the distribution of printed text. Many illustrators and photographers also sell their work to greetings cards, calendar, or postcard manufacturers, or provide images for brochures or corporate publications. Here, the influence of marketing pressures and corporate values will clearly limit the type of work that can be used. Fine artists wishing to exhibit their work are often faced with even higher barriers. Most exhibitions take place in private galleries which survive by taking large commissions on the sale of work, and can therefore not afford to take risks; public galleries, such as the National Gallery or the Museum of Modern Art, if they exhibit work by living artists at all, normally only present the work of established artists. Even if an unknown or unfashionable artist can manage to get their work exhibited, it will only reach the few people who are able to be physically present in the exhibition space. Whereas printed images can be turned into digital form and disseminated through the Internet or on CD-ROMs, the very nature of paintings and other art works means that digital dissemination can only be done at the cost of a substantial degradation of quality, and with a loss of the presence that may well be the essence of the work. However, fine artists, with their trained habits of exploring media, may be in the best position to take advantage of new opportunities that digital media and multimedia offer.

Most recorded music is distributed by a few labels, owned by huge multinational corporations, whose sales dwarf those of the independent labels. Because of the investment that a record label must make to sign, record and promote a band, companies try to ensure that they only make records that will sell, hence the notorious difficulty faced by new musicians trying to break in to the business. Again, contrast this with the ease with which anybody's songs can be placed on the Internet for downloading.

The enormous cost of making even a 'low budget' film means that access to the medium of film is even more tightly restricted than the other forms we have considered. Among the established media, only television can begin to provide any sort of broad access. The relatively low cost of video equipment compared to film, and the absence of any specialized processing requirements, means that video can be produced on a very restricted budget — although not to broadcast quality. Transmission, however, is controlled, as much as in any other medium, if not more, and opportunities for public access to broadcasting are limited to a few derisory slots on a minority channel in the UK, or one cable channel among fifty or so in New York, for example. We will see in Chapters 10 and 15 that emerging network and digital video technologies mean that anyone with a video camera, a contemporary desktop computer and a Web site can broadcast their own video over the Internet.

In discussing access to multimedia production, we have concentrated on the World Wide Web. This is because methods of offline delivery do not offer the same possibilities. Since a CD-ROM is a physical object, it must be manufactured and distributed. Additionally, a significant amount of content is needed to fill a CD-ROM. Production and distribution of CD-ROMs therefore more closely resembles that of audio CDs or videotapes than the unique forms of online delivery. Nevertheless, the relatively cheap cost of CD writers and the very low cost of the CD media itself, has meant that to some extent a new form of self-publishing has opened up for those people with sufficient funds and motivation. It is now possible for artists of all kinds to distribute high quality portfolios or catalogues of their work themselves, or for musicians to make their own cheap CDs.

While the description we have just given is incomplete and simplified, it should convey the fact that access to traditional media is highly exclusive in comparison with access to the World Wide Web — which has the bonus of supporting interactive multimedia. It is hard to imagine a publisher issuing a book directed only at pedestrians, which lists gardens that are open to the public and can be visited on foot or by public transport, but one man is preparing a charming Web site on that subject as a personal Millennium project. Unfortunately for proponents of democratic access to multimedia, putting up a Web site is not the end of the story. People also have to visit it.

Control of Multimedia

In theory, all Web sites have equal claim on our attention, but, in practice, some exert a stronger appeal than others. Recent evidence suggests that the metaphor of 'Web surfing' — following links at hazard in the hope of coming across something useful or interesting — is no longer an accurate way of describing people's behaviour (if it ever was). Most people have a limited number of sites that they visit habitually, and only a small number of sites, compared with the vast number that exist, attract more than a few visitors. Many of the most popular sites on the World Wide Web are the home sites of Internet Service Providers, which typically provide a news service, little articles about technology or lifestyles, horoscopes, advertisements, and, increasingly, online shopping. In fact, these Web portals, as they are called, are online Sunday supplements, echoing the early development of film, with its impersonation of newspapers. This Sunday supplement model of a Web site is now also provided by sites such as AltaVista and Excite, which originally functioned purely as search engines.[6] The main competition to Web portals are online stores, that is, digital mail-order catalogues. They may be more convenient for home shoppers, but the form of the online catalogues closely mimics their off-line originals, and very little extra functionality is offered beyond basic search facilities.

In addition to importing established forms, the World Wide Web is also importing content from established media. In 1999, Apple launched QuickTime TV, providing streamed video over the Internet. Like a conventional cable or satellite service, QuickTime TV is organized as a collection of channels. The current collection of channels includes ABC News, Fox Sports, Warner Brothers, VH-1, Rolling Stone, WGBH, and the BBC World Service. There is little evidence of public access provision here.

It is hardly surprising that companies should follow established advertising and marketing patterns when confronted with a new medium, or that Web content providers should turn to existing conventional sources for news and entertainment. The notion that the comparative ease with which anybody can put up a Web site provides an opportunity to wrest control of the means of distribution of ideas away from the old establishment does not bear much scrutiny, and is not supported by the evidence. The pattern of domination by identifiable large companies, corporations and institutions has, by and large, been transferred intact to the Internet. However, in addition to this predictable transference to the new media, the Internet has enabled the beginnings of something more interesting: the opportunity to create new kinds of relationship between consumers and producers of multimedia. As well as the one-to-many relationship of conventional broadcasting, publication and marketing, which the Web portals echo, we are seeing a new development of few-to-few relationships, between small groups of individuals with specialized interests, who in reality may be physically very isolated from one another. The structure of the Internet and the low production cost of Web pages provide a support for such relationships which is denied by the politics and economics of the established media.

[6] We will describe search engines and Internet portals in Chapter 9.

The Control of Content

"Then we must speak to our poets and compel them to impress upon their poems only the image of the good, or not to make poetry in our city. [...] He that cannot obey must not be allowed to ply his trade in our city. For we would not have our Guardians reared among images of evil, and [...] gather many impressions from all that surrounds them, taking them all in until at last a great mass of evil gathers in their inmost souls, and they know it not."

Plato, *The Republic*, Book III

Some issues do not go away or become resolved in a hurry: more than 2300 years have passed since Plato wrote *The Republic*, and yet very similar concerns about the protection of impressionable young people are frequently expressed today. It is precisely the enduring nature of ethical problems — which deal with fundamental issues of how human beings can and should conduct themselves — and the great difficulty in finding satisfactory and practical solutions, that has fuelled the continuation of both academic and real-world debates on these subjects.

Human history to date has shown that any sufficiently complex society has sought to exert some control over what people may read, see, hear, or do, whether this is done by explicit policing or by economic or other, less tangible, means. In the twentieth century this control has usually been justified as a benevolent protective measure, and may well be broadly supported by the electorate. It is interesting that the same type of justification has been used by political systems which were radically opposed to one another — each supposedly protecting their own population from the potentially harmful influences of the other, for example in the idealogical combat known as the Cold War.

The rapid growth of the Internet, which offers an unprecedented dissemination of possibly unacceptable or unsuitable material, has given a new impetus to debates about censorship. However, the ethical issues surrounding censorship are complicated, and a very long history of discussion has produced no enduring conclusion or consensus about them. The novelty of the contemporary situation lies in the international reach of the Internet, and its availablility within people's homes. Multimedia has not yet managed to raise any new issues of content — and it seems improbable that people will have anything to express in this medium which has not already been expressed many times before in other forms. The new dimension is entirely one of access and distribution. Because the World Wide Web is currently the most effective means of distributing multimedia, anybody who is involved in multimedia production may find themselves drawn into the censorship debate, even though they may be poorly equipped to resolve the difficult ethical problems which have evaded a solution for centuries, and which have only been made more complicated by the very wide reach of the Internet. If you have a serious interest in this debate you will need to do a substantial amount of further reading in ethics generally, and censorship issues in particular, to develop a well-rounded and informed view of the situation. If you prefer to concentrate your efforts elsewhere — as most people working in traditional media have chosen to do — you would be prudent to avoid being drawn in unnecessarily, but you will need to be aware of how the debate is focussed, and of the nature of the main issues.

Within a single modern state, mechanisms generally exist for controlling publication, exhibition, performance and so on, which are consistent with the political conditions of that state. The plurality of political and social systems that exist in the world means that it is unrealistic to expect a single model of censorship — whether that means no censorship, rigid centralized control, self-regulation, or any other set of mechanisms — to be acceptable everywhere. And yet, Internet content is available everywhere[7] and the way the network is organized makes it difficult to assign responsibility for the dissemination of content, even in cases where it is desired to do so.

A society's history, its state of development, political system, and religious institutions and beliefs are among the factors that determine precisely what sorts of media or multimedia content are considered objectionable within that society. Some groups of societies may have a broadly similar history and culture, deriving from certain shared roots, which will mean that their basic concerns are concentrated within the same areas. But even in these cases there is often disagreement about the precise limits of acceptability. For example, the level of female semi-nudity that has been commonplace on beaches in the south of France for many years has been, during the same period, considered indecent on the other side of the English Channel (where it is still daring, at the least), while at the same time complete nudity of both sexes is acceptable at bathing places in Scandinavia. Where there is greater

[7] Subject to the limits we described earlier.

cultural diversity, there is a corresponding diversity of opinion about decency.

In any society there is often a substantial separation between what is acceptable in a portrayal of life — i.e. in media — and what is acceptable in reality. In some societies, much greater leniency is extended towards representations of behaviour than to the real thing, but in others there is a much closer correspondence of the restrictions which govern media on the one hand and actual human behaviour on the other. In some there are very stringent restraints on specific media, for example on image making. These different sets of standards will be further sub-divided along lines of content: a society may sanction representations of violence while punishing those who actually do violence on others, and at the same time censor images of the sexual activity which in practice that society condones. A quick look at the films of the United States in the 1930s will bear this out. At this period even happily married couples had to be shown as sleeping in separate beds, and never apparently enjoyed any intimacy beyond a kiss and a cuddle, but at the same time gangster films and thrillers portrayed scenes of violence and murder which were illegal and generally condemned in reality. When the complexity of the standards for censorship within a single society is multiplied by the number of societies within the world, and then by the cultural and religious groups within those societies, the result is a set of principles, beliefs and practices which are very diverse indeed.

There is no alternative to accepting this diversity, so how can it be accommodated in the context of the Internet and other modern communication systems? To simply permit everything is not accommodation — restrictions, even when we may not personally approve of them, are part of the diversity. Just as it would be unacceptable to impose one society's set of restrictions on everybody else, so is it unacceptable to impose one society's freedoms on everybody else. (We are not talking here about matters which fall within the scope of 'Human Rights', which is a separate area of ethics in which an international consensus has now more or less been established through, for example, the United Nations Declaration of Human Rights.)

The *Platform for Internet Content Selection (PICS)* is an attempt to provide a mechanism that supports a diversity of attitudes towards content and censorship. The difference between PICS and conventional mechanisms, such as the banning of books or the seizure of video tapes and magazines, is that it restricts reception, not distribution.

> ⇨ An established pattern for the censorship of films, as practised in the United Kingdom and other countries with similar systems, tries to combine restrictions upon reception with some control of distribution. In extreme cases censors can refuse certification, effectively preventing the showing of that film in most venues, or they may require scenes to be edited out before a certificate can be awarded. These measures affect distribution, but by labelling each film which 'passes' with a certificate that defines what age group may legally be admitted to view that film, the censors also attempt to control reception — although in practice in the UK, for instance, this is not put into effect very rigorously. A significant flaw in this system, which has been addressed by certain video tape distributors, is that no indication is given of the nature of the content which was deemed unsuitable. This makes it very difficult for anyone to exercise their own control over what they are exposed to at the movies, unless they have access to information about the film's level of violence, or whatever, from other sources.

Acknowledging the virtual impossibility of preventing material that is unacceptable to somebody somewhere appearing on the World Wide Web, PICS aims to allow the user themselves to prevent unacceptable material reaching them. The underlying idea is simple: labels are associated with each Web page, to provide a rating of its content that can be used to assess its desirability. Undesirable material is rejected by screening software that is either incorporated in a Web browser, or implemented at a lower level to intervene in all network traffic carrying Web pages. All that PICS stipulates is a standard format for content labelling; it says nothing about what the labels should contain, nor how software should deal with them. PICS thus allows for arbitrary criteria to be used for assessing content, and arbitrary strategies for dealing with it.

PICS may sound like a laudably flexible, value-independent, means of blocking the reception of certain sorts of material, but it achieves this by deferring the difficult decisions. Who does determine the criteria for blocking, who ensures that software can apply the criteria, and who attaches the necessary labels to Web pages? In theory, PICS would allow pages to be labelled with, say, a green rating, so that pages that advocated environmentally damaging practices and policies could be filtered out. For this to work, filtering software would have to be made aware of the existence of the green label, so that it could be configured to use it as a filtering criterion.

A PICS *service description* is a document that specifies the format of some rating service's labels. Filtering software can download service descriptions so that they can allow users to specify controls based on the labels they describe.

Most filtering is performed by Web browsers which, by default, only understand the labelling system of the Recreational Software Advisory Council (RSAC). This system is restricted to a crude set of measures of violence, sex, nudity, and language. Other rating systems can be used if appropriate service descriptions are available, but there is no guarantee that any sites will be labelled in accordance with them, whereas RSAC ratings are widely used. In practice, therefore, instead of the flexible, multi-dimensional content screening that PICS promises, we find the crude and culturally specific RSAC system.

The PICS architecture allows labels to be attached either by the page's author, or by some third party vetting organization. Where third parties do the labelling, a mechanism is available for associating labels with URLs without attaching them to the page itself, and for enabling the label to be inspected, to determine whether to allow access to the page. The green label could only be applied if concern about green issues was so widespread that Web authors felt an obligation to include green ratings, or if an ecological organization was able to set up a ratings bureau and vet Web pages. (There are more Web pages than people on this planet.) However, if there is no third party who shares your concerns, and you do not necessarily trust Web authors to label their pages, you have no means of preventing material you do not want to see from reaching you, except by complete avoidance of anything you think may be unacceptable. Even if a third party service description does embody some criteria you wish to apply to content, you have to trust the labelling service. There is no regulation of such services, so there is room for abuse. There have been allegations that some labelling bureaux have blocked access to pages critical of their own policies, for example.

PICS labels could be used for a number of purposes other than those for which they were intended. For example, it would be trivial to write a Web searching program which sought out Web pages with certain types of content, simply by looking at the PICS labels. Ironically, the mechanism intended to restrict access to certain material is itself the means by which that material may be most easily located.

A New Medium

During the very brief history of digital multimedia to date, designers have consistently looked back to older forms for inspiration, adapting established ideas to the digital realm. This has its good side — the fundamental principles of fine typography, or the conventions of film editing provide a firm foundation for their digital counterparts — but it means that, as yet, new purely digital forms have hardly begun to emerge.

Nowhere is this more evident than on the World Wide Web. When the first browsers became available for personal computers, all Web pages looked the same. Dominated by text in Times Roman, with rudimentary layout and a few images, these pages were distinguished from printed papers only by the presence of links to other pages that looked just the same. Information content was all. Now, several years later, the Web can support elaborate layout, typography, animation, video, sound, and interactivity. Yet behind the different sorts of bells and whistles that these new elements permit, there is an essential uniformity to Web sites. They are electronic brochures, shopping catalogues, and Sunday supplements, whose multimedia elements are seen as embellishments whose primary purpose is to attract attention and gain hits. During the heyday of the multimedia CD-ROM, the same kind of formal limitation occurred: it was an electronic presentation, reference book, or again, a brochure.

These limitations have come to define a Web site; the boundaries of the form have become the boundaries of the concept. The technology we will describe in the remainder of this book is not contained by those boundaries.

Consider an Internet application that is far removed from an electronic brochure, the SETI@home project, which is based at the University of California at Berkeley. SETI stands for the Search for ExtraTerrestrial Intelligence. One way of searching for extraterrestrial intelligence is to look for radio signals concentrated in a narrow wavelength band. Orbiting radio telescopes that continually scan the sky generate enormous quantities of data about the radio signals that are reaching the Earth. Processing their raw data to search for signals with characteristics that might indicate an intelligent origin requires correspondingly enormous amounts of computer power. Searchers for extraterrestrial intelligence are not generously funded, so research groups cannot command the necessary amount of time on supercomputers. But nowadays, supercomputers are not needed, if a sufficient number of less powerful computers can be made to cooperate on the computation.

The Internet connects a huge number of computers, most of which are not doing anything useful most of the time. Quite a lot of the time, many computers are displaying flying toasters, goldfish, and space-filling curves, allegedly to prevent their monitor phospors burning out during periods of inactivity. The SETI@home project makes use of that time to look for intelligent signals from outer space. Instead of running a conventional screen saver, participants are running special software that downloads a small portion of the radio telescope data and, at times when the screen saver would normally kick in (i.e. when the user has gone to a meeting, or is having a long phone call), the program does some data analysis.[8] When it has finished with its current data set, it sends the results of the analysis back to the project's headquarters, and gets some more.

Here is a project that exposes the true nature of the Internet: it is a collection of connected *computers*. This is what the restricted concepts of 'Web site', and 'multimedia CD-ROM' omit. Naturally, they depend on the processing power of computers to display media and handle interactivity, but in doing so, they reduce the computer to a flexible media player, and deny its power to perform computation. To develop multimedia forms that accommodate and exploit that power will require a leap of imagination and understanding, comparable to the leap of imagination that took computer graphics and placed it at the centre of the user interface. Until that happens, the promise of digital multimedia is still unfulfilled.

[8] It shows a graphic display of its progress, so you still get your screen saved.

Further Information

Some of the issues raised in this chapter are taken up from a different perspective by Brent MacGregor in his Afterword.

If you are interested in the cultural theory of multimedia, [Bar77] is a good starting point. For analyses of established media, consult [Wil74] (televison) or [Kaw92] (film); [McC96] considers the impact of new technology on craftsmanship, and [Lis95] looks at digital images. [RM96] is a short introduction to the mechanics of PICS. [EF99] and [App93a] both provide a management-oriented view of multimedia that complements our technical treatment. [Fau98] gives a concise introduction to interface design.

Exercises

1. Is the idea of making a film of this book an absurd one?
2. The late Kingsley Amis reportedly once asserted that he would never read another book unless it began *A shot rang out!*. How would this opening be realized in

   (a) A radio play;

   (b) A stage play;

   (c) A film;

   (d) An animation;

   (e) A single created image;

   (f) A photograph;

   (g) A comic strip;

   (h) A song;

   (i) A symphony?

   If you have access to suitable facilities, produce a rough version of each of your answers.
3. A commonly used means of multimedia delivery is the *information kiosk*, a computer system with limited functionality provided for the public to learn about, for example, the exhibits in a museum, or tourist facilities in a city. Discuss whether information kiosks are a qualitatively different form of delivery from those described in this chapter.
4. Choose a Web site and a multimedia CD-ROM available to you. For each of them, identify at least five (and preferably ten) ways in which you think it could be improved. State the criteria you use to judge them by.
5. Look closely at the user interface of any application of your choice, and identify ten features of that interface which you

personally do not like. Explain the reasons for your choices, and suggest possible improvements.

6. Choose a straightforward subject, such as a guide to guest accommodation in your neighbourhood. Design a simple one-page layout of graphical material and text, which optimises intelligibility and access to information for potential users of all levels of ability.

7. We stated on page 13 that 'ultimate control over "the content and flow of information" remains with the producer not the user'. Ignoring feasibility, discuss whether it would be desirable for true control to be transferred to the user.

8. If you are working within a group of people, have each member of the group devise their own set of six filtering labels, which they would choose to apply to a Web site of their making for the guidance of users, and then compare and discuss results. If you are in a position to do so, extend this exercise to include other groups in other countries.

9. A committed vegetarian couple wish to prevent their children from being exposed to images of meat on the World Wide Web. What difficulties would they face in using PICS ratings for this purpose?

2 Enabling Technologies

The production and consumption of digital multimedia as we know it depends on the ability of digital computers to perform operations at high-speed. In order to benefit from this ability, media data must be in digital form. That is, the rich variety of sensory inputs that make up images, text, moving pictures, and sound, must be reduced to patterns of binary digits inside a computer. Once this remarkable transformation has been effected, programs can be used to change, combine, store and display media of all types. Furthermore, the same data can be transmitted over networks, to be distributed anywhere in the world, or conveyed to remote destinations on removable storage such as CD-ROMs or DVDs.

Although a digital representation of data is fundamental, general purpose computers are not the only devices that can manipulate digital data. Digital video can be played from DVD by special players, digital TV only requires a simple set-top box, and digital audio can be played from CD by any CD player. At present, only a programmable computer can add full interactivity to multimedia, but it seems almost inevitable that, as the technology matures, consumer devices that function as multimedia players will be brought to market. It is to be expected that such devices will be significantly cheaper than personal computers, and that they may bring about the transformation of digital multimedia into a true form of mass communication.

Digital Representations

Computers are built out of devices that can only be in one of two states. Physically, this means that certain points in the devices' circuits are only stable at one of two well-defined voltages. In more abstract terms, we can say that these devices store and operate on *bits*, units of data that can only have one of two values. We might call these values 0V and 3.5V, or on and off, true and false, yin and yang, but conventionally we denote them by 0 and 1. Bits are usually grouped into larger units such as *bytes*, which consist of an ordered sequence of eight bits, or *words*, whose length depends on the particular model of computer, but is often four bytes, that is, thirty two bits. Adopting the convention that a bit is either 0 or 1 suggests an interpretation of these larger groups: they can be read as numbers to base 2, whose digits are the component bits. Thus, the byte containing the eight bits 0, 1, 1, 0, 0, 0, 0, and 1 can be read as the binary number 01100001, which is 97 in decimal. Not only can we interpret bytes and words in this way, we can also build electronic devices that perform the basic arithmetic operations of addition, subtraction, and (with a bit more effort), multiplication and division, and produce an answer in the same format. Hence the once popular notion that computers are just giant high-speed calculating machines.

There is, however, nothing intrinsically numerical about bits and bytes. It is only the way that we choose to interpret them, and the operations that we build into our computers, that make these bits and bytes into numbers. We can choose to interpret patterns of bits in different ways, and this is how the data belonging to different media can be represented digitally. The pattern 01100001 might denote a particular shade of grey occurring in an image, if we wrote software and built display hardware to interpret it in that way.

It is easiest to see what is going on if we describe the interpretation of bit patterns in terms of the interpretation of numbers, since we know how numbers and bit patterns can be associated, and numbers are easier to write down, and have familiar properties. Thus, we represent characters of text by associating a unique number with each letter, digit or other sign we need, in the manner of a code. The widely used ASCII character set, for example, is an association between characters and numbers; it assigns the value 97 (i.e. the bit pattern 01100001) to the lower case letter a, 98 to b, and so on, for 96 different printable characters. For this association to mean anything, we must arrange that hardware — such as keyboards and printers — and software behave in a way that is consistent with it. For example, when you press the key labelled A on your keyboard without holding down the shift key, the number 97 (or rather a sequence of electrical pulses that can be interpreted as the corresponding pattern of bits) is sent to your computer.

As we will see in later chapters, a similar association can be made between numbers and quantities such as the brightness of an image at a point, or the instantaneous amplitude of a sound wave. Although the association is essentially arbitrary, there will normally be some structure to the way numbers are assigned to the quantities they represent: e.g. numbers denoting high brightness have greater values than those denoting low brightness. Such properties tend to fall out naturally from the nature of the quantities represented.

⇨ Ada Augusta, Countess of Lovelace, appears to have understood the nature of digital representations in 1844, when she wrote:

> "[Babbage's Analytical Engine] can arrange and combine its numerical quantities exactly as if they were letters or any other general symbols; in fact it might bring out its result in algebraical notation *were provisions made accordingly.*" [Our italics.]

Bits are arranged into bytes, and in the memory of a computer, bytes are arranged in a linear sequence so that each byte[1] can be identified by its position in the sequence, which we usually call its address. Addresses behave like numbers, so they can be represented by bit patterns and stored and manipulated like other quantities. This makes it possible to organize collections of bytes into *data structures.* For example, a black and white image is often represented by the values corresponding to the brightness at each point on a fine rectangular grid. We can store these values in a sequence of bytes, and then use the address of the first byte to access the image data; a simple computation allows us to work out the address of the byte corresponding to any grid point, and

[1] Some computers use a different addressable unit, but the principle is the same.

access the stored value. If we need a sequence of such images, representing successive frames in an animation, say, we can store with each image the address of the next and previous ones, so that we can easily traverse the sequence in either direction.

The most important interpretation of bit patterns in a computer is only incidentally related to multimedia: bit patterns can represent instructions that cause the processor to carry out operations on values stored in memory. Again, this interpretation relies on the fact that the hardware is constructed so that the intended effect is achieved when the instruction is presented to the processor. Because instructions are bit patterns, sequences of instructions — *programs* — can be stored in memory and executed. This is the defining characteristic of a computer: it is a *stored program* machine. This is what makes it possible for the same machine to be used for many diverse tasks, from the computation of tax allowances to the precise editing of video footage.

To reiterate a point we made in Chapter 1: the common representation as collections of bits means that data belonging to all media can be manipulated by programs, both individually and in combination. At present, there are established ways of representing text, images, sound, video and animation in bits; we can be fully confident that any media developed in the future will also be representable digitally.

Digitization

Multimedia data sometimes originates in digital form, as it does when images are made using a painting program, but it is frequently necessary to convert from some analogue representation to a digital one, for example, by scanning an image.

A banal illustration of the differences between analogue and digital representations is provided by contemporary time-pieces. Figure 2.1 shows the face of an analogue wristwatch and the display of a digital alarm clock. The minute hand of the watch moves steadily, and can point anywhere on the rim; it is not constrained to point only at the marks indicating exact minutes. This alarm clock can only display whole numbers of minutes, and its displayed value only changes once every sixty seconds. The digital representation is more convenient than the analogue one for precise computation: for example, if we want to know how long it is until half past twelve, we need only subtract the displayed time from 12:30, whereas for the analogue watch we must estimate the difference based on the angular difference between the current position of the hour hand, and a position half way between twelve and one (or else read the watch, and convert the time to its digital form in our heads). This particular analogue representation is more detailed: the alarm clock cannot show how far between 7:47 and 7:48 it is, whereas this can be estimated (to an arbitrary level of precision) from a close examination of the position of the watch's minute hand.

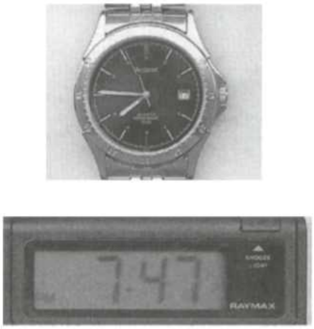


Figure 2.1
Analogue and digital representations

⇨ If you are feeling contrary, you will point out that the watch's hands do not really move continuously — there are gears and stepping motors and things inside the watch that make it actually move in a series of small jerks. (Nowadays, new watches are almost all digital inside, and so, even where the display is analogue, the hands can only move in discrete steps, as is evident if you watch the second hand. Even older, spring-powered, mechanisms do not move truly continuously, though, because of the finite size of the moving parts.) There is no such thing as continuous physical movement, if only because of the quantum nature of electrons and photons; the infinitesimally small is a mathematical abstraction. However, as in the case of the watch, a continuous (analogue) model of behaviour is often more tractable than a digital one, and provides a more accurate description of what is going on than any manageable discrete (digital) model could do.

In multimedia, we encounter values that change continuously in several ways. For example, the amplitude of a sound wave varies continuously over time, as does the amplitude of an electrical signal produced by a microphone in response to a sound wave. The brightness of any point on a black and white photograph could in principle have any value, though in practice this will be restricted by the physical constraints of the process of photography. As you see, we may be measuring different quantities, and they may be varying either over time or over space (or perhaps, as in the case of moving pictures, both). For this general discussion, we will follow tradition, and refer to the value we are measuring, whatever it may be, as a *signal*, not usually distinguishing between time-varying and space-varying signals.

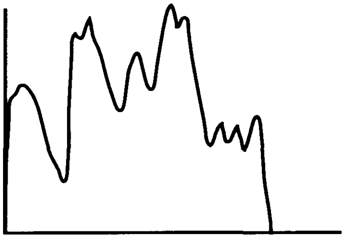


Figure 2.2
An analogue signal

You will observe that, when we have a continuously varying signal, such as the one shown in Figure 2.2, both the value we measure, and the intervals at which we can measure it, can vary infinitesimally. In contrast, if we were to convert it to a digital signal, we would have to restrict both of these to a set of discrete values. That is, *digitization*

— which is what we call the process of converting a signal from analogue to digital form — consists of two steps: *sampling*, when we measure the signal's value at discrete intervals, and *quantization*, when we restrict the value to a fixed set of levels. Sampling and quantization can be carried out in either order; Figure 2.3 shows a signal being first sampled and then quantized. These processes are normally carried out by special hardware devices, generically referred to as *analogue to digital converters (ADCs)*, whose internal workings we will not examine. We will only consider the (almost invariable) case where the interval between successive samples is fixed; the number of samples in a fixed amount of time or space is known as the *sampling rate*. Similarly, we will generally assume that the levels to which a signal is quantized — the *quantization levels* — are equally spaced, although, in this case, other arrangements are sometimes used.

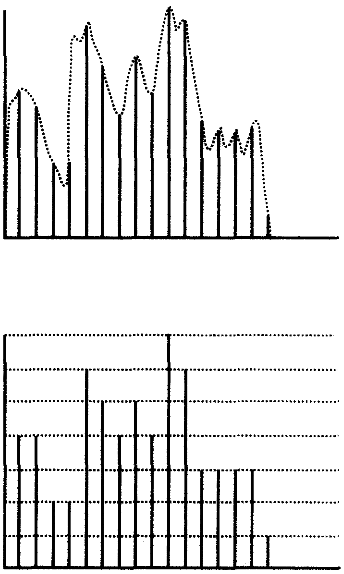


Figure 2.3
Sampling and quantization

One of the great advantages that digital representations have over analogue ones stems from the fact that only certain signal values — those at the quantization levels — are valid. If a signal is transmitted over a wire or stored on some physical medium such as magnetic tape, inevitably some random noise is introduced, either because of interference from stray magnetic fields, or simply because of the unavoidable fluctuations in thermal energy of the transmission medium. This noise will cause the signal value to be changed. If the signal is an analogue one, these changes will be more or less undetectable — *any* analogue value is legitimate, and so a signal polluted by noise cannot be distinguished from a clean one. If the signal is a digital one, though, any minor fluctuations caused by noise will usually transform a legal value into an illegal one, between quantization levels. It is then a simple matter to restore the original signal by quantizing again. Only if the noise is sufficiently bad to alter the signal to a different level will an error in the transmission occur. Even then, because digital signals can be described numerically, schemes to detect and correct errors on the basis of arithmetic properties of groups of bits can be devised and implemented. Digital signals are therefore much more robust than analogue ones, and do not suffer degradation when they are copied, or transmitted over noisy media.

However, looking at Figure 2.3, it is evident that some information has been lost during the digitization process. How can we claim that the digitized result is in any sense an accurate representation of the original analogue signal? The only meaningful measure of accuracy must be how closely the original can be reconstructed. In order to reconstruct an analogue signal from a set of samples, what we need to do, informally speaking, is decide what to put in the gaps between the samples. We can describe the reconstruction process precisely in mathematical terms, and that description provides an exact specification of the theoretically best way to generate the required signal. In practice, simpler methods that can easily be implemented in fast hardware are used.

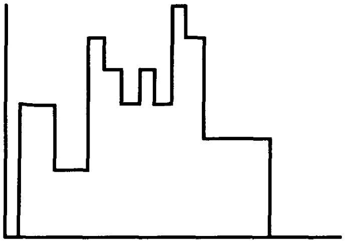


Figure 2.4
Sample and hold reconstruction

One possibility is to 'sample and hold': that is, the value of a sample is used for the entire extent between it and the following sample. As Figure 2.4 shows, this produces a signal with abrupt transitions, which is not really a very good approximation to the original. However, when such a signal is passed to an output device, such as a CRT display or a loudspeaker, for display or playback, the lags and imperfections inherent in the physical device will cause these discontinuities to be smoothed out, and the result actually approximates the theoretical optimum quite well. (However, in the future, improvements in the technology for the playback of sound and picture will demand matching improvements in the signal.)

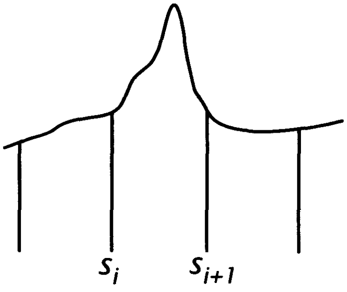


Figure 2.5
Under-sampling

Clearly, though, if the original samples were too far apart, *any* reconstruction is going to be inadequate, because there may be details in the analogue signal that, as it were, slipped between samples. Figure 2.5 shows an example: the values of the consecutive samples taken at s_i and s_{i+1} are identical, and there cannot possibly be any way of inferring from them the presence of the spike in between those two points — the signal could as easily have dipped down or stayed at the same level. The effects of such *under-sampling* on the way in which the reconstructed signal will be perceived depend on what the signal represents — sound, image, and so on — and whether it is time-varying or space-varying. We will describe specific instances later. Suffice it to say, for now, that they are manifested as distortions and artefacts which are always undesirable.

It is easy enough to see that if the sampling rate is too low some detail will be lost in the sampling. It is less easy to see whether there is ever any rate at which we can be sure samples are close enough together to allow the signal to be accurately reconstructed, and if there is, how close is close enough. To get a better understanding of these matters, we need to consider an alternative way of representing a signal. Later, this will also help us to understand some related aspects of sound and image processing.

You are probably familiar with the idea that a musical note played on an instrument consists of waveforms of several different frequencies added together. There is the fundamental, which is the pitch associated with the note, but depending on the instrument, different numbers of harmonics, or overtones, are also present, and these give the note its distinctive timbre. The fundamental and each harmonic are pure tones — sine waves of a single frequency. Any periodic waveform can be decomposed into a collection of different frequency components, each a pure sine wave, in a similar way; in fact, with a little mathematical gymnastics, assuming you don't mind infinite frequencies, any waveform can be decomposed into its frequency components. We are using the word 'frequency' in a generalized sense, to go with our generalized concept of a signal. Normally, we think of frequency only in the context of time-varying signals, such as sound, radio, or light waves, when it is the number of times a periodic signal repeats during a unit of time. When we consider signals that vary periodically in space, such as the one shown in Figure 2.6, it makes equal sense to consider its frequency as the number of times it repeats over a unit of distance, and we shall often do so. Hence, in general discussions, frequencies, like signals, may be either temporal or spatial.[2]

Figure 2.6
A periodic fluctuation of brightness

Figure 2.7 shows one of the classic examples of how pure sine waves at different frequencies combine to produce more complex waveforms. Starting with a pure sine wave of frequency f, we successively add components to it with frequencies of $3f$, $5f$, $7f$, and so on, whose amplitudes are one third, one fifth, one seventh,... the amplitude of the original signal. As you can see, as we add more 'harmonics', the signal begins to look more and more like a square wave; the more frequency components we add, the better the approximation.

Although it may not have the same immediate perspicuity, we could use the frequencies and amplitudes of its components to represent our signal. The collection of frequencies and amplitudes is the signal's representation in the *frequency domain*.[3] It can be computed using a mathematical operation known as the *Fourier Transform*. The result of applying the Fourier Transform to a signal can be displayed, like the original signal itself, in the form of a graph, where the horizontal axis represents frequencies and the vertical axis amplitude. A typical signal's frequency spectrum, as this representation is called, will consist of a set of spikes at different frequencies, corresponding to the different components.

[2] You will probably realize that a spatially varying signal may vary in two dimensions, not just one. We will consider this complication in Chapter 5, but it can safely be ignored for now.

[3] Strictly speaking, we also need to know the phase of the components, since it is sometimes necessary to displace them relative to each other, but we will ignore this complication, too.

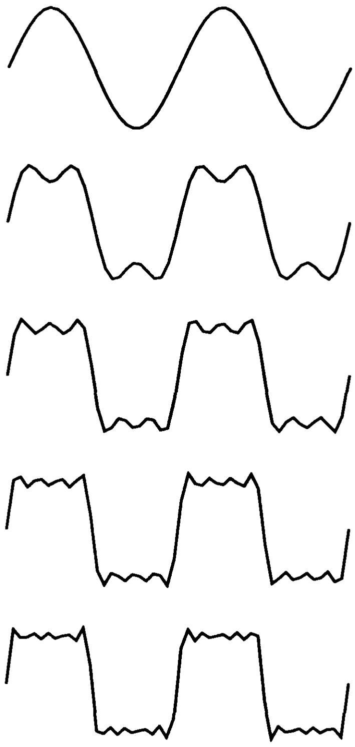


Figure 2.7
Frequency components of a square wave

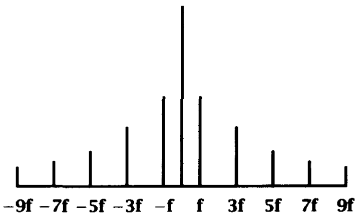


Figure 2.8
Square wave transformed into the frequency domain

Figure 2.8 shows our square wave in the frequency domain. You will notice that it includes negative frequencies. There is no need to worry about this: negative frequencies are a notational convenience that allow us to deal with phase shifts; where none is present, as here, the representation will be symmetrical, with the negative frequencies matching positive ones. There is also a spike at a frequency of zero, which is called the *DC component*, from the long-standing use of frequency domain representations in electrical engineering. You can think of it as the average value of the signal.

The square wave example demonstrates a phenomenon that can be shown to be generally true: *higher frequency components are associated with abrupt transitions.* As we add higher frequencies, the leading and falling edges of the waveform become more nearly vertical. Looking at this from the other end, as we omit high frequency components, such abrupt changes get smoothed out. Hence, operations such as sharpening or smoothing an image can be described and implemented in terms of *filters* that remove certain frequencies — this is of fundamental importance in the processing of graphics, as we will see in Chapter 5.

⇨ The *Inverse Fourier Transform*, as its name suggests, performs the opposite operation to the Fourier Transform, taking a signal from the frequency domain to the time domain.

Returning to the question of whether there is a sampling rate that guarantees accurate reconstruction of a signal, we can now give a precise answer. The *Sampling Theorem* states that, if the highest frequency component of a signal is at f_h, the signal can be properly reconstructed if it has been sampled at a frequency greater than $2f_h$. This limiting value is known as the *Nyquist rate.*

⇨ Some authors, especially in the field of audio, use 'Nyquist rate' to denote the highest frequency component that can be accurately reproduced. That is, if a signal is sampled at f_s, their Nyquist rate is $f_s/2$. The fact that the term is used with both meanings is unfortunate, but any ambiguity is usually easily resolved by context. We will always use the term in the sense defined in the main text.

The proof of the Sampling Theorem is technical, but the essence of the underlying effect can be illustrated simply. Suppose we have a circular disk, with a single radial line marked on it, which is spinning in a clockwise direction at a rate of n rotations per second, and suppose that we 'sample' this rotation as a movie camera would, by taking snapshots of the disk at equal time intervals. Figure 2.9 shows the snapshots we would obtain by sampling $4n$ times a second. Looking at this sequence (ignoring, for now, the dashed boxes) it is clear that the disk is rotating clockwise, and you can imagine that if these images were successive frames of a film, we would see the disk rotating in its proper direction when the film was projected. Considered as a periodic signal, the rotating disk has a frequency of n and we have sampled at $4n$, comfortably above the Nyquist rate. Consider now what happens if we sample at a rate of $4n/3$; the samples we obtain are the ones identified by dashed boxes. Looking at the sequence comprising only those samples, it appears from the successive positions of the line more as if the disk is rotating anti-clockwise at a rate of $n/3$. (The phenomenon may be familiar to you from Western movies, in which stagecoach wheels frequently appear to rotate backwards, because the rate at which the film frames were shot is less than the Nyquist rate relative to the actual rotational speed of the wheels.) Note that we must go beyond the Nyquist rate: if we sample our disk at a rate of exactly $2n$, we get samples in which the line alternates between the 12 o'clock and 6 o'clock positions, so that it is impossible to determine whether the disk is rotating clockwise or anti-clockwise.

> The Sampling Theorem only holds true if the signal is reconstructed from its samples in a particular way — technically, it is necessary to perform the operation in the time domain in such a way that it is equivalent to multiplying the sampled signal by a perfect pulse function in the frequency domain, that is, a function whose value is 1 between a specific pair of frequency values, and 0 everywhere else. By manipulating Fourier Transforms and their inverses, it is possible to arrive at a definition of a function that must be applied to the samples in the time domain to achieve this effect. Unfortunately, this function has non-zero values at points arbitrarily far from the origin. It is therefore impossible to realize the reconstruction operation in this way. In reality, we can only approximate it, with functions that are finitely wide. The practical effect is that it is necessary to sample at a rate still higher than the Nyquist rate to achieve perfect reconstruction.

In general, if we *undersample* a signal — sample it at less than the Nyquist rate — some frequency components in the original will get transformed into other frequencies when the signal is reconstructed, just as our rotating disk's frequency was transformed when we undersampled it. This phenomenon is known as *aliasing*, and is perceived in different ways in different media. With

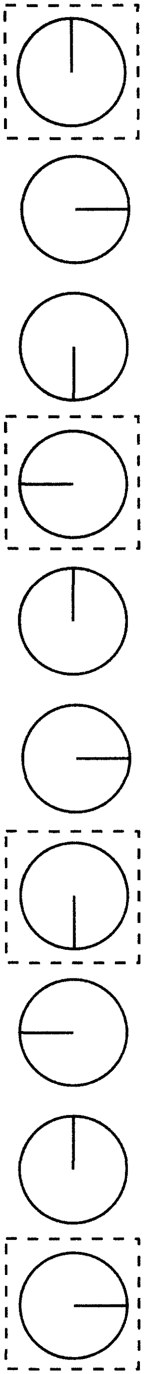


Figure 2.9
Sampling and undersampling

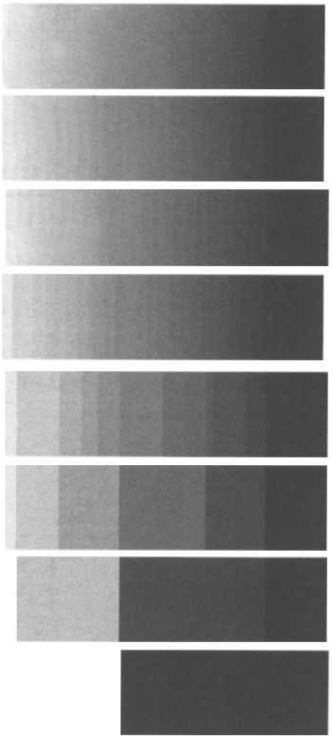


Figure 2.10
256, 128,..., 2 grey levels

sound, it is heard as distortion; in images, it is usually seen in the form of jagged edges, or, where the image contains fine repeating details, Moiré patterns (see Plate 2); in moving pictures, temporal undersampling leads to jerkiness of motion, as well as phenomena similar to the retrograde disk just described.

The effects of an insufficient number of quantization levels are generally easier to grasp intuitively than those of inadequate sampling rates. If we can only represent a limited number of different values, we will be unable to make fine distinctions among those that fall between. In images, it is as if we were forced to make do with only a few different colours, and so had to use, say, crimson for every shade of red we needed. The difference between scarlet and carmine would be lost, and any boundaries between areas of those colours would be elided. The effect on black and white images can be seen clearly in Figure 2.10 which shows a gradient swatch using 256, 128, 64, 32, 16, 8, 4, and 2 different grey levels. The original gradient varies linearly from pure white to pure black, and you can see how, as we reduce the number of different greys, values band together as they are quantized increasingly coarsely. In a less regular image, the effect manifests itself as posterization, more formally known as brightness contouring, where coloured areas coalesce, somewhat like a cheaply printed poster.

The numbers of grey levels in our example were not chosen at random. The most common reason for limiting the number of quantization levels (in any type of signal) is to reduce the amount of memory occupied by the digitized data by restricting the number of bits used to hold each sample value. Here we have used 8, 7, 6, 5, 4, 3, and 2 bits, and finally a single bit. As you can see, although the effect is visible with 128 greys, it only becomes intrusive when the number of levels falls to 32, after which deterioration is rapid.

When sound is quantized to too few amplitude levels, the result is perceived as a form of distortion, sometimes referred to as *quantization noise*, because its worst manifestation is a coarse hiss. It also leads to the loss of quiet passages, and a general fuzziness in sound (rather like a mobile phone in an area of low signal strength). Quantization noise is clearly discernible when sound is sampled to 8 bits (256 levels), but not (except in the opinion of some audiophiles) at the 16 bits (65536 levels) used for audio CDs.

⇨ If you have ever used a drum machine or sequencer for making music, you will have met another form of quantization: these devices quantize notes in time, so that they fall on exact beats. A poor drummer can be brought into a tight rhythm by quantizing to sixteenth notes, for example. However, if you quantize to eighth notes, any funky syncopated rhythms will be lost — there are too few quantization levels. (On the other hand, if your drummer is really egregious, and you quantize to sixteenth notes when they are playing in straight eights, you may let in some of their mistakes.)

Hardware Requirements

If representing media digitally is conceptually so simple, and if digitization is so well understood mathematically, why is it that digital multimedia is such a relatively new development in computing? What we have so far failed to address is the resource requirements. As subsequent chapters will show, sound, video and most images require large amounts of disk space and memory. Time-based media also have to be captured and played back in real time, so the very large quantities of data to be handled imply a requirement for fast processors and data buses capable of high speed transfers. Accessing media data over a network demands high bandwidth. It is only recently that fast enough computers with sufficient memory and large high-speed disks have become available. At the time of writing, in 1999, domestic network connections are still not fast enough to support multimedia at anything but the minimum acceptable quality. As new access technologies boost Internet bandwidth, a further explosion in the use of multimedia is widely anticipated.

When we look at the hardware requirements of multimedia, there are two distinct cases to consider: requirements for multimedia consumption and requirements for production. Realistically, in the case of consumption, it is more sensible to consider the capabilities of typical domestic systems, and the limitations which those impose, than some putative set of requirements for ideal playback.

A series of specifications produced by a consortium of PC manufacturers has defined the *multimedia PC (MPC)*. The latest of these defines the Level 3 MPC as having a 75MHz Pentium processor,[4] with 8 Mbytes of RAM, running Windows 3.11 (or a binary compatible OS), with at least 500Mbytes of hard disk, a 4× CD-ROM drive, CD quality audio output, and the capability to play quarter-frame video

[4] Or a lesser processor, provided additional hardware support for video playback is provided.

at full frame rate in 15-bit colour. This probably sounds like a bit of a joke, but you should be aware that at least some owners of machines satisfying the MPC specification probably believe that their hardware is capable of 'doing multimedia'.[5] More realistically, a 'revision C' iMac, running MacOS 8 on a 333MHz PowerPC 750 processor, with 32 Mbytes of RAM, 6 Gbytes hard disk, a 24× CD-ROM drive, a built-in V90 modem, built-in stereo speakers, a powerful graphics accelerator card and a multiple scan monitor capable of 24-bit colour (or an equivalent Windows 98 PC system based on a Pentium II or III processor) represents the sort of hardware available to digital multimedia consumers at the end of the 1990s. Even these relatively powerful machines cannot play back full-frame, full-motion video at VHS quality without additional hardware.

While one can be confident that the best available desktop consumer machines will soon be capable of flawless video playback, at the same time a new class of less powerful devices is coming into use. Hand-held 'personal digital assistants' (PDAs) are evolving from simple digital address books into capable computational systems. Their small size makes it difficult to achieve the same performance as desktop machines, and imposes limitations on the amount of storage that can be accommodated and the size of the display. The communication facilities of these devices are often based on the cellular telephone network, which provides less bandwidth than the fixed network. It may seem foolish to even consider using PDAs for multimedia, yet it seems inevitable that they will increasingly be used for Internet access, including Web browsing. This presents a challenge to Web designers to produce pages that are accessible both to powerful desktop machines and to PDAs. At the system level, it means that multimedia architectures must be scaleable, allowing different versions of a production to be made available transparently to devices with different capabilities. We will return to this question in Chapter 15.

Although the hardware requirements for high quality multimedia production are more demanding than those for its consumption, a great deal of multimedia is produced on high-end PCs running Windows NT and on Macintosh computers. These systems are often augmented with special-purpose graphics acceleration hardware and input devices for video and audio. For example, at the time the iMac described earlier was a typical consumer machine, a G3 Power Macintosh with a 450 MHz version of the same processor fitted

[5] In fact, the label 'multimedia PC' can be attached to a machine satisfying the Level 2 or even Level 1 specifications — they will have a CD-ROM drive and a sound card.

with 1 Mbyte of high-speed cache memory, 128 or 256 Mbytes of main memory, and a built-in 128-bit graphics accelerator board with 16Mbytes of video memory, would be typical of the better desktop machines used for multimedia production. This would probably be fitted with additional high-speed disks, a video capture card, and other specialized peripherals, as described below. More modest equipment can also be used to produce multimedia — many Web pages are produced on ordinary desktop PCs — but the range of material that can be created and manipulated on such machines is limited. At the opposite extreme, high-powered workstations, such as those produced by SGI, are also used, especially for computationally intensive tasks, such as rendering 3-D animation and video effects, but their use is more common in the film and television industries. The incorporation of special-purpose array processing instructions into both Pentium and PowerPC processors (MMX and AltiVec, respectively) is blurring the distinction between high-end personal computers and workstations.

Raw processing power, high-speed data buses, large main memories, and powerful graphics boards are a necessity for producing multimedia content; fast high-capacity secondary storage is equally necessary. We have already noted that sound, video and images require large amounts of storage. During their preparation, they often require substantially more storage than the finished product will need. For example, images may be composed in layers (see Chapter 5), and 'flattened', by combining the separate layers, for delivery. Each layer occupies roughly the same amount of space as the final image, and designers often use up to a hundred layers. High quality audio and, above all, video source material can occupy a very large amount of storage indeed. The degree of compression required for final delivery, especially of video or over a network, usually entails a loss of quality that cannot be tolerated during production, so less compressed, and therefore much larger, files will be in use throughout the production process, with high compression only being applied at the final stage.

Manufacturers are regularly increasing the capacity of hard disks, and several can be used together if necessary. The speed of data transfer to and from large disks is a more serious limiting factor, particularly for digital video, where transfer rates in excess of 5 Mbytes per second — preferably well in excess, up to 30 Mbytes per second — are desirable during capture. The IDE and SCSI-1 disk drives normally fitted to desktop machines cannot cope with such rates. However, newer revisions of the SCSI standard provide much higher data rates: fast and wide SCSI-2 supports 40 Mbytes per second, while Ultra SCSI-2 can support 80 Mbytes per second; development of the SCSI interface standard is continuing, with a specification of a 160 Mbytes per second version already drafted (in 1999). An alternative standard for connecting mass storage devices is *FireWire*, or IEEE 1394 as it is more formally known, which not only provides transfer rates up to 50 Mbytes per second, but is also used as the standard interface for connecting consumer electronic devices, especially DV cameras (see Chapter 10) and digital stills cameras to computers.

⇨ For video capture it is essential that the data transfer to disk be maintained at a constant rate, or frames may be lost. This is only feasible if the free areas on the disk are contiguous. If a disk becomes fragmented, so that free areas are scattered over its surface in small pieces, the data transfer will be intermittently interrupted, as the disk seeks to a new position. It is important, therefore, that disks to be used for video capture be periodically de-fragmented, using a disk optimizing utility, or simply completely erased before a clip is captured, if this is feasible.

When peripherals are connected via one of the fast SCSI or FireWire buses, the limiting factor becomes the speed at which data can be moved off or onto the disks. With current disk speeds, a single, modestly priced, high-capacity disk connected via FireWire or Fast SCSI 2 to a dedicated computer will provide adequate performance for multimedia production. If, however, it is desired to share data between workstations, using a central server and a local area network, higher performance will be required. Fibre channel or Gigabit Ethernet can provide the transfer speeds. Relatively expensive high-speed disks can be used for storage, but a popular alternative is a *RAID array*. RAID stands for Redundant Array of Ine~~xpensive D~~isks, and a RAID array is indeed put together out of cheaper, relatively slow, disks: improved performance is achieved by reading and writing from and to them in parallel.

More precisely, RAID specifies eight different levels, which offer different degrees of performance and fault-tolerance. The emphasis of most of these levels is on the latter, which explains the 'Redundant' in the expansion of RAID. However, the lowest level, Level 0 RAID, specifies a 'data striping' technique, whereby a data block being written to the array is split into segments which are written to separate disks. The write operations can be overlapped, so that data can be written to the drive buffer faster than it can be physically transferred to any single drive in the array. Read operations can be similarly overlapped. The array looks like a single high-capacity fast drive. RAID 0 offers no protection against disk failure — if one disk fails, the whole array fails. Higher RAID levels are designed to protect against failures, by incorporating redundancy, with more or less sophistication, into the data storage scheme. Level 1 is the least sophisticated: it mirrors disks, which, instead of improving performance, decreases it. Higher levels use more sophisticated schemes, but a popular option is to combine RAID levels 0 and 1, thereby gaining the performance of data striping and the protection of disk mirroring.

⇨ Marketing literature sometimes uses the term ‘AV disk’, which is supposed to indicate disks that are particularly suitable for use with audio and video. Originally, this indicated that the disk did not perform periodic recalibration to compensate for thermal expansion. Such calibration operations interfere with data transfers, and since video and audio require transfer at a consistent sustained rate, disks that performed these operations would be unsuitable for use with multimedia data. Some modern disks do not perform recalibration in the same way, and the term ‘AV’ may be loosely used simply to indicate ‘fast’ — but almost all modern disks are, in themselves, fast enough anyway, except for use in servers.

In addition to large fast disks, multimedia production requires other, more specialized, peripherals. A graphics tablet with a pressure-sensitive pen is necessary for all kinds of graphical work (with the possible exception of some image correction tasks) — you cannot properly draw or paint with a mouse. A large (often 20 inch) high-resolution monitor capable of accurately reproducing millions of colours is also desirable (although it should always be remembered that the end user will probably have a smaller, and possibly lower quality, display). It is common practice to attach two monitors to a computer for preparing images, one to display the image, the other, which may be smaller, to hold the plethora of tool bars and palettes that make up the user interface of most graphics programs. High resolution scanners are commonly used for capturing images from photographs and printed media. Digital cameras are also becoming popular, since they allow photographs to be downloaded directly to disk, without the expense, time, or specialist laboratories required to process film. However, only the most expensive digital cameras offer anything like the picture quality of conventional 35mm SLR cameras. A video camera and sound recording equipment are also needed for recording these media. Nowadays, this equipment may itself be digital and produce digital output that can be sent directly to the computer, over a FireWire connection, for example. Video and audio capture from conventional analogue equipment requires analogue to digital converters and compression hardware, which will be described in Chapters 10 and 12. The amount and quality of additional studio equipment — microphones, mixers, tripods and heads, lighting, and so on — required will depend on how ambitious and professional a multimedia producer is. Generally, the expectations of domestic consumers are high, so for productions intended for the domestic market, production values must be correspondingly high.

Software

We have emphasized that data of all media types are represented digitally as collections of bits, and can therefore be manipulated by programs on a computer. The things that you might naturally want to do to the bits that represent an image are not the same, though, as the things that you might naturally want to do to the bits that represent a piece of text or an audio clip. Different applications have therefore been developed for the different media types: image editing, painting and drawing programs for graphics; editors and layout programs for text; capture, editing and post-production software for video; special packages for motion graphics and animation; recording, editing, and effects programs for sound, and music synthesizers and sequencers, to name just some.

The essence of multimedia is the combination of the different media elements that can be produced by these programs. One way of carrying out this combination would be by writing a program, but, while this provides the maximum flexibility, it requires advanced programming skills (which are not always found in conjunction with the necessary skills for effective presentation of multimedia), while being repetitive and error-prone. *Authoring systems*[6] have been devised to carry out the task within an organizational framework that permits the automation of much of the work that would otherwise require programming. Authoring systems may be based on a layout model with a markup language, or on a timeline, as we will describe in detail in Chapter 13. Some programming, at least in

[6] 'Author' isn't a verb, of course, but the usage is too well established to quibble over.

a scripting language, is usually still required, if interactivity is to be provided.

The upshot of all this is that the making of a multimedia production can involve a host of software tools and skills, some of them special to multimedia, others belonging to existing disciplines. A full mastery of any one of the tools required may take years to acquire — Photoshop ping can be a career in itself, for example — and each demands a different kind of talent. It is a rare artist who can both write good programs and create high quality images, or direct videos, for example — so professional quality multimedia production will generally require a team. Which is not to say that one person working alone cannot produce multimedia, just that, in most cases, the result is unlikely to be of professional quality. This might be perfectly acceptable in some contexts: personal Web pages probably ought to look amateur, in a sense; scientific visualization is not judged by the same criteria as science fantasy creations; diagrams are judged differently from works of art, and corporate presentations have their own aesthetic. At the low end of the multimedia software market there are packages intended to make multimedia production, especially Web page design, accessible to non-specialists. Such packages hide technical details, such as HTML tags and JavaScript code, and usually provide 'wizards' or 'assistants' that guide the production process. Collections of clip art, basic animations and other prefabricated media elements are available, so simple multimedia productions can be produced quickly with the minimum of effort. They all come out looking much the same, of course.

An essential part of our definition of multimedia is the condition that the combined media be sufficiently well integrated to be presented via a unified interface, and manipulated by a single computer program. This is in complete contrast to the situation we have just described with respect to multimedia production, with its multitude of different software tools and team input. The key to integration is a framework that can accommodate a multiplicity of media and present them to the user. Three distinct approaches can be distinguished (although, as we will see, elements of these approaches can be, and are, combined).

The World Wide Web epitomizes one approach: define a format (in this case, a markup language, HTML or XML) that can accommodate different media, and view it using a dedicated browser. Here, all multimedia productions are collections of Web pages, which are basically text, but usually contain embedded graphics and other media elements. Any Web page can be displayed by a sufficiently up-to-date Web browser.

The second approach is to define an architecture, comprising a format that can contain different media types, together with an API (Application Programming Interface) that provides a rich set of functions to manipulate data in that format. The prime example here is QuickTime. A QuickTime movie can contain video, sound, animation, images, virtual reality panoramas, and interactive elements, all synchronized and coordinated. The QuickTime API provides functions for playing and interacting with movies, as well as for editing them. Any program can use the API to provide multimedia functionality; QuickTime movies can be played by the QuickTime player that is distributed with QuickTime, in the same way that a Web page can be displayed in a browser, but they can also be emdedded in word processor documents, or within games, or any special-purpose program that needs to manipulate multimedia as well as performing some other tasks. One can imagine adding code which interprets HTML to any program, of course, but since QuickTime is implemented at the system level and is available to any program, the effort required to add QuickTime support is minimal in comparison.

The third approach is quite the opposite to the previous one: deliver the multimedia production in a 'stand alone' form, that needs no additional software to be used. Director provides a well-known example of this approach: movies may be saved in the form of a 'projector', which can be delivered on a CD-ROM, for example, and will play on any user's machine (of the right architecture) without their needing Director itself, or any browser or other movie-playing program.

Figure 2.11 summarizes these three approaches, and identifies some of the many applications that are available for each part of the multimedia production process. Although we have shown the process as one of production followed by immediate consumption, you should realize that in between these stages a multimedia production may be transmitted over a network, stored in a database, or written to an offline delivery medium. Conversely, although our examples show authoring systems that are used to combine media prior to delivery, the combination may occur 'on the fly' in response to user input. Finally, it should be borne in mind that these three approaches are not necessarily separate or incompatible.

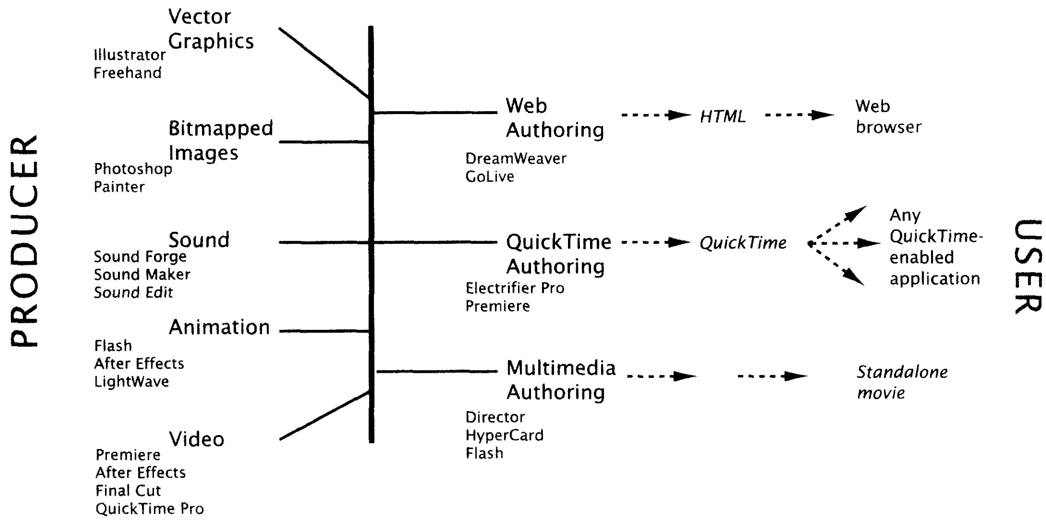


Figure 2.11
Multimedia production and delivery

QuickTime is frequently used purely as a video format with synchronized sound, for example, and so QuickTime movies can often be found embedded in Web pages or in Director movies. Similarly, Director movies can be saved in a form called Shockwave, and embedded in Web pages to be delivered over the Internet and played inside a browser. In order to accommodate these forms of embedded media, Web browsers must be extended with plug-ins, as we will describe in Chapter 13.

Networks

Modern computer systems rarely operate in isolation. Local area networks, usually employing some form of Ethernet technology, are used to connect computers within a small organization, or within a single site belonging to a large one. Local area networks are connected together by routers, bridges and switches to form internets. *The* Internet is a global network of networks, communicating via a standard set of protocols loosely referred to collectively as *TCP/IP*. Most of these networks are operated by commercial Internet Service Providers (ISPs) who lease access to other organizations via dedicated lines and routers, and to individuals via the telephone or cable TV networks. The growth of the Internet since the removal of restrictions on its commercial use in 1991 has been more than adequately described and commented on elsewhere. Although, as we pointed out in Chapter 1, its ubiquity can be exaggerated, in developed countries the Internet is rapidly achieving the status of a public data network, analogous to the public communication network of the telephone system.

Networks, and the Internet in particular, offer valuable opportunities for distributing multimedia, but they also present formidable technical difficulties. In the case of the Internet, an additional factor to be considered is the presence of a largely technologically unsophisticated community of users, who have high expectations of technical quality based on their experience of television, videos, photography and print media. Where multimedia content delivered over the Internet attempts to compete with traditional media, as it is beginning to in news or sports coverage, it must provide some additional value (usually in the form of interactivity) to compensate for inferior picture and sound quality. This situation will, however, change as new, high-speed, access technologies come into widespread use.

For the present, most domestic Internet access is provided over a 'dial-up connection', with a modem and the telephone system providing a temporary connection between a user's computer and an ISP's network. The maximum bandwidth available through a modem is 56 kbits per second (kbps).[7] Many new computers are fitted with modems that conform to the V90 standard, which should provide this speed for data flowing 'downstream' from the telephone exchange to the modem, with a maximum of 33.6 kbps in the opposite direction, although factors such as noise and the distance between the modem and the telephone exchange make it rare for these figures to be achieved in practice. Speeds between 34 kbps and 48 kbps downstream are more realistic. Many older modems are still in use, providing speeds down to 14.4 kbps. For multimedia delivery it is normal to assume that it is unreasonable to expect to be able to use anything slower than 14.4 kbps, although it is considerate to provide some alternative to images, sound and video — usually a text substitute — for users with slow connections.

There is good reason to suppose that 56 kbps is the absolute maximum bandwidth achievable using analogue connections between the modem and the exchange.[8] Various new technologies avoid this analogue connection to make direct use of the digital telephone system's potentially higher bandwidth. *ISDN* (Integrated Services Digital Network) is based on the principle of combining multiple

[7] Note that in this section, speeds are quoted, following normal usage in the networking literature, in *bits* per second, although elsewhere, following mainstream computing usage, we quote file sizes and so on in bytes.

[8] But that's what was said about 33.6 kbps previously.

Table 2.1
Data transfer rates over the Internet

	kbps	6KB text page	100KB image	4MB movie
Slow modem	14.4	3 seconds	56 seconds	37 minutes
Fast modem	56	1 second	14 seconds	9 minutes
Basic rate ISDN	128	< 1 second	6 seconds	4.3 minutes
T1 line	1544	< 1 second	1 second	21 seconds
Cable modem/ ADSL (typical)	6000	< 1 second	< 1 second	5 seconds
T3 line	44736	< 1 second	< 1 second	1 second

digital 'channels', each of which supports 64 kpbs. Its 'basic rate' can use two digital connections simultaneously to carry data at up to 128 kbps.[9] Although ISDN was a promising technology when it was introduced in the mid-1980s, non-technical factors have prevented its widespread deployment, and other technologies have since caught up with or overtaken it. Currently, *ADSL* (Asymmetric Digital Subscriber Line) appears to be the leading new method for access over existing copper telephone wires (albeit with different switching equipment). It provides speeds of up to 6.1 Mbps in the downstream direction, and up to 640 kbps upstream (hence 'assymmetric').[10] ADSL is being installed in homes in some areas of the United States in 1999, and European telecommunications carriers, including British Telecom, have announced their intention to make ADSL available in 2000. Another way of avoiding the restriction of the analogue connection between the home and the telephone exchange is to use the cable television network instead of the telephone network, in areas where cable TV is available. Cable modems can transfer data at rates from 500 kbps to 10 Mbps; devices capable of 30 Mbps will be available by 2000. Cable modems have an additional advantage of being permanently connected.

Commercial Internet users, especially those operating their own Web servers (see below), usually prefer to lease dedicated lines to provide a fixed connection to ISPs. *T1* and *T3* lines provide 1.544 Mbps and 44.736 Mbps, respectively. T1 and T3 lines are also used by small local ISPs to connect to the larger ISPs who are directly connected to the Internet backbone.

⇨ Although 'primary rate' ISDN, which utilizes up to 30 channels (in Europe) or 23 (elsewhere) is available, it is most often used for transferring files directly between, for example, a graphic design studio and a pre-press bureau, rather than for Internet access.

Table 2.1 shows the typical times taken to transfer different media elements over various types of connection, assuming the maximum speed is attained. Although the comparison is striking, it only tells

[9] A third 16kpbs channel is used to carry control signals.

[10] Other DSL variants offer lower but symmetrical rates (for example HDSL offers up to 1.5Mbps in both directions) or even higher assymetrical rates, up to 50 Mbps downstream).

part of the story. The connection between the user and their ISP is not always the factor that limits the speed of data transfer over the Internet. The capacity of the connections between ISPs' networks, and the computing power of the machines from which data is being downloaded may be just as important. It is quite common to find that a file download is not even using all the available speed of a V90 modem. However, anecdotal evidence suggests that both cable modems and ADSL can provide qualitative improvements in download speeds from powerful servers. For example, downloading the 25 Mbyte QuickTime movie of the trailer for *Star Wars: The Phantom Menace* from Apple's QuickTime site is typically reported to take less than three minutes using cable or ADSL — nowhere near the theoretical speed, but considerably better than any conventional modem. However, it is quite easy to see that the advantages of faster access between the user and the ISP can only be maintained if the connections between ISPs are upgraded proportionally as more users acquire faster access, and if data can be delivered by the machines at the far end of the connection at the same rate at which it can be transferred. Since data bus speeds are still considerably higher than network speeds, the latter condition is easily met for a single connection, but as the number of Internet users increases, popular sites will need to service many requests at once.

Local area networks provide much faster data transfer than the Internet. The most common technology for small organizations' LANs is 10 base T Ethernet, which provides 10 Mbps; increasingly, this is being superseded by 100 base T, at 100 Mbps, and, during 1999, the first Gigabit Ethernet equipment began to appear. This bandwidth must be shared between all the transfers taking place at any time, so the effective bandwidth between two computers on a LAN will be lower; it is still sufficiently high for multimedia applications that are presently infeasible over the Internet. In particular, video conferencing and other video applications, which can only be implemented on the Internet at very low quality, are possible using a high-speed LAN.

Clients and Servers

Online distribution of multimedia over LANs or the Internet is almost always based on the client/server model of distributed computation. In this model, programs called *servers* 'listen' on a communication channel for *requests* from other programs, called *clients*, which are generally running on a different machine elsewhere on the network. Whenever a server receives a request, it sends a *response*, which provides some service or data to the client. The requests and responses conform to a *protocol*, a set of rules governing their format and the actions to be taken by a server or client when it receives a request or response.

The most popular form of online multimedia delivery is the World Wide Web, whose implementation is an example of the client/server model. Web servers and clients communicate with each other using the *HyperText Transfer Protocol*, usually abbreviated to *HTTP*. HTTP is a very simple protocol, designed for the fast transmission of hypertext information, which is usually in the form of documents marked up using the HyperText Markup Language, HTML, which will be described in Chapter 8. However, the information in the World Wide Web is more properly described as hypermedia, and may include graphics, sound, video, MIDI and other sorts of data, and even programs.

HTTP provides communication between Web servers and their clients. A client first contacts a server, and then (usually) sends a request for a Web page. The identity of the server and the location of the file containing the Web page's data are normally extracted from a URL, the familiar 'Web address', such as `http://www.wiley.com/compbooks/chapman/index.html`, where `www.wiley.com` is the name of a machine running a Web server, and `compbooks/chapman/index.html` identifies a file in that machine's file system. The server responds by sending the contents of the designated file, if it exists, wrapped up in the form of an HTTP response with some extra information, such as the type of the data (HTML text, GIF graphics, sound, etc.) it is sending. This type information is specified in a format which will be described in the next section.

World Wide Web clients are usually browsers, such as Netscape Navigator or Internet Explorer, which allow us to access Web pages interactively. Despite the foregoing description, Web browsers are usually multi-protocol clients, which can communicate using other protocols, too. Nearly all browsers allow you to download files using the File Transfer Protocol (FTP), for example, although the interface to that protocol is integrated transparently into the Web browsing interface used for HTTP. Modern browsers also support real-time data streaming for audio and video using several special protocols for this purpose.

Web servers often run on dedicated powerful machines, usually under Unix, to enable them to cope with heavy traffic. A common arrangement is for ISPs to provide Web space on one or more of their machines running a server for the benefit of their customers; small companies and individuals wishing to put up personal Web pages can use the ISP's facilities. Larger companies maintain their own sites and servers. It is perfectly possible to run a Web server on a desktop machine, provided only a reasonable number of hits are expected — Watford football club's site uses a server running on an iMac, for example. The machine running the server must be permanently connected to the Internet, though, which will usually entail leasing a line, so it is not a viable option for most people.

The more powerful servers often augment their basic Web page serving function with interfaces to other programs running on the same machine. This allows them to generate Web pages dynamically, incorporating, for example, information retrieved from a database, and to perform computations based on information supplied through a form on a Web page. The Common Gateway Interface (CGI) is a *de facto* standard for such interfaces, but other proprietary mechanisms, such as Microsoft's Active Server Pages, and Apple's WebObjects, are increasingly preferred, largely because of efficiency and their tighter integration with database management systems.

Many local area networks are based on the same TCP/IP protocols that are used on the Internet, and often they use the same high level protocols, so that, for example, multimedia data can be served over a LAN using HTTP. A LAN organized in this way is often called an *intranet*. It allows familiar Internet clients, in particular, Web browsers, to be used with the private network to access information within the organization, but at higher speeds than those that can be attained over the public Internet. Alternatively, proprietary protocols may be used for multimedia applications operating over a LAN. These may provide services that are not available via the Internet, and may be more suitable for private use by a single organization.

MIME Types

The transmission of disparate types of media data across networks connecting heterogeneous computer systems calls for some way of identifying the sort of data contained in a file or data stream. Individual operating systems have their own methods of identifying the type of a file. Most commonly, the extension of a file's name distinguishes the type: a file whose name ends .JPG on a Windows system is assumed to contain a JPEG image, one whose name ends in .MOV, a QuickTime movie, and so on. There is nothing standard about the mapping from content type to extension, though — Unix systems may use different extensions from Windows — and not all systems use extensions for this purpose. The MacOS has its own way of storing file type and creator codes with files, and not all media data is stored in a file. Some other means of identifying the content types is required in a networked environment.

MIME (Multipurpose Internet Mail Extension) is an extension to the Internet mail protocols that supports the inclusion of data other than plain ASCII text in mail messages. Some of its features have been adopted by HTTP, where they provide a simple and convenient way of specifying the type of data included in a server's response to a client's request for a resource. In particular, an HTTP response includes a MIME content type header, which takes the form:

`Content-type:` *type/subtype*

where *type* provides a broad indication of the sort of data, such as text, image, or sound, and *subtype* specifies the precise format, such as HTML, GIF, or AIFF. For example, HTML pages have the MIME content type `text/html`, while GIF images have type `image/gif`.

The available types are `text`, `image`, `audio`, and `video`, which have the obvious meanings; `model` for 3D model data, such as VRML; `message`, which indicates an email message, and `application`, which means binary data, including executable programs, that must be processed in some way: for example, PostScript would usually have the MIME type `application/postscript`, since it must be passed to a PostScript interpreter. (Conceivably, you might send PostScript with type `text/plain`, if you wanted the client to view the PostScript code itself instead of rendering it).[11] The range of subtypes is extensive (none more so than the subtypes of `application`) and supports most multimedia file formats, although some use 'experimental' subtypes, identified by the prefix `x-`, which, while not included in the list of supported MIME types maintained by the Internet Assigned Numbers Authority (IANA), are widely supported by Web browsers. For example, the MIME content type `video/x-msvideo` is recognized by Internet Explorer and Netscape

[11] MIME also allows for arbitrary experimental types, which must begin with the characters `x-`, and for a `multipart` type, which has been used experimentally with HTTP for form data, but is more often used for mail messages with attachments.

Navigator as identifying AVI movies, although this format is not among the video subtypes registered with IANA. QuickTime movies, on the other hand, have MIME content type `video/quicktime`: since QuickTime is registered with IANA, the subtype does not have to begin with `x-`.

We will decribe how MIME types are used by Web servers and browsers in more detail in Chapter 13.

Standards

The International Organization for Standardization (ISO)[12] offers this description of standards:

> "Standards are documented agreements containing technical specifications or other precise criteria to be used consistently as rules, guidelines, or definitions of characteristics, to ensure that materials, products, processes and services are fit for their purpose."

Since standards are agreements, we can assume that things that conform to the same standards are interchangeable. For example, an important set of ISO standards is concerned with screw threads. Any manufacturer's standard screws can be fitted in place of any other manufacturer's standard screws, which means that equipment that is assembled using standard screws is not tied to one screw manufacturer. In a market economy, this is supposed to lead to healthy competition. In practical terms, it means that the bankruptcy of one screw manufacturer does not imply that everything built with their screws becomes irreparable. However, only some standards have any legal status, so conforming to a standard is not something that anybody can usually be compelled to do. Nor does conforming to a standard necessarily mean doing something the best way; it just means doing something the standard way.

In multimedia, standards define interfaces, file formats, markup languages, network protocols, and so on, in precise, and usually formal, terms. The rôle of these standards is broadly analogous to that of the standards for screw threads. If we have a standard file format for image data, for example, then we can incorporate images into a multimedia production without having to worry about which

[12] ISO is not a mistyping of IOS, and it does not stand for International Standards Organization. ISO is not an acronym, but a play on words on the prefix iso- meaning 'the same'.

program was used to prepare them originally. Similarly, standards for all the other media types enable multimedia authoring systems to be constructed independently of the applications used to prepare the individual media elements. The alternative would be for each manufacturer of multimedia software to produce a system that could only use its own formats. Whereas this may be attractive to some manufacturers, such closed systems are unacceptable in a world where several different hardware and software platforms are in use, communicating with each other over networks; open systems that can accommodate data from any platform or source are needed.[13]

Standards are of particular importance in networking, precisely because a modern network should be capable of connecting different makes of computer running different operating systems. It is only by conforming to standards for protocols and for electrical connections agreed to by all the manufacturers that this can be possible. Because modern networks cross national boundaries, and carry data all over the world, international standards are essential. In general, international standards are increasingly seen as being important because they facilitate global trade.

Three organizations are involved in making international standards that are relevant to multimedia: ISO, the International Electrotechnical Commission (IEC), and the ITU (International Telecommunication Union). ISO takes responsibility for formulating standards in all technical fields except electrical and electronic engineering, which are the responsibility of the IEC. Information technology defies categorization, and is dealt with by a joint ISO/IEC technical committee. ISO works through the national standards bodies of its member countries — ANSI (the American National Standards Institute) in the United States, BSI (the British Standards Institution) in the United Kingdom, DIN (Deutsches Institut für Normung) in Germany, and so on — who administer the committees that formulate standards, with ISO itself largely operating as a coordinating agency. The IEC operates in a similar way with the relevant national bodies responsible for electrotechnical standards.

Whereas ISO and the IEC are non-governmental agencies — commercial companies, in fact, albeit of a rather special nature — the ITU is an agency of the United Nations.[14] It has a more regulatory function than the other two international standards bodies. For example, it is the ITU that allocates frequencies to radio services; these allocations must be followed to prevent interference, so they have a mandatory

[13] We use the word 'open' in a general sense, without implying the free availability of source code required by adherents to the Open Software credo.

[14] Although its history goes back much further than the UN's, to 1865.

status according to international treaty. ITU standards covering video formats and telecommunication are those most relevant to multimedia.

Although there is broad agreement on the desirability of standards, the process of agreeing standards through these official bodies is a fraught and often a long drawn out one. Since standards will only be observed if there is a consensus among the concerned parties, a great deal of politics and compromise may be involved in the production of a standard. The major standards bodies require extensive consultative procedures to be followed, designed to ensure that a standard has the broadest possible support, before a draft document can be endorsed as a full standard. These procedures work well for such things as screw threads, where a standard may be expected to remain relevant for very many years after it has been created, but in the rapidly changing environment of computers, networks, and multimedia, standards are often obsolete before they have passed through all the necessary stages and national and international committees. Furthermore, international standards bodies derive some of their income from the sale of standards, and are therefore reluctant to make their documents freely available. This causes some resentment in the computing community, which has become used to free documentation on the World Wide Web — and the high prices charged for some standards put them out of the reach of small software companies. As a result, semi-formal standards and *ad hoc* arrangements play a greater rôle in these areas than they do in more traditional fields of engineering.

Internet standards are a paradigm of this semi-formal standardization. Since the Internet is, by definition, an open network architecture, it relies on standard protocols to enable different networks to be interconnected. The Internet grew out of Arpanet and NSFNET, which had some degree of central administration; the TCP/IP protocols could be imposed in the early days of internetworking and were inherited as the basis of the Internet. Responsibility for the further development of protocols and for administering information required for the protocols to operate now resides with the Internet Architecture Board (IAB) and its subsidiaries, including the Internet Engineering Task Force (IETF), which deals with technical development, and the Internet Assigned Numbers Authority (IANA), which registers MIME types, language codes, and so on. These bodies have no formal standing outside the Internet community, and no statutory powers.[15] Similarly, the

[15] IETF documents, including Internet standards, are almost diffidently called 'Requests for Comments'.

organization responsible for defining World Wide Web standards, the World Wide Web Consortium (W^{3}C), has no official status but its Recommendations are treated as standards. As you would expect, these bodies make use of the Internet as a means of disseminating standards, both after they have been adopted, and during the standardization process, thereby providing an opportunity for a wide audience to comment on drafts and proposals.

The advantage of such an *ad hoc* approach to standards is that it accommodates rapid change. The disadvantage is that manufacturers feel less compunction in ignoring, adapting, or extending standards. There is a fine line between such behaviour and legitimate experimentation aimed at advancing a standard. This is illustrated by the history of HTML, where the two main competing browser companies, Netscape and Microsoft, each implemented their own extensions to HTML 2.0, leading to incompatibilities, but ultimately most of the extensions were incorporated into later versions of the HTML standard.

Sometimes, standards are established without the intervention of standards bodies of any sort. One company's product may come to dominate the market to such an extent that it becomes a standard in all but name. The PostScript page description language and the QuickTime multimedia architecture are examples. In both cases, their perceived technical superiority, possibly reinforced by the marketing muscle of Adobe and Apple, respectively, has caused most rivals to be abandoned, leaving them as standards by default. In some cases, a *de facto* standard of this sort may be more widely used than a competing official standard. For example, at the time of writing, the W^{3}C is developing a standard for vector graphics on the Web, SVG, but, while this has not been implemented, Macromedia's Flash format is supported by the major browsers, and is in wide use as the 'standard' Web vector format.

Further Information

Sampling and quantization are described in most books on image processing or sound, e.g. [Bax94] and [Poh95]. Hardware and software develop at such a rate that it is usually best to consult trade journals and magazines for the latest developments. We return to the topic of multimedia and networks in Chapter 15. [ISO] gives ISO's perspective on standards, while [Bra96] describes the IETF's standardization procedure for the Internet.

Exercises

1. Identify three phenomena in the natural world that exhibit continuous change.
2. The second hand on the watch in Figure 2.1 appears to be broken in one place. Explain why it looks like this. (Hint: The picture was made by scanning an actual wristwatch.) What can you deduce about the way the second hand moves?
3. Is the sampling rate of 44.1 kHz used for audio CDs adequate to reproduce musical sounds accurately? Justify your answer.
4. Suppose a piece of film depicting a moving stagecoach is shot at 24 frames per second, and that the wheels are rotating at such a speed that, when the film is projected at 24 frames per second the wheels appear to move backwards. What would you expect to see if the same film is projected at (a) 12 frames per second; (b) 30 frames per second; (c) 60 frames per second?
5. Digital representations of sounds and images are sometimes accused of being 'limited' (e.g. by the number of quantization levels or the sampling rate), and consequently not to be 'true' representations. Are analogue representations also limited? Explain your answer and give examples.
6. Prove (at least to your own satisfaction) that, if we double the number of bits used to hold a quantized value, then we square the number of quantization levels.
7. Draw up a list of the hardware and software you would recommend for equipping:

   (a) A small company specializing in Web page design.

   (b) A community college or Further Education college laboratory for introductory courses on multimedia.

   (c) A public library's IT facility for community multimedia access.

If possible, talk to people working in the relevant sectors to get an idea of realistic budgets, and try to fit your recommendations within them.

8. Which of the following types of media production are likely to be succesful on a Web site, given current technology, and which may cause problems?

   (a) A live sports broadcast, (i) sound only, and (ii) sound and video.

   (b) The personal home page of a stamp collecting enthusiast.

   (c) A video clip from a pop promo.

   (d) An art gallery's catalogue of its current exhibition, with text and still images.

   (e) An interactive computer game with high quality graphics.

   (f) A display of up-to-the-minute share prices for distribution to market traders.

   (g) A hi-fi live broadcast of a classical music concert.

   Explain each of your answers. For those productions that you do not consider likely to be succesful, what high-end equipment or presently envisaged technological advances would be needed to make them feasible?

9. The ISO standard ISO 216 defines a collection of standard paper sizes, including the A sizes, A0, A1, A2, and so on, of which A4 is the most popular for office use. Paper made to this standard is used almost everywhere in the world except in the United States. What problems does this exception cause for hardware and software manufacturers and users? What factors prevent the adoption of ISO standard paper sizes in the United States?

3 Introduction to Computer Graphics

We use the term *graphics* in a broad sense to refer to the software and hardware technologies used in a computer system to create, modify and display still images stored in a digital form.

Graphics in this sense is of fundamental importance in multimedia, not only because it allows us to generate and display still pictures, but also because it underlies the display of moving pictures and text. Graphics should not, therefore, be considered as just one of the media that make up what we call multimedia, in the sense that sound is, for example. Rather, it is the enabling technology for all the visual elements of multimedia. To understand how it fulfils that rôle, we need to examine how it works in isolation first. In other words, we need to consider the production and display of still images.

Digital images may originate in a number of different ways. They might already exist in some non-digital medium, and be digitized by a scanner, or they might be captured from the external world in digital form by a digital camera or video frame grabber. Other images might be created on a computer system, by an artist, designer or illustrator using a graphics package, or they might be built up by a programmer using a graphics language. Extensive collections of digital images are available on CD-ROM and on commercial and amateur Web sites. Finally, images might be generated by a computer program operating on some data, mapping it to a simple visual representation such as a pie-chart or a more elaborate visualization, such as a simulated picture of the wave structure of electrons in a solid.

There is a long history of images being made and used as art, entertainment, information, inspiration, devotion, titillation, education, amusement, decoration and communication. In a way, this makes still images the easiest of media: we can draw on centuries of experience when creating images, and, at least within a specific cultural context, we can reasonably expect our audience to understand an extensive range of cultural and visual conventions, such as perspective or caricature. This very familiarity can also make images difficult to deal with in a digital system, because it raises expectations of what can be achieved, which the limitations of the systems available to us often frustrate.

Although some painting programs can simulate the effects of real art materials to a remarkable extent, and high quality scanners can capture much of the detail and subtlety of an original, in the final multimedia production your images will almost certainly be displayed on a low resolution monitor which cannot match even the quality of reproduction provided by photographic methods in glossy magazines. Worse, the available range of colours may be limited, and will be reproduced differently on different systems. It is necessary to understand these limitations, so that you can take appropriate steps to ensure that your artwork looks as good as it can do within them.

Graphic elements of multimedia productions will usually be delivered on CD-ROM, DVD or over a network, often the Internet. However, digital images are also widely used for print media. It may be necessary to adapt such images for use in multimedia or, contrariwise, to allow for the printing on paper of images originally made for display on a monitor. To do this effectively, it is necessary to take account of the different characteristics of displays and printers, particularly in respect of colour, a topic we will examine in detail in Chapter 6.

There is no doubt that, ultimately, display technology will improve to a point where much higher image quality is achievable. It will always remain the case, though, that an image works — conveys its meaning, evokes its response — differently when it is in a digital form than it does when it is reproduced in print, or framed and hung on a wall. A notable feature of the World Wide Web's shift from a text-based medium to a heavily graphical medium has been the way traditional graphic designers failed to appreciate this difference, and tried to transplant established idioms from print-based design on to the Web. Often the results were cumbersome and unreadable pages that took many minutes to download, looked terrible on most ordinary monitors, and served only to obstruct users trying to find information. Latterly, the Web has begun to develop its own visual vocabulary, which takes account of its limitations and takes advantage of its particular capabilities.

The use of icons as components of graphical user interfaces provides one example of the way in which images can be put to new use in a computer system. Another example is the way in which visual representations of data can be generated and displayed to help make large quantities of information more readily comprehensible, as we mentioned in Chapter 1. Such visualizations depend on computers to perform the necessary calculations to generate the image from the data; they are therefore designed with the limitations of display technology in mind.

Vector Graphics and Bitmapped Graphics

The display of images is ultimately controlled by some program: a dedicated picture-displaying application, an image editor, or a Web browser, for example. Monitors display pictures as a rectangular array of *pixels* — small, usually square, dots of colour, which merge optically when viewed at a suitable distance to produce the impression of continuous tones. To display an image on the monitor, the program must set each pixel to an appropriate colour or shade of grey, in order that the pattern of pixels on the screen produces the desired image. The low level operations required to set pixel values are usually performed by a graphics library, which communicates with the display hardware, and provides a higher level interface to the application program.

A graphics application program must somehow keep an internal model of the image to be displayed. The process of generating a pattern of pixels from a model is called *rendering*. The graphic model will usually take the form of an explicit data structure that holds a description of the image, but it may be implicit in the sequence of calls to the graphics library which are made as the program executes. Where picture data must be persistent (i.e. must live on after the execution of the program, so that the image can be displayed again at a later time, possibly by a different program), a similar model must be kept in a file. The sequence of events for displaying an image then begins with a program reading an image file, from which it constructs an internal data structure corresponding to the image description in the file; the program then renders the image for display by calling functions from a graphics library, supplying arguments derived from its image model.

⇨ Where images are being generated as visualizations of data, another level of modelling may be present. For example, if a program is generating weather maps from satellite data, then it must have a model of the visual elements of the map — isobars, weather fronts, the underlying geographical features — but it must also maintain a mathematical model of the state of the atmosphere, based on the data, from which the map model is generated.

It is usual to distinguish between two different approaches to graphical modelling: *bitmapped graphics* and *vector graphics*.

In bitmapped graphics, the image is modelled by an array of pixel values. Where it is necessary to emphasize the distinction between these stored values and the physical dots on a display screen we will call them *logical pixels*, and the latter *physical pixels*. In the simplest case, the logical pixels correspond one-to-one to the physical pixels: the model is a map of the displayed image. More generally, the model may be stored at a different resolution from the displayed image, so that some scaling has to be applied to the logical pixels before the image is displayed; the model may also describe a larger image than that to be displayed, so some clipping will have to be applied to extract the desired part for display. Scaling and clipping are the only computations that need to be performed to display a bitmapped image.

In vector graphics, the image is stored as a mathematical description of a collection of individual lines, curves and shapes making up the image. Displaying a vector image requires some computation to be performed in order to interpret the model and generate an array of pixels to be displayed. For example, the model will represent a line by storing its end-points. When the model is rendered for display, the co-ordinates of all the pixels lying on the straight line between those end-points must be computed so the pixels can be set to the appropriate colour. For persistent storage in a disk file, such a model is often implemented as a program in a graphics language, for example, PostScript or PDF.

⇨ Neither of the terms 'vector graphics' or 'bitmapped graphics' is entirely accurate. A more accurate term for what we are calling vector graphics would be 'object-oriented graphics', were it not for the potential confusion caused by the meaning of the term 'object-oriented' in the field of programming languages. Bitmapped graphics does not use *bit*maps (except for purely monochrome images) it uses pixel maps, or *pixmaps* for short, but the term 'pixmapped graphics' has never acquired any widespread currency. You will sometimes see the name 'graphics' reserved for vector graphics, and 'images' used to mean bitmapped images, but this usage can cause confusion with the colloquial meaning of these words. Throughout this book, we will stick to the terms introduced in the preceding paragraphs, and hope you will not be misled.

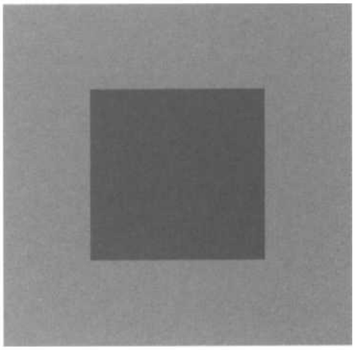


Figure 3.1
A simple picture

There are profound differences between vector and bitmapped graphics. It should be evident that they will make different demands on your computer system: a bitmapped image must record the value of every pixel, but a vector description can be much more compact. For example, consider an extremely simple picture consisting of a magenta square inside a green one. (See Figure 3.1, although it lacks the colour.) The complete picture is 45mm square; if it was stored with 72 logical pixels per inch (meaning that it could be displayed on a 72dpi monitor at its natural size without scaling), then it would be 128 pixels square, so its bitmap would consist of $128^2 = 16384$ pixels. If we assume that the intended destination is a consumer-level monitor capable of displaying 256 colours at a time, then we need 8 bits to distinguish the possible colours (see Chapter 6), which means that each pixel occupies one byte, and the entire image occupies 16 kilobytes of memory. The same picture, on the other hand, could be stored in the form of a description of its component rectangles and their colours. One possible format would be a short program in the PostScript page description language, such as the following, which occupies a total of just 78 bytes:

```
0 1 0 setrgbcolor
0 0 128 128 rectfill
```

```
1 0 1 setrgbcolor
32 32 64 64 rectfill
```

The first line sets the 'paint' colour to green, the second draws the outer square and fills it with that paint; the second pair of lines similarly construct the inner, magenta, square. Whereas displaying the bitmapped image only requires the pixels to be copied onto the monitor, displaying the PostScript version requires some software, in this case a PostScript interpreter, to translate it into a displayable form. This not only slows down the display of the picture, it also relies on a PostScript interpreter being available on the computer where the image is to be displayed.

The sizes of bitmaps and vectors are affected by the content of images in different ways. The memory requirement for any 45mm square, 256 colour bitmapped image at a resolution of 72 pixels per inch is 16 kbytes, no matter how complicated the image may be. In a bitmapped image we always store the value of every logical pixel, so the size of the image and the resolution at which it is stored are the only factors determining the amount of memory occupied.[1] In a vector representation, we store a description of all the objects making up the image; the more complex the picture, the more objects there will be, and the larger the description that will be required, but since we do not actually store the pixels, the size is independent of any resolution.

At least as important as the technical differences between the two different types of graphics are the differences in what you can easily do to your images using each approach. Although many image editing programs now allow you to combine vector and bitmapped graphics, traditionally a distinction has been made between *painting* programs, which operate on bitmaps, and *drawing* programs, which work with vector representations. Even though the distinction is less clear-cut than it used to be, it is still the case that packages such as Illustrator or Freehand are primarily intended for creating and editing vector graphics, and only provide limited facilities for dealing with bitmaps, while packages such as Photoshop and Painter, while providing extensive support for bitmap manipulation, either offer little support for vector graphics, or provide a poorly integrated drawing sub-system.

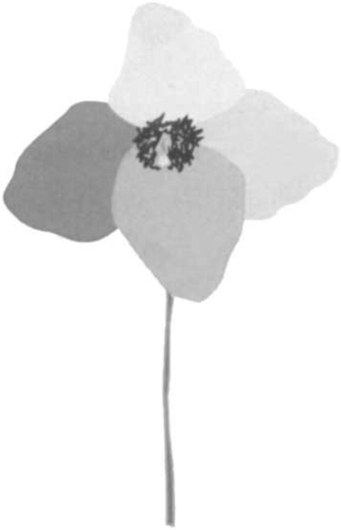


Figure 3.2
A vector poppy

Figures 3.2 and 3.3 show two flowers; the first is a vector drawing of a poppy, made in Illustrator, the other is a bitmapped image, scanned from a painting of an iris, executed in gouache. (Plates 3 and 4 show the same two flowers in colour.) The bitmap

[1] Unless we apply compression to it — see Chapter 5.

captures the textures of the original picture and provides a good reproduction of the continuous tones and translucent quality of the paint. The vector drawing has a quite different character, with clearly delineated shapes, made out of a collection of smooth curves, filled in with flat colour.

⇨ These examples have been deliberately chosen to emphasize the differences between the two types of image. A good illustrator working with a professional drawing package can produce much more subtle effects, but vector images are always based on the same elements of filled shapes.

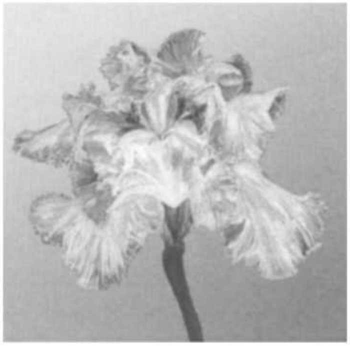


Figure 3.3
A bitmapped iris

Figures 3.4 and 3.5 demonstrate the difference between the two formats in another way. With the vector representation, it is easy to select the individual shapes — petals, stem, stamens — and move, rotate or otherwise transform them independently. Each element of the picture retains its identity and can be edited as an object because the position and attributes of each object are stored in the image model. To achieve a similar effect with the bitmapped image would require a painstaking process of masking out the required pixels and then re-touching the gap where the petal had been removed, since this image is just an array of pixels, with no underlying model to indicate which belong to the stem, and which to the petals. As a result, editing of parts of the image must be performed by selecting areas, either by painstakingly drawing round them or semi-automatically (and somewhat unreliably) by searching for tonal discontinuities or areas of similar colour. On the other hand, applying a special effect, such as distortion or blurring, to the bitmapped image is simple, whereas producing the same distortion on the vector image could only be done by first transforming it to a bitmapped format, since such effects transform each pixel, possibly using the value of its neighbours, but independently of whether they belong to the same object.

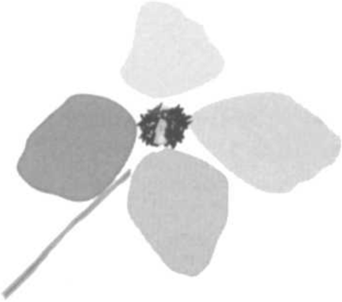


Figure 3.4
Transforming a vector image

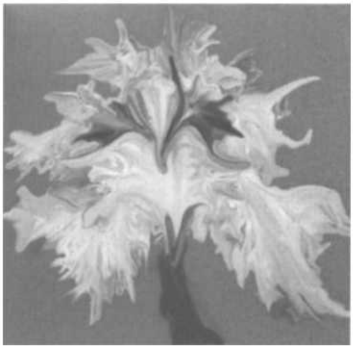


Figure 3.5
Applying effects to a bitmap

Another major difference between vector and bitmapped graphics is the way they behave when scaled or re-sized. If a bitmapped image is to be displayed at greater than its natural size, each logical pixel must be mapped to more than one physical pixel on the final output device. This can be achieved either by multiplying up the logical pixels, effectively increasing their size (for example, to double the linear dimensions, each pixel value in the model should be used to set a block of four pixels on the display), or by interpolating new pixels in between the stored ones. In either case, the effect will usually be a readily perceptible loss of quality. Since a vector image consists of a description of the component shapes of the picture, not the values of pixels, scaling can be performed easily as a simple mathematical operation, before the pixel values are calculated. As a result, curves, for example, will remain smooth, no matter how much a vector image is blown up, whereas they will become jagged or blurred if a bitmapped image is scaled up. Figures 3.6 and 3.7 illustrate this effect, being details of the two flower images scaled up by a factor of 8. The outlines of the poppy remain smooth, whereas the iris has become coarse, and the blocks of the original pixels are clearly visible, especially along the edge of the petal.

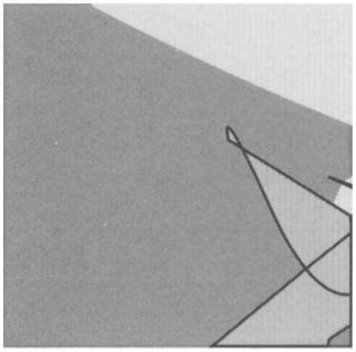


Figure 3.6
Scaling a vector image

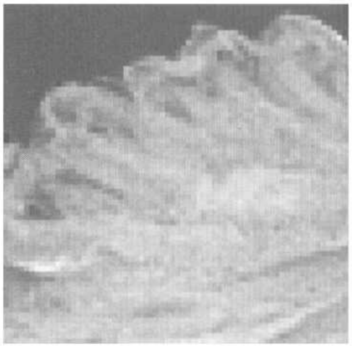


Figure 3.7
Scaling a bitmap

⇨ Related problems arise when bitmapped images are displayed on devices with different resolutions, since either they will be the wrong size, or they will exhibit the same loss of quality as occurs when they are scaled. This is more of a problem when images are being prepared for printing than it is with multimedia, since in the latter case we know that our images are going to be displayed on a monitor. Monitor resolutions do vary: whereas standard Macintosh displays have a resolution of 72 dpi, PC monitors usually use a resolution of 96 dpi, while multiple scan displays may provide a resolution of as much as 115 dpi at their higher settings. As a result, images prepared using such a display at a high resolution will appear larger on an ordinary PC or Macintosh. (Users of multiple scan displays will be familiar with the way in which all their desktop icons change size when they change the resolution.) This is usually considered acceptable, but should be borne in mind by designers.

The terms 'drawing program' and 'painting program' introduced earlier express the difference between the visual characteristics of vector and bitmapped graphics. A drawing program lets you build up a vector image out of individual curves, lines and shapes, so the result usually has much of the character of a pen and ink illustration, an air-brushed painting, or a technical diagram: shapes are defined by outlines, and colour is applied to the regions defined by those shapes. A painting program allows you to make a wide range of different marks, and apply colour to arbitrary areas of the image; modern painting programs do a pretty good job at simulating the appearance of natural media such as charcoal, pastel, watercolour or oil paint and the interaction of those media with textured and absorbent supports, such as coarse watercolour paper or canvas.

Drawing and painting programs use different sets of tools to perform the operations that are most appropriate to vector graphics and bitmapped images, respectively. Drawing programs have tools for constructing shapes such as rectangles and ellipses, and for drawing lines and curves, selecting objects, and moving and transforming them by scaling, rotation, reflection and skewing. Painting programs have a quite different set of tools, including brushes for making marks in a variety of styles, colour correction and retouching tools and filters for altering images, and selection tools for picking out areas and groups of pixels.

Painting programs generally offer more expressive possibilities, but at the expense of high memory requirements and scalability problems. For many applications, such as the display of data for scientific or business purposes, technical illustration, and some sorts of graphic design, the expressiveness of painting programs and bitmapped graphics is unnecessary, or even positively undesirable, and vector graphics produced by drawing programs will be preferred.

Memory requirements, the visual characteristics of the image produced, and the possibilities for transformations and effects might influence your decision as to which format to work with. Another important factor is the source of your image. Scanned images, screen shots, photographs from a digital camera, and captured video frames are all inherently bitmaps, because of the way the hardware from which they originate works. Accordingly, for the manipulation and re-touching of photographs and for most prepress work, painting programs are the only possible choice. Charts, diagrams, and other data visualizations generated by a program from data usually, but not invariably, use vector graphics. Artwork made on a computer can be in either form, with the artist's personal preferences, together with the factors mentioned above, and the availability of suitable software, determining the choice.

Combining Vectors and Bitmaps

The different qualities of vector and bitmapped graphics, deriving from the fundamentally different image representations each employs, make them suitable for different tasks. It is not uncommon to find tasks requiring a combination of these qualities that call for images containing elements of both types. For example, a graphic designer might wish to use a scanned image as a background over which drawn (vector) shapes are arranged. There are several ways in which this can be achieved.

The first is to transform vectors into bitmaps, or *vice versa*. Hence, our designer might prepare their drawn elements in a vector graphics progam, turn the drawing into a bitmap, and composite that with the scanned background in a bitmapped image manipulation program. It is relatively easy to turn a vector graphic into a bitmapped image. The process of interpreting the vector description, known as *rasterizing*, can be accomplished using the same algorithms that are used to display the image on a monitor. The rasterized image loses all its vector properties, though — it ceases to be resolution-independent, so a resolution must be chosen when the rasterization takes place, and the individual shapes can no longer be selected and transformed, they just become areas of pixels. However, as a result they can be treated to the full range of bitmapped effects.

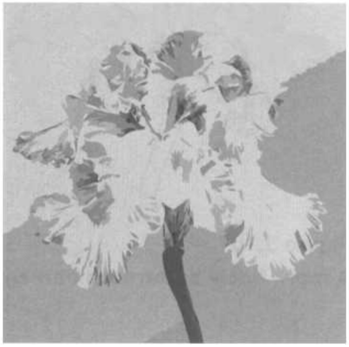


Figure 3.8
A vectorized bitmap

Going in the opposite direction, from pixels to vectors, is more problematical. It requires software to identify the boundaries of shapes within the image, then to approximate that boundary using the available sorts of curves and lines, and to colour them in appropriately. When the original image features subtle tonal graduations, and soft outlines, the process is not likely to be very successful. Figure 3.8 shows the result of an attempt to vectorize the bitmapped flower image from Figure 3.3. (It was done with Flash.[2]) The resulting vector image is made out of many small curves and shapes, which means that it is neither as compact nor as easy to edit as an image made using vector graphics from the start. (In fact, the file containing this particular vector image is bigger than the bitmapped version at screen resolution it was traced from.) Resizing, rotation, and other transformations that can be performed naturally and simply on vectors can be applied to the vectorized image.

⇨ Vectorization can be used in other ways than simply as a means of changing the representation of an image. It can be used in a controlled fashion to generate a starting point for a new vector image. Figure 3.9 shows the result of applying Illustrator's autotrace tool to the iris painting. This tool merely traces the outline of a shape, producing the path shown here. While this is barely recognizable as a version of the original, it could now be worked on inside Illustrator as part of a new composition based on the organic shape of the flower.

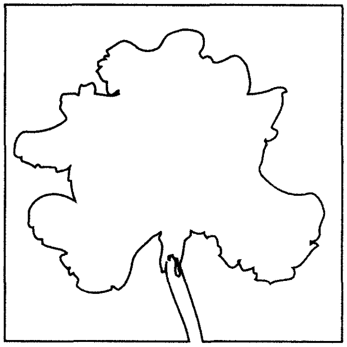


Figure 3.9
An autotraced bitmap

The autotrace tool will find different shapes depending on where it is applied. Figure 3.10 shows the paths produced by deliberately choosing parts of the image to produce a collection of curves that

[2] And to be fair to Flash, we could have adjusted some parameters to make a better job of it, at the expense of creating a picture made out of an enormous number of tiny vectors.

make a better approximation to the iris. Again, these could be used as the basis of a new drawing, retaining the shapes but exploiting the capabilities of the vector tools.

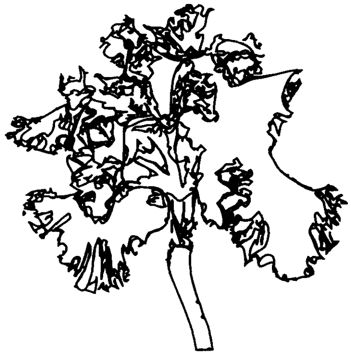


Figure 3.10
A more finely autotraced bitmap

Most drawing programs allow you to import bitmaps without vectorizing them. A bitmap imported in this way is treated as an indivisible object; it can be moved, and some transformations may be applied to it (although they might affect the quality of the image). However, it cannot be broken into its component shapes, the way a vectorized image or any shape drawn in vector form can be, and certain filter effects that rely on being able to change individual strokes cannot be applied to it. Commonly, drawing programs allow pointers to bitmapped images to be imported instead of making a copy of the image itself. This means that the bitmap can be edited in a painting program, after it has been incorporated into a vector image, and the changes will be seen in the vector image.

Sometimes it is not actually necessary to incorporate bitmaps into vector graphics; all that is wanted is for the vectors to have the appearance of bitmapped images. Until quite recently this could only be achieved by rasterizing and retouching in an image editor. Nowadays, some drawing programs — MetaCreations' Expression and version 8.0 of Illustrator, for example — allow 'brush strokes' to be applied to the lines and curves making up vector shapes, so that they appear to have been made with natural media, such as watercolour or pencil; calligraphic effects, such as a variation in the thickness of a line depending on its direction, can also be achieved.[3] This application of a natural media appearance to vector strokes is quite distinct from using a painting tool to draw a smooth curve, for example. In the latter case, the pixels on the path following the brush's movement are coloured to simulate the effect of a painted or drawn line; in the former case, the path is stored in the usual vector form, as the defining parameters of a collection of curves, and the associated appearance is added to it when it is displayed. The path retains all the desirable qualities of vector graphics: it can be transformed, altered, and displayed at any resolution, but it does not have the uniform appearance normally associated with vector graphics. Perhaps most interestingly, the brush stroke applied to a path can be changed, so the appearance of the marks can be altered without re-drawing the path. As far as the appearance of the resulting artwork is concerned, these effects blur the distinction between drawing and painting programs, but internally the strokes are still being applied algorithmically to paths stored in vector form.

[3] Similar technology is used in Painter to allow vector strokes to be used in a bitmapped image editor, with all the natural media from that program's bitmapped world available to them.

Layers

The arrangement of artwork on *layers* is an organizational device that is common to both vector and bitmapped images. Since the introduction of the concept of layers in Photoshop 3, it has become one of the most significant ways in which digital technology has affected how artists, designers and illustrators work. As we will see in Chapters 10, 11 and 14, the intuitive appeal of the layer metaphor has led to its adoption in other media, too.

A layer is often likened to a digital version of a sheet of clear acetate material, like an overhead projector transparency. You can draw or paint on parts of the layer, leaving some of it transparent. An image can be constructed by stacking layers on top of each other; where a layer is transparent, the layer below it shows through. This may not sound very exciting — you can always draw things on top of other things — but a layer allows you to collect together part of an image and treat it as a unit.

An immediate consequence of this ability is that it provides a way of distinguishing objects in a bitmapped image. Normally, as we have shown, if you make a picture of a flower as a bitmapped image, there is no discrete object corresponding to each petal, just areas of pixels. By placing each petal on a different layer, though, they can be moved or modified individually, much as the individual shapes making up a vector image can be. One specific way in which artists take advantage of the separation that layers allow is by using one layer as a background against which objects on other layers are superimposed. These objects can then be moved about over the background, until a satisfying arrangement is found. If they were not on separate layers, whenever an object was moved, it would be necessary to touch in the background where it had been. In a similar way, the use of layers makes it easy to apply effects to parts of an image, since effects can be applied to individual layers. Thus, for example, a background layer might be blurred so that elements on layers placed over it will stand out.

Layers facilitate a graphic style resembling collage. You will probably recognize the Layered Look in much graphic design that was produced in the 1990s.

A different way of using layers is as digital tracing paper. For example, rather than try to vectorize an image like our iris, you might prefer to import it into a drawing program in its bitmapped form, and then create a new layer on top of it, on which you could draw with the vector tools, using the imported image as a guide. (Drawing programs usually allow you to dim layers so that you can more easily see what you are doing on another layer.) Once that task has been achieved, the guide layer can be deleted without affecting any other layer.

Yet another way of using layers is for experimentation. Layers can be reordered without affecting their contents, so that different stacking arrangements can be tried out. Layers can be duplicated, and the duplicates altered separately; any layer can be made invisible, so different versions of a layer can be displayed in turn, to see which is better. When a picture is finished, it is normal to delete any invisible layers, and merge all the remaining ones down to a single layer, since this uses less space.

We have likened layers to transparent sheets of acetate, but, as data structures inside a computer are not subject to the physical limitations of materials, they can behave in ways that acetate cannot. In particular, the degree of transparency can be varied.[4] By using layers that are only partially transparent, backgrounds can be dimmed. The precise way in which separate layers are combined may also be modified. The normal behaviour of layers is for the areas that are not transparent to cover any layers beneath. Sometimes, it may be preferable for these areas to be blended with lower layers instead, or dissolved into them. Transparency may be made conditional on the brightness of one or other of the layers, so that blending takes place below a threshold value, but above that value the superposed layer conceals what is below it.

Figure 3.11 is a simple illustration of combining layers. The starting point is the two photographs shown at the top, one an Alpine hillside in winter, the other a piece of woodland. These were scanned, corrected for tonal balance, and cropped to the same size. The light areas of the woodland scene were selected and filled with pure white. This modified image was then pasted as a new layer over the background of the hillside. The hillside is unaffected, being on a separate layer. At the bottom, we show these two layers combined, with different threshold values being used to control the layer blending. On the left, the lower level shows through, producing a slightly disturbing incoherent scene; this was achieved

[4] In Photoshop you actually change the opacity, which is one minus the transparency.

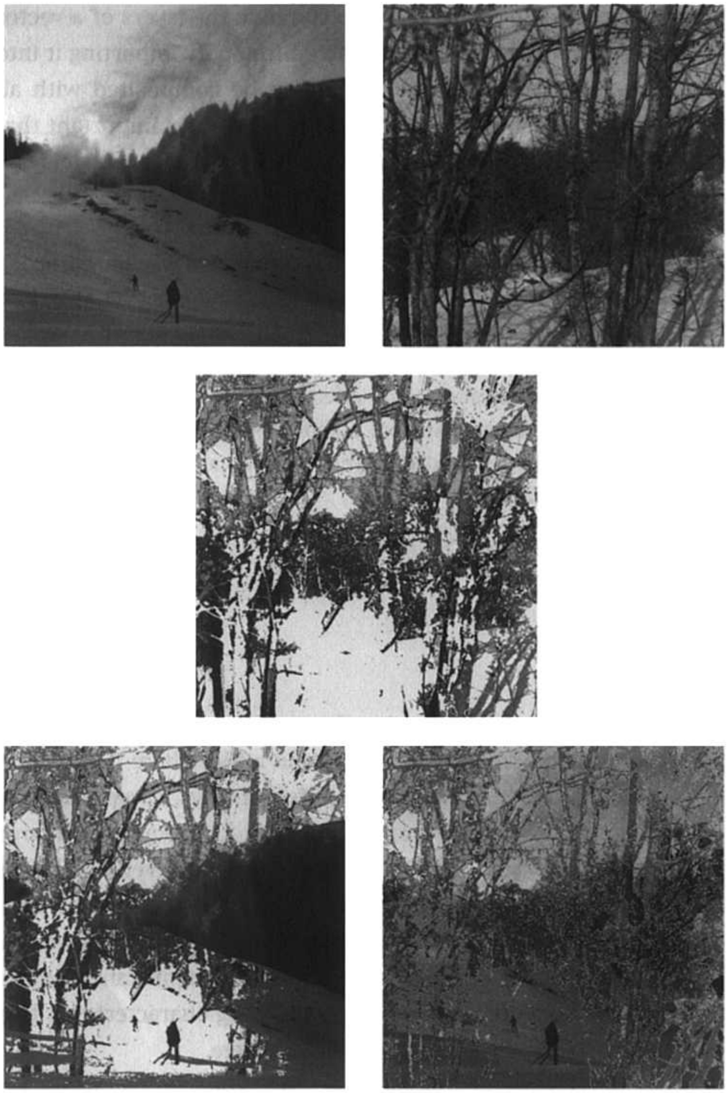


Figure 3.11
Producing artificial compositions with layers

by using a low white threshold and a high black one for the blending on the upper layer. On the right, by raising the black threshold for the background layer to quite a high value, the dark bulk of the hill on the right hand side of the picture has been effectively obscured, and the trees appear to be part of the same scene as the skiers — we have manufactured a new landscape.

Layer compositing of this sort can only be performed on bitmapped images, because it is done by computing the value of each pixel individually. When it is necessary to combine the layers of a vector illustration, it must be converted into a bitmap by importing it into a painting program, where its layers can be composited with all the flexibility available. For this to be possible, it is important that the layers be preserved when the image is imported — only certain combinations of programs can do this correctly.

⇨ The layer metaphor has found such favour as a way of organizing images that it has been extended to incorporate effects as well as image elements. Photoshop's *adjustment layers* are described as being a layer through which you can look at the image through a medium that applies some effect, such as a tonal adjustment. In practice, this makes them a tool for applying effects without changing the pixels on image layers. They thus provide a safe means of experimentation.

File Formats

In order to preserve images and exchange them between programs, it must be possible to store image data in a file. There is considerable scope for encoding the data in different ways, compressing it, and adding supplementary information to it. Consequently, a large number of different graphics file formats has been developed.[5] As is the case with programming languages, most of these are only used by a limited circle of enthusiasts, for some specialized applications, or on particular platforms. There remains a significant number of different formats in wide use, with differing characteristics that make them suitable for different types of image.

For bitmapped images, one of the main differences among file formats is the way in which they compress image data. Bitmapped images can contain a large number of pixels, so they require large files, often occupying several megabytes per image at medium to high resolutions. In order to reduce the storage and bandwidth requirements of such images, data compression techniques are often applied to them. We will describe image compression in more detail in Chapter 5, but you need to be aware at this stage of the distinction between *lossless* and *lossy* compression. Lossless compression algorithms have the property that it is always possible to reconstruct the original data exactly from its compressed version; lossy algorithms discard some data — in the case of images, data

[5] The *Encyclopedia of Graphics File Formats* [Mv96] describes nearly 100, with more on its associated Web site.

representing visually insignificant details — in order to achieve greater compression.

We also defer a full description of colour to a later chapter (Chapter 6) but note that one way of reducing the size of a bitmapped image is to restrict the number of different colours that it can contain. If an image uses at most 256 different colours, each pixel only requires a single byte, whereas if it is to be allowed the full range of millions of colours which most monitors are capable of displaying, three bytes per pixel are needed.

The advent of the World Wide Web has had something of a standardizing influence; although it does not specify any particular graphics file formats, the necessity for cross-platform compatibility has led to the adoption of certain formats as *ad hoc* standards.

The first of these is *GIF*, originally developed by CompuServe as a common format for exchanging bitmapped images between different platforms. GIF files use a lossless compression technique, and are restricted to 256 colours. One of this format's most useful features is that one colour can be designated as transparent, so that, if the GIF image is displayed against a coloured background or another image, the background will show through the transparent areas. This, in effect, allows you to produce images that are not rectangular. GIFs are most suitable for simple images, such as cartoon-style drawings and synthetic images produced on computers. They are less succesful with scanned and photographic images, which may have wide colour ranges and tonal variations.

For these images, *JPEG* is preferred. Strictly speaking, JPEG is a compression technique,[6] and images that have been compressed using it may be stored in any of several file formats — the JPEG standard does not specify any. Colloquially, though, the name 'JPEG file' is used to refer to what are correctly called *JFIF* — JPEG File Interchange Format — files. To complicate matters further, the more recently developed SPIFF format is actually the officially endorsed file format for JPEG images, but as we remarked, JPEG data can be embedded in other files, including TIFF and EPS, which we will say more about shortly.

The third, and newest, file format that is widely supported on the Web is *PNG*,[7] which was devised to supersede GIFs. The trouble with GIF is that the compression algorithm it employs is covered by a patent owned by Unisys, who require a licence fee to be paid for any program that implements GIF compression or decompression.

[6] *Really strictly* speaking, JPEG is the Joint Photographic Experts Group, who developed the compression technique, and after whom it is named.

[7] Pronounced 'ping'.

PNG, on the other hand, uses a different lossless technique that is not encumbered in this way, and can therefore be implemented freely by anybody. Additionally, PNG is not restricted to 256 colours and it offers a more sophisticated form of transparency than does GIF. The PNG format was developed under the aegis of the W^{3}C, and its specification, published in 1996, has the status of a W^{3}C Recommendation. Support for PNG has been slow to develop, but it is likely to be supported by all the major Web browsers and graphics programs by the end of 1999.

Outside the World Wide Web, other commonly encountered bitmap graphics file formats include *TIFF*, *BMP*, and *TGA* (often called Targa). TIFF (Tag Image File Format) is an elaborate extensible file format that can store full colour bitmaps using several different compression schemes, including JPEG. It is supported by most painting programs on all platforms, although not all programs are equal in the comprehensiveness of their support, so that TIFF files created by one program cannot always be read by another. TIFF is natively supported by Windows, as is BMP. Indeed, BMP is more properly called the Microsoft Windows Bitmap format. As such, it is platform-dependent, but the near ubiquity of the Windows platforms means that it is widely understood by programs on other systems. Unlike most other bitmapped formats, BMP only supports a simple form of lossless compression, and BMP files are usually stored uncompressed. TGA files have achieved wide currency, because the format was one of the earliest to support more than 256 colours on PCs. It was designed to accompany a proprietary range of video capture boards, but has become more widely supported by programs on most platforms, although its use is probably declining.

All of the formats mentioned so far store bitmapped images. The situation with respect to vector graphics is slightly different. Vector graphics is dominated by PostScript. PostScript, developed by Adobe Systems in the mid-1980s, could be described as a programming language with built-in graphics capabilities that allow it to specify the appearance of pages in the form of a program describing how to place graphic elements — paths, shapes, and fills — on the page. Procedures can be defined, so that an application that generates PostScript (it is not the sort of programming language that anyone would want to write by hand) can create a set of operations suited to its own needs and view of page layout.

➲ Illustrator, for example, when writing PostScript output, prepends a set of definitions which define procedures corresponding to its drawing primitives, in terms of PostScript's own, lower-level, primitives.

PostScript is intended as a page layout language, which makes it unsuitable for storing single images in isolation, rather than as components of a page. *EPS* (encapsulated PostScript) applies certain conventions to the use of PostScript to ensure that the images in an EPS file are self-contained so that they can be incorporated in other documents. In particular, an EPS file must contain a *bounding box comment*, describing the dimensions of the image. EPS is one of the most widely-used vector graphics formats, but a full PostScript interpreter is required to display EPS images.

In February 1999, the W^{3}C issued a first working draft specification of a Scaleable Vector Graphics format, *SVG*, which is defined in XML (see Chapter 8), the new extensible markup language for the Web. In essence, though, SVG is a derivative of PostScript that uses the same imaging model but a fixed repertoire of operations and is thus easier to implement, and is more compact for transmission over networks.[8] At the same time, the *SWF* format, originally developed for vector animations using Macromedia's Flash (see Chapter 11) but now an open standard, is in wide use for vector images. Although SWF does not have the sanction of the W^{3}C, it has been supported, either via a plug-in or directly, in the major Web browsers for some years,[9] which gives it *ad hoc* status as a standard. SWF is a highly compact format, and can be rendered very quickly. Although, because of its origins, SWF is mostly used for animations, its use for still images is increasing, and may yet pre-empt SVG.

EPS, SWF, and SVG are not just vector formats, although their vector capabilities are their essential feature. It is possible to incorporate bitmaps into these files, too, as self-contained objects. This sort of file format that accommodates both vector and bitmapped graphics, and usually text, too, is sometimes called a *graphics metafile.* Other graphics metafile formats include Macintosh PICT files and Microsoft's Windows Metafiles (WMF). Purely vector-based formats are in fact quite rare and, for the most part, associated with computer-aided design packages: AutoCAD DXF is a complex file format widely used for the exchange of vector data, for example. Vector-based formats are also widely used for three-dimensional models, which we will look at in Chapter 4.

As well as the general purpose formats we have described, certain proprietary formats associated with popular programs are also

[8] SVG is closely based on a proposal from Adobe, originally designated PGML (Precision Graphics Markup Language) before the post-HTML vogue for names ending in ML passed.

[9] Unlike SVG, which, at the time of writing, has not been implemented at all.

widely used. In particular, Photoshop and Illustrator files are commonly used as interchange formats for bitmapped and vector graphics, respectively, in the pre-press and publishing industries. One noteworthy feature of these formats is that they preserve any layers that might have been used in making the image. Exporting, say, a Photoshop image to a PICT file will 'flatten' it, combining all the layers into one. This is undesirable if it is intended to make further changes to the image.

Evidently, with so many different file formats in use, conversions between formats are often required. These vary in complexity from a simple re-ordering of bytes to the sort of vectorization operation described in the previous section. In this context, mention should be made of QuickTime. Although QuickTime is most often considered to be a digital video format (and we will describe it further in Chapter 10) it is better described as a multimedia architecture, which provides a framework and a set of software components for storing and manipulating various digital media formats, of which video is only one — still images are another. QuickTime has its own image format, which generally contains JPEG compressed image data, but more importantly, it provides a collection of import and export components, that allow any program that uses QuickTime to work with files in many formats, including JFIF, PNG, GIF, TGA, PICT, TIFF, Photoshop and SWF (but not EPS).[10]

[10] Some of these are only supported by version 4.0 or higher.

Further Information

The topics introduced in this chapter will be taken up in more depth in Chapters 4 to 6. The references given at the end of those chapters should be consulted for further details. [Mv96] describes many graphics file formats; [Ado90b] is the definitive description of the PostScript language.

Exercises

1. Describe three significant differences between vector and bitmapped graphics.

2. For each of the following kinds of image, which would be more suitable, bitmapped images or vector graphics?

   (a) Circuit diagrams.

   (b) Architectural drawings.

   (c) Botanical drawings.

   (d) Pie charts.

   (e) Fingerprints.

   (f) A map of the world.

   (g) Brain scans.

   (h) Illustrations for a children's alphabet book.

   (i) A reproduction of the Mona Lisa.

   (j) A simple cartoon character, such as Mickey Mouse.

3. Since GIF files are most suitable for the same sort of images as vector graphics, why would you use them in preference to EPS or some other vector format?

4. What file format would you choose to present

   (a) A photograph of yourself.

   (b) A comic strip cartoon drawing.

   (c) The floor plan of a museum.

   for each of the following purposes:

   (a) Incorporation in a Web page.

   (b) Incorporation in a CD-ROM multimedia encyclopedia.

   (c) Transfer to another platform.

   Justify your choice in each case.

5. Do you think the 'layered look' in graphic design is a result of the introduction of layers in graphics programs, a response to the conditions of contemporary life, a reflection of recent ideas in philosophy, or some other cause or combination of causes?

4 Vector Graphics

Vector graphics provide an elegant way of constructing digital images whose representation is compact, scaleable, resolution-independent, and easy to edit. The compactness of vector graphics makes them particularly attractive for networked multimedia, where the indiscriminate use of bitmapped images can lead to excessive download times, because of the large sizes of the image files. A common complaint amongst domestic Internet users is that the increasing trend towards graphically rich Web sites means that pages take too long to download. Equally, Web designers who are aware of this complaint feel inhibited about using images freely. Vector images can be a fraction of the size of bitmaps, but the absence of any standard format for vector graphics on the Web left little opportunity for using them. As the official SVG and *de facto* SWF standards are adopted, this will change.

Although vector graphics has been eclipsed in recent years by bitmapped representations for two-dimensional images, for three-dimensional work — that is, images that are constructed as projections of a 3-D model — vector techniques are mandatory, since the use of models made out of the three-dimensional equivalent of pixels (*voxels*) is impractical on all but the most powerful equipment.

Fundamentals

In vector graphics, images are built up using shapes that can easily be described mathematically. We expect many readers will be familiar with at least the rudiments of coordinate geometry, the field of mathematics underlying the representation of shapes in vector graphics. For the benefit of those who may not be used to thinking about shapes and outlines in terms of coordinates and equations, we will begin with a very brief review of the basic concepts.

Coordinates and Vectors

Since an image is stored as a rectangular array of pixels, a natural way of identifying any single pixel is by giving its column and row number in that rectangular array. If we number the columns from left to right, and the rows from the bottom of the image to the top, both starting at zero, then any pixel is uniquely identified by the pair (x, y), called its *coordinates*, where x is the column number and y the row. In Figure 4.1, the pixel labelled A is at $(3, 7)$, while B is at $(7, 3)$. The point labelled O is at the *origin* $(0, 0)$.

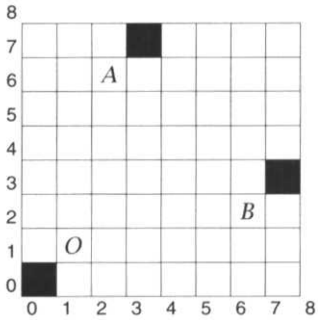


Figure 4.1
Pixel coordinates

The coordinates of pixels in an image must be integer values between zero and the horizontal (for x coordinates) or vertical (for y) dimensions of the image. For the purposes of modelling shapes in a device-independent way, we need to generalize to a coordinate system where coordinates can have any real value. That is, instead of identifying a finite pixel on a grid, coordinates identify infinitely small geometrical points, so that there are infinitely many of them between, for example, $(0, 0)$ and $(1, 0)$. Additionally, the values are not restricted to any finite maximum. We also allow negative coordinates: points with negative x coordinates lie to the left of the origin, while those with negative y coordinates lie below it. Thevertical line running through the origin, which consists of all the points with an x coordinate of zero, is called the *y-axis*, while the horizontal line through the origin is the *x-axis*. We can label the x- and y-axes with the x and y coordinates of equally spaced points to produce the familiar graph axes, as shown in Figure 4.2, from which coordinates can easily be read off. Vector drawing programs usually allow you to display axes (usually called *rulers*) along the edges of your drawing.

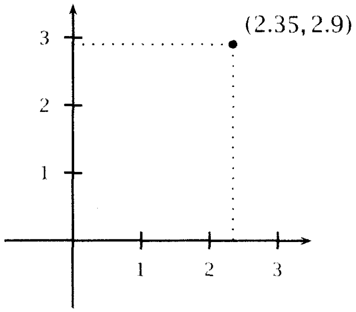


Figure 4.2
Real coordinates and axes

⇨ Although drawing programs (and PostScript) usually follow mathematical convention by using a coordinate system where y values increase upwards, lower level graphics packages, such as QuickDraw or the Java Abstract Windowing Toolkit, often use the opposite convention, with y values increasing downwards. This same convention is employed in some graphics file formats, too. It corresponds more closely to the way pixels are drawn onto physical output devices. You generally should not have to concern yourself with the coordinate system used by the output device. If your drawing program produces PostScript output, then the PostScript interpreter will convert from its coordinate system to that of whichever device your drawing is rendered on.

This conversion is an example of a *coordinate transformation*, whereby coordinates in one system (the *user space*) are transformed into a different one (the *device space*). Coordinate transformations are inevitable if we are to produce device-independent graphics since, in general, we cannot know the coordinate space of the output device. Another example of a coordinate transformation occurs when an image is rendered in a window on a display. Since we cannot know when the image is prepared whereabouts on the screen the window will be positioned, there is no possible way of using absolute screen coordinates to specify the objects in the drawing. Instead, the drawing is prepared in the user coordinate space, and transformed to the device space when it is displayed.

Pairs of coordinates can be used not only to define points, but also to define displacements. For example, to get from A to B in Figure 4.1 we must move 4 units to the right, and 4 units down, or, putting it another way, -4 units up. So we can specify the displacement from A to B by the pair $(4, -4)$. In general, for any pair of points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$, the displacement from P_1 to P_2 is $(x_2 - x_1, y_2 - y_1)$, which we write as $P_2 - P_1$ (see Figure 4.3). When a pair of values is used to specify a displacement in this way, we call it a two-dimensional *vector*. Note that a vector has a direction: $P_2 - P_1$ is not the same as $P_1 - P_2$, since moving from P_1 to P_2 is different from moving in the opposite direction from P_2 to P_1.

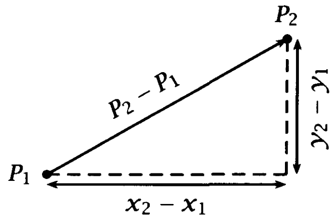


Figure 4.3
A vector

⇨ The term *vector graphics* was originally used to refer to the production of images on output devices where the position of the electron beam of a cathode ray tube was controlled directly by setting its (x, y) coordinates, rather than scanning the screen in a raster pattern. By changing the coordinate values, the beam was made to trace out vectors, hence the name. This type of graphics device only lends itself to the drawing of simple shapes, which is why the name is retained for the shape-based graphics systems we are describing here.

A coordinate system lets us identify points in space. The power of coordinate geometry comes from using letters to represent 'unknown' values and using equations in those values to specify relationships between coordinates that characterize geometrical shapes. For example, if (x, y) is *any* point on a straight line that passes through the origin at an angle of 45° from south-west to north-east, then it must be the case that $x = y$, as you can show using simple geometry. We can use the methods of coordinate geometry to derive equations for arbitrary straight lines, circles, ellipses, and so on. Using such equations we can represent shapes simply by storing the appropriate constants that appear in the equations, since it is these which distinguish between different shapes belonging to the same class.

➲ Practical considerations might lead us to use a slightly less obvious representation. For example, the equation of a straight line is usually written as $y = mx + c$, where the constants m and c are the slope and intercept, respectively. However, since we can only use finite segments of lines, it is necessary to store the endpoints. The values of m and c can be deduced from these. A bit of simple algebraic manipulation, which many readers will have done at school, demonstrates that, if the line passes through (x_1, y_1) and (x_2, y_2), m is equal to $(y_2 - y_1)/(x_2 - x_1)$ and c is equal to $(x_2y_1 - x_1y_2)/(x_2 - x_1)$. Hence, the cordinates of the endpoints are enough on their own to specify both the extent of the line and the constants in its equation.

In a similar way, the values actually stored for other shapes are not necessarily those that a simple mathematical analysis would suggest.

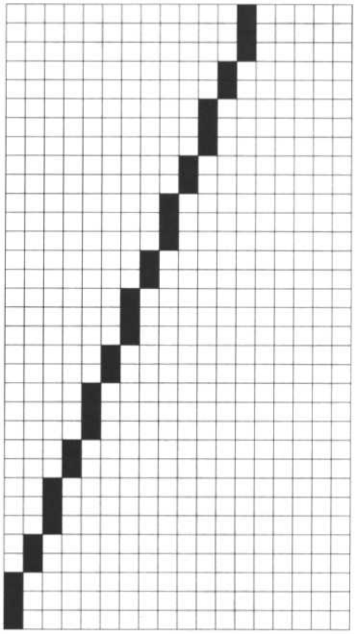


Figure 4.4
Approximating a straight line

When it becomes necessary to render a vector drawing, the stored values are used, in conjunction with the general form of the description of each class of object, to set the values of pixels to form an image of the object described. For example, if a line has endpoints $(0, 1)$ and $(12, 31)$, we could compute the y coordinate corresponding to each integer value as x was stepped from 0 to 12. Remember that a pixel's coordinates are always integers (whole numbers), and we cannot set the value of just part of a pixel. The pixel image can, therefore, only ever approximate the ideal mathematical object which the vector model describes. For example, the line just described has equation $y = 5x/2 + 1$, so for any odd integer value of x, y must be rounded up (or down — as long as it is done consistently) to get its corresponding integral value. The coordinates of pixels along the line would be $(0,1),(1,4),(2,6),(3,9)\ldots$. To get a continuous line, we set blocks of pixels, but the height of the blocks alternates between 2 and 3 pixels, so that the ideal straight line is approximated by an uneven staircase, as shown in Figure 4.4. This is inevitable, since the output devices we are using are based on a grid of discrete pixels. If the resolution of the output device is low, (i.e. the pixels are relatively large) the jaggedness of lines, and other effects due to the same cause, can become offensive. This phenomenon is colloquially known as 'staircasing' or, more colourfully, 'the jaggies'.

Anti-aliasing

The process of rendering a vector object to produce an image made up of pixels can usefully be considered as a form of sampling and reconstruction. The abstract line that we would like to draw is a continuous signal — the x and y coordinates can vary infinitesimally — which has to be approximated by a sequence of pixel values at fixed finite intervals. Seen in this light, jaggies are a form of aliasing caused by undersampling. This is consistent with the common-sense observation that as the resolution of the output device increases — that is, we sample at a higher rate — the individual pixels get smaller, so that the jagged effect becomes less noticeable.

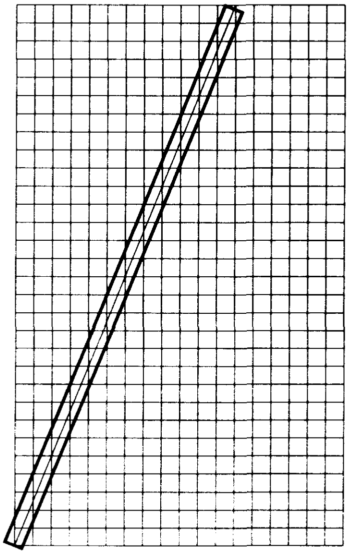


Figure 4.5
The single pixel approximation to a line.

You will recall from Chapter 2 that it is necessary to sample at a rate greater than twice the highest frequency in a signal in order to reconstruct it accurately, and that high frequencies are associated with abrupt changes. If an image contains a sharp hard-edged boundary, its brightness or colour will change directly from one value to another crossing the boundary without any intermediate gradation. The representation in the (spatial) frequency domain of such a discontinuity in the spatial domain will include infinitely high frequencies. Consequently, no sampling rate will be adequate to ensure perfect reconstruction. In other words, jaggies are always possible, no matter how high the resolution that is used for rendering a vector shape.

In any case, practical and financial considerations impose a limit on the available resolution. In particular, vector graphics that are used in multimedia presentations will usually be rendered on a monitor with a resolution between 72 and 120 dots per inch, and aliasing will be readily visible. To reduce its impact a technique known as *anti-aliasing* is often employed.

Looking back at Figure 4.4, you will see that the staircase effect is a result of the pronounced contrast between black and white pixels. We can soften the effect by using intermediate grey values for some pixels. In terms of the frequency domain representation, we are removing the spurious high frequencies, and replacing them with lower frequencies. We cannot simply tone down the black pixels to produce a grey line instead of a black one; we want to try to use a range of greys to somehow convey to the eye and brain of someone looking at the displayed line the appearance of a smoothness that cannot actually be achieved by the finite pixels.

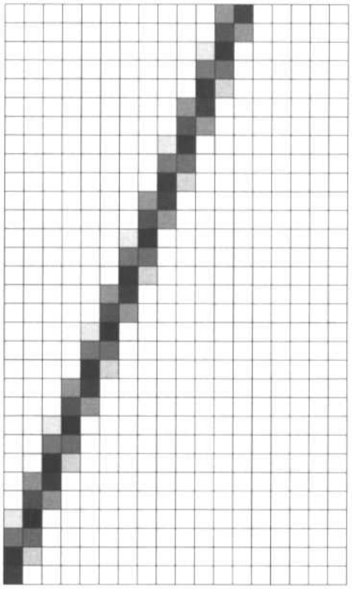


Figure 4.6
Anti-aliased line

If we did not have to be concerned about the orientation of the pixel grid, at best we could produce a smooth line one pixel wide, as shown in Figure 4.5. Our original attempt at rendering the line on the basis of its defining equation had the effect of setting to black those pixels whose intersection with this one pixel wide rectangle was at least half the area of the pixel. Anti-aliasing is achieved by colouring each pixel in a shade of grey whose brightness is proportional to the area of the intersection, as shown in Figure 4.6. The number of pixels that are no longer white is greater than before, but if we take the grey pixels and multiply the area of each one by the value used to colour it, the total amount of greyness, as it were, is the same as that produced by only using a single black level to colour the original collection of pixels. At this magnification, the result looks fairly incoherent, although it should be apparent that the jaggedness has been subdued somewhat. At normal viewing resolutions, anti-aliasing can significantly reduce aliasing effects, albeit at the expense of a certain fuzziness.

Shapes

When using a drawing program you are restricted to the shapes it provides, which are generally shapes with a simple mathematical representation that can be stored compactly and rendered efficiently. Usually the repertoire of shapes is restricted to rectangles and squares (possibly with rounded corners), ellipses and circles, straight lines, polygons, and a class of smooth curves, called *Bézier curves*, although spirals and stars are sometimes supplied too. Shapes built up out of these elements can be filled with colour, patterns or gradients. Because the program works with a description of the shape, not a map of its pixels, it is easy to move, rotate, scale and skew shapes. It may sound as though vector programs are very limited in their graphic capabilities, but they can be used to achieve complex and subtle effects, especially once you understand how to work with Bézier curves. Vector drawing programs must be approached in a different way from a freehand drawing medium, though.

A good way of appreciating the potential and limitations of vector graphics is by looking at the capabilities offered by a typical drawing program. We will use Illustrator as an example; you will find the same concepts and facilities in any other professional drawing package. Illustrator generates PostScript, so the description of Illustrator's facilities provides an indirect account of how graphical objects are constructed by PostScript programs. Generally, the way in which a drawing package allows you to work with shapes is a reflection of the way in which those shapes are represented inside the program or in PostScript code. For example, you draw a line by selecting a pen tool and clicking the mouse or pressure-sensitive pen at each end of the line: internally, a line is represented by the coordinates of its end-points; PostScript's `lineto` operator takes the coordinates of a point and draws a line to it from the current point, established by its `moveto` operator.

A sequence of connected lines, such as the one shown in Figure 4.7 is sometimes considered as a single object, called a *polyline*. Closed polylines, whose first and last points coincide, form regular or irregular polygons, and can be used to draw rectangles, for example.

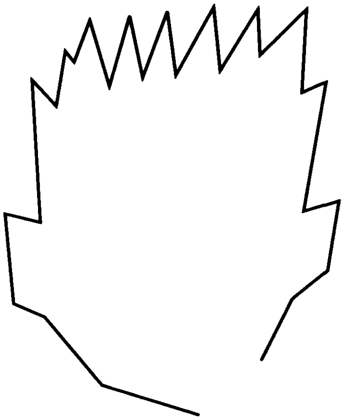


Figure 4.7
A polyline

Alternatively, a rectangle whose sides are parallel to the axes can be drawn by selecting the rectangle tool, holding down the mouse button where you want one corner, and dragging to the opposite corner: a rectangle can obviously be completely described by the coordinates of its opposite corners. In Illustrator, rectangles can also be drawn with the centred rectangle tool. With this, you begin at the point where you want the centre to be, and then drag out one corner. It should be clear that it is possible for the program to compute the coordinates of opposite corners from those of the centre and one corner, and *vice versa*.

Ellipses can be drawn by selecting the appropriate tool and dragging from one point on the perimeter to the opposite point. A pair of points is sufficient to determine the shape and position of the ellipse, and their coordinates can be transformed into one of many convenient representations of the object.

Squares and circles are special cases of rectangles and ellipses, respectively, and so do not need any special representation of their own. It is helpful when drawing to be able to ask the program to restrict rectangles and ellipses to squares and circles. In Illustrator this is done by holding down the shift key while using the rectangle or ellipse tool.

Curves

Lines, polylines, rectangles and ellipses are sufficient for drawing many sorts of technical diagrams (particularly when your lines can be decorated with arrowheads, as they can be in all professional drawing programs). Less constrained drawing and illustration requires more versatile shapes, which are supplied by Bézier curves.

Bézier curves are a class of curve that, as we will shortly demonstrate, have several properties that make them especially useful for graphics. A Bézier curve is completely specified by just four points: its two endpoints, and two more points, called *direction points*, which do not usually lie on the curve itself. The endpoints and direction points are collectively referred to as *control points*. The name 'direction points' indicates the purpose of these points: they show the direction in which the curve sets off from each endpoint. This is shown in Figure 4.8: P_1 and P_4 are the endpoints of the curve, P_2 and P_3 are its direction points, and you can see that the curve is accurately described by saying that it begins at P_1, setting off towards P_2, curving round so that is arrives at P_4 from the direction of P_3.

The length of the lines from each endpoint to its direction point determine how wide a sweep the curve makes. You can think of the lengths of these lines as representing the speed with which the curve sets off towards the direction point: the faster it goes, the further out it will curve.

This characterization of the curve is the basis for the way Bézier curves are drawn interactively. After selecting the appropriate tool (usually a pen), you click at the first endpoint, and then drag out

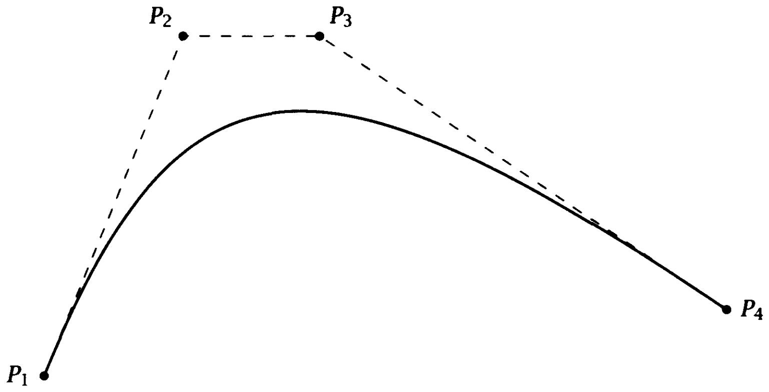


Figure 4.8
A Bézier curve

towards the first control point, as if you were pulling the curve towards it. You will usually see a *direction line* showing you how far you have pulled. In most applications, for reasons that will be explained in the next section, the direction line usually extends away from the endpoint both in the direction you drag and symmetrically in the opposite direction. Once you have the first one right, you click at the point where you want the curve to end, and drag away from the direction point (see Figure 4.9). You will see the curve being formed as you move the cursor. If you do not like the result when you have finished, you can subsequently select any of the control points and drag it around to change the shape and extent of your curve.

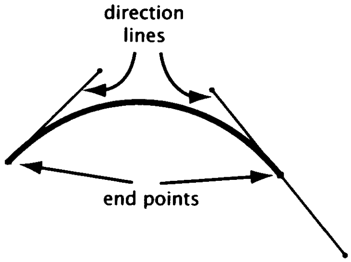


Figure 4.9
Drawing a curve with the pen tool in Illustrator

⇨ It is perhaps not immediately obvious that a curve related to four points in the manner just described is unique, or even that it always exists. Since this is not a book about the mathematics of Bézier curves, you will have to take it on trust that it is and does, or consult one of the references at the end of the chapter. However, it may help to contemplate the following construction for building curves from a pair of endpoints P_1 and P_4 and a pair of direction points P_2 and P_3.

Begin by finding the mid-points of the lines between P_1 and P_2, P_2 and P_3, and P_3 and P_4. Call these P_{12}, P_{23} and P_{34}, respectively, and construct the lines between P_{12} and P_{23} and between P_{23} and P_{34} (see Figure 4.10). Next, find the mid-points P_{123} and P_{234} of these new lines, join *them* together (Figure 4.11) and find the mid-point of this last line. This final mid-point, Q lies on the curve. (Figure 4.12.)

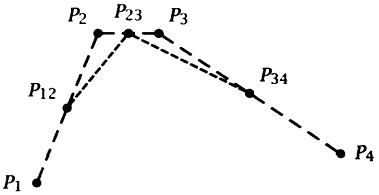


Figure 4.10
Constructing a Bézier curve, step 1

Q lies on the curve, because we assert that this construction is how our curve is to be built. We proceed by taking the two sets of four points P_1, P_{12}, P_{123} and Q, and Q, P_{234}, P_{34} and P_4, and asserting that the curves built using them as control points will be the left and right halves of our complete curve. We therefore apply the same technique of finding mid-points and so on to produce two points on the portions of the curve to the left and right of Q. This in turn leaves us with new sets of control points from which to construct the portions of the curve lying to each side of the new points we have found.

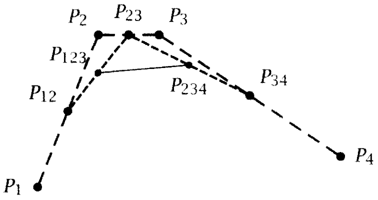


Figure 4.11
Constructing a Bézier curve, step 2

If we were mathematicians, we would go on bisecting our curve forever, until the curve segments became infinitesimally small. If you actually try this construction on paper, or write a program to perform it for you, you will find that the segments very rapidly become indistinguishable from straight lines and your curve becomes as smooth as your drawing medium or display will allow. Figure 4.13 shows how the left half of the curve produced by the construction fits between its control points the same way the complete curve fits between P_1, P_2, P_3 and P_4.

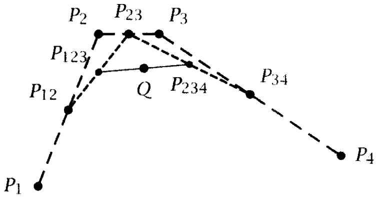


Figure 4.12
Constructing a Bézier curve, step 3

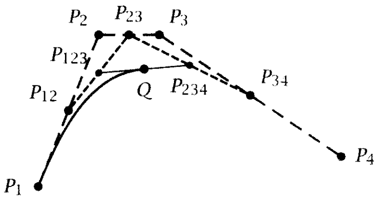


Figure 4.13
The left half of the curve

However much bisecting we do, the construction lines that pass through the endpoints P_1 and P_4 are part of the lines between P_1 and P_2 and between P_3 and P_4. This remains the case, even if we were to go on bisecting to infinity, so that, in the limit, these lines would be the tangents to the curves at P_1 and P_4. Which is to say that the curve leaves P_1 heading towards P_2, and approaches P_4 from the direction of P_3, as originally described. Furthermore, the further away P_2 is from P_1 the further away will be the midpoint, and, eventually, the curve points we construct, so that the length of the tangents controls how widely the curve sweeps round.

Mathematicians will find the preceding account a bit glib, and may feel more comfortable with an alternative description of Bézier curves.

The curves used in most drawing programs are, strictly speaking, Bézier *cubic* curves, a form of cubic with certain desirable properties. They are usually defined by the *parametric equations*:

$$\begin{aligned} x(t) &= a_x t^3 + b_x t^2 + c_x t + x_1 \\ y(t) &= a_y t^3 + b_y t^2 + c_y t + y_1 \end{aligned}$$

where the endpoint $P_1 = (x_1, y_1)$. The other control points are then given by:

$$\begin{aligned} P_2 &= (x_2, y_2) &&= (x_1 + c_x/3, y_1 + c_x/3) \\ P_3 &= (x_3, y_3) &&= (x_2 + (c_x + b_x)/3, y_2 + (c_y + b_y)/3) \\ P_4 &= (x_4, y_4) &&= (x_1 + c_x + b_x + a_x, y_1 + c_y + b_y + a_y) \end{aligned}$$

The curve defined by these four control points is the path traced out by $x(t)$ and $y(t)$ as t varies from 0 to 1. We invite you to show that

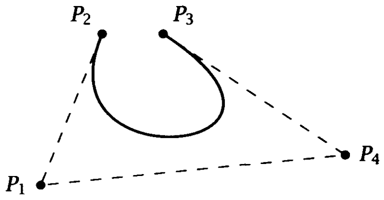


Figure 4.14
P_2, P_1, P_4, P_3

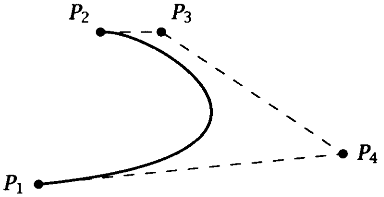


Figure 4.15
P_1, P_4, P_3, P_2

the lines P_1P_2 and P_3P_4 are the tangent vectors to the curve at its endpoints.

There is an interesting physical interpretation of this parametric formulation of the Bézier curve: if t denotes time, then $x(t)$ is a function showing how x, which could denote horizontal displacement, varies over time, while $y(t)$ describes how vertical displacement varies with time. The two equations together thus describe a two-dimensional trajectory, traced by a point over time from $t = 0$ to $t = 1$. The Bézier curve can be extended into three dimensions. In that case, the equations might describe the trajectory of, for example, a camera through a three-dimensional space. As we will see in Chapter 11, such trajectories are commonly used in three dimensional computer-generated animation.

You can construct a Bézier curve using any set of four control points, but the result is not necessarily going to be useful or lovely. Nor is it always obvious, until you have acquired some experience using these curves, exactly what the curve built from any four control points is going to look like. Figures 4.14 to 4.18 show the curves produced from the same set of points as were used in Figure 4.8, but in different orders.

Paths

A single Bézier curve on its own is rarely something you want in a drawing. Except in stylized applications for which rectangles, ellipses and straight lines serve adequately, we need to be able to draw a variety of irregular curves and shapes, such as those making up the petals and stamens of the poppy in Figure 3.2 on page 71. In principle, because the pixels on monitors and printers are finite in size, any shape, no matter how curvaceous, can be approximated as well by a collection of straight lines as by any other method. However, in practice, to produce acceptable approximations to curved shapes, we need to use a lot of very short lines. We thus lose much of the advantage of vector graphics, inasmuch as our descriptions of shapes become large, unwieldy and difficult to work with and edit interactively.

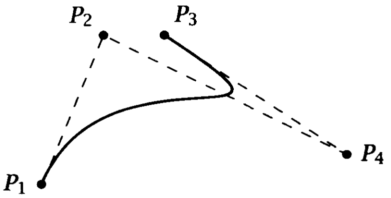


Figure 4.16
P_1, P_2, P_4, P_3

What makes Bézier curves useful is the ease with which they can be combined to make more elaborate curves and irregular shapes. Remember that a Bézier curve with control points P_1, P_2, P_3 and P_4 approaches its endpoint P_4 from the direction of P_3. If we construct a second curve with control points P_4, P_5, P_6 and P_7 (so that it is joined to the original curve at P_4), it will set off from P_4 in the direction of P_5. Provided we ensure that P_3, P_4 and P_5 are in a straight line, with P_5 on the opposite side of P_4 to P_3, the curve segments will both be travelling in the same direction through P_4, so there will be a smooth join between them, as shown in Figure 4.19. The join in Figure 4.20 is smoother still, because we have made sure that the length of the direction lines is the same on each side of P_4. If you think of the line as a trajectory through space, this ensures that it passes through the shared endpoint at a constant velocity, whereas, if we only make sure the three points are in a straight line, the direction is constant, but the speed changes.

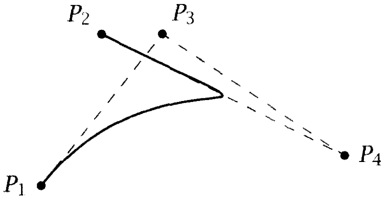


Figure 4.17
P_1, P_3, P_4, P_2

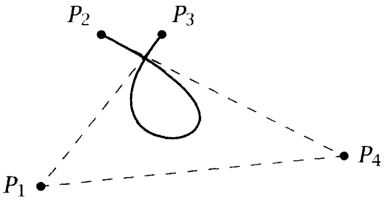


Figure 4.18
P_3, P_1, P_4, P_2

The smoothness of joins when control points line up and direction lines are the same length is the reason behind the display of direction lines in drawing programs. When you drag towards an endpoint, as we saw earlier, direction lines are displayed both to and from it. In other words, you are shown two direction points: one belonging to the curve you are just finishing, the other which could belong to a new curve which joins smoothly on to the end of it. Thus, you can rapidly build up compound curves using a sequence of dragging movements at the points where you want Bézier segments to join.

⇨ This property of continuity is not shared by other sorts of curve which might seem to be candidates for the job of curve-drawing primitive. You cannot generally make arcs of circles, parabolas or ellipses join smoothly.

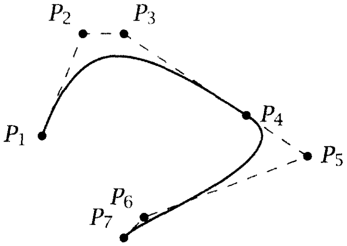


Figure 4.19
Joining two Bézier curves

Sometimes you will want your curve to change direction instead of continuing smoothly. To do so, you simply need to arrange that the direction lines of adjacent segments are not lined up (see Figure 4.21). In a drawing program, some expedient is required to break out of the default behaviour of smoothly joining curves. In Illustrator, for example, you hold down the option (or alt) key after creating the direction lines at a point; you are then able to drag the direction point for the new curve segment round to where you want it, to make an abrupt corner. By mixing clicking and dragging, it is possible to combine curves and straight lines in arbitrary ways.

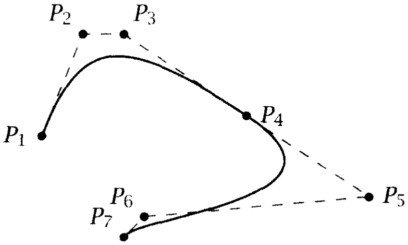


Figure 4.20
A smoother join

Some additional terminology is used to talk about joined curves and lines. A collection of lines and curves is called a *path*. If a path joins up on itself, it is said to be *closed*, otherwise it is *open*. Figure 4.23 shows a closed path; Figure 4.22 shows an open one. Open paths have endpoints, closed paths do not. Each individual

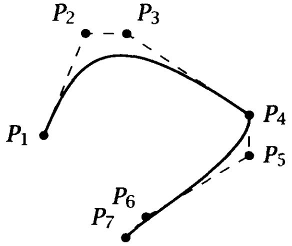


Figure 4.21
A corner point

line or curve is called a *segment* of the path; the points where segments join (the original endpoints of the segments) are called the path's *anchor points*. Note that any collection of curves and lines may be considered as a path, they do not all have to be connected.

The usual way to construct a path is by constructing the individual segments with the pen tool, which provides great control and permits you to construct very accurate curves. However, if you wish to create artwork with a hand-drawn look, you can use Illustrator's pencil tool. With this tool selected, you can just drag with the mouse or pressure-sensitive pen, as if you were drawing freehand with a pencil or pen. You are not doing so: Bézier curve segments and straight lines are being created to approximate the path your cursor follows. Once the path is complete, you can select it and see anchor points that you can adjust in the usual way. The faithfulness of the approximation can be controlled by a tolerance setting. A higher tolerance leads to a more efficient path, with fewer anchor points, which may, however, smooth out some of the smaller movements you made with the pencil tool.

Stroke and Fill

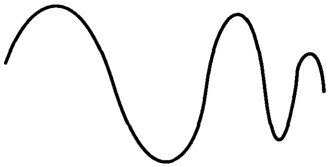


Figure 4.22
An open path

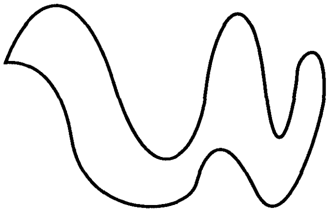


Figure 4.23
A closed path

A path, strictly speaking, is an abstract mathematical entity: just as points are infinitesimally small, so a path is infinitesimally thin. Although we have talked about drawing curves, and shown you pictures of paths, you cannot really see a path. You can, however, use it as a specification of something you *can* see. You can do this in two different ways. Either you apply a *stroke* to the path, making it visible as if you had traced it with ink or some other medium, or you treat the path as the outline of a shape, and *fill* it, as if with ink or paint. Or both. Since computer graphics is not bounded by the physical limitations of real art materials, you can stroke or fill paths with more elaborate things than flat colour, such as patterns or gradients.

⇨ Practically, you have to be able to see your path while you are drawing it, of course, so Illustrator shows you the path as a thin stroke. Once it is complete, each path segment is stroked or filled straight away, so you can see what you are doing.

Consider first applying a stroke to a path. Like physical strokes on paper, the strokes applied by a drawing program have characteristics, such as weight and colour, which determine their appearance.

These characteristics can be set and changed by the user of a drawing program. The weight is usually set by specifying the width of strokes numerically, in whatever units are convenient. Specification of colours is more complex, and is described in Chapter 6.

> ➩ As we noted in Chapter 3, some drawing programs can apply strokes that simulate natural media, such as charcoal or painted brush-strokes.

It is customary for drawing programs to support dashed strokes as well as solid ones. Ideally, the length of dashes and of the gaps between them can be specified. Again, this is usually done by the user entering numerical values in appropriate units.

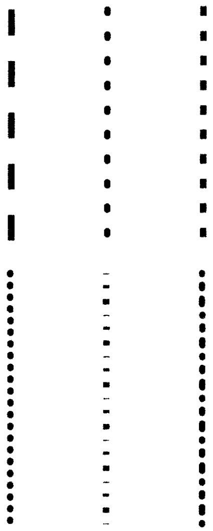


Figure 4.24
Dashed effects

A more subtle feature of strokes is the shape of their ends — the *line cap.* If a stroke has any appreciable thickness, cutting it off square at the ends with a *butt cap* may produce an undesirable and ugly effect. It may be preferable to use a *round cap,* where the line is finished off with a filled-in semicircle built across its end. A third option is to use a *projecting cap,* with the stroke continued beyond the endpoint of the path by half the width, so that the weight of the stroke relative to the path is the same in all directions. These three line cap options are provided by PostScript, and are consequently supported by drawing programs like Illustrator that produce PostScript output. Combining different line caps with dash patterns provides a range of effects, as shown in Figure 4.24.

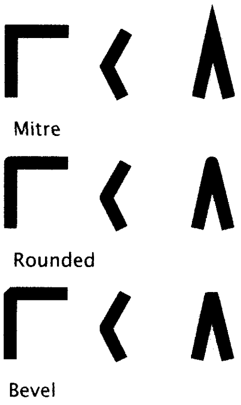


Figure 4.25
Joining styles

Joins at corner points also need consideration, because wide lines can only meet cleanly if they do so at 90°; when they meet at any other angle an unsightly gap or overlap will result. This can be removed in several ways. The three styles of line join provided by PostScript are a *mitre* — as in a picture frame, the outside edges of the lines are extended to meet at a point; *round* — a circular arc is used to produce a rounded corner; and *bevel* — the segments are finished off square where they join, and the resulting notch is filled in with a triangle to produce a flat ended joint. If mitred joins are used on segments that meet at a very narrow angle, long projecting spikes will result. To avoid this, a limit can be specified, and if the ratio of the spike's length to the stroke width exceeds it, mitres will be replaced by bevels. Figure 4.25 illustrates the different joining styles.

As well as applying a stroke to a path, you can use it as an outline and fill it. Astute readers will observe that you can only fill a closed path, but most drawing programs also allow you to fill an open path — the filling operation implicitly closes the path with a straight line between its endpoints.

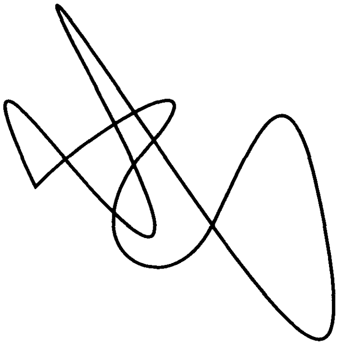


Figure 4.26
A complex path...

The simplest fill is a single colour, and this is often all that is required. When a fill (or, indeed, a stroke) is applied it completely obscures anything underneath it. There is no mixing of overlaid colours, as you would expect if you were using watercolour paints, for example. This means that, among other possibilities, you can use a shape filled with the background colour to knock out areas of objects underneath it.

More interesting and sometimes more attractive effects can be produced by using *gradient fills* and *patterns.*

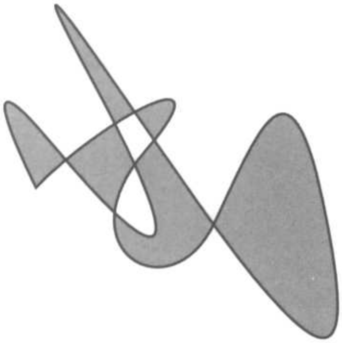


Figure 4.27
...filled

Plate 5 shows two examples of gradient fills. This type of fill is characterized by a gradual transition between colours or tones. In the simplest case — a *linear* gradient — the colours at each end of a region are specified, and a smooth blend of intermediate colours is generated in between. Illustrator provides controls to let you specify intermediate colours, adjust the mid-point of the gradient (where the two colours being blended are of equal intensity) and the line along which the gradient varies. An alternative form of blending, also shown in Plate 5, has the colour varying outwards from a centre point to the outside of the fill. This is called a *radial* gradient. The more sophisticated gradients used in the Plate 6 were created using Illustrator's gradient mesh tool, which allows the designer or artist to set up a two-dimensional mesh of points, and specify the colours at each mesh point; the colours are blended in the spaces between the mesh points. The shape of the mesh can be adjusted dynamically until the desired effect is achieved.

Gradient fills are very widely used in artwork created in vector drawing programs, and contribute to the characteristic air-brushed look of much of the graphic design that is produced using these programs.

Pattern fills, as the name suggests, allow you to use a repeating pattern to fill in an area, as shown in Plate 7. Patterns are built out of elements called *tiles.* A tile is just a small piece of artwork, made using the facilities provided by your drawing program (possibly including the facility to import bitmaps as objects). The name embodies the analogy normally employed for describing how an area is filled with a pattern. Imagine that the artwork is rendered onto a rectangular ceramic tile, such as you might use in your bathroom. Copies of the tile are arranged in rows and columns parallel to the x- and y-axes, butted together as if by a skilled tiler. The resulting pattern is clipped to the area being filled. Constructing tiles that join seamlessly requires a certain amount of skill — in graphics as in bathroom design. Tiles may be used to produce geometrically patterned areas, such as might be appropriate for drawing textiles or wallpaper (or bathrooms), but they can also be designed so that they produce textured effects. Pattern fills are often used as backgrounds.

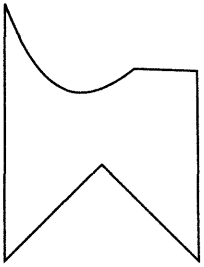


Figure 4.28
A simple object...

Some drawing programs allow you to use patterns to stroke paths, producing a textured outline, for example. This is more difficult than using patterns as fills, because one naturally wants the tiles to be arranged perpendicular to the path, not horizontally. This in turn makes it difficult to get tiles to go round corners, so that a pattern intended for tiling a path must include special corner tiles.

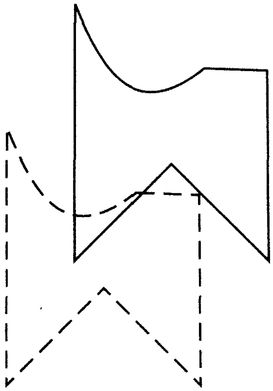


Figure 4.29
...translated...

If you want to fill a path, you need to know which areas are inside it. For simple shapes, this is a trivial question, but a path may be arbitrarily complex, crossing itself several times. Figure 4.26 shows a single path. Which areas are inside it, and which outside? There is no absolute answer; several different interpretations of the concept of insideness have equal validity. Figure 4.27 shows Illustrator's answer, which is based on the *non-zero winding number rule*, which may be expressed as an algorithm as follows: To determine whether a point is inside a path, draw a (conceptually infinite) line from the point in any direction. Start by setting the winding number to zero. Follow the constructed line, and every time the path crosses it from left to right, add one to the winding number; every time the path crosses from right to left, subtract one from the winding number. After all crossings have been counted, if the winding number is zero, the point is outside the path, otherwise it is inside. Note that this algorithm depends on the path's having a direction, which will depend on the order in which anchor points were added to it.

Transformations and Filters

The objects that make up a vector image are stored in the form of a few values that are sufficient to describe them accurately: a line by its endpoints, a rectangle by its corners, and so on. The actual pixel values that make up the image need not be computed until it is

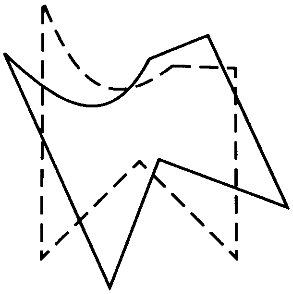


Figure 4.30
...rotated...

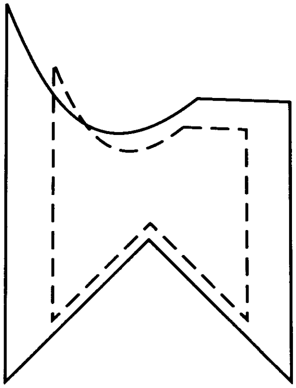


Figure 4.31
...scaled...

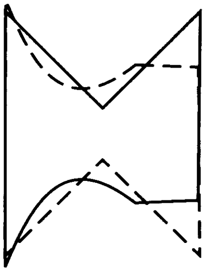


Figure 4.32
...reflected...

displayed. It is easy to manipulate objects by changing these stored values. For example, if a line runs parallel to the x-axis from $(4, 2)$ to $(10, 2)$, all we need do to move it up by 5 units is add 5 to the y-coordinates of its endpoints, giving $(4, 7)$ and $(10, 7)$, the endpoints of a line running parallel to the x-axis, but higher up. To use a term we introduced earlier in this chapter, we have transformed the image by editing the *model* that is stored in the computer.

Only certain transformations can be naturally produced in this way. The most important are *translation* — a linear movement of the object — *scaling, rotation* about a point, *reflection* about a line, and *shearing* — a distortion of the angles of the axes of an object. These transformations are illustrated in Figures 4.28 to 4.33. Any modern drawing program will allow you to perform these transformations by direct manipulation of objects on the screen. For example, you would translate an object simply by dragging it to its new position.

Briefly, the operations on the model which achieve these transformations are as follows. Any translation can be done by adding a displacement to each of the x and y coordinates stored in the model of the object. That is, to move an object Δ_x to the right and Δ_y upwards, change each stored point (x, y) to $(x + \Delta_x, y + \Delta_y)$. Negative Δs move in the opposite direction. Scaling is performed by multiplying coordinates by appropriate values. Different factors may be used to scale in the x and y directions: to increase lengths in the x direction by a factor of s_x and in the y direction by s_y, (x, y) must be changed to $(s_x x, s_y y)$. (Values of s less than one cause the object to shrink.) Thus, to double the size of an object, its stored coordinates must be multiplied by two. However, this has the effect of simultaneously displacing the object. (For example, if a unit square has its corners at $(1, 2)$ and $(2, 1)$, multiplying by two moves them to $(2, 4)$ and $(4, 2)$, which are the corners of a square whose side is of length 2, but it is now in the wrong place.) To scale an object in place, the multiplication must be followed by a suitable, easily computed, displacement to restore it to its original position.

Rotation about the origin and reflection about an axis are simple to achieve. To rotate a point (x, y) around the origin in a clockwise direction by an angle θ, you transform it to the point $(x \cos\theta - y \sin\theta, x \sin\theta + y \cos\theta)$ (which you can prove by simple trigonometry if you wish). To reflect it in the x-axis, simply move it to $(x, -y)$; in the y-axis, to $(-x, y)$. Applying these operations to all the points of an object will transform the entire object. The more general operations of rotation about an arbitrary point and reflection in an arbitrary line require more complex, but conceptually simple, transformations. The details are left as an exercise.

Finally, when an object is sheared, it is as if we took the x-axis and skewed it upwards, through an angle α, say, and skewed the y-axis through an angle β (see Figure 4.34). You can show that the transformation can be achieved by moving (x, y) to $(x + y \tan\beta, y + x \tan\alpha)$.

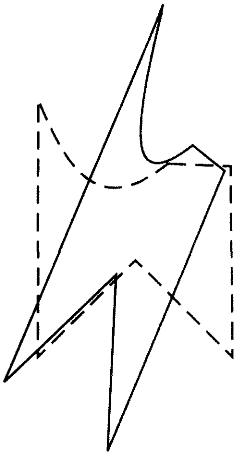


Figure 4.33
...sheared

Other, less structured, transformations can be achieved by moving (i.e., changing the coordinates of) the anchor points and control points of paths. This would normally be done by interactive manipulation in a drawing program. Anchor points and control points may also be added and deleted from paths, so that a designer or artist can fine-tune the shape of objects.

Some commonly required effects which fall between the highly structured transformations and the free manipulation of control points are provided as *filters* in Illustrator and similar programs. (The term 'filter' is taken from photography, where optical filters are used to produce similar effects.) An object is selected, and a filter operation is chosen from a menu of those available. The chosen effect is then applied to the selected object. Filters available in Illustrator include roughening, which produces a rough edge to an object by moving the anchor points of its path in a jagged pattern, scribbling, which moves the anchor points in a random fashion, and rounding corners, which converts corner points into smooth curves. Most filters are parameterized in values such as the maximum distance an anchor point may be moved. The parameters can be set by the user via controls such as sliders.

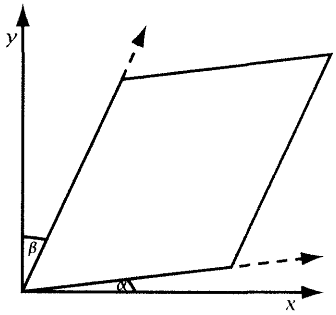


Figure 4.34
Skewed axes

The important thing to understand about all these transformations is that they are achieved simply by altering the coordinates of the defining points of objects, altering the stored model using nothing but arithmetical operations which can be performed efficiently. Although every pixel of the object must be transformed in the final displayed image, only the relatively few points that are needed to define the object within the model need to be re-computed beforehand. All the pixels will appear in the desired place when the changed model is rendered on the basis of these changed values.

3-DGraphics

Pictures on a screen are always two-dimensional, but that doesn't mean that the models from which they are generated need to be restricted to flat two-dimensional shapes. Models of three-dimensional objects correspond more closely to the way we perceive space. They enable us to generate two-dimensional pictures as perspective projections — or perhaps other sorts of projection — onto a plane, as if we were able to photograph the model. Sometimes, this may be easier than constructing the two-dimensional image from scratch, particularly if we can begin with a numerical description of an object's dimensions, as we might if we were designing some mechanical component, for example, or constructing a visualization on the basis of a simulation. A three-dimensional model allows us to generate many different images of the same objects. For example, if we have a model of a house, we can produce a view of it from the front, from the back, from each side, from close up, far away, overhead, and so on, all using the same model. If we were working in only two dimensions, each of these images would have to be drawn separately. Working in three dimensions really begins to open up new creative possibilities when we start to generate a sequence of images as we move models and cameras in space, as we will see in Chapter 11.

3-D graphics, as we call vector graphics based on three-dimensional models, is a complicated subject, though, requiring tools that are hard to master, and should be left to specialists most of the time. Here, we can only outline the main features of 3-D technology, in order to provide an appreciation of the difficulties it presents and the opportunities it has to offer.

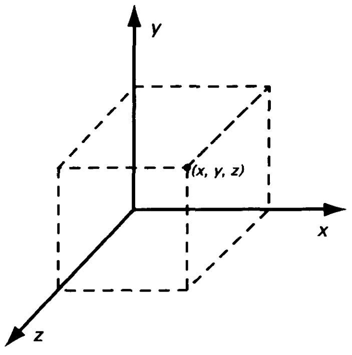


Figure 4.35
Axes in three dimensions

In abstract mathematical terms, generalizing coordinate geometry from two dimensions to three is straightforward. The x- and y-axes of a two-dimensional coordinate system are perpendicular to each other. If you imagine drawing a set of axes on a flat sheet of paper and pinning it to a vertical wall, you can see that we can place a third axis perpendicular to the other two, coming out of the paper horizontally. Just as we can use x- and y coordinates to define a point's horizontal and vertical distance along the wall from the origin, we can use a z-coordinate, measured along our third axis, to define its distance from the wall. The three coordinates together define a point in a three-dimensional space (see Figure 4.35).

The primitive geometrical shapes of 2-D graphics are replaced by 3-D objects: instead of a circle, we have a sphere, instead of a square, a cube, and so on. By an obvious extension, a three-dimensional vector defines a displacement in the same way the two-dimensional vectors we used earlier did. This allows us to describe translations of objects in three-dimensional space. We certainly have a basis for 3-D vector graphics.

Even at this stage, it is apparent that three dimensions may be more than one and a half times as complicated as two. Consider rotation. In two dimensions we rotate about a point; in three, we must rotate about a line. Rotations about an arbitrary line can be built out of rotations about the axes, but that still leaves three distinct rotations to consider, as shown in Figure 4.36, which introduces the names used for each of them.

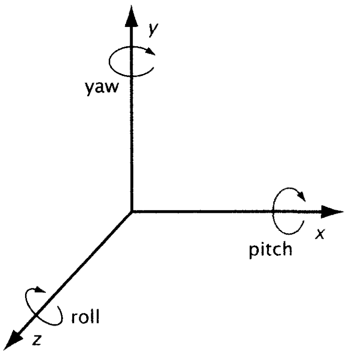


Figure 4.36
Rotations in three dimensions

⇨ When we introduced a third dimension, we stated that the z-axis points out of the imaginary wall of the x-y plane. Why should it not point into it? There is no reason. The coordinate system we have adopted is no more than a convention. It is known as a *right-handed* coordinate system, a name that Figure 4.37 should explain. A left-handed system is equally valid, and is sometimes used in computer graphics. However, the right-handed system is more widely used, especially by mathematicians, and we will employ it exclusively.

We have also adopted the convention that the vertical axis is labelled y, as it was before, but some systems use the x- and y-axes to define a horizontal *ground plane*, with the z-axis adding height to it. In that case, the names of the three rotations are assigned differently to the three variables, although they retain the same spatial meaning — roll is always a movement around the front-to-back axis, and so on.

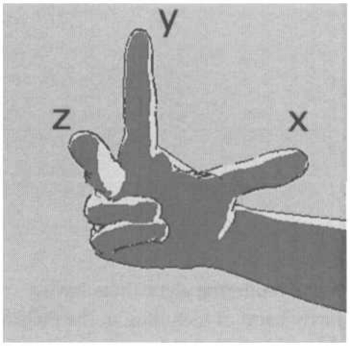


Figure 4.37
Right-handed coordinate system

The added complexity of graphics employing the third dimension goes much further than an extra couple of rotations. Instead of defining shapes by paths, we must define objects by surfaces, which require more complicated mathematics and are much harder for most people to visualize. The fact that we can only work with two-dimensional representations while building models makes visualization generally difficult. This problem is further exacerbated by the fact that only very powerful workstations can provide real-time rendering of 3-D models, so it is usually necessary to work with cruder approximations during the design process.

Once a collection of objects has been modelled, they are arranged in space, usually interactively. Often, a complete scene will be constructed from a few objects that are specially modelled and others taken from a library, either developed in earlier projects or bought in as the 3-D equivalent of clip art. (Indeed, some 3-D applications intended for consumer or corporate use allow you to do little more than arrange ready-built models in space.)

As well as the spatial relationships between separate objects, we may also wish to consider the relationships between objects that form parts of a larger object. Most complex objects have a hierarchical structure. They can be described as a collection of sub-objects, each of which might be a collection of sub-sub-objects, and so on, until we reach the smallest component objects, which can be modelled in a simple way. For example, a bicycle consists of a frame, two wheels, a saddle, handlebars and front fork. Each wheel is a tyre around a rim, with spokes and a hub. We could model the hub as a cylinder, without decomposing it further. Hierarchical modelling is a well-established way of coping with complexity in many fields. In 3-D it becomes of particular importance when objects are animated, because we can then take advantage of the relationships between components to deduce how the object must move as a whole.

Rendering is no longer a relatively simple matter of generating pixels from a mathematical description. In 3-D, we have a mathematical model of objects in space, but we need a flat picture. Assuming that we want to use conventional Renaissance perspective to project the 3-D model onto a plane surface, we will need to consider the viewpoint, or the position of an imaginary camera, and ensure that the rendered picture corresponds to what the camera sees,[1] while exhibiting the scaling with distance that we expect from perspective.

We also need to consider lighting — the position of light sources, and the intensity and type of illumination they cast, whether it is a diffuse glow or a concentrated beam, for example. The interaction of the light with the surface of objects, and perhaps — in the case of an underwater scene or a smoke-filled room — with the atmosphere must also be modelled and rendered. If we want to achieve any sort of realism, our models must include not just the geometrical features of objects, but also their surface characteristics — not just the colours, but also the texture — so that the surface's appearance can be rendered convincingly in different positions and under different lighting conditions. Although 3-D systems are reasonably good at using lighting models derived from the underlying physics, they are not perfect, and designers sometimes have to resort to

[1] Naïve rendering algorithms have a nasty habit of including in the picture things that are behind the camera or on the hidden side of objects.

physical impossibilities, such as negative spotlights, for absorbing unwanted light.

These added complications mean that the theory of 3-D graphics is a great deal more elaborate than that of 2-D graphics. It also means that 3-D software is more complex and difficult to use. Finally, it means that rendering 3-D models can be an extremely computationally expensive process, that often requires additional hardware (often in the form of 3-D accelerator PCI cards) to achieve acceptable performance on desktop machines.

3-D Models

Broadly speaking, there are three general approaches to modelling objects, which are often used in conjunction. The simplest approach, which goes by the name of *constructive solid geometry*, uses a few primitive geometric solids, such as the cube, cylinder, sphere and pyramid, as elements from which to construct more complex objects. These elements can be distorted, by squashing or stretching, to produce variations on the simple forms. They can also be combined, using operations usually described in terms of the set theoretical operators union, intersection, and difference.

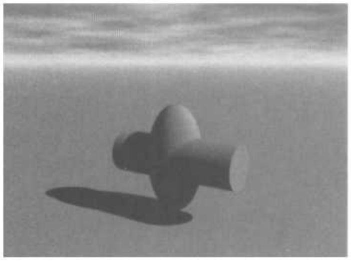


Figure 4.38
Union

These operations only do anything to two objects that are placed in such a way that they share some of the space that they occupy — normally a physical impossibility, but no problem for a computer model. Their *union* is a new object made out of the space occupied by the two together. Figure 4.38 shows the object formed from the union of a horizontal cylinder and a vertical ellipsoid.[2] The *intersection* of two objects is the space that the two have in common; Figure 4.39 shows what happens when the same cylinder and ellipsoid are intersected: the shape that is produced is otherwise difficult to describe. Finally, the *difference* of two objects is the space occupied by the first but not the second. This operation is useful for knocking holes out of solid objects, as Figure 4.40 shows.

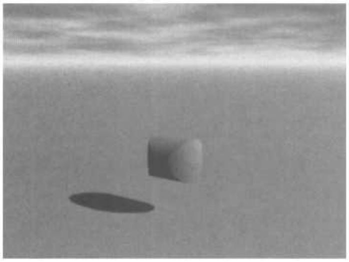


Figure 4.39
Intersection

Constructive solid geometry is especially useful for modelling man-made objects and architectural features, which are often built out of the same elementary solids. It is often found in computer-aided design systems. It can take you a long way with modelling camshafts, triumphal arches and toy steam trains. However, as a glance around your immediate environment will show, many objects

[2] The shadow and background are just cosmetic additions.

in the real world are not made out of geometric solids combined by set operations; these need a different approach.

Free form modelling uses a representation of an object's boundary surface as the basis of its model. This approach is a 3-D generalization of the use of paths to enclose shapes in two dimensions. Instead of building paths out of lines and curves, we must build surfaces out of flat polygons or curved patches. Surfaces constructed as a *mesh* of polygons can be rendered relatively efficiently, making this a popular representation.[3] However, they suffer from the drawback that, like straight line segments used to approximate a curve, polygons cannot fit together smoothly when they are used to approximate curved surfaces. This is more serious than it may sound, because it affects the way light is reflected off the surface. If reflections are broken up, any irregularities will be readily apparent.

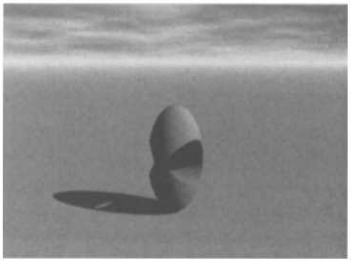


Figure 4.40
Difference

➲ It is possible to generalize Bézier curves to three-dimensional surfaces in order to produce curved patches that can be fitted together smoothly. A cubic Bézier surface patch requires sixteen control points, instead of the corresponding curve's four. Just as we could join curves together smoothly by ensuring that they meet at an anchor point and the connected control points are in a straight line, so we can join patches by ensuring that they meet at a common edge curve and that the connected control points lie in the same plane. This is easy to say, but Bézier patches are hard to work with in interactive 3-D applications, largely because moving a single control point can affect the geometry of the whole patch in a way that may be hard to predict. A more tractable kind of patch is based on surfaces called *non-rational B-splines* or *NURBs*, which, by using a more complicated construction, ensure that the effect of moving control points is localized. In effect, this makes it possible to sculpt a smooth surface by pulling and pushing it into the desired shape. NURBs are largely confined to high-end systems.

The generality and lack of structure of boundary representations can make it hard to get started with a model. To overcome this problem, most 3-D programs provide a means of generating objects with a certain regular structure or symmetry from 2-D shapes. The resulting objects can then either be used directly, or their surface mesh can be altered to produce a less constrained object.

The basic idea is to treat a two-dimensional shape as a cross section, and to define a volume by sweeping the cross section along a path. The simplest path is a straight line. A shape creates an object

[3] Where fast rendering is required, as in 3-D games, the polygons are often restricted to triangles, which makes rendering more efficient. It also guarantees that the polygons are flat.

with a uniform cross section as it travels along a straight line. For example, a circle creates a cylinder in this way. This process is known as *extrusion*, since the objects it produces resemble those that can be made by industrial processes in which plastic or metal is forced through an opening. Extruded text is an application of this technique that has been so widely used in producing corporate logos as to have become a cliché. To produce more elaborate objects, a curved path can be used, and the size of the cross section can be altered as it moves along it. If the path is a conventional Bézier path, organic shapes can be generated. If it is a circle, the resulting objects exhibit radial symmetry. If a suitable shape is chosen, circular paths can be used to generate many types of drinking vessel and vase, as well as mechanical components. Because of the resemblance of the resulting objects to traditional turned artefacts, this special case is often called *lathing*. Figure 4.41 shows some simple objects constructed by extrusion and lathing.[4]

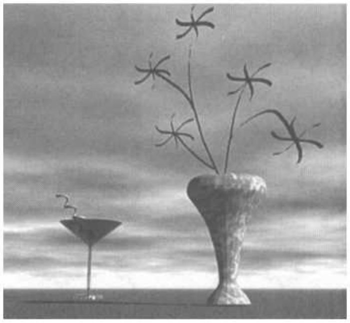


Figure 4.41
Lathed and extruded objects

The third appraoch to modelling is *procedural modelling*. Here, instead of using models that can be described by equations, and storing only the constants that define a particular instance, we use objects that are described by an algorithm or procedure. Thus, returning to two dimensions for a moment, instead of defining a circle by the equation $x^2 + y^2 = r^2$, we could define it by some procedure such as 'draw a curve that maintains a constant distance r from the origin'. In this case, the procedural representation is not very helpful — the equation tells you how to implement the procedure — but for naturally occurring objects with a more complex, less mathematically tractable, structure, algorithms may provide a description where equations cannot.

The best known procedural modelling techniques are based on *fractals*. These are often described as shapes that exhibit the same structure at all levels of detail. Figure 4.42 shows a famous example of such a shape, and its construction. We start with the shape at the top, consisting of four equal line segments, arranged as a straight line with a bump in the middle. We then replace each segment by a scaled down copy of the entire shape, as shown on the second line. We continue in the same way, replacing each of the segments of each of the scaled down copies with a copy of the original shape scaled down further, and so on. The bottom two lines show later stages in the procedure. You can imagine that, if we were able to continue in this way to infinity, we would end up with a crinkly curve, with a bulge in the middle and two lesser crinkly bulges to each side. If

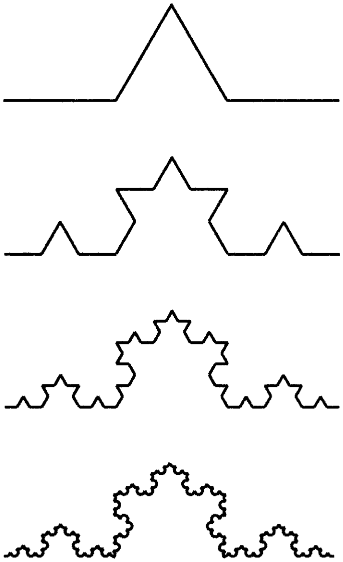


Figure 4.42
Constructing a well-known fractal curve

[4] The flowers are slightly extruded from a drawing you can find in Chapter 13. This is enough to make them into objects that can be positioned in three dimensions.

you were then to magnify any part of the curve and look at it you would see a crinkly curve, with a bulge in the middle and two lesser crinkly bulges to each side, which, if you were to magnify any part of it . . .

The appearance of certain natural features, such as coastlines, mountains and the edges of clouds, approximates this property: their small scale structure is the same as their large scale structure. Figure 4.43 shows that three of the curves we just constructed can be put together to make a snowflake. Fractals can be extended to three-dimensional structures that exhibit the same sort of similarity at different scales. Whereas this particular sort of structure cannot be described by equations, it is, as we have just demonstrated, easily described by a recursive algorithm.

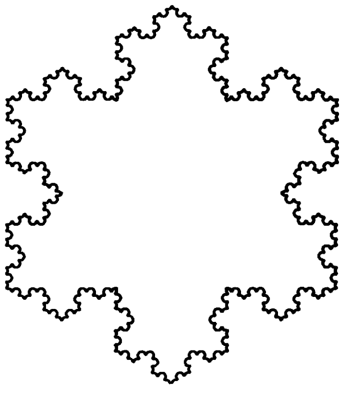


Figure 4.43
A fractal snowflake

Where fractal algorithms are used to model natural phenomena, an element of randomness is usually introduced. For example, a very simple fractal might be made by splitting a line into two parts, moving the centre point a certain distance, and then applying the construction recursively to the two halves. If the distance moved is not a constant proportion of the length of the segment but a random distance, the resulting shape will still display a recognizable similarity at different scales, without being exactly internally replicated. Figure 4.44 shows a curve being constructed in the manner just described; it could be said to resemble a mountain slope. In three dimensions, a similar construction can be applied to construct terrains out of internally sub-divided triangular areas. By joining the mid-points of the three sides of a triangle, it can be divided into four smaller triangles. The mid-points can be randomly displaced perpendicularly to the plane of the original triangle, as indicated in Figure 4.45, and then the construction can be applied to each of the small triangles. This process can be applied to arbitrary polygons, and repeated indefinitely to produce arbitrarily fine detail with the irregular appearance of natural terrain, as exhibited by the landscape in Figure 4.46.

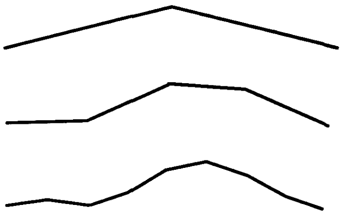


Figure 4.44
Constructing a fractal mountainside

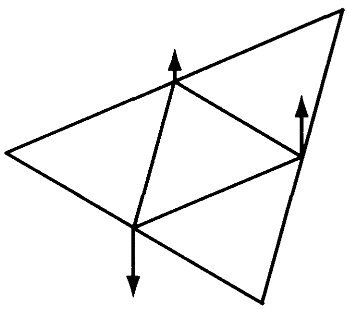


Figure 4.45
3-D random fractal construction

Note that the structures used in this sort of modelling are not, strictly mathematically speaking, fractals. A mathematician's fractal shows the same structure at *all* scales, right on to infinity. Fractals are said to be *self-similar*. In computer graphics, we cannot do this, nor do we want to: we can't show any detail smaller than a single pixel, and in any case, to model natural features we must introduce a random element. The randomized structures commonly used in modelling are a finite approximation to a class of structures

> known as *random fractals*, which have the property that at any scale the components have the same statistical distribution as the whole structure. This property is a random equivalent of the property of true fractals that at any scale the components are identical to the whole. To ensure such *statistical self-similarity*, constraints must be placed on the random factors introduced at each stage of the construction of these objects.

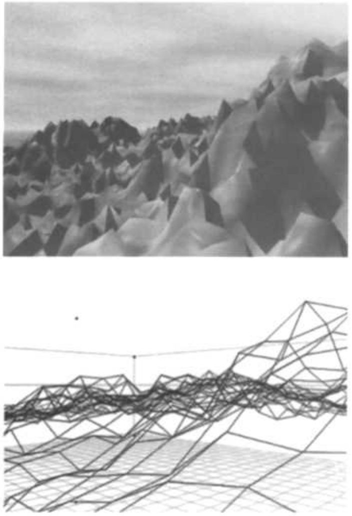


Figure 4.46
Fractal terrain

Two other procedural modelling techniques deserve a brief mention. *Metaballs* are sometimes used to model soft objects. Metaballs are just spheres, but their interaction is modelled using the same sort of mathematics as physicists use to model the electric fields around charged spheres. When two metaballs are placed close together, the fields coalesce into a composite field with a smooth boundary (see Figure 4.47). In 3-D graphics, this boundary is used as the surface of a composite shape. Complex objects can be built by sticking metaballs together. The resulting shapes have a soft, organic quality that is difficult to achieve using other methods of modelling. The process of constructing objects from metaballs somewhat resembles modelling using clay, so tools based on this approach can have a more intuitive feel than more traditional, geometry-based, systems. The fields around other shapes besides spheres can be modelled in a similar way, to allow a wider range of soft objects to be constructed easily. These models count as procedural ones because the disposition of the surfaces is computed algorithmically from the positions of the balls; there is no need to shape the actual surface, in the way polygon meshes must be shaped by the designer.

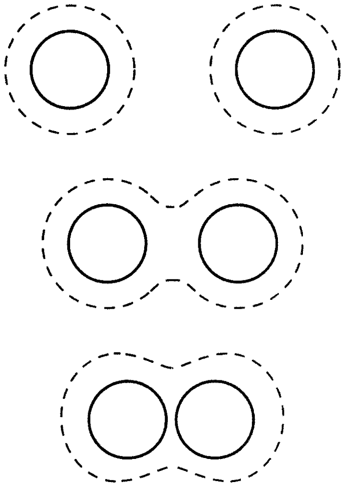


Figure 4.47
Coalescing fields around metaballs

None of the techniques we have described so far is particularly suitable for modelling phenomena such as rain, fountains, fireworks, or grass, consisting of a large number of semi-independent elements, with some shared properties. Often, the elements are moving, and the laws of motion impose certain patterns on them, as in the case of a fountain. To model such things by placing individual drops of water, blades of grass, sparks, and so on, calls for a dedication to detail, and an amount of time, that many 3-D graphic artists lack. *Particle systems* allow features made out of many particles to be specified in terms of a few parameters, from which the positions of the individual particles can be computed algorithmically. Particle systems were first used for film special effects[5] but are now available, at varying levels of sophistication, in desktop 3-D systems.

[5] Their first reported use was in creating the 'Genesis effect' in *Star Trek II: The Wrath of Khan.*

The modelling technique used in particle systems leads on to the ultimate procedural modelling system: physics. Established modelling techniques are primarily based on the desired appearance of the objects. If, instead, we base our models on the physical characteristics of objects — their mass and its distribution, elasticity, optical properties, and so on — the appearance can be deduced using physical laws. Such physical models could be tremendously powerful, since a single description of an object would suffice to determine its appearance in any position or environment. Combined with the laws of motion, physical models could be used to describe moving objects, and so to generate animation. Unfortunately, although physicists feel able to construct models of the evolution of the entire universe, modelling such things as the way clothing drapes over a figure is beyond current theories. A great deal of research has been carried out in recent years, particularly into modelling textiles, and it is beginning to bear fruit, but it requires intensive computation and its use is still confined to research laboratories and high-budget film production.

Rendering

3-D models only exist inside computers, all we can see is two-dimensional images generated from them.[6] This is the case both when we wish to produce a final image from a model, and when we are working on a model in a 3-D application. In both cases, a rendering operation is needed; in the latter case, this must be done rapidly enough to provide visual feedback. As we will shortly see, rendering high quality images is time consuming, so it is usually necessary to compromise on quality while a model is being built. Because of the complexity of rendering, and its computational demands, it is common for it to be handled by a specialized module, usually referred to as a *rendering engine*. It may be the case that a rendering engine is optimized for multiprocessing, so that rendering can be speeded up by using a collection of processors operating in parallel.

Almost all contemporary 3-D graphics aspires to the sort of realism associated with photographs. That is, it uses Renaissance perspective conventions (distant objects are smaller than near ones) and attempts to represent the effects of light on different surfaces in a way that is consistent with the laws of optics. The mathematics of perspective projection has been understood for a long time, so it

[6] Research work is being done into generating 3-D holographic projections directly from models, but it still has a long way to go.

is relatively easy for a rendering engine to work out which points on an image plane correspond to each of the nodes of a 3-D model. By joining these points to show the edges of objects and of the polygons making up surfaces, it is possible to produce a *wire frame* image of a model, as shown in Figure 4.48. Wire frames are often used as preview images in modelling programs. They can also be useful in computer-aided design systems, since they contain enough information to provide answers to questions such as 'Does the door provide adequate clearance for the piano when it is fully open?' For most purposes, though, they are not acceptable.

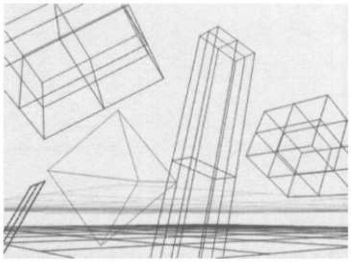


Figure 4.48
Wire frame rendering of objects

Wire frames' most noticeable feature is the absence of any surfaces. This has the obvious disadvantage that no surface detail can be shown, and the less obvious disadvantage that they do not contain enough information to enable us to determine objects' orientations unambiguously — in Figure 4.48, can you determine the form and orientation of every object, and which of their vertices is nearest to you? (Look at Plate 8 to check your answers.) In order to show the surfaces of objects, it is necessary to determine which of them are visible. The far sides of objects are considered hidden in perspective views[7], as are surfaces, or parts of surfaces, with other objects in front of them. Determining which surfaces are visible, starting only from a model based on the coordinates of corners of objects, is not a trivial task, but it is one that has received attention almost from the beginning of computer graphics. There are several tried and tested algorithms for performing *hidden surface removal.* Once this has been done, the next task is to determine how to render the visible surfaces.

One answer is to colour them arbitrarily, ensuring that no two adjacent faces are the same colour. This approach makes no pretence at photographic realism, but ensures that the shapes of objects are clearly visible. An alternative is to assign a colour to each object, and render its entire surface in that colour. This is more consistent with our expectations about surfaces, but leaves some visual ambiguity. To resolve this ambiguity, and to begin to produce images that resemble photographs, we must take account of the way light interacts with surfaces. This means that our models must include sufficient information about lights and surface properties. In most 3-D modelling programs, surface characteristics can be associated with an object by setting the values of some parameters, including its colour and reflectivity. (Some systems allow the designer to paint on the surface of models in order to specify

[7] One could envisage a cubist projection in which this was not the case.

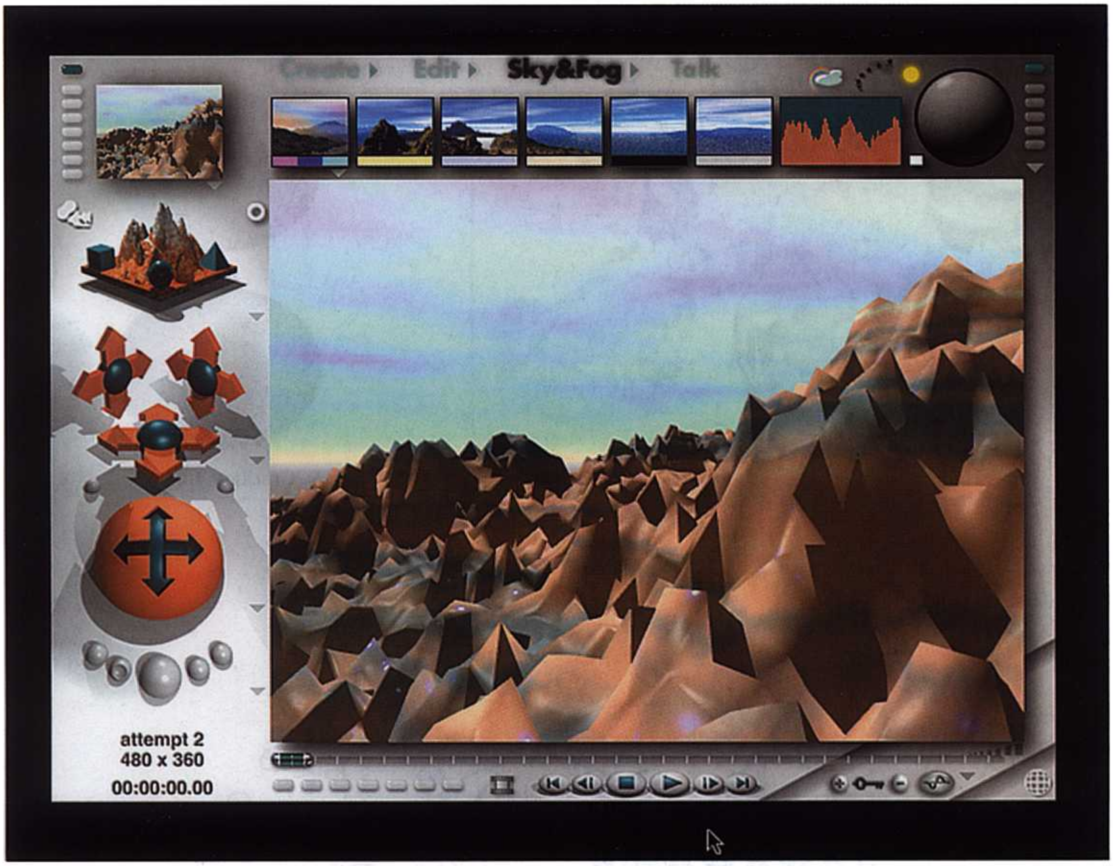
their surface characteristics indirectly.) Lights can be created and positioned in the same way as other objects; the designer merely selects a type of light — spotlight or point source (such as the sun), for example — and specifies its position and intensity. It is up to the rendering engine to produce an image that is consistent with the lights and materials that have been used in the model.

Different algorithms — usually called *shading algorithms* — are employed for this task, each incorporating different models of how light interacts with surfaces,[8] and each using different approximations to allow rendering to be performed with reasonable efficiency. Rendering engines usually allow you to choose the most appropriate shading algorithm for the needs of a particular model. There is usually a trade-off between the final quality of the image — or at least, of its treatment of light — and the amount of computation required.

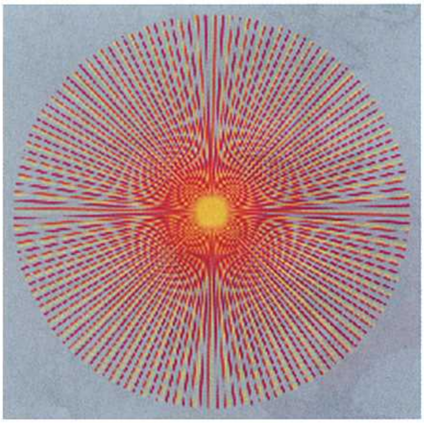
The simplest way of shading an object whose surface is made out of polygons is to calculate a colour value for each polygon, based on the light striking it and its own optical properties, and use that value to colour the whole polygon. Adjacent polygons will often have different colours — they may be at a different angle to the light source, for example — and, as we mentioned earlier, this will make the discontinuities between polygons readily visible. To disguise this effect, and to produce a more convincing rendering of the polygons, more sophisticated algorithms interpolate the colour across each polygon, on the basis of colour values calculated at the vertices. One such interpolation scheme is *Gouraud shading*, named after its inventor. An alternative, which does the interpolation differently, is *Phong shading*. Phong's approach works better when the illumination model being used takes account of *specular reflection* — the light that bounces off the surface of shiny (or partially shiny) objects. The resulting highlights are rendered by Phong shading in a quite convincing way.

The shading algorithms mentioned so far only deal with each object in isolation, but in reality, the appearance of an object may be affected by the presence of others. As an extreme example, the appearance of a glass full of water will depend on what is behind it, while that of a mirror will depend on the objects it reflects. Generally, light may bounce off several objects before reaching the eye, and be influenced by their colour and surface. *Ray tracing* is a shading algorithm that attempts to take account of this. It works by tracing the path of a ray of light back from each pixel in the the

[8] The models of illumination are loosely derived from the physics, but also incorporate heuristics aimed at producing an acceptable result and not an optical simulation.



1. An unconventional user interface (Bryce)

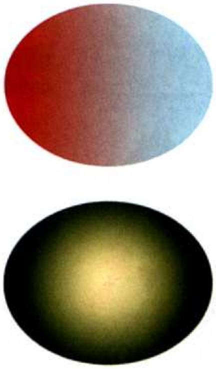


2. Moiré patterns

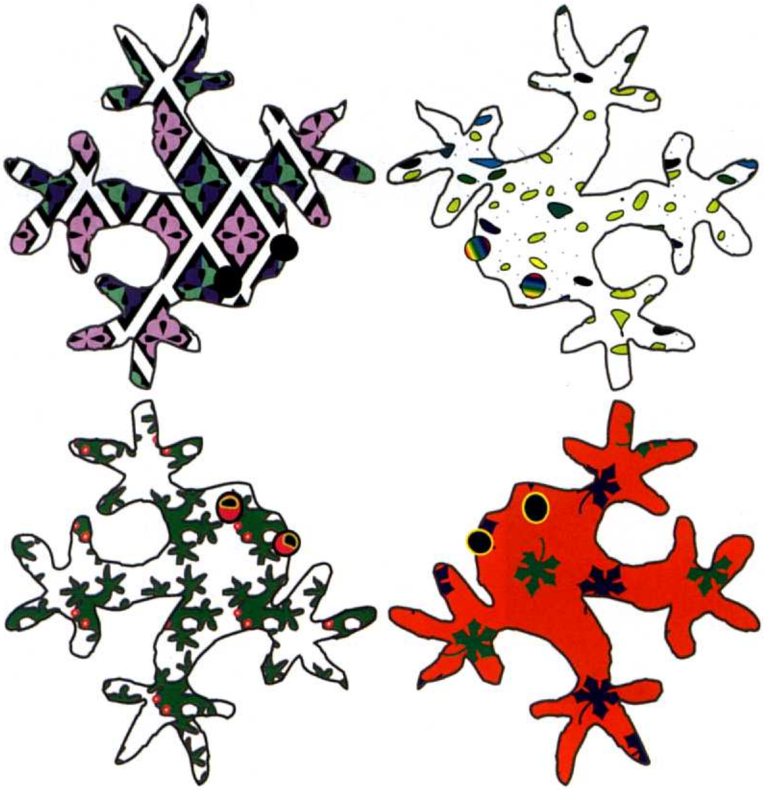
3. Vector image of a poppy

4. Bitmapped image of an iris

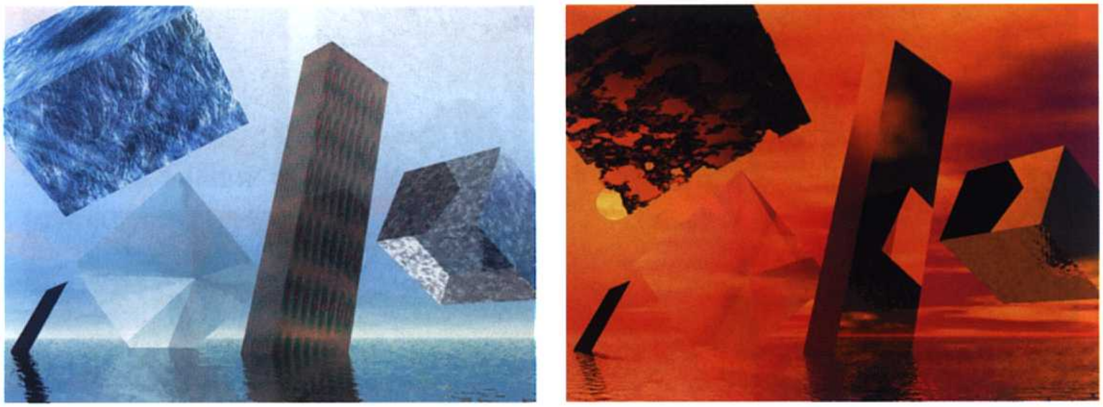
6. Use of gradient meshes



5. Gradient fills



7. Pattern fills

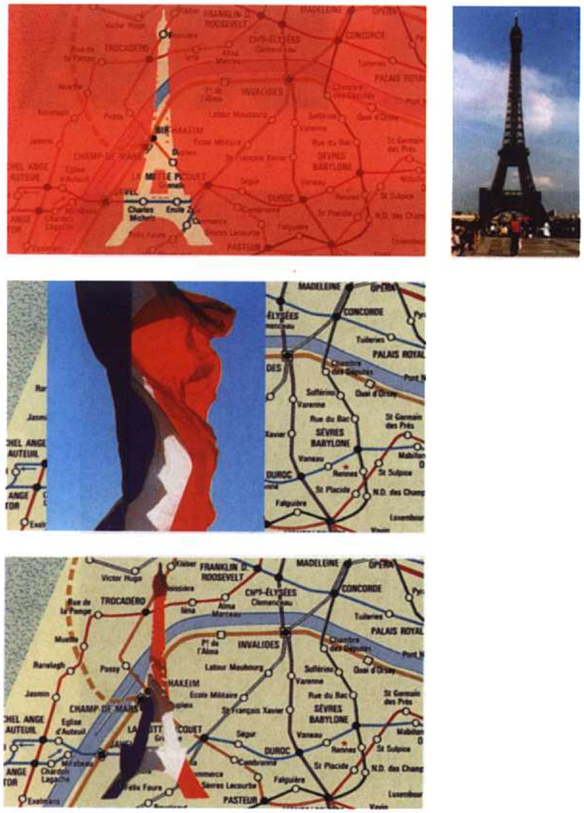


8. Ray traced rendering with different lighting and surface characteristics

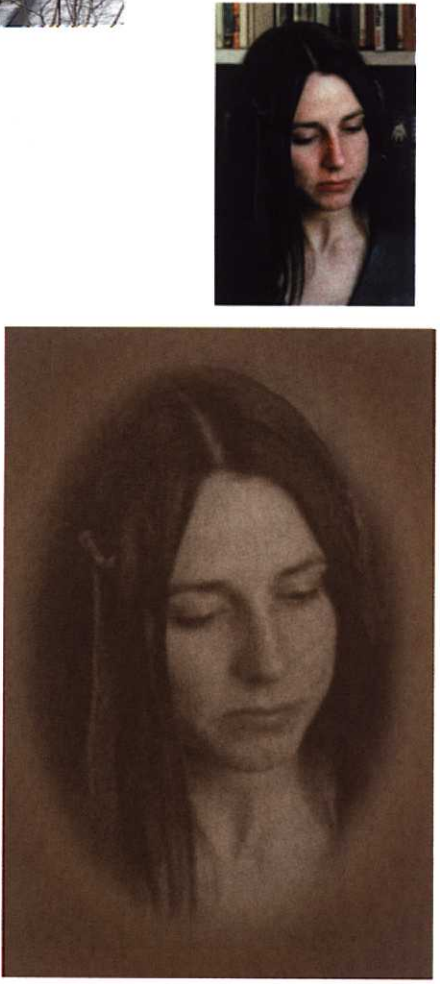
9. A figure modelled and posed in Poser

10. A landscape modelled in Bryce

11. Fading with an alpha channel gradient



12. Compositing with an alpha channel



13. A vignette

rendered image to a light source. On the way, the ray may bounce off several surfaces, and the ray tracer will take account of their effect, using a model of illumination and information about the materials of the objects in the scene, to compute the colour and intensity of the starting pixel. Plate 8 shows two renderings of Figure 4.48 by a ray tracer, and illustrates the qualities of different surfaces that can be applied to the objects. The ray tracing computation is quite complex, and it must be repeated for every pixel in the rendered image — very possibly millions of pixels. Until recently, therefore, ray tracing was confined to high-performance workstations, but the extraordinary advance in the speed of personal computers has made it feasible to use ray tracing in desktop 3-D systems. It is now the preferred shading algorithm for 'photo-realistic' graphics. It can produce excellent results, particularly in scenes including transparent or semi-transparent objects.

An alternative approach to the interactions between objects is *radiosity*, which attempts to model the complex reflections that occur between surfaces that are close together. This provides a more accurate representation of scattered and ambient light, and is especially useful for interior scenes. Radiosity is more accurately based on the physics of light than other shading algorithms. It also differs from them in computing the lighting on a model independently of any rendered view of it; essentially, it adds the computed light values to the model. This means that the final rendering of an image can be done very efficiently, although the initial computation of lighting values is slow.

These shading algorithms depend on information about the material of which an object is composed. Where the surface contains a great deal of detail, it may not be feasible to specify this information precisely enough for every small feature to be rendered accurately. A popular way of adding surface details to 3-D models is *texture mapping*. An image, typically a pattern representing some particular sort of surface's appearance, such as fur, bark, sand, marble, and so on, is mathematically wrapped over the surface of the object. That is, mathematical transformations are applied to the pixels of the rendered object, based on those of the image, to produce the appearance of an object with the image wrapped around it. To see the effect of this, imagine cutting a picture out of a magazine, wrapping it round a physical object, and sticking it down. This will work quite well for boxes, producing a box whose surface has the appearance of the picture, but it will work much less well for spherical or irregular shaped objects. If you can further imagine that the picture was printed on a very thin sheet of rubber, you can see that it would be possible to glue it to these shapes, but that it would become distorted in the process. Texture mapping is similar to gluing rubber pictures to objects, and may introduce the same sort of distortion, but it can also provide a convenient means of applying detail to a surface. Related operations include *bump mapping*, where, instead of using the picture to form the surface appearance, we use it to apply bumpiness or roughness, and *transparency mapping* and *reflection mapping*, which modify the corresponding optical characteristics on the basis of a two-dimensional map in the same way.

⇨ It is by no means uncommon for a rendered image to be imported into an image manipulation program, such as Photoshop, for further processing of the sort we will describe in Chapter 5.

Specialized Types of 3-D Graphics

The complexity of modelling and rendering arbitrary 3-D objects, and the difficulty of mastering general-purpose 3-D tools, strongly suggest that, where possible, specialized tools dedicated to a specific type of model should be used. With a specialized modelling program, a designer can use higher level abstractions that belong to a particular domain without having to provide general modelling tools for any and all types of object. For example, in a figure-modelling program you would build a body out of arms and legs, and so on, instead of trying to construct it as a polygon mesh. The rendering engine can use algorithms that are optimized for the characteristic models produced within the limits set by the modeller. A further potential advantage is that manipulating objects that correspond to things in the world, instead of manipulating their geometrical components, is better fitted to the ways of seeing that artists and designers are experienced with. (Traditional 3-D systems, on the other hand, are generally more suited to the ways of seeing of engineers and scientists, who may find them easier and more familiar than the specialized applications we will describe next.)

Two prominent examples of such tools (both from MetaCreations) are *Bryce*, for landscapes, and *Poser*, for human and animal figures.

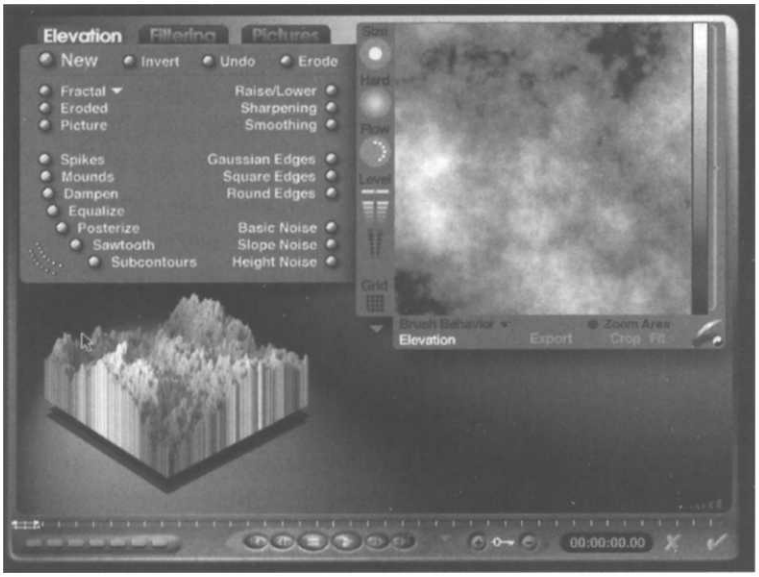


Figure 4.49
Bryce's terrain editor

In Bryce, modelling terrain is a basic activity, not, for example, one way of using polygon mesh objects. A terrain object represents the shape of a landscape; it is constructed from a greyscale image, whose brightness represents the height of the 3-D terrain model. Figure 4.49 shows Bryce's terrain editor, with a preview of a terrain, and the image (a fractal image, in this case) from which it is being generated. Terrains can be based on an imported image, or one painted by hand; they can be generated automatically using fractals, or built from US Geological Survey satellite data; they can be eroded, as if by rain, and given features such as ridges or canyons. Elaborate facilities are provided for creating textures and materials to apply to terrains, to simulate the appearance of natural or fantastic phenomena.

To complement its terrain modelling, Bryce provides facilities for modelling sky and atmosphere. Clouds, fog, haze, the sun, moon and stars, even rainbows can be added to the sky. All these features can be customized to a fine degree.

Bryce uses ray tracing, optimized for landscape models, to render its final output. Although Bryce is by no means a trivial program, it allows a designer without the specialized skills required by general 3-D applications to produce landscapes, such as the one shown in Plate 10, with relative ease.

Poser is dedicated to posing figures in a realistic way. Figures are a digital equivalent of the traditional manikins, or wooden jointed models used by artists as a substitute for live models. The limbs of manikins can be positioned to imitate lifelike poses, which can be drawn by an artist. In a similar way, the limbs and other body parts of a Poser figure can be placed in a pose, and then rendered to produce an image of a figure. Just as the joints of a well-made manikin only move in ways that real people's joints can move, so Poser figures are subject to constraints that force their poses to be physically realizable: the hand on a figure of a person cannot turn 360° on their wrist, for example. Recent releases of Poser provide extensive control over facial expressions as well as poses.

Poser has a library of figures, including variations of male and female human figures, and some animals (although these are not as fully realized as the human figures). Most aspects of their physiology can be customized, and they can be given clothes, which can also be customized; textures and other maps can be applied to people and clothes, either to create realistic skin and textiles, or to produce fantastic effects, such as transparent people. Poser treats figures as figures, not as 3-D models of the kind we described earlier. It thus provides an intuitive way of producing 3-D models representing people and animals. Because of our particularly intimate understanding of figures, and the notorious difficulty of producing convincing representations of them, this is an especially suitable domain for a specialized modelling approach. Whether results such as those shown in Plate 9 have any worth is a matter of opinion. (There exist more powerful figure modelling programs, which are used by film effects units to produce more lifelike results.)

⇨ Poser provides a good example of the need for ingenious computer programs to be supplemented by human skills. In order to produce convincing results, whether in posing figures or in creating new ones, it is necessary to understand how people are put together and how they move. This sort of understanding can only come from lengthy study, as practised in traditional life drawing classes. Although the computer might make it unnecessary to know how to handle charcoal or paint, it does not affect the need for the real skills of seeing and understanding that traditional art education seeks to develop.

Poser and Bryce can be used together. For example, a figure made in Poser can be imported into Bryce to be placed in a landscape. Both applications can also export models in formats that can subsequently be imported into a general purpose modelling application. Models constructed in both Poser and Bryce can be animated — see Chapter 11.

⇨ Another of MetaCreations' programs is specialized in a different way. *Canoma* is dedicated to a single way of constructing models: *image-assisted* modelling. The starting point for a model is a photograph (or several photographs), and the model is constructed by 'pinning' the nodes of a wire frame to corresponding points on the photograph. The photographic image is then mapped on to the surface of the resulting solid, so, in effect, the original picture is extended into the third dimension. Clearly, in order to produce an accurate model, pictures of at least two sides of the object must be used in conjunction with each other. The pinning operation works best on objects, such as buildings and domestic appliances, which are fundamentally simple geometric solids. Hence, although Canoma is not dedicated to a specific area, in the way that Bryce and Poser are, this method of modelling is best suited for certain applications, such as producing architectural models.

Further Information

Animation of vector graphics, including 3-D, is described in Chapter 11. [FvDFH96] contains a comprehensive technical account of most aspects of vector graphics; [WW92] provides details of ray tracing and radiosity. [Vin92] and [O'R98] are more accessible introductions to 3-D graphics.

Exercises

1. True or false: anti-aliasing never affects the appearance of vertical and horizontal lines?
2. (For mathematicians.) For many purposes, it is convenient to know the *bounding box* of a vector graphics object, that is, the smallest rectangle that entirely encloses the object.
   (a) Show that the bounding box on its own is an adequate representation for an ellipse.
   (b) Can a Bézier curve be represented by a bounding box?

3. What happens if two Bézier curves with control points P_1, P_2, P_3 and P_4, and P_4, P_5, P_6 and P_7 are joined at P_4 so that P_3, P_4 and P_5 are in a straight line but P_5 and P_3 are on the same side of P_4?

4. Gradient fills are not directly provided in PostScript and other low-level graphics languages. Describe how you would implement linear and radial gradients using the other vector graphics primitives described in this chapter.

5. (a) The algorithm for computing the winding number of a point relative to a path does not tell you what to do if the line you construct coincides with part of the path or touches it tangentially. How would you cope with these possibilities?

   (b) An alternative to the non-zero winding number rule that is sometimes used instead is the *odd-even rule*, which is computed in a similar way by constructing a line from a point and counting crossings, only this time you just count the number of times the path crosses the line. If it is even, the point is outside the path, if it is odd, it is inside. Does this rule always give the same result as the non-zero winding number rule? If not, when will they disagree?

6. The difference operation in constructive solid geometry is not commutative, that is, if we subtract shape A from shape B the result is different from what we get by subtracting shape B from shape A. Figure 4.40 shows the effect of subtracting the cylinder from the ellipsoid. Sketch the object that would be obtained by subtracting the ellipsoid from the cylinder.

7. Under what circumstances would you *not* use ray tracing to render a 3-D scene?

8. Suggest two types of specialized 3-D graphics, besides landscapes and figures, for which it would be sensible to use special-purpose tools instead of general 3-D graphics packages. Outline the main features you would expect in the corresponding tools — what abstract components would they support, and how might they take advantage of characteristics of your problem domains to optimize rendering performance?

Bitmapped Images

5

Conceptually, bitmapped images are much simpler than vector graphics. There is no need for any mathematical modelling of shapes, we merely record the value of every pixel in the image. This means that when the image is created, a value has to be assigned to every pixel, but many images are created from external sources, such as scanners or digital cameras, which operate in a bitmapped fashion anyway. For creating original digital images, programs such as Painter allow visual artists to use familiar techniques (at least metaphorically) to paint images. As we pointed out earlier, the main cost for this simplicity is in the size of image files. There is, though, one area where bitmaps are less simple than vectors, and that is resolution. We touched on this in Chapter 3, but now we shall look at it in more detail.

Resolution

The concept of resolution is a simple one, but the different ways in which the word is used can be confusing. Resolution is a measure of how finely a device approximates continuous images using finite pixels. It is thus closely related to sampling, and some of the ideas about sampling rates introduced in Chapter 2 are relevant to the way resolution affects the appearance of images.

There are two common ways of specifying resolution. For printers and scanners, the resolution is usually stated as the number of dots per unit length. Usually, in English-speaking countries, it is specified anachronistically in units of dots per inch (dpi). At the time of writing, desktop printers typically have a resolution of 600 dpi, while imagesetters (as used for book production) have a resolution of about 1200 to 2700 dpi; flatbed scanners' resolution ranges from 300 dpi at the most basic level to 3600 dpi; transparency scanners and drum scanners used for high quality work have even higher resolutions.

In video, resolution is normally specified by giving the size of a frame, measured in pixels — its *pixel dimensions*. For example, a PAL frame is 768 by 576 pixels; an NTSC frame is 640 by 480.[1] Obviously, if you know the physical dimensions of your TV set or video monitor, you can translate resolutions specified in this form into dots per inch. For video it makes more sense to specify image resolution in the form of the pixel dimensions, because the same pixel grid is used to display the picture on any monitor (using the same video standard) irrespective of its size. Similar considerations apply to digital cameras, whose resolution is also specified in terms of the pixel dimensions of the image.

Knowing the pixel dimensions, you know how much detail is contained in the image; the number of dots per inch on the output device tells you how big that same image will be, and how easy it will be to see the individual pixels.

Computer monitors are based on the same technology as video monitors, so it is common to see their resolution specified as an image size, such as 640 by 480 (for example, VGA), or 1024 by 768. However, monitor resolution is sometimes quoted in dots per inch, because of the tendency in computer systems to keep this value fixed and to increase the pixel dimensions of the displayed image when a larger display is used. Thus, a 14 inch monitor provides a 640 by 480 display at roughly 72 dpi; a 17 inch monitor will provide 832 by 624 pixels at the same number of dots per inch.

⇨ There is an extra complication with colour printers. As we will see in Chapter 6, in order to produce a full range of colours using just four or six inks, colour printers arrange dots in groups, using a pattern of

[1] Sort of — see Chapter 10.

> different coloured inks within each group to produce the required colour by optical mixing. Hence, the size of the coloured pixel is greater than the size of an individual dot of ink. The resolution of a printer taking account of this way of mixing colours is quoted in *lines per inch* (or other unit of length), following established printing practice.[2] The number of lines per inch will be as much as five times lower than the number of dots per inch — the exact ratio depends on how the dots are arranged, which will vary between printers, and may be adjustable by the operator. You should realise that, although a colour printer may have a resolution of 1200 *dots* per inch, this does not mean that you need to use such a high resolution for your images. A line resolution of 137 per inch is commonly used for printing magazines; the colour plates in this book are printed at a resolution of 150 lines per inch.

Now consider bitmapped images. An image is an array of pixel values, so it necessarily has pixel dimensions. Unlike an input or output device, it has no *physical* dimensions. In the absence of any further information, the physical size of an image when it is displayed will depend on the resolution of the device it is to be displayed on. For example, the square in Figure 3.1 on page 70 is 128 pixels wide. When displayed at 72 dpi, as it will be on a Macintosh monitor, for example, it will be 45mm square. Displayed without scaling on a higher resolution monitor at 115 dpi, it will only be a little over 28mm square. Printed on a 600 dpi printer, it will be about 5mm square. (Many readers will probably have had the experience of seeing an image appear on screen at a sensible size, only to be printed the size of a postage stamp.) In general, we have:

$$\textit{physical dimension} = \textit{pixel dimension} / \textit{device resolution}$$

where the device resolution is measured in pixels per unit length. (If the device resolution is specified in pixels per inch, the physical dimension will be in inches, and so on.)

Images have a natural size, though: the size of an original before it is scanned, or the size of canvas used to create an image in a painting program. We often wish the image to be displayed at its natural size, and not shrink or expand with the resolution of the output device. In order to allow this, most image formats record a resolution with the image data. This resolution is usually quoted in units of *pixels* per inch (ppi), to distinguish it from the resolution of physical devices. The stored resolution will usually be that of the device from which the image originated. For example, if the image is scanned at 600 dpi, the stored *image resolution*

[2] This figure is sometimes called the *screen ruling*, again following established terminology from the traditional printing industry.

will be 600 ppi. Since the pixels in the image were generated at this resolution, the physical dimensions of the image can be calculated from the pixel dimensions and the image resolution. It is then a simple matter for software that displays the image to ensure that it appears at its natural size, by scaling it by a factor of *device resolution/image resolution.* For example, if a photograph measured 6 inches by 4 inches, and it was scanned at 600dpi, its bitmap would be 3600 by 2400 pixels in size. Displayed in a simple-minded fashion on a 72dpi monitor, the image would appear 50 inches by 33.3 inches (and, presumably, require scroll bars). To make it appear at the desired size, it must be scaled by 72/600 = 0.12, which, as you can easily verify, reduces it to its original size.

If an image's resolution is lower than that of the device on which it is to be displayed, it must be scaled up, a process which will require the interpolation of pixels. This can never be done without loss of image quality, so you should try to ensure that any images you use in your multimedia productions have an image resolution at least as high as the monitors on which you expect your work to be viewed.

If, on the other hand, the image's resolution is higher than that of the output device, pixels must be discarded when the image is scaled down for display at its natural size. This process is called *downsampling.* Here we come to an apparent paradox. The subjective quality of a high resolution image that has been downsampled for display at a low resolution will often be better than that of an image whose resolution is equal to the display resolution. For example, when a 600 dpi scan is displayed on screen at 72 dpi, it will often look better than a 72 dpi scan, even though there are no more pixels in the displayed image. This is because the scanner samples discrete points; at lower resolutions these points are more widely spaced than at high resolutions, so some image detail is missed. The high resolution scan contains information that is absent in the low resolution one, and this information can be used when the image is downsampled for display. For example, the colour of a pixel might be determined as the average of the values in a corresponding block, instead of the single point value that is available in the low resolution scan. This can result in smoother colour gradients and less jagged lines for some images. The technique of sampling images (or any other signal) at a higher resolution than that at which it is ultimately displayed is called *oversampling.*

The apparently superior quality of oversampled images will only be obtained if the software performing the downsampling does so in a sufficiently sophisticated way to make use of the extra information available in the high-resolution image. Web browsers are notoriously poor at down-sampling, and usually produce a result no better than that which would be obtained by starting from a low-resolution original. For that reason, images intended for the World Wide Web should be downsampled in advance using a program such as Photoshop.

We will describe resampling in more detail later in this chapter, when we consider applying geometrical transformations to bit-mapped images.

Information once discarded can never be regained. The conclusion would seem to be that one should always keep high resolution bit-mapped images, downsampling only when it is necessary for display purposes or to prepare a version of the image for display software that does not downsample well. However, the disadvantage of high resolution images soon becomes clear. High resolution images contain more pixels and thus occupy more disk space and take longer to transfer over networks. The size of an image increases as the square of the resolution, so, despite the possible gains in quality that might come from using high resolutions, in practice we more often need to use the *lowest* resolution we can get away with. This must be at least as good as an average monitor if the displayed quality is to be acceptable. Even at resolutions as low as 72 or 96 ppi, image files can become unmanageable, especially over networks. To reduce their size without unacceptable loss of quality we must use techniques of data compression.

Image Compression

Consider again Figure 3.1. We stated on page 70 that its bitmap representation required 16 kilobytes. This estimate was based on the assumption that the image was stored as an array, with one byte per pixel. In order to display the image or manipulate it, it would have to be represented in this form, but when we only wish to record the values of its pixels, for storage or transmission over a network, we can use a much more compact data representation. Instead of storing the value of each pixel explicitly, we could instead store a value, followed by a count to indicate a number of consecutive pixels of that value. For example, the first row consists of 128 pixels, all of the same colour, so instead of using 128 bytes to store that row, we could use just two, one to store the value corresponding to that colour, another to record the number of occurrences. Indeed, if there was no advantage to be gained from preserving the identity of individual rows, we could go further, since the first 4128 pixels of this particular image are all the same colour. These are followed by a run of 64 pixels of another colour, which again can be stored using a count and a colour value in two bytes, instead of as 64 separate bytes all of the same value.

This simple technique of replacing a run of consecutive pixels of the same colour by a single copy of the colour value and a count of the number of pixels in the run is called *run-length encoding (RLE)*. In common with other methods of compression, it requires some computation in order to achieve a saving in space. Another feature it shares with other methods of compression is that its effectiveness depends on the image that is being compressed. In this example, a large saving in storage can be achieved, because the image is extremely simple and consists of large areas of the same colour, which give rise to long runs of identical pixel values. If, instead, the image had consisted of alternating pixels of the two colours, applying RLE in a naïve fashion would have led to an increase in the storage requirement, since each 'run' would have had a length of one, which would have to be recorded in addition to the pixel value. More realistically, images with continuously blended tones will not give rise to runs that can be efficiently encoded, whereas images with areas of flat colour will.

It is a general property of any compression scheme that there will be some data for which the 'compressed' version is actually larger than the uncompressed. This must be so: if we had an algorithm that could always achieve some compression, no matter what input data it was given, it would be possible to apply it to its own output to achieve extra compression, and then to the new output, and so on, arbitrarily many times, so that any data could be compressed down to a single byte (assuming we do not deal with smaller units of data). Even though this is clearly absurd, from time to time people claim to have developed such an algorithm, and have even been granted patents.

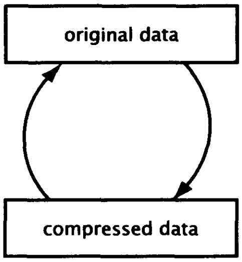


Figure 5.1
Lossless compression

Run-length encoding has an important property: it is always possible to decompress run-length encoded data and retrieve an

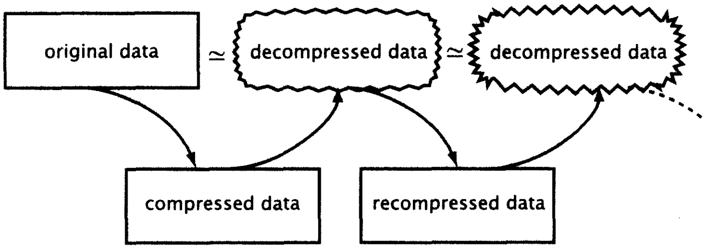


Figure 5.2
Lossy compression

exact copy of the original data as it was before compression. If we take the 16 kilobyte array representing Figure 3.1 and apply RLE compression to it, we will be able to apply an obvious decompression algorithm to the result to get back the original array. RLE is an example of a *lossless* compression technique, since no information is lost during a compression/decompression cycle (see Figure 5.1). In contrast, *lossy* compression techniques work by discarding some information during the compression process. Once discarded, information can never be retrieved, so when data that has been lossily compressed is decompressed, the result is only an approximation to the original data (see Figure 5.2). Lossy compression is well suited to data, such as images and sound, that originates in analogue form, since, as we saw in Chapter 2, its digital representation is an approximation anyway. Clever design of compression algorithms can ensure that only data that is insignificant to perception of the image or sound is discarded, so that substantial savings in storage can be achieved with little or no perceived loss in quality. However, because information is lost during every compression/decompression cycle, if data is repeatedly compressed, decompressed and then compressed again, its quality will progressively deteriorate.

A full description of the mechanics of compression lies beyond the scope of this book; we will just introduce the main algorithms you are likely to encounter.

Lossless Compression

RLE is the simplest lossless compression algorithm to understand, but it is far from the most effective. The more sophisticated lossless algorithms you are likely to encounter fall into two classes. Algorithms of the first class work by re-encoding data so that the most frequent values occupy the fewest bits. For example, if an image uses 256 colours, each pixel would normally occupy eight bits (see Chapter 6). If, however, we could assign codes of different lengths to the colours, so that the code for the most common colour was only a single bit long, two-bit codes were used for the next most frequent colours, and so on, a saving in space would be achieved for most images. This approach to encoding, using *variable-length codes*, dates back to the earliest work on data compression and information theory, carried out in the late 1940s. The best known algorithm belonging to this class is *Huffman coding.*[3]

Although Huffman coding and its derivatives are still used as part of other, more complex, compression techniques, since the late 1970s variable-length coding schemes have been superseded to a large extent by *dictionary-based* compression schemes. Dictionary-based compression works by constructing a table, or dictionary, into which are entered strings of bytes (not necessarily corresponding to characters) that are encountered in the input data; all occurrences of a string are then replaced by a pointer into the dictionary. The process is similar to the tokenization of names carried out by the lexical analyser of a compiler using a symbol table. In contrast to variable-length coding schemes, dictionary-based schemes use fixed-length codes, but these point to variable-length strings in the dictionary. The effectiveness of this type of compression depends on choosing the strings to enter in the dictionary so that a saving of space is produced by replacing them by their codes. Ideally, the dictionary entries should be long strings that occur frequently.

Two techniques for constructing dictionaries and using them for compression were described in papers published in 1977 and 1978 by two researchers called Abraham Lempel and Jacob Ziv, thus the techniques are usually called LZ77 and LZ78.[4] A variation of LZ78, devised by another researcher, Terry Welch, and therefore known as LZW compression, is one of the most widely used compression methods, being the basis of the Unix `compress` utility and of GIF files. The difference between LZ77 and LZ78 lies in the way in which the dictionary is constructed, while LZW is really just an improved implementation for LZ77. LZW compression has one drawback: as we mentioned in Chapter 3, it is patented by Unisys, who charge a licence fee for its use. As a result of this, the compression method used in PNG files is based on the legally unencumbered LZ77, as are several widely used general-purpose compression programs, such as PKZIP.

[3] You will find Huffman coding described in many books on data structures, because its implementation provides an interesting example of the use of trees.

[4] It is a curious, but probably insignificant, fact that the papers' authors were actually given as Ziv and Lempel, not the other way round, but the algorithms are never referred to as ZL7x.

JPEG Compression

Lossless compresssion can be applied to any sort of data; it is the *only* sort of compression that can be applied to certain sorts, such as binary executable programs, spreadsheet data, or text, since corruption of even one bit of such data may invalidate it. Image data, though, can tolerate a certain amount of data loss, so lossy compression can be used effectively for images. The most important lossy image compression technique is *JPEG compression*. JPEG stands for the Joint Photographic Experts Group,[5] which draws attention to a significant feature of JPEG compression: it is best suited to photographs and similar images which are characterised by fine detail and continuous tones — the same characteristics as bitmapped images exhibit in general.

In Chapter 2, we considered the brightness or colour values of an image as a signal, which could be decomposed into its constituent frequencies. One way of envisaging this is to forget that the pixel values stored in an image denote colours, but merely consider them as the values of some variable z; each pixel gives a z value at its x and y coordinates, so the image defines a three-dimensional shape. (See Figure 5.3, which shows a detail from the iris image, rendered as a 3-D surface, using its brightness values to control the height.) Such a shape can be considered as a complex 3-D waveform. We also explained that any waveform can be transformed into the frequency domain using the Fourier Transform operation. Finally, we pointed out that the high frequency components are associated with abrupt changes in intensity. An additional fact, based on extensive experimental evidence, is that people do not perceive the effect of high frequencies very accurately, especially not in colour images.

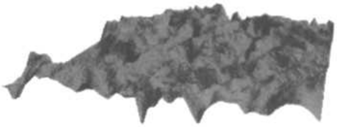


Figure 5.3
Pixel values interpreted as height

Up until now, we have considered the frequency domain representation only as a way of thinking about the properties of a signal. JPEG compression works by actually transforming an image into its frequency components. This is done, not by computing the Fourier Transform, but using a related operation called the *Discrete Cosine Transform (DCT)*. Although the DCT is defined differently from the Fourier Transform, and has some different properties, it too analyses a signal into its frequency components. In computational terms, it takes an array of pixels and produces an array of coefficients, representing the amplitude of the frequency

[5] 'Joint' refers to the fact that JPEG is a collaboration between two standards bodies, ISO and CCITT (now ITU).

components in the image. Since we start with a two-dimensional image, whose intensity can vary in both the x and y directions, we end up with a two-dimensional array of coefficients, corresponding to spatial frequencies in these two directions. This array will be the same size as the image's array of pixels.

Applying a DCT operation to an image of any size is computationally expensive, so it is only with the widespread availability of powerful processors that it has been practical to perform this sort of compression — and, more importantly, decompression — without the aid of dedicated hardware. Even now, it remains impractical to apply DCT to an entire image at once. Instead, images are divided into 8 × 8 pixel squares, each of which is transformed separately.

The DCT of an $N \times N$ pixel image $[p_{uv}, 0 \le u < N, 0 \le v < N]$ is an array of coefficients $[\mathrm{DCT}_{uv}, 0 \le u < N, 0 \le v < N]$ given by

$$\mathrm{DCT}_{uv} = \frac{1}{\sqrt{2N}} C_u C_v \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} p_{xy} \cos\left[\frac{(2x+1)u\pi}{2N}\right] \cos\left[\frac{(2y+1)v\pi}{2N}\right]$$

where

$$C_u, C_v = \begin{cases} 1/\sqrt{2} & \text{for } u, v = 0 \\ 1 & \text{otherwise} \end{cases}$$

On its own, this doesn't tell you much about why the DCT coefficients are in fact the coefficients of the frequency components of the image, but it does indicate something about the complexity of computing the DCT. For a start, it shows that the array of DCT coefficients is the same size as the original array of pixels. It also shows that the computation must take the form of a nested double loop, iterating over the two dimensions of the array, so the computational time for each coefficient is proportional to the image size in pixels, and the entire DCT computation is proportional to the square of the size. While this is not intractable, it does grow fairly rapidly as the image's size increases, and bitmapped images measuring thousands of pixels in each direction are not uncommon. Applying the DCT to small blocks instead of the entire image reduces the computational time by a significant factor.

Since the two cosine terms are independent of the value of p_{ij}, they can be precomputed and stored in an array to avoid repeated computations. This provides an additional optimization when the DCT computation is applied repeatedly to separate blocks of an image.

Transforming an image into the frequency domain does not, in itself, perform any compression. It does, however, change the data into a form which can be compressed in a way that minimizes the perceptible effect of discarding information, because the frequency components are now explicitly separated. This allows information about the high frequencies, which do not contribute much to the perceived quality of the image, to be discarded. This is done by distinguishing fewer different possible values for higher frequency components. If, for example, the value produced by the DCT for each frequency could range from 0 to 255, the lowest frequency coefficients might be allowed to have any integer value within this range; slightly higher frequencies might only be allowed to take on values divisible by 4, while the highest frequencies might only be allowed to have the value 0 or 128. Putting this another way, the different frequencies are quantized to different numbers of levels, with fewer levels being used for high frequencies. In JPEG compression, the number of quantization levels to be used for each frequency coefficient can be specified separately in a quantization matrix, containing a value for each coefficient.

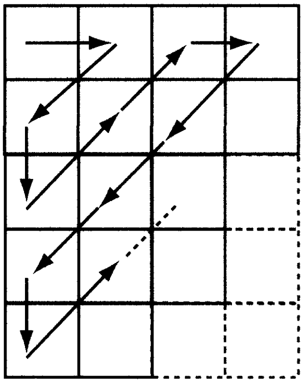


Figure 5.4
The zig-zag sequence

This quantization process reduces the space needed to store the image in two ways. First, after quantization, many components will end up with zero coefficients. Second, fewer bits are needed to store the non-zero coeficients. To take advantage of the redundancy which has thus been generated in the data representation, two lossless compression methods are applied to the array of quantized coefficients. Zeroes are run-length encoded; Huffman coding is applied to the remaining values. In order to maximize the length of the runs of zeroes, the coefficients are processed in what is called the *zig-zag sequence*, as shown in Figure 5.4. This is effective because the frequencies increase as we move away from the top left corner of the array in both directions. In other words, the perceptible information in the image is concentrated in the top left part of the array, and the likelihood is that the bottom right part will be full of zeroes. The zig-zag sequence is thus likely to encounter long runs of zeroes, which would be broken up if the array were traversed more conventionally by rows or columns.

Decompressing JPEG data is done by reversing the compression process. The runs are expanded and the Huffman encoded coefficients are decompressed and then an *Inverse Discrete Cosine Transform* is applied to take the data back from the frequency domain into the spatial domain, where the values can once again be treated as pixels in an image. The inverse DCT is defined very similarly to the DCT itself; the computation of the inverse transform requires the same amount of time as that of the forward transform, so JPEG compression and decompression take roughly the same time.[6] Note that there is no 'inverse quantization' step. The information that was lost during quantization is gone forever, which is why the decompressed image only approximates the original. Generally, though, the approximation is a good one.

One highly useful feature of JPEG compression is that it is possible to control the degree of compression, and thus the quality of the compressed image, by altering the values in the quantization matrix. Programs that implement JPEG compression allow you to choose a quality setting, so that you can choose a suitable compromise between image quality and compression. You should be aware that, even at the highest quality settings, JPEG compression is still lossy in the literal sense that the decompressed image will not be an exact bit-for-bit duplicate of the original. It will often, however, be visually indistinguishable, but that is not what we mean by 'lossless'.

⇨ The JPEG standard does define a lossless mode, but this uses a completely different algorithm from the DCT-based method used by lossy JPEG. Lossless JPEG compression has never been popular. It has been superseded by a new standard JPEG-LS, but this, too, shows little sign of being widely adopted.

Yet another JPEG standard is also under construction: JPEG2000 aims to be the image compression standard for the new century (or at least the first ten years of it). The ISO's call for contributions to the JPEG2000 effort describes its aims as follows:

> "[...] to create a new image coding system for different types of still images (bi-level, gray-level, color) with different characteristics (natural images, scientific, medical, remote sensing imagery, text, rendered graphics, etc.) allowing different imaging models (client/server, real-time transmission, image library archival, limited buffer and bandwidth resources, etc.) preferably within a unified system."

JPEG compression is highly effective when applied to the sort of images for which it is designed: photographic and scanned images with continuous tones. Such images can be compressed to as little as 5% of their original size without apparent loss of quality. Lossless compression techniques are nothing like as effective on such images. Still higher levels of compression can be obtained by using a lower quality setting, i.e. by using coarser quantization that discards more information. When this is done,

[6] On the same machine.

the boundaries of the 8 × 8 squares to which the DCT is applied tend to become visible, because the discontinuities between them mean that different frequency components are discarded in each square. At low compression levels (i.e. high quality settings) this does not matter, since enough information is retained for the common features of adjacent squares to produce appropriately similar results, but as more and more information is discarded, the common features becomes lost and the boundaries show up.

Such unwanted features in a compressed image are called *compression artefacts.* Other artefacts arise when an image containing sharp edges is compressed by JPEG. Here, the smoothing that is the essence of JPEG compression is to blame; sharp edges come out blurred. This is rarely a problem with the photographically originated material for which JPEG is intended, but can be a problem if images created on a computer are compressed. In particular, if text, especially small text, occurs as part of an image, JPEG is likely to blur the edges, often making the text unreadable. For images with many sharp edges, JPEG should be avoided. Instead, images should be saved in a format such as PNG, which uses lossless LZ77 compression.

Image Manipulation

A bitmapped image explicitly stores a value for every pixel, so we can, if we wish, alter the value of any pixel or group of pixels to change the image. The sheer number of pixels in most images means that editing them individually is both time-consuming and confusing — how is one to judge the effect on the appearance of the whole image of particular changes to certain pixels? Or which pixels must be altered, and how, in order to sharpen the fuzzy edges in an out-of-focus photograph? In order for image editing to be convenient, it is necessary that operations be provided at a higher level than that of altering a single pixel. Many useful operations can be described by analogy with pre-existing techniques for altering photographic images, in particular, the use of filters and masks.

Before we describe how images can be manipulated, we ought first to examine why one might wish to manipulate them. There are two broad reasons: one is to correct deficiencies in an image, caused by poor equipment or technique used in its creation or digitization, the other is to create images that are difficult or impossible to make naturally. An example of the former type is the removal of 'red-eye', the red glow apparently emanating from the eyes of a person whose portrait has been taken face-on with a camera using a flash set too close to the lens.[7] Consumer-oriented image manipulation programs often provide commands that encapsulate a sequence of manipulations to perform common tasks, such as red-eye removal, with a single key stroke. The manipulations performed for the second reason generally fall into the category of special effects, such as creating a glow around an object. Image manipulation programs, such as Photoshop, generally supply a set of built-in filters and effects, and a kit of tools for making selections and low-level adjustments. Photoshop uses an open architecture that allows third parties to produce plug-ins which provide additional effects and tools — the development of Photoshop plug-ins has become an industry in its own right, and many other programs can now use them.

Many of the manipulations commonly performed on bitmapped images are concerned purely with preparing digital images for print, and are therefore not relevant to multimedia. An operation that is typical of multimedia work, on the other hand, is the changing of an image's resolution or size (which, as we saw previously, are essentially equivalent operations). Very often, images which are required for display on a monitor originate at higher resolutions and must be down-sampled; an image's size may need adjusting to fit a layout, such as a Web page.

⇨ Photoshop is, without doubt, the leading application for image manipulation, being a *de facto* industry standard, and we will use it as our example. There are other programs, though, in particular a powerful package known as *The Gimp*, which has similar capabilities and a similar set of tools to Photoshop's, is distributed under an open software licence for Unix systems. Both of these programs include many features concerned with preparing images for print, which are not relevant to multimedia. Recently, a number of packages, such as Adobe's ImageReady and Macromedia's Fireworks, which are dedicated to preparation of images for the World Wide Web have appeared; these omit print-oriented features, replacing them with others more appropriate to work on the Web, such as facilities for slicing an image into smaller pieces to accelerate downloading, or adding hotspots to create an 'image map' (see Chapter 13).

[7] Red-eye is caused by light reflecting off the subject's retinas.

Selections, Masks and Alpha Channels

As we have repeatedly stressed, a bitmapped image is not stored as a collection of separate objects, it is just an array of pixels. Even if we can look at the picture and see a square or a circle, say, we cannot select that shape when we are editing the image with a program, the way we could if it were a vector image, because the shape's identity is not part of the information that is explicitly available to the program, it is something our eyes and brain have identified. Some other means must be employed to select parts of an image when it is being manipulated by a mere computer program.

Ironically, perhaps, the tools that are used to make selections from bitmapped images are more or less the same tools that are used to draw shapes in vector graphics. Most selections are made by drawing around an area, much as a traditional paste-up artist would cut out a shape from a printed image using a scalpel. The simplest selection tools are the rectangular and elliptical *marquee* tools, which let you select an area by dragging out a rectangle or ellipse, just as you would draw these shapes in a drawing program. It is important to realize that you are not drawing, though, you are defining an area within the image.

More often than not, the area you wish to select will not be a neat rectangle or ellipse. To accommodate irregular shapes, thinly disguised versions of the other standard drawing tools may be used: the lasso tool is a less powerful version of Illustrator's pencil tool, which can be used to draw freehand curves around an area to be selected; the polygon lasso is like a pen tool used to draw polylines, rather than curves; a fully-fledged Bézier drawing pen is also available. These tools allow selections to be outlined with considerable precision and flexibility, although their use can be laborious. To ease the task of making selections, two tools are available that make use of pixel values to help define the selected area. These are the magic wand and the magnetic lasso.

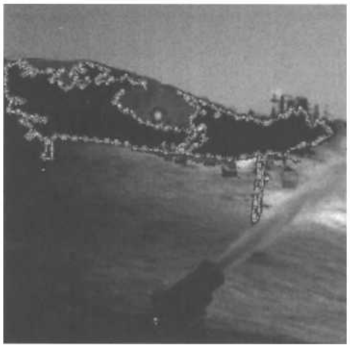


Figure 5.5
Selecting with the magic wand

The magic wand is used to select areas on the basis of their colour. With this tool selected, clicking on the image causes all pixels adjacent to the cursor which are similar in colour to the pixel under the cursor to be selected. Figure 5.5 shows an example of the magic wand selecting a highly irregular shape. The tolerance, that is, the amount by which a colour may differ but still be considered sufficiently similar to include in the selection, may be specified. The magnetic lasso works on a different principle. Like the other lasso tools, it is dragged around the area to be selected, but instead of simply following the outline drawn by the user, it adjusts itself so that the outline snaps to edges within a specified distance of the cursor. Any sufficiently large change in contrast is considered to be an edge. Both the distance within which edges are detected, and the degree of contrast variation that is considered to constitute an edge may be specified. Where an image has well-defined edges, for example, both of these can be set to a high value, so that drawing roughly round an object will cause it to be selected as the outline snaps to the high contrast edges. Where the edges are less well defined, it will be necessary to allow a lower contrast level to indicate an edge, and consequently the outline will have to be drawn with more care, using a narrower detection width.

These semi-automatic selection tools can be somewhat erratic, and cannot generally cope with such demanding tasks as selecting hair or delicate foliage. It is often necessary to make a preliminary selection with one of these tools, and then refine it. The outline of a selection is essentially a vector path, and adjustments are made by moving, adding or deleting control points.

Once a selection has been made, using any of the tools just described, any changes you make to the image, such as applying filters, are restricted to the pixels within the selected area. Another way of describing this is to say that the selection defines a *mask* — the area that is not selected, which is protected from any changes. Image manipulation programs allow you to store one or more masks with an image, so that a selection can be remembered and used for more than one operation — an ordinary selection is ephemeral, and is lost as soon as a different one is made.

The technique of masking off parts of an image has long been used by artists and photographers, who use physical masks and stencils to keep out light or paint. A cardboard stencil, for example, either completely allows paint through or completely stops it. We could store a digital mask with similar 'all or nothing' behaviour by using a single bit for each pixel in the image, setting it to one for all the masked out pixels, and to zero for those in the selection. Thus, the mask is itself an array of pixels, and we can think of it as being another image. If just one bit is used for each pixel, this image will be purely monochromatic. By analogy with photographic masks, the white parts of the image are considered transparent, the black ones opaque.

Digital masks have properties which are difficult to realize with physical media. By using more than one bit, so that the mask becomes a greyscale image, we can specify different degrees of transparency. For reasons which will be elaborated on in Chapter 6, a greyscale mask of this sort is often called an *alpha channel* . Any painting, filtering or other modifications made to pixels covered by semi-transparent areas of the mask will be applied in a degree proportional to the value stored in the alpha channel. It is common to use eight bits for each pixel of a mask, allowing for 256 different transparency values.

To return to the analogy of a stencil, an alpha channel is like a stencil made out of a material that can allow varying amounts of paint to pass through it, depending on the transparency value at each point. One use for such a stencil would be to produce a soft edge around a cut out shape. In a similar way, the edge of a selection can be 'feathered', which means that the hard transition from black to white in the alpha channel is replaced by a gradient, passing through intermediate grey values, which correspond to partial masking. Any effects that are applied will fade over this transitional zone, instead of stopping abruptly at the boundary. A less drastic way of exploiting alpha channels is to apply anti-aliasing to the edge of a mask, reducing the jagged effect that may otherwise occur. Although anti-aliasing resembles feathering over a very narrow region, the intention is quite different: feathering is supposed to be visible, causing effects to fade out, whereas anti-aliasing is intended to unobtrusively conceal the jagged edges of the selection.

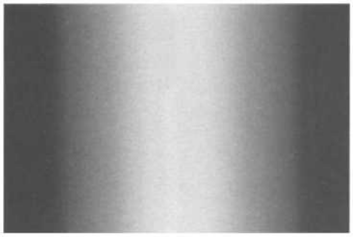


Figure 5.6
Alpha channel for Plate 11

Normally, if an image is pasted into another, it obscures everything underneath it, by overwriting the pixels. However, if the pasted image has an alpha channel, its transparency values are used to mix together the two images, so that the original will show through in the transparent areas of the mask. The value of a pixel p in the resulting composited image is computed as $p = \alpha p_1 + (1 - \alpha)p_2$, where p_1 and p_2 are the values of pixels in the two original images, and α is normalized to lie between 0 and 1.[8] Some familiar effects can be achieved by constructing a mask by hand. For example, if we use a gradient as the mask, we can make one image fade into another. Plate 11 shows an example: the mask shown in Figure 5.6 is used to combine a colour and a black and white version of the same photograph, producing a coloured 'magic corridor' down the middle of the composite image. Another example is shown in Plate 12.

[8] That is, if the α value is stored in 8-bits we divide it by 255.

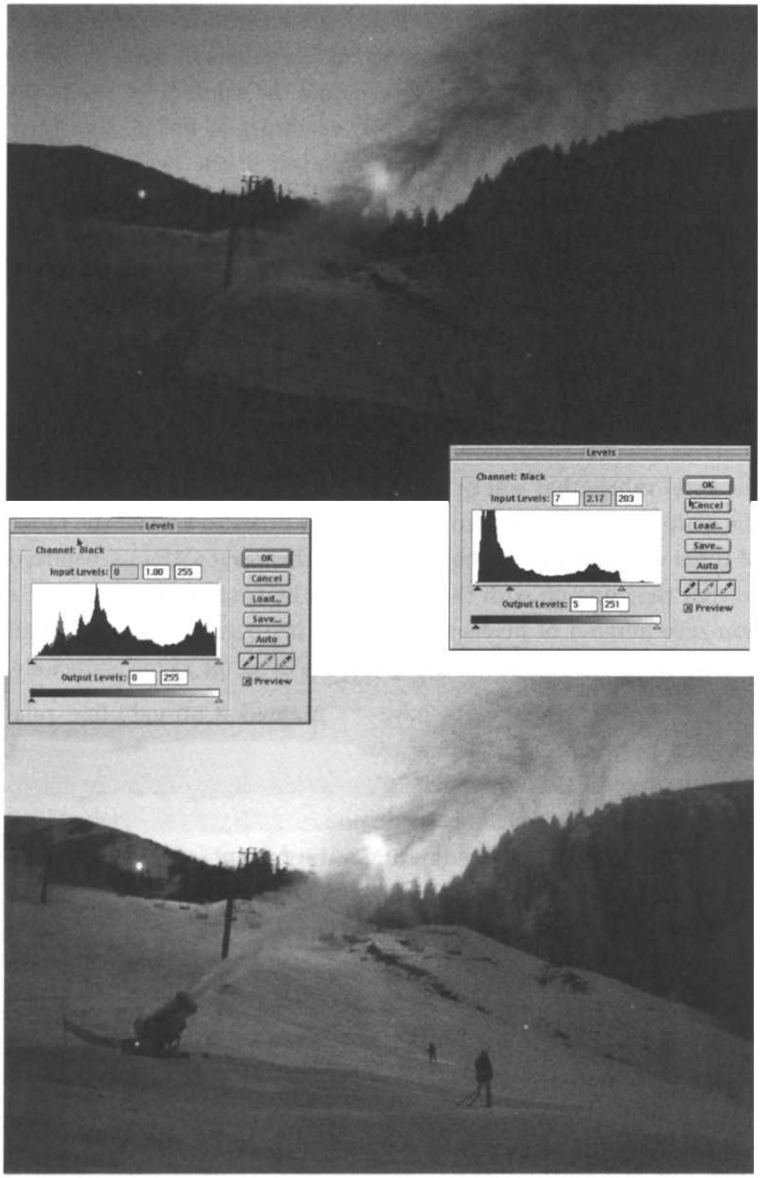
Here, a mask has been created by saving a magic wand selection from the photograph of the Eiffel tower. This mask has then been attached to the map of Paris and used to combine it with the French flag.

Pixel Point Processing

Image processing is performed by computing a new value for each pixel in an image. The simplest methods compute a pixel's new value solely on the basis of its old value, without regard to any other pixel. So for a pixel with value p, we compute a new value $p' = f(p)$, where f is some function, which we will call the *mapping function.* Such functions perform *pixel point processing.* A simple, if only rarely useful, example of pixel point processing is the construction of a negative from a greyscale image. Here, $f(p) = W - p$, where W is the pixel value representing white.

The most sophisticated pixel point processing is concerned with colour correction and alteration. We will not describe this fully until Chapter 6. Here, we will only consider the brightness and contrast alterations that are the typical applications of pixel point processing to greyscale images. Colour processing is an extension — although not a trivial one — of these greyscale adjustments. Once again, we will use Photoshop's tools to provide a concrete example, but any image editing software will offer the same functions, with roughly the same interface.

The crudest adjustments are made with the brightness and contrast sliders, which work like the corresponding controls on a monitor or television set. Brightness adjusts the value of each pixel up or down uniformly, so increasing the brightness makes every pixel lighter, decreasing it makes every pixel darker. Contrast is a little more subtle: it adjusts the range of values, either enhancing or reducing the difference between the lightest and darkest areas of the image. Increasing contrast makes the light areas very light and the dark areas very dark, decreasing it moves all values towards an intermediate grey. In terms of mapping functions, both of these adjustments produce a linear relationship that would be represented as a straight line on a graph: adjusting the brightness changes the intercept between the line and the y-axis; adjusting the contrast alters the gradient of the line.


Levels

Channel: Black

Input Levels: 7 2.17 203

Output Levels: 5 251

☒ Preview

Levels

Channel: Black

Input Levels: 0 1.00 255

Output Levels: 0 255

☒ Preview

Figure 5.7
Level adjustments

More control over the shape of the mapping function is provided by the levels dialogue, which allows you to move the endpoints of a linear mapping function individually, thereby setting the white and black levels in the image. Graphically, these adjustments stretch or shrink the mapping function horizontally and vertically. To help with choosing suitable levels, a display called the *image histogram* is used. This is a histogram showing the distribution of pixel values: the horizontal axis represents the possible values (from 0 to 255 in an 8-bit greyscale image), the bars show the number of pixels set to each value. Figure 5.7 shows two examples of images and their histograms. The histograms are displayed in the levels dialogue, with two sets of sliders below them, as shown. The upper set controls the range of input values. The slider at the left controls the pixel value that will be mapped to black, so, in graphical terms, it moves the intercept of the mapping function's line along the x-axis. The slider at the right controls the pixel value that is mapped to white, so it moves the top end of the line along the horizontal line corresponding to the maximum pixel value. The lower slider controls affect the output values in a similar way, i.e. they determine the pixel values that will be used for black and white, so they move the endpoints of the line up and down. In order to spread the range of tonal values evenly across the image, the input sliders are moved so that they line up with the lowest and highest values that have a non-zero number of pixels shown in the histogram. Moving beyond these points will compress or expand the dynamic range artificially.

So far, all the adjustments have maintained a straight-line relationship between old and new pixel values. The third slider that you can see on the upper levels control in Figure 5.7 allows you to produce a more flexible correspondence between original and modified pixel values, by adjusting a third point, corresponding to the mid-tones in the image. If an image's brightness is concentrated in a particular range, you can move the mid-point slider under the corresponding point on the histogram, so that the brightness values are adjusted to put this range in the centre of the available scale of values. Figure 5.7 shows the effect that level adjustments can achieve in bringing out detail that has been lost in an under-exposed photograph. The top image was shot at dusk, with too short an exposure; the levels dialogue below it shows the positions of the sliders that were used to produce the lower image. The changed image histogram can be seen in the lower levels dialogue box.

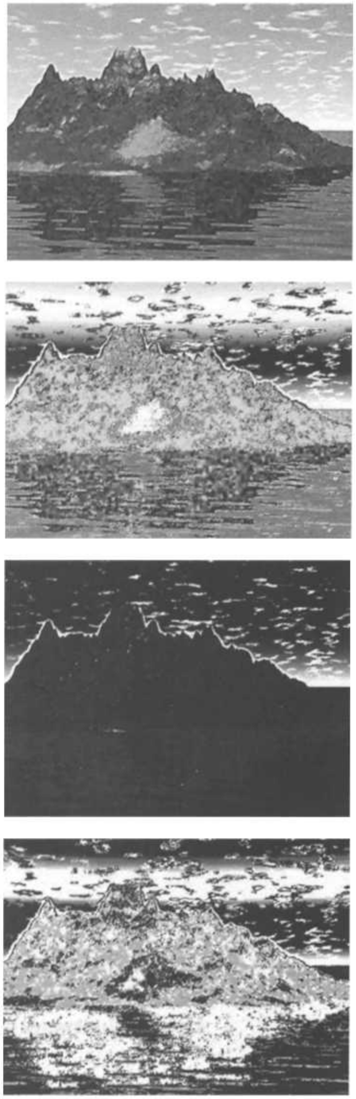


Figure 5.8
The effect of different curves on a single image

All of the brightness and contrast adjustment facilities described so far can be considered as making specialized alterations to the graph of the mapping function f to achieve particular commonly required adjustments to the values of individual pixels. In Photoshop, it is possible to take detailed control of this graph in the curves dialogue, where it can be reshaped by dragging control points, or completely redrawn with a pencil tool. The almost complete freedom to map grey levels to new values that this provides permits some strange effects, but it also makes it easy to apply subtle corrections to incorrectly exposed photographs, or to compensate for improperly calibrated scanners.

Before any adjustments are made, the curve is a straight line with slope equal to one: the output and input are identical, f is an identity function. Arbitrary reshaping of the curve will cause artificial highlights and shadows. Figure 5.8 shows a single image with four different curves applied to it, to bring out quite different features. More restrained changes are used to perform tonal adjustments with much more control than the simple contrast and brightness sliders provide. For example, an S-shaped curve such as the one illustrated in Figure 5.9 is often used to increase the contrast of an image: the mid-point is fixed and the shadows are darkened by pulling down the quarter-point, while the highlights are lightened by pulling up the three-quarter-point. The gentle curvature means that, while the overall contrast is increased, the total tonal range is maintained and there are no abrupt changes in brightness.

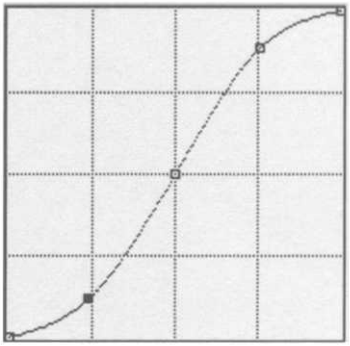


Figure 5.9
The S-curve for enhancing contrast

➲ The adjustments we have just described compute a pixel's new value as a function of its old value. We can look at compositing as another form of pixel point processing, where a pixel's value is computed as a function of the values of two corresponding pixels in the images or layers being combined. That is, when we merge pixels p_1 and p_2 in two separate images, we compute a result pixel with value $p' = p_1 \oplus p_2$, where $\oplus$ is some operator. The different blending modes that are provided by image processing programs correspond to different choices of $\oplus$. Generally, non-linear operators are needed to perform useful merging operations. In particular, it is common to use a threshold function, of the form $p' = p_1$ if $p_1 > t$, $p' = p_2$, otherwise, where the threshold value t corresponds to the opacity setting for the layer.

Pixel Group Processing

A second class of processing transformations works by computing each pixel's new value as a function not just of its old value, but also of the values of neighbouring pixels. Functions of this sort perform *pixel group processing*, which produces qualitatively different effects from the pixel point processing operations we described in the preceding section. In terms of the concepts we introduced in Chapter 2, these operations remove or attenuate certain spatial frequencies in an image. Such *filtering* operations can be implemented as operations that combine the value of a pixel with those of its neighbours, because the relative values of a pixel and its neighbours incorporate some information about the way the brightness or colour is changing in the region of that pixel. A suitably defined operation that combines pixel values alters these relationships, modifying the frequency make-up of the image. The mathematics behind this sort of processing is complicated, but the outcome is a family of operations with a simple structure.

It turns out that, instead of transforming our image to the frequency domain (for example, using a DCT) and performing a filtering operation by selecting a range of frequency components, we can perform the filtering in the spatial domain — that is, on the original image data — by computing a weighted average of the pixels and its neighbours. The weights applied to each pixel value determine the particular filtering operation, and thus the effect that is produced on the image's appearance. A particular filter can be specified in the form of a two-dimensional array of those weights. For example, if we were to apply a filter by taking the value of a pixel and all eight of its immediate neighbours, dividing them each by nine and adding them together to obtain the new value for the pixel, we could write the filter in the form:

$$\begin{array}{ccc} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{array}$$

The array of weights is called a *convolution mask* and the set of pixels used in the computation is called the *convolution kernel* (because the equivalent of the multiplication operation that performs filtering in the frequency domain is an operation in the spatial domain called convolution).

Generally, if a pixel has coordinates (x, y), so that it has neighbours at $(x - 1, y + 1)$, $(x, y + 1) \ldots (x, y - 1)$, $(x + 1, y - 1)$, and we apply a filter with a convolution mask in the form:

$$\begin{array}{ccc} a & b & c \\ d & e & f \\ g & h & i \end{array}$$

the value p' computed for the new pixel at (x, y) is

$$
\begin{aligned}
p' &= ap_{x-1,y+1} + bp_{x,y+1} + cp_{x+1,y+1} \\
&\quad + dp_{x-1,y} + ep_{x,y} + fp_{x+1,y} \\
&\quad + gp_{x-1,y-1} + hp_{x,y-1} + ip_{x+1,y-1}
\end{aligned}
$$

where $p_{x,y}$ is the value of the pixel at (x, y), and so on. The process is illustrated graphically in Figure 5.10.

Convolution is a computationally intensive process. As the formula just given shows, with a 3 × 3 convolution kernel, computing a new value for each pixel requires nine multiplications and eight additions. For a modestly sized image of 480 × 320 pixels, the total number of operations will therefore be 1382400 multiplications and 1228800 additions, i.e. over two and a half million operations. Convolution masks need not be only three pixels square — although they are usually square, with an odd number of pixels on each side — and the larger the mask, and hence the kernel, the more computation is required.[9]

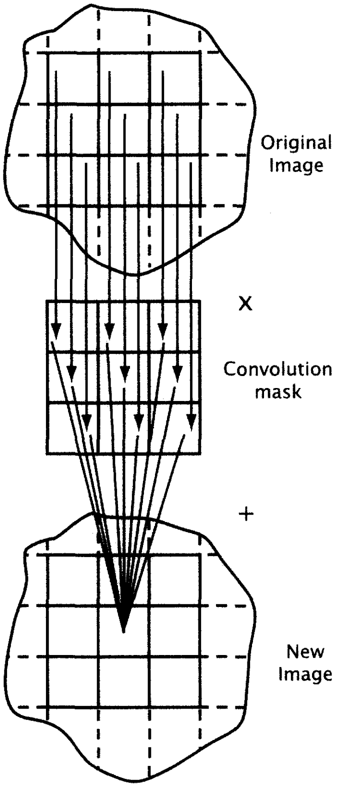


Figure 5.10
Pixel group processing with a convolution mask

This is all very well, but what are the visible effects of spatial filtering? Consider again the simple convolution mask comprising nine values equal to 1/9. If all nine pixels being convolved have the same value, let us say 117, then the filter has no effect: 117/9 × 9 = 117. That is, over regions of constant colour or brightness, this filter leaves pixels alone. However, suppose it is applied at a region including a sharp vertical edge. The convolution kernel might have the following values:

117	117	27
117	117	27
117	117	27

then the new value computed for the centre pixel will be 105. Moving further into the lighter region, to an area that looks like this:

117	27	27
117	27	27
117	27	27

gives a new pixel value of 57. So the hard edge from 117 to 27 has been replaced by a more gradual transition via the intermediate values 105 and 57. The effect is seen as a *blurring*. One way of thinking about what has happened is to imagine that the edges have

[9] Applying certain filters, particularly Gaussian blur (described below) to a large image file is often used to provide 'real world' benchmarking data for new computers.

been softened by rubbing together the colour values of the pixels, in the same way as you blur edges in a pastel drawing by rubbing them with your finger. An alternative view, based on the concepts of signal processing, is that this operation produces a smoothing effect on the spatial waveform of the image, by filtering out high frequencies. (Engineers would refer to the operation as a *low pass filter.*)

Blurring is often used in retouching scans. It is useful for mitigating the effects of digital artefacts, such as the jagged edges produced by undersampling, Moiré patterns, and the blockiness resulting from excessive JPEG compression.

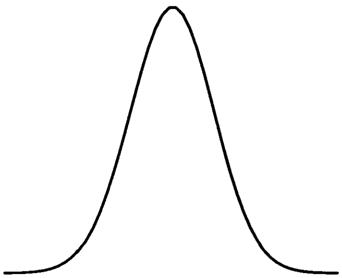


Figure 5.11
The Gaussian bell curve

Although the convolution mask we have just described is a classical blur filter, it produces a noticeably unnatural effect, because of the limited region over which it operates, and the all or nothing effect caused by the uniform coefficients. At the same time, the amount of blurring is small and fixed. A more generally useful alternative is *Gaussian blur*, where the coefficients fall off gradually from the centre of the mask, following the Gaussian 'bell curve' shown in Figure 5.11, to produce a blurring that similar to those found in nature. The extent of the blur — that is, the width of the bell curve, and hence the number of pixels included in the convolution calculation — can be controlled. Photoshop's dialogue allows the user to specify a 'radius' value, in pixels, for the filter. A radius of 0.1 pixels produces a very subtle effect; values between 0.2 and 0.8 pixels are good for removing aliasing artefacts. Higher values are used to produce a deliberate effect. A common application of this sort is the production of drop-shadows: an object is selected, copied onto its own layer, filled with black and displaced slightly to produce the shadow. A Gaussian blur with a radius between 4 and 12 pixels applied to the shadow softens its edges to produce a more realistic effect. A radius of 100 pixels or more blurs the entire image into incoherence; one of 250 pixels (the maximum) just averages all the pixels in the area the filter is applied to. Note that the radius specified is not in fact the limit of the blurring effect, but a parameter that specifies the shape of the bell curve — the blurring extends well beyond the radius, but its effect is more concentrated within it, with roughly 70% of the contribution to the value of the centre pixel coming from pixels within the radius.[10]

Figure 5.12 shows a typical application of Gaussian blur: a scanned watercolour painting has had a small blur, followed by a slight sharpening (see below), applied to remove scanning artefacts. The

[10] The so-called radius is really the standard deviation of the normal distribution corresponding to the bell curve.

result is an image that closely resembles the original, with the filters themselves indiscernible — the blurriness you see is the characteristic spreading of thin watercolour and has nothing to do with the filters. In contrast, the blur in Figure 5.13, with a radius of 29 pixels, transforms the image into something quite different. These pictures, and the remaining illustrations of filters in this section, are reproduced in colour in Plate 14, where the effects can be better appreciated.

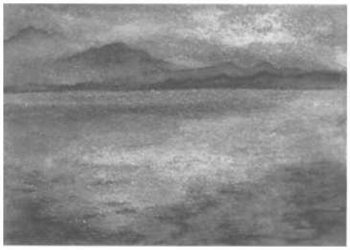


Figure 5.12
A scanned image, corrected by filtering

Other types of blur are directional, and can be used to indicate motion. The final example in Plate 14, also shown in Figure 5.14, shows radial blur with the zoom option, which gives an effect that might suggest headlong flight towards the focal point of the zoom.

Blurring is a surprisingly useful effect when applied to digitized images — you might expect blur to be an undesirable feature of an image, but it conceals their characteristic imperfections; in the case of Gaussian blur, it does this in a visually natural way. Sometimes, though, we want to do the opposite, and enhance detail by sharpening the edges in an image. A convolution mask that is often used for this purpose is:

$$\begin{array}{ccc} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{array}$$

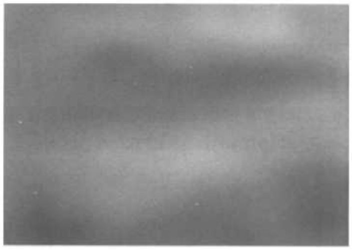


Figure 5.13
A large amount of Gaussian blur

This mask filters out low frequency components, leaving the higher frequencies that are associated with discontinuities. Like the simple blurring filter that removed high frequencies, this one will have no effect over regions where the pixels all have the same value. In more intuitive terms, by subtracting the values of adjacent pixels, while multiplying the central value by a large coefficient, it eliminates any value that is common to the central pixel and its surroundings, so that it isolates details from their context.

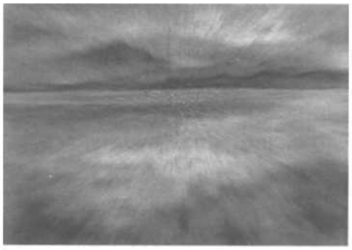


Figure 5.14
Zooming radial blur

If we apply this mask to a convolution kernel where there is a gradual discontinuity, such as

$$\begin{array}{ccc} 117 & 51 & 27 \\ 117 & 51 & 27 \\ 117 & 51 & 27 \end{array}$$

assuming that this occurs in a context where all the pixels to the left have the value 117 and those to the right 27, the new values computed for the three pixels on the central row will be 317, −75 and −45; since we cannot allow negative pixel values, the last two will be set to 0 (i.e. black). The gradual transition will have been replaced by a hard line, while the regions of constant value to either side will be left alone. A filter such as this will therefore enhance detail.

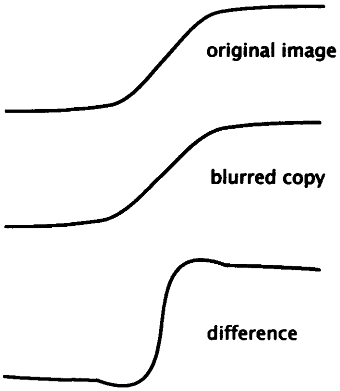


Figure 5.15
Unsharp masking

As you might guess from the example above, sharpening with a convolution mask produces harsh edges; it is more appropriate for analysing an image than for enhancing detail in a realistic way. For this task, it is more usual to use an *unsharp masking* operation. This is easiest to understand in terms of filtering operations: a blurring operation filters out high frequencies, so if we could take a blurred image away from its original, we would be left with only the frequencies that had been removed by the blurring — the ones that correspond to sharp edges. This isn't quite what we usually want to do: we would prefer to accentuate the edges, but retain the other parts of the image as well. Unsharp masking[11] is therefore performed by constructing a copy of the original image, applying a Gaussian blur to it, and then subtracting the pixel values in this blurred mask from the corresponding values in the original multiplied by a suitable scaling factor. As you can easily verify, using a scale factor of 2 leaves areas of constant value alone. In the region of a discontinuity, though, an enhancement occurs. This is shown graphically in Figure 5.15. The top curve shows the possible change of pixel values across an edge, from a region of low intensity on the left, to one of higher intensity on the right. (We have shown a continuous change, to bring out what is happening, but any real image will be made from discrete pixels, of course.) The middle curve illustrates the effect of applying a Gaussian blur: the transition is softened, with a gentler slope that extends further into the areas of constant value. At the bottom, we show (not to scale) the result of subtracting this curve from twice the original. The slope of the transition is steeper, and overshoots at the limits of the original edge, so visually, the contrast is increased. The net result is an enhancement of the edge, as illustrated in Figure 5.16, where an exaggerated amount of unsharp masking has been applied to the original image.

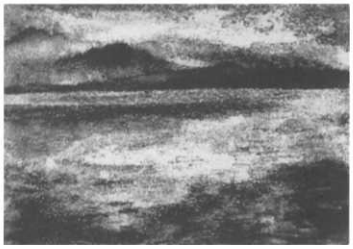


Figure 5.16
Enhancing edges by unsharp masking

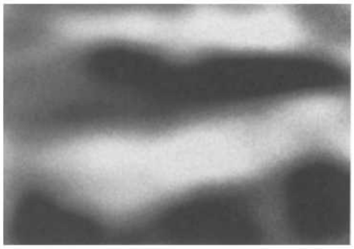


Figure 5.17
Unsharp masking applied after extreme Gaussian blur

The amount of blur applied to the mask can be controlled, since it is just a Gaussian blur, and this affects the extent of sharpening. It is common also to allow the user to specify a threshold; where the difference between the original pixel and the mask is less than the threshold value, no sharpening is performed. This prevents the operation from enhancing noise by sharpening the visible artefacts

[11] Originally a process of combining a photograph with its blurred negative.

it produces. (Notice how in Figure 5.16, the grain of the watercolour paper has been emphasized.)

Although sharpening operations enhance features of an image, it should be understood that they add no information to it. On the contrary, information is actually lost, although, if the sharpening is succesful, the lost information will be irrelevant or distracting. (It's more intuitively obvious that information is lost by blurring an image.) It should also be understood that although, in a sense, blurring and sharpening are opposites, they are not true inverses. That is, if you take an image, blur it and then sharpen it, or sharpen it and then blur it, you will not end up with the image you started with. The information that is lost when these operations are applied cannot be restored, although it is interesting to see features re-emerging in Figure 5.17 when Figure 5.13 is treated with an unsharp mask. This demonstrates how much information is actually preserved even under intense blurring.

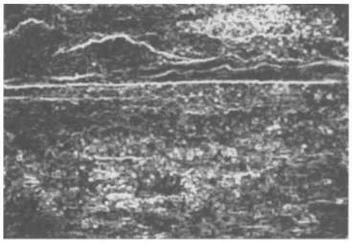


Figure 5.18
Glowing edges

Blurring and sharpening are central to the established scientific and military applications of image processing, but now that image manipulation software is also used for more creative purposes, some rather different effects are called for as well. Photoshop provides a bewildering variety of filters, and third party plug-ins add many more. Many of them are based on the type of pixel group processing we have described, with convolution masks chosen to produce effects that resemble different photographic processes or, with more or less success, the appearance of real art materials. These filters usually work by picking out edges or areas of the same colour, and modifying them; they are not too far removed from the more conventional blur and sharpen operations. Figure 5.18 shows the 'glowing edges' filter's effect on the seascape painting.

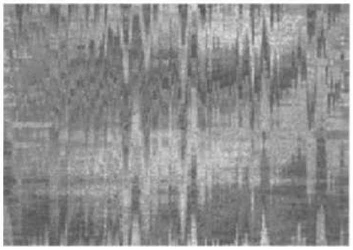


Figure 5.19
Distortion with a square wave

⇨ If you have access to Photoshop, you should investigate the Custom filter (on the Other sub-menu of the Filter menu). This allows you to construct your own convolution mask, by entering coefficients into a 5 × 5 matrix. The results are instructive and sometimes surprising.

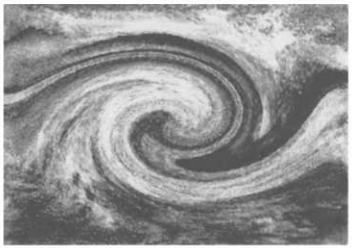


Figure 5.20
Twirled image

Another group of filters is based on a different principle, that of moving selected pixels within an image. These produce various sorts of distortion. Figure 5.19 shows the seascape image modified by a square-wave pattern, while Figure 5.20 shows the twirl filter applied to its sharpened version from Figure 5.15, producing an entirely new image. As this example indicates, filters may be combined. It is not untypical for designers with a taste for digital effects to combine many different filters in order to generate imagery that could not easily be made any other way.

Geometrical Transformations

Scaling, translation, reflection, rotation and shearing are collectively referred to as *geometrical transformations.* As we saw in Chapter 4, these transformations can be be applied to vector shapes in a very natural way, by simply moving the defining points according to geometry and then rendering the transformed model. Applying geometrical transformations to bitmapped images is less straightforward, since we have to transform every pixel, and this will often require the image to be resampled.

Nevertheless, the basic approach is a valid one: for each pixel in the image, apply the transformation using the same equations as we gave in Chapter 4 to obtain a new position in which to place that pixel in the transformed image. This suggests an algorithm which scans the original image and computes new positions for each of its pixels. An alternative, which will often be more successful is to compute the transformed image, by finding, for each pixel, a pixel in the original image. So instead of mapping the original's coordinate space to that of the transformed image, we compute the inverse mapping. The advantage of proceeding in this direction is that we compute all the pixel values we need and no more. However, both mappings run into problems because of the finite size of pixels.

For example, suppose we wish to scale up an image by a factor s. (For simplicity, we will assume we want to use the same factor in both the vertical and horizontal directions.) Thinking about the scaling operation on vector shapes, and choosing the inverse mapping, we might suppose that all that is required is to set the pixel at coordinates (x', y') in the enlarged image to the value at $(x, y) = (x'/s, y'/s)$ in the original. In general, though, x'/s and y'/s will not be integers and hence will not identify a pixel. Looking at this operation in the opposite direction, we might instead think about taking the value of the pixel at coordinates (x, y) in our original and mapping it to $(x', y') = (sx, sy)$ in the enlargement. Again, though, unless s is an integer, this new value will sometimes fall between pixels. Even if s is an integer only some of the pixels in the new image will receive values. For example, if $s = 2$, only the even numbered pixels in even numbered rows of the image correspond to any pixel in the original under this mapping, leaving three quarters of the pixels in the enlarged image undefined. This emphasizes that, in constructing a scaled-up image we must use some interpolation method to compute values for some pixels.

However, it is not just in scaling up that interpolation is required. Whenever we appy a geometrical transformation to an image, it can send values into the gaps between pixels. Consider, for example, something as simple as moving a selected area of an image stored at 72 ppi one fifth of an inch to the right. Even scaling an image *down* can result in the same phenomenon unless the scale factor is a whole number. It should be clear from our discussion earlier in this chapter that changing the resolution of an image leads to similar problems.

A useful way of thinking about what is going on is to imagine that we are reconstructing a continuous image, so that we can find the required values in between the pixels of our sampled image, and then resampling it. Thus, the general problem is the same as the one we introduced when we discussed digitization in Chapter 2: how to reconstruct a signal from its samples. In practice, of course, we combine the reconstruction and resampling into a single operation, because we can only work with discrete representations.

We know that, for general images which may contain arbitrarily high frequencies because of sharp edges, the reconstruction cannot be done perfectly. We also know from sampling theory that the best possible reconstruction is not feasible. All we can hope to do is approximate the reconstruction to an acceptable degree of accuracy by using some method of *interpolation* to deduce the intervening values on the basis of the stored pixels. Several interpolation schemes are commonly employed; Photoshop provides three, for example, which we will describe next. As is usual in computing, the more elaborate and computationally expensive the algorithm used, the better the approximation that results.

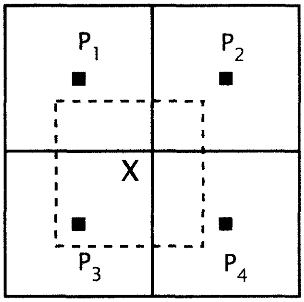


Figure 5.21
Pixel interpolation

Suppose that we are applying some geometrical transformation, and we calculate that the pixel at the point (x', y') in the resulting image should have the same value as some point (x, y) in the original, but x and y are not integers. We wish to sample the original image at (x, y), at the same resolution at which it is stored, so we can imagine drawing a pixel — call it the *target pixel* — centred at (x, y) which will be the sample we need. As Figure 5.21 shows, in general this pixel may overlap four pixels in the original image. In the diagram, X marks the centre of the target pixel, which is shown dashed; P_1, P_2, P_3, and P_4 are the surrounding pixels, whose centres are marked by the small black squares.

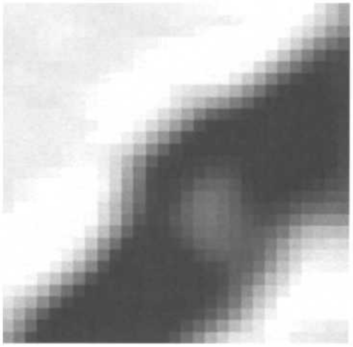


Figure 5.22
Nearest neighbour interpolation

The simplest interpolation scheme is to use the *nearest neighbour*, i.e. we use the value of the pixel whose centre is nearest to (x, y), in this case P_3. In general — most obviously in the case of upsampling or enlarging an image — the same pixel will be chosen as the nearest neighbour of several target pixels whose centres, although different, are close together enough to fall within the same pixel. As a result, the transformed image will show all the symptoms of undersampling, with visible blocks of pixels and jagged edges. An image enlarged using nearest neighbour interpolation will, in fact, look as if its original pixels had been blown up with a magnifying glass.

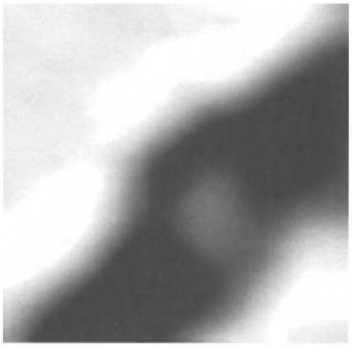


Figure 5.23
Bi-linear interpolation

A better result is obtained by using *bi-linear interpolation*, which uses the values of all four adjacent pixels. They are combined in proportion to the area of their intersection with the target pixel. Thus, in Figure 5.21, the value of P_1 will be multiplied by the area enclosed by the dashed lines and the solid intersecting lines in the north-west quadrant, and added to the values of the other three pixels, multiplied by the corresponding areas.

➲ If a and b are the fractional parts of x and y, respectively, then some simple mathematics shows that the value of the pixel at (x', y') in the result, whose target pixel is centred at (x, y) will be equal to

$$(1-a)(1-b)p_1 + a(1-b)p_2 + (1-a)bp_3 + abp_4$$

where p_i is the value of the pixel P_i, for $1 \leq i \leq 4$.

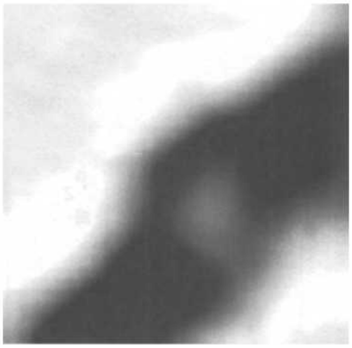


Figure 5.24
Bi-cubic interpolation

This simple area calculation is implicitly based on the assumption that the values change linearly in both directions (hence '*bi*-linearly') across the region of the four pixels. An alternative way of arriving at the same value is to imagine computing the values vertically above and below (x, y) by combining the values of the two pairs of pixels in proportion to their horizontal distances from x, and then combining those values in proportion to their vertical distances from y. In practice, the values are unlikely to vary in such a simple way, so that the bi-linearly interpolated values will exhibit discontinuities. To obtain a better result, *bi-cubic interpolation* can be used instead. Here, the interpolation is based on cubic splines,[12] that is, the intermediate values are assumed to lie along a Bézier curve connecting the stored pixels, instead of a straight line. These are used for the same reason they are used for drawing curves: they join together smoothly. As a result, the resampled image is smooth as well.

Bi-cubic interpolation does take longer than the other two methods but relatively efficient algorithms have been developed and modern machines are sufficiently fast for this not to be a problem on single images. The only drawback of this interpolation method is that it can cause blurring of sharp edges.

Figures 5.22 to 5.24 show the same image enlarged using nearest neighbour, bi-linear, and bi-cubic interpolation.

[12] We will not try to intimidate you with the equations for bi-cubic interpolation.

Further Information

[Bax94] is a good introduction to image manipulation. [NG96] describes many compression algorithms, including JPEG. [FvDFH96] includes a detailed account of re-sampling.

Exercises

1. Suppose you want to change the size of a bitmapped image, and its resolution. Will it make any difference which order you perform these operations in?
2. On page 126 we suggest representing the first 4128 pixels of the image in Figure 3.1 by a single count and value pair. Why is this not, in general, a sensible way to encode images? What would be a better way?
3. Our argument that no algorithm can achieve compression for all inputs rests on common sense. Produce a more formal proof, by considering the number of different inputs that can be stored in a file of N bytes.
4. We lied to you about Plate 4: the original painting was made on black paper (see Figure 5.25). In order to make it easier to reproduce, we replaced that background with the blue

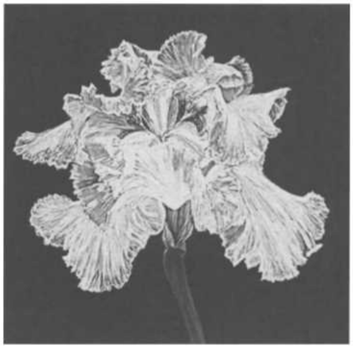


Figure 5.25
The original iris scan

gradient you see in the colour plate. Describe as many ways as you can in which we might have done this.

5. Describe how you would convert a photograph into a vignette, such as the one shown in Plate 13, by adding an oval border that fades to white, in the fashion of late nineteenth century portrait photographers. How would you put an ornamental frame around it?

6. Explain why it is necessary to use an alpha channel for anti-aliased masks.

7. Compare and contrast the use of alpha channels and layer transparency for compositing.

8. Describe how the input-output curve of an image should be changed to produce the same effect as moving (a) the brightness, and (b) the contrast sliders. How would these adjustments affect the histogram of an image?

9. Describe the shape of the curve you would use to correct an image with too much contrast. Why would it be better than simply lowering the contrast with the contrast slider?

10. If asked to 'sharpen up' a scanned image, most experts would first apply a slight Gaussian blur before using the sharpen or unsharp mask filter. Why?

11. Motion blur is the smearing effect produced when a moving object is photographed using an insufficiently fast shutter speed. It is sometimes deliberately added to images to convey an impression of speed. Devise a convolution mask for a motion blur filter. How would you allow a user to alter the amount of motion blur? What other properties of the blurring should be alterable?

12. Why are the screen shots published in tutorial articles in computing magazines often hard to read?

13. Explain carefully why pixel interpolation may be required if a rotation is applied to a bitmapped image.

14. An alternative to using bi-linear or bi-cubic pixel interpolation when downsampling an image is to apply a low-pass filter (blur) first, and then use the nearest neighbour. Explain why this works.

Colour

6

Both vector graphics and bitmapped images can use colour. For most people, colour is such a commonplace experience that it is somehow surprising to discover that it is actually a rather complex phenomenon, in both its objective and subjective aspects. Representing colour in digital images and reproducing it accurately on output devices are consequently not at all straightforward.

Perhaps the most important thing to realize about colour is that you don't always need it. The existence of black and white photography and film demonstrates that people are perfectly well able to recognize and understand an image in the absence of colour — variations in brightness are quite adequate. Indeed, comparing the luminous elegance of a Fred Astaire film from the 1930s with the garish Technicolor of a 1950s MGM musical is enough to demonstrate that the addition of colour is not necessarily an improvement — a fact well understood by advertising agencies and the makers of music videos. Pragmatically, there are advantages to using images without colour. As we will see shortly, black and white bitmapped image files can be much smaller than coloured ones. Furthermore, black and white images are largely immune to the variations in colour reproduction of different monitors. You should not forget that some people do not have colour monitors, or prefer to use them in monochrome mode, and that a few people cannot see in colour at all, while many more cannot always distinguish between certain colours. By working in black and white (or, more accurately, shades of grey) you avoid producing images that may not be seen properly for these reasons.

But people have come to expect colour, and colour can add a great deal to an image, if it is used effectively. Sometimes colour is vital to the purpose for which an image is being used: for example, the colours in a clothing catalogue will influence people's buying decisions, and must be accurate to avoid disappointment and complaints.

The effective use of colour is not something that can be summarized in a few rules. The experience of artists and designers working in traditional media can be put to good use, and in large-scale multimedia production it is wise to ensure that there are artistically trained specialists available, and to leave colour decisions to them. As in most of the other areas we will describe, certain constraints associated with the digital nature of the images we can use in our multimedia productions will modify how we approach colour, so artistic sensibility must be augmented with some technical knowledge.

Colour and Science

Colour is a subjective sensation produced in the brain. In order to reproduce colour electronically, or manipulate it digitally, we need a model of colour which relates that subjective sensation to measurable and reproducible physical phenomena. This turns out to be a surprisingly difficult task to accomplish successfully.

Since light is a form of electromagnetic radiation, we can measure its wavelength — the wavelength of visible light lies roughly between 400nm and 700nm — and its intensity. We can combine these measurements into a *spectral power distribution (SPD)*, a description of how the intensity of light from some particular source varies with wavelength. Figure 6.1 shows the SPD of typical daylight. (Notice that it extends beyond the visible spectrum). In effect, an SPD is constructed by splitting light into its component wavelengths, much as a prism splits a beam of light into a spectrum when we repeat Isaac Newton's optics experiments in school, and measuring the intensity of each component.[1] Subjective experiments show that an SPD corresponds closely to what we mean by 'colour', in the sense that observers can successfully match light with a particular SPD to

[1] In theory, an SPD ought to be a continuous function, but it is satisfactorily approximated by using samples at wavelengths separated by a suitable interval, for example 10nm, giving an SPD consisting of 31 components.

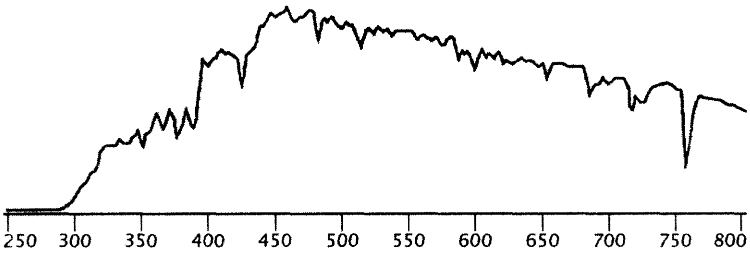


Figure 6.1
The spectral power distribution of daylight

a specific colour. However, SPDs are too cumbersome to work with when we are specifying colours for use in computer graphics, so we need to adopt a different approach.

You may have been told at some point in your education that the human eye contains two different sorts of receptor cells: *rods*, which provide night-vision and cannot distinguish colour, and *cones*, which in turn come in three different sorts, which respond to different wavelengths of light. The fact that our perception of colour derives from the eye's response to three different groups of wavelengths leads to the theory — called the *tristimulus* theory — that any colour can be specified by just three values, giving the weights of each of three components.

The tristimulus theory of colour is often summarized (inaccurately) by saying that each type of cone responds to one of red, green or blue light. It follows that the sensation of any colour can be produced by mixing together suitable amounts of red, green and blue light. We call red, green and blue the *additive primary colours*.[2] Putting that another way, we can define a particular colour by giving the proportions of red, green and blue light that it contains. It follows that we can construct television screens and computer monitors using pixels each made up of three dots of different types of phosphor, emitting red, green and blue light, and exciting them using three electron beams, one for each colour. To produce any desired colour we just have to adjust the intensity of each electron beam, and hence the intensity of the light emitted by the corresponding phosphors. Optical mixing of the light emitted by the three component dots of any pixel will make it look like a single pixel of the desired colour.

Since we *can* construct monitors like that, the simplified tristimulus theory is evidently more or less right, and much of the time we

[2] These are not the artist's primary colours, which will be described later.

can proceed on the basis of it. However, it *is* a simplified theory, and some subtle problems can arise if we ignore the more complex nature of colour. Fortunately, the worst of these are connected with printing, and do not often occur in multimedia work. Reproducing colour on a computer monitor, as we more often need to do, is more straightforward.

RGB Colour

The idea that colours can be constructed out of red, green and blue light leads to the *RGB colour model*, in which a colour is represented by three values, giving the proportions of red (R), green (G) and blue (B) light[3] which must be combined to make up light of the desired colour. The first question that arises is 'What do we mean by red, green and blue?'. The answer ought to be that these are the names of three colours corresponding to three standard primary SPDs, and in the television and video industries several such standards do exist. In computing, however, there is no universally accepted standard, although monitors are increasingly being built to use the primaries specified for High Definition TV (HDTV) by the ITU in its Recommendation ITU-R BT.709, and a standard version of RGB colour derived from this recommendation has been proposed. However, there is no real standard for red, green and blue on computer displays, and the colours produced in response to the any particular RGB value can vary wildly between monitors.

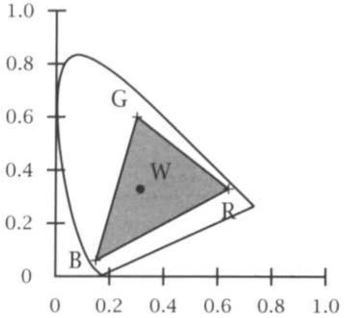


Figure 6.2
The RGB colour gamut

It is also important to be aware that it is *not* possible to represent any visible colour as a combination of red, green and blue components, however defined. Figure 6.2 shows a pictorial representation of the relationship between the colours that can be represented in that way — the so-called RGB *colour gamut* — and all the visible colours. The fin-shaped area is a spatial representation of all the possible colours. (For an explanation of exactly what is being plotted see the detailed comments below.) Shades of green lie towards the tip, with reds at the bottom right and blues at the bottom left. The triangular area of the RGB colour gamut is entirely enclosed within the fin, showing that there are some colours that cannot be produced by adding red, green and blue. It should be noted, though, that red, green and blue primaries do produce the largest gamut possible from simple addition of three primaries. Putting that another way, if you were to draw the largest triangle you

[3] The primary blue is actually better described as 'blue-violet', and the red is an orangey shade.

could fit inside the area of all possible colours, its vertices would correspond to colours you would call red, blue and green.

Exactly what is being plotted in Figure 6.2? For a full explanation, you will have to consult the references given at the end of this chapter, but a rough outline is as follows. The diagram derives from work done under the auspices of the *Commission Internationale de l'Éclairage (CIE)* in 1931. In experiments in which people were asked to mix quantities of red, green and blue lights to produce a colour that matched a displayed sample, it was found that it was necessary to add negative amounts of primaries (by adding them to the displayed sample instead of the attempted match) in order to match some colours. This is now understood to be because the responses of the three type of cone cell are not simply used as colour stimuli, the way the voltages on the three electron guns of a monitor are, but are combined in a more complex way. The CIE defined a set of three primaries, known simply as X, Y and Z, whose spectral distributions matched the inferred spectral response of the eye of their 'standard observer', and could thus be used to produce any visible colour purely by addition. The Y component is essentially the brightness (more properly called *luminance* in this context). If we put

$$x = \frac{X}{X+Y+Z}, y = \frac{Y}{X+Y+Z} \text{ and } z = \frac{Z}{X+Y+Z}$$

you should be able to see that any colour can be fully specified by its x, y and Y values, since X and Z can be recovered from the equations. (Note that z is redundant.) Since Y is the luminance, x and y together specify the colour, independently of its brightness. It is x and y that label the axes in Figure 6.2. (This diagram is called the *CIE chromaticity diagram*, because it plots colour information, independent of brightness.) The curved area is obtained by plotting the value of an SPD comprising a single wavelength as its value varies from 400nm to 700nm. The area is closed by a straight line joining the extreme low and high frequencies — the 'line of purples', which corresponds to those colours which do not have a single identifiable wavelength as their dominant spectral component. The RGB gamut is contained inside a triangle with red, green and blue (as defined by the CIE) at its vertices.

Unfortunately, the primaries X, Y and Z, cannot be realized by physical light sources.

In practice, the majority of colours perceived in the world do fall within the RGB gamut, so the RGB model provides a useful, simple and efficient way of representing colours. A colour can be represented by three values. We can write this representation in the form (r,g,b), where r, g and b are the amounts of red, green and blue light making up the colour. By 'amount', we mean the proportion of pure ('*saturated*') light of that primary. For example, if we express the proportions as percentages,(100%, 0%, 0%) represents pure saturated primary red, and (50%, 0%, 0%) a darker red, while (100%, 50%, 100%) represents a rather violent shade of mauve. Since black is an absence of light, its RGB colour value is (0%, 0%, 0%). White light is produced by mixing equal proportions of saturated light of all three primaries, so white's RGB colour value is (100%, 100%, 100%). We emphasize that the three values represent the amounts of *light* of the three primary colours which must be mixed to produce light of the specified colour. Do not confuse this *additive mixing* of colours with paint mixing, which is a *subtractive mixing* process, since paint absorbs light.[4] Computer monitors emit light, so any consideration of how colours are produced must be based on an additive model. Scanners work by detecting light reflected from the scanned document, so they too work with additive colours.

The three numbers r, g and b are not absolute values in any sense, it is only their relative values that matter, so we can choose any convenient range, provided it allows us to distinguish enough different values. Like many important questions about colour, 'How many is enough?' can only be answered subjectively. Different cultures have different ideas about when two colours are different, and people differ in their ability to distinguish between different colours, but few, if any, can distinguish more than the nearly 16.8 million distinct combinations provided by using 256 different values for each of the red, green and blue component values.

[4] This is a simplification — see below.

Colour Depth

Of course, 256 is a very convenient number to use in a digital representation, since a single 8-bit byte can hold exactly that many different values, usually considered as numbers in the range 0 to 255. Thus, an RGB colour can be represented in three bytes, or 24 bits. The number of bits used to hold a colour value is often referred to as the *colour depth*: when three bytes are used, the colour depth is 24. We often also refer to *24 bit colour*, as a shorthand for colour with a 24 bit colour depth. In 24 bits, our mauve shade would be stored as (255, 127, 255), black is (0, 0, 0), as always, and white is $(255, 255, 255)$. (Remember that the values start at zero).

You may well suppose that 24 is not the only possible colour depth, and this is indeed the case. One other possibility, less common than it used to be, is 1-bit (*bi-level*) colour. A single bit allows us to distinguish two different colours. Usually these will be considered black and white, although the actual colours displayed will depend on the output device — orange and black monitors enjoyed a vogue for a while on ergonomic grounds, for example. A colour depth of 4 bits allows 16 different colours, which is clearly inadequate for displaying anything but the most simple of colour images; 16 different grey levels, on the other hand, can produce respectable greyscale images. Since greys are represented by RGB colour values (r, g, b), with $r = g = b$, a single value is sufficient to specify a grey level, the other two RGB components are redundant.

The 8 bit colour depth, where a single byte is used to hold a colour value, providing 256 colours, is in widespread use, both in monitors for domestic personal computers and on the World Wide Web. It affords several points of particular interest, and we will return to it shortly. Some computer systems use 16 bits to hold colour values. 16 is not divisible by 3, so when RGB values are stored in a 16 bit format, either one bit is left unused, or different numbers of bits are assigned to the three components. Typically, red and blue each use 5 bits, and green is allocated 6 bits, allowing twice as many different green values to be distinguished. This allocation of bits is based on the observation that the human eye is more sensitive to green light than to the other two primaries. (The cone cells in the eye are especially insensitive to blue, to compensate for the high levels of blue in daylight. The high sensitivity to green is presumably an evolutionary response to the preponderance of green in the environment in which human beings evolved.)

Although 24 bits are sufficient to represent more colours than the eye can distinguish, higher colour depths, such as 30, 36 or even 48 bits are increasingly being used, especially by scanners. Support for 48 bit colour is included in the specification of the PNG file format. These very large colour depths serve two purposes. Firstly, the additional information held in the extra bits makes it possible to use more accurate approximations when the image is reduced to a lower colour depth for display (just as images stored at high resolution will look better when displayed on a low resolution monitor than images stored at the monitor resolution). Secondly, it is possible to make extremely fine distinctions between colours, so that effects such as chroma-key (see Chapter 10) can be applied very accurately.

The common colour depths are sometimes distinguished by the terms *millions of colours* (24 bit), *thousands of colours* (16 bit) and *256 colours* (8 bit), for obvious reasons (although 16 bits allows 65,536 values, so tens of thousands of colours would be a more accurate description). The names *true colour* and *hi colour* are sometimes used for 24 and 16 bit colour, respectively, but they are often used more loosely by marketing people. We will avoid these terms.

Colour depth is a crucial factor in determining the size of a bitmapped image: each logical pixel requires 24 bits for 24 bit colour, but only 8 for 8 bit, and just a single bit for 1 bit colour. Hence, if it possible to reduce the colour depth of an image from 24 bits to 8, the image file size will decrease by a factor of three (ignoring any fixed-size housekeeping information that is held in addition to the image data). Returning to an earlier point, an image made up of 256 different shades of grey — more than most people can distinguish — will be one third the size of the same image in millions of colours. Using an arbitrarily chosen set of 256 colours, on the other hand, is unlikely to produce an acceptable result. If the colour depth must be reduced, and grey-scale is not desired, then an alternative strategy must be employed.

Indexed Colour

So far, we have implicitly assumed that the stored R, G and B values are used directly to control the intensity of the monitor's three electron beams, thus determining the colour that is displayed. Often this is just what is done, especially in the case of 24 bit colour.[5] This arrangement is called *direct colour*. There is, however, an alternative, which is very widely used on low-end computer systems and the World Wide Web, known as *indexed colour*.

The screen of any colour monitor is capable of displaying the millions of colours that can be represented by 24-bit colour values, but it may well be the case that the video RAM (VRAM) provided in the monitor is not sufficient to hold a full-screen image at 3 bytes per pixel. (Table 6.1 shows the amount of VRAM needed to support each colour depth at a selection of common screen sizes.) Another

[5] The relationship between the voltage applied to an electron beam and the intensity of the light emitted by a phosphor when it strikes it is non-linear, as is the response of the eye to the intensity of light which enters it, so the value is not simply used as a voltage, but the hardware is built to compensate somewhat for these non-linearities.

Table 6.1
Colour depth and VRAM

VRAM (Mbytes)	Screen Resolution	Max Colour Depth
1	640 × 480	16
	832 × 624	16
	1024 × 768	8
	1152 × 870	8
2	640 × 480	24
	832 × 624	24
	1024 × 768	16
	1152 × 870	16
4	640 × 480	24
	832 × 624	24
	1024 × 768	24
	1152 × 870	24
	1280 × 1024	16
8	640 × 480	24
	832 × 624	24
	1024 × 768	24
	1152 × 870	24
	1280 × 1024	24

constraint on colour depth is that an image file in 24-bit colour may be considered too large for the available disk space, or it may be felt to take too long to transmit over a network. If, for any of these reasons, we are constrained to use only one byte for each pixel, we can use at most 256 different colours in any one image. Using a standard set of 256 colours for all the images we might need, i.e. attempting to use 8-bit direct colour, is unacceptably restrictive. Indexed colour provides a means of associating a *palette* of 256 specific colours with each image. For example, if we wished to produce a pastiche of Picasso's 'blue period' paintings, we could use a palette holding shades of blue and obtain a reasonable result. If we had had to use only the blues from a set of 256 colours spread evenly over the spectrum, all of the subtlety and possibly much of the content would be lost.

One way of thinking about indexed colour is as a digital equivalent of painting by numbers. In a painting by numbers kit, areas on the painting are labelled with small numbers, which identify pots of paint of a particular colour. When we use indexed colour, pixels store a small number that identifies a 24-bit colour from the palette associated with the image. Just as each painting by numbers outfit includes only those paints that are needed to colour in one picture, so the palette includes only the 24-bit RGB values for the colours used in the image. When the image is displayed, the graphics system looks up the colour from the palette corresponding to each single-byte value stored at each pixel, and uses the value it finds as the colour of that pixel.

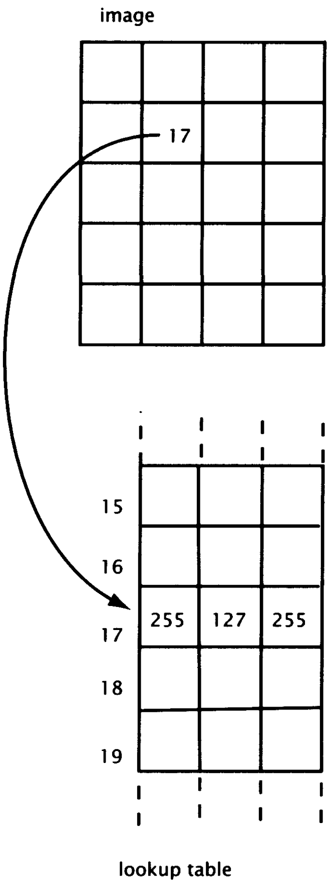


Figure 6.3
Using a colour lookup table

Experienced programmers will immediately recognize the strategy: it is an example of the folk axiom that all problems can be solved by adding a level of indirection. Hence, instead of trying to use eight bits to hold an RGB value, we use it to hold an index into a table, with 256 entries, each of which holds a full 24-bit RGB colour value. For example, if a particular pixel was to be displayed as the colour whose 24-bit RGB value was (255, 127, 255), instead of storing that value in the image, we could store an index identifying the table location in which it was stored. Supposing (255, 127, 255) was held in the 18th entry of the table, the pixel would hold the offset 17 (see Figure 6.3). Such an indexed table of colour values is called a *Colour Lookup Table (CLUT)* or, more simply, a *palette.* You may find it helpful to consider the CLUT as carrying out a mapping from *logical colours*, stored for logical pixels, to *physical colours*, displayed by physical pixels. The looking up operation that performs this mapping is usually carried out by the monitor's video hardware; all the displaying program needs to do is load the correct palette. (Even if the display hardware supports 24-bit direct colour, indexed colour may sometimes be used to reduce the size of image files; in that case, the CLUT lookup may be done by software.) If no palette is supplied, a default system palette will be used (the system palettes on the major platforms are different).

It may help you to understand what is going on when indexed colour is being used if you consider what happens if a program displays two images in different windows at the same time, and those images use different palettes. Unless the program has been designed to operate in an unusual way, the colour palette associated with the active window will be loaded. Say this palette is mostly made up of shades of blue, and the 5th entry is a pale sky-blue. Then every pixel with a value of 4 (palette offsets start at 0) will be coloured that pale sky-blue — including pixels in the other image, if its window is still visible. This other image may be made up of shades of brown, with the 5th element of its palette a reddish-brown; nevertheless, while the blue image's window is active, the pixels that should be reddish-brown will be pale sky-blue, since, although the palette is associated with the image, it is applied to the entire screen. Hence, if the brown image becomes active, the pale sky-blues in the first window will turn reddish-brown. A logical colour value does not identify a colour absolutely, only relative to a palette. You will appreciate that there is a problem if two images are to be displayed in the same window, as they must be if they both occur on the same Web page. It will be necessary to use a single palette to display both. This may require the number of colours in each individual image to be restricted to an even smaller number, unless the colour make-up of both images is the same.

If indexed colour is to be used an image file needs to store the palette along with the actual image data, but the image data itself can be smaller by a factor of three, since it only needs eight bits per pixel. For any reasonably sized image, there will be a net saving of space. This saving may be increased if the file format allows palettes with fewer than 256 entries to be stored with images that have fewer colours, and uses the minimum number of bits required to index the palette for each colour value.

Of the file formats mentioned in Chapter 3, PNG, BMP, TGA and TIFF allow the use of a colour palette; GIF requires it as it only supports 8-bit indexed colour images; an SGI image file may have a palette associated with it in a separate file. JFIF and SPIFF files do not support indexed colour: JPEG images stored in 24-bit colour must be reduced to 8-bit if necessary at the time they are displayed. PostScript and its derivatives, such as EPS and PDF, provide sophisticated support for indexed colour, although its use is usually confined to bitmapped images embedded in the PostScript, since its use with vector shapes, where the colour specification is a small part of the image model, is less compelling.

For created images, a palette of 256 colours will often be tolerable and, since you have control over the colours you use, you can work within the restriction, though subtle variations will be lost. However, with photographic or scanned material, or images that already exist in 24-bit colour, it will be necessary to cut down the number of colours when you prepare a version that uses 8-bit indexed colours. What can be done with the areas of the image that should be displayed in some colour which is not in the reduced palette?

Obviously, the missing colour must be replaced by one of those that is in the palette. There are two popular ways of doing this. The first is to replace the colour value of each individual pixel with the CLUT index of the nearest colour. This can have undesirable effects: not only may the colours be distorted, but detail may be lost when two similar colours are replaced by the same one, and banding and other visible artefacts may appear where gradations of colour are replaced by sharp boundaries. (These effects are collectively known as *posterization*; posterization is sometimes used deliberately as an effect, for example, to simulate cheaply printed late 1960s underground posters.)

> What do we mean by 'nearest' colour? A simple-minded view is that the colour nearest to some value (r,g,b) is the one with RGB value (r',g',b') such that $\sqrt{(r'-r)^2+(g'-g)^2+(b'-b)^2}$ (the Euclidean distance between the two colour values) is minimized. This computation does not take account of the non-linearity of our perceptions of colour differences, so some correction should be applied to the given formula, although computer graphics programs often fail to do so.

When posterization is unacceptable, the replacement of colours missing from the palette can be done another way. Areas of a single colour are replaced by a pattern of dots of several different colours, in such a way that optical mixing in the eye produces the effect of a colour which is not really present. For example, say our image includes an area of pale pink, but the palette that we must use does not include that colour. We can attempt to simulate it by colouring some of the pixels within that area red and some white. This process is called *dithering*.[6] It is done by grouping together pixels in the area to be coloured, and applying colour to individual pixels within each group in a suitable pattern. The process is an extension of the use of half-toning, used to print greyscale images on presses that can only produce dots of pure black or white. At low resolutions, dithering may produce poor results, so it is better suited to high resolution work.

Figure 6.4 shows five distinct ways of distributing black and white among a 2×2 block of pixels. If the pixels were small enough and the image was viewed from a sufficient distance, these patterns would be interpreted as five different shades of grey (including black and white) — see Figure 6.5, where we have exaggerated the size of the pixels to show how the half-tones are built up. In general, a block of $n \times n$ pixels can simulate $n^2 + 1$ different grey levels, so you can see that as the size of the block over which we dither is increased, so is the number of grey levels, but at the expense of resolution.

If each pixel can be one of 256 different colours, instead of just black or white, then the corresponding patterns can simulate

[6] You may sometimes see the term 'dithering' used incorrectly just to mean reducing the number of colours.

millions of colours. However, these simulated colours are, in effect, being applied to pixels four times the area of those actually on the screen, hence the effective resolution of the image is being halved, resulting in a loss of detail. While this is usually acceptable for printing, where resolutions in excess of 600dpi are common, it is often intrusive when images are being displayed on 72dpi monitors. Other artefacts may also be generated when the patterns are superimposed; these can be minimized by clever choice of dot patterns for the generated colours, although this has the effect of cutting down the number of different colours that can be simulated.

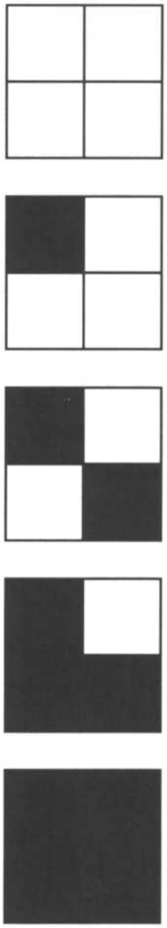


Figure 6.4
Dithering patterns

This leaves the question of which colours should be used in a palette. Ideally, you will fill the palette with the most important colours in your image. Often, these will be the most common, in which case it is easy for a program like Photoshop to construct the palette automatically by examining the original 24-bit version when it converts it to indexed colour. Sometimes, though, the use of colour will be more complex, and it may be necessary to construct the palette by hand (or, more likely, edit an automatically constructed palette) to ensure that all the vital colours are present.

You cannot guarantee that every program that displays your image will necessarily use your palette — some may just fall back on the default system palette, with dismaying results. If you have reason to believe that that is a possibility, it is usually better to convert to a system palette yourself when preparing the image, and make any adjustments you can to retrieve some quality instead of letting the system do its own colour conversion. Unfortunately, the system palettes for the major platforms are different. A restricted set of 216 colours, usually referred to as the *Web-safe palette*, is the only palette you can rely on to be reproduced by Web browsers on any system using 8-bit colour, and if you must know which colours are going to be displayed on any platform the Web-safe palette is your best option.

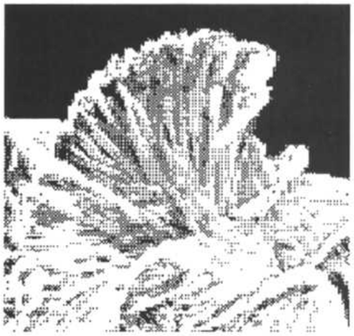


Figure 6.5
Grey produced by dithering

Plate 15 summarizes this section. The left-hand column shows, at the top, a 24-bit colour image, with, below it, the same image using a custom palette of its 256 most common colours; the palette is shown at the bottom. There is little discernible difference between the two images. The top picture in the middle column shows the same image again, this time using the Web-safe palette, with dithering applied. Again, and perhaps surprisingly, there is little discernible difference, but, if you look closely, you can see that the dithering has introduced some slight pixelation. Below this, is the same image, using the same palette, but without dithering, now badly posterized; the Web-safe palette is shown below. In the final column, the top image is a greyscale version, showing how much detail is retained despite the absence of colour. The lower image is indexed colour again, this time using the palette shown below it, which was constructed from the most common colours in the image that is reproduced as Plate 17.

Other Colour Models

The RGB colour model is the most important means of representing colours used in images for multimedia, because it corresponds to the way in which colour is produced on computer monitors, and it is also how colour is detected by scanners. Several other colour models are in use, and you should know something about them because you may be required to work with images that have already been prepared for some other medium, especially print, or to use software which is based on a different view of colour from RGB.

CMYK

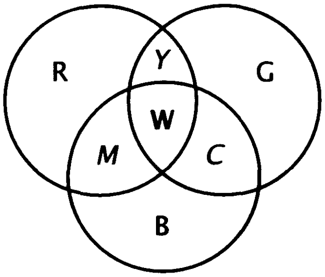


Figure 6.6
Complementary colours

You have probably seen at some time the experiment, originally devised by Thomas Young in 1801, illustrated in Plate 16. Beams of the three additive primaries red, green and blue are shone onto a white surface, which reflects all of the light falling on it, so that they overlap. Where only one colour lands, we see that colour; where all three overlap, we see white, as we would expect from the previous section. Where two colours overlap, we see a new colour formed by the mixing of the two additive primaries. These new colours are a pale shade of blue, usually called *cyan*, a slightly blueish red, called *magenta*, and a shade of *yellow* (see Figure 6.6). Each of these three is formed by adding two of the additive primaries. Since all three additive primaries combine to form white light, we could equally say that cyan, for example, which is the mixture of blue and green, is produced by subtracting the remaining primary, red, from white light.

We can express this effect pseudo-algebraically. Writing R, G and B for red, green and blue, C, M and Y for cyan, magenta and yellow,[7] and W for white, and using + to mean additive mixing of light,

[7] The Y that stands for yellow is not to be confused with the Y that stands for luminance elsewhere in this chapter.

and − to mean subtraction of light, we have

$$\begin{array}{lllll} C & = & G+B & = & W-R \\ M & = & R+B & = & W-G \\ Y & = & R+G & = & W-B \end{array}$$

In each equation, the colour on the left is called the *complementary colour* of the one at the extreme right; for example, magenta is the complementary colour of green.

The relevance of this experiment is two-fold. Firstly, it is the basis for a theory of colour aesthetics which has had a great influence on the use of colour in art and design. Secondly, the idea of forming colours by subtraction of light instead of addition provides a colour model appropriate to ink and paint, since these are substances which owe their coloured appearance to the way they absorb light.

An important point to grasp is that the light that is reflected from a coloured surface is not changed in colour by the process of reflection. For example, when you hold a glossy photograph under a bright white light, the light that bounces off it will produce white reflections that interfere with the colour of the photograph itself. The dyes on a surface such as paper do not supply colour to light reflected off the surface, but to light that penetrates through them and gets reflected or scattered back from beneath it. During the light's journey through the particles of dye, ink or paint, the pigments absorb light at some frequencies. The light that emerges thus appears to be coloured. When paints are mixed or dyes are overlaid, the combination absorbs all the frequencies absorbed by the individual components. When, for example, we talk of 'cyan ink', we mean ink that, when it is applied to white paper and illuminated by white light will absorb the red component, allowing the green and blue, which combine to produce the cyan colour, to be reflected back (see Figure 6.7). If we apply a layer of such an ink to white paper, and then add a layer of magenta, the magenta ink will absorb incident green light, so the combination of the cyan and magenta inks produces a blue colour (the additive primary blue, that is), as shown in Figure 6.8. Similarly, combining cyan and yellow inks produces green, while magenta combined with yellow gives red. A combination of all three colours will absorb all incident light, producing black. Mixtures containing different proportions of cyan, magenta and yellow ink will absorb red, green and blue light in corresponding proportions, thus (in theory) producing the same range of colours as the addition of red, green and blue

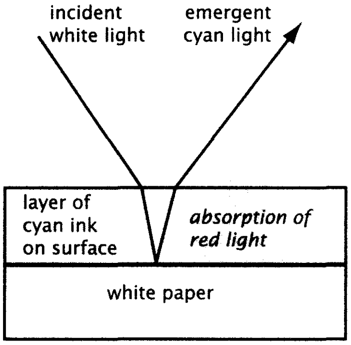


Figure 6.7
Effect of coloured ink

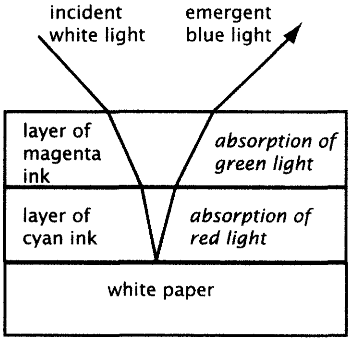


Figure 6.8
Mixing coloured inks

primary lights. Cyan, magenta and yellow are called the *subtractive primaries* for this reason. These subtractive primaries are the primary colours which the artist working in conventional media must use.

In practice, it is not possible to manufacture inks which absorb only light of precisely the complementary colour. Inevitably, some unwanted colours are absorbed at the same time. As a result, the gamut of colours that can be printed using cyan, magenta and yellow is not the same as the RGB gamut. Furthermore, combining actual inks of all three colours does not produce a very good black. On top of this, applying three different inks is not very good for your paper and leads to longer drying times. For these reasons, in magazine and book printing, the three subtractive primaries are augmented with black. The four colours cyan, magenta, yellow and black, when used in printing, are called *process colours*, and identified by their initials CMYK. Figure 6.9 shows the CMYK colour gamut and compares it with the RGB gamut displayed earlier. You will see that the CMYK gamut is smaller than RGB, but is not a strict subset — there are CMYK colours that fall outside the RGB gamut, and vice versa. You will also see that, because of the different colour mixing method and the presence of the black, the gamut is not triangular.

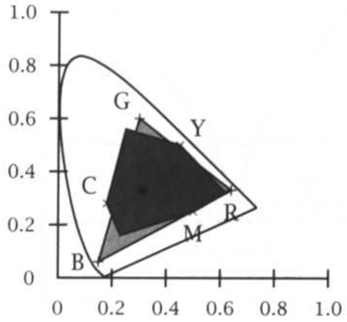


Figure 6.9
The CMYK colour gamut

For printing, an understanding of the subtractive CMYK model of colour, and also of the properties of inks and papers is vital to high quality colour reproduction. For multimedia, CMYK is of less direct relevance. If it should happen that you wish users of your multimedia production to be able to print copies of the images it contains, in colour, then you should ensure that they only contain colours within the CMYK gamut, otherwise there will be colour shifts on the printed version. If you are incorporating images that have previously been used in some print medium, you may find they are stored in a format in which colours are specified as *C*, *M*, *Y* and *K* values, instead of *R*, *G* and *B*. When the images are converted to a suitable file format for multimedia, it may be necessary also to convert the CMYK colour values into RGB. (Programs such as Photoshop can do this.) Having done so, you may be able to apply some colour transformations to improve the appearance by taking advantage of the larger RGB gamut.

Finally, if you come from a printing or painting background, or any pre-digital art and design discipline except photography, film or video, you may find it more natural to specify colours using

CMYK, since the subtractive mixing of CMYK colours is more like the mixing of inks and paints — the primaries are the same, for example. You should also realize that painting programs which simulate natural media will simulate the subtractive colour mixing that comes from overlaying different coloured paint. Plate 17 shows the effect of painting with thin layers of yellow, magenta and cyan using the simulated paints of Painter. The red, green, and blue, and the other colours are all formed by overlaying thin layers of these primaries with a simulated airbrush; the mixing is subtractive.

⇨ Many people's thinking about primary colours is confused by the habit school teachers have of telling children that the primary colours are red, yellow and blue, referring to the cheerful poster-paint primaries familiar from the classroom, not the artists' primary pigments, which are given different names by different manufacturers, but whose appearance corresponds well to the subtractive primaries magenta, yellow and cyan. The confusion stems from the breadth with which we apply the names of different hues: mixing any yellow with any blue will produce something we would call green, for example. Of the colours available in the sort of paints commonly found in junior schools, red, yellow and blue provide the closest approximations to the primaries. They can be mixed to give a range of colours, but not the full range obtainable when true primary pigments are available. (Purples are especially disappointing.)

HSV

Breaking a colour down into its component primaries makes sense from a theoretical point of view, and reflects the way monitors, scanners and colour printers work, but as a way of describing colour, it does not correspond to the way in which we experience colours in the world. Looking at a pale (cyan) blue, you probably don't relate it to a mixture of green and blue-violet light, but rather to other blues, according to how nearly pure blue it is, or how much of a green-ish or purple-ish blue it appears, and to how light or dark it is.

In more formal terms, we can consider a colour's *hue*, its *saturation*, and its *brightness*. In physicists's terms, hue is the particular wavelength at which most of the energy of the light is concentrated (the *dominant wavelength*); if you like, hue is the pure colour of light. In less scientific terms, we usually identify hues by names. Famously, Isaac Newton identified seven hues in the spectrum, the familiar red, orange, green, blue, indigo and violet of the rainbow. Newton had his own mystical reasons for wanting seven colours in the rainbow and it is more normal to distinguish just four: red, yellow, green and blue, and define hue informally as the extent to which a colour resembles one, or a mixture of two, of these.

A pure hue can be more or less 'diluted' by mixing it with white: the dominant hue (wavelength) remains the same, but the presence of other hues makes the colour paler. A colour's saturation is a measure of its purity. Saturated colours are pure hues; as white is mixed in, the saturation decreases. A colour's appearance will be modified by the intensity of the light: less light makes it appear darker. The brightness of a colour is a measure of how light or dark it is.

⇨ The equation of hue with the measurable quantity dominant wavelength is appealing, but not quite as simple as one would hope. Some hues cannot be identified with a single wavelength. Most people would consider purple to be a hue, but the SPD for purple has peaks at both short and long wavelengths (red and blue). We know, though, from the discussion of subtractive mixing, that we could consider purple to be made by subtracting light at the dominant wavelength of a shade of green from white light. In other words, the hue purple is associated with a dominant wavelength, but the dominance takes the form of a negative peak. This notion is, perhaps, no more incongruous than the idea that a hole is where an electron isn't.

In terms of paint, a hue is a pure colour. Adding white decreases its saturation, producing a tint. Adding black decreases its brightness, producing a tone.

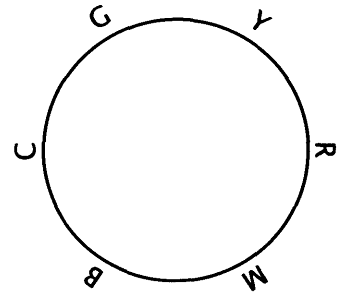


Figure 6.10
A colour wheel

Since the early nineteenth century, painters and other visual artists who like to systematize their thinking about colours have organized them into a 'colour wheel': the different hues are organized in a circle, with the subtractive primaries equally spaced around the perimeter, and the additive primaries in between them so that each primary is opposite its complement (see Figure 6.10). In art terms, it is the subtractive primaries which are usually referred to as primary colours. Their complements are called secondary colours, and the colours produced by mixing primaries and secondaries are tertiary colours. (The tertiary colours are sometimes added to the colour wheel in between the primaries and secondaries.)

This colour wheel can be extended to produce an alternative colour model. First, the distinct primary, secondary and tertiary hues can be augmented to produce a continuum extending all the way round a circle. Any hue can be specified as an angle around the circle, usually measured counter-clockwise, relative to red at 0°. Saturation can be added to the model by putting white at the centre of the circle, and then showing a gradation of tints from the saturated hues on the perimeter to the pure white at the centre. To add brightness to this model, we need a third dimension. Imagine the colour disk just described as being on top of a cylinder, made up of similar slices whose brightness decreases as we move downwards, so that, instead of white at the centre we have increasingly dark greys, until at the bottom, there is black. On each slice, the colours will be correspondingly toned down. (See plate 18, which shows three such slices.) A particular colour can be identified by its hue *H* (how far round the circle from red it is), its saturation *S* (how near the central axis) and its brightness value *V* (how far up the cylinder it is), hence this form of colour description is called the HSV model.

⇨ As we move towards the ends of the HSV cylinder, the variability in saturation will be smaller. At the bottom, there is less light, so less scope for adding white, while at the top, there is more light, and less scope for adding black. Accordingly, a better model is actually provided by a double cone, rather than a cylinder, which narrows towards a single black point at the bottom, and a single white point at the top. The colour model based on this geometry is called *HLS*, which stands for hue, saturation, and lightness.

You may sometimes see references to the *HSB* model. This is identical to the HSV model just described, using the B to stand for brightness instead of the less obvious V for value.

Industrial designers use a different approach to colour specification, based on swatches of standard colours that can be reproduced accurately. The Pantone system is widely used for this purpose, and has been adapted for incorporation into computer graphics applications.

Although the HSV model superficially resembles one of the ways in which painters think about colour, it should be remembered that the arrangement of hues around the circle is really no more than a visual mnemonic for colour harmony — it is otherwise arbitrary and has no physical basis.

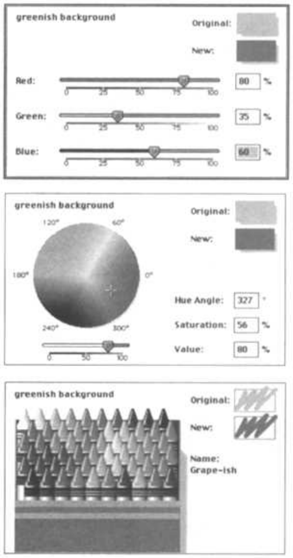
All colour models provide a way of specifying a colour as a set of numbers, and of visualizing their relationships in a three-dimensional space. This way of describing colour is independent of the many subjective factors which normally play a part in our perception of colour. For computer programs, a numerical, objective, colour specification is necessary. For people, it is less so, and most people find the numerical specification of colours non-intuitive. Within graphics programs, though, it is normal to choose colours interactively using a *colour picker* dialogue. These may be based on any of the colour models we have discussed, and allow you to choose colours by manipulating controls that affect the values of the components, while seeing a sample of the resulting colour displayed. Figure 6.11 shows the RGB and HSV colour pickers provided by the Macintosh system software. In the HSV colour picker, a slider is used to control the *V* value, while the corresponding slices through the cylinder are displayed alongside, and a colour is selected by clicking on a point in the disk. The RGB picker simply uses sliders to control the three components.

Figure 6.11 also shows the witty 'crayon picker', which can be used to choose a colour from the system palette. More conventional palette pickers just present swatches of the available colours.

Colour Spaces Based on Colour Differences

For some purposes it is useful to separate the brightness information of an image from its colour. Historically, an important motivation for doing this was to produce a means of transmitting colour television signals that would be compatible with older black and white receivers. By separating brightness and colour, it is possible to transmit a picture in such a way that the colour information is undetected by a black and white receiver, which can treat the brightness as a monochrome signal. Once a means of separating colour and brightness had been devised, it became possible to process the two signals separately in other ways. In particular, it is common practice in analogue broadcasting to use less bandwidth to transmit the colour than the brightness, since the eye is less sensitive to colour than to brightness. There is evidence to suggest that the human visual apparatus processes brightness and colour separately, too, so this form of representation also holds some physiological interest.

The basis of the RGB representation of colour is that light can always be broken down into three components. What we consider as brightness — the extent to which a pixel appears to be lighter or darker — is clearly related to the values of the red, green and blue components of the colour, since these tell us how much light of each primary is being emitted. It would be tempting to suppose that the brightness could be modelled simply as $(R + G + B)/3$, but this formula is not quite adequate, because it takes no account of the eye's differing sensitivity to the different primaries. Green contributes far more to the perceived brightness of a colour than red, and blue contributes hardly at all. Hence to produce a measure of brightness from an RGB value, we must weight the three components separately. There are several different formulae in use for performing this computation; that recommended for modern cathode ray tubes, as used in most computer monitors and TV sets is:



Red: 80 %
Green: 35 %
Blue: 60 %

Hue Angle: 327 °
Saturation: 56 %
Value: 80 %

Name: Grape-ish

Figure 6.11
Colour pickers

$$Y = 0.2125R + 0.7154G + 0.0721B$$

The quantity Y defined here is called *luminance*.

You should be able to see that red, green and blue values can be reconstructed from luminance and any two of the primaries. It is normal, though, for technical reasons, to use some variant of a representation consisting of Y together with two *colour difference* values, usually $B - Y$ and $R - Y$. We say 'some variant' because different applications require that the three components be manipulated in different ways. For analogue television, a non-linearly scaled version of Y, properly denoted Y', is used together with a pair of weighted colour difference values, denoted U and V. Out of carelessness, the term *YUV colour* is often used to refer to any colour space consisting of a brightness component together with two colour difference components. Digital television uses a variant called $Y'C_BC_R$, which is like Y'UV but uses different weights for the colour difference computation.

Device-independent Colour Spaces

RGB and CMYK are the most commonly used colour models, but they are both *device-dependent*: different monitors provide different red, green and blue primaries; different sorts of ink and paper produce different cyan, magenta, yellow and even black. Given an RGB value such as $(255, 127, 255)$, you don't know exactly what colour is going to be produced on any particular monitor, and you have no guarantee that it will be the same on any two monitors, or even on the same monitor at different times.

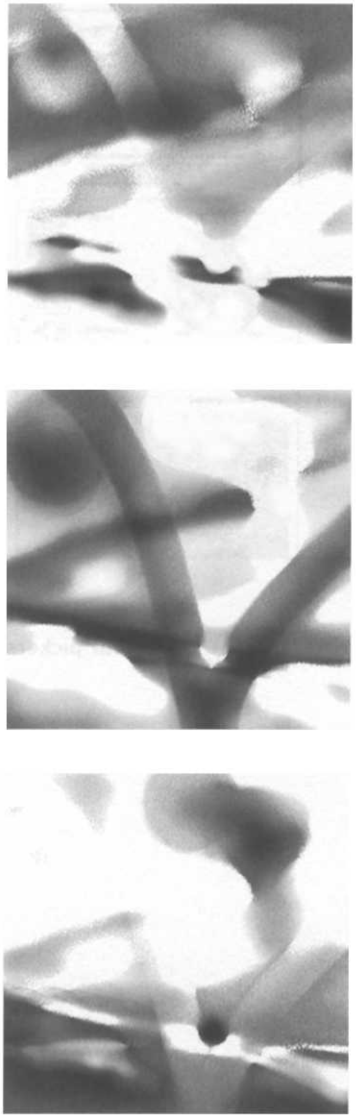


Figure 6.12
Red, green and blue channels of Plate 17

As we mentioned in the technical note on page 157, the *Commission Internationale de l'Éclairage (CIE)* has carried out work to produce an objective, *device-independent* definition of colours. Its basic model, the *CIE XYZ colour space*, uses three components X, Y and Z to approximate the three stimuli to which the colour-sensitive parts of our eyes respond. This model is device-independent, but as well as being awkward to work with and impossible to realize with physical light sources, it suffers (as do the RGB, CMYK, HSV and most other colour models) from not being *perceptually uniform*. A perceptually uniform model would be one in which the same change in one of the values produced the same change in appearance, no matter what the original value was. If, for example, RGB was a perceptually uniform model then changing the R component from 1 to 11 would produce the same increase in perceived redness as changing it from 101 to 111. However, it doesn't. This means that computations of the distance between two colours based on their RGB values do not reflect what we consider the degree of difference between them, which in turn makes finding a closest colour match when reducing an image to a restricted palette difficult.

⇨ You will be familiar with perceptual non-uniformity if you have ever owned a cheap audio amplifier: very small movements of the volume control at low levels make the sound much louder or quieter, while much larger movements at the top end make little discernible difference to how loud the sound seems.

For many years, the CIE tried to produce a colour model that was perceptually uniform, but never quite succeeded. Two models, refined from the original XYZ model to be much more nearly perceptually uniform were produced in 1976. These are the L*a*b* and L*u*v* models (often written simply Lab and Luv, or CIELAB and CIELUV).

In both models, the L* component is a uniform luminance and the other two components are colour differences. In L*a*b*, they specify colour in a way that combines subtractively, as in CMYK. 'Lab colour' is widely used in the pre-press industry as a standard specification of colour difference values for printing. L*u*v* is a similar model, but the colour difference values work additively, as in RGB, so L*u*v* provides a device-independent model of colour suitable for monitors and scanners.

These models do not correspond to any easily understandable way of thinking about colour, so you are unlikely to use them directly.

However, they do provide a basis for the device-independent treatment of colour, and are used as a reference point for systems that try to guarantee that the colours we specify in our images will look as nearly as possible the same on whatever output device is being used.

Channels and Colour Correction

Earlier, we implied that in a 24-bit RGB colour image, a single 24-bit value is stored for each pixel. While this is the way colours are stored in some file formats, there is another possibility: for each pixel, three separate 8-bit values, one for each of red, green and blue, are stored. That is, in terms of data structures, instead of storing an image in a single array of 24-bit values, we store it in three arrays, each consisting of single bytes. This representation has some computational advantages, since 3-byte quantities are awkward to manipulate on most machines. It also provides a useful way of organizing colour images conceptually, irrespective of whether the physical storage layout is actually of this form.

Each of the three arrays can itself be considered as an image. When we consider each colour separately, it only makes sense to consider these as greyscale images. Each of the three greyscale images making up the colour image is called a *channel*, so we speak of the red channel, or the green or blue channel. Figure 6.12 shows the three channels of the image in Plate 17.

Each channel can be manipulated separately: since it is an image, the manipulations can be done using the usual image editing tools provided by Photoshop and similar programs. In particular, levels and curves can be used to alter the brightness and contrast of each channel independently. When the altered channels are recombined, the colour balance of the composite image will have changed. Colour correction operations performed in this way are frequently required to compensate for the deficiencies of scanners and other input devices. Plate 19 shows an example: the scanned photograph on the left has a marked colour cast; in the middle image, this has been corrected; the image on the right shows how colour adjustment can be taken further to produce an artifically coloured image.

The previous paragraph may make adjusting the levels of each channel sound like a simple operation. In practice, although some colour adjustments are straightforward — maximizing the contrast in all three channels, for example — producing a desired result can be very time-consuming, calling for considerable experience and fine judgement. Some image manipulation programs come equipped with 'wizards' or 'assistants', software modules with some degree of artificial intelligence, to help in some common situations calling for colour correction.

Sometimes, though, a simpler approach is adequate. Just as the brightness and contrast controls encapsulate a class of tonal adjustments, so the *colour balance* and *hue and saturation* adjustments provide a less refined interface to the adjustment of levels in the three colour channels. Colour balance provides three sliders, one for each of the pairs of complements cyan and red, magenta and green, and yellow and blue, which allow you to adjust the relative proportions of each pair. The adjustment can be applied to shadows, midtones, or highlights separately. The hue and saturation adjustment performs a similar operation, by allowing you to vary each of the H, S, and V components of colours within specified ranges: reds, greens, blues, and so on. This provides a more intuitive way of adjusting the colours if you are used to working with HSV specifications.

A different way of making colour changes which, like the two sets of controls just described, works across all the channels at once, is to replace a specified colour wherever it appears with a different one. This is like doing a search and replace operation on a text file using an editor, except that the target and replacement are colours not text strings. Photoshop's version of this operation is somewhat more sophisticated: you can set a tolerance value, which causes pixels within a certain percentage of the target value to be altered, and use H, S, and V sliders to specify a change to those components of pixels that match the target, instead of just specifying an exact replacement colour. In effect, this command creates a mask, in a similar manner to the magic wand described in Chapter 5 and then makes colour adjustments to selected areas of the image.

Although we are only discussing RGB colour, images stored in any colour space can be separated into channels. In particular, the channels of a CMYK image correspond to the colour separations needed by commercial printing processes, while the channels of a $Y'C_BC_R$ image correspond to the components of a video signal.

Furthermore, the idea that a greyscale image can be considered to be one channel of a composite image is extended in the concept of alpha channel, which we introduced in Chapter 5. Although alpha channels behave completely differently from the colour channels that make up the displayed image, their common representation has some useful consequences.

One of the most common applications of this duality is in inserting people into a separately photographed scene, often one where it is impossible for them to be in reality. This can be done by photographing the person against a blue background (ensuring that none of their clothing is blue). All of the image data for the person will be in the red and green channels; the blue channel will be an image of the area not occupied by the person. An alpha channel constructed as a copy of the blue channel (an operation built in to any software that manipulates image channels) will therefore mask out the background, isolating the figure. It is then a simple matter to copy the figure, using the mask to define it as the selection, and paste it into, for example, an underwater scene. This *blue screen* technique is commonly used in video and film-making, as well as in constructing fake images.

Other sorts of colour image processing can be implemented by applying some greyscale processing to each of the channels separately. An important example is JPEG compression, which, although it is most commmonly applied to colour images, is defined in the standard to operate on 8-bit quantities, so that each channel is compressed individually. One advantage of this is that the compression algorithm is unaffected by the colour space used for the image, it just takes whatever channels are there and compresses them. This offers the freedom to choose a colour space, and apply pre-processing to the channels. It is common for JPEG compressors to transform an image into $Y'C_BC_R$, and then downsample the colour difference components, since the eye is less sensitive to colour variation than to brightness variations. This same 'chroma downsampling' step is applied to video signals, both for compression and transmission, as we will see in Chapter 10.

Consistent Colour

Colour adjustment is messy, and getting it wrong can cause irreparable damage to an image. It would be much better to get things right first time, but the varying colour characteristics of different monitors and scanners make this difficult. Some recent developments in software are aimed at compensating for these differences. They are based on the use of *profiles*, describing how devices detect and reproduce colour.

We don't need much information to give a reasonable description of the colour properties of any particular monitor. We need to know exactly which colours the red, green and blue phosphors emit (the R, G and B *chromaticities*). These can be measured using a suitable scientific instrument, and then expressed in terms of one of the CIE device-independent colour spaces. We also need to know the maximum saturation each component is capable of, i.e. we need to know what happens when each electron beam is full on. We can deduce this if we can characterize the make-up and intensity of white, since this tells us what the (24-bit) RGB value (255, 255, 255) corresponds to. Again, the value of white — the monitor's *white point* — can be specified in a device-independent colour space.

> ⇨ You will sometimes see white point specified as a *colour temperature*, in degrees absolute. This form of specification is based on the observation that the spectral make-up of light emitted by a perfect radiator (a so-called 'black body') depends only on its temperature, so a black body temperature provides a concise description of an SPD. Most colour monitors for computers use a white point designed to correspond to a colour temperature of 9300°K. This is far higher than daylight (around 7500°K), or a conventional television monitor (around 6500°K), in order to generate the high light intensity required for a device that will normally be viewed under office lighting conditions. (Televisions are designed on the assumption that they will be watched in dimly lit rooms.) The 'white' light emitted by a monitor when all its three colours are at full intensity is actually quite blue.
>
> Computer monitors are not actually black bodies, and so their SPDs deviate from the shape of the black body radiation, which means that colour temperature is only an approximation of the characteristic of the white point, which is better specified using CIE colour values.

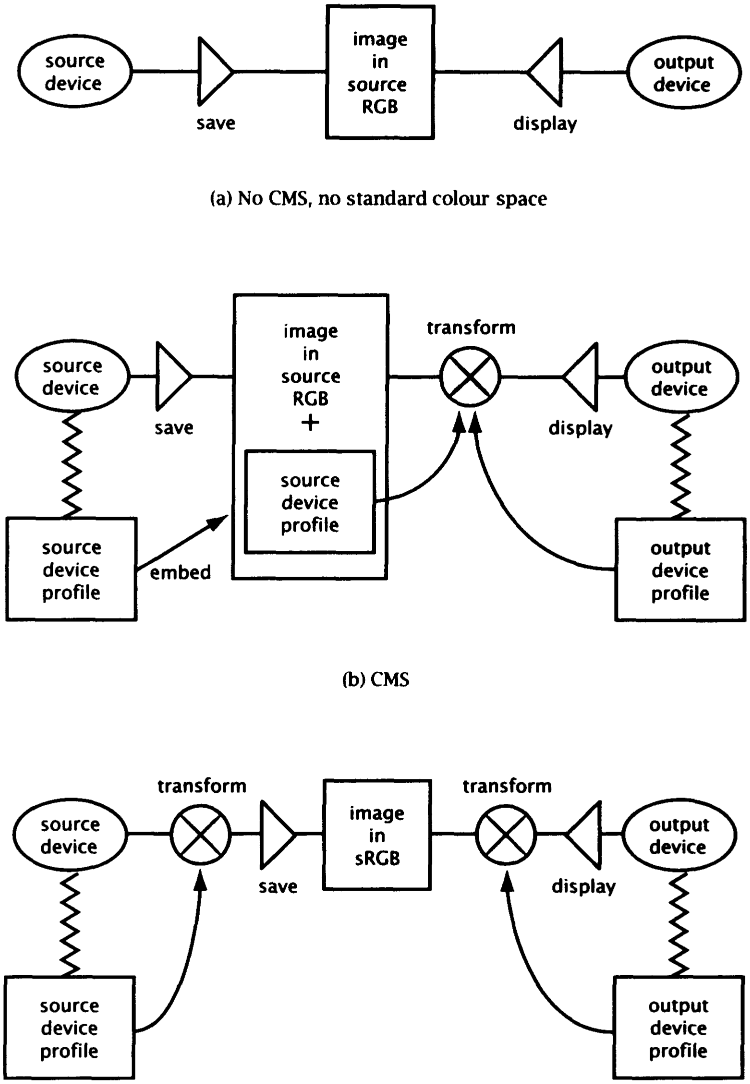
Finally, the most complex element in the monitor's behaviour is the relationship between the RGB values presented to it by the graphics controller, and the intensity of the light emitted in response. This relationship is not a simple linear one: the intensity of light emitted in response to an input of 100 is not ten times that produced by an input of 10, which in turn is not ten times that produced by an input of 1. Physical devices do not work as conveniently as that. However, the response can be captured to some extent using a single number, referred to as the display's γ (often written out as *gamma*).

Why γ? Because the *transfer characteristic* of a display — the relationship between the light intensity emitted by the screen (I) and the voltage applied to the electron gun (V) is often modelled by the transfer function $I = V^{\gamma}$, where γ is a constant. Unfortunately, this model is not entirely accurate, and one of the sources of variability between monitors lies in the use of incorrect values for γ which attempt to compensate for errors in the formula. Another is the fact that some display controllers attempt to compensate for the non-linearity by adjusting values according to an inverse transfer function before they are applied to the electron guns, while others do not.

Armed with device-independent values for red, green and blue, the white point and γ, it is possible to translate any RGB colour value into an absolute, device-independent, colour value in a CIE colour space that exactly describes the colour produced by the monitor in response to that RGB value. This is the principle behind the practice of *colour management.*

Figure 6.13(a) shows a typical situation calling for colour management: an image is prepared using some input device, which might be a monitor used as the display by a graphics editor, or a scanner. In either case, the image will be stored in a file, using RGB values which reflect the way the input device maps colours to colour values -- its *colour profile.* We say that the image is stored using the input device's colour space. Later, the same image may be displayed on a different monitor. Now the RGB values stored in the image file are mapped by the output device, which probably has a different profile from the input device. The colours, stored in the colour space of the input device are interpreted as if they were in the colour space of the output device. In other words, they come out wrong.

One way of correcting this is shown in Figure 6.13(b). Information from the input device's profile is embedded in the image file. EPS, TIFF, JFIF and PNG files are able to accommodate such information, in varying degrees of detail. At the least, the R, G and B chromaticities, white point and γ can be included in the file. Software on the machine being used to display the image can, in principle, use this information to map the RGB values it finds in the image file to colour values in a device-independent colour space. Then, using the device profile of the output monitor, map those



(a) No CMS, no standard colour space

(b) CMS

(c) Using the sRGB colour space

Figure 6.13
Colour management

values to the colour space of the output device, so that the colours are displayed exactly as they were on the input device. In practice, it is more likely that the two profiles would be combined and used to map from the input device colour space to the output device colour space directly. It is, of course, possible, that some of the colours in the input device colour space are not available in the output device colour space, so colour management can only actually guarantee that the colours will be displayed exactly as they were intended within the capabilities of the output device. If software uses colour management consistently, the approximations made to accommodate a restricted colour gamut will be the same, and the output's colour will be predictable. If the display software has not been written to take advantage of device profiles, the display will be no better than it would be without colour management.

An alternative way of using colour management software is to modify the colours displayed on a monitor using the profile of a chosen output device. In this way, colour can be accurately previewed. This mode of working is especially useful in pre-press work, where actually producing printed output may be expensive or time-consuming.

To obtain really accurate colour reproduction across a range of devices, device profiles need to provide more information than simply the RGB chromaticities, white point and a single figure for γ. In practice, for example, the γs for the three different colours are not necessarily the same. As already stated, the actual transfer characteristics are not really correctly represented by γ; a more accurate representation is needed. If, as well as display on a monitor, we also wished to be able to manage colour reproduction on printers, where it is necessary to take account of a host of issues, including the C, M, Y and K chromaticities of the inks, spreading characteristics of the ink, and absorbency and reflectiveness of the paper, even more information would be required — and different information still for printing to film, or video.

Since the original impetus for colour management software came from the pre-press and printing industries, colour management has already been developed to accommodate these requirements. The *International Colour Consortium (ICC)* has defined a standard device profile, which supports extremely elaborate descriptions of the colour characteristics of a wide range of devices. ICC device profiles are used by colour management software such as Apple's ColorSync, and the Kodak Precision Color Management System, to provide colour management services. ColorSync provides colour management at a system software level, making it easy for individual applications to incorporate extremely sophisticated colour management services. Manufacturers of scanners, monitors and printers routinely produce ICC profiles of their devices. TIFF and EPS files can accommodate complete ICC profiles.

⇨ Colour management is not much use unless accurate profiles are available. Unfortunately, no two devices are exactly identical. In the case of monitors, the situation is even worse, since tubes age and their colour characteristics change over time. Although a generic profile produced by the manufacturer for one line of monitors is helpful, to take full advantage of colour management it is necessary to calibrate individual devices, at relatively frequent intervals (once a month is often advised). Some high end monitors are able to calibrate themselves automatically. For others, it is necessary to use a special measuring device in conjunction with software that displays a sequence of colour values and, on the basis of the measured output of the screen, generates an accurate profile.

You may wonder why the profile data is embedded in the file. Why is it not used at the input end to map the colour values to a device-independent form, which can then be mapped to the output colour space when the image is displayed? The work is split between the two ends and no extra data has to be added to the file (see Figure 6.13(c)). The reason is that most existing software does not work with device-independent colour values, so it could not display the images at all. If it ignores a device profile, things are no worse than they would have been if it was not there.

Clearly, though, it would be desirable to use a device-independent colour space for stored colour values. The sRGB (standard RGB) colour model attempts to provide such a space. As you can guess, it is based on the RGB model, and specifies standard values for the R, G and B chromaticities, white point, and γ. The standard values have been chosen to be typical of the values found on most monitors.[8] As a result, if the display software is not aware of the sRGB colour space and simply displays the image without any colour transformation, it should still look reasonable. Use of sRGB colour is especially suitable for graphics on the World Wide Web, because there most images are only ever destined to be displayed on a terminal. A standard colour space based on RGB is of no help in maintaining consistency of colours for printing.

Having read this section, you may well ask 'Why bother?' It may not seem like the end of the world if the colours in your image are a bit off when it is displayed on somebody else's monitor. Consider, for example, the use of Web pages as online shopping catalogues. For many products, the colour is important. (To take an extreme example, think about buying paint over the Internet.) One of the factors driving the development of colour management

[8] Or so its proponents claim. Other experts believe the chosen values are not the best. Photoshop 5.0 used sRGB as its default colour space; in response to users' reactions, this default was almost immediately changed in the 5.01 update.

and its incorporation into Web browsers is the desire to facilitate online shopping. As well as the development of the sRGB colour space, this has led to the development of browser plug-ins providing full colour management facilities based on ColorSync. If online shopping seems crassly commercial, consider instead the use of the World Wide Web and CD-ROMs by art galleries to deliver catalogues. Here again the best possible colour reproduction is vital if the catalogues are to serve their function properly.

Further Information

[JMF94] surveys many aspects of colour; [Got98] considers colour in digital media from a designer's point of view. [Poy96] includes technical material on colour, especially in the context of video. [FvDFH96] also has a somewhat technical description of colour spaces and the physics of colour. [SACM] is the document that defines the sRGB colour space.

Exercises

1. Comment on the advisability of using translucent tangerine coloured plastics in the surround of a computer monitor.

2. If you consider the three components of an RGB colour to be cartesian coordinates in a 3-dimensional space, and normalize them to lie between 0 and 1, you can visualize *RGB colour space* as a unit cube, as shown in Figure 6.14. What colours correspond to the eight corners of this cube? What does the straight line running from the origin to $(1, 1, 1)$ shown in the figure represent? Comment on the usefulness of this representation as a means of visualizing colour.

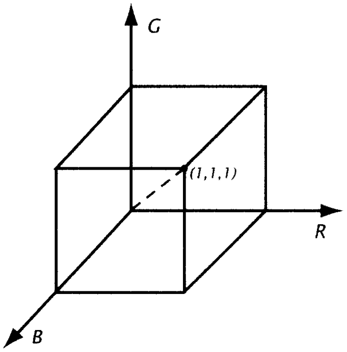


Figure 6.14
The RGB colour space

3. Why do RGB colour values (r, g, b), with $r = g = b$ represent shades of grey?

4. Why do you think we chose to show the CIE chromaticity diagram in black and white as Figure 6.2, instead of providing a colour plate showing the actual colours?

5. If a CLUT has 256 entries (so pixels only need a single byte), each 3 bytes wide, how large must an image be before using indexed colour provides a net saving of space?

6. For which of the following images would the use of indexed colour be satisfactory?

   (a) A reproduction of your national flag.

   (b) A photograph of the surface of Mars.

   (c) A photograph of yourself on the beach.

   (d) A still from a black and white Fred Astaire movie.

   In which cases is the system palette likely to be acceptable without dithering?

7. Give an example of an image whose most important colour is not one of its most commonly occurring colours. When might this cause a problem?

8. If you gave a child a paintbox containing red, green and blue paints only, will they be able to mix an adequate range of colours to paint the ceiling of the Sistine chapel?

9. Which colour space should be used for storing images and why?

10. Suppose you had an image with too much red in it, perhaps because of poor lighting. Explain what adjustments you would make to its red channel to compensate. What undesirable side-effect would your adjustment have, and what would you do about it?

11. Superman's uniform is blue, with a yellow S on the chest, and a red cape. How would you make an image of Superman flying, without recourse to his superpowers?

Characters and Fonts

7

Text has a dual nature: it is a visual representation of language, and a graphic element in its own right. Text in digital form must also be a representation of language; that is, we need to relate bit patterns stored in a computer's memory or transmitted over a network to the symbols of a written language. When we consider the display of stored text, its visual aspect becomes relevant. We then become concerned with such issues as the precise shape of characters, their spacing and the layout of lines, paragraphs and larger divisions of text on the screen or page. These issues of display are traditionally the concern of the art of *typography*. Much of the accumulated typographical practice of the last several centuries can be adapted to the display of the textual elements of multimedia.

In this chapter, we consider how the fundamental units of written languages — characters — can be represented in a digital form, and how the digital representation of characters can be turned into a visual representation for display. We will show how digital font technology makes it possible to approximate the typographical richness of printed text in the textual components of multimedia.

Character Sets

In keeping with text's dual nature, it is convenient to distinguish between the lexical *content* of a piece of text and its *appearance*. By content we mean the characters that make up the words and other units, such as punctuation or mathematical symbols. (At this stage we are not considering 'content' in the sense of the meaning or message contained in the text.) The appearance of the text comprises its visual attributes, such as the precise shape of the characters, their size, and the way the content is arranged on the page or screen. For example, the content of the following two sentences[1] is identical, their appearance is not:

```
The Right Hon was a tubby little chap who looked as if
he had been poured into his clothes and had forgotten
to say 'When!'
```

The Right Hon was a tubby little chap who looked as if he had been *poured* into his clothes and had forgotten to say 'When!'

We all readily understand that the first symbol in each version of this sentence is a capital T, even though one is several times as large as the other, has some additional strokes, and is darker. To express their fundamental identity, we distinguish between an *abstract character* and its graphic representations, of which there is a potentially infinite number. Here, we have two graphic representations of the same abstract character.

As a slight over-simplification, we could say that the content is the part of a text that carries its meaning or semantics, while the appearance is a surface attribute that may affect how easy the text is to read, or how pleasant it is to look at, but does not substantially alter its meaning. In the example just given, the fixed-width, typewriter-like font of the first version clearly differs from the more formal book font used for most of the second, but this and the gratuitous initial dropped cap and use of different fonts do not alter the joke. Note, however, that the italicization of the word 'poured' in the second version, although we would normally consider it an aspect of the appearance like the use of the sans serif font for 'Right Hon', implies an emphasis on the word that is missing in the plain version (and also in the original story). Hence, the distinction between appearance and content is not quite as clear-cut as one might think. Nevertheless, it is a useful distinction, because

[1] From the short story *Jeeves and the Impending Doom* by P.G. Wodehouse.

it permits a separation of concerns between these two qualities that text possesses.

Abstract characters are grouped into alphabets. Each particular alphabet forms the basis of the written form of a certain language or group of languages. We consider any set of distinct symbols to be an alphabet.[2] This includes the set of symbols used in an ideographic writing system, such as those used for Chinese and Japanese, where each character represents a whole word or concept, as well as the phonetic letters of Western-style alphabets, and the intermediate syllabic alphabets, such as Korean Hangul. In contrast to colloquial usage, we include punctuation marks, numerals, and mathematical symbols in an alphabet, and treat upper- and lower-case versions of the same letter as different symbols. Thus, for our purposes, the English alphabet includes the letters A, B, C,... Z, and a, b, c,... z, but also punctuation marks, such as comma and exclamation mark, the digits 0, 1,..., 9, and common symbols such as + and =.

To represent text digitally, it is necessary to define a mapping between (abstract) characters in some alphabet and values that can be stored in a computer system. The only values that we can store are bit patterns. As we explained in Chapter 2, these can be interpreted as integers to base 2, so the problem becomes one of mapping characters to integers. As an abstract problem, this is trivial: *any* mapping will do, provided it associates each character of interest with exactly one number. Such an association is called, with little respect for mathematical usage, a *character set*; its domain (the alphabet for which the mapping is defined) is called the *character repertoire.* For each character in the repertoire, the character set defines a *code value* in its range, which is sometimes called the set of *code points.* The character repertoire for a character set intended for written English text would include the twenty-six letters of the alphabet in both upper- and lower-case forms, as well as the ten digits and the usual collection of punctuation marks. The character repertoire for a character set intended for Russian would include the letters of the Cyrillic alphabet. Both of these character sets could use the same set of code points; provided it was not necessary to use both character sets simultaneously (for example, in a bilingual document) a character in the English alphabet could have the same code value as one in the Cyrillic alphabet. The character repertoire for a character set intended for the Japanese Kanji alphabet must contain at least the 1945 ideograms for common use and 166 for names sanctioned by the Japanese Ministry of

[2] We do not define 'symbol'. In the abstract, an alphabet can be any set at all, but in practical terms, the only symbols of interest will be those used for writing down some language.

Education, and could contain over 6000 characters; consequently, it requires far more distinct code points than an English or Cyrillic character set.

The mere existence of a character set is adequate to support operations such as editing and searching of text, since it allows us to store characters as their code values, and to compare two characters for equality by comparing the corresponding integers; it only requires some means of input and output. In simple terms, this means that it is necessary to arrange that when a key is pressed on a keyboard, or the equivalent operation is performed on some other input device, a command is transmitted to the computer, causing the bit pattern corresponding to the character for that key to be passed to the program currently receiving input. Conversely, when a value is transmitted to a monitor or other output device, a representation of the corresponding character should appear.

There are advantages to using a character set with some structure to it, instead of a completely arbitrary assignment of numbers to abstract characters. In particular, it is useful to use integers within a comparatively small range that can easily be manipulated by a computer. It can be helpful, too, if the code values for consecutive letters are consecutive numbers, since this simplifies some operations on text, such as sorting.

Standards

The most important consideration concerning character sets is standardization. Transferring text between different makes of computer, interfacing peripheral devices from different manufacturers, and communicating over networks are everyday activities, and continual translation between different manufacturers' character codes would not be acceptable, so a standard character code is required. The following description of character code standards is necessarily somewhat dry, but an understanding of them is necessary if you are to avoid the pitfalls of incompatibility and the resulting corruption of texts.

Unfortunately, standardization is never a straightforward business, and the situation with respect to character codes remains somewhat unsatisfactory.

ASCII (American Standard Code for Information Interchange) has been the dominant character set since the 1970s. It uses 7 bits to store each code value, so there is a total of 128 code points. The character repertoire of ASCII only comprises 95 characters, however. The values 0 to 31 and 127 are assigned to *control characters*, such as form-feed, carriage return and delete, which have traditionally been used to control the operation of output devices. The control characters are a legacy from ASCII's origins in early teletype character sets. Many of them no longer have any useful meaning, and are often appropriated by application programs for their own purposes. Table 7.1 shows the ASCII character set. (The character with code value 32 is a space.)

Table 7.1
The printable ASCII characters

32		33	!
34	"	35	#
36	$	37	%
38	&	39	'
40	(	41	)
42	*	43	+
44	,	45	-
46	.	47	/
48	0	49	1
50	2	51	3
52	4	53	5
54	6	55	7
56	8	57	9
58	:	59	;
60	<	61	=
62	>	63	?
64	@	65	A
66	B	67	C
68	D	69	E
70	F	71	G
72	H	73	I
74	J	75	K
76	L	77	M
78	N	79	O
80	P	81	Q
82	R	83	S
84	T	85	U
86	V	87	W
88	X	89	Y
90	Z	91	[
92	\	93	]
94	^	95	_
96	`	97	a
98	b	99	c
100	d	101	e
102	f	103	g
104	h	105	i
106	j	107	k
108	l	109	m
110	n	111	o
112	p	113	q
114	r	115	s
116	t	117	u
118	v	119	w
120	x	121	y
122	z	123	{
124	|	125	}
126	~		

American English is one of the few languages in the world for which ASCII provides an adequate character repertoire. Attempts by the standardization bodies to provide better support for a wider range of languages began when ASCII was adopted as an ISO standard (ISO 646) in 1972. ISO 646 incorporates several national variants on the version of ASCII used in the United States, to accommodate, for example, some accented letters and national currency symbols.

A standard with variants is no real solution to the problem of accommodating different languages. If a file prepared in one country is sent to another and read on a computer set up to use a different national variant of ISO 646, some of the characters will be displayed incorrectly. For example, a hash character (#) typed in the United States would be displayed as a pounds sign (£) in the UK (and vice versa) if the British user's computer used the UK variant of ISO 646. (More likely, the hash would display correctly, but the Briton would be unable to type a pound sign, because it is more convenient to use US ASCII (ISO 646-US) anyway, to prevent such problems.)

➲ The problem of national variants of ISO 646 is particularly acute for programmers. Many programming languages, particularly those whose syntax has been influenced by that of C, make use of some of the characters in the ASCII repertoire which are rarely needed for text as delimiters and operators. For example, the curly brackets { and } are often used to delimit compound statements, function bodies, and other syntactical units; the vertical bar | is used as a component of several operator symbols; the backslash \ is widely used as an escape character within strings. Unfortunately, these otherwise rarely-used symbols are precisely those whose code values correspond to different characters in variants of ISO 646 used outside the US; usually they are used for letters not found in English. Bjarne Stroustrup gives an example of the grief this can cause. The following program:

```
int main(int argc, char* argv[])
{
  if (argc < 1 || *argv[1] == '\0') return 0;
  printf("Hello, %s\n", argv[1]);
}
```

would appear on a terminal using the Danish variant of ISO 646 as[3]

```
int main(int argc, char* argvÆ Å)
æ
  if (argc < 1 øø *argvÆ1Å == 'Ø0') return 0;
  printf("Hello, %sØn", argvÆ1Å);
å
```

A better solution than national variants of the 7-bit ISO 646 character set lies in the provision of a character set with more code points, such that the ASCII character repertoire is mapped to the values 0–127, thus assuring compatibility, and additional symbols required outside the US or for specialized purposes are mapped to other values. Doubling the set of code points was easy: the seven bits of an ASCII character are invariably stored in an 8-bit byte. It was originally envisaged that the remaining bit would be used as a parity bit for error detection. As data transmission became more reliable, and superior error checking was built in to higher level protocols, this parity bit fell into disuse, effectively becoming available as the high order bit of an 8-bit character.

As is customary, the different manufacturers each developed their own incompatible 8-bit extensions to ASCII. These all shared some general features: the lower half (code points 0–127) was identical to ASCII; the upper half (code points 128–255) held accented letters and extra punctuation and mathematical symbols. Since a set of 256 values is insufficient to accommodate all the characters required for every alphabet in use, each 8-bit character code had different variants; for example, one for Western European languages, another for languages written using the Cyrillic script, and so on. [4]

Despite these commonalities, the character repertoires and the code values assigned by the different manufacturers' character sets are different. For example, the character é (e with an acute accent) has code value 142 in the Macintosh Standard Roman character set, whereas it has the code value 233 in the corresponding Windows character set, in which 142 is not assigned as the value for any character; 233 in Macintosh Standard Roman, on the other hand, is È. Because the repertoires of the character sets are different, it is

[3] See B. Stroustrup, *The Design and Evolution of C++*, Addison-Wesley, 1994, page 159.

[4] Under MS-DOS and Windows, these variants are called *code pages*.

not even always possible to perform a translation between them, so transfer of text between platforms is problematical.

Clearly, standardization of 8-bit character sets was required. During the 1980s the multi-part standard ISO 8859 was produced. This defines a collection of 8-bit character sets, each designed to accommodate the needs of a group of languages (usually geographically related). The first part of the standard, ISO 8859-1 is usually referred to as *ISO Latin1*, and covers most Western European languages. Like all the ISO 8859 character sets, the lower half of ISO Latin1 is identical to ASCII (i.e. ISO 646-US); the code points 128–159 are mostly unused, although a few are used for various diacritical marks; Table 7.2 shows the 96 additional code values provided for accented letters and symbols. (The character with code value 160 is a 'non-breaking' space.)

⇨ The Windows Roman character set is sometimes claimed to be the same as ISO Latin1, but it uses some of the code points between 128 and 159 for characters which are not present in ISO 8859-1's repertoire.

Other parts of ISO 8859 are designed for use with Eastern European languages, including Czech, Slovak and Croatian (ISO 8859-2 or Latin2), for languages that use the Cyrillic alphabet (ISO 8859-5), for modern Greek (ISO 8859-7), Hebrew (ISO 8859-8), and others — there is a total of ten parts to ISO 8859, with more projected, notably an ISO Latin0, which includes the Euro currency symbol.

ISO 8859 has several shortcomings. In practical terms, the worst of these is the continued use of manufacturers' proprietary non-standard character sets. These are firmly entrenched in use and entangled with their corresponding operating systems and other software. Experience shows that, in time, standards are adopted if they can be seen to provide benefits, but there are fundamental problems with 8-bit character sets, which make it seem likely that ISO 8859 will not achieve universal adoption, but that newer standards will render it obsolete. The main problem is simply that 256 is not enough code points — not enough to represent ideographically based alphabets, and not enough to enable us to work with several languages at a time (unless they all happen to use the same variant of ISO 8859).

Table 7.2
The top part of the ISO Latin1 character set

160		161	¡
162	¢	163	£
164	¤	165	¥
166	¦	167	§
168	¨	169	©
170	ª	171	«
172	¬	173	-
174	®	175	¯
176	°	177	±
178	²	179	³
180	´	181	µ
182	¶	183	·
184	¸	185	¹
186	º	187	»
188	¼	189	½
190	¾	191	¿
192	À	193	Á
194	Â	195	Ã
196	Ä	197	Å
198	Æ	199	Ç
200	È	201	É
202	Ê	203	Ë
204	Ì	205	Í
206	Î	207	Ï
208	Ð	209	Ñ
210	Ò	211	Ó
212	Ô	213	Õ
214	Ö	215	×
216	Ø	217	Ù
218	Ú	219	Û
220	Ü	221	Ý
222	Þ	223	ß
224	à	225	á
226	â	227	ã
228	ä	229	å
230	æ	231	ç
232	è	233	é
234	ê	235	ë
236	ì	237	í
238	î	239	ï
240	ð	241	ñ
242	ò	243	ó
244	ô	245	õ
246	ö	247	÷
248	ø	249	ù
250	ú	251	û
252	ü	253	ý
254	þ	255	ÿ

Unicode and ISO 10646

The only possible solution is to use more than one byte for each code value. A 16-bit character set has 65,536 code points; putting it another way, it can accommodate 256 variants of an 8-bit character set simultaneously. Similarly, a 24-bit character set can accommodate 256 16-bit character sets, and a 32-bit character set can accommodate 256 of those. The ISO (in conjunction with the IEC) set out to develop a 32-bit character set, designated ISO 10646, structured along the lines of the previous two sentences: a collection of 2^{32} characters can be arranged as a hypercube (a 4-dimensional cube) consisting of 256 groups, each of which consists of 256 planes of 256 rows, each comprising 256 characters (which might be the character repertoire of an 8-bit character set). The intention was to structure the immense character repertoire allowed by a 32-bit character set with alphabets distributed among the planes in a linguistically sensible way, so that the resulting character set would have a clear logical structure. Each character can be identified by specifying its group g, its plane p, and a row r and column c (see Figure 7.1). Each of g, p, r and c is an 8-bit quantity, which can fit in one byte; four bytes thus identify a unique character, so, inverting our viewpoint, the code value for any character is the 32-bit value which specifies its position within the hypercube.

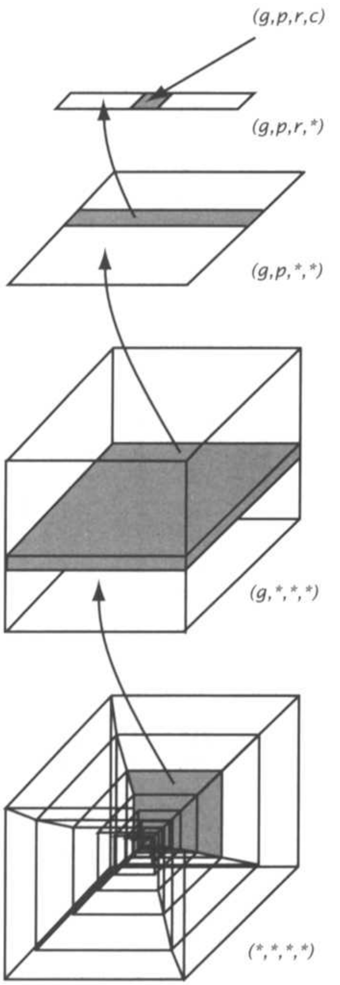


Figure 7.1
The four-dimensional structure of ISO 10646

To make the structure of the character set evident, we usually write code points as quadruples (g,p,r,c). By extension, such a quadruple also identifies a subset of the character set using a $*$ to denote all values in the range 0–255. Thus $(0,0,0,*)$ is the subset with all but the lowest-order byte zero. In ISO 10646 this subset is identical to ISO Latin1.

At the same time as the ISO was developing this elegant framework for its character set, an industry consortium was working on a 16-bit character set, known as *Unicode*. As we noted above, a 16-bit character set has 65,536 code points. This is not sufficient to accommodate all the characters required for Chinese, Japanese and Korean scripts in discrete positions. These three languages and their writing systems share a common ancestry, so there are thousands of identical ideographs in their scripts. The Unicode committee adopted a process they called *CJK consolidation*,[5] whereby characters used in writing Chinese, Japanese and Korean are given the same code value if they look the same, irrespective of

[5] Some documents use the name *Han unification* instead.

which language they belong to, and whether or not they mean the same thing in the different languages. There is clearly a cultural bias involved here, since the same process is not applied to, for example, upper-case A and the Greek capital alpha, which are identical in appearance but have separate Unicode code values. The pragmatic justification is that, with Chinese, Japanese and Korean, thousands of characters are involved, whereas with the European and Cyrillic languages, there are relatively few. Furthermore, consolidation of those languages would interfere with compatility with existing standards.

Unicode provides code values for all the characters used to write contemporary 'major' languages, as well as the classical forms of some languages. The alphabets available include Latin, Greek, Cyrillic, Armenian, Hebrew, Arabic, Devanagari, Bengali, Gurmukhi, Gujarati, Oriya, Tamil, Telugu, Kannada, Malayalam, Thai, Lao, Georgian, and Tibetan, as well as the Chinese, Japanese and Korean ideograms and the Japanese and Korean phonetic and syllabic scripts. It also includes punctuation marks, technical and mathematical symbols, arrows, and the miscellaneous symbols usually referred to as dingbats (pointing hands, stars, and so on). In addition to the accented letters included in many of the alphabets, separate diacritical marks, such as accents and tildes, are available and a mechanism is provided for building composite characters by combining these marks with other symbols. (This not only provides an alternative way of making accented letters, it also allows for the habit mathematicians have of making up new symbols by decorating old ones.) Code values for nearly 39,000 symbols are provided, leaving some code points unused. Others are reserved for the UTF-16 expansion method (described briefly later on), while a set of 6,400 code points is reserved for private use, allowing organizations and individuals to define codes for their own use. Even though these codes are not part of the Unicode standard, it is guaranteed that they will never be assigned to any character by the standard, so their use will never conflict with any standard character, although it might conflict with other individuals'.

Unicode is restricted to characters used in text. It specifically does not attempt to provide symbols for music notation or other symbolic writing systems that do not represent language.

Unicode and ISO 10646 were brought into line in 1991 when the ISO agreed that the plane $(0, 0, *, *)$, known as the *Basic Multilingual Plane (BMP)*, should be identical to Unicode. ISO 10646 thus utilizes

CJK consolidation, even though its 32-bit code space does not require it. The overwhelming advantage of this arrangement is that the two standards are compatible (and the respective committees have pledged that they will remain so). To understand how it is possible to take advantage of this compatibility, we must introduce the concept of a character set *encoding.*

An encoding is another layer of mapping, which transforms a code value into a sequence of bytes for storage and transmission. When each code value occupies exactly one byte, it might seem that the only sensible encoding is an identity mapping: each code value is stored or sent as itself in a single byte. Even in this case, though, a more complex encoding may be required. Because 7-bit ASCII was the dominant character code for such a long time, there are network protocols that assume that all character data is ASCII, and remove or mangle the top bit of any 8-bit byte. To get round this limitation, it may be necessary to encode 8-bit characters as sequences of 7-bit characters. One encoding used for this purpose is called *Quoted-Printable (QP).* This works quite simply: any character with a code in the range 128-255 is encoded as a sequence of three bytes: the first is always the ASCII code for =, the remaining two are the codes for the hexadecimal digits of the code value. For example, é, has value 233 in ISO Latin1, which is E9 in hexadecimal, so it is encoded in QP as the ASCII string =E9. Most characters with codes less than 128 are left alone. An important exception is = itself, which has to be encoded, otherwise it would appear to be the first byte of the encoded version of some other character. Hence, = appears as =3D.

To interpret a sequence of bytes correctly we need to know which encoding and which character set is being employed. Many systems rely on convention, or stipulate (or quietly assume) that all text is encoded in ISO Latin1. A more flexible approach is to use an additional feature of the MIME content type specifications which we introduced in Chapter 2. Following the type and subtype, a character set specification of the form ; charset = *character set* may appear. A set of 'official' character set names is maintained by IANA. For example, the type of a Web page composed in ISO Latin1 is

```
text/html; charset = ISO-8859-1
```

This information should be included in HTTP response headers and MIME-encoded email messages. It can also be specified using appropriate tags in a mark-up language, as explained in Chapter 8.

For ISO 10646 the obvious encoding scheme, known as UCS-4, employs four bytes to hold each code value. Any value on the BMP will have the top two bytes set to zero; since most values that are currently defined are on the BMP, and since economic reality suggests that for the foreseeable future most characters used in computer systems and transmitted over networks will be on the BMP, the UCS-4 encoding wastes space. ISO 10646 therefore supports an alternative encoding, UCS-2, which does the obvious thing and drops the top two bytes. UCS-2 is identical to Unicode.

Unicode encodings go further. There are three *UCS Transformation Formats (UTFs)* which can be applied to Unicode code values. UTF-8 takes the reasoning we just applied to 32-bit values a step further. ASCII code values are likely to be more common in most text than any other values. Accordingly, UTF-8 encodes UCS-2 values so that if their high-order byte is zero and the low-order byte is less than 128, the value is encoded as the single low-order byte. That is, ASCII characters are sent as themselves. Otherwise, the two bytes of the UCS-2 value are encoded using up to six bytes, with the highest bit of each byte set to 1 to indicate it is part of an encoded string and not an ASCII character. Text encoded with UTF-8 is thus a string of 8-bit bytes, and is therefore vulnerable to mangling by protocols that can only handle ASCII. UTF-7 is an alternative encoding which uses a technique similar to that described for QP to turn Unicode characters into streams of pure ASCII text, which can be transmitted safely.

The UTF-16 encoding has a different emphasis. This encoding allows pairs of 16-bit values to be combined into a single 32-bit value, thus extending the repertoire of Unicode beyond the BMP. Only values in a limited range can be combined this way, with the result that UTF-16 only provides access to an additional 15 planes of the full ISO 10646 character set. These comprise nearly a million characters under UTF-16, which seems to be sufficient for present purposes.

To summarize: ISO 10646 is a 32-bit character code, arranged in 256 groups, each of which consists of 256 planes accommodating 65,536 characters each. The UCS-4 encoding utilizes four bytes to hold the full 32-bit code value for any character; the UCS-2 encoding utilizes just two bytes, to hold 16-bit values for characters on the $(0, 0, *, *)$ plane. UCS-2 is identical to Unicode. ISO Latin1 is the eight bit code equivalent to the $(0, 0, 0, *)$ row of ISO 10646, and ASCII is a subset of ISO Latin1. UTF-8 allows any ISO 10646 or

Unicode value to be encoded as a sequence of 8-bit bytes, such that ASCII values are left unchanged in a single byte. UTF-16 is an extension mechanism which provides Unicode with access to an extra 15 planes of the full ISO 10646 character set.

The markup languages HTML and XML, described in Chapters 8 and 13, and the Java programming language use Unicode as their character set. Once it also becomes adopted as the native character set of the major operating systems, the task of transferring text that uses characters beyond the US-Anglophone ASCII repertoire should be substantially eased.

Fonts

To display a piece of text, each stored character value must be mapped to a visual representation of the character's shape. Such a representation is called a *glyph*. As Figure 7.2 shows, a single character can be represented by a multitude of different glyphs. Many small details can be changed without destroying the fundamental identity of the abstract character being represented. We can recognize all the glyphs in Figure 7.2 as lower-case letter q. In addition to the variations in shape which that figure illustrates, glyphs can vary in size, from tiny subscripts to banner headlines over an inch tall. If we are to be able to use glyphs systematically, it is necessary to impose some organization on them.

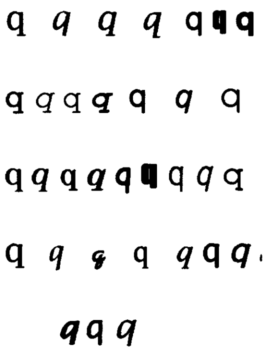


Figure 7.2
Glyphs for lower case q

Glyphs are arranged into collections called *fonts*. The name has been taken from traditional printing. In the letterpress (or 'cold metal') technique a page is assembled out of individual pieces of type, each one of which is a metal block, with a mirror image of a letter or other symbol in relief on one end. These blocks are clamped in a frame together with spacers, and pages are printed by inking the faces of the type and pressing the paper against them. In this context, a font is a collection of pieces of type — actual metal objects. Usually, the typesetter works from a cabinet in which the type is arranged in drawers holding multiple instances of each letter shape at each size that is required. All the shapes in a particular font will have been made from masters produced by a type designer so that they share certain visual characteristics, and combine well with each other to make a harmonious and readable page.

Letterpress has been largely superseded, first by mechanical hot metal technologies, such as Monotype and Linotype machines, and now by digital typesetting. The concept of a font has been retained through these technological changes, although its realization has evolved, first into a set of moulds into which hot lead is poured to make letters for typesetting machines, and then into a computer file that holds a description of the graphic forms of the glyphs. The idea that a font combines a set of glyphs that are visually related and designed to work together has not changed. In fact, many of the fonts available for use on computers today are versions of traditional typesetters' fonts, some of which are based on original designs from the 15th century.

Accessing Fonts

Text originates as a stream of character codes. To turn it into a form that can be displayed, these must be replaced by glyphs chosen from one or more fonts. Once this replacement has been performed, the content of the text (the character codes) becomes lost and can only be retrieved with difficulty. Essentially, the text turns into graphics, which cannot easily be searched or edited. (A person can still read the text, of course — in fact, they can't read it in any other form — but a program cannot manipulate it in any way.) What is more, since a stream of character codes is much more compact than an image representing the same text, replacement of character codes by glyphs also increases the size of the text. The graphical representation of text is inefficient because it takes no advantage of the repetitive nature of text: the same character will typically occur many times, so if glyphs are substituted, many copies of the same glyph will be used, and each will be much larger than the corresponding character code.

⇨ There is an exception to our observation that text originates as a stream of character codes. If a page of printed text is scanned the result is a bitmapped image of the page. If you want to get hold of the characters from a scanned text you need to use special *optical character recognition (OCR)* software, which uses pattern recognition techniques to analyse the shapes within the bitmapped image and deduce the characters they represent. OCR is a mature technology, but 100% accurate recognition is rarely achieved.

Because of these considerations, it is normal to keep text in some character-based form, and only to use glyphs when it is actually displayed or incorporated into a graphic image. The simplest case is that of *monostyled* text: text that is displayed in a single font. In that case, a text file need only contain the characters of the text itself: it is the familiar 'plain ASCII' (or perhaps plain ISO 8859) file. When such a file is displayed, the character codes are used to select glyphs from a font, either a default system font, or a single font selected by the user from within the program being used to display the text.

Monostyled text is rarely adequate for multimedia production, where a richer typographic experience is usually required. The principle of selecting glyphs using character codes remains the same, but additional information is needed to control the selection of fonts. We will describe various forms this information may take in Chapter 8. At this point, we must consider the question: Where might fonts be found?

There are only two possibilities. Glyphs must be taken either from fonts stored on the system[6] being used to display the text, or from fonts embedded in the text file. In the latter case, the fonts must originally have been stored on the system used to prepare the text, and therein lies the most important advantage of this approach: multimedia designers can use fonts of their own choosing, confident that they will be available when they are needed, because they are embedded in the same file as the text. If they are not, then only the fonts on the user's system will be available, and there is no reason to suppose that the user's selection will be the same as the designer's, so there is a possibility that some font required by the text will not be available. The probability of this occurring increases if the designer chooses esoteric fonts — as designers sometimes will.

Why then might you prefer not to embed fonts in your text files? One potent reason is that not all text file formats allow you to do so. In particular, HTML presently has no facility for embedding fonts,[7] although, in conjunction with CSS stylesheets, it does allow you to specify the fonts you wish to be used to display textual elements of Web pages. The other reason for preferring to rely on fonts on the user's system is that fonts are fairly bulky objects: the fonts used for this book, for example, vary in size between about 32 and 50 kilobytes each. It is quite easy to find yourself using half a dozen fonts on a page, so including the fonts with the text can lead to bloated files and consequently to extended download times.

[6] Exactly what is meant by 'stored on the system' is system-dependent: on a MacOS system, for example, fonts must be placed in the `Fonts` sub-folder of the `System Folder` (unless a font management utility is being used); on a system running X11 under Unix, fonts must be in some directory included on the user's FONTPATH.

[7] Although several mechanisms for downloading fonts have been proposed, none has been adopted as a standard at the time of writing.

If a font is not available, various more or less undesirable consequences can result. In the extreme, the result might be a system crash. More likely, text in the missing font will not be displayed, or it will be displayed in some other font that has been substituted. The last case is the least serious, and the most common, but it is far from ideal. As we will see in the next section, every font has its own distinctive character, so substituting another in its place can seriously impair a carefully thought out design. Furthermore, the widths of the glyphs in the substituted font will probably be different from those in the intended font, so that the positioning of individual glyphs on the page will be incorrect. Depending on how the layout of text for display is done, this might result in ragged margins where straight ones were intended, uneven gaps between words, or gaps within words. Any of these defects will have a detrimental effect on the appearance of your text, and may make it hard to read.

Classification and Choice of Fonts

There are thousands of fonts available, each with its own special personality. Some broad characteristics are used to help classify them.

A major distinction is that between *monospaced* (or *fixed-width*) and *proportional* fonts. In a monospaced font, every character occupies the same amount of space horizontally, regardless of its shape. This means that some letters have more white space around them than others. For example, the narrow shape of a lower-case l must be surrounded with white to make it the same width as a lower-case m. In contrast, in a proportional font, the space each letter occupies depends on the width of the letter shape. Paradoxically, this produces a more even appearance, which is generally felt to be easier to read in most contexts. In addition, it allows you to fit more words on to a line. Text in a monospaced font looks as if it was produced on a typewriter or a teletype machine. It can sometimes be used effectively for headings, but is especially suitable for typesetting computer program listings. It is also useful for conveying a 'low-tech' appearance, for example in correspondence where you wish to convey a slighty informal impression. Text in a proportional font has the appearance of a traditional book, and is usually preferred for setting lengthy texts. It is generally felt to be more readable, since letters appear to be

Monospaced Font: Lucida Typewriter
Each letter occupies the same amount of horizontal space, so that the text looks as if it was typed on a typewriter.

Figure 7.3
A monospaced font

tightly bound together into words. Figures 7.3 and 7.4 illustrate the difference between monospaced and proportional fonts.

Proportional Font: Lucida Bright
Each letter occupies an amount of horizontal space proportional to the width of the glyph, so that the text looks as if it was printed in a book.

Figure 7.4
A proportional font

Probably the most widely used monospaced font is Courier, which was originally designed for IBM typewriters, and has achieved wide currency because it is one of the fonts shipped as standard with all PostScript printers. Many proportional fonts are in common use. As you can no doubt discern, the font used for the main text in this book, Lucida Bright, is proportionally spaced. Most of the classical book fonts, such as Times, Baskerville, Bembo, and Garamond are also proportional, as are many newer fonts, such as Helvetica.

Another very broad distinction is between *serifed* and *sans serif* (sometimes *sanserif*) fonts. *Serifs* are the little strokes added to the ends of character shapes (see Figure 7.5). These strokes are present in serifed fonts, but omitted in sans serif fonts, which consequently have a plainer look. Serifs originate in marks produced by chisels on Roman stone inscriptions, so serifed fonts are sometimes called Roman fonts. Figure 7.6 shows some text in Lucida Sans, a sans serif version of the Lucida Bright font.

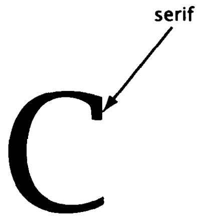


Figure 7.5
Serifs

Sans serif fonts are a comparatively new development, on a typographical time scale, and have only gradually gained acceptance for general use — in some typographical catalogues they are still identified by the alternative name of *grotesques*. In the nineteenth century, sans serif fonts were indeed rather grotesque, being crude designs used mostly for advertising and posters. Only in the twentieth century have they become accepted for use in books, with the development of more elegant and refined designs. The best known sans serif font is probably Helvetica, which again is one of the standard PostScript fonts. Other sans serif fonts you may meet include Univers and Arial (the latter popularized by Microsoft). Gill Sans, another very popular sans serif font, was based on the font used for the signs on the London Underground, which had helped to generate interest in sans serif type.

Sans Serif Font: Lucida Sans
The letters of a sans serif (or sanserif) font lack the tiny strokes known as serifs, hence the name. They have a plain, perhaps utilitarian, appearance.

Figure 7.6
A sans serif font

There is contradictory evidence as to whether serifed or sans serif fonts are more readable — spacing probably makes more difference. However, serifs are very small features, and therefore difficult to render accurately at low resolutions, which can mean that text in a serifed font is hard to read on a computer screen, simply because the letters are not being accurately reproduced. Sans serif fonts are widely used for such features as window titles and menu entries on these grounds.

Spacing and serifs are independent properties: sans serif and serifed fonts can equally well be either monospaced or proportional.

A third classification of fonts is based on broad categories of *shape*. In particular, we distinguish between fonts with an *upright* shape, and those with an *italic* shape. Upright fonts, as the name implies, have characters whose vertical strokes (stems) are indeed vertical. Italic fonts imitate a certain style of handwriting, and have letters that are slanted to the right. Additionally, the letter shapes in an italic font are formed differently from those in an upright font, so that they share some of the characteristics of italic handwriting. When digital fonts first appeared, 'italic' fonts were sometimes produced simply by applying a shear transformation to an upright font. The effect is rather different, since the calligraphic features of true italic fonts are missing. Such fonts are now used in their own right, not as substitutes for italics. They are said to have a *slanted* shape. The difference is illustrated in Figures 7.7 and 7.8, which show italic and slanted versions of the upright font used for the main text of this book.

Italic Font: Lucida Bright Italic
The letters of an italic font slope to the right, and are formed as if they were made with an italic pen nib. Italics are conventionally used for emphasis, and for identifying foreign words and expressions.

Figure 7.7
An italic font

Most italic fonts are variations on or designed as companions to upright fonts. For example, Times Italic is an italic version of Times Roman. There are, however, some fonts with an italic shape which are designed on their own; these are usually intended to have the character of handwriting, and to be used where something more human than a conventional typeface is desired. Among the best known calligraphic fonts are several versions of Chancery, including Zapf Chancery. Specialist fonts based on samples of real handwriting are also available. Figures 7.9 and 7.10 are examples of a relatively formal calligraphic font and a handwriting font.

Slanted Font: Lucida Bright Oblique
The letters of a slanted font share the rightward slope of italic fonts, but lack their calligraphic quality. Slanted fonts are sometimes used when a suitable italic font is not available, but may be preferred to italics when a more modern look is wanted.

Figure 7.8
A slanted font

⇨ One type of font that doesn't quite fit our classifications is the *small caps* font, in which small versions of the upper case letters are used as lower case letters. This variation is sometimes considered to be a shape, although the sense is somewhat different from that in which italic is considered a shape. Small caps fonts are most useful for trademarks and so on, which are normally written entirely in upper case: full size upper case words stand out awkwardly in ordinary text, but small caps are less intrusive.

Calligraphic Font: Lucida Calligraphy
Calligraphic fonts usually resemble 'round hand' or 'copperplate' handwriting, unlike italic fonts.

Figure 7.9
A calligraphic font

Digital technology has made some shapes widely available that previously could not be easily produced. In particular, outline fonts, which have hollow letters, and fonts with drop shadows, as illustrated in Figure 7.11, can be easily constructed using graphics effects. Such fonts should be viewed as gimmicks and used judiciously, if at all.

Handwriting Font: Lucida Handwriting Handwriting fonts are based on samples of real people's handwriting, so they are often quite idiosyncratic.

Figure 7.10
A handwriting font

Drop shadows create a slight 3-D effect

Figure 7.11
A font with drop shadows

Some fonts appear somewhat squashed horizontally, compared with the normal proportions of most fonts. They are referred to as *condensed* fonts, and are intended for applications, such as marginal notes or narrow newspaper columns, where it is desirable to be able to fit text in as tightly as possible. In contrast, some fonts, described as *extended* are stretched out horizontally, making them more suitable for headings, and other isolated text elements.

Finally, fonts can be classified according to their *weight*, that is, the thickness of the strokes making up the letters. Thicker strokes make text look darker and more solid. Conventionally, we call fonts with a heavy weight (thick strokes) *boldface* or simply *bold*. Like italics, bold fonts are usually versions of other fonts with a lighter weight. As a result, boldness is not an absolute property: a bold version of a font whose normal weight is particularly light may be lighter than the normal version of a heavy font, whose own bold version will be even heavier. Some fonts may exist in several versions, exhibiting varying degrees of boldness. Under these circumstances, individual styles are described by terms such as *ultra-bold*, *semi-bold*, *light* and *ultra-light*.

Conventional typesetting wisdom has it that boldface is intrusive, and should be reserved for headings and similar uses. Because of the limitations of computer displays, it is sometimes advisable to use boldface more widely than would be appropriate in text intended for paper. For example, conventionally you *never* use boldface for emphasis, always italics. However, italic text, because of its slant, often renders badly at low resolutions, making it hard to read. Under these circumstances, the use of bold text for emphasis may be justified. Bold fonts are also used quite widely for window titles and menu items, because they show up well.

⇨ Word processors often treat underlining as a styling option similar to italicization and emboldening, so you might think that there would be underlined versions of fonts, too. Underlined fonts are extremely rare, though: underlining is scorned in typographic circles, where it is considered to be a poor substitute for italicization, only suitable for typewriters which lack italics. Like most conventional typographical wisdom, the undesirability of underlining is no longer unquestioned. Since more flexible effects can be produced by combining ordinary fonts with lines of various thicknesses, this questioning has not led to an outbreak of underlined fonts.

Because it makes sense to talk about an italic version or a bold version of an upright font, fonts can be grouped into families.[8] Usually, an upright serifed font is considered to be the normal form, which is augmented by versions in different weights, perhaps an italic form or a sans serif form. Variations are often combined to produce additional versions, such as bold italics, or slanted sans serif. For example, the Lucida Bright family consists of twenty fonts:[9] Lucida Bright is the upright serifed font you are reading now; it is also available in bold, italic, and bold italic. Lucida Sans is the sans serif version, which comes in the same four versions, as do Lucida Typewriter, a fixed width Lucida font, and Lucida Fax, a variant form designed to be especially readable at low resolution. Three calligraphic fonts and a slanted font complete the family. All twenty fonts share a similar feel so that they can be combined without any obtrusive visual discontinuities. In contrast, when fonts from different families are combined, their differences can be very noticeable, as Figure 7.12 shows. Traditionally, such discontinuities have been carefuly avoided, but in recent years designers have taken advantage of the ease with which desktop publishing software allows them to combine fonts to explore new combinations that defy the established conventions.

Lucida Bright goes well with *Lucida Bright italic,* ***and bold italic,*** but not nearly so well with Palatino.

Figure 7.12
Mixing fonts from different families

In the days of letterpress and hot metal typesetting, to produce text at different sizes required different sets of type, which would qualify as separate fonts. Only a limited number of different sizes could easily be made available. With digital fonts, arbitrary scaling can be applied to glyphs, so that fonts can be printed at any size. Purists maintain that simple scaling is not adequate, and that letter shapes should be designed to look their best at one particular size, and should only be used at that size. Contemporary font technology, however, tends to the idea that a font should exist at only one size and should be scaled to produce any other size that is needed. Special information contained in the font is used to help maintain a pleasing appearance at all sizes, as we will describe in a later section.

As well as the objective factors of spacing, serifs, shape, weight and size that can be used to classify type, there is a more subjective classification based on the sort of jobs for which a font is most suitable. The basic distinction is between *text fonts* and *display fonts.* The terminology is slightly silly, since all fonts are used for text. The distinction is between fonts suitable for continuous text, such as the body of a book or article, and those suitable for

[8] A font family corresponds closely to what is traditionally called a *typeface.* A font is a particular style of some typeface. In pre-digital typography each different size would be considered a separate font, but nowadays we usually consider that the same font can be rendered at different sizes.

[9] Actually, there are 25, the additional ones are for mathematics.

short pieces of isolated text, such as headings, signs or advertising slogans on posters.

⇨ Sometimes, a finer distinction is drawn within the class of display fonts. *Decorative* fonts are those for which appearance is the primary design consideration. They often incorporate ornaments and other features that make them quite unsuitable for extended use. *Headline* fonts, as the name implies, are designed for use in headlines and other situations where it is important to attract the reader's attention. This leaves a category to which the name display fonts is often attached, consisting of fonts intended especially for use at large sizes, with other features designed to take advantage of the possibilities offered by large characters; for example the serifs might be especially fine.

Text fonts must be unobtrusive, so that they do not intrude on the reader and interfere with the primary message of the text. They must also be easy to read, so that they do not cause fatigue when they are read for hours at a time. To some extent, whether or not a font is intrusive depends on whether it is familiar; at the same time, the criteria for selecting text fonts have not changed over the years. The combination of these factors means that text fonts tend to be conservative. Invariably they are upright, more often serifed than not, and of a medium weight.

A Display Font: Bodoni Highlight

Display fonts are designed for short pieces of text, such as headlines. They are not intended for use in lengthy passages.

Figure 7.13
A display font

Display fonts are another matter. Here, the intention is to get across a short message. Garish design that would be offensive in a text font becomes eye-catching; innovation attracts attention. There is a far greater diversity of design among display fonts than among text fonts (Figure 7.13 shows just one example), and whereas the same text fonts continue to be used year after year, display fonts are subject to fashion.

⇨ Digital font technology has made the mechanics of producing new fonts much simpler than it was when type had to be created out of metal. It has also made possible some effects that could not previously be physically achieved. Consequently, recent years have seen an acceleration in the rate of innovation in font design, and the introduction of aggressively unconventional display fonts. Opinion is predictably divided over the virtues of 'grunge typography', as this new design movement is sometimes called.[10] In some circles it is condemned as unreadable and ugly, while in others it is considered to be the most exciting development in type design since the invention of the printing press. The ease with which grunge typefaces have been absorbed into mainstream advertising suggests that they might not be especially revolutionary after all.

[10] Although you probably won't be surprised to learn that it is also sometimes called 'post-modern' or (what else?) 'deconstructivist'.

Conventional ideas about font usage are based on centuries of experience of making books, pamphlets, posters, packaging, road signs, shop fronts and other familiar forms of printed matter. While much of this experience can be applied to the textual components of multimedia, some aspects of the new media demand a fresh approach. Generally, display fonts can be used on a computer monitor in much the same way as they can on paper, so where text is combined with images, fonts can be used as they would be on a poster; attention-grabbing headings can effectively use the same display fonts as you might find in a book. Continuous text is more problematical: the low resolution of most monitors can lead to distortion of letter shapes, making fonts that work well on paper hard to read, especially at small sizes. The obvious solution is to use fonts at larger sizes than is customary in books, and this is often done: text for electronic display is often set as much as 60% larger than text in ordinary books. An alternative is to look for a font that has been designed to be readable at low resolution; Monaco and Arial are examples of such fonts. Sans serif fonts tend to survive better at low resolutions, which makes them more suitable for this purpose.

The way the text in multimedia productions is arranged gives rise to some less technical questions concerning fonts. Long continuous passages covering many pages are cumbersome and tiring to read on a screen, and do not integrate well with other media. It is common, therefore, to find multimedia text elements constructed as small pieces that fit onto one or two screen-fulls. These pieces will often resemble, in form and style, the short explanatory placards attached to museum displays and zoo exhibits. This sort of text partakes of the character of both continuous text for which we would use a text font — it has a non-trivial extent and content — and the shorter texts normally set in a display font — it is succinct and usually has a message to get across. A restrained display font will often work well in this situation, as will some quite unconventional solutions, such as a text font in a large bold version, or coloured text on a contrasting background.

Text for multimedia is often prepared using the same tools as are used in ‘desktop publishing’ (DTP). It is important to remember that the output from conventional DTP programs is intended for printing on paper. We reiterate that text that will look excellent when printed on paper may well be unreadable when viewed on a computer monitor. It is necessary to adapt the way you use these tools, bearing in mind the actual medium for which your output is destined, or to use tools that have support for output to alternative media.

One final consideration which is unique to digital text is that, in many cases, the multimedia designer has no control over the fonts that will be used when text is finally displayed. As we explained previously, it may be necessary for completely different fonts to be substituted on different computer systems; in some cases, the software used for display may let users override the original fonts with those of their own choosing. Consequently, unless you know that neither of these circumstances will arise, there is no point in carefully exploiting the features of a particular font to achieve some special effect — the effect may never be seen by anyone except you.

Most of the preceding description of fonts has concentrated on letters, implicitly in the Latin alphabet. However, as the discussion of character sets in the previous section indicated, we use far more characters than these letters, and just as we need character sets to provide code values for a large character repertoire, so we need fonts to provide glyphs for them. In fact, you can think of a font as a mapping from abstract characters to glyphs, in much the same way as a character set is a mapping from abstract characters to code points. Like a character set, a font is only defined for a specific character repertoire. Most fonts' repertoires consist of the letters from some alphabet, together with additional symbols, such as punctuation marks. You might be forgiven for hoping that fonts would be structured around one of the character set standards, such as ISO Latin1, but this is not usually the case. It is generally possible to access the individual glyphs in a font using a numerical index, but, although the printable ASCII characters are usually in their expected positions, other symbols may well be in positions quite different from that corresponding to their code values in any ISO character set.

The way in which glyphs are grouped into fonts owes more to printing tradition than to the influence of character code standards, with the alphabet being the focus of design. Even then, some fonts may not include lower case letters, if the designer intended the font only to be used for headlines, for example. Specialized non-alphabetic symbols, such as mathematical symbols are usually grouped into their own fonts, known as *symbol fonts* or *pi fonts*. Some fonts consist entirely of graphic images: you may encounter fonts of arrows, or fonts of pointing hands, for example. Some images are considered to be symbols and are included in character sets — fonts containing these symbols are called *dingbat fonts* — but others are put together by font designers for special applications, and do not fit into any character set framework.

⇨ A particularly interesting example is available as we write (in mid-1998): the impending adoption of a single currency, the Euro, by most of the member states of the European Union has led to the design of a new currency symbol to designate it. This symbol (which looks a bit like €) is not present in any fonts predating the Euro. Some font vendors have responded by producing fonts consisting of nothing but Euro symbols, in different styles. Instead of choosing the Euro symbol from your text font, you use the Euro font and choose whichever version most closely matches the style of the surrounding font.

As a result of all this, another level of encoding is required in order to map stored character codes in some character set to glyphs. Additionally, some mechanism may be needed to combine separate fonts, for example, an alphabetic font and a symbol font, to provide glyphs for all the characters used in a document — a problem which will become acute as the wider range of characters provided by Unicode comes into wider use. Fortunately, the mechanisms for accessing glyphs are usually handled transparently by text layout software.

Font Terminology

Typography has its own specialized vocabulary, some of which it is necessary to understand if you are to find your way around among fonts and their descriptions.

Much of the description of a font's characteristics consists of measurements. These are usually given in units of *points (pt)*. In digital typography, one point is 1/72 of an inch, which makes 1pt equal to just under 0.3528mm. A point is thus a very small unit, suitable for measuring the dimensions of such small objects as typeset characters. For slightly larger quantities, such as the distance between lines of text, we often use a *pica (pc)*, which is equal to 12pt (or 1/6in or 4.2333mm).

⇨ Unlike other units of measurement, such as the metre or the foot, the point does not have an internationally agreed standard magnitude. The value of exactly 1/72in is a comparatively recent innovation, pioneered by PostScript. In English-speaking countries, the printer's point is 1/72.27in, and some software, such as TeX, still uses this value by default. In France, the Didot point, which is about 7% larger than the printer's point, is preferred.

A font's size is quoted in points, as in '12pt Times Roman' or '10pt Lucida Sans'. The value specified is technically the font's *body size.* In the days of letter press, the body size was the height of the individual pieces of metal type, each of which were the same size so that they could be fitted together. Hence, the body size was the smallest height that could accommodate every symbol in the font. This remains the case: the size is not necessarily equal to the height of any character in the font (although parentheses are often as tall as the body size), it is the height between the top of the highest character and the bottom of the lowest.

⇨ This is often the case, but the body height is actually an arbitrary distance, chosen to provide a suitable vertical space for the letters to sit in when they are set in lines. A font with a particularly open character might have a body size somewhat larger than is necessary just to accommodate all its characters, so that there is always some extra space around them when text is set in that font on conventional baselines.

In normal text, characters are arranged so that they all sit on the same horizontal line, just as, when we write on lined paper, we ensure that all our letters are on the lines. This line is called the *baseline.* An important font dimension is the height between the baseline and the top of a lower-case letter x. This value is the font's *x-height*; the bodies of most lower-case letters fit in between the baseline and the x-height. Some letters, such as h, have strokes that rise above the x-height; these are called *ascenders.* Similarly, letters such as y extend below the baseline; the extending strokes are called *descenders.* These terms are illustrated in Figure 7.14. Sometimes, the size of the largest ascender (the distance between the x-height and the top of the body) is called the *ascent*, and the corresponding descending dimension is called the *descent* of the font.

Ascenders are not the only things that extend above the x-height: capital letters do so as well. In many fonts, though, the capitals do not extend to the full height of ascenders, so we need another quantity, the *cap height*, also shown in Figure 7.14, to characterize the vertical extent of capitals. (In English text, the difference between cap height and ascent is most obvious when the combination 'Th' occurs at the beginning of a sentence.)

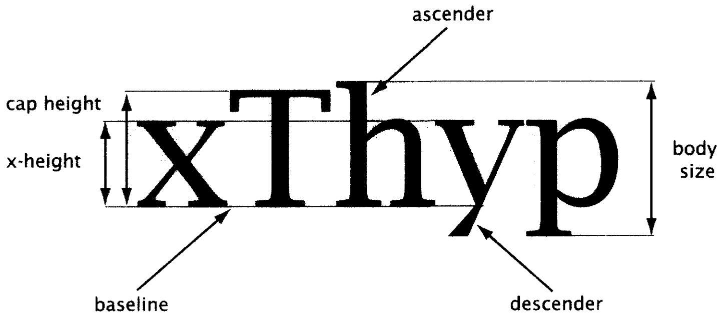


Figure 7.14
Some font terminology

The ratio of the x-height to the body size is one of the most important visual characteristics of a font. Since the bulk of the text lies between the baseline and the x-height, a font with a relatively high x-height will look bigger than a font with a lower x-height at the same size. If you look back at Figure 7.12, you will be able to see that one of the things that prevents the text in Garamond merging happily with that in Lucida Bright is that the latter font has a much higher x-height. This feature is common to the entire Lucida family, so the Lucida Italic mixes easily with the upright Lucida Bright.

⇨ If you look closely at Figure 7.14 you will see that the curved tops of the h and p actually extend through the x-height. This phenomenon is called *overshoot*, and helps make the letters look more uniform in size. The extent of the overshoot is another factor that characterizes the look of a font.

The x-height of a font is used as a unit of measurement, usually written ex. It has the useful property of not being an absolute unit, like a point, but a relative one; it changes with the font's size, and is different for different fonts, but always stands in the same relation to the height of lower-case letters, so it provides a convenient way of expressing vertical measurements that should change in proportion to this quantity. A similar unit of horizontal distance is the em. Traditionally, 1em is the width of a capital letter M. In many fonts, M is as wide as the body size, so the meaning of 1em has migrated over the years, and is now often taken as a unit of length equal to the font size;[11] for a 10pt font, 1em is equal to 10pt, for a 12pt font it is equal to 12pt, and so on. Long dashes — like these — used for parenthetic phrases are 1em long, so they are called *em-dashes.* You will sometimes see another relative unit, the *en*, which is the width of a capital N, and usually defined as 0.5em. An *en-dash* is 1en long; en-dashes are used for page or date ranges, such as 1998–99.[12]

[11] The em unit is defined like this in CSS (see Chapter 8), for example.

[12] There are no fixed rules about dash usage. Some publishers stipulate the use of en-dashes for all purposes. You should never use a hyphen instead of a dash, though.

Other features which characterize the look of a font include the size and shape of serifs, and the ratio of the thickness of thick and thin strokes. For example, so-called *modern* fonts (actually based on designs about 200 years old) are characterized by high contrast between the thick and thin strokes, and serifs without brackets (the curves that join the serif to its stem). Fans of these fonts[13] believe these features produce a brilliant, sophisticated appearance on the page, while their detractors find them affected and illegible, compared to the *old-style* fonts with their more uniform solid appearance, or the cleaner lines of twentieth century sans serif fonts. The implications for display at low resolution should by now be clear.

The features of individual letters are not all we need to consider in a font. We also have to look at how letters are combined. As we stated earlier, most text fonts are proportionally spaced, with each letter occupying as much horizontal space as it needs to. As Figure 7.15 shows, each letter has a *bounding box*, which is the smallest box that can enclose it. Normally, a slight gap is left between the bounding boxes of adjacent characters. In other words, when drawing a glyph immediately following another one, it is drawn relative to a *glyph origin*, which lies outside of the bounding box, usually to its left, by a distance known as the character's *left side bearing*, as shown in Figure 7.15 (where we have considerably exaggerated the magnitude of the side bearing).

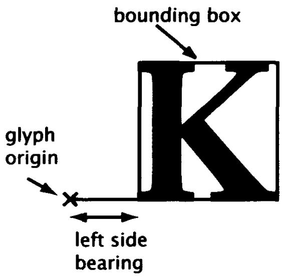


Figure 7.15
Side bearings

Sometimes, when two particular letters are placed next to each other, the total amount of space between them looks too great or too small. Typographers normally adjust the spacing to make it look more uniform. (Once again, you will notice the apparent contradiction that it is necessary to introduce non-uniformity to achieve the subjective appearance of uniformity.) This process of adjustment is called *kerning*, and is illustrated in Figure 7.16. (Look closely, kerning is subtle.) The kerning pairs for a font are defined by its designer. Carefully produced fonts may have hundreds of kerning pairs, each with its own spacing. The information about which pairs of characters are to be kerned, and by how much, is stored as part of the font.

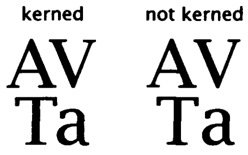


Figure 7.16
Kerning

The measurements which describe the size of individual characters and the spacing between them are collectively known as *font metrics*. Programs that perform typesetting need access to font metric information, in order to determine where to place each glyph as they build up lines, paragraphs and pages. The organization of

[13] Yes, fonts have their fans. Indeed, once you start to look closely at fonts, you may find you start to develop very strong feelings about them. You have been warned.

this information depends on the font format, computer system and typesetting software being used.

⇨ Two operations related to kerning are *letter spacing* and *tracking.* Letter spacing is the process of changing the space between all letters in a font, stretching or squeezing the text. This may be done either to change the appearance of the font, or to fit an awkward line of text on to a page. (The latter operation is severely frowned on in the best typesetting circles.) Tracking is a systematic application of letter spacing depending on the font size: large letters need proportionally less space between them to look right.

Certain character combinations just will not look right, no matter how you space them. Printers traditionally replace occurrences of such troublesome sequences by single composite characters, known as *ligatures.* In English text, commonly used ligatures include ff, fl, fi and ffl. There is no standard set of ligatures, and some fonts provide more of them than others. Monospaced fonts do not usually provide any. Figure 7.17 shows why we need ligatures, and the improvement achieved by using them. Ligatures are stored as extra characters in the font.

without: fine fluffy soufflés
with: fine fluffy soufflés

Figure 7.17
Ligatures

High quality text layout software will automatically deal with kerning and ligatures, making the appropriate spacing corrections and character subsitutions when it encounters the codes for characters that require such treatment. Word processors and Web browsers cannot generally do this.

Digital Font Technology

Glyphs are just small images, which can be stored in a font using either bitmaps or vector graphics. This leads to two different sorts of fonts, *bitmapped* fonts and *outline* fonts, based on these two graphics technologies. Glyphs in bitmapped fonts exhibit the same characteristics as bitmapped images, while outline glyphs share the characteristics of vector graphics. Since character shapes are usually built out of simple curves and strokes,[14] and lack any subtle texture, there is little advantage in using bitmapped fonts on the grounds of expressiveness. The main advantage of bitmapped fonts is that glyphs can be rendered on-screen simply and rapidly, using a bit copying operation. This advantage is being eroded by the wider availability of efficient graphics hardware and faster processors, but bitmapped fonts remain widely available.

[14] Indeed, there is a long tradition of using mathematical constructions to define glyphs. A recent example of this approach to font design is the METAFONT system, described in Donald E. Knuth, *The METAFONT Book*, Addison-Wesley, 1986.

The considerable disadvantage of bitmapped fonts is that, like any other bitmapped image, they cannot be scaled gracefully. The need for scaling arises in two different contexts: when a font is required at a different size from that at which it was prepared, and when it is required at a different resolution. The latter case is unlikely to arise with multimedia productions designed for display on screen, since the bitmapped fonts installed on computer systems are usually designed for use at screen resolution, but it presents serious problems if text must also be printed on a high resolution printer. The first case — scaling bitmapped fonts to produce type at a different size — always presents a problem. The irregularities that result from scaling a bitmap are especially intrusive with character shapes, owing to the amount of fine detail in their design. When bitmapped fonts are used, it is normal to provide versions of each font at several different sizes. Characters will then look as good as is possible at those sizes, but not at any others, so the designer is forced to use only the standard sizes.[15]

Each platform has its own native bitmapped font format, usually optimized in some way to work efficiently with the system software routines responsible for text display. In contrast, outline fonts are usually stored in a cross-platform format. The two most widely used formats are Adobe *Type 1* (often simply referred to as *PostScript fonts*, although there are other types of PostScript fonts) and *TrueType*.[16]

Outline fonts can be scaled arbitrarily, and the same font can be used for display at low resolution and for printing at high resolution, provided that the display software can interpret the outline shapes. The utility Adobe Type Manager (ATM) performs this function at the system level for Type 1 fonts; rendering of TrueType fonts is built into both the Windows and MacOS operating systems, hence outline fonts of both types can be used by any application program. Because of the supremacy of text preparation software directed towards print-based media, outline fonts are more widely used than bitmaps for elaborately formatted text.

Outline fonts in either of the popular formats can contain up to 256 glyphs. The glyphs in a Type 1 font are simply small programs, written in a restricted subset of PostScript. The restrictions are intended to make it possible for the glyphs to be rendered efficiently on screen by programs such as ATM, at sufficient quality to be acceptable at screen resolution. TrueType is an alternative format, based on quadratic curves instead of the cubic Bézier curves used

[15] This is no different from the situation in the days before digital fonts were available when, as we remarked earlier, each size of a typeface in a particular style was a separate font.

[16] The TrueType format was originally devised by Apple, but is more widely used on Windows than on MacOS, where, because of the popularity of Adobe software, Type 1 fonts are most common.

by PostScript. A character outline in a TrueType font is stored as a series of points which define the lines and curves making up its shape. A newly developed format, *OpenType* unifies Type 1 and TrueType fonts, by providing a file format that allows either sort of outline to be stored.[17]

⇨ You might also meet Adobe Type 3 fonts, although these are becoming rare — they date from a period when some aspects of the Type 1 specification were kept secret, and people wishing to develop their own PostScript fonts were obliged to use the inferior Type 3 format. There is also a Type 0 format, in which the elements of the font are not character glyphs, but other fonts. The Type 0 format is intended for storing composite fonts, each built out of several Type 1 fonts. Its main application is in providing fonts with a large character repertoire, such as might be required for use in conjunction with Unicode.

As well as descriptions of the glyph shapes, Type 1 and TrueType fonts both include extra information that can be used by a rendering program to improve the appearance of text at low resolutions. This information concerns rather subtle features, which tend to lose their subtlety when only a few pixels can be used for each stroke. In Type 1 fonts, it takes the form of declarative *hints*, which provide the values of parameters that can be used to direct glyph rendering at low resolutions. For example, some hint information is intended to cause overshoots to be suppressed. As we stated on page 209, overshoots normally help improve the regular appearance of a font. However, at low resolutions, when the amount of overshoot is rounded to the nearest pixel, the effect will be exaggerated and text will appear to be more irregular than it would if all lower-case letters were exactly as high as the x-height. The point at which overshoot suppression is effective depends on the design of the font, so it cannot be applied without information from the font designer. The Type 1 hints allow this information to be supplied. Other hints similarly suppress variations in stem width, which while aesthetically pleasing at high resolutions, merely look uneven at low resolutions. Another example of the use of hints concerns bold fonts. It is quite conceivable that the rounding effect of rendering at low resolution will cause the strokes of a normal font to thicken to such an extent that it becomes indistinguishable from the boldface version of the same font. A hint allows rendering software to add additional width to boldface fonts to ensure that they always appear bold. TrueType fonts achieve the same sort of improvements

[17] OpenType was developed by Adobe and Microsoft in 1997-8. OpenType fonts began appearing in 1998.

using a more procedural approach, allowing the font designer to write *instructions* that specify how features of a character should be mapped to points on a pixel grid at any resolution.

⇨ Although the use of hints and instructions improves the appearance of fonts at small sizes as well as at low resolutions, it only causes minor changes of shape to compensate for the small number of pixels available. Traditional typographers sometimes produce separate designs of the same font at different sizes, which is said to produce a more pleasing appearance than simply scaling one design to fit. Such niceties are rarely encountered with digital fonts, although the Computer Modern fonts, designed to accompany the TeX typesetting system, were produced in this way.

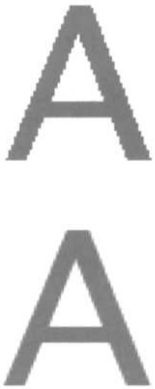


Figure 7.18
Anti-aliased text

As with other sorts of vector graphics, the appearance of outline fonts can sometimes be improved by anti-aliasing — softening a hard, and inevitably jagged, edge between those pixels that are on and off by using a pattern of grey pixels over a wider area. The smoothing effect thereby achieved is illustrated in Figure 7.18: the upper letter A is not anti-aliased, the lower one is. As this figure shows, anti-aliasing is very effective with large characters; it also works well at medium resolutions, such as that of a laser printer. At low resolutions, small type (below about 12pt), although noticeably smoother, also begins to look blurred, which may leave it harder to read than it would be if left jagged, so anti-aliasing should be applied to fonts judiciously.

Multiple Master Fonts

Before digital typography, each font provided a typeface in one style at one size. With most digital formats, each font provides a typeface in one style, and each size is generated from it by scaling, with the help of hints. Multiple Master fonts are a new development, based on Adobe Type 1 fonts, wherein each font provides an entire typeface, from which any style may be generated at any size.

Multiple Master fonts are conceptually fairly simple. Several characteristics of a font family can be considered as values along a line. For example, the weight of a particular font lies somewhere between ultra-bold and ultra-light (at least, relative to other members of the same family with different weights). A Multiple Master typeface comprises Type 1 fonts for each of the extremities of such a line, or *design axis*, as it is called. Fonts lying between the extremities can be constructed, as *instances* of the Multiple Master typeface, by interpolation: the weight of glyphs for a medium weight font might lie half-way between the ultra-bold and ultra-light glyphs. Potentially, an infinite number of fonts, differing finely in their weight, can be generated from the two extremities of the weight design axis.[18]

A single design axis is not always very exciting, but design axes can be combined. The Multiple Master format allows the ends of up to four axes to be stored in the same typeface file. Although the format is liberal enough to allow any characteristics to be varied along design axes, four specific ones are registered as part of the specification. They are weight, width (e.g. from condensed to extended), optical size, and serif style. The optical size axis is particularly interesting, since it permits features such as the proportions of letters, their weight and spacing to vary with the point size. This allows a closer approximation to the traditional practice of modifying character shapes to suit the size of type than is provided by Type 1 hints and TrueType instructions (which are primarily concerned with maintaining the quality of rendering). Figure 7.19 shows how three design axes can be combined into a font design space. By labelling each axis with values from 0 to 1, any point within the cube defined by the axes can be identified by three coordinates. The point corresponds to a font instance; the coordinates express the proportions of the characteristics of each design axis relative to its end-points that are present in the instance.

A Multiple Master typeface is larger than a single Type 1 font, since it must include a set of glyphs for each end of each design axis. However, it can be used to generate an entire font family — indeed, a family comprising many more individual font instances than you would normally find in a font family provided as a collection of individual fonts.

Multiple Master typefaces can also be used to provide a partial answer to the problem of font substitution. There are two problems with embedding fonts when you wish to distribute a document. The first, purely practical, is that incorporating fonts into a document may seriously increase its size. Not only does this make the document more difficult to distribute, but the recipient may not wish to accumulate large numbers of fonts, or multiple copies of the same font on their systems. The second problem is that fonts are subject to copyright, and it may not be legal to distribute them freely in a form that allows the recipient to use them for any

[18] Although as we are working with finite digital values, the actual number of variations is restricted.

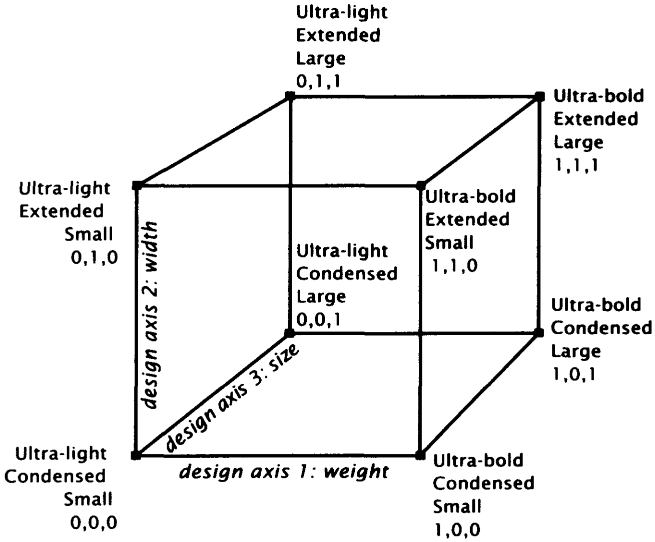


Figure 7.19
Multiple Master design axes

purpose other than to view a particular document. Ensuring that a distributed font can only be used for a single purpose is not always feasible. However, if fonts are not distributed with the document, font substitution will probably occur, destroying the document's carefully designed appearance. Designers wishing to use fonts other than the small number of rather dull fonts that can be found on most computer systems are left in a quandary: to distribute or not?

With Multiple Masters a new solution becomes possible. Instead of distributing the font, include a sufficient description of it to enable an instance of a Multiple Master typeface to be constructed that approximates its appearance and matches its spacing. That way, if a font must be substituted, the deleterious effect on the document's appearance is minimized. This form of font substitution is supported by the *Portable Document Format (PDF)*; the Acrobat Reader software that displays PDF files will instantiate fonts if necessary. A pair of general-purpose Multiple Master typefaces (one serifed, the other sans serif) is distributed with Acrobat Reader, so it is always possible for some approximation to the intended font to be produced. Other Multiple Master typefaces can be obtained, which have distinctive characteristics like a conventional font family.

Further Information

The layout of text is described in Chapter 8.

Appendix B of [GQ99] is a good account of character sets and related issues; [App93b] includes some interesting information about the diversity of writing systems used around the world.

[Bla92] is a history of fonts in the 20th century; [Cli94] and [HF97] concentrate on recent faces. [Ado90a] is the specification of the Type 1 font format, and [Ado97] is Adobe's guide to Multiple Master fonts.

Exercises

1. In Chapter 3, we described how a graphics application program keeps track of a model of an image, which it renders to make it visible. Discuss the extent to which a sequence of character codes can be considered a model of text, which has to be rendered as glyphs.

2. Naïve commentators sometimes claim that using Unicode or ISO 10646 is wasteful of space, when documents can be adequately represented in ASCII. Explain why this is not so.

3. Although Unicode can represent nearly a million different characters, keyboards cannot have that many keys. Discuss ways in which keyboards can be made to accommodate the full range of characters.

4. Look at a newspaper. Compare and contrast the way fonts are used in news stories and in advertisements.

5. Find out which fonts are available on your computer, and print samples of each. Which would you consider to be suitable for display purposes and which for text?

6. Which of the fonts available on your system would you use for each of the following jobs?

   (a) The opening credits of a science fiction movie.

   (b) Your *curriculum vitae.*

(c) The body copy of a community newsletter.

(d) A wedding invitation.

(e) The list of ingredients on the back of a jar of jam.

Explain each of your choices.

7. Is there a meaningful distinction to be made between fonts that are *readable* and those that are *legible*? If so, what is it?

8. Fonts designed to be used on low-resolution devices usually have a relatively high x-height. Why?

9. Design a font for Scrabble tiles. You only need the uppercase letters. You will have to ensure that your letter forms fit harmoniously and legibly into the square tile (and don't forget the little number for the letter's value). If you don't have access to font design software, such as Fontographer or Metafont, you should design the character shapes on squared paper in sufficient detail to be mapped to pixels; otherwise, use the software available to you to generate a font in a suitable format and use it to typeset some sample words.

Layout

8

In Chapter 7, we considered the storage and display of individual characters. When characters are combined into words, sentences and extended passages of text, we need to think about how they should be arranged on the screen. How are paragraphs to be broken into lines? How are headings, lists and other textual structures to be identified and laid out? How are different fonts to be used?

As well as being a vital component of multimedia, text can play an additional role by specifying the structure of a multimedia production. Layout commands and visual characteristics can all be specified using a text-based *markup language*. In later chapters we will show how links to other documents, the location of images and video clips, interaction with users, and synchronization of time-based media can also be specified textually in a markup language. The best known such language is HTML, which we will introduce in this chapter, together with the newer and more powerful XML.

Our primary focus is on markup and the way it is expressed in languages, not on the tools, such as word processors and desktop publishing packages, that present users with an environment for preparing text. Multimedia authoring systems usually include similar facilities for preparing text. It is, however, common practice to use a dedicated system to prepare text which is then imported into the authoring environment, already formatted. In either case, the formatting operations available are equivalent to what can be expressed using the languages described later in this chapter.

Text in Graphics

The maximum flexibility in the layout of text can be obtained by treating text as graphics and manipulating it with a graphics program. Text can then be placed at will, possibly using layers to overlay it, and treated with effects and transformations. The integration of text and graphics occurs naturally, so this approach to text is ideally suited to graphic design incorporating text, such as posters, packaging, company logos and letterheads, Web pages, book jackets, and CD covers. As always, there is a choice between vector and bit-mapped graphics, which determines how the text can be manipulated.

If text items are treated as objects in a vector graphics program, they can be arranged on the page arbitrarily, and all the effects that can be applied to other graphical objects can be applied to them.[1] Words and letters can be scaled, rotated, reflected and sheared. They can be stroked with colours, gradients, or even patterns. The leading drawing programs provide extra facilities for laying out text, which take account of its sequential nature. Tools allow you to fill areas with text, arranged either horizontally or vertically. (Vertical text is intended for languages that are normal written that way, but can be applied to horizontal languages for a special effect.) The text flows automatically to stay within the area, with line breaks being created automatically as required. Text can also be placed along a path.

Figure 8.1 shows some typical ways of treating text in a vector graphics program. The design down the left was made by setting the name Shakespeare vertically, using a mid-grey to colour the text, taking a copy and reflecting it about the vertical axis. The mirrored copy was then filled with a lighter shade of grey and displaced slightly vertically and horizontally. Glendower's speech is set on a Bézier path, while Hotspur's reply is centred within an oval. Again, slightly different shades have been used to distinguish the components. The attribution has been rotated; all the elements were scaled to fit their spaces. The text remains editable: spelling mistakes can be corrected — you can even use a spelling checker — and the fonts and type size can be changed. The shapes into which the text has flowed, and the paths along which it is placed can be changed using the normal vector editing tools, and the text will move to accommodate the changes. By treating text as bitmapped

[1] But you must use outline fonts.

graphics, quite different results can be obtained. Once text has been converted into pixels, it becomes susceptible to all the retouching and filtering that painting programs provide. (However, unless the text is set in a large font, many effects either do not show up, or obliterate it entirely.) Text can be warped and distorted, embossed and bevelled, set in burning metal, or weathered and distressed. Figure 8.2 provides some examples. Different elements of the design have been blurred and sharpened; the line 'This is the worst' has been copied onto several layers, displaced relative to each other to produce a 3-D effect, which is enhanced by the streaking that has been painted on. The text 'So long as we can say' is not placed on a path, as it would be in Illustrator, it has been distorted using Photoshop's polar coordinates filter. This text has been incorporated into a bitmapped image and can no longer be edited as text, but must be retouched like any other part of the image. Using the terminolgy from Chapter 7, text in a vector graphics program retains its lexical content, whereas in a bitmapped image it is reduced entirely to appearance.

Evidently, both approaches to text in graphics have their uses, and they can be profitably combined. For most purposes, vector-based text is more manageable; it has been common practice for designers to prepare text in Illustrator, then import it into Photoshop, where it is rasterized so that bitmapped effects can be applied. With the release of Photoshop 5.0, vector-based text was incorporated directly into Photoshop.Text on its own layer can be edited, as it can in Illustrator, although the full range of vector effects is not available. The text must still be rasterized before bitmapped effects can be used. The distinction between the two approaches to graphics and text remains: programs are increasingly combining the two, but the underlying representations remain distinct.

Graphics programs offer complete control over the appearance and placement of text, but this is not always what is required. Quite the contrary, in fact. If you are preparing some conventional text, such as a letter or an article, you want to be able to just type the words, and have a program do as much of the layout as possible for you. For this, we have to turn to a different class of programs, and consider some new questions, in particular, how do we exercise control over layout without having to place everything explicitly?

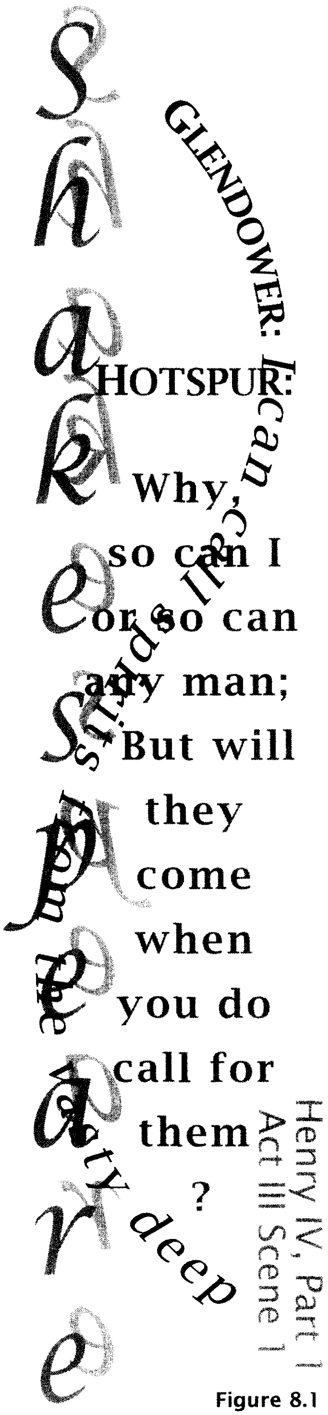


Figure 8.1
Vector text

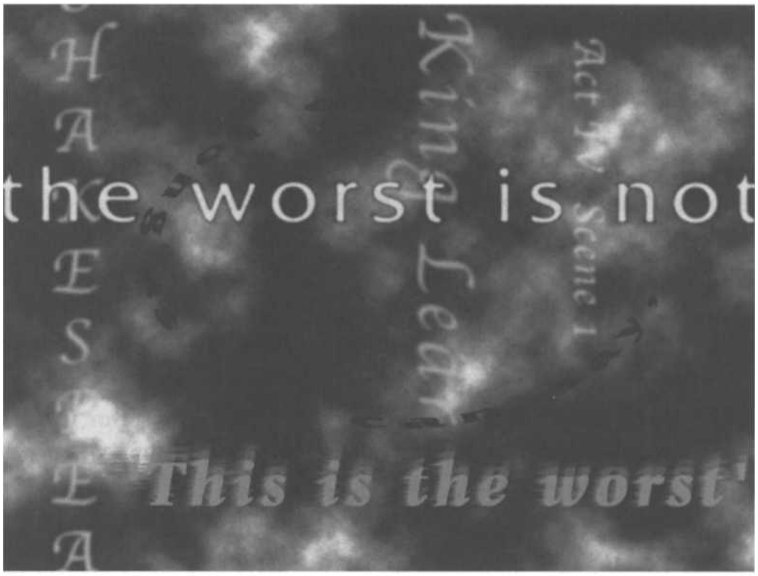


Figure 8.2
Bitmapped text

Markup

In the days before desktop publishing and word-processors, authors' manuscripts were usually produced on a typewriter. Consequently, they could not be laid out in exactly the same form as the final published book or paper. The author, the publisher's copy editor and possibly a book designer would annotate the manuscript, using a variety of different coloured pens and a vocabulary of special symbols, to indicate how the text should be formatted by the typesetter when it was printed. This process of annotation was called *marking up* and the instructions themselves were often referred to as *markup*. With the transition to computer-based methods of publication, the function of markup has been transferred to the instructions inserted (often by the author) into the text file that now corresponds to the manuscript. The form of these instructions is determined by the program being used to prepare the manuscript.

A distinction is often drawn between *WYSIWYG* formatting systems, and *tag-based* systems. WYSIWYG stands for 'What you see is what you get', a phrase which succinctly captures the essence of such systems: as you type, text appears on the screen laid out just as it will be when printed or displayed. Font and size changes, indentation, tabulation, and other layout features are controlled by menus, command keys or toolbar icons, and their effect is seen immediately. The markup, although present, is invisible; only its effect can be seen, giving the illusion that the formatting commands do not insert markup, but actually perform the formatting before your very eyes, as it were. In contrast, with a tag-based system, the document is normally prepared using a plain text editor; the text is interspersed with the markup, which takes the form of special layout commands, often known as *tags*. Tags are lexically distinguished from the text proper, by being enclosed in angle brackets or beginning with a backslash character, for example.[2] Tags do the same job as the commands in a WYSIWYG system, but their effect is not necessarily immediately visible. A separate processing phase is usually required, during which the formatting described by the tags is applied to the text, which is converted into a form that can be displayed or printed, such as PostScript. Word processors such as MS Word are invariably WYSIWYG, as are most commercial page layout tools, such as Quark Express and Adobe inDesign. Tag-based layout has, until recently, been largely confined to the academic and technical sectors, where tools such as `troff` and LaTeX have been in use since long before the phrase 'desktop publishing' was coined. In recent years, the popularity of HTML has brought tag-based text formatting to a wider audience.

Although the experience of preparing text using these two different types of system is very different, calling for different skills and appealing to different personalities, a moment's reflection will show you that their differences are mostly superficial and concern the interface much more than the way in which layout is actually controlled. Underneath any WYSIWYG system is a tag-based text formatter, although the tags are more often binary control codes than readable text tags, and are inaccessible to the user of the WYSIWYG editor. The rash of WYSIWYG HTML editors and the re-targeting of page layout packages to produce HTML demonstrate that there is no real incompatibility between using tags to control layout and using a WYSIWYG approach to document preparation.

We will concentrate on layout using textual tags, partly because it is easier to see what is going on, and partly because, after years in the shade of layout based on hidden control codes, it has acquired a new importance because of the World Wide Web. An advantage of using textual markup, instead of binary codes or some data structure, is that plain text can be read on any computer system and

[2] A sufficiently powerful editor, such as Emacs or Alpha, will allow you to insert tags with a single keystroke or menu selection.

can be transmitted unscathed over a network. This is particularly important for the Internet, with its heterogeneous architectures. Here, HTML has brought tagging into wide use. XML, and a clutch of languages developed from it, promise to extend the use of this sort of markup into other areas of multimedia.

The difference between tag-based and WYSIWYG layout is merely a cosmetic difference between interfaces. A more profound distinction is that between *visual* markup and *structural* markup. In visual markup, tags or commands are used to specify aspects of the appearance of the text, such as fonts and type sizes. In structural markup, tags identify logical elements of a document, such as headings, lists or tables, and the visual appearance of each type of element is specified separately.

The distinction can be illustrated using LaTeX, an example from a previous generation of tag-based markup languages, with which either approach can be used. For example, suppose we were using LaTeX to prepare a document which was divided into sections, each of which had a section heading. If we were taking a visual approach, we might decide that each section heading should be centred and set in large boldface type. This can be accomplished using commands such as the following for each section:

```
\begin{center}
\fontfamily{hlh}\fontseries{b}\fontshape{n}
\fontsize{14}{16}\selectfont Introduction
\end{center}
```

whose meaning should be fairly evident: LaTeX tags begin with a backslash, and some of them take arguments enclosed in curly brackets. Here, `\begin{center}` and `\end{center}` define the extent of an 'environment' within which text is centred. The next, more enigmatic, commands select the font: `hlh` identifies the Lucida Bright family,[3] `b` its boldface series and `n` its normal shape; next we stipulate a size of 14pt on a 16pt baseline, and finally select the font. A cut-and-paste editor, or one that supported keystroke macros, would make it relatively easy to ensure that every section was laid out in the same way.

However, if we were to adopt the structural approach to our markup, we would simply tag each section heading *as* a section heading:

```
\sectionheading{Introduction}
```

[3] LaTeX fonts are identified with apparently cryptic names because of the LaTeX community's decision to adopt a naming scheme that could be made consistent with the primitive file naming conventions of MS-DOS when font specifications are mapped to font file names.

Elsewhere, probably in a separate file, known as a *document style*, containing definitions for all the structural elements of a particular type of document, we would provide a definition of the \sectionheading command that specified the layout we wished to apply. For example, if we still wanted each section heading to be centred and set in large boldface type, we could define \sectionheading like this:

```
\newcommand{\sectionheading}[1]{
  \begin{center}
    \bfseries\Large #1
  \end{center}
}
```

Again, the meaning should not be too obscure: \newcommand defines a new command, whose name appears next in curly brackets. The [1] indicates that the command will take one argument, in this case the text of the section heading. The commands between the next pair of curly brackets form the body of the definition; they are substituted in place of \sectionheading whenever it is used. The \bfseries command chooses a boldface version of the current font and \Large switches to a larger point size. (These two commands are themselves abstractions of the primitive font selection commands we used previously. Both of them change the font characteristics relative to the body font of the document, which would be set globally elsewhere.) Within the body, #1 will be replaced by the argument when the command is used, so \sectionheading{Introduction} will lead to #1 being replaced by Introduction. The net effect will be to centre the heading in large boldface type, as it was previously.

Structural markup has distinct advantages over visual markup. Perhaps most obviously, it allows us to change the appearance of a document globally by changing the definitions of markup tags just once. If, for example, we wanted to use a more imaginative layout for our section headings — right-aligned in uppercase boldface italics, say — we would only have to change the definition of \sectionheading — for example, to:

```
\newcommand{\sectionheading}[1]{
  \begin{flushright}
    \bfseries\itshape\Large \uppercase{#1}
  \end{flushright}
}
```

and every section would be re-formatted in the new style. Consistency is ensured. In contrast, if we had marked up each section heading individually, we would have had to find and change the markup of each one.

The ease with which structural markup allows layout to be changed offers more than a licence to experiment with different typographical effects at whim. A more substantial advantage is the easy localization of documents which it permits. Centred layouts, for example, which are fairly standard in the United States, are much less popular in Europe. Changing the layout of a document such as a software manual to make it look more agreeable to customers in different parts of the world can be achieved by redefining the heading commands, while leaving the marked up text of the manual alone. Similarly, while an author is working on a manuscript, it might be desirable to preview it using a simple layout, perhaps double-spaced. By using a different set of definitions for the tags, the same markup can be used to produce a final version, laid out in a more professional way.

Even better, a sufficiently powerful system based on structural markup would allow you to redefine the effect of your markup tags for different output media. For example, if you wished to produce a version of your document to be 'read' to blind people by a speech synthesizer, you might redefine a section heading so that the words of the heading were read with increased emphasis.[4] More generally, since the structural tags themselves do not specify any form of processing, they can be used as the basis for presentation of the document in any medium, including media not invented when the document was written.

A related advantage of structural markup is that it permits a separation of concerns, and a division of labour, between the appearance of a document and its structure. The writer need only concentrate on identifying the logical divisions within the document while he or she is writing. The question of what typographical features should be used to identify those divisions in a perspicuous and pleasing way can be dealt with separately, ideally by a specialist book designer.[5] If visual markup is used, the processes of writing and layout become intertwined, and control over appearance is left to the whim of the author.

> Programmers will recognize that visual markup is an *abstraction* mechanism, similar to the use of functions and classes in programming. The separation of appearance from markup resembles the separation of a class's interface from its implementation.

[4] This has actually been done for LaTeX. See *An Audio View of LaTeX Documents - Part II* by T.V. Raman, in TUGboat, 16,3, September 1995.

[5] This is the situation that prevailed in the publishing industry prior to the arrival of 'desktop publishing'.

The final advantage of structural markup is that tags identifying the structural elements make it easy for a computer program to analyse the structure of the document. Although people can identify the structural elements from their appearance, this depends on knowledge and understanding, things which it is notoriously difficult to incorporate into computer programs. Programs are extremely good at searching text for strings with a specific form, such as LaTeX commands, though. This makes it easy to write programs to extract tables of contents or other tagged pieces of information. As a simple example, the Alpha editor is able to construct a pop-up menu of all the section headings and sub-headings in a LaTeX document — provided they are marked up using the `\section` and `\subsection` commands, and not an *ad hoc* sequence of visual markup — so that, when a section name is selected from the menu, the insertion point is moved to the start of that section. In a more complex environment, structural tags make it possible to display an outline of a document, at varying levels of detail, or to allow readers to search for strings only when they are used in a particular way, as part of a heading or a figure caption, for example.

We must emphasize that, although we have used LaTeX in our examples, and will continue to use tag-based markup in the remainder of this chapter, structural markup can be applied using modern WYSIWYG word-processors and page layout programs. The mechanisms and their names vary, but macros, stylesheets, and master pages all offer the possibility of separating the appearance of a document from its structure.

SGML

In the preceding section we gave examples of both structural and visual markup using LaTeX. If you are familiar with LaTeX, you will know that it was designed specifically as a structural markup system built on top of the low-level formatting features provided by the powerful typesetting system TeX. Despite this, in LaTeX we could choose visual formatting, because the same language is used for markup of the text and for defining new commands. To perform the latter function it must provide facilities for visual formatting, and there is nothing to stop you using these visual facilities to mark up your text.

There is much to be said for providing computer users with the choice between different ways of working, but many of the benefits of structural markup can only be obtained if it is used consistently and exclusively, and the only way to guarantee this happening is by completely separating markup from visual layout. The easiest way to enforce this separation is via a language that provides nothing but facilities for structural markup. This philosophy finds its purest embodiment in *SGML (The Standard Generalized Markup Language).*

SGML is arguably mis-named, since, properly speaking, it is not a markup language, but a markup *metalanguage*. That is, it is a language for defining markup languages; it allows you to define your own tags, each of which is used to mark a particular sort of *document element*, such as a paragraph, section heading, or list. In SGML, defining a tag consists of giving it a name and some attributes, and specifying some restrictions on the context in which the tag may be used and *nothing more.* This is in contrast to LaTeX and similar systems, where the only way to define a new tag is as a command, whose definition specifies the layout to be applied when it is used, so that a LaTeX document class defines the structure *and* the appearance of a class of documents. The SGML equivalent, known as a *Document Type Definition (DTD)* defines only the structure. In other words, a DTD describes all the documents of a particular type, in terms of the tags that may be used to mark them up. A document marked up with SGML tags from a DTD thus contains a description of its own structure. Any program that needs to understand the document's structure can deduce it by reading the tags.

> If you are familiar with programming language definitions, you may find it helpful to consider a DTD as being like an attribute grammar defining a class of documents, in a similar way to that in which a grammar defines the syntax of a class of programs — and thereby defines a language.

We are used to thinking of a document as a physical object, and of its content as what appears on the paper, but SGML's notion of a document goes beyond the limitations of physical media, to encompass more of the potential of computer-based storage. A document may include elements which are not intended to be displayed, but which describe its contents in some way. For example, a document may include a tag which indicates the language in which its text content is written. This information is largely redundant for display — it will soon become evident to the reader — but might be valuable to a document retrieval system, allowing it to select only documents in a language that could be understood by the person doing the retrieving. The SGML tag would distinguish the language information from the other elements of the document. SGML can thus be used not just for traditional text markup, but for marking up any sort of data as a self-describing document. Remember, though, that in order for this to be possible, a DTD defining the class of documents in question must be available: in the example just cited, the language tag can only be included in documents based on a DTD which includes it.

From a structural markup perspective, displaying is only one of the things that might be done to a document, but from our multimedia perspective, it is the most natural, so we must consider mechanisms for controlling the layout of a structurally marked-up document. SGML provides no such mechanism, because of its determination to separate the description of a document's structure from its appearance. But, precisely because of this, it permits the use of *any* mechanism based on structural information. Several approaches are possible.

If a DTD describes a sufficiently large class of documents, it may make sense to construct display software specifically for that class of documents, and to hardwire the layout rules into the software. That way, every document will be laid out consistently by a particular display program. A variation on this approach is to build facilities into the display program allowing its user to control some aspects of the layout. This is the approach adopted by the first generations of Web browsers[6] (and remains the default in more recent ones). Most of the formatting of headers, lists, tables and other elements is controlled by the browser, but some details, specifically fonts, type sizes and colours, may be controlled by preferences set by the user. This is quite a radical departure from traditional publishing, since it gives the reader control over some aspects of the appearance of the document.

The alternative is to provide a separate specification of the layout, parallel to and complementing the DTD. Such a layout specification is usually called a *stylesheet*. For each tag defined in the DTD, a stylesheet provides a rule describing the way in which elements with that tag should be laid out. There may be more than one stylesheet for a particular DTD, providing a different appearance to the same structure, as, for example, one corresponding to typographical styles favoured in the US, and another corresponding to those in Western Europe. Stylesheets are written using a *stylesheet language*, which is designed for the purpose.

[6] Web pages may be described by an SGML DTD — it's called HTML.

Ideally, the use of stylesheets in conjunction with SGML leads to scenarios such as the following. A technical publisher devises a DTD for books, defining tags for each element that might occur in a technical book, such as chapters, sections, paragraphs, quotations, tables, computer program listings, footnotes, and so on, hopefully exhaustively. Authors mark up their manuscripts with the tags defined in this DTD. For each book, or series, a book designer constructs a stylesheet, which has the effect of laying out the book in a style appropriate to its contents and intended market, taking account of any budget constraints and house style rules which might affect the layout.

This ideal scenario is not always achieved, however, not least because desktop publishing software has accustomed authors to having control over the appearance of their work. In practice, SGML has been used in the publishing industry mainly as an archiving format for electronic versions of printed material, since its independence from any formatting mechanism or technology means that the markup will in principle remain valid forever and can be used as the basis for publication at any time in the future, using whatever technology has become available. Furthermore, the markup can be used for indexing and document retrieval, thereby making the archived material readily accessible. However, SGML never remotely approached the status of a universal markup language, despite being enshrined in an international standard.[7]

One reason for this was the lack of any perceived need for structural markup. This perception changed with the increased use of heterogeneous computer networks, such as Arpanet and subsequently the Internet. Since one can make no assumptions about what hardware or software might be on the other end of a network connection, visual markup that is tied to one piece of software or depends on certain hardware facilities for correct display cannot be relied upon to be intelligible or effective. Since structural markup can be interpreted on any platform in a logically consistent — if not necessarily visually consistent — manner, it is ideally suited for the markup of documents which are to be transmitted over networks.

⇨ At one time, the only type of text that could be reliably transmitted over wide area networks and be guaranteed to be readable on any computer was plain ASCII, a situation that led to many important documents, such as Internet RFCs, being stored on archives with no more formatting than could have been achieved on a typewriter.

[7] ISO 8879:1986(E). *Information Processing — Text and Office Systems — Standard Generalized Markup Language (SGML).*

When the World Wide Web was being designed, therefore, the structural approach was adopted for the markup of Web pages. SGML was used as the basis of the *Hypertext Markup Language (HTML)* devised for this purpose. Unlike SGML, HTML is only a markup language: it provides a set of tags suitable for marking up Web pages, but does not provide any way of defining new tags. In fact, HTML is defined by an SGML DTD,[8] which describes the structure of the class of documents known as Web pages. Like SGML, it is concerned solely with identifying the structural elements of a page, and not with their appearance.

Originally, the World Wide Web was intended as a means of dissemination of scientific research, so the DTD for Web pages contained tags corresponding to the main elements of a scientific paper: a title, headings that can be nested several levels deep to introduce sections, subsections, subsubsections and so on, and lists of various types such as enumerations and bullet points. Additionally, HTML provides some typographical control, by allowing you to mark text as italicized or bold, although it does not allow you to specify particular fonts;[9] HTML also provides generic styling tags, to mark text as, for example, emphasized or strong, without specifying anything about the typographical rendering of such text — these tags are more in the spirit of structural markup than are the explicit typographical tags. Later revisions of HTML added tables and interactive forms to its repertoire. As well as these layout tags, HTML crucially includes tags for hypertext linkage and for inclusion of images and other media elements, as we will describe in later chapters.

XML

Although they are adequate for marking up the simple, mainly textual, papers for which they were originally intended, HTML's layout tags are by no means sufficient for all the diverse types of material that have found their way onto the World Wide Web, particularly since its commercialization during the middle of the 1990s. During that period two undesirable responses to HTML's deficiencies were made by browser manufacturers and Web page designers. The first response, from the manufacturers, was the addition of *ad hoc* proprietary extensions to HTML. Usually such additions were only supported by one browser, thus destroying the platform-independence of HTML markup, and often they were not

[8] The original version of HTML was not, but all versions subsequent to HTML 2.0 have been.

[9] Font selection was introduced in HTML 3.2, having been provided as an extension in popular browsers previously, but is deprecated in HTML 4.0 in favour of stylesheets.

in the spirit of structural markup.[10] The second response, from designers, was to put the available tags to purposes for which they were never intended. The most widespread form of HTML abuse — ensconced in more than one HTML authoring program and encouraged by any number of HTML tutorials — is the use of tables for grid-based layout. This trick enables designers to use established design ideas from traditional media to produce more interesting and readable page layouts than the simple sequence of paragraphs that HTML provides.

Another trick that has been widely adopted is to use graphics instead of text for parts of a document that cannot be satisfactorily formatted with HTML. Mathematical equations provide an example: there are no facilities in HTML for typesetting mathematics, so the only satisfactory way to incorporate equations and formulae into a Web page is to typeset them using some system, such as LaTeX, that can handle them, and then transform the typeset result into a GIF image that is pasted into the Web page. The same approach has been used by designers who are deeply concerned about the appearance of their pages to make sure that text is set in exactly the font they envisaged, and laid out just as they wished it to be.

Even if you believe that designers should live with HTML layout when they are working on the Web, there are many cases where the information they have to present simply does not fit in to an HTML document. Consider, for example, a restaurant wishing to post its menu on a Web page. What tags should be used for each of the dishes on offer? Probably, using a level 2 or 3 header would produce appropriate formatting, but dishes from a menu are not headings, any more than blocks of text laid out on a grid are entries in a table, or mathematical equations are illustrations. If structural markup does not describe the structure of a document, but rather another structure that happens to look like what the designer had in mind as the visual appearance of the document, then much of the point of structural markup is lost. In particular, automatic searching and indexing programs cannot correctly identify the elements of the document. For example, it would be hard to write a program to search for restaurants that served chocolate mousse unless it was possible to distinguish occurrences of the phrase 'chocolate mousse' that announced its presence on a menu from all others, such as headings in on-line recipe collections or studies on nutrition.

[10] The most blatant example is perhaps the <BLINK> tag.

Since HTML's tags are not sufficient for the class of Web pages as it has evolved, new tags are required. Any attempt to add tags to HTML to accommodate absolutely any sort document that someone might one day want to make into a Web page is doomed to failure, and would leave HTML bloated and unmanageable. A much better solution is to provide Web designers with a facility for defining their own tags. This is where we came in: SGML has just that facility. SGML is not, however, entirely suitable for use over the Internet; certain of its features make parsing it too difficult to be done efficiently enough to provide the required response times. Work on adapting SGML to the Internet led to the definition of a subset, known as *XML (eXtensible Markup Language)*, which provides all the important facilities of SGML without the overhead imposed by full SGML. In particular, XML allows Web designers to define their own DTDs for any type of document. Web pages are thus freed from the limitations of HTML's definition of a document.

XML 1.0 was adopted as a W^{3}C Recommendation early in 1998. Because of its complexity compared with HTML, it is unlikely that XML will be used directly to mark up individual Web pages by many designers. It is more likely that XML will be used indirectly: experts will produce XML DTDs that define new markup languages for specific tasks. Although this will lead to a proliferation of special-purpose languages, they will all share some syntactic features, such as the form of tags, and will all be based on the underlying technology of XML, so that it will not be necessary to add interpreters for each new language to every browser. So long as it can interpret XML DTDs, a browser can interpret pages marked up in any language defined by one. XML can thus function as an infrastructure supporting a wide variety of mechanisms for organizing text and multimedia content. There is no reason for it to be restricted to the World Wide Web in this rôle; XML can function equally well with other delivery media.

The process of defining new languages as XML DTDs began before XML 1.0 was even adopted as a Recommendation. Among the DTDs produced are *SMIL (Synchronized Multimedia Integration Language)*, *PGML (Precision Graphics Markup Language)*,[11] *MathML (Math Markup Language)* and *XFDL (eXtensible Forms Description Language)*.[12] A somewhat different application of XML is as a concrete syntax for *RDF (Resource Description Framework)*, which is intended to provide a standard for *metadata* — data about a document, such as its language, or labels describing its content for the benefit of filtering services.

[11] Now superseded by SVG, also defined by an XML DTD.

[12] HTML is defined by an *SGML* DTD, not an XML one, since it uses some of the features of SGML which were dropped in the XML subset. Work is proceeding on an XML DTD for HTML, which will bring it into line with these other markup languages.

Although we have introduced the idea of structural markup in the context of text layout, the ideas behind it may be applied to other media and to their integration into multimedia. We will show in later chapters how XML is used in some of the languages just listed to manipulate richer media than pure text. HTML, too, has features for combining other media with text, and these will be introduced when we come to consider multimedia proper.

Text Layout Using HTML and CSS

Our main interest in describing HTML is to introduce you to the syntax of markup in languages of the SGML family, and to illustrate structural markup. Similarly, with CSS we intend to show how stylesheets can provide sophisticated layout control to complement the markup. We assume that most readers will have some knowledge of HTML, but will not necessarily be acquainted with CSS. Those readers needing additional detailed information should consult the suggestions for further reading at the end of this chapter.

HTML is a language that has evolved — not always in an entirely controlled manner — and its design goals have changed over the years. As a result it displays inconsistencies, especially between its current structural markup ethos and left-over visual markup features from an earlier phase of its evolution. Thus, whereas some tags are little more than hooks on which to hang stylesheet-based formatting, others provide direct control over typography. Because of the need for browsers to be able to display old Web pages, HTML must provide backwards compatibility with its older versions, so this mixture of features will persist. Looking at this situation in a positive light, HTML will continue to support a plurality of approaches to markup.

⇨ The features of HTML described in this book correspond to the HTML 4.0 specification, adopted as a W^{3}C Recommendation in December 1997. Some of them are not supported by earlier versions of HTML. Similarly, we include some features from the CSS2 Recommendation of May 1998 which are not present in CSS1.

Although we have referred to Web pages being displayed by a Web browser, such as Netscape Navigator or Internet Explorer, the structural markup of HTML can also be interpreted by other programs, such as a text-to-speech converter. In the following pages, we will adopt the name *user agent*, used in the HTML specification, to mean any program that interprets HTML markup. A user agent, such as a conventional Web browser, that formats HTML documents and displays them on a monitor, is an example of a *visual user agent*; a text-to-speech converter is an example of a non-visual user agent.

Elements, Tags, Attributes and Rules

HTML markup divides a document into *elements*, corresponding to its logical divisions, such as sections and paragraphs. In general, elements may contain other elements: a section typically contains several paragraphs, for example, and this containment is reflected in the markup. Each element is introduced by a *start tag* and ends with an *end tag*; between these two tags is the element's *content*, which, as we just observed, may include other elements with their own start and end tags. Elements must be properly nested, that is, if an element starts inside another, it must end inside it, too. This means that each element forms part of the content of a unique *parent* element that immediately encloses it.

Every type of element available in HTML has a name. Its start tag consists of the name enclosed in angle brackets, as, for example, <P>, which is the start tag of the P element, used for paragraphs. The end tag is similar, except that the element's name is preceded by a slash: the end tag for paragraphs is </P>. So every paragraph in an HTML document has the form:[13]

```
<P>
   content of the paragraph element
</P>
```

All other elements have the same form — start tag, content, end tag — but with their own identifying tags. The case in which an element name is written in a tag is ignored: <P> and <p> are both the same tag and begin a P element. The case in an end tag may be different from that in the start tag it is paired with.

Some other lexical details have to be considered. Prior to the HTML 4.0 specification, ISO 8859-1 was the designated character set for HTML. In HTML 4.0, the character set is ISO 10646; no encoding is specified, you are allowed to choose your own, provided you specify it with the appropriate MIME type, and the HTML

[13] But see page 238.

specification defines 'encoding' liberally, to encompass subsetting. Hence, most HTML documents continue to be written in the ISO 8859-1 subset of Unicode. As true Unicode support begins to be provided by user agents, this situation should change.

An inevitable problem arises if tags are written in the same character set as the text proper: we must lexically distinguish tags from text, using some special characters (in HTML the angle brackets that surround the element name in a tag) so we need some mechanism for representing those special characters when they appear in the text as themselves. In HTML, a < symbol can be represented by the *character entity reference* &lt; (the terminating semi-colon is part of the entity reference). This now leaves us with the problem of representing an & so as not to cause confusion. Again, a character entity reference is used; this time it is &amp;. Strictly speaking, there should be no trouble using > symbols in text — there is always sufficient context to determine whether or not it is part of a tag — but some user agents do get confused by them, so it is advisable to use &gt; except at the end of a tag.

⇨ As you may have guessed, character entity references provide a general mechanism for inserting characters that are hard to type or are not available in the chosen character encoding. References are available for all the characters in the top half of ISO 8859-1, as well as for mathematical symbols and Greek letters, and for layout characters, such as an em-dash. Some examples are given in Table 8.1; for the full list, consult section 24 of the HTML 4.0 specification. There are many Unicode characters without character entity references. For these you can use a *numeric character reference*, which specifies a character using its ISO 10646 character code. A character with code *D* is written as &#*D*;, so, for example, &#60; is an alternative to &lt;. (If you are more comfortable with base 16 numbers, you can write your numeric entity references in hexadecimal, putting an x after the # to indicate you are doing so. Another alternative to &lt; is &#x3c.[14])

Table 8.1
A few examples of character entity references

&iexcl;	¡
&pound;	£
&sect;	§
&Aring;	Å
&aring;	å
&piv;	ϖ
&forall;	∀
&exist;	∃
&isin;	∈
&ni;	∋
&mdash;	—
&dagger;	†
&Dagger;	‡

Whitespace consists of space, tab and formfeed characters, and line breaks, which may be of the form used on any of the major platforms: carriage return, linefeed or both. In text, whitespace separates words in the usual way. Visual user agents will format the text in accordance with the conventions of the writing system being used; in English this will usually mean that runs of spaces are replaced by a single space. (In PRE elements, described later, whitespace is left exactly as it is typed.) Whitespace immediately following a start tag or preceding an end tag is ignored, so that the

[14] Or &#X3C — case is not significant.

following two examples will be displayed identically.

```
<P>
Shall I compare thee to a summer's day?
</P>
<P>Shall I compare thee to a summer's day?</P>
```

An HTML document may be annotated with *comments*. These are introduced by the character sequence <!-- and terminated by -->. For example,

```
<!-- This section was revised on
     21st September 1998
-->
```

As the example shows, comments may occupy more than one line. Comments are not displayed with the rest of an HTML document. They are intended for information about the document, for the benefit of its author or anyone who needs to read the marked up source. However, as we will see later, comments have also been appropriated for the purpose of maintaining compatibility between current and older versions of HTML, by providing a mechanism for hiding content that older browsers cannot deal with.

⇨ As if the syntax for comments was not odd enough already, HTML allows you to leave space between the -- and the closing > (but not between the opening <! and its --). As a result, HTML comments are very difficult to parse correctly, and most browsers don't really try. Hence, it is very unwise to include strings of hyphens within an HTML comment, and you cannot rely on comments nesting correctly.

In HTML you are allowed to omit end tags where the context makes it clear that they should be there. For example, most documents will consist of a series of paragraphs, which should be of the form:

```
<P>
  content of the first paragraph
</P>
<P>
  content of the second paragraph
</P>
<P>
  content of the third paragraph
</P>
```

and so on. Even a computer program can tell that if you are starting a new paragraph, the previous one must have ended, so the </P> tags can be left out:

```
<P>
   content of the first paragraph
<P>
   content of the second paragraph
<P>
   content of the third paragraph
```

Any good book on HTML will tell you which end tags can be omitted and when. There is no particular merit in doing so, unless you actually type all the tags yourself instead of using an intelligent editor or HTML authoring application. In our examples we will show all end tags.[15]

There is one case in which end tags *must* be omitted, and that is the case of *empty elements*. These elements have no content. An example is the HR element, which produces a horizontal rule (line). There is no sensible content that can be supplied to such an element — it just *is* a horizontal rule — so there is nothing for an end tag to do and you must leave it out.

Although it does not have any content, a horizontal rule element has some properties, including its width and colour. In HTML[16] such properties are called *attributes*. The attributes associated with a particular element are specified as part of its definition in the DTD. Values for each attribute may be assigned within the start tag of an element. For example, to set the length of a rule to one half of the width of the displayed page and its width to 12pt, we would use the following tag:

```
<HR size = "12" width = "50%">
```

The case of attribute names is ignored; we use lower case to emphasize the distinction from element names. Values are assigned as shown using an = sign. Any number of assignments may occur inside the tag, separated by spaces. Attribute values are written enclosed in either single or double quotes. (If you need a double quote inside an attribute value enclosed in them, use the character entity reference &quot;.) The treatment of Boolean attributes (which may take the value true or false) is singular: there are no Boolean constants, and a Boolean attribute is set to true by assigning its own name to it, as in:

[15] The omission of end tags is an example of *markup minimization*, a feature of SGML that has been carried over into HTML but is not present in XML. This is one of the reasons that HTML cannot easily be defined by an XML DTD.

[16] And SGML and XML.

```
<HR size = "12" width = "50%" noshade = "noshade">
```

(The noshade attribute, when true, causes the rule to be displayed as a solid block, and not as the pseudo-3D groove that is normally used.) The notation for setting Booleans is clumsy, so a contracted form is allowed, where the mere presence of the attribute is sufficient to set it to true. The previous example would more normally be written:

```
<HR size = "12" width = "50%" noshade>
```

Booleans are set to false by omitting them from the tag, as in the first 12pt rule example.[17]

HTML experts might cringe at the preceding examples, because all the attributes to HR that we have used are *deprecated* in HTML 4.0. Actually, most of the attributes of simple text elements are deprecated, because they control some aspect of appearance. This can now be controlled much better using stylesheets. In principle, since stylesheet information is separate from markup, any stylesheet language can be used with HTML. In practice, most Web browsers only support *Cascading Style Sheets (CSS)*, a simple stylesheet language which works well with HTML, and can be easily mastered.

Let's return to paragraphs to demonstrate how a stylesheet can be used to control layout. For now, do not worry about how the stylesheet and HTML document are combined.

A paragraph element is a logical division of the text of a document. In the absence of any stylesheet, a user agent is free to express this division in whatever way it chooses. In English text, several different conventions are employed for laying out paragraphs. The visually simplest relies only on the insertion of additional vertical space between paragraphs; commonly this visual cue is augmented or replaced by applying an indentation to the first line of each paragraph. Most, if not all, visual user agents use the simplest convention by default. Suppose we want to use the more elaborate form, with indentation.

CSS allows us to specify various visual properties of each document element. One of these properties is, luckily for our example, the indentation of the first line. We specify that the first line of each paragraph should be indented by 4pc with a CSS rule like this:

[17] User agents commonly recognize the values true and false, although these are not part of the HTML 4.0 standard.

```
P {
    text-indent: 4pc;
}
```

The rule has two parts: a *selector* (here P) which indicates which elements the rule applies to, and some *declarations* (here there is only one) which provide values for some visual properties. Here the declaration specifies a `text-indent` (the name of the property controlling the amount of indentation applied to the first line) of 4pc. The declaration part of a rule is enclosed in curly brackets, as you can see.[18]

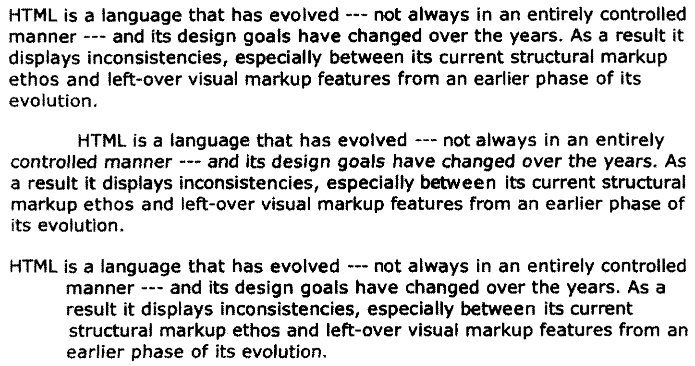
Whenever the rule just given is in force, every paragraph will be displayed with its first line indented by any user agent that implements CSS stylesheets (and is configured to do so). Suppose now that we did not wish to apply this indentation to every paragraph. Many manuals of style suggest that the first paragraph of a section should not be indented, for example. In order to be more selective about which paragraphs a rule applies to, we need some way of distinguishing between different classes of paragraph, such as those that are indented and those that are not. The HTML attribute `class` can be used for this purpose. This attribute is a property of virtually every HTML element; its value is a distinguishing name that identifies a subset of that element. For example, we might use a class `noindent` for paragraphs we wished to be displayed with no indentation.

In a CSS rule, a selector can consist of an element name followed by a dot and a class name. To specify that paragraphs of class `noindent` should not be indented, we would add the following rule to the previous one:

```
P.noindent {
    text-indent: 0pc;
}
```

As you would probably expect, when there is a general rule for some element, like our first example, and a more specific rule for some class of that element, like our second example, the more specific rule is applied to elements of that class, and the general rule is only applied to elements belonging to no class, or to some class for which no specific rule is available. If, for example, we has some paragraphs of class `unindent`, they would be displayed with a 4pc first line indent, since the only rule that applies to them is the general one with selector P.

[18] If you have ever written an Awk script, this syntax should look familiar.



HTML is a language that has evolved --- not always in an entirely controlled manner --- and its design goals have changed over the years. As a result it displays inconsistencies, especially between its current structural markup ethos and left-over visual markup features from an earlier phase of its evolution.

HTML is a language that has evolved --- not always in an entirely controlled manner --- and its design goals have changed over the years. As a result it displays inconsistencies, especially between its current structural markup ethos and left-over visual markup features from an earlier phase of its evolution.

HTML is a language that has evolved --- not always in an entirely controlled manner --- and its design goals have changed over the years. As a result it displays inconsistencies, especially between its current structural markup ethos and left-over visual markup features from an earlier phase of its evolution.

Figure 8.3
Browser display of paragraphs with different indents

Rules can be used to control a range of properties; even a simple rule with only a couple of declarations can produce effects that are not usually available in HTML. For example, the following rule causes paragraphs to be displayed with a hanging indent.

```
P.hang {
    text-indent: -40pt;
    margin-left: 4pc;
}
```

The effect of these three rules is shown in Figure 8.3, which shows the display of the HTML in Figure 8.4 by a user agent that understands stylesheets.

Figure 8.4 also illustrates some features of HTML that have not been described yet. The very first line is a *document type declaration*, which identifies the DTD defining the version of HTML being used. The enigmatic form of the DTD specification should just be treated as red tape; the declaration is not actually needed by current browsers, but it can be used by verification programs to check the legitimacy of the HTML markup. Next comes the start tag for the HTML element. This is the root of the entire document structure; all the elements of the document proper are contained within it. There are only two document elements that can come immediately inside the HTML element: HEAD followed by BODY. The head of the document contains information about the document, which is not displayed within it; the body contains the real text.

There are three elements contained in the head of the document in Figure 8.4. The first META element should also be treated as red tape: it specifies that the character set is ISO 8859-1. This is the default

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
                          Transitional//EN">

<HTML>
<HEAD>

<META http-equiv="content-type"
      content="text/html;charset=iso-8859-1">
<TITLE>Paragraphs</TITLE>

<STYLE type="text/css">
<!-- /* Hide content from old browsers */
P {
text-indent: 4pc;
}
P.noindent {
text-indent: 0pc;
}
P.hang {
text-indent: -40pt;
margin-left: 4pc;
}
/* end hiding content from old browsers */ -->
</STYLE>
/HEAD>

<BODY>
<P class="noindent">
HTML is a language that has evolved
--- not always in an entirely controlled manner --- and
its design goals have changed over the years.  As a
result it displays inconsistencies, especially between
its current structural markup ethos and left-over
visual markup features from an earlier phase of its
evolution.
</P>
<P class="unindent">
HTML is a language ...
</P>
<P class="hang">
HTML is a language ....
</P>
</BODY>
</HTML>
```

Figure 8.4
HTML source for Figure 8.3

usually applied, but it is good practice to specify the character set explicitly in this way. If you do so, the META element should be the first thing in the HEAD (for reasons that are hopefully obvious).[19] The TITLE element contains a short title, which is often displayed in the title bar of the browser's window. It is not displayed within the window, because it is not part of the text contained in the body.

The next element in the HEAD is a STYLE, which is where the stylesheet rules governing the layout of this page are to be found. (This is not the only place they can be, we will see an alternative later.) The start tag has an attribute `type` whose value is the MIME type of the stylesheet. For CSS rules, this is always `text/css`. The next line is a trick we will see more than once. The usual reaction of a Web browser to tags it does not understand is to ignore them completely: the tags are skipped, so the content would be displayed as if it were text. This is sensible behaviour much of the time, but quite inappropriate for stylesheets. The syntax of CSS allows an entire set of rules to be enclosed in an SGML/HTML comment, which has the effect of hiding it from browsers that do not recognise the <STYLE> tag. The remarks on the same lines as the HTML comment tags are CSS comments, which indicate what is going on.[20]

The body of the document contains the paragraphs themselves, with their class attributes. The style in the head is applied to them to produce the desired display.

To summarize: text in the body of an HTML document is marked up with tags that delineate the document elements corresponding to the logical divisions of the text. Stylesheet rules, which may be put in a STYLE element in the document's head, control the layout of these elements; the selector of each rule determines which elements it applies to, and the corresponding declarations set properties that control the appearance on screen. The `class` attribute is used to distinguish between different subsets of an element type, so that finer control can be exerted over their layout.

We are now in a position to describe the HTML tags that can be used to mark up text, and the CSS properties that can be used to control its layout. Although we are describing these particular languages, you should appreciate that the underlying principles of markup and layout apply to any system of text preparation.

Please note that the account we are about to give is not exhaustive. Some additional details will be provided later — in particular, we will look at facilities for incorporating other media besides text

[19] If your chosen character set does not provide ASCII as a subset you have got problems.

[20] HTML editors and authoring programs often insert these comments automatically.

in later chapters — but we are not attempting to give a definitive reference guide to or tutorial on HTML and CSS. For that, consult the specifications or suggestions for further reading at the end of the chapter.

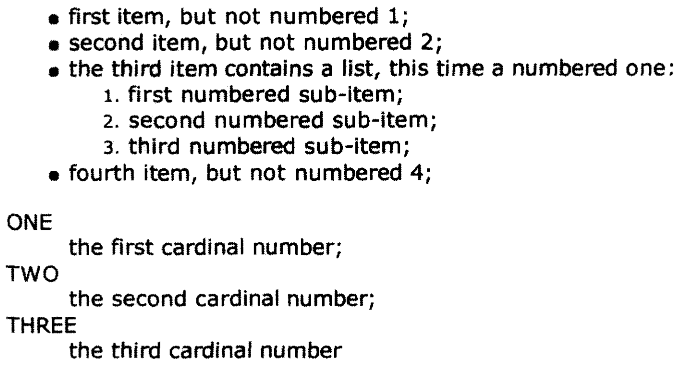
HTML Elements and Attributes

The HTML 4.0 specification defines 91 elements, of which 10 are deprecated, meaning that they may be removed from future revisions of HTML, since there are now preferred ways of achieving the same effect. Only a few of these elements are concerned purely with text layout. Those that are can conveniently be divided into *block-level* and *inline* elements. Block-level elements are those, such as paragraphs, that are normally formatted as discrete blocks; i.e. their start and end are marked by line breaks. Inline elements do not cause such breaks; they are run in to the surrounding text.

The most frequently used block-level textual element is the paragraph (P) element, which we have looked at already. Other block-level elements concerned purely with text layout include level 1 to level 6 headers, with element names H1, H2...H6; BR, which causes a line break; and HR, the horizontal rule element which we saw previously. The BLOCKQUOTE element is used for long quotations, which are normally displayed as indented paragraphs. Note, though, that using BLOCKQUOTE as a way of producing an indented paragraph is an example of the sort of structural markup abuse that should be avoided: markup is not intended to control layout. This being so, you will appreciate that the PRE element, which is used for 'pre-formatted' text and causes its content to be displayed exactly as it is laid out, is something of an anomaly, yet it is useful for cases where the other available elements do not serve, and elaborate stylesheet formatting is not worthwhile.

The only elaborate structures that HTML supports as block-level elements are lists and tables. Tables are relatively complex constructions (as they inevitably must be to accommodate the range of layouts commonly used for tabulation); since their use is somewhat specialized we omit any detailed description. Lists, in contrast, are quite simple. HTML provides three types:[21] ordered lists, in the form of OL elements, unordered lists, UL elements, and definition lists, DL elements. OL and UL elements both contain a sequence of list items (LI elements), which are laid out appropriately, usually as separate blocks with hanging indentation. The difference is that

[21] Two other special-purpose lists are also available, but both are deprecated.



- first item, but not numbered 1;
- second item, but not numbered 2;
- the third item contains a list, this time a numbered one:
    1. first numbered sub-item;
    2. second numbered sub-item;
    3. third numbered sub-item;
- fourth item, but not numbered 4;

ONE
    the first cardinal number;
TWO
    the second cardinal number;
THREE
    the third cardinal number

Figure 8.5
Browser display of HTML lists

```
<UL>
    <LI>first item, but not numbered 1;
    <LI>second item, but not numbered 2;
    <LI>the third item contains a list, this time a numbered one:

    <OL>
        <LI>first numbered sub-item;
        <LI>second numbered sub-item;
        <LI>third numbered sub-item;
    </OL>
    <LI>fourth item, but not numbered 4;
</UL>

<DL>
    <DT>ONE <DD>the first cardinal number;
    <DT>TWO <DD>the second cardinal number;
    <DT>THREE   <DD>the third cardinal number
</DL>
```

Figure 8.6
HTML source for Figure 8.5

user agents will number the items in an ordered list; the items in an unordered list are marked by some suitable character, often a bullet. The items of a DL element are somewhat different, in that each consists of two elements: a term (DT) and a definition (DD). The intended use of a DL is, as its name suggests, to set lists of definitions; typically each item consists of a term being defined, which will often be exdented, followed by its definition. Figures 8.5 and 8.6 show the use of lists. Note that a list item element can contain a list, giving nested lists.

The most abstract block-level element is DIV, which simply identifies a division within a document that is to be treated as a unit. Usually, a division is to be formatted in some special way. The `class` attribute is used to identify types of division, and a stylesheet can be used to apply formatting to everything that falls within any division belonging to that class. We will see some examples in the following sections. Even in the absence of a stylesheet, classes of divisions can be used to express the organizational structure of a document.

⇨ Although we emphasize the use of the `class` attribute and stylesheets for formatting, HTML provides most textual elements with attributes that provide some primitive layout control. For example, the paragraph element has an attribute `align`, which may take the values `left`, `center`, `right` or `justify`, causing the paragraph to be left-aligned (ragged right), centred, right-aligned (ragged left) or justified (fitted flush to both margins by adjusting the inter-word spacing). The `align` attribute may also be used with HR elements and all levels of heading. When it is used with DIV, it causes all the block-level elements contained within the division to be aligned according to its argument. For example,

```
<DIV align="center">
<P> first paragraph </P>
<P> second paragraph </P>
<P align="left"> third paragraph </P>
<P> fourth paragraph </P>
</DIV>
```

would cause all but the third paragraph to be centred. The third paragraph would be left-aligned, because explicit alignment of an element within a division overrides that of the DIV. Although stylesheets provide superior formatting facilities, it may be necessary to use attributes such as `align` so that user agents that do not support stylesheets can produce an approximation to the intended layout.

Inline elements are used to specify formatting of phrases within a block-level element. As such, they might be seen as being in conflict with the intention of structural mark-up. It is, however, possible to identify certain phrases as having special significance that should be expressed in their appearance without compromising the principle of separating structure from appearance. Examples of elements that work in this way are EM, for emphasis, and STRONG for strong emphasis. Often the content of these elements will be displayed by a visual user agent as italicized and bold text, respectively, but they need not be. In contrast, the I and B elements explicitly specify italic and bold text. These elements *are* incompatible with structural markup and should be avoided. (Especially since a stylesheet can be used to change their effect.)

There is an inline equivalent to DIV: a SPAN element identifies a sequence of inline text that should be treated in some special way. In conjunction with the class attribute, SPAN can be used to apply arbitrary formatting to text, as we will see.

In this section we have considered textual elements that can appear in the body element of an HTML document, but there is one element that can *replace* the body, and that is the FRAMESET. Framesets are interesting because they provide a type of layout that, while deriving from traditional practice, exploits the unique characteristics of screen displays. A frameset divides a page up into a collection of individual *frames*, arranged in a rectangular grid. To that extent, it is just an analogue of the grid often used in magazine layout. However, HTML frames can be updated independently (in ways which will be described later), which opens up new possibilities. Typically, some frames are kept constant, displaying headlines, advertisements, navigational information or copyrights and attributions, for example, while others are used to display changing content. This general form of layout has become something of a cliché on commercial Web sites, but at least it is a cliché based on the form of the medium, and not one transplanted from some other medium.

The FRAMESET element has among its attributes rows and columns, which allow you to specify the arrangement of the grid. Several possibilities exist, including using absolute sizes, but the easiest option is to specify the width of the columns and the height of the rows as percentages of the size of the whole page. For example,

```
<FRAMESET rows="80%,20%" cols="20%,80%">
```

specifies a grid of four frames, arranged in two rows and two columns, with the first row occupying 80% of the height of the page, and the first column 20% of its width. As the example demonstrates, the values of both the rows and cols attributes are strings describing the size of each row and column; the number of rows and columns is implicit, commas are used to separate the individual parts of each attribute value. If either rows or cols is omitted, its value defaults to "100%".

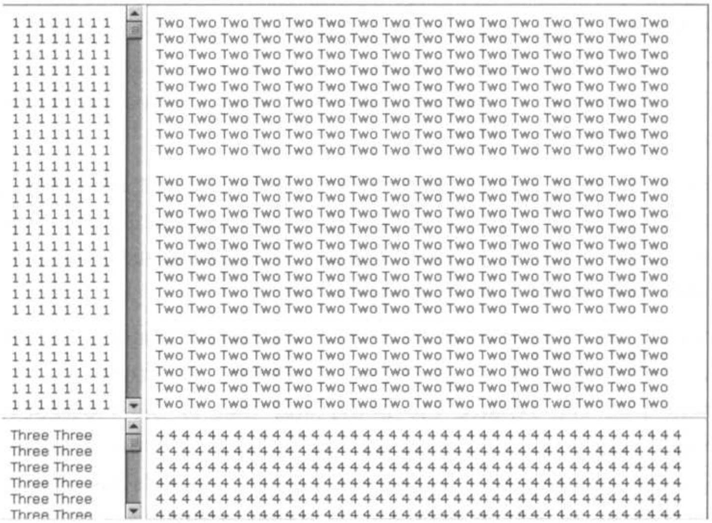


Figure 8.7
Browser display of a simple frameset

A FRAMESET element can only contain FRAME elements or other framesets. Each frame is (perhaps unexpectedly) an empty element; it uses an attribute named `src` ('source') to specify another HTML file, whose contents will be displayed within the frame. The value of the `src` attribute is a URL.[22] The FRAME elements are fitted into the frames specified by their parent frameset in left-to-right, top-to-bottom order, as shown in Figure 8.7, which was produced by the HTML in Figure 8.8. These figures also show how the `scrolling` attribute can be used to add scrollbars to individual frames. (Note also in Figure 8.8 that the document type for a document using frames is different from that which we saw previously.) Nested framesets can be used to produce more complicated frame layouts.

⇨ Framesets must be used judiciously. There is ample anecdotal evidence to suggest that many users find them confusing, precisely because the elements of a page become independent of each other, which undermines the metaphor of a printed page that is generally used to make sense of Web browsing.

All the elements we have described can possess a `class` attribute, which permits subsetting. Additionally, each may have an `id` attribute, which is used to specify a unique identifier for a particular occurrence of the element. For example,

```
<P id="para1">
```

is the start tag of a paragraph identified as `para1`. This identifier can

[22] See Chapter 9 if you have managed to avoid knowing what a URL is.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN">
<HTML>
<HEAD>
<TITLE>Frames Example</TITLE>
</HEAD>

<FRAMESET rows="80%,20%" cols="20%,80%">
<FRAME src="frame1.html">
<FRAME src="frame2.html" scrolling="no">
<FRAME src="frame3.html">
<FRAME src="frame4.html" scrolling="no">
</FRAMESET>
</HTML>
```

Figure 8.8
HTML source for Figure 8.7

be used in various ways, one of which is in a CSS selector, where it must be prefixed by a # symbol instead of a dot. For example,

`P#para1` { *formatting to apply to this paragraph only* }

CSS Properties

It is now time to see how CSS can be used to transform the sparse text markup provided in HTML into an expressive and flexible formatting system. Again, you should note that the principles of layout — the values that can be changed and the ways they can be combined, for example — apply to many text preparation systems. Master pages in a DTP system such as FrameMaker, document styles in LaTeX, and even macros in a word-processor perform a very similar function to that of stylesheets and are applied to the text of a document via its markup in a similar way. In turn, these formatting operations closely resemble the tasks that have been performed in setting type by hand for hundreds of years.

It should be emphasized that CSS is a language for visual formatting, so it can only be meaningfully interpreted by visual user agents. Non-visual user agents have their own stylesheet languages.

CSS allows you to control the typography of your document, by choosing fonts and setting the type size. Five properties control the font characteristics described in Chapter 7. Several of them display some ingenuity in coping with the inevitable uncertainty about the capabilities of user agents and the availability of fonts. This is most evident in the `font-family` property: its value may be a list of font names (separated by commas) in decreasing order of preference. For example:

```
P.elegant  { font-family: "The Sans", Verdana,
                           Helvetica, sans-serif }
```

says, in effect, that we would ideally like text in paragraphs of class `elegant` to be set in a rather recherché font called 'The Sans'. (Note that we must surround the name by double quotes in the CSS declaration because it contains a space.) Should that font not be available, we will settle for Verdana, failing which Helvetica will do. If even Helvetica is not possible, any sans serif font should be used. Our fallback choices are based on pragmatic considerations: Verdana is similar to the preferred font, and is distributed with Internet Explorer, so there is a good chance that it will be available. Next we try for a sans serif, which, although not as good a match, is almost certain to be on any user's system. Finally, we have used a generic font family, in the expectation that a user agent will substitute an appropriate font that falls into that family. CSS provides five such generic families: `serif`, `sans-serif`, `monospace`, `cursive` and `fantasy`. The first four correspond to font styles introduced in Chapter 7; the fantasy family is a catch-all for display fonts, experimental fonts and general eccentricity. The actual font selected when a generic font must be used will depend on the configuration of the browser. There is no actual guarantee that it will fall into the class identified by the name: the standard set of fonts included on a MacOS system does not include any font that would qualify as a fantasy, for example, so something more down-to-earth would have to be used.

> ⇨ The naming of fonts is not standardized, so you must be careful to include in a list of font family names all the pseudonyms by which your chosen font is known. For example, Times New Roman is often identified simply as Times or Times-Roman. If you wanted to be as sure as possible that this font would be used you would have to include all three names at the beginning of your font family list.

The search for a font is carried out for each character — remember that HTML uses ISO 10646 as its character set. Where a document contains text in several languages, or mixes mathematical symbols with text, the glyphs for different characters will be found in different fonts. The list of fonts in a declaration for the `font-family` property can be used to group together a set of fonts that should be used to provide glyphs for all the characters that might occur in a document.

The two properties `font-style` and `font-variant` are used to select different font shapes. The `font-style` property can have the values `normal`, `italic`, or `oblique`, `normal` being upright, and `oblique` what we have termed 'slanted'. The value of `font-variant` may be `normal` or `small-caps` — CSS considers small-caps a variant form rather than a different style, in theory allowing for small-caps italic fonts, and so on. The effect of declarations for these properties is to select an appropriate member of the font family chosen on the basis of the value of `font-family`. A slanted font is considered to be an appropriate choice for the font style `italic` if no real italic font is available.

The `font-weight` property has to deal with the fact that the terms used for font weights only make sense within a font family; as we remarked in Chapter 7, the bold version of some typeface may well be lighter than the medium version of another. For simple situations, CSS lets you use the values `normal` and `bold` for this property, with the expected effect. However, many font families provide more than two different weights, but with no universal naming convention to distinguish between them. Instead of imposing a single naming scheme, CSS uses numbers to identify different weights. You can set `font-weight` to any of the nine values 100, 200, ..., 900, which represent an ordered sequence of fonts, such that, as the `font-weight` value increases (numerically), the font's weight will not decrease, and may increase. Thus, if there are nine different weights (TrueType fonts always have nine different weights), each value will select a different one, but if the font has fewer, some values will be mapped to the same weight of font. Finally, you can use the values `bolder` and `lighter`. This requires a slight digression on the subject of inheritance.

The formatting described by a CSS rule's declaration is applied to any document element that matches the rule's selector. As you know, each element has a parent element. Any properties that are not explicitly changed by a rule for an element are left with the values they had in the parent — the properties' values are *inherited*. This is almost certainly what you would expect. It introduces the possibility of specifying property values not in absolute terms, but relative to the inherited values. Values for `font-weight` of `bolder` and `lighter` are the first example of this that we have seen. Their effect is to set the font weight to the next larger or smaller numerical value that corresponds to a different font from the inherited one, if such a font exists. (You cannot, for example, make a font any bolder than 900.)

A similar option is available for the `font-size` property. This may take on the values `smaller` or `larger`, which cause a relative size change. Font sizes can also be specified as a percentage of the parent element's font size, or as a multiple of the em or ex of the inherited font. Sizes can also be specified independently of the parent font. Here the range of values can be chosen from `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large` and `xx-large`; it is suggested that these values correspond to sizes forming a geometric progression, with a ratio of 1.5. The absolute values will be determined by the choice of size for `medium`, which will be the 'natural' size of the font, as determined by the user agent. There is no guarantee that all user agents will produce the same set of sizes, even for the same font. Sizes can also be specified as absolute lengths, in any unit, although points will usually be preferred.

Normally, when we select a type size, we also specify the line spacing — the vertical distance between baselines. In CSS, the `line-height` property is used for this purpose. The default value, `normal`, allows the user agent to select a 'reasonable' size. This is invariably too small: most user agents (following the advice in the CSS specification) choose a value between 1.0 and 1.2 times the font size. This follows convention for printed matter. This convention is partly dictated by economic considerations: if the lines are closer together, it is possible to fit more words on a page. This consideration is irrelevant for screen display, where the more open appearance lent to text by more widely spaced lines can compensate for the physical strain most people experience when reading from a screen. A line height of about 1.5 times the font size is more suitable. This can be set with any of the following declarations, which illustrate some of the different values available for this property.

```
line-height: 150%;
line-height: 1.5;
line-height: 1.5em;
```

Line heights may be expressed as a percentage, a ratio or in units of ems, all of which are relative to the font in use in the current element. Heights may also be specified as absolute values in any units.

All of the font properties may be combined in a shorthand declaration for the `font` property. Its value is a list comprising the values for the five font properties; the individual property values are separated by spaces, leaving enough context to identify a list of font styles separated by commas.[23] A value for `line-height` can be combined with the `font-size`, separating the two with a slash. This closely follows printers' conventions: 12pt/14pt is the way a printer specifies a 12pt font on 14pt baselines. A typical font declaration would look like:

```
P  {  font: italic bold 14pt/21pt
            "The Sans", Verdana, Helvetica, sans-serif
   }
```

Where no attribute is supplied for a font property (such as `font-variant` here) the default is used. This is usually the value inherited from the parent element.

⇨ The value for `font` can also be one of the names `caption`, `icon`, `menu`, `messagebox`, `smallcaption` and `statusbar`, which refer to the font, in all its aspects, used by the host system to label components of its standard user interface: `messagebox` is the font used in system dialogue boxes, for example. These fonts can be used if you wish to harmonize the appearance of your document with that of the host system's desktop.

One of the important differences between printing and displaying text is that printing in colour is an expensive process, but displaying in colour is free — although you cannot guarantee that everyone has a colour monitor. Therefore, using coloured text and text on coloured backgrounds is an option which should be considered and is often to be preferred to imitating ink on paper by using black text on a white background. This is because the white areas of a screen are actually emitting light, which is usually quite bright and, as we noted in Chapter 6 has a very high colour temperature; it can therefore be quite a strain to look at for long — it is as if the monitor was shining a bright light in your eyes. Furthermore, the high contrast between black text and a white background can result in an optical illusion, whereby the type appears thinner, as if the background had spread over it. The same effect occurs in reverse if white text is placed on a black background. A combination of pale grey text on a dark grey background is surprisingly effective: the dark background reduces the strain caused by the light, and the lessened contrast reduces or eliminates the apparent distortion.[24]

[23] It should not matter what order the values appear in, but browsers are sometimes picky: style, variant, weight, size, family is a safe choice.

[24] The mass protest over the Communications Decency Act, when many Web pages' backgrounds were switched to black, improved the appearance of the Web overnight.

In some circumstances, the use of coloured type can serve other ends. For example, if you are making a multimedia production for small children or primary school teachers, the use of primary colours for the text may make it appear more cheerful and inviting. Red on black may produce a more sinister effect, suitable for a digital Fanzine for a Goth band, perhaps. Many style guides will tell you to be restrained in your use of colour, especially bright colours and harsh contrasts, and your own experience probably tells you that over use of coloured text can be initially irritating and ultimately boring. But a timid and conservative approach to colour is boring too.

The two CSS properties `background-color` and `color` control the colour of the background and the text, respectively. Their values specify colours in the sRGB colour space (see Chapter 6), although user agents are given considerable latitude in how faithfully they approximate sRGB colours. Several formats are provided for colour specifications. The most intuitive takes the form rgb(*r*%, *g*%, *b*%), where *r*, *g* and *b* are the percentages of red, green and blue in the desired colour. Instead of percentages, you may use numbers in the range 0–255 if you prefer. The most commonly used form of colour specification is the least readable: the three components are expressed in hexadecimal (base 16), and combined into a single, six-digit number, preceded by a #. The following specifications all describe the same shade of mauve: `rgb(80%, 40%, 80%)`, `rgb(204, 102, 204)` and #CC66CC.

> ⇨ If, as in this example, each of the component values consists of a pair of repeated digits, the specification can be abbreviated to a three digit form, with each being given just once, as in #C6C. If the digits are taken only from the set 0, 3, 6, 9, C and F, the resulting colour will necessarily belong to the 216 colours of the 'Web-safe' palette.
>
> You can also use names for some colours, but the only ones sanctioned by the standard are the sixteen belonging to the VGA colour palette, so this option is of limited use.

Applying colour, fonts and line spacing to different document elements, such as the body text, headers, emphasized passages and general DIVs and SPANs goes a long way to producing attractive and expressive text for use in multimedia. CSS — and DTP packages — provide control over several more aspects of a document's appearance. We will briefly describe some of the more useful of these, leaving the details to the references given at the end of this chapter.

Lines of text may be aligned in four different ways. In *left-aligned* (or 'ranged left') text, the left margin is straight and words are added to each line until there is not enough room left for another. At that point a new line is started. Since lines will not all be the same length, this results in a ragged right margin. *Right-aligned* text is the same in principle, but the right margin is kept straight by inserting extra space at the left of each line, so that it is the left margin that looks ragged. *Justified* text keeps both margins straight by inserting extra space between words, while *centred* text allows both margins to be ragged, arranging lines symmetrically beween them. In all cases, words might be hyphenated to prevent excessively uneven lines in aligned or centred text or inter-word spacing that is too wide in justified text.

In printed books, justified text is the rule, although left-alignment has many advocates, including respected book designers; it has been enjoying one of its periodic upsurges in popularity during the 1990s. Fully justified text has a neat and orderly appearance, whereas ragged right is less formal, some say more restless. However, even if you wish to present a formal appearance, justification is best avoided for text to be displayed on a monitor. Because of the low resolution of monitors, text is usually set in relatively large sizes, which means that fewer words can be fitted on to a line of reasonable length; this in turn provides fewer gaps between which to distribute the extra space needed for justification, with the result that the text may acquire large or uneven gaps, rivers of white, and other undesirable artefacts that interfere with readability. Right alignment and centring are usually reserved for headings and similar displayed material, although occassionally setting entire paragraphs in one of these modes can be jolly.

Control over alignment in CSS is provided by the `text-align` property, which takes values `left`, `right`, `center` or `justify`. For example, to set the body text of a document ragged right (i.e. left-aligned), except for paragraphs of class `display`, which should be centred, the following rules could be used.

```
BODY      {  text-align: left  }
P.display {  text-align: center  }
```

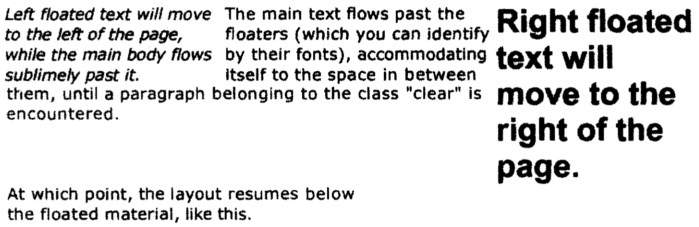
Text layout for multimedia has more in common with magazine and advertising layout than with mainstream book design. Rather than trying to construct a layout for extended reading of flowing text, the multimedia designer is concerned with arranging text on a screen, so that each screenful stands on its own and communicates its message or makes its impact as effectively as possible. An established approach to such layout is to place individual blocks of text on a grid. Designers have been using HTML tables as grids, positioning each block in a table cell, but CSS provides direct control over the positioning of document elements, a facility comparable to the control offered by page layout programs such as PageMaker. To understand positioning in CSS you need to know about CSS's model of text rendering, which is typical of the models employed by layout programs, although less sophisticated than, for example, that of TeX.

The model is fundamentally simple, although the fine detail (which we will gloss over) gets quite involved in places. You will recall that HTML document elements can be classified as either block-level or inline. The layout algorithm reflects this classification. Each element is notionally placed into a box. Text that is not contained in any inline element is placed into an anonymous box. These anonymous boxes and those containing inline elements are placed next to each other horizontally, and then this line of boxes is, as it were, folded up to fit into the available width. The alignment specified for the parent element is used to control the distribution of space between the boxes. In this folding process, some boxes may be split between adjacent lines; for example, an EM element may need to be broken across lines. When the inline elements and text contained in a block-level element have been arranged as described, they are placed in a box corresponding to the whole block. The boxes for blocks are placed vertically, one below another.[25] For displays, there is no notion of 'available height' — windows scroll — so no further adjustment or selection of page breaks is necessary.

In the absence of any further stylesheet information, this layout algorithm produces the familiar sequence of uniformly spaced blocks of text, all fitted into lines of the same width. CSS lets you exert some control over the disposition of text, to produce more interesting and effective layouts.

⇨ Lists are treated slightly differently, since we usually want to display the label or marker for each list element outside the block containing the element itself. The modification to the basic algorithm is simple. A somewhat more complicated algorithm is used to lay out tables, although in all cases, layout is performed by arranging boxes next to each other horizontally and vertically. Full details can be found in the CSS specification.

[25] TeXperts will recognize vertical and horizontal mode.



Left floated text will move to the left of the page, while the main body flows sublimely past it.

The main text flows past the floaters (which you can identify by their fonts), accommodating itself to the space in between them, until a paragraph belonging to the class "clear" is encountered.

Right floated text will move to the right of the page.

At which point, the layout resumes below the floated material, like this.

Figure 8.9
Floating text

Each box can be surrounded by a *border*, which is separated from the box's contents by some *padding*; beyond the border, *margins* can be used to separate the box from its neighbours, or from the edges of its enclosing box. Each box's width and height can be specified explicitly, subject to certain constraints. Colour and backgound can be specified separately for each box. These properties permit such effects as shaded and outlined boxes, narrowed columns and superimposed or overlapping elements (using negative margins). Borders, margins and padding are controlled by a plethora of CSS properties, which provide control over each of them on each of the four sides of the box.

It is also possible to make boxes float to the left or right margin, while text flows around them. This facility is most often used for embedding images into paragraphs of text, but it can also be used, in conjunction with margin settings, to float text past text, as shown in Figure 8.9, produced from Figure 8.10. The `float` property can take the values `left` or `right`, with the expected effect. A complementary property `clear` is used to control the placement of text that might be flowing past a floated box. It takes values `left`, `right`, `both` or `none`, indicating which sides of the box may not be adjacent to a floating box. Putting it another way, a value of `left` for `clear` forces a box to go below the bottom of any left-floating element, and so on.

The ultimate in layout control comes from using *absolute positioning* of elements. If an element is formatted according to a rule that sets the property `position` to `absolute`, then you can assign lengths to the properties `top` and `left`, thus determining the position of the top left hand corner of the box. In conjunction with `width` and `height` these properties allow you to place boxes arbitrarily on the screen. This, of course, allows you to place boxes on top of each other, which raises the question of their stacking order. The `z-order` property can be used to control this: its value

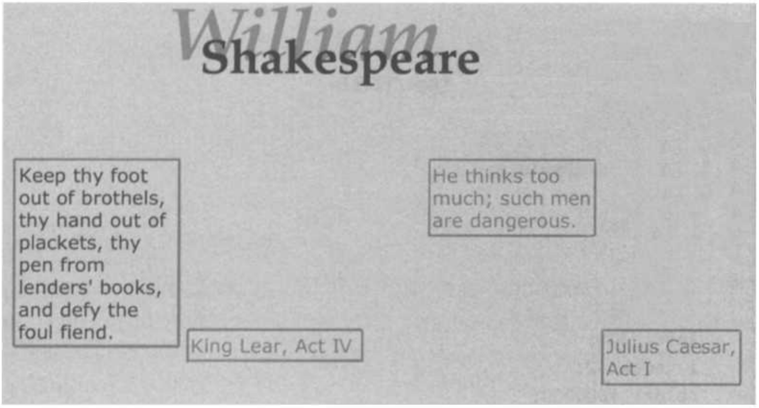
```
<HTML>
<HEAD>
<TITLE>Box effects</TITLE>
<STYLE TYPE="text/css">
P.leftfloater {
   margin-left: 0;
   float: left;
   width: 30%;
   font: italic;
}
P.rightfloater {
   margin-right: 0;
   float: right;
   width: 30%;
   font: bold x-large sans-serif;
}
P.clear {
   clear: both;
   width: 50%;
}
</STYLE>
</HEAD>

<BODY>
<P class="leftfloater">
Left floated text will move ...
<P class="Rightfloater">
Right floated text will move ...
<P class=boxed>
The main text flows ...
<P class="clear">
At which point,...
</BODY>
</HTML>
```

Figure 8.10
HTML and CSS source for Figure 8.9

is a number; elements with higher `z-order` values are placed in front of those with lower values. Use of the absolute positioning properties allows you to lay out text with the same degree of control over its positioning as you could achieve using a graphics program, as described in the first section of this chapter, but without having to convert to a graphics format, and without losing the text content or its markup.

The calculations and measurements needed for absolute positioning can be tedious, so most designers will prefer to use software that enables them to manipulate text boxes interactively on their screens, and generates the markup and stylesheet automatically. Such



William Shakespeare

Keep thy foot out of brothels, thy hand out of plackets, thy pen from lenders' books, and defy the foul fiend.

King Lear, Act IV

He thinks too much; such men are dangerous.

Julius Caesar, Act I

Figure 8.11
Absolutely positioned text

programs essentially provide the layout capabilities of conventional page layout packages.

Figure 8.11, produced from Figure 8.12 is a simple demonstration of absoute positioning.

Several words of caution is in order. You should never forget that user agents may ignore (or fail to correctly interpret) stylesheets.[26] It can be quite difficult to ensure that a document that is laid out using absolute positioning remains comprehensible when it is laid out using a user agent's default interpretation of the structural markup alone. The effect of positioning directives interacts with font sizes, which may be overridden by user preferences. You should also remember that your documents may be rendered by a non-visual user agent, perhaps for the benefit of people with impaired vision. Here again you should avoid producing documents whose comprehensibility depends on their layout.

In our examples, we have shown stylesheets embedded in HTML documents. Where a stylesheet is to be applied to more than one document, duplicating it in every one is wasteful and leads to maintenance problems. Under these circumstances, a stylesheet is usually stored in its own file, and incorporated into every HTML document that needs it, by way of a LINK element, which for this purpose, has the form:

```
<LINK href=" stylesheet's URL"
      rel="stylesheet" type="text/css">
```

(We will describe LINK elements more fully in Chapter 9.)

[26] Absolute positioning is also one of the areas where fourth generation Web browsers are most buggy.

```
<HTML>
<HEAD>
<TITLE>Absolutely Positioning</TITLE>
<STYLE TYPE="text/css">
H1 {
    position: absolute;
    top: 0%;
    left: 25%
}
H1.over {
    top: 5%;
    left: 30%;
    z-index: 2;
    color: #000000;
    font: normal bold 36pt palatino, times, serif;
}
H1.under {
    z-index: 1;
    color: #808080;
    font: normal italic 54pt palatino, times, serif;
}
P  {
    border-color: red;
    border-width: thin
}
P.q1 {
    position: absolute;
    width: 125pt;
    height: auto;
    left: 10pt;
    top: 120pt
}
P.src1 {
    position: absolute;
    left: 140pt;
    top: 250pt
}
```

Figure 8.12
HTML and CSS source for Figure 8.11

Our intention in presenting this description of CSS is to illustrate the sort of control over text layout that is available to designers, by examining in some detail one particularly accessible means of achieving it. We have by no means described all its features, and have simplified some of those we have discussed. Two features in particular that are worthy of note in passing are context-sensitive selectors and media types. The first of these allows you to stipulate that a declaration should only be applied to elements that occur within another specified element type. For example, you might

```
P.q2 {
    position: absolute;
    width: 125pt;
    height: auto;
    left: 320pt;
    top: 120pt;
}
P.src2 {
    position: absolute;
    left: 450pt;
    top: 250pt;
}
BODY {
    color: #AAAAAAA;
    background: #FFFFCC;
}
</STYLE>
</HEAD>
<BODY>
<H1 CLASS="over">Shakespeare</H1>
<H1 CLASS="under">William</H1>
<P CLASS="q1">
Keep thy foot ...
</P>
<P CLASS="src1">
King Lear, Act IV
</P>
<P CLASS="q2">
He thinks ...
</P>
<P CLASS="src2">
Julius Caesar, Act I
</P>
</BODY>
</HTML>
```

Figure 8.12
HTML and CSS source for
Figure 8.11 (continued)

wish to apply a special sort of formatting to EM elements that occur within other EM elements, or within paragraphs belonging to the class `special`, say. Media types allow you to specify that certain parts of a stylesheet apply only when the document is being rendered to certain media. Thus, you can specify different fonts for printing on paper and for displaying on screen, for example.

⇨ You may be wondering why *Cascading* Style Sheets. Stylesheets may be provided by document designers, but they may also be provided by the user; user agents must necessarily provide a default stylesheet. The term 'cascading' refers to the way in which stylesheets from these sources are combined. Basically, designers' stylesheets supplement or override the users', which in turn supplement or override the user agent's. A user's stylesheet may, however, designate certain rules as *important*, in which case they override any conflicting rules in the designer's stylesheet. A user with poor eyesight might, for example, use an important rule to ensure that text is always set in a large size. However, designers' rules can also be important, and in that case, override any users' rules. This is a facility that should be used with discretion — it is, at the least, impolite to insist on imposing your vision on all users, even though you are a sensitive artist, when you don't know what special needs they might have.

Portable Documents

We have argued in favour of structural markup, but we must now concede that there are occasions when it is not adequate. The reasons for this can be found in two inherent limitations of the model of markup and display derived from SGML.

The first of these is the delegation to the user agent of the task of combining markup with stylesheets. What this means is that the designer cannot be sure what is going to happen when his or her marked up text is eventually displayed. Users might override parts of a stylesheet with important rules of their own; a user agent might ignore stylesheet information, or make a poor job of interpreting it; resizing a window may cause a user agent to reflow text, completely changing the layout of a page. Without knowing which user agent is being employed and how it is configured, you cannot know what the displayed text will finally look like. If you have been convinced by the arguments in favour of structural markup, you will retort that that is precisely the point. If, however, you have seen a carefully devised layout turned into an unrecognizable travesty by a Web browser that cannot interpret the stylesheet that describes it, you may infer that the point of structural markup is not entirely one in its favour.

The second limitation of structural markup concerns the finer points of layout. It is widely accepted among typographers that what distinguishes really fine layout is not the marks but the spaces. The adjustment of inter-word spacing on a justified line to produce a visually uniform appearance is an operation that depends on an appreciation of how the shapes of the adjacent characters affects their apparent spacing. A clearly visible example is the need to add extra space when the font changes from a slanted or italic shape to an upright shape, unless punctutation immediately follows the italics. Compare this *good* example with this *bad* example. Notice how, in the latter, the ascender of the *d* seems to lean over too far towards the following word. Failure to make this *italic correction* looks particularly bad at low resolution on screen; Web browsers do not make it, and sometimes the result is almost illegible as the inter-word space disappears under the overhang of an italic letter. Other similar niceties that form part of the traditional typesetter's craft include correct hyphenation, the choice of the ratio of the inter-word spacing to the line length, and the spacing after punctuation. Although we have come a long way from word processors that could only use monospaced fonts, performed justification by inserting fixed-width spaces, and hyphenated 'God', only the most classy document preparation and page layout packages provide a level of control over these details approaching that of the best hand typesetting, and in most cases, some user intervention in the form of visual markup is required to produce the best results. This markup cannot be applied to structural elements because it is determined by the way lines are made up, which is a feature that structural markup abstracts away. Therefore, if you use structural markup, you must entrust these fine details of layout to user agents. Contemporary (fourth generation) Web browsers make a very poor effort at this task. In principle, much better results can be achieved automatically, but in practice even such sophisticated spacing and hyphenation algorithms as those employed by TEX cannot match the results that can be produced by hand.

Often in multimedia production, these considerations will be irrelevant. Low resolution displays preclude the most elegant spacing, and it will often be possible to design text components in such a way that their precise layout is not important. Sometimes, though, only the best will do, or a particular layout may be an integral component of the way information is being presented. What can be done then?

Achieving the desired appearance is not the problem, provided one is prepared to live with visual markup and the consequent loss of flexibility and abstraction. Distributing the resulting document is not so easy. To confine ourselves to pure text for the moment, let us suppose that we have prepared a document using LATEX, and hand-tweaked it to perfection. If we want to send it to you, what can we do? We could send you the marked-up source. This will be no use to you at all unless you have up-to-date versions of TEX and LATEX, and all the macro packages we might have used, installed on your computer. Even if you do, we cannot rely on your having the fonts we have used, unless we restrict ourselves to a very limited choice; we cannot send you our fonts, because that is probably illegal. We have the same problems if, instead of sending the source, we send the device-independent output produced by TEX, since you still need a TEX back-end driver to process it (even though you will no longer need all the LATEX packages). We can translate this output into PostScript and hope that you can interpret that. This is a popular option, but retains some difficulties. Chief among these is the necessity for you to have a PostScript interpreter of some sort available. PostScript printers remain more expensive than other types, and although free software interpreters, such as GhostScript, are available, which can also render PostScript to the screen, they are slow and clumsy. We still have our usual problem with fonts: either we must hope that you share our taste in fonts, or must embed fonts in the PostScript, thereby increasing its size considerably. In desperation, we might convert the output to GIF or some other widely used bitmapped graphics format, but, apart from the inevitably substantial size of the resulting graphics files, this would require us to choose a resolution, which might not be appropriate.

Although we have used LATEX as our example here, much the same situation prevails with any other page layout program: either the author has to hope that the reader possesses the same software and fonts, or the document must be distributed as PostScript or some other graphics format. Multimedia authoring packages such as Director do not provide a solution to this dilemna, since their own built-in text handling facilities are no more sophisticated than a typical word-processor and do not offer the high-quality typesetting we need. Only by importing the output from a high-end page layout program as graphics can we incorporate properly laid out text in a Director movie.

What is needed is an *portable format* for typeset text, and preferably one that can accommodate other elements of multimedia too. Adobe's *Portable Document Format (PDF)* is an attempt to provide exactly that. PDF can be thought of as a simplified PostScript, with which it shares its imaging model. That is, PDF allows shapes to be described in vector terms, and supports the precise placement of characters from outline fonts. This means that a PDF file can accurately represent the effect of any markup that has been applied, so that the document will be displayed exactly as it was intended to be.

PDF also provides a solution to the problem of fonts. The preferred way of dealing with the fonts in a document is to embed in the PDF file not the font itself but a description of it, consisting of its name, character widths and sufficient other font metric information to characterize the font's general appearance. If the named font is present when the PDF document is displayed, it will be used; if it is not, then the font metrics will be used to construct an instance of a Multiple Master font with the same character widths (so that layout is not disturbed) and similar font characteristics, so that the appearance of the page — the density of the print and its lightness or darkness — closely approximates the effect that would be produced by the specified font. If this is not considered acceptable, Type 1 fonts can be physically embedded in PDF. (Symbol and dingbat fonts, in fact, must be embedded, because font substitution only works for text fonts.)

Although it was originally designed as a text and graphics format, in later releases, PDF acquired facilities for hypertext links, inclusion of sound and video, and integration with the World Wide Web. It can now be seen as a platform-independent multimedia format, based on visual layout.

But, like any other 'portable' format, PDF is only any use to someone with a program that can display it. Adobe overcame this problem by the sublimely simple strategy of producing such a program — Acrobat Reader — for all major platforms and giving it away free.[27] They also published the specification of PDF so that anyone can write a PDF interpreter. The Acrobat Reader package includes the two basic Multiple Master fonts (serif and sans serif) so that the font substitution mechanism is also assured to work.

All of which is all very well, but how is PDF to be generated? There are two mechanisms. On MacOS and Windows systems, where printing is performed as a system-level function, the PDFWriter program can be used as if it was a printer driver: if it is selected for printing in the same way as any real printer, any program that produces printed output can produce PDF simply by sending it to PDFWriter, using the same command as would be used for printing.[28] Where this mechanism is not appropriate, and on Unix

[27] We have lost count of the number of copies of Acrobat Reader we have received on CD-ROMs and as part of the distribution of other programs. You probably have, too.

[28] There are some limitations on how PDFWriter handles embedded images. In particular, EPS is usually rendered at low resolution.

systems, an alternative means of generating PDF is provided in the form of Acrobat Distiller, a program that takes PostScript as its input and generates equivalent PDF. Since almost any high quality layout program can produce PostScript, this mechanism is almost as general as using PDFWriter. If video, links, or sound are required, they must be added in a PDF editing program; Acrobat Exchange is provided for this purpose. Needless to say, PDFWriter and Acrobat Distiller and Exchange are not given away free.

PDF has had mixed fortunes. It has become a favoured format for distributing machine-readable manuals and other documentation, but it has not achieved the status of a universal portable format to which it aspires. Factors contributing to this lack of acceptance include the bulkiness of PDF files, relative to plain text or HTML, the unattractiveness of the interface to early versions of Acrobat Reader, and the limited multimedia support provided by Acrobat Exchange. Possibly, though, the need for PDF is not widely perceived: within a particular community, *de facto* interchange formats already exist; $\LaTeX$ within the academic technical community, MS Word in the business community, PostScript in printing and pre-press,[29] and so on. HTML, despite its shortcomings, perhaps already has some claim to be a universal portable document format, Web browsers being even more ubiquitous than Acrobat Reader. It has the further advantage that it can be generated with any text editor. For many purposes, now that CSS offers the potential for control over formatting, HTML is acceptable. It possesses the advantages of structural markup which we have previously extolled, and is easily distributable, being the native language of the World Wide Web.

[29] Somewhat ironically, it is in pre-press that PDF has been most enthusiastically received, being easier to handle than PostScript.

Further Information

[Wil93] describes the principles of traditional book design; [Sas93], [Got98] and [Car97] consider, from different viewpoints, the impact of digital technology on those principles. [GQ99] includes an annotated version of the XML specification [XML]. [TDT96] is a short introduction to SGML from the writer's point of view. [Gra96] and [Gra97] are good descriptions of HTML and CSS, respectively; [HTMa] and [CSS] are the corresponding standards. [Ado93] describes the format of PDF documents.

Exercises

1. What advantages would be gained from re-defining HTML using an XML DTD instead of an SGML DTD? What would be the disadvantages?

2. By default, EM elements are usually displayed in italics. As we stated in the previous chapter, for display on screen, it may be preferable to use boldface for emphasis. Write a CSS rule to specify this appearance for EM elements.

3. How would you specify that an entire HTML document should be displayed with one and a half times the normal leading?

4. Write a CSS rule to set level 1 headings as 'call outs', that is, the heading text should be typeset in the margin with its top on a level with the top of the first paragraph of the following text. (You will need to specify a wide margin.)

5. For pages that use CSS extensively, it would be polite to display some text that could only be seen by browsers that could not interpret stylesheets, explaining that the layout was being ignored by the browser. There is no simple way of discovering whether a user agent supports stylesheets. Devise a means of embedding such a message in a page, using only HTML and CSS.

6. Construct a CSS stylesheet in which all the block-level and inline HTML elements described in this chapter are distinguished purely by their colour. Choose a colour scheme that is visually appealing, and ensures that people with defective colour vision can still distinguish the different elements.

7. Construct a simple example of a document that has a different meaning depending on whether or not a user agent correctly interprets stylesheets.

8. Find a Web page that uses visual layout tags and attributes. (You will have to view the HTML source to discover this.) Take a copy of the page, remove the visual layout and replace it with a CSS stylesheet that produces an identical layout on a suitably equipped browser.

9. Rewrite your CSS stylesheet from the previous exercise to change every aspect of the page's layout without altering the HTML tags.

10. Could PDF be used as an alternative to HTML for Web page layout?

Hypertext

9

In Chapters 7 and 8 we described the character of digital text, one of the most important components of multimedia. As well as the simple text described there, multimedia productions often make heavy use of *hyper*text. Hypertext is text augmented with *links* — pointers to other pieces of text, possibly elsewhere in the same document, or in another document, perhaps stored at a different location. A navigational metaphor is usually employed to describe how these links work: the place a link occurs is called its source, the place it points to is its destination. A user 'follows' a link from its source to its destination: when a representation of the source is selected, usually by clicking with a mouse, the part of the document containing the destination is displayed, as if the user had jumped from the source to the destination, much as one follows a cross-reference in an instruction manual. Hypertext thus allows us to store a collection of texts and browse among them.

Because the user's interaction with hypertext is more complex than that with text, we must consider more than just appearance when we look at the interface to hypertext. As well as the question of how a link is to be displayed, we must consider whether and how to store a record of links that have been followed, so as to permit backtracking, and we must consider what additional navigational facilities should be provided. For most people, the most familiar example of hypertext is the World Wide Web, and the navigational facilities provided by the popular Web browsers illustrate the sort of facilities that are required if users are to be able to browse a collection of hypertext documents comfortably.

You will almost certainly be aware, though, that the World Wide Web is not just made up of hypertext. Web pages can have images, sounds, video, and animation embedded in them, and scripts associated with them to perform interaction with the user. This network of media elements connected by links is an example of *hypermedia*, of which hypertext is a special case, restricted to the single medium of text. In describing hypertext in this chapter we will be laying the groundwork for a description of hypermedia in Chapter 13. Despite the added technical complexity of dealing with a multitude of media types, the generalization from hypertext to hypermedia is conceptually a simple one, so we can safely begin our technical discussion by considering hypertext. The development of the two is inseparable, so, as we wish to begin by reviewing this development, we will start by considering hypertext and hypermedia together.

A Short History of Hypertext and Hypermedia

Although popular perceptions of hypermedia focus on the World Wide Web, the concept has a long history. Its origin is generally traced to an article[1] written in 1945 by Vannevar Bush, in which he described a machine for browsing and annotating a large collection of documents. The *Memex*, as Bush's device was known, included a mechanism for creating links between documents, allowing documents related to the one currently being read to be retrieved, in much the same way as Web pages related to the current one can be accessed by following a link. The Memex was a mechanical device, based on photosensors and microdots, and it is hardly surprising that it was never built — digital computer technology provides a much more suitable mechanism for linked data retrieval.

Bush's contention was that association of ideas was fundamental to the way people think, and that document storage systems should work in a way that reflects these associations, hence the need for links. This interest in matching retrieval to a hypothesized way of thinking persisted in the earliest work on computer-based hypertext: one of the first working hypertext systems was developed under the aegis of an institution with the grandiose

[1] V. Bush, "As We May Think", *Atlantic Monthly*, July 1945.

title of the Augmented Human Intellect Research Center.[2] Another feature of early writing and research on hypertext is the assumption that links would be added dynamically by different users of the system, so that the linked structure would develop over time on the basis of interactions between a community of users. There remains a considerable overlap between hypermedia systems and computer-supported collaborative working.

Among several experimental systems developed in the late 1960s and early 1970s, Ted Nelson's Xanadu project deserves mention, because it was intended to be a global system, storing the entire world's literature. In this sense, it foreshadowed modern distributed hypermedia systems, unlike the smaller scale, but more practical, systems that were developed in the years that followed. It was in connection with Xanadu that the word 'hypertext' was devised.

A number of experimental systems were developed throughout the 1970s and early 1980s. As experience with hypertext grew, some of these systems began to be used in earnest, particularly in information kiosks and similar applications. At this stage, much attention was paid to human interface issues, and to developing models of hypertext systems. Further work on these lines was rendered largely irrelevant by the arrival of *Hypercard* in 1987.

Hypercard, developed by Apple, was for several years distributed free with every Macintosh computer. The consequent sudden mass distribution of a hypertext system allowed Hypercard to establish itself as the paradigm of such a system — for a while, Hypercard 'stacks' become synonymous with hypertext — irrespective of any flaws in its interface and model of hypertext. Most, if not all, of the features to be found in Hypercard were derived from earlier systems, particularly Xerox's NoteCards; its importance lies not in innovation, but in popularization. Among those features were a card-based metaphor for organizing linked material, support for a variety of media, and the provision of a scripting language that allows actions to be associated with events and controls on a card. This language, HyperTalk, is sufficiently powerful to have allowed users to develop sophisticated applications in HyperCard — *Myst* is a HyperCard stack, for example.[3]

HyperCard brought hypermedia out of the laboratories and into the world; the World Wide Web made it into a cultural phenomenon. Unlike earlier systems, which used proprietary formats and were largely closed and self-sufficient, the Web uses publicly available

[2] We leave it to the reader to judge whether the majority of contemporary Web sites can be said to augment the human intellect.

[3] The last version of HyperCard was marketed as an application development tool, not a hypertext system.

technology — anyone can create a Web page with a text editor, and plug-ins and helper applications allow browsers to handle arbitrary media types. This technology is, in fact, fairly crude, and it took over five years for the World Wide Web to evolve from a simple distributed hypertext system to a full hypermedia system with interactivity. The ubiquity of the World Wide Web means that its continuing evolution is likely to define the nature of hypermedia for some time to come.

The Nature of Hypertext

Text becomes hypertext with the addition of links which connect separate locations within a collection of hypertext documents. Links are active: using some simple gesture, usually a mouse click, a user can follow a link to read the hypertext it points to. To make this happen, a piece of software, called a *browser* is required.[4] Usually, when you follow a link, the browser remembers where you came from, so that you can backtrack if you need to. The World Wide Web is an example of a (distributed) hypertext[5] system, and a Web browser is a particular sort of browser.

Browsers tend to encourage people to read hypertext in a *non-linear* fashion: instead of starting at the beginning and reading steadily, you might break off to follow a link, which in turn might lead you to another link that you can follow, and so on. At some point, you might go back to resume reading where you left off originally, or you may find it more fruitful to go on pursuing links.

It is a common misconception that such non-linear browsing is the distinctive innovation of hypertext. The following quotation is an extreme expression of this view:

> "All traditional text, whether in printed form or in computer files, is sequential. This means that there is a single linear sequence defining the order in which the text is to be read. First, you read page one. Then you read page two. Then you read page three. And you don't have to be much of a mathematician to generalize the formula to determine what page to read next."[6]

Leaving aside the mathematical invalidity of the proposed generalization, a glance at any magazine will show just how mistaken

[4] A browser may be an independent program that reads hypertext documents as data, or it may be integrated with the document itself, to produce 'standalone' hypertext.

[5] Actually, hypermedia, as we have seen.

[6] Jakob Nielsen, from 'Usability considerations in introducing hypertext' in *Hypermedia/Hypertext*, ed. Heather Brown, Chapman and Hall (1991).

is the notion of a single linear sequence for reading printed text. What we find is a collection of articles that can be read in any order, broken up by pages of advertisements, to be skipped over or glanced at between paragraphs; a contents page contains pointers to each article. Many magazines feature 'sidebars', with information tangential to the main text of the article; picture captions interrupt the linear flow of text; and articles may be split ('continued on page 52'), with their continuation many pages further on and parts of several other articles intervening.

Other forms of printed matter also display non-linearity. An encyclopedia is made up of short articles, which are often cross-referenced, so that having read one you may be led on to another. Cross-referencing is also common in manuals for software and in all sorts of technical reference material. The tiresome scholarly apparatus of footnotes and bibliographic references comprises a sort of system of links, and much academic research consists in following those links. Even novels, which usually have a linear structure are often read non-linearly: a reader might find it necessary to go back to an earlier chapter ('Was Piotr Ivan's brother-in-law or his faithful servant?') or even to go to the end to find out who murdered the Colonel; it may be necessary to stop reading the novel and resort to a dictionary to find out what 'tergiversate' means. Some novels explicitly use non-linear structures: Vladimir Nabokov's *Pale Fire*, and Flann O'Brien's *The Third Policeman*, for example, both include extensive notes, arranged in the same way as academic endnotes and footnotes, which must, therefore, be read out of the linear sequence of the text, but nevertheless play a major role in telling the story.[7]

It would not be going too far to say that, taken over all the varieties of reading, non-linearity is the norm when it comes to reading text. What is novel and distinctive about computer-based hypertext is the immediacy with which links can be followed, which creates a qualitatively different experience. Contrast waiting six weeks for a copy of a conference paper you have ordered on inter-library loan because you saw a reference to it in a journal with clicking on a link in a Web page and being shown the text of the same paper, as stored on a computer somewhere on the other side of the world, within a few seconds (on a day when Internet traffic is flowing briskly). Although text on paper does not have a unique linear sequence for reading, it does have a unique linear physical sequence, so following cross-references, even within a

[7] It might be argued that the notes in *The Third Policeman* actually tell part of the story of the same author's *The Dalkey Archive*, thereby providing an example of inter-document linkage.

single text, requires the reader to flip forwards and backwards between pages, using page numbers to identify the destinations of cross-references. Even though computer memory is also arranged linearly and addressed sequentially, layers of abstraction imposed by software on top of this arrangement, together with the speed of operation of modern computers and networks, makes this linearity transparent, so that hypertext systems can provide the illusion that links connect together disparate pieces of text into a network (you might even call it a web) that you can travel on.

Links

Hypertext raises some new issues of storage and display. How are links to be embedded in a document? How are their destinations to be identified? How are they to be distinguished typographically from the surrounding text? To appreciate the answers to those questions, we need a clear understanding of what a link is, and what sort of thing links connect.

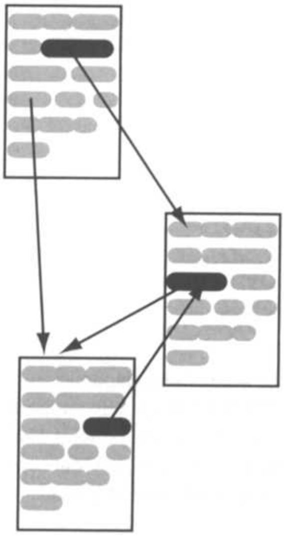


Figure 9.1
Simple uni-directional links

The simplest case is exemplified by the World Wide Web. If we confine ourselves for the moment to pages consisting purely of text, these are self-contained passages, which may be of any length, but usually fit on a few screens. Their only connection with other pages is through links (even though a link may be labelled 'next' or 'previous', such sequential connections between pages must be explicit). Within a page, though, the normal sequential structure of text is exhibited: you can sensibly read a page from beginning to end (although you don't have to), and elements such as headings and paragraphs are used to structure the content of the page. The HTML source of a Web page is often held in a file, but some pages are generated dynamically, so you cannot always identify pages with files, or indeed, with any persistent stored representation.

Hypertext systems are generally constructed out of self-contained elements, analogous to Web pages, that hold textual content. In general, these elements are called *nodes*. Some systems impose restrictions on their size and format — many early hypertext systems were built on the analogy of 3 by 5 index cards, for example — whereas others allow arbitrarily large or complex nodes.

In general, hypertext links are connections between nodes, but since a node has content and structure, links need not simply associate two entire nodes — usually the source of a link is embedded somewhere within the node's content. To return to the World Wide Web, when a page is displayed, the presence of a link is indicated by highlighted text somewhere on the page, and not, for example, by a pop-up menu of links from that page. Furthermore, a link may point either to another page, or to a different point on the same page, or to a specific point on another page. Hence, Web links should be considered as relating specific locations within pages, and, generally, links connect parts of nodes (see Figure 9.1).

In HTML, each link connects a single point in one page with a point (often implicitly the start) in another, and can be followed from its *source* in the first page to its *destination* in the other. We call links of this type *simple uni-directional links.* XML and other more elaborate hypertext systems provide a more general notion of linking, allowing the ends of a link to be regions within a page (*regional links*), links that can be followed in either direction (*bi-directional links*), and links that have more than just two ends (*multi-links*). Figure 9.2 illustrates these generalized forms of link.

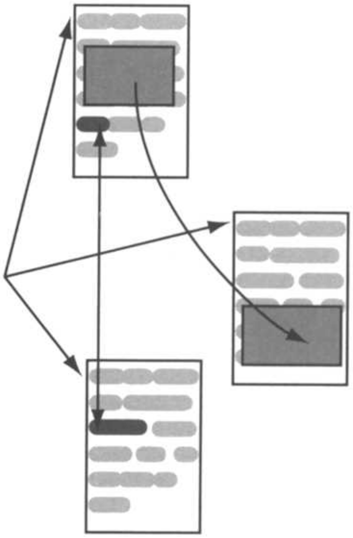


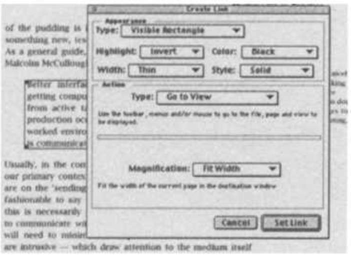
Figure 9.2
Regional, bi-directional and multi-links

Links in PDF

Adobe's Portable Document Format (PDF) supports hypertext linkage. PDF links are uni-directional, but not quite simple, since a restricted form of regional link is provided: each end of a link is a rectangular area on a single page. Since Acrobat Distiller and PDFWriter make it possible to convert just about any text document to PDF, and links can then be added using Acrobat Exchange, this means that hypertext links can be added to any document, however it was originally prepared.

➲ In PDF, links are considered to be annotations, that is, they are not part of the page's contents proper. This is reflected in the way links are implemented. A PDF document comprises a hierarchically organized collection of objects. (In a PDF file, the objects are stored in a textual form, using keywords to identify their type and attributes.) A document object consists of an `outline` object and a `pages` object, which is organized as a tree of page objects. Within each page, separate objects are used to hold the imageable content and the annotations. Link objects, containing the coordinates, in a device-independent coordinate space, of the source rectangle (on the parent page), a specification of the destination, and some information describing how links appear when the document is displayed, are stored within the `annotations` object. If the destination is within the same document as the source, it may be specified in the `link` object as a page sequence number and the destination rectangle, together with additional information specifying the size at which the destination is to be displayed after the link is followed. If it is in another file, a separate `destination` object is used, which holds this information together with the name of the file containing the destination. A pointer to this `destination` object is then stored in the link. (This indirect form may also be used for links within a document; the special treatment of these internal links is provided for efficiency reasons, and to maintain compatibility with PDF version 1.0, which only supported links within a single document, and only allowed the destination to be specified directly within the link object.)

Links may be displayed by Acrobat Reader in a variety of styles. The default representation consists of a rectangle outlining the region that is the source of the link. When a user clicks within this rectangle it is highlighted — by default, colours within the region are inverted — and then the viewer displays the page containing the destination region. The actual destination, always a rectangular region, is highlighted. When the link is created, the magnification to be used when the destination is displayed can be specified, which makes it possible to zoom in to the destination region as the link is followed.



Type: Visible Rectangle

Highlight: Invert Color: Black

Width: Thin Style: Solid

Type: Go to View

Use the toolbar, menus and/or mouse to go to the file, page and view to be displayed.

Magnification: Fit Width

Fit the width of the current page in the destination window

Figure 9.3
Adding a link to a PDF document

Adding links in Acrobat Exchange is a two-stage process. First, you select the link tool and use it to drag out the source region of your link. By default, this region is an arbitrary rectangle; by holding down a modifier key, you can constrain it to be the tightest possible rectangle that can enclose text that you select with the tool. Thus, sources can appear to be words in the text, if that seems appropriate. Once the source region has been selected, a dialogue box appears in which you can select the display styles for the source (see Figure 9.3[8]). In the second stage, you use the navigational controls of the program to move to the page containing the link's desired destination; if necessary, you can open a different file at this stage. If your destination is a specific region on this page, you can use a selection tool to draw its outline; otherwise, the top of the page is used as the destination. Once the destination has been selected, clicking on the button labelled `Set Link` causes the link to be stored.

[8] The pop-up menu labelled Type in the `Action` pane will be explained in Chapter 14

Links in HTML

The links that can be embedded in a Web page composed in HTML are simple and uni-directional. What distinguishes them from links in earlier hypertext systems is the use of *Uniform Resource Locators (URLs)* to identify destinations.

➲ A URL uniformly locates a resource, but the concept of ‘resource’ thus located is quite hard to pin down.[9] In practice, a resource is anything that can be accessed by one of the higher level Internet protocols such as HTTP, FTP, or SMTP. Often, but by no means always, a resource is a file, or some data, but the way in which you can access that data is constrained by the protocol you use. For example, a `mailto` resource identifies a user's mailbox, but only allows you to access the mailbox by sending a message to it. An `ftp` resource identifies a file, and might even be used to identify a user's mailbox on systems where mailboxes are stored as files, but an `ftp` resource can be fetched over the network from its remote location. A resource is thus something like an abstract data type, identifying some data and providing a set of operations that can be performed on it.

⇨ The HTML 4.0 specification stipulates the use of Uniform Resource *Identifiers* (URIs), rather than URLs. URIs are a superset of URLs, which also include *Uniform Resource Names* (URNs). URNs are intended to provide a location-independent way of referring to a network resource, unlike URLs which pin a resource down to a specific location for all time. In practice, Web pages invariably use URLs, so we will stick to the more familiar term and concept.

The URL syntax provides a general mechanism for specifying the information required to access a resource over a network. For Web pages, three pieces of information are required: the *protocol* to use when transferring the data, which is always HTTP, a *domain name* identifying a network host running a server using that protocol, and a *path* describing the whereabouts on the host of the page or a script that can be run to generate it dynamically. The basic syntax will be familiar: every Web page URL begins with the prefix `http://`, identifying the HTTP protocol. Next comes the domain name, a sequence of sub-names separated by dots,[10] for example `www.wiley.co.uk`, which usually identifies a machine within an organization, within a sector within a country. (In the US, the country is usually implicit, and the top-level domain is the sector — commercial, educational, government, etc.) There may be more

[9] The specifications are not unambiguously helpful here: “A resource can be anything that has identity [. . .] The resource is the conceptual mapping to an entity or set of entities, not necessarily the entity which corresponds to that mapping at any particular instance in time.” (IETF RFC 2396 *Uniform Resource Identifiers (URI): Generic Syntax*). “ [R]esource: A network data object or service that can be identified by a URI,. . . ” (IETF RFC 2068 *Hypertext Transfer Protocol - HTTP/1.1.*

[10] Occasionally, the numerical IP address is used here (see Chapter 15).

intermediate domains in a domain name than we have shown in this example: universities, for example, typically assign sub-domains to departments, as in `www.cs.ucl.ac.uk`, a Web server in the computer science (CS) department of University College London, an academic institution in the United Kingdom.

After the domain name in a URL comes the path, giving the location of the page on the host identified by the preceding domain name. A path looks very much like a Unix pathname: it consists of a /, followed by an arbitrary number of segments separated by / characters. These segments identify components within some hierarchical naming scheme. In practice, they will usually be the names of directories in a hierarchical directory tree, but this does not mean that the path part of a URL is the same as the pathname of a file on the host — not even after the minor cosmetic transformations necessary for operating systems that use a character other than / to separate pathname components. For security and other reasons, URL paths are usually resolved relative to some directory other than the root of the entire directory tree. However, there is no reason for the path to refer to directories at all; it might be some access path within a document database, for example.

⇨ In advertisements and other published material, you will frequently see the URLs of Web pages given with the leading `http://` omitted, and some Web browsers allow you to omit the protocol when typing URLs, making intelligent guesses on the basis of the rest. Although such usage in these contexts is sanctioned in the relevant standards, within an HTML document, you must always use a complete URL, or a partial URL of the form, and with the meaning, described below.

The only characters that can be used in a URL belong to the ASCII character set, because it is important that URL data can be transmitted safely over networks, and only ASCII characters can be considered safe for this purpose — and not even all ASCII characters. Certain characters that may get corrupted, removed or misinterpreted must be represented by escape sequences, consisting of a `%` character followed by the two hexadecimal digits of the character's ASCII code.[11] In particular, spaces must be written in a URL as `%20`.

A URL as described so far identifies a Web page in one of three ways. In all cases, the domain name identifies the host running an HTTP server. The path might be a complete specification of the location of a file containing HTML, as in `http://www.wiley.co.uk/`

[11] Compare this with the QP encoding described in Chapter 7.

compbooks/chapman/index.html or, if it ends in a / character, the specification of the location of a directory. As a special case, if the path consists only of a /, it may be omitted: http://www.wiley.co.uk/ and http://www.wiley.co.uk both identify the root directory of the UK mirror of John Wiley and Sons' Web site. Where a path identifies a directory, the configuration of the Web server determines what the URL specifies. Often, it is a file within that directory with a standard name, such as index.html.

The third way in which a URL identifies a Web page is via a program that generates the content dynamically. Again, the precise mechanism depends on the server's configuration, but usually such programs must reside in a specific directory, often called cgi-bin, or must have a special extension, such as .acgi. The occurrence of the letters CGI in both of these conventions indicates that a mechanism known as the *Common Gateway Interface* is being invoked. This mechanism provides a means for a server to pass information to and receive it from other programs, known as CGI scripts. CGI scripts are commonly used to provide an interface — or *gateway* — between the Web server and databases or other facilities. Several mechanisms are provided to pass parameters to CGI scripts, including appending a *query string* to the end of the URL used to invoke the script. The query string is separated from the path by a ?, as in http://ink.yahoo.co.uk/bin/query_uk?p=macavon.[12]

⇨ Although the Common Gateway Interface was intended to provide a standard for interaction between a Web server and other resources, many Web server programs and database systems provide their own methods for performing the same function. These proprietary solutions are often more efficient than CGI scripts, and have been widely adopted in commercial Web sites.

We have described a hypertext network as consisting of nodes connected by links, but the nodes (i.e. pages) of the World Wide Web are grouped into Web *sites*, each comprising a relatively small number of pages, usually held on the same machine, maintained by the same organization or individual, and dealing with related topics. Within a site, links pointing to other pages on the same site are common. *Partial URLs* provide a convenient shorthand for such local links.

Informally, a partial URL is a URL with some of its leading components (the protocol, domain name, or initial segments of the path) missing. When a partial URL is used to retrieve a

[12] For more information on the Common Gateway Interface and CGI scripting see Chapter 15.

resource, the missing components are filled in from the *base URL* of the document in which the partial URL occurs. This base URL is usually the URL that was used to retrieve the document, although sometimes it might be specified explicitly (see below). The way in which base and partial URLs are combined is very similar to the way in which a current directory and a relative path are combined in hierarchical file systems, particularly Unix. For example, if the relative URL `video/index.html` occurs within the document retrieved via the URL `http://www.wiley.co.uk/compbooks/chapman/index.html` it will be equivalent to the full URL `http://www.wiley.co.uk/compbooks/chapman/video/index.html` Within the same document, `/catalog/` is equivalent to `http://www.wiley.co.uk/catalog/`. The special segments `.` and `..` are used to denote the current 'directory' (component in the hierarchy described by the URL path) and its immediate parent, so, again within the same document, the relative URL `../graham/index.html` is the same as `http://www.wiley.co.uk/compbooks/graham/index.html`. Generally, if a relative URL begins with a `/`, the corresponding complete URL is constructed by removing the pathname from the base URL and replacing it with the relative URL. Otherwise, just the last segment of the pathname is replaced, with `.` and `..` being interpreted as described.[13]

⇨ A little thought will show you that, if relative URLs are resolved using the URL of the document in which they occur as the base URL, it is possible to move an entire hierarchy of Web pages to a new location without invalidating any of its internal links. This is usually something you want to be able to do, but occasionally you may need the base URL to be independent of the location of the document containing relative URLs. Suppose, for example, that you have constructed a table of contents for some Web site. You will want to use relative URLs, because all your links point to pages on the same server. If, however, you wish to be able to duplicate the table of contents (but not the site itself) on several sites, you will not want to use each copy's own location as the base URL for these relative URLs; instead, you will always want to use a URL that points to the site you are indexing. To achieve this, you can explicitly set the base URL in a document using the HTML BASE element. This is an empty element, with one attribute `href`, whose value is a URL to be used as a base for resolving relative URLs inside the document. The BASE element can only appear within a document's HEAD. Base URLs specified in this way take precedence over the document's own URL. Hence, if a document included the

[13] It is possible to describe this interpretation as a purely lexical transformation on the constructed URL. We leave this as an exercise.

> element <BASE href="http://www.wiley.co.uk/compbooks">, then, no matter where the document itself was stored, the relative URL chapman/index.html would be resolved as http://www.wiley.co.uk/compbooks/chapman/index.html.

One more refinement of URLs is needed before they can be used to implement simple unidirectional links as we have described them. We emphasized that links connect parts of nodes, but a URL specifies a complete page. To identify a location within a page — a particular heading, or the beginning of a specific paragraph, for example — the URL needs to be extended with a *fragment identifier*, consisting of a # character followed by a sequence of ASCII characters. The fragment identifier functions roughly like a label in assembly language programs: it can be attached to a particular location within a document (we will see how shortly) and then used as the destination for a 'jump'. A fragment identifier is not really part of a URL: it is not used by HTTP requests, but is stripped off by the user agent making the request, which retains it. When the requested document is returned, the user agent will find the location designated by the fragment identifier, and display the document with that part of it in the window.

We have glossed over some of the fine detail of HTTP URLs — if you need to know everything about URLs consult the specifications — but the description just given should suffice to show that URLs provide the information required to access a Web page from anywhere on the Internet. The HTTP protocol provides a means of retrieving a Web page, given its URL, and therefore the combination of URLs and HTTP can be used to implement hypertext links in the World Wide Web provided we have a means of embedding URLs as links in HTML documents. This is provided by the A (anchor) element, which can be used as both the source and destination of a link, depending on which of the attributes href, name and id are provided.

The href attribute is used when an anchor is to serve as the source of a link. Its value is a URL, which may be absolute, if the link points to a document elsewhere on the World Wide Web, or relative, if it points to a document within the same hierarchy. The URL may have a fragment identifier appended to designate a specific location within the destination document, or it may consist solely of a fragment identifier, for links within the same document. The anchor element has content, which is displayed by user agents in some special way to indicate that it is serving as the source of a hypertext link. For example, popular browsers by default show the content of anchor elements in blue and underlined; when a link's destination has been visited, a different colour (often green) is used to display the visited link whenever the document containing the source is displayed again within a designated period of time.

⇨ A CSS stylesheet can provide rules for formatting A elements, just like any other element. To cope with the desire to change the appearance of a link when it is visited, CSS provides special *pseudo-classes* `link`, `visited` and `active` that can be applied to A selectors. They identify subsets of the class of anchor elements, just as a class name does, but whether an anchor belongs to the subset depends on its condition in the browser, not on any attribute values. To distinguish pseudo-classes from ordinary classes, they are separated from the element name in selectors by a colon instead of a dot.

The following rules stipulate that links should be set in boldface until they have been visited, when they revert to the normal text font. While the user clicks on the link, it will appear extra large.

```
A:link {
    font: bold;
}
A:visited {
    font: normal;
}
A:active {
   font: xx-large;
}
```

Pseudo-classes, although they are only used for links in CSS2 provide a more general mechanism that can potentially be applied to other HTML elements.

Despite the elegance of this way of controlling the appearance of links, it should be used judiciously, if at all. Users rely on their chosen browser's default for the appearance of links in order to recognize them. Interfering with this recognition can detract from the usability of your Web pages.

An anchor element with a `name` attribute is presently the most common way of attaching a name to a location within a document. The value of the name can be any string that would be legal HTML, although when the name is used as a fragment identifier any characters other than the subset of ASCII that is legal in a URL must be escaped. Although an anchor attribute that is being

```
<H1>Links and URLs</H1>

<H2 id="links">Links</H2>
Hypertext links are implemented in HTML using the
&lt;A&gt; element and <A href="#urls">URLs</A>.
<P>
[etc, etc.]
</P>
<A name="urls"><H2>URLs</H2></A>
An introduction to the use of URLs in HTML can be found in
<A href=
 "http://www.w3.org/TR/REC-html40/intro/intro.html#h-2.1.1">
the HTML4.0 specification</A>.
They are the basis of <A href="#links">links</A>
in Web pages.
<P>
[etc, etc.]
</P>
```

Figure 9.4
Links and Anchors

used as the destination for hypertext links in this way has content, this is not usually displayed in any special way by user agents. An alternative to anchors with names is provided in HTML 4.0: *any* element may have an `id` (identifier) attribute; this may be used for several purposes, among them the specification of link destinations, by using the identifier as a fragment identifier. The use of identifiers instead of anchor names for this purpose has the advantage that any document element — for example, a header — can be used as the destination of a link, without the necessity of enclosing it in a dummy anchor. The disadvantage is that older browsers do not recognize identifiers.

There is nothing to stop you using the same anchor as both the source of a link, and a potential destination of other links, by providing it with an `href` attribute and a name or identifier. Within a document, each name or identifier must be unique, for obvious reasons. Although it would appear to be possible to distinguish between identifiers and names with the same value, this is not done, and it is not permitted for a name and an identifier to be the same.[14] The behaviour of a user agent in the face of name clashes is not defined.

Figure 9.4 shows examples of links within a document, using both named anchors and a header with an identifier; a link to a section of a separate document is also shown.

[14] In computer language jargon, names and identifiers share the same namespace.

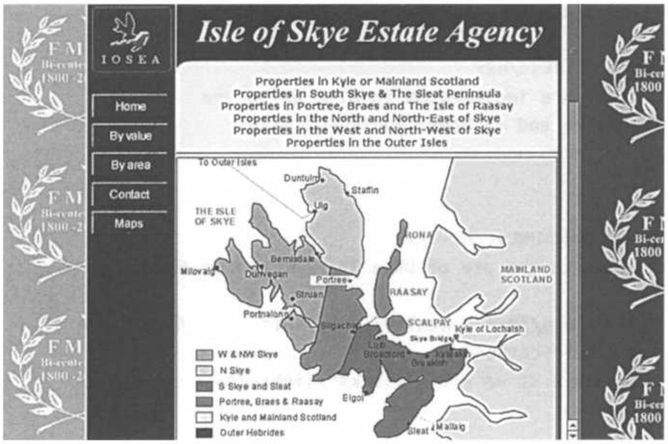


Figure 9.5
Frame-based navigation

Colloquially, we say that when a user clicks on a highlighted anchor element in a Web page, the browser 'goes to' the page that is the destination of the link; by extension, we talk about 'visiting' Web sites and pages. This metaphor of visiting has achieved almost universal currency, despite the fact that, in reality, the opposite takes place: we do not visit Web pages, they come to us. Clicking on a link that references a separate document causes the resource identified by the URL that is the value of the `href` attribute of the anchor to be retrieved via HTTP. Assuming that the resource is a Web page[15] the browser interprets the HTML and any associated stylesheet and displays the page. If a fragment identifier is appended to the URL, the browser will find the corresponding anchor (assuming it is there) and scroll the display to it. Usually, the newly retrieved page replaces any page being currently displayed by the browser, hence the idea of 'going to' the new page. Sometimes, though, the browser may open a new window.

An interesting question arises if the link occurs within a frame being displayed as part of a frameset. Should the new page replace just that frame, or the entire frameset, or should it replace some other frame in the frameset? Any of these may be appropriate. Consider, for example, the typical frame layout shown in Figure 9.5, with a navigational menu in a small frame on the left, and the remainder of the window occupied by a page devoted to one of the topics on the menu. The menu will consist of links to the specific pages; since the intention of this style of layout is to ease navigation through the site by having the menu available all or most of the time, it is obvious that clicking on a link in the menu frame should cause

[15] We describe in Chapter 13 what may happen if it is not.

the retrieved page to be displayed in the main frame. If a page displayed in the main frame contains a link to another part of the same site, though, when it is followed, the new page should replace the old one. Finally, if a page contains a link to some other site, the navigational menu will become irrelevant when the link is followed, so the new page should be displayed in the whole window.

HTML provides control over the placement of retrieved pages within a frameset by allowing you to name frames, using the `name` attribute of the `FRAME` element, and specify the frame in which to display the destination of a link, using the `target` attribute of the `A` element. The value of this attribute may be the name of a frame, in which case, when the link is followed, the destination page is displayed in the corresponding frame. Alternatively, the target may be one of four special names, `_self` (the retrieved page is displayed in the frame containing the link), `_blank` (the retrieved page is displayed in a new window), `_top` (the retrieved page is displayed in the entire window, removing any frameset structure), or `_parent` (the retrieved page is displayed in the frameset containing the frame containing the link; this is the different from `_top` in the case of nested framesets). If no target is specified, `_self` is assumed. In a frameset layout such as that of Figure 9.5, the main frame would have a name such as `mainframe`, which would be used as the target for links in the navigation menu. Links within the site specify no target, so their destinations are displayed in the same frame as their sources, typically the main frame. Links which point beyond the site would have a target of `_top`, so that this frame structure would be effaced when a user leaves the site.

⇨ HTML anchors provide a flexible and general linking mechanism, but this very generality may be its weakness. There is nothing to suggest why a link has been established; in other words, simple `A` elements do not capture any of the semantics of linking. Thus, a user can have no way of knowing what sort of relationship the link expresses, and hence whether it is worth following, and programs cannot provide automatic navigational hints based on, for example, similarities between a link and those previously followed. The `rel` (relationship) and `rev` (reverse relationship) attributes are an attempt to capture more of links' semantics. Their value is a *link type*, which is a string designating a type of relationship that may be expressed through linkage. For example, an occurrence of a technical word in a document may be linked to an entry in a glossary; the `rel` attribute of the source anchor for this link could have the value `glossary`, as could the rev attribute of the destination anchor at the entry for that word in the glossary:

```
... by means of a
  <A href="../terms.html#url" rel="glossary">URL
</A> ...
```

and

```
<DT><A name="url" rev="glossary">URL</A>
  <DD>Uniform Resource Locator ...
```

A related issue concerns relationships between entire documents, such as one document being a translation of another into a different language, that are not expressed naturally by hypertext links between anchors within documents. The LINK element is provided to express such relationships. LINKs can only appear in a document's HEAD; they have an href attribute, whose value is a URL identifying the linked document, and may have rel or rev attributes, serving the same function as they do with anchors. Since they are in the document's head, link elements are not displayed by browsers — in fact, they have no content. It is envisaged that they might be used to construct special purpose menus or toolbars to assist with navigation among a set of related documents.

A collection of standard link types is defined to designate common relationships between documents, such as Next, Prev, and Start, which can be used to connect together a sequence of documents, or Chapter, Section, Subsection, and Contents which can be used to organise a collection of documents in the form of a conventional book. We have previously seen the use of a LINK element, with a rev value of Stylesheet to connect a document to an external stylesheet definition. In addition to the standard relationships, Web page authors are free to invent their own, although these are of less use in helping programs and users understand the structure of a collection of documents.

Unfortunately, capturing the semantics of linkage seems to have little appeal to Web authors and browser vendors: rev and rel have not been widely adopted, and with the exception of the special case of stylesheets, most browsers ignore <LINK> tags. The functionality of these HTML features has been incorporated into XML's linking facilities, described in the next section, where they will hopefully fare better.

XPointer and XLink

XML allows you to define your own document elements, and following the precedent of HTML, it is natural to want to include linking elements among them. It would be simple to stipulate that attributes whose value is a URL (or URI) should be treated as hypertext links, but this would mean that XML was restricted to the same class of simple uni-directional links as HTML provides. This was felt to be inadequate, so a more general mechanism has been proposed. It consists of two parts: XPointer extends HTML's fragment identifier, and XLink generalizes its anchors. At the time of writing, these proposals have not been adopted as W^{3}C Recommendations, so the following description should be treated as an outline of a possible way of implementing the more general forms of link identified earlier in this chapter.

Adding a fragment identifier to a URL used as the value of the `href` attribute of an HTML anchor allows you to link to a specific named anchor or element with an identifier in a document. If there is no such element, you cannot create a link. XPointer defines a syntax that enables you to describe the destination of a link in terms of its position within the document. This means, among other things, that you can create links to arbitrary points in documents whose source you do not have access to. In addition, XPointer provides a way of defining a region within a document, so that it supports regional hypertext links.

As we pointed out in Chapter 8, every element of an HTML document has a unique *parent* element. The same applies to XML document elements. As a consequence, the structure of a document can be represented in the form of a tree, with each element being a child of its parent. Any element can be identified by its position in the tree, relative to the root or a named node, such as 'the second paragraph after the third level two header', or 'the third child of the second element after the element with id "orinoco" that has the same parent'. Xpointer is a means of expressing such positions in a way that can easily be processed by software. XPointer expressions — themselves known as XPointers — are appended to URLs in the same way as fragment identifiers are. (Recall that fragment identifiers are stripped off from a URL before the resource is retrieved, so altering the syntax of what may follow the URL proper does not interfere with protocols that interpret URLs.)

An XPointer is a sequence of terms, separated by dots. You can think of it as a set of instructions giving you directions to follow in the order specified to reach the intended point in the document. The first term usually refers to a fixed location, such as an element with an id, and subsequent terms tell you how to get from there to where you are going, by choosing, for example, the third child or the fourth paragraph. Every term consists of a keyword followed by some brackets, usually containing arguments. The terms `root()` and `id(`*name*`)` are used to establish the starting points for XPointers. The first form refers to the outermost element of a document — the `HTML` element of an HTML page, for example; the second refers to an element with an `id` attribute whose value is *name*,[16] any legitimate XML name. This latter form is similar to the use of fragment identifiers in HTML; for the specially common case of HTML anchor nodes with name attributes, a term of the form `html(`*name*`)` can be used. Thus, the XPointer `id(orinoco)` identifies the element (which should be unique) whose `id` attribute's value is `orinoco`, while `html(tobermoray)` identifies an element with start tag `<A name="tobermoray">`.

Once a starting point has been established, relative motion can be specified by terms using the keywords `child`, `descendant`, `ancestor`, `preceding`, `following`, `psibling`, or `fsibling`. The intent of most of these should be evident: if `E` is an element, a child is any element that is immediately contained in `E` (i.e., `E` is its parent), while a descendant is any element within the content of `E`; conversely, an ancestor is any element within whose content `E` appears. Elements are considered to precede `E` if their start tags appear before it, and to follow it if their end tags appear after it. The siblings of `E` are those elements with the same parent: `psibling` selects siblings that precede `E`, `fsibling` those that follow it. Figure 9.6 illustrates these relationships in terms of the tree structure of a document.

These keywords take between one and four arguments. The first is a number, allowing a particular element to be identified by counting. For example, the term `child(3)` identifies the third child of a node. The second argument, if it is present, is an element type name, and restricts the counting to elements of that type: `child(3, P)` identifies the third child that is a `P` element. The special value `#element` can be used to mean 'any element': `child(3, #element)` is thus the same as `child(3)`. The remaining two arguments are an attribute name and value, and restrict matters further to elements

[16] Since XML allows you to redefine almost everything, the situation is actually slightly more complicated, but most of the time this will be the case.

with the specified attribute having the given value: child(3, P, lang, de) identifies the third paragraph child whose lang attribute has the value “de”.

Although this description has been slightly simplified, it should suffice to show you that the examples cited at the beginning of this section can easily be expressed as XPointers: ‘the second paragraph after the third level two header’ is root().child(3, H2).following(P,2) (assuming HTML element names) and ‘the third child of the second element after the element with id “orinoco” that has the same parent’ is id(orinoco).fsibling(2).child(3). (You have to read them backwards to get the English version, but navigation in the tree is specified from left to right.)

So far, XPointers have enabled us to specify single elements in a document. Another keyword, span, allows them to specify regions that span more than one element. It takes two XPointers as arguments, with the effect of identifying the region extending from the element identified by the first through that identified by the second. For example id(orinoco).span(child(3), child(6)) identifies the third through sixth child elements of the element identified as “orinoco”.

Finally, XPointers include a crude facility for identifying positions by text content: a term of the form string(*n, string*) identifies the n^{th} occurrence of the string *string*. For example, root().string(3, "fish") points to the third occurrence of the string “fish” in a document. (Additional embellishments, which will not be described here, allow you to refine string terms to point to, for example, the three characters following a string, and so on.)

XPointers extend the power of links by providing a flexible means of locating elements within documents. XLink extends their power in a different direction, by generalizing the notion of what a link is. It does this in a way that may be familiar from mathematics. A link is a connection or association between two locations. In the abstract, we can consider the ordered pair comprising the link’s source and destination to *be* the link — nothing extra is really required. If we want bi-directional links, we only need to forget the ordering of the pair, so neither location is distinguished as either source or destination, but they remain connected. Once you have got this far, allowing as many locations as you want to be part of a link is a relatively small next step — in the abstract world. In practice, it raises some thorny implementation issues.

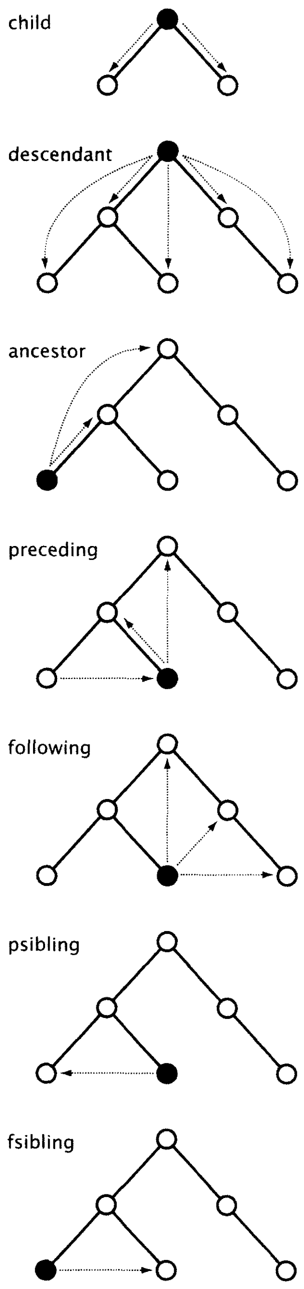


Figure 9.6
Relationships within a document’s tree structure

XLink provides two sorts of link: *simple* and *extended*. Simple links are almost the same as HTML links and will not be described here. Extended links correspond to the generalization described in the previous paragraph. Syntactically, an extended link element needs to specify several locators (URLs or XPointers); since an element cannot have more than one attribute with the same name (such as href), XLink proposes separate element types to denote the presence of an extended link and the locations being linked, the latter appearing in the content of the former. If we assume that a DTD defines element types exlink and locator (leaving aside the question of how these are to be identified as link and locator elements), an extended link might look like this:[17]

```
<exlink>
  <locator link = "../text.xml"/>
  <locator link = "../graphics.xml"/>
  <locator link = "../video.xml"/>
  <locator link = "../animation.xml"/>
  <locator link = "../sound.xml"/>
</exlink>
```

This is all very well — an extended link is a collection of linked resources, as required — but where should elements such as exlink appear? In one of the linked documents? In all of them? In a separate file? The XLink specification is rather vague on the matter. It provides for the possibility that a link might be inline or out-of-line. Technically, an inline link is one that includes itself as one of the linked objects; practically, this means that it appears in one of the documents being linked. An out-of-line link does not, so typically out-of-line links will be stored in separate linking files, devoted to the links connecting a group of documents. This begs the question of how out-of-line links are to be located by user agents.

For the common case where several documents can be considered as a group, with a collection of out-of-line links connecting parts of them in different ways, the links can be stored in an *extended link group* element, and a specialized locator called an *extended link document* element can be placed in each of the documents in the group to point to the links.

The idea of extended links is not new, but previously its implementation has been confined to hypertext systems on a smaller scale than the World Wide Web. It remains to be seen whether succesful implementations can be constructed for the Web, and whether extended links will be considered a worthwhile extension by Web users.

[17] In XML the tags for empty elements end with />, as in the locator elements here.

Navigation and Structure in Hypertext

The possibility of disorientation — being 'lost in hyperspace' as the cliché happily puts it — has been anticipated since the earliest experiments with hypertext. The division of text into nodes leads to fragmentation and a disruption of the context supplied by conventional linear structures; the multiple branching paths provided by links, while offering ample opportunity to browse endlessly, can resemble a maze to a user searching for one specific piece of information.

Much early thinking about this problem concentrated on the provision of graphical views[18] of the link structure between a collection of nodes. This approach has many problems — for any non-trivial collection of nodes and links the diagram itself becomes a confusing tangle. It is difficult to provide enough information about a node's content for users to determine whether it is interesting, or remind them whether they have visited it before — and, despite various ingenious embellishments, such as the use of 'fish-eye' views, showing nodes close to the current one in more detail than those further away, these graphical navigation aids have been largely abandoned in favour of navigation based on browsing history, indexing and searching.

Often, browsing in hypertext entails following side-tracks from a linear sequence and then returning to pursue it further. Providing a 'go back' command is a natural way of supporting this mode of reading. To do so requires the browser to maintain a stack of recently visited nodes, so it is straightforward to extend the facility by making this stack visible to the user in the form of a *history list*, and allowing users to jump directly to any node on the list, going back an arbitrary number of steps. A simple 'go back' facility provides a form of backtracking whereby a path is followed through the hypertext until it is no longer fruitful, and then the user backs up to a point where there was a different choice of link to follow and tries another path. In an alternative way of working, direct transfers to nodes on the history list allow a collection of recently visited nodes to be treated as a cluster, with arbitrary jumps being made between them. In effect, the history list provides an *ad hoc* set of dynamically constructed links.

History lists are always of limited size. Even if they were not, a complete history of your browsing activity will soon become

[18] Often confusingly called 'browsers' in the literature of the time.

as confusing to find your way around as the hypertext network itself. It is more useful to summarize this activity over a period of time in the form of a collection of pointers to previously visited interesting nodes. On the World Wide Web, pointers are URLs, and, since these are just text strings, maintaining a collection of *bookmarks*, as URLs of interesting sites are usually called[19] is simple. Typically, Web browsers allow you to bookmark a page when you visit it, recording its URL together with the page title (this can be replaced by a mnemonic name subsequently, if the title is not sufficiently meaningful); bookmarks can be organized into a hierarchy and sorted. A menu of bookmarks is provided by the browser, which shows their names; selecting a bookmark from this menu causes the browser to retrieve the corresponding page. Instead of following trails of links, users can construct a collection of bookmarks that enables them to go directly to their chosen sites. This way of working is extremely common: you will notice that it entails rejecting the browsing approach to finding information in favour of direct access to a relatively samll number of familiar sites.

Browsing is based on looking for information by its associations. An alternative, and often more natural procedure, is to look for information on the basis of its content. This approach is closer to information retrieval and database querying; it can serve as a way of finding a starting point for browsing as well as being an alternative to it. In a small hypertext network, stored on a single computer, it makes sense to search through the whole network for a keyword or phrase by applying a conventional text searching algorithm to each of the stored nodes. For larger networks, and especially for distributed networks such as the World Wide Web, this approach is not practical. Instead, we would like to construct an index (which can be kept on a single Web site), consisting of keywords together with URLs of nodes that they describe, and search that. Such an index could also be organized into categories using a suitable classification scheme, so that it can serve as a subject index, rather like a library's classified catalogue.

Many small indexes have been constructed simply by users publishing their bookmarks, sometimes supplying a simple CGI script to search through them. Where these form a coherent set of pages on a single topic, for example MPEG video, the result can be a useful access point for specialized browsing on that topic. Wider coverage is available from commercial index sites, such as Yahoo! and AltaVista. Here, huge numbers of URLs have been

[19] Internet Explorer prefers to call them 'favourites'.

collected, classified, and provided with a powerful search engine that supports the full range of Boolean query operations normally found in information retrieval systems.

There are two broad approaches to constructing index sites: manual and automatic. Using the manual approach, sites are classified on the basis of people's evaluation of their content. A URL for a Web site is normally added to the index in response to a request from its author, who suggests an initial position in the classification — usually a hierarchy of topics — and supplies a brief description of the contents. The site is then visited, and the classification is refined if necessary. The description is kept along with the URL, so it is possible to find a site either by navigating through the classification hierarchy, or by searching through the descriptions for keywords.

> ⇨ The search facility helps to compensate for the limitations of hierarchical classifications, which are well known to information scientists and will be familiar if you have ever used a hierarchical library catalogue: a site cannot be in more than one place in the hierarchy[20] even though there is often good reason to classify it in more than one way. For example, should a Web site devoted to the use of MPEG-2 video for broadcast television be classified under television, which may be a sub-category of entertainment, or under digital video, a sub-category of multimedia within the computing category? Evidently, a great deal depends on the structure of the hierarchy. Most classifications used by Web index sites are heavily slanted towards computing, communications and physical sciences, reflecting the origins of the World Wide Web. For example, Yahoo!'s top-level categories include one devoted entirely to computers and Internet, while another is devoted to everything concerning business and economy.[21] Such imbalances may be alleviated by replacing the simple hierarchical classification by one of the more sophisticated schemes developed by information scientists, or by changing the structure of the classification as the content of the material being indexed changes. Whereas libraries' classification schemes tend to ossify, because of their relationship to the physical books and their shelving, a Web index, which exists and deals with objects that exist only on computers, can be changed much more easily.

While the manual approach to constructing indexes has the considerable advantage that human intelligence is applied to classifying and filtering the entries, it is labour-intensive — to be done properly, it requires somebody to actually go and visit each Web site submitted for indexing — which makes it difficult for the indexes to keep up with the rapid pace of change of the World

[20] Although most indexes support the use of aliases that can make it appear almost as if it were.

[21] And computer science is a sub-category of science.

Wide Web. Automatically constructed indexes avoid this problem. They are built by programs variously known as *robots, spiders* or *Web crawlers*, which simply follow links, collecting URLs and keywords from the pages they encounter. The keywords from a page are then used as the basis for its classification, and for searching. The success of this approach depends on the robot's success in extracting meaningful keywords, on the heuristics used to classify sites on the basis of keywords, and on the efficiency and sophistication of the search engine used to query the database of keywords and URLs.

AltaVista is the oldest and probably still the largest of the automatically constructed indexes. Actually, construction is not entirely automatic: for a Web site to be included, the URL of its root page must be explicitly submitted to Alta Vista. Subsequently, the Alta Vista spider[22] will retrieve that page and, by following links, all the other pages on the site. It then indexes all the text on those pages. A highly efficient search engine running on some very powerful computers is used to process queries against the database of indexed words. When a user submits a query containing any of the words found on a page, the first few lines of that page, together with its URL, are displayed among the query results.

This brute force approach to indexing can produce some idiosyncratic results, as you will probably know if you have ever used AltaVista. The spider cannot in any sense understand natural language, or differentiate between different uses of a word in different contexts, so, for example a search for 'the blues' will find Web pages devoted to Birmingham City Football Club and the St. Louis Blues NHL hockey team as well those dealing with blues music.[23] It is also possible, though, that the actual text of a page may not include the words one would normally use to describe its subject: that may be implicit in the content, or the author may have used a synonym. (For example, the paragraph preceding this one does not use the word 'robot', only 'spider'.) Furthermore, the opening lines of text from a Web page do not always provide a very helpful indication of its content. The current state of the art in natural language understanding by software does not provide a practical way of improving matters. Instead, the problem has to be approached from the other end by providing a means of adding a description that can be easily processed by a simple program to a Web page.

[22] Actually, they call it a Superspider.

[23] You also get an alarming number of sites devoted to the Moody Blues, but it wouldn't be fair to blame the search engines for that.

HTML uses the META element for this purpose. META is a general mechanism for specifying *metadata* — data about data; in this case, data about a page, as against data that is part of the content of a page. A description for the benefit of indexing software is just one useful example of metadata. (PICS ratings are another.) META is an empty element that can only appear in the head of an HTML document. Two attributes, name and content, are used to allow it to specify arbitrary metadata as a pair of values, the name serving to identify the type of information, the content providing the corresponding value. This scheme avoids a proliferation of elements and leaves the maximum flexibility for choosing properties to use to describe a document.

➲ The inevitable drawback that accompanies this flexibility is that metadata lacks any structure. The proposed *Resource Description Format (RDF)* is intended to address this drawback, by incorporating ideas from conventional database modelling and object-oriented technology to provide an architecture for metadata that retains the flexibility of the simple META element approach, but allows metadata to be organized more formally.

AltaVista suggest the use of two named properties to assist their spider: description and keywords, whose intent is clear from the chosen names. The description, if present, is used instead of the first lines of the page when the results of a query are displayed. The keywords provide additional index terms, which may not be present in the text itself. Thus, if a Web page was devoted to the life and times of the blues singer Robert Johnson, it could usefully include the following:

```
<META name="description"
      content="The life and times of the
               blues singer Robert Johnson">
<META  name="keywords" content="blues, music,
                     King of the Delta Blues">
```

➲ Instead of the name attribute, a META element may include an hhtp-equiv attribute, which causes the corresponding content value to be incorporated into HTTP responses as a header. For example, PICS data can be incorporated into an HTML document using a META tag with its http-equiv attribute set to PICS-Label. This produces a Pics-Label header in an HTTP response for that page, with the value of the META element's content attribute as data. This way, metadata can be retrieved using an HTTP HEAD request, so the page itself does not have to be downloaded.

⇨ Sites that provide an index to the World Wide Web are naturally popular, since they provide a starting point for browsing as well as direct access to specific information one may be looking for. By adding additional services, such as a news feed and on-line shopping, to the basic indexing functions, sites such as Yahoo! and AltaVista are evolving into *portals*: ways in to the tangle of the rest of the Web that, for some users, may remove the need for browsing entirely. As a result, they are among the few Web sites that are guaranteed to be visited many times, and hence they are attractive to advertisers. They are becoming seen as an answer to the question that has vexed business since the commercialization of the World Wide Web: how can you actually make money out of it?

If, as we suggested at the beginning of this section, disorientation in hypertext is due, at least partly, to the loss of contextual structure that results from the fragmentation of text into nodes, then one solution must lie in the development of new contextual structures appropriate to hypertext. In this connection, the concept of a *home* node has been used in many hypertext systems. For a Web site, it denotes the first page that is normally retrieved from that site; generally, it denotes a similar entry point to a collection of linked documents. Most nodes in the collection have a link back to the home node, so that it is always possible to return to a known point in the network.

Very often, a Web site or hypertext document collection has a hierarchical structure, usually with cross-links. The major divisions of the hierarchy provide an organizational and navigational framework. Within such an organization, the home node can be seen as the root of a tree of nodes; it will contain links to the roots of the sub-trees containing nodes belonging to the major sub-divisions of the site. These in turn may contain links to nodes at a lower level in the hierarchy. A frame-based display, such as we saw earlier in this chapter, can be used to provide navigation within a hierarchy: a navigational frame containing links to the roots of all the major sub-trees and the home node can be continuously displayed so that, at any time, a user can move between divisions. In a more sophisticated variation, the navigational frame can change so that, as well as the top-level links, it always contains links to important nodes within the sub-tree currently being explored. In the absence of frames, each node must explicitly contain links to the major divisions (again, a form of organization that should be familiar). This is less satisfactory, because of the duplication of link information and the consequent problems of maintaining consistency when changes are made to the structure of the hypertext network.

Hierarchical structures do not always make sense, and sometimes a simple sequential structure is preferable. Here, every node needs a link to the previous and next one in the sequence, and possibly a link to the start of the sequence. Often a hybrid structure with sequential and hierarchical components will fit the material best.

On a scale larger than a single Web site or document collection, few structures have been developed. The index sites previously described serve, in some ways, as higher level home nodes, but it is not clear how navigational assistance, analogous to navigation frames, can be provided on this scale. Some *ad hoc* structures that make it easier to find one's way around a collection of Web sites have been developed, most notably, the 'Web ring', where a set of site maintainers agree to add links to their sites so that they form a circular sequence: once you have arrived at one of them, you will be led around the whole ring.

The structures just described possess little of the richness of long-established textual strategies used by writers to organize their material. The complexity and flexibility of language offers possibilities that an essentially crude system of nodes and links cannot approach. Despite the claims that have been made for hypertext's non-linearity freeing readers from the limitations of conventional text, in the end, hypertextual structures are far more restricted. The usefulness of hypertext (and hypermedia) lies in the way it allows us to add connections, not remove linear restrictions.

Further Information

Although it is dated, [Con87] remains a useful survey of hypertext issues and early systems. [BLFM98] contains the formal definition of a URL; the HTML references given in Chapter 8 describe how they are used. We will describe the mechanisms underlying World Wide Web links in Chapter 15. The specifications for XPointer and XLink can be traced from [XML].

Exercises

1. Show how you can simulate bi-directional links and multi-links with simple uni-directional links. What advantages are there in using the more general forms of link instead of simulating them in this way?

2. In PDF, the source and destination of a link are arbitrary rectangular areas that may contain some text. Why cannot HTML use the same model for anchors?

3. Regional links allow for the possibility of links whose sources overlap. How would you cope with the problems this causes?

4. In a frame-based Web site, why might a target other than `_top` be used for links which point beyond the site?

5. According to Figure 9.6, a node's parent both precedes and follows it. Explain how this can be so.

6. Assuming HTML tags and attributes, write down XPointers for the following:

   (a) The third level 1 header in a document.

   (b) The second paragraph after the anchor with name `References`

   (c) The third anchor element in the fourth paragraph following the second level 2 header.

   (d) The first span element in the same paragraph as the anchor with name `References`.

7. Suggest a way of displaying the structure of a Web site that would provide assistance with navigation, assuming the site has (a) 10, (b) 100 and (c) 1000 pages.

8. Devise a classification scheme for Web pages that avoids the shortcomings of hierarchical schemes identified in this chapter.

9. Give examples showing how metadata could be used to assist search engines in avoiding (a) false negatives and (b) false positives.

10. The *Encyclopedia Britannica* and several good dictionaries are now available online, so, in principle, every term used in any Web page could be linked to its definition in one of these reference works. Would this be (a) practical and (b) useful?

10 Video

All current methods of displaying moving pictures, whether they are on film or video, broadcast to a television, or stored on a computer system or digital storage medium, depend on the phenomenon known as *persistence of vision* — a lag in the eye's response to visual stimuli which results in 'after-images' — for their effect. Because of persistence of vision, if a sequence of still images is presented to our eyes at a sufficiently high rate, above what is called the *fusion frequency*, we experience a continuous visual sensation rather than perceiving the individual images. If consecutive images only differ by a small amount, any changes from one to the next will be perceived as movement of elements within the images. The fusion frequency depends on the brightness of the image relative to the viewing environment, but is around 40 images per second. Below this frequency, a flickering effect will be perceived, which becomes more pronounced as the rate at which successive images are presented is decreased until, eventually, all illusion of motion is lost and we see the sequence of still images for what it really is.

⇨ Contemporary film is shot at a rate of 24 frames per second, but film projectors are equipped with a device that interrupts the projection, effectively displaying each frame twice, so that the projected frame rate is 48. In countries that use the NTSC system for television and video, frames are displayed at a rate of roughly 30 per second, but again the rate at which the picture is refreshed is effectively doubled, because each frame is broken into two interlaced halves, known as fields, which are displayed one after the other (see below). The PAL and SECAM systems work in a similar manner, at a rate of 50 fields (25 frames) per second. When we are displaying images on a computer monitor, we can choose the rate at which they are displayed. The relatively high refresh rate of the monitor avoids flicker, so digital display does not require interlaced fields, but images must still be displayed rapidly enough to produce the illusion of motion. A rate of 12–15 frames per second is just about sufficient, although higher rates are to be preferred.

There are two different ways of generating moving pictures in a digital form for inclusion in a multimedia production. We can use a video camera to capture a sequence of frames recording actual motion as it is occurring in the real world, or we can create each frame individually, either within the computer or by capturing single images one at a time. We use the word *video* for the first option, and *animation* for the second. Since both video and animation are forms of moving pictures they share some features, but there are also technical differences between them. For live-action video it is necessary both to record images fast enough to achieve a convincing representation of real-time motion, and to interface to equipment that conforms to requirements and standards which were defined for broadcast television — even though, when we come to play back the video on a computer, these requirements and standards are largely irrelevant. To make matters worse, there are several different standards in use in different parts of the world.

As we will see, video places considerable strains on the current processing, storage and data transmission capabilities of computer systems. Video is also much in demand from consumers, whose expectations are typically based on their experience of broadcast television; it is probably the area of multimedia which is presently most subject to technical change. Typically, video targetted at consumer equipment will be played back at reduced frame rates, possibly even with jittering from dropped frames, in windows much smaller than the smallest domestic television set, and it will often exhibit visible compression artefacts. In order to accommodate the limitations of low-end PCs, considerable compromises over quality must be made. Video should therefore be used judiciously in multimedia productions intended for low-end platforms. Consumers who are used to broadcast digital TV will not be impressed by what has been disparagingly, but not unjustifiably, described as 'dancing postage stamps'. Good video production for general consumption

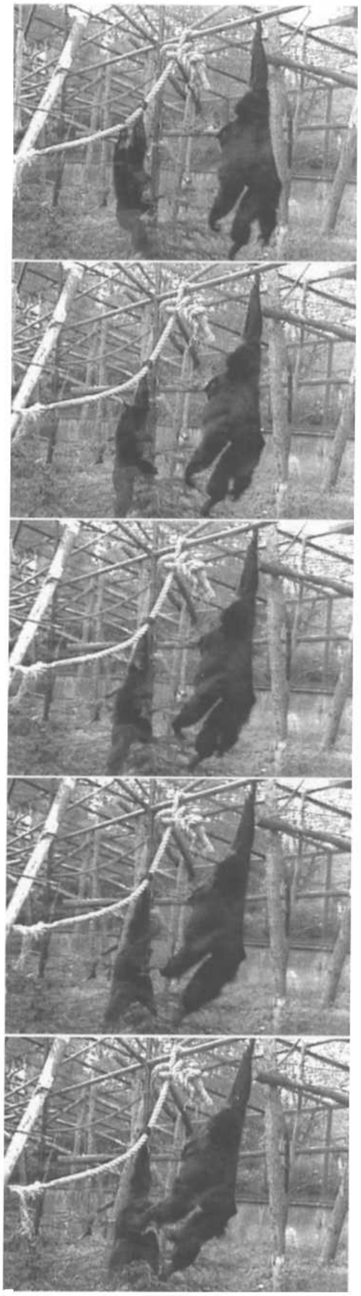


Figure 10.1
A sequence of video frames

therefore requires both a careful choice of material and a mode of presentation that does not draw attention to its defects.

Digitizing Video

The one thing that you need to keep constantly in mind when considering digital video is its size. A video sequence consists of a number of frames, each of which is a single image produced by digitizing the time-varying signal generated by the sensors in a video camera. We invariably use bitmapped images for video frames (because that is what video equipment generates). The size of the image produced for each frame of NTSC video, as used in North America and Japan, is 640 pixels wide by 480 pixels high. If we use 24-bit colour, each frame occupies 640 × 480 × 3 bytes, which is 900 kilobytes. One second of uncompressed NTSC video comprises almost exactly 30 frames, so each second occupies just over 26 megabytes, each minute about 1.6 gigabytes. The figures for the PAL system used in Western Europe and Australia are slightly higher: 768 × 576 at 25 frames per second gives 31 megabytes per second or 1.85 gigabytes for each minute. At these rates, you could not get very much video on to a CD-ROM or DVD; nor could you transmit it over any but the fastest network — certainly not the Internet as we know it. By using a good sized disk array, you would be able to store an entire feature film, and the required data transfer rate is within the reach of the latest SCSI standards. Storing video in such a form is therefore a feasible option for film and TV studios, but not (yet) for the domestic consumer.

Notice that the need for very high rates of data transfer arises from the fact that these are *moving* pictures, and we must deliver them one after another, fast enough for them to be perceived as a representation of continuous motion. The high volume of each frame arises from the storage requirements of bitmapped images. As we saw in Chapter 5, we can reduce this requirement by applying some form of compression to our images. It should come as no surprise, therefore, to learn that we also apply compression to video. For transmission over a relatively slow network such as the Internet, or for playback directly from CD-ROM, we must apply severe compression, as well as using other methods, such as limiting the frame size, to reduce the volume of data. This results in a significant loss of quality, despite the claims that are sometimes made to the contrary. For capturing video in real-time, this compression must be carried out so fast that dedicated hardware is needed. In the remainder of this discussion, we will ignore the possibility of working with uncompressed video, since this is only feasible on very high-end equipment at the present time.

Digital video may either be captured directly from a camera, or indirectly from a video tape recorder (VTR) or (leaving aside questions of legality) from a broadcast signal. Current technology offers two sites for performing the digitization and compression: in the computer, or in the camera. If a VTR intervenes, or a signal is broadcast, the nature of the signal is not changed by the recording or broadcasting process — digital VTRs record digital signals, analogue VTRs record analogue signals.

In the case of digitization being performed in the computer, an analogue video signal, conforming to some broadcast video standard, is fed to the input of a *video capture card* attached to the computer. Within this card, the analogue signal is converted to digital form. Usually, the resulting digital data is compressed in the card, before being passed on for storage on disk or transmission over a network, although sometimes it may be possible to perform the compression using software running on the computer's central processor.

The alternative option is to digitize and compress the video signal using circuitry inside a camera. The purely digital signal (now better described as a data stream) is then passed to the computer via a high speed interface. The *IEEE 1394* interface, often called by its more colourful code name *FireWire*, is commonly used for this purpose, although low quality cameras, typically used for video conferencing, may use a USB connection.

⇨ The expression 'digital video' has acquired several different meanings. It is sometimes used in a general sense to refer to the storage and manipulation of video data in a digital form, but it is also used more specifically to refer to the technology just described, which performs digitization in camera. An emerging consumer-level technology for digital video in this second sense is based on the *DV* standard, where DV stands for... digital video, as does (or did originally) the DV in DVD. We will usually use the term in its general sense; context will indicate any occasions on which it is used in some more restricted sense.

Digitization in the camera has one great advantage over digitization in the computer. When an analogue signal is transmitted over a cable, even over a short distance, it will inevitably be corrupted, if only to a small extent, by noise. Noise will also creep in when analogue data is stored on magnetic tape. *Composite* video signals, the type normally used in domestic equipment, are also subject to distortion caused by interference between the colour and brightness information, especially when stored on VHS tape. We have all become used to noise and distortion in broadcast television and VHS video, and don't usually consider it objectionable, but these phenomena can reduce the effectiveness of compression. They typically produce small random fluctuations in colour, so that, for example, areas that should be composed of a single colour, and would compress well, will lose their coherence and compress less well. Some video compression techniques are based on the similarity between adjacent frames. Again, the random nature of any signal degradation will undermine this similarity, making compression less effective, and the resulting picture less satisfactory. If digitization is carried out inside the camera, only digital signals, which, as we pointed out in Chapter 2, are resistant to corruption by noise and interference, are transmitted down cables and stored on tape. Compression is applied to a clean signal at its source and so it will be more effective.

The disadvantage of digitization performed in the camera is that the user has no control over it. The data stream produced by a digital video camera must conform to an appropriate standard, such as the DV standard for domestic and semi-professional camcorders, which stipulate the data rate for the data stream, and thus the amount of compression to be applied. Analogue video capture boards and their associated software normally offer the user control over the compression parameters, allowing a trade-off to be made between picture quality and data rate (and hence file size).

Video capture boards can usually not only perform digitization and compression, but also their inverse, decompression and digital to analogue conversion. Devices which compress and decompress signals are known as compressor/decompressors, invariably contracted to *codecs*. Using a hardware codec, it is possible to capture video signals, store them on a computer, and then play them back at full motion to an external video monitor (i.e. a TV set) attached to the video card's output. Although playback to the computer screen is not impossible, most hardware codecs cannot provide full screen full motion video to the monitor. More importantly for multimedia producers, we cannot know whether our audience will have any hardware codec available, or if so, which one. So, we need a *software codec*: a program that performs the same function as the dedicated hardware codecs we have described, in order to ensure that the audience will be able to play back the video on a computer's ordinary monitor. Generally, software is slower than dedicated hardware, so playing full-screen video at broadcast frame rates using a software codec is only possible on the fastest of desktop computers, and we have to sacrifice many aspects of quality in video that is intended for playback in this way. Generally, compression algorithms that are suitable for software codecs are different from those suitable for hardware implementation, so we will often have to recompress captured video material — resulting in a type of generational loss of quality — in order to prepare it for incorporation in a multimedia production.

Video Standards

Analogue Broadcast Standards

There are three sets of standards in use for analogue broadcast colour television. The oldest of these is *NTSC*, named after the (US) National Television Systems Committee, which designed it. It is used in North America, Japan, Taiwan and parts of the Caribbean and of South America. In most of Western Europe a standard known as *PAL*, which stands for Phase Alternating Line, referring to the way the signal is encoded, is used, but in France *SECAM* (*Séquential Couleur avec Mémoire*, a similar reference to the signal encoding) is preferred. PAL is also used in Australia and New Zealand and in China, while SECAM is used extensively in the former Soviet Union and in Eastern Europe. The standards adopted in Africa and Asia for the most part follow the pattern of European colonial history. The situation in South America is somewhat confused, with NTSC and local variations of PAL being used in different countries there.

The NTSC, PAL and SECAM standards are concerned with technical details of the way colour television pictures are encoded as broadcast signals, but their names are used loosely to refer to other characteristics associated with them, in particular the frame rate and the number of lines in each frame. To appreciate what these figures refer to, it is necessary to understand how television pictures are displayed.

Like computer monitors, television sets work on a raster scanning principle. Conceptually, the screen is divided into horizontal lines, like the lines of text on a page. In a CRT (cathode ray tube) set, three electron beams, one for each additive primary colour, are emitted and deflected by a magnetic field so that they sweep across the screen, tracing one line, then moving down to trace the next, and so on. Their intensity is modified according to the incoming signal so that the phosphor dots emit an appropriate amount of light when electrons hit them. The picture you see is thus built up from top to bottom as a sequence of horizontal lines. (You can see the lines if you look closely at a large TV screen.) Once again, persistence of vision comes into play, to make this series of lines appear as a single picture.

As we remarked above, the screen must be refreshed about 40 times a second if flickering is to be avoided. Transmitting an entire picture that many times a second requires an amount of bandwidth that was considered impractical at the time the standards were being developed. Instead, each frame is divided into two *fields*, one consisting of the odd-numbered lines of each frame, the other of the even lines. These are transmitted one after the other, so that each frame (still picture) is built up by *interlacing* the fields. The fields are variously known as odd and even, upper and lower, and field 1 and field 2. Originally, the rate at which fields were transmitted was chosen to match the local AC line frequency, so in Western Europe, a field rate of 50 per second, and hence a frame rate of 25 per second, is used for PAL; in North America a field rate of 60 per second was used for black and white transmission, but when a colour signal was added for NTSC, it was found to cause interference with the sound, so the field rate was multiplied by a factor of 1000/1001, giving 59.94 fields per second. Although the NTSC frame rate is often quoted as 30 frames per second, it is actually 29.97.

When video is played back on a computer monitor, it is not generally interlaced. Instead, the lines of each frame are written to a frame buffer from top to bottom, in the obvious way. This is known as *progressive scanning*. Since the whole screen is refreshed from the frame buffer at a high rate, flickering does not occur, and in fact much lower frame rates can be used than those necessary for broadcast. Fields may be combined into frames when an analogue video signal is digitized, or they may be stored separately and only combined when the material is played back. Since fields are actually separated in time, combining them in this way can cause undesirable effects. If an object is moving rapidly, it may change position between the two fields. Figure 10.2 shows an extreme example of a flash of light on the surface of water. When the fields are combined into a single frame for progressive display, there are discontinuities, since adjacent lines come from different fields. Even where the movement is less rapid, the same effect will occur in a less extreme form: the edges of moving objects will have a comb-like appearance where they are displaced between fields, as shown in Figure 10.3. To prevent this effect, it may be necessary to 'de-interlace' by averaging the two fields when constructing a single frame; this, however, is a relatively poor compromise. Another option is to discard half the fields — say all the even fields — and to interpolate the missing information for the fields that remain, to give full frames. This also gives predictably poor results. On the other hand, the fact that there are 60 or 50 fields per second in analogue video can sometimes usefully be exploited by converting each field into a single frame, thus giving two seconds of slower but fluid video for each second captured. If the resulting video is either played at full speed (30 or 25 fps), or at a small size, the loss of image quality resulting from the interpolation necessary to create frames from fields will scarcely be noticeable.

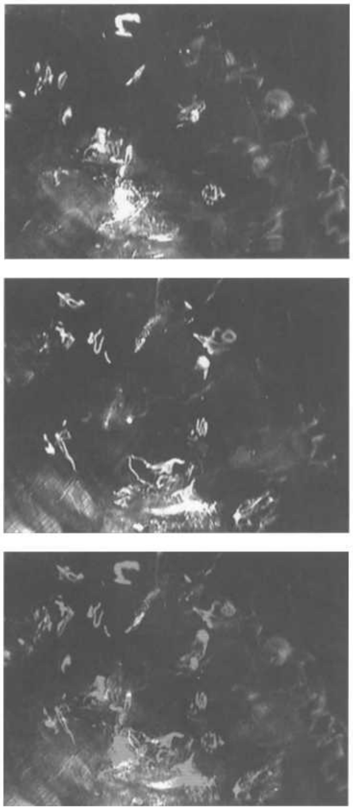


Figure 10.2
Fields and interlaced frame with very rapid change

Each broadcast standard defines a pattern of signals to indicate the start of each line, and a way of encoding the picture information itself within the line. In addition to the lines we can see on the picture, some extra lines are transmitted in each frame, containing synchronization and other information. An NTSC frame contains 525 lines, of which 480 are picture; PAL and SECAM use 625 lines, of which 576 are picture. It is common to quote the number of lines and the field rate together to characterize a particular scanning standard; what we usually call NTSC, for example, is 525/59.94.

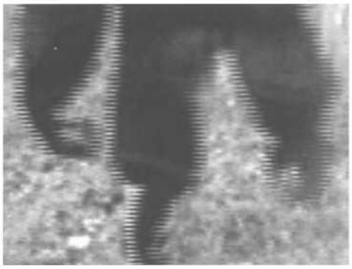


Figure 10.3
Interlacing artefacts

⇨ It is possible that you might need to digitize material that was originally made on film and has been transferred to video tape. This would be the case if you were making a multimedia film guide, for example. Most film footage is projected at 24 frames per second so there is a mismatch with all the video standards. In order to fit 24 film frames into (nearly) 30 NTSC video frames, a stratagem known as *3-2 pulldown* is employed. The first film frame is recorded for the first three video fields, the second for two, the third for three again, and so on. If you are starting with material in this form, it is best to remove the 3-2 pulldown after it has been digitized (a straightforward operation with professional video editing software) and revert to the original frame rate of 24 per second. Using PAL, films are simply shown slightly too fast, so it is sufficient to adjust the frame rate.[1]

In production and post-production studios, analogue video is represented in *component* form: three separate signals, representing luminance and two colour difference values, are used. As we explained on page 172, the components are said to represent picture information in *YUV colour*. For transmission, the three components are combined into a single *composite* signal, with Y, U and V being transmitted on a single carrier.[2] Each colour difference component (U and V) is allocated half as much bandwidth as the luminance (Y), a form of analogue data compression, which is justified by the empirical observation that human eyes are less sensitive to variations in colour than to variations in brightness. As we remarked above, there is inevitably some interference between the components, especially when a composite signal is stored on tape, although for capture direct from a good camera a composite signal will be perfectly adequate for anything other than broadcast quality productions. A compromise between component video and composite video separates the luminance from the two colour difference (which remain combined into one *chrominance* signal). This representation is employed in high-level consumer and semi-professional video equipment that uses the S-VHS and Hi-8 tape formats; the term *S-video* is used to describe this form of signal. Despite being a compromise solution, the significantly better quality of the video tapes available for recording this signal means that S-video is very widely used for sourcing video footage for low-cost multimedia production; it is also frequently used when 'amateur' footage is to be shown on television, as in public access programming.

In practical terms, if you are capturing analogue video to use in a multimedia production, it is necessary to ensure that your capture board is compatible with the signals produced by your camera or VTR. Composite and S-video inputs and outputs are found on most capture cards, but only more expensive professional equipment can handle component signals.

[1] 'Made for TV' movies are usually shot at the prevailing TV frame rate.

[2] Actually Y′, U and V.

Digital Video Standards

The standards situation for digital video is no less complex than that for analogue video. This is inevitable, because of the need for backward compatibility with existing equipment — the use of a digital data stream instead of an analogue signal is orthogonal to scanning formats and field rates, so digital video formats must be capable of representing both 625/50 and 525/59.94. The emerging HDTV standards should also be accommodated. Some attempt has been made to unify the two current formats, but unfortunately, different digital standards for consumer use and for professional use and transmission have been adopted.

Like any analogue data, video must be sampled to be converted into a digital form. A standard officially entitled *Rec. ITU-R BT.601*, but more often referred to as *CCIR 601*[3] defines sampling of digital video. Since a video frame is two-dimensional, it must be sampled in both directions. The scan lines provide an obvious vertical arrangement; CCIR 601 defines a horizontal sampling picture format consisting of 720 luminance samples and two sets of 360 colour difference samples per line, irrespective of the scanning standard. Thus, ignoring chrominance and interlacing for a moment, an NTSC frame sampled according to CCIR 601 will consist of 720×480 pixels, while a PAL frame will consist of 720×576 pixels.

Observant readers will find this perplexing, in view of our earlier statement that the sizes of PAL and NTSC frames are 768×576 and 640×480 pixels, respectively, so it is necessary to clarify the situation. PAL and NTSC are analogue standards. Frames are divided vertically into lines, but each line is generated by a continuous signal, it is not really broken into pixels in the way that a digital image is. The value for the number of pixels in a line is produced by taking the number of image lines and multiplying it by the *aspect ratio* (the ratio of width to height) of the frame. This aspect ratio is 4:3 in both PAL and NTSC systems, which gives the sizes originally quoted. Video capture cards which digitize analogue signals typically produce frames in the form of bitmaps with these dimensions. The assumption underlying the calculation is that pixels are square. By relaxing this assumption, CCIR 601 is able to specify a sampling rate that is identical for both systems, and has desirable properties such as providing the same number

[3] CCIR was the old name of the organization now known as ITU-R.

of horizontal samples in every line. CCIR 601 pixels, then, are not square: for 625 line systems, they are slightly wider than high, for 525 line systems they are slightly higher than wide. Equipment displaying video that has been sampled according to CCIR 601 must be set up to use pixels of the appropriate shape.

⇨ We have simplified somewhat. There are actually 858 (NTSC) or 864 (PAL) samples per line, but only some of these (the 'active samples') constitute part of the picture. While digital video editing software generally works with 720 pixels per line, one expert on digital video has made the following remark (with reference to 525/59.94 scanning):

> "[...]the debate over the true number of active samples is the trigger for many hair-pulling cat-fights at TV engineering seminars and conventions, so it is healthier to say that the number lies somewhere between 704 and 720. Likewise, only 480 lines out of the 525 lines contain active picture information. Again, the actual number is somewhere between 480 and 496."[4]

Like component analogue video, video sampled according to CCIR 601 consists of a luminance component and two colour difference components. The colour space is technically $Y'C_BC_R$ (see page 172) — you can consider the three components to be luminance Y, and the differences B−Y and R−Y. We described earlier how analogue video signals are compressed by allocating half as much bandwidth to each colour difference signal as to the luminance. In a similar way, and with the same justification, digital video is compressed by taking only half as many samples for each of the colour difference values as for luminance, a process known as *chrominance sub-sampling.* The arrangement of samples used in CCIR 601 is called *4:2:2* sampling; it is illustrated in Figure 10.4. In each line there are twice as many Y samples as there are samples of each of B−Y and R−Y. Since, for those pixels whose colour is sampled, all three values are sampled at the same point, the samples are said to be *co-sited.* The resulting data rate for CCIR 601 video is, using eight bits for each component, 166 Mbits (just over 20 Mbytes) per second.

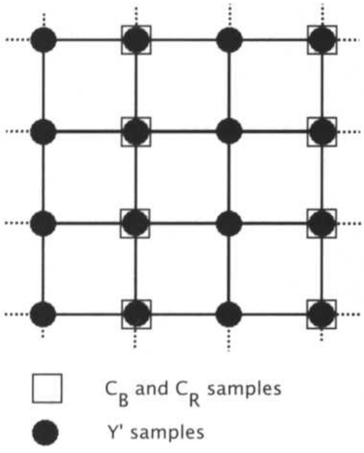


Figure 10.4
4:2:2 sampling

Why 4:2:2 and not 2:1:1? Because other sampling arrangements are possible. In particular, some standards for digital video employ either 4:1:1 sampling, where only every fourth pixel on each line is sampled for colour, or 4:2:0, where the colour values are sub-sampled by a factor of 2 in both the horizontal and vertical

[4] Chad Fogg, *The MPEG2 FAQ.*

directions[5] — a somewhat more complex process than it might at first appear, because of interlacing. (The notation 4:2:0 is inconsistent; it certainly does not mean that only one of the colour difference values is sampled.)

Sampling produces a digital representation of a video signal. This must be compressed and then formed into a data stream for transmission. Further standards are needed to specify the compression algorithm and the format of the data stream. Two separate standards are emerging, both based on $Y'C_BC_R$ components, scanned according to CCIR 601, but with further chrominance sub-sampling. Digital video equipment intended for consumer and semi-professional use (such as corporate training video production) and for news-gathering is based on the *DV* standard; studio equipment, digital broadcast TV and DVD are based on *MPEG-2*. Neither of these is a single standard, although the situation is not entirely chaotic. The main variations on DV — DVCAM and DVPRO — concern tape formats, and use the same compression algorithm and data stream as DV. There are, however, a high quality DVPRO, and a professional Digital-S format, which use 4:2:2 sampling, unlike DV, which uses 4:1:1. MPEG-2 was designed as a family of standards, organized into different *profiles* and *levels*. Each profile defines a subset of features of the MPEG-2 data stream; as we will see in the next section, MPEG-2 allows some scope for different ways of encoding the digital video data. Each level defines certain parameters, notably the maximum frame size and data rate, and chrominance sub-sampling. Each profile may be implemented at one or more of the levels, although not every combination of level and profile is defined. The most common combination is *Main Profile at Main Level (MP@ML)*, which uses CCIR 601 scanning with 4:2:0 chrominance sub-sampling; it supports a data rate of 15 Megabits per second and allows for the most elaborate representation of compressed data provided by MPEG-2. MP@ML is the format used for digital television broadcasts and for DVD video.

We should emphasize that the standards outlined in the preceding paragraph concern the data streams produced by equipment such as digital video cameras and VTRs. Once the data has entered a computer a whole new range of possibilities is opened up. Likewise, video data that has been digitized by a capture board from an analogue signal is unlikely to conform to either the DV or MPEG-2 standards.

[5] In 4:2:0 sub-sampling, the chroma and luminance samples are not co-sited. The chrominance samples are positioned halfway between lines, i.e. they are averaged between two lines.

Digital video formats are intimately bound up with compression, so before we can consider how video may be represented and manipulated inside a computer we must look at compression techniques that are applied to it.

Introduction to Video Compression

We have seen that video data is usually compressed when it is digitized. The form of compression applied by digital video cameras and video capture boards is usually optimized for playback through the same devices. When we want to use a video sequence as part of a multimedia production, we cannot rely on our users having any particular video hardware, so the only safe assumption is that the video is going to be played back using their computer's processor. Additionally, we have to assume that the video data must be delivered from a hard disk, at best, or from a CD-ROM or DVD, or even over a network connection. We will usually prepare video material using a software codec, to apply a form of compression that is more suited to the capabilities of the hardware that is likely to be used by consumers. Typically, therefore, our video data will need to be compressed twice: first during capture and then again when it is prepared for distribution. To appreciate the implications of this, we need to understand the general features of the compression algorithms employed at both of these stages.

All video compression algorithms operate on digitized video consisting of a sequence of bit-mapped images. There are two ways in which this sequence can be compressed: each individual image can be compressed in isolation, using the techniques introduced in Chapter 5, or sub-sequences of frames can be compressed by only storing the differences between them. These two techniques are usually called *spatial compression* and *temporal compression*, respectively, although the more accurate terms *intra-frame* and *inter-frame* compression are also used, especially in the context of MPEG. Naturally, spatial and temporal compression can be used together.

Since spatial compression is really just image compression applied to a sequence of images, it makes sense to distinguish between lossless and lossy methods — the distinction, and the trade-offs implied, are the same as they are for still images. Generally, lossless methods do not produce sufficiently high compression ratios to reduce video data to manageable proportions, except on synthetically generated material (such as we will consider in Chapter 11). However, although compressing and recompressing video usually leads to a deterioration in image quality, and should be avoided if possible, recompression is often unavoidable, since the compressors used for capture are not suitable for delivery for multimedia. Furthermore, for post-production work, such as the creation of special effects, or even fairly basic corrections to the footage, it is usually necessary to decompress the video so that changes can be made to the individual pixels of each frame. For this reason, it is wise, if you have sufficient disk space, to work with uncompressed video during the post-production phase. That is, once the footage has been captured and selected, decompress it and use uncompressed data while you edit and apply effects, only recompressing the finished product for delivery.

⇨ There is considerable irony in the fact that one of the most loudly voiced advantages of digital video (and audio) is that it suffers no 'generational loss', unlike analogue video, which inevitably degrades every time you copy a tape, as you must whenever you perform any editing, and when you prepare material for distribution. Digital data can be copied as many times as you like, without any loss of quality — provided all you are doing is making an exact copy. As soon as you start decompressing and recompressing between copies, something very much like generational loss, only uglier, happens after all. The irony is compounded by the fact that considerable effort is being expended on devising methods to prevent people making exact copies, for fear of widespread piracy of digital video.

The principle underlying temporal compression algorithms is simple to grasp. Certain frames in a sequence are designated as *key frames*. Often, key frames are specified to occur at regular intervals — every sixth frame, for example — which can be chosen when the compressor is invoked. These key frames are either left uncompressed, or more likely, only spatially compressed. Each of the frames between the key frames is replaced by a *difference frame*, which records only the differences between the frame originally in that position and either the most recent key frame or the preceding frame, depending on the sophistication of the decompressor. For many sequences, the differences will only affect a small part of the frame. Consider, for example, a typical 'talking head' shot, of a person's head and upper body against a background, such as you might see on a news report. Most of the time, only parts of the speaker's face will be moving; the background, and perhaps the speaker's upper body, will remain static, so the pixels corresponding to these elements will keep the same values from frame to frame. Therefore, each difference frame will have much less information in it than a complete frame. This information can be stored in much less space than is required for the complete frame.

You will notice that we have described these compression techniques in terms of frames. This is because we are normally going to be concerned with video intended for progressively scanned playback on a computer. However, the techniques described can be equally well applied to fields of interlaced video; while this is somewhat more complex, it is conceptually no different.

Compression and decompression of a piece of video need not take the same time. If they do, the codec is said to be *symmetrical*, otherwise it is *asymmetrical*. In theory, this asymmetry could be in either direction, but generally it is taken to mean that compression takes longer — sometimes much longer — than decompression. This is acceptable, except during capture, but since playback must take place at a reasonably fast frame rate, codecs which take much longer to decompress video than to compress it are essentially useless.

⇨ The quality of lossily compressed video can only be judged subjectively — we can measure how much information has been discarded, but not whether it matters to the viewer. Consequently, any terms used to describe video quality are vague at best. One of the most commonly employed terms is 'broadcast quality', which does not mean the quality you receive on a TV set, it means good enough *to be* broadcast or used in post-production — a much higher quality, usually only achieved in a professional studio, using high end equipment. 'Near broadcast quality', a favourite term of marketing people, means 'not broadcast quality'. 'VHS quality' is a vague term, as VHS tape is available in a very wide range of grades, from the cheapest commonly used in domestic VCRs to tapes described as 'professional' or 'broadcast'. Generally, though, it is reasonable to take 'VHS quality' to mean 'just about acceptable'.

Motion JPEG and DV

At present, the most popular approach to compressing analogue video during capture is to apply JPEG compression to each frame, with no temporal compression. JPEG compression is applied to the three components of a colour image separately, and works the same way irrespective of the colour space used to store image data. Analogue video data is usually stored using $Y'C_BC_R$ colour, with chrominance sub-sampling, as we have seen; JPEG compression can be applied directly to this data, taking advantage of the compression already achieved by this sub-sampling.

The technique of compressing video sequences by applying JPEG compression to each frame is referred to as *motion JPEG* or *MJPEG* compression[6], although you should be aware that, whereas JPEG is a standard, MJPEG is only a loosely defined way of referring to this type of video compression. Special-purpose hardware is usually employed, allowing the complex computations required by the JPEG algorithm to be carried out fast enough to keep up with the incoming video signal. Individual hardware manufacturers have implemented motion JPEG independently, and in the absence of any standard, implementations differ slightly, especially in the way that the compressed data is stored. Recently, a consortium of digital video companies has agreed a standard format known as *MJPEG-A*, which is supported by QuickTime (see below), ensuring that compressed data can be exchanged between systems using different hardware.

Like still-image JPEG, motion JPEG codecs allow the user to specify a quality setting, trading off high compression against image quality. Instead of specifying the quality directly, video codecs often allow the specification of a maximum data rate, from which a suitable quality setting is deduced. This is appropriate, for example, if you wish your video to be played back from a DVD, where the maximum data rate is known in advance.

Data rates of around 3 Mbytes per second, corresponding to a compression ratio of around 7:1 are commonly achieved by low- to mid-range capture cards. More expensive cards offer higher data rates (lower compression ratios) and therefore better quality.[7] However, for multimedia use, it is invariably necessary to apply a high degree of compression to the video data for delivery, so although the principle of starting with the highest possible quality material is a sound one, it may not be worthwhile working with the best capture boards if the final product is destined for CD-ROM or the Internet.

Provided the device on which the compressed video data is stored is capable of delivering it at the rate required by the card, MJPEG

[6] Not to be confused with MPEG.

[7] This is perhaps counter-intuitive: the more you pay for compression, the less of it you get.

video can be played back at full frame size at standard frame rates. Such 'full-frame, full-motion' playback cannot presently be achieved without the aid of special-purpose hardware.

Although in multimedia it is most common to use MJPEG hardware to compress video as it is captured, sometimes we will want to go in the opposite direction, and use software to perform motion JPEG compression so that a video sequence that is already in digital form can be played back through a hardware codec. MJPEG compression is also sometimes used as an archiving format for video, since it provides higher quality than the codecs that are needed to compress material for delivery. However, since JPEG compression is lossy, so is MJPEG compression. This means that whenever video is compressed using MJPEG, some information, and hence picture quality, is immediately lost. Additionally, any subsequent decompression and recompression will exacerbate this loss of quality. It is therefore preferable, if at all possible, to archive uncompressed video, and only to use MJPEG where it is necessary — at the point of capture.

DV equipment uses a compression algorithm similar to MJPEG, though it is not identical. DV is chrominance sub-sampled to 4:1:1. Like JPEG, DV compression is based on the Discrete Cosine Transform (see Chapter 5); in addition it performs some temporal compression, but only between the two fields of each frame. It uses motion compensation, which will be described under MPEG shortly, to help achieve this. Because of the demands of DV-based VTRs, DV must maintain a constant data rate of 25 Mbits (3.25 Mbytes) per second, a compression ratio of 6.6:1 relative to CCIR 601, or, to give a better idea of how much compression is really being achieved by the algorithm, 5:1 relative to the 4:1:1 sampled data stream. The quality is varied dynamically to maintain this rate; this means that frames with relatively little motion will be spatially compressed less than, and hence will be of a higher quality than, those with more motion, because in the latter case less temporal compression will be possible. DV data should be of better quality than MJPEG compressed analogue video that has been compressed by a capture board to a similar compression ratio, because of the absence of noise, which, as we stated earlier, interferes with the effectiveness of compression.

Software Codecs for Multimedia

Among the many software codecs that have been developed, four are presently considered especially suitable for compressing video destined for delivery on CD-ROM or (with a little patience) over the Internet. The most elaborate of these, MPEG-1, will be described in the next section. The other three are called *Cinepak*, *Intel Indeo*, and *Sorenson*.

All three are based on a technique known as *vector quantization*, which works in the following manner: each frame is divided up into small rectangular blocks of pixels, the 'vectors' in vector quantization. The codec uses a collection of constant vectors, known as a *code book*; the code book vectors represent typical patterns that might occur in an image, such as flat areas of colour, sharp or soft edges, or different textures. Quantization is the process of allocating each vector in the image to the vector in the code book which approximates it most closely. (It is a generalization of scalar quantization, familiar from our description of digitization, when a value from some continuous range is approximated by one of a number of fixed levels.) Vector quantization provides compression, because each vector in the image can be replaced by an index into the code book. The image can be reconstructed from these indices and the code book by a simple process of putting together copies of the code book vectors corresponding to the stored indices. Decompression is thus very efficient, and can be carried out without special-purpose hardware. Compression, on the other hand, is a computationally intensive process, so these codecs are highly asymmetrical: compressing a frame using Cinepak, for example, typically takes more than 150 times as long as decompressing it.

Cinepak, Intel Indeo and Sorenson all augment their vector quantization compression with temporal compression using key frames and difference frames. The first two use a straightforward differencing scheme, while Sorenson uses a more sophisticated scheme, including motion compensation, which is closer to that used in MPEG compression.

Even though the decompression process is relatively efficient, full-motion full-screen playback cannot be achieved on mid-range processors with these codecs. Therefore, they are usually applied to small frames to be played back at reduced frame rates.

Respectable (VHS quality) images can be achieved at quarter-frame size (320×240 pixels) and 12 frames per second — certainly not good enough for the discerning viewer, but acceptable for many purposes. The Sorenson codec (the most efficient of the three) can compress video with these parameters so that it has a data rate of only 50 kilobytes per second, which is well within the capabilities of a 'multimedia PC' and can even be delivered by a 1× speed CD-ROM drive. (It should be borne in mind, though, that this data rate is still well in excess of that of an ISDN line, let alone a modem, so there is no chance of using these codecs to achieve real-time playback over a dial-up Internet connection. They could be used in this way over a 10 base T ethernet, though.)

Of the three, Cinepak has been established the longest. It remains popular, because of the high compression ratios it can achieve, and the efficiency with which it plays back compressed material, which makes it especially suitable for use on older computers. Intel's Indeo is similar in its general characteristics, but is less asymmetrical, being roughly 30% faster than Cinepak at compressing. It was originally intended for hardware implementation, but is now available in an efficient software version. It is generally held that Cinepak is superior for material with a lot of motion, but Indeo is better for more static material, for which it does a better job of preserving colours accurately. Sorenson is the most recently produced of these three codecs, and is widely hyped as being superior to the other two, both in quality and in the compression ratios it can achieve. It is, however, only available with QuickTime 3.0 or higher.[8]

MPEG Video

Although all of the codecs described so far are widely available, none of them is anything more than a *de facto* standard. Several international standards for compressed digital video are grouped together under the name *MPEG*, which stands for Motion Picture Experts Group — the resemblance to JPEG is not coincidental.

Earlier in this chapter, we briefly described the MPEG-2 format for broadcast digital video. For video to be incorporated in multimedia, the earlier MPEG-1 format is more relevant, being designed for lower data rates and progressive rendering. An emerging MPEG-4 standard provides support for integrated multimedia, including

[8] The version of the Sorenson compressor distributed free with QuickTime is a basic one that lacks some of the features that make the full version attractive. The full version is relatively expensive.

video, at data rates ranging from as low as 5 kbits per second to around 4 Mbytes per second. It is to be expected that this ambitious standard will have a growing importance for distributed multimedia, but at the time of writing, it is not published and no implementations based on it exist, so we will concentrate on MPEG-1 — in this chapter, its video aspects only.

The MPEG-1 standard[9] doesn't actually define a compression algorithm: it defines a data stream syntax and a decompressor, allowing manufacturers to develop different compressors, thereby leaving scope for 'competitive advantage in the marketplace'. In practice, the compressor is fairly thoroughly defined implicitly, so we can describe MPEG-1 compression, which combines temporal compression based on motion compensation with spatial compression based, like JPEG, on quantization and coding of frequency coefficients produced by a discrete cosine transformation of the data.

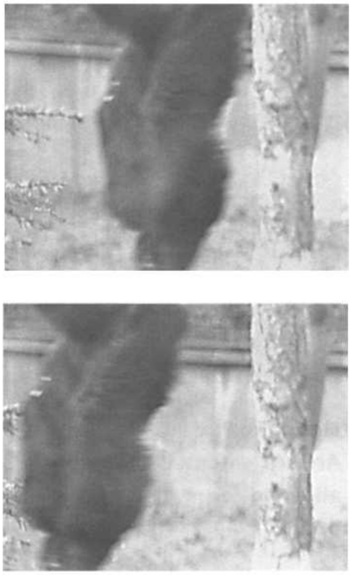


Figure 10.5
An object moving between frames

A naïve approach to temporal compression consists of subtracting the value of each pixel in a frame from the corresponding pixel in the previous frame, producing a difference frame. In areas of the picture where there is no change between frames, the result of this subtraction will be zero. If change is localized, difference frames will contain large numbers of zero pixels, and so they will compress well — much better than a key frame. Often, though, we may be able to do better, because pictures are composed of objects that move as a whole: a person might walk along a street, a football might be kicked, or the camera might pan across a landscape with trees. Figure 10.5 shows a simple example, consisting of an object — actually the lower body and legs of a gibbon, swinging from a rope — moving through two consecutive frames of a movie. As the dark shape of the gibbon moves across the screen, it obscures different parts of the vegetation behind it. Figure 10.6 shows the area within which pixels may have changed. If we could somehow identify the coherent area corresponding to the animal, we would only need to record its movement together with the changed pixels in the area shown in Figure 10.7. *Motion compensation* is an attempt to do this.

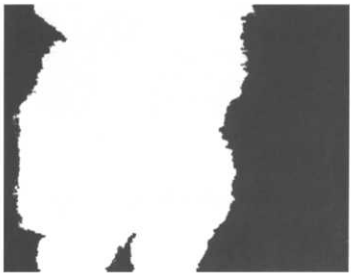


Figure 10.6
Area of potential change

MPEG compressors do not attempt to identify objects in a scene. Instead, they divide each frame into blocks of 16×16 pixels known as *macroblocks* (to distinguish them from the smaller blocks used in the DCT phase of compression), and attempt to predict the whereabouts of the corresponding macroblock in the next frame. No high-powered artificial intelligence is used in this prediction: all possible displacements within a limited range are tried, and the

[9] ISO/IEC 11172: 'Coding of moving pictures and associated audio for digital storage media at up to about 1.5Mbit/s'

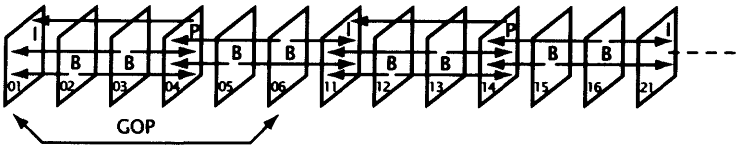


Figure 10.9
An MPEG sequence in display order

best match is chosen.[10] The difference frame is then constructed by subtracting each macroblock from its predicted counterpart, which should result in fewer non-zero pixels, and a smaller difference frame after spatial compression. The price to be paid is that, in addition to the difference frame, we now have to keep the motion vectors describing the predicted displacement of macroblocks between frames.

Figure 10.7
Area of potential change, allowing for motion of the object

Temporal compression has to start somewhere. MPEG key frames are called *I-pictures*, where I stands for *intra*. These frames are purely spatially (intra-frame) compressed. Difference frames that use previous frames are called *P-pictures*, or 'predictive pictures'. P-pictures can be based on an earlier I-picture or P-picture. MPEG goes further and allows for frames that are predicted from later frames; these are *B-pictures*. Figure 10.8 shows why backward prediction can be useful. In this frame, which precedes those shown in Figure 10.5, the tree trunk on the right of the picture is hidden. If we wished to use this frame as an I-picture, the following P-picture would have to record all the pixels of the tree trunk. However, the trunk is present in the following frame (the lower one in Figure 10.5), so if the middle frame is predicted from the one before itself and the one after, further compression will be achieved. B-pictures can use motion compensation from the next I- or P-pictures, or both, hence their full name 'bi-directionally predictive' pictures.

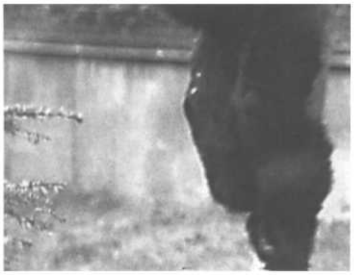


Figure 10.8
I-picture with a hidden object

A video clip can be encoded in compressed form as a sequence of I-, P- and B-pictures. It is not a requirement that this sequence be regular, but encoders typically use a repeating sequence, known as a *Group of Pictures* or *GOP*, which always begins with an I-picture. Figure 10.9 shows a typical example.[11] The GOP sequence is IBBPBB; the picture shows two such groups: frames 01 to 06 and 11 to 16. The arrows indicate the forward and bi-directional prediction. For example, the P-picture 04 depends on the I-picture 01 at the start of its GOP; the B-pictures 05 and 06 depend on the preceding P-picture 04 and the following I-picture 11. All three types of picture are compressed using the MPEG version of JPEG

[10] Extensive searching is not a good idea, since it implies the need for larger displacement vectors.

[11] Read it from left to right, like a time line.

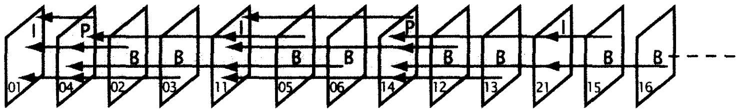


Figure 10.10
An MPEG sequence in bitstream order

compression. Published measurements[12] indicate that, typically, P-pictures compress three times as much as I-pictures, and B-pictures one and a half times as much as P-pictures. However, reconstructing B-pictures is more complex than reconstructing the other types, so there is a trade-off to be made between compression and computational complexity when choosing the pattern of a GOP. An additional factor is that random access to frames corresponding to B- and P-pictures is difficult, so it is customary to include I-pictures sufficiently often to allow random access to several frames each second. Popular GOP patterns include IBBPBBPBB and IBBPBBPBBPBB. However, as we remarked, the MPEG specification does not require the sequence of pictures to form a regular pattern, and sophisticated encoders will adjust the frequency of I-pictures in response to the nature of the video stream being compressed.

For the decoder, there is an obvious problem with B-pictures: some of the information required to reconstruct the corresponding frame is contained in an I- or P-picture that comes later in the sequence. This problem is solved by reordering the sequence. The sequence of pictures corresponding to the actual order of frames is said to be in 'display order'; it must be rearranged into a suitable 'bitstream order' for transmission. Figure 10.10 shows the bitstream order of the sequence shown in display order in Figure 10.9. All the arrows showing prediction now run from right to left, i.e. every predicted frame comes later in the sequence than the pictures it depends on.[13] You will notice that the first GOP is reordered differently from the second; any subsequent groups will extend the pattern established by the second.

Before any of this compression is done, MPEG video data is chroma sub-sampled to 4:2:0. If, in addition to this, the frame size is restricted to 352×240[14], video at a frame rate of 30 fps can be compressed to a data rate of 1.86 Mbits per second — the data rate specified for compact disc video. This is the typical format for MPEG-1 video, although it can be used with larger frame sizes and other frame rates. MPEG-1 cannot, however, handle interlacing or HDTV formats, hence the need for MPEG-2 for broadcasting and studio work. For multimedia, however, MPEG-1 is an excellent

[12] In [Tud95].

[13] For the B-pictures, we have run the arrows to the relevant P- and I-pictures together, with an intermediate arrowhead, in an attempt to keep the diagram less cluttered.

[14] 4:2:0 video of this size is said to be in *Source Input Format (SIF)*.

form of compression that provides good quality compared to the software codecs described in the previous section.

The preceding description should have made it clear that MPEG compression and decompression are computationally expensive tasks — and there are further complications which we have glossed over. Initially, MPEG video, like MJPEG, could only be played back using dedicated hardware. Indeed, the parameters used for CD video were chosen largely so that MPEG decoders could be accommodated in VLSI chips at the time the standard was drawn up (1993). Advances in processor speed mean that it has since become feasible to play back MPEG at only slightly lower frame rates and sizes using software only — a G3 PowerMac takes 320×240 at 24 fps in its stride, producing smooth VHS quality playback. The associated data rate of 264 kbytes per second can be delivered by a 2× speed CD-ROM drive. File sizes are by no means small, however. A 650 Mbyte CD-ROM will only hold just over 40 minutes of video at that rate; an 8.75 Gbyte DVD has room for over nine hours. (You would only use MPEG-1 on DVD if your video was part of a multimedia production, though. DVDs employ MPEG-2 when they are Digital *Video* Disks.)

QuickTime

As preceding sections have shown, there is a plethora of digital video compression schemes and data formats in existence: DV, MPEG-1 and MPEG-2, several varieties of MJPEG, various software compressors, including the ones described earlier, proprietary schemes devised by particular manufacturers of digital video workstations, not forgetting uncompressed video. Each of these schemes requires the information to be encoded in a different way; within each encoding there is scope for defining incompatible file formats — even some standards, such as the MPEGs, only define a data stream, not a file format. Because of the diversity of requirements, there is little hope of being able to design a universal video file format, and the usual political considerations make it unlikely that all the parties involved would agree to it, even if one could be devised. A more profitable approach to standardization is to base it on an architectural framework, defined at a sufficiently abstract level to accommodate a multiplicity of concrete video representations. Several such approaches have been proposed, but *QuickTime* has established itself as a *de facto* standard.

QuickTime was introduced by Apple in 1991, and reached a mature state with the release of version 3.0 in mid-1998.[15] The objects that QuickTime manipulates are *movies*. For the present, you can consider a movie to be an abstraction of a video sequence, although, in fact, QuickTime movies can contain other types of media as well. Although we speak of a movie containing data, the movie itself is really just a framework for organizing, accessing and manipulating the data that represent the actual video frames, which may be stored separately from the movie itself.

Originally, QuickTime's focus was on the temporal aspects of time-based media, including video. Every movie has a *time base*, which records the rate at which it should be played back and its current position; both are specified relative to a time coordinate system which allows the time base to be synchronized to a clock so that movies can be played back at the correct speed on any system. When a movie is played, if the frames cannot be displayed fast enough to maintain the required frame rate, some are dropped so that the overall duration of the movie will be correct and synchronization (for example with an associated sound track) will be maintained. The time coordinate system also makes it possible to identify specific points in a movie, so that individual frames can be accessed; this in turn facilitates non-linear editing of movies — a topic which will be described in the next section.

The success of QuickTime is largely due to its being a component-based architecture. This means that it is possible to plug components (small programs conforming to certain interface conventions) into the QuickTime structure in order to deal with new formats and provide new implementations of operations. A set of standard components is supplied as part of the distribution. These include a number of *compressor components*, implementing Sorenson, Cinepak, and Intel Indeo codecs, as well as several others developed for specific tasks such as compressing computer-generated animation. *Sequence grabber components* are used to digitize video. They use lower level components to obtain any necessary parameters from the user, and to communicate with digitizing hardware. A standard *movie controller* component is used to provide a user interface for playing movies; it is illustrated in Figure 10.11, where you can see that the control buttons are based on those normally found on domestic VCRs, so that their use is obvious to most people. There are also components called *transcoders*, which translate data between formats that use the same

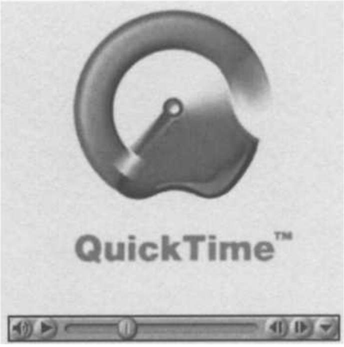


Figure 10.11
A standard QuickTime movie controller

[15] In 1999, QuickTime 4.0 added support for streaming, which will be described at the end of this chapter.

compression algorithm, without the need for decompression and recompression.

QuickTime does have its own file format, which provides a very flexible way of storing video and other media.[16] To avoid imposing this format on everybody, and to enable application software based on QuickTime to access other types of file, components have been added to make it possible to manipulate files in other formats as if they were native QuickTime. Formats supported in this way include MPEG-1 and DV, OMF (a high-end professional format), and Microsoft's AVI, and its extension OpenDML. As this demonstrates, QuickTime's component architecture makes it easily extensible. Another common example occurs when a new video capture card is developed, with its own features and idiosyncrasies. Provided its manufacturer writes a (relatively simple) *video digitizer component*, any application program based on QuickTime can capture and play back video using the new card, edit that video, and convert it to other formats. Looking at this from the other side, anybody who devises a new video format or codec does not have to implement all the high level editing operations that are necessary to make it useful — QuickTime provides all that, using the interface supplied by the components associated with the type of data in question. This abstraction away from specific formats is passed on to end users of video editing and playback software. For example, when a video editor wishes to save an edited movie, he or she is presented with options for selecting a compression method. If a new codec is added to their system, it will show up as a new alternative within the same interface.

QuickTime 4.0 is available in a fully compatible form on the 32-bit Windows platforms as well as MacOS, and QuickTime is also supported on SGI workstations for professional use. A version of QuickTime for Java has also been developed. As well as the basic functionality implemented in the QuickTime system software, a plug-in for popular Web browsers is distributed with QuickTime, which means that QuickTime files can be used as a format for distributing video over the World Wide Web.

The only other formats that you might encounter or consider for use in multimedia are MPEG and AVI. We have stated that MPEG does not define a file format, but there is a rather obvious way of storing an MPEG data stream in a file: just record the bytes in the order they appear in the stream. Files of this form are becoming quite common.[17] They can be played back by QuickTime.

[16] The QuickTime file format has also been adopted as the file format for storing MPEG-4 data.

[17] Often they are identified by the extension .mpg.

So can AVI files, although they are the native format for Microsoft's Video for Windows, a system with similar objectives to QuickTime. Because of the pervasiveness of Windows operating systems, AVI has been widely used, but Video for Windows is generally held to be technically inferior to QuickTime, and has indeed been abandoned by Microsoft, whose attempts at a successor have been overshadowed by the adoption of QuickTime as a *de facto* cross-platform standard. As we stated earlier, QuickTime 3.0 and later versions can handle AVI files directly.

Digital Video Editing and Post-Production

Shooting and recording video only provides raw material. Creating a finished piece of video — whether it is a feature film or a small clip for a Web site — requires additional work. *Editing* is the process of making a constructed whole from a collection of parts. It comprises the selection, trimming and organization of the raw footage, and where sound is used, its combination with the picture. *Transitions*, such as dissolves, may be applied between shots, but no changes are made to the footage itself. We contrast this with *post-production*, which is concerned with making changes or adding to the material. Many of the changes made at this stage are generalizations of the image manipulation operations we described in Chapter 5: colour and contrast corrections, blurring or sharpening, and so on. Compositing — the combination or overlaying of elements from different shots into one composite sequence — is often carried out during post-production; for example, figures may be inserted into background scenes that were shot separately. Elements may be animated during post-production, and animation may be combined with live action, in the manner that has become characteristic of film special effects.

People have been making films for over a hundred years, and during that time an elaborate set of conventions about how film — and, by extension, video — is edited have developed. For example, action sequences in Hollywood films are typically cut 'on the action'; that is, cuts are made while something is happening, in order to distract the viewer's attention from the cut itself. In contrast, flashback sequences are often introduced by a long dissolve, typically starting from a shot of a person starting to tell a story about earlier events, with the narration continuing over the first shot of the flashback. Here, the transition is used to signify the change of time frame, and is supposed to be evident to the viewer, or this method of signification will not work. Viewers expect film to be edited in accordance with conventions such as these; if it is not, as in certain types of avant garde work, then that fact in itself makes a statement. Thus, in contrast to Hollywood directors, some of the French *nouvelle vague* directors of the 1960s deliberately used 'jump cuts', occurring apparently gratuitously in the middle of a scene, to draw attention to the constructed nature of their films, and cut straight to flashbacks instead of using dissolves, acknowledging that film is only seen in its own time. This kind of editing requires the audience to be more alert — to work harder — if they are to appreciate what the film-maker is doing (or even follow the narrative, in some cases).[18] Early film makers, such as Dziga Vertov and Sergei Eisenstein, experimented with editing as much as with the process of shooting film, and their work is still referred to by students of film today.

All film and video is constructed. A common example is the way in which a conversation between two people facing each other is conventionally presented: each person occupies the field of view while they speak; when the speaker changes, a cut occurs to a 'reverse shot', with the camera apparently moving instantaneously through nearly 180° to face the new speaker — as if the camera were each person in turn, watching the other speak. You have probably seen this so often that you barely think about it, but, in fact, the two actors in the scene were probably not carrying on a conversation in real time while the dialogue was shot; they may not even have been filmed speaking their lines on the same day. The conversation is created in the editing, on the basis of a convention that, if you do stop to think about it, does not resemble the way anybody could see that conversation if it really took place. This is a particularly blatant example of the artifice involved in constructing sequences of moving pictures, but some artifice is always involved — even the picture from a live Webcam is framed in a certain way. While most Hollywood film makers seek to make the construction of a film invisible, and concentrate on telling a story, other directors, such as Jean-Luc Godard, prefer to expose the mechanics of film making, and still others, such as Eric Rohmer, prefer to minimize their reliance on convention, and shoot and edit in a more realistic way, using third person viewpoints and extended shots for dialogue

[18] It is interesting to observe that when Woody Allen, working within the commercial milieu, used similar devices in films such as *Husbands and Wives*, they were dismissed by mainstream critics as being intrusive and distracting.

scenes, with the sound recorded live, for example. But, however a piece of film or video is to be constructed, it will be necessary both to shoot footage and to edit it.

Even if nobody had ever wanted to display video on a computer, incorporate it into a multimedia production, or broadcast it digitally, video would have been digitized, because the advantages of non-linear editing which digitization brings are too compelling to resist. To appreciate this, and to understand the metaphors used by digital editing systems, you have to consider traditional methods of film and video editing.

Film and Video Editing

Editing film is a physical process. To a film editor, a 'cut' is not merely a figurative reference to an edit that produces a discontinuity in the flow of pictures, it is a real severing of the film itself, which divides a strip of film into two clips which may then be spliced together with others to compose a scene. Clips can be trimmed, cut and spliced, arranged and rearranged, for as long as the fabric of the film holds up. The main problem with this basic editing is keeping track of everything — a good deal of labelling and hanging strips of film from hooks is called for.

Although making straight cuts in film is straightforward, creating other types of transition between clips is much less so, and requires special equipment. Suppose, for example, you wanted a scene of someone falling asleep to dissolve into a dream sequence. Such an effect is usually achieved using a device called an *optical printer*, which is a rig that directs the light from a pair of projectors into a camera.[19] Optical filters and masks can be interposed to control the amount of light from each projector reaching the camera. For our dissolve, we would put the end of the falling asleep scene on one projector and the beginning of the dream sequence on the other; initially, filters would be set up so that all the light from the first projector and none from the second reached the camera. Gradually, the filters would be adjusted, so that the two beams were mixed, with the proportion from the second projector increasing as that from the first decreased. The result would be a piece of film with the required dissolve effect; this would then be spliced in place of the corresponding frames of the two original clips.

[19] There are different types of optical printer. Some only use a single projector, but expose the film in the camera more than once, to achieve the same effects as the set-up described here.

Despite the apparent simplicity of the set-up, exceptionally sophisticated effects can be achieved using such 'opticals', in conjunction with techniques such as matte painting or with models.[20] One drawback is that opticals are usually done by a specialist laboratory, so the film editor cannot actually see what the transition looks like until the film has been developed. This leaves little room for experimentation. It is no coincidence that the straight cut forms the basis of most films' structure.

Traditional video editing, although the same as film editing in principle, is quite different in practice. It is virtually impossible to cut video tape accurately, or splice it together, without destroying it. The only way to rearrange pictures recorded on video tape was to copy them onto a new tape in the desired order. If, for example, we wanted to cut from an outside view of a house to a scene in a room inside it, and had first recorded interior scenes and then added some outside shots to the end of the same tape, we could use two tape machines (call them A and B) to assemble the material in the required order. The output signal from machine A is fed to the input of machine B, so that B records what A plays. The original tape is put on machine A and wound forward to the point at which the desired part of the exterior scene begins, then B is started and records that scene. Recording is stopped while machine A is rewound to the exact beginning of the interior, which is then copied onto the tape in machine B, which now holds the required material in the right order.

A more powerful arrangement is to use three machines, designated A, B and C. If you can arrange that your two scenes are on separate tapes, you load one on to each of A and B, and set the *in point* (first frame) and *out point* (last frame) of each. A device known as an *edit controller* can now start machine A at the in point of the first scene, copying it on to machine C until it reaches its out point, when machine B is started at its in point, adding the second scene to the tape on machine C. One virtue of this arrangement is that the cut can be previewed, by running machines A and B without actually recording on C, so it is possible to experiment with adjusting in and out points until a perfect cut is obtained. The signals can be mixed and modified electronically, so this arrangement also provides a video equivalent of the optical printer, for some standard operations. A rich variety of transitions can be produced this way, and, unlike film transitions, they can be reviewed straight away, and parameters such as the speed of a dissolve can be controlled in

[20] The effects in the originally released version of *Star Wars* were purely optical, not digital.

real time. Three machines also make insert editing straightforward. Suppose, for example, that part of a scene has had to be reshot after a first edit of the scene has been done. The original edit can be played off one machine, until the point where the defective footage occurs, at which point the signal is taken from the other machine, and then switched back after the replacement shot has been inserted.

Note that these operations require that the machines be able to start and stop in precisely the right place,[21] which not only requires extremely accurate control mechanisms, but also some means of identifying positions on tapes. *Timecode* is used for this purpose. There are several standards in use, but the only one of any importance is SMPTE timecode. A timecode value consists of four pairs of digits, separated by colons, such as 01:14:35:06, representing hours, minutes, seconds, and frames, so that the complete value identifies a precise frame. A trivially obvious scheme, you may well think — the tricky bit is writing the code onto the tape so that its current frame can be read by a machine. Standards for doing so are in place, and so 'frame-accurate' positioning of tape is possible.

⇨ Timecode behaves differently depending on the frame rate: for a PAL system, the final component (which identifies the frame number) ranges from 0 to 24, for NTSC it ranges from 0 to 29, but not in the obvious way, because the NTSC frame rate is 29.97. Since there is not an exact number of NTSC frames in a second, SMPTE timecode, which must use exactly 30, drifts with respect to the elapsed time. The expedient adopted to work round this is called *drop frame timecode*, in which frames 00:00 and 00:01 are omitted at the start of every minute except the tenth. (It's a bit like a leap year.) So your count jumps from, say, 00:00:59:29 to 00:01:00:02, but runs smoothly from 00:09:59:29 through 00:10:00:00 to 00:10:00:01. Whether or not it handles drop frame timecode correctly is one measure of how professional a digital video editing program is.

Editing video on two or three machines requires tapes to be copied every time a change is made, and it is inevitable that there will be generational loss as noise is introduced by the copying process. Even the best analogue systems used for commercial video always introduce some loss every time a copy is made. With VHS tape, just two copying operations is usually sufficient to produce serious loss of quality. A second drawback is that the final tape has to

[21] It is physically impossible to start a tape transport instantaneously, so some *pre-roll* is always required as the machine comes up to speed.

be constructed linearly, from beginning to end, in contrast to film, which can be spliced together in any order. This imposes a less flexible way of working, which inevitably has implications for the nature and quality of video editing.

Digital Video Editing

The unique visual qualities of film might make the process of cutting and splicing worthwhile, but video is such a poor medium that it offers little return for the hardships that must be endured in a three machine edit suite. Digitization has opened the way to a different mode of working, that brings video editing closer in kind to film editing, but without the physical process. An imperfect, but useful, analogy of the difference between linear analogue and non-linear digital video editing is the difference between writing with a typewriter and using a word processor. On a traditional typewriter, words have to be written in their final order, with the potential for corrections limited to what can be achieved with Tipp-Ex, unless an entire sheet is thrown away and retyped — which might upset subsequent pagination. In the latter case, corrections can be made anywhere, text can be composed in any order, without regard to pagination or layout. The ability to randomly access and change data, which is fundamental to the way we build computers, engenders these capabilities in text processing software, and analogous ones in video editing software.

Besides random access, digital video editing's other big advantage is that it is non-destructive. Source clips are never changed. This means that it is possible to cut and recut, potentially forever, as the editor changes his or her mind. Furthermore, in contrast to film, edited digital video can be played back as soon as the hardware on which it is being edited allows. With top-end equipment, this is instantaneously; on desktop machines, there is usually a delay — sometimes a considerable delay — but the delays are measured in minutes and hours, not the days that it may take for film to be processed. Within a few years, at most, even desktop equipment and software will be providing instant playback of edited digital video.

⇨ Despite these advantages, it has been observed by film and video editors who have adopted digital technology that it does not save time. Editors tend to be perfectionists, and the opportunity to experiment with different cuts and transitions can lead to considerable amounts of time being spent trying to achieve the perfect edit.

Developments in hardware, including DV and FireWire and the increased speed of processors and capacity of hard disks, have led to a broadening of interest in digital video editing. Formerly, only a few professional systems were available, but there is now a range from basic consumer-oriented editors, designed for editing home videos, up to the state of the art editing suites used by film and television studios. Adobe Premiere, which could be described as a mid-range video editing application that incorporates some post-production facilities, is the most widely used video application on desktop platforms, so we will use it as a concrete example,[22] but all video editing programs work in roughly the same way, using the same metaphors derived from traditional film and video editing practices.

Premiere's interface can be customized to fit different ways of working, but, for most people, editing takes place in three main windows. This is a major difference between video editing and image, graphics, or text editing, where a single document window suffices; the more complex, time-based, nature of video requires that the user be able to view the material in several different ways at once. As we will see in later chapters, other time-based media and multimedia applications have the same requirement for multiple views.

Premiere's three main windows are the project, timeline, and monitor. The project window is used to collect the raw material for a project: video and audio clips, and usually, still images. Video clips may be captured within Premiere from tape or camera, or they may be imported from disk, having been previously captured, possibly with some other application. Within the project window, each clip may be displayed as an icon, with its name, duration, codec, and other information. Related clips can be organized into bins — rather as files are organized in sub-directories — and ordered in whatever way seems perspicuous to the user.

The timeline window provides a visual display of the linear extent of the completed movie, showing the order of its component clips. A simple example is shown in Figure 10.12. Time increases from left to right, and is linearly related to the length of segments of the timeline, so the movie shown here begins with `clip1`, followed by `clip2`, which is roughly twice as long, then the short `clip3`

[22] Note that our descriptions refer to Premiere 5. Earlier releases use a somewhat different interface.

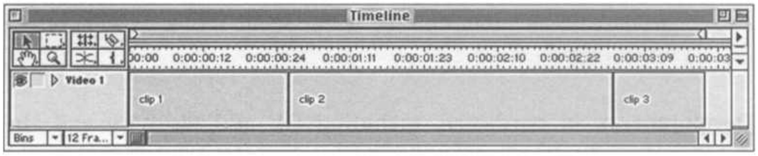


Figure 10.12
A simple composition on the timeline in Premiere

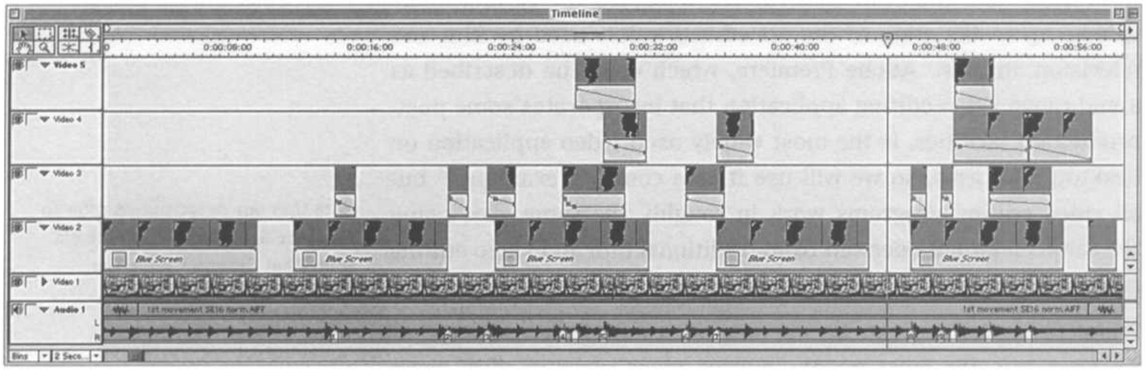
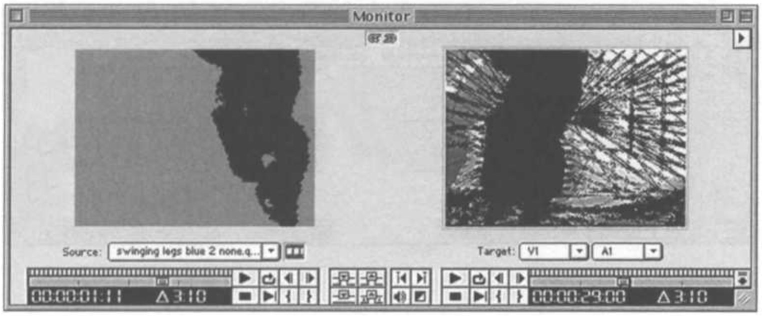


Figure 10.13
A complex Premiere composition

concludes it. In general, timelines may have several video *tracks*, as well as audio tracks. Figure 10.13 shows an example of a complex composition with a sound track. The multiple tracks are used for transitions and overlays, as we will describe shortly.

The third window, the monitor, is where you actually see your picture. (In the other two windows you only see iconic or text representations of clips.) In Premiere 5, the monitor is split into two panes, as shown in Figure 10.14, each with separate playback controls, including jog and shuttle, for stepping through a frame at a time, or scrubbing forwards and backwards. The left half shows a source clip; the clip can be trimmed by scrubbing to the correct frames in the clip in this window and setting in and out points (as in a conventional analogue editing system). The right half of the monitor window is used to display the movie as it has been assembled on the timeline, so that the effects on the whole movie of trims and other edits can be previewed.

Each editor will evolve their own method of working with a particular program, and Premiere, for example, offers alternative means for achieving the same ends. One simple, idealized procedure for editing with Premiere would begin with assembling all the clips for



Source: swinging legs blue 2 none.q...

Target: V1 A1

Figure 10.14
Premiere's monitor window

a project — capturing them where necessary, and importing them into the project window, where they are arranged for convenient access. Next, each clip is opened in the monitor window, and roughly trimmed. Note that trimming does not actually discard any frames, as it would with film, it merely suppresses those before the in point and after the out point. If necessary, the in and out points can be readjusted later; if the out point is subsequently moved to a later frame in the clip, or the in point is moved to an earlier one, frames between the old and new points will reappear. The initial trimming operation will leave the editor with the material that they believe should appear in the final movie. The next step is to drag clips from the project window and drop them onto the timeline, assembling them into a rough cut, which can be previewed. Still images can be dragged to the timeline and given a duration; they will behave as clips with no motion. For movies with sound, the picture and sound track can be combined. Almost always, adjustments will have to made, particularly if it is necessary to match up sound and picture. Clips may need to be trimmed again, or more drastic changes may be required, such as the substitution of completely different material when ideas fail to work out.

For some projects, editing will be complete at this stage, but for others, the desired transition from one clip to the next cannot be achieved simply by cutting directly. Using other transitions changes the style, rhythm and mood of a piece. A dissolve, for example, in which one clip fades into another, is less emphatic than a cut, and tends to convey a sense of gradual change or smooth flow from one thing to another. More fanciful transitions, such as wipes, spins and page turns, draw attention to themselves, and function almost as decoration. As most transitions can be described relatively easily in terms of mathematical operations on the two clips involved, digital

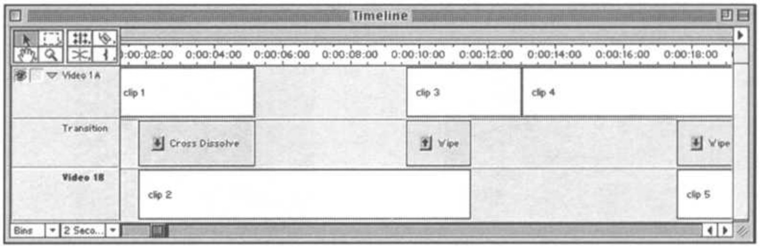


Figure 10.15
Applying transitions in Premiere

video editing software usually offers a vast range of possibilities — Premiere has 75 transitions built in (most of them showy gimmicks), plus a means of defining your own — of which the majority will never be used by a good editor.

Premiere's approach to adding transitions other than cuts is based on the three machine method of editing video. Each transition always involves exactly two clips, conventionally referred to as its A and B clips. These are combined on video track 1, which — uniquely — can be expanded into tracks 1A and 1B, with a special transition track between them. The A and B clips are positioned on tracks 1A and 1B so that they overlap in time; a transition is dragged from the transition palette onto the transition track where A and B overlap (see Figure 10.15). Its duration is automatically adjusted to fill the region of overlap. Many transitions are parameterized: for example, the 'clock wipe' transition may play clockwise or anticlockwise. Parameters can be set in a dialogue box, brought up by double-clicking the transition on the timeline.

There are two important practical differences between cuts and other transitions. Firstly, in a cut, the two clips are butted; in all other transitions, they overlap, and some part of each contributes to the resulting picture. For sequences which are not going to be edited just with basic cuts it is necessary to ensure that when shot and captured to disk, each clip has sufficient extra frames, beyond the period where it is to play alone, to cover the transition. Secondly, because image processing is required to construct the transitional frames, transitions must be rendered, unlike cuts, which can be implemented simply by copying. Hence, there will inevitably be some loss of image quality where dissolves and so on are used instead of straight cuts.

Digital Video Post-Production

Video editing is primarily concerned with the arrangement of picture through time, and its synchronization with sound. While frame accuracy may be required to set precise in and out points, or to eliminate faulty footage, for example, it is rarely necessary while editing to consider whole sequences of footage as individual frames. In post-production procedures, however, the temporal dimension may assume less importance, and a movie may be dealt with as a sequence of individual still images. Most digital post-production tasks can be seen as applications of the image manipulation operations we described in Chapter 5 to the images in such a sequence. Two important classes of post-production operation that can be described in these terms are those concerned with image correction, and with compositing.

A video sequence may suffer from the same defects as a single image: for example, it may be over- or under-exposed or out of focus; it may have a colour cast, or it may display unacceptable digitization artefacts. Each of these defects has its characteristic remedy: adjust the levels, sharpen, or apply a Gaussian blur. Post-production systems therefore provide the same set of adjustments as image manipulation programs — some support the use of Photoshop plug-ins — but allow them to be applied to sequences instead of a single image.

Most adjustments have parameters, such as the slider positions for the levels controls. When adjustments are made to sequences, it may be appropriate to use the same parameter values for each image, or it may be preferable for them to change. If, for example, a whole, mostly static, sequence has been shot under incorrect lighting, the same correction will probably be needed for every frame, so the levels can be set for the first, and the adjustment will be applied to as many frames as the user specifies. If, however, the light fades during a sequence, when it was intended to remain constant, it will be necessary to increase the brightness gradually to compensate. It is possible to apply a suitable correction to each frame individually, and this may occassionally be necessary, but often it is adequate to specify parameters at a few key frames and allow their values at intermediate frames to be interpolated. We will show in Chapter 11 how varying parameter values over time can be used to achieve certain special effects.

Just as some of the image manipulation operations we described in Chapter 5 combined separate images or layers into a composite result, so some post-production operations combine separate video tracks into a composite. In Premiere, as we have seen, video track 1 is special, inasmuch as it can be used for transitions. This gives it a privileged status. All other video tracks can only be used for superimposing picture over video track 1. As with still images, for superimposition to achieve anything useful, some parts of the superimposed tracks must be transparent. In video, selecting transparent areas is called *keying.* Good video editing and post-production software will offer several different keying methods.

The method of blue screening, which we described for single images in Chapter 6, has long been used in video for inserting isolated elements into shots. Traditional examples of its use include adding models to live footage, or placing actors in impossible or dangerous situations. Digital post-production systems support both traditional blue screening, where the actor or model is shot in front of a screen that is a particular shade of blue and then the blue channel is removed, and a more general form of *chroma keying,* where any colour in a scene can be selected and designated as transparent.[23] Chroma keying is essentially the same as building an alpha channel from a selection made using a magic wand tool. An alternative is *luma keying,* where a brightness threshold is used to determine which areas are transparent. Compositing on the basis of luma keying closely resembles the layer blending we demonstrated in Chapter 3.

As you might expect, it is possible to select a transparent area explicitly, using selection tools to create a mask. In film and video, a mask used for compositing is called a *matte.* Mattes are frequently used for removing unwanted elements, such as microphone booms, from a scene,[24] or for allowing live footage to be combined with a still image. A typical example of such a use of mattes occurs when actors are filmed on a set, containing just a few foreground elements such as trees and plants. The top of the picture (typically just a plain backdrop on the set) is matted out, and subsequently replaced with a painting of a landscape, so that the foreground scene appears to be taking place in front of a mountain range, a medieval castle, or whatever. Mattes can also be used for split-screen effects.

Another way of creating transparency is to use an alpha channel created in some other application. This is often the most satisfactory method of creating mattes to be used in conjunction with

[23] Compare this with the use of one colour from a palette to designate transparency in a GIF image.

[24] In this case, the matte is called a *garbage matte.*

still images, since Photoshop provides much more sophisticated selection tools than Premiere, for example. Alternatively, an imported greyscale image can be used as a matte.

Since a video clip is a sequence of images, a new possibility arises that is not present when single images are being composited: transparency can be made to vary over the course of the sequence (that is, to vary over time when it is played back). This happens automatically with chroma and luma keying, as the colour and brightness distribution changes between frames. In order to produce masking that varies over time, it is necessary to use a sequence of masks as the matte. Such a sequence — often called a *travelling matte* — is naturally stored in a separate video track, when it is called a *track matte*. Although, in principle, the content of any suitable video track could be used as a travelling matte, a specially designed sequence of masks will often be required in practice. Track mattes may be created painstakingly by hand, but are more usually generated from a single still image, by applying simple geometrical transformations over time to create a varying sequence of mattes, in a way that we will describe in Chapter 11. Travelling mattes are often used in title sequences, where they are especially favoured for news bulletins.

All of the post-production facilities we have described can be found in Premiere and other desktop video editing applications, but only in a relatively crude form. For serious post-production, dedicated applications such as Adobe After Effects are preferred. Although these do not offer any radically different operations, they do provide much greater control over their application, and implement them more accurately. Where an editing program may allow you to apply a filter to a clip, and vary it over time, a post-production program will provide a wide range of controls for the filter's parameters, and for the more sophisticated interpolation between key frames that it offers. For example, in Premiere, the parameter values are interpolated linearly between key frames, whereas in After Effects, the interpolation can also use Bézier curves, which can be specified using handles and control points, as they can in Illustrator.

⇨ Another post-production task that is perhaps less important for multimedia production than for commercial film and video is titling. Editing programs' titling facilities are usually barely adequate, simply allowing the superposition of text over video — Premiere did not provide any means of creating rolling credits until version 5.0. Sophisticated title and credit sequences are often best made by using graphics applications to create the text and graphic elements, which are then animated using a post-production tool such as After Effects, in a way that will be described in Chapter 11.

Preparing Video for Multimedia Delivery

Editing and post-production are performed in roughly the same way whether the final video is intended for multimedia delivery — off-line or on-line — or for conventional transmission or recording on video tape. For multimedia, an additional step of preparing the material for delivery is usually required. The reason for this step is the need to cope with limitations of the final delivery medium and playback platforms that are not usually relevant on the platform used for editing and post-production. (Recall from Chapter 2 that multimedia production usually takes place on specially equipped top-of-the-range desktop machines or workstations, whose performance far exceeds that of consumer machines.) Compromises must be made at this stage, which will involve choosing what is to be sacrificed in order to bring the resource requirements of video within the capabilities of delivery media and low-end machines. Different material will permit and suggest different choices.

What might be sacrificed in this way? The possibilities include frame size, frame rate, colour depth, and image quality. Reducing the size of video frames is usually a relatively painless way of reducing the size and bandwidth of video files. People usually sit close to their monitors, so a large picture is not necessary, and monitor resolution is usually better than that of television sets, so a downsampled image on a computer screen can appear smoother than the images we are familiar with on television (assuming the downsampling is done intelligently). Similarly, reducing the frame rate is often acceptable: the illusion of continuous motion can often be satisfactorily maintained with frame rates around 12 fps. Higher frame rates are needed to eliminate flicker only if the display is refreshed at the frame rate. This is not the case with computer monitors, which are refreshed at a much higher rate from VRAM. By using quarter frames and reducing the frame rate from 30 fps to 15, the volume of data is reduced by a factor of eight.

A further factor of three can be obtained by reducing the colour depth from the 24 bits usually used for video to eight bits. As we explained in Chapter 6, this can be done by limiting the colours to those in a standard palette (usually the Web-safe palette), using indexed colour with a custom palette, or by reducing to 256 shades of grey. We argued in Chapter 6 that the last option may be preferable to using a limited range of colours. Unfortunately, not all video codecs support the use of greyscale images. Among those that do not are Sorenson, Intel Indeo, and H.263[25], all of which are otherwise well suited to multimedia.

If these compromises are unacceptable, or — as may easily be the case — they do not sufficiently reduce the resource requirements of your video, it will be necessary to squeeze bytes out with compression, at the cost of a loss of image quality. We have already described suitable codecs, and their characteristics. It should be emphasized that, despite all the clever algorithms they use and despite the claims that are sometimes made, all codecs introduce visible artefacts when very high compression ratios are needed. It may be necessary to apply filters in order to minimize their effects, but this will also result in a further loss of picture information.

As well as taking steps to minimize the size of video, it is also necessary to ensure that it is fit to play on any platform. QuickTime movies prepared on a Macintosh computer must be flattened, for example, so that they do not store any data in the resource fork that is unique to MacOS files. Where necessary, movies must be made self-contained. That is, where the file format allows pointers to other files to be used as part of a movie, these pointers must be replaced by the data they refer to. It may be necessary to produce different versions of a movie for different platforms, to compensate for the different gamma of PC and Mac monitors, for example.[26] For video that is to be delivered over a network it is common practice to produce a range of different versions matched to the speed of users' network connections. QuickTime allows different versions to be combined into a single movie.

⇨ The processing described in this section is not always necessary even for video that is being incorporated into a multimedia production. If the production is to be presented in a kiosk or as part of an exhibition, where the producer determines what hardware is to be used, the video can be delivered at whatever quality the hardware supports. If a suitable video card is known to be available, for example, the codec that was used for capture can be used for delivery; if terabytes of storage can be supplied, uncompressed video can be used, and so on. Cross-platform issues are irrelevant in such situations.

[25] See page 344.

[26] It may sometimes be necessary to produce platform-specific versions at the editing stage, too. For example, if a movie contains product advertisements, there is little point in including advertisements for products that only run under Windows as part of the Macintosh version.

Video editing programs provide facilities for performing the necessary preparation for delivery, but as with post-production tasks, their facilities are relatively crude, and it is better to use a dedicated application, such as Media Cleaner Pro. This provides improved control over settings for codecs, and integrates compression with the other tasks we have described for preparation. Suitable filters can be applied to mitigate the effects of aggressive compression, and multiple versions can be constructed at the same time, and integrated. Additional facilities that are especially useful at this stage of production include a data rate analyzer, and a split-pane preview window for showing the effects of different settings.

Streamed Video and Video Conferencing

It's one thing to play a pre-recorded video clip from a hard disk, DVD or even a CD-ROM; it's quite another to play video over a network. By this we mean delivering a video data stream from a remote server, to be displayed *as it arrives*, as against downloading an entire video clip to disk and playing it from there. Such *streamed video* resembles broadcast television, in that the source video is held on the server, which acts like a TV transmitter sending out the signal, which is played back straight away on a client machine. Downloading the entire clip is as though the TV company sent a courier round with a videotape whenever you wanted to watch a programme. Streamed video opens up the possibility of delivering live video, bringing one of the modes of conventional broadcasting to video on computers. It goes beyond conventional broadcast TV in this area, though, because it is not restricted to a single transmitter broadcasting to many consumers: any suitably equipped computer can act both as receiver and transmitter, so users on several machines can communicate visually, taking part in what is usually called a *video conference.*

The fundamental obstacle to streamed video is bandwidth. As we showed earlier, even the heavily compressed and down-sampled quarter-frames of SIF MPEG-1 video require a bandwidth of 1.86 Mbits per second. For now, therefore, decent quality streamed video is restricted to local area networks and T1 lines, ADSL and cable modems; dial-up Internet connections using V90 modems or basic rate ISDN (never mind slower modems) cannot handle the required data rate. Even where the bandwidth is available, the network has to be capable of delivering data with the minimum of delay, and without undue 'jitter' — a variation in the delay that can cause independently delivered video and audio streams to lose synchronization. We will return to this subject in Chapter 15.

⇨ It is likely that a large proportion of the population will be restricted to lower speed connections for some time to come. In the domestic sector, video is most likely to be streamed over broadcast digital television; video conferencing is unlikely to be an option, and the shape of interactivity over the digital TV networks will depend on market forces and politics at least as much as on the technological capabilities of the delivery medium.

This is not to say that streaming video over slow dial-up Internet connections is impossible, just that the quality is poor.

It may help you to understand the nature of what we will sometimes call *true streaming* by contrasting it with alternative methods of video delivery you may meet on the World Wide Web. The simplest method is embedded video, where a movie file is transferred from a server to the user's machine, where it is played back from disk once the entire file has arrived. A refinement of this method is called *progressive download* or *HTTP streaming*. With this mode of delivery, the file is still transferred to the user's disk, but it starts playing as soon as enough of it has arrived. This will be when the time it will take for the remainder to be downloaded is equal to the duration of the entire movie. This is illustrated in Figure 10.16. There is usually an appreciable delay before playback starts, since progressively downloaded movies are typically made with a data rate that exceeds the bandwidth of the network connection, in order to maintain quality. The movie file usually remains on the user's hard disk — at least in their Web browser's cache — after playback is completed. Thus, enough disk space to store the whole movie must be available, so progressive download cannot be used for huge files, such as complete feature films. Also, since an entire file is downloaded, this method of delivery cannot be used for live video, nor does it allow you to skip over parts of the file without downloading them.

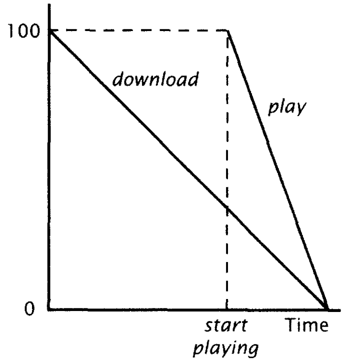


Figure 10.16
Progressive download

In contrast, true streaming video is never stored on the user's disk. A small buffer may be used to smooth out jitter, but effectively each frame in the stream is played as soon as it arrives over the network. This means that streams can be open-ended, so true streaming can be used for live video, and the length of a recorded movie that is streamed is limited only by the amount of storage available at the server, not by the viewer's machine. Random access to specific points in a stream is possible, except for live streams. True streaming is thus suitable for 'video on demand' applications.[27] Its drawback is that the network must be able to deliver the data stream fast enough for playback. Looked at from the other side, this means that the movie's data rate, and thus its quality, is restricted to what the network can deliver. Even with the best connections, Internet streamed video will not provide the picture quality of broadcast television.

⇨ Why bother, then? The answer is sometimes summed up in the slogan, 'Every Web site a TV station'. Streaming video server software is available free, or (for a small number of simultaneous streams) at a modest cost. Anyone with a permanent Internet connection can therefore transmit streamed video content; many ISPs provide streaming facilities to their customers who do not run their own servers. Internet streaming is thus much more accessible than conventional TV broadcasting facilities, and so it provides a new communication channel for minority voices and unconventional video (experimental and underground work, as well as the more notorious sort) that is denied accesss to conventional channels. As we will explain in Chapter 15, though, the broadcasting establishment may not be undermined to the extent that might seem possible... or desirable.

Another answer, more often given by conventional broadcasters, is that streamed video can be interactive. It is not entirely clear, though, what sort of interaction, if any, people want to have with video. Facilities that have been proposed for broadcast digital TV appear to be restricted to allowing viewers to select between different camera angles, and to call up 'instant replays' on demand. Initial provision of such facilities, for example by the satellite TV company BSkyB, is being targeted at sports coverage. It is not clear that this sort of interactivity, when provided over the Internet, would compensate for poor picture quality.

The two factors of democratic access and interactivity are combined in video conferencing, which may currently be the most successful application of streamed video, but is somewhat marginal from our viewpoint on multimedia, opening up, as it does, questions better considered in the context of computer-supported collaborative working.

The two leading architectures for streaming video over the Internet are *Streaming QuickTime* ,[28] and Real Networks' *RealVideo* . They

[27] The fact that no copy of the entire movie is held on the user's machine makes it even more attractive where copyright material is involved.

[28] True streaming was added to QuickTime with version 4.0.

have several features in common. Both are based on open Internet standard protocols, particularly RTSP, the Real Time Streaming Protocol, which is used to control the playback of video streams, which are carried over the network using the Real Time Protocol, RTP. These protocols are described in Chapter 15. RealVideo and Streaming QuickTime streams can both be embedded in Web pages; Streaming QuickTime streams can also be embedded in any application that incorporates QuickTime. Both architectures provide a means of providing several different versions of a movie compressed to match the requirements of different types of connections — a 28.8 kbps version, a 56 kbps version, a T1 version, and a cable modem version may all be provided so that, to the user, there only appears to be one movie; the server chooses the appropriate one to fit the speed of the user's connection. Tools are available for integrating servers for either format with capture facilities, allowing live video to be streamed to the Internet. As well as true streaming, progressive download is supported by both QuickTime and RealVideo. There is really little to choose between the technologies, but, since RealVideo has been around for longer, the production tools and server are more mature, and it is more likely that users will have the necessary plug-in and player to view it. In Streaming QuickTime's favour is the fact that it is just a variety of QuickTime, which can be embedded in any application, not just in a Web browser or its own special-purpose player.

Codecs for Streamed Video and Video Conferencing

Since Streaming QuickTime is essentially just QuickTime delivered in a special way, all of the QuickTime codecs can be used for streaming. However, to obtain sufficient compression to stream a movie over most network connections, these have to be used on their lowest quality settings, and the movie will generally have to be scaled down to a small frame size. Better results may be obtained using codecs designed specifically for video conferencing, which are designed for streaming video at low bit rates.

H.261 (named after the ITU-T Recommendation that defines it) was designed for video conferencing over ISDN. Since each ISDN data channel has a bandwidth of 64 kbps, H.261 was defined to operate at data rates that are an exact multiple of this rate. Hence, it is sometimes referred to as $p \times 64$, where, as p ranges from 1 to 30 (the maximum number of ISDN channels), the frame rate and the quality of the video stream scale up. An important feature of H.261 is that the recommendation specifies a maximum delay of 150 milliseconds due to the codec. This is because a video conference cannot tolerate longer delays without becoming disjointed — participants require immediate visual feedback to preserve the illusion of presence. Another singular feature is that the frame size is restricted to one of two values: *Common Intermediate Format (CIF)* is 352×288 pixels;[29] at lower data rates, a frame size of 176 × 144 pixels — i.e. one quarter that of CIF, hence *QCIF* — is used. Generally, if $p < 6$, that is, the available bandwidth is less than 384 kbps, QCIF must be used. There is a non-standard 'sub-QCIF' format of 128 × 96 for very low bandwidth applications. Whichever size is used, the video signal is chroma sub-sampled to 4:2:0.

H.261 uses a DCT-based compression algorithm, with motion compensation. It can be seen as a precursor of MPEG-1, but it does not use anything like B-pictures. Based on the additional experience gained from MPEG, a successor to H.261, *H.263* has been developed. This is very similar to MPEG, but it goes further by providing a means of combining a P-picture and a B-picture into a single *PB-frame.* H.263 was targeted at very low bit rates indeed: its primary target rate is about 27 kbps — the speed of a V.34 modem — and its specification is for bit rates less than 64 kbps. It achieves these rates by using all the compression techniques of MPEG — 4:2:0 chroma sub-sampling, DCT compression with quantization, run-length and variable-length encoding of coefficients, and motion compensation with forward and backward prediction — to compress a QCIF picture at frame rates as low as 3.5 frames per second. It does produce recognizable moving pictures at sufficiently low rates to be used for streaming and conferencing over dial-up Internet connections, but the quality is best described as 'better than nothing'.

For streaming over faster connections, the Sorenson codec is recommended for Streaming QuickTime. RealVideo uses its own proprietary codec.

Further Information

[Poy96] provides an expert description of digital video technology. [Oha93] and [Tho93] describe different aspects of film and video

[29] Depending on how you measure your active samples.

editing; parts of [Kaw92] are relevant, too. [Tud95] is a good short introduction to MPEG compression. [VD96] is an entertaining account of the use of digital techniques in high-budget film production. We will return to the technology underlying streaming video in Chapter 15.

Exercises

1. Under what circumstances will you need a hardware codec when working with video?
2. Specify the ideal video components of a system for the following projects:
   (a) A video interview with a celebrity, to be recorded at their home, and incorporated into a multimedia production on CD ROM for which you will need to extract high quality still images as well as the video footage.
   (b) A live video conference to be conducted across the Internet.
   (c) A video recording of the close finish of a 100 metre sprint, for a multimedia production on the Olympic games, which will allow the user to determine the winner for themselves.
   (d) A surveillance system in a bank which would provide visual evidence for a forensic multimedia presentation in a court of law.
   (e) Video footage of a fast flowing mountain stream, which is to be presented as smoothly as possible in slow motion, in a small sized window on a web page.
   (f) Time-lapse footage of the opening of a flower, for use on a Web site devoted to gardening.
3. Specify a practical system, based only on the equipment available to you, for realising the same projects.
4. Suppose you were involved in the design of a multimedia software application for domestic use, intended to allow users to create 'home' multimedia productions such as a record of a child's birthday to send on CD ROM to a grandparent

on the other side of the world. What assumptions will you make about video for this program, and what facilities will you supply?

5. How much storage would be occupied by a 90 minute feature film stored in each of the following forms?

   (a) CCIR 601.

   (b) MP@ML MPEG-2.

   (c) DV.

   (d) MPEG-1 in SIF.

   For each format, indicate what applications it is suitable for.

6. What effect will each of the following common video idioms have on a compression scheme that includes temporal compression?

   (a) Cuts;

   (b) Dissolves;

   (c) Hand-held camera work;

   (d) Zooms;

   (e) Pans.

   In which cases does motion compensation help?

7. Suppose an MPEG encoder uses the nine frame sequence IBBPBBPBB as a GOP. Draw a diagram showing the dependencies between the first eighteen frames of a compressed clip produced by this encoder. Show how the pictures would be reordered into bitstream order. Explain carefully why the pattern of I, P and B pictures in the bitstream order of the first nine frames is different from that of the second nine frames.

8. Capture a short clip[30] from TV or a video tape of each of the following:

   (a) A classic black and white *film noir.*

   (b) A contemporary soap opera.

   (c) A silent comic piece by Chaplin or Keaton.

   (d) A serious documentary.

   (e) A musical comedy in Glorious Technicolor.

[30] Pay attention to copyright and use these clips only for your own work. Do not attempt to publish or broadcast them in any medium, including from a Web site.

Observe how each has been edited to give it its particular character.

Re-edit each clip to give it the style and feel of one of the others. For example, make a silent comedy clip out of the documentary, or *film noir* clip out of the musical comedy.

(a) Do this simply with basic transitions, i.e. cuts, dissolves, and wipes.

(b) If post-production tools are available to you, take this further, by changing colours, contrast, or the quality of sound. For example, render the contemporary soap opera in grainy black and white with crackly sound, or the re-edited *film noir* in three-colour Technicolor.

9. A user starts to download a 905 kilobyte movie of 30 seconds duration, using progressive download over a connection that provides an average data rate of 2500 bytes per second. How long will it be before the movie starts to play? Why might the user experience jerky playback?

11 Animation

Animation may be defined as the creation of moving pictures one frame at a time; the word is also used to mean the sequences produced in this way. Throughout the twentieth century, animation has been used for entertainment, advertising, instruction, art and propaganda on film, and latterly on video; it is now also widely employed on the World Wide Web and in multimedia presentations.

To see how animation works, consider making a sequence of drawings or paintings on paper, in which those elements or characters intended to change or move during the sequence are altered or repositioned in each drawing. The changes between one drawing and the next may be very subtle, or much more noticeable. Once the drawings are complete, the sequence of drawings is photographed in the correct order, using a specially adapted movie camera that can advance the film a single frame at a time. When the film is played back, this sequence of still images is perceived in just the same way as the sequence of frames exposed when live action has been filmed in real time: persistence of vision causes the succession of still images to be perceived as a continuous moving image. If you wish to convey the illusion of fast movement or change, the differences between successive images in the sequence must be much greater than if the change is to be gradual, or the movement slow.

Etymologically, 'animate' means 'to bring to life', which captures the essence of the process: when played back at normal film or video speeds, the still characters, objects, abstract shapes, or whatever, that have been photographed in sequence, appear to come to life.

As film is projected at 24 frames per second, drawn animation, as we have just described it, technically requires 24 drawings for each second of film, that is, 1440 drawings for every minute — and even more for animation made on video. In practice, animation that does not require seamlessly smooth movement can be shot 'on 2s', which means that two frames of each drawing, or whatever, are captured rather than just one. This gives an effective frame rate of 12 frames per second for film, or 15 for NTSC video.

If an animation is made solely from drawings or paintings on paper, every aspect of the image has to be repeated for every single frame that is shot. In an effort to reduce the enormous amount of labour this process involves, as well as in a continuing search for new expressive possibilities, many other techniques of animation have been devised. The most well known and widely used — at least until very recently — has been *cel animation*. In this method of working, those elements in a scene that might move — Homer Simpson, for example — are drawn on sheets of transparent material known as 'cel', and laid over a background — the Simpsons' living room, perhaps — drawn separately. In producing a sequence, only the moving elements on the cel need to be redrawn for each frame; the fixed part of the scene need only be made once. Many cels might be overlaid together, with changes being made to different ones between different frames to achieve a greater complexity in the scene. To take the approach further, the background can be drawn on a long sheet, extending well beyond the bounds of a single frame, and moved between shots behind the cels, to produce an effect of travelling through a scene. The concepts and techniques of traditional cel animation have proved particularly suitable for transfer to the digital realm (see Figure 11.1).

Largely because of the huge influence of the Walt Disney studios, where cel animation was refined to a high degree, with the use of multi-plane set-ups that added a sense of three-dimensionality to the work, cel has dominated the popular perception of animation. It was used in nearly all the major cartoon series, from *Popeye* to the *Simpsons* and beyond, as well as in many full-length feature films, starting with *Snow White and the Seven Dwarfs* in 1937. However, from the very beginnings of moving pictures in the 1890s, animation has been successfully created by employing a variety of other means. Many artists do indeed work by drawing each frame

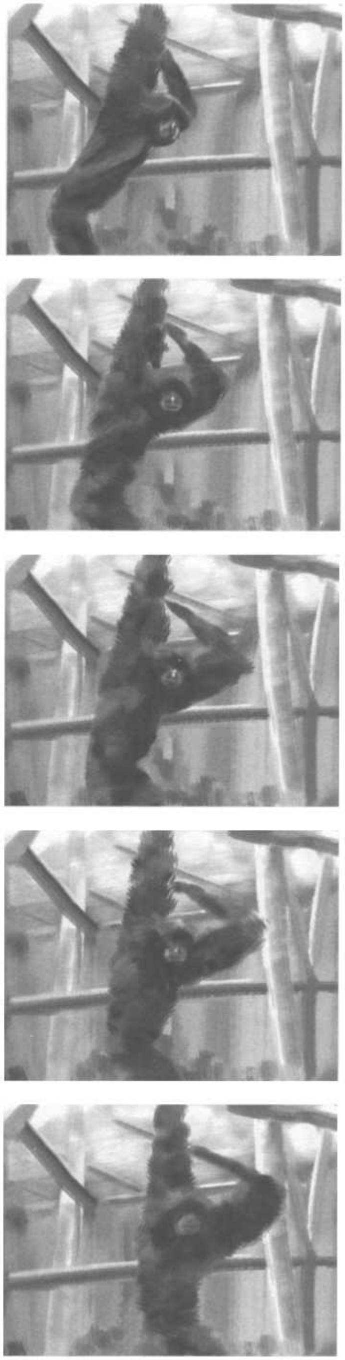


Figure 11.1
A cel-like digital animation

separately on paper, while others, even more painstaking, have painted directly on to film, or scratched the emulsion of blackened film stock; others have worked with sand or oil paint on glass, or chalks on paper or card, making changes to the created image between every shot; still others have manipulated front or back lit cut-outs under the camera — Terry Gilliam's very strange work for the *Monty Python* TV series is a well-known example of cut-out animation. Sometimes animators have invented a completely new way of working for themselves, such as Alexeieff and Parker's pin screen, in which closely spaced pins are selectively pushed through a board and lit so that the shadows they cast form an image, which is changed between each shot.

A distinct alternative to all of these essentially two-dimensional forms is three-dimensional, or *stop-motion* animation. This encompasses several techniques, but all use miniature three-dimensional sets, like stage sets, on which objects are moved carefully between shots. The objects may include articulated figures, whose limbs can be repositioned, or solid figures whose parts are replaced, or substituted, between shots, to produce an effect of gestures, walking, and so on. Figures and other objects made out of a malleable modelling material, such as Plasticine, may be used instead; these can be manipulated between shots, to produce both natural movement, and otherwise impossible changes and transformations. This latter form of animation — often called *clay animation* — has achieved recent prominence with the work of the Aardman studios whose output includes the *Wallace and Gromit* films.

Although it may be convenient to consider the various techniques of animation separately, hybrid forms of animation are often produced — mixing cel and 3-D, for example. There is also a long tradition of combining animation with live footage. The most celebrated example of this is perhaps *Who Framed Roger Rabbit?* (1988), but a mixture of live action and animation was employed in some of the earliest films ever made, including Georges Méliès well known 'trick films', and Max Fleischer's *Out of the Inkwell* series of the 1920s, which did much to popularize animation as a form of entertainment. Recently, the eager adoption of digital technology by the film industry has led to a substantially increased use of animation in conjunction with live action, particularly in special effects movies, such as *Jurassic Park*, that have provided some of the major box-office hits of the 1990s.

⇨ It is perhaps not always realised by an audience that much of what they perceive as 'special effects' has been achieved by basic animation techniques, whether traditional, as in the 1933 classic *King Kong* and many other monster movies, or digital, as in the TV series *Babylon 5*, for example.

All of the established forms of animation have their counterparts in the digital realm. Moreover, digital technology affords new opportunities for using animation and techniques derived from it in new contexts.

Captured Animation and Image Sequences

As we will see, digital technology has brought new ways of creating animation, but computers can also be used effectively in conjunction with the older methods discussed above, to produce animation in a digital form, suitable for incorporation in multimedia productions. Currently, preparing animation in this way — using digital technology together with a video camera and traditional animation methods — offers much richer expressive possibilities to the animator working in digital media than the purely computer-generated methods we will describe later in this chapter.

Instead of recording your animation on film or videotape, a video camera (either a digital camera or an analogue camera connected through a video capture card) is connected directly to a computer, to capture each frame of animation to disk — whether it is drawn on paper or cel, constructed on a 3-D set, or made using any other technique that does not depend on actually marking the film. Instead of storing the entire data stream arriving from the camera, as you would if you were capturing live video, you only store the digital version of a single frame each time you have set up a shot correctly. Most digital video editing applications provide a facility for *frame grabbing* of this sort. Premiere, for example, offers a `Stop Frame` command on its `Capture` menu. This causes a recording window to be displayed, showing the current view through the camera. You can use this to check the shot, then press a key to capture one frame, either to a still image file, or to be appended to an AVI or QuickTime movie sequence. You then change your drawing, alter the position of your models, or whatever, and take another shot. Frames that are unsatisfactory can be deleted; an option allows you to see a ghost image of the previously captured frame, to help with alignment and making the appropriate changes. When you have captured a set of frames that forms a sequence, you can save it as a QuickTime movie or a set of sequentially numbered image files (see below). The latter option is useful if you want to manipulate individual images in Photoshop, for example.

For certain types of traditional animation, it is not even necessary to use a camera. If you have made a series of drawings or paintings on paper, you can use a scanner to produce a set of image files from them. You can also manipulate cut-outs on the bed of a scanner, almost as easily as under a camera. A film scanner will even allow you to digitize animation made directly onto film stock. You might be able to use a digital stills camera instead of a video camera, provided it allows you to download images directly to disk. In all of these cases you are able to work at higher resolution, and with a larger colour gamut, than is possible with a video camera.

For drawn or painted animation you can dispense with the external form and the digitization process entirely by using a graphics program to make your artwork, and save your work as a movie or as a sequence of image files.

⇨ Sequences of image files provide a very flexible representation of an animation. Individual files can be opened in a graphics program to be altered; single files can be removed from the sequence, replaced or added. The sequence can then be imported into a video editing application and converted into an AVI or QuickTime movie. However, managing a collection of image files can become complicated, especially if you eventually want to import them into a video editing program. In order for this to be possible without tying you to a particular combination of programs, the files' names must conform to some convention. For example, on the Macintosh, Premiere can only import a sequence of PICT files if they are all in the same folder, and all the files have a suffix consisting of a period followed by the same number of digits, for example `Animation.001`, `Animation.002`, `...Animation.449`. (Failure to provide the necessary leading zeroes will have consequences that you can probably guess at.) If you make any changes to the set of images, you must take care not to disturb the numbering, or to adjust it if necessary.

Painter provides special support for animators wishing to draw or paint animations digitally,[1] in the form of features that resemble some of the ways in which traditional animators work with a stack

[1] Recall from Chapter 3 that Painter provides tools that simulate natural media, so you can make pictures that look (somewhat) as if they were made with conventional art materials.

of drawings. A Painter *frame stack* is a set of images, all of the same dimensions and colour depth. When you create or open a frame stack, you are provided with controls for navigating through it — advancing or retreating one frame, or going to the beginning or end of the stack — and playing it by having each frame in the stack displayed in turn. You create an animated sequence by painting on each of the frames in the stack. By turning on *onion skinning* you can make up to four frames adjacent to the one you are currently working on visible as ghost images, to help you line up the static parts of the animation and judge the amount by which you need to change the moving elements. This practice is modelled on the way animators sometimes use a light box to help construct a series of frames. By working on thin paper (hence the name 'onion skin') on a surface illuminated from below, they can lay the sheet they are working on over the one preceding or following it, which will show through the top sheet as a ghost image in just the same way.

Plate 20 is a screenshot of a Painter framestack being worked on, showing the navigational controls and the ghost images of the nearby frames. Note also the thumbnails of these frames.

Because it is a computer program, Painter (and similar applications) can help the animator in ways that are difficult or impossible when working with physical media. The most characteristically computerized facility is the ability to record a sequence of operations as a *script* that can subsequently be played back to apply the same operations to a new image. For example, you could record everything you did while drawing a static element in the first frame of a stack, and then, when you came to work on the next frame, simply play back the recording to redraw that element automatically, leaving you with just the changing parts of the animation to do by hand. You can also use a script to record the application of a combination of effects, such as adjustments to brightness and contrast, or colour balance, to one image, and then have the script applied to an entire frame stack to alter every frame in the same way.

Alternatively, a framestack may be used to create animation by progressively altering one single image, simulating traditional methods of working in paint or sand on glass, etc. Painter is especially appropriate for this, given its extensive and customisable range of natural media and brushes. In this case each new frame starts as a copy of the preceding frame, which is then partly erased and repainted to create the animation. (Painter allows you to add frames on to a framestack one at a time, with an option to make each new frame added a copy of the last in the stack at that time.)

As well as letting you save a frame stack as a sequence of image files, Painter will let you save it as a QuickTime or AVI movie. It will also let you open a movie, transforming it into a frame stack when you do so. This offers new possibilities. You can, for example, paint onto or otherwise alter original video material, which is one way of adding animation to live action.[2] Another option is to trace, frame by frame on a new framestack, selected elements from a live action video clip opened as a separate framestack. (Painter provides special facilities for doing this). This process, whether achieved digitally or by older means, is what is properly referred to as rotoscoping, and has long been used to create animation that accurately reproduces the forms and natural movements of people and animals.

⇨ Rotoscoping is named after the *rotoscope*, a device patented by Max Fleischer (of *Betty Boop* and original animated *Popeye* fame) in 1915. Fleischer's device projected movie footage, one frame at a time, onto a light table, giving a back projected still image over which the animator could place a sheet of animation paper. When the tracing of one frame was complete, the film was advanced to the next by means of a hand crank.

Instead of using a set of still image files to hold an animation sequence, you can sometimes use a single 'image' file to hold several images. While a surprising number of file formats — including, but not exclusively, formats intended for use with animation software — offer this facility, by far the most common is GIF.

GIF files' ability to store a sequence of images has been used to provide a cheap and cheerful form of animation for Web pages. Most Web browsers will display each image contained in a GIF file in turn when they load the file. If the displaying happens fast enough, the images will be seen as an animation. The GIF89a version of the format provides for some optional data items that control the behaviour of an *animated GIF*, as these files are called. In particular, a flag can be set to cause the animation to loop, either for a stipulated number of times or indefinitely, and a minimum delay between frames, and hence a frame rate, can be specified. However, animated GIFs do not provide a very reliable way of adding animated features to Web pages. As with most aspects of a browser's behaviour, the way in which animated GIFs are displayed can be changed by users — looping can be turned off, animation can be

[2] In computer graphics circles, this process of painting on to existing video frames is sometimes called 'rotoscoping', but the use of the term is inaccurate, as explained below.

prevented, and if image loading is disabled, animated GIFs will not appear at all — and not all browsers offer a proper implementation. The main advantage of animated GIFs is that they do not rely on any plug-in, or the use of scripting (see Chapter 14).

Several free or inexpensive utilities are available on the major platforms for combining a set of images into a single animated GIF; Painter can export a frame stack as an animated GIF, and Premiere and Flash allow you to save a movie in this form, too. Potentially, therefore, GIF files can be used to store any form of animation. However, even when GIF animation is properly implemented and enabled, it has many shortcomings. You cannot add sound; you are restricted to a 256 colour palette; your images are losslessly compressed, which may conserve their quality, but does not provide much compression, a serious consideration that effectively prevents the use of this format for any extended animation sequences. Usually, each frame of an animated GIF is displayed by the browser as it arrives. Network speeds mean that there may be excessive, and probably irregular, delays between frames, making any frame rate that may be specified in the file irrelevant. However, if an animation is set to loop, once it has played through the first time it will have been copied into the browser's local cache (unless it is too big), and subsequent loops will play at a speed only limited by the user's processor and disk (which are completely unknown to the animator). In general, there is little chance of an animated GIF consistently playing back at a sufficiently high frame rate to give smooth animation, unless it is small (and even then, not on older computers). Usually, therefore, animated GIFs are not used for real animation, but for more stylized changing images, often resembling neon advertising signs. Possibly for this reason, by association of ideas, Web page advertising is what animated GIFs are most often used for. It is probably fair to say that, because of the ease with which animated advertisements can be incorporated into a Web page by almost anybody, they have been used for some of the worst animation ever produced.

For animation of any duration, especially if it is accompanied by sound, the best results will be achieved using a video format, and QuickTime has become the standard. Once you have captured or painted an animation sequence and saved it as, or converted it to, QuickTime in the manner previously described, what you have is just an ordinary QuickTime movie, so it can be edited, combined with other clips, have effects applied, and be prepared for inclusion in a multimedia production, just like any other video clip. However, animation clips may have some distinctive features which affect the way you deal with them. In particular, certain styles of drawn animation tend to feature simplified shapes and areas of flat colour. (This is *not* invariably the case, the characteristics of the images depend on the individual animator's style.) Material of this type may be more amenable to lossless compression than other types of video. QuickTime's *Animation* codec is designed to take advantage of the characteristics of simple cartoon-style drawn animation, which, as we will see later in this chapter, are often shared by computer-generated 3-D animation. Compression is based on run-length encoding (RLE), and, when the codec is used at its highest quality setting, is lossless. There is also a lossy mode that can be used to achieve higher compression ratios. Because it is based on RLE, this codec can compress areas of flat colour well, which is what makes it suitable for animation in the particular styles just mentioned. It works best on images made in the computer: the signal noise that may be introduced by capturing from a video camera can interfere with its effectiveness.

'Digital Cel' and Sprite Animation

Our earlier description of cel animation may have put you in mind of layers, as described in Chapter 3. Layers allow you to create separate parts of a still image — for example, a person and the background of a scene they are walking through — so that each can be altered or moved independently. The frames of an animated sequence can be made by combining a background layer, which remains static, with one of more animation layers, in which any changes that take place between frames are made. Thus, to create an animation, you begin by creating the background layer in the image for the first frame. Next, on separate layers, you create the elements that will move; you may want to use additional static layers in between these moving layers if you need to create an illusion of depth. After saving the first frame, you begin the next by pasting the background layer from the first; then, you add the other layers, incorporating the changes that are needed for your animation. In this way, you do not need to recreate the static elements of each frame, not even using a script.

Where the motion in an animation is simple, it may only be necessary to reposition or transform the images on some of the layers. To take a simple example, suppose we wish to animate the movement of a planet across a background of stars. The first frame could consist of a background layer containing the star field, and a foreground layer with an image of our planet. To create the next frame, we would copy these two layers, and then, using the move tool, displace the planet's image a small amount. By continuing in this way, we could produce a sequence in which the planet moved across the background. (If we did not want the planet to move in a straight line, it would be necessary to rotate the image as well as displace it, to keep it tangential to the motion path.) Simple motion of this sort is ripe for automation, and we will see in a later section how After Effects can be used to animate Photoshop layers semi-automatically.

Using layers as the digital equivalent of cel saves the animator time, but, as we have described it, it does not affect the way in which the completed animation is stored: each frame is saved as an image file, and the sequence will later be transformed into a QuickTime movie, an animated GIF, or any other conventional representation. Yet there is clearly a great deal of redundancy in a sequence whose frames are all built out of the same set of elements. Possibly, when the sequence comes to be compressed, the redundant information will be squeezed out, but compressing after the event is unlikely to be as successful as storing the sequence in a form that exploits its redundancy in the first place. In general terms, this would mean storing a single copy of all the static layers and all the objects (that is, the non-transparent parts) on the other layers, together with a description of how the moving elements are transformed between frames.

This form of animation, based on moving objects, is called *sprite animation*, with the objects being referred to as *sprites*. Slightly more sophisticated motion can be achieved by associating a set of images, sometimes called *faces*, with each sprite. This would be suitable to create a 'walk cycle' for a humanoid character, for example (see Figure 11.2). By advancing the position of the sprite and cycling through the faces, the character can be made to walk.

QuickTime supports sprite tracks, which store an animation in the form of a 'key frame sample' followed by some 'override samples'. The key frame sample contains the images for all the faces of all the sprites used in this animation, and values for the spatial properties

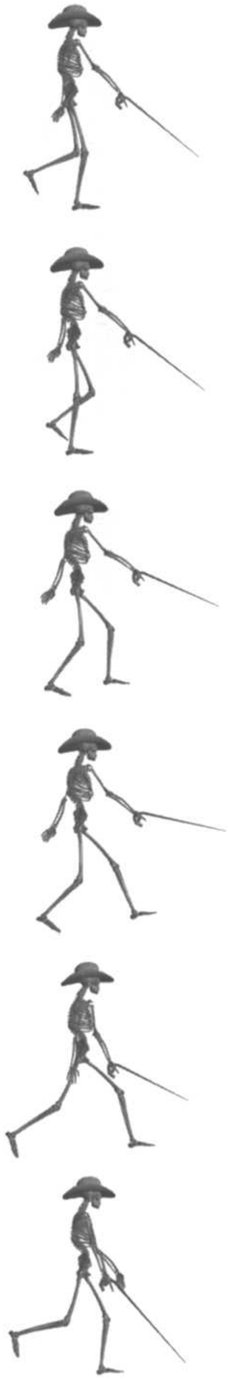


Figure 11.2
Sprite faces for a walk cycle

(position, orientation, visibility, and so on) of each sprite, as well as an indication of which face is to be displayed. The override samples contain no image data, only new values for the properties of any sprites that have changed in any way. They can therefore be very small. QuickTime sprite tracks can be combined with ordinary video and sound tracks in a movie.

We have described sprite animation as a way of storing an animated sequence, but it is often used in a different way. Instead of storing the changes to the properties of the sprites, the changed values can be generated dynamically by a program. Simple motion sequences that can be described algorithmically can be held in an even more compact form, therefore, but, more interestingly, the computation of sprite properties can be made to depend upon external events, such as mouse movements and other user input. In other words, the movement and appearance of animated objects can be controlled by the user. This way of using sprites has been extensively used in two-dimensional computer games, but it can also be used to provide a dynamic form of interaction in other contexts, for example, simulations. We will see something similar in Chapter 14 when we consider animating text on Web pages.

Key Frame Animation

During the 1930s and 1940s, the large American cartoon producers, led by Walt Disney, developed a mass production approach to animation. Central to this development was division of labour. Just as Henry Ford's assembly line approach to manufacturing motor cars relied on breaking down complex tasks into small repetitive sub-tasks that could be carried out by relatively unskilled workers, so Disney's approach to manufacturing dwarfs relied on breaking down the production of a sequence of drawings into sub-tasks, some of which, at least, could be performed by relatively unskilled staff. Disney was less successful at de-skilling animation than Ford was at de-skilling manufacture — character design, concept art, storyboards, tests, and some of the animation, always had to be done by experienced and talented artists. But when it came to the production of the final cels for a film, the rôle of trained animators was largely confined to the creation of *key frames.*

We have met this expression already, in the context of video compression and also in connection with QuickTime sprite tracks. There, key frames were those which were stored in their entirety, while the frames in between them were stored as differences only. In animation, the meaning has a slightly different twist: key frames are typically drawn by a 'chief animator' to provide the pose and detailed characteristics of characters[3] at important points in the animation. Usually, key frames occur at the extremes of a movement — the beginning and end of a walk, the top and bottom of a fall, and so on — which determine more or less entirely what happens in between, but they may be used for any point which marks a significant change. The intermediate frames can then be drawn almost mechanically by 'in-betweeners'. Each chief animator could have several in-betweeners working with him[4] to multiply his productivity. (In addition, the tedious task of transferring drawings to cel and colouring them in was also delegated to subordinates.)

In-betweening (which is what in-betweeners do) resembles what mathematicians call *interpolation*: the calculation of values of a function lying in between known points. Interpolation is something that computer programs are very good at, provided the values to be computed and the relationship between them can be expressed numerically. Generally, the relationship between two key frames of a hand-drawn animation is too complex to be reduced to numbers in a way that is amenable to computer processing. But this does not prevent people trying — because of the potential labour savings.

All digital images are represented numerically, in a sense, but the numerical representation of vector images is much simpler than that of bitmapped images, making them more amenable to numerical interpolation. To be more precise, the transformations that can be applied to vector shapes — translation, rotation, scaling, reflection and shearing — are arithmetical operations that can be interpolated. Thus, movement that consists of a combination of these operations can be generated by a process of numerical in-betweening starting from a pair of key frames.

Macromedia's *Flash* is designed for producing simple vector animations, primarily for display on the World Wide Web. Although it can only produce very crude animation, best suited for such ornaments of contemporary civilization as rotating logos and dancing text, it illustrates in a simple form the use of key frames and interpolation as an aid to animation.[5]

[3] The mass production approach to animation is almost invariably associated with cartoons featuring characters.

[4] This 'him' is not a casual slip: the big cartoon studios of those days did not have what we would consider an enlightened attitude to women as animators.

[5] You will also encounter some of the same concepts when we describe time-based multimedia authoring in Chapter 13.

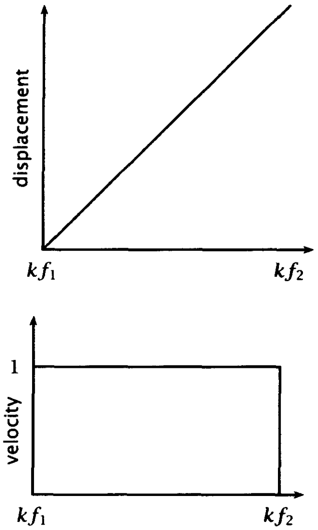


Figure 11.3
Linearly interpolated motion

A Flash animation is organized using a *timeline*, a graphical representation of a sequence of frames, similar to the timeline in video editing applications. Animations *can* be built up a single frame at a time, by inserting individual frames into the timeline, but here we are more interested in inserting just the key frames. A key frame is created simply by clicking in the timeline at the appropriate point and using the `insert keyframe` menu command or its command key equivalent. This just establishes the existence of the key frame; its content must also be added before in-betweening can be specified.

Flash's *stage* is a sub-window in which frames are created by arranging objects, and which is also used to preview animations. Objects can be created on the stage using some built-in drawing tools, similar to but less comprehensive than those of Illustrator or Freehand, or they can be imported from those applications. Bitmap images, in formats including JPEG and PNG, may also be imported and auto-traced to make vector objects; images can be used in bitmap form within a Flash frame, but cannot then be rotated or scaled without potentially degrading the image. Comprehensive support is provided for text; characters in outline fonts can be decomposed into their component outline paths, which can be edited or animated separately.

Graphical objects can be stored in a library in a special form, called a *symbol*, that allows them to be reused. Since interpolated animations, almost by definition, reuse objects, key frames can only be built out of symbols. To animate a symbol, a key frame is selected in the timeline, and the symbol is placed on the stage; the current frame is then moved to the next key frame, and the symbol is moved, scaled, or otherwise transformed, to create the new frame. (The symbol is behaving like a sprite.) Double-clicking anywhere in the timeline between the two key frames brings up a dialogue, which allows you to specify that the in-between frames should be interpolated ('tweened', as Flash puts it). The current frame is moved to the next key frame, if there is one, and the process is repeated. The animation is built up as a sequence of automatically tweened segments, between key frames that have been arranged by hand.

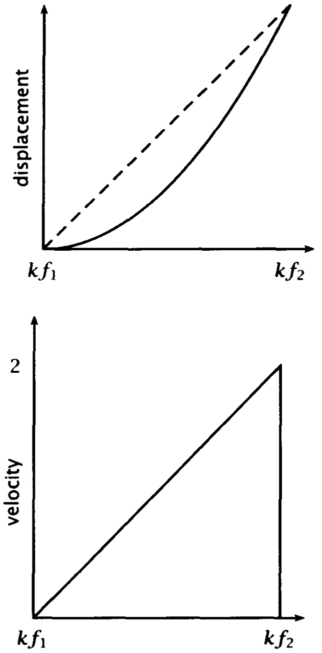


Figure 11.4
Easing in

Moving a single symbol about the stage offers little in the way of artistic gratification. Each Flash animation, though, can have an arbitrary number of layers, allowing the independent animation of many symbols. To further ease the work of animating motion, an object can be moved along a path drawn on a hidden layer; this *motion path* need not be a straight line, so movements that would have to be constructed using many key frames if only rectilinear movements were possible can be achieved in a single tweened sequence with just two key frames. Finally, a symbol can itself be a small animation, that plays within the main movie, so that different motions can be combined. For example, a symbol can consist of an object rotating; this can then be animated on a path, so that the object rotates as it follows the path. The analogy with traditional cel animation — of a basic kind — is fairly complete.

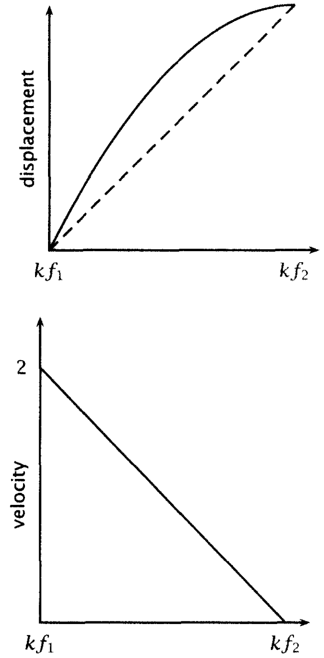


Figure 11.5
Easing out

Flash's interpolation is linear. If we just consider interpolated motion in a straight line, this means that a symbol moves an equal distance between each frame, the distance moved being the total distance between its positions in the starting and ending key frames, divided by the number of frames in the sequence. Putting it more simply, the symbol moves at a constant velocity, which causes two problems.

First, motion begins and ends instantaneously, with objects attaining their full velocity as soon as they start to move, and maintaining it until they stop. Nothing really moves like that. To produce a more natural movement, Flash borrows a technique from hand-made animation: the transition from stasis to movement is made more gradual by using smaller, slowly increasing, increments between the first few frames (i.e. the object accelerates from a standstill to its final velocity), a process referred to as *easing in*. The converse process of deceleration is called *easing out*. Figure 11.3 shows the way the horizontal displacement and velocity of an object changes with time when it is moved from an initial position in key frame 1 of $(0, 0)$ to a final position in key frame 2 of $(50, 50)$, using linear interpolation over 50 frames. Figures 11.4 and 11.5 show how the change might be modified when the motion is eased in or out — we have shown a style of easing that uses quadratic interpolation, that is, the acceleration is constant. More complicated styles are possible and might be preferred. Flash allows the animator to set the degree of easing using a slider that moves from maximum easing in, through a constant velocity, to maximum easing out. In effect, this moves the displacement curve from one like Figure 11.4, via similar curves with less pronounced bulge, through Figure 11.3 and beyond to Figure 11.5. (That is, the acceleration goes from some maximum positive value, through zero, to a maximum negative value.)

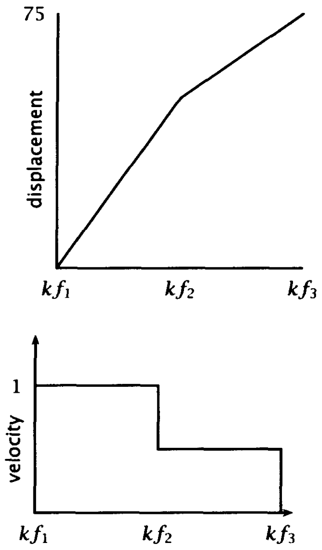


Figure 11.6
Abrupt change of velocity

The second problem with linear interpolation can be seen in Figure 11.6, which shows how displacement and velocity change if we now append to our original sequence a second one of 50 frames, during which our object moves from its position in key frame 2 of (50, 50) to a new position at (75, 75) in key frame 3. Because each sequence is interpolated separately as a straight line, there is a sharp discontinuity at key frame 2; as the velocity graph clearly shows, this will appear as a sudden deceleration at that point in the animation. Again, this is an unnatural sort of movement, that will rarely be what is desired. By clever manipulation of the easing slider, it would be possible to smooth out this abruptness, but a more general solution to the problem is available. In Chapter 4, we stressed that Bèzier curves' most attractive property is that they can be joined together smoothly by aligning their tangent vectors. By using Bèzier curves instead of straight lines to interpolate between key frames, smooth motion can be achieved. Note that we do not mean that objects should follow Bèzier shaped paths, but that the *rate* at which their properties change should be interpolated using a Bèzier curve. Flash does not offer the option of any form of non-linear interpolation but other, more elaborate, key frame animation systems do. Although we have only considered displacement in one direction, you should be able to see that similar considerations apply to any transformations that might be interpolated.

Although we claimed earlier that interpolation of vector graphics is easier than interpolation of bitmapped images, you can probably see that there is little advantage, for any but the most basic animation, in using vectors — rather the reverse, because of the stylistic limitation imposed by vector drawing. We will see in the next section that key frame animation can be applied to bitmaps to produce more visually interesting results. The pay-off from using vectors comes in the compact representation that can be used for the final animation. As with vector still image formats, objects can be represented compactly; additionally, as with sprite tracks, their movements can also be represented compactly. For example, the native file format of Flash is *SWF ('Shockwave Flash')*. An SWF file consists of items, which are divided into two broad classes: definitions and control items. The former are used to store definitions of the symbols used in an animation into a dictionary; the latter are instructions to place, remove, or move a symbol (identified by its name in the dictionary). Placement and movement are specified using transformation matrices, so that the position and any scaling or rotation are specifed at the same time. An SWF file is thus rather like a program, comprising as it does definitions of some objects and instructions that manipulate them. SWF data is encoded in a binary form and compressed, resulting in very small files. For example, an extremely simple tweened animation consisting of twenty frames, during which a red circle moves along a motion path, growing in size as it does so, occupies 450 bytes as an SWF file; the same animation, stored as a QuickTime movie using the Animation compressor (which should be particularly effective on such material) at a medium quality setting, occupies over 19 kbytes.

Motion Graphics

Interpolation between key frames can be applied to bitmapped images. Since bitmaps do not contain identifiable objects, the use of layers to isolate different elements of an animation is essential. The analogy with cel animations is more or less complete — each layer is like a transparent sheet of acetate with something painted on it. Layers can be moved independently, so an animation can be constructed by placing different elements on different layers, and moving or altering the layers between frames. Where the movement or alteration is easily described algorithmically, it can be interpolated between key frames, just as in-betweeners interpolate between a chief animator's key frames. Typically, between key frames, a layer may be moved to a different position, rotated or scaled. These geometrical transformations are easily interpolated, but since we are now concerned with bitmapped images, they may require resampling, and consequently cause a loss of image quality, as we explained in Chapter 5.

After Effects is the leading desktop application for animation of this kind. Because of their shared provenance, After Effects works well in conjunction with Photoshop and Illustrator. A Photoshop image can be imported into After Effects, with all its layers — including adjustment layers — and alpha channels intact; an Illustrator drawing can be imported and rasterized, again with its layers intact. A common mode of working, therefore, is to use the tools and facilities of Photoshop or Illustrator to prepare the elements of an animation on separate layers, and import the result into After Effects where the layers are animated. Photoshop images should be prepared at an appropriate resolution and size for your intended delivery medium. If they are to be scaled down, they must be large enough to accommodate the maximum reduction that will be applied. Illustrator files can either be rasterized when they are imported and then treated as bitmaps, or they can be continuously rasterized for each frame in the animation. This means that if they are scaled, for example, no detail will be lost.

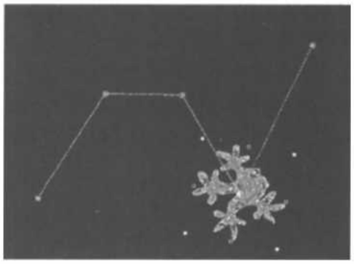


Figure 11.7
Linear spatial interpolation

The simplest animations are made by repositioning layers, either by dragging them or by entering coordinates, and interpolating motion between key frames. After Effects supports both linear and Bézier interpolation, in both space and time. Figure 11.7 shows a tree frog moving along a motion path, defined by linear interpolation between five key frames — the changes in direction are abrupt, as if the frog was a ball bouncing off a hard surface. Figure 11.8 shows the motion path through the same key frames, with Bézier interpolation[6] — the changes in direction are smooth, like a well steered bicycle. These are different forms of spatial interpolation, which are set by moving the layer in the window that shows the image. Temporal interpolation affects the rate of change of position with respect to time. Again, this may be linear, with a constant velocity and instantaneous starting and stopping, as discussed earlier in connection with Flash, or Bézier, where the acceleration is smooth. Temporal interpolation is set in a *value graph* for the corresponding property. The value graphs displayed by After Effects for our frog's velocity with linear and Bézier interpolation are shown in Figures 11.9 and 11.10, respectively. The temporal and spatial interpolation methods are independent: you can use linear temporal interpolation with Bézier motion paths, and vice versa.

Interpolation can be applied to other properties of a layer. In particular, its angle can be varied, so that it appears to rotate. Angles may be set by hand in key frames and interpolated, or the rotation may be determined automatically in conjunction with movement, to maintain the orientation of a layer with respect to its motion path. Scaling, which may be used as a perspective effect to convey the impression of approaching or receding movement, or as a zoom in or out, can also be set in key frames and interpolated.

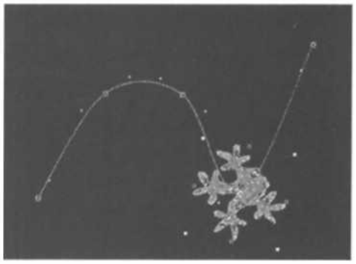


Figure 11.8
Bézier spatial interpolation

The degree of control over the interpolation of these spatial properties offered by After Effects is considerable. Using a conventional Bézier pen tool, the graphs showing how a value varies with time may be redrawn. Key frames are inserted automatically when control points are added to the graph. Absolute values may be entered numerically, allowing complete control over positioning

[6] After Effects actually provides 'auto Bézier' interpolation, where the handles are set automatically on the basis of the neighbouring key frames; 'continuous Bézier' interpolation, where handles are adjusted manually, as in Illustrator, with the direction lines at joins constrained to be the same length and colinear; and 'Bézier' interpolation, where the two direction lines at a join are moved independently.

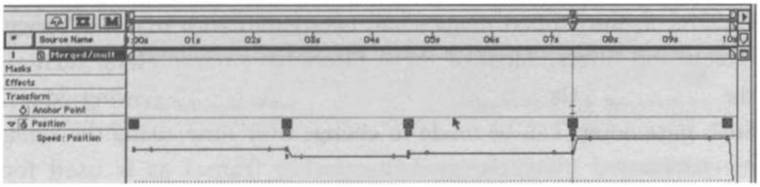


Figure 11.9
Velocity graph for linear temporal interpolation

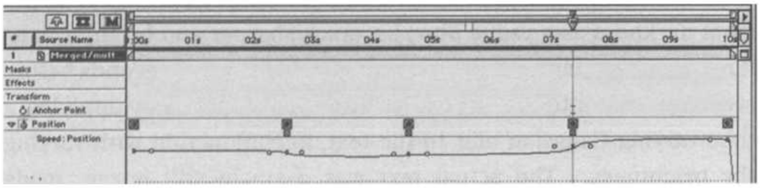


Figure 11.10
Velocity graph for Bézier temporal interpolation

and the rate of movement. Nevertheless, the type of motion that can be produced by interpolating the position, angle, and size of a single layer is restricted. Objects appear to move as a whole, with an unrealistic gliding motion, resembling that seen in simple, low-budget, cut-out animation, as favoured for some pre-school entertainment and education on television. Production of realistic movement — for example, making our frog walk or jump — requires different parts of an object, such as the legs, body, and eyes of a frog, to be placed on separately animated layers. An understanding of the way things move in real life is needed to coordinate the movement of the layers, and there may be little scope for automation.

Key frame animation of bitmapped images is therefore more frequently used for stylized motion. As we mentioned in Chapter 10, travelling mattes are often made by animating a still image in After Effects. Another popular application is the animation of text. Individual characters or words can be placed on layers and animated, just like any other layer, or text may be placed on a path, as in Illustrator, and then moved along that path over time.

The bitmapped representation allows other properties of the image besides its position, angle and size to be altered over time. So, in addition to geometrical transformations, more radical time-based alterations of the layers can be achieved. As we described in

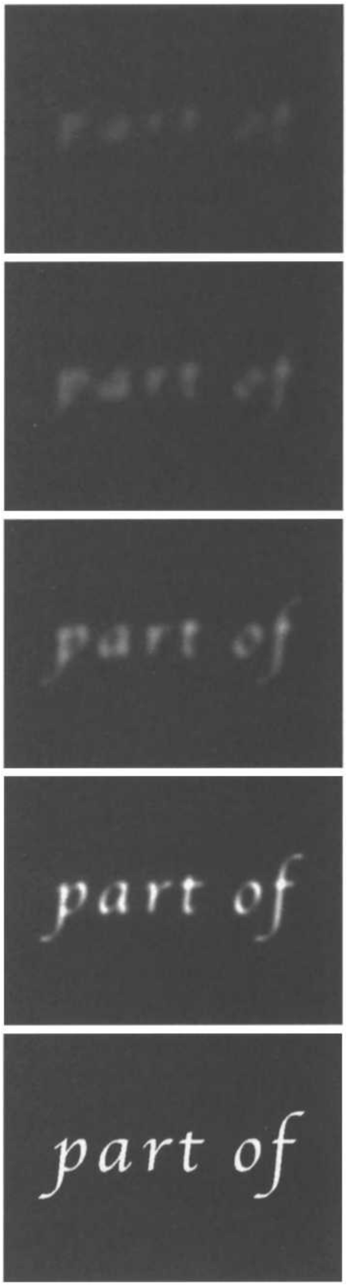


Figure 11.11
Frames from a title sequence

Chapter 5, bitmapped images can be treated with many different effects and filters. Most of these filters have parameters, such as the radius of a Gaussian blur, or the brightness of glowing edges. Such parameters can be made to change over time, using the same mechanism of interpolation between key frames as is used for interpolating motion. Doing so allows some unique effects to be generated.

For example, Figure 11.11 shows a sequence of frames extracted from the title sequence of a short film. The title, 'part of', emerges from darkness as a vague blur, becomes sharper and brighter until it reaches maximum clarity, where it is held for a few seconds before receding back into the darkness. This was achieved by applying a time-varying Gaussian blur to the text, in conjunction with varying the brightness. The actual text was a single still image, made in Photoshop, that was otherwise unaltered. Figure 11.12 shows the graphs of the Gaussian blur and brightness values that were used. The blur starts with a very high value, which renders the text illegible; in the same key frame, the brightness is substantially reduced. The values are interpolated to a point where the blur is removed and the brightness brought right up. Bézier interpolation is used to ensure a smooth fade up. The values are held constant between the middle two key frames, as shown by the flat portions of the graphs, and then, to make the title fade back into nothing, a symmetrical interpolation is used to a final key frame where the values are identical to those at the beginning.

The effects that can be achieved using time-varying filters on bitmapped images have more in common with graphic design than with mainstream cartoons or art animations. They are often known by the more suggestive name of *motion graphics*. Many of the techniques first appeared in title sequences for feature films, and credit sequences remain a typical application.

Motion graphic effects can be applied to live video as well as to still images. That is, a video clip can be moved across the screen, scaled and rotated, and treated with time-varying filters. By arranging clips on separately animated layers, elaborate compositions can be constructed. The result is fast becoming the time-based equivalent of the Layered Look in graphic design.

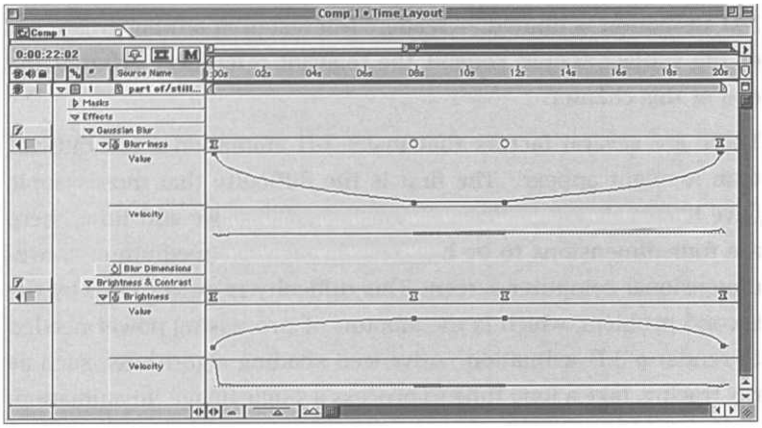


Figure 11.12
Value graphs used for the title sequence

3-D Animation

3-D animation is easy to describe, but much harder to do. No new concepts beyond those already introduced in this chapter and in Chapter 4 are needed to understand the essence of the process. The properties of 3-D models are defined by numerical quantities. Changing the numbers changes properties such as an object's position in space, its rotation, its surface characteristics, and even its shape. The intensity and direction of light sources and the position and orientation of a camera are also numerically defined. In order to animate a three-dimensional scene, therefore, all that is necessary is to set up an initial scene and render it as the first frame of the animation, make some changes to parameters, render the next frame, and so on. Because the values that are being changed are numerical, some kinds of change can be interpolated between key frames; a time line can be used as a convenient way of organizing the animation, and motion paths in three dimensions (often 3-D Bézier splines) can be used to describe movement.

Whereas simple 3-D animations, such as tumbling logos and rotating globes, really can be made very easily — and there are dedicated packages available for such tasks — high-quality photo-realistic animations, such as those employed in television advertisements, music videos, and film special effects, require huge resources: time, processor power and memory, dedicated software, and above all, highly skilled specialized personnel. Multimedia production can rarely afford these resources. For this reason our description of

3-D animation is limited — readers interested in a fuller treatment of the subject should consult the relevant references given at the end of this chapter.

There are several factors that make 3-D animation more difficult than it might appear. The first is the difficulty that most people have in visualizing in three dimensions. When we add time, there are four dimensions to be handled through the medium of a two-dimensional computer screen. This difficulty is exacerbated by the second problem, which is the amount of processing power needed to render a 3-D animation. Advanced shading algorithms, such as ray tracing, take a long time to process a single image. In animation, we need at least twelve, and up to thirty, images to be processed for every second of completed animation. This makes generating fully rendered previews impossible on anything but the most powerful workstations or networks of distributed processors.

Large budgets, patience and practice can overcome these problems, but what remains is the necessity to change a large number of parameters in such a way as to produce convincing movements. At the very highest level of 3-D computer-generated animation, the solution that is adopted is to provide a rich interface giving the animator complete control over movement. For animating figures, this resembles the type of control used by a puppeteer to control a mannequin — body suits equipped with motion sensors, like those used to control animatronic puppets, are even used sometimes.[7] This is in marked contrast to the approach taken to animation by computer programmers, whose well-trained instinct is to try to automate everything. In an attempt to overcome the limitations of simple interpolation schemes, considerable research efforts have been expended on ways of producing convincing motion in three dimensions automatically.

One of the key approaches is to provide certain kinds of behaviour that can be applied to objects, and the way they interact. A simple type of behaviour consists of making one object point at another. This is most useful when the pointing object is a camera or light. If a camera is pointed at an object, it will maintain that object in its field of view, no matter where it moves; a spotlight pointed at an object will automatically follow it, as a real spotlight follows a dancer on a stage, for example. Actual objects in the scene can also be made to point at each other: a sunflower can be made to always point at the sun, for example. A variation on this behaviour is to have one object track another, i.e. follow its motion at a distance. This can be used

[7] At this level, computer-generated animation is certainly no simpler than traditional animation, and is only used in order to produce results that cannot be achieved economically using physical means.

crudely to animate chase scenes. Like pointing, it can be applied to a camera, allowing it to follow an object or character through a scene, even in places where it would be physically impossible for a real camera to go.

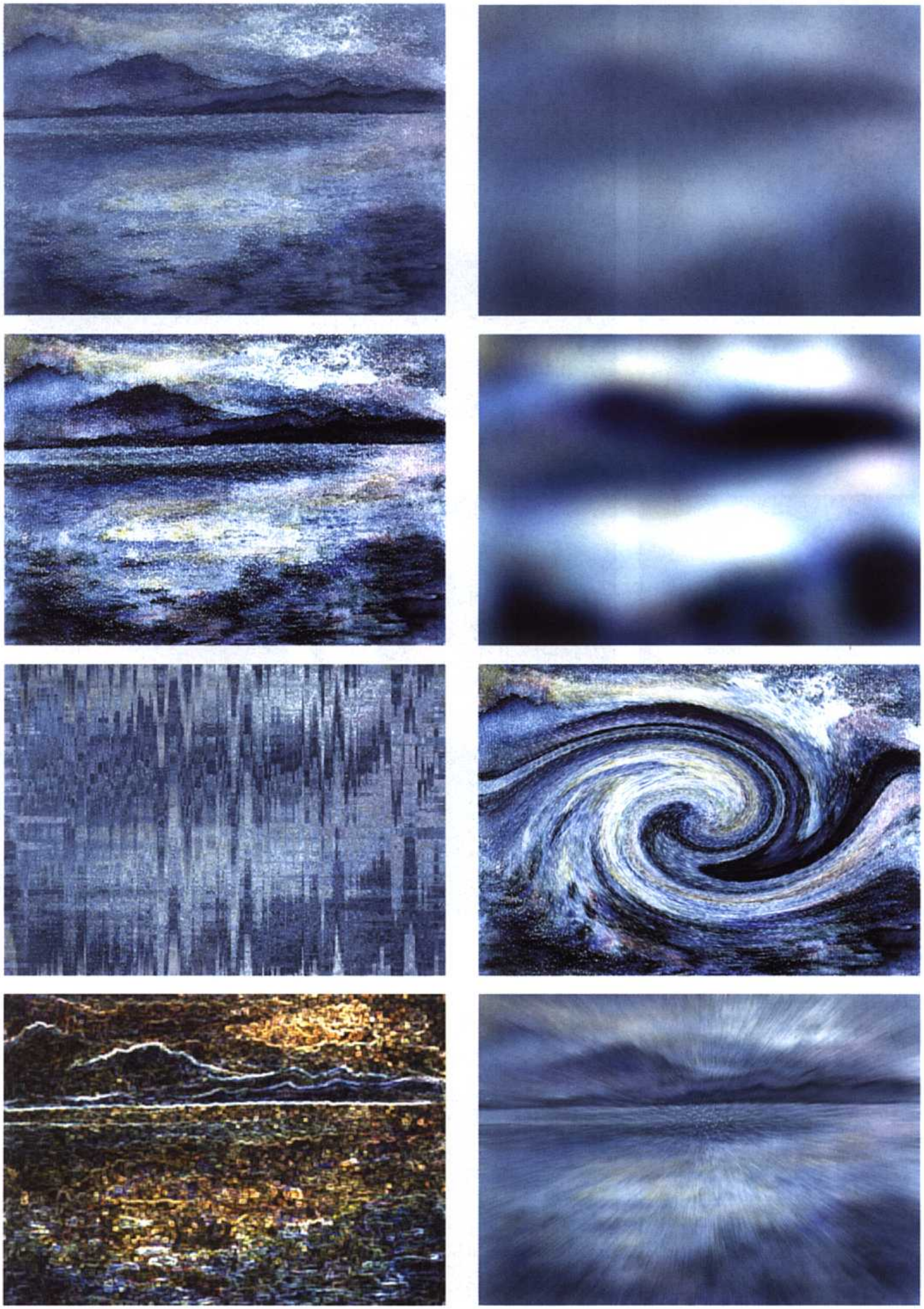
Some 3-D animation systems incorporate behaviour based on the physical laws of motion. For example, they allow the user to specify that an object should accelerate from zero under the influence of an external force whose magnitude is specified as a parameter. Taking this further, moving objects can be made to collide realistically, or bounce off solid surfaces. These behaviours are based on simple laws of physics that encapsulate the possible motion in a few equations. Unfortunately, many realistic types of movement cannot be so easily described, and other methods must be used.

Kinematics is the study of the motion of bodies without reference to mass or force. That is, it is only concerned with how things can move, rather than what makes them do so. In animation, it is most useful in connection with jointed structures, such as the limbs of human or animal figures. Because they are joined together, the various parts of your arm, for example, can only move in certain ways. To produce realistic movement, a 3-D model of an arm must obey the same *kinematic constraints* as a real arm: if the upper arm is raised, the lower arm and hand must come with it, for example. Whereas, in reality, it is the motion of the upper arm that propels the rest of the arm, in an animation system, modelling movement in this way — from the beginning of a chain of connected elements to the end — is not very helpful to the animator. It is more useful to be able to position the object which is at the end of the chain — a hand, say — and then make the other elements — the rest of the arm — move to accommodate it. It is usually the extremities that impose limitations on movement; when walking, a foot must stay above the ground, resting on it at each step, for example. It takes considerable understanding of the way limbs move to ensure that this will happen correctly by moving the thigh, so it is preferable for the software to work out the movements of the leg from the animator's placement of the foot, and so on. This type of modelling is called *inverse kinematics*, since it works backwards from effect to cause. Inverse kinematics can be applied to any structure that is modelled as a linked chain of objects. It is routinely provided by 3-D animation programs that support such structures. Poser, for example, can be set to automatically apply inverse kinematics to the arms and legs of figures.

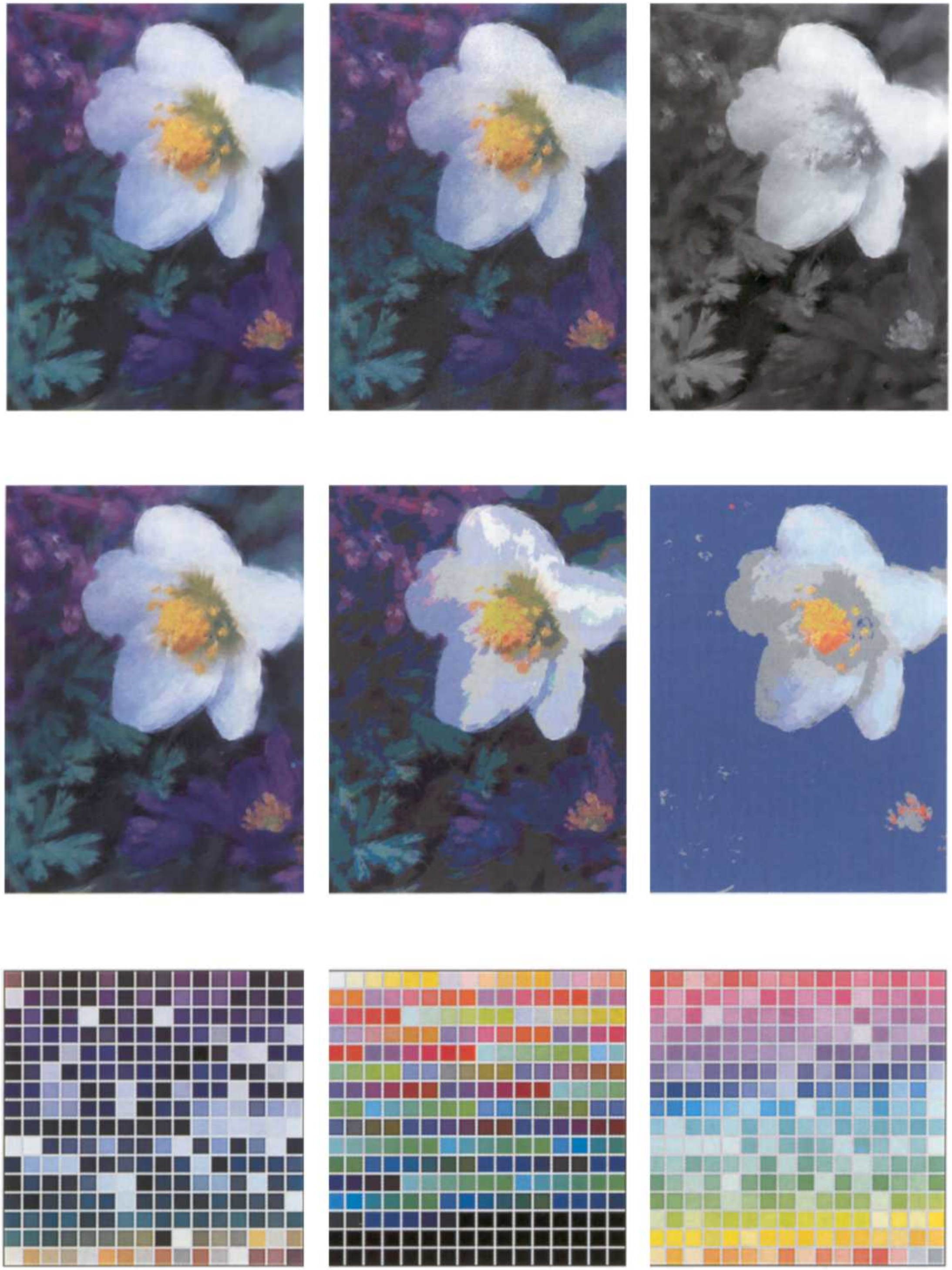
A little experimentation will show that computation of movement using inverse kinematics is not entirely straightforward. In particular, the kinematic constraints on arms and legs do not uniquely determine the possible movements that the limbs can make to accommodate movements of the extremities. Try lifting your right hand to touch the tip of your nose with your first finger. How many different ways can your right elbow move while you do so? In order to fix a particular type of movement, extra constraints, such as minimizing the potential energy of the whole structure, must be added. To produce movements that defy these constraints, while still being physically possible, inverse kinematics must be abandoned, and the parts must be positioned by hand.

Despite what we said earlier, there are some types of 3-D animation that can be done effectively using simple interpolation of objects' properties. As you might expect from our remarks in Chapter 4, specialized applications, such as Bryce, are generally the most succesful. Bryce does allow basic animation of the motion of objects, by interpolating between key frames — the interface is very similar in spirit to that used in After Effects for animating layers, with Bézier motion paths and velocity graphs that can be modified. The most effective application is animating a camera to obtain a 'fly-through' of a scene; the camera can optionally be made to point at an object while it follows its motion path. This inherently artificial type of motion is much more effective when interpolated than more natural movements can be.

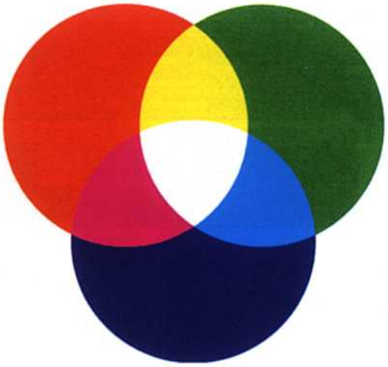
Bryce also includes some specialized animation techniques that are suited to its particular domain of terrain modelling. Most aspects of the sky can be animated, to produce weather effects and to model the passage of time. Clouds can be made to move, as if by the wind, and their shape, density and colour can be changed, to indicate a gathering storm, for example. Rainbows can be made to appear and fade; fog can swirl and dissolve. The sun and moon can rise and set; as the sun passes overhead, shadows will automatically move correctly. Even geological processes can be modelled: mountains can be made to erode into hills, and a landscape can be shaken by an earthquake. All of these effects can be achieved with relative simplicity by animating properties of the scene using key frames and interpolation.



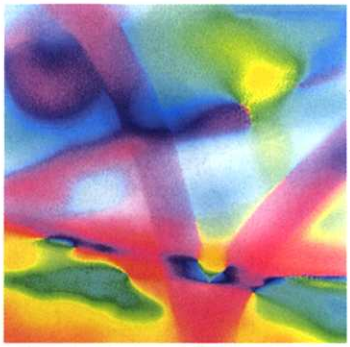
14. Filters applied to a bitmapped image



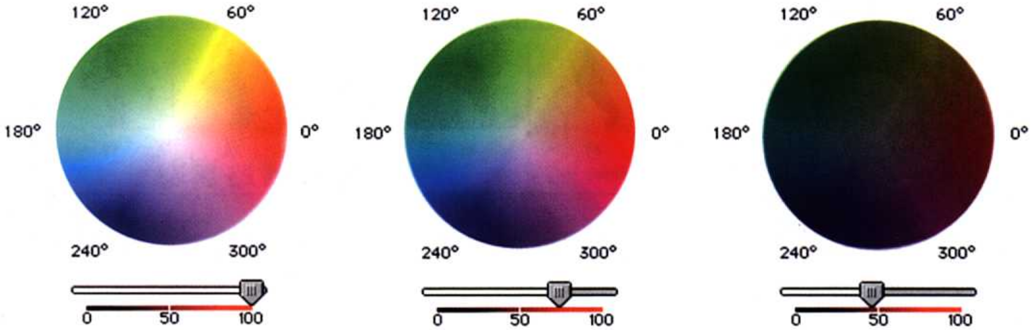
15. Indexed colour



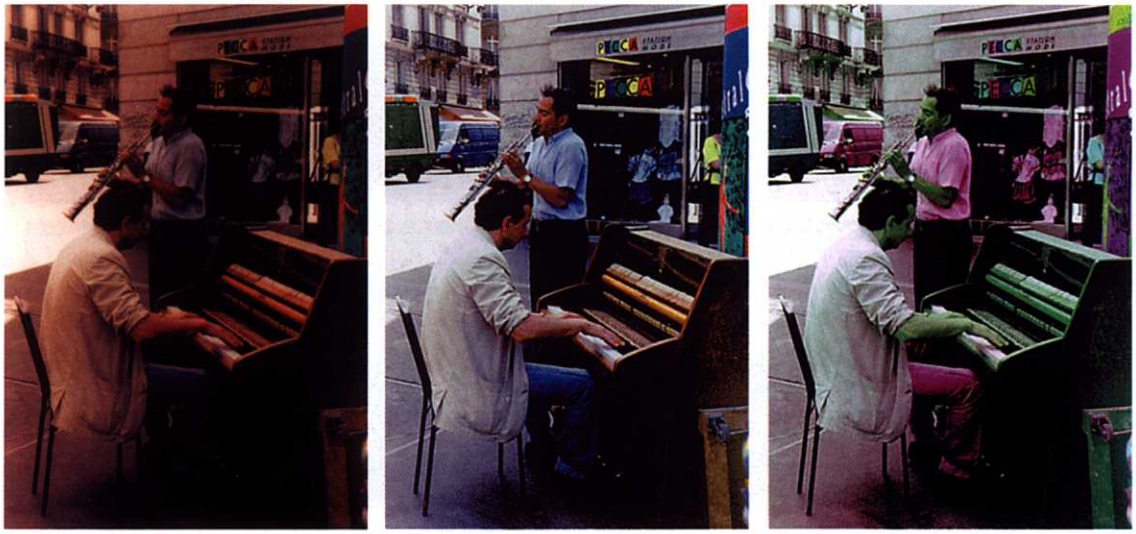
16. Complementary colours



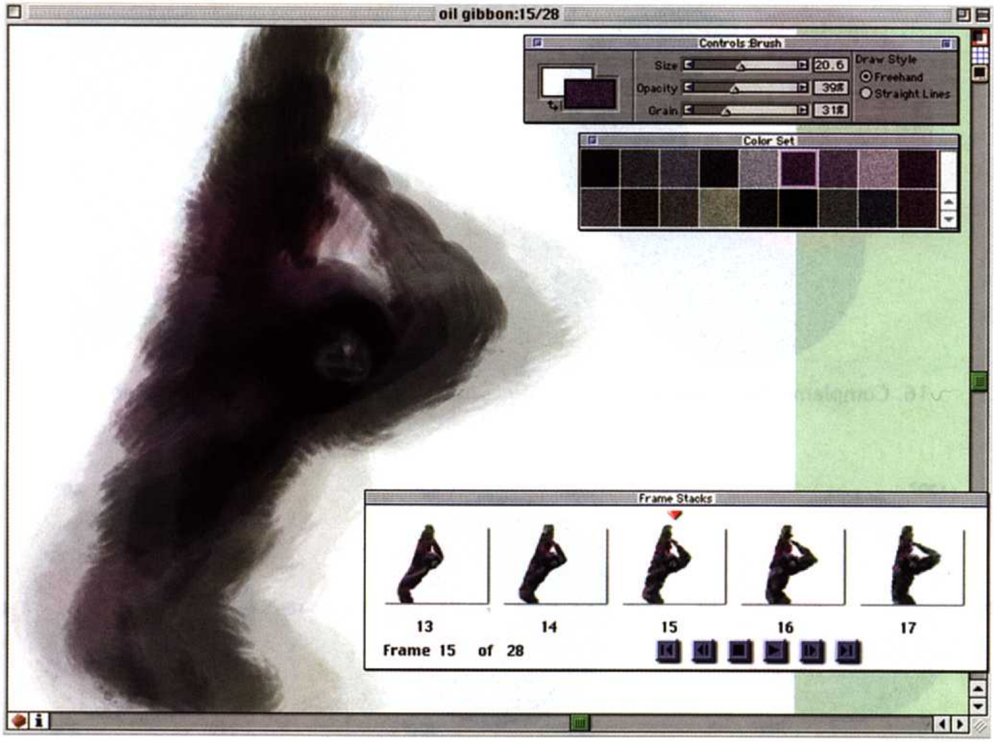
17. Subtractive colour mixing



18. Hue and saturation



19. Colour correction



oil gibbon:15/28

Controls Brush

Size 20.6

Opacity 39%

Grain 31%

Draw Style

● Freehand

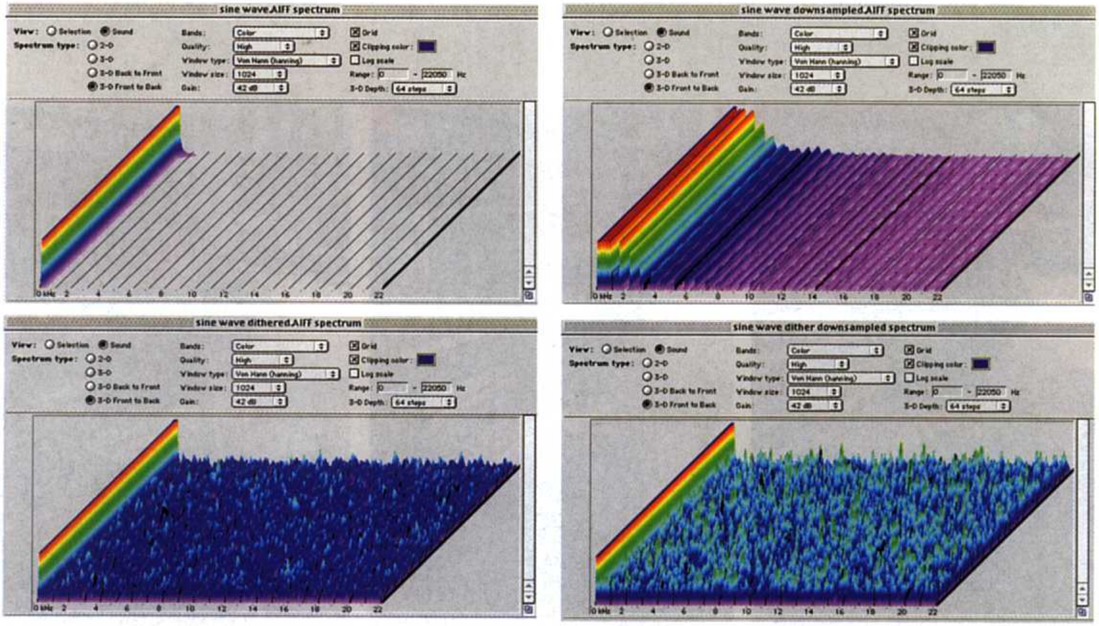
○ Straight Lines

Color Set

Frame Stacks

Frame 15 of 28

20. Animating using a framestack



sine wave.AIFF spectrum

View: ○ Selection ● Sound

Spectrum type: ○ 2-D ○ 3-D ○ 3-D Back to Front ● 3-D Front to Back

Bands: Color

Quality: High

Window type: Von Hann (hanning)

Window size: 1024

Gain: 42 dB

☒ Grid

☒ Clipping color:

□ Log scale

Range: 0 - 22050 Hz

3-D Depth: 64 steps

sine wave downsampled.AIFF spectrum

sine wave dithered.AIFF spectrum

sine wave dither downsampled spectrum

21. Effect of sampling and dithering on audio frequency spectrum

10d

Virtual Reality

Originally, the phrase 'virtual reality' was used to describe an immersive sensory experience of a synthetic world. Head-mounted displays, which are sensitive to head movements, are used to project images on the user's eyes, modifying them as the head is moved, so that the user appears to be inside a 3-D world, looking around. Data gloves track hand movements, allowing the display to incorporate an image of the user's arm; haptic interfaces provide tactile feedback, so that users can touch and feel objects in the virtual world. Taken to the extreme, virtual reality of this sort would be the ultimate in multimedia, stimulating all the senses at once.

The high cost of the interface hardware required by immersive virtual reality (together with the understandable reluctance on the part of most of the adult population to immerse their body in strange electronic devices) has confined it to flight and industrial simulations, and specialist games arcades. A more modest vision of virtual reality (VR), as 3-D graphics that can be explored, has evolved. Even this version of VR has not yet achieved widespread acceptance, largely because the heavy demands it makes on processor power lead to disappointing results on desktop computers. Two VR technologies deserve a brief mention, since they can be incorporated in Web pages and other multimedia productions with some success, and promise to become more important as computer power catches up with the vision of VR enthusiasts.

VRML

The *Virtual Reality Modeling Language (VRML)* was created on a wave of enthusiasm for VR and the World Wide Web in 1994. The intention was to provide a mechanism for distributing virtual worlds over the Internet, using Web browsers as the interface. To this end, VRML was a text-based language, that allowed 3-D objects and scenes to be described in a programming language-like notation. VRML 1.0 did little more; the main additional feature was the capability of embedding hyperlinks in scenes, using URLs. Subsequently VRML 2.0 added support for interactivity, via scripting of the sort to be described in Chapter 14, and allowed for video and audio to be embedded in VRML worlds (so, for example, a television set could be made to show a movie). VRML became an ISO standard[8] in December 1997.

VRML allows the specification of objects, in terms of their geometry (whether they are a cube, cylinder, sphere, and so on) and the material of which they are composed. Textures can be mapped onto the surfaces of objects, which can be placed in 3-D space using transformations. Scenes can be lit in a variety of ways, by specifying the type and position of light objects. The basic language thus provides a way of describing the sorts of scenes that can be constructed with conventional 3-D modelling programs, although it lacks some of the more elaborate features, such as NURBS and metaballs. The description is explicit: for example, a terrain might be modelled as an elevation grid, a type of VRML object that specifies a set of points, forming a grid, each at a different height. In VRML, the dimensions of the grid and the height of each grid point must be explicitly specified. Constructing VRML scenes by hand is thus a painstaking and error-prone business. Most 3-D modellers will generate VRML as output from the normal interactive modelling tools, however, which provides an easier way of constructing scenes.

It might appear that VRML is no more than an alternative representation of the output of a modeller, and, as far as the language goes, this is more or less the case. It does, as we noted earlier, provide additional features for interactivity and embedding multimedia, but the main distinctive feature lies not so much in the language as in the way it is displayed. Once a VRML file has been downloaded into a suitable browser — either a Web browser with an appropriate plug-in, or a dedicated VRML browser — the user can explore the 3-D world it describes. That is, they can move the viewpoint through the space of the model, as if they were moving about in it.[9] To that extent, VRML deserves to be considered a form of virtual reality.

To create the illusion of moving through a 3-D space, VRML must be rendered in real-time. As we have stated several times, realistic rendering of 3-D models is a computationally intensive task, which is usually only feasible with special hardware, such as 3-D accelerator cards. This is one of the main obstacles to the widespread use of VRML, although a lack of commitment to the format by major software vendors may be more significant. At the time of writing, a host of competing — mostly proprietary — formats for delivering 3-D models over the World Wide Web is available, with none, as yet, achieving any widespread use.

[8] ISO/IEC-14772-1:1997. The language defined by the standard is identical to VRML 2.0, but is sometimes referred to as VRML 97.

[9] The experience will, of course, be familiar to players of certain types of computer game.

QuickTime VR

QuickTime VR (or QTVR, for short), part of QuickTime, offers a very basic VR experience. There are two types of QuickTime VR movies: panoramic movies and object movies. The former presents a 360° view of a scene — the interior of a room, or a valley surrounded by mountains, for example. Using their mouse, a user can drag the scene, as if looking around. It is also possible to zoom in or out, in order to view the scene in more or less detail. Object movies, in contrast, allow the user to examine an object from different angles, as if by walking round it, again by dragging with the mouse. QTVR movies of either sort may contain *hot spots*, which are active areas that contain links to other movies. Clicking on a hot spot causes the linked movie to replace the current one. A typical use of hot spots is to allow a user to go through a door from one room into another.

QTVR movies can be generated from some 3-D programs, such as Bryce. They can also be made from photographs, allowing them to represent real scenes and objects. To achieve the best results, a special rotating rig is used to hold the camera (for panoramas) or an object. A sequence of pictures is taken, with the rig being rotated a fixed amount between each picture. These are then scanned (or read in to the computer if a digital camera is being used) and special software 'stitches' them together and renders the result as QTVR. The purpose of the rig is to ensure that the individual pictures fit together perfectly. If an ordinary tripod is used for panoramas, or a record turntable or similar device for object movies, there may be discontinuities; stitching software will attempt to compensate for these, but the result may be distortion of the images.

Since QTVR is part of QuickTime, panoramas and object movies can be combined with audio and video. Most usefully, they can be viewed by any software that uses QuickTime; in particular, the QuickTime plug-in for Web browsers allows QTVR to be embedded in Web pages, in the same way as video clips can be (see Chapter 13).

QuickTime VR and VRML might be considered travesties of the original vision of immersive virtual reality, but they have the advantage of being implementable without special interface devices or powerful workstations. They offer the possibility of new approaches to interfaces to multimedia, based on the organization of media in three-dimensional spaces.

Further Information

[Lay98] is the best starting point for learning more about all aspects of animation. [Vin92] and [O'R98] deal specifically with computer-generated 3-D animation, and [WW92] describes the algorithms it employs. [HW96] is a full description of VRML. [McC96] offers some interesting thoughts about VR and user interfaces.

Exercises

1. What are the advantages and disadvantages of using a scanner or a digital stills camera to capture traditional art work as animation sequences? For what types of animation would you have to use a video camera connected to a computer?

2. How could you incorporate drawn animation into a live-action video sequence without using a special effects program?

3. If an animation sequence is to be saved as QuickTime, what factors will influence your choice of codec? Under what circumstances would it be appropriate to treat the animated sequence exactly like a live-action video sequence?

4. When would it be appropriate to use an animated GIF for an animation sequence? What problems are associated with animated GIFs?

5. In what ways is a sprite animation track radically different from a video track containing animation?

6. For what type of work would sprite animation be particularly suitable and why?

7. In the example of a planet moving across a starscape, how could you add rotation of the planet on its axis to the sprite animation?

8. What problems are associated with using basic linear interpolation to do 'in-betweening' between key frames?

9. How would you use Flash's easing facility to set up a movement that eases in *and* out?

10. Describe the motion of an object whose position is animated in After Effects using Bézier interpolation for the motion path, and linear interpolation for the velocity.

11. Create a very simple title for a video clip as a single image in a bitmapped graphics application such as Photoshop or Painter, and save it as a still image file. Using whatever tools are available (Premiere, After Effects, etc.), create a pleasing ten second title sequence by simply applying time-varying effects and filters to this single image. (If you want to go for a more sophisticated result, and have the necessary tools, you might create your original image on several layers and animate them separately.)

12. Do the models generated by 3-D applications contain enough information to be used in conjunction with a haptic interface to provide tactile feedback to a user? If not, what extra information is needed?

12 Sound

Sound is different in kind from any of the other digital media types we have considered. All other media are primarily visual,[1] being perceived through our eyes, while sound is perceived through the different sense of hearing. Our ears detect vibrations in the air in a completely different way from that in which our eyes detect light, and our brains respond differently to the resulting nerve impulses. Sound does have much in common with one other topic we have considered, though. Although sound is, for most of us, a familiar everyday phenomenon, like colour, it is a complex mixture of physical and psychological factors, which is difficult to model accurately.

Another feature that sound has in common with colour is that you may not always need it. Whereas a multimedia encyclopedia of musical instruments will be vastly enriched by the addition of recordings of each instrument, few, if any, Web pages need to play a fanfare every time they are visited. Sounds can be peculiarly irritating; even one's favourite pieces of music can become a jarring and unwelcome intrusion on the ears when inflicted repeatedly by a neighbour's sound system. Almost everyone has at some time been infuriated by the electronic noises of a portable games console, the cuter varieties of ringing tone of a mobile phone, or the rhythmic hiss that leaks out of the headphones of a personal stereo. The thoughtless use of such devices has become a fact of modern life; a similar thoughtlessness in the use of sound in multimedia

[1] Text may exceptionally be rendered in other ways, but the graphic representation is the norm.

productions should be avoided. At the very least, it should always be possible for users to turn the sound off.

Some users, though, don't need to be able to turn sounds off, because they can't hear them anyway. Not only are some people unable to hear, many others use computers that are not equipped to reproduce sound. Although the Multimedia PC specification[2] requires a sound card (and all Macintosh computers have sound capabilities), older PCs, and those used in the business environment, are rarely fitted with them. It is always considerate to provide some alternative to sound, such as captions or transcripts of speech, for the benefit of those who cannot hear. If you know that your multimedia production is destined to be used in an environment where sound hardware is not typically available, then it may be advisable to avoid the use of sound altogether.

There are two types of sound that are special: music and speech. These are also the most commonly used types of sound in multimedia productions. The cultural status of music, and the linguistic content of speech mean that these two varieties of sound function in a different way from other sounds and noises, and play special rôles in multimedia. Representations specific to music and speech have been developed, to take advantage of their unique characteristics. In particular, compression algorithms tailored to speech are often employed, while music is sometimes represented not as sound, but as instructions for playing virtual instruments.

The Nature of Sound

If a tuning fork is struck sharply on a hard surface, the tines will vibrate at a precise frequency. As they move backwards and forwards, the air is compressed and rarefied in time with the vibrations. Interactions between adjacent air molecules cause this periodic pressure fluctuation to be propagated as a wave. When the sound wave reaches the ear, it causes the eardrum to vibrate at the same frequency. The vibration is then transmitted through the mechanism of the inner ear, and converted into nerve impulses, which we interpret as the sound of the pure tone produced by the tuning fork.

All sounds are produced by the conversion of energy into vibrations in the air or some other elastic medium. Generally, the entire

[2] See page 45.

process may involve several steps, in which the energy may be converted into different forms. For example, if one of the strings of an acoustic guitar is picked with a plectrum, the kinetic energy of the musician's hand is converted to a vibration in the string, which is then transmitted via the bridge of the instrument to the resonant cavity of its body, where it is amplified and enriched by the distinctive resonances of the guitar, and then transmitted through the sound hole. If one of the strings of an electric guitar is picked instead, the vibration of the string as it passes through the magnetic fields of the pickups induces fluctuations in the current which is sent through the guitar lead to an amplifier, where it is amplified and used to drive a loudspeaker. Variations in the signal sent to the speaker coil cause magnetic variations, which are used to drive the speaker cone, which then behaves as a sound source, compressing and rarefying the adjacent air.

While the tines of a good tuning fork vibrate cleanly at a single frequency, most other sound sources vibrate in more complicated ways, giving rise to the rich variety of sounds and noises we are familiar with. As we mentioned in Chapter 2, a single note, such as that produced by a guitar string, is composed of several components, at frequencies that are multiples of the fundamental pitch of the note. Some percussive sounds and most natural sounds do not even have a single identifiable fundamental frequency, but can still be decomposed into a collection — often a very complex one — of frequency components. As in the general case of representing a signal in the frequency domain, which we described in Chapter 2, we refer to a sound's description in terms of the relative amplitudes of its frequency components as its frequency spectrum.

The human ear is generally considered to be able to detect frequencies in the range between 20 Hz and 20 kHz, although individuals' frequency responses vary greatly. In particular, the upper limit decreases fairly rapidly with increasing age: few adults can hear sounds as high as 20 kHz, although children can. Frequencies at the top end of the range generally only occur as components of the transient attack of sounds. (The general rule that high frequencies are associated with abrupt transitions applies here.) The highest note on an ordinary piano — which more or less defines the limit of most Western music — has a fundamental frequency of only 4186 Hz when in concert pitch.[3] However, it is the transient behaviour of notes that contributes most to the distinctive timbre

[3] That is, using even temperament, with the A above middle C equal to 440 Hz.

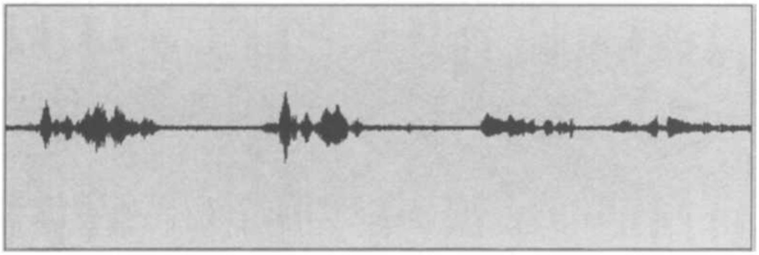


Figure 12.1
'Feisty teenager'

of instruments: if the attack portion is removed from recordings of an oboe, violin, and soprano playing or singing the same note, the steady portions are indistinguishable.

Interesting sounds change over time. As we just observed, a single musical note has a distinctive attack, and subsequently it will decay, changing its frequency spectrum first as it grows, and then as it dies away. Sounds that extend over longer periods of time, such as speech or music, exhibit a constantly changing frequency spectrum. We can display the *waveform* of any sound by plotting its amplitude against time. Examination of waveforms can help us characterize certain types of sound.

➲ The idea of a sound's frequency spectrum changing might be slightly confusing, if you accept that any complex waveform is built out of a collection of frequency components. Strictly, Fourier analysis (as introduced in Chapter 2) can only be applied to *periodic* signals (i.e. ones that repeat indefinitely). When analysing signals with a finite duration, various expedients must be adopted to fit into the analytic framework. One approach is to treat the entirety of a signal as one cycle of a periodic waveform; this is roughly what is done when images are broken down into their frequency components. An alternative is to use a brief section of the signal as if it were a cycle, thus obtaining a snapshot of the frequency make-up at one point. For audio signals, this provides more useful information. A spectrum analysis is typically obtained by sliding a window through the waveform to obtain a sequence of spectra, showing how the signal's frequency components change over time.

Figures 12.1 to 12.7 show waveforms for a range of types of sound. Figure 12.1 is a short example of speech: the main speaker repeats the phrase 'Feisty teenager' twice, then a more distant voice responds. You can clearly identify the syllables, and recognize that the same phrase is repeated, the second time faster and with more emphasis. In between the phrases there is almost silence —

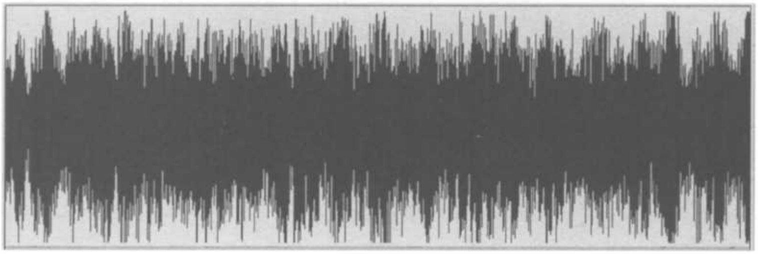


Figure 12.2
Didgeridoo

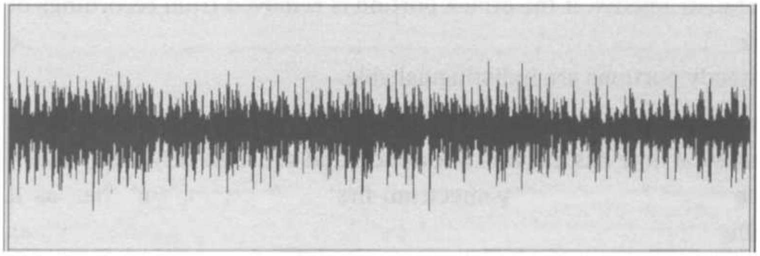


Figure 12.3
Boogie-woogie

the sound was recorded in the open air and there is background noise, which is visible as the thin band running along the axis. You can see that it could be possible to extract individual syllables and recombine them to synthesize new words, and that, if it were necessary to compress speech, a lot could be achieved by removing the silences between phrases. The clearly demarcated syllables also provide a good basis for synchronizing sound with video, as we will see later.

The next four figures show the waveforms of some different types of music. The first three are purely instrumental, and do not exhibit the same character as speech. The first is taken from an Australian aboriginal didgeridoo piece. This is characterized by a continuous drone, which requires the musician to employ a 'circular breathing' technique to maintain it. The waveform shows this drone, as the thick continuous black region, with its rhythmic modulation. Figure 12.3 shows the waveform of a piece of boogie-woogie, played by a pianist accompanied by a small group. The rhythm is clearly visible, but it is not possible to distinguish the melody played in the right hand (unless, perhaps, you are a very experienced audio technician). Figure 12.4 is a completely different waveform, corresponding to a very different piece of music: a contemporary work arranged for violin, cello, and piano. It shows

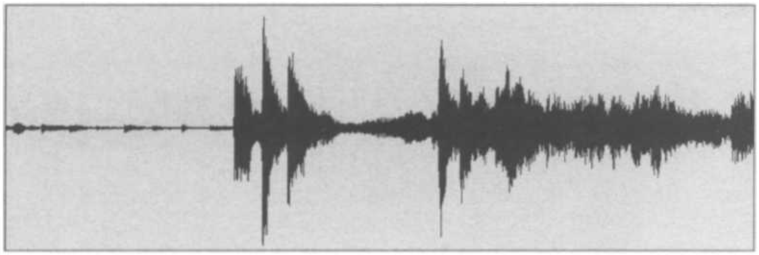


Figure 12.4
Violin, cello and piano

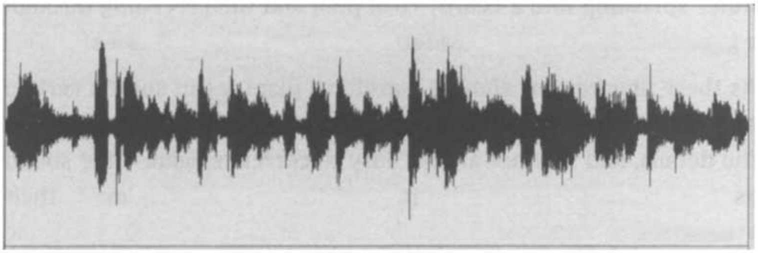


Figure 12.5
'Men grow cold...'

a great dynamic range (difference between the loudest and quietest sounds). Although the steep attack of the louder phrases tells you something about the likely sound of this music, there is no obvious rhythm, and it is not possible to pick out the different instruments (although they can be clearly identified when listening to the music).

As you would expect, singing combines characteristics of speech and music. Figure 12.5 is typical: the syllables of each word are easily identifiable, as is the rhythm, but the gaps between sung phrases are filled with the musical accompaniment. It is possible to see the singer's phrasing, but quite impossible to deduce the lyrics,[4] and, although voice prints are unique to each individual, we doubt whether any readers could identify the singer from this waveform, despite her distinctive voice. (It's Marilyn Monroe.)

Figures 12.6 and 12.7 are both natural water sounds. The first is a recording of the trickling sound of water in a small stream; it is almost continuous. The random spikes do not correspond to any audible clicks or other abrupt sound, they are just slight variations in the water's flow, and some background noise. The second waveform was recorded on the seashore. There is a constant background of surf and two distinct events. The first is a wave breaking fairly close to the microphone, while the second is the

[4] *Men grow cold, as girls grow old/And we all lose our charms in the end.*

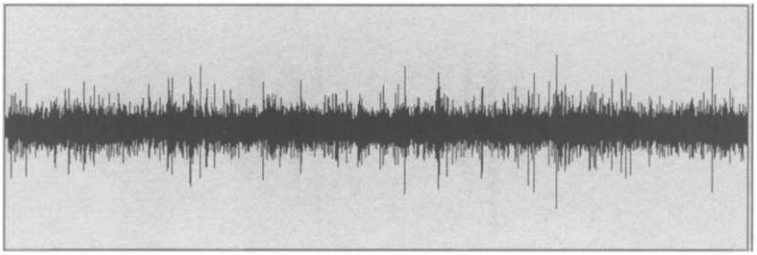


Figure 12.6
A trickling stream

water splashing into a nearby rock pool and then receding through a gap in the rocks. This waveform can almost be read as a story.

As these illustrations show, a waveform display can show a certain amount of the gross character of a sound, but it does not convey the details, and it is not always easy to correlate against the sound as it is heard. The main advantage of these visual displays is their static nature. A piece of sound can be seen in its entirety at one time, with relationships such as the intervals between syllables or musical beats visible. This makes analysis of the sound's temporal structure — which is especially useful for synchronization purposes — relatively simple, compared to performing the same analysis on the dynamically changing sound itself, which is only heard an instant at a time.

Waveforms and the physics of sound are only part of the story. Sound only truly exists as a sensation in the mind, and the perception of sound is not a simple registering of the physical characteristics of the waves reaching the ears. Proofs of this abound, both in the literature and in everyday experience. For example, if a pure 200 Hz tone is played, first softly, then louder, most listeners will believe that the louder tone has a lower pitch than the quieter one, although the same illusion is not perceived with higher frequency tones. Similarly, complex tones sometimes seem to have a lower pitch than pure tones of the same frequency. Most people with good hearing can distinguish the sound of their own name spoken on the opposite side of a noisy room, even if the rest of what is said is inaudible, or carry on a successful conversation with someone speaking at a volume lower than that of the ambient noise.

One of the most useful illusions in sound perception is stereophony. The brain identifies the source of a sound on the basis of the differences in intensity and phase between the signals received from

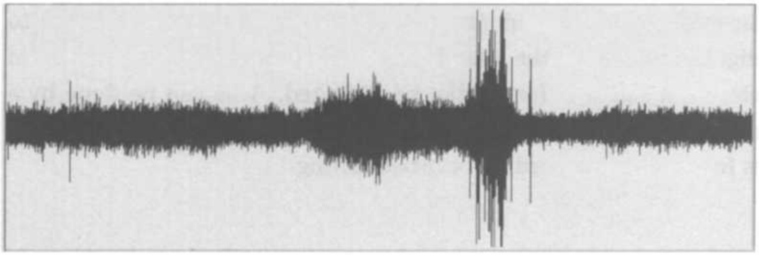


Figure 12.7
The sea

the left and right ears. If identical signals are sent to both ears, the brain interprets the sound as coming from a non-existent source that lies straight ahead. By extension, if a sound is recorded using a pair of microphones to produce two monophonic channels, which are then fed to two speakers that are a suitable distance apart, the apparent location of the sound will depend on the relative intensity of the two channels: if they are equal it will appear in the middle, if the left channel is louder (because the original sound source was nearer to the left hand microphone) it will appear to the left, and so on. In this way, the familiar illusion of a sound stage between the speakers is constructed.

Because of the psychological dimension of sound, it is unwise, when considering its digitization and reproduction, to place too much reliance on mathematics and measurable quantities. Pohlmann's comments[5] about the nature of sound and its reproduction should be borne in mind:

> "Given the evident complexity of acoustical signals, it would be naive to believe that analog or digital technologies are sufficiently advanced to capture fully and convey the complete listening experience. To complicate matters, the precise limits of human perception are not known. One thing is certain: at best, even with the most sophisticated technology, what we hear being reproduced through an audio system is an approximation of the actual sound."

Digitizing Sound

The digitization of sound is a fairly straightforward example of the processes of quantization and sampling described in Chapter 2.

[5] [Poh95], p 5.

Since these operations are carried out in electronic analogue to digital converters, the sound information must be converted to an electrical signal before it can be digitized. This can be done by a microphone or other transducer, such as a guitar pickup, just as it is for analogue recording or broadcasting.

Sampling

A sampling rate must be chosen that will preserve at least the full range of audible frequencies, if high-fidelity reproduction is desired. If the limit of hearing is taken to be 20 kHz[6], a minimum rate of 40 kHz is required by the Sampling Theorem. The sampling rate used for audio CDs is 44.1 kHz — the precise figure being chosen by manufacturers to produce a desired playing time[7] given the size of the medium. Because of the ubiquity of the audio CD, the same rate is commonly used by the sound cards fitted to computers, to provide compatibility. Where a lower sound quality is acceptable, or is demanded by limited bandwidth, sub-multiples of 44.1 kHz are used: 22.05 kHz is commonly used for audio destined for delivery over the Internet, while 11.025 kHz is sometimes used for speech. Another important sampling rate is that used by DAT (digital audio tape) recorders, and also supported by the better sound cards. Although these commonly offer a variety of sampling rates, 48 kHz is used when the best quality is desired. DAT is a very suitable medium for live recording and low budget studio work, and is often used for capturing sound for multimedia.

DAT and CD players both have the advantage that they can generate digital output, which can be read in by a suitably equipped computer without the need for extra digitizing hardware. In this respect, they resemble DV cameras. Where a digital signal cannot be produced, or where the computer is not fitted with the appropriate digital audio input, a digitizing sound card must be fitted to the computer, in the same way as a video capture board must be used for analogue video. Digital audio inputs are surprisingly uncommon, so it is often necessary for the (analogue) line output of a DAT or CD player to be redigitized by the sound card. This is clearly unfortunate, since it is preferable to work entirely with digital data and prevent noise and signal degradataion. It does, however, avoid the problem of incompatible sampling rates that can occur if, say, a recording on DAT is to be combined with an extract from a CD. Resampling audio is as undesirable as resampling images.

[6] Which makes some assumptions about the auditory system's transient response, which may or may not be valid.

[7] According to legend, the time to play Karajan's recording of Beethoven's 9th symphony.

⇨ The necessity to resample data sampled at 48 kHz often occurs if the sound is to be combined with video. Some video applications do not yet support the higher sampling rate, even though DAT is widely used for capturing sound, and sound cards that support 48 kHz are becoming common. For multimedia work it may therefore be preferable to sample sound at 44.1 kHz, which is supported by all the major desktop video editing programs.

Sampling relies on highly accurate clock pulses to determine the intervals between samples. If the clock drifts, so will the intervals. Such timing variations are called *jitter*. The effect of jitter is to introduce noise into the reconstructed signal. At the high sampling frequencies required by sound, there is little tolerance for jitter: it has been estimated that for CD quality sound, the jitter in the ADC must be less than 200 picoseconds (200×10^{-12} seconds).

Even if they are inaudible, frequencies in excess of 20 kHz are present in the spectra of many sounds. If a sampling rate of around 40 kHz is used, these inaudible components will manifest themselves as aliasing when the signal is reconstructed. In order to avoid this, a filter is used to remove any frequencies higher than half the sampling rate before the signal is sampled.

Quantization

We mentioned in Chapter 2 that the number of quantization levels for analogue to digital conversion in any medium is usually chosen to fit into a convenient number of bits. For sound, the most common choice of sample size is 16 bits, as used for CD audio, giving 65 536 quantization levels. This is generally sufficient to eliminate quantization noise, if the signal is dithered, as we will describe shortly. As with images, smaller samples sizes (lower bit-depths, as we would say in the context of images) are sometimes needed to maintain small file sizes and bit rates. The minimum acceptable is 8-bit sound, and even this has audible quantization noise, so it can only be used for applications such as voice communication, where the distortion can be tolerated. In the search for higher fidelity reproduction, as many as 24 bits are sometimes used to record audio samples, but this imposes considerable demands on the accuracy of ADC circuitry.

Quantization noise will be worst for signals of small amplitude. In the extreme, when the amplitude is comparable to the difference

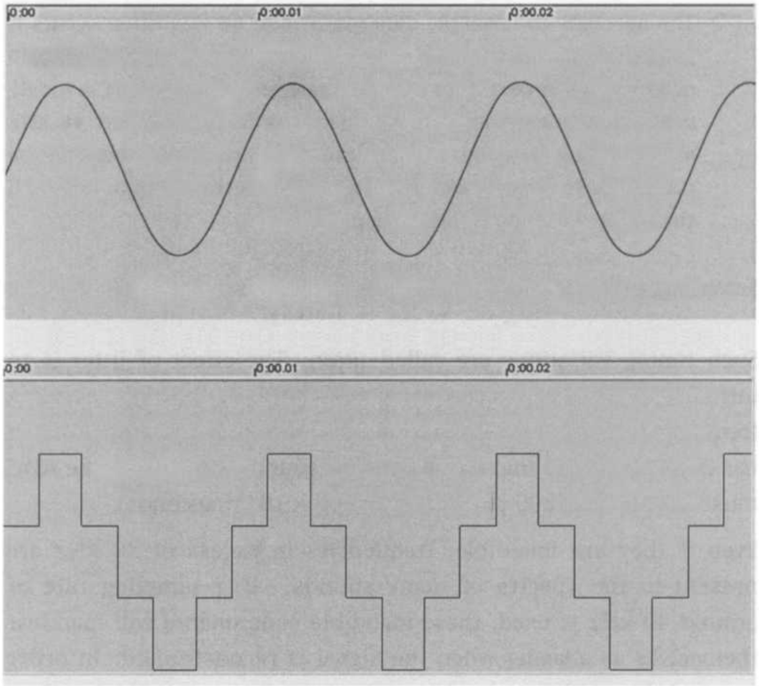


Figure 12.8
Undersampling a pure sine wave

between quantization levels, an analogue signal will be coarsely approximated by samples that jump between just a few quantized values. This is shown in Figure 12.8. The upper waveform is a pure sine wave; below it is a digitized version, where only four levels are available to accommodate the amplitude range of the original signal.[8] Evidently, the sampled waveform is a poor approximation of the original. The approximation could be improved by increasing the number of bits for each sample, but a more economical technique, resembling the anti-aliasing applied when rendering vector graphics, is usually employed. Its operation is somewhat counter-intuitive.

Before sampling, a small amount of random noise is added to the analogue signal. The word 'dithering' (which we used with a somewhat different meaning in Chapter 6) is used in the audio field to refer to this injection of noise. The effect on sampling is illustrated in Figure 12.9. The upper waveform is the original sine wave with added dither;[9] The lower waveform is a sampled version of this dithered signal. What has happened is that the presence of the noise has caused the samples to alternate rapidly between quantization levels, instead of jumping cleanly and abruptly from

[8] If you want to be scrupulous, since these images were prepared using a digital audio application, the top waveform is a 16-bit sampled sine wave (a very good approximation), the lower is the same waveform downsampled to 2 bits.

[9] We have used rather more noise than is normal, in order to show the effect more clearly.

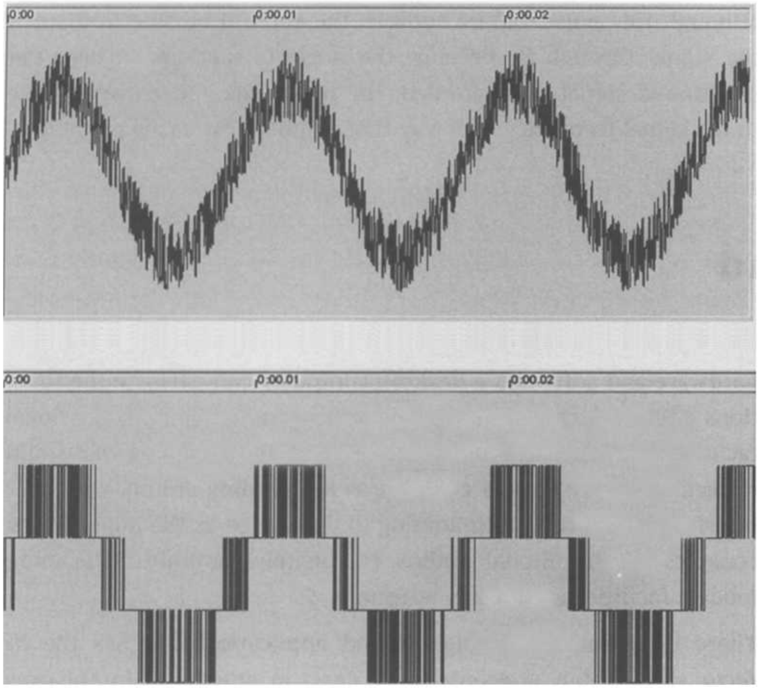


Figure 12.9
Dithering

one to the next, as they do in Figure 12.8. The sharp transitions have been softened. Putting it another way, the quantization error has been randomized. The price to be paid for the resulting improvement in sound quality is the additional random noise that has been introduced. This is, however, less intrusive than the quantization noise it has eliminated.

Plate 21 illustrates the effect of sampling and dithering on the signal's frequency spectrum. In these pictures, the horizontal x-axis represents frequency, the vertical y-axis amplitude, with the colours being used as an extra visual indication of intensity, and the back-to-front z-axis represents time. The first spectrum is the pure sine wave; as you would expect, it is a spike at the wave's frequency, which is constant over time. To its right is the spectrum of the sampled signal: spurious frequencies and noise have been introduced. These correspond to the frequency components of the sharp edges. Below the pure sine wave is the spectrum of the dithered version. The extra noise is randomly distributed across frequencies and over time. In the bottom left is the sampled version of this signal. The pure frequency has re-emerged clearly, but random noise is present where before there was none. However, although this noise will be audible, the ear will be able to discern the signal through it, because the noise is random. Where the undithered signal was sampled, the noise was concentrated near to the signal frequency, in a way that is much less easily ignored.

Processing Sound

With the addition of suitable audio input, output and processing hardware and software, a desktop computer can perform the functions of a modern multi-track recording studio. Such professional facilities are expensive and demanding on resources, as you would expect. They are also as complex as a recording studio, with user interfaces that are as intimidating to the novice as the huge mixing consoles of conventional studios. Fortunately, for multimedia, more modest facilities are usually adequate.

There is presently no single sound application that has the *de facto* status of a cross-platform desktop standard, in the way that Photoshop and Premiere, for example, do in their respective fields. Several different packages, some of which require special hardware support, are in use. Most of the well known ones are biased towards music, with integrated support for MIDI sequencing (see page 405) and multi-track recording. Several more modest programs provide simple recording and effects processing facilities; where hardware support is not provided, real-time effects are not usually achievable. Video editing packages usually include some integrated sound editing and processing facilities, and some offer basic sound recording. These facilities may be adequate for multimedia production in the absence of special sound software, and are especially convenient when the audio is intended as a soundtrack to accompany picture.

Given the absence of an industry standard sound application for desktop use, we will describe the facilities offered by sound programs in general terms only, without using any specific example.

Recording and Importing Sound

Many desktop computers are fitted with built-in microphones, and it is tempting to think that these are adequate for recording sounds.

It is almost impossible to obtain satisfactory results with these, however — not only because the microphones themselves are usually small and cheap, but because they are inevitably close to the machine's fan and disk drives, which means that they will pick up noises from these components. It is much better to plug an external microphone into a sound card, but if possible, you should do the actual recording onto DAT (or good quality analogue tape) using a professional microphone, and capture it in a separate operation. Where sound quality is important, or for recording music to a high standard, it will be necessary to use a properly equipped studio. Although a computer can form the basis of a studio, it must be augmented with microphones and other equipment in a suitable acoustic environment, so it is not really practical for a multimedia producer to set up a studio for one-off recordings. It may be necessary to hire a professional studio, which offers the advantage that professional personnel will generally be available.

Before recording, it is necessary to select a sampling rate and sample size. Where the sound originates in analogue form, the choice will be determined by considerations of file size and bandwidth, which will depend on the final use to which the sound is to be put, and the facilities available for sound processing. As a general rule, the highest possible sampling rate and sample size should be used,[10] to minimize deterioration of the signal when it is processed. If a compromise must be made, the effect on quality of reducing the sample size is more drastic than that of reducing the sampling rate: the same reduction in size can be produced by halving the sampling rate or halving the sample size; the former is better. If the signal is originally a digital one — the digital output from a DAT recorder, for example — the sample size should be matched to the incoming rate, if possible.

A simple calculation suffices to show the size of digitized audio. The sampling rate *is* the number of samples generated each second, so if the rate is r Hz and the sample size is s bits, each second of digitized sound will occupy $rs/8$ bytes. Hence, for CD-quality, $r = 44.1 \times 10^3$ and $s = 16$, so each second occupies just over 86 kbytes,[11] each minute roughly 5 Mbytes. These calculations are based on a single channel, but audio is almost always recorded in stereo, so the estimates should be doubled. Conversely, where stereo effects are not required, the space occupied can be halved by recording in mono.

[10] But remember that if the sound is to be combined with video in an editing application that cannot handle 48 kHz, then it is better to record at 44.1 kHz in the first place.

[11] In kHz, k represents 1000, following normal usage, but in kbytes, the k is 1024, in accordance with the conventions of computing.

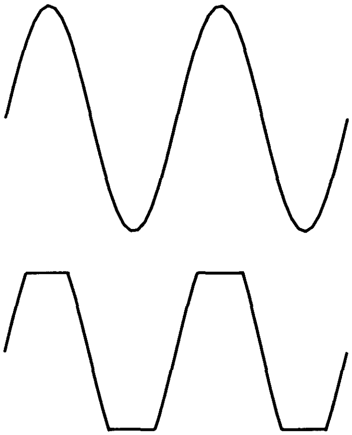


Figure 12.10
Clipping

Professional sound applications will record directly to disk, so that the possible length of recordings is limited only by the available disk space and any file size limitations built in to the operating system. Many lower-level programs record to RAM, however, and subsequently carry out all their processing in memory. While this is more efficient, it imposes severe restrictions on the length of sound that can be managed.

The most vexatious aspect of recording is getting the levels right. If the level of the incoming signal is too low, the resulting recording will be quiet, and more susceptible to noise; if the level is too high, clipping will occur, that is, at some points, the amplitude of the incoming signal will exceed the maximum value that can be recorded. The value of the corresponding sample will be set to the maximum, so the recorded waveform will apparently be clipped off straight at this threshold. (Figure 12.10 shows the effect on a pure sine wave.) The result is heard as a particularly unpleasant sort of distortion. Ideally, a signal should be recorded at the highest possible level that avoids clipping. Sound applications usually provide level meters, so that the level can be monitored, with clipping alerts. Where the sound card supports it, a gain control can be used to alter the level. Some sound cards do not provide this function, so that the only option is to adjust the output level of the equipment from which the signal originates. Setting the level correctly is easier said than done, especially where live recordings are being made: to preserve the dynamic range of the recording, the same gain must be used throughout, but the optimum can only be determined at the loudest point. When the sound is live, this cannot be known in advance, and only experience can be used to choose gain settings. Where the material already exists on tape or CD, it is possible — and usually necessary — to make several passes in order to find the best values.

Some software includes automatic gain controls, which vary the gain dynamically according to the amplitude of the signal, in order to prevent clipping. They must, therefore, reduce the volume of louder passages, so as a side-effect, they reduce the dynamic range of the recording. This is generally undesirable, but may be necessary if suitable levels cannot be maintained throughout the recording.

⇨ It may be obvious, but it seems worth emphasizing: once a signal has been clipped, nothing can be done to restore it. Reducing the amplitude subsequently just produces a smaller clipped signal. There is no way to recover the lost waveform. Similarly, although sound programs often provide a facility for 'normalizing' a sound after recording, by amplifying it as much as possible without causing clipping, this stretches the dynamic range of the original without adding any more detail. In practice it may be necessary to use this facility, or to select and amplify particularly quiet passages within a sound editing application after the recording has been made. In principle, though, the gain should always be set correctly, both when recording to tape, and when recording or capturing to disk.

A technically simpler alternative to recording sound is to import it from an audio CD. Although audio CDs use a different format from CD-ROM, they nevertheless are a structured collection of digital data, so they can be read by suitable software. QuickTime includes an audio CD import component that allows any sound application based on QuickTime to open tracks on a CD just like any other file. This is the simplest way of importing sounds, but most recorded music is copyrighted, so it is necessary to obtain permissions first. Copyright-free collections can be obtained, much like royalty-free image libraries, although they tend to the anodyne. Composers and musicians with access to professional recording facilities may supply their work on CD, avoiding the need for the multimedia producer to deal with the sound recording process. However, even when importing sounds from CDs there can be difficulty in getting the levels right.

The Internet is a rich source of ready-made sounds, but many are made available illegally, and others may not be legally reproduced without payment of a fee. Increasingly, though, record companies are arriving at mechanisms to provide music online, usually in the form of MP3 files (see below). While it may be legal to download these files and listen to them, it remains generally illegal to use them in any published form without obtaining clearance from the copyright holders.

Sound Editing and Effects

We can identify several classes of operation that we might want to apply to recorded sounds. Most of them have counterparts in video editing, and are performed for similar reasons.

First, there is editing, in the sense of trimming, combining and rearranging clips. The essentially time-based nature of sound naturally lends itself to an editing interface based on a timeline. A typical sound editing window is divided into tracks, in imitation of the separate tape tracks used on traditional recording equipment, providing a clear graphic representation of the sound through time. The sound in each track may usually be displayed as a waveform; the time and amplitude axes can be scaled, allowing the sound to be examined in varying degrees of detail. Editing is done by cutting and pasting, or dragging and dropping, selected parts of the track. Each stereo recording will occupy two tracks, one for each channel. During the editing process many tracks may be used to combine sounds from separate recordings. Subsequently, these will be mixed down onto one or two tracks, for the final mono or stereo output. When mixing, the relative levels of each of the tracks can be adjusted to produce the desired balance — between different instruments, for example.

A special type of edit has become common in audio: the creation of loops. Very short loops are needed to create voices for the electronic musical instruments known as samplers (whose functions are increasingly performed by software). Here, the idea is to create a section of sound that represents the sustained tone of an instrument, such as a guitar, so that arbitrarily long notes can be produced by interpolating copies of the section between a sample of the instrument's attack and one of its decay. It is vital that the sustained sample loops cleanly; there must not be abrupt discontinuities between its end and start, otherwise audible clicks will occur where the copies fit together. Although some software makes such loops automatically, using built-in heuristics such as choosing zero crossings for each end of the loop, the best results require a detailed examination of the waveform by a person. Longer loops are used in certain styles of dance music — techno and drum'n'bass, for example — which are based on the combination of repeating sections. Again, there is a requirement for clean looping, but this time, at the coarser level of rhythmic continuity.

As well as editing, audio has its equivalent of post-production: altering sounds to correct defects, enhance quality, or otherwise modify their character. Just as image correction is described in terms of filters, which are a digital equivalent of traditional optical devices, so sound alteration is described in terms of gates and filters, by analogy with the established technology. Whereas analogue gates and filters are based on circuitry whose response produces a desired effect, digital processing is performed by algorithmic manipulation of the samples making up the signal. The range of effects, and the degree of control over them, that can be achieved in this way is much greater than is possible with analogue circuits. Several standard plug-in formats are in use, that allow effects to be shared among programs. Although it is not an audio application, Premiere's effects plug-in format is becoming widely used; at a more professional level, the formats associated with Cubase VST and with DigiDesign ProTools are popular.

The most frequently required correction is the removal of unwanted noise. For example, in Figure 12.1, it might be considered desirable to remove the background noises that were unavoidably picked up by the microphone, since the recording was made in the open. A *noise gate* is a blunt instrument that is used for this purpose. It eliminates all samples whose value falls below a specified threshold; samples above the threshold are left alone. As well as specifying the threshold, it is usual to specify a minimum time that must elapse before a sequence of low amplitude samples counts as a silence, and a similar limit before a sequence whose values exceed the threshold counts as sound. This prevents the gate being turned on or off by transient glitches. By setting the threshold just above the maximum value of the background noise, the gaps between words will become entirely silent. Since the noise gate has no effect on the speaker's words, the accompanying background noise will cut in and out as he speaks, which may well turn out to be more distracting than the original noise. This illustrates a general problem with noise removal: the noise is intimately combined with the signal, and although people can discriminate between the two, computer programs generally cannot.

Noise gates can be effective at removing hiss from music, since, in this case, the noise is hidden except in silent passages, where it will be removed by the noise gate. There are more sophisticated ways of reducing noise than the all-or-nothing filtering of the noise gate, though. Filters that remove certain bands of frequencies can be applied to noise that falls within a specific frequency range. *Low pass* filters, which allow low frequencies to pass through them, removing high frequencies, can be used to take out hiss; *high pass filters*, which pass the high frequencies and block the low, are used to remove 'rumble': low frequency noise caused by mechanical vibrations. A *notch filter* removes a single narrow frequency band. The commonest use of notch filters is to remove hum picked up from the mains: this will have a frequency of exactly 50 or 60 Hz, depending on the part of the world in which the noise was recorded. Some sophisticated programs offer the user the ultimate facility of being able to redraw the waveform, rubbing out the spikes that correspond to clicks, and so on. To do this effectively, however, requires considerable experience and the ability to interpret the visual display of a waveform in acoustic terms, which, as the examples shown earlier demonstrate, is not always easy.

Specialized filters are available for dealing with certain common recording defects. A *de-esser* is a filter that is intended to remove the sibilance that results from speaking or singing into a microphone placed too close to the performer. *Click repairers* are intended to remove clicks from recordings taken from damaged or dirty vinyl records.[12] Although these filters are more discriminating than a noise gate, they are not infallible. The only sure way to get perfect sound is to start with a perfect take — microphones should be positioned to avoid sibilance, and kept well away from fans and disk drives, cables should be screened to avoid picking up hum, and so on.

⇨ Although the noise reduction facilities available in desktop sound applications are fairly crude and ineffectual, more elaborate — and computationally expensive — approaches have been developed. One approach is based on attempting to analyse the acoustic properties of the original recording apparatus on the basis of the make-up of the noise in quiet passages, and then compensating for it in the music. Sophisticated noise reduction techniques are used to restore old records from the early part of the 20th century, and also to reconstruct other damaged recordings, such as the tapes from voice recorders of crashed aircraft.

When we consider effects that alter the quality of a sound, there is a continuum from those that perform minor embellishments to compensate for poor performance and recording, to those that radically alter the sound, or create new sounds out of the original. A single effect may be used in different ways, at different points in this continuum, depending on the values of parameters that affect its operation. For example, a *reverb* effect is produced digitally by adding copies of a signal, delayed in time and attenuated, to the original. These copies model reflections from surrounding surfaces, with the delay corresponding to the size of the enclosing space and the degree of attenuation modelling surfaces with different acoustic reflectivity. By using small delays and low reflectivity, a recording can be made to sound as if it had been made inside a small

[12] There are also effects plug-ins that attempt to add authentic-sounding vinyl noise to digital recordings.

room. This degree of reverb is often a necessary enhancement when the output from electric instruments has been recorded directly without going through a speaker and microphone. Although cleaner recordings are produced this way, they are often too dry acoustically to sound convincing. Longer reverb times can produce the illusion of a concert hall or a stadium.[13] Still longer times, with the delayed signals being amplified instead of attenuated, can be used creatively to generate sustained rhythm patterns from a single chord or note.

Other effects can be put to a variety of uses in a similar way. These include *graphic equalization*, which transforms the spectrum of a sound using a bank of filters, each controlled by its own slider, and each affecting a fairly narrow band of frequencies. (Analogue graphic equalizers are commonly found on mid-range domestic sound systems.) These can be used to compensate for recording equippment with idiosyncratic frequency response, or to artificially enhance the bass, for example, to produce a desired frequency balance. *Envelope shaping* operations change the outline of a waveform. The most general envelope shapers allow the user to draw a new envelope around the waveform, altering its attack and decay and introducing arbitrary fluctuations of amplitude. Specialized versions of envelope shaping include *faders*, which allow a sound's volume to be gradually increased or decreased, and *tremolo*, which causes the amplitude to oscillate periodically from zero to its maximum value.[14]

Time stretching and *pitch alteration* are two closely related effects that are especially well-suited to digital sound. With analogue recordings, altering the duration of a sound can only be achieved by altering the speed at which it is played back, and this alters the pitch. With digital sound, the duration can be changed without altering the pitch, by inserting or removing samples. Conversely, the pitch can be altered without affecting the duration.

Time stretching is most often required when sound is being synchronized to video or another sound. If, for example, a voice-over is slightly too long to fit over the scene it describes, the soundtrack can be shrunk in time, without raising the pitch of the speaker's voice, which would happen if the voice track was simply played at a faster speed.

Pitch alteration can be used in several ways. It can be applied uniformly to alter the pitch of an instrument, compensating for an out-of-tune guitar, for example. It can be applied periodically to add a vibrato (periodic fluctuation of pitch) to a voice or instrument, or

[13] One sound application offers the values 'Empty room', 'Concert Hall', 'Stadium', and 'Outer Space' as settings for its reverb effect.

[14] To classical musicians, 'tremolo' means the rapid repetition of a single note — this does produce a periodic oscillation of amplitude. The 'tremolo arm' fitted to Fender Stratocasters and other electric guitars actually produces a periodic change of pitch, more accurately referred to as 'vibrato'.

it can be applied gradually, to produce a 'bent note', in the same way a blues guitarist changes the tone of a note by bending the string while it sounds. The all-important shape of the bend can be specified by drawing a curve showing how the pitch changes over time.

Beyond these effects lie what are euphemistically called 'creative' sound effects. Effects such as flanging, phasing, chorus, ring modulation, reversal, Doppler shift, and wah-wah, which were pioneered in the 1960s on albums such as *Sergeant Pepper's Lonely Hearts Club Band* and *Electric Ladyland* have been reproduced digitally, and joined by new extreme effects such as roboticization. These effects, if used judiciously, can enhance a recording, but they are easily over-used, and are generally best enjoyed in private.

Compression

While the data rate for CD-quality audio is nothing like as demanding as that for video, it still exceeds the bandwidth of dial-up Internet connections, and lengthy recordings rapidly consume disk space. A single three minute song, recorded in stereo, will occupy over 25 Mbytes. Hence, where audio is used in multimedia, and especially when it is delivered over the Internet, there is a need for compression. The complex and unpredictable nature of sound waveforms makes them difficult to compress using lossless methods. Huffman coding can be effective in cases where the amplitude of the sound mainly falls below the maximum level that can be represented in the sample size being used. In that case, the signal could have been represented in a smaller sample size, and the Huffman algorithm, by assigning short codes to the values it does encounter, will effectively do this automatically. This is a special case, though, and, in general, some form of lossy compression will be required.

An obvious compression technique that can be applied to speech is the removal of silence. That is, instead of using 44,100 samples with the value of zero for each second of silence (assuming a 44.1 kHz sampling rate) we record the length of the silence. This technique appears to be a special case of run-length encoding, which, as we said in Chapter 5, is lossless. However, as Figure 12.1 shows, 'silence' is rarely absolute. We would obtain little compression if we simply run-length encoded samples whose value was exactly zero; instead, we must treat samples falling below a threshold as if they were zero. The effect of doing this is equivalent to applying a noise gate, and is not strictly lossless, since the decompressed signal will not be identical to the original.

The principles behind lossy audio compression are different from those used in lossy image compression, because of the differences in the way we perceive the two media. In particular, whereas the high frequencies associated with rapid changes of colour in an image can safely be discarded, the high frequencies associated with rapid changes of sound are highly significant, so some other principle must be used to decide what data can be discarded.

Speech Compression

Telephone companies have been using digital audio since the early 1960s, and have been forced by the limited bandwidth of telephone lines to develop compression techniques that can be effectively applied to speech. An important contribution of this early work is the technique known as *companding*. The idea is to use non-linear quantization levels, with the higher levels spaced further apart than the low ones, so that quiet sounds are represented in greater detail than louder ones. This matches the way in which we perceive differences in volume.

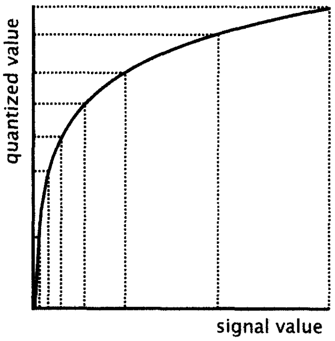


Figure 12.11
Non-linear quantization

Figure 12.11 shows an example of non-linear quantization. The signal value required to produce an increase of one in the quantized value goes up logarithmically. This produces compression, because fewer bits are needed to represent the full range of possible input values than a linear quantization scheme would require. When the signal is reconstructed an inverse process of expansion is required, hence the name 'companding' — itself a compressed version of 'compressing/expanding'.

Different non-linear companding functions can be used. The principle important ones are defined by ITU Recommendations for use in telecommunications. Recommendation G.711 defines a function called the *μ-law*, which is used in North America and Japan. It has been adopted for compressing audio on Sun and NeXT systems, and files compressed in accordance with it are commonly found on the Internet. A different ITU Recommendation is used in the rest of the world, based on a function known as the *A-law*.

Telephone signals are usually sampled at only 8 kHz. At this rate, μ-law compression is able to squeeze a dynamic range of 12 bits into just 8 bits, giving a one-third reduction in data rate.

The μ-law is defined by the equation:

$$y = \frac{\log(1 + \mu x)}{\log(1 + \mu)} \text{ for } x \geq 0$$

where μ is a parameter that determines the amount of companding; $\mu = 255$ is used for telephony.

The A-law is:

$$y = \begin{cases} \dfrac{Ax}{1 + \log A} & \text{for } 0 \leq |x| \leq 1/A \\ \dfrac{1 + \log Ax}{1 + \log A} & \text{for } 1/A \leq |x| \leq 1 \end{cases}$$

Another important technique that was originally developed for, and is widely used in, the telecommunications industry is *Adaptive Differential Pulse Code Modulation (ADPCM).* This is related to inter-frame compression of video, in that it is based on storing the difference between consecutive samples, instead of the absolute value of each sample. Because of the different nature of audio and video, and its origins in hardware encoding of transmitted signals, ADPCM works somewhat less straightforwardly than a simple scheme based on the difference between samples.

Storing differences will only produce compression if the differences can be stored in fewer bits than the sample. Audio waveforms can change rapidly, so, unlike consecutive video frames, there is no reason to assume that the difference will necessarily be much less than the value. Basic *Differential Pulse Code Modulation (DPCM)*[15] therefore computes a predicted value for a sample, based on preceding samples, and stores the difference between the prediction and the actual value. If the prediction is good, the difference will be small. *Adaptive* DPCM obtains further compression by dynamically varying the step size used to represent the quantized differences. Large differences are quantized using large steps, small differences using small steps, so the amount of detail that is preserved scales with the size of the difference. The details of how this is done are complicated, but as with companding, the effect is to make efficient use of bits to store information, taking account of its rate of change.

ITU Recommendation G.721 specifies a form of ADPCM representation for use in telephony, with data rates of 16 kbps and 32 kbps.

[15] Pulse Code Modulation is the term used in audio and communications circles for encoding digital data as a sequence of pulses representing ones and zeroes. Whereas this is more or less the only sensible representation for computer use, alternatives, such as Pulse Width Modulation, exist where the data is to be represented as a stream for transmission, rather than as stored values.

Lower rates can be obtained by a much more radical approach to compression. *Linear Predictive Coding* uses a mathematical model of the state of the vocal tract as its representation of speech. Instead of transmitting the speech as audio samples, it sends parameters describing the corresponding state of the model. At the receiving end, these parameters can be used to construct the speech, by applying them to the model. The details of the model and how the parameters are derived from the speech lie beyond the scope of this book. Speech compressed in this way can be transmitted at speeds as low as 2.4 kbps. Because the sound is reconstructed algorithmically, it has a machine-like quality, so it is only suitable for applications where the content of the speech is more important than a faithful rendition of someone's voice.

Perceptually Based Compression

The secret of effective lossy compression is to identify data that doesn't matter — in the sense of not affecting perception of the signal — and throw it away. If an audio signal is digitized in a straightforward way, data corresponding to sounds that are inaudible may be included in the digitized version. This is because the signal records all the physical variations in air pressure that cause sound, but the perception of sound is a sensation produced in the brain, via the ear, and the ear and brain do not respond to the sound waves in a simple way.

Two phenomena in particular cause some sounds not to be heard, despite being physically present. Both are familiar experiences: a sound may be too quiet to be heard, or it may be obscured by some other sound. Neither phenomenon is quite as straightforward as it might appear.

The *threshold of hearing* is the minimum level at which a sound can be heard. It varies non-linearly with frequency, as shown in Figure 12.12. A very low or very high frequency sound must be much louder than a mid-range tone to be heard. It is surely no coincidence that we are most sensitive to sounds in the frequency range that corresponds to human speech. When compressing sound, there is no point in retaining sounds that fall below the threshold of hearing, so a compression algorithm can discard the corresponding data. To do this, the algorithm must use a *psycho-acoustical model* — a mathematical description of aspects of the

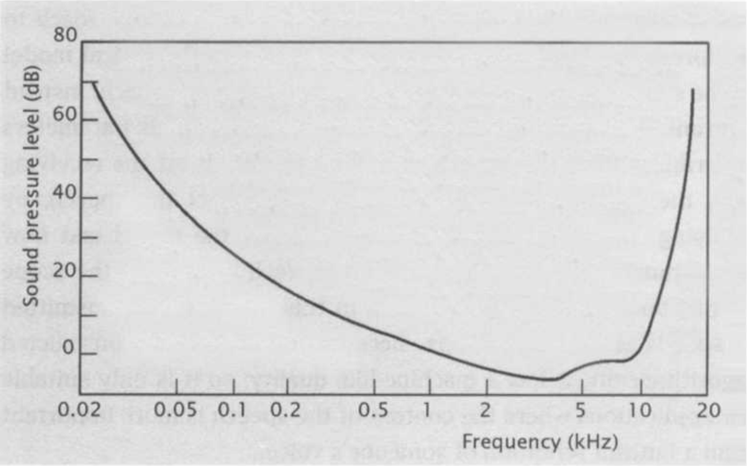


Figure 12.12
The threshold of hearing

way the ear and brain perceive sounds. In this case, what is needed is a description of the way the threshold of hearing varies with frequency.

Loud tones can obscure softer tones that occur at the same time.[16] This is not simply a case of the loud tone 'drowning out' the softer one; the effect is more complex, and depends on the relative frequencies of the two tones. *Masking*, as this phenomenon is known, can be conveniently described as a modification of the threshold of hearing curve in the region of a loud tone. As Figure 12.13 shows, the threshold is raised in the neighburhood of the masking tone. The raised portion, or *masking curve* is non-linear, and assymmetrical, rising faster than it falls. Any sound that lies within the masking curve will be inaudible, even though it rises above the unmodified threshold of hearing. Thus, there is an additional opportunity to discard data. Masking can be used more cleverly, though. Because masking hides noise as well as some components of the signal, quantization noise can be masked. Where a masking sound is present, the signal can be quantized relatively coarsely, using fewer bits than would otherwise be needed, because the resulting quantization noise can be hidden under the masking curve.

It is evident that the phenomena just described offer the potential for additional compression. It is not obvious how a compression algorithm can be implemented to take advantage of this potential. The approach usually adopted is to use a bank of filters to split the signal into bands of frequencies; thirty two bands are commonly

[16] In fact, they can also obscure softer tones that occur a little later or, strange as it may seem, slightly earlier.

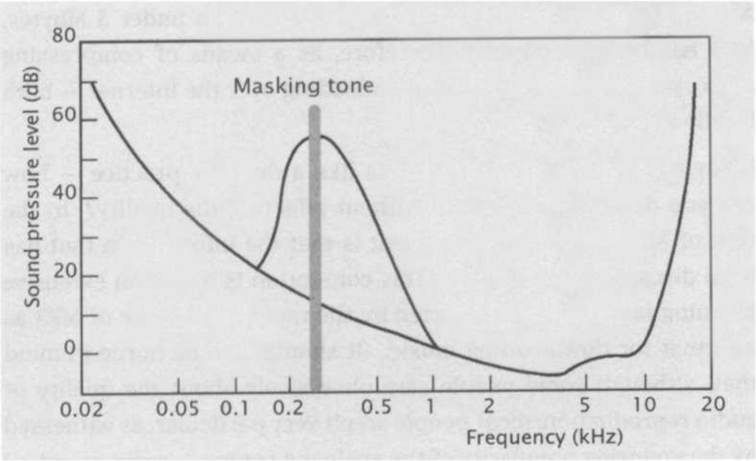


Figure 12.13
Masking

used. The average signal level in each band is calculated, and using these values and a psycho-acoustical model, a masking level for each band is computed. That is, it is assumed that the masking curve within each band can be approximated by a single value. If the signal in a band falls entirely below its masking level, that band is discarded. Otherwise, the signal is quantized using the least number of bits that causes the quantization noise to be masked.

Turning the preceding sketch into a working algorithm involves many technical details that lie beyond the scope of this book. The best known algorithms that have been developed are those specified for audio compression in the MPEG standards. MPEG-1 and MPEG-2 are primarily video standards, but, since most video has sound associated with it, they also include audio compression. MPEG audio has been so successful that it is often used on its own purely for compressing sound, especially music.

MPEG-1 specifies three *layers* of audio compression. All three layers are based on the principles just outlined. The encoding process increases in complexity from Layer 1 to Layer 3, while as a result, the data rate of the compressed audio decreases, from 192 kbps for each channel at Layer 1, to 128 kbps at Layer 2, and 64 kbps at Layer 3. (These data rates will be doubled for stereo.) The audio part of the MPEG-2 standard is essentially identically with MPEG-1 audio, except for some extensions to cope with surround sound.

MPEG-1 Layer 3 audio, or *MP3* as it is usually called,[17] achieves compression ratios of around 10:1, while maintaining high quality.

[17] MP3 does not, despite what you will sometimes read, stand for MPEG-3. There is no MPEG-3.

A typical track from a CD can be compressed to under 3 Mbytes. MP3 has become popular, therefore, as a means of compressing audio, particularly music, for downloading over the Internet — both legally and illegally.

Lossy compression always sounds like a dubious practice — how can you discard information without affecting the quality? In the case of MPEG audio, the argument is that the information that has been discarded is inaudible. This contention is based on extensive listening tests, and is supported by the rapid acceptance of MP3 as a format for downloading music. (It should also be borne in mind that, although some people care obsessively about the quality of audio reproduction, most people aren't very particular, as witnessed by the enduring popularity of the analogue compact audio cassette.) As with any lossy form of compression, though, MPEG audio will deteriorate progressively if it is decompressed and recompressed a number of times. It is therefore only suitable as a delivery format, and should not be used during production, when uncompressed audio should be used whenever possible.

Formats

Most of the development of digital audio has taken place in the recording and broadcast industries, where the emphasis is on physical data representations on media such as compact disc and digital audio tape, and on data streams for transmission and playback. There are standards in these areas that are widely adhered to. The use of digital sound on computers is a much less thoroughly regulated area, where a wide range of incompatible proprietary formats and *ad hoc* standards can be found. The standardizing influence of the Internet is, as yet, less pronounced in audio than it is in graphics.

File Formats

Each of the three major platforms has its own sound file format: AIFF for MacOS, WAV (or WAVE) for Windows, and AU[18] for Unix. All three file types have evolved over the years until they now provide broadly similar capabilities. Each can store audio data at a variety of commonly used sampling rates and sample sizes, including the CD and DAT standard values. Each supports uncompressed or

[18] More properly, WAV is the Audio Waveform file format, actually a variety of Microsoft RIFF file, and AU is the NeXT/Sun audio file format.

compressed data, with a range of compressors, including μ-law and A-law (although not, at the time of writing, MP3). They are all, to some extent, extensible formats, which will be able to accommodate new compressors and sampling rates in the future. But each one has its own idiosyncrasies; in particular, AU files impose some constraints on the combinations of compressor, sampling rate and sample size allowed.

MP3 has its own file format, which does not accommodate sound compressed by any other method.

As well as pure audio formats, video and other multimedia formats can accommodate sound as one media type among several. QuickTime and AVI files usually include one or more audio tracks in addition to video, and they are sometimes used purely for audio. SWF (Flash) files can also be used to store sound.

Although no standard audio file format has emerged, support for AIFF, WAV, and AU files is common on all platforms. In particular, QuickTime — and, therefore, any program that uses it — is capable of reading and writing sound in all three of these formats, as well as MP3 and some other formats associated with particular sound applications.

Streaming Audio

In Chapter 10, we explained that streamed video resembles broadcast television. Streamed audio resembles broadcast radio. That is, sound is delivered over a network and played as it arrives, without having to be stored on the user's machine first. As with video, this allows live transmission and the playing of files that are too big to be held on an average-sized hard disk. Because of the lower bandwidth required by audio, streaming is more successful for sound than it is for video.

The first succesful streaming audio format was Real Networks' RealAudio, a companion to RealVideo, which has now been incorporated into Real G2. This uses a proprietary compressor to maintain reasonable sound quality at data rates suitable for dial-up Internet connections. Streaming QuickTime can also be used for audio, on its own as well as accompanying video. Both formats are used for broadcasting live concerts and for the Internet equivalent of radio stations, and also for providing 'play on demand' music — the Internet as the biggest jukebox in the world.

MP3 requires too much bandwidth to be streamed over modems, although it is well within the capacity of cable modems and ADSL. A 'lo-fi' MP3 format is in use, however, which sacrifices some quality to bandwidth, allowing MP3 audio to be streamed at lower data rates.

⇨ Another streamed audio format has a somewhat different emphasis. Liquid Audio is intended as an architecture for distributing music. Liquid Audio can be streamed, in order to preview a song, for example, but the primary intention is for the user then to download a CD-quality file, which can then be burned on to a recordable CD. Mechanisms for payment and the prevention of piracy are built in to the architecture.

MIDI

If we had written a piece of music, there are two ways we could send it to you. We could play it, record the performance, and send you the recording, or we could write it down using some form of notation, indicating the arrangement, and send you the sheet music, so you could play the piece for yourself. There is a parallel here with bitmapped and vector graphics. In the first case, we send you the actual sound, in the second, we send you what amounts to a set of instructions telling you how to produce the sound. In either case we are making some assumptions about what you can do: for the recording, we assume you have a machine capable of playing back whichever medium we have recorded our performance on. For the sheet music, we are making the more demanding assumptions that you can read our chosen music notation, have access to the instrument or instruments indicated in the arrangement, and can either play yourself or can get musicians to play those instruments. If the music is arranged for a symphony orchestra, this may present some difficulties for you, whereas, if we were to send a recording, all the difficulties would lie at our end.

In the digital realm, there is a similar choice of options for delivering music. So far, we have considered ways of delivering digitized sound, that is, the equivalent of recordings. There also exists an equivalent to delivering the sheet music, i.e. a way of delivering instructions about how to produce the music, that can be interpreted by suitable software or hardware. Similar assumptions must be made: for sound files, you must have software that can read them — as we have seen, this is not a demanding requirement.

For instructions, you must have software that can interpret the instructions, and some means of producing sounds that correspond to the appropriate instruments.

MIDI (The Musical Instruments Digital Interface) provides a basis for satisfying these requirements. Originally, MIDI was devised as a standard protocol for communicating between electronic instruments, such as synthesizers, samplers, and drum machines. By defining a standard hardware interface, and a set of instructions indicating such things as the start and end of a note, it provided a means of controlling a collection of such instruments from a single keyboard, removing the requirement for the huge banks of keyboards beloved of certain 1970s rock stars, and opening the way for playing traditional keyboard instruments, particularly synthesizers, with other controllers, such as drum pads or wind instruments.

More significantly, perhaps, MIDI allowed instruments to be controlled automatically by devices that could be programmed to send out sequences of MIDI instructions. Originally, *sequencers*, as these devices are known, were dedicated hardware devices, programmed using their own built-in, relatively clumsy, interfaces. It was not long before it was realized that computer programs could offer a more convenient and flexible means of sequencing, provided that a computer could be fitted with a MIDI interface so that it could send the necessary signals to other MIDI devices. Such an interface is a relatively simple and inexpensive device, so computer-based sequencers rapidly became available. A software sequencer provides editing and compositional functions, so it needs to store MIDI sequences in files. This requirement led to the development of a standard file format for MIDI files, that is, a way of storing MIDI on disk. Clearly, such files can be exchanged between computers equipped with MIDI software. They can also be incorporated into multimedia productions.

Playing back MIDI files requires an instrument that understands MIDI, but a computer, equipped with suitable hardware or software, can be such an instrument itself. Sounds can either be synthesized on a sound card, or held on disk in the form of samples, to be played back in response to MIDI instructions. MIDI files are therefore a means of communicating music. Because they do not contain any audio data, they can be much more compact than actual digitized sound files. For the same reason, though, they cannot guarantee the same fidelity: the samples available when the file is produced may be of higher quality than those used to play it back — just as the musician who plays a piece of music from a score may not be sufficiently accomplished to realise the composer's intentions. In both cases, the result is unpredictable.

MIDI Messages

A MIDI *message* is an instruction that controls some aspect of the performance of an instrument. Messages are encoded in much the same way as machine instructions: a *status byte* indicates the type of the message, and is followed by one or two *data bytes* giving the values of parameters. Although wind instruments, drum pads, and guitars are used as MIDI controllers (as devices that transmit MIDI signals are called), MIDI is markedly biased towards keyboard instruments. Thus, for example, the most commonly used message is 'Note On', which takes two parameters: a number between 0 and 127 indicating the note to be sounded, and a key velocity, indicating how fast the key was pressed, and hence the attack of the note. When an actual keyboard is being used to generate MIDI messages, these values will be sensed by the keyboard's hardware as the musician plays the key. When the message is being generated by software, the values are specified by the user.

Other notable MIDI messages include 'Note Off', which ends a note, 'Key Pressure', which indicates the degree of 'aftertouch' to be applied, and 'Pitch Bend', to change note values dynamically, as a guitarist does by bending the string (on MIDI keyboards, a wheel is used for this function).

The status and data bytes in a stream of MIDI instructions are distinguishable by the value of their most significant bit. This makes possible an optimization: where a sequence of messages all have the same status byte, it may be omitted from the second and subsequent messages, for which it will be inferred from the most recent value. This arrangement is called *running status*; it can save an appreciable number of bytes where a sequence of notes is being played with no modifications. Using the convention that the end of a note can be indicated by a 'Note On' message with a velocity of zero, the whole sequence can consist of a single 'Note On' status byte, followed by a series of data bytes giving the notes to be played and the velocities to be applied to them.

When a MIDI message is interpreted, we say that an event occurs. In live performance, the timing of events is determined by the player in real time. In a MIDI file, it is necessary to record the time of each event. Each message is preceded by a *delta time*, that is, a measure of the time since the preceding event. Near the beginning of the file is a specification of the units to be used for times.

General MIDI and the QuickTime Music Architecture

The preceding account indicates how notes are produced, but leaves unanswered the question of how they are to be associated with particular sounds. Typically, the sorts of instruments controlled by MIDI — synthesizers and samplers — provide a variety of *voices*. In the case of synthesizers, these are different synthesized sounds;[19] in the case of samplers, different instrument samples. A MIDI 'Program Change' message selects a new voice, using a value between 0 and 127. The mapping from these values to voices is not specified in the MIDI standard, and may depend on the particular instrument being controlled. There is thus a possibility that a MIDI file intended to specify a piece for piano and violin might end up being played on trombone and kettle drum. To help overcome this unsatisfactory situation, an addendum to the MIDI standard, known as *General MIDI*, was produced, which specifies 128 standard voices to correspond to the values used by Program Change messages. The assignments are shown in Table 12.1.

For drum machines and percussion samplers, Program Change values are interpreted differently as elements of drum kits — cymbals of various sorts, snares, tom-toms, and so on (see Table 12.2).

General MIDI only associates program numbers with voice *names*. There is no guarantee that identical sounds will be generated for each name by different instruments — a cheap sound card may attempt to synthesize all of them, while a good sampler may use high quality samples of the corresponding real instruments. Adherence to General MIDI offers some guarantee of consistency, though, which is otherwise entirely missing.

QuickTime incorporates MIDI-like functionality. QuickTime Musical Instruments provide a set of instrument samples[20], and the QuickTime Music Architecture incorporates a superset of the features of MIDI. QuickTime can read standard MIDI files, so any computer with QuickTime installed can play MIDI music using software alone.

[19] Often called *patches* by synthesists.

[20] The Roland Sound Canvas set is distributed with QuickTime.

Table 12.1
General MIDI voice numbers

1	Acoustic Grand Piano
2	Bright Acoustic Piano
3	Electric Grand Piano
4	Honky-tonk Piano
5	Rhodes Piano
6	Chorused Piano
7	Harpsichord
8	Clavinet
9	Celesta
10	Glockenspiel
11	Music Box
12	Vibraphone
13	Marimba
14	Xylophone
15	Tubular bells
16	Dulcimer
17	Draw Organ
18	Percussive Organ
19	Rock Organ
20	Church Organ
21	Reed Organ
22	Accordion
23	Harmonica
24	Tango Accordion
25	Acoustic Nylon Guitar
26	Acoustic Steel Guitar
27	Electric Jazz Guitar
28	Electric clean Guitar
29	Electric Guitar muted
30	Overdriven Guitar
31	Distortion Guitar
32	Guitar Harmonics
33	Wood Bass
34	Electric Bass Fingered
35	Electric Bass Picked
36	Fretless Bass
37	Slap Bass 1
38	Slap Bass 2
39	Synth Bass 1
40	Synth Bass 2
41	Violin
42	Viola
43	Cello
44	Contrabass
45	Tremolo Strings
46	Pizzicato Strings
47	Orchestral Harp
48	Timpani
49	Acoustic String Ensemble 1
50	Acoustic String Ensemble 2
51	Synth Strings 1
52	Synth Strings 2
53	Aah Choir
54	Ooh Choir
55	Synvox
56	Orchestra Hit
57	Trumpet
58	Trombone
59	Tuba
60	Muted Trumpet
61	French Horn
62	Brass Section
63	Synth Brass 1
64	Synth Brass 2
65	Soprano Sax
66	Alto Sax
67	Tenor Sax
68	Baritone Sax
69	Oboe
70	English Horn
71	Bassoon
72	Clarinet
73	Piccolo
74	Flute
75	Recorder
76	Pan Flute
77	Bottle blow
78	Shakuhachi
79	Whistle
80	Ocarina
81	Square Lead
82	Saw Lead
83	Calliope
84	Chiffer
85	Synth Lead 5
86	Synth Lead 6
87	Synth Lead 7
88	Synth Lead 8
89	Synth Pad 1
90	Synth Pad 2
91	Synth Pad 3
92	Synth Pad 4
93	Synth Pad 5
94	Synth Pad 6
95	Synth Pad 7
96	Synth Pad 8
97	Ice Rain
98	Soundtracks
99	Crystal
100	Atmosphere
101	Bright
102	Goblin
103	Echoes
104	Space
105	Sitar
106	Banjo
107	Shamisen
108	Koto
109	Kalimba
110	Bagpipe
111	Fiddle
112	Shanai
113	Tinkle bell
114	Agogo
115	Steel Drums
116	Woodblock
117	Taiko Drum
118	Melodic Tom
119	Synth Tom
120	Reverse Cymbal
121	Guitar Fret Noise
122	Breath Noise
123	Seashore
124	Bird Tweet
125	Telephone Ring
126	Helicopter
127	Applause
128	Gunshot

QuickTime can also control external MIDI devices. MIDI tracks can be combined with audio, video or any of the other media types supported by QuickTime.

MIDI Software

MIDI sequencing programs, such as Cakewalk Metro and Cubase, perform capture and editing functions equivalent to those of video editing software. They support multiple tracks, which can be allocated to different voices, thus allowing polytimbral music to be constructed. In addition, such packages support composition.

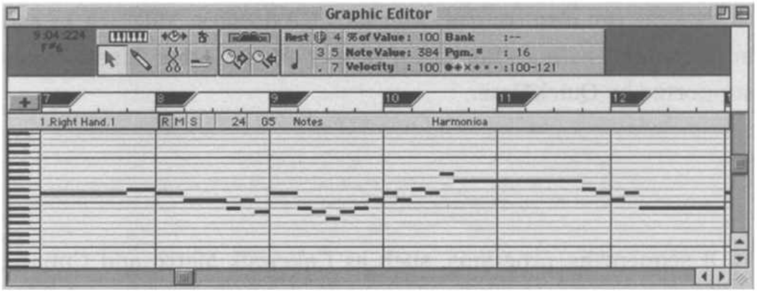
Table 12.2
General MIDI Drum Kit Numbers

35	Acoustic Bass Drum
36	Bass Drum 1
37	Side Stick
38	Acoustic Snare
39	Hand Clap
40	Electric Snare
41	Lo Floor Tom
42	Closed Hi Hat
43	Hi Floor Tom
44	Pedal Hi Hat
45	Lo Tom Tom
46	Open Hi Hat
47	Low -Mid Tom Tom
48	Hi Mid Tom Tom
49	Crash Cymbal 1
50	Hi Tom Tom
51	Ride Cymbal 1
52	Chinese Cymbal
53	Ride Bell
54	Tambourine
55	Splash Cymbal
56	Cowbell
57	Crash Cymbal 2
58	Vibraslap
59	Ride Cymbal 2
60	Hi Bongo
61	Low Bongo
62	Mute Hi Conga
63	Open Hi Conga
64	Low Conga
65	Hi Timbale
66	Lo Timbale

Music can be captured as it is played from MIDI controllers attached to a computer via a MIDI interface. The sequencer can generate metronome ticks to assist the player to maintain an accurate tempo. Although it is common to simply use the sequencer as if it were a tape recorder, to capture a performance in real time, sometimes MIDI data is entered one note at a time, which allows musicians to 'play' music that would otherwise be beyond their competence. Facilities normally found in conventional audio recording software are also available, in particular, the ability to 'punch in': the start and end point of a defective passage are marked, the sequencer starts playing before the beginning, and then switches to record mode, allowing a new version of the passage to be recorded to replace the original.

Sequencers will optionally *quantize* tempo during recording, fitting the length of notes to exact sixteenth notes, or eighth note triplets, or whatever duration is specified. This allows rhythmically loose playing to be brought into strict tempo, which may be felt desirable for certain styles of music, although the result often has a machine-like feel, since live musicians rarely play exactly to the beat. Quantization is usually necessary if it is desired to produce a transcription of the music, since otherwise the program will transcribe exactly what was played, even if that involves dotted sixty-fourth notes and rests.

Most programs allow music to be entered using classical music notation, often by dragging and dropping notes and other symbols onto a stave. Some programs allow printed sheet music to be scanned, and will perform optical character recognition to transform the music into MIDI. The opposite transformation, from MIDI to a printed score, is also often provided, enabling transcriptions



Graphic Editor

9.04.224 F#6

Rest 4 % of Value : 100 Bank : --

3 5 Note Value : 384 Pgm. # : 16

. 7 Velocity : 100 ●◆×◆•• :100-121

1 Right Hand 1 R M S 24 G5 Notes Harmonica

Figure 12.14
The 'piano-roll' interface to a sequencer

of performed music to be made automatically. Those who do not read music usually prefer to use the 'piano-roll' interface, illustrated in Figure 12.14, which allows the duration of notes to be specified graphically, essentially using a timeline. Their pitch is specified with reference to the piano keyboard shown on the left. For music constructed out of repeating sections, loops can be defined and reused many times.

Once a piece of music has been recorded or entered, it can be edited: individual notes' pitch and duration can be altered, sections can be cut and pasted, or global changes can be made, such as transposing the entire piece into a different key, or changing the time signature. The parameters of individual MIDI events can be changed — the velocity of a note can be altered, for example. Voices can be changed to assign different instruments to the parts of the arrangement.

Because digital audio is very demanding of computer resources, but MIDI is much less so, the two forms of music representation were originally separated, with different software being used for each. Now that computers have become powerful enough to take audio in their stride, the two are commonly integrated in a single application, which allows MIDI tracks to be combined and synchronized with full audio. This arrangement overcomes one of the major limitations of MIDI, namely the impossibility of representing vocals (except for 'Oohs' and 'Aahs'). MIDI can be transformed into audio, much as vector graphics can be rasterized and transformed into pixels. The reverse transformation is sometimes supported, too, although it is more difficult to implement. MIDI captures the musical structure of sound, since MIDI events correspond to notes. Being able to transform audio into MIDI allows music to be recorded from ordinary instruments instead of MIDI controllers — it can even be recorded from somebody's whistling — and then edited or transcribed in terms of musical notes.[21]

[21] It is sometimes claimed that facilities of this sort allow people who cannot play instruments to compose tunes by whistling. You have to be able to whistle in tune, though.

⇨ The *MOD file* is a file format which is being increasingly used for music. Originating on the Amiga platform, MOD (short for 'module') files combine the features of audio and MIDI files. In particular, they can include the necessary samples to use to play back a piece of music whose structure is described by MIDI information. Programs for playing MOD files are called *trackers.*

Combining Sound and Picture

Sound is frequently used as a part of a video or animation production. When it is, synchronization between sound and picture becomes a matter of considerable importance. This is seen most clearly where the picture shows a person talking, and the soundtrack contains their speech. If synchronization is slightly out, the result will be disconcerting; if it is considerably out, the result will at best be unintentionally funny, but more likely, incoherent. Although speech makes the most exacting demands on synchronization, wherever sound and picture are related, it is necessary that the temporal relationship between them is maintained. Voice-overs should match the picture they describe, music will often be related to edits, and natural sounds will be associated with events on screen.

In order to establish synchronization, it is necessary to be able to identify specific points in time. Film is divided into frames, which provide a natural means of identifying times. Video tape does not have physical frames, but, as we mentioned in Chapter 10, timecode can be written to a video tape, allowing the frames to be identified. Audio tape can be similarly augmented with timecode, and the codes can be used to synchronize tape recorders, both audio and video. As long as the timecodes on two machines agree, they must be in synch.

Sound is effectively continuous, though, even in the digital domain — the high sampling rates used for digital sound mean that a single sample defines too short a time interval to be useful. For sound, the division into frames imposed by timecode is just a useful fiction. This fictional division continues to be used when synching digital audio and video. It enables sound and picture tracks in a video editing application such as Premiere to be arranged on the same timeline. Unlike the soundtrack on a piece of film or a video tape, a sound track in Premiere is physically independent of the video it accompanies, so it is easy to move the sound in time relative to the picture, simply by sliding the sound track along the timeline. This is not something you would normally want to do if the sound and picture had originally been recorded together. In that case, you will usually want to maintain their synchronization during editing. For this purpose, tracks can be locked together, so that, for example, cutting out part of the video track will remove the accompanying sound.

Audio tracks may be displayed as waveforms. When a sound track has been made independently of the picture — a voice-over or musical accompaniment, for example — it will be necessary to fit the sound to the picture. By looking at the waveform to identify the start of syllables in speech, or stressed beats in music, an editor can identify meaningful points in the sound track, which can be lined up with appropriate picture frames. Performing this matching by eye is difficult, so a method that is often used is to scrub through the sound to identify the precise cue point by ear, and place a marker (a facility offered by Premiere, for example) which can then be lined up on the timeline with the video frame (which can also be marked for identificiation). Sometimes, it may be necessary to apply a time-stretching filter to adjust the duration of the sound to fit the picture, as described earlier.

Synchronization can thus be established in a video editing program, but it must then be maintained when the video and its sound track are played back — possibly over a network. If the sound and video are physically independent — travelling over separate network connections, for example — synchronization will sometimes be lost. This is a fact of life and cannot be avoided. Audio and video data streams must therefore carry the equivalent of timecode, so that their synchronization can be checked, and they can be resynched, if necessary. Usually, this will require some video frames to be dropped, so that picture can catch up with sound — the greater data rate of the video means that it is the more likely to fall behind.

Where video and audio are played back from a local hard disk, it is easier to maintain synchronization, although it still cannot be guaranteed, owing to the different speeds of the components involved. For very short clips, it is possible to load the entire sound track into memory before playback begins, eliminating any potential delays caused by reading from disk. This is impractical for movies of any significant duration. For these, it is normal to *interleave* the audio and video, that is, the audio is divided into chunks, which are interspersed between video frames.[22] The size of chunk can be varied, and optimized for particular hardware configurations.

➲ QuickTime provides highly flexible support for specifying time within a movie. A movie has a *time base*, specifying the current time and the rate (and direction) at which time passes, relative to a *time coordinate system*, which specifies the units in which time is to be measured. The time coordinate system is chosen to suit the data; for example, sound sampled at 44.1 kHz may use a time coordinate in which each time unit is 1/44.1 ms. A *clock component* is used to obtain real time values, usually from the clock chip built in to the computer on which the movie is being played. By synchronizing the time base to the clock, the required playback rate is maintained. Whenever some data, such as a frame of video, is to be played, the time base is passed to the software component responsible for playing the data. The current movie time can then be used to determine the appropriate portion of the data to play. If sound and picture start to drift, this will ensure that they are brought back into the correct temporal relationship.

Further Information

[Poh95] is an excellent introduction to all aspects of digital audio, and [Rot92] is a good, brief account of MIDI. [RM94] describes the techniques of sound recording.

Exercises

1. Give an example of a natural sound that has an identifiable pitch.
2. Why are the sampling frequencies normally used for 'lo-fi' digital sound exact sub-multiples of 44.1 kHz?
3. Given that singing has characteristics of both speech and music, which compression algorithms would you expect to be most successful on songs?
4. A problem commonly encountered when recording in the open air is that a microphone will pick up the sounds made

[22] AVI stands for 'audio and video interleaved', even though by default they aren't.

by the wind blowing against it. Describe how you would attempt to remove such noise from a digitized recording. How successful would you expect your attempts to be? Suggest an alternative approach to eliminating wind noise.

5. When we described anti-aliasing of vector graphics in Chapter 4, it was as an antidote to insufficiently high sampling rates. In this chapter, we have described dithering as a way of mitigating the effect of insufficient quantization levels. Would this kind of dithering help improve the apparent quality of under-sampled audio? Is there any connection between dithering sound and dithering colour, as described in Chapter 6?

6. Explain how you would apply a 'cross-fade' to a stereo recording of a solo instrument, so that the sound would appear to move gradually from the extreme left to the extreme right of the sound stage.

7. What could you do to correct a sound that was digitized with its levels (a) too high; (b) too low. How would you prepare a sound recording with an extremely wide dynamic range for a multimedia production?

8. Is there a limit on how far you can (a) stretch, (b) contract a digitized sound successfully? What aspects of particular kinds of sound might affect the limits?

9. (a) Describe the alterations that must be made to the digital representation of a sound to raise its pitch by an octave, without changing its duration. (Raising the pitch by an octave is the same as doubling the frequency.)

   (b) Describe the alteration that must be made to a MIDI Note-On message to raise the note's pitch by an octave.

10. Under what circumstances might you expect to lose synchronization between sound and picture in a multimedia production? What steps could you take to minimize the chances of this happening.

Combining Media

13

Broadly speaking, there are two models currently in use for combining elements of different media types: page-based and synchronization-based.

In the first, text, images, and video are laid out in a two-dimensional arrangement that resembles the way text and images are laid out in books and magazines. Time-based elements, such as video clips and sound, are embedded in the page as if they were images, occupying a fixed area; controls may be provided to start and stop playback. Individual pages can be combined using the linking mechanism of hypertext. Such linked page-based multimedia productions are known as *hypermedia*. The best known example of a hypermedia system is the World Wide Web.

Hypermedia takes its inspiration from paper-based media that are essentially static, and grafts time-based features onto a spatially organized framework. In contrast, synchronization-based multimedia makes time the central organizing principle. Elements are arranged in time, often using a timeline, so that they are presented as a sequence, resembling a slide show. *Transitions*, such as dissolves and wipes, may be used to go from one element in the sequence to the next. Unlike most (but not all) slide shows, however, a multimedia *presentation* (as such timeline-based productions are usually known) will often incorporate parallelism: several video clips may be shown at the same time, perhaps overlaid against a static image, or a sound track may play during an animation. Elements may be synchronized, so that, for example, some text may be displayed as long as a video clip is playing, or an image will be displayed ten seconds after the beginning of a video clip.

> ⇨ A third model for combining media is exemplified by VRML, as we described briefly in Chapter 11. Here, the elements are laid out in a three-dimensional scene. The user can move about in the 3-D space and inspect images or objects that they encounter. Links may be incorporated, to provide hypertext functionality, and interactivity can be provided, using the scripting mechanisms that are applied to the other models of multimedia, as we will describe in Chapter 14. Scene-based multimedia of this sort is presently most often found in games, although these are not usually thought of as multimedia productions. The apparent failure of VRML so far to achieve wide acceptance means that currently there is no standard form for scene-based multimedia. The proposed MPEG-4 standard for multimedia, presently under development, uses a scene-based model, but has, as yet, only been implemented experimentally.

Multimedia productions based on both of these models are often augmented with interactive features that allow the user to control their progress. As we will describe in Chapter 14, a scripting facility is usually provided that allows multimedia authors to write simple programs that cause actions to occur in response to user input. Scripts can also be written to cause actions to occur after a certain period of time, or in response to events such as a video clip finishing. By using scripts of this sort, temporal organization can be added to page-based multimedia. Non-linear branching can be provided in a synchronized presentation, either using scripting or by extending the timeline metaphor to support linking. Since it is also the case that the elements of a synchronization-based multimedia presentation must be arranged spatially, the distinction between our models of media combination is somewhat blurred. Nevertheless, their different emphases make it useful to consider them, at least to begin with, as distinct ways of combining media.

> ⇨ More than one introduction to multimedia, when describing the combination of media into multimedia, does so in a chapter entitled 'Putting it all together'. Aside from the regrettable descent into cliché, this title carries the suggestion that it is somehow necessary,

once you have moved beyond a single medium, to combine the whole lot: text, images, video, animation, and sound. It isn't, and may often be undesirable to do so.

We can illustrate this using familiar pre-digital media. Laurence Bergreen has written an excellent biography of Louis Armstrong.[1] However, as it is a book, it offers no opportunity to hear the music, which is something of a deficiency, given its subject. All the important recordings are available, but obtaining them all, collating them with the text, and actually getting round to listening to them, requires a certain amount of effort, discipline and money. Surely a case for multimedia.

Such was presumably the thought in the minds of the producers of the television documentary *The Wonderful World of Louis Armstrong*, which broadly covers the same material as Bergreen's book. The multimedia possibilities of television allowed them to combine extracts from Armstrong's recordings with spoken commentary and reminiscences from those who knew him. However, the documentary makers apparently had little access to any relevant footage or images of the trumpeter, and since they evidently believe in the dictum that television is a visual medium, they were left with the quandary of what to do when there are no appropriate pictures. They chose to use computer graphics and post-production techniques to produce 'evocative' imagery and altered video to accompany the words and music. The result was unwatchable, but not untypical of much contemporary television.

The trouble is that working in a visual medium demands images, whether appropriate ones are available or not. The forms of digital multimedia offer the opportunity to select and combine those media that are suitable to a particular production, without necessarily putting it *all* together.

Hypermedia

Hypertext, as we described it in Chapter 8, never generated much excitement. It required the addition of first graphics and then other media elements to the linking mechanism of hypertext to produce something that captured people's imaginations: *hypermedia*. It is easy to see how this generalization can be achieved by considering HTML and the World Wide Web, though you should realize from the history in Chapter 9, that the Web was by no means the first hypermedia system.

[1] Laurence Bergreen, *Louis Armstrong: An Extravagant Life*, Harper Collins, 1997.

HTML and Hypermedia

In HTML, links are implemented as anchors, with an `href` attribute whose value is a URL pointing to the destination. In previous sections, we assumed that the URL points to a page of HTML. What if it doesn't?

HTTP doesn't care. If you ask a server to return the resource identified by a URL it will do so. As we mentioned in Chapter 2, it will include in its response, as well as the resource's data, an indication of what sort of data it is, in the form of a MIME content type. The presence of MIME content types in HTTP responses answers a question that may have bothered you: how does a Web browser know that what it is receiving is an HTML document? Because the content type is specified as `text/html`. The question begs another: How does an HTTP server know what MIME type to specify when it sends a response? It would be unrealistic to expect the server to analyze the data and deduce its type; a more prosaic solution is adopted. The server has access to a configuration database, maintained by whoever looks after the server, which provides a mapping from filename extensions, or file type and creator codes, to MIME types.

The upshot of all this is that when a browser receives a document it also receives information about the type of data it contains, in the form of a MIME content type. This brings us back to the original question, which we can now rephrase: What does the browser do if the content type is not `text/html`? Either the browser will be intelligent enough to deal with it anyway, or it will have to get another program to do so.

Consider the second option first. Web browsers can be configured so that, for each MIME type, a program is nominated to deal with documents of that type. The nominated programs are usually called *helper applications*. When a document arrives that the browser cannot display itself, it starts up the appropriate helper application to do so instead. For example, suppose (on a Macintosh) your browser had been configured so that the video player Sparkle was the helper application for data of type `video/mpeg` (MPEG video clips). If you click on an anchor whose `href` attribute's URL points to an MPEG video clip, the data will be retrieved as usual. When it arrives, the browser will use the OS's facilities to start up another process to run Sparkle, handing it the retrieved data. The video will be displayed in a new window belonging to Sparkle, with that program's interface, independent of the Web browser. Thus, although you can play the video clip after retrieving it via a Web page, it is not integrated with any other Web data — we have not really achieved hypermedia, although we can link together different media using hyperlinks.

In order to properly integrate the display of media besides formatted text into a Web browser we must extend the capabilities of the browser so that it can render other media, and we must provide some extra HTML tags to control the layout of pages with embedded graphics, video and sound. It is not realistic to expect a Web browser to be able to cope with absolutely any imaginable type of data; for some obscure types a helper application will always be the best option. On the other hand, some types, especially image types and plain text, are so common that it is reasonable to expect browsers to have code to display them built in. Other types, such as video and audio fall in between: it would be nice to handle them in the browser, but they are not yet so common that the necessary implementation effort on the part of the browser manufacturer is justified, and not all users will be happy with the resulting increased resource requirements of their browsers. The solution to this dilemna is *plug-ins*: software modules that can be installed independently by users who need them. Plug-ins are loaded by a browser when it starts up and add functionality to it, specifically the ability to deal with additional media types. The major browsers all use the same interface that allows them to incorporate plug-ins, and they can be configured to use a plug-in instead of a helper application for certain media types. An example of a widely used plug-in is Apple's QuickTime plug-in, which transparently incorporates all the functionality of QuickTime into any browser that implements the Netscape plug-in interface (this includes Internet Explorer). It is written by the QuickTime engineers, who presumably know more about QuickTime than anyone else, so it provides a better implementation of QuickTime displaying than is likely to have been provided if the browsers had been extended by their own developers. Users only need to install the plug-in (usually simply by placing it in an appropriate place) if they want to view QuickTime within their browsers; this way, users with no interest in video do not incur the associated overhead.

Once a browser becomes capable of rendering non-textual data without the aid of a helper, the possibility exists of integrating the display of such data with the other elements of Web pages. This leads to a mode of multimedia presentation based on a *page layout* model. This model is derived from long-established practice in print-based media for combining text and graphics. It can be fairly naturally extended to incorporate video and animation, by treating them as if they were pictures to be placed on the page, but with the added property that they can be made to move. Sound sits much less comfortably on a page, being purely non-visual. An expedient often employed is to represent a sound by an icon or set of controls, which can be treated as a graphic element, and then activated to play the sound in the same way as a video element is activated. An alternative is simply to associate a sound with a page, and have it play when the page is loaded. This approach has the disadvantage that users have no control over whether the sound is played.

Special markup is required for embedding these new elements in Web pages. In HTML 4.0, the OBJECT element is provided for embedded media of all types — it is flexible enough to accommodate new media types that have not yet been implemented, as well as embedded executable content, in particular, Java applets. Previous versions of HTML only provided support for bitmapped images, in the form of the IMG element. Use of this element has become so well established that, for the most popular image formats — JPEG, GIF, and possibly PNG — it is unlikely to be superseded by OBJECT. Before HTML 4.0 there was no officially endorsed method for embedding any other type of media, but Netscape implemented an EMBED element (subsequently also adoped by Microsoft in Internet Explorer) for including video, sound, and other media types. Although EMBED never had any official status, it has been widely used, and is routinely generated by HTML authoring programs.

Part of the enduring attraction of IMG is its simplicity. It is an empty element; in the simplest case, it has a single attribute, src, whose value is a URL pointing to an image file.[2] IMG is an inline element, so the image is displayed where the tag occurs. This enables you to run images in with text, use them as headings or labels on list items, and so on; alternatively, you can isolate an image by enclosing it in its own paragraph.

To understand the necessity for IMG, consider the following two fragments of HTML.

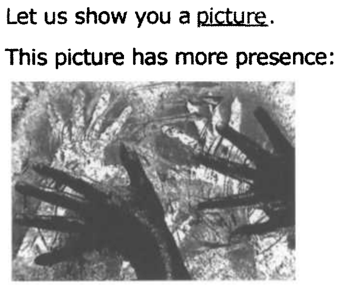
```
Let us show you a <A href="still1.jpg">picture</A>.
```

and

[2] Strictly, in HTML 4, one other attribute, alt, is compulsory — see below.

```
This picture has more presence:
<P>
<IMG src="still1.jpg">
</P>
```

Figure 13.1 shows the corresponding parts of the displayed page. In the first case, the word 'picture' is highlighted as a link, just as if it pointed to another page. When the highlighted text is clicked, the image is displayed alone in the browser window, replacing the current page (or frame). In the second case, the image is displayed as part of the page, like a picture in a newspaper or magazine.



Let us show you a picture.

This picture has more presence:

Figure 13.1
Linked and embedded images

Such clumsy layout would not cut much ice in the magazine design world, though, nor on today's World Wide Web. Like any other HTML document element, an IMG can be laid out in a more imaginative and pleasing way using CSS rules. Margins and borders can be set around images, which can be aligned and floated, using the properties introduced in Chapter 8. For complete layout control, absolute positioning, including z-index, can be used to put images exactly where you want them — subject always to the proviso that not all user agents interpret stylesheet information. (For the benefit of older browsers, IMG elements have a set of attributes, including height, width, border and align to provide some degree of control over their placement.)

OBJECT elements are the preferred way of embedding multimedia and executable content in Web pages.[3] We will not consider the latter at this stage, but we note that there are additional attributes for this purpose. For embedding images, video, sound, and so on, the data attribute is used; the value is a URL that points to the data to be rendered as the object's content. It is advisable also to specify the MIME type of the data, as the value of the type attribute. Although the server will supply this information if the data is requested, providing it in the object's start tag means that a user agent that cannot render data of that type need not waste time downloading it — possibly a non-trivial consideration if the object is a large video clip.

The way in which OBJECT provides for the possibility of a user agent's being unable to display the specified object is ingenious, but slightly counter-intuitive. Unlike an IMG element, an OBJECT has content, but the content is not the object — that is specified by the data attribute. The content is displayed only if the user agent is unable to display the object. This means that the content can be

[3] Preferred, that is, by the W^{3}C. At the time of writing, the implementation of OBJECT in contemporary (fourth generation) browsers is patchy, but EMBED is almost universally supported.

used to provide a replacement version of the object. The following use is typical:

```
<P>Here's a groovy moovy
<P>
<OBJECT data="movies/clip2.mov"
        type="video/quicktime">
  <OBJECT data="images/still2.jpg" type="image/jpeg">
    A 5 second video clip.
  </OBJECT>
</OBJECT>
```

If possible, a movie clip is displayed. If the necessary plug-in is not there, or some other reason prevents playing the movie, a still image is put in its place. If even this cannot be done, some text describing the missing object is used.

It is often necessary or desirable to specify some aspects of the way in which an object is displayed. For example, a video clip might be shown in a standard video controller component, with play, pause and fast-forward controls, or it might be displayed with no controls and play itself automatically. A page designer would want to be able to specify which of these styles is used. However, different options make sense for different media: you cannot sensibly attach a video controller component to a still image, for example. Furthermore, OBJECT elements are intended to be able to support arbitrary media, including those not yet invented. Consequently, it is not possible to define a set of attributes for the OBJECT element type that will provide for all possible eventualities.[4] Instead, parameters required to control the display of objects are set using a special element type: PARAM. This is similar to META, in that it has attributes name and value, which are used to provide a value for a named parameter. The parameters for an object are set in PARAM elements contained in the OBJECT element. The precise set of names that is applicable depends on the type of the object's data (or, on the plug-in used to display the object).

For example, the QuickTime plug-in understands several parameters that can have the value true or false, including controller, which determines whether a movie is displayed with a standard movie controller component, autoplay, which causes the movie to start playing as soon as the page is displayed, if it is true, and loop, which makes it play as a continuous loop. Hence, to make a movie play automatically forever without any visible controls, the following HTML markup could be used.

[4] This argument has extra force when you consider that OBJECTs can be used to contain applets, which may need to be supplied with parameters when they are run. The set of parameters required will be determined by the applet's programmer, and cannot be predicted.

```
<OBJECT data="movies/clip2.mov"
        type="video/quicktime">
  <PARAM name = "controller" value = "false">
  <PARAM name = "autoplay" value = "true">
  <PARAM name = "loop" value = "true">
</OBJECT>
```

⇨ The EMBED element type has attributes with the same names as these parameters.

Links

One of the places you can use images is between the start and end tags of an A element with an href attribute. The effect is to produce a clickable image that serves as the source of a link. This is usually indicated by outlining it in a special colour. A common use of this facility is to create clickable icons or buttons; another is to produce image catalogues consisting of small 'thumbnail' pictures:[5] clicking on a thumbnail causes a full-sized version of the picture to be displayed. Alternatively, it may just be considered more appropriate to have an image serve as the linking element, if it somehow captures the semantics of what it links to. For example, an image of a book's cover may be used as the source of a link to a page of an on-line book store from which it can be purchased. (It is, however, more difficult to convey a link's meaning unambiguously using an image than it is using words.)

⇨ You can use objects as link sources in the same way, but for media other than images it is not clear that this is a useful thing to do. In particular, for time-based media, there is something counter-intuitive about considering the whole object as a link. It seems more natural to use events or particular frames in this way. This raises many questions, since to do this requires us to incorporate time into the hitherto simple, and essentially spatial, model of nodes and links. We will defer consideration of how to go about doing so until the second half of this chapter.

A slightly more elaborate way of using images as links is extremely popular: an *image map* is an image containing 'hot' areas, which are associated with URLs. Clicking on such an area causes the resource identified by its associated URL to be retrieved. Figure 13.2 shows such an image map: it looks like just a picture, but each of the

[5] Web graphics applications often provide a means of constructing such catalogues automatically, just by selecting a directory of image files.

flower heads is the source of a link. To make an image into an image map, it must be associated with a MAP element, by giving it a usemap attribute whose value is a fragment identifier. This must match the value of the name attribute of a MAP. Thus, for example,

```
<IMG src="flower1.jpeg" usemap="#image-map-1">
```

associates the image flower1.jpeg with a map whose start tag looks like:

```
<MAP name="image-map-1">
```

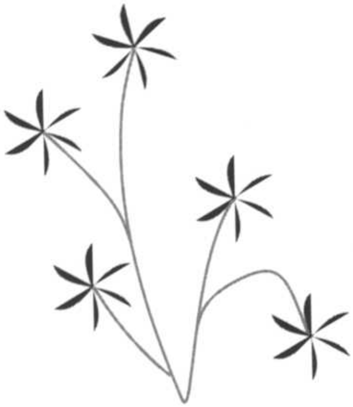


Figure 13.2
An image map

The content of a MAP is a series of AREA elements, each having an href attribute whose value is a URL in the usual way, together with two attributes shape and coords, which together describe a region within the image which is to be linked to the specified URL. The options for shape and the corresponding interpretation of coords are listed in Table 13.1. The MAP element associated with the flower image looks like this:

```
<MAP name="image-map-1">
<AREA shape="rect" coords="65,351,166,413"
      href="section1.html">
<AREA shape="rect" coords="64,353,167,410"
      href="section1.html">
<AREA shape="rect" coords="106,324,124,442"
      href="section1.html">
<AREA shape="rect" coords="351,420,452,482"
      href="section2.html">
 eleven more AREA elements
 </MAP>
```

(As you can see, AREA is an empty element type.) Each flower head is approximated by three intersecting rectangles.

Table 13.1
Shapes and co-ordinates

shape	coords
rect	*left-x, top-y, right-x, bottom-y*
circle	*center-x, center-y, radius*
poly	$x_1, y_1, x_2, y_2, \ldots, x_N, y_N$

Almost nobody will want to measure or calculate the coordinates of areas such as these. Fortunately, many tools are available that enable you to construct a map by selecting an area and typing a URL. Most graphics packages are also capable of constructing image maps. The particular example just shown was made in Illustrator, where it suffices to select an object and type a URL in a text field in the attributes palette. If you then export the image as a GIF or JPEG file you are given the option of having an HTML file containing the image map generated at the same time. Note, however, that whatever the shape of the object you select, only rectangles, circles and polygons can be used as areas in the HTML code.

Synchronization-Based Presentations

Pure hypermedia possesses no temporal structure. The way in which a hypermedia production unfolds in time is determined by the user's choice of links to follow or controls to activate. This is not always a desirable situation. For example, one might want to produce a multimedia demo showing off a computer system's sound, graphics and video capabilities that can be left running unattended at an exhibition or in a shop. For such applications, the order in which the elements of the production are played must be determined beforehand by the author. A popular framework for organizing multimedia in time is the timeline.

Timelines for multimedia authoring work in much the same way as the video editing and animation timelines described in Chapters 10 and 11. The sophistication with which they can be used varies widely. At the most basic end of the scale, simple slide show packages, such as Power Show, allow video clips and images to be sequenced by dragging them onto a 'filmstrip', which is simply a representation of the clips and images in the order they are to be displayed. The duration of each can be set, and transitions — wipes, dissolves, ripples, page turns, and so on — can be applied between them. Text can be superimposed onto the images, and animated, and sound can be associated with each element of the presentation. Although all of this could be achieved with a video editor such as Premiere or post-production package such as After Effects, slide show programs are simpler to use, cheaper to buy, and need not render their output as a movie; instead, they can just produce a set of instructions for playing the various items in the slide show in the designated order.

Timeline-based multimedia authoring can be much more complex, as demonstrated by Macromedia Director. In Director, a collection of media objects is known as a 'cast', with the individual objects being 'cast members'. A Director movie, as the finished production is called, is constructed by arranging cast members in time on a timeline, and in space. More precisely, since cast members can be used many times, pointers to them, known as 'sprites'[6] are arranged to make the movie. The timeline becomes the 'score', which shows, for each frame of the movie, all of the sprites that are active. The spatial arrangement of sprites is performed on the 'stage', which is simply an editable version of the display. Sprites can be animated

[6] The terminology betrays Director's origins as an animation program.

using key frames, as in Flash. To construct a movie, the author drags sprites onto the score, and specifies the number of frames for which they are active (usually by dragging over the score) or sets key frames and specifies tweening parameters.

The playing of a Director movie is described in terms of a playback head travelling through the score one frame at a time, at a rate prescribed by the author.[7] When the playback head is in a particular frame, all the sprites that are shown in the score at the corresponding point are displayed at the appropriate position — either where they have been placed on the stage, or at coordinates interpolated from their positions at nearby key frames. After the correct frame time has elapsed, the playback head moves on to the next frame, whose sprites are displayed. A sprite can refer to a sound or video clip, or even another Director movie, and these have their own natural duration. To ensure that they are played in their entirety (assuming that that is what is desired) they can either be extended over a sufficient number of frames, or the sprite can be placed into a single frame, and using a facility provided for the purpose, the playback head can be made to remain in that frame for as long as it takes the entire clip to play.

Sprite properties such as transparency can be animated as well as their position, so Director can produce motion graphics effects in the same way as a program like After Effects. Transitions can also be used in a similar way to Premiere and other video editing applications. With all these possibilities, the timeline, in the guise of Director's score, can be a vehicle for producing elaborate presentations using all the media types we have considered. The facilities we have described only permit linear presentations; the non-linearity that characterizes hypermedia is missing. It can be provided by the use of additional facilities, which we will describe in Chapter 14.

SMIL

The timeline is a visual metaphor that makes it easy to organize multimedia presentations in time, but it is not the only way that such organization can be expressed. *SMIL (Synchronized Multimedia Integration Language)* is a tag-based language for specifying the temporal structure of a presentation, in a similar way to that in which the structure of a hypermedia document can be specified in HTML. Like HTML, SMIL, being a purely text-based language,

[7] Or as near to that rate as the user's computer can manage.

can be written using any text editor, it doesn't require special authoring tools (although, again like HTML, elaborate tools can be used to generate SMIL code from a more comfortable environment — for example, a timeline). For our purposes, SMIL has the additional advantage that its textual representation lays bare some of the details of synchronization that are concealed by the timeline metaphor, and does so in a way that provides a direct comparison with the page-based hypermedia facilities of HTML.

The SMIL 1.0 specification was adopted as a W^{3}C Recommendation in June 1998. As this suggests, it is intended that SMIL presentations be played over the Internet, using URLs to identify individual media resources stored on a server. The widely used RealPlayer G2, although generally thought of as a program for watching streamed video over the Internet, is actually a SMIL player, and is capable of showing synchronized clips in parallel and sequentially. Similarly, QuickTime 4.1 incorporates SMIL.

SMIL is defined by an XML DTD, so the syntax of its tags will be familiar. At the outermost level, the document structure resembles that of an HTML document: the entire document is a `smil` element, whose content comprises a `head` followed by a `body`. Note that the element names, and thus their start and end tags are written in lower-case; unlike HTML, XML is case sensitive. The `head` may contain `meta` elements, which, again as in HTML, provide an open-ended mechanism for including in the document information about the document. The `head` may also include layout information, describing the spatial disposition of the elements of the presentation on the screen. This takes the form of a `layout` element, which contains definitions written either in CSS2, using its absolute positioning facilities, or in a simpler notation called the *SMIL Basic Layout Language*.[8] In either case, it is conventional (in SMIL Basic it is necessary) to define the location and dimensions of a collection of *regions*, each associated with one or more of the elements that occur in the document body. These have a `region` attribute identifying which of the defined regions should be used to display the element. We will not describe the syntactical details here — consult the SMIL specification if you need to know about the SMIL Basic Layout Language — and in the examples that follow we have elided references to layout and regions to avoid cluttering the presentation.

As you would expect, the real substance of a SMIL document is its body. The most interesting elements that may appear in the

[8] SMIL allows for the possibility of providing alternative layouts in different languages. See the specification for details.

body's content are the *synchronization elements*, for which temporal properties can be specified. These include two compound elements, par (short for 'parallel', and nothing to do with paragraphs) and seq (sequence). Each may contain *media object elements*, which specify the actual images, video clips, sound, and so on, that are used in the presentation (the sprites, in Director's terms). To accommodate complex synchronization relationships, compound synchronization elements may be nested. Elements that occur within a par element may overlap in time, whereas those within a seq are displayed in sequence. A simple example will illustrate how this works, and serve to introduce the attributes that SMIL uses to specify timing and synchronization.

Suppose we wish to assemble four elements for a presentation: two images, a QuickTime movie, and a sound clip, and that we want to present them so that the movie starts to play at the beginning of the presentation, and plays in its entirety. Initially, the first image is displayed together with the movie, but after five seconds it is replaced by the other image, which is displayed for ten seconds before the first image comes back for another fifteen seconds. While this is going on, the sound should start playing after five seconds, and continue until twenty seconds into the presentation. (You can safely assume that the layout defined for this presentation causes the various components to be arranged in a sensible way on the screen.) We have three things going on in parallel here: a movie, images, and sound, with the images themselves being displayed in a sequence, so the structure of the SMIL document's body will be a par element containing media objects for the movie and sound and a seq element containing the sequence of images. The complete body is as follows:[9]

```
<body>
  <par>
    <video src="movies/m1.mov"
           type="video/quicktime"/>
    <seq>
      <img src="images/image1.jpeg"
           type="image/jpeg"
           dur="5s"/>
      <img src="images/image2.jpeg"
           type="image/jpeg"
           dur="10s"/>
      <img src="images/image1.jpeg"
           type="image/jpeg"
```

[9] Indentation has been used to emphasise the structure; it has no syntactical significance.

```
            dur="15s"/>
    </seq>
    <audio src="sounds/sound1"
            type="audio/aiff"
            begin="5s" end="20s"/>
  </par>
</body>
```

The media object elements are all empty;[10] they all use a `src` attribute to provide a URL specifying the whereabouts of the media data. (Here we have used relative URLs.) The media object elements allowed are `animation`, `audio`, `img`, `ref`, `text`, `textstream`, and `video`. The intention of most of these should be clear. The `ref` element is a catch-all for media that are not accurately described by any of the other tags. In fact, the other tags are purely descriptive synonyms for `ref`: the type of data is determined by the `type` attribute, which specifies a MIME type, or by information provided by the server.

The first element in the `par` thus specifies the movie. The next is a `seq` for the sequenced images. We must use three `img` elements, even though two of them relate to the same image — each element corresponds to an occurrence of the image during a particular period of time. The elements within a `seq` are displayed in the order they appear textually, as you would probably expect. The duration of each can be specified using the `dur` attribute, which takes a SMIL *clock value*. Clock values can either be specified as a number of hours, minutes, seconds (as in our example) or milliseconds, using the abbreviations `h`, `m`, `s`, and `ms`, or in the form *hours:mins:secs.fraction*, which is much like a timecode, except that a fraction is used instead of a number of frames, since, in general, we cannot rely on any particular frame rate. The *hours:* and *.fraction* parts are optional. A special value `indefinite` can be used for `dur` to specify that we wish an element to continue playing indefinitely.

Instead of specifying a duration, we can specify a start and end time for the display of a media object, using attributes `begin` and `end`. Whereas durations are time differences, and therefore need no frame of reference, start and end points must be specified relative to some time origin. What this is depends on the enclosing synchronization element. For elements occurring within a `par`, times are measured relative to the start of the whole `par` element; for those occurring within a `seq`, they are measured relative to the end time of the preceding element. Hence, when we specified the

[10] Since this is XML, their tags end with />.

begin attribute of the audio clip in the example above as 5s, this made the sound start playing five seconds after the beginning of the presentation (since the containing par element is the outermost synchronization element of the whole document); specifying its end as 20s truncates the clip's duration to fifteen seconds. Figure 13.3 illustrates the synchronization of this presentation on a timeline. The natural duration of the video clip, which is used in the absence of any explicit attributes, is 10 seconds.

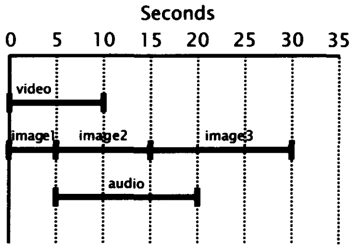


Figure 13.3
Effect of SMIL synchronization elements

⇨ The SMIL 1.0 specification includes rules to ensure that synchronization attributes can only be used sensibly. For example, an element's end value must not precede its begin.

The timing of the sequential elements in this presentation was specified purely by their duration, so they butt together in time. If we use begin and end, we can create pauses between them. For example, if we change the second img element as follows:

```
<img src="images/image2.jpeg"
     type="image/jpeg"
     region="upper"
     begin="5s" end="20s"/>
```

there will be a five second gap between the display of image1 and image2 — because this is a seq element, the start time is not evaluated relative to the start of the synchronization element, but to the end of the preceding element. Figure 13.4 shows the effect of this change on the presentation's timeline.

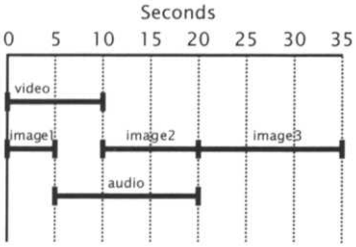


Figure 13.4
Introducing a delay in a seq element

⇨ The begin and end attributes can be thought of as specifying the position of SMIL elements on the presentation's timeline. Media elements such as video and audio clips or streamed text have their own internal timeline; the attributes clip-begin and clip-end can be used to specify offsets within an internal timeline, in order to extract a section from within a clip. For example, if the movie movies/fullmovie.mov was one minute long, the element

```
<video src="movies/fullmovie.mov"
       clip-begin="20s"
       clip-end="40s">
```

would define a twenty second clip from the middle of it. The use of these attributes removes the need to create a separate movie for the extract.

Elements within a par element can be synchronized directly with reference to each other. For this to be possible, each media element must have an id attribute with a unique name as its value, which identifies the element in the same way as id attributes identify HTML document elements. A time value can then be specified in the form id(*element-id*)(*time-value*), where the *time-value* can either be a clock value of the form just described, or one of the special forms begin or end. These two forms denote the start and end times of the element identified by *element-id*, respectively; the more general form denotes a time *time-value* later than the start of that element. For example, to delay the onset of the sound track of our example presentation until after the video had finished playing, we could make the following modifications:

```
<video src="movies/m1.mov"
       id="vid"
       type="video/quicktime"/>
<audio src="sounds/sound1"
       type="audio/aiff"
       begin="id(vid)(end)"/>
```

This way of specifying times, using what the SMIL specification calls an *event value*, is often more convenient than specifying them relative to the start of the enclosing synchronizing element; if the duration of some of the media elements is not known, it is essential. An event value can only be used by an element within the same synchronization element as the one it identifies. In the above example, since the video and audio elements are both in the content of the same par, the use of the event value id(vid)(end) as the beginning of the audio is legal. However, we could not use an event value id(img2)(2s) to specify that the audio should start two seconds after the display of the second image begins (assuming the second image element has id img2), because the image is inside the enclosed seq element. This restriction somewhat reduces the usefulness of event values.

When a par element contains elements that do not all finish at the same time, the question arises of when the whole par ends. Looking back at Figure 13.3, you will see that there are at least two reasonable candidates for the end time of the combination of video, images, and sound: at ten seconds — the earliest point at which one of the elements finishes, and at thirty seconds — the latest. Probably, the latter seems the more sensible interpretation, and that is what SMIL takes as the default: the end time of a par element is equal to the latest end time of any of its children (the elements it contains). The endsync attribute may be used to change this behaviour. Its value may be `last`, which is the same as the default; `first`, which makes the end time of the `par` be equal to the earliest end time of any of its children; or an expression of the form `id(`*element-id*`)`, which makes the end of the `par` equal to the end of its child with `id` equal to *element-id*. For example, to make our presentation finish as soon as the video ended, we would change the start tag of the `par` element to:

```
<par endsync="first">
```

or

```
<par endsync="id(vid)">
```

If we use any `endsync` value other than `first`, what should happen to elements that terminate before the end of the `par`? Looking at Figure 13.3 again, you will see that there is a twenty second gap between the end of the video and the end of the whole presentation. What should happen to the video during this gap? Should it disappear from its allotted region, or should it be held frozen on its final frame? SMIL lets us choose between these two alternatives using the `fill` attribute, which can take either of the values `remove` or `freeze` (the default). To make our video disappear after it has reached its end, we would use the following tag:

```
<video src="movies/m1.mov"
       fill="remove"
       type="video/quicktime"/>
```

An alternative you might prefer to either of these options is to have the video start playing again from the beginning. Generally, you can arrange for any synchronization element to be repeated a specific number of times using the `repeat` attribute, whose value is a number, indicating the number of times the element is to play.[11] Since our video is ten seconds long, we can fill the entire presentation using three iterations of the video. To do this, we must put the `video` element inside its own `seq`, because repeat counts cannot be specified for media elements.

```
<seq repeat="3">
  <video src="movies/m1.mov"
         type="video/quicktime"/>
</seq>
```

The value of `repeat` can also be `indefinite`, which as you would expect, means that the element loops indefinitely — not, though,

[11] The semantics of English suggests that it should be one less, but in SMIL each iteration, including the first, is a repetition.

forever. If an element has an indefinite repeat count, it is ignored in the calculation of the duration of its enclosing synchronization element.[12] The effect of this is to ensure that it runs for as long as it needs to in order to 'fill up' an enclosing `par`. If, in our example, we had enclosed the `video` element in a `seq` with the following start tag:

```
<seq repeat="indefinite">
```

it would play until the last image had been displayed for its full duration, no matter how long the video clip was.

At first sight, although they may be awkward to work with, SMIL's synchronization elements and time-related attributes appear to be sufficient to allow a multimedia author to specify precisely any possible temporal relationship between the individual media elements of a presentation. It doesn't require much imagination to see how SMIL, or its equivalent, could be generated from a timeline of the sort we described earlier, so it would appear that implementing synchronized multimedia presentations is fairly trivial. In reality, things are not quite so rosy. SMIL's synchronization mechanism depends on the assumption that we can unambiguously measure the time taken to play any media element. For time-based media such as audio and video this is not the case. There are two distinct ways of measuring the time that elapses during playback. Consider a video clip that is intended to play at a rate of 15 frames per second. Assuming its `id` attribute's value is `clip1`, what time does the event value `id(clip1)(3s)` specify? Is it the time at which playback of the clip reaches its 46th frame, or is it three seconds after the clip starts to play, as measured by the clock of the computer system on which it is playing? There is no guarantee that the two will be the same, because there can be no guarantee that the clip will really play at the correct frame rate: the processor or disk may not be fast enough, or, if the clip is being streamed over a network, transmission delays may occur, causing some frames to be delivered too late. Hence, we must distinguish beween *media time*, the time measured by counting how much of the clip has been played, and *presentation time*, the elapsed time measured with reference to an external clock. SMIL clock values could be interpreted as either, producing different results. If they are interpreted as media time, then media elements wait for each other if a delay occurs; if they are interpreted as presentation time, synchronization may be lost, but the total running time of

[12] Strictly, it is treated as if its end time were immediately after its start time.

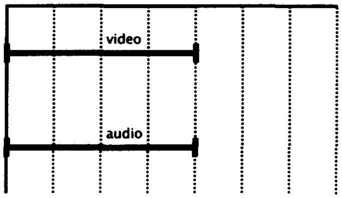


Figure 13.5
Video and audio perfectly synchronized

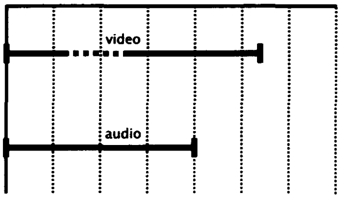


Figure 13.6
Soft synchronization

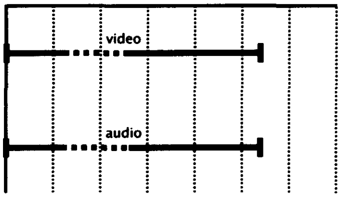


Figure 13.7
Hard sync: audio delay introduced

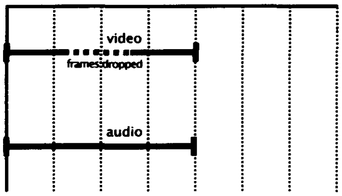


Figure 13.8
Hard sync: video frames dropped

a presentation will be maintained. The SMIL specification permits either interpretation, but recommends media time.

The same effect shows up in a different guise when we consider the moment-to-moment synchronization of elements within a `par`. If, for example, an audio clip and a video clip are playing simultaneously, we would expect them both to play at the same rate; in other words, media time in the two should pass at the same speed. If this is not the case, then the audio and video could slip relative to each other, leading to undesirable effects, such as loss of lip-sync, or the displacement of cuts in the video relative to the tempo of music on an accompanying sound track. On real computers and over real networks, independent delays can occur to either audio or video. Again, there is a choice between two possible responses, and a SMIL player is permitted to use either: it can use *hard synchronization*, with the elements within a `par` being synchronized to a common clock (thereby effectively matching their presentation times); or *soft synchronization*, with each having its own clock. Soft synchronization is really no synchronization, but it ensures that each element is played in its entirety. Hard synchronization maintains the temporal relationships between the elements (as nearly as is possible in the presence of delays), but distorts playback. Suppose, for example, that we have one video and one audio clip of identical durations, playing in parallel (see Figure 13.5), and that the video element experiences a delay of some sort during playback. If soft synchronization is used, the audio will carry on playing, and when the video resumes it will be out of sync (see Figure 13.6, where the dotted portion of the video line indicates a delay). To preserve hard synchronization, one of two things must be done. Either the audio must also be stopped until the delayed video data arrives (Figure 13.7), or some video frames must be dropped until the picture catches up with the sound again (Figure 13.8). The latter option is usually less intrusive, but, in the worst case, the picture may never catch up to the sound, so neither is very satisfactory.

⇨ The synchronization of separate tracks in a QuickTime movie, which we sketched in Chapter 12, is equivalent to hard synchronization, with frame dropping. Note that the use of the QuickTime time base is essential to detecting and correcting for loss of synchronization.

The problems just outlined are not a specific shortcoming of SMIL but are inevitable where synchronization is desired, but unpredictable delays can occur. Ideally, one would like to ensure that delays cannot occur, or are kept so short that they do not interfere with playback. This can never be guaranteed, but we can assist by selecting versions of video and audio clips whose data rate is matched to the capabilities of the user's computer and network connection (the latter usually being the limiting factor). SMIL's switch element may be used for this purpose.

A switch may contain any number of other elements, each of which has one or more *test attributes*, which behave in a somewhat different way from other attributes. Their value is tested against some system parameter; if the test succeeds, the element may be selected and rendered. Only one of the elements within a switch is selected; they are considered in the order they appear in the document, and the first one whose test succeeds is chosen. The test attribute that is most relevant to our present discussion is system-bitrate, whose value is a number that is interpreted as a bandwidth in bits per second, and tested against the system bandwidth. This latter value may be simply taken from a user preference, for example specifying a modem connection rate, or it may be calculated from actual measurements of the system's performance. The mode of calculation is implementation-dependent. A test of this attribute succeeds if the available system bandwidth is greater than or equal to its value. Hence, a sequence of media elements can be provided in decreasing order of bandwidth requirements, with corresponding system-bitrate values, within a switch, so that a version can be chosen that is within the capabilities of the system. For example, we could prepare several different versions of a video clip, at different frame sizes and compression quality settings, and choose one of them as follows:

```
<switch>
    <video src="movies/mjepg-full-size-movie.mov"
           system-bitrate="56000"
           type="video/quicktime"/>
    <video src="movies/cinepak-full-size-movie.mov"
           system-bitrate="28800"
           type="video/quicktime"/>
    <video src="movies/cinepak-quarter-size-movie.mov"
           system-bitrate="14400"
           type="video/quicktime"/>
    <img src="images/movie-still.jpeg"
         system-bitrate="0"
         type="image/jpeg"/>
</switch>
```

Note the final element in the switch: this is a default option (presumably the test will always succeed); a still image is selected, which does not place any strain on the capabilities of a low bandwidth system.

⇨ Other test attributes include system-language, which is used to select between different languages for text and speech, based on the user's preferences; system-captions, which can be set to the value on to display subtitling captions, for example, to supply a transcript of an audio sound track for the benefit of deaf people; system-screen-size and system-screen-depth, which are used to choose between different versions of images designed for particular screen sizes or colour depths.

We remarked earlier that the distinction between page-based and synchronization-based multimedia is not an absolute one: each style may possess characteristics of the other. As a prime example, SMIL provides simple uni-directional links, using linking elements modelled on HTML's anchors; however, the extra dimension of time that is central to SMIL calls for some refinement of the notion of linkage.

There are two linking element types in SMIL, the simpler being a, which behaves as a link source, much as HTML's a element type can. (Unlike HTML, SMIL does not use named a elements as link destinations, it makes use of id attributes instead.) The href attribute holds the URL of the link's destination, and clicking on the a element's content causes a jump to the destination, which will usually be another SMIL presentation. Formally, a elements have no synchronization properties and do not affect the temporal behaviour of any elements they contain. Since the content of an a is all that is actually displayed, and since the elements in the content will, in general, have a specified duration, the a element will behave as if it is only there while its content is being displayed. Consider the following simple presentation, for example.

```
<seq>
  <a href="presentation1.smil"
    <img src="images/image1.jpeg"
         type="image/jpeg"
         dur="15s"/>
  </a>
  <a href="presentation2.smil"
    <img src="images/image2.jpeg"
         type="image/jpeg"
```

```
          dur="15s"/>
  </a>
  <a href="presentation3.smil"
    <img src="images/image3.jpeg"
          type="image/jpeg"
          dur="15s"/>
  </a>
</seq>
```

Three images are displayed in sequence. Each of the images is contained in an a link, whose href points to a presentation — presumably one which expands on the content of the corresponding image. During the display of an image, clicking on it will activate the enclosing anchor, causing the linked presentation to begin playing. By default, the new presentation replaces the old one; this behaviour can be modified using the show attribute, which has one of the values replace (the default behaviour), new, which causes the new presentation to start up in a new window, leaving the old one playing in its original window, and pause, which causes the new presentation to start up in its own window, but leaves the old presentation paused at its current point, from which it resumes after the new presentation has finished. (Note how it is the temporal dimension of presentations that requires separate replace and new options.)

⇨ The SMIL 1.0 specification is actually rather vague about the duration of a elements, stating only that, " For synchronization purposes, the a element is transparent, i.e. it does not influence the synchronization of its child elements." The interpretation just given seems to be the only logical one, and corresponds to how the SMIL players available to us actually behave.

Like HTML, SMIL supports the use of fragment identifiers in URLs, so that links can point to individual elements (identified by the value of their id attribute) within a presentation. Again, the temporal dimension complicates matters. Consider a link such as the following:

```
  <a href="presentation2.smil#vid1"
    <img src="images/image2.jpeg"
          type="image/jpeg"
          dur="15s"/>
  </a>
```

and suppose that presentation2.smil has the following body:

```
<seq>
  <video src="movies/m1.mov"
         id="vid0"
         type="video/quicktime"/>
  <par>
    <seq>
      <img src="images/image1.jpeg"
           type="image/jpeg"
           id="img1"
           region="upper"
           dur="5s"/>
      <video src="movies/m1.mov"
             id="vid1"
             type="video/quicktime"/>
      <video src="movies/m1.mov"
             id="vid2"
             type="video/quicktime"/>
    </seq>
    <audio src="sounds/sound1"
           type="audio/aiff"
           begin="5s"/>
  </par>
</seq>
```

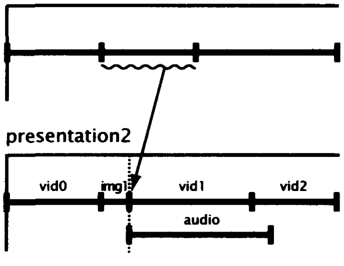


Figure 13.9
Effect of a fragment identifier

(The lower part of Figure 13.9 shows its timeline.) When the link is activated (at any time during the display of the image it contains, as shown in the diagram), a jump is made to the element with `id` equal to `vid1`. This element is not displayed until five seconds into the `par` element that follows the first video clip (`vid0`), so the effect of the jump must be to start `presentation2.smil` at that point, skipping the initial video clip and image. In general, when the destination URL of a link has a fragment identifier, the effect of following the link is to start the presentation containing the element it identifies, but as if the user had fast-forwarded to the point at which that element begins. Note that, where the element in question is within a `par`, all of the elements that are specified to play at the same time will be displayed. Hence, in the example just given, when the link is followed the sound will play as well as the video clip `vid1`.

⇨ If the destination of a link is an element with a `repeat` attribute, the presentation is started before the first iteration of the element. If it is contained within an element with a `repeat` attribute (which might itself be inside an element with a `repeat` attribute, and so on) matters become slightly complicated. The SMIL 1.0 specification describes how this case should be handled to ensure that the presentation behaves as expected.

SMIL's a elements, in conjunction with fragment identifiers, provide links between entire elements; in other words, they connect identifiable points on the timelines of presentations. SMIL allows a more refined sort of link, based on identifying regions within the individual timelines of media elements. It provides the anchor element type for this purpose. These elements are the only ones that can appear in the content of a media element. (Previously we have always supposed that media elements were empty.)

An anchor is like an HTML A element, in that it may serve as either the source or the destination of a hyperlink, or both. The element type is furnished with href and id attributes for these purposes. It may also have a coords attribute, whose value is a set of four length values (with the same syntactical possibilities as in CSS), separated by commas. These define a rectangular area: the first two numbers are the upper left, the last two the lower right corner. An anchor with coords and an href thus defines a hot spot within the region in which the media element that contains it is displayed.[13] Several such anchors behave in the same way as an image map in HTML, although they can be associated with other media besides images.

An image map divides up the spatial region occupied by an object, associating links with different sub-regions. Since SMIL elements have a temporal dimension, it is also possible to divide up the time occupied by an object, and associate links with different intervals. The begin and end attributes of the anchor element type are used for this purpose. Suppose, for example, that the file trailers.mov contains a thirty second movie, consisting of three ten second segments, each of which is a trailer for some other movie (say an original Hollywood blockbuster and its two sequels). The following SMIL code will cause the trailer movie to play indefinitely; during each segment, if a user clicks on it, the corresponding full movie will be streamed from a video server.[14] Notice that, for the first time, the video element has separate start and end tags, because it is not empty.

```
<video src="trailers.mov"
       type="video/quicktime"
       repeat="indefinite">
  <anchor href="rtsp://movies.com/blockbuster.mov"
          id="trailer1"
          begin="0s" end="10s"/>
  <anchor href="rtsp://movies.com/first_sequel.mov"
          id="trailer2"
```

[13] The four values are subject to constraints that ensure that the rectangle's top is above its bottom, and so on. The rectangle is clipped to the media element's region.

[14] See Chapter 15 for an explanation of rtsp://

```
        begin="10s" end="20s"/>
  <anchor href="rtsp://movies.com/second_sequel.mov"
        id="trailer3"
        begin="20s" end="30s"/>
</video>
```

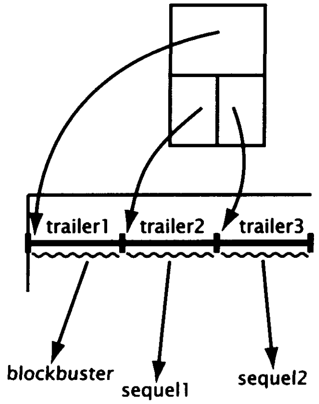


Figure 13.10
Anchors as link sources and destinations

The `id` attributes of `anchor` elements can be used as fragment identifiers in URLs, so that segments of a movie can be used as link destinations. For example, if the previous example appears in a file called `trailers.smil`, a presentation in the same directory could include an image map built out of anchors, such as the following:

```
<img src="poster.jpeg"
    type="image/jpeg">
  <anchor href="trailers.smil#trailer1"
        coords="0, 0, 100%, 50%"/>
  <anchor href="trailers.smil#trailer2"
        coords="0, 50%, 50%, 100%"/>
  <anchor href="trailers.smil#trailer3"
        coords="50%, 50%, 100%, 100%"/>
</img>
```

Assuming that the top half of the image holds a still from the original blockbuster, while the bottom half is divided into two, showing stills from the two sequels, clicking on a still will cause the trailers video to start playing at the beginning of the trailer for the appropriate movie (see Figure 13.10).

HTML+Time

SMIL provides a mechanism for producing synchronized presentations that can be played over the Internet, but it is largely separate from the World Wide Web. SMIL presentations can be embedded in Web pages and played with the aid of a suitable plug-in, much as video clips can, and HTML can be embedded in SMIL using `text` elements in a similar way, but the two languages cannot be truly integrated. For example, paragraphs within an HTML document cannot be arranged in time so that they are displayed in sequence.[15] *HTML+Time* (the 'Time' part stands for 'Timed Interactive Multimedia Extensions') is a proposal for applying the synchronization facilities of SMIL directly to HTML document elements. HTML+Time has not presently been endorsed by the W^{3}C, but it does have the influential backing of Microsoft and other major companies.

[15] See Chapter 14 for a description of a mechanism whereby such effects can be achieved by another means, albeit with more effort.

HTML+Time's basic idea is a simple one: additional attributes are introduced for every document element that can appear in the body of an HTML document. These attributes include `begin`, `end`, and `dur`, which take clock values in the same style as SMIL. Their effect on most document elements is to restrict their visibility to the specified time. When applied to those HTML elements that control style, such as `EM`, or to an element that uses the `style` attribute for in-line stylesheet information, the timing attributes control the duration of the indicated style. SMIL's event values cannot be used for `begin` and `end`. Instead, HTML+Time provides an attribute `beginWith`, which has as its value the `id` of some other element; clock values used for other attributes of the same element are interpreted relative to the start of the element with that id. For example,

```
<P id="start" begin="5s">
 content of start paragraph
 </P>
<P beginWith="start" begin="10s">
 text that appears 10 seconds
   after the start paragraph
 </P>
```

To produce the effect of a SMIL `par` element, an attribute `par` can be used with `DIV` or `SPAN`. Attributes corresponding to those that SMIL provides for its `par` element type can be used in conjunction with the `par` attribute to achieve similar effects. Inconsistently, HTML+Time does not offer a `seq` attribute, but instead adds SMIL's `seq` *element type* to HTML.

Since HTML+Time is only a proposal that has been submitted to the W^{3}C for comment, it would be unfair to judge it by the same criteria that are applied to mature W^{3}C Recommendations. The idea of adding synchronization facilities to HTML is potentially a productive one. Certain time-based effects, such as slide shows, which can only presently be achieved by stepping outside the declarative framework of HTML, may be produced relatively easily within the extended framework of HTML+Time. However, extending HTML goes against current trends in the World Wide Web. Stylistic properties have been separated from content, using stylesheets; tying temporal properties to content seems like a backward step. A 'temporal stylesheet' approach would seem to be more in keeping with the spirit of structural markup, and would have the additional advantage that the temporal specification mechanism would not necessarily be tied to HTML alone, but could be applied to XML and other markup languages based on it.

⇨ The HTML+Time proposal addresses another of SMIL's shortcomings in passing only, without offering any improvement: the failure to provide any form of transitions, such as dissolves or page-turns, between elements of a presentation. These effects are standard fare in the mainstream, timeline-based, presentation tools. Without them, SMIL presentations have an abruptness that can be disconcerting, and usually makes them look distinctly amateurish.

Accessibility

Images, movies and sound may provide vital content in a multimedia production. Even if plain text can convey all the essential content, adding these other media can add interest, transmit the information more effectively, and make navigation more intuitive, but only if the images and video can be seen and the sounds can be heard. Not everybody can see and hear; not everybody who can see and hear can do so easily, or always understand what they see or hear. One of the great benefits that computers can bring to the world is in helping people who are blind, partially sighted, or deaf, or have cognitive difficulties or other physical disabilities. Special hardware and software can render text as speech or braille, or provide magnification; speech can be transcribed into text; information can be presented at a rate adapted to the user's abilities. All of this can be derailed by thoughtless use of multimedia elements. The W^{3}C has made strenuous efforts to ensure that, wherever possible, their recommendations include facilities for making information on the Web accessible to everybody with an Internet connection, irrespective of disability.

The use of images as links is a common example of an idiom that can frustrate users with poor eyesight if it used thoughtlessly. Unless some alternative is provided, any semantic content attached to the icons is lost to people who cannot see them. This does not mean, though, that all Web pages should revert to pure text in the interests of accessibility. HTML provides facilities that allows you to use multimedia elements, but still cater for people who, for some reason, are unable to perceive them. As the W^{3}C's accessibility guidelines for page authors put it:

> "Accessibility does not mean minimal page design, it means thoughtful page design. ...In general, authors should not be discouraged from using multimedia, but rather should use it in a manner which ensures that the material they publish is accessible to the widest possible audience."

⇨ Although questions of accessibility are particularly compelling in connection with disabilities, it is worth remembering that people might access Web sites using slow modems, so that the time taken to download images is not acceptable, or via hand-held mobile devices with screens too small to display images. Many PCs used in business do not sport sound cards. There are countless reasons why images and sounds might not be perceived, and it is only sensible to design your multimedia productions so that, as far as possible, they can be used — and make sense — without them.

Making your work accessible to the hard of hearing is not difficult, provided you make some effort: transcripts of speech and song lyrics can be provided;[16] alert sounds can be supplemented by visual cues, such as flashing icons; if you use speech to interact with users (as in some games involving dialogue) there should be an option to replace it with captions or speech bubbles.

The heavy use of graphically-based elements in multimedia means that almost every multimedia producer should be making some effort to compensate for visual problems. This means providing some alternative, usually in a textual form that can be rendered by a non-visual user agent, for every graphic element. This alternative must perform the same function as the graphic, which is not the same as simply describing it. For example, a navigational button requires a textual link, containing whatever clue the button's icon gave to the destination of the link, so a button labelled with the conventional icon of a house, denoting a link to a home page should not be augmented with some text saying 'little house on a red background', but a link labelled 'home'.

Not all authoring tools and formats provide much help, but HTML 4 has features especially designed to assist in making Web pages accessible to everybody. Chief among these is the `alt` attribute of IMG and AREA elements. Its value is a text string that serves as a substitute for the element if it cannot be rendered normally. So if the HTML is being rendered by a non-visual user agent, such as a speech synthesizer, or by a Web browser with images disabled, the `alt` value will be available in place of the image; in the case

[16] QuickTime 3 and higher allows captions and video descriptions to be added to the clip itself.

of AREA elements, the alt value provides a textual link instead of the corresponding hot spot in the image map. The HTML 4.0 specification stipulates that alt must always be supplied with IMG and AREA elements; of course, Web browsers do little or no checking of the legality of HTML, but HTML verifiers should insist on the presence of these attributes.[17] Since alt text will be rendered in place, it should be relatively short. If a longer description of an image is required, the longdesc attribute can be set to its URL. In that case, the alt value should ideally provide enough information for a user to decide whether to pursue the link and read the long description. For example, the image map example of the previous section could have been improved like this:

```
<IMG src="flower1.jpeg" usemap="#image-map-1"
          alt="Flower image map"
          longdesc="flowermap.html">
```

where the file flowermap.html provided a text-based set of links to the same destinations as those offered by the image map.

Several other attributes introduced in HTML 4.0 can be used to provide information that enables a user agent to help people whose disabilities make 'normal' browsing difficult. Consult the specification and the Web Accessibility Initiative's guidelines[18] for details. Unfortunately, Web browser vendors seem less interested in fully implementing these features than in adding marginally useful user interface gimmicks, so it is advisable to provide redundant textual links and other information to ensure that the graphical content of your Web pages does not make them inaccessible.

We alluded in Chapter 8 to one other way of enhancing accessibility: use structural markup to separate content from appearance. If a page is consistently tagged to reflect the semantics of its content, software can use the information in the tags to assist with presenting it in different media. More helpfully still, if appearance is controlled by a stylesheet, alternative stylesheet information can be provided for non-visual user agents. CSS2 proposes several properties to control aural presentation, such as volume and voice characteristics including stress, pitch and speech rate, and the use of 'aural icons' as cues. Consult the CSS proposal for more details.

The techniques we have just described can be complemented by use of the title attribute (not to be confused with the TITLE element type), which is an attribute of almost all elements in HTML 4.0. Its function is to provide a short description of the element. Its value is

[17] This does not prevent graphics and Web authoring tools from routinely failing to generate them.

[18] See the *Further Information* at the end of the chapter.

not rendered as part of the page's content, but it may be displayed by a user agent in the style of a 'tool tip' or 'help balloon' when the cursor is over the element. Since titles can be spoken by non-visual user agents, this information can be made available to all users in some form. This can be helpful, since it enables a user to scan all the elements of a page and quickly get an idea of their function within the document. In particular, the title of a link may be used to indicate what the link is for.

SMIL shares some of the features in HTML that are intended to improve accessibility: all media elements support `alt` and `longdesc` attributes, linking elements have a `title`, while `par` and `seq` have an `abstract` attribute, which can be used to provide a textual description of their contents. However, a shortcoming of SMIL in this respect is that it offers no mechanism for providing users with control over the playback speed of presentations. Users with restricted movement or cognitive difficulties may not be able to respond quickly enough to anchors with limited duration, for example. An option to slow down a presentation would assist such users;[19] it is not clear how easy it would be to incorporate variable playback speeds into SMIL's model of time and synchronization.

[19] Some SMIL players provide a means of pausing playback, but that is not what is required.

Further Information

The HTML references given at the end of Chapter 8 describe the use of HTML for hypermedia. [SMI] and [HTMb] are the definitions of SMIL and HTML+Time, respectively. [WAI] gives the Web Accessibility Initiative's guidelines for authors. [LH99] looks at the issues surrounding hypermedia development, which we have not considered. The use of scripting with combined media is the subject of the next chapter.

Exercises

The best way to understand combined media is to implement some multimedia productions. We suggest that you look at the chapter entitled *Multimedia Practice* for some suggestions. The following short exercises are merely concerned with some details of the languages described in the present chapter.

1. An AREA element may have an attribute nohref, which, if true, specifies that the designated area has no URL associated with it. Since no part of the image is associated with a URL unless it is within the shape defined by some AREA, what possible use could you make of this attribute?

2. Explain how you would use SMIL's synchronization elements to make a simple looping animation from a sequence of still images. How would you add a synchronized sound track?

3. In the absence of any explicit coords, begin and end attributes, a SMIL anchor element is associated with the entire media element it is contained in. Does this make SMIL's a element type redundant?

4. What difficulties would somebody who was not able to use a mouse experience in trying to access the World Wide Web? What attributes introduced in HTML 4.0 can be used to help such people, and how?[20]

[20] You will have to read the HTML 4.0 specification or a good HTML book.

14 Events, Scripts and Interactivity

The graphical user interface provided by contemporary operating systems is a paradigm of an *event-driven* system: the user does something (double-clicks an icon, for example), and something happens in response (an application program is launched). Hypermedia navigation, as we described it in Chapter 9, can be seen as a special case of this general model of interaction: clicking on some highlighted text, an image, or a hot spot in an image map, causes the destination of a link to be displayed. While this form of interaction can take you a long way, incorporating more general event-driven interaction into a multimedia production opens up additional possibilities.

The idea is to associate *actions* with *events*. We first need to identify a set of events. Most of these are initiated by the user — mouse clicks and movements, key presses, and so on — but some events are generated internally, for example, when a movie finishes playing, or simply after a specific period of time has elapsed.[1] The set of available events is not fixed, but depends on the particular combination of hardware and software being used. As well as identifying the set of events, we need to be able to specify actions. Some multimedia environments provide a pre-defined set of actions

[1] There is often a *null* or *idle* event that is generated when nothing else happens.

(often called *behaviours*) to perform common tasks, such as opening a file, playing a movie or replacing one image with another. More sophisticated environments let you define your own, by providing a *scripting language*: a small programming language with facilities for controlling user interface elements and multimedia objects. You write a script that performs the desired action when it is executed; you then associate that script with an event, using a tag or some command in an authoring environment, so that it gets executed when the user clicks on a particular image, for example.

> ➲ Here, we have assumed that actions can be associated with events, either by an authoring system, or using some tagging mechanism, and that they are executed under the control of some other program, such as a Web browser or a multimedia player. Such a program takes the form of a loop that is repeatedly executed. Every time round the loop, a check is made to see whether any event has occurred. The precise details of how this information is passed to the program depend on the operating system, but commonly, when an event occurs it causes an interrupt. The operating system then determines what event has occurred and assembles a data structure describing it — for example, if a mouse click occurs, this structure might contain the mouse coordinates and the state of the buttons and any relevant modifier keys — which it places in a queue before resuming the interrupted task. The event loop uses a system call to obtain the first event from the queue, and then takes a branch according to the type of event. Thus, associating an event with an action causes the branch corresponding to that event to execute the code you have written to perform the action. Sometimes, perhaps for efficiency reasons, it is not adequate to use the event loop and scripting language provided by a package, and you must write your own. The Java programming language provides for a flexible model of event handling that is used by the *Java Media Framework* to enable you to write programs that interact with multimedia objects.

One reason for associating actions with events is to provide interactivity: if the system can respond to events generated by the user, then the user can control the system's behaviour.[2] In particular, users can direct the flow of information that they receive. Events that occur at specific times provide another reason for using associated actions: they allow time-based behaviour and synchronization to be introduced into systems, such as the World Wide Web, that do not support it directly as part of their model of multimedia. Contrariwise, actions embedded in the timeline of a synchronized multimedia production can be used to provide non-

[2] Within limits — see the discussion in Chapter 1.

linearity. It is only a slight over-simplification to say that scripting can add the characteristic features of hypermedia to time-based presentations; it can add a temporal dimension to hypermedia; and it can add interactivity to both, so that multimedia productions that make use of scripts can provide the same experience to every user, no matter which model of media combination they start from.

Scripting Fundamentals

One definition of a *scripting language*, taken from the ECMAScript specification [ECM97], is

> "... a programming language that is used to manipulate, customize, and automate the facilities of an existing system."

The authors of the specification go on:

> "In such systems, useful functionality is already available through a user interface, and the scripting language is a mechanism for exposing that functionality to program control. In this way, the existing system is said to provide a *host environment* of objects and facilities which completes the capabilities of the scripting language."[3]

Scripting languages can be distinguished from mainstream programming languages such as C++ or Java, which provide some control structures, abstraction mechanisms, and built-in data types, such as numbers and pointers or references; the abstraction mechanisms of a modern programming language allow you to define your own data types, constructed out of the basic types provided by the language. A scripting language provides some control structures and a few basic types too — and may provide some abstraction mechanisms — but it also provides objects and data types that belong to some 'existing system'. For example, a scripting language for use with a relational database system will provide data types corresponding to relations and tuples; one for use as an operating system command language will provide data types corresponding to processes and files. (Note that here we are using the expression

[3] Our italics.

'data types' in the modern sense of *abstract data types*: a set of values together with a set of operations.)

Our concern is with multimedia, so the scripting languages we are interested in must provide objects corresponding to the elements of a multimedia production. It is not adequate to provide a type for each medium — text, sound, video, and so on — the types of objects available need to take account of the way in which media objects are combined within the production. (Recall from the passage just cited that a scripting language provides program control equivalent to the user interface.) Thus, in a scripting language for use with XML or HTML, objects must be provided that correspond to document elements and to elements of the user interface, such as windows. In a language whose host environment is a time-line based system such as Flash, we need objects corresponding to frames and the various elements that may appear inside them.[4] The scripting language will allow us to perform computations on the attributes of these objects, or create new ones, thereby affecting their appearance and behaviour. This computation will be triggered by events. Some of these events will be initiated by the user, so scripted actions provide a means of allowing user input to affect the flow of control, or putting it another way, they allow certain elements of a multimedia production to function as controls providing interactivity . For example, an action may be attached to an image, with the effect that clicking on the image causes a particular movie to start playing in another window.

World Wide Web Client-Side Scripting

The World Wide Web offers two potential host environments for a scripting language — servers and browsers (clients) — giving rise to two types of Web scripting: *server-side* and *client-side*. Server-side scripting is used to enable an HTTP server to communicate with other resources, such as databases, and incorporate data obtained from them into its responses. In particular, server side scripts are used to enable a server to construct Web pages dynamically from time-varying data. We will discuss server-side scripting in Chapter 15. In this chapter, we will confine ourselves to client-side scripting, which is used to control the display of media elements that have been retrieved from a server. World Wide Web scripting on the client-side is a comparatively recent development, which has helped transform the appearance and behaviour of Web pages.

[4] Although we talk of the host system providing a set of 'objects', a scripting language does not need to be explicitly object-oriented, although most are.

Executing a client-side script means allowing executable code that has been downloaded from the Internet to run on your machine. Since you have no real guarantee that the person whose code you have downloaded is not malicious, deranged, or in the pay of a market research organization, this would be a foolhardy thing to do if scripts were able to perform arbitrary computation. So they aren't. In particular, scripts running in a Web browser cannot access any local resources, such as files on your hard disks, or make any network connections, and their interaction with the server from which they originated is limited to requesting new resources and posting information from HTML forms.[5] As a result of these restrictions, client-side scripts are (reasonably) secure, but limited in the useful work they can do. In effect, they cannot do more than modify the browser's display in response to events. Generally, client-side scripts are used to provide feedback to users (for example, by changing the colour of a clickable item when the cursor is over it) or, more often, just to add interest to a site by providing a richer user interface than the simple one consisting of loading a new page when a link is clicked.[6]

The first scripting language for the World Wide Web was a proprietary product called LiveScript, which was embedded in Netscape's Navigator Web browser; its name was changed to JavaScript, in what is generally understood to be a marketing-inspired move to capitalize on the interest generated by Java — a largely unrelated programming language. Microsoft rapidly produced an almost compatible scripting language, disingenuously called JScript, for their Internet Explorer browser. In an attempt to prevent a Web scripting schism, the European Computer Manufacturers' Association (ECMA) was asked to produce a scripting language standard, based on JavaScript and JScript. ECMA did something smarter: they defined a 'core scripting language', called ECMAScript, which could be adapted to a variety of host environments — a Web browser being just one of them.

In order for ECMAScript to to be able to interact with a Web browser, there must be a well-defined interface between the two. Since ECMAScript is an object-based language, this can be provided in the form of a set of objects representing the elements of an HTML (or, in time, XML) document and any styles applied to them, and the components of the browser interface. The W^{3}C's *Document Object Model (DOM)* is intended to provide this interface. Unfortunately, at the time of writing, only part of the DOM has been defined, and the

[5] To prevent abuses of this facility, there are additional restrictions on the elements of the browser state which a script can read.

[6] Where forms are embedded in Web pages, client-side scripts are often used to validate input to fields, without the need to contact the server and run a CGI script.

two major browsers use document models of their own, which are not entirely compatible. Nevertheless, it is possible to identify most of what an object model must look like, so that we can describe client-side Web scripting. You should be aware, though, that in practice a lot of effort and ingenuity is required to ensure that scripts will work as intended on every browser that supports client-side scripting.

> ⇨ Both Netscape and Microsoft claim to have brought their scripting languages into line with ECMAScript. In practice, therefore, 'JavaScript' ought to mean ECMAScript with Netscape's version of the DOM, and 'JScript' ECMAScript with Microsoft's, but 'JavaScript' is often used loosely to mean either. To complete the confusion, the expression *dynamic HTML*, which seems to have been devised by Microsoft to refer to the use of their object model and JScript or VBasic for client-side Web scripting, has been adopted by Netscape to refer to client-side scripting using the aspects of *their* object model that are concerned with style sheets and absolute positioning,[7] but is often used loosely to refer to anything to do with client-side scripting, style sheets, or both.

Although ECMAScript is a simple language, compared with C++ for example, it would still require more space than we have available in this book to provide a complete description of all its features and the way in which it can be used. Instead of attempting to do that, we will use a set of examples to try to provide an impression of what can be achieved and how, using client-side scripting to automate multimedia elements of Web pages.[8] In the interests of simplicity we will only introduce the minimum of syntax, so much of our coding will be less elegant than it could be — ECMAScript allows some sophisticated programming techniques to be employed by those who know how. Since the W^{3}C DOM remains incomplete and has not been implemented in any browser at the time of writing, we will present scripts using the set of host objects that works with existing fourth generation Web browsers, with a preference for Internet Explorer 4 where there are differences, since its object model is closer to the emerging DOM. The main difference between the current model and the Level 1 W^{3}C DOM is that, since the latter is designed to support XML as well as HTML, its object names are prefaced with `HTML`. Thus, where we refer to `document`, the standard, when complete, will refer to `HTMLdocument`, and so on. We expect that the old names will continue to work for some time, in the interests of backwards compatibility.

[7] Which Netscape have helpfully implemented using their own <LAYER> tag instead of (latterly, as well as) the appropriate CSS2 properties.

[8] Much of this will be concerned with manipulating image elements, so you should remember that whenever such techniques are employed, it is necessary to provide alternatives for non-visual user agents.

In the following sections we assume some familiarity with programming and programming languages. If you are not comfortable with these ideas, you may still benefit from skimming through the rest of this chapter, just to get a feeling for what can be achieved with scripts.

ECMAScript Syntax Outline

If there is any justification for the similarity between the names JavaScript and Java, it is that the sytax of statements and expressions of JavaScript — now standardized in ECMAScript — closely resembles that of Java, and hence of C and C++. If you know any of those languages, the form of ECMAScript programs will contain few surprises. If you are only familiar with Visual Basic, or with the Algol and Pascal family, you may find the detailed concrete syntax slightly odd, but the statement and expression forms available will be roughly as you expect: assignment statements, a variety of loops and conditionals, function definition, call and return, and so on. A major difference between ECMAScript and Java, C++, Pascal and most other mainstream programming languages is that ECMAScript is an *untyped* language. To be precise (since the term 'untyped' does not have a unique meaning), a variable is not restricted, either by declaration or any other mechanism or convention, to holding values of just one type. The same variable can hold a string at one time, a number at another, and a Boolean at another.[9] There can therefore be no type checking. As is the case with other untyped languages, there is no need to declare variables before you use them, although it is good practice to do so.

The primitive values supplied by ECMAScript belong to the types `Boolean`, containing the two values `true` and `false`; `Number`, containing floating point numbers;[10] and `String`, containing strings of Unicode characters (although implementations commonly only support ISO 8859-1). The usual operations on these objects — Boolean algebra for `Boolean`, arithmetic for `Number`, and concatenation (denoted + in ECMAScript) for `String` — are provided. Primitive values can be combined into arrays and objects. This gives you the capability of creating your own data types, but doing so is only really appropriate if you are designing re-usable script libraries, perhaps to serve as behaviours (see page 466). Most of the time, casual scripts of the sort we will display in this section make use of ECMAScript's support for objects by manipulating its *built-in objects*

[9] Not that we recommend the practice.

[10] Double-precision 64-bit format IEEE 754 values.

— the equivalent of a run-time library, including the `Math` and `Date` objects — and *host objects*, which provide the interface to the host environment — in the case we are considering, a Web browser.

Objects

ECMAScript is usually described as an *object-based* language. This means that it supports the creation and manipulation of objects, but these are not systematically organized into hierarchical classes as they are in an *object-oriented* language, such as Java. An object in ECMAScript is just a collection of named data items, known as *properties*,[11] and functions, known as *methods*. Any collection will do, and properties and methods can be added to or removed from an object at any time. If a variable called, for example, `the_object`, holds an object, you can access any property, such as `a_property` using the notation `the_object.a_property`, and call a method, `a_method`, with `the_object.a_method()`. If the method takes any arguments, they are written inside the brackets, separated by commas, as in `the_object.b_method('string', true)`.

> Objects are implemented simply as associative arrays, indexed by the method and property names. This works because, in ECMAScript, functions are 'first-class' values, and can be stored in an array like any other value. The apparently anarchic properties of ECMAScript objects follow from this implementation. If you are writing complex scripts that require you to define your own objects and you wish to impose some order on them, in the way you would do with classes in C++ or Java, you can do so by using a constructor to build them. Whenever a particular constructor *K* is called to build an object, it will always do the same thing, so, apart from any differences deriving from the arguments to *K*, the objects it initializes will be the same — they will have the same methods and properties, although the initial values of the properties may be different. We could very well say the objects built by *K* are all instances of the same class.
>
> ECMAScript provides syntax for calling constructors when objects are created, and even supports a form of inheritance (based on object *prototypes*). Since we are not going to require these more advanced features in our example scripts we will not pursue these matters further. If you are interested in these features, or need them, consult the detailed references provided at the end of the chapter.

[11] Strictly speaking, the property is a container for the data item. This allows attributes to be associated with a property — such as whether it is read-only — in a semantically clean manner. Most of the time, the distinction is academic.

Among the host objects provided by Web browsers to allow scripts to manipulate HTML documents,[12] the document object plays a primary rôle. As you would expect from its name, it provides an interface to an HTML document. It contains properties holding the document's title and various pieces of information derived from the HTTP request used to access it, such as the URL and referrer; there are also properties which are arrays, containing all the links, images, and embedded objects in the document, and a very useful property called all, which is an object having, for each document element with an id attribute, a property with a matching name holding an object corresponding to the element.[13] For example, if a document contains a paragraph element with id value 'opening', the object document.all.opening will be an object describing that paragraph. The document object also has several methods, including write, which writes its argument into the current document.

➲ Note that the document object is not the root of a parse tree of the document structure, as you might expect. The Microsoft object model does provide methods children and contains which can be used to navigate the document hierarchy, but these have not been incorporated into the W^{3}C DOM.

Which document is the document object related to? It is the document currently being displayed when the script is executed. But how is a script to be executed in conjunction with a document? The HTML SCRIPT element is used to embed scripts in documents. When a page is loaded, all the scripts embedded in it are executed, with the document object providing its interface to that page. A simple example will make this clear. Consider the following:

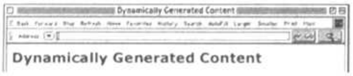
```
<HTML>
<HEAD>
<TITLE>Dynamically Generated Content</TITLE>
</HEAD>

<BODY>
<SCRIPT type="text/javascript">
<!-- /* Hide content from old browsers */
document.write('<H1>', document.title, '</H1>');
// end hiding content from old browsers  -->
</SCRIPT>
</BODY>
</HTML>
```

[12] We will not consider XML.

[13] The W^{3}C DOM specifies methods getElementById and getElementByName for accessing these objects, instead of making them into properties of the document.all object.

When this page is loaded into a browser, it produces the display shown in Figure 14.1. The interesting part is the SCRIPT element, enclosed between start and end tags in the usual way. The start tag sets the value of the type attribute to the MIME type `text/javascript` — other scripting languages may be used, so we must identify the one being used; despite the anachronism the `javascript` sub-type is the appropriate one. Next comes a trick, similar to that demonstrated in Chapter 8 to protect stylesheets from old browsers. There are two forms of ECMAScript comments: bracketed comments, delimited by the sequences /* and */, and end of line comments, extending from the sequence // to the end of line. Both of these are used here in conjunction with HTML comments to hide the script from browsers that don't understand it. In this case, the script is just the single method call:



Dynamically Generated Content

Figure 14.1
Dynamically generated content

```
document.write('<H1>', document.title, '</H1>');
```

The script is executed at the point it is encountered by the Web browser as it parses the HTML document. The `write` method of the `document` object may take an arbitrary number of arguments. Here, we have given it three: the first and last are literal strings containing the start and end tags of an H1 element; the middle argument is the value of the `title` property of the document, and this is a string containing the content of the TITLE element of this page, i.e. `'Dynamically Generated Content'`. The `write` method does what you probably expect, writes its argument into the document. Since the script is executed when it is encountered during the parsing of the document, the text it writes replaces the SCRIPT element, so it is as if the document's body were:

```
<H1>Dynamically Generated Content</H1>
```

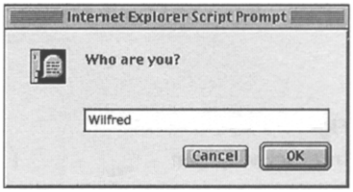
and, indeed, the display produced looks exactly as if it was.

It doesn't take a very astute reader to recognize this script as totally pointless — we knew the text of the title and could have more economically embedded it directly in the document, in the form just shown, without executing any script. Now that we know how to combine script and document, via the SCRIPT element and the `document` object, though, we can go on to produce genuinely dynamic pages.

A clumsy but effective means of generating dynamic page content is to build it out of text solicited from the user. The function `prompt`[14] can be used for this purpose. It takes a single string argument,

[14] Technically, `prompt` is a method belonging to the `window` object that is attached to the window in which the current document is displayed. All unqualified names (including `document`) are implicitly methods or properties of this object, so, nearly all the time, you do not need to be aware of it.

and when called it causes a dialogue box to be displayed, with its argument as a prompt and a text entry field. Any string typed into this field is returned as the result of prompt when the user dismisses the dialogue. For example, the following code will cause the dialogue shown in Figure 14.2 to be displayed; if the OK button is clicked with the text entered as shown, the variable their_name will be set to the string 'Wilfred'.



Who are you?

Wilfred

Figure 14.2
Prompting for user input

```
var their_name = prompt('Who are you?');
```

The value of this variable is in scope in all other script fragments embedded in the document after this call, so it can be used to put together some text incorporating the user's name, which can be put into the document with document.write. A good place for the call is in the document's HEAD. The script is executed — so the dialogue is displayed — when it is encountered. The HEAD is completely processed before the content of the BODY is displayed, so the prompt will appear before the page itself. If prompt is called from within the page BODY, the dialogue may appear in the middle of the page being rendered, which will usually be distracting. Hence, we might use a prompt in the following manner:[15]

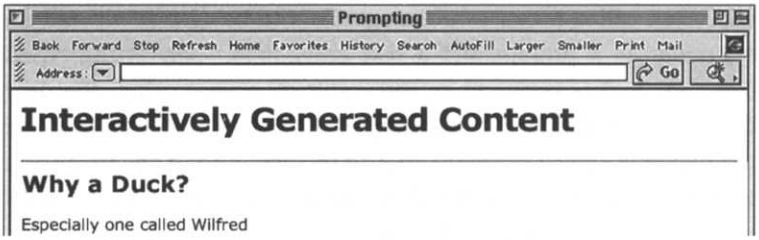
```
<HTML>
<HEAD>
<TITLE>Prompting</TITLE>
<SCRIPT type="text/javascript">
var their_name = prompt('Who are you?');
</SCRIPT>
</HEAD>

<BODY>
<H1>Interactively Generated Content</H1>
<HR>
<SCRIPT type="text/javascript">
document.write('<H2>Why a Duck?</H2>');
document.write('<P>Especially one called ',
                   their_name, '</P>');
</SCRIPT>
</BODY>
</HTML>
```

The dialogue box will appear as we saw previously, and then the page shown in Figure 14.3 will be displayed.

⇨ You probably want to know what happens if the user clicks the cancel button in the dialogue box. An undefined object is

[15] We have suppressed the script-hiding comments, and will continue to do so from now on.


Prompting
Back Forward Stop Refresh Home Favorites History Search AutoFill Larger Smaller Print Mail
Address:
Go

Interactively Generated Content

Why a Duck?

Especially one called Wilfred

Figure 14.3
Interactively generated content

returned, and production code should check for this eventuality and do something sensible. If you want to see what happens without any check, try it.

This example is fatuous, but it demonstrates a serious point: information obtained on the client machine at the time the data is downloaded from the server is incorporated dynamically into the page when it is displayed. In other words, interaction with the user has influenced the appearance of the page.

Event Handlers

Much of the interaction required from multimedia presentations and Web pages is concerned with controlling the flow of information. This can be achieved gracefully by using on-screen controls — icons, images, or text — that respond to user events, such as mouse clicks. A simple mechanism for associating a script with an event and a document element is provided by HTML 4. Almost any document element can have attributes whose name identifies a class of events and whose value is a short piece of code to be executed when the event occurs to that element. Table 14.1 shows the names of the event-related attributes[16] that can be used with most elements, and the events that cause the script that is their value to be executed. Certain elements (for example HEAD) cannot sensibly have these attributes; others, particularly those representing HTML forms may have extra attributes, but we will not go into the detail of these here, since forms are not often used in conjunction with multimedia pages.[17]

The code which is the value of an event-related attribute is known as an *event handler*. It is normal for it to be a call to a function. This way, the code that must be included in line with the element tag is compact, and that which performs the actions can be kept separate

[16] The eccentric use of capital letters is a common convention for showing the component words of these names. It has no actual significance — HTML is insensitive to the case in which attribute names are written.

[17] Note, though, that in versions of HTML earlier than 4.0, *only* form elements had any event handlers, so they were often used for purposes other than data entry where event handling was required.

and easy to read. In this case, we usually refer to the function as the event handler.

One use of event handlers has become a cliché of client-side Web scripting: the 'rollover button'. This is a button — i.e. a small image that serves as a control — whose appearance changes when the cursor is moved over it, to indicate that a mouse click at that point will activate it. This provides some useful feedback to the user, if it is not over-done. The essence of the effect is captured in the following example.

The change of appearance is achieved simply by assigning to the `src` property of an object corresponding to an image. As we mentioned earlier, the `document.all` object has a property for each element with an `id` attribute. The following two event handlers update the `src` of an element with `id` equal to `circlesquare`, the first setting it to a GIF whose name suggests it is a picture of a circle, the second to a similar image of a square.

```
function in_image() {
    document.all.circlesquare.src = 'images/circle.gif';
}

function out_of_image() {
    document.all.circlesquare.src = 'images/square.gif';
}
```

These definitions would be placed in the `HEAD` of the document. The names of the functions have no semantic significance; they are chosen purely to indicate their function to human readers. They are associated with events in the `IMG` tag of the `circlesquare` element:

```
<IMG src="images/square.gif" alt="square or circle"
  onMouseOver="in_image()" onMouseOut="out_of_image()"
  id="circlesquare">
```

Setting the value of the `onMouseOver` attribute to the string `"in_image()"` ensures that whenever the cursor is moved over the image, the function `in_image` will be called, to set the image to a circle. Similarly, the `onMouseOut` attribute has been set to call `out_image` to make the image revert to the original square. The combination of these handlers ensures that the image displays as a circle whenever the cursor is within its bounds, and as a square the rest of the time. In a practical situation, the images would portray, for example, a push-button in different colours, to indicate readiness when the cursor was over it. Additional `onMouseDown`

Table 14.1
HTML 4.0 Event Handlers

`onClick`	a mouse button was clicked
`onDblClick`	a mouse button was double-clicked
`onKeyDown`	a key was pressed down
`onKeyPress`	a key was pressed and released
`onKeyUp`	a key was released
`onMouseDown`	a mouse button was pressed down
`onMouseMove`	the cursor was moved
`onMouseOut`	the cursor was moved away from the element
`onMouseOver`	the cursor was moved over the element
`onMouseUp`	a mouse button was released

and onMouseUp handlers could be defined to show the button being pressed and released.

The technique of updating the src property of an image object can be used to add animation to a Web page, under the control of a script. This is an improvement on the use of animated GIFs, since these cannot be controlled: they play once on loading, or for a specific number of times or forever in a loop. Using a script, we can arrange for an animation to be started and stopped by a control or some other means. In the following example, we make a simple animation play only as long as the cursor is over it — a moving version of the rollover we just described.

The strategy is easily grasped, but a number of tricks are required to make it work properly. We want to associate a sequence of frames — we will use ten — with an image element, and cycle through them in turn. The onMouseOver event handler for our image will start the cycle going, and the onMouseOut will stop it. We can write the image tag straight away:

```
<IMG src="images/animation/frame1.gif"
     alt="simple animation"
     id="the_image"
     onMouseOver="start_animation()"
     onMouseOut="stop_animation()">
```

We are assuming that the images for the animated sequence are in a directory images/animation below the document, as files named frame1.gif, frame2.gif, ... frame10.gif. The first frame is displayed in the image element with id the_image when the page is loaded; the two relevant handlers are set to call functions to start and stop the animation. All we have to do is define those functions, and supply any supporting code they require. The appropriate place to do so is in the document's HEAD.

We know that what we need to do is assign a new value to the_image.src at regular intervals — let us say every eighty milliseconds, to give a frame rate of 12.5 fps — and we know what those values should be (the URLs of the frames of the animation). If we assume that we have a function animate that does that as long as a Boolean variable continuing is true, we can define the handlers in terms of it:

```
function start_animation()
{
   continuing = true;
   animate();
}

function stop_animation()
{
    continuing = false;
}
```

We don't need to declare `continuing` — undeclared or uninitialized variables have an undefined value, which evaluates to false in a Boolean context, when they are first used — but we can if we like (before the functions):

```
var continuing = false;
```

Our problems now seem to be concentrated in the `animate` function. We know from the rollover example how to update images, and it's fairly clear how to use a counter variable to cycle through the frames, but how are we to produce the necessary delay between them? The answer lies in a built-in function that we have not yet described: `setTimeout`. This takes two arguments. The first is a string of code, the second is a number, which will be treated as a time interval in milliseconds. The effect of `setTimeout(`*code*`,` *delay*`)`, is to execute *code* after a delay of *delay* milliseconds. In order to repeat an action at regular intervals, all we need to do is make *code* be a call to a function that calls `setTimeout` with a call to itself as the first argument. This is what our `animate` function does. We need some globals first:

```
var delay = 80;
var num_frames = 10;
var i = 0;
```

Notice that two of these are really just constants, but ECMAScript does not provide anything like C++'s `const`, so if you want named constants you must use variables. Now we can write `animate`:

```
function animate() {
    document.all.the_image.src =  'images/animation/frame'
                                     + (++i) + '.gif';
    if (i == num_frames)
        i = 0;
   if (continuing)
        setTimeout('animate()', delay);
}
```

The expression on the right of the first assignment builds the name of the next frame using the value of the variable `i` as the frame number — the prefix `++` operator pre-increments the value, which ensures that it cycles correctly through the frames. The test for equality with `num_frames` is used to make the value wrap round to zero when it reaches the last frame number, so that we do truly cycle. The last conditional is the key: we test `continuing` to see whether to go on — remember that this variable is set to false when `stop_animation` is called. If we are continuing, we set a timeout, so that `animate` is called again after another delay.

The code just given works after a fashion, but has a major defect: the first time a URL is assigned to the image's `src`, the image file has to be fetched over the network, which may cause an appreciable delay, so the first cycle of animation may be slower than intended. Subsequently, the file will be cached, and loaded much faster. We would like to be able to preload the images so that every cycle runs at the proper speed. The way this is usually done is by constructing an array of `Image` objects, and populating it with the frames of the animation:

```
frames = new Array;
for (var j = 0; j < num_frames; ++j)
{
    frames[j] = new Image;
    frames[j].src = 'images/animation/frame'
                          + (j + 1) + '.gif';
}
```

(Array indices start at zero.) Although these images are not displayed, this code is sufficent to cause them to be downloaded and cached. We can now copy the `src` of one of the images in the array when performing the animation:

```
document.all.the_image.src = frames[i++].src;
```

which is slightly faster than assigning the URL directly and means that we only have to change the initialization code to get a different animation.

Scripts and Stylesheets

In a well-designed Web page, content is separated from appearance, with the latter being controlled by stylesheets. In order to be able to make changes to the appearance, scripts need to be able to manipulate stylesheets. This is possible, but here, above all, there is no standard model that can be used: Level 2 of the W^{3}C DOM, which deals with styles, is only a first draft as we write, and there are significant differences between the available implementations. Conceptually, though, there is no great difficulty, so although the details of the relevant objects may change, the general approach to controlling appearance with scripts should remain roughly as we describe it.[18]

There are two ways in which we can change styles: either by changing the style applied to a particular document element, or by altering the stylesheet applied to an entire document. The former is the easier option: each object in `document.all` has a `style` property, which is itself an object with properties (in the ECMAScript sense) corresponding to the CSS properties applied to the element. By assigning to these, its appearance can be changed. Suppose, for example, a document has the following stylesheet applied to it:

```
H1 {
      color: lime;
}
H2 {
      color: blue;
}
```

Then the following HTML element will produce a lime-coloured header:

```
<H1 id="intro">Introduction to Limes</H1>
```

Its colour can be reset to a more sober value by the following assignment, which might be put into an event handler:

```
document.all.intro.style.color = 'black';
```

Altering the stylesheet itself is slightly more complicated. The `document` object has a property `styleSheets`, which is an array containing an object for each STYLE element in the document. This in turn includes an array `rules` containing an object for each rule in the STYLE element. Each rule object has a `selectorText` property, which allows you to extract the selectors from the corresponding rule, and a `style` property, which provides access to the values of the CSS properties set by the rule. By assigning to these properties,

[18] Our description is of the DOM as implemented by Internet Explorer 4.

changes to the styles affecting the entire document can be made. For example, executing

```
document.styleSheets[0].rules[1].style.color = "fuchsia";
```

would change the colour of every level 2 header in the document. Access to STYLE elements and the rules within them is by numerical index, *starting at zero*. Here, `styleSheets[0]` is the first STYLE; assuming its content is the rules shown above, then `styleSheets[0].rules[1]` is the object corresponding to

```
H2 {
      color: blue;
}
```

so our assignment has the same effect as transforming this rule into

```
H2 {
      color: fuchsia;
}
```

and applying it, with garish results.

Combining the scriptability of styles with absolute positioning enables you to move document elements about the screen. In conjunction with the use of `setTimeout` described earlier, this allows you, among other things, to animate text, in a style often favoured for the opening credits of television news bulletins and sports programmes for example.

> ⇨ Because absolutely positioned elements with different `z-order` values behave rather like layers in Photoshop, and because the first implementation of absolute positioning that was available was based on Netscape's proprietary (and now deprecated) <LAYER> tag, it has become customary to refer to the process of moving absolutely positioned elements with a script as 'animating layers'.

As a crude example of text animation, we will construct a Web page in which the word MOTION moves diagonally from top left to bottom right; when it has gone a certain distance, it stops, and the word BLUR appears superimposed over it for a while, before a new page is loaded.[19] The body of the document looks like this:

```
<SPAN ID="layer1">MOTION</SPAN>
<SPAN ID="layer2">BLUR</SPAN>
```

[19] If you try this, you'll see why we chose the text 'MOTION BLUR'.

```
<SCRIPT TYPE="text/javascript">
   move_the_text();
</SCRIPT>
```

Each word is enclosed in a SPAN element with its own id, so that we can apply positioning style to it. The script will be executed when the page is loaded; it calls a function to do the work. Before looking at that function, we need to see the stylesheet applied to the two spans.

```
<STYLE TYPE="text/css">
#layer1 {
    position: absolute;
    top: 35;
    left: 40;
    font: x-large ;
    color: blue;
}

#layer2 {
    position: absolute;
    top: 300;
    left: 540;
    font: x-large ;
    color: red;
    visibility: hidden;
}
</STYLE>
```

Recall that we use fragment identifiers as selectors to apply a style to a specific element. The first rule thus positions the first layer (MOTION) at (40, 35), using absolute positioning; an extra large blue font is used to make the word conspicuous. The same font, in red, is used for the other layer, which we position down and to the right at (540, 300) (vertical coordinates increase downwards); the visibility property is set to hidden — we do not want to see BLUR until MOTION has stopped moving. We do not bother to specify z-order explicitly; the elements are stacked by default in the order they appear in the stylesheet.

Now to make things move. We use exactly the same technique as we used to animate a sequence of images: the function move_the_text updates the top and left properties of the layer1 object's style, thereby moving it to a new position, and then sets a timeout, with itself as the code to be executed after the delay. To ensure that the process terminates, we use a counter num_steps, initialized to

50 and decremented until it reaches zero. At that point, we make layer2 visible. A feature we have not seen before is then used: the `location` object[20] has a property called `href`, which holds the URL of the document currently being displayed. Updating this property causes a new page to be loaded. We want a pause — so the user can admire the final state of the animation — before this happens, so we set a timeout, this time to a longer value. Here is the complete code of the function, with its associated variables:

```
var delay = 40;
var num_steps = 50;
var x = 35, y = 40;
function move_the_text() {
    x += 10; y += 5;
    document.all.layer1.style.left = x;
    document.all.layer1.style.top = y;
    if (--num_steps)
        setTimeout('move_the_text()', delay);
    else
    {
       layer2.style.visibility = 'visible';
       setTimeout('location.href = "image-swap.html"',
                                            delay*35);
    }
}
```

We should emphasize again that in these examples we have only sought to demonstrate how a scripting language in conjunction with a document object model can be used to add interactivity and dynamic effects to Web pages. We have not tried to demonstrate good programming style, nor to address the real practical issues concerning compatibility between different browsers and different versions of browsers.

Behaviours

Scripts offer the potential for controlling the user interface to a multimedia production on the basis of elaborate computation, carried out using most of the facilities of a modern programming language. Much of the time, authors only require a limited repertoire of interface enhancements and animation facilities, such as those illustrated in the previous section.[21] If an authoring system

[20] Actually a property of `window`.

[21] Since user interfaces are subject to fashion, the desired repertoire may change over time.

can provide a set of parameterized actions that is adequate to satisfy most needs, and a suitable interface for attaching them to elements of a multimedia production, then much of the necessity for scripting is removed. This means that designers can add interactivity and animation to their productions without having to acquire programming skills on top of their design skills. (Conversely, it means that programmers can employ their skills in a way that can be exploited by designers, rather than involving themselves in design.) Parameterized actions provided by an authoring system to be used in this manner are usually called *behaviours*. Since behaviours are going to be extensively re-used, considerable care must be taken in writing them, so that, for example, ECMAScript behaviours cope gracefully with the divergent object models and capabilities of different user agents.

What do we mean by describing behaviours as *parameterized* actions? Simply that each behaviour is an abstraction that captures a class of actions — such as displaying a message to a Web browser's status line — from which individual actions, such as writing the specific message `Please wait` to the status line, can be generated by providing a value for some variable, the behaviour's *parameter*. In this case, the parameter is the message to be displayed; in general, a behaviour may have several parameters. It follows that an authoring system that supports behaviours must provide some means for authors to provide parameter values, and be able to generate code by combining these values with a behaviour. They must also be able to attach the resulting actions to events chosen by the author.

The simplest approach is to build a limited set of behaviours into the browser. You might consider the link-following capabilities of all Web browsers as an instance of this approach. A slightly fuller set of behaviours is supported by PDF files. Recall that in Chapter 9 we described how to add links to a PDF file using Acrobat Exchange: after selecting an active area, a dialogue box appears, which previously we used to set up the parameters of a link. However, the pop-up menu labelled Type in that box (see Figure 9.3 on page 276) can be used to associate different actions with the active region. The possibilities include automatically executing a menu command, playing a QuickTime or AVI movie or a sound clip, launching a different program to open a file of some type other than PDF, or opening a file on the World Wide Web identified by a URL.

The path names of the movie, sound clip, or file, and the URL are the parameters to these behaviours.

The repertoire of behaviours available in Acrobat is fixed, and the actions themselves are coded as part of the reading software. In contrast, modern Web browsers include code to interpret a scripting language, so that arbitrary actions can be written in that language. You should be able to see in outline how, instead of being written by hand like the examples in the previous section, ECMAScript actions can be parameterized as behaviours and incorporated into HTML semi-automatically. In order to fill in the details, we will describe the interfaces to behaviours provided by a representative HTML authoring program, Macromedia Dreamweaver, and sketch their implementation.

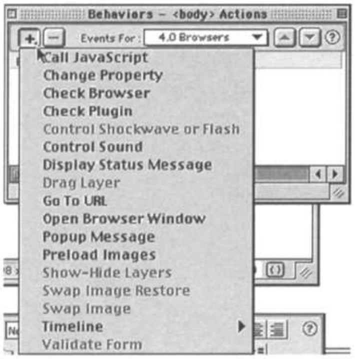


Figure 14.4
Dreamweaver's behaviours palette

Most HTML authoring programs (as against general-purpose text editors with HTML modes, such as Alpha or Emacs) use a similar interface, comprising a document window providing a WYSIWYG view of the document being composed, and several context-sensitive floating palettes, whose contents change according to the element selected in the document window. These palettes allow attributes of the selected element to be changed. Dreamweaver's behaviours palette (shown in Figure 14.4) is used to attach actions to events associated with the selected element. Clicking the + button causes a pop-up menu to appear, as shown, listing the names of all the available behaviours — Dreamweaver ships with a small collection, others can be added, as described later. Menu entries for behaviours that, for some reason, are not applicable to the currently selected element are disabled. For example, unless an image is selected the `Swap Image` behaviour will not be available. When a behaviour has been selected, a dialogue box is displayed, with fields for any parameters that are needed to generate the code. Once these have been supplied, the event to which the action is to provide a response is selected; each behaviour has a default, but any other event supported by the DOM can be chosen instead.

Since Dreamweaver is an HTML editor, its basic function is to change the contents of HTML documents. It can therefore embody the association of document element, event and action by writing suitable ECMAScript code in the document's `HEAD` to define a handler function, and then inserting an `on`*Event* attribute into the start tag of the selected element. For example, suppose an `IMG` element is selected, and two instances of the `Display Status Message` behaviour are attached to it, one for the event `MouseOver`, the other, with a blank message, for MouseOut, so that the first message will be displayed for as long as the cursor is over the image. Dreamweaver will insert a script equivalent to the following one into the document head:

```
<SCRIPT type="text/javascript">
function displayStatusMsg(msgStr) {
  status = msgStr;
}
</SCRIPT>
```

and decorate the <IMG> tag with attributes similar to the following:

```
onMouseOver=
    "displayStatusMsg('The cursor is inside the image')"
onMouseOut=
     "displayStatusMsg(' ')"
```

The collection of behaviours shipped with Dreamweaver provides an insight into the range of actions that can be readily added to a Web page in this way. (Looked at more cynically, since most of the other powerful HTML tools presently available support roughly the same set of actions, this collection provides an insight into Web page design clichés — at least until the next round of updates to these programs is released.)

These behaviours can be divided into groups. The first, consisting of Display Status Message and Popup Message display messages in response to events, in the status line and a dialogue box respectively. The second group consists of functions that check the browser's capabilities. Check Browser is used to determine which version of which browser is being used, and to select different URLs on the basis of this information, so that, for example, a user viewing a page with an old version of Navigator would be transferred to a simplified version of a page. Check Plugin can be used similarly to send users to different pages depending on whether they have installed a particular plug-in or not. For example, users without the QuickTime plug-in would be directed away from pages with embedded movies.

The next group of behaviours offers control over browsers and page elements. If a page has embedded Flash or Shockwave movies, or sounds, the Control Shockwave or Flash and Control Sound behaviours can be used to start or stop them, or for movies, rewind or go to a specific frame, all in response to any of the

DOM events. For example, a sound could be made to play when a page is loaded, and stop when the user clicks on an image. The apparently redundant Go To URL is used to load a new Web page; the reason it is useful is that it understands about framesets, and can be used to change the contents of several frames with a single click, without the need for an intermediate frameset. Open Browser Window spawns a separate window to display a new page.[22]

[22] Compare show="new" in SMIL.

Probably the most popular group is concerned with images and motion graphics. Swap Image provides the action needed to implement rollover buttons, or to perform image replacements in response to any chosen events. Preload Images performs the preloading operation described previously, to ensure that image swaps work smoothly. It is usually included automatically with Swap Image.

> ⇨ Image rollovers are such a common requirement for fashionable Web pages that Dreamweaver provides an Insert Rollover command, which allows you to create a rollover effect simply by selecting the images for the two states. Inserting one of them and creating the necessary code is all done in one step. Most HTML authoring packages provide this function, as do many graphics packages that have been augmented with Web capabilities or designed for preparing Web graphics. It can only be a matter of time before your emailer does too.

Show-Hide Layers controls the visibility of layers, and Change Property is a general-purpose behaviour for altering the style of a document element. The final behaviour in this group is Drag Layer, which can be used to add basic drag and drop functionality to absolutely positioned elements of a page, or to implement analogue-like controls, such as dials and sliders. It provides a powerful argument in favour of the use of behaviours, since implementing dragging within the confines of ECMAScript and the DOM requires some non-trivial programming: all mouse movements must be tracked and the position of the dragged object updated in response. Care must be taken to observe the physical constraints of the page. If you also want, for example, to reserve certain areas of an image as 'drag handles'; to provide a target for each dragged object, allowing it to snap to its destination when it is close enough; to support several draggable layers on the same page; and to work with as many versions of different browsers as possible, you would have to write quite a lot of code. As it is, it may be necessary to add hand-made code to the basic behaviour to monitor the position of the moving element.

➲ The behaviours that we have described are not built in to Dreamweaver in the same way that PDF behaviours are built in to Acrobat, new behaviours can be added to extend the repertoire. The mechanism by which this is achieved is an interesting exercise in economy, since all it requires is HTML and ECMAScript — which Dreamweaver has to be able to interpret already.

A behaviour is just an HTML file, whose `HEAD` contains some ECMAScript definitions, and whose `BODY` is a document that will be rendered in a dialogue box when a user selects this behaviour from the pop-up menu in the `Behaviours` palette. Usually, the document will contain a `FORM` element, with fields for accepting the values of the behaviour's parameters.[23] When the form is filled in, these values become available to scripts within the behaviour via objects in the same way as elements' attribute values.

The author of the behaviour must provide two functions: `behaviorFunction` and `applyBehavior`, which both return strings to be inserted in the HTML document being edited. The first returns the code for a handler function definition, the second a call to that handler, to be inserted into the selected element's start tag. Typically, the strings returned by both these functions will incorporate values elicited from the user by the form in the behaviour's `BODY`. Behaviours may include several other function definitions, notably `canAcceptBehavior`, which returns a list of events with which the behaviour can be associated, allowing Dreamweaver to correctly set up the pop-up menu by which the event is chosen when a behaviour is added to a document. A `displayHelp` function can also be defined, which returns a string describing the behaviour when the help button in the behaviour dialogue is pressed.

One can imagine simpler ways of implementing behaviours — using action templates, for example — but this approach permits great flexibility, since all the features of ECMAScript are available for creating the function definitions and calls. (If you know about JavaBeans, you might like to compare the approach with the way a programmer implements the `PropertyEditor` interface so that visual development environments can manipulate the properties of beans.)

Earlier, we showed how layers can be crudely animated using a script. The code for performing this type of animation is as amenable to prefabrication as the behaviours we have just seen, but a different interface is required for conveniently attaching the code to a document. The familiar timeline is provided by Dreamweaver and other authoring tools that support animated layers. Given their shared provenance, it is not surprising that Dreamweaver's timeline resembles a simplified version of Flash's. It appears in its own

[23] We have not decribed HTML forms, since they are quite complex and not particularly interesting in the context of multimedia. Consult one of the HTML references for details on forms, the elements that may be used within them, and the special events that are associated with form elements.

palette, with the document view serving the same function as the stage in Flash. A layer can be dragged to the timeline, where a bar appears, representing its extent in time. This can be stretched or shrunk as necessary. Key frames can be added to the timeline at any point, and the layer can be moved to its desired position in that key frame. Movement between key frames is linearly interpolated by the script generated by Dreamweaver from the timeline.

➲ The code generated by Dreamweaver, although it is based on the same principle as the code we wrote to animate some text in the previous section — move the element to its new position and set a time out — is slightly smarter, because it precomputes all the necessary positions, and stores them in an array, which leads to more efficient animation.

Timeline Scripting and Behaviours

Scripting can, as we have shown, free hypermedia from its purely spatial model, and add an element of temporal organization to it. Where multimedia is primarily organized on a timeline, scripting can provide interactivity and non-linearity. Timelines provide new sites for actions — an action can be attached to a frame and triggered when playback reaches that frame — and new sorts of action, ones that control playback. These two innovations offer three new opportunities to multimedia authors. By attaching playback control actions to static elements — such as buttons — that can respond to events, timeline controls — such as fast forward, rewind, and pause buttons — can be implemented. By attaching other actions to frames, their effects can be choreographed in time. By attaching timeline-controlling actions to frames, the timeline can be made to loop. Notice that the effect of actions attached to frames is fully determined — only elements that can respond to events initiated by users can produce non-linear behaviour.

We shall begin by considering Flash, which, although it is primarily an animation tool, provides some simple behaviours. Actions generated by supplying parameters for these can be attached to key frames (not interpolated frames) or to special symbols called *buttons* (irrespective of their appearance). An action is attached to a key frame simply by selecting the key frame and the appropriate menu command, choosing one of the available behaviours, and entering values for its parameters. The action is triggered when playback reaches the frame to which it is attached. Attaching actions to buttons is achieved by a similar sequence of operations; first, however, the button must be created. Flash buttons are special symbols (in the specialized sense in which Flash uses the word), consisting of four individual frames, the first three of which correspond to the button's appearance when it is in one of the states 'up' (the cursor is not over it), 'over' (it is), or 'down' (the mouse is clicked while the cursor is over the button). The fourth frame defines the 'hit area', the region that responds to the mouse. Once a button has been created, it can be placed on the stage like any other symbol, animated, and have actions attached to it.

The behaviours Flash offers to control timelines are elementary: `Play`, `Stop`, and `Go To`. The last of these takes a frame number or a label — which can be attached to a frame as a mnemonic — as its parameter; it also has options for selecting the next or previous frame as the destination. Whatever the destination, a further option to the `Go To` behaviour can be used to determine whether the movie should play from the new frame or stop there. It should be evident how these behaviours can be used with buttons to start and stop a movie or jump to a selected point. They are of limited use when they are attached to the timeline; using a `Go To` action to produce a loop is the only effective option. Combining a looped movie with buttons that cause a jump to a selected frame is, however, sufficient to provide a selection of different interfaces for offering a user some meaningful control over a presentation.

A straightforward example of such a combination allows us to provide a hypermedia-like interface. Suppose a movie consists of several distinct scenes,[24] each one associated with a particular subject. For example, if we wished to demonstrate the effects that could be achieved using Flash's built-in drawing tools, we might produce a set of short animations, each of which showed a picture being built up using one of the tools: we could do some motion painting with the brush tool, move some shapes about with the arrow tool, and so on. We want to present the user with a welcome screen containing icons of each of the tools, so that clicking on a tool causes the corresponding animated demonstration to play, after which the welcome screen returns (see Figure 14.5).

This is easily done. The welcome screen can be a single frame — let's say the first frame, though it doesn't have to be. Attached to this frame is a `Go To` action, with the frame itself as the destination;

[24] In fact, Flash allows you to split movies into separate named scenes that can be treated as distinct entities.

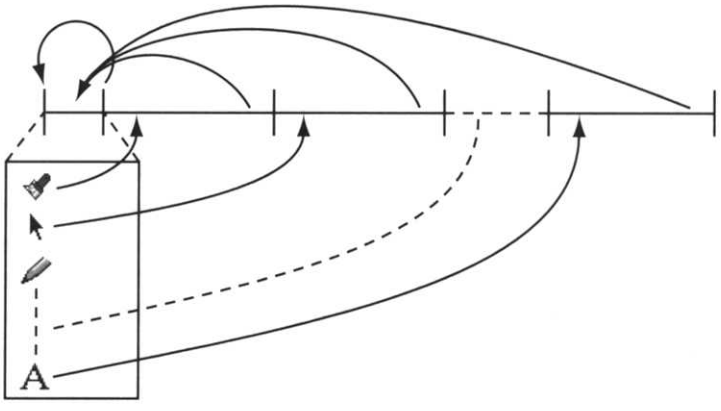


Figure 14.5
Structuring a timeline with selections

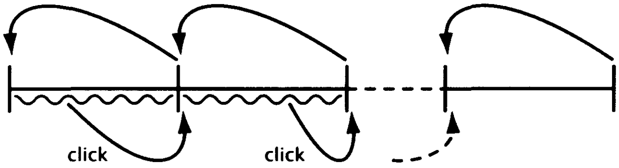


Figure 14.6
Structuring a timeline as levels

i.e. the movie will hold on this frame forever, unless some action causes it to do otherwise. Each of the icons in this frame will be a button with a Go To attached, with the starting frame of the corresponding demonstration as the destination. The final frame of each demonstration will have `Go To Frame 1` attached, to ensure that the welcome screen returns after it has finished. To complete the presentation, an exit button with a `Stop` action should be added to the first frame, to give the user a way out of the infinite loop.

Because buttons in Flash are symbols, they can be animated, which allows for a type of interface that is not available in hypermedia (without the aid of scripting to do the animation). A slightly fanciful example will demonstrate this. A little exercise for helping new computer users to become proficient in using the mouse might be organized as a series of levels of increasing difficulty. At each level, the task is to click on a moving object. When this is done succesfully, the user moves on to the next level, where the object is smaller and moving faster in a more complicated path, possibly with extra objects moving around and getting in the way (see Figure 14.6).

The scenario more or less determines the implementation. Each level is a short animation, with a Go To action attached to its final frame, taking it back to its first frame. These animations are constructed separately, each with its own object moving on its path, so that its position in the final frame is placed to join up with the position in the first. The objects are Flash buttons, with an action that goes to the first frame of the next level. Since actions are triggered by mouse clicks, all that the user needs to do to advance to the next level is catch the object with the mouse. After a suitable number of levels, the action on the object at the hardest level takes the user to a short congratulatory animation, after which they can get on with using their mouse in earnest. We could (and probably should) have added a facility for going back a level, or giving up, and possibly a way of selecting a starting point.

As well as demonstrating the power of behaviours — putting together a prototype took us hardly any time — this example shows some of their inadequacy. The mouse trainer would be more useful if it could take account of how long a user took to master a level, by counting the number of iterations of the loop before they caught the object, so that if a user was obviously having difficulty they could be taken on at a slower pace than those whose eye and hand coordination was well developed. This cannot be done, since Flash provides no mechanism for storing state information. A scripting language could be used to do this[25] — but even a scripting language could not be used to save the state of a movie so that it could be restarted at a later date from the same point, unless a facility to do so was included as part of the object model. Generally, this is not done — multimedia scripting languages do not usually provide any facilities for writing to disk.

Flash's most advanced behaviour is designated `Tell Target`, which is used to send an event from one object to another. This is commonly employed to enable buttons to control individual elements of a movie. You will recall that a Flash movie can be included in another movie, in the form of a symbol. The included movie has its own timeline, which can be controlled by actions. `Tell Target` enables a different symbol to trigger one of those actions. To make identifying the target easier, symbols can be given names. Suppose, as a simple example, we have a movie consisting of nine frames, containing the digits 1 to 9. A `Stop` action attached to the first frame prevents the movie from playing. We now construct a single frame movie, by placing an instance of this movie on the stage, giving it the name `counter`, together with two other buttons, one representing a + sign, the other a - sign. We attach

[25] Version 4 of Flash added a primitive scripting language. Unfortunately, it uses its own idiosyncratic syntax, instead of being based on ECMAScript.

to the + button a Tell Target action, with its target parameter set to counter and the action for the target to perform specified as Go To Next Frame; a similar action is attached to the - button, telling counter to go to the previous frame. When this movie is played, nothing happens spontaneously — there is only one frame — but if the + button is pressed, the displayed digit is incremented, provided it is less than 9, and if the - button is pressed, the digit is decremented if it is greater than 1.

You should be able to see how Tell Target actions can be used to implement controls that affect individual elements of a movie. Again, the restricted nature of the available behaviours imposes some limitations on what can be achieved. For instance, it is not possible for the + to disable itself when the displayed count reaches 9. However, with ingenuity, complex controls such as pop-up menus can be assembled out of buttons and movies, using the built-in behaviours.

When we consider how techniques similar to those we have described in the preceding paragraphs can be applied in the context of the World Wide Web, we find a somewhat different situation. Timeline animation on a Web page is itself implemented using scripting, as we have previously described, so actions that are to be triggered when an animation reaches a certain frame can simply be embedded in the function that performs the animation, to be executed conditionally when some current frame counter reaches the appropriate value. For example, if we wished to modify the 'motion blur' example from page 464 so that the colour of the moving word MOTION changed to fuchsia at the fifteenth frame, we could do so very easily, by adding a new global variable

```
var action_frame = 15;
```

and inserting two lines of extra code in the function move_the_text():

```
if (--action_frame == 0)
    document.all.layer1.style.color = 'fuchsia';
```

If you compare these two lines of code with the two lines that move the text layer in the same function, it is easy to appreciate that in the context of Web pages, animation is just a special case of time-based execution of scripting code. Conceptually, though, it may often be helpful to treat the timeline as an organizing framework, just as we do in an intrinsically time-based system such as Flash. In that case, what is required is a reusable framework for arranging actions, including animation, in time. If this can be done, the timeline can be presented as part of the interface to an HTML authoring system. The three opportunities identified at the beginning of this section as being offered by the combination of actions and timelines become available to Web designers.

We briefly described at the end of the last section how Dreamweaver does offer such a timeline interface, which it uses to generate scripts using timeouts to implement animated layers. The facilities go further: the timeline has a behaviour channel, which allows actions to be attached to any frame. Arbitrary ECMAScript code can be used, as can any of the available behaviours. Several different layers can be controlled by the same timeline, just as several different symbols can be animated in Flash. In addition, several different timelines, each with its own name, can be attached to the same page. This enables different animation sequences to be controlled independently by events. The behaviours `Play Timeline`, `Stop Timeline`, and `Go To Timeline Frame` are all parameterized in a timeline's name, as well as being associated with events and document elements in the usual way. It is thus possible to provide a set of controls for playing an animation by associating these behaviours with images and binding them to the animation's timeline. Multiple sets of controls, controlling multiple animations on the same page, then follow naturally from the ability to define several timelines for the same document.

Because putting together a reusable scripting framework that supports time-based actions, interpolation and key frames is not a trivial exercise, it is usually advisable to adopt an existing one if it is available. It may then sometimes be necessary to mix hand-crafted scripting with prefabricated behaviours to achieve the desired effect — since a fully-featured scripting language is available, effects that are not possible within a fixed repertoire of actions can be implemented. For example, suppose we wanted to re-work our earlier Flash example of a counter controlled by two buttons as a scripted Web page. We could place GIF images for the two buttons and the first digit each on their own layer so that we could position them where we wanted. By dragging the digit's layer on to the timeline palette, creating nine key frames and resetting the `src` attribute of the image in each key frame we could create an animation that ran through the nine images of digits. By not attaching any `Play Timeline` action we would ensure that, as before, no change took place spontaneously. Adding a `Preload`

Images action to the document's HEAD will ensure that when we do want the digits to change they do so smoothly.

All we need to do to complete the example is add actions to the + and - images — which are serving as controls — to cause the animation to go to its next and previous frame respectively, in order to achieve an effect identical to that of our earlier effort with Flash. Dreamweaver only ships with a behaviour that causes a timeline to go to a specific numbered frame, not to the next or previous one. By inserting such an action and inspecting the generated code, we discover that Dreamweaver uses a function called MM_timelineGoto, which takes the name of a timeline as its first argument, and a frame number as its second, to implement its Go To Timeline Frame behaviour. Armed with this knowledge, we can not only implement the required operations, we can make something sensible happen if the digit's value goes outside the range one to nine — we choose to make it wrap round.

We begin by declaring a variable, and asserting that, except while an action is being performed, its numerical value will be equal to that denoted by the displayed digit. To establish this invariant we must initialize the variable to one.

```
var current_digit = 1;
```

Considering the layer holding the digit's image element as an animation, our assertion also amounts to saying that this variable's value is equal to the current frame number. We can therefore use it in a pair of functions that cause the next and previous frame to be displayed by incrementing or decrementing its value and then passing it to MM_timelineGoto as its second argument. If current_digit is about to go out of range we first make a suitable adjustment to produce the intended effect.

```
function up()
{
  if (current_digit == 9) current_digit = 0;
  MM_timelineGoto('Timeline1', ++current_digit);
}

function down()
{
  if (current_digit == 1) current_digit = 10;
  MM_timelineGoto('Timeline1', --current_digit);
}
```

(Timeline1 is the name of the timeline corresponding to the sequence of digits.) The functions up and down can be used as handlers for a mouseDown event in each of the two control images.

Combining behaviours, timelines and *ad hoc* scripts within an HTML authoring environment is a relatively easy way of producing dynamic Web pages that exhibit change with time and respond to user input in a variety of ways. It is always the case, though, that what can be achieved is restricted by the limitations imposed on Web scripting languages for the sake of security.

The combination of scripts with a timeline is taken to the extreme (and possibly beyond) in Macromedia Director. To begin with, Director provides behaviours that can be attached to frames or sprites to implement much the same range of time-based actions and controls as Flash, but in the richer multimedia environment provided by Director. A Message Sprite behaviour can be used like Tell Target in Flash to control one sprite from another. At the next level of sophistication, Director allows authors to compose their own behaviours by combining actions — chosen from a built-in set — into a sequence, which is then treated as a single behaviour. Composition is done simply by repeated selection from a menu, so no real scripting skill is required. The set of available actions includes several to control playback, such as Go to Frame and Wait on Current Frame, others to make global changes to the playback environment, such as Set Volume and Change Palette, as well as some to control sprites and cast members: Play Cast Member plays a sound from among the cast, and Change Cast Member, which takes a sprite as one of its parameters, associates a new cast member with that sprite, so that its appearance changes. Behaviours attached to a sprite are triggered by the usual user-initiated events: a key or mouse button being pressed or released, the cursor entering, leaving, or being over the sprite. Those attached to a frame may be triggered when the playback head enters the frame or when it leaves it; an additonal event, Prepare Frame, which occurs between the playback head's leaving the previous frame and entering the current one, can be used to provide an even finer degree of control over the precise sequencing of actions.

The full range of Director — the thing that made it the leading multimedia authoring environment of its time — depends on the use of its scripting language *Lingo*. Lingo belongs to the tradition of 'English-like' scripting languages rather than those, such as ECMAScript, derived from C and other mainstream programming languages. To anyone used to more established languages, the most noticeable aspect of Lingo is its verbosity: a property p of an object obj is referred to as the p of obj, for example, giving rise to expressions like the rect of the member of sprite 19; assignment uses the set keyword, in the style of Basic, as in set name = "Wilfred".[26] At a sightly deeper level, Lingo ostensibly has no user-defined functions or subroutines. Instead, users can define handlers for 'custom events' — events that aren't predefined. In fact, custom handlers are functions, and custom events are calls to them. Handlers can take arguments and return values. For example, a handler might be defined in the following way:

```
on mean a, b
    set m = (a + b)/2
    return m
end mean
```

and invoked as follows:

```
set average = mean(843, 438)
```

In the case of built-in events, the syntax used for the handler's definition conflates it with the association with an event; using the same syntax for user-defined functions, and calling them custom handlers, is merely a piece of conceptual economy, with no deep semantic significance.

Underneath the cosmetic differences, Lingo has roughly the same facilities and expressive power as ECMAScript, which it also resembles in being untyped. It may have some claim to be object-oriented, rather than object-based,[27] since it has constructs that behave somewhat like classes and inheritance, though again unconventional terminology is used to describe it ('parent scripts' for classes, and 'ancestor scripts' for base classes). The object model is not separated from the language, as it is in World Wide Web scripting, but there is clearly an object model embedded in Lingo, which encompasses all the elements of a Director movie — score, cast, sprites, and so on. Lingo therefore provides a completely flexible way of controlling all those elements on the basis of user input and computation carried out by scripts.

Not only do Lingo scripts offer extra flexibility, compared with behaviours, they can be attached to the production in more places, and be triggered by some extra events. Scripts are divided into different categories, depending on which elements of the

[26] With the release of Director 7, some changes were made to Lingo, so that its syntax is beginning to converge with ECMAScript. Object properties can now be accessed using a dot operator, and set is no longer required. (On the other hand, to can, and always could, be used instead of = in an assignment.) The old syntax is retained for compatibility.

[27] In the absence of static type checking, there is little significant difference between the two approaches to objects.

production they are attached to. 'Score scripts' are the same as behaviours, and may be attached to sprites or frames. These scripts can respond to mouse, key and frame-related events, with the sprite's handler taking precedence over the frame's if there is any conflict. A sprite may also respond to `beginSprite` and `endSprite`, which occur in the first and last frame, respectively, of its duration. 'Scripts of cast members' are attached, as the name suggests, to a cast member itself; remember that a cast member is a media object, such as an image or a sound, whereas a sprite is an instance of that object, placed on the stage, with a specific duration. These scripts respond to the same events as frame scripts, although the latter take precedence. Every sprite that uses a cast member automatically acquires the scripts of that cast member. Moving further outwards in the structure of a production, 'movie scripts' are attached to an entire movie, and 'primary event handlers' to the entire production (which may comprise several nested movies). Movie scripts respond to three additional events: `prepareMovie` (the very first event that occurs), `startMovie` and `stopMovie`. Primary event handlers can only be set up for key presses, mouse clicks, and timeouts.

The final category of scripts is, in some ways, the most interesting. 'Parent scripts' contain the definitions of a collection of handlers. An object may be constructed from a parent script using the `new` statement, which causes an object to be created, and attaches all the handlers in the parent script to the object. Parent scripts usually include a special handler for the `new` event, which is invoked when an object is created, and used to initialize it. If you are familiar with object-oriented programming, you will recognize the `new` handler as a constructor, and the handlers in the parent scripts as methods belonging to a class which the parent script defines. Parent scripts may also include some property definitions; any object created from the script acquires the same properties. (We might call them 'instance variables'.) An object created from a parent script is called a 'child object' (rather confusingly, since inheritance is not yet involved,): it is just an object in the classical sense of some code and some data structured so that it responds to certain messages. Every child object created from a particular parent script responds to the same set of events in the same way; the values of each child object's properties may change independently, so that each child object is an instance of the particular 'sort of thing' that its parent script implements. If a parent script has a sprite as one of its properties, it can be used to create generically similar sprites that respond to the same set of events. This is clearly more manageable than creating sprites and attaching the same scripts to each one individually. A parent script may declare another parent script to be its ancestor, meaning that any events that cannot be handled in the original parent script will be passed on to the ancestor — in other words, the parent script *inherits* handlers from its ancestor. Ancestors may have ancestors of their own, so inheritance hierarchies can be constructed.

We have seen that custom handlers can be called from any script. The properties of sprites, such as their location, size, cast member, and so on, can be changed within handlers. In other words, a sprite can be controlled by scripts. Together with the structuring facilities provided by parent scripts, this offers the possibility of an alternative to the timeline for organizing movies: organize them as programs. For example, instead of implementing a bouncing ball by adding key frames to the timeline at the extremities of its motion, and interpolating a sequence of frames in between, it could be implemented using a single looped frame, with a handler triggered each time the frame was entered that computed a new position for the ball sprite and updated its location accordingly.

The advantage of this way of controlling the elements of a movie is that scripts can perform arbitrary computation. In the example just cited, the ball's position could be calculated from equations derived from the laws of motion governing its trajectory, thus producing a more realistic bouncing movement than that which would be produced by interpolation[28] between the extremities of the motion. Parameters such as the ball's elasticity could easily be changed by editing the script. In more elaborate productions, scripts can be used to endow sprites with artificial intelligence, enabling them to react to user input, or to each other, in less predictable and possibly more satisfying ways than the simple branching behaviour that is the natural concomitant of timeline organization.

⇨ There have been attempts to create multimedia authoring environments that started with the interactions between objects as their basic organizing principle. Such object-oriented environments, as they are inevitably called, have failed to achieve widespread popularity. Despite an enthusiastic cult following, the most succesful such system, Digital Chisel's mTropolis (sic), was killed off shortly after being acquired by Quark, leaving no serious object-oriented authoring environments at the time of writing.

[28] Director uses Bézier interpolation by default.

Beyond Scripting

Traditionally, programmers have thought in terms of creating an application that does something, and adding a user interface to it. The more enlightened programmers will have designed the interface as an integral part of a complete system, and will probably have used some application framework or set of reusable components to build the interface, but the implicit model of program structure is one in which the user interface is a subsidiary component of a system intended to perform some task. Where we need a multimedia interface, or to manipulate media objects, it may be more appropriate to think about creating the user interface in an authoring environment and adding the functionality inside it, so that the interface becomes the organizing principle, with the code to perform the computational task added to the interface instead of the opposite way round. The rôle of scripting, though, has traditionally been seen as one of controlling the elements of a production and reacting to input, and scripting languages usually lack features, such as file input and output, that are necessary for many applications. (ECMAScript on the Web, as we have seen, must lack such features, for security reasons.) Hence, while the scripting facilities and behaviours we have described so far can do plenty of things to make the displayed media elements respond to the user, they are really only enhancing the user interface, not adding real functionality. To do that, some additional mechanism is required.

One approach is exemplified by Director. *Xtras* are code modules, written in C or C++, which can be used to extend Director, adding capabilities that the basic application does not provide. These can range from support for new media types to networking and database management.

There are four types of Xtra. Transition Xtras provide new forms of transition, in addition to the built-in dissolves, wipes, and so on; tool Xtras add new functions to the authoring environment. The remaining two kinds are the most germane to our present discussion. Sprite Xtras are used to add new classes of objects to the range of cast members. A sprite Xtra usually defines some visible representation, and a set of properties and behaviours, so sprites created from it can handle events and perform computation. The difference between sprite Xtras and parent scripts is that the former, being written in C, are not subject to any restrictions on the kind of operations they can perform. Sprite Xtras can be written to interact with databases or other persistent data, or to represent and display elements of a complex simulation, for example. The final type of Xtra is the scripting Xtra, which adds new commands to Lingo itself. An example is the `fileio` extra, which provides commands such as `open` and `write` for accessing files; these commands are not otherwise available in Lingo.

Xtras must be written to conform to an interface specification that allows Director to recognize them and manipulate them like elements of its own environment. Director prescribes the interface — the function names and their signatures — and the Xtra programmer provides the implementations. Every Xtra must implement a function that registers it, passing various identifiers and information about the Xtra's capabilities to Director, so that it can be manipulated in the authoring environment. For example, its name will be added to appropriate menus, and, if it implements a custom editor for changing its properties, this will be made available when it is required. Additionally, a sprite Xtra must implement functions to create instances of itself, draw a representation of itself, if one is needed, and respond to events; a scripting Xtra must provide a despatching function that allows it to be called from Lingo. Behind these interfaces, Xtras can perform any computation at all, so that, as we suggested before, an arbitrarily complex multimedia application can be constructed by embedding computational elements, in the form of Xtras, inside a multimedia production made in Director.

⇨ Besides adding new functionality, there is another reason for resorting to Xtras and similar extension mechanisms that allow programmers to mix C++ or a similar language with scripting. At the end of the previous section, we mentioned the possibility of using scripts to endow sprites with some artifical intelligence. One could do this in Lingo. However, when things have got as far as artificial intelligence, many programmers would doubt the wisdom of continuing to use a scripting language instead of a programming language that offers more sophisticated scoping, compile-time type checking, and the object-oriented facilities that can be built on them. Lingo was designed to be accessible to non-programmers, but using it in advanced ways is programming, and anyone who can program in Lingo will be able to program in Java or C++, and do so more effectively with the assistance of language features designed to help in the task of structuring complex programs.

The most loudly trumpeted mechanism for extending multimedia productions through conventional programming must be Java *applets* — small programs, written in Java, that can be embedded in HTML documents. With the advent of applets, executable programs became, in effect, another media type that can be included in Web pages. Java is a complete object-oriented programming language, and it has a set of powerful libraries that allow applets to display graphics and animation, play sounds and movies and handle events. Subject to security restrictions described below, they may also communicate over networks, access databases, control domestic appliances, and carry out many other computational tasks.[29] The technology required to execute applets comprises two components: a way of embedding applets in HTML, and a way of executing code from within a Web browser.

Originally, the first was provided by a special-purpose APPLET document element, but, in accordance with the view that an applet is a media element like a QuickTime movie, APPLET is deprecated in HTML 4.0 in favour of the OBJECT element, which we introduced in Chapter 13. For executable OBJECTs, the classid attribute is used to specify a URL where the implementation can be found. Any OBJECT can contain PARAM elements, as we described in conjunction with using OBJECTs to embed movies in HTML documents. For applets, these can be used to pass named parameters to the applet. For example,

```
<OBJECT classid="NervousText.class">
<PARAM name=text value="Hooray for Captain Spaulding!">
The most famous applet should be executed here.
</OBJECT>
```

will embed an applet that runs the notorious 'nervous text' demo, with the string "Hooray for Captain Spaulding!" being passed to it as the value of its text parameter.

An applet OBJECT can be formatted and positioned using CSS, like any other document element. (An OBJECT is an inline element, like an IMG.) The height, width, and co-ordinates, if specified, of an OBJECT define an area within the browser window into which the applet can draw and write, treating it as a Java frame. Applets can also create their own windows as they run, but these always carry an intimidating warning in their title bar, so that downloaded applets cannot masquerade as system windows, and elicit information to which they have no right from unsuspecting users.

[29] Applets are not the only programs that can be written in Java. Standalone applications that do not depend upon a Web browser for their execution can also be generated.

➲ Applets do not actually have to be written in Java, although they usually are. The `classid` attribute's name seems to embody the assumption that applets are Java classes, but, providing a suitable method for executing the code is implemented in the browser, Java does not have a monopoly on applets. The `OBJECT` element has an attribute `codetype` that allows the type of executable code to be specified, just as its `type` attribute allows the type of data to be specified.

Java class files only store the code for a single class; in the case of an applet, the class that implements it, which must extend `applet`. If an applet uses other classes, and perhaps some data files that are always the same, you can package everything it needs into a single archive, and use the `archive` attribute to inform the browser of its whereabouts, so that the entire archive can be retrieved at once, without having to make multiple connections to get all the pieces. You still need the `classid` attribute, to identify the applet from among the classes in the archive.

If you need to preserve the state of an applet between invocations, you can use Java's object serialization facility, and specify its serialized form in the `<OBJECT>` tag. In this case, you must use the `data` attribute, not `classid`.

One of Java's best publicized features is its *platform independence*, often summed up in the slogan 'Write once, run anywhere'. This is clearly a necessity if applets are to be able to run in any browser, on any machine, under any operating system. Equally clearly, platform independence cannot be achieved by a conventional compiler that generates machine code for a specific machine. Java's solution is to compile the source code into code for a *virtual machine*. A virtual machine is a fully specified machine, with its own instruction set, data types, and so on, but it does not exist in hardware. Its instructions are interpreted by a piece of software — an *interpreter* — which emulates the behaviour of the virtual machine. Interpreters for Java's virtual machine code have been written for all major platforms. Dissatisfaction with the performance of early interpreters has led to the adoption of more efficient techniques. In particular, 'just-in-time' compilers are used to generate native machine code on the fly from the virtual machine code, when doing so would provide better overall performance.

⇨ If you have read some of the more enthusiastic accounts of Java, you may be forgiven for thinking that the technique of compiling to a virtual machine code which can be interpreted on a range of host machines is one of its unique innovations. In fact, the technique is an old one, by the standards of computing, dating back at least as far as the 1960s.

To execute an applet, a Web browser must first download the virtual machine code generated from the Java source, and then invoke the interpreter on the local machine to run it. The Web browser sets up the appropriate graphical context, on the basis of the OBJECT tag and any stylesheet applied to it, and performs security checks on the applet's code to ensure that it cannot compromise the client machine.

Security is a subject that has been given great consideration in the design and implementation of Java. A description of how the security features are implemented is beyond the scope of this book, but their effect is that applets can be run with different levels of privilege, so that applets downloaded from trusted sources can be permitted to perform operations that are denied to those whose provenance is dubious. In particular, applets that are downloaded from the World Wide Web are usually run under the same restrictions as are applied to scripts embedded in HTML. They may not read or write local files, run programs on the client machine, determine any but the most general information about the machine they are running on, or open network connections to any other machine than the host from which they are downloaded. Applets running under these restrictions are colloquially said to be running in the 'sandbox' (like children playing harmlessly). These restrictions mean that applets running in the sandbox are not much use, but they are absolutely necessary, given the completely open nature of the Internet. The sandbox restrictions are unnecessarily severe for applets that are downloaded from a server on an organization's internal local area network, though. There are mechanisms for determining whether an applet comes from where it purports to, and whether it has been tampered with. This means that it is possible to verify that local applets really are local, and should be allowed out of the sandbox to do useful computation. Unfortunately, with each successive release of Java, well publicized bugs have been discovered in the implementation of the security features, leading to a decline in confidence in their reliability.

By now you may be wondering whether there is any point to applets, since, inside the sandbox, there isn't really anything you can do with an applet that you can't do with a script, stylesheet or plug-in, and outside it you could use an ordinary program. It is certainly true that the case for using Java as a means of enhancing the interface to Web pages is much less compelling now than it was when Java first appeared, since, at that time, there was no client-side scripting, multimedia plug-ins were not widely available, and Web designers had little or no control over typography. Many early applets were intended to overcome these limitations, with animation being particularly popular (and particularly bad). Applets were also the only means of providing interactive Web pages. With these functions now implemented more accessibly by scripts and plug-ins, Java has had to work harder to justify itself.[30]

One argument in favour of applets is that they can perform arbitrary computation, and display graphics and other media, without the need for any plug-in. Thus, they can add new functionality to a Web browser, using no resources beyond the Java Virtual Machine interpreter provided on the host machine. Theoretically, this argument has some force, but in practice browser vendors seem to prefer to add extra functionality to their programs, and users seem content to tolerate the resulting heavy memory demands. Another occasional advantage is that, whereas the source of a script is physically part of a Web page and therefore available for all to see, an applet is download in binary form, so that its workings can be kept secret.

Generally, though, applets, like Xtras, only become worthwhile where complex computation must be carried out and its result integrated dynamically into a multimedia production. Most of the time, useful computation requires cooperation between the client (Web browser) and a server. As we have tried to show, server-side computation alone cannot do much more than enhance the user interface, and this can be done easily and adequately using scripts, as we described earlier. The subject of server-side computation, and communication between server and client will be examined in Chapter 15.

⇨ Some of Java's more vociferous supporters have advocated the use of applets as a means of software distribution — replacing conventional installers — which adumbrates new marketing models, and new types of computer. The idea is that applets will be kept on servers, either a central repository serving all the machines connected to an organization's local area network, or a software vendor's server on the Internet. Anyone wishing to perform some task on their computer will download a suitable applet. If data is also kept on a server, as it usually is within organizations, and conceivably could be on the Internet, the local computer only needs

[30] On the Web, that is. Java applications are subject to different criteria. Their portability and the scope of the available libraries remain significant advantages.

to be able to run a Java interpreter; it does not require local storage (apart, perhaps, from a cache) and can be slimmed down to the bare minimum — a *network computer*. Since users do not keep copies of applications, they can be charged for use, not ownership. The use of trusted servers to hold applets that need to get out of the sandbox, and a solid implementation of the Java security model, are necessary if this mode of distribution is to be useful.

Enthusiasm for network computers peaked around 1997. Within organizations, the concept may yet prove viable, but domestic users currently show no signs of enthusiasm for abandoning their local disks in favour of hiring space on a server, or for downloading an email program every time they need to send a message. However, services such as 'video on demand' may be implemented using similar ideas.

The mechanisms we have described allow computation to be embedded inside a multimedia production, but sometimes it is necessary to write code that manipulates media data directly — somebody has to write the multimedia authoring programs, Web browsers and plug-ins, if nothing else. This does not mean that everything must be done from scratch, though. Most media formats can be manipulated using the objects or functions supplied by an accompanying library and its application programming interface (API). The QuickTime API, for example, which has interfaces to C (or C++) or Java, and has been integrated into Perl[31] includes a wealth of functions for playing, editing, and obtaining information about movies. A simple movie player, with cut and paste editing functions, can be implemented in a few pages of C, much of which is concerned with initialization, event handling, and window management. For Java programmers, the Java 2D and Java 3D APIs provide classes for creating and manipulating two- and three-dimensional graphics, while Java Sound provides audio and MIDI capabilities. The Java Media Framework is a collection of classes that support the capture and playback of synchronized time-based media, both from disk and as streams, and multimedia conferencing. While multimedia programming cannot be simple, APIs such as these enable it to be done at a level of abstraction that conceals much of the problematic detail, allowing multimedia to be relatively easily incorporated into many types of application.

[31] MacPerl only.

Further Information

[Fla97] provides a full and readable description of Web scripting. [ECM97] is the formal definition of the ECMAScript language, and [DOMa] and [DOMb] define the W^3C DOM. [Sma99] is an interesting book about Lingo scripting, concentrating on its object-oriented aspects. If you are a Java expert, [SWDB98] will tell you about the Java Media Framework.

Exercises

Note: Most of these exercises require some scripting ability. If you do not feel confident about writing scripts, see how far you can get using behaviours in an authoring program such as Dreamweaver.

1. Modify the 'Why a duck?' script on page 457 so that it checks for an undefined return value and takes some appropriate action.
2. Why does the HEAD element of an HTML document never have any event handlers attached to it?
3. Construct an HTML page containing two square buttons. Add event handlers, so that, when the cursor is over one of the buttons, the other one changes into a circle.
4. Construct an HTML page that behaves like an animation player. That is, there should be buttons to start, stop, and single-step through an animation. Add event handlers to make these buttons play a simple animation, held as a sequence of individual GIF files on the server.
5. Using CSS, construct a page of HTML that uses complex styling — special fonts, colours, and so on. Add a button to the page which, when clicked, will cause all the styling to be ignored, and the page to be displayed in the browser's default style.
6. How would you make text tumble across the screen in a Web browser?
7. Create an image map in which clickable areas will only be active on, after, or between certain dates (rather like a digital

Advent Calendar, for example). When the clickable areas are activated, a thumbnail image is revealed which may be blown up to full size.

8. Suppose a Web site contains a large number of similar pages, each containing details of a feature film. Implement an interface to this site, using only client-side scripting, that provides the following: a welcome page contains two buttons, one marked 'view by date', the other marked 'view by director'. All the pages for individual films have a button marked 'next' and one marked 'previous'. If the user clicks the 'view by date' button on the welcome page, the page for the earliest film is loaded, and, henceforth the 'next' button takes them to the following film in chronological order, while 'previous' takes them back chronologically. If, however, they click 'view by director', the films are traversed in alphabetical order of their director's name. Assume that the collection of films whose details are recorded on the site is fixed.

9. Implement the mouse training game described on page 474 in Flash. Add the 'go back a level' and 'give up' functions mentioned in the text.

10. Redo exercise 3 as a Flash movie.

11. If you have access to Director, make a movie of a bouncing ball, implemented as suggested on page 482, as a looped one-frame movie. Use the equations of motion to ensure that the ball behaves in accordance with physics (as far as your knowledge of mechanics goes).

12. Give an example of a task that could be performed using a Java applet but not using client-side scripting.

15 Multimedia and Networks

The relationship between computer networks and multimedia is a contradictory one: they are incompatible perfect partners. As the preceding chapters have shown, multimedia places considerable demands on resources: files are often very large, complex processing, such as decompression, may be required, response times must be short, and tight, sometimes complex, synchronization constraints must be respected. Networks are particularly poorly placed to satisfy these demands, because of the inherent limitations of present day technology, and because of the complex and unpredictable effects of the interactions between network components and patterns of network traffic. At the same time, it makes a great deal of sense to maintain multimedia data in a central repository, to be accessed over a network. Wasteful duplication of those same large files is avoided, there is no need for any physical medium, such as a CD-ROM, to be used for distribution, and the approach fits in with the current trend towards distributed systems of desktop machines connected to central servers. Furthermore, the combination of networks and multimedia makes new applications, such as video conferencing and live multimedia presentations, possible.

The considerations in favour of delivering multimedia over networks are generally held to be so compelling as to make it worthwhile living with and trying to overcome the limitations.

Established multimedia authoring environments have been extended to support delivery over networks, especially by adding features to integrate them into the World Wide Web. For example, Director movies can be transformed into *Shockwave* movies, a compressed format which retains all of the media and interactivity of a Director movie, but is optimized for delivery over the Internet. A Shockwave plug-in for Web browsers is freely available, so that all the functionality of Director movies can be provided inside a browser. Some extensions to Director's scripting language Lingo have been made to support URLs and the fetching of data over the network, and Shockwave movies can interact with the browser through scripting and the document object model. This departure from the original model of offline delivery that served Director well during its early years is evidence of a recognition of the growing importance of online delivery of multimedia.

In this chapter we will take a closer look at the mechanisms, which we have so far only glanced at, that are provided to make it possible to deliver multimedia over a network. This material is among the most technical in the book, and you probably won't find it of much interest unless you have some knowledge of how computer systems and networks operate. Readers without this technical background may prefer to skim the opening paragraphs of each section to get some idea of what is involved in meeting the demands of distributed multimedia. Computer scientists and engineers will be able to appreciate why the main thrust of academic and industrial research in multimedia is concentrated on the matters covered in this chapter.

We will concentrate exclusively on TCP/IP networks. Bear in mind that this includes not only the Internet — although that was where these protocols were developed — but also 'intranets': local area networks utilizing the same set of protocols. Because of their generally higher speeds and more centralized management, intranets are better able to support some of the distributed multimedia applications, such as video conferencing, that can be implemented using the real-time protocols we will describe. For the Internet, universal deployment of these applications is still in the future.

Protocols

Protocols are the rules governing the exchange of data over networks. They are conceptually organized into layers, stacked on top of each other, with the lowest layer protocols dealing with the actual physical signals transmitted over wires and optical fibres. Above these, slightly higher level protocols handle the transfer of packets of raw data, and ensure that they reach their correct destination. The highest layers implement more application-oriented services, such as file transfer or Web browsing. The protocols on each layer are implemented using those on the layer below.

When a Web server and client exchange information using HTTP, for example, it appears as though the client sends HTTP requests to the server, which replies with HTTP responses. To the Web software, no other protocols appear to be involved, but in fact, what actually happens is that the HTTP messages are translated into streams of TCP packets, which are themselves translated into the format of the actual networks over which the data must pass; at the receiving end, these packets are used to reconstruct the incoming HTTP message. Thus, we need a little understanding of the lower level TCP/IP protocols before we can appreciate the characteristics of the application-oriented protocols that are used for multimedia.

TCP/IP networks are *packet-switched* networks, which means that all messages transmitted over the network are split into small pieces, called *packets*, which are sent separately. This enables network bandwidth to be shared efficiently between many messages, since, if the rate at which packets are generated for one message is lower than the available carrying capacity, packets belonging to other messages can be transmitted at the same time; we say messages are *multiplexed*. Contrast this with the telephone network, where a connection is established between the caller and the person being called, and this connection is held open for the exclusive use of their conversation until the caller hangs up, whether or not anybody is speaking. The advantages of this *circuit-switched* approach, which can be used for data exchange as well as spoken conversations, is that there is no need to distinguish between packets belonging to different messages on the same circuit — there is only ever one message — and that the two parties can rely on the availability of all the bandwidth provided by their circuit. On a packet-switched network, neither of these is the case: packets must carry additional information identifying (at least) their source and destination, and the rate at which packets can be transmitted between two points will depend on what other traffic the network is carrying. For real-time distributed multimedia applications this last factor is particularly problematical.

Network and Transport Protocols

IP[1] is the *Internet Protocol*, both in the sense that that's what IP stands for, and in the more fundamental sense that it's what makes internets and the Internet possible. IP defines a basic unit of transfer, called a *datagram*, and provides a mechanism for getting datagrams from their source to their destination through a network of networks. The machines that actually exchange messages — the sources and destinations of datagrams — are called *hosts*. Each host will be connected to one of the networks making up the internet. It is identified by its *IP address*, a set of four numbers uniquely identifying the network and host, which is used by IP to send datagrams to the right place.

⇨ IP addresses have a hierarchical structure; the assignment of numbers at the top level of the hierarchy is done by IANA. The organizations to which the top-level addresses are assigned take responsibility for assigning sub-addresses at the next level down, and so on. (ISPs sometimes assign IP addresses to their dial-up customers dynamically.) In this way, the uniqueness of IP addresses is ensured.

Datagrams belong to the category of data structures comprising a header, containing administrative information about the datagram, followed by the actual data. The header contains the source and destination IP addresses, as well as some additional information concerning routing and optionally security. The detailed layout of an IP datagram depends upon the version of IP being used. IPv4 is the 'classical' IP that has been in use for many years; IPv6 is a newer, more flexible, version. An important difference is that, whereas IPv4 addresses are 32 bits long, IPv6 addresses are 128 bits long, so the new version can accommodate many more hosts and networks. A datagram may be made up out of several packets on the underlying physical network that is carrying it.

[1] It's impossible to talk about networks without using sets of initials and acronyms — that's how network people talk. We've tried to keep them to a minimum, and will always tell you what they stand for.

As well as hosts, an internet includes machines called *routers*, connected between its component networks, which maintain information about the network topology so that they can work out where to send a datagram to. In general, an IP datagram may have to pass through several routers during its travels over several networks. By inspecting the destination address in a datagram's header, a router can determine whether or not the datagram should be sent to a host connected to the same network as it is. If it should, the router translates the IP address to the native address format of the network, if necessary, and sends the data to its destination. Otherwise, the datagram is passed on to another router. Dynamically updated tables are used to determine which of the accessible routers a datagram should be forwarded to.

IP treats each datagram individually, recognizing no connection between the datagrams that make up a Web page, for example. Furthermore, it cannot identify the application that generated a piece of data, nor the one that should receive it. All that IP does is attempt to deliver individual datagrams from one host to another. It doesn't even guarantee to succeed. In particular, if some datagram has failed to reach its destination after a certain length of time,[2] it will be discarded. This is more efficient than continuing to try to deliver it — its destination is probably unreachable for some reason — and has the beneficial side-effect of preventing rogue datagrams going round and round forever. It follows, though, that if an application sends a stream of data, it may arrive at its destination with some pieces missing. It is also possible for some pieces to arrive in the wrong order. Since IP treats each datagram individually, and calculates routes dynamically, it is possible for a packet to overtake others sent before it. For example, a router that was down, forcing data to be sent round by a long route, may come back on line, and later packets will take a shorter, faster route to their destination, arriving earlier than packets sent before them.

The majority of applications that communicate over TCP/IP networks require the data they send to arrive intact and in the right order at the receiving end. Consider an email system, for example. If we send you a message, you won't consider it very satisfactory if it arrives in your mailbox with key passages missing, and others in the wrong order. If applications had to rely on IP alone, this could happen, so, to provide the required service, the application would have to implement some mechanism for putting packets back in order, and requesting the retransmission of any that are

[2] In practice, after it has passed through a certain number of routers.

missing. It would be unreasonable to expect every application to implement these functions; instead, a reliable transport protocol, *TCP (Transmission Control Protocol)*, is layered on top of IP.

TCP provides reliable delivery of sequenced packets. It does this using a system of acknowledgement. A simplified version of the protocols would work as follows. When the destination receives a packet, it sends an acknowledgement, and the sender does not transmit another packet until it gets the acknowledgement. If the acknowledgement is not received within a specified period of time (the *time-out*), the packet is sent again. The algorithm as described is prohibitively slow, and TCP actually uses a more sophisticated technique, based on the same principle, using a 'sliding window' of unacknowledged packets. Instead of sending a single packet and waiting for an acknowledgement, it sends several, let's say eight for concreteness. As each packet is received, it is acknowledged; because of the time packets and acknowledgements take to traverse the network, the acknowledgement of the first packet will probably not reach the sender until several others have been sent. That is, there may be several unacknowledged packets in transit at one time. If the acknowledgement of the first packet has not been received by the time the eighth has been sent, the sender waits, and retransmits, as before, if the acknowledgement does not come in time. On the other hand, once the acknowledgement of the first packet is received, the limit on the number of packets to send before waiting is incremented, so the sender will send up to the ninth packet before waiting for the acknowledgement of the second, no matter at which point in the transmission of the first eight the acknowledgment arrives. Once the second acknowledgment is in, the limit is advanced to packet number ten, and so on (see Figure 15.1).

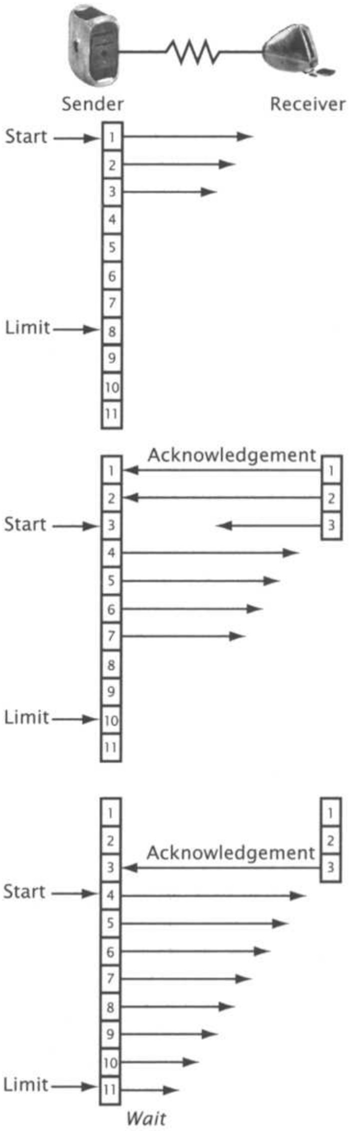


Figure 15.1
Sliding window acknowledgement

Notice that retransmitted packets may arrive out of sequence, and also that the possibility exists for packets to be sent twice, if an acknowledgment goes astray, for example, or a packet turns up long after it should. TCP thus introduces new opportunities for packets to arrive out of sequence. However, the finite number of packets that is allowed to be transmitted without acknowledgment imposes an upper bound on how far out of sequence a packet can be.

An analogy that is often used to describe the difference between IP and TCP concerns pouring water on to a burning building. IP is like a bucket brigade: water (data) is is poured out of a hydrant into buckets (datagrams) that are carried by a disorganized crowd of volunteers (the network). A bucket may be carried some distance, then passed from hand to hand to the next volunteer. Some buckets will get dropped in the process; others will work their way through the crowd at different speeds, so that the order in which they arrive at their destination bears little resemblance to the order in which they are filled up and sent on their way. In contrast, TCP is like a firehose that carries the water straight from the hydrant to the fire; the water just goes in one end and comes out the other, in an orderly stream, without loss.

The analogy is not perfect. A firehose is a much more efficient way of transporting water than a crowd of bucket carriers, but TCP is less efficient than IP, because it is actually implemented using IP, but then adds extra overheads with its system of acknowledgments and retransmission.

Besides the overhead incurred by acknowledgement, TCP also incurs a cost in setting up a connection. This is necessary, in order for the acknowledgements to work, and in order to ensure that packets are sent to the appropriate application. Recall that IP addresses only identify hosts, and that IP accordingly only attempts to transport data between hosts. In order to identify a particular application running on a host, the IP address must be extended with a number, called a *port number*. Ports are associated with applications according to the protocol they use. For example, programs communicating via HTTP conventionally use port 80. The combination of an IP address and a port number is called a *transport address*, and enables the protocols running on top of IP to provide communication facilities between programs running on different hosts. TCP uses a pair of transport addresses to set up a connection between two programs, and subsequently these can communicate as if there was a reliable data conduit between them.

For some networked multimedia applications, the possibility of lost packets is more acceptable than the overhead of TCP. Streamed video and audio are the prime examples. Suppose, for example, a speech is being transmitted live over a network. If the video data were sent using TCP, then every frame would arrive complete and in the correct order, but the rate at which they would arrive would be unpredictable. Jerks and pauses might occur as packets were retransmitted and acknowledgements were awaited. Almost inevitably, the display would fall behind the actual delivery of the speech. The transient glitches caused by the occassional loss of a frame or fragment of audio would be less intrusive, if they permitted an otherwise steady frame rate to be maintained instead. An alternative to TCP is needed to transport data with these characteristics.

The *User Datagram Protocol (UDP)* is built on top of IP, like TCP, but is much simpler, doing little more than ensure that datagrams are passed to the correct application when they arrive at their destination host. Like IP, UDP only tries its best to deliver datagrams, it does not offer the reliable delivery provided by TCP. Nor does it set up connections: port numbers are included in the source and destination fields of every UDP packet to ensure data finds its way to the right place. UDP does perform some elementary error checking that is not provided by IP, to help ensure that datagrams are not corrupted.[3] These features of UDP make it a suitable basis for building protocols for delivering data such as streamed video and audio, for which real-time constraints are more important than totally reliable delivery.

The low-cost delivery of UDP is not enough, on its own, for such purposes. The *Real-Time Transport Protocol (RTP)* typically runs on top of UDP, adding extra features that are needed for synchronization, sequencing, and identifying different types of data — or *payloads* as they are called in this context. RTP itself still does not guarantee delivery and it does not prevent packets arriving in the wrong order, but it does enable applications to reconstruct a sequence of packets and detect whether any are missing. It does this by including a sequence number in the header of each packet belonging to a particular stream — RTP sets up connections between applications, so that the stream of data belonging to a particular connection is an identifiable entity. Each time a packet is sent over a connection, the sequence number is incremented; thus, it is clear to the receiver what order packets should be in. Depending on its requirements and the way in which it treats data once it has arrived, a receiving application can reconstruct the correct sequence, discard packets that are received out of sequence, or insert them in their correct places in some data structure being built from them.

An RTP packet's header also identifies the payload type — whether it is video, audio, and so on — which determines, by implication, the format of the data contained in the rest of the packet. This allows different payload types to employ formats that are optimized for their special characteristics; in particular, it allows appropriate forms of compression to be applied to images, video, and sound. Where several different media types are being transmitted —

[3] We might say UDP notices if water sloshes out of the buckets, extending the analogy we used earlier.

typically video with accompanying sound — they must be sent over separate RTP streams with different payload types. It is therefore necessary to synchronize them when they are received. A timestamp is included in the header for this purpose. It records the instant at which the first byte contained in a packet was sampled. This can be collated with the timestamps in other, related, streams, to ensure that simultaneously sampled data is played back at the same instant.

RTP can be tailored to different applications' requirements and extended to cope with new types of payload — it is a framework for an entire family of protocols, rather than a protocol in the traditional sense. RTP *profiles* define subsets of the protocol's features that are suitable for particular applications. In particular, the audio-visual profile is intended for use with audio and video streams.

➲ A complementary protocol, the *RTP Control Protocol (RTCP)* can be used in conjunction with RTP to provide feedback on the quality of the data delivery — statistics such as the number of packets lost and the variance of the time between packets' arrival. This data can be used by the application sending the data, to adjust the rate at which it does so, for example.

Multicasting

Previously, when we considered bandwidth limitations and how they affect online delivery of multimedia, we concentrated on the speed of users' connections to the Internet, via modems or cable and so on. These connections are usually the slowest link in the communication chain, but the high-speed networks that provide the skeleton of the Internet are also of finite bandwidth. Here, too, new technologies are leading to increased speeds, but traffic is also growing at a considerable rate, and new types of traffic — especially that associated with bulky time-based media such as video — are adding to the demands placed on the Internet as a whole. Speeding up the network is one way of coping with these demands; reducing the volume of traffic by eliminating duplication is another.

A common situation where data is unnecessarily duplicated arises when a group of Internet users require access to the same resource at the same time. This situation is often associated with multimedia.

Suppose, for example, a well-known musician — let's call him Tim Linkinwater — has arranged to transmit a live video feed from a concert over the Internet. Every Tim Linkinwater fan with Internet access will want to watch this Webcast. The data being sent to every fan is identical, but conventional (*unicast*) transmission will require that the server from which the video is being streamed send a copy of it to everybody who has set up a connection to watch this concert. If the concert is taking place in New York, but Tim Linkinwater is big in Europe, many copies of the video data will be sent across the Atlantic, putting a strain on both the transatlantic links and the video server. This strain could be reduced by sending a single copy over the Atlantic and not duplicating it until it became necessary to do so in order to distribute the data to all the people who wanted it.

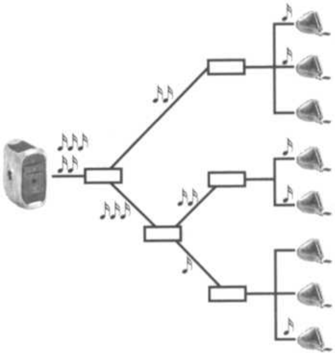


Figure 15.2
Unicast transmission

The scenario just outlined is the basis of *multicasting*. Figures 15.2 and 15.3 illustrate the difference between unicasting and multicasting. In unicasting, a separate packet is sent to each user; in multicasting, a single packet is sent, and is duplicated along the way whenever routes to different users diverge. For this to be possible, hosts must be assigned to *host groups*, with certain network addresses identifying groups instead of single hosts. A range of IP addresses is reserved for this purpose. From the sender's point of view a multicast address behaves rather like a mailing list — although mailing lists are implemented quite differently (using unicast). One difference is that the sender does not know who belongs to the host group, if anybody.

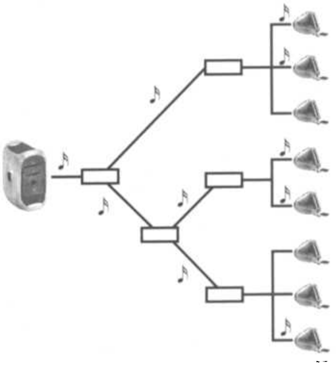


Figure 15.3
Multicast transmission

A packet that is intended to go to a group is sent to the appropriate multicast address. Routers must be enhanced to cope with multicast addresses, duplicating packets where necessary. They must also be able to perform the routing calculations necessary to determine optimal routes — which may be different from optimal unicast routes. These technical issues, while difficult, are amenable to technical solutions. Other problems that arise when multicasting is attempted on the scale of the Internet are less straightforward.

⇨ On a LAN, there are no routers, so multicasting is simpler. Multicast packets will go to the network interface on every host. Each host just has to look at the multicast address and determine whether it belongs to the corresponding host group; if it does, it accepts the packet, in the same way as it accepts a packet sent to its individual address. Multicast applications are therefore more common on LANs than the Internet, at present.

The main problems concern management of host groups. These will typically be spread across different networks, so the information about which hosts belong to a group must be distributed to routers. Group membership has to be dynamic, to accommodate applications such as the Webcast described earlier, so a mechanism is required for a host to join a group, and subsequently to leave it. Routers must then be able to adjust to the new state of group membership. At a still higher level, some mechanism is needed for users to find out about the existence of host groups, what their multicast addresses are, and how to join them. (Again, compare this with discovering mailing lists.) Essentially, when a multicast session is set up, an address must be chosen from the available range, and then advertised — by email, newsgroup, Web site, or a dedicated application that serves the function of a listings magazine.

Many distributed multimedia applications, including live video streaming, and conferencing, have characteristics that make them suitable for multicasting. Protocols such as RTP can run on top of IP multicast.[4] Considerable experience of doing so has been gained using an experimental sub-network of the Internet, known as the *MBone (Multicast Backbone)*, and multicast capabilities are now being provided in routers as a matter of course. ISPs are beginning to offer multicast services, but this emergence of multicast into the world of the commercial Internet raises new questions. Above all, how should multicasting be charged for?

⇨ There is a third alternative to both unicast and multicast transmission: *broadcast*. A broadcast transmission is sent to every host on the network, and it is up to the host to decide whether or not to ignore it. This is very efficient for the sender, and it may be an efficient way of using network bandwidth, but for most purposes it imposes an unacceptable load on hosts, since it requires them to look at every packet sent to a broadcast address and determine whether it is interesting. Broadcasting is used on the Internet for monitoring the status of the network and for maintaining routing information, amongst other things.

Application Protocols for Multimedia

The network and transport protocols we have described do no more than deliver packets of data, more or less reliably, to their

[4] TCP cannot, however.

designated destinations. Higher level protocols must run on top of them in order to provide services suitable for distributed multimedia applications. We will describe two such protocols: HTTP, which is the basis of the World Wide Web, and RTSP, a newer protocol designed to control streamed media. Our description is not exhaustive, but is intended to show what is involved in mapping the requirements of some kinds of distributed multimedia applications on to the transport facilities provided by the protocols described in preceding sections.

HTTP

Interaction between a Web client and server over HTTP takes the form of a disciplined conversation, with the client sending requests, which are met by responses from the server. The conversation is begun by the client — it is a basic property of the client/server model that servers do nothing but listen for requests, unless they are asked to. To start things off, the client opens a TCP connection to a server. The identity of the server is usually extracted from a URL. As we saw in Chapter 9, the URL contains a domain name; TCP only works with numeric IP addresses, though, so the name must be translated into an address. This translation is done transparently, as far as HTTP is concerned, by DNS (Domain Naming Services), so HTTP can always work in terms of names (although numeric addresses are occasionally used instead). By default, Web servers listen to port 80, so this will normally be the port number used when the connection is opened.

➲ Originally, prior to version 1.1 of HTTP, a connection was used for each request and its response only: that is, the client opened a TCP connection and sent one request; the server sent a response to that request and closed the connection. This way of using connections is very efficient from the server's point of view: as soon as it has sent the response and closed the connection it can forget about the transaction and get on with something else, without having to keep the connection open and wait for further requests to come over it, or close the connection if it times out. The disadvantage is that accesses to Web servers tend to come in clusters. If a page contains ten images most browsers will make eleven requests in rapid succession — one for the page itself and one for each image — so eleven connections between the same pair of hosts must be opened and closed. For technical reasons, to help minimize lost packets, a TCP/IP connection starts out at a slow rate and gradually

> works up to the best speed it can attain with acceptable losses in the existing state of the network. Hence, in our example, the data for the page and its images will not be sent as fast over the eleven connections as it would over a single connection, since each new connection has to start over and work up to speed. HTTP version 1.1 has therefore introduced the possibility of a persistent connection, which must be explicitly closed by the client (although the server will close it after a specified time has elapsed without requests). Nevertheless, the structure of an HTTP session retains the form of a sequence of requests that evoke responses. No state information is retained in the server, which is therefore unaware of any logical connections between any requests.

HTTP requests and responses are collectively known as messages. Both consist of a string of 8-bit characters, so they can be treated as text by programs that read them (and can be read by humans if they have a program that can eavesdrop on HTTP). Messages conform to a simple rigid structure, consisting of an initial line (the *request line* for a request, the *status line* for a response) containing the essential message, followed by one or more *headers*, containing various parameters and modifiers. These may be followed by the message *body*, which contains data, such as the contents of a file being sent by the server, if there is any. Headers are separated from the data by a blank line.[5]

A request line comprises three elements. The *method* is a name identifying the service being requested; the most commonly used method is GET, which is used to request a file or other resource (we will not consider other methods in any detail). The *identifier* comes next, and tells the server which resource is being requested, for example by giving the path name of a file. Finally, the HTTP *version* indicates which protocol version the client is using. For example, if a user clicked on a link with URL `http://www.wiley.com/compbooks/chapman/index.html`, their Web browser would connect to the host with name `www.wiley.com`, using port 80 so that communication would go to the Web server on that machine. It would then send an HTTP request, whose request line was:

```
GET /compbooks/chapman/index.html HTTP/1.1
```

The headers that may follow a request line all take the form of a header name followed by a colon and some arguments. For example, the following two headers can be found after the request line shown above:

[5] On the Internet, the combination of a carriage return followed by a linefeed is defined as the line terminator; this is not the same convention as all systems use, so HTTP clients and servers must take care with line termination. Not all do so.

```
Host: www.wiley.com
User-Agent: Mozilla/4.0 (compatible; MSIE 4.5; Mac_PowerPC)
```

The Host header tells the server the host name that the request is directed at;[6] User-Agent identifies the Web browser (or other user agent) that is making the request. This will allow the server to make allowances for any known problems with this particular browser, if it chooses to.

One of the most commonly seen headers in GET requests is Accept. Its arguments indicate, using MIME content types, the range of types of data that the browser can deal with. For example,

```
Accept: image/gif, image/x-xbitmap, image/jpeg
```

is sent by a browser to indicate that it is capable of and willing to display GIF, JPEG and X bitmap images. Browsers may send several Accept headers instead of combining them. Web servers will only send them data that is of an acceptable type. In a MIME type used in this way, a * can be used as a wildcard character, so

```
Accept: text/*
```

indicates that the browser will accept any text format.

The value specified in an Accept header can be modified by two qualifiers. The first is a 'quality factor', q, which expresses the desirability of a type, using a real number between zero and one. Attaching different quality factors to the sub-types of the same MIME type allows a Web client to express a preference among different formats. For example,

```
Accept: image/png;q=0.8
Accept: image/jpeg;q=0.6
```

indicates that PNG files are preferred over JPEGs. The second modifer, mxb, signifies the maximum number of bytes that will be accepted in files of the designated type. If a browser only preferred PNG files over JPEGs if they were not too big, the following headers could be used:

```
Accept: image/png;q=0.8;mxb=30000
Accept: image/jpeg;q=0.6
```

As the examples show, the modifiers are separated from the type and from each other by semi-colons.

Despite the potential usefulness of this header in allowing servers to offer alternative versions of images and other content, and negotiate with clients to choose the most appropriate, most browsers appear to send

[6] Sometimes, several host names correspond to a single IP address, and this might make request identifiers ambiguous.

```
Accept: */*
```

among their headers by default.

Two similar headers, Accept-Charset and Accept-Language are used by browsers to inform servers about the character sets and languages they will accept. The following would be typical of the headers sent by a browser set up in an English-speaking country:

```
Accept-Charset: iso-8859-1
Accept-Language: en
```

Since a GET request does not send any data, its body is empty — the message terminates with the blank line.

The first line of an HTTP server's response is the status line, indicating how it coped with the request. This line begins with the protocol version, telling the client which HTTP version the server is using. Next comes a numerical status code, whose meaning is defined by the HTTP standard, followed by a short phrase, explaining to human readers what the code means. If all goes well, the code will be 200, which means OK, as shown:

```
HTTP/1.1 200 OK
```

Just as the client told the server who it was, in the User-Agent header, the server introduces itself in the Server header. It also dates and times the response, and, as we saw in Chapter 9, uses the Content-type header to inform the client about the MIME type of the data being returned. For example:

```
Server: Netscape-Enterprise/3.5.1G
Date: Sat, 17 Jul 1999 14:27:17 GMT
Content-type: text/html
```

We will describe some other noteworthy headers shortly.

Typically, a server's response does contain some data; in the case of a response to a GET request, this will be the contents of the file that was requested, for example.

➲ Where a persistent connection is being used, the client must be able to determine when the end of the data has been reached — when servers closed the connection immediately after sending the response this was not a problem, because data obviously finished when the connection closed. Two mechanisms are now provided. Either a Content-length header can be sent, or the response can be broken into 'chunks', each of which is self-delimiting. For the details, consult the HTTP 1.1 standard.

The status code in a response is not always 200, unfortunately. The HTTP 1.1 standard defines many 3-digit return codes. These are divided into groups, by their first digit. Codes less than 200 are informative; the only such codes presently defined are 100, which means 'continuing', and is only applicable to persistent connections (see the technical asides earlier in this section), and 101, sent when the server switches to a different protocol, for example to stream some real-time data. Codes in the range 200–299 indicate success of one sort or another. Those in the three hundreds are used when the server must redirect the request to a different URL. For example, a code of 301 ('moved permanently') indicates that the resource identified by the URL in a GET request has been permanently moved to a new location (the server sends a Location header giving the new URL).

Codes that begin with a 4 or a 5 represent errors, by the client and server, respectively. Probably the two most commonly encountered error codes are 400 and 404. 400 means 'bad request' — the request was malformed in some way. This error usually occurs when requests are manufactured by programs other than Web browsers. Error code 404 is 'not found' — the URL in a GET request does not correspond to any resource on the server. This is the code that results from broken links. Other error codes in this range correspond to requests for services that are denied by the server, such as attempts to access protected files. Another interesting code is 406 ('not acceptable'), which is sent if the requested resource's MIME type is not listed in the Accept headers of the request. Server errors include 500 ('internal server error'), which should never happen, and 501 ('not implemented'), which is sent when the server does not know how to implement a requested method.

⇨ Where confidential information is being transmitted between a Web client and server — credit card details being supplied to an online store being the most prominent example — the transactions must be secure. Mechanisms have been added to HTTP for guaranteeing that a server really is what it claims to be, and for encrypting all the data sent between the server and client.

Caching

The World Wide Web has made information accessible over the Internet in a way that it had never been previously. This has led to an increase in network traffic as people take advantage of this new-found accessibility. In order to help alleviate the strain this could cause on the network, HTTP provides support for a system of *caching*, which allows copies of pages that have been received to be kept on a user's machine, and on other intermediate machines between it and the server from which they originate. When a page is requested more than once, it can be retrieved after the first time from a cache instead of from the server. If the cache is on the local machine, this may not require any network access or action by the server at all.

The full extent of HTTP 1.1's support for caching is quite extensive, supplying several alternative mechanisms; we will only describe a subset.

The trouble with caching is that a version of the requested resource that is newer than the version in the cache might have appeared on the server. In that case, the new version should be retrieved, and the old one in the cache should be discarded. The simplest way for the client to find out whether that is the case is to ask. It does so by sending an `If-Modified-Since` header, giving the date and time of its cached copy (which it knows from the `Date` header in the response to its original request). The request is then said to be *conditional*, and the server only sends the requested page if the condition is satisfied, that is, if the page has been modified since the date specified in the header. If it has, the server sends a response containing the modified page, as before. If not, it sends a response with status code 304, which means 'not modified'. On receiving this code, the browser displays the page from its cache.

Conditional requests of this sort eliminate the need for servers to send complete responses in all cases, thus reducing the volume of data transmitted. A further facility potentially eliminates the need for servers to send any response, by eliminating the need for clients to send requests. A server can include an `Expires` header in its response to a GET request, indicating the date and time after which the data it returns should no longer be considered up-to-date. Until that time, the client is free to use a cached copy of the data to satisfy any subsequent requests for the same resource. Thus no network activity at all is required to obtain the same page until it reaches its expiry date.

While this mechanism is sound, it begs the question of how servers are to assign expiry dates to arbitrary pages. Most of them appear not to try, although one can imagine tightly controlled environments in which it would be known how long a page would remain valid.[7] In the absence of any widespread use of `Expires` headers, Web clients fall back on their own *ad hoc* devices for avoiding unnecessary network accesses. For example, Internet Explorer offers an offline browsing mode, in which all requests are met from the local cache, if possible. This allows previously visited Web sites to be accessed without the user even being connected to the Internet.

So far, we have assumed that a cache is maintained on a user's machine. There are other places that data can be cached, though, sometimes more effectively. In particular, *Web proxies* usually maintain large caches. A proxy is a machine that handles requests directed to some other server. So when a client that is configured to use a proxy sends a request, the request is sent to the proxy, which then forwards it to its designated destination, receives the response and passes it back to the client. This apparently pointless exercise is often needed when clients are operating behind a 'firewall' — a specially modified router that filters packets to provide additional security. Firewalls prevent users inside them from making HTTP requests, so a proxy is used, which has access to the outside world and the protected machines. Security can be added to the proxy to prevent unauthorized access across the firewall, in either direction. This implies that all responses to requests sent by any machine inside the firewall pass through the proxy, which means that a cache on the proxy will end up holding a pool of data that is of interest to people in the organization maintaining the firewall. This is likely to mean that many requests can be met from the proxy's cache.

Proxies can provide effective caches whenever they are placed somewhere that many requests pass through. Large ISPs, in particular, can employ proxies with extremely large caches to intercept a large proportion of their customers' requests.

Caches cannot help with pages that are dynamically generated, as an increasing number are, in ways that we will describe briefly in a later section. Nor do they help users who never visit the same page twice unless it has been updated. And caches are finite, so that, sooner or later, cached copies of pages must be discarded, and retrieved again if they are requested subsequently. Nevertheless, they are widely considered to be making a useful contribution to keeping network traffic within bounds.

[7] The HTTP standard suggests the use of heuristics based on, for example, the `Last-modified` header, but does not specify any detailed algorithms. It recommends that heuristics be applied with caution.

RTSP

HTTP provides the services necessary for implementing a distributed hypertext system: it should be easy to see how HTTP requests can be constructed in response to mouse-clicks on links, and how a browser displaying the data in their responses can provide the experience of following a link to a new page. HTTP can cope with embedded images, sounds, and video, by downloading a complete data file. Streaming audio and video require a different treatment, though.

In the first place, HTTP runs on top of TCP, and, as we explained earlier, the overheads incurred by the reliable delivery that TCP provides are unacceptable for streamed media, which are better served by a less reliable but more efficient protocol. Its use of TCP also makes HTTP unsuitable for multicasting. RTP, as described earlier, is adequate for carrying streaming media data, and can be used for multicasting, but it does not provide all the necessary functionality required by streams of such data. We usually want to be able to start, stop, and pause them, and possibly (for streams that are not being transmitted live) go to a particular point in the stream, identified by a timecode. For live streams, we might want to schedule a time at which to start the display; for example, if a concert was being transmitted over the Web, we might want to skip the support band and start listening when the headline act was due to go on stage.

The *Real Time Streaming Protocol (RTSP)* is intended to provide these services. It is often described as an 'Internet VCR remote control protocol', which conveys much of the flavour of what it does. Syntactically, it closely resembles HTTP, with request and status lines and headers, many of which are the same as in HTTP, but there are some differences. In particular, RTSP uses ISO 10646 (with UTF-8 encoding) as its character set, unlike HTTP which is restricted to ASCII in its messages.[8] In RTSP requests that include an identifier, an absolute URL must be used, instead of the pathname that may be used in HTTP; RTSP consequently does not need a separate `Host` header. Most importantly, RTSP responses do not carry the media data, the way that HTTP responses carry Web page data. Instead, the data is transmitted separately, often using RTP; RTSP merely coordinates the transmission.

[8] But not its data, of course.

Before an RTSP session can be set up, the client must obtain a *presentation description*, which contains information about the data streams that are to be controlled, which together make up a multimedia presentation. The RTSP specification does not stipulate the format of a presentation description; the *Session Description Protocol (SDP)*'s format is commonly used, but the prevailing applications using RTSP (Streaming QuickTime and RealPlayer G2) take advantage of that format's extensibility to customize it to their own requirements, so, in effect, presentation descriptions largely conform to proprietary formats.[9] A typical description will include one or more *media announcements*, including the transport address and protocol (e.g. RTP) of one of the session's component streams, and information about the encoding used, including the type of compression, if any. Each stream will have an `rtsp://` URL. It is quite possible for different streams to be served from different hosts. The presentation description will also provide a *connection address*, to which subsequent requests are addressed. This is necessary because RTSP does not specify the mechanism whereby presentation descriptions are to be retrieved. There is a `DESCRIBE` request that is often used for this purpose, but it is permissible for the presentation description to be retrieved using HTTP, or even sent to a user by email.

In practice, both RealPlayer and Streaming QuickTime use `DESCRIBE` requests, but need a bootstrapping step before they can find out where to send it to. Typically, HTTP is used to obtain the URL of the presentation description; this will begin with `rtsp://`, to indicate that it can be accessed using RTSP. QuickTime uses a self-contained approach: a small movie containing URLs for different versions of a session corresponding to different connection speeds is presented as the representation of a streamed movie[10] — for example, it might be embedded in a Web page, where a user can click on it, causing it to be retrieved by HTTP. RealPlayer uses a small file with a distinctive extension for the same purpose, although it does not provide for different versions.

Once the necessary URLs have been obtained by the client, it sends a `DESCRIBE` request, to which the server responds with a message containing the required presentation description. The client's request can include an `ACCEPT` header, indicating the description formats it understands. Once it has a presentation description, the client can send a `SETUP` request. The server responds with a message that includes a *session identifier*, an arbitrary string that

[9] These are fairly readily comprehensible, though.

[10] This is just a special case of the use of a master movie to orchestrate a set of alternatives, as described in Chapter 10.

is used by both client and server to identify messages associated with the same session. (RTSP does not use connections, the way TCP does, partly because the actual streamed data is sent over a separate transport protocol, and partly to enable it to be used for controlling multicast streams.) Everything is then in place for the streaming to begin.

⇨ A significant difference between the way QuickTime and RealPlayer G2 use RTSP lies in their approach to presentations comprising several streams, such as a video stream and accompanying audio. QuickTime treats these as separate tracks of a single movie (although they will be transmitted separately), so the presentation description provides details of every stream. RealPlayer G2 is a SMIL player (see Chapter 13), and uses SMIL to coordinate the separate streams. The presentation description it originally retrieves only describes a single SMIL document; this contains `rtsp://` URLs for each of the streams in the presentation, and the player sets up separate RTSP sessions for each.

In the simplest mode of operation, an RTSP client sends `PLAY` requests to cause the server to begin sending data, and `PAUSE` requests to temporarily halt it. As we mentioned earlier, a `PLAY` request may include a header specifying a range within the duration of the stream; SMPTE timecodes or clock values in a standard format may be used for this purpose. A session is ended by the client sending a `TEARDOWN` request, which causes the server to deallocate any resources associated with the session. It is, of course, conceivable that something might go wrong. Status codes are used in responses to indicate any errors that might occur, just as they are in HTTP. Indeed, wherever it makes sense, RTSP uses the same status codes as HTTP.

We emphasise that RTSP messages travel separately from the streamed data they relate to. Usually, RTSP will operate on top of TCP, so that it can rely on delivery of its messages, although it can as easily run on top of UDP, where efficiency is important. For the streamed data, there is no such choice. As we explained previously, TCP is not suitable for carrying the streamed data itself, and this will usually be carried over RTP, which in turn runs on top of UDP. Nevertheless, there is always a logical connection between the streamed data and the RTSP messages. For example, RTSP supports the same headers for controlling caching as HTTP does, but, although the headers are transmitted as part of RTSP messages, it is the streamed data that is cached, not the messages.

RTSP can be used for unicast and multicast streams. Where unicast is used, it provides services such as video on demand: a client requests, say, a movie to be streamed from a server, and the data is sent exclusively to the client who controls its playback with RTSP requests. Multicasting provides services more like conventional television broadcast: a client joins a multicast session that has been set up beforehand, to transmit a concert, for example. Where RTSP is being used with a multicast service, the main difference from the client's point of view is that the server has to tell it the multicast address for the transmission, whereas, in the unicast case, it is the client that has to tell the server where to direct the data stream to.

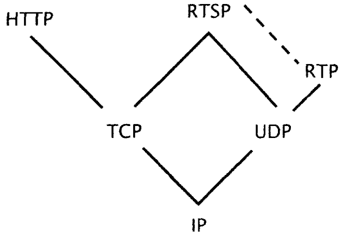


Figure 15.4
Relationships between TCP/IP protocols used for multimedia

Figure 15.4 summarizes the relationships between the protocols we have described. Video conferencing and similar applications require additional protocols for session management and the equivalent of rules of procedure. We will not describe the protocols presently available for these purposes, since they are still largely experimental, and take us too far away from our main concerns.

Quality of Service

Normally, when data is being transmitted over a packet-switched network, it is subject to the vagaries of network performance, which depends on many factors, most of them unpredictable. This can interfere with the satisfactory streaming of media in several ways. First, *delays* can be introduced. There is always some delay due to the distance signals must travel at a finite speed; the operations that must be performed by routers contribute additional delays. Whereas traditional network applications, such as email and file transfer can tolerate considerable delays, real-time and streamed multimedia are much less tolerant. You are probably familiar with the slightly disconcerting effect of the transmission delays that occur when live interviews are conducted over transatlantic links for television news bulletins. The effect of similar, or longer, delays on a real-time application, such as a video conference, can seriously interfere with its effectiveness. Streamed applications can tolerate longer delays, although, the appeal of live streaming often lies in its immediacy, which is compromised by excessive delays.

Typically, the delay experienced by a particular stream of data will not be constant, but will vary continuously. The variation in delay is called *jitter*; it means that the time between the arrival of successive packets will vary. Jitter causes two kinds of problem for multimedia. First, the variation in the time between packets can result in time base errors; that is, samples will be played back at the wrong time. This is most likely to affect audio streams, where it manifests itself as noise. Second, where a presentation comprises several streams that are transmitted independently, as they will be over RTP, for example, the streams may jitter independently, resulting in a loss of synchronization between them (see Figure 15.5). Unless this can be corrected at the receiving end, disturbing — or unintentionally comical — effects may occur, since people are sensitive to tiny errors in lip-sync. As with delay, different applications can tolerate different amounts of jitter. For example, a video application might buffer incoming data streams, so that it could smooth out jitter and synchronize video and audio streams, provided the jitter was within certain limits.

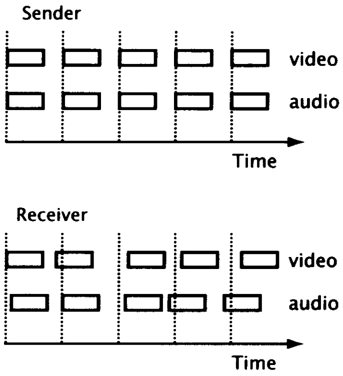


Figure 15.5
Effect of delay jitter on synchronization

In addition to delay and jitter, there may be packet loss, since, as we explained earlier, streamed media cannot be sensibly transported using a reliable protocol such as TCP. Once again, different applications can tolerate different amounts of packet loss.

Delay, jitter, and packet loss are measurable quantities. We can, therefore, quantify the amounts of each which a particular application can tolerate. The values of these parameters, together with the bandwidth it needs, define the *quality of service (QoS)* required by an application.

Established types of network service do not require much in the way of quality of service. Mostly, they are based on the transfer of entire files, such as a mail message or the HTML and embedded images of a Web page. They can tolerate as much delay as users' patience allows, and jitter is irrelevant, since nothing will be done with the data until it has all arrived. However, it is generally the case that packet loss is completely unacceptable. These characteristics led to the development of transport protocols like TCP that are reliable but pay no other attention to the requirements of applications. On the contrary, their performance is driven by the availability of bandwidth, and their objective is to make the best use of that bandwidth — which is how delay and jitter are introduced, since the bandwidth available to any one connection is not constant or guaranteed. Integrating support for streamed and real-time applications with their more demanding requirements for quality of service is thus quite a challenge.

New high-speed network technologies are coming into use, which are able to reserve resources for individual connections. The best known of these is *ATM*, which stands for *Asynchronous Transfer Mode.* A pure ATM network can offer guarantees about the quality of service it will provide. Applications can reserve resources to satisfy their requirements, and the network will honour those reservations, so that the application can be assured that delay, jitter and packet loss will be kept within tolerable limits. In order to maintain the quality of service to some applications, the network may have to restrict the resources it allocates to others, perhaps even refusing to accept any new traffic after a certain point. The network is thus behaving much more like a circuit-switched network, such as the telephone system, than an ordinary packet-switched network.

Making an open heterogeneous network of networks, such as the Internet, behave in the same way, while still providing the traditional services that users have come to expect, is problematical. Parts of the Internet are presently ATM networks, but other parts are not, so the quality of service support has to be implemented at a higher level, by way of protocols. A framework known as the *Integrated Services Architecture* has been developed for this purpose.[11] It defines different classes of service, such as 'best effort', which is the traditional IP approach to delivery, and 'guaranteed', where an absolute upper bound is imposed on the delay of packets. Several mechanisms for controlling network traffic to ensure that specified classes of service are actually provided are defined, together with a protocol for reserving resources, *RSVP (Resource Reservation Protocol.* Some effort has been made to ensure that the facilities of the Integrated Services Architecture work with multicasting, since it is expected that many applications, such as video conferencing and live video streaming, requiring guaranteed quality of service will employ multicasting.

⇨ One thing seems fairly certain: when ISPs start providing guarantees about quality of service, they will make additional charges for them. It is quite likely, therefore, that streamed video with guaranteed performance will not be something that can be provided for everybody. More likely, when (or if) the Internet can provide such services, it will enable established broadcasting companies to use the Internet as an alternative delivery medium for certain sorts of content. This is already being seen, with organizations such as CNN, Disney, FoxTV and the BBC providing Real Video and Streaming QuickTime channels, albeit subject to the vagaries of ordinary IP delivery. Very probably, when these services evolve to

[11] At the time of writing, it is only experimental.

provide higher quality interactive TV, they will only be provided on a subscription basis. The notion that streaming video technology can turn every Web page into a TV station may need some qualification.

Server-side Computation

In previous sections we described protocols that allow multimedia data to be carried over a network, but we have not considered where that data ultimately comes from. Consider the World Wide Web, in particular. Using client-side scripting, as described in Chapter 14, we can produce Web pages that are responsive to user input in a limited way, but they must be made up from a fixed collection of elements — typically stored in files accessible to a server, although video and audio streams can also be incorporated by a server that understands the appropriate protocols. Using this approach, we cannot even embed the time of day in a Web page.

Server-side scripting is used to enable an HTTP server to communicate with other resources, such as databases, and incorporate data obtained from them into its responses. In particular, server side scripts are used to enable a server to construct Web pages dynamically from time-varying data. The *Common Gateway Interface (CGI)* provides a standard mechanism for communication between a server and server-side scripts, although, as we mentioned in Chapter 2, proprietary scripting interfaces tied to a particular server are also widely used. Server-side scripting is not used much for multimedia yet, but as the limitations of client-side scripting become more apparent, and multimedia databases become more widely available, server-side scripting will become more prominent.

The Common Gateway Interface

Earlier in this chapter, we described how a client sends HTTP requests to a server, which sends responses back. The Common Gateway Interface provides a mechanism for the server to pass on data in a client's request to a script[12] — referred to as a *CGI script* — running on the server's machine, and for the script to pass data back, which the server then sends on to the client. An important point to realize is that the communication between the client and

[12] Or program, but most server-side computation is done using scripting languages, especially Perl.

server takes place using HTTP, so the output of a CGI script must be something from which the server can construct an HTTP response. Some scripts[13] actually generate complete HTTP responses, but more often a CGI script produces some headers, together with the response body, and leaves the server to add the status line and other headers, such as `Server`, which it is better placed to generate.

CGI is a mechanism for communication in both directions between client and CGI script, via the server. A CGI script is called by the server when the identifier in the request line denotes a script (usually because it refers to an executable file in a designated script directory). How does the client send data to the script? There are three possible places.[14] First, there are the headers in the HTTP request. Some of these might contain information that is useful to a CGI script. Second, as we mentioned in Chapter 9, a *query string* may be appended to the end of the URL — it is then added to the identifier in the request line. Third, a similar string can be sent as the body of a request whose method is POST. We must begin by looking at the format of these query strings, and how they are constructed by browsers.

A query string assigns values to named parameters. In this respect, it is rather like the <PARAM> tag in HTML, but its syntax is quite different. A query string consists of a sequence of assignments of the form *name* = *value*, separated by & characters. Within the *value*, spaces are encoded as + signs, and other special characters are replaced by code sequences in the same way as they are in URLs (see Chapter 9). Thus, an HTTP request for a CGI script called `register.cgi`, with a query string setting the value of a parameter called `Name` to the string `MacBean of Acharacle`, and that of `Profession` to `Piper` would look like this:

```
GET /cgi-bin/register.cgi?Name=MacBean+of+Acharacle&Profession=Piper HTTP/1.1
```

Such a request can be generated by any program that has opened an HTTP connection, but most commonly HTTP requests are constructed and sent by Web browsers. There isn't much point in using URLs with fixed query strings in anchors; to be useful, query strings need to be constructed dynamically from user input. The traditional way of doing this is by using HTML's facilities for constructing and submitting forms, comprising text boxes, radio buttons, pop-up menus, and check boxes. These facilities are quite elaborate, and we will not describe them in any detail. Suffice it to say that each element within the form has a `name` attribute, whose

[13] Technically known as 'non-parsed header' scripts.

[14] Actually, there's a fourth, but it is something of a security risk, and therefore is not used much any more, so we won't tell you about it.

value is used as a parameter name when the browser constructs a query string. The value of the parameter is taken from the user's input in the form element. Text fields are HTML INPUT elements with a type attribute of text, so a simple form that was used to elicit a user's name and profession for registration purposes might be constructed in the following way:

```
<FORM method="get" action="cgi-bin/register.cgi">
  <P>Your name:
    <INPUT type="text" name="Name" size="54">
  </P>
  <P>Your profession:
    <INPUT type="text" name="Profession" size="49">
  </P>
  <P>
    <INPUT type="submit" name="Submit" value="Submit">
  </P>
</FORM>
```

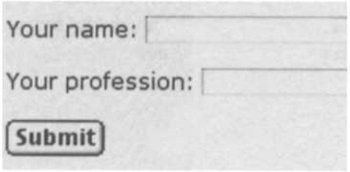


Figure 15.6
A simple HTML form

The displayed page produced by this code is shown (truncated) in Figure 15.6. The method attribute of the FORM element is used to determine the method in the HTTP request (GET or POST), and its action attribute identifies the CGI script. The INPUT element whose type is submit is a button that can be clicked on to submit the form. This causes the browser to send the HTTP request shown earlier to the server, which in turn passes the parameter values and request headers to the script register.cgi.

The mechanisms whereby the parameter values and headers are passed to the CGI script are of interest only to programmers. In fact, a muddle of different mechanisms is used, which can make writing CGI scripts from scratch excessively tedious. Fortunately, libraries exist which encapsulate all the tedious processing and provide a clean interface that allows the CGI programmer to access the values that the server has passed to the script with a minimum of effort. If you are going to write CGI scripts in Perl, Lincoln Stein's CGI module will save you a lot of effort. A brief introduction to its use can be found in Chapter 10 of *Perl: The Programmer's Companion* by Nigel Chapman (John Wiley & Sons, 1997). A full account is given in Lincoln Stein's own book: *Lincoln Stein's Guide to CGI.pm* (John Wiley & Sons, 1998).

As we remarked earlier, a CGI script must pass back to the HTTP server (most of) an HTTP response. Often, CGI scripts construct HTML by carrying out some computation based on the input they receive, and place it in the body of their response; that is, they provide a means of building Web pages dynamically. They can, however, send data of any type in the response, providing they supply a suitable Content-type header specifying the appropriate MIME type. Browsers cannot distinguish between a response generated by a CGI script and one sent directly by an HTTP server,[15] and will display any type of data that is generated by a script, using plug-ins and helper applications where necessary. CGI can therefore form the basis of a distributed multimedia application, with a Web browser providing a front-end to elicit user input with a form and display multimedia elements generated by a script's computation. Often this computation will include accessing a database, possibly containing the multimedia data itself, or possibly just providing information about the data, which is stored separately. In that case, users will be able to construct queries specifying desired properties of the media they wish to retrieve.

HTML forms provide a familiar and effective interface for eliciting user input, and you can use stylesheets to format them in a way that harmonizes with the rest of your page. (You can even animate the form elements if you want to, though this may not be a good idea.) However, there are other ways of invoking CGI scripts that may be more suitable for some sorts of application.

We mentioned in Chapter 14 that assigning to the href property of a window's location object has the effect of loading a new page. If we assign the URL of a CGI script, and append a query string, the new page will be constructed from the script's output in the usual way. The query string can be constructed by a script running on the client machine, on the basis of whatever logic is written into it, within the restrictions that must be applied to client-side scripts. Conventional form elements may not be required to obtain the necessary user input to construct the query; it may be sufficient to detect events on elements of the page, such as images that can be chosen with a mouse click. In a similar spirit, the additional scripting facilities introduced into Flash with the release of version 4.0 include the capability to send an HTTP GET or POST request, with a query string constructed by a script associated with elements of a Flash animation.

[15] This isn't quite true. In fact, they need to be able to distinguish between them, since it is not sensible to cache the output of CGI scripts.

Beyond CGI

The CGI mechanism has two significant limitations. The first is that it can be slow, which makes it unsuitable for applications that may receive many hits. Various techniques have been and are being developed for improving performance while retaining the essence of the interface. A more fundamental limitation is that the output of a CGI script is returned to the client as an HTTP response. In many respects, this is actually a good thing. It means that any HTTP client can communicate with any CGI script, and that caching and security are handled automatically by HTTP. At the same time, it limits the way in which the client and script can interact to the possibilities supported by HTTP — a protocol explicitly designed for the efficient transport of hypermedia, which is not the only model of multimedia, as we have seen.

A major shortcoming of HTTP for performing complex transactions between clients and servers is the lack of any persistent state in the server. Even though HTTP 1.1 allows connections to be held open, the server still deals with each request in isolation. This extends to requests that invoke CGI scripts: each such request causes a new process to be started to run the script, and terminated once it has done its work.[16] In order to maintain continuity between a series of requests from the same user, various more or less underhand ruses have to be employed, such as embedding a connection key in a hidden form field in the script's output, using 'cookies' to remember state information on the client's machine (if the client accepts cookies), or using non-standard server features instead of CGI.

An alternative strategy is to bypass CGI and HTTP entirely, by writing a special-purpose server and client to implement a specific distributed application. This can be where Java comes into its own, not necessarily because of any virtue it may possess as a programming language, but because the libraries distributed with it provide extensive support for network programming that is (relatively) easy to use — opening a TCP connection and creating an input stream that enables you to read from it as if it were a file only takes a couple of lines of code, for example. UDP connections are also supported. Java also supports remote method invocation, meaning that a program can be truly distributed, with code running on a client calling methods on objects running on a server, in almost

[16] This is the main source of CGI scripts' inefficiency.

the same way as it calls methods on local objects. This provides for seamless cooperation between the client and server.

In Chapter 14, we briefly described the range of facilities offered by Java's APIs for multimedia programming on the client side. It should be evident from that description and the preceding paragraphs that it is quite possible to construct distributed multimedia applications that combine media types — possibly including new or special-purpose ones that are not supported by mainstream HTTP clients — and protocols — again, possibly including ones that are not otherwise supported. The applet mechanism makes it possible to embed the functionality of such applications in the ubiquitous Web browser, and to allow users to download the client software when they need it. Alternatively, the reusable component technology known as Java Beans makes it possible to construct clients in a form that can be easily incorporated in other applications.

Where the computation in the server is mainly concerned with accessing databases, as it often is, the *Java Database Connectivity (JDBC)* API can be used. This provides platform-independent access to databases that use SQL as their query language. This is the vast majority of serious conventional relational databases. Although commercial database systems do not allow media data to be stored directly in databases, they can be used to store and retrieve data about such data. For example, an index to a collection of video clips, storing their title, length, the codec used to compress them, the date they were made, the director, and a summary of their contents could be stored as a set of relations. A URL at which it was stored could be used to represent each clip — being a string, a URL can be stored in a database where a QuickTime movie cannot. A client program could provide a user-friendly interface to such a database, and construct SQL queries from a user's input. A server could then execute these queries, via JDBC, and send the URLs of clips satisfying the user's criteria back to the client, which could then retrieve the clips and display them, using the facilities of the Java Media Framework or QuickTime for Java.

Further Information

[CHW99] is an expert summary of the protocols and related matters introduced in this chapter; [Kos98] is an accessible account of multicasting over the Internet. [MS99] includes much informed speculation about the future of the Internet, including the impact of multimedia. The full details of the major multimedia protocols can be found in their defining standards: [FGM+97] (HTTP), [SRL98] (RTSP) and [SCFJ96] (RTP).

Exercises

1. True or false: file transfer applications such as FTP can only run on top of a reliable protocol such as TCP?
2. Why is a sequence number on its own not sufficient to synchronize separate RTP streams?
3. Explain why TCP cannot be used as a multicast protocol.
4. Since the Internet is a network of networks, packets often travel over networks belonging to several different ISPs. Normally, ISPs operate 'peering agreements', whereby they provide their services free to other ISPs of roughly the same size, in return for reciprocal services. This way, ISPs don't have to keep track of where packets come from and go to, and charge each other for carrying them. Discuss how the commercial provision of multicasting may affect such arrangements.
5. The HEAD method of HTTP is much like GET, except that it causes the server only to return headers, and no body, in its response. What might this method be used for?
6. If RTSP is used to control a multicast stream, what would you expect to happen if a client sent a PAUSE request to the server? Discuss the implications of your answer.
7. Is it true that, as we claimed on page 517, 'There isn't much point in using URLs with fixed query strings in anchors'?

8. RTSP allows the client to randomly access any frame in a video stream (provided it is not live) by specifying its SMPTE timecode. If you were designing an application that allowed users to control streamed video, what controls would you provide to give them this facility? Would your design task be any easier if additional information about the video stream could be transmitted separately? If so, what information would you use?

9. Video streams are often described as 'bursty': that is, the data rate is not constant, but exhibits bursts or intermittent peaks. Explain why this pattern occurs, and describe the problems it presents to networks.

10. The application outlined in the last paragraph of this chapter could largely be implemented using client-side scripting, CGI and HTTP. Describe how this could be done. What desirable features of such an application require you to step outside this technology?

Multimedia Practice

This chapter is different from the rest of the book, both in its form and in the nature of its content. It is entirely concerned with putting ideas into practice — with learning through doing — and is presented as a series of multimedia projects, preceded by some general advice and explanation. It is not intended that these projects should represent a general cross-section of real-world briefs — their purpose is to focus learning through solving a range of different problems. Each one has been designed to highlight certain areas of difficulty in practical multimedia, and to stimulate original and well thought out responses. Nevertheless, many of the briefs do draw their inspiration from existing or potentially real multimedia productions, or from actual problems we have been asked to solve in practice. There are no trick questions — the challenge lies only in planning or carrying out the projects themselves. In order to assist you in reaching appropriate and satisfying solutions, each project is discussed in some detail, and the main areas of difficulty are clearly identified.

We know that very different kinds of practical facilities will be available to different readers, and that the amount of time it will be possible for you to allocate to any project will also vary considerably. Some of the projects are small, some could be very ambitious indeed, and one or two could not actually be implemented with currently available technology. However, for each one, a variety of approaches is possible, and you should tailor your response to fit your personal situation. What you cannot execute in practice, you might still be able to plan fully on paper; where only some of the necessary media tools are available to you, you can try to improvise an alternative implementation of the brief with the facilities that you have got. However, some of the briefs have been limited with certain restrictions for a good reason, which will be clearly indicated — to ignore the restrictions in these projects would be to miss the point. Even if you cannot do any practical work at all, much can be achieved simply by reading, thinking, and planning.

You may be working independently, or in a group — this too will determine how much you can achieve. Do not set out to do what is clearly impossible, but try to be realistic. To put ideas into practice successfully, you need to make an accurate assessment of what you, or your group, can actually produce. You will need to distinguish between the more ambitious and imaginative projects, and those briefs which are relatively simple and realizable. It is intended that some of the projects will function primarily to stimulate thought and problem-solving abilities, and these may be very difficult or impossible to actually carry through in practice. Others — the majority — could be realized without too much difficulty or the need for unusual equipment. If you are working in a group, there will be additional questions of project management and the division of labour to address, which will add a further challenge to the realization of each project. Groups need to agree upon an effective and productive organization of work, taking into account the personal strengths, weaknesses, skills and experience of group members. We do not offer guidance on specific management strategies, as each project is open to many possible approaches, both by individuals and by groups of varying sizes and abilities.

Inevitably, when it comes to actually creating a digital multimedia production, you are dealing with design: design of an interface; design of a coherent production; maybe even content design and creation. It is most important that you make an honest assessment of your own competence in this respect. Unless you have spent several years on a studio-based course in a design school, or have equivalent work-based experience, you need to recognize that you do not have the advantages and experience which others, properly trained in their field, are expected to have. Never be afraid to seek whatever help you can get from specialists — whether from fully-trained professionals or from people studying in appropriate areas. This is a fundamental part of good professional practice, and your work can only be the richer for adding other expertise to your own. However, if no such help is available, clear thinking, common sense, and a keen awareness of other people's work and your audience's needs or desires, can take you a long way. By studying both traditional and new media, and using the World Wide Web, you can gain unprecedented access to other people's work and ideas; if you analyse contemporary practice in individual media and multimedia, and examine design precedents carefully, you can build up a sound and informed basis for your own work. Much time and effort — not to mention disappointment — can be saved simply by learning from other people's successes and mistakes. It may sound like a piece of advice too obvious to be voiced, but it is so often overlooked, or scarcely acted upon, that it can hardly be repeated too often.

What is true for technical people who lack training in design, applies even more to multimedia designers lacking technical expertise. If you fall into this latter category, you too need to acknowledge your shortcomings, and to seek skilled help. But for you, if help is not readily available, a study of other people's work will not serve as a substitute. Too often we have seen multimedia presentations — especially Director productions — where inadequate technical knowledge has resulted in serious flaws in functionality, such as leaving users without a way of stopping a certain part of the presentation, or without any exit option at certain points. Where scripting or other programming skills, or highly technical knowledge, is necessary to make a presentation work, do not try to take a shortcut around this requirement. If it is impossible to obtain skilled assistance, you must either look for a solution to the problem which will fall comfortably within your own — or your group's — sphere of competence, or you must undertake to acquire the missing skills or knowledge at an adequate level.

Underlying principles and theory may be learnt from a book, such as this one — but to put this theory into good practice is another matter. Your multimedia work will be judged not by what you know, but by the practical results. Make yourself aware of the context in which you are working, make good use of specialist skills and your knowledge of precedents, and then *practise.* We all make many mistakes, and where practical work is involved these will quickly become apparent. Test out your designs and ideas at all stages — whether they are still in your head, planned on paper, or actually realized as multimedia projects. Avoid choosing your test audience from the people you believe will respond most favourably, but if you can, seek out impartial or unlikely test subjects — and then listen to what they say. You may ultimately dismiss some people's responses as being irrelevant to what you are doing, or you may decide that you are right and they are wrong, but make sure that you have both heard and understood their criticisms, and can give good reasons for dismissing those that you reject.

Any practical challenge offers scope for the imagination and for ingenuity. In some cases the result could be wildly fantastic, and in others a simple but elegant piece of classic design. The key to the successful realization of a project is never to pre-judge or copy, but to think through each task that has been set until you have reached a clear personal vision of your best solution. Then keep that vision in mind throughout all the stages of development, to use as a measure of your progress. Sometimes it will prove necessary to amend your original plan — so be it. The ability to adapt can be a key virtue in a designer — but do not lose sight of your goal, even though you may have to alter the route by which it is reached.

Rather than present some patent set of design rules or guidelines — a list of 'dos and don'ts' — we simply identify, within the descriptions of the projects, some of the more important issues you will need to address. So far as they concern layout, presentation and content, ideas of 'good' design are very much a matter of fashion and cultural sensitivity. It is true that a few classics of design may be widely admired and achieve a relatively long life, but on the whole, what looks wonderful in one decade (or even one year) becomes out-moded, or positively unthinkable in the next,[1] and what may be the height of good taste and fashion for one group of people could be grossly offensive to another. Just be aware of what you are doing, and who you are doing it for. Even at a particular moment in time, and in a single place, many different standards of design will obtain among different social, economic and cultural groups. With multimedia productions often intended to reach a worldwide audience, the picture is extremely complex and unclear. It is highly probable that what seems good to you, or in your immediate social and cultural context, will most certainly seem bad to at least some part of a world audience. So, the wider the audience you wish to reach, the greater the challenge — but where you have a specific and restricted target audience, your task will be less difficult.

Interface design differs from other design elements within a project in so far as it is concerned with functionality and ergonomics. Some of the projects have a specific focus on interface design, and for these we have indicated some of the things you will need to think about — but remember that, even here, nothing is written in stone. It was not so very long ago that the now almost ubiquitous 'windows, icons and menus' interface, with its pointing and dragging, was unthought of. Before its invention — and for a considerable time afterwards — most computer users interacted with their machines in an utterly different way. Unless new designs depart from existing paradigms, there can be no significant progress. There is always room for good new ideas, and designers of interfaces — like designers of anything — are likely to continue to try to push forward and break new ground. As a general guide, it may be useful to bear in mind what Malcolm McCullough has expressed in the following way:[2]

> "Better interfaces relieve attention overload by getting computing out of the way ... changing it from active task into passive context. Digital production occurs within a highly visual and networked environment where our primary context is communication."

Usually, in the contemporary world of digital multimedia, our primary context is indeed communication, whether we are on the 'sending' or the 'receiving' end. It is no longer fashionable to say that "the medium is the message", but this is necessarily the case to some extent. If you wish to communicate *within* the medium as far as possible, you will need to minimize all aspects of that medium which are intrusive — which draw attention to the medium itself or to the business of interaction. For example, if you are displaying a list of product photos and descriptions for consumers to choose between on a Web page, it is unlikely that incorporating a large, jumping

[1] Fashion in clothing illustrates this most effectively — consider the history of flared trousers, or the bustle. But it applies equally to all aspects of design, from fonts to table-ware to billboards and TV advertisements.

[2] [McC96, p. 262]. This expresses very nicely a current state of thinking about computer use. Future developments will no doubt bring unpredictable changes to the way we feel about computing.

and distorting animated title as a header to that list will best serve the intention of having potential customers focus on the products. Or, if it is not made clear in a piece of interactive multimedia what inputs from the user will give what results — or if it is difficult to achieve those inputs — your interface is itself presenting a challenge in interaction. This is fine if that is what you intended — for example in a game or certain types of educational product — but not otherwise.

Several recent books (for example, [LH99] and [EF99]) have proposed formal methodologies — derived more or less explicitly from current practice in software engineering — for multimedia development. If you are familiar with software engineering practice, you may find it helpful to try and apply the same principles to your multimedia practice, but you should bear in mind that there are fundamental differences — centring on content — between the two disciplines. Multimedia is a new area, even by the standards of computing technology, and we still lack a sufficient understanding of its nature to be able to quantify the processes of production and reduce them to a set of rules and procedures.

In thinking about and carrying out the projects described in the remainder of this chapter, you should feel free to ignore any of the advice we have just given. This is a chance to explore multimedia production in a safe context, free from the pressures of deadlines, clients, and the test of public opinion. Hopefully, the experience you gain will help you avoid some of the worst pitfalls when you are exposed to those pressures on real projects.

Note that the projects are not presented in any meaningful order. This is partly because what presents a big challenge to some readers will be an easier task for others, so ordering according to levels of difficulty is not really appropriate. (The one exception to this is the final project, which represents the culmination of the whole book.) And partly because while several projects may deal with similar media areas — they may share a focus on sound, for example — they will also be concerned with quite different issues in each case. So you may approach the following tasks in either a linear or non-linear way, according to your own preference. It really doesn't matter which.

The Projects

Choose a personal hero or heroine — living or dead, famous or unrecognized, real or imaginary, human or non-human — and design a *single* Web page (i.e. one without links), using whatever media and software are available to you, to present your chosen figure in the best light to a worldwide audience. Make the design of your Web page 'fit' your hero or heroine in every possible respect, and keep the total size of the content below 250 kbytes. (If you want to make this a bit harder, keep the content down to 100 kbytes.)

This project is about effective communication in a very small space, and about sensitivity in design. You will need to be highly selective about the material you choose, process it sufficiently to reduce its size to meet the target, and design and organize your page in an appropriate way. In asking you to make the design 'fit' the figure you have chosen, we mean that you should choose and integrate all the elements of the page in a way that is sensitive to your subject. For example, if you choose a great political or religious leader, you might wish to encourage viewers in a feeling of respect and admiration for their achievements or philosphy — an irreverent comic strip or a quirky font would be out of place. If you were to choose a film star, on the other hand, a touch of glamour and a showbiz aura for the whole page might not go amiss ...unless, perhaps, you wanted to convey their personal strength in some aspect of life unconnected with the film industry. If you were to choose your pet dog, or Bugs Bunny, a completely different solution would be appropriate. So think hard about exactly what it is that you are trying to communicate, and then make sure that every aspect of your design works towards that end. Be aware of the difficulties in addressing a world audience effectively, and think about ways of getting your message across to as many different kinds of people as possible. Do not be discouraged if you have only limited technical resources — results can often be more effective when the means are simple. If you do have substantial resources, excercise discretion in their use, concentrating primarily on effective communication within the limits set. Don't forget that less is quite often more.

Traditionally, museums and galleries prevent visitors from approaching valuable objects or works of art too closely. Many items are protected by glass cases or barriers, and very delicate objects may be displayed under low lighting conditions. It is almost never possible to see the objects displayed in any context or arrangement other than the one ordained by the exhibition organizers.

Design a simple Web site or a CD-ROM, which exhibits a number of artefacts of your choice, and allows the user to view and examine the items as they might wish.

This brief may seem straightforward — there is no particular difficulty in setting up a virtual exhibition. The challenge lies in going beyond the confines of the exhibition paradigm. Imagine what visitors might like to do with paintings, if they were allowed to: take them off the wall and look at the back, examine them very closely, or from a distance, perhaps judge the effect of looking at them upside down. What about an exhibition of early Renaissance paintings, or aboriginal cave paintings, which are known to cover earlier paintings lying beneath? And different artefacts raise different possibilities. What might you want to do in an exhibition of ancient Egyption artefacts, or a display of stained glass window panels? It is your task to enable the virtual museum visitor to control the way in which they view and interact with the virtual objects to the greatest extent possible. Note that the emphasis here is strictly on the visual — the intention is not so much to inform as to provide a wide, user-defined range of visual interaction with the objects on display.

You will need to design an interface which allows satisfying displays of the objects while providing a simple and effective way of interacting with those displays to change them in a number of different ways. As direct interaction with the artefacts on display is the primary focus of this project, the business of having to interact with the computer as an intermediary will be intrusive, and your interface should minimize this disturbance as far as possible.

More ambitious solutions to this project will require considerable expertise and a wide range of software. So, be aware of what it is not possible for you to achieve — and therefore of areas which you should steer clear of.

Clearly, if you have the necessary software and the skill to use it, one fruitful approach to this project could be based on 3-D representations and VR.

Devise a multimedia application to assist in the identification of birds in a specific geographical area. This should use still images, sound and text — and video if possible (but if not, do without it). It should include an interface that allows the user to obtain, for example, a view of typical tail twitching behaviour, or a sample of warbling song, as well as stock responses to form-filling.

The aim of this project is to make the end product serve a precise requirement in the best possible way. The requirement is for an aid to the *identification* of birds — not a catalogue or other attractive presentation. Birds are usually identified by information such as size, colouring and markings, voice, habitat, behaviour, and so on. You need to decide (or research) what features will aid identification, and then assist the user both in recognizing these features and matching them to the birds of your chosen area. Try to think through the ideal solution to this brief from the ground up — you may learn by looking at similar multimedia applications which already exist, but avoid copying them.

An artist wishes to convert a many-layered screen print into a multimedia gallery presentation. The original printed image consists of 24 layers, which will be scanned in separately. Each layer contains one coloured abstract shape or mark; some are opaque and some are semi-transparent. In the presentation — on a computer screen installed in an art gallery — these shapes and marks are to move around in an apparently random manner, independently of one another, giving an almost infinite possible number of permutations of the image. The ultimate purpose of this display is to enable the viewer to halt the movement at any time, and request a printed version of the particular permutation of the image which is displayed at that moment.

This is another brief which appears deceptively simple: several of the technologies described in earlier chapters, including Flash, Director, and JavaScript, will enable you to animate layers. However, you must ensure that the technology you choose does justice to the material. Does it handle transparency correctly, for example? There is also an interesting problem in deciding what constitutes random motion of the layers. Presumably, an unpredictable swirling effect is required. How will you achieve this? And if the user's reactions are a little too slow, such that they stop the movement beyond the state which they wished to freeze, how will you ensure that they can retrieve the image they actually wanted?

The real difficulty of the project is concealed in the innocuous final requirement that the user be able to request a printed version of the current display. This is a fine art project, so screen resolution prints will not be acceptable. Accurate colour is important, both on-screen and when printed. How will you arrange that a high resolution image can be printed that is identical to what is displayed on the screen at the instant the print is requested?

An old harbour town has decided to mount an exciting multimedia installation as a tourist attraction — to be sited in a boat moored by the waterfront. Inside the boat, visitors are to be given access to a vast amount of information and material in all media on the subject of 'harbours through the ages'. Devise an effective and realistic implementation of this installation, with an interface suitable for constant use by the general public, and some measure of control which will prevent the visitor from being overwhelmed by excess material or getting disoriented.

The first problem here is obtaining the material. A possible approach is to hook up the boat to the Internet, but a conventional Web browser is too general a tool. You might write a dedicated Web client that automatically queried the major search engines at intervals and then provided a set of links for users to follow (but how would you prevent them getting lost in hyperspace, or stepping outside the material on harbours?). Alternatively, you could download the material, and convert it to some standalone form of multimedia production. How would you store, organize and retrieve the material — and make sure it stays up to date — in this case? As a third alternative, consider how you would obtain suitable material from conventional sources, and digitize it. Make some estimates of the costs of this approach, and consider the problems of copyright and clearances.

The second challenge lies in designing a suitable interface for use by people of all ages and abilities. This is a classic design problem, without an emphasis on special needs. For this particular project it would be a bonus if the interface could relate to the subject of the installation (harbours) in some way.

Construct a Web site for use by blind and severely visually handicapped people who are computer literate. The site is to provide a guide, tailored to the needs and interests of the visually handicapped, to a city of your choice.

In one sense, this is easy. The W^{3}C accessibility guidelines, which ought to be compulsory reading for all Web designers, describe how to use the features of HTML 4.0 to design pages that can be appreciated by blind and visually handicapped people. So this is partly an exercise in applying the guidelines — but technical fixes are not the whole solution. You need to think this through for yourself — to choose appropriate material, and organize it in a way that does not depend upon the visual character of a Web page. If possible, consult somebody with visual difficulties before you start, and then have them evaluate the site you produce to see how well you have appreciated their problems and needs.

Create a simple Web site which contains basic but perhaps rather dull information (or use an existing one), but add to it all that is necessary for the user to build and experience their own sound environment (within reasonable limits) to enhance their experience of using the site. This sound environment should not relate to the other content of the Web site in any way — its function is solely to make the experience of using the site more pleasant.

We observed in Chapter 12 that sound can be particularly irritating, and that it is a bad idea to add music or other sound to a Web site unless a user can turn it off. In this project you are invited to go further, and allow the user to select background sounds that they like to accompany their browsing of this rather dull site. There are three sub-problems: providing the sound, constructing an interface, and playing the sound. For the first step, don't feel restricted to music; natural sounds, such as running water or rustling leaves, might have more appeal. This project offers you the opportunity to experiment with making sound recordings and digitizing them. The interface could simply allow the user to preview sounds, and select one to play in a loop or for a certain time, or it could allow for sounds to be mixed or altered to create a 'soundscape'. The latter will require programming, but if you cannot program, just designing the interface offers a challenge in itself. You will need to consider how the sound is to be delivered from the server to the client. Solving these problems should determine how you can implement the actual realization of the sound environment.

Create a CD-ROM and/or a Web site guide to an art college's annual degree show. This needs to display a wide range of work in such disciplines as glass, ceramics, fashion design, theatre design, graphics, illustration, photography, video, animation, painting and drawing, sculpture, etc. It should convey the scale of work, and the spaces in which it is displayed in the real show, where that is appropriate, and it should allow for each student to include a written statement and short personal profile to accompany their degree display. Your production should provide adequate navigational controls for the user to find their way around both the virtual and the real degree show.

This might almost be called a classical problem in multimedia, so commonly is it required. Remember that, in this case, the multimedia production is subsidiary to the real show. Although it should be attactive and useful in its own right, its primary function is to advertise the degree show and serve as a guide to it. Thus you should, in some way or other, represent the physical space of the show, whether as a map or using QTVR or some other 3-D representation. You must also respect the students' work. Where it is represented, you should do it justice, and not subsume it to some grand design scheme of your own. Different types of work — painting, sculpture, video, and so on — need different treatment to convey their nature. Do not fall into the trap of trying to produce a solution that will work for any show. Fit your solution to specific work.

The best results will be produced if you are able to work with an actual art college or faculty on real students' work.

A chain of real estate agents wishes to provide computers in each of their branch offices which will access a central database holding pictures, video and VR material relating to the properties for sale. Devise such a system — using the Internet and not a local network — which allows for this centrally held material to be pulled down for display to walk-in clients.

This brief is representative of a class of multimedia applications that are being deployed in the world, although at present, intranets are usually used and not the Internet. It is a large-scale, technical project, and should only really be attempted by a team that includes a high proportion of technically skilled people. Its importance lies in the need for design and computation both on the server and client side. On the server, it will be necessary to design and implement a database that can hold (or at least point to) multimedia data. On the client side, an interface that can be used by office staff to display property details in a suitably impressive manner to clients is required. The client and server must communicate. HTTP and CGI may be adequate with a conventional Web browser interface, but you may be able to design a better (at least a more efficient) solution.

Produce an interesting and enticing presentation of life on your campus which uses just recorded and remixed sound and still images (no video), to play as a looped show on a simple fixed installation without interactivity, in a foyer or waiting room where potential students are waiting to be interviewed.

This is an exercise in production and communication skills using relatively simple means. The result will essentially be a looped slide show. To compensate for this simplicity and to avoid possible boredom on the part of the viewers, make your production as imaginative and interesting as possible. Avoid the obvious: consider using devices such as humorous or ironic juxtapositions of pictures and/or sound, and consider how to achieve emphatic effects. Think about the pace and flow of the production — try to create variety and dynamic relationships between elements of your material. Make the most of the media you are working with — still images could come from many different types of sources and may be treated or combined digitally. Sound alone can be extremely powerful, so make the most of its potential.

A maker of short animated films wishes to give the audience a new level of interaction with her work. A ten-minute film has been made frame by frame with traditional animation techniques, and captured to computer disk, complete with sound track. The film-maker would like the audience to understand more about how the film was made, and to be able to view individual frames or selected sequences at will. More challengingly, she would like any viewer to be able to re-edit the film, and play back their own edited version. This version would persist until either the computer was instructed to return to playing the original version, or another re-edit took place. Devise an implementation of this brief which would be suitable for installation in a public space, with the display of the film versions maintained on a TV monitor or video projector separate from the computer interface.

It's hard to see how this can be implemented without some custom programming. You might consider using one of the consumer-oriented desktop video editors, such as Apple's iMovie, but before doing so, you should think very carefully about whether an untrained audience could make use of it, without getting into difficulties. If you prefer a custom solution, you will need to devise a minimal interface that lets the user perform enough re-editing to obtain some feeling of control, but is simple enough to be mastered without instruction. (Non-programmers can usefully try to design such an interface without implementing it.)

The technically minded should also consider the problem of how to maintain the display of one version of the movie while somebody is creating a new edit.

Create a very simple, low-budget presentation of a set of instructions of your choice — for example, the best way to brush your teeth, or how to give basic first-aid. Assume that this presentation may have to be installed on a very old computer. The instructions are to be text-based, with links via clear and appropriate icons to a pure graphic exposition of the same instructions, for those who, for whatever reason, cannot read the text.

This apparently simple project is not easy to do well. The challenge is to communicate absolutely clearly, in either text *or* pictures, but not in both at the same time. The design of the presentation should also promote clarity, both in the general layout and ordering and in the design and placing of the icons for the links. Think about the presentation of material one screen at a time. If you do not have graphic design or illustration skills, you could usefully collaborate with someone who does for this project, as the purely graphic exposition of instructional material presents a substantial challenge in itself. However, if this isn't possible, you could use photographs or existing material (providing you observe copyright restrictions).

Remember that the brief specifies that this production must be capable of running on a very old and basic system — don't deny information to people who do not have access to the latest and most powerful technology.

Imagine a portable tourist's companion of the future. When turned on, this hand-held digital device will: tell you where you are in the world (or galaxy); ask you what you would like to know; show you appropriate multimedia material; give you directions on how to get where you want to be from where you are now; make reservations for travel, accommodation, etc. on your behalf; and connect to Web sites for further information where appropriate. It will update itself to remove redundancy, and hold a memory of your personal use for user-sensitive assistance. Specify a design for such a system, and if you feel you have the skills and necessary facilities, implement such parts of it as you can.

Which may be none of it... This is an ambitious project that goes beyond the capabilities of today's technology (doesn't it?). This should not prevent you from thinking about what is required, from hardware, software and telecommunications. A possible starting point would be to look at how nearly you can approximate this device with the facilities currently available: hand-held PDAs, mobile phones, and the Internet. It may not be too big a technological step from there to the full system. However, you should also think about what is required beyond the technology: Who is going to provide the information? How is it going to be paid for?

You may find it possible to implement some of the sub-tasks; if so, try to combine as many as possible into a single system that approximates the intended implementation. Make an honest assessment of how nearly your prototype approaches a genuine realization of the brief. (Try to see this from an ordinary consumer's point of view.)

Thinking about how hypertext, displays, frames, etc. work, design a layout for an on-screen presentation of a substantial and heavily annotated manuscript such as the Bible, the Koran, the Buddhist Sutras, the complete works of Shakespeare, or whatever.

Footnotes and endnotes, as we know them, are a device entirely dictated by the physical form of books and the process of printing. Digital media frees us from those forms, but what are we to do instead? Is putting an HTML anchor node wherever you would place a footnote marker any more convenient than putting numbered notes at the end of the text? The aim of this brief is to utilize the dynamic nature of screen layout to devise a mechanism for associating annotations with a text — a mechanism that is convenient, placing the annotation where it is needed, but unobtrusive. It is a good discipline to try and work within the limits of Web technology (you will probably need to write some scripts), and this may allow you to produce a prototype. You could also think about ways of automatically generating texts in your layout from XML markup.

Create an instructional road safety installation specifically for teaching blind or severely visually impaired children how to interpret sounds at the roadside. This should incorporate some very simple interaction with an appropriate interface for these users — to allow, for example, for repetition of elements, starting and stopping the instructions, and so on.

This is not simply a re-run of the earlier project for blind people. Here, sound is a vital part of the system, but must be interpreted. What is required is not a guide, or a collection of information, but a complete learning environment. This offers an opportunity for you to research and think about educational uses of multimedia, as well as designing both for children, and for special needs. The interface does not just duplicate the familiar functions of a Web browser, but has to provide special controls suited to the teaching purpose of the installation. Flash [3] may be a suitable tool for implementing at least a prototype.

Import a short (say five minutes or so) but already edited piece of video or an animation into a video editing application. Study it carefully and choose 20 or 30 single frames which you feel will best convey the narrative or import of the whole clip on their own. Produce these as (a) a storyboard, and/or (b) a time-based animatic.[4]

This is an exercise in reverse storyboarding. Its purpose is to make you think hard about how time-based visual media are constructed and develop, and to enable you to identify and subsequently plan structures and narrative development for such work of your own. The challenge lies in identifying a few key frames out of some 9000 candidates. You should test out your result on someone who has not seen the original piece you started from — your success will be measured by how accurately they are able to understand the story or development of the piece from your storyboard or animatic.

[3] You will need Flash 4 for the extended scripting features.

[4] An animatic is a sort of sketch of a movie composed entirely of still images, each held for the duration of the scene envisaged for the final, full-motion, movie. Each still image should capture the essence of a scene. The animatic will show the rhythm of the edits that are proposed (or, in this case, have been made) and the flow of the whole movie.

Choose a popular computer game, and carry out a reverse storyboarding process similar to the one described above — this time for a non-linear structure — to convey the essence and possible developments of the game to someone who doesn't know it. This project may be carried out either on paper — in which case devise a suitable non-linear structure, or in a digital form. The number of elements in your non-linear storyboard will be determined by the game you choose, but you should keep them to the minimum possible.

Carrying out the brief should not, in itself, present too many problems. To learn from this project you should ask yourself how well the storyboard manages to convey the non-linear structure, compared with the way it worked for the linear structure in the previous project. Can you devise an alternative representation that performs the same function but is better suited to non-linearity? Given the non-linear nature of much multimedia delivery, the ability to visualize, plan and represent non-linear structures is of fundamental importance. Is it appropriate to develop a personal way of representing such structures? Do you think the problems and constraints will be different or broadly similar for each non-linear project you might undertake? These issues are fundamental to the production of non-linear multimedia productions at the present time.

Why would it be pointless to try to construct an animatic for such a structure? Can you think of any way of producing a time-based representation of a non-linear production that takes account of user-driven events?

In Chapter 1, we invited you to choose a Web site and identify at least five ways in which you thought it could be improved. Return to your chosen site and redesign it, using its existing content, so that it is as good as you can make it.

This ought to be a simple project by now. You should be able to determine which of your original suggestions for improvement are implementable, and implement them. It may be that, having now learned more about multimedia technology, you must accept that some of the site's features which you found inadequate are unavoidable consequences of Web technology. For these, consider ways of changing the site to reduce their impact. In order to make your changes, you should either rewrite the HTML by hand, or import the site into an authoring application and change it there.

The original exercise in Chapter 1 also asked you to criticize a CD-ROM. You may like to consider, for this case too, whether your original suggestions for improvement can be implemented, and how.

Design an interface — and make a mock-up if appropriate tools are available to you — for an international tele-conference for students to exchange thoughts on their experiences of multimedia courses.

Think about what you need to control; what you need to indicate and the best means of doing this; what analogies you should be thinking in terms of (is this to be like using the phone, watching TV, being in a conference centre, or what?); and what assumptions about the underlying technologies you are or should be making.

There are several tele-conferencing applications in use, and you should look at their interfaces as a starting point. None of those that we have seen give the impression of having been designed — each participant is put in his or her own window, equipped with a few controls. Try to think of an interface that fits the application, not the desktop context in which it will be used. One possibility is to use a three-dimensional representation of the conference.

Create for yourself a show-piece Web site which you could use to exhibit your design skills to a potential employer or client. This should be a no-budget production (except for using such equipment as is available to you) — the content should be entirely comprised of royalty free material trawled from the Web.

The purpose of this project is to help you to understand and maximize your own particular Web design skills, without being concerned with the production of content. However, you should ensure that you choose and organize your found content in such a way as to create a successful and coherent Web site. Make use of the full range of Web technologies if you wish, but ensure that your page degrades gracefully on older browsers, or when viewed over a slow link. And be careful to test it on as many different platforms as you can — you won't impress many clients if the first thing they see is a scripting error message.

Using the material from your college's Web site (or a part of it), make a Director movie that conveys the same messages and information, but more effectively and in a way that is better suited for distribution on CD-ROM. Add appropriate supplementary material if you can, to take advantage of the change of delivery medium.

The rise of the Internet as a delivery medium means that more and more content is being 'repurposed' for the World Wide Web. In this project, we are asking you to go in the opposite direction, in order to gain an insight into the different requirements and opportunities of online and offline delivery. You could take the exercise further. For example, you could produce a print version, using text and images only. At each stage, see how the form must change to fit in with the medium. Alternatively, try to avoid changing anything, and see whether the form and content used in one medium fit comfortably in another.

This project should be done in groups. Each of you should obtain or make one or more pieces of text, audio, video, animation, and one or more still images — you each need a total number of media elements sufficient to send a different one to each of the other members of the group. (So if your group has ten members, for example, you each need to collect and send nine media elements.) Each group member will end up with a collection of elements, with no particular logic to their combination. The task is to create a multimedia production out of those elements, first by working individually, and subsequently by combining your individual productions into a larger whole.

It is important for group members not to collaborate beforehand; the idea is that each of you is presented with a collection of elements over which you have no prior control. It may be that one member ends up with nothing but pieces of text, or a large collection of sound files, or things may work out so that everybody ends up with a balanced number of elements of each type. The point is to make the best of whatever turns up. This means looking for ways of using the particular mix of media that you have been sent, looking for connections or contrasts between media originating from different people, and so on.

This project can be carried out on a small scale — within your particular class, for example — or on a much larger scale, by collaborating with other groups around the world. You can choose whether or not to focus it around a specific subject, or to leave it wide open (which is more difficult to do well).

Malcolm McCullough suggests "we might say that postmodern consumers are ceasing to spin their own yarns, figuratively, every bit as much as the artisans of industrializing Britain stopped spinning their own yarns, literally."[5]
Thinking again of the dark and stormy night in the Introduction to this book, design an exciting and enabling CD-ROM or Web site which will allow postmodern consumers to spin their own yarns once again, with the only constraint being that the yarn commences with the expression (not necessarily in words) "It was a dark and stormy night ... ".

This is one to dream about and plan; it is not expected that you will attempt it in practice — unless you really feel that you are in a position to do so. This is not about providing a ready-made story with alternative branches — or anything like that. It is about trying to invent a mew multimedia form for the future. What would be required from a multimedia production that enabled each user to spin a yarn, using all possible combinations of media, which was completely of their own making but from a single starting point? Give free scope to your imagination, but temper it with some hard thinking about what is required for this to become possible. Somebody has to lead the way ahead ...

[5] [McC96, p. 73].

Afterword

The Creative Challenge

Brent MacGregor

Head of School of Visual Communication, Edinburgh College of Art

Multimedia has arrived. Developments in hardware and software now make it possible for things only dreamt of a few short years ago to be created easily. Domestic machines can play complex multimedia product and easy to use software tools allow the creation of imaginative work on the domestic desktop. Once created, such products can be recorded cheaply onto silver discs or instantly published on the Internet. The technology is now enabling, not inhibiting; it has delivered its promise. What is needed now is for a creative infrastructure to develop alongside the software and hardware. New creative dreams will be dreamt and realised as multimedia matures and develops in the hands of a wide range of imaginative people. This revolution is only beginning.

User base

I remember an old joke from the early days of multimedia. A developer arrives at the pearly gates having received the final error message. 'Who are you and what have you done in your life?' asks a holy personage. 'I created multimedia' comes the proud reply. St. Peter answers 'That's nothing, God created the world in six days.' 'But He didn't have to worry about an installed user base,' comes the frustrated response.

I remember a time, not so long ago, when the phrase 'multimedia computer' was used to describe a PC with a double speed CD-ROM and a sound card. Multimedia developers worked in niche markets where the required hardware for consumption could be afforded. Today every sub-$1000 PC is multimedia capable with a fast CD-ROM if not DVD player, sound and video cards. Complex multimedia products can be used on ordinary mass market computers. The once desired user base has arrived. What is needed is the creative infrastructure to complement this 'kit'.

Just when every PC is shipped with a CD-ROM and several free multimedia products; when the physical manufacture of silver discs is cheap, the world has changed and DVD has arrived with its increased and much desired capacity. Will there will be another time lag as the installed user base arrives sufficient to justify huge investment in newly developed product? Fortunately, as with VHS, the predominant software for the DVD revolution already exists in the form of films originally produced for the cinema. This new technology offers high quality domestic reproduction of movies which may assist take up rates, but will the DVD player sit by the television in the living room or be part of the computer on the desktop or both? Will DVD offer unlimited creative possibilities, or will it just become another way of delivering Hollywood product?

Creative possibilities

These are just quibbles. It really is time to start getting excited about what is now or very soon possible. The printed book dates from the fifteenth century. Photography was developed in the 1830s, while film was invented in 1896. Radio programmes developed during the 1920s and by the mid-1930s television programmes were routinely possible. All these forms have been available to domestic consumers using easily affordable equipment for some time. In short, they are everyday commonplaces. The creative content growing from the technologies in question is mature but none have had their full creative potential realised let alone exhausted. All have been refined and developed as creative forms since the technology made them possible. Will multimedia be the same and develop over years, decades and even centuries or will it go the way of the telegram, telex or the newsreel and become a technologically enabled form quickly eclipsed by subsequent developments? Will the CD-ROM and its silver successors become the 'new papyrus'[1] or just a footnote in the history of creative forms? Will the development of multimedia forms and the growth of broadband network distribution be a creative revolution, a Gutenburg shift or just more noise? The answer to this question lies in the hands of creative people in design studios, in garages, in University computer rooms, in front of screens all over the world.

[1] *Multimedia: The Complete Guide*, London: Dorling Kindersley, 1998, p.8.

A common sense description of multimedia might be the ability to combine the creative possibilities of radio and television programmes, newspapers, books, magazines, comic books, animated films and music CDs into one set of computer files accessed by the same piece of software to provide an integrated seamless experience where user input to some extent determines the manner in which the material is accessed. It is therefore interactive. The computer's ability to have rapid random access to the files which constitute this material makes the linear model of the radio or television programme seems old fashioned and limited. Interactivity (over and above the level of interactivity involved in the very act of reading or viewing a fixed, linear text) wherein the user to some extent determines the text or, more accurately, the order in which the text unfolds, offers great creative potential. It also offers a great creative challenge with the notion of an interactive movie very likely to give the traditional scriptwriter something approaching a migraine headache to the power of 5.

Multimedia already means different things to different types of creative people. Film and television programme makers quickly saw the possibility for interactive narratives with the arrival of the laser disc in the 1980s. Yet interestingly there are no great examples of this form to point to as creative landmarks. Graphic designers who began to work with computers to create two dimensional printed work, saw the ability to animate pages, to add sound and video as revolutionary extension of what they do and they found in the World Wide Web a distribution medium undreamt of. Fine artists have enthusiastically embraced the technology (Laurie Anderson, *Puppet Motel*, 1995[2] and Zoe Beloff, *Beyond*, 1996[3]) and will continue to act as the leading edge of development in a way we may not understand the significance of for many years. Animators often respond to the vision of multimedia by saying 'We've been doing this for years'. The only difference arising with the new technology is random access/interactivity. In fact, there is a case to be made that animation is the most helpful paradigm to apply as we seek to understand and create multimedia from the technical possibilities offered. It is the closest analogue analogue.

Interactivity

Traditional animation and film and television has a linear text unfolding in only one way, one narrative order, that determined by the creator. With the arrival of multimedia new possibilities have come to the fore: first hypertext and now hypermedia. This important notion presents the greatest creative challenge since the arrival not of printing but of writing. The move from linear analogue forms to digital non-linear creative structures will be a gradual one driven by organic imaginative growth over relatively long periods of time. New multimedia forms will grow, as all

[2] http://voyager.learntech.com/cdrom/catalogpage.cgi?puppet

[3] http://www.users.interport.net/~zoe/

new media do, from their immediate predecessors. Television drama grew from its radio, cinema and theatre origins over a period of decades but is now a creative form itself.

Old wine in new bottles?

Digital tools are now routinely used to create traditional linear product. The ubiquitous word processor gives the writer increased non-linear manipulative possibilities in construction but the product that emerges from this process is an orthodox text consumed, as the author intended, in a defined linear way. A similar process occurs in film, video and animation production where digital 'nonlinear' editors are now used to create traditional linear narratives. High quality contemporary audio visual product is shot on film for its still superior image quality and contrast ratio, but this film is digitised and edited 'non-linear' because of the creative flexibility and speed afforded while exhibition is still on film because of the quality and reliability. It is a case of the best horse for the course. Sitting in front of a computer screen is clearly not the best way to read *War and Peace*, which is doubtless best enjoyed as a printed book but teaching children to read using a speaking multimedia book running on a computer which records their progress adds value and creates a new order of experience.

The encyclopaedia, whether printed book(s) or silver disc, has always been interactive. Its contents are accessed as the user wishes, in a different order for each user with certain devices to help access. The content of the traditional printed reference work consisted of text and images. The digital multimedia version can add sound, it can animate the illustrations and add full motion video. Its search tools can be incredibly sophisticated and its content can be arranged and presented in a variety of ways. Crucially however, it still needs a huge team to produce the content. This content is fundamental and its creation requires a vast repertoire of traditional production skills ranging from making phone calls through organizing and managing teams to new skills such as cyberspace architecture. In this regard multimedia production is most like movie making, a team endeavour where technical specialists from best boy through gaffer and re-recordist are orchestrated by production managers and producers to realise the creative vision of scriptwriters and directors. In multimedia these roles are still evolving.

If a creative infrastructure is to develop to unleash the potential of multimedia, one must develop the equivalent skills of the producers, directors and the script writers who toil in the television and film industry. It is worth making the point here that television as it exists is already mixed media. The electronically transmitted moving pictures which define television are routinely mixed with voice and music, stills, text and graphics. In fact the only thing really missing for making the mixed media of present day television into multimedia is the element of interactivity. The viewer sits before the screen and television flows past in all its ephemeral glory. Even if you record it, you can only spin backwards and forwards very ponderously without being able to access additional audio-visual material or shift from motion video to text. Television then already has all the elements of multimedia save the ability to view the material in any order the viewer chooses. True interactivity is still missing.

The role of the graphic design profession in multimedia development should not be underestimated. The typographical knowledge and skills, the sense of page layout applied to screen design and the ability to communicate concisely in visual language are all skills which are essential to the creation of any multimedia product. Multimedia screens or web pages all need this creative skills if they are to look anything but amateur.

Taxonomy of multimedia

The technical problems have been solved for the most part, creative challenge remains great. What is likely to be created can be seen by a brief taxonomy of existing multimedia products. From these cave paintings of the multimedia age we may be able to glimpse the mature forms that will emerge.

As mentioned above, the reference book, be it the dictionary or encyclopaedia, has been an obvious type of product to produce in multimedia format. In these early days, the interactivity offered by multimedia seems to work best if delivered using the book model which has always been both linear and interactive. The multimedia encyclopaedia is already with us. In the reference book you have an ideal multimedia product which allows for interactivity, indeed demands it. However, a multimedia atlas will never replace the printed version if the maps are not very good and the production of high quality maps is not a task to be undertaken lightly. A searchable illustrated multimedia pronouncing dictionary adds value but does it add enough to lure the consumer to part with cash? Virtual museums and galleries can give us access to images and objects physically distant and multimedia technology can be used in the same physical museum space to enhance the actual physical exhibition.

Education is an area where the potential for development is enormous especially as the young are the most likely adopters of new digitally enable products. 'Living books', children's stories which read to their user and help them to learn to read in the process are already among us. Mathematics and drawing packages already fill Christmas stockings put there by dutiful parents acting out of the same noble motives that used to keep door to door encyclopaedia salesmen in business. Languages can be taught with multimedia methods and huge, entertaining databases can make all the sciences accessible and exciting.

Computer games be they action adventure, puzzle or simulation already generate more income than Hollywood films and have the added benefit of creating a generation of enthusiastic users unafraid of technology. Some of these end users will go on to become sophisticated creators of multimedia product.

However, in terms of adult product for the mature, non-games playing male and female audience there is still a golden opportunity to produce a mass market product. Interactive narrative is often mentioned in this context but this promise remains unfulfilled since it was first made possible with the laser disc in the 1980s. Simply stated there has been no 'killer application' in multimedia. One can talk of software products or games which define an era but there are no multimedia equivalents, no paradigm creating breakthroughs. The form is more difficult both to create and consume, it is demanding but it equally has more potential. The challenge remains; the opportunity is still there.

Conclusion

We live in interesting times, times when the young and the creative are taking the new digital technologies and appropriating them to their own uses; uses that could never have been imagined by those who developed the technologies. To work today in the creative uses of digital media is akin to what it must have been like to work in the film business with Eisenstein or Griffith as they defined the working practices and the very grammar of the film form, or what is was like to have been working in television during the fifties and early 1960s when that form defined itself creatively as a distinct medium different from both radio and the cinema. The situation is all the more complicated today as technologies very often become obsolete before they can be fully understood and used. Whatever the technology and however great the pace of change and development, however much the unexpected happens, analogue creativity must inform digital technologies. We who use any communications media whether new or old, digital or analogue: we still must have something interesting and worthwhile to say. A mastery of the medium and its techniques is crucial but so is vision and creative daring, imagination and flair and the ability to work hard to make visions real.

Annotated Bibliography

The references that follow have been selected as suitable sources of additional information, or to provide useful background. Since this is an introductory text, we have concentrated on books and tutorial articles, and have made no attempt to provide a guide to the research literature.

Because of the changing and ephemeral nature of the World Wide Web, we have only included a few Web pages among our references. For the most part, these contain standards documents, on the assumption that their location is likely to remain fixed for some time. The Web site that accompanies this book, at `www.wiley.com/digital_multimedia`, includes an extensive collection of links to relevant Web sites.

We have adapted the convention used for asides in the body of the text to indicate the nature of some of the references. Where the annotation begins with ⇨, the book in question is only tangentially relevant; where it begins with ⊃, its content is of a technical nature, and will require some programming or mathematical expertise.

[Ado90a] Adobe Systems Incorporated. *Adobe Type 1 Font Format.* Addison-Wesley, 1990.

➲ This specification provides the information on how PostScript fonts are stored, and what hints can be provided by the font designer.

[Ado90b] Adobe Systems Incorporated. *PostScript Language Reference Manual.* Addison-Wesley, 2nd edition, 1990.

➲ Most of the time, most people don't want to know what is going on inside a PostScript file, but for the times when you do, the answers are all here. While this book is the specification, there are also more accessible introductions to using PostScript if you ever need to.

[Ado93] Adobe Systems Incorporated. *Portable Document Format Reference Manual.* Addison-Wesley, 1993.

➲ In case you ever wondered what PDF is actually like.

[Ado97] Designing multiple master typefaces. Adobe Technical Note 5091, Adobe Systems Incorporated, 1997.

This note contains an overview of the multiple master technology and guidelines for designing font families based on it.

[App93a] Apple Computer Inc. *Demystifying Multimedia: A Guide for Multimedia Developers from Apple Computer Inc.* Apple Computer Inc, 1993.

⇨ While this book, which is directed primarily at managers, has negligible technical content and is dated (it is mostly about CD-ROM production), it is good on the process of multimedia production — especially on the different jobs involved — and remains worth reading. It includes interviews with people working in multimedia, and case studies.

[App93b] Apple Computer, Inc. *Inside Macintosh: Text.* Addison-Wesley, 1993.

➲ Most of this book is the reference manual for the (soon to be obsolete) MacOS text APIs, and therefore is of limited interest. However, the introduction contains one of the best available treatments of scripts and writing systems, and the difficulties they present for displaying text on computers for an international audience. Worth looking at, therefore, if you come across a copy.

[Bar77] Roland Barthes. *Image - Music - Text.* Fontana, 1977.

Nobody who is at all interested in the cultural aspects of media and multimedia can afford to neglect this collection of essays. Barthes has written several other books, and if you find his ideas interesting and wish to pursue them further, you should track them down.

[Bax94] Gregory A. Baxes. *Digital Image Processing: Principles and Applications.* John Wiley & Sons, 1994.

Most books on image processing (apart from introductions to tricks you can get up to with Photoshop) are highly mathematical and mainly concerned with image analysis and recognition, so this book is especially valuable, as it covers a range of image manipulations in an accessible way. Many examples of the effects of the processes on sample images are included.

[Bla92] Lewis Blackwell. *20th Century Type.* Laurence King Publishing, 1992.

An extremely good history of the important developments in typography this century. Anyone who has any interest in fonts and typography should read this book.

[BLFM98] Tim Berners-Lee, Roy T. Fielding and Larry Masinter. Uniform Resource Identifiers (URI): Generic syntax. RFC 2396, IETF, 1998.

➲ The formal definition of URIs, which includes URLs as a subset.

[Bra96] Scott O. Bradner. The Internet standards process — revision 3. RFC 2026, IETF, 1996.

How a document becomes an Internet standard: this description of the process provides an insight into what it means to be a standard.

[Car97] Rob Carter. *Experimental Typography.* Working With Computer Type 4. RotoVision, 1997.

More of a guide than other books on the subject, which tend to be mere showcases. This book includes a description of the rules traditionally used for layout, and then goes on to offer advice about breaking them. There are plenty of examples.

[CHW99] Jon Crowcroft, Mark Handley and Ian Wakeman. *Internetworking Multimedia.* Taylor & Francis, 1999.

➲ A comprehensive account of the issues we described briefly in Chapter 15, including multicasting, protocols and videoconferencing.

[Cli94] Stafford Cliff. *The Best in Cutting Edge Typography.* Quarto, 1994.

This book captures the grunge typography movement at its peak, with lots of examples including record covers, logos and stationery, posters and advertisements, all showing type being used as a graphic element in new ways, partly because of the possibilities opened up by digital fonts.

[Con87] Jeff Conklin. Hypertext: An introduction and survey. *IEEE Computer*, 20(9):17-41, September 1987.

An early standard survey reference on hypertext and hypermedia, which has not dated as much as you might expect. Although the World Wide Web has appeared and grown up since the publication of this paper, many of the issues Conklin considers remain relevant, and are only now being addressed with the development of XML.

[CSS] Cascading Style Sheets, level 2, `http://www.w3.org/TR/REC-CSS2`.

➲ This document defines the CSS language, including absolute positioning facilities. Hopefully, by the time you read this, Web browsers will have correctly implemented it.

[DOMa] Document Object Model (DOM) level 1 specification, `http://www.w3.org/TR/REC-DOM-Level-1`.

➲ Unfortunately, this document only defines the basic parts of the DOM, which has left browser manufacturers free to implement the interesting bits as they see fit. See also [DOMb].

[DOMb] Document Object Model (DOM) level 2 specification, `http://www.w3.org/TR/WD-DOM-Level-2`.

➲ Level 2 of the DOM defines the parts of the model dealing with events and stylesheets, i.e. the parts used in 'dynamic HTML'. The standard is not, at the time of writing, much observed, but should be the shape of the DOM in future.

[ECM97] ECMA. *ECMAScript: A general purpose, cross-platform programming language.* Standard ECMA-262, 1997.

➲ The reference manual for the language usually known as JavaScript. Does not describe any host systems, so cannot be used as a complete reference for Web scripting.

[EF99] Elaine England and Andy Finney. *Managing Multimedia: Project Management for Interactive Media.* Addison-Wesley, 2nd edition, 1999.

The complexity of many multimedia projects implies the need for management — a topic we have barely touched on. This book is devoted to it.

[Fau98] Christine Faulkner. *The Essence of Human-Computer Interaction.* Prentice-Hall, 1998.

A good, concise survey of mainstream thinking about HCI and usability.

[FGM$^+$97] Roy T. Fielding, Jim Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen and Tim Berners-Lee. Hypertext Transfer Protocol — HTTP/1.1. RFC 2068, IETF, 1997.

➲ Full details of the latest version of HTTP, with all the request and response headers, and status codes.

[Fla97] David Flanagan. *JavaScript: The Definitive Guide.* O'Reilly & Associates, Inc., 2nd edition, 1997.

➲ There are many books currently available on JavaScript, JScript, or DHTML, but this is one of the best, providing a clear and thorough account of the JavaScript language and its uses on the World Wide Web. The book does not cover the standards (ECMAScript and the DOM), but it does cover the real implementations as they stood at the time of its publication. Expect a revised edition when the browser manufacturers come into line with the W^{3}C Recommendations.

[FvDFH96] James D. Foley, Andries van Dam, Steven K. Feiner and John F. Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley, 2nd edition, 1996.

➲ This well-established academic text contains a thorough account of the theory of computer graphics. It concentrates largely on vector graphics, and includes an extensive account of all aspects of 3-D graphics. The authors' approach is mainly mathematical, and many readers will find it dry. Available in both Pascal and C editions.

[Got98] Veruschka Gotz. *Digital Media Design: Color and Type for the Screen.* RotoVision, 1998.

An excellent succinct introduction to the issues raised by the differences between display and print in connection with colour and typography. Instead of laying down rules based on established practice, the author encourages us to look at digital media in their own right, and to take advantage of their special qualities.

[GQ99] Ian S. Graham and Liam Quin. *XML Specification Guide.* John Wiley & Sons, 1999.

➲ The title parses as 'a guide to the XML specification': the book is mostly an annotated version of the specification [XML]. For anyone seriously interested in XML, this is essential reading.

[Gra96] Ian S. Graham. *HTML Sourcebook.* John Wiley & Sons, 4th edition, 1998.

One of the better books on HTML. Although there are many others to choose from, this is clear and accurate.

[Gra97] Ian S. Graham. *HTML Stylesheet Sourcebook.* John Wiley & Sons, 1997.

A good description of CSS. It includes a clear presentation of the arguments in favour of stylesheets, a description of CSS1 and most of CSS2 (although this was only a proposal at the time this book was written), with many examples of the layout and typographic effects that can be achieved. Includes rather too much description of the bugs in browsers' implementations of CSS, which should be out of date by now.

[HF97] Steven Heller and Anne Fink. *Faces on the Edge: Type in the Digital Age.* Van Nostrand Reinhold, 1997.

A collection of experimental typefaces, made using digital technology. The book itself is an example of some of the layout ideas that go with experimental faces.

[HTMa] HTML 4.0 specification, `http://www.w3.org/TR/REC-html40`.

➲ The current HTML specification is based on an SGML DTD, which makes it tough reading if you are not familiar with SGML or XML. This is the standard, and you should try to stick with it, even though many media-rich Web pages rely on non-standard elements, such as `EMBED`.

[HTMb] Timed Interactive Multimedia Extensions for HTML (HTML+TIME), `http://www.w3.org/TR/NOTE-HTMLplusTIME`.

➲ This document only has the status of a 'note', and the proposal has not been adopted in this form by the W^{3}C, but it is worth reading, nevertheless, especially since its sponsors include some heavyweight and influential software manufacturers.

[HW96] Jed Hartman and Josie Wernecke. *The VRML 2.0 Handbook: Building Moving Worlds on the Web.* Addison-Wesley, 1996.

➲ A tutorial and reference on VRML, showing how the language is used to construct models, arrange them into scenes and add interactivity.

[ICC97] ICC profile format specification, version 3.4. International Color Consortium, 1997.

➲ The complete definition of the format of the profiles used by colour management software, such as ColorSync. Much of it is concerned with print issues, but multimedia authors must be increasingly aware of colour management. The specification is presently available in PDF from the ICC's Web site at `http://www.color.org/profiles.html`.

[ISO] Introduction to ISO, `http://www.iso.ch/infoe/intro.htm`.

ISO's own account of their history and rôle in the standards process. The page includes a good discussion of the nature of standards.

[JMF94] Richard Jackson, Lindsay MacDonald and Ken Freeman. *Computer Generated Color: A Practical Guide to Presentation and Display.* John Wiley & Sons, 1994.

A short introduction to the perception, representation and display of colour, covering physics, psychology and physiology, as well as computational topics. The book is arranged as a series of short articles, which gives it a superficial flavour, reminiscent of popular science magazines, but it actually contains much useful information on a topic that is generally somewhat neglected.

[Kaw92] Bruce F. Kawin. *How Movies Work.* University of California Press, 1992.

Film shares many qualities with multimedia, but it is a more mature technology, with well-developed conventions and working practices. Understanding film can provide useful insights into multimedia, and possibly show us some ways in which it might develop. This book is a comprehensive introduction to all aspects of film. (The author is a professor of English and film studies, so his account of the importance of critics should be taken with a pinch of salt.)

[Kos98] Dave Kosiur. *IP Multicasting: The Complete Guide To Interactive Corporate Networks.* John Wiley & Sons, 1998.

➲ Despite the intimidating sub-title, this is actually one of the most readable books on networking that we know. It clearly describes the motivation behind multicasting as well as the technology itself, including the main protocols used for multimedia.

[Lay98] Kit Laybourne. *The Animation Book.* Three Rivers Press, 'new digital' edition, 1998.

An excellent practical introduction to the techniques and equipment used in animation. As the 'new digital' tag indicates, the current edition takes account of the impact of computers on the animator's craft, but it still includes an extensive coverage of traditional media, and the author is at pains to emphasize the common underpinnings of both approaches.

[LH99] David Lowe and Wendy Hall. *Hypermedia and the Web: An Engineering Approach.* John Wiley & Sons, 1999.

The authors of this book believe that the application of software engineering principles to multimedia production will improve quality and reduce costs. This seems to be a somewhat dubious claim from several points of view, but if you share their faith, you may find this book helpful.

[Lis95] Martin Lister, editor. *The Photographic Image in Digital Culture.* Routledge, 1995.

A collection of analytical essays, which are relevant inasmuch as the 'photographic image' is broadly defined to include video, and some of the contributors consider interactivity, too.

[McC96] Malcolm McCullough. *Abstracting Craft: The Practiced Digital Hand.* MIT Press, 1996.

The expression 'thought-provoking' is overworked, but it is hard to describe this book any other way. The author raises many questions about the relationship between traditional craft and digital technology, and, indeed, between people and digital technology. His answers may not always be convincing, but the issues are of considerable importance to anybody working in multimedia. If you only read one book from this list, this should be it.

[MS99] Daniel Minoli and Andrew Schmidt. *Internet Architectures.* John Wiley & Sons, 1999.

➲ A classic example of network writing, full of acronyms, neologisms, and poorly constructed sentences, this book is hard going, but contains valuable information about where the Internet is heading.

[Mv96] James D. Murray and William vanRyper. *Encyclopedia of Graphics File Formats.* O'Reilly & Associates, Inc., 2nd edition, 1996.

➲ The bulk of this book comprises descriptions of the internal layout of nearly 100 graphics file formats, including those commonly encountered in multimedia (GIF, JFIF, PNG, etc.). As such, it will only be of interest to programmers. The introductory chapters provide a useful review of some aspects of computer graphics and a general description of how the necessary information can be stored in a file. Extensive pointers to defining documents, standards, and so on are included, and the book has a companion Web site, where updated information and descriptions of new formats are available.

[NG96] Mark Nelson and Jean-Loup Gailly. *The Data Compression Book.* M&T Books, 2nd edition, 1996.

➲ A general account of data compression algorithms, aimed clearly at programmers — all the main algorithms are implemented in C in the text. The emphasis is on practicalities — this book will not satisfy mathematicians. Among the algorithms included are Huffman, Lempel-Ziv in all varieties, and JPEG.

[Oha93] Thomas A. Ohanian. *Digital Nonlinear Editing.* Focal Press, 1993.

Although quite badly dated, this book provides useful insights into digital editing, from the perspective of film editing in particular.

[O'R98] Michael O'Rourke. *Principles of Three-Dimensional Computer Animation.* W.W. Norton & Company, revised edition, 1998.

A good accessible survey of 3-D techniques.

[Poh95] Ken C. Pohlmann. *Principles of Digital Audio.* McGraw-Hill, 3rd edition, 1995.

➲ Since this is the only serious technical book on digital audio, it's fortunate that it is very good.

[Poy96] Charles A. Poynton. *A Technical Introduction to Digital Video.* John Wiley & Sons, 1996.

➲ More technical than introduction, this is the best book on digital video technologies; it also provides the analogue background in some detail. Sadly, it is now somewhat out of date. A second edition, due to be published early in 2001, will be called 'Digital Video and HDTV: Pixels, Pictures and Perception'. If it lives up to the promise of the existing edition, it should be an essential reference for all aspects of digital video.

[RM94] Francis Rumsey and Tim McCormick. *Sound and Recording: An Introduction.* Focal Press, 2nd edition, 1994.

⇨ A practical introduction to the techniques of sound recording — what goes on before digitization.

[RM96] Paul Resnick and James Miller. Pics: Internet access controls without censorship. *Communications of the ACM,* 39(10):87–93, 1996.

A tutorial introduction to the mechanics of PICS, and the rationale behind it. Also available online at `http://www.w3.org/PICS/iacwcv2.htm`.

[Rot92] Joseph Rothstein. *MIDI: A Comprehensive Introduction.* Oxford University Press, 1992.

The description of MIDI software is out of date in the details, but overall, this is a straightforward, accurate and accessible description of the MIDI protocol and how it is used.

[SACM] Michael Stokes, Matthew Anderson, Srinivasan Chandrasekar and Ricardo Motta. A standard default color space for the internet — sRGB, `http://www.w3.org/Graphics/Color/sRGB`.

➲ The document that defines the controversial sRGB colour space.

[Sas93] Rosemary Sassoon, editor. *Computers and Typography.* Intellect Books, 1993.

A nice little collection of essays, which concentrates on the relevance of established typographical practice to digital typesetting.

[SCFJ96] Henning Schulzrinne, Stephen L. Casner, Ron Frederick and Van Jacobson. RTP: A transport protocol for real-time applications. RFC 1889, IETF, 1996.

➲ Full details of RTP and RTCP. Note that a slightly revised draft of this document was circulated in 1998, which will form the basis of an Internet standard.

[Sma99] Peter Small. *Lingo Sorcery.* John Wiley & Sons, 2nd edition, 1999.

This distinctly eccentric book should be approached with some caution, especially, perhaps, by non-programmers. The author presents a view of object-oriented programming in the context of Director, but without the perspective of conventional object-oriented programming, the result is ...odd. Nevertheless, he succeeds in showing how Lingo can be used in ways that are not immediately obvious, to produce complex multimedia applications.

[SMI] Synchronized Multimedia Integration Language (SMIL) 1.0 specification, `http://www.w3.org/TR/REC-smil`.

➲ The formal definition of SMIL, as an annotated XML DTD.

[SRL98] Henning Schulzrinne, Anup Rao and Robert Lanphier. Real Time Streaming Protocol (RTSP). RFC 2326, IETF, 1998.

➲ Full details of RTSP, with all the methods, status codes, and so on.

[SWDB98] Sean C. Sullivan, Loren Winzeler, Jeannie Deagen and Deanna Brown. *Programming with the Java Media Framework.* John Wiley & Sons, 1998.

➲ Contains reference material on the Java APIs for playing audio and video, and tutorials on how to use them. Purely for Java programmers.

[TDT96] Ronald C. Turner, Timothy A. Douglass and Audrey J. Turner. *README.1ST: SGML for Writers and Editors.* Prentice-Hall, 1996.

⇨ An accessible introduction to SGML, written, as the sub-title indicates, from the perspective of writers and editors. This perspective helps clarify the arguments in favour of structural markup.

[Tho93] Roy Thompson. *Grammar of the Edit.* Focal Press, 1993.

The title may strike you as strange, if you are not used to thinking about structures in grammatical terms, but you should not be put off. This book is simply an account of different types of shots and transitions used in film and video, how they may be combined, and the effects that result. Useful both as a guide for video editing and as a reference point when considering ways of combining media, especially in time-based multimedia.

[Tud95] P. N. Tudor. MPEG-2 video compression. *Electronics and Communication Engineering Journal*, December 1995.

A very clear short description of MPEG-2 (and, by implication, MPEG-1) compression. Also available (at the time of writing) online at `http://www.bbc.co.uk/rd/pubs/papers/paper_14/paper_14.htm`.

[Vau98] Tay Vaughan. *Multimedia: Making It Work.* Osborne McGraw-Hill, 4th edition, 1998.

Vaughan covers much of the same ground as *Digital Multimedia*, but much more superficially. He does, however, deal with issues such as project management, contracts, and so on, which we have not considered. His style, which is informal and anecdotal, may not be to everybody's taste.

[VD96] Mark Cotta Vaz and Patricia Rose Duignan. *Industrial Light and Magic: Into the Digital Realm.* A Del Rey book published by Ballantine Books, 1996.

⇨ Essentially just an entertaining public relations exercise for ILM, but one that provides an insight into the extraordinary effort (and the technology) that goes into high-end digital special effects. It usefully puts desktop and multimedia post-production efforts into perspective.

[Vin92] John Vince. *3-D Computer Animation.* Addison-Wesley, 1992.

An unpretentious introduction to 3-D modelling, rendering and animation techniques.

[WAI] WAI accessibility guidelines: Page authoring, `http://www.w3.org/TR/WD-WAI-PAGEAUTH`.

Compulsory reading for anybody designing Web pages.

[Wil74] Raymond Williams. *Television: Technology and Cultural Form.* Fontana, 1974.

⇨ An influential study of the cultural aspects of television, which provides an important reference point for any attempts to analyse the cultural impact of multimedia.

[Wil93] Adrian Wilson. *The Design of Books.* Chronicle Books, 1993.

⇨ A description of the practice and values of traditional book design, from a leading designer. Pre-dates the digital revolution, but contains many things that remain relevant.

[WW92] Alan Watt and Mark Watt. *Advanced Animation and Rendering Techniques: Theory and Practice.* ACM Press/Addison-Wesley, 1992.

➲ If you want to know how ray tracing, radiosity, and other advanced 3-D algorithms work, this book will tell you, in considerable technical detail.

[XML] Extensible Markup Language (XML) 1.0, `http://www.w3.org/TR/REC-xml`.

➲ The official XML specification is dense and quite hard to read; generally, you are better off with [GQ99]. Related specifications (XPointer, XLink, XSL, and others) can be traced from `http://www.w3.org/XML`.

Index