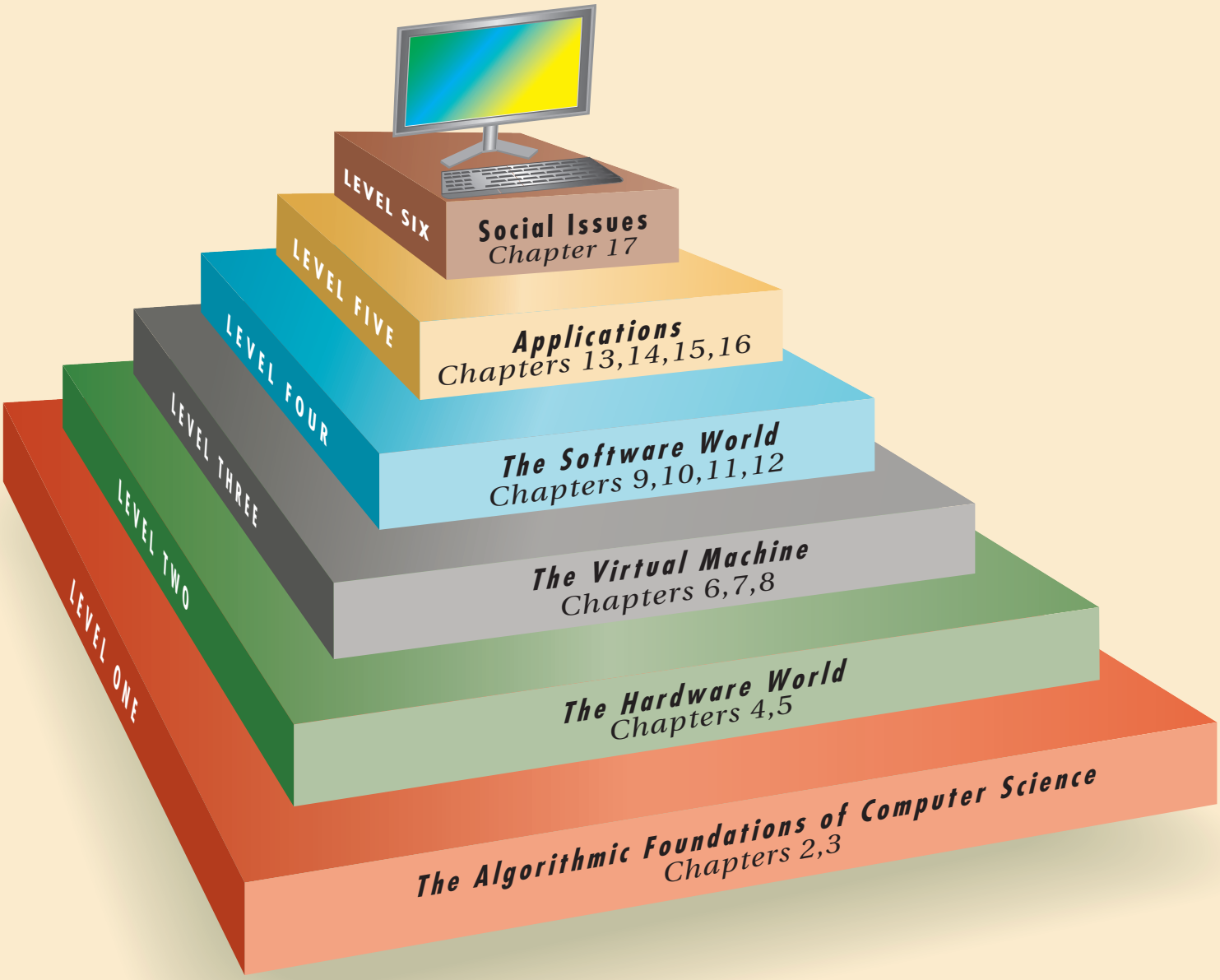

FIFTH EDITION
INVITATION TO
COMPUTER SCIENCE
G. Michael Schneider • Judith L. Gersting

5TH EDITION

Invitation to Computer Science


LEVEL SIX
Social Issues
Chapter 17
LEVEL FIVE
Applications
Chapters 13,14,15,16
LEVEL FOUR
The Software World
Chapters 9,10,11,12
LEVEL THREE
The Virtual Machine
Chapters 6,7,8
LEVEL TWO
The Hardware World
Chapters 4,5
LEVEL ONE
The Algorithmic Foundations of Computer Science
Chapters 2,3

5TH EDITION

Invitation to Computer Science

G. Michael Schneider
Macalester College

Judith L. Gersting
University of Hawaii, Hilo

Contributing author:
Keith Miller
University of Illinois, Springfield

COURSE TECHNOLOGY
CENGAGE Learning

Australia • Brazil • Japan • Korea • Mexico • Singapore • Spain • United Kingdom • United States

Invitation to Computer Science, Fifth Edition
G. Michael Schneider and Judith L. Gersting

Executive Editor: Marie Lee

Acquisitions Editor: Amy Jollymore

Senior Product Manager: Alyssa Pratt

Development Editor: Deb Kaufmann

Editorial Assistant: Julia Leroux-Lindsey

Marketing Manager: Bryant Chrzan

Content Project Manager: Jennifer K. Feltri

Art Director: Faith Brosnan

Cover Designer: RHDG/Tim Heraldo

Cover Artwork: Fotolia.com (Royalty Free), Image # 375162

Compositor: Integra

ISBN-13: 978-0-324-78859-4
ISBN-10: 0-324-78859-2

Course Technology
20 Channel Center Street
Boston, MA 02210
USA

Cengage Learning is a leading provider of customized learning solutions with office locations around the globe, including Singapore, the United Kingdom, Australia, Mexico, Brazil, and Japan. Locate your local office at:
international.cengage.com/region

Cengage Learning products are represented in Canada by Nelson Education, Ltd.

For your lifelong learning solutions, visit **course.cengage.com**
Visit our corporate website at **cengage.com**.

Printed in Canada
1 2 3 4 5 6 7 13 12 11 10 09

BRIEF CONTENTS

CHAPTER 16

Computer Graphics and Entertainment: Movies, Games, and Virtual Communities

16.1 Introduction

The first commercially marketed computer was the UNIVAC I, manufactured by Remington Rand. On March 31, 1952, the company delivered its first machine to the U.S. Census Bureau. Later systems went to the U.S. Army, the U.S. Air Force, the Atomic Energy Commission, U.S. Steel, General Electric, and CBS, which used it to predict the outcome of the 1952 presidential election. (See the special interest box in Chapter 1 entitled "Good Evening, This Is Walter Cronkite.")

In 1952 a UNIVAC I cost $1,500,000, about $12 million in today's dollars. It weighed 15 tons, contained 5200 vacuum tubes, consumed 125,000 watts of electricity (generating an enormous amount of heat), and occupied floor space equal in size to a small apartment. These early machines were prohibitively expensive and required a massive financial investment in space, power, cooling, and support staff. Because of these costs, computers of the 1950s and early 1960s were only available to large organizations and only for what were deemed "important" purposes—classified military work, corporate research and development, or governmental policy analysis. The idea of using these systems for such frivolous pastimes as playing games or watching videos would have been unimaginable.

However, conditions changed dramatically in the late 1960s due to the development of transistors and integrated circuits, introduced in Chapter 4. Computers became more compact, more reliable, and less costly. In 1965 Digital Equipment Corp. (DEC) rolled out the PDP-8, the world's first "minicomputer," a term coined to describe a computer system that was smaller and less expensive than the unwieldy mainframes of the 1950s and early 60s. A DEC PDP-8 minicomputer could be purchased for as little as $16,000 (about $100,000 in current dollars) and only took up as much space as two or three refrigerators.

Although still not cheap by today's standards, this lower price meant that computers were no longer accessible only to the military, government, and wealthy corporations; instead, they were now within the financial reach of colleges, universities, and small businesses. Some of the first computer games were created in the early 1970s by college students experimenting after hours to see what these new minicomputers were capable of doing. Games like Space Wars, Adventure, and Dungeons & Dragons were played on university computers, arcade machines, or custom-designed home consoles well before personal computers arrived on the scene.

In 1972 Nolen Bushnell, an electrical engineering graduate of the University of Utah, started a company called Atari (named after a board position in the game of Go), which released its first product in 1974, an arcade game called Pong. It was wildly successful and quickly became the most popular computer

game in the country with 40,000 units sold nationwide and hundreds of thousands of players eager to stuff coins into a slot just for the privilege of playing a primitive electronic version of Ping-Pong, as shown in Figure 16.1. (It is amazing to see how far video game technology has progressed in just 35 years!)

In 1976 Atari produced a home version of Pong that allowed users to play the game on their televisions using a console, complete with joysticks and onscreen scoring. It sold hundreds of thousands of units and became one of the most popular Christmas gifts of the late 1970s. Other games soon followed, and the decade of the late 1970s and early 1980s is termed the "golden period" of video arcade systems. By early 1982 Atari had become a $2 billion corporation and the fastest-growing company in the United States.

Using computers for entertainment (once considered frivolous and a waste of scientific resources) had by the late 1970s become an important, not to mention financially lucrative, industry, and that growth has continued unabated. Today, computer-based entertainment is a $15–20 billion industry employing tens of thousands of talented designers, artists, computer scientists, and engineers. Gamers stand in line for hours to purchase the latest and greatest video game release. Hollywood spends massive amounts of money and manpower producing computer-generated images that amaze and enthrall, all in the hope of reaping hundreds of millions of dollars in movie ticket and DVD sales. Virtual worlds enroll millions of subscribers who spend hours wandering imaginary spaces, joining online virtual communities, and making virtual friends.

Using information technology to amuse, fascinate, and frighten is no longer viewed as a waste of time, at least not by the millions who participate and play. Instead, it is seen as an application that contributes significantly to the national economy and brings enjoyment to many people. Just as the Jeep and Hummer evolved from specialized military vehicles to passenger cars used for off-road fun and adventure, so too has the computer evolved from a research tool of the military and government to something available for our personal pleasure. By the start of the twenty-first century, entertainment had

FIGURE 16.1

Pong–One of the First Computer-Based Video Games

become an application that stands alongside traditionally "important" uses of computers such as mathematical modeling (Chapter 13), electronic commerce (Chapter 14), and robotics (Chapter 15).

16.2 Computer-Generated Imagery (CGI)

16.2.1 *Introduction to CGI*

On March 2, 1933, a sellout audience at Radio City Music Hall in New York City was treated to the premiere of the science-fiction movie *King Kong*. This was the first feature-length film (rather than cartoon) to have its central character, in this case a giant gorilla, generated using animation. Animation had been used before in feature films, but only in a few short scenes or to animate minor characters. Since the movie's premiere occurred years before the appearance of the first commercial computer, Kong's movements were created using a manual technique called **stop-motion animation**. The special effects staff built a small-scale clay mockup of the creature, positioned it, and snapped a single photograph, called a **frame**. Then they made a tiny change in the position of the model to represent its location a fraction of a second later and shot another frame. This process of "move the model, shoot a frame" was repeated thousands of times and, when the frames were shown in sequence without interruption, Kong appeared to come alive on-screen. (This is similar to the "flip-book" style of animation, in which pages of a notebook are filled with drawings and riffled to produce the effect of motion.) Stop-motion special effects were used in many fantasy, adventure, and science-fiction movies of the 1940s through 1970s.

Although it is possible to produce reasonably good images using either hand-drawn frames or stop-motion animation (*King Kong* was voted one of the 100 best films of all time by the American Film Institute), both of these techniques have serious limitations. Hand-drawing frames can be a painstakingly slow process, requiring dozens or even hundreds of highly skilled artists. The most notable problem with stop-motion techniques is the difficulty of repositioning a clay model with a sufficient degree of accuracy so that the model's movements do not appear jerky and artificial. At 30 frames per second (the standard rate for video; film uses 24), one hour of stop-motion animation requires 108,000 separate images, each of which must be manually positioned and photographed. The effort required can make this a painfully slow and expensive way to create special effects.

However, until the early 1990s, there were really no other choices. As we will soon learn, using a computer to produce realistic images requires enormous amounts of computational speed and power, and the mainframes of earlier decades were generally not up to the task. In addition, the algorithms used to create realistic human and animal replicas were in their infancy and not well understood. There were some early attempts to produce computer-animated movies—for example, *Tron* (1982) and *The Last Starfighter* (1984)—but the quality of that early work was poor, and most movie directors opted to stay with stop-motion or hand-drawn animation for their special effects.

Two groundbreaking movies of the early 1990s quickly changed Hollywood's mind: *Terminator 2: Judgment Day* (1991) used a computer to create the T-100 Terminator character and the special effects used in action sequences. *Jurassic Park* (1993) used computers to create and animate the movie's dinosaurs and paste them seamlessly into the background. The

quality of those early 1990s images was an order of magnitude improvement over what had been available just 8 to 10 years earlier. Both movies were huge financial as well as artistic successes, and they clearly demonstrated the rapidly improving capability of **computer-generated imagery**, usually referred to by its acronym **CGI**. By the mid-1990s computer hardware could handle the computational demands required to create realistic three-dimensional images, and CGI software development had reached a point where its final product was as good as, if not better than, the manual output of human animators. By the beginning of the twenty-first century, CGI had become the method of choice for virtually all film and TV animation and special effects.

CGI has many advantages over manual techniques. It can produce extremely high-quality, lifelike images, called **photorealistic animation**, that are difficult to create using hand-drawn pictures or stop-motion models because of the level of detail. CGI can generate images that are prohibitively expensive to produce manually, such as massive crowd scenes containing thousands of characters. Without CGI, directors would either have to hire thousands of extras, animate the scenes by hand, or produce thousands of miniature models, all of which would be quite costly. Computers can be used to produce scenes that would be dangerous if filmed using human subjects, such as car chases and explosions. Finally, CGI produces frames using only a single animator and a single tool—the computer—instead of a team of animators, model builders, camera crew, and lighting staff. This can reduce costs and speed up the animation process.

Today, computer imaging is a multibillion-dollar industry, and the CGI budget for a wide-release feature film can easily exceed $40–50 million. Furthermore, CGI techniques have moved well beyond the Hollywood sound stage and are now used in such fields as video gaming (discussed later in this chapter), computer software, scientific and medical imaging, television, advertising, flight simulation, and the production of still images for books and magazines.

Although most of us are well aware that computers generate many of the images we see in the theater or on TV, few of us know how this is done. In the following section, we describe some of the fundamental algorithms used to produce computer-generated images.

Computer Horsepower

Industrial Light and Magic (ILM) is one of the largest visual effects companies in the world. It was founded by George Lucas in 1975 to produce special effects for his science-fiction movie *Star Wars*. ILM went on to do the CGI work for *The Empire Strikes Back, Jurassic Park, Star Trek, Back to the Future, Pirates of the Caribbean*, the Harry Potter series, and dozens of other well-known and highly profitable films. The company has won 38 Oscars for its outstanding visual effects work.

None of this would be possible, however, without an enormous amount of computing resources. At its California headquarters, ILM has the computational power equivalent of 15,000 home computers, and enough online storage to hold 5000 trillion bytes of data. These systems are interconnected by a high-speed network capable of transmitting 12 trillion bits of information per second. This massive collection of equipment draws 2.4 million megawatts of power and is cooled by 32 air-conditioning units that require 25 tons of coolant to keep the building inhabitable—all of this computing power just to make sure that the alien Death Star spaceship looks realistic while you sit back and munch popcorn!

16.2.2 *How It's Done: The Graphics Pipeline*

The production of computer-generated images is a complex subject that takes years of study to understand fully. In this short chapter we cannot possibly do justice to its many facets and details. Instead, we introduce and describe a few of the fundamental operations required to generate and display images. Our goal is to provide a basic understanding of CGI techniques and an appreciation for the powerful computers and sophisticated algorithms required to carry out this difficult task.

There is a sequence of operations that must be completed successfully to produce a realistic three-dimensional image. This sequence is termed the **graphics pipeline**. There is no agreement on exactly which steps should or should not be included, and the number of distinct items in a pipeline diagram can vary from three to eight, depending on the type of image being produced and whether certain operations are grouped together or listed separately. The following sections describe the three stages shown in the simplified graphics pipeline of Figure 16.2: object modeling, object motion, and rendering and display. (Exercise 1 at the end of the chapter asks you to find other examples of a graphics pipeline diagram to determine which steps were omitted from the version shown in Figure 16.2.)

16.2.3 *Object Modeling*

The first step in generating a three-dimensional image is **object modeling**—the creation of a mathematical or computational model of a three-dimensional object that can be stored in memory and manipulated algorithmically. There are many approaches to object modeling, but one of the most well known and widely used is **wireframe modeling**. In this technique, the object's surface, but not its interior, is represented mathematically using a set of simple polygons, usually triangles or rectangles.

For example, to create a scene containing a dolphin, we start by inputting an image of that object. There are a number of ways to provide this input—an artist could draw a picture by hand, or we could scan an existing photograph. Next, using special CGI software and an algorithm called **tessellation**, the object is subdivided into a set of plane figures that covers its surface. An

FIGURE 16.2
A Simplified Three-Stage Graphics Pipeline

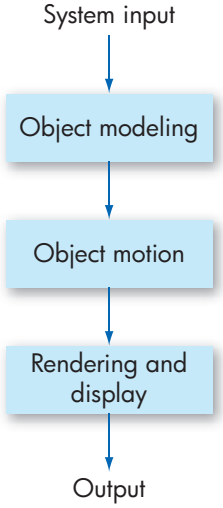
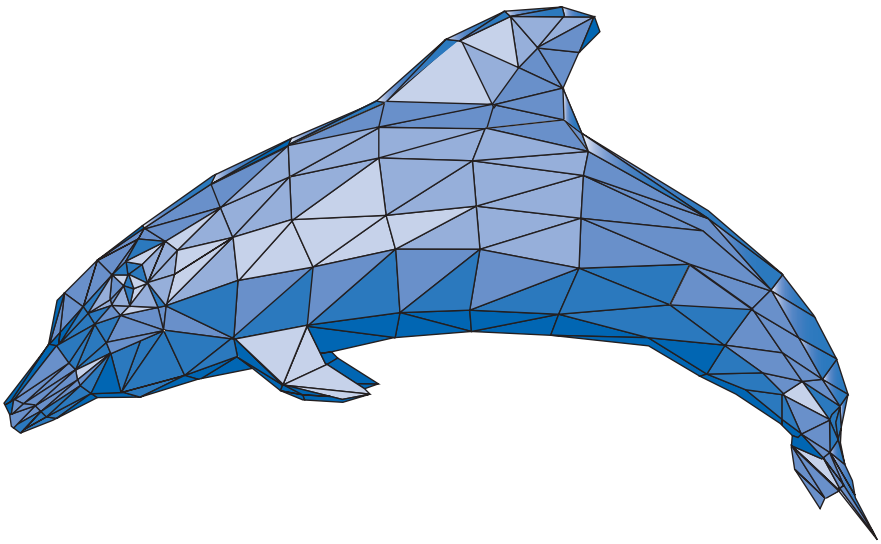


FIGURE 16.3

Wireframe Model of a Dolphin (based on image in Wikipedia entry on polygon meshes)



example of this tessellation process is shown in Figure 16.3, using triangles. Figure 16.3 also explains the reason behind the name of this technique. The polygonal outline on the surface, called a polygon mesh, produces a model that looks as if it were built from many thin pieces of wire.

Once the object's surface is tessellated, information about the polygons is stored in memory, usually in the form of a **vertex list**. This is a table giving the coordinates of each vertex on the object's surface and the identity of all other vertices to which this one is connected. In order to enter the proper (x, y, z) coordinates of each vertex, we need to know the **origin** of the coordinate system, that is, the (0, 0, 0) reference point. For simplicity, one of the vertices is usually specified as the origin. It does not matter which is chosen, as long as the computer knows its identity.

As an example, the four triangles in Figure 16.4(a) might produce the vertex list shown in Figure 16.4(b) using vertex v_1 as the origin point.

The simple four-triangle object in Figure 16.4(a) produced the vertex list of Figure 16.4(b) containing about three dozen pieces of information.

FIGURE 16.4(a)

Tesselation Producing Three Triangles and Five Vertices

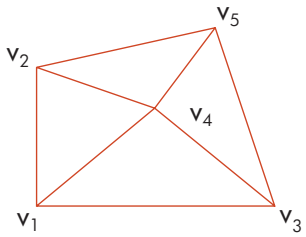


FIGURE 16.4(b)

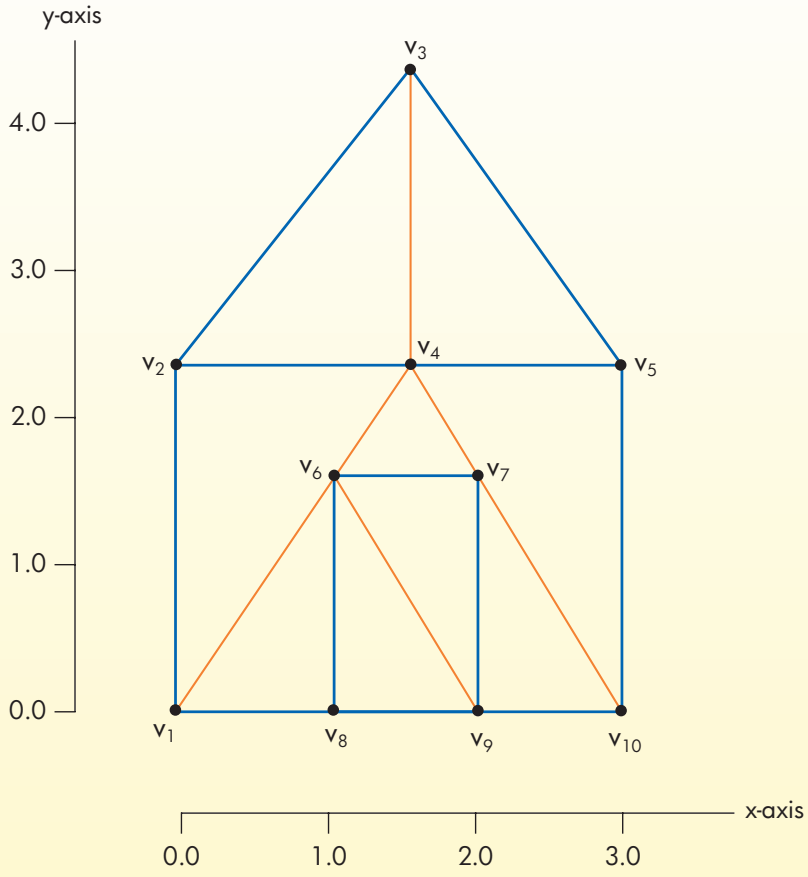
Vertex List Representation of the Model in Figure 16.4(a)

	Vertex	x	y	z	Connected to
(origin)	v_1	0	0	0	v_2, v_3, v_4
	v_2	0	1.0	0	v_1, v_4, v_5
	v_3	1.6	0	0	v_1, v_4, v_5
	v_4	0.7	0.5	0.5	v_1, v_2, v_3, v_5
	v_5	1.4	1.1	0	v_2, v_3, v_4

Realistic objects such as the dolphin of Figure 16.3 can result in enormous tables that consume huge amounts of memory and massive amounts of computer time. Furthermore, our dolphin may be only one of hundreds of objects (e.g., rocks, coral, algae, fish, water, sky) present in a single frame, all of which must be modeled and stored. You can now begin to understand the reasons why CGI places such huge processing and storage demands on a computer system.

PRACTICE PROBLEM

The following is a polygonal mesh representation of a two-dimensional drawing of a house created using tessellation:



Using v_1 as the origin, show the vertex list generated by this wireframe model. (*Note*: Since this is a two-, rather than three-, dimensional model, the z entry in each column of Figure 16.4(b) will be 0.) How many distinct pieces of information are required to store the information about this model?

16.2.4 *Object Motion*

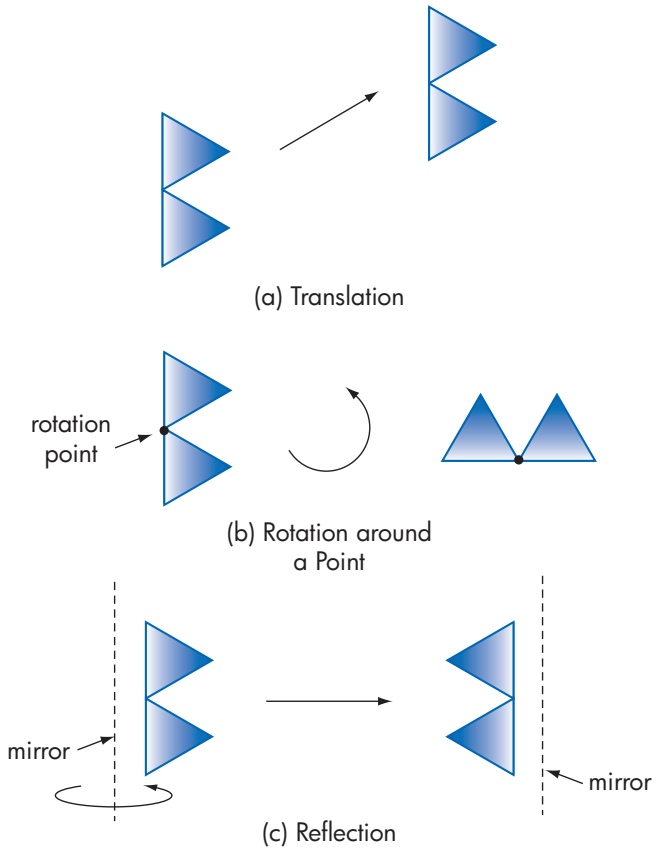
After we have captured and stored a model of our object in a computational format, we can begin the next stage in the graphics pipeline of Figure 16.2—moving that object to its proper position in the next frame.

There are three types of **rigid motion** (motion that does not bend or deform an object): translation, rotation, and reflection. These three motions are illustrated in Figure 16.5.

Translation, shown in Figure 16.5(a), is the lateral (up, down, right, left, in, out) movement of every point in an object by the same amount and in the same direction. **Rotation**, Figure 16.5(b), is the circular movement of an object around some fixed axis of rotation, much as a merry-go-round horse revolves around the ride's central mechanism. Finally, **reflection**, Figure 16.5(c), is a special type of rotation. It produces a mirror image of an object such that every point in the reflected image is the same distance from the mirror as the original object, but on the opposite side of the mirror.

The movement of an object from its location in one frame to its new location in the next is often described not by a single type of motion but a combination of two or more of these basic operations. For example, to model the motion of an airplane taking off and banking to the left, we might use translation to move the airplane forward and upward in space and rotation to model the turning operation. (*Note*: Some movements cannot be described using just these three operations. Motions that deform or change the shape of an object, such as scaling, squeezing, stretching, or ripping, are widely used in computer graphics, but they need additional operators not described here.)

FIGURE 16.5
The Three Types of Rigid Motion



To implement these three motions, we use a mathematical structure called a **transformation matrix**. When a vector containing the (x, y, z) coordinates of a single vertex point is multiplied by the matrix, the result is a new vector containing the translated, rotated, or reflected (x′, y′, z′) coordinates of that vertex point in the next frame. This same multiplication operation is then applied to every vertex in the vertex list, generating the new coordinates for the entire object. Thus, in CGI the abstract concept of motion is defined in terms of matrix multiplication, an algorithmic operation easily programmed on a computer. However, even though it may be easy to implement, the potentially huge number of multiplications can make this a time-consuming task. Animating an object containing thousands of vertices, like the dolphin of Figure 16.3, can require millions or even billions of arithmetic operations. Again, we can begin to understand and appreciate the need for high-performance computers in the field of CGI.

Let's illustrate how this is done using translation, the straight-line movement of a single point. To move a single vertex point located at coordinates (x, y, z) to a new position at location (x + a, y + b, z + c), we multiply the current coordinates by the 4 × 4 **translation matrix** in Figure 16.6. (See pages 343–344 for an explanation of matrix multiplication.) The results of this operation are the coordinates of this vertex point in the next frame after it has been moved by *a* units along the x-axis, *b* units along the y-axis, and *c* units along the z-axis.

After the operation of Figure 16.6 has been applied to every vertex in the vertex list, the entire object will appear to move laterally as a single unit. This behavior is diagrammed in Figure 16.7, in which the object of Figure 16.4(a) is moved *a* units right in the x-direction and *b* units up in the y-direction. (Assume zero movement in the z-direction to make the picture easier to visualize.)

What actually happened in Figure 16.7 is that the (x, y, z) coordinates of each of the five vertices in the vertex list of Figure 16.4(b)—i.e., columns 2, 3, and 4—were multiplied by the 4 × 4 translation matrix in Figure 16.6, with *c* set to 0 since there is no movement in the z-direction. The newly generated (x′, y′, z′) coordinates of each vertex point are copied back into columns 2, 3, and 4 of the vertex list, replacing the old coordinates. Now, when this object is displayed in the next frame it will be in its proper location. If we repeat this operation 30 times, moving the object a tiny amount each time (i.e., using small values for *a*, *b*, and *c*), then when these 30 frames are shown in sequence, the result will be one second of animated motion. Our eyes will not see 30 separate and distinct movements like those in Figure 16.7, but one second of smooth, flowing motion.

Both rotation and reflection operations are implemented in a similar way, but using the appropriate rotation or reflection matrix in place of the translation matrix shown in Figure 16.6. (Exercises 7 and 8 at the end of the chapter ask you to determine what these two matrices look like when working in two dimensions rather than three.)

FIGURE 16.6
Using Matrix Multiplication to Implement Object Translation

$$\begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + a \\ y + b \\ z + c \\ 1 \end{bmatrix}$$

Translation Matrix *Current Vertex Coordinates* *New Coordinates*

One of the advantages of CGI over manual systems is that a computer can perform many of the required operations without the assistance of a human designer, speeding up the animation process. For example, assume the translation motion in Figure 16.7 takes place over one second. At 30 frames per second, the standard rate for video, an animator would need to generate 30 frames to obtain the desired effect. However, using a CGI technique called **keyframing**, a human animator only needs to produce the first frame, containing the starting location of the object; the last frame, containing the final location of the object; and the elapsed time, in this case one second. Using this information, a computer can automatically generate the 28 required intermediate frames, called **in-between frames** or, more simply, **tweeners**. The computer adds 1/29th of the distance between the object's location in the first and last frames to the coordinates of the object in the current frame to position it correctly, since with N total frames there are $N - 2$ in-between frames and $N - 1$ intervals. The work of the animator is reduced from creating 30 frames to creating two, the first and the last. This is a huge shift in workload from human being to computer.

In our discussions of motion we have moved the entire object as a single entity in relation to a single origin point. For example, the polygon in Figure 16.7 moved up and to the right as a complete unit. However, sometimes we want to move different parts of an object in relation to different points or axes, rather than one, in order to achieve a specific effect. For example, Figure 16.8 shows a figure-8 object with two axes of rotation, labeled A and B, with A lying outside the object and B lying at the center of the right circle. (*Note*: Assume the axes of rotation are parallel to the z-axis and are coming out of the page.)

FIGURE 16.7

Example of a Translation Performed on the Object Shown in Figure 16.4(a)

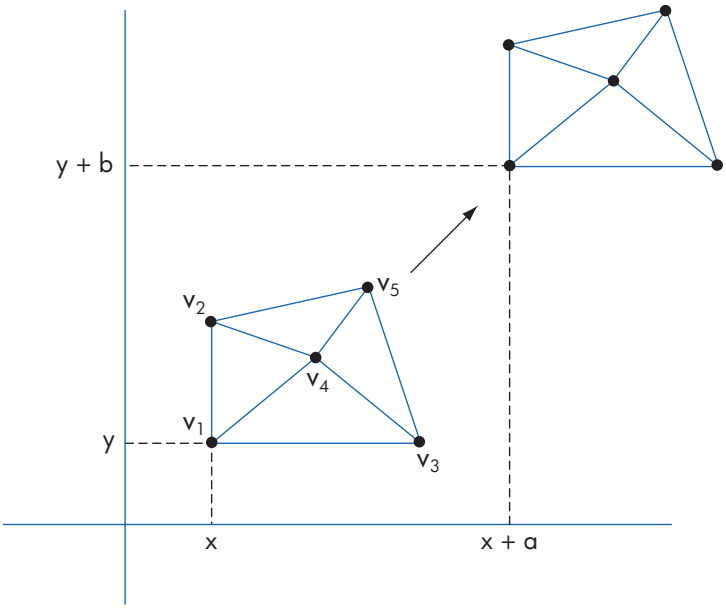
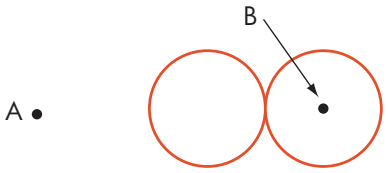


FIGURE 16.8

Figure-8 Object



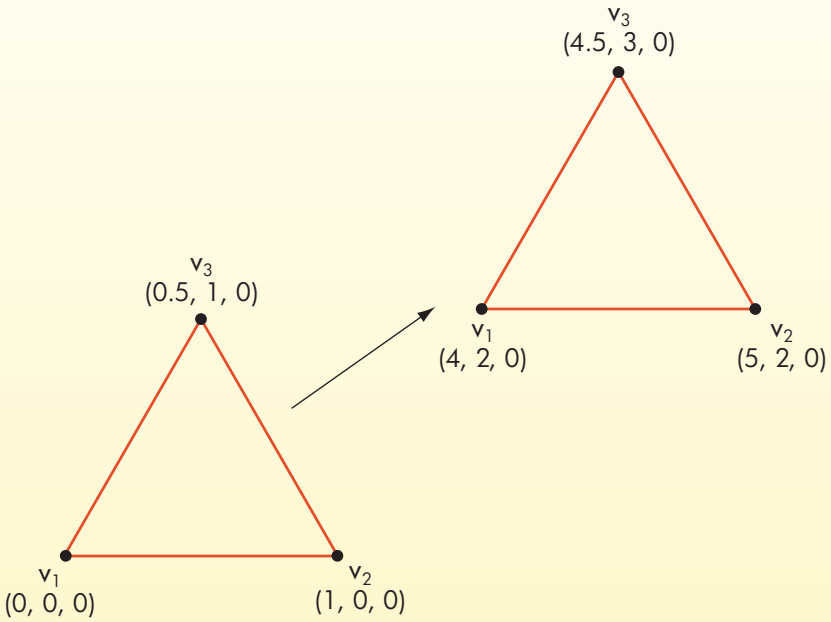
PRACTICE PROBLEM

Assume that you want to animate the lateral motion of a triangle from its current location, in which:

vertex v_1 is at (0, 0, 0), vertex v_2 is at (1, 0, 0), and vertex v_3 is at (0.5, 1, 0) to a new location such that

vertex v_1 is at (4, 2, 0), vertex v_2 is at (5, 2, 0), and vertex v_3 is at (4.5, 3, 0)

This movement is shown in the following diagram:



This motion will take two seconds to complete. Show the translation matrix that can be used to generate all the necessary in-between frames.

If we perform a rotation operation around axis B on just the rightmost circle, that circle rotates like a wheel around its axle. If we do a second rotation on the entire figure-8 object, this time using axis A, the figure-8 flies around A like the earth around the sun. We will have created two distinct types of motion—one part of the object spinning like a wheel, while both parts of the object are flying around in a circle. We have achieved this complex set of motions by using two different points of control. A point or axis used to control the motion of an object is called a **control point**, also called an **animation variable** or simply an **avar**.

In an object like the dolphin of Figure 16-3 there may be dozens or hundreds of distinct control points that allow us to move the object's head, body, and tail in multiple ways. Similarly, if we are animating the image of a

human face, we might want control points for the smiling and frowning movements of the mouth; control points for each eye, to allow them to move in different directions; and control points for the head, to allow it to turn left and right as well as swivel up and down. It would not be unusual for an animated image of a human being to have as many as 500 separate control points, to allow it to move in many different ways, just as people are able to do in the real world.

16.2.5 *Rendering and Display*

We now have a polygon mesh composed of a set of plane figures, such as triangles, correctly positioned within the new frame after motion has taken place. The final step in the graphics pipeline of Figure 16.2 is rendering and displaying the final image. **Rendering** means taking an object stored as a mathematical model, such as the vertex list of Figure 16.4(b), and converting it into a fully formed, visually pleasing three-dimensional image.

Rendering is a complex set of operations that often consumes the vast majority of computer time required to produce an image. Some of the issues addressed during the rendering process are:

- *Lighting*. We specify the location and intensity of all light sources illuminating the image and determine the effect these light sources have on the final appearance.
- *Color shading*. We initially assign a single color or gray level to each vertex in the model and then interpolate those colors across the face of the polygon. We also determine if there are any modifications to the intensity or shade of that color due to the incidence of light falling on that plane.
- *Shadows*. We modify the color and brightness of each plane figure because of shadows cast on that plane by opaque objects.
- *Texture mapping*. In the first two stages of CGI, we assume that each plane is a homogeneous, detail-free surface. However, real surfaces like human skin or tree bark are far from homogeneous. Texture mapping allows us to add surface details (bumps, grain, indentations) to each of the plane figures.
- *Blur*. If an object is moving rapidly from one frame to the next, we may choose to blur the final image to represent that motion.

The operations just described (and many others not listed here) are carried out by CGI software running special-purpose rendering algorithms. Figure 16.9 shows a fully rendered color image generated from a polygon mesh representation of each object—glasses, pitcher, dice, ashtray, tiled walls, and table. This image clearly illustrates the many difficult issues that rendering software must deal with in producing a finished image—the color shading of the ashtray from bright green to almost black; the transparency of the glass objects, revealing objects located behind them; the opaqueness of the pitcher; shadows on the wall; reflection of light off the glass surfaces; and the complex texture on the bottom of the water glass.

Rendering a complex image like Figure 16.9 can be difficult and time-consuming, especially when there are numerous objects and many light

FIGURE 16.9

Example of a Fully Rendered Frame

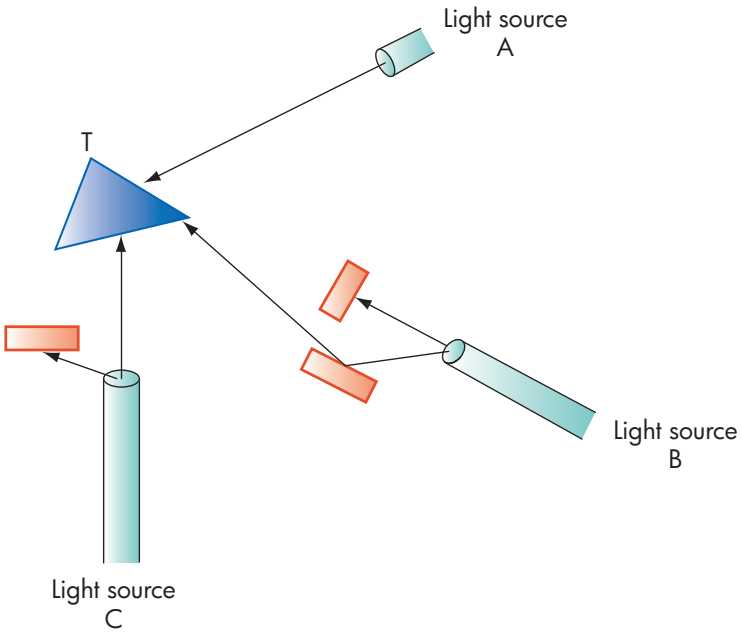
Courtesy Gilles Tran

sources. It would not be unusual for a computer, even a powerful one, to spend hours producing a single frame such as Figure 16.9.

There are many algorithms for carrying out the rendering operations just described, with names such as **ray tracing, rasterization**, and **radiosity**. However, most use a similar approach—determining the amount and direction of light falling on each plane surface in the model's vertex list. For example, in Figure 16.10 there are three light sources illuminating triangle T, where T is a single triangle on the object's surface. Light source A shines directly onto the surface of T. Light source B is blocked by an opaque object, so it does not contribute any direct light, although it does contribute some indirect lighting due to reflection off another surface. Light source C is partially, but not completely, blocked by an opaque object, so it contributes a fraction of its potential light. The contributions of each light source are summed to determine the total amount of light falling on the face of triangle T. This value, along with a knowledge of the object's orientation in space, allows us to determine the

FIGURE 16.10

Three Light Sources Illuminating Triangle T



proper intensity, color shading, and brightness of that face and render it in a visually appropriate manner.

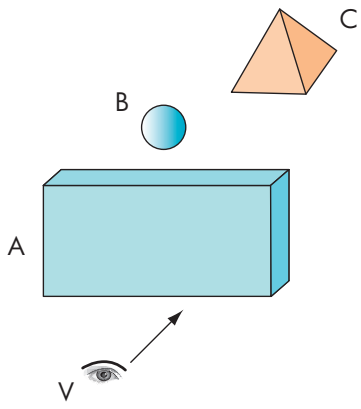
Tracing the individual rays of light falling on every face of a model can be an extremely slow operation, especially if there are numerous light sources and we are rendering a "busy" object, such as leaves on a tree or strands of hair on a human head. For these special surfaces CGI often uses algorithms designed to render just this type of object.

The end product of rendering is a fully colored and textured three-dimensional image ready for display. The final step in the process is changing that three-dimensional image into a two-dimensional one for display on a computer, game console, or movie screen. This is a relatively simple step based on the position of each object in the frame, the location of the viewer, and some simple geometry. For example, Figure 16.11(a) contains three three-dimensional objects labeled A, B, and C, and a viewer whose position is indicated by the letter V. Knowing the three-dimensional coordinates of A, B, C, and V, we can determine that, from the perspective of point V, sphere B is totally obscured while pyramid C is partially obscured. This information can be used to produce a two-dimensional screen representation of what can be seen by a viewer from location V. This is shown in Figure 16.11(b).

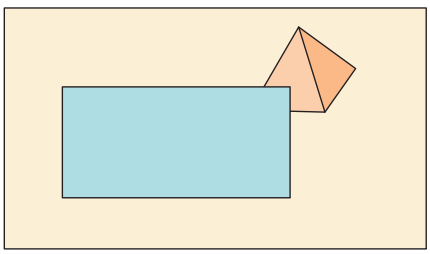
16.2.6 *The Future of CGI*

High-quality CGI is one of the most computationally demanding applications of computers, and only in the last 10–15 years have processors become sufficiently powerful and memory units grown sufficiently large to carry out the operations described in this section in a reasonable amount of time. However,

FIGURE 16.11
Converting an Image to a Two-Dimensional Representation



(a) Three-dimensional representation



(b) Two-dimensional representation

as we learned in Chapter 5, parallel and multicore computers are becoming more common, and this parallelism is allowing computers to overcome Moore's law and continue to gain in speed. In addition, computer scientists are discovering newer and better algorithms for such common CGI operations as modeling, animation, and rendering. The future of CGI is bright, and it is quite likely that the next 35 years will see improvements in the quality of computer-generated imagery that may be as (or more) dramatic than the change from the primitive Pong image in Figure 16.1 to the elegant still life of Figure 16.9.

16.3 Video Gaming

The computer science issues involved in producing video games are much the same as those addressed by CGI because game images displayed on a laptop, arcade system, or console must still be modeled, animated, and rendered as described in the last section. However, there is one huge difference between CGI and video gaming that makes an enormous difference in how we approach and implement these two applications.

A movie is not an interactive environment. There is no change to the plot or action on the screen based on what the user is thinking or doing. If you watch a movie 10 times, you see exactly the same images in exactly the same order 10 times. Therefore, movie animators can spend as much time as they want rendering each frame, even hours if necessary, because once each frame is completed its content never changes, and the order in which the frames are shown never changes. Simply put, a movie is a **static environment** that is created once and shown as often as desired.

On the other hand, a video game is a highly **interactive environment**. Using an input device such as a joystick, wireless controller, or keyboard arrow, a user dynamically controls the action and makes instant decisions about what happens next—Should I shoot that alien? Will I go through this trapdoor? The content of the next frame depends on what the user does at this instant. Therefore, we cannot render all frames in advance, since we don't know how the objects will move or behave. When the game is in progress, we must generate the frames fast enough so that action on the screen appears to happen at roughly the same rate as it would happen in the real world. For example, if I use my game controller to swing a virtual golf club, the screen image must immediately display the ball's flight based on the properties of the swing I just made. If the processor cannot work that quickly, the action will be sluggish, and the game will be far less enjoyable to play.

The branch of computer graphics that studies methods for creating images at a rate matching that of the real world is called **real-time graphics**, and video gaming is an excellent example of a real-time application. This means that instead of having minutes or hours to render a frame, we have, at thirty frames per second, only 1/30th of a second to get the user's input, determine what took place, generate a new frame representing the result of that action, and display the final image. That is a severe time constraint, and because of this limitation the operative principle in producing video game images is:

If necessary, sacrifice image quality for speed of display.

One of the most common techniques for increasing imaging speed, termed the **frame rate**, is to use a **GPU**, an acronym for **Graphics Processing Unit**. A GPU is an independent Von Neumann processor, much like those described in

Chapter 5 and diagrammed in Figure 5.18. A GPU executes instructions in parallel with the CPU, the main processor, and carries out all graphics operations described in this chapter—modeling, motion, rendering, and display. If there is no GPU, these operations must be handled by the CPU in addition to its many other responsibilities—running user programs, updating disks, handling input/output, and managing network connections.

With a GPU, all imaging responsibilities are offloaded from the CPU to the GPU, and the two processors run in tandem, an excellent example of the multi-core parallelism introduced in Section 5.4. Since a GPU does not have to do general-purpose computing, only image processing, its instruction set can be optimized to perform the specific operations needed by CGI. These might include rendering and drawing triangles like those in the dolphin model of Figure 16.3 or searching two-dimensional matrices like the vertex list of Figure 16.4(b).

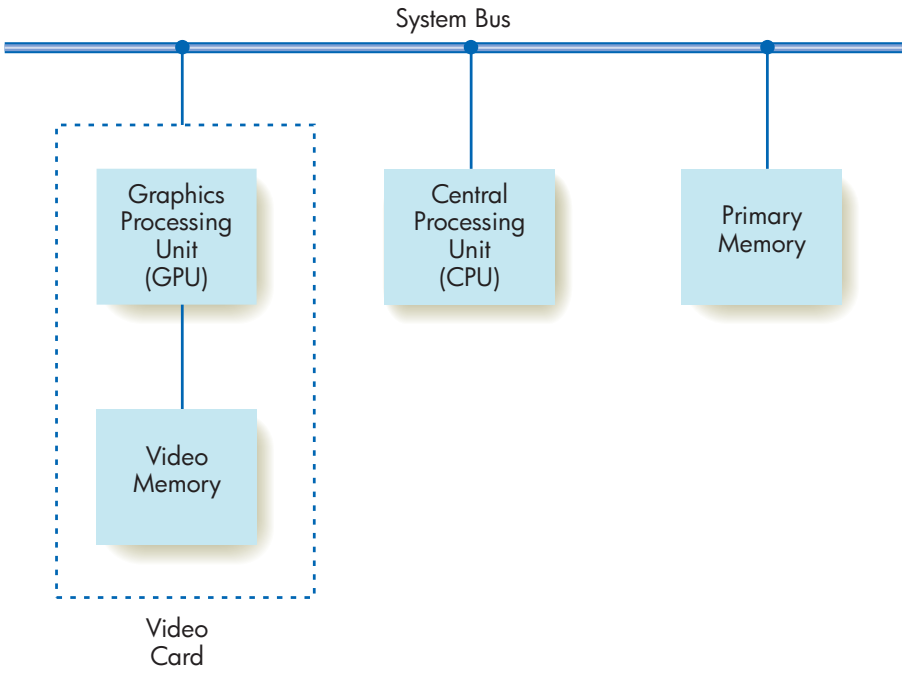
Typically, a GPU has its own dedicated random access memory where it stores its image data and which is separate from primary memory. The GPU along with this dedicated RAM, referred to as **video memory**, is located on a **video card** connected to the main CPU and memory either through a plug-in expansion slot or via the system bus. This architecture is diagrammed in Figure 16.12.

The configuration shown in Figure 16.12 allows the GPU to access image data (e.g., vertex list, color information, location of light sources) from video memory without having to compete with, and be slowed down by, the CPU as it tries to access the primary memory. Today, the great majority of computer systems and video game consoles contain a dedicated video card and GPU architecture similar to the one shown in Figure 16.12.

Another way to achieve speedup in real-time graphics is to avoid the use of algorithms that, although they produce high-quality images, simply take too much time. An excellent example is the ray-tracing algorithm introduced in Section 16.2.5 and diagrammed in Figure 16.10. Following millions (or billions) of light rays from their source to an object's surface and any subsequent reflections can produce truly lifelike images, such as the still life of Figure 16.9, but it can take minutes or hours to render a single frame. In a

FIGURE 16.12

Typical Architecture of a GPU and Video Memory



real-time environment we don't have hours or minutes, only 1/30th of a second, to complete this task.

We can gain considerable speedup by rendering an entire plane (i.e., a single triangle) using a uniform color, shade, and texture. As was mentioned in the previous section, many rendering algorithms assign a color and texture to each vertex and then use this information to interpolate shading and intensity changes across the face of the triangle. We can eliminate this step and instead assign a single color and texture to the entire face; subtle color differences or brightness changes within a single plane would not be allowed. This reduces our workload significantly, but it comes at the cost of a less lifelike image.

Another technique to speed up rendering and display is **culling**. Rather than rendering every plane in the wireframe model and then determining which planes are visible from the user's perspective, as diagrammed in Figure 16.11, we could turn those two operations around. First determine which planes can be seen from the user's point of view, based on location and opaqueness, and then render only those objects visible in the next frame, omitting all operations on hidden surfaces.

Finally, video gaming often makes use of a technique called **cut-ins**. These are fully modeled and fully rendered objects stored in a video library in video memory. These already prepared objects can be dropped into a frame as is, producing a significant speedup in frame creation. These cut-ins often include images of the main game characters as well as standard background objects—cars, castles, weapons—that appear in many of the frames.

The end result of these optimizations (and many others not mentioned here) is the ability to accept user input, determine what action should be taken in response to this input, and render and display a frame representing

The Good, the Bad, and the Ugly

In the mid-nineteenth century, a British literary genre called "penny dreadfuls" was extremely popular among English teenagers. These cheap magazines, costing a penny, contained lurid tales and drawings of horror, sex, and violence. Respectable Victorians detested these publications and fought to censor them, claiming they warped the minds of young, impressionable British lads. This was one of the earliest attempts at censoring a style of youth culture deemed inappropriate and degrading by adults of that era.

Similar attempts at censorship have been repeated numerous times against other popular pastimes such as comic books, rock and roll, R- and X-rated movies, and rap music. Most recently there has been a good deal of vocal opposition to modern video games.

There is no doubt that video games contain a good deal of controversial content—drug use, criminal behavior, sexual acts, violence, and strong language. In many cases, this extreme content forms the central theme of the game. For example, the primary goal of one popular game is to steal a car and run down pedestrians. A widely played video game includes the simulated shooting of police officers and the rape and murder of prostitutes. Another game involves violent gang activity between competing ethnic groups. These games often use a "first person shooter" format, in which the player views the activity through the eyes of the main character, actually pointing the gun and shooting the victims.

This is rough stuff and has led to public criticism of the video game industry from politicians, schools, parents, religious groups, and mental health organizations. To address these concerns, the industry has adopted a voluntary rating system, but it is strongly opposed to any additional forms of censorship. The game industry argues that adults should be free to play these games if they so desire, and that there are few scientific studies linking the playing of violent video games to real-world crime or changes in the personality or behavior of their players.

This controversy will no doubt continue to grow, as governments around the world have passed, or are considering, legislation to regulate or restrict the production and sale of violent video games.

the game state after that action has been completed. And all of this in only 1/30th of a second!

Today the quality of a typical video game image does not approach the level achieved in high-quality, feature-film CGI because of the time constraints placed on real-time graphics. However, as processors grow faster and as higher degrees of multi-core parallelism (4, 8, 16 GPUs per system) become both technically and financially feasible, the quality of real-time video game images will certainly improve and perhaps begin to approach the level of the computer-generated imagery found in today's best feature films.

16.4 Multiplayer Games and Virtual Communities

Most video games involve a small number of players, typically one to four. However, the last 10 years have seen the development of a new game genre called **massively multiplayer on-line games**, abbreviated **MMOG**. These games allow a large number of players, often thousands or tens of thousands, to interact, form groups, and compete against one another within a simulated virtual world.

The world in which the action takes place is created and managed by special computers called **game servers**. Depending on the game complexity and the number of players, there may be one or two game servers or many thousands. In an MMOG the virtual world in which the game is played is **persistent**. This means the **server software** that creates the world is always running and always available, and it always remembers the current state of every player. This is unlike games that can be turned off and on at will, but that lose state information when turned off, and must be restarted from the beginning.

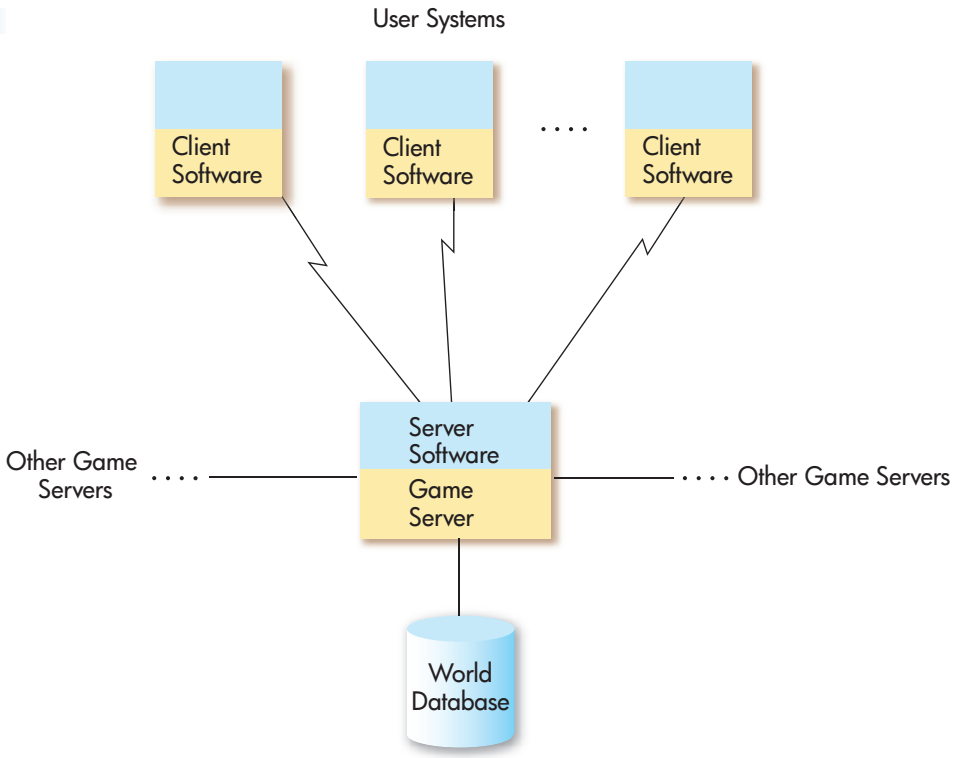
Users log on to the game server whenever they wish, using **client software** running on their home computer or laptop. This client software may be either proprietary code purchased from the gaming company or a freely available program such as a Web browser. Thus, the architecture of an MMOG, shown in Figure 16.13, is virtually identical to the client-server network model introduced and diagrammed in Figure 7.18.

The development of an MMOG incorporates a number of important computer-science-related research topics. For example, the three-dimensional images displayed on the user's computer employ all the real-time graphics algorithms discussed in the previous section, but with the added complexity and tighter time constraints caused by delays across the network. In addition to rendering the game images, designers of MMOG must also address and solve the following technical problems:

- *Registration management*. There may be tens of thousands of existing users at various points in the game, as well as thousands of new users joining every day. The responsibilities of the server software that manages this user base include ensuring that new users correctly join the community, saving the game state of existing users when they log off, and restoring that state when they log back in. This is similar to the "receptionist" responsibility of the operating system discussed in Section 6.4.1.
- *Client/server protocols*. In an MMOG there are tens of thousands of users simultaneously accessing hundreds or thousands of game servers across multiple communication channels. The game designers must implement

FIGURE 16.13

Architecture of an MMOG



the network protocols that support this vast communications array. We discussed the topic of computer networks and protocols in Chapter 7.

- *Security*. An MMOG must keep track of each user's activity to ensure that his or her actions do not incorrectly or inappropriately affect the actions of other players. Furthermore, the system must ensure that all users adhere to the rules of the game and do not attempt to carry out illegal operations. The topic of computer security was discussed in Chapter 8.
- *Database design*. The world database of Figure 16.13 can be a truly massive structure holding trillions of bytes of data. The game designers are responsible for implementing this database and making sure that it can be accessed quickly enough to provide real-time response to user actions. Databases were introduced and described in Section 14.3.

Because of these many technical complexities, the cost of developing a sophisticated MMOG can run to tens of millions of dollars and take hundreds or thousands of person-years to design and implement.

When we think of the word "game" we often assume an environment based on competition, scoring, winners, and losers. However, a recent development in MMOG design is the concept of a **noncompetitive MMOG**, sometimes called a **metauniverse**, or just a **metaverse**. This is a simulated virtual world, much like what we have just described, but where the goal is not to destroy your opponent or get the highest score. Instead, the purpose of entering this metaverse is simply to explore the virtual world, interact with other people in the world (often called "residents"), form communities of residents with similar interests, and create new economic entities that have (virtual) value. Players behave in this metaverse in many of the same ways they do in the real world—communicating, working, building, and moving around. There

is no winning or losing in a noncompetitive MMOG, just the enjoyment of experiencing a new environment and meeting new people—not unlike traveling (for real) to a foreign country.

The most widely used and well-known metaverse is Second Life, a virtual world created by Linden Labs in 2003. Many of the items in this virtual world (houses, cars, clothing) are user-generated objects constructed by individuals or groups using a CGI modeling tool that allows residents to customize their virtual environment. (This tool performs the operations listed in the graphics pipeline diagram of Figure 16.2.) According to many of its residents, it is the collaborative and creative activities, not competition, that make Second Life so popular.

Second Life uses the client-server model diagrammed in Figure 16.13. The client software that provides access to the virtual world is a free, downloadable program called Second Life Viewer. Currently there are several thousand server computers and over 100 trillion (10^{14}) bytes of data in the Second Life world database. However, even with this vast amount of computing power, the popularity and growth of Second Life is beginning to strain the computational resources of Linden Labs, making it difficult to keep up with the growth of their virtual world. (This problem is not unlike the problems encountered in rapidly growing real-world cities whose resources strain to keep up with an expanding population.)

Currently there are 15 million residents of Second Life, which would make it the 63rd-largest country in the world (if it were a country), a little smaller than the Netherlands but larger than Ecuador, Greece, Portugal, Hungary, and Sweden. At any instant in time, there are approximately 40,000 people logged on to Second Life, wandering this virtual world, chatting with members of virtual communities, and creating virtual economic wealth.

16.5 Conclusion

Not long ago, all input to a computer was textual. Communication with the operating system was via cryptic textual commands that were difficult to understand. The primary applications of the 1960s, 1970s, and 1980s were also textual—e-mail, word processing, databases, spreadsheets—and they often produced reams of incomprehensible textual output. The appearance of the first graphical user interfaces demonstrated the power of visualization, using icons, windows, and buttons to enhance understanding. By the early 1990s, graphics had moved into the scientific domain via charts, diagrams, and images that made it easier to interpret the output of scientific programs. (This was described in Chapter 13 and shown in Figures 13.11, 13.12, and 13.13.) Businesses began to use graphics in the form of computer-aided design (CAD) tools that gave architects and manufacturers the power to create and edit designs online. Soon visualization was being added to just about every popular application, such as the ability to place images in documents or enhance JPEG photographs and attach them to e-mail.

But it is in the last 10 years that visualization has found one of its most compelling and exciting uses—the ability to amuse us, entertain us, and enhance our pleasure. The use of CGI in feature movies, TV, advertisements, and online videos is growing to the point where nearly every environment and action can be created on a computer and displayed in a photorealistic fashion. The use of interactive real-time graphics takes us beyond passive viewing

The Computer Will See You Now

Computer imaging can be used to amaze, enthrall, and entertain. It can also be used to diagnose, treat, and heal—applications that most of us would agree are far more important.

Medical imaging is a rapidly growing area of computer and biological science research in which computers and graphics software are used to produce highly accurate two- and three-dimensional images of the human body without surgery or other invasive procedures. The instrumentation that generates the image data may be x-ray, MRI, PET scans, or ultrasound. However, in all cases, these raw data would be useless for diagnostic purposes if they could not be converted into high-quality, lifelike images that can be examined and analyzed by health professionals. To do this, the raw data from these instruments are input to a graphics pipeline similar to the one in Figure 16.3. This allows a computer to model, render, and display images in a photorealistic fashion. Today, medical imaging algorithms are helping physicians detect the early stages of breast cancer, perform delicate brain surgery, and track fetal development in the womb.

The algorithms that produce these medical visualizations are quite similar to, sometimes identical to, algorithms originally developed to generate images of alien invaders, prehistoric dinosaurs, and virtual worlds. This is an excellent demonstration of the importance of basic scientific research—you never know where that work may lead, or in which fields it may ultimately make fundamental contributions.

of virtual worlds to letting us play in them, function in them, and even live in them.

As graphics processors and visualization algorithms grow faster and more sophisticated, and as parallel computing environments become even more common in laptops, home computers, and game consoles, the quality of computer-generated images will continue to improve, and the feeling of actually being inside that virtual world will grow more real. Perhaps the simulated-reality "holodeck" technology from the 1980s TV series *Star Trek: The Next Generation*, usually thought of as a comic book fantasy, is no longer an unattainable goal.

16.6 Summary of Level 5

At the beginning of Level 5, "Applications," we said that we would be able to cover only a sampling of the important applications of computers. After looking at simulation and modeling, electronic commerce and databases, artificial intelligence, and computer graphics, we hope you will seek to learn more about application areas that interest you but that were not covered in detail here.

There is one more level to our story. With all the capabilities that exist today and that will be developed tomorrow, what is the larger picture of computer technology within society? What are the ethical, legal, and social consequences of these capabilities? What should we welcome? What should we monitor or regulate? Is there anything we should prohibit? (Some of these issues were raised earlier in the special interest box in this chapter entitled "The Good, the Bad, and the Ugly.") Are there any tools that can help us clarify thorny ethical decisions? Level 6 raises these questions in more detail, though, of course, it provides no definitive answers. Individuals, armed with adequate knowledge, must hammer out their own position on many of these complex social issues. This is one of the responsibilities that comes with our unprecedented opportunity to enjoy the benefits of computer technology.