Advanced Graphics Behind Medical Virtual Reality: Evolution of Algorithms, Hardware, and Software Interfaces

ZIV SOFERMAN, DAVID BLYTHE, MEMBER, IEEE, AND NIGEL W. JOHN

Invited Paper

Applications of virtual reality (VR) and augmented reality (AR) in medicine require real-time visualization and modeling of large three-dimensional data sets. Consequently, these applications require powerful computation, extensive high-bandwidth memory, and fast communication links.

In the past, the manufacturers of medical imaging equipment produced their own special-purpose proprietary hardware for image processing and solid graphics. Due to the developments in computer hardware in general and in graphics accelerators in particular, there is a trend toward replacing the proprietary hardware with off-the-shelf (OTS) equipment. Computer graphics itself has advanced in its quest for realism. Generic algorithms such as shading, texture mapping, and volume rendering have been developed to meet the resultant ever increasing requirements. Advances in both the OTS CPU and graphics hardware have enabled real-time implementations of these algorithms, thereby facilitating many of the medical VR/AR applications used today. The development of graphics libraries such as OpenGL has also been an important factor. These libraries provide an underlying portable software platform that optimizes the utilization of the available graphics hardware. OpenGL has become a standard graphics application programming interface, particularly for graphics-intensive applications, and more and more OTS systems provide hardware implementations of OpenGL commands.

This review paper follows the evolution of these technologies and examines their crucial role in enabling the appearance of the current VR/AR applications in medicine and provides a look at current trends and future possibilities.

Keywords— *Augmented reality, computer graphics, evolution, graphics accelerator, graphics algorithms, graphics hardware, hardware evolution, history, medical imaging, medicine, OpenGL, surgery, virtual reality, volume rendering.*

Manuscript received July 3, 1997; revised November 14, 1997.

Z. Soferman is with Silicon Graphics Biomedical, Ltd., Malha, Jerusalem 91487 Israel (e-mail: ziv@jerusalem.sgi.com).

D. Blythe is with Silicon Graphics, Inc., Mountain View, CA 94043 USA (e-mail: blythe@sgi.com).

N. W. John is with Silicon Graphics Biomedical, Ltd., Salford Quays, Manchester M5 2XW U.K. (e-mail: nigelj@manchester.sgi.com).

Publisher Item Identifier S 0018-9219(98)01906-9.

NOMENCLATURE

ASIC	Application-specific integrated circuit.
Data glove	Glove with electronic sensors that relay hand movements to the computer.
Depth cueing	Using intensity to represent the depth of an image; parts of an object that are farthest from the viewer have lower intensity.
Fill rate	The rate of filling a frame buffer with the pixel values, given in pixels/s.
Fragment	A grid square (pixel) along with its associated values of color, z (depth), and texture coordinates.
Frame buffer	The memory used to hold the pixel values that will be written to the screen.
Graphics hardware	(or graphics accelerator) Computational hardware that is capable of executing a reduced set of graphics operations, usually done much faster than the host processor.
Haptic	Pertaining to the sense of touch.
Perspective	A geometric transformation from a three-dimensional (3-D) world to the two-dimensional (2-D) screen where the center of projection is at a finite distance from the projection plane. This process mimics the human visual system.
Pipeline	A sequence of functional units (stages) performing a task in several steps, like an assembly line in a factory. Each functional unit takes inputs and produces outputs that are stored in its output buffer. The output buffer of one stage is the input buffer of the next stage. This ar-

	rangement allows all the stages to work in parallel, thus giving greater throughput than if each input had to pass through the whole pipeline before the next input could enter.
Raster	A method of image representation by the computer. The image is represented by a set of samples of its gray or color values on a discrete Cartesian grid. The image is represented as a 2- or 3-D array (raster pixels or a 3-D raster of voxels).
Rasterization	Converting the primitives (vertices, line segments, and polygons) to a 2-D raster image.

I. Introduction

We are witnessing a significant event in the revolution brought about by the information age—the encounter between medicine and virtual reality (VR). This event is characterized by a flurry of medical VR [1]–[3]. Computer graphics processing has been more significant in this area than raw CPU power. The goal of this paper is to examine how the development of this technology has enabled real-time visualization in medical VR/augmented reality (AR). It provides a background to the computer graphics techniques employed by medical VR/AR applications, together with details of their computational needs. It is these computer graphics techniques that have driven hardware designers, but the performance that the latter can achieve has always been dictated by currently available technologies (components, memory, etc.). The minimum processing requirements of medical VR have now been met, however, and this has given rise to an increase in medical VR activity.

The main types of applications in VR/AR in medicine are described in Section II with an emphasis being placed on their computational requirements. Augmented reality, virtual endoscopy, surgical simulations, and more are covered. Although there is much worldwide activity in this area, there are only a few commercial products available today, and most of the applications described are still being developed.

An overview of the main computer graphics methods and algorithms used today is given in Section III. This background is necessary in order to understand which functions have been implemented in each hardware generation, what graphics capabilities can be expected, and their relevance to VR and, in this case, to medical VR. The overview covers object representation in computer graphics, basic illumination models, shading methods, and two-dimensional (2-D) texture mapping. The aliasing problem (the appearance of "jaggies" or "staircase" artifacts) is also described. Antialiasing techniques combat this problem and are very important in computer graphics. While aliasing is tolerable in some applications, e.g., computer games, in medical VR it is unacceptable. Next, volume rendering, an important technique for medical visualization and VR, is described in detail.

It is important to distinguish between "computer graphics" and "image processing." In many cases, the differentiation is not a clear cut. Classically, computer graphics is concerned with the visual display of computer-generated models and objects, and the goal is to achieve better understanding and intuitive treatment of the model (e.g., computer-aided design and manufacturing) or of large data sets in general, as in the case of scientific visualization. Another goal became to achieve maximum photorealism, in which the computer-generated objects appear as "real" as the actual physical objects they are meant to represent. In image processing, the source of the image is usually a sensor such as a camera or a medical imaging device, such as computed tomography (CT) or magnetic resonance imaging (MRI), which acquires an image, digitizes it, and places it in computer memory for subsequent viewing and reviewing on a screen. In most cases, the goal of image processing is to perform image manipulations allowing better visibility or automatic recognition of the object or analysis of the image. Classical image processing is concerned with extracting more accurate details, aiming at reliability, while computer graphics is concerned with how objects will appear on the screen.

In Section IV, the evolution of graphics hardware is summarized. The provision of graphics acceleration requires a computational device dedicated to performing a set of specialized operations very rapidly. A factor of 100–1000 times faster than the host CPU for some of these operations is typical. Parallel and/or pipeline architectures are used to achieve that performance. The many advances in graphics hardware performance since the early 1980's can be shown to outstrip the corresponding growth in CPU performance. Volume rendering is used as an example in this paper to highlight this performance difference. Akeley [4] uses a generation nomenclature for describing graphics architectures according to the primary capabilities for which the architecture was designed. In this paper, we use the same ideas for grouping features and for describing the evolution of these capabilities.

Application software development has a significant cost associated with it. If portions of the software can be reused with new hardware, then substantial savings can be realized, so there has been a trend toward decoupling the application software from the underlying hardware implementation. Section V describes the emergence of the graphics application programming interface (API) and its role in allowing software components to outlive the hardware on which they were first developed. This in turn places a new emphasis on universality and longevity in the development of graphics standards such as OpenGL since the API must now have a long lifetime to allow a longer lifetime for the application software to be realized.

The above factors have contributed to a noticeable increase in medical VR/AR application development in the last five years. This paper ends with a discussion of some possible future directions suggested by the continuation of this trend.

II. Examples of Virtual Reality/Augmented Reality Applications in Medicine

This section lists the different categories of VR in medicine and provides a sampling of some of the current state-of-the-art applications. A detailed survey is beyond the scope of this paper. Here, the main categories of applications are identified and examined from the point of view of their computational needs. Computer graphics terms like polygons, Z-buffering, texture mapping, etc. are fully explained in Section III. For a short historical account of the use of computer graphics in medicine, see [5].

Creating a medical simulation that is extremely efficient and realistic in look and feel and that does not suffer from unacceptable latency is not a trivial challenge, especially when considering the demands of motion and object interaction (including solid object computation and collision detection). To reduce latency, the VR developer employs fast devices (e.g., position trackers that report position quickly and often) and fast image generators. Also critical is a system architecture that distributes the computation and makes use of multiple processors to parallelize the simulation processing, the handling of input/output, and rendering. Fast processing speed is essential in medical VR, which involves the support of multiple input/output devices, some of which require multiple (heterogeneous) platforms on which to run. (The use of multiple platforms permits the harnessing of the strengths of different platforms yet adds another level of complexity to software design.) Input/output technologies, software, and computer hardware are changing rapidly to overcome these challenges.

One generally agreed-upon benchmark for medical simulation is real-time output of at least 10–15 frames/s [6]. From a perception standpoint, "real time" implies perceived simultaneity, which is vital for interactions in virtual environments (e.g., when a medical student moves the scalpel, he must experience the appropriate visual, auditory, and haptic display without perceived delay or lag). From a system-performance standpoint, real-time image generation means that images are generated quickly enough to be displayed at a chosen frame rate, ideally 30–60 frames/s, and the delay between user input and the system response must be less than 100 ms and ideally less than 10 ms. The update of a haptic feedback device is even more demanding and is estimated, for many applications, to be more than 100 updates/s.

Even with the most advanced graphics computers of just a few years ago, VR developers have to make a tradeoff between photorealism and scene complexity to increase responsiveness by reducing latency. Integrating multiple I/O devices such as surgical instruments and synchronizing multisensory displays such as visual output, force feedback, and tactile feedback place additional demands on the system. The responsiveness required to generate and display changes in the environment or model in response to user input, especially position-tracked changes in head or hand position, has generally taken priority over image quality (detail, resolution, antialiasing, etc.) and complexity. Consequently, techniques to provide both responsiveness and image quality have been developed. Improvements in image quality can occur automatically and instantly as soon as the participant stops moving objects. Or, perhaps, the simulation can display organs that look extremely realistic but do not deform realistically or change position at real-time frame rates.

More recent advances in high-end graphics computers support sophisticated load management, data base paging, dynamic nonlinear image mapping, continuous terrain, Z-buffering, unlimited moving objects, character animation, live video input, pixel-fill load management, morphing, and integrated image processing. It is now possible to display 960 megapixels/s or a volume of $256 \times 256 \times 512$ at a frame rate of 30 Hz. For many VR applications, a tradeoff is no longer required. The high bandwidth and low latency of the latest architectures also make it possible to combine multiple graphics pipelines on a single rendered scene. Called "multipipe rendering," this architecture delivers increased update rates for complex geometry and allows the developer to combine the multiple pipeline texture memories to render huge volumes. We anticipate the results that surgical simulation developers will be able to achieve with this technology.

Another image display feature that is useful in medical simulation is stereoscopic display. It facilitates the understanding of spatial relationships between objects in the simulation (such as determining the correct depth of epidural needle placement and other motor tasks). This feature is so critical that it is now part of the display hardware for high-end graphics systems. As a standard feature, stereoscopic hardware can drive head-mount displays (HMD's) without the optional connection boxes previously required. Similarly, from the most advanced graphics hardware down to desktop workstations, on-screen stereoscopic image display is supported for use with LCD stereo shutter glasses, such as StereoGraphics CrystalEyes.

The remainder of this section highlights some of the work that is going on in the key areas of medical simulation.

A. Virtual Endoscopy

Endoscopy is used for the internal examination of various human body cavities without surgery and to look for macroscopic anatomical abnormalities located within the cavity or in its wall, e.g., tumors, ulcers, etc. It requires that a video camera or bundle of fiber optics be inserted into the body, e.g., the nasal cavity, thus allowing the physician to view or display on a television monitor whatever happens to be inside that cavity.

Virtual endoscopy is a VR application that is meant to teach, simulate, and provide an alternative to the unpleasant real endoscopy procedure. Virtual endoscopy can be exploratory or diagnostic without any data manipulation. In surgical endoscopy, the model is manipulated and deformed. Here, we relate only to the fly-through part.

A three-dimensional (3-D) input from a CT/MRI or even an ultrasound is used to simulate the endoscopic procedure. The data used can be either patient specific or nonspecific

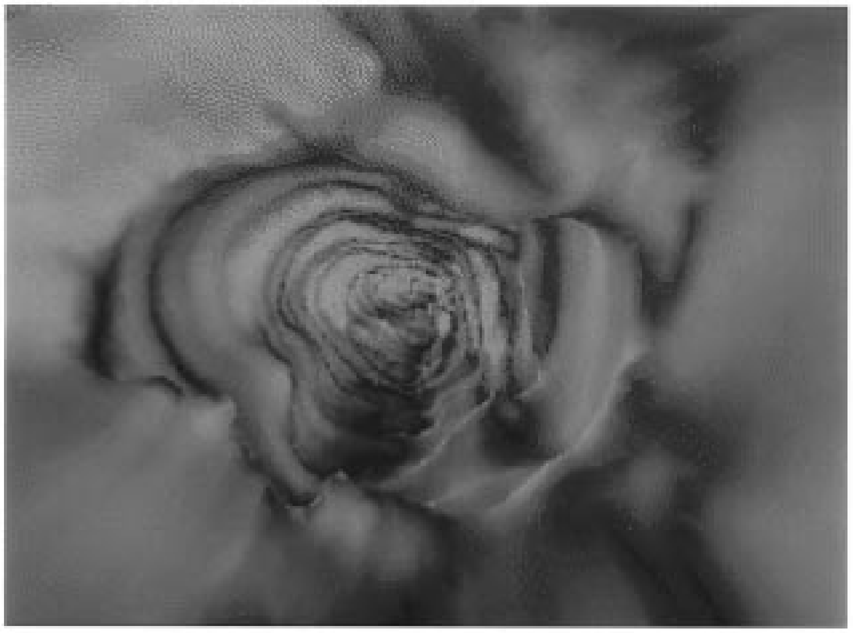


Fig. 1. This project employs several visualization techniques to achieve virtual imaging and exploration of the musosal surface of the human colon. Helical CT scans of the human abdomen are reconstructed into a 3-D volume and, subsequently, the human colon is visualized (shown in the figure) with various visualization and navigation techniques implemented in VolVis, which is a comprehensive volume visualization system. This noninvasive procedure is intended for mass screening for colon disorders. (Copyright and courtesy of Prof. A. Kaufman, State University of New York at Stony Brook.)

such as on visible human data (in which case much more work can be invested in preprocessing).

The typical procedure for preparing a virtual endoscopy comprises mainly:

1) obtaining 3-D data of the body region of interest using the appropriate imaging device;
2) segmenting the desired organ from the data so that the surface bounding the desired organ is obtained;
3) approximating that surface by polygons (polygonalize surface);
4) performing the "fly through" in the space bounded by the surface; the surface will often be texture mapped to add more realism (see Fig. 1).

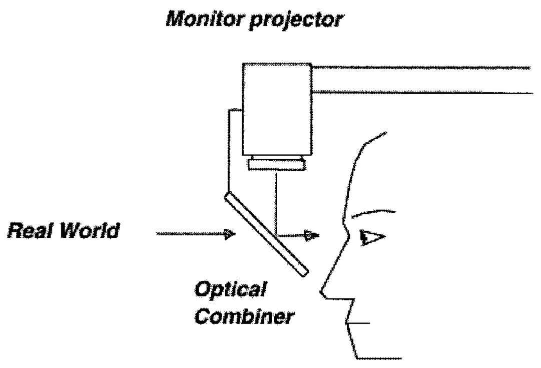


Fig. 2. A see-through HMD.

B. Augmented Reality

The essence of AR is the combination of a view of the real world with computer-generated (CG) objects. There are two main methods employed: the use of an optical combiner (Fig. 2) and video technology. The optical combiner uses a semitransparent mirror on a video display, thus allowing the real world to be seen through the mirror, while a CG image is simultaneously reflected onto it. The second approach consists of viewing the outside world via a video camera fixed on an HMD. The video image is combined digitally with the CG image and displayed on the HMD (optionally stereo). A conventional monitor could also be used instead of an HMD. For a description of the tradeoff between these methods, see [7]. For a survey on AR, see [8]. AR has some very interesting applications in medicine but they are still in the prototype stage.

The ultrasound group at the University of North Carolina, Chapel Hill, has built a real-time AR prototype in which the input is taken from both a video camera and ultrasound video. The video camera is displayed in the background, while the ultrasound video is texture mapped and displayed onto polygons appearing in a synthetic opening in the scanned patient. The polygon orientation corresponds to that of the ultrasound slice, and the displayed polygon is made to fade gradually after the probe has left the corresponding position. This may help the viewer to obtain a 3-D understanding of the scanned patient from a set of such polygons. More recent research has focused on using

AR to guide a needle biopsy of breast lesions in which the needle is superimposed on the breast image [9]. An important problem in AR is precise registration between the real world and the CG images, which is based on precise localization of the ultrasound slice relative to the patient. Maintaining the localization in real time is called "tracking." Precise tracking in real time is as essential to medical AR applications as the ability to move a CG object in real time.

Other AR projects aim at guiding a surgical knife, needle, etc. in the brain, for example, by looking at its "twin" virtual knife, which is at a corresponding location within a volumetric CT/MRI image of the brain. Skull-based surgery is an area of great interest. The Institute of Laryngology and Otology, U.K., is another group actively developing an AR system utilizing real-time volume rendering, registration with video data of the patient, and tracking of surgical instruments.

C. Surgical Simulation

A pioneering VR application for surgical simulation was built in 1991 by R. Satava, Jr., M.D., and VR pioneer/inventor J. Lanier of VPL Research. Wearing VPL's EyePhones head-mounted display and DataGlove, the viewer could "enter" and "fly through" a 3-D simulation of an abdomen containing simplified organs. The simulation was built with VPL's BodyElectric software.

Much progress has occurred since then, in terms of both maturation of the technologies and techniques and awareness of surgical simulation throughout the medical community. The advances in computer graphics over the past five years have made it possible to accomplish much more in less time. Today, surgical simulation results from a merging of the most sophisticated, high-resolution, high-contrast patient image acquisition, innovative computer graphics rendering algorithms, physically based modeling linked to image data, and real-time, high-performance, multiprocessor-based computer graphics systems.

The category of surgical simulation can be divided into its primary application areas of education, training, diagnosis, preoperative planning, rehearsal, and surgical instrument prototyping and analysis. Each category has differing computational demands but also a lot of similarity.

One example is the research being carried out at the Laboratory for Integrated Medical Interface Technology (LIMIT) at the Human Interface Technology (HIT) Laboratory of the University of Washington, Seattle, which also performs clinical testing of endoscopic sinus-surgery simulation. The HIT Lab is involved in a joint project to construct and evaluate a VR-based simulator for training physicians in endoscopic sinus surgery. The two-year project joins HIT Lab staff with physicians from nearby Madigan Army Medical Center and computer scientists at Lockheed Martin, Immersion Corporation, and the Ohio Supercomputer Center. This project represents the evolution of the endoscopic sinus-surgery (ESS) simulator developed by the Ohio Supercomputer Center and The Ohio State University. This simulator provides intuitive interaction with complex volume data and haptic (force) feedback sensation—see Fig. 3. In addition to the graphics computer, the software, and the surgical instrument (an endoscope), the system consists of a mock patient head, which contains the electromechanical mechanisms to provide the force feedback. The current visual interface produces frame rates up to 20 Hz (see [10] and [11]).

At the National Cancer Center Hospital, Tokyo, Japan, a group of researchers has been engaged in an ongoing project on a VR enhanced surgical conference system for the purpose of education, training, and oncological[1] surgical planning. The system, which incorporates two video cameras, simulates virtual organs and tumors, viewed through a head tracked display, and supports surgical procedures performed on the cancerous model (created from patient-specific CT/MRI data, employing 3-D texture mapping) while overlaying the interaction on live video images of the patient organ. The internal structure of the organ is visualized as the simulated resection takes place. The simulator achieves frame rates of 11 or 12 frames/s for monocular viewing and 6 or 7 frames/s for stereoscopic viewing of a scene, comprising 15 000 visible polygons.

D. Body-Movement Simulation

Another research-and-development field is body-movement simulation. Physically based modeling incorporates physical characteristics into models, permitting mathematical simulation of their behavior. An example is the human-trunk-movement simulator [12]. Other researchers have developed a 3-D virtual environment for modeling mechanical cardiopulmonary interactions, using 3-D deformable body dynamics, and associating physiological parameters with corresponding 3-D anatomy [13].

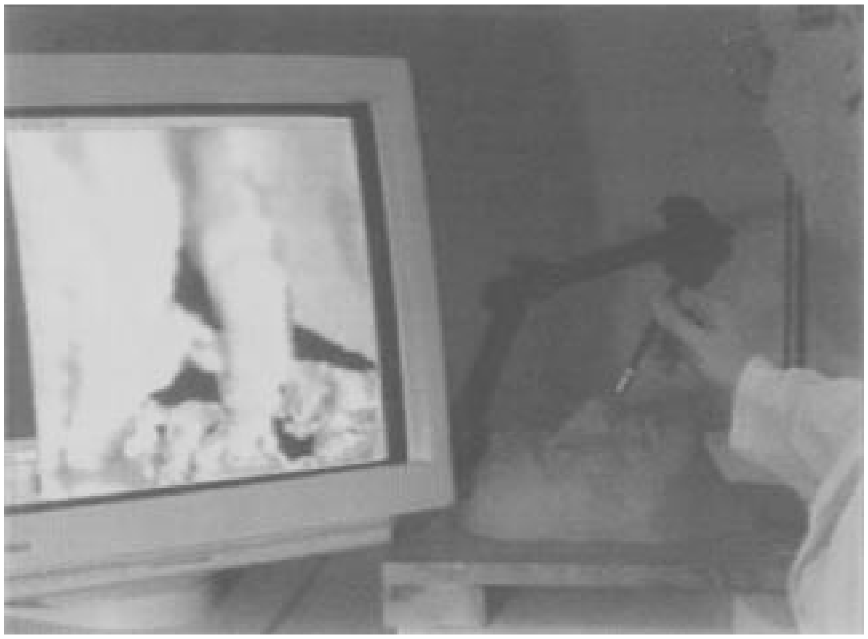
E. Finite Element Simulation of Heart Defibrillation

Recent research has been initiated to find a practical way of implanting electrodes within the body to defibrillate a person automatically upon the onset of cardiac fibrillation. Finite element methods (FEM's) are used to simulate the multitude of electrode configurations, their sizes, and the magnitudes of defibrillation shocks via computer simulation. A typical model is composed of more than 1.5 million tetrahedral elements with 250 000 degrees of freedom and requires approximately 4 billion floating-point operations to compute a detailed solution. The results obtained are displayed using hundreds of thousands of polygons—see Fig. 4. The system used is a 14-processor supercomputer, which is able to solve this problem in semiinteractive time [14]. Others use FEM's to simulate the interaction with models that take into consideration the physical properties of tissues.

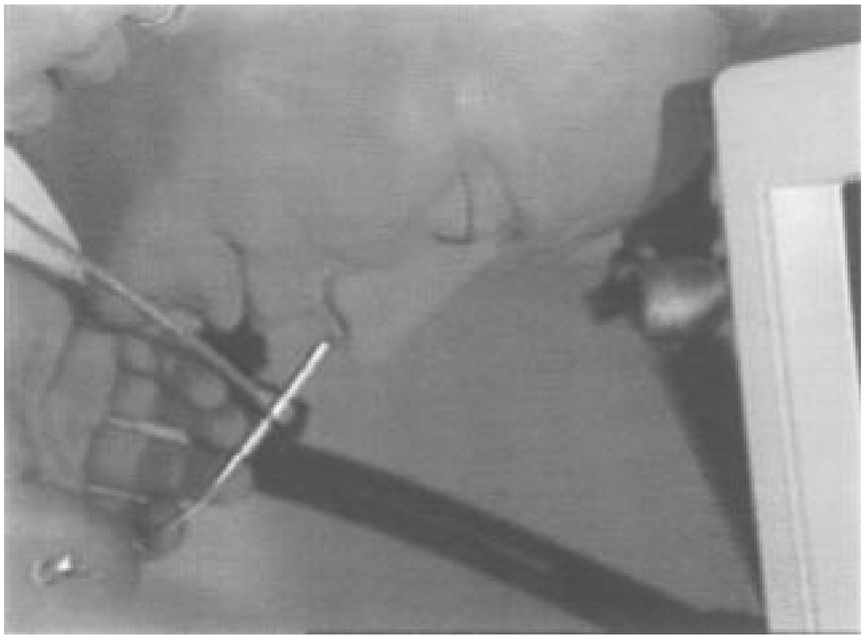
F. Notes on Computational Demands

How do the computational demands of the applications described above relate to the performance of available

[1] Oncology—the medical field dealing with malignant diseases.



(a)



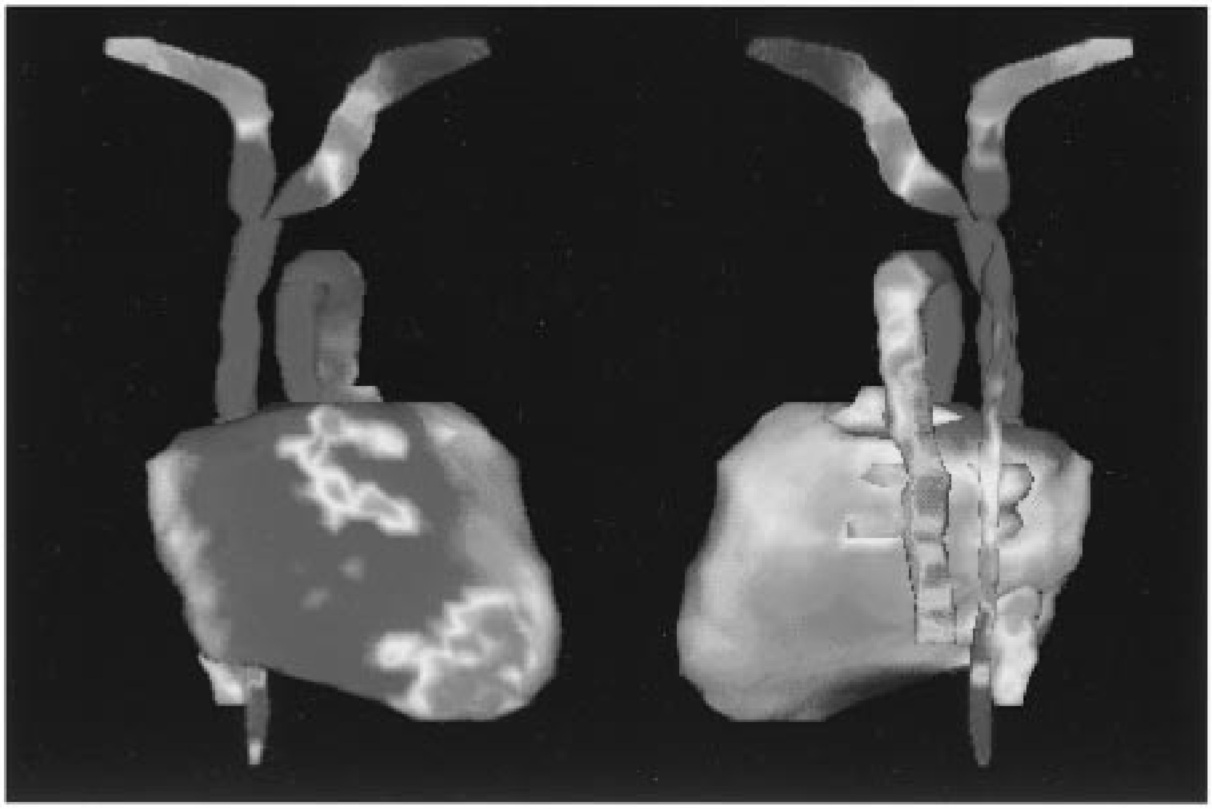
(b)

Fig. 3. (a) A multisensory ESS simulation system. (b) A haptic feedback device is housed inside a mannequin head. A motion-tracking device guides the display, which allows real-time volume-based rendering as well as surgery-like operations and deformations. (Courtesy of Prof. R. Yagel, The Ohio State University.)

graphics hardware? It is difficult to define exact computational demands—the complexity of the human body cannot be fully simulated by any of the computers available today. Additional computational power will always be welcome. The following general guidelines are suggested.

A generally agreed-upon requirement for medical simulation is for real-time output of at least 10–15 frames/s. The delay between user input and the system response must be less than 100 ms and ideally less then 10 ms.

The number of polygons needed for different applications varies considerably. For some applications, 10 000–15 000 visible polygons provide sufficient realism. Other projects, e.g., the ESS project at The Ohio State University, find that a model with less than 50 000 polygons is not realistic enough, even for a fly through. For data manipulation, and in smaller areas, much higher resolution is required. Physicians claim that in this case, the resolution needs to be at least 2000×2000 to make it useful for diagnostics. For example, a FEM simulation of heart defibrillation uses a model composed of more than 1.5 million tetrahedral elements, and the results are displayed using hundreds of thousands of polygons [14].



(a) (b)

Fig. 4. The two images show the current density magnitudes resulting from the electrode configuration in this figure. (a) is an anterior view and (b) is a posterior view of the heart and its major vessels. Red denotes regions of large current densities and blue denotes regions of low current densities. (Courtesy of Prof. C. Johnson, University of Utah.)

Data sets will be huge. The Visible Human digital image data set consists of 13.4 Gbyte.[2] After segmentation and polygonalization with two polygons/voxel, about 9 billion polygons are obtained [15]. When determining the number of polygons required for modeling, the effects of fluids such as inside the data model must also be considered. Such dynamics are computationally expensive, as is the display of the model.

The numbers of polygons and the frame rates used are actually dictated by the available computational power. The appearance of the third-generation graphics hardware allows about 15 000–20 000 polygons to be manipulated at 30 frames/s. This allows at least a minimal level of realism to be obtained and explains the surge in the development of VR applications that can be seen today.

III. Basic Computer Graphics Concepts and Methods

To succeed, a VR application must be "realistic." Simulating the exact interaction between each light photon and each minuscule surface feature in a realistic scene is far beyond the computational power of current computers. Thus, the story of computer graphics in general, and of VR in particular, concerns the achievement of maximum realism through contemporary hardware performance. The computational power can be used to pursue two different directions. The first is to approximate the physics of the interaction between light and objects, representing it by simplified yet well-approximated models. The second direction is to tap into the natural abilities of the human visual system to understand a three-dimensional world. The human visual system can understand depth or shape from shading, stereo vision, small object movements, and blurring. Computer graphics algorithms use this knowledge to achieve increased realism by simulating these effects. This explains why illumination models, shading methods, texture mapping, etc. have been invented and why they are so extensively used in computer graphics. The primary algorithms to achieve these goals were developed in the early 1970's, with considerable advancements made throughout the 1980's and 1990's. Only in the middle and latter part of this decade, however, have the functions been implemented in commercial hardware, thereby enabling the real-time, high-fidelity performance required for a successful VR application. This section provides a basis for understanding the technology concepts that have driven this evolution. Comprehensive treatments of all of these topics can be found elsewhere [16], [17].

A. Object Representation

Object surfaces (or models) in computer graphics are usually represented as polygons because polygons can be

[2] We use abbreviations such as Gbyte for giga (billion) bytes and Mbyte for mega (million) bytes. Likewise Gpixel, Mpixel, and Kpixel.

represented by their vertices. However, there are several primary methods of representing objects, or models, each with its own advantages and disadvantages. The criteria for selecting a method include:

- computational and memory complexity;
- desired image quality;
- ease of building and editing an object description;
- cost required to design and build hardware for manipulating objects in a given representation.

1) Wire-Frame Representation: This is the simplest method, in which only the polygon edges are displayed, not the facets. Characterized by relatively low computational complexity, it was the earliest method to be implemented in hardware for real-time performance. This method is suitable for overlaying the wire frame on continuous pixel images in AR applications [see Fig. 7(a)].

2) Polygonal Representation: Object surfaces are represented as a set of flat two-dimensional polygons that combine to form an approximation to the surface. Curved surfaces are decomposed into polygons (tessellated), which are small enough that the individual facets are not visible. The main advantage of a polygonal representation is simplicity. It is dominant in all surface representation in VR, such as in virtual endoscopy [3]. Each polygon is uniquely defined by its list of vertex coordinates, and geometric manipulations such as affine transformations need only be applied to the vertices rather than to each facet pixel. This is computationally efficient and widely used in graphics hardware devices. Much of the medical 3-D visualization carried out today is also polygon based. An algorithm such as Marching Cubes [18] can be used to extract a 3-D surface from volumetric CT data. A disadvantage with the surface extraction method is that much of the original data are discarded; only the surface at a particular threshold value is maintained.

3) Bicubic Parametric Patches: In this method, the surface is approximated by 2-D piece-wise bicubic parametric "patches," which are visually superior to polygons. The advantage is a more accurate representation of the model. An edge list must also be maintained, however, with the result that the complexity of any transformation is increased.

4) Constructive Solid Geometry: Here, the graphic object is expressed as a set of "primitive" bodies (spheres, cylinders, cubes, cones, and tori) combined by Boolean relationships such as union, intersection, subtraction, etc.

5) Volumetric Representation: The object is represented as a 3-D array of "voxels" (volume elements, the 3-D extension of the 2-D "pixels"). Unlike the previous methods that represent only the surface of an object, this method represents the entire volume. It is very useful to represent 3-D medical data such as CT, MRI, and 3-D ultrasound data in this manner since it corresponds closely to the form in which the data were acquired. Volume visualization is needed to define a function that will transform the volume onto the screen in a way that maintains the true representation of the desired details despite the suppression of one dimension. The technique used to visualize volumes is generally known as "volume rendering."

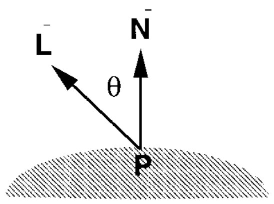


Fig. 5. Diffuse reflection.

6) Deformable Modeling: In many medical VR applications, it is desirable to represent skin, soft-tissue, and other "elastic" models and to be able to display the deformation of the object in real time. This is usually achieved through the use of models based on elasticity theories. The cost is computational and cannot be calculated solely on the graphics hardware because of the differential equations and other mathematical operations involved—which are performed on the CPU.

B. Object Shading

The manner in which the surface color at any point on an object is computed is referred to as the shading method. The shading of CG models enhances their visual fidelity and simplifies the underlying physics of computing and displaying realistic-looking models. An early (1967) shading method is described in [19], where each polygon vertex is assigned an intensity according to its distance from the viewing point and the intensities are interpolated across the polygonal facets. Today, the selected approach to shading depends upon the object representation, the illumination model, and the shading algorithm. Shading models are typically characterized by their accuracy, e.g., how apparent the underlying polygonal representation is, and by their computational efficiency, i.e., whether they can be implemented cost effectively. We will describe surface shading in terms of two aspects: illumination models and shading algorithms.

1) Illumination Models: Realistic shading requires the modeling of the interaction of a light source with the object being shaded. An illumination model can be thought of as an equation that is evaluated at each surface point. The equation can accept a number of input parameters: viewer position, position of surface point, surface characteristics, light-source position, light-source characteristics, etc. Modern lighting models reflect a compromise between physical foundations and less theoretical enhancements that provide good empirical results. Current lighting models typically include contributions from several terms:

a) Ambient: This corresponds to a constant, nondirectional source.

b) Diffuse: Dull surfaces such as cloth or chalk exhibit diffuse reflections in which the light is reflected with equal intensity in all directions. The surface brightness is proportional to the cosine of the angle (θ) between the surface normal and the light direction (see Fig. 5).

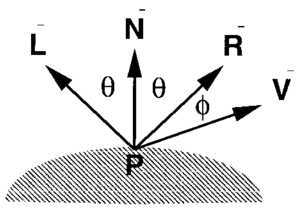


Fig. 6. Specular reflection.

c) Specular: Shiny surfaces exhibit highlights as a result of specular reflection. Specular highlights depend on the viewer's position as well as the light position. Phong [20] developed a model that uses the cosine of the angle (ϕ) between the viewer and the light reflected from the source. The degree to which the surface acts as a perfect reflector is modeled by raising the cosine term to a power referred to as the specular exponent (the n in $\cos^n \Phi$). Surfaces that are good reflectors have a large exponent, resulting in a very small focused highlight, and surfaces that are poor reflectors will have a small exponent and a gradual highlight (see Fig. 6).

There are a number of other effects that can be included in the lighting computation, such as model attenuation, light shape (spot light, point light), light colors, object surface (material) colors, etc.

2) Shading Algorithms: The illumination model determines the effect of one or more light sources on a surface point. The shading algorithm determines at which points the illumination model is evaluated and how those results are used to decide upon pixel colors. Popular shading techniques include the following methods (see [16] and [17]).

a) Flat shading: This is a very simple technique in which every polygonal facet has a constant intensity determined by evaluating the illumination model once for each facet using the facet normal. While computationally efficient, this technique has a serious disadvantage in that the edges between the facets are sharp, emphasizing the underlying polygonal structure [21].

b) Gouraud shading (due to H. Gouraud [22]): The aim of this approach is to ensure that a continuous, curved surface is rendered, one having a smooth appearance hiding the underlying polygonal representation. Gouraud shading achieves continuous shading by evaluating the illumination equation at each polygon vertex and then linearly interpolating those values to produce individual pixel colors. If vertex normals are not available, they can be computed by averaging the normals of adjacent facets. Gouraud shading is more expensive computationally, but the result is a smooth appearance. However, since illumination is only computed at the vertices, sharp specular highlights within facet interiors will not be represented.

c) Phong shading (due to B. Phong [20]): This method preserves specular highlights by evaluating the illumination model at each pixel. A surface normal is computed at each pixel by linearly interpolating the vertex normal at each pixel and normalizing the resulting vector. This shading algorithm is intense computationally since a complex illumination model is evaluated at each pixel. It is possible to reduce the computational burden slightly by using hybrid schemes that compute Gouraud shading for diffuse illumination and Phong shading for specular illumination.

C. Z-Buffer Algorithm

Developed by Catmull [23], the Z-buffer algorithm is a method for performing visible surface resolution so that only the visible surfaces of an object are rendered, i.e., those not occluded by another object. The Z-buffer is as ubiquitous in computer graphics as shading algorithms and interpolators.

Computer graphics algorithms typically represent the spatial location of an object according to its Cartesian position (x, y, z) in relation to the viewer, where x represents the horizontal position, y the vertical, and z the distance from the viewer. The Z-buffer algorithm operates in image (screen) space and associates a z (depth) value with each (x, y) pixel coordinate. The Z-buffer holds the smallest of all z values at the same (x, y). This value corresponds to the pixel at the location (x, y) closest to the viewer. As each new pixel is computed, its z value is compared with the z value currently saved at the corresponding (x, y) location. If the z value is less than the saved value (nearer the viewer), the new intensity and z values replace the saved values; if not, they are discarded. Once all of the pixels have been processed, only those nearest the viewer will be left. The advantage of this method is simplicity, but it does require additional storage for a z value for each screen (x, y) location.

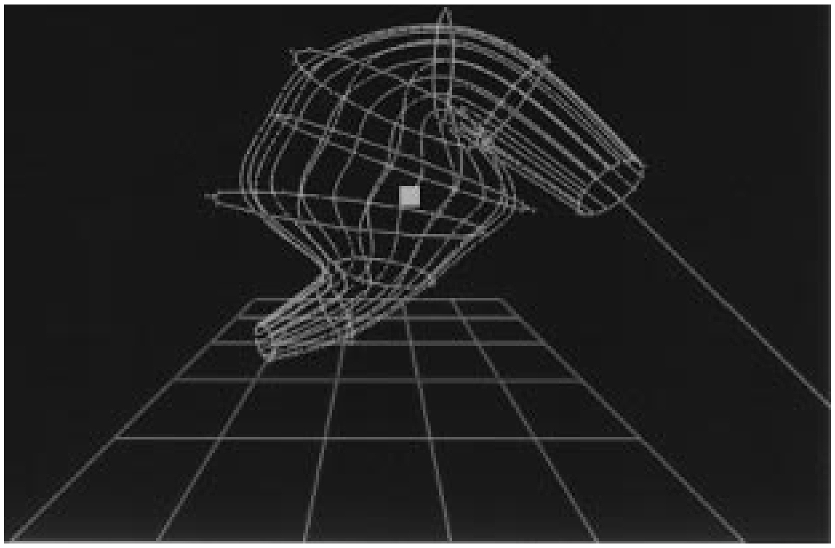
D. 2-D Texture Mapping

Also due to Catmull [24], texture mapping was the most significant step toward realism after Gouraud and Phong shading. Catmull realized that most objects in real life have very rich and detailed surfaces. He came up with the idea that if people apply texture to objects, like wallpaper on a wall, the same could be done to computer-generated 3-D objects. The original motivation for texture mapping was to overcome the effect of "shiny plastic" resulting from the much simpler Phong shading and also to allow "projection" of different images on the same surface rather than just color.

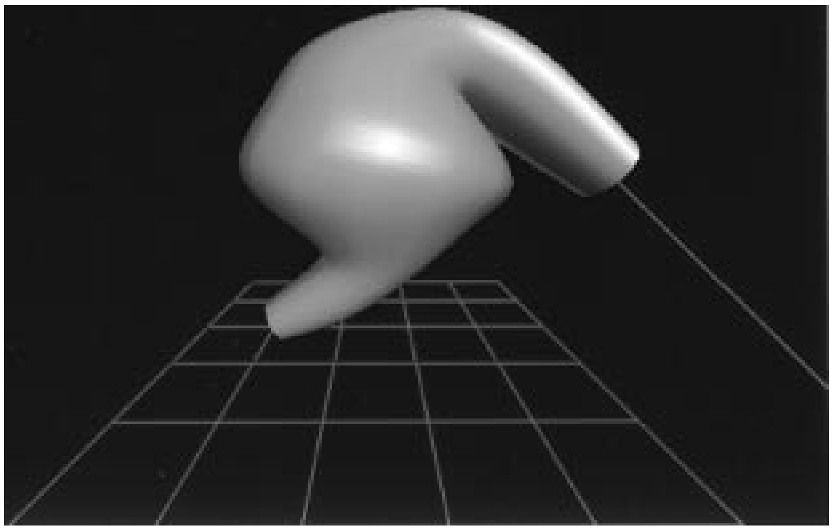
The 2-D texture space contains the "wallpaper" texture as pixelized, raster information, i.e., as numbers on a discrete Cartesian grid. The texture is transformed according to the geometric structure of the surface, imitating "gluing" wallpaper onto it. Then a second (usually perspective) transformation maps the texture from the surface to the screen.

Texture mapping is widely used in medical applications of computer graphics, particularly in VR/AR applications such as the display of 2-D ultrasound acquisition on a plane representing the orientation of the plane in the 3-D referential womb.

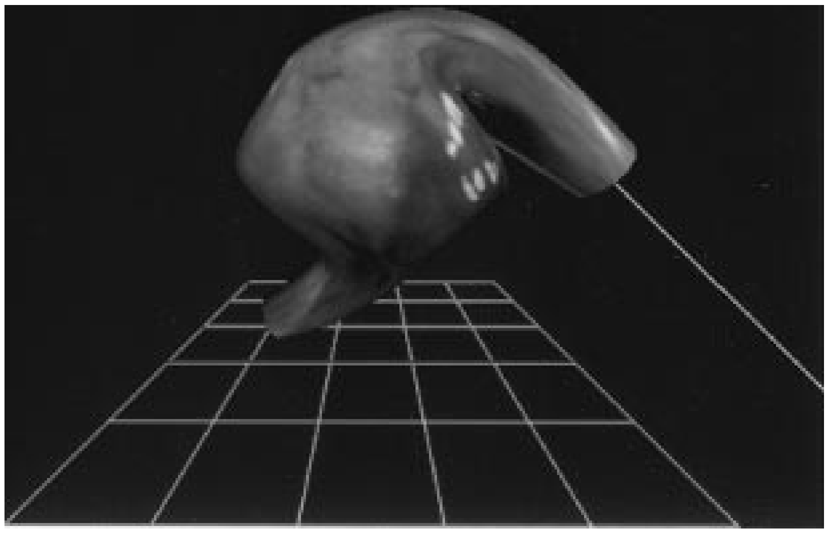
The three images in Fig. 7 show a generic tube displayed according to three representation methods.



(a)



(b)



(c)

Fig. 7. A generic tube. (a) Wire frame (the square indicates the center of mass). (b) Phong shading. (c) Texture mapping. (Courtesy of HT Medical.)

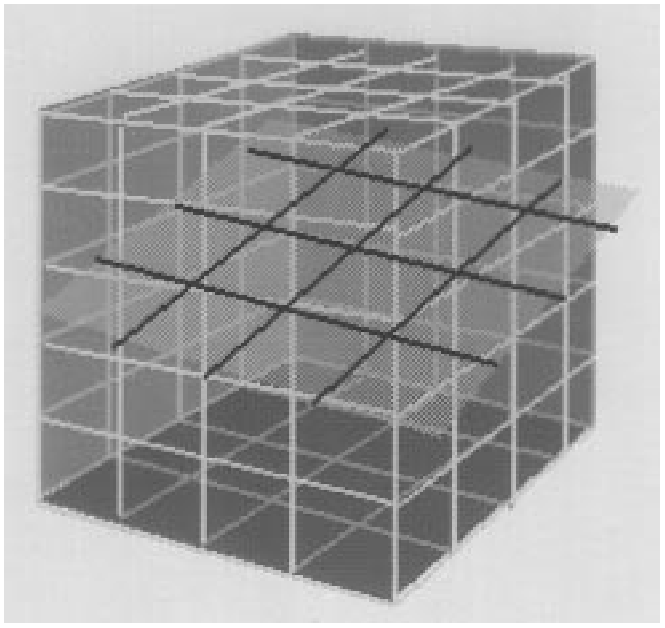


Fig. 8. Three-dimensional texture mapping. An oblique plane cuts through a 3-D grid. The value of each pixel lying on the plane is interpolated from the texture elements nearest it.

E. 3-D Texture Mapping

A more complicated version of texture mapping was described independently by Peachey [25] and Perlin [26] in 1985. In 3-D texture mapping both the object and the texture space are three dimensional. A surface is produced as a "cut" in the volumetric texture space. Regard the surface as being "embedded" in the texture volume. Then the resulting texture on the surface appears as if sculpted from the solid material. The surface pixel values are obtained through 3-D trilinear interpolation from the nearby voxels (see Fig. 8). When it became supported in hardware, the full 3-D texture data were stored in memory, enabling 3-D texture mapping with arbitrary data. Before that, the data were synthetic and calculated near the surface only (procedurally generated).

In medical applications, 3-D texture mapping plays a central role in many medical VR systems in which human body texture appears on virtual surfaces, from virtual endoscopic fly-through to clipping planes in CT/MRI data to clipping planes in Visible Human-based applications.

F. Aliasing and Antialiasing

Regardless of the shading technique used, a solution to the ever present aliasing problem must be included. This issue is very important in computer graphics, particularly when high-quality visualizations are required. For example, if a straight edge in a binary image is oriented diagonally, it will display "staircasing" or "jaggies." Another example is objects smaller than one pixel, which may appear or disappear depending upon whether they are "on" or "out" of a sampling point. Long objects that are thinner than a pixel may break up or disappear completely or periodically if they go in between the pixels, not intersecting with some or all of them on the way. The effect is strongly dependent upon the orientation of the object relative to the screen coordinate axes. Aliasing is also a problem in animation, where an object may appear and disappear (scintillate) or Moiré patterns may be visible on a texture-mapped surface. In general, aliasing happens when an object, continuous or discrete, appears to be visually distorted due to its discrete representation. Or it can be said to happen when high-resolution information is misrepresented by low-resolution means (see Fig. 9). See [27] for a comprehensive treatment.

There are three major methods for reducing the aliasing artifacts: supersampling, prefiltering, and stochastic sampling.

Supersampling, also known as postfiltering or filtering after sampling, is the most popular method. Sampling is performed at higher resolution, such as a factor of eight more in each direction followed by low pass filtering (i.e., smear, blur, average), then resampling at the screen resolution. The complexity is independent of image richness. It is implemented in modern graphics hardware.

Prefiltering, or filtering before sampling, is due to Catmull [28]. It is an area sampling method. The contribution to a pixel due to any portion of the continuous image is proportional to the area of intersection between the square pixel and that portion. Prefiltering is equivalent to sampling after convolving the continuous image with a square box, pixel-size filter. Each of the subpixels has an area Si and intensity Ii, the area of the entire pixel being one. The total intensity I is the weighted average $I = I_1 \times S_1 + I_2 \times S_2 + I_3 \times S_3 \cdots$ of the subpixel intensities. It is computationally intense and was later improved [29], [30] by using a bit mask to approximate subpixelization, i.e., the intensities of the visible subpixels are sampled in higher resolution. The sampled intensities are averaged within each pixel to give its value. Carpenter [29] used the method for Z-buffer, calling it an A-buffer (antialiased, area-averaged, accumulator buffer).

1) Stochastic Filtering: In signal-processing terminology, aliasing can be said to result from sampling a signal with equally spaced samples below the Nyquist rate. This is true even if the signal is supersampled and filtered to assign a value to the pixels. In stochastic filtering, the samples are not equally spaced, trading aliasing for noise, this being found to be a more bearable artifact. The method is implemented by performing sampling at a sampling grid whose coordinates are perturbed, followed by applying a reconstruction filter to estimate values at the unperturbed sampling grid points. The method was used by Cook (see [31] and [32]).

G. Antialiasing and Texture Mapping

Texture mapping in general is only successful if used in conjunction with a good antialiasing method. Otherwise there is a strong tendency for visible artifacts to emerge from point sampling of periodic or otherwise coherent textures. Graphics hardware performing high-quality texture mapping will need to take this into account. The hardware implementation of antialiasing in texture mapping is responsible for much of the computational demand.

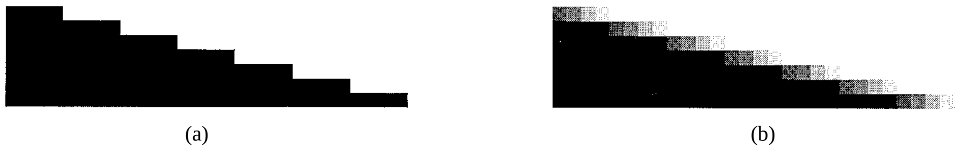


Fig. 9. Antialiasing. (a) Aliasing effect of "stairs" or "jaggies." (b) Antialiasing correction by postfiltering.

A good example is the case where many texture pixels are mapped onto one screen pixel, e.g., in a chessboard reduced so that a 2×2 region with two white and two black texture pixels (texels) will be mapped into each screen pixel. The correct result is for each screen pixel to appear gray—the intensities of the two black and the two white pixels averaged together. However, subsampling will cause each screen pixel to be incorrectly regarded as either black or white. This is a severe aliasing distortion of the visual representation of the chessboard.

The process of combining several texels to one screen pixel is generally known as filtering. The square pixel is inversely mapped onto the texel coordinate space, deriving the weights accordingly. Since the shape of the preimage changes from pixel to pixel, the filter is spatially variant. There are several ways of approximating the filter. Catmull [24] averaged with equal weight all texels corresponding to a screen pixel. Since then, Blinn and Newell [33] have used a similar approach with a pyramid weighting function, taking into account the inverse map of the center of the screen pixel in addition to its four corners. Other space-variant filtering methods are described in [34] and [35]. A method claimed to be efficient and accurate is the elliptical weighted average (EWA) filter proposed by Greene and Heckbert [36]. The idea is that the pixel in the screen space is approximated by a circle. Its inverse mapping is always an ellipse in the texel space. The ellipse can be described by its size, eccentricity, and orientation, thus enabling fast calculation.

Another approach is prefiltering, an important example for which there is the method called mip-mapping [37], which is essentially a multiresolution method (mip—*multum in parvo*—many things in a small place). The image is held in multiresolution, each time coarsened (scaled down) by a factor of two in each direction, from which any in-between scale can be achieved through interpolation. This avoids the need to average many pixels from the beginning in case of scaling/zooming of the image and thus saves a lot of computation. Mip-mapping is hardware accelerated in high-end and midrange graphics hardware.

H. Transparency and Image Compositing

Another important computer graphics problem involves rendering objects that are semitransparent. Such objects allow some light to pass through them, including light reflected from objects that are behind them. One way to simulate transparency is to use stipple patterns or masks in which some of the pixels of a semitransparent polygon are left unmodified so that they retain the color of the polygon behind them. The more transparent a polygon is, the more pixels there will be that are left unmodified. This technique relies on the human visual system to integrate the pixels across the polygon to achieve the transparency effect.

Another technique for rendering semitransparent surfaces is the use of a *blending* operation in which a new pixel color is determined by linearly interpolating the color of the semitransparent pixel and that of the corresponding pixel from the object behind the semitransparent one

$$C_{\text{new}} = \alpha^* C_{\text{transparent}} + (1 - \alpha)^* C_{\text{opaque}}.$$

The weight alpha corresponds to the transparency of the surface. To achieve the correct results, the polygons must be drawn in back-to-front order relative to the viewer. This operation can be supported in graphics hardware accelerators by maintaining an alpha value as part of the surface color and using it in interpolation operations so as to combine new pixel values with old pixel values.

This blending capability can be further generalized as performing compositing operations to combine images [38] and performing more general filtering and integration computations, such as those required for volume visualization.

I. Volume Rendering

Volume visualization is very important to VR/AR in medicine and will be described here in some detail.

Since the human body is a three-dimensional volume and sampled volumetric images can be acquired through CT, MRI, positron emission tomography, and ultrasound, among other imaging technologies, the issue of computer-generated three-dimensional volumes representing the human body is integral to the application of computer graphics in medicine.

The main problem in volume visualization is how to display this sampled volumetric information on a 2-D screen or how to use the 2-D screen to create a volumetric understanding of an object, tapping into the capabilities of the human visual perception system to "see" and understand what something is by viewing the way it is shaped and shaded and how it moves or behaves.

Early algorithms of volume visualization utilize the "additive projection," which computes an image by averaging the voxel intensities along parallel rays from the rotated volume to the image plane [39], [40]. This simulates an X-ray image and does not provide information about depth relationships. An additional method is the "source-attenuation reprojection," also referred to as "opacity," allowing object obscuration [41], [42]. Another important method is maximum intensity projection, which displays the maximum value found along a virtual ray from the screen to the volume. This method may distort the correct perspective because the maximal value overrides the others on the ray path. The method is widely used for magnetic resonance angiography visualization, where the voxel value represents the amount of flow in the vessel. Volume rendering in its present form (in particular for medical applications) was pioneered by Drebin *et al.* [43] and Levoy [44], although some of the ideas were mentioned earlier [45].

These algorithms were executed on general-purpose computers to produce a static (nonreal-time) image. The views provided the clinician with greater insights into the data than were possible through the traditional method of viewing a series of two-dimensional slices presented in a histological fashion. More important, they allowed those with less training to understand nearly as much as an experienced radiologist.

1) Method (see [17] and [46]): Suppose that in the volume to be rendered, each voxel is indexed as $X = (x, y, z)$ and has a color $C(X)$ and opacity $a(X)$; $a = 0$ implies a transparent voxel and $a = 1$ an opaque voxel. Each screen pixel is associated with one virtual ray (or multiple rays for antialiasing), which ends at that pixel after passing through the volume. For a screen pixel whose coordinates are (i, j), we assign a ray vector that is $R = (i, j, k)$, defining $r = (i, j)$ and $k = 1, 2, \cdots, K$ to be the distance along the ray in a front-to-back order. The volumetric data are resampled along the sequence of equally spaced points $R(k)$, $k = 1, 2, \cdots, K$ defined along the ray, using trilinear interpolation to calculate the color and opacity values at those points. The color and opacity values are accumulated while progressing along the ray from front to back. For each ray, we use the standard transparency formula to update the accumulated color and opacity for each k in the following manner:

$$\begin{aligned} C^*_{\text{out}}(r, R) &= C^*_{\text{in}}(r, R) + C^*(R)[1 - \alpha_{\text{in}}(r, R)] \\ \alpha^*_{\text{out}}(r, R) &= \alpha^*_{\text{in}}(r, R) + \alpha^*(R)[1 - \alpha_{\text{in}}(r, R)] \end{aligned}$$

where

$$\begin{aligned} C^*_{\text{in}}(r, R) &= C_{\text{in}}(r, R)\alpha_{\text{in}}(r, R) \\ C^*_{\text{out}}(r, R) &= C_{\text{out}}(r, R)\alpha_{\text{out}}(r, R) \\ C^*(r, R) &= C(r, R)\alpha(r, R). \end{aligned}$$

Note that the colors are multiplied by the opacities, following the intuition that a transparent voxel contributes zero to the color. The final color for each ray is obtained at the end of the ray by

$$C(r) = \frac{C^*_{\text{out}}(r, R)}{\alpha_{\text{out}}(r, R)}, \qquad \text{for } R = (i, j, K).$$

Other, more complicated recipes are possible, such as adding a surface property to each voxel, using it to simulate reflection and shading terms. Depth cueing can be simulated by multiplying the colors by an attenuation factor before blending.

As for practical implementation on graphics hardware, instead of performing the above calculation for each ray sequentially, resampling and value accumulation are calculated along the rays in parallel for all the rays simultaneously. The set of all first points of all rays together forms a plane parallel to the screen, as does the set of all second points, etc. In practice, this is done as follows: define a 3-D space whose XY plane is the screen. A 3-D perspective transformation between the volume to be rendered and the said space is defined. The screen is mapped into the volume using 3-D texture mapping, extracting cross-sectional slices parallel to the mapped screen. Each new slice is blended with the accumulated result of the previous slices. The last result is displayed on the screen.

In earlier, nonreal-time systems, movement around the volume was simulated by creating multiple views from the original volumetric data set and then scrolling through this series of precomputed images. In the late 1980's and early 1990's, hardware acceleration support for volume rendering was incorporated into some research and commercial graphics accelerators. Notably, Pixel-Planes 5 [47] was capable of rendering 128 $\times$ 128 $\times$ 128 volumes at approximately 2 frames/s. In 1992, hardware-accelerated 3-D texture mapping became commercially available [4], which allowed interactive volume rendering for volumes up to 128 $\times$ 128 $\times$ 128 at rates of 30 frames/s or more. As a result, view position, direction, and scale could all be varied arbitrarily to obtain more detailed diagnostic information. Using this technology, a physician can manipulate the volume to see, for example, over and under a patient's rib cage in order to inspect any structure or anomaly behind it.

With real-time hardware support, operations could be as simple as slicing the volume with an arbitrary cutting plane or performing volume rendering of the entire data set. When combined with color look-up tables, the density information can be classified (or segmented) into a color-coded tissue type (see Fig. 10, for example). By applying cutting planes, a selected portion of the volume may be viewed with an arbitrary cross-section capping the partial volume. By categorizing tissue according to density, it is possible to make various tissue types transparent. The internal surfaces between tissue types can be enhanced by evaluating lighting equations at each voxel, using gradients to compute approximations to the surface normals. Specular and diffuse lighting effects can be incorporated into 3-D texture mapping algorithms using multipass rendering techniques [48]. The result is somewhat like a "visible human" model residing in the computer.

As shown in 1994 by Cabral *et al.* [49], 3-D volumetric viewing comprises the dual operation of 3-D reconstruction. This allows a single piece of hardware, if properly constructed, to perform both the reconstruction function and a volumetric viewing function. This may prove to be an important cost savings in coming generations of patient image scanners. As demonstrated by the duality, there is an equivalence between the volume stored as a 3-D array of density values in Cartesian space and that stored as a

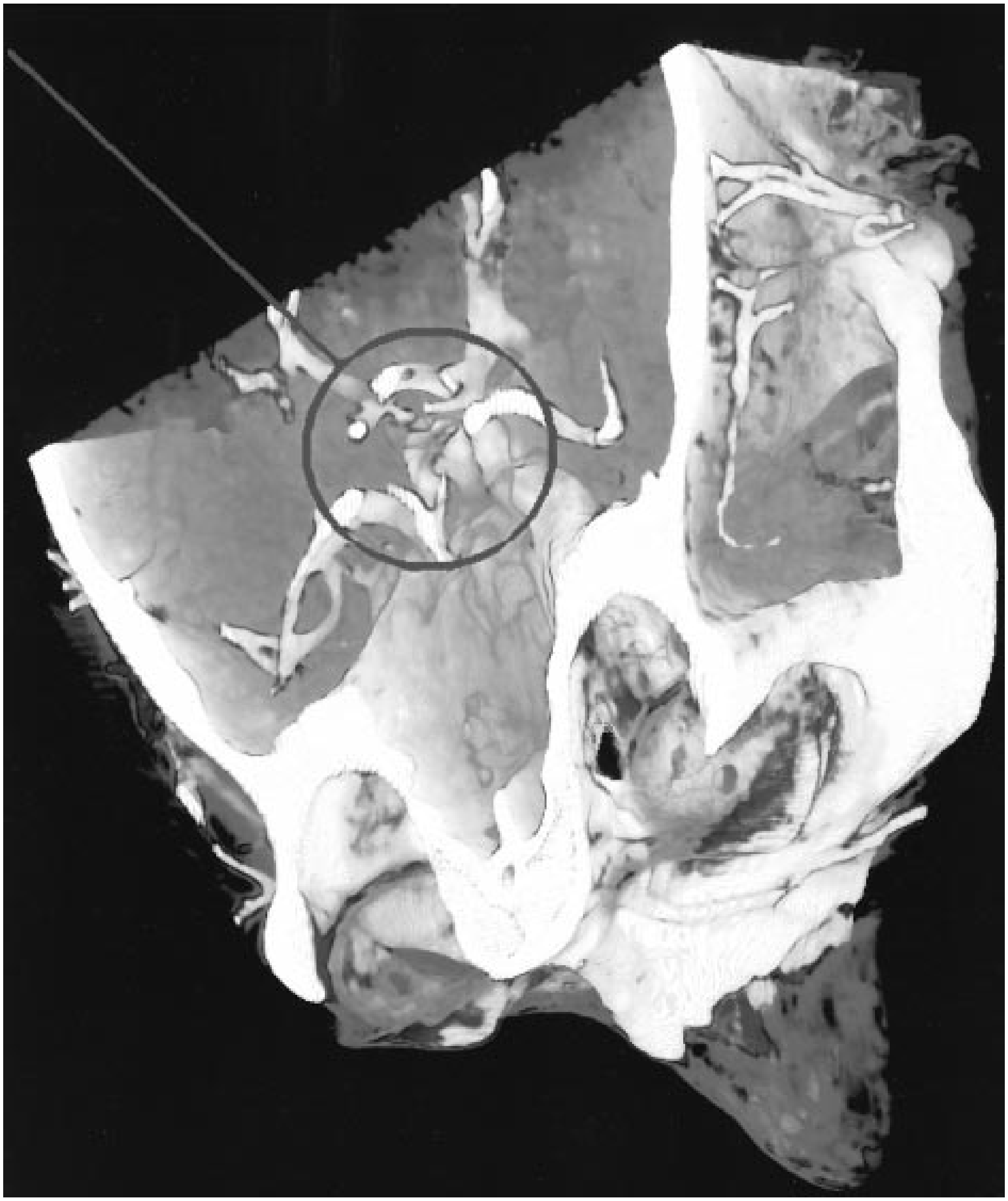


Fig. 10. Volume rendering of a three-dimensional reconstruction of the frontal region of the skull with CT angiography. The vascular map of the Circle of Willis is well defined. (Courtesy of Prof. E. K. Fishman, Department of Radiology, The Johns Hopkins Hospital.)

3-D non-Cartesian set of line integrals, as obtained by the diagnostic imaging equipment.

IV. Evolution of Graphics Hardware

Many of the features that distinguished the advanced graphics hardware implementations of a few years ago are now available on inexpensive personal computer (PC) systems. At the same time, the capabilities of more powerful workstations have continued to grow.

A. Some Evolutionary Steps for Graphics Hardware

Many of the ideas and algorithms for rendering realistic images have been known for some time and used in noninteractive applications. The past 15 years have seen a steady transition of many of those algorithms to sophisticated hardware accelerated implementations, as improvements in semiconductor technology and design skills, coupled with entrepreneurial drive, made them commercially viable.

1) Flight Simulators: Much of the early real-time work was focused around the visual simulation system, it being one of the few application areas where the high cost of building such a system was acceptable. Early simulation systems cost hundreds of thousands or even millions of dollars to build and emphasized constant frame rate coupled with very dynamic scene content. Much of the flight-simulator evolution addressed increasing scene realism rather than raw polygon performance [50], [51].

2) Vector Displays: In parallel with the flight-simulator work, a large amount of attention was devoted to computer-aided design (CAD) and manufacturing (CAM) applications throughout the late 1960's and 1970's. Much of the CAD/CAM work made use of vector graphic displays (or terminals) connected to large centralized computers. Vector systems displayed an image as a series of line segments. The electron beam of the display traced out each line segment. These vector displays excelled at drawing monochrome, high-quality lines but were not well suited to displaying solid shapes or shaded surfaces. Also, the early computers did not provide enough computational power to manipulate sophisticated wire-frame models interactively. Early vector terminals ranged in price from \$10 000–100 000 depending on the resolution and speed of the display [52], [53].

3) Raster Displays: Around the late 1970's, raster display systems started to appear in both personal/hobbyist computers and in terminals connected to larger time-sharing machines. Raster systems store and display the images as a two-dimensional rectangle of pixels. The electron beam of the display scans each raster location and illuminates the screen proportionally to the value stored in the corresponding memory location. Such displays became popular as the cost of semiconductor memory declined to a level where it was practical for the display to be used to hold a moderate resolution monochrome image. For example, a 1024×1024 1-bit display requires 128 kbytes of memory. As memory densities increased, color display systems became feasible. Early color displays used a table lookup scheme (color map) to convert a shallow index value (e.g., 4 or 8 bit) to a wider 24-bit red-green-blue (RGB) color value. As densities increased further, true-color RGB displays became feasible.

4) Frame Buffers: During the late 1970's and early 1980's, graphics researchers typically computed images on a host computer and displayed them in a color raster display device called a *frame buffer* attached to the host. These frame buffers typically had little or no hardware acceleration for 3-D operations.

5) First-Generation Acceleration: By the mid-1970's, the notion of packaging a minicomputer, a terminal, and turnkey application software for CAD/CAM had bloomed, and by the early 1980's, it had evolved into the concept of the workstation. By the mid-1980's, workstations and terminals providing accelerated support for mechanical CAD operations were appearing. The workflow for CAD created the demand for high-quality antialiased lines (a problem absent in vector displays) and eventually for filled surfaces. CAD and similar applications demanded interactivity. To provide smooth interaction, the technique of *double buffering*—displaying the current frame in one buffer while the next frame is being computed in a second buffer—became popular and increased the amount of memory needed for storing images. Around this time, what has since become the mainstream *pipeline,* or sequence of steps used to generate 3-D images for raster graphics, started to become prevalent in hardware accelerators. Machines such as the Silicon Graphics IRIS 3000 and the Apollo DN570 were among the first generation of 3-D-graphics accelerated workstations. Workstations with these hardware accelerated pipelines cost close to \$100 000.

6) Second-Generation Acceleration: In the latter part of the 1980's, density and speed improvements in semiconductor technology made possible the addition of Gouraud shading with support for a small number of illumination models—for example, Phong lighting with support for directional, point, and spot light sources as well as support for hidden surface elimination through Z-buffering. The largest market for these machines continued to be industrial design and science, though entertainment and simulation applications were emerging. The rendering pipelines also included support for handling transparent surfaces. Much of this support came from the addition of a fourth color component known as *alpha,* which could be used to represent the opacity of the surface. Blending operations were added to allow generated pixel values to be combined with pixel values already stored in the frame buffer. These blending operations paved the way for image-processing operations and early volume-rendering algorithms. Workstations that supported these features made up the second generation of 3-D hardware acceleration and included machines such as the Apollo DN590 and Silicon Graphics GTX.

In the late 1980's and early 1990's, attention turned to the emerging VR/AR markets and the features that were required to support them. These features included texture mapping with advanced texture *filtering* to reduce aliasing artifacts in the textured image and to gain support for antialiasing of polygon edges—both critical for visual simulation applications such as flight simulators. They are also the necessary features to allow medical VR and AR applications to be built. Again, these features were only available in machines costing roughly \$100 000, but other features such as lighting, smooth shading, and double buffering migrated down to \$20 000–\$30 000 machines such as the Silicon Graphics Elan [54] and Sun ZX [55].

Early machines that supported hardware accelerated texture mapping and limited antialiasing of polygons such as the Hewlett Packard VRX, Apollo DN10000 [56], and Silicon Graphics VGX are better described as second- rather than third-generation machines their performance degraded significantly when texturing was enabled or when features such as polygon antialiasing were only partially supported.

7) Third-Generation Acceleration: Continuing in the early to mid-1990's, texture mapping and antialiasing capability were further enhanced. Again, combinations of semiconductor technology improvements and market demand (which translates to higher volumes) allowed these capabilities to migrate to \$20 000–30 000 machines while lighting, smooth shading, etc. became available in machines costing less than \$10 000. At the high end, support continued for more advanced texture-mapping features such as three- and four-dimensional textures, more texture storage, and much higher performance levels. Also, the image-processing market emerged, and the opportunity arose to reuse some of the components in the rendering pipeline to do simple image-processing operations such as convolution and lookup tables. Higher resolution color components also appeared to support these more advanced operations, and texture mapping took on a new role for manipulating images.

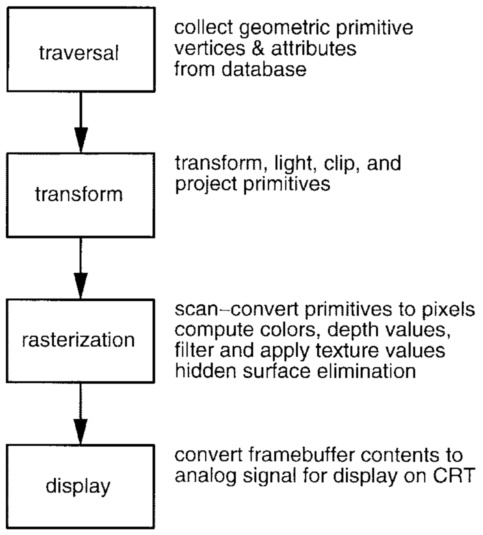
In the mid-1990's, entertainment started to become a more interesting market. Technologies such as the World Wide Web and advanced video games caused a growth in interest in systems to run entertainment applications.[3] These applications share many of the feature requirements of visual simulation, though often with greatly reduced quality requirements (e.g., reduced spatial and color resolution—320 $\times$ 200 and 5 bits per color component, frame rate, texture filtering, etc.). This has increased the demand for texture mapping and other advanced features in very inexpensive systems and is allowing simpler medical VR and AR applications to be deployed on PC-class hardware. These accelerators can arguably be classified as between second and third generation.

B. The Rendering Pipeline

Graphics hardware evolution has not just been a product of market demand but has also been made possible by advances in semiconductor and design technology. To understand the evolution of the features available in graphics accelerators, it is beneficial to understand the evolution of the implementations of the rendering pipeline.

The rendering pipeline is the sequence of processing steps needed to convert a polygonal description of an object into a set of pixels on the screen. The pipeline can be broken up into four major stages: traversal, transform, rasterization, and display (Fig. 11).

1) Traversal: The traversal stage consists of "walking through" a data structure containing the transformations, geometry, and shading attributes—the data base for the scene to be rendered and for determining the parts to be rendered in the current frame. In a large fly-through application, only part of the data base is visible, and only these parts should be sent for further processing. In some architectures, this task is left to the host processor; in others, the geometry data base is converted (compiled) into a format called a *display list*, which can be interpreted or traversed directly by the hardware accelerator.



traversal
collect geometric primitive vertices & attributes from database
transform
transform, light, clip, and project primitives
rasterization
scan-convert primitives to pixels compute colors, depth values, filter and apply texture values hidden surface elimination
display
convert framebuffer contents to analog signal for display on CRT

Fig. 11. Rendering pipeline.

2) Transform: The transform stage can be further expanded into the standard geometry pipeline [16, pp. 855–922] (see Fig. 12). The geometry pipeline takes as its input a collection of geometric primitives (points, lines, and polygons) described by their vertices. A vertex consists of its coordinate in world space (3-D space) and surface attributes such as color and normal vector. The geometry pipeline transforms the vertices into a canonical coordinate system using a modeling and viewing transformation and then discards any primitives that cannot be seen by the viewer (e.g., behind the viewer). Next, surface shading colors are computed at each vertex by evaluating an illumination model in the canonical coordinate system using the vertex attributes. Last, any partially visible primitives are *clipped* so that only the visible portion and the canonical, 3-D, lighted, and clipped vertex coordinates are projected to 2-D *screen* coordinates using an *orthographic* or *perspective* projection.

3) Computational Demands: The geometry pipeline is implemented using floating-point arithmetic to support the large dynamic range in the vertex coordinates. The geometry pipeline requires approximately 100 floating-point operations to process each vertex, assuming the simplest of illumination models and no clipping, so roughly 100 Mflops of computational power are required to process 1 million vertices/s. Using the crude approximation that connected triangular strips are used for drawing (see Fig. 13), each new vertex adds a new triangle so that 100 Mflops optimistically correspond to 1 million triangles/s. One million triangles/s corresponds to 16 000 triangles/frame at 60 frames/s.

In real applications, clipping of some primitives is unavoidable. Clipping also requires computing new shading values at any new vertices that are generated when a primitive is clipped. The addition of texture-mapping support

[3]See Angel Studios, http://www.angel.com/; MagicEdge, http://www.magic-edge-ctr.com/; and Newfire, http://www.newfire.com/.

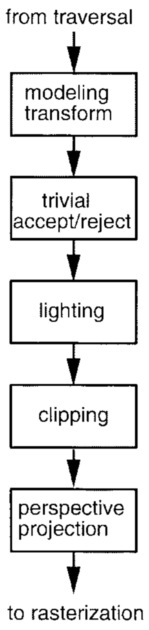


Fig. 12. Transformation (geometry) pipeline.

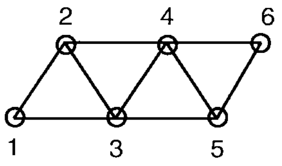


Fig. 13. Connected triangle strip (six vertices = four triangles).

requires additional operations for transforming and clipping the texture coordinates as well as the geometric coordinates. Therefore, the number of floating-point operations required to sustain triangle rates is often much higher than estimated above.

4) Rasterization: The rasterization stage follows the transformation stage. Simple primitives such as triangles and lines described by coordinates transformed to screen space are converted to pixel values and are written to the frame buffer. This stage can also be refined into a pipeline (see Fig. 14) consisting of *scan conversion* or rasterization, in which the primitive is converted into a set of discrete pixel values, using forward differences or some other iteration scheme. These pixel values consist of screen x, y, and z coordinates, an RGB color, and a set of texture coordinates. The texture coordinates are used to look up a set of texel values, i.e., the individual elements of the texture map, which are combined (filtered) to produce a color. The color computed as a result of filtering is combined with the interpolated pixel color to produce a final color. Last, the computed z value is used in the depth test for visibility, and if it passes, the color is written to the frame buffer. Over time, additional operations have been added to the rasterization stage. These include fog computations to simulate atmospheric effects, alpha testing and blending to support rendering semitransparent surfaces, stenciling (an extended depth test) and accumulation buffer operations for higher precision color computations, etc.

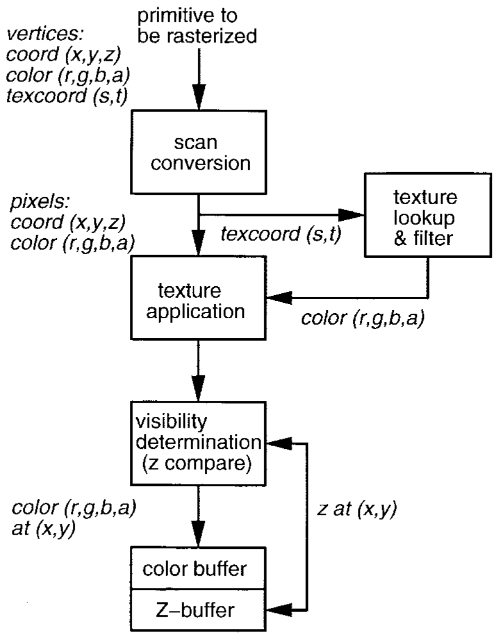


Fig. 14. Rasterization pipeline.

The rasterization stage differs from the transform stage in that the computations can be performed using fixed-point arithmetic, allowing simpler circuits. A second difference is that while the operations are simpler, the number of computations and memory operations are substantially higher than in the geometry stage.

5) Computational Demands: Assuming that the frame buffer can store a 1024 × 1024 image, we note that to write every screen pixel at a rate of 60 frames/s corresponds to 60 million rasterized pixels/s. In addition, the color and Z-buffer must be cleared with each frame. Assuming that 8-bit color components and a 24-bit Z-buffer translate to 360 Mbytes/s to clear the color and depth, and assuming that each pixel is written once and requires a single read and write to the Z-buffer, all this implies an additional 540 Mbytes/s of memory bandwidth for a total of 900 Mbytes/s. Of course, real scenes require many of the pixels to be written more than once, so the numbers can be much higher. Texture mapping increases the memory bandwidth demands even more—or at least one texel access for each rasterized pixel.

6) Display: The display portion of the pipeline involves scanning out the contents of the frame buffer and performing a conversion from the digital representation to analog signals for driving the video display. Again, to scan out a 1024 × 1024 24-bit color buffer at 60 frames/s requires an additional 180 Mbytes/s of memory bandwidth.

From this simplified description of the rendering pipeline, it is clear that substantial memory bandwidth and computational power are required. A typical 1987 processor at best provided just 10 million instructions/s of integer processing power and 1 Mflop of floating-point power.

7) Geometry Acceleration: Fortunately, the computer architecture techniques of pipelining, parallelism, and replication of critical resources are highly amenable to the rendering pipeline. In 1982, Clark [57] proposed a custom very-large-scale-integration (VLSI) circuit design for accelerating the geometry processing steps. The first machines built using these ideas used a pipeline of geometry processing modules that matched the conceptual stages of the geometry pipeline already described. The early machines provided about 5 Mflops of processing power and could process approximately 60 000 vertices/s. The actual polygon rate was much lower since the geometry pipeline was also responsible for doing setup calculations, such as computing parameters for the edge equations used by the rasterization stage for the generation of pixels.

8) Parallelism: Pipelining improves performance by breaking a task up into a sequence of operations or *stages* and allowing each stage of the pipeline to proceed concurrently, with each subsequent pipeline stage working on the results generated by the preceding stage. The pipeline approach is somewhat hindered by the need to balance the amount of work being done by each pipeline stage in order to achieve peak performance. If one of the pipeline stages, such as lighting or clipping, becomes the bottleneck, the whole pipeline slows down and the overall efficiency of the pipeline is reduced. This makes it difficult to experiment with new computationally complex features without disturbing the pipeline balance. Parallel processing is also used in some of the pipeline stages to allow greater flexibility in managing the computational load. Pipeline stages in the early Clark design consisted of multiple modules operating in parallel to perform a single task. For example, a coordinate transform consists of multiplying a four-element vector by a 4×4 matrix. This was accomplished by using four processors in parallel.

Parts of the pipeline were changed from custom VLSI chips to commercial off-the-shelf parts whenever practical. For example, floating-point digital signal processors and floating-point accelerators from Advanced Micro Devices, Inc., Weitek, Texas Instruments, and Intel [58], [59] were commonly used.

By the early 1990's, reduced instruction set computer (RISC) processors provided enough computational power to replace custom VLSI and off-the-shelf geometry processors in low-end accelerators. In 1992, a commercial RISC processor could process approximately 15 000 triangles/s. In 1997, a commercial processor is capable of supporting the geometry processing for more than 100 000 triangles/s. At the same time, high-end customized hardware is capable of using multiple processors with both pipelining and parallelization to process more than 10 million triangles/s.

9) Parallel Rasterization: Parallelism was exploited earlier on for the rasterization process. There are a number of opportunities during the rasterization steps where parallelism can be used. Once the coordinates and color information have been determined for each vertex in a primitive, each pixel can be computed independently of the others. For example, a four-vertex polygon (quadrilateral) measuring 10 pixels wide and 10 pixels high requires 100 pixel values to be computed. In the extreme case, all 100 of these pixel values could be computed in parallel. Not

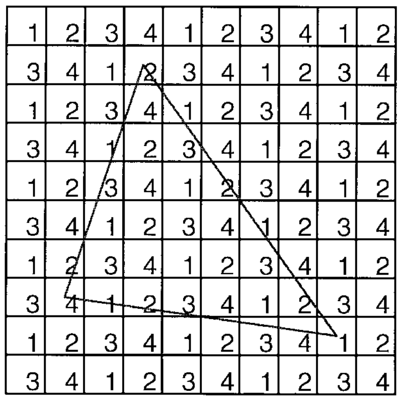


Fig. 15. Frame-buffer tiling.

only can the pixel color values but also the visibility results can be determined independently. These observations have led to the use of multiple VLSI chips to perform parallel rasterization of pixels for a given primitive [60], [61].

10) Memory Bandwidth: The use of parallelism also helps solve some problems with memory bandwidth. Frame-buffer memory was originally constructed using commercial memory technology. A single dynamic random-access memory (DRAM) chip from 1985 might have been able to support a write every 200 ns, which amounts to 5 million writes/s. To support the large aggregate bandwidths required for rasterization rates in the tens of millions of pixels/second, the individual rasterizer chips were paired with separate memory chips. Simple direct mapping schemes between screen locations and memory locations could be used to give good performance. In such a scheme, each rasterizer was sent a given primitive and each rasterizer generated only the pixels that lay within the screen region "owned" by the rasterizer. For example, if there were four rasterizers, then each rasterizer owned one-fourth of the screen. Rather than dividing the screen into quadrants, however, better load balancing could be achieved by using a fine-grain interleave pattern in which a given rasterizer "would own" one pixel in every 2×2 region (see Fig. 15).

11) Video RAM: Despite the use of multiple memories, as much as half of the memory bandwidth was used to scan out the frame-buffer contents to the video display. As raster display systems became more widespread, DRAM manufacturers found it economically viable to produce customized video memory chips (VRAM) [62], which featured a mechanism to support access for the video scan-out without interfering with the normal frame-buffer memory accesses. Throughout the late 1980's and early 1990's, pixel rates on the order of 40–80 Mpixels/s were common and sufficient for CAD applications and simple VR applications.

12) Logic Enhanced Memory: In the early 1990's, as 3-D graphics became still more popular, it became economically viable for RAM manufacturers to move some of the frame-buffer processing (e.g., depth test and blending operations) directly onto the memory chip [63]. This had the benefit of reducing the system cost and providing higher bandwidth between the memory device and the logic operations operating on the memory, with the result that fewer devices were required. The disadvantage was that to be economically viable, the chips only provided the features required by higher volume markets, so esoteric features such as support for higher resolution color or more complex frame-buffer arithmetic operations were not present. By the mid-1990's, processors such as Intel's Pentium [64] also included some of the arithmetic operations required for rasterization. A fast 1997 processor can cope with a smooth shade and depth test of 30 million pixels/s. At the high end, a massively parallel system costing several hundreds of thousands of dollars can process approximately 1 billion pixels/s.

13) Texture Mapping: When texture mapping was added to the rasterization stage, the memory bandwidth problem was exacerbated. The use of parallel rasterization techniques created a problem since each rasterizer needed access to texture memory. Unlike rasterized pixels, there was no independence in the texture memory accesses for each textured pixel. During rasterization of a triangle, different rasterizers might have needed to read the identical texel values. A brute-force solution to the bandwidth problem is simply to replicate the texture memory for each rasterizer. If the number of rasterizers is small, then the cost may be tolerable [4]. The pressure is that to keep the number of rasterizers small, to minimize texture replication, and to increase the number of frame-buffer memory chips so as to improve the aggregate memory bandwidth causes a natural decoupling or split in the raster pipeline. The scan-conversion portion of the rasterization pipeline (which includes the generation of the color, z, texture coordinates, and possibly filtered texture colors) is separated from the latter part of the pipeline consisting of the combining of the texture color and pixel color, visibility test, etc. The front end of the pipeline can be replicated less and possibly can be designed to run faster, while the back end can use more parallelism.

14) Texture Memory Bandwidth: Advanced texture filtering techniques such as mip-mapping and 3-D texture mapping also add to the bandwidth requirement. If 2-D mip-mapping or 3-D linear (trilinear) filtering is used, eight texel values are required to produce a final color. If 5-bit color components are used, then an RGB texel fits into 2 bytes. In our 60 Mpixels/s rasterization example, 960-Mbytes/s access to texture memory is required to match that rate. Fortunately, the access patterns for some sophisticated filters are regular enough so that the data can simply be partitioned into the memories and the parallel access techniques can be used to meet the bandwidth requirements. Volume rendering may use less bandwidth if 8-bit data values are used and then mapped to full-color values using a lookup table after the filtering operation. Again, filtering and address generation logic can be combined on-chip with RAM to improve bandwidth and cost [65].

Current high-end machines can process 2- and 3-D textures with sophisticated filters at speeds approaching 1 Gpixel/s. Fast CPU's with extra support for processing pixels (such as MMX [64]) can process 10 million pixels/s with smooth shading, depth testing, and the simplest of texture filters, but additional hardware acceleration is required to achieve acceptable performance for sophisticated filtering. The cost of such accelerators is very low, however—30–40 Mpixels/s for 2-D mip-mapping would rate a few hundred dollars.

During this evolution, the ability to support medical applications has improved tremendously. The first- and second-generation systems were capable of supporting scientific visualization applications and some surgical visualization. The second-generation systems were capable of supporting some interactive volume-rendering applications [66] and had large enough polygon rates to support some VR applications with limited realism. It was the development of third-generation systems supporting general AR and VR applications, however, that enabled medical AR/VR applications. In addition, a number of additional features have come together to allow truly innovative medical VR and AR applications. These features include 3-D texturing, image-processing operations, and real-time video transfer to texture memory, combined with the high polygon rates and scene realism capabilities of third-generation systems.

C. Example of Performance Comparison Between the Graphics Hardware and the CPU

Volume rendering is a basic requirement in medical-volume visualization, and in particular in AR 3-D applications. The computational complexity of volume rendering is proportional to the number of voxels ($M \times M \times M$) in the data set. Furthermore, with forward projection, the number of pixels needed to fill the screen is proportional to the screen size ($N \times N$) and the number of cross sections accumulated (usually $\sim M$). Thus, the total number of pixels required for filling the screen is $N \times N \times M$. For each of these pixels, trilinear interpolation plus blending is applied. For example, a 256 × 256 × 256 volume displayed in a 256 × 256 window will require 16 million pixels to be processed per frame. A rate of 20 frames/s requires a sustained processing rate of more than 320 million pixels/s, and the trilinear filtering requires access to more than 2.5 billion voxels/s. This rate currently falls within the capability of high-end accelerators. Using textured pixel fill rates to estimate performance, a contemporary low-end accelerator with a fill rate of 40 Mpixel/s [67] can render at the rate of 0.4/s, and a high-end accelerator with a 1-Gpixel/s fill rate [68] can sustain a rate of 62 frames/s. A 1997 host processor can support a trilinear texture rate of approximately 0.25 Mpixels/s, however, corresponding to a frame rate of 1/64 of a frame/second. This represents a performance improvement by a factor of 160 for the low end to a factor of 4000 for the high end and demonstrates the importance of hardware acceleration in enabling interactive medical applications.

D. Performance Evolution

Table 1 has been created using the published performance ratings of some commercial workstations available

Table 1 Performance Growth Rate

CPU	**Graphics hardware**						
	Generic rendering - not Z-buffered, no lighting or shading, no texture mapping or antialiasing			Z-buffered, lighted, Gouraud shaded.		Z-buffered, lighted, Gouraud shaded and antialiased texture mapping.	
	Points	Triangles	Pixels	Triangles	Pixels	Triangles	Pixels
Below: factor of growth in performance **1984-1996** for each operation (12 years):						Below: factor of growth for **1992-1996** (4 years) only:	
130	157	1100	19.56	10000	9000	10.5	3.1
Below: average factor of growth per year							
1.5	1.52	1.79	1.28	2.15	2.14	1.8	1.32

during each of the three hardware generations specified previously. The higher growth rate in graphics hardware performance compared to CPU performance is due both to an improvement in the algorithms used and to the relative freedom in graphics hardware design, particularly in exploiting parallelism, as compared to CPU design. The 1.5 factor for the CPU performance growth rate is the average between the "rule of thumb" anticipating doubling every two years ($\sim$1.41/year) and Moore's law—(1.6/year).

V. The Development of Application Programming Interfaces

To simplify the task of writing applications and to allow them to be run on newer generations of hardware without the need to rewrite the application, it is necessary to provide an API to hide the complexities of the underlying hardware under a simple abstraction. This has been a particular problem in the medical market, where the product life of a scanner (CT, MRI, etc.) can be six to ten years, while the computer hardware used by the scanner becomes obsolete much sooner.

To maximize the utility of the API, it needs to be designed with the following considerations in mind.

1) Provide sufficient expressiveness so as not to hinder the programmer from achieving the desired result (flexibility).
2) Design the API so that it can be implemented efficiently on a wide variety of hardware (scalability).
3) Provide room for growth as new techniques are discovered (extensibility).

If any of these considerations are inadequately addressed, the programmer may be tempted to bypass the API or to use a new and incompatible API, resulting in the failure of the principal goal of the API in providing a portable abstraction for application writers.

OpenGL is one solution to the programming problem [69]. It is a software interface consisting of several hundred subroutines, which allow a programmer to specify geometric objects and commands for controlling the rendering of these objects into a frame buffer. OpenGL can be thought of as a state machine that controls a set of specific drawing operations. The state machine is described by a processing pipeline, which is very similar to the rendering pipeline described in Section IV-B.

OpenGL has demonstrated application flexibility and efficient implementations on graphics hardware ranging in cost from a hundred dollars to hundreds of thousands of dollars. OpenGL is deployed in a wide variety of applications, including visual simulation, CAD, film and video production, scientific visualization, image processing, and medical imaging. It is supported by most of the major computer vendors and is actively being extended to provide new capability, such as 3-D texturing, new blending and image processing operations, etc.

VI. Summary

We are witnessing a significant event in the revolution brought about by the information age—the meeting of medicine and virtual reality. VR applications have been around for some time now, with early examples including flight simulators, the movie *Tron* [a fantasy world inside a video game (1980)], and the virtual fly through of the Martian terrain (1985). It is only in the last few years, however, that there has been a surge in the number of medical VR/AR applications available. The large-scale meeting between medicine and VR/AR has had to wait for a good reason. Most VR/AR applications demand a minimum hardware performance that in general was not met until the appearance of the third generation of graphics hardware in 1992. At the same time, continued development occurred in several discrete phases. However, there was no basis for a large-scale emergence of VR/AR applications based on real-time volume visualization when only a volume size of $32 \times 32 \times 16$ could be supported. Consequently, the performance threshold is now based upon the need for:

- real time (15–20 frames/s);
- the data set size of manipulated volumes having to be at least millions of voxels;
- sufficient available memory to contain the volume.

Both volume rendering and three-dimensional antialiased texture mapping should be performed in real time. The demands of virtual endoscopy without texture mapping are below this threshold, but then the quality is lower, too. It must be noted that computational requirements increase significantly when moving from passive observation of the data to active participation with the data. Therefore, instead of just flying through, one also needs to deform the data in real time *and* in a realistic manner, as well as to render the results in real time. Computational demands will continue to grow, and it will be some time before high-quality real-time modeling of the behavior of the body and its responses to interventions can be carried out. The complexity of the human body far outstrips any computers available today.

Additional computational power will always be welcomed. The graphics hardware accelerator has been and will continue to be vital in satisfying the demands for realism in VR/AR. Processor performance will doubtlessly both continue to improve and incorporate circuitry to assist in graphics processing. However, the same technology improvements combined with algorithmic enhancements will continue to support more rapid advancement in graphics hardware accelerator features and performance, enabling new levels of realism and interactivity in VR/AR just as it has in the past. Another field that is starting to enjoy the benefits of the fast hardware evolution is medical image processing. There has been a long-standing requirement for fast implementation of CT reconstruction by backprojection, image sharpening, and other image operations (which are also important in the context of VR/AR, supplying 3-D data in computer-aided surgery). Use of the high computational power available on the graphics hardware for image processing is therefore desirable. In most cases, however, direct support for image-processing operations is minimal and so requires the hardware to be utilized in novel and sophisticated ways—for example, in the use of texture mapping for reconstruction by backprojection [49]. It can be predicted that in the future, there will be hardware support for both graphics and image processing. The medical market will benefit greatly from such developments.

In general, the medical market has not influenced the graphics hardware evolution and has used whatever technology has been available. The experience of the authors with scanner manufacturers and other producers of imaging equipment shows that the generation problem has always been a central issue (see Section V on API's).

Computational hardware is the major technological factor responsible for the advancement and assimilation of VR/AR in medicine. Other elements are necessary for the acceptance of VR/AR application by the whole medical community, including hardware technologies (AR position sensors, ultrasound sensors, display technologies, higher screen resolution, higher dynamic range of screen intensity) as well as the man-machine interface. In addition, knowledge of body behavior and responses to surgical, electrical, or pharmacological interventions must be modeled. The software challenge is equally as important as the hardware one. With the increasing fidelity of VR/AR applications, software is going to be more and more sophisticated and time consuming to write. Good development tools will be required with many programmers in the field, and algorithms will be needed to make applications computationally efficient. The educational challenge is to make practitioners more familiar and confident with the computer.

VII. Future Perspective

Looking into the future, two basic axes are considered: the expected growth in high-end performance and continuing price reductions in the hardware, propelling its widespread use.

1) Fourth-Generation Hardware: There is enough information available now to speculate about the capabilities of fourth-generation hardware. We can anticipate that fourth-generation capabilities might include support for more advanced shading such as lighting computations performed at each pixel during rasterization, bump mapping, multiple textures applied to each polygon, and texture images being used in the per-pixel lighting computations. The supporting evidence for these conclusions includes the observation that many of these capabilities can be simulated now using multipass techniques on third-generation machines [48], and there is some suggestion of these features in vendor publications [70]. Based on the rule of thumb of at least a ten times increase in performance every four years, it is expected that in the year 2000, the graphics hardware will enable geometric processing of 100–200 million triangles/s. That means a frame rate of 30/s and about 5 million triangles/frame, which is quite an improvement in resolution.

Another potentially important development that can be expected is computational holography, i.e., using a holographic 3-D display for VR/AR. Only a real-time electronic holographic ("holovideo") display can truly create a 3-D computer graphics image with all of the depth cues (motion parallax, ocular accommodation, occlusion, etc.) and with a sufficient resolution to provide extreme realism. Video holography today requires a supercomputer to process the wave-propagation calculations. Billions of arithmetic operations are needed for even a simple and small holographic image. Moreover, the operations are based on trigonometric functions, square roots, and other computationally expensive functions. For this to be feasible, the price-performance target would need to approach 1 billion multiply-accumulate operations for each $100. The target is expected to be achieved by 2002 [71].

Off-the-shelf (OTS) computers are meeting more and more of the requirements of the manufacturers of imaging devices. This trend may revolutionize the diagnostic-imaging field by having the scanner manufacturers building only the imaging parts of the machine, letting the computation be done by OTS computers using the appropriate software.

As larger graphics and texture memories with higher bandwidth become available, system performance will increase to the point where image data representing the entire human body can be stored at the full resolution as provided by the image-acquisition system. It will also be possible to construct these graphics systems with fewer components than are required today, thus contributing to the lower cost points. Indeed, it is likely that the development of meaningful diagnostic procedures will take longer than the time required for today's highest performance research systems to become available at a cost point that supports the widespread, broad-band deployment of this technology.

As an analogy, consider that the capabilities of the most advanced graphic workstation in 1990 have arrived today in the form of consumer electronics products such as the Nintendo Ultra 64 video-game system. This represents a reduction in selling price from $100 000 to less than $150. The hardware capable of performing the ultimate in diagnostic imaging can follow that same price curve, but only if the imaging hardware community commits to making its technology widely available. In fact, the computational power of a game system such as Ultra

64 is higher than that required for low-end ultrasound applications. This raises the possibility that these devices may become ubiquitous, allowing every physician to have a three-dimensional ultrasound imaging device available.

The medical community is not yet aware of all of the possibilities that VR/AR offers. Some people think that it is starting to revolutionize medicine, predicting, for example, that virtual endoscopy will replace a significant number of "routine screening" procedures [72], and moreover predicting to the extreme that in this information age, medicine will be just "information with a medical flavor" [73]. On the other hand, objections are often heard in open discussions in conferences that although volume visualization and rendering look good, they offer no significant added value, or that surgical simulation for education and training is oversimplified and the student is misled into the belief that real surgery is also that simple. The authors of this paper believe that it is only a matter of time before VR/AR is accepted as an integral part of medicine in the same way as flight simulators have been accepted as a training tool. It seems that the acceptance of medical VR for practical use will grow with the growth in the application quality.

Acknowledgment

The authors wish to thank M. Berman and O. Shochet from Silicon Graphics Biomedical, Ltd. (SGBL); L. Jacobson, J. Foran, and P. Lacroute from Silicon Graphics, Inc. (SGI); P. Bouchaud and K. Bishop from SGI; Prof. R. Yagel from The Ohio State University for reading the manuscript and giving many useful comments; the researchers and companies for allowing the authors to use their images; E. Neria; R. Zhukov; and all SGBL people.

References

[1] S. J. Weghorst, H. B. Sieburg, and K. S. Morgan, Eds., *Proceedings of Medicine Meets Virtual Reality 4*. San Diego, CA: IOS Press, 1996.

[2] K. S. Morgan, H. M. Hoffman, D. Stredney, and S. J. Weghorst, Eds., *Proceedings of Medicine Meets Virtual Reality 5*. San Diego, CA: IOS Press, 1997.

[3] J. Proccaz, E. Grimson, and R. Mosges, Eds., *Proceedings of the First Joint Conference on Computer Vision, Virtual Reality and Robotics in Medicine and Medical Robotics and Computer-Assisted Surgery*. Grenoble, France: Springer, 1997.

[4] K. Akeley, "Reality engine graphics," in *Proc. SIGGRAPH'93,* Anaheim, CA, 1993, pp. 109–116.

[5] M. L. Rhodes, "Computer graphics in medicine: The past decade," *IEEE Comput. Graphics Applicat. Mag.,* vol. 11, no. 1, pp. 52–54, 1991.

[6] G. J. Wiet, D. Strendney, R. Yagel, J. E. Swan, N. Shareef, P. Schmalbrock, K. Wright, J. Smith, and D. E. Schuller, "Cranial base tumor visualization through high performance computing," in *Proceedings of Medicine Meets Virtual Reality 4,* S. J. Weghorst, H. B. Sieburg, and K. S. Morgan, Eds. San Diego, CA: IOS Press, 1996, pp. 43–59.

[7] J. Rolland, R. Holloway, and H. Fuchs, "A comparison of optical and video see-through head-mounted displays," in *Proc. SPIE: Telemanipulator and Telepresence Technologies,* Boston, MA, Oct. 31–Nov. 4, 1994, vol. 2351, pp. 293–307.

[8] R. T. Azuma, "A survey of augmented reality," *Presence: Teleoperators Virtual Environ.,* vol. 6, no. 4, pp. 355–385, Aug. 1997.

[9] A. State, M. A. Livingston, G. Hirota, W. F. Garret, M. C. Witton, H. Fuchs, and E. Pissano, "Techniques for augmented reality systems: Realising ultrasound-guided needle biopsies," in *Proc. SIGGRAPH'96,* New Orleans, LA, Aug. 4–9, 1996, pp. 439–446.

[10] G. J. Wiet, R. Yagel, D. Stredney, P. Schmalbrock, D. J. Sessanna, Y. Kurzion, L. Rosenberg, M. Levin, and K. Martin, "A volumetric approach to virtual simulation of functional endoscopic sinus surgery," in *Proceedings of Medicine Meets Virtual Reality 5,* K. S. Morgan, H. M. Hoffman, D. Stredney, and S. J. Weghorst, Eds. San Diego, CA: IOS Press, 1997, pp. 167–179.

[11] C. V. Edmond, D. Heskamp, D. Sluis, D. Stredney, G. J. Wiet, D. J. Sessanna, R. Yagel, S. J. Weghorst, P. Oppenheimer, J. Miller, M. Levin, and L. Rosenberg, "Simulation for ENT endoscopic surgical training," in *Proceedings of Medicine Meets Virtual Reality 5,* K. S. Morgan, H. M. Hoffman, D. Stredney, and S. J. Weghorst, Eds. San Diego, CA: IOS Press, 1997, pp. 518–528.

[12] E. Promayon, P. Baconnier, and C. Puech, "Physically-based model for simulating the human trunk respiration movements," in *Proceedings of the First Joint Conference on Computer Vision, Virtual Reality and Robotics in Medicine and Medical Robotics and Computer-Assisted Surgery,* J. Proccaz, E. Grimson, and R. Mosges, Eds. Grenoble, France: Springer, 1997, pp. 379–388.

[13] J. Kaye, D. N. Metaxas, and F. P. Primiano, Jr., "A 3D virtual environment for modeling mechanical cardiopulmonary interactions," in *Proceedings of the First Joint Conference on Computer Vision, Virtual Reality and Robotics in Medicine and Medical Robotics and Computer-Assisted Surgery,* J. Proccaz, E. Grimson, and R. Mosges, Eds. Grenoble, France: Springer, 1997, pp. 389–398.

[14] C. Johnson. (June 1997). Compuational steering. [Online]. Available: http://www.cs.utah.edu/~sci/projects/sci_comp.html.

[15] W. Lorensen. Marching through the visible man. [Online]. Available: http://www.crd.ge.com/esl/cgsp/projects/vm/.

[16] J. Foley, A. van Dam, S. Feiner, and J. Hughes, *Computer Graphics, Principles and Practice,* 2nd ed. Reading, MA: Addison-Wesley, 1990.

[17] A. Watt, *3D Computer Graphics.* Reading, MA: Addison-Wesley, 1993.

[18] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *Comput. Graphics,* vol. 21, no. 4, pp. 163–169, 1987.

[19] C. Wylie, G. W. Romney, D. C. Evans, and A. Erdahl, "Halftone perspective drawing by computer," in *Proc. AFIPS, Fall Joint Computer Conf. 31,* 1967, pp. 49–58.

[20] B. Phong, "Illumination for computer-generated pictures," *Commun. ACM,* vol. 18, no. 6, pp. 311–317, 1975.

[21] W. J. Bouknight, "A procedure for the generation of three-dimensional half-toned computer graphic presentations," *Commun. ACM,* vol. 13, no. 9, pp. 527–536, 1970.

[22] H. Gouraud, "Continuous shading of curved surfaces," *IEEE Trans. Computers,* vol. C-20, pp. 623–629, June 1971.

[23] E. Catmull, "Computer display of curved surfaces," in *Proc. IEEE Conf. Computer Graphics, Pattern Recognition and Data Structures,* Los Angeles, CA, May 1975, pp. 11–17; reprinted in H. Freeman, Ed., *Tutorial and Selected Readings in Interactive Computer Graphics.* New York: IEEE, 1980, pp. 309–315.

[24] ——, "Subdivision algorithm for the display of curved surfaces," Ph.D. dissertation, University of Utah, Salt Lake City, 1974.

[25] D. R. Peachy, "Solid texturing of complex surfaces," *Comput. Graphics,* vol. 19, no. 3, pp. 279–286, 1985.

[26] K. Perlin, "An image synthesizer," *Comput. Graphics,* vol. 19, no. 3, pp. 287–296, 1985.

[27] A. V. Oppenheim and R. W. Shafer, *Digital Signal Processing.* Englewood Cliffs, NJ: Prentice-Hall, 1975.

[28] E. Catmull, "A hidden surface algorithm with anti-aliasing," *Comput. Graphics,* vol. 12, no. 3, pp. 6–10, 1978.

[29] L. C. Carpenter, "The A-buffer, an anti-aliased hidden surface method," *Comput. Graphics,* vol. 18, no. 3, pp. 103–108, 1984.

[30] E. Fiume, A. Fournier, and L. Rudolph, "A parallel scan conversion algorithm with anti-aliasing for a general purpose ultracomputer," *Comput. Graphics,* vol. 17, no. 3, pp. 141–150, 1983.

[31] R. L. Cook, "Stochastic sampling in computer graphics," *ACM Trans. Comput. Graphics,* vol. 5, no. 1, pp. 51–72, 1986.

[32] R. L. Cook, T. Porter, and L. Carpenter, "Distributed ray tracing," *Comput. Graphics,* vol. 18, no. 3, pp. 137–145, 1984.

[33] J. F. Blinn and M. E. Newell, "Texture and reflection in computer generated images," *Commun. ACM*, vol. 19, no. 10, pp. 362–367, 1976.
[34] E. A. Feibush, M. and R. L. Cook, "Synthetic texturing using digital filters," *Comput. Graphics*, vol. 14, no. 3, pp. 294–301, 1980.
[35] M. Ganget, D. Perny, and P. Coueignoux, "Perspective mapping of plannar textures," in *Proceedings of EUROGRAPHICS'82.* Amsterdam, The Netherlands: North-Holland, 1982, pp. 57–71.
[36] N. Greene and P. S. Heckbert, "Creating raster omnimax images using the elliptically weighted average filter," *IEEE Comput. Graphics and Applications Mag.*, vol. 6, no. 6, pp. 21–27, 1986.
[37] L. Williams, "Pyramid parametrics," *Comput. Graphics*, vol. 17, no. 3, pp. 1–11, 1983.
[38] T. Porter and T. Duff, "Compositing digital images," in *Proc. SIGGRAPH'84*, Minneapolis, MN, Aug. 1984, pp. 253–259.
[39] R. A. Harris, D. Lowell, *et. al.*, "Non-invasive numerical dissection and display of anatomic structure using computerized X-ray tomography," in *Proc. SPIE*, vol. 152, 1978, pp. 10–18.
[40] K. H. Hoehne, R. L. Delapaz, R. B. Stein, and R. C. Taylor, "Combined surface display and reformatting for the three-dimensional analysis of tomographic data," in *Investigative Radiology*, vol. 27, no. 7, pp. 658–664, July 1987.
[41] S. M. Jaffey and K. Dutta, "Digital perspective correction for cylindrical holographic stereogram," in *Proc. SPIE.*, Aug. 1982, vol. 367, pp. 130–140.
[42] D. S. Schlusselberg, W. K. Smith, D. J. Woodward, "Three-dimensional display of medical image volumes," in *Proc. NCGA*, Mar. 1986.
[43] R. A. Drebin, L. Carpenter, and P. Hanrahan, "Volume rendering," in *Proc. SIGGRAPH'88*, J. Dill, Ed., Atlanta, GA, Aug. 1988, vol. 22, pp. 65–74.
[44] M. Levoy, "Display of surfaces from volume data," *Comput. Graphics Applicat.*, vol. 8, no. 3, pp. 29–37, May 1988.
[45] S. M. Jaffey, M. Stephen, and K. Dutta, "Digital reconstruction methods for three-dimensional image visualization," in *Proc. SPIE: Processing and Display of Three-Dimensional Data II*, vol. 507, 1984, pp. 155–163.
[46] M. Levoy, "Efficient ray tracing of volume data," *ACM Trans. Graphics*, vol. 9, no. 3, pp. 245–261, 1990.
[47] H. Fuchs, J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, B. Tebbs, and L. Israel, "Pixel-planes 5: A heterogeneous multiprocessor graphics system using processor-enhanced memories," in *Proc. SIGGRAPH'89*, Boston, MA, pp. 79–88.
[48] T. McReynolds, D. Blythe, C. Fowler, P. Womack, S. Hui, and B. Grantham, "Programming with OpenGL: Advanced techniques," *SIGGRAPH'97 Course Notes*, Los Angeles, CA, 1997, pp. 71–80.
[49] B. Cabral, N. Cam, and J. Foran, "Accelerated volume rendering and tomographic reconstruction using texture mapping hardware," in *Proceedings of the ACM/IEEE Symposium on Volume Visualization.* New York: IEEE Press, 1994, pp. 91–98.
[50] M. Bunker and R. Economy, "Evolution of the GE CIG systems," General Electric Company, Daytona Beach, FL, SCSD Doc., 1989.
[51] R. Schumacker, "A new visual system architecture," in *Proc. 2nd Interservice/Industry Training Equipment Conf.*, Salt Lake City, UT, Nov. 1980, pp. 16–20.
[52] C. Machover, "A brief, personal history of computer graphics," *Computer*, pp. 38–45, Nov. 1978.
[53] W. Myers, "Interactive computer graphics: Poised for takeoff?" *Computer*, pp. 60–74, Jan. 1978.
[54] C. Harell and F. Fouladi, "Graphics rendering architecture for a high performance desktop workstation," in *Proc. SIGGRAPH'93*, Anaheim, CA, Aug. 1993, pp. 93–100.
[55] M. Deering and S. R. Nelson, "Leo: A system for cost effective 3D shaded graphics," in *Proc. SIGGRAPH'93*, Aug. 1993, pp. 101–108.
[56] D. Kirk and D. Voorhies, "The rendering architecture of the DN10000VS," in *Proc. SIGGRAPH'90*, Dallas, TX, Aug. 1990, pp. 299–308.
[57] J. Clark, "The geometry engine: A VLSI geometry subsystem for graphics," in *Proc. SIGGRAPH'82*, Boston, MA, Aug. 1982, pp. 127–133.
[58] K. Akeley, "The silicon graphics 4D/240GTX superworkstation," *Comput. Graphics Applicat.*, vol. 9, no. 4, pp. 71–83, July 1989.
[59] J. Grimes, L. Kohn, and R. Bharadhwaj, "The Intel i860 64-bit processor: A general purpose CPU with 3D graphics capabilities," *Comput. Graphics Applicat.*, vol. 9, no. 4, pp. 85–94, July 1989.
[60] J. Clark and M. Hannah, "Distributed processing in a high-performance smart image memory," *Lambda(VLSI Design)*, vol. 1, no. 3, pp. 40–45, 1980.
[61] H. Fuchs and J. Poulton, "Pixel-planes: A VLSI-oriented design for a raster graphics engine," *Lambda(VLSI Design)*, vol. 2, no. 3, pp. 20–28, 1981.
[62] R. Pinkham, M. Novak, and K. Guttag, "Video RAM exceeds at fast graphics," *Electron. Design*, vol. 31, no. 17, pp. 161–182, Aug. 18, 1983.
[63] M. Deering, S. Schlapp, and M. Lavelle, "FBRAM: A new form of memory optimized for 3D graphics," in *Proc. SIGGRAPH'94*, Orlando, FL, Aug. 1994, pp. 167–174.
[64] A. Peleg, S. Wilkie, and U. Weiser, "Intel MMX for multimedia PC's," *Commun. ACM*, vol. 40, no. 1, pp. 25–38, Jan. 1997.
[65] A. Schilling, G. Knittel, and W. Strasser, "Texram: A smart memory for texturing," *Comput. Graphics Applicat.*, vol. 16, pp. 32–41, May 1996.
[66] P. LoPiccolo, D. Phillips Mahoney, A. Vasilopoulis, and S. Porter, "The visible volume," *Comput. Graphics World*, vol. 14, no. 4, pp. 44–80, Apr. 1991.
[67] M. Kilgard, "Realizing OpenGL: Two implementations of one architecture," in *Proc. 1997 SIGGRAPH Eurographics Workshop*, Los Angeles, CA, Aug. 1997, pp. 45–55.
[68] J. S. Montrym, D. R. Baum, D. L. Dignam, and C. J. Migdal, "InfiniteReality: A real-time graphics system," in *Proc. SIGGRAPH'97*, Los Angeles, CA, Aug. 1997, pp. 293–302.
[69] J. Neider, T. Davis, and M. Woo, *OpenGL Programming Guide.* Reading, MA: Addison-Wesley, 1993.
[70] M. A. Cosman and R. L. Grange, "CIG scene realism: The world tomorrow," in *Proc. 18th I/ITSEC Conf.*, Orlando, FL, Dec. 1996, pp. 627–639.
[71] M. Lucente, "Interactive three-dimensional holographic displays: Seeing the future in depth," *Comput. Graphics*, vol. 31, no. 2, pp. 63–67, May 1997.
[72] S. Jones and R. M. Satava, "Virtual endoscopy of the head and neck," in *Proc. MMVR'96*, San Diego, CA, pp. 152–156.
[73] R. M. Satava, "Medical virtual reality: The current status of the future," in *Proc. MMVR'96*, San Diego, CA, pp. 100–106.

Ziv Soferman received the B.Sc. degree in mathematics and physics and the M.Sc. degree in physics from The Hebrew University of Jerusalem, Israel, in 1976 and 1982, respectively. He received the Ph.D. degree in applied mathematics and computer science from the Weizmann Institute of Science, Israel, in 1989 in the field of image processing.

From 1989 to 1991, he conducted postdoctoral research at the University of Colorado, Denver. During this period and afterwards, he was a Consultant and Inventor for high-tech companies in Israel and the United States in the areas of image processing and computer vision for air and space imagery as well as for the semiconductor and biological industries, where some of his inventions were made into products. In 1995, he joined Silicon Graphics Biomedical, Ltd., Jerusalem, a subsidiary of Silicon Graphics, Inc., where he currently is Chief Scientist. He developed novel ways of using mathematical techniques to increase the computational capabilities of graphics hardware and to support efficient design of future models. He also developed new methods for two- and three-dimensional medical imaging. His fields of interest are image processing, computer vision, medical imaging, computerized microscopy, and algorithmic methods for hardware exploitation.

David Blythe (Member, IEEE) received the B.Sc. and M.Sc. degrees in computer science from the University of Toronto, Ont., Canada, in 1983 and 1985, respectively.

From 1985 to 1988, he was a Research Assistant for the Computer Systems Research Group at the University of Toronto. From 1989 to 1991, he worked in the visualization laboratory of the Ontario Centre for Large Scale Computation. Since 1991, he has been with the Advanced Systems Division of Silicon Graphics, Inc., Mountain View, CA, where he has contributed to the development of the RealityEngine and InfiniteReality graphics accelerators and the OpenGL graphics library standard. He currently is a Principal Engineer in the Advanced Software Group, where his interests include computer graphics algorithms and high-level toolkit design.

Nigel W. John received the B.Sc. degree in mathematics and computing and the Ph.D. degree in mathematical sciences from the University of Bath, U.K., in 1987 and 1990, respectively.

He spent five years in the chemical/pharmaceutical industry, where he was involved in the development of an early three-dimensional ultrasound system for use in pharmaceutical research. He currently is a Project Leader for Silicon Graphics Biomedical, Ltd., Manchester, U.K. He is a Member of the Eurographics U.K. Executive Committee. He has been active in computer graphics since 1987. His current interests lie with developing medical imaging and visualization solutions.