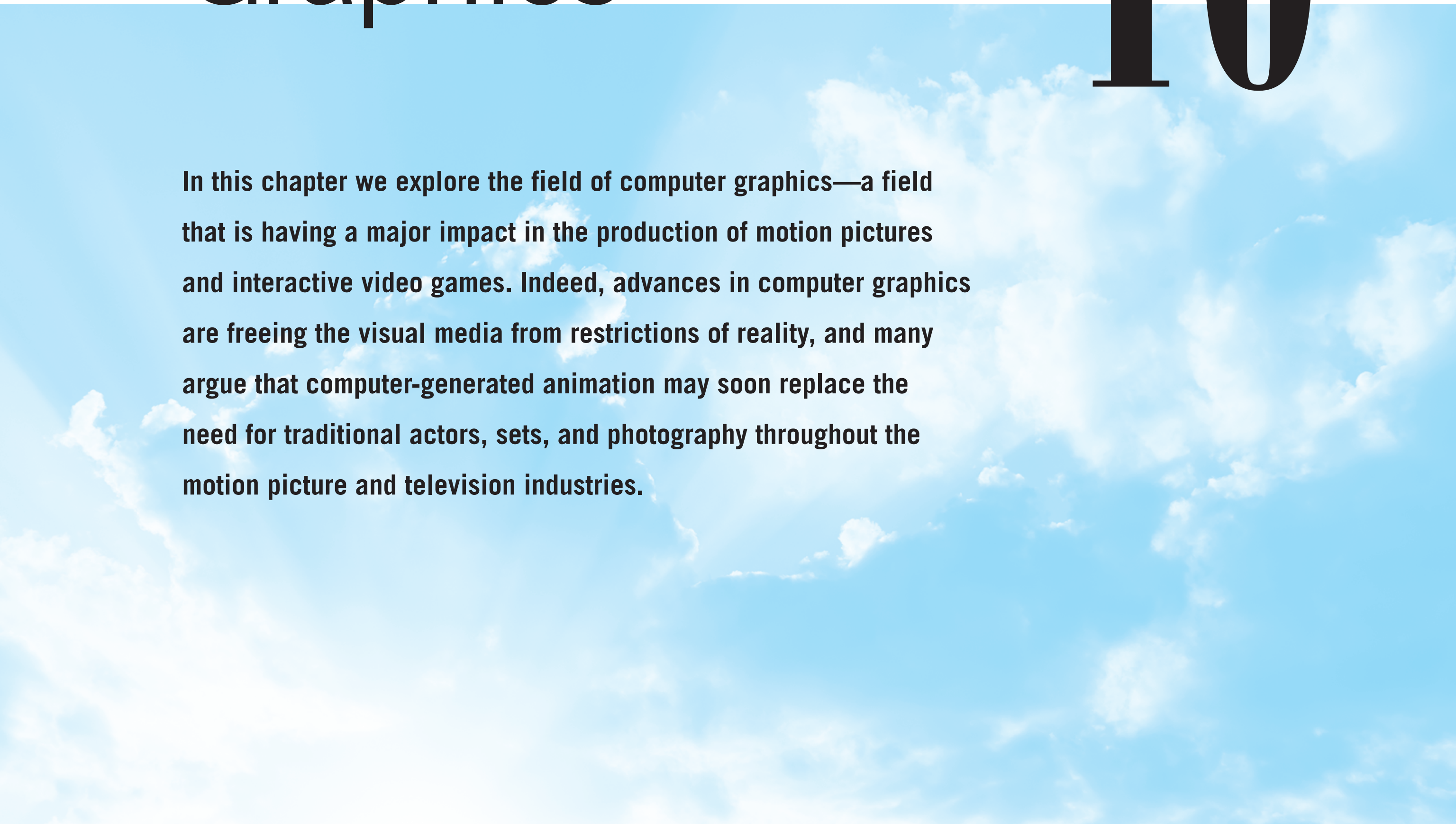
CHAPTER 10

Computer Graphics



In this chapter we explore the field of computer graphics—a field that is having a major impact in the production of motion pictures and interactive video games. Indeed, advances in computer graphics are freeing the visual media from restrictions of reality, and many argue that computer-generated animation may soon replace the need for traditional actors, sets, and photography throughout the motion picture and television industries.

**Asterisks indicate suggestions for optional sections.*

Computer graphics is the branch of computer science that applies computer technology to the production and manipulation of visual representations. It is associated with a wide assortment of topics including the presentation of text, the construction of graphs and charts, the development of graphical user interfaces, the manipulation of photographs, the production of video games, and the creation of animated motion pictures. However, the term *computer graphics* is increasingly being used in reference to the specific field called 3D graphics, and most of this chapter concentrates on this topic. We begin by defining 3D graphics and clarifying its role within the broader interpretation of computer graphics.

10.1 The Scope of Computer Graphics

With the emergence of digital cameras, the popularity of software for manipulating digitally encoded images has rapidly expanded. This software allows one to touch up photographs by removing blemishes and the dreaded "red eye," as well as cutting and pasting portions from different photographs to create images that do not necessarily reflect reality.

Similar techniques are often applied to create special effects in the motion picture and television industries. In fact, such applications were major motivating factors for these industries shifting from analog systems such as film to digitally encoded images. Applications include removing the appearance of support wires, overlaying multiple images, or producing short sequences of new images that are used to alter the action that was originally captured by a camera.

In addition to software for manipulating digital photographs and motion picture frames, there is now a wide variety of utility/application software packages that assist in producing two-dimensional images ranging from simple line drawings to sophisticated art. (A well-known elementary example is Microsoft's application called Paint.) At a minimum, these programs allow the user to draw dots and lines, insert simple geometric shapes such as ovals and rectangles, fill regions with color, and cut and paste designated portions of a drawing.

Note that all the preceding applications deal with the manipulation of flat two-dimensional shapes and images. They are, therefore, examples of two related fields of research: One is **2D graphics,** the other is **image processing.** The distinction is that 2D graphics focuses on the task of converting two-dimensional shapes (circles, rectangles, letters, etc.) into patterns of pixels to produce an image, whereas image processing, which we will meet later in our study of artificial intelligence, focuses on analyzing the pixels in an image in order to identify patterns that can be used to enhance or perhaps "understand" the image. In short, 2D graphics deals with producing images while image processing deals with analyzing images.

In contrast to converting two-dimensional shapes into images as in 2D graphics, the field of **3D graphics** deals with converting three-dimensional shapes into images. The process is to construct digitally encoded versions of three-dimensional scenes and then to simulate the photographic process to produce images of those scenes. The theme is analogous to that of traditional photography, except that the scene that is "photographed" using 3D graphics techniques does not exist as a physical reality but instead "exists" merely as a collection of data and algorithms. Thus, 3D graphics involves "photographing" virtual worlds, whereas traditional photography involves photographing the real world.

It is important to note that the creation of an image using 3D graphics encompasses two distinct steps. One is the creation, encoding, storage, and manipulation of the scene to be photographed. The other is the process of producing the image. The former is a creative, artistic process; the latter is a computationally intense process. These are topics that we will explore in the next four sections.

The fact that 3D graphics produces "photographs" of virtual scenes makes it ideal for use in interactive video games and animated motion picture productions where the shackles of reality would otherwise limit the action. An interactive video game consists of an encoded three-dimensional virtual environment with which the game player interacts. The images that the player sees are produced by means of 3D graphics technology. Animated motion pictures are created in a similar manner, except that it is the human animator who interacts with the virtual environment rather than the ultimate viewer. The product ultimately distributed to the public is a sequence of two-dimensional images as determined by the production's director/producer.

We will investigate the use of 3D graphics in animation more thoroughly in Section 10.6. For now, let us close this section by imagining where these applications may lead as 3D graphics technology advances. Today, motion pictures are distributed as sequences of two-dimensional images. Although the projectors that display this information have progressed from analog devices with reels of film to digital technology using DVD players and flat panel displays, they still deal only with two-dimensional representations.

Imagine, however, how this may change as our ability to create and manipulate realistic three-dimensional virtual worlds improves. Rather than "photographing" these virtual worlds and distributing a motion picture in the form of two-dimensional images, we could distribute the virtual worlds. A potential viewer would receive access to the motion picture set rather than just the motion picture. This three-dimensional set would then be viewed by means of a "3D graphics projector" in much the same way that video games are viewed by special-purpose "game boxes." One might first watch a "suggested plot" that would result in viewing the motion picture as the director/producer envisioned. But, the viewer could also interact with the virtual set in a manner reminiscent of a video game to produce other scenarios. The possibilities are extensive, especially when we also consider the potentials of the three-dimensional human-machine interfaces that are being developed.

Questions & Exercises

1. Summarize the distinction between image processing, 2D graphics, and 3D graphics.
2. How does 3D graphics differ from traditional photography?
3. What are the two major steps in producing a "photograph" using 3D graphics?

10.2 Overview of 3D Graphics

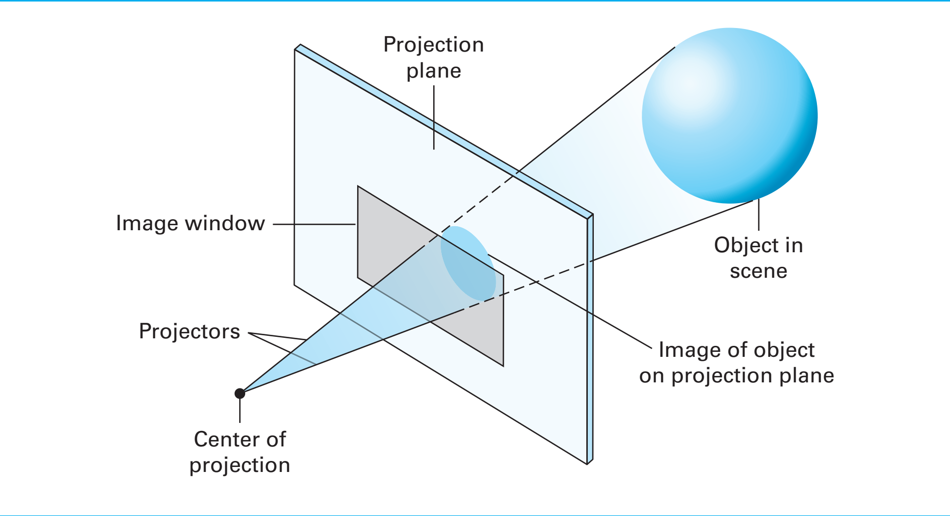
Let us begin our study of 3D graphics by considering the entire process of creating and displaying images—a process that consists of three steps: modeling, rendering, and displaying. The modeling step (which we will explore in detail in Section 10.3) is analogous to designing and constructing a set in the traditional motion picture industry, except that the 3D graphics scene is "constructed" from digitally encoded data and algorithms. Thus, the scene produced in the context of computer graphics may never exist in reality.

The next step is to produce a two-dimensional image of the scene by computing how the objects in the scene would appear in a photograph made by a camera at a specified position. This step is called **rendering**—the subject of Sections 10.4 and 10.5. Rendering involves applying the mathematics of analytic geometry to compute the projection of the objects in the scene onto a flat surface known as the **projection plane** in a manner analogous to a camera projecting a scene onto film (Figure 10.1). The type of projection applied is a **perspective projection,** which means that all objects are projected along straight lines, called **projectors,** that extend from a common point called the **center of projection,** or the **view point.** (This is in contrast to a **parallel projection** in which the projectors are parallel. A perspective projection produces a projection similar to that seen by the human eye, whereas a parallel projection produces a "true" profile of an object, which is often useful in the context of engineering drawings.)

The restricted portion of the projection plane that defines the boundaries of the final image is known as the **image window.** It corresponds to the rectangle that is displayed in the viewfinder of most cameras to indicate the boundaries of the potential picture. Indeed, the viewfinder of most cameras allows you to view more of the camera's projection plane than merely its image window. (You may see the top of Aunt Martha's head in the viewfinder, but unless the top of her head is within the image window, it will not appear in the final picture.)

Once the portion of the scene that projects into the image window is identified, the appearance of each pixel in the final image is computed. This

Figure 10.1 The 3D graphics paradigm



pixel-by-pixel process can be computationally complex because it requires determining how the objects in the scene interact with light—a hard, shiny surface in bright light should be rendered differently than a soft, transparent surface in indirect light. In turn, the rendering process borrows heavily from numerous fields including material science and physics. Moreover, determining the appearance of one object often requires knowledge about other objects in the scene. The object may be in the shadow of another object, or the object may be a mirror whose appearance is essentially that of another object.

As the appearance of each pixel is determined, the results are stored collectively as a bit map representation of the image in a storage area called the **frame buffer.** This buffer may be an area of main memory or, in the case of hardware designed specifically for graphics applications, it may be a block of special purpose memory circuitry.

Finally, the image stored in the frame buffer is either displayed for viewing or transferred to more permanent storage for later display. If the image is being produced for use in a motion picture, it may be stored and perhaps even modified before final presentation. However, in an interactive video game or flight simulator, images must be displayed as they are produced on a real-time basis, a requirement that often limits the quality of the images created. This is why the graphics quality of full-feature animated productions distributed by the motion picture industry exceeds that of today's interactive video games.

We close our introduction to 3D graphics by analyzing a typical video game system. The game itself is essentially an encoded virtual world together with software that allows that world to be manipulated by the game player. As the player manipulates that world, the game system repeatedly renders the scene and stores the image in the image buffer. To overcome real-world time constraints, much of this rendering process is handled by special-purpose hardware. Indeed, the presence of this hardware is a distinguishing feature between a game system and a generic personal computer. Finally, the display device in the game system displays the contents of the frame buffer, giving the player the illusion of a changing scene.

Questions & Exercises

1. Summarize the three steps involved in producing an image using 3D graphics.
2. What is the difference between the projection plane and the image window?
3. What is a frame buffer?

10.3 Modeling

A 3D computer graphics project begins in much the same way as a theatrical stage production—a set must be designed and the required props must be collected or constructed. In computer graphics terminology, the set is called a **scene** and the props are called **objects.** Keep in mind that a 3D graphics scene is virtual because it consists of objects that are "constructed" as digitally encoded models rather than tangible, physical structures.

In this section we will explore topics related to "constructing" objects and scenes. We begin with issues of modeling individual objects and conclude by considering the task of collecting those objects to form a scene.

Modeling Individual Objects

In a stage production, the extent to which a prop conforms to reality depends on how it will be used in the scene. We may not need an entire automobile, the telephone does not have to be functional, and the background scenery may be painted on a flat backdrop. Likewise, in the case of computer graphics, the degree to which the software model of an object accurately reflects the true properties of the object depends on the requirements of the situation. More detail is necessary to model objects in the foreground than objects in the background. Moreover, more detail can be produced in those cases that are not under stringent, real-time constraints.

Thus, some object models may be relatively simple whereas others may be extremely complex. As a general rule, more precise models lead to higher-quality images but longer rendering times. In turn, much of the ongoing research in computer graphics seeks the development of techniques for constructing highly detailed, yet efficient, object models. Some of this research deals with developing models that can provide different levels of detail depending on the object's ultimate role in the scene, the result being a single object model that can be used in a changing environment.

The information required to describe an object includes the object's shape as well as additional properties, such as surface characteristics that determine how the object interacts with light. For now, let us consider the task of modeling shape.

Shape

The shape of an object in 3D graphics is usually described as a collection of small flat surfaces called **planar patches,** each of which is the shape of a polygon. Collectively, these polygons form a **polygonal mesh** that approximates the shape of the object being described (Figure 10.2). By using small planar patches, the approximation can be made as precise as needed.

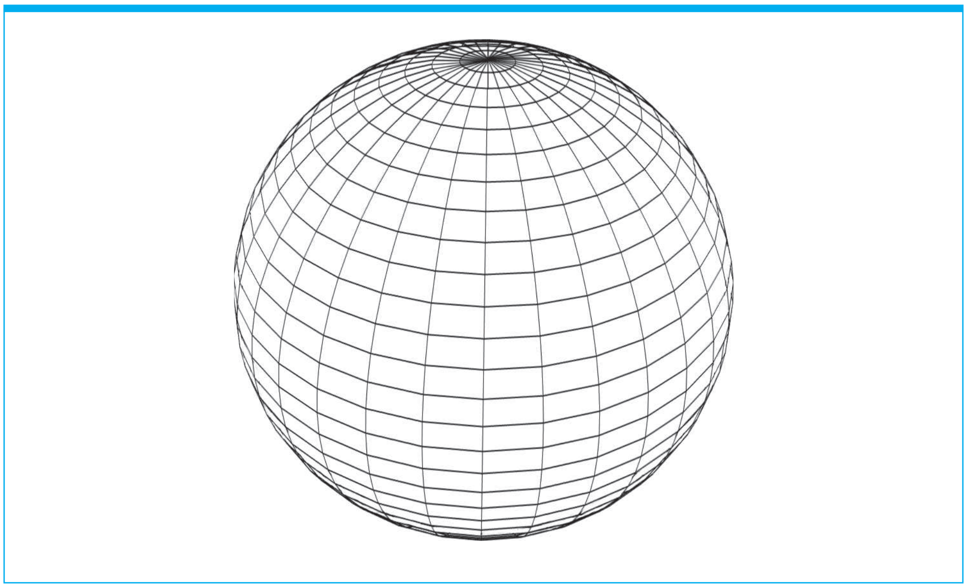
The planar patches in a polygonal mesh are often chosen to be triangles because each triangle can be represented by its three vertices, which is the minimum number of points required to identify a flat surface in three-dimensional space. In any case a polygonal mesh is represented as the collection of the vertices of its planar patches.

A polygonal mesh representation of an object can be obtained in a variety of ways. One is to begin with a precise geometric description of the desired shape, and then use that description to construct a polygonal mesh. For example, analytic geometry tells us that a sphere (centered at the origin) with radius r is described by the equation

$$r^2 = x^2 + y^2 + z^2$$

Based on this formula, we can establish equations for lines of latitude and longitude on the sphere, identify the points where these lines intersect, and then use these points as the vertices of a polygonal mesh. Similar techniques can be applied to other traditional geometric shapes, and this is why characters in

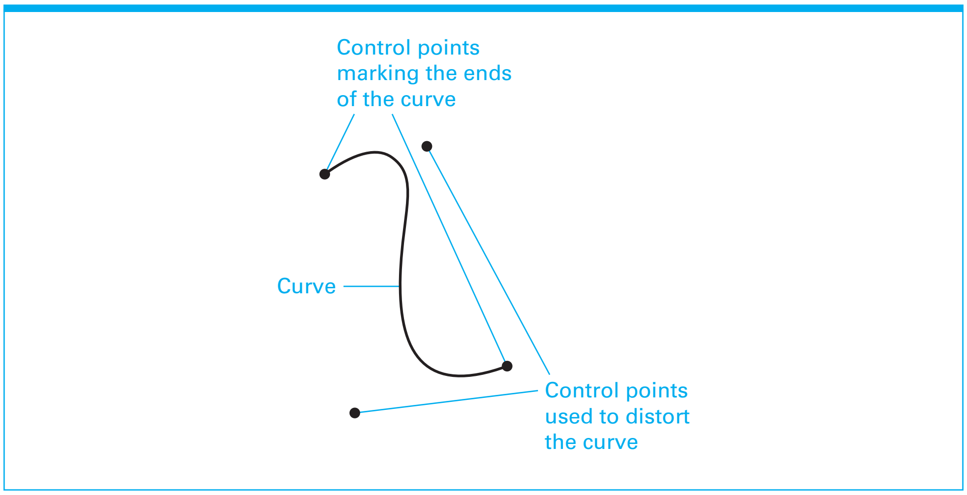
Figure 10.2 A polygonal mesh for a sphere



less-expensive computer-generated animations often appear to be pieced together from such structures as spheres, cylinders, and cones.

More general shapes can be described by more sophisticated analytical means. One is based on the use of **Bezier curves** (named after Pierre Bezier who developed the concept in the early 1970s as an engineer for the Renault car company), which allow a curved line segment in three-dimensional space to be defined by only a few points called control points—two of which represent the ends of the curve segment while the others indicate how the curve is distorted. As an example, Figure 10.3 displays a curve defined by four control points. Note how the curve appears to be pulled toward the two control points that do not identify the segment ends. By moving these points, the curve can be twisted into

Figure 10.3 A Bezier curve


Control points marking the ends of the curve
Curve
Control points used to distort the curve

different shapes. (You may have experienced such techniques when constructing curved lines using drawing software packages such as Microsoft's Paint.) Although we will not pursue the topic here, Bezier's techniques for describing curves can be extended to describe three-dimensional surfaces, known as **Bezier surfaces.** In turn, Bezier surfaces have proven to be an efficient first step in the process of obtaining polygonal meshes for complex surfaces.

You may ask why it is necessary to convert a precise description of a shape, such as the concise formula of a sphere or the formulas describing a Bezier surface, into an approximation of the shape using a polygonal mesh. The answer is that representing the shape of all objects by polygonal meshes establishes a uniform approach to the rendering process—a trait that allows entire scenes to be rendered more efficiently. Thus, although geometric formulas provide precise descriptions of shapes, they serve merely as tools for constructing polygonal meshes.

Another way of obtaining a polygonal mesh is to construct the mesh in a brute force manner. This approach is popular in cases where a shape defies representation by elegant mathematical techniques. The procedure is to build a physical model of the object and then to record the location of points on the surface of the model by touching the surface with a pen device that records its position in three-dimensional space—a process known as **digitizing.** The collection of points obtained can then be used as vertices to obtain a polygonal mesh describing the shape.

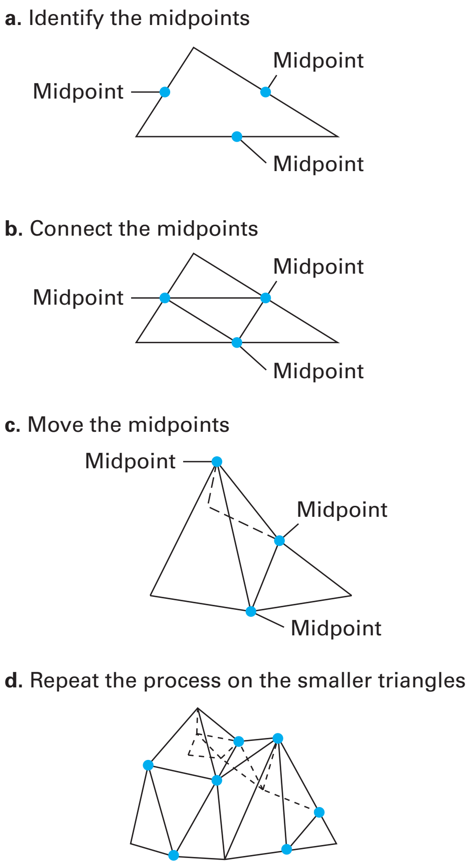
Unfortunately, some shapes are so complex that obtaining realistic models by geometric modeling or manual digitizing is unfeasible. Examples include intricate plant structures such as trees, complex terrain such as mountain ranges, and gaseous substances such as clouds, smoke, or the flames of a fire. In these cases, polygonal meshes can be obtained by writing programs that construct the desired shape automatically. Such programs are collectively known as **procedural models.** In other words, a procedural model is a program unit that applies an algorithm to generate a desired structure.

As an example, procedural models have been used to generate mountain ranges by executing the following steps: Starting with a single triangle, identify the midpoints of the triangle's edges (Figure 10.4a). Then, connect these midpoints to form a total of four smaller triangles (Figure 10.4b). Now, while holding the original triangle's vertices fixed, move the midpoints in three-dimensional space (allowing the triangle's edge lines to stretch or contract), thus distorting the triangular shapes (Figure 10.4c). Repeat this process with each of the smaller triangles (Figure 10.4d), and continue repeating the process until the desired detail is obtained.

Procedural models provide an efficient means of producing multiple complex objects that are similar yet unique. For instance, a procedural model can be used to construct a variety of realistic tree objects—each with its own, yet similar, branching structure. One approach to such tree models is to apply branching rules to "grow" tree objects in much the same way that a parser (Section 6.4) constructs a parse tree from grammar rules. In fact, the collection of branching rules used in these cases is often called a grammar. One grammar may be designed for "growing" pine trees whereas another may be designed for "growing" oaks.

Another method of constructing procedural models is to simulate the underlying structure of an object as a large collection of particles. Such models are called **particle systems.** Usually particle systems apply certain predefined rules to move the particles in the system, perhaps in a manner reminiscent of molecular

Figure 10.4 Growing a polygonal mesh for a mountain range


a. Identify the midpoints
Midpoint
Midpoint
Midpoint
b. Connect the midpoints
Midpoint
Midpoint
Midpoint
c. Move the midpoints
Midpoint
Midpoint
Midpoint
d. Repeat the process on the smaller triangles

interactions, to produce the desired shape. As an example, particle systems have been used to produce an animation of sloshing water as we will see later in our discussion of animation. (Imagine a bucket of water modeled as a bucket of marbles. As the bucket rocks, the marbles tumble around, simulating the movement of water.) Other examples of particle system applications include flickering fire flames, clouds, and crowd scenes.

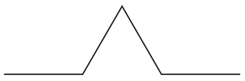
The output of a procedural model is usually a polygonal mesh that approximates the shape of the desired object. In some cases, such as generating a mountain range from triangles, the mesh is a natural consequence of the generating process. In other cases, such as growing a tree from branching rules, the mesh may be constructed as an additional, final step. For example, in the case of particle systems, the particles on the outer edge of the system are natural candidates for the vertices of the final polygonal mesh.

How precise the mesh generated by a procedural model may be depends on the situation. A procedural model for a tree in a scene's background may produce a course mesh that reflects only the basic shape of the tree, whereas a procedural model for a tree in the foreground may produce a mesh that distinguishes individual branches and leaves.

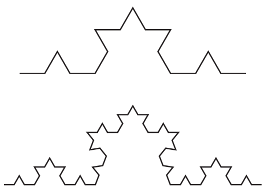
Fractals

The construction of a mountain range by means of a procedural model as described in the text (see Figure 10.4) is an example of the role that fractals play in 3D graphics. Technically speaking, a **fractal** is a geometric object whose "Hausdorff dimension is greater than its topological dimension." What this means intuitively is that the object is formed by "packing together" copies of an object of a lower dimension. (Think of the illusion of width created by "packing together" multiple parallel line segments.) A fractal is usually formed by means of a recursive process, where each activation in the recursion "packs together" additional (yet smaller) copies of the pattern being used to build the fractal. The fractal that results is self-similar in that each portion, when magnified, appears as a copy of itself.

A traditional example of a fractal is the von Koch snowflake that is formed by repeatedly replacing the straight-line segments in the structure



with smaller versions of the same structure. This leads to a sequence of refinements that proceeds as follows:



Fractals are often the backbone of procedural models in the field of 3D graphics. Indeed, they have been used to produce realistic images of mountain ranges, vegetation, clouds, and smoke.

Surface Characteristics A model consisting merely of a polygonal mesh captures only the shape of an object. Most rendering systems are capable of enriching such models during the rendering process to simulate various surface characteristics as requested by the user. For example, by applying different shading techniques (which we will introduce in Section 10.4) a user may specify that a polygonal mesh for a ball be rendered as a red smooth ball or a green rough ball. In some cases, such flexibility is desirable. But in situations requiring faithful rendering of an original object, more specific information about the object must be included in the model so that the rendering system will know what it should do.

There are a variety of techniques for encoding information about an object in addition to its shape. For instance, along with each vertex in a polygonal mesh, one might encode the color of the original object at that point on the object. This information could then be used during rendering to recreate the appearance of the original object.

In other instances, color patterns can be associated with an object's surface by means of a process known as **texture mapping.** Texture mapping is similar to the process of applying wallpaper in that it associates a predefined image with the surface of an object. This image might be a digital photograph, an artist's rendering, or perhaps a computer-generated image. Traditional texture images include brick walls, wood grained surfaces, and marble facades.

Suppose, for example, that we wanted to model a stone wall. We could represent the shape of the wall with a simple polygonal mesh depicting a long rectangular solid. Then, with this mesh we could supply a two-dimensional image of stone masonry. Later, during the rendering process, this image could be applied to the rectangular solid to produce the appearance of a stone wall. More precisely, each time the rendering process needed to determine the appearance of a point on the wall it would simply use the appearance of the corresponding point in the masonry image.

Texture mapping works best when applied to relatively flat surfaces. The result can look artificial if the texture image must be distorted significantly to cover a curved surface (imagine the problems of trying to wallpaper a beach ball) or if the texture image wraps completely around an object causing a seam where the texture pattern may not blend with itself. Nonetheless, texture mapping has proven to be an efficient means of simulating texture and is widely used in situations that are real-time sensitive—a prime example being interactive video games.

The Search for Realism Building object models that lead to realistic images is a topic of ongoing research. Of special interest are materials associated with living characters such as skin, hair, fur, and feathers. Much of this research is specific to a particular substance and encompasses both modeling and rendering techniques. For instance, to obtain realistic models of human skin, some researchers have incorporated the degree to which light penetrates the epidermal and dermal skin layers and how the contents of those layers affect the skin's appearance.

Another example involves the modeling of human hair. If hair is to be seen from a distance, then more traditional modeling techniques may suffice. But, for close-up views, realistic-appearing hair can be challenging. Problems include translucent properties, textural depth, draping characteristics, and the manner in which hair responds to external forces such as wind. To overcome these hurtles, some applications have resorted to modeling individual strands of hair—a daunting task because the number of hairs on a human head can be on the order of 100,000. More astounding, however, is that some researchers have constructed hair models that address the scaled texture, color variation, and mechanical dynamics of each individual strand.

Still another example in which considerable detail has been pursued is in modeling cloth. In this case the intricacies of weaving patterns have been used to produce proper textural distinctions between fabric types such as twill versus satin, and detailed properties of yarn have been combined with knitting pattern data to create models of knit fabric that lead to extremely realistic close-up images. In addition, knowledge of physics and mechanical engineering has been applied to individual threads to compute images of draped material that account for such aspects as stretching of threads and shearing of the weave.

The production of realistic images is an active area of research that, as we have said, incorporates techniques in both the modeling and rendering processes. Typically, as progress is made, the new techniques are first incorporated in applications that are not subject to real-time constraints, such as the graphics software of motion picture production studios where there is a significant delay between the modeling/rendering process and the ultimate presentation of images. As these new techniques are refined and streamlined, they find their way into real-time applications, and the quality of the graphics in these environments improves as well. Truly realistic real-time interaction with virtual worlds may not be too far in the future.

Modeling Entire Scenes

Once the objects in a scene have been adequately described and digitally encoded, they are each assigned a location, size, and an orientation within the scene. This collection of information is then linked to form a data structure called a **scene graph.** In addition, the scene graph contains links to special objects representing light sources as well as a particular object representing the camera. It is here that the location, orientation, and focal properties of the camera are recorded.

Thus, a scene graph is analogous to a studio set-up in traditional photography. It contains the camera, lights, props, and background scenery—everything that will contribute to the appearance of the photograph when the shutter is snapped—all in their proper places. The difference is that a traditional photography setup contains physical objects whereas a scene graph contains digitally encoded representations of objects. In short, the scene graph describes a virtual world.

The positioning of the camera within a scene has many repercussions. As we mentioned earlier, the detail with which objects are modeled depends on the object's location in the scene. Foreground objects require more detail than background objects, and the distinction between foreground and background depends on the camera position. If the scene is to be used in a context analogous to a theatrical stage setting, then the roles of foreground and background are well established and the object models can be constructed accordingly. If, however, the context requires that the camera's position be altered for different images, the detail provided by the object models might need to be adjusted between "photographs." This is an area of current research. One envisions a scene consisting of "intelligent" models that refine their polygonal meshes and other features as the camera moves within the scene.

An interesting example of a moving camera scenario occurs in virtual reality systems with which a human user is allowed to experience the sensation of

3D Television

Several technologies exist to produce 3D imagery in the context of television, but all rely on the same stereoscopic visual effect—two slightly different images arriving at the left and right eyes are interpreted by the brain as depth. The most inexpensive mechanisms for this require special glasses with filter lenses. Older colored lenses (used in cinema in the 1950s) or the more modern polarized lenses filter out different aspects of a single image from the screen, resulting in different images reaching different eyes. More costly technology involves "active" glasses that alternately shutter left and right lenses in synchronization with a 3D television that switches quickly between the left and right images. Finally, 3D televisions are being developed that do not require special glasses or head gear. They use elaborate arrays of filters or magnifying lens on the surface of the screen to project the left and right images toward a viewer's head at slightly different angles, meaning that the left and right eyes of the viewer see different images.

moving around within an imaginary three-dimensional world. The imaginary world is represented by a scene graph, and the human views this environment by means of a camera that moves within the scene as dictated by the movements of the human. Actually, to provide the depth perception of three dimensions, two cameras are used: one representing the human's right eye, the other representing the human's left eye. By displaying the image obtained by each camera in front of the appropriate eye, the human gets the illusion of residing inside the three-dimensional scene, and when audio and tactile sensations are added to the experience, the illusion can become quite realistic.

In closing, we should note that the construction of a scene graph resides at a pivotal position in the 3D graphics process. Because it contains all the information required to produce the final image, its completion marks the end of the artistic modeling process and the beginning of the computationally intensive process of rendering the image. Indeed, once the scene graph is constructed, the graphics task becomes that of computing projections, determining surface details at specific points, and simulating the effects of light—a collection of tasks that is largely independent of the particular application.

Questions & Exercises

1. The following are four points (encoded using the traditional rectangular coordinate system) that represent the vertices of a planar patch. Describe the shape of the patch. (For those without a background in analytic geometry, each triple tells you how to reach the point in question beginning at the corner of a room. The first number tells you how far to walk along the seam between the floor and the wall on your right. The second number tells you how far to walk out into the room in a direction parallel to the wall on your left. The third number tells you how far to climb up from the floor. If a number is negative, you will have to pretend that you are a ghost and can walk backward through walls and floors.)

   (0, 0, 0)   (0, 1, 1)   (0, 2, 1)   (0, 1, 0)

2. What is a procedural model?
3. List some of the objects that might be present in a scene graph used to produce an image in a park.
4. Why are shapes represented by polygonal meshes even though they could be represented more precisely by geometric equations?
5. What is texture mapping?

10.4 Rendering

It is time now to consider the process of rendering, which involves determining how the objects in a scene graph would appear when projected onto the projection plane. There are several ways to accomplish the rendering task. This section focuses on the traditional approach that is used by most of the more popular graphics systems (video games, home computers, etc.) on the "consumer market" today. The following section investigates two alternatives to this approach.

We begin with some background information on the interaction between light and objects. After all, the appearance of an object is determined by the light emitted from that object, and thus determining an object's appearance ultimately becomes the task of simulating the behavior of light.

Light-Surface Interaction

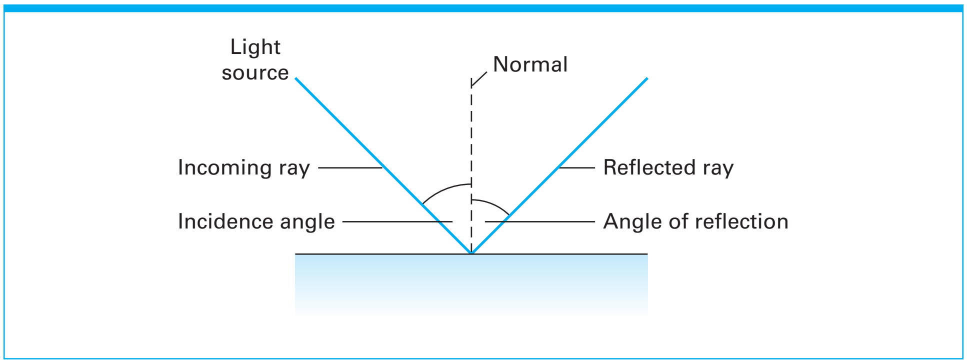
Depending on the material properties of an object, light striking its surface may be absorbed, bounce off the surface as reflected light, or pass through the surface (and be bent) as refracted light.

Reflection Let us consider a ray of light that is reflected off a flat opaque surface. The ray arrives traveling in a straight line and strikes the surface at an angle called the **incidence angle.** The angle at which the ray is reflected is identical to the incidence angle. As shown in Figure 10.5, these angles are measured relative to a line perpendicular (or **normal**) to the surface. (A line normal to a surface is often referred to as simply "the normal" as in "The incidence angle is measured relative to the normal.") The incoming ray, the reflected ray, and the normal all lie in the same plane.

If a surface is smooth, parallel light rays (such as those arriving from the same light source) that strike the surface in the same area will be reflected in essentially the same direction and travel away from the object as parallel rays. Such reflected light is called **specular light.** Note that specular light can be observed only when the orientation of the surface and the light source causes the light to be reflected in the viewer's direction. Thus, it normally appears as bright highlights on a surface. Moreover, because specular light has minimal contact with the surface, it tends to portray the color of the original light source.

Surfaces, however, are rarely perfectly smooth, and therefore many light rays may strike the surface at points whose orientations differ from that of the prevailing surface. Moreover, light rays often penetrate the immediate boundary of a surface and ricochet among the surface particles before finally departing as reflected light. The result is that many rays will be scattered in different directions. This scattered light is called **diffuse light.** Unlike specular light,

Figure 10.5 Reflected light

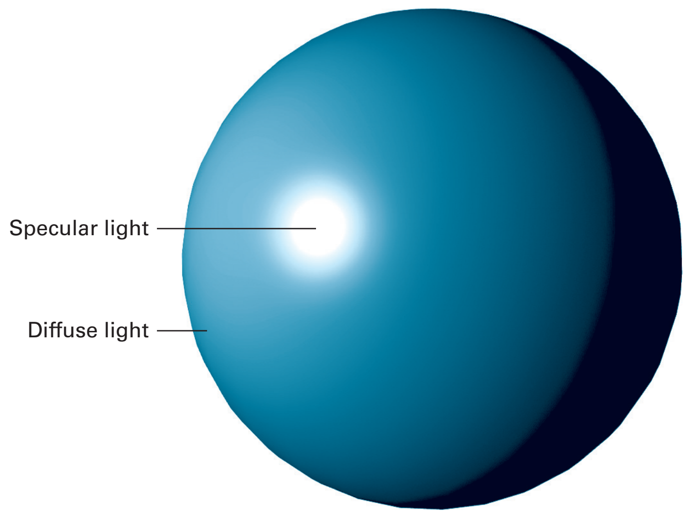


diffuse light is visible within a wide range of directions. And, because it tends to have prolonged contact with the surface, diffuse light is more susceptible to the absorption properties of the material and therefore tends to portray the color of the object.

Figure 10.6 presents a ball that is illuminated by a single light source. The bright highlight on the ball is produced by specular light. The rest of the hemisphere facing the light source is seen by means of diffuse light. Note that the hemisphere facing away from the primary light source is not visible by means of light being reflected directly from that source. The ability to see this portion of the ball is due to **ambient light,** which is "stray" or scattered light that is not associated with any particular source or direction. Portions of surfaces illuminated by ambient light often appear as a uniform dark color.

Most surfaces reflect both specular and diffuse light. The characteristics of the surface determine the proportion of each. Smooth surfaces appear shiny because they reflect more specular light than diffuse light. Rough surfaces appear dull because they reflect more diffuse light than specular light. Moreover, due to minute properties of some surfaces, the ratio of specular to diffuse light varies depending on the direction of the incoming light. Light striking such a surface from one direction may be reflected primarily as specular light, whereas light striking the surface from another direction may be reflected primarily as diffuse light. Thus, the appearance of the surface will shift from shiny to dull as it is rotated. Such surfaces are called **anisotropic surfaces,** as opposed to **isotropic surfaces** whose reflection patterns are symmetric. Examples of anisotropic surfaces are found in fabric such as satin, where the nap of the cloth alters the material's appearance depending on its orientation. Another example is the grassy surface of an athletic field, where the lie of the grass (usually determined by the manner in which the grass is cut) produces anisotropic visual effects such as light and dark striped patterns.

Figure 10.6 Specular versus diffuse light



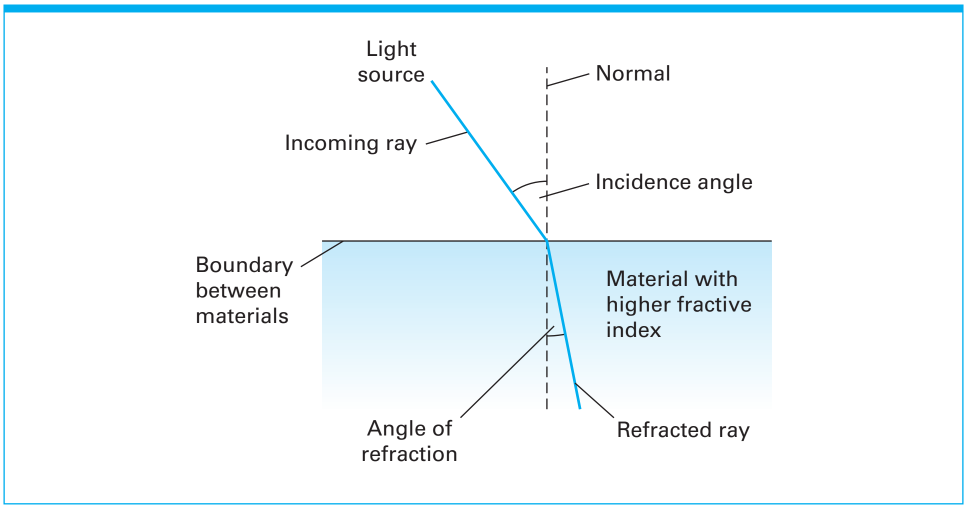
Refraction Now consider light striking an object that is transparent rather than opaque. In this case light rays pass through the object rather than bounce off its surface. As the rays penetrate the surface, their direction is altered—a phenomenon called **refraction** (Figure 10.7). The degree of refraction is determined by the refractive index of the materials involved. The refractive index is related to the density of the material. Dense materials tend to have a higher refractive index than less dense materials. As a light ray passes into a material with a higher reflective index (such as passing from air into water), it bends toward the normal at the point of entry. If it passes into a material with a lower refractive index, it bends away from the normal.

To render transparent objects properly, rendering software must know the refractive indexes of the materials involved. But this is not the whole story. The rendering software must also know which side of an object's surface represents the inside of an object as opposed to the outside. Is the light entering the object or exiting the object? Techniques for obtaining this information are sometimes quite subtle. For example, if we agree to always list the vertices of each polygon in a polygonal mesh in a counter-clockwise order as seen from outside the object, then the list cleverly encodes which side of the patch represents the outside of the object.

Clipping, Scan Conversion, and Hidden-Surface Removal

Let us now focus on the process of producing an image from a scene graph. Our approach for now is to follow the techniques used in most interactive video game systems. Collectively, these techniques form a well-established paradigm known as the **rendering pipeline.** We will consider some of the pros and cons of this approach at the end of this section and explore two alternatives in the following section. For now it is helpful to note that the rendering pipeline deals with opaque objects and thus refraction is not an issue. Moreover, it ignores interactions between objects so that, for now, we will not be concerned with mirrors and shadows.

Figure 10.7 Refracted light



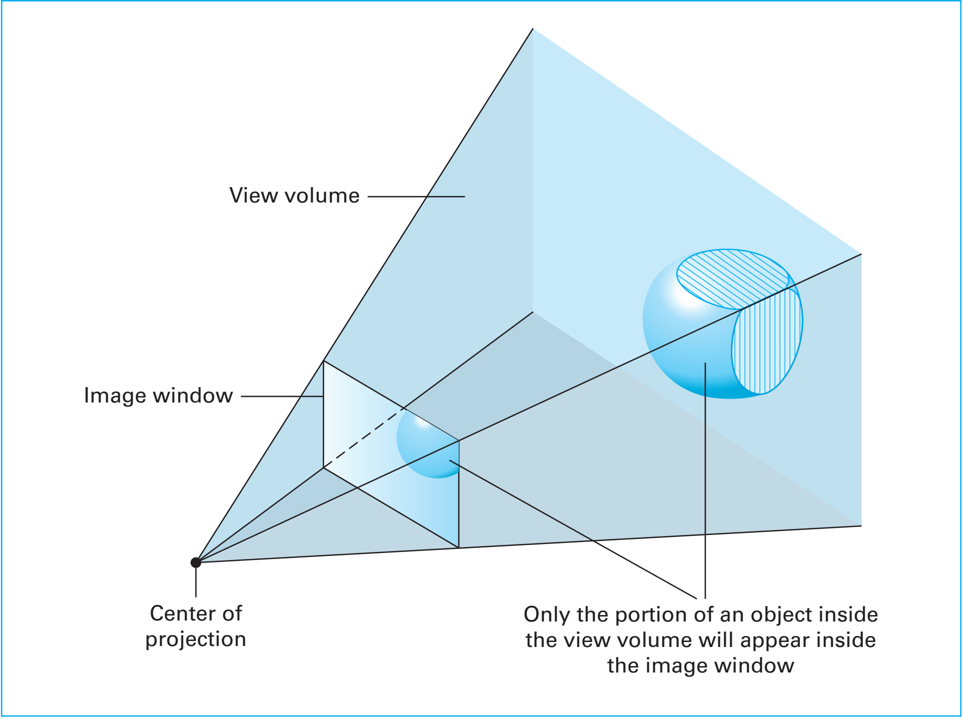
The rendering pipeline begins by identifying the region in a three-dimensional scene that contains the objects (or parts of objects) that can be "seen" by the camera. This region, called the **view volume,** is the space within the pyramid defined by straight lines extending from the center of projection through the corners of the image window (Figure 10.8).

Once the view volume is identified, the task is to discard from consideration those objects or parts of objects that do not intersect the view volume. After all, the projection of that portion of the scene will fall outside the image window and therefore not appear in the final image. The first step is to discard all those objects that are totally outside the view volume. To streamline this process, a scene graph may be organized in a tree structure in which objects in different regions of the scene are stored in different branches. In turn, large sections of the scene graph can be discarded merely by ignoring entire branches in the tree.

After identifying and discarding the objects that do not intersect the view volume, the remaining objects are trimmed by a process known as **clipping,** which essentially slices off the portion of each object that lies outside the view volume. More precisely, clipping is the process of comparing each individual planar patch to the boundaries of the view volume and trimming off those portions of the patch that fall outside. The results are polygonal meshes (of possibly trimmed planar patches) that lie entirely within the view volume.

The next step in the rendering pipeline is to identify the points on the remaining planar patches that are to be associated with pixel positions in the final image. It is important to realize that only these points will contribute to the final

Figure 10.8 Identifying the region of the scene that lies inside the view volume


View volume
Image window
Center of projection
Only the portion of an object inside the view volume will appear inside the image window

Aliasing

Have you ever noticed the weird "glittery" appearance that striped shirts and ties have on television screens? This is the result of the phenomenon called **aliasing,** which occurs when a pattern in the desired image meshes inappropriately with the density of the pixels comprising the image. As an example, suppose a portion of the desired image consists of alternating black and white stripes but the center of all the pixels happens to fall only on the black stripes. The object would then be rendered as being completely black. But, if the object moves slightly, the center of all the pixels may fall on the white stripes, meaning that the object would suddenly change to white. There are a variety of ways to compensate for this annoying effect. One is to render each pixel as the average of a small area in the image rather than as the appearance of a precise single point.

picture. If a detail on an object falls between the pixel positions, it will not be represented by a pixel and therefore not be visible in the final image. This is why pixel counts are heavily advertised in the digital camera market. The more pixels there are, the more likely that small details will be captured in a photograph.

The process of associating pixel positions with points in the scene is called **scan conversion** (because it involves converting the patches into horizontal rows of pixels called scan lines) or **rasterization** (because an array of pixels is known as a raster). Scan conversion is accomplished by extending straight lines (projectors) from the center of projection through each pixel position in the image window and then identifying the points at which these projectors intersect the planar patches. These, then, are the points on the patches at which we must determine an object's appearance. Indeed, these are the points that will be represented by the pixels in the final image.

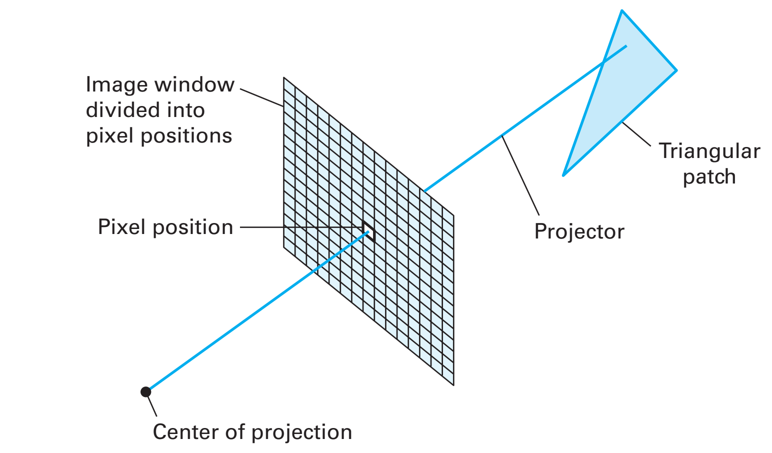
Figure 10.9 depicts the scan conversion of a single triangular patch. Part a of the figure shows how a projector is used to associate a pixel position with a point on the patch. Part b shows the pixel image of the patch as determined by the scan conversion. The entire array of pixels (the raster) is represented by a grid, and the pixels associated with the triangle have been shaded. Note that the figure also demonstrates the distortion that can occur when scan converting a shape whose features are small relative to the size of the pixels. Such jagged edges are familiar to users of most personal computer display screens.

Unfortunately, scan conversion of an entire scene (or even a single object) is not as straightforward as scan converting a single patch. This is because when multiple patches are involved, one patch may block the view of another. Thus, even though a projector intersects a planar patch, that point on the patch may not be visible in the final image. Identifying and discarding points in a scene that are blocked from view is the process called **hidden-surface removal.**

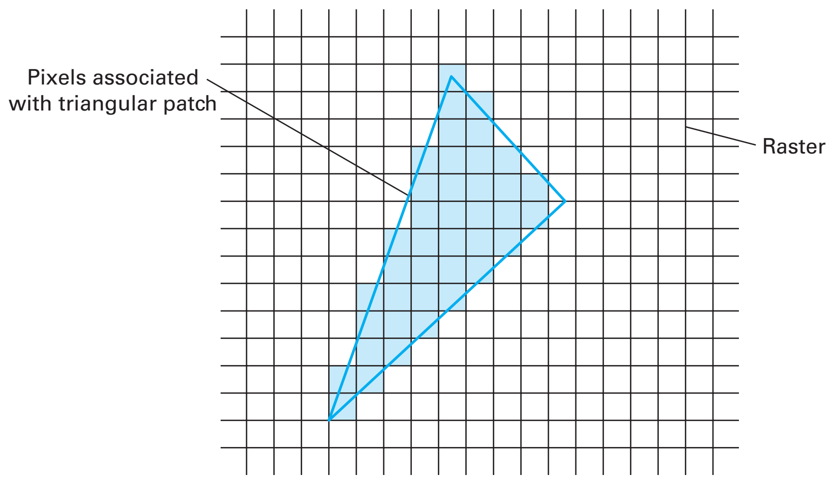
A specific version of hidden-surface removal is **back face elimination,** which involves discarding from consideration those patches in a polygonal mesh that represent the "back side" of an object. Note that back face elimination is relatively straightforward because the patches on the back side of an object can be identified as those whose orientation faces away from the camera.

Solving the complete hidden-surface removal problem, however, requires much more than back face elimination. Imagine, for example, a scene of an

Figure 10.9 The scan conversion of a triangular patch



a. The scan conversion process



b. Raster showing the "projected shape" of the triangular patch

automobile in front of a building. Planar patches from both the automobile and the building will project into the same area of the image window. Where overlaps occur, the pixel data ultimately stored in the frame buffer should indicate the appearance of the object in the foreground (the automobile) rather than the object in the background (the building). In short, if a projector intersects more than one planar patch, it is the point on the patch nearest the image window that should be rendered.

A simplistic approach to solving this "foreground/background" problem, known as the **painter's algorithm,** is to arrange the objects in the scene according to their distances from the camera and then to scan convert the more distant objects first, allowing the results of scan converting closer objects to override any previous results. Unfortunately, the painter's algorithm fails to handle cases in which objects are intertwined. Part of a tree may be behind another object while another part of the tree may be in front of that object.

More encompassing solutions to the "foreground/background" problem are obtained by focusing on individual pixels rather than entire objects. A popular technique uses an extra storage area, called a **z-buffer** (also a depth buffer), which contains an entry for each pixel in the image (or, equivalently, each pixel entry in the frame buffer). Each position in the z-buffer is used to store the distance along the appropriate projector from the camera to the object currently represented by the corresponding entry in the frame buffer. Thus, with the aid of a z-buffer, the "foreground/background" problem can be resolved by computing and storing the appearance of a pixel only if data for that pixel has not yet been placed in the frame buffer or if the point on the object currently being considered is closer than that of the previously rendered object as determined by the distance information recorded in the z-buffer.

More precisely, when using a z-buffer, the rendering process can proceed as follows: Set all entries in the z-buffer to a value representing the maximum distance from the camera that an object will be considered for rendering. Then, each time a new point on a planar patch is considered for rendering, first compare its distance from the camera to the value in the z-buffer associated with the current pixel position. If that distance is less than the value found in the z-buffer, compute the appearance of the point, record the results in the frame buffer, and replace the old entry in the z-buffer with the distance to the point just rendered. (Note that if the distance to the point is greater than the value found in the z-buffer, no action needs to be taken because the point on the patch is too far away to be considered or it is blocked from view by a closer point that has already been rendered.)

Shading

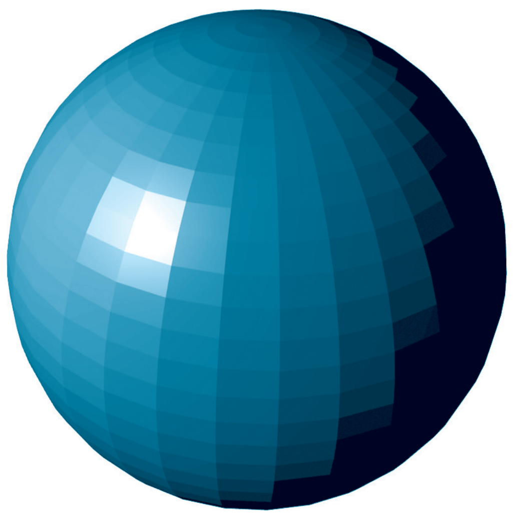
Once scan conversion has identified a point on a planar patch that is to appear in the final image, the rendering task becomes that of determining the appearance of the patch at that point. This process is called **shading.** Note that shading involves computing the characteristics of the light projected toward the camera from the point in question, which, in turn, depends on the orientation of the surface at that point. After all, it is the orientation of the surface at the point that determines the degree of specular, diffuse, and ambient light witnessed by the camera.

A straightforward solution to the shading problem, called **flat shading,** is to use the orientation of a planar patch as the orientation of each point on the patch—that is, to assume the surface over each patch is flat. The result, however, is that the final image will appear faceted as depicted in Figure 10.10 rather than rounded as shown in Figure 10.6. In a sense, flat shading produces an image of the polygonal mesh itself rather than the object being modeled by the mesh.

To produce a more realistic image, the rendering process must blend the appearance of individual planar patches into a smoothly curved appearing surface. This is accomplished by estimating the true orientation of the original surface at each individual point being rendered.

Such estimation schemes normally begin with data indicating the surface orientation at the vertices of the polygonal mesh. There are several ways to obtain this data. One is to encode the orientation of the original surface at each vertex and attach this data to the polygonal mesh as part of the modeling process. This produces a polygonal mesh with arrows, called **normal vectors,** attached to each

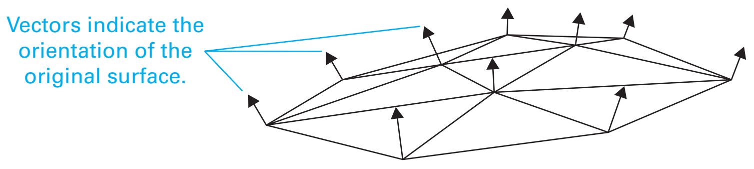
Figure 10.10 A sphere as it might appear when rendered by flat shading



vertex. Each normal vector points outward in the direction perpendicular to the original surface. The result is a polygonal mesh that can be envisioned as shown in Figure 10.11. (Another approach is to compute the orientation of each patch adjacent to a vertex and then use an "average" of those orientations as an estimate of the surface's orientation at the vertex.)

Regardless of how the orientation of the original surface at a polygonal mesh's vertices is obtained, several strategies are available for shading a planar patch based on this data. These include **Gouraud shading** and **Phong shading,** the distinction between which is subtle. Both begin by using the information about the surface orientation at a patch's vertices to approximate the surface orientation along the boundaries of the patch. Gouraud shading then applies that information to determine the appearance of the surface along the patch boundaries and, finally, interpolates that boundary appearance to estimate the appearance of the surface at points in the interior of the patch. In contrast, Phong shading interpolates the orientation of the surface along the patch's boundaries to estimate the surface orientation at points within the patch's interior and only then considers questions of appearance. (In short, Gouraud shading converts orientation information into color information and then interpolates the color information. Phong

Figure 10.11 A conceptual view of a polygonal mesh with normal vectors at its vertices



shading interpolates orientation information until the orientation of the point in question is estimated, and then converts that orientation information into color information.) The result is that Phong shading is more likely to detect specular light within a patch's interior because it is more responsive to changes in surface orientation. (See question 3 at the end of this section.)

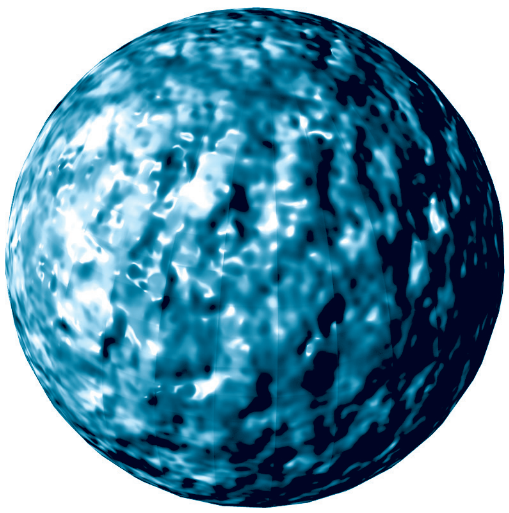
Finally, we should note that basic shading techniques can be augmented to add the appearance of texture to a surface. An example, called **bump mapping,** is essentially a way of generating small variations in the apparent orientation of a surface so that the surface appears to be rough. More precisely, bump mapping adds a degree of randomness to the interpolation process applied by traditional shading algorithms so that the overall surface appears to have texture, as demonstrated in Figure 10.12.

Rendering-Pipeline Hardware

As we have already said, the processes of clipping, scan conversion, hidden- surface removal, and shading are viewed collectively as a sequence known as the rendering pipeline. Moreover, efficient algorithms for performing these tasks are well-known and have been implemented directly in electronic circuitry, which has been miniaturized by VLSI technology to produce chips that perform the entire rendering pipeline automatically. Today, even inexpensive examples are capable of rendering millions of planar patches per second.

Most computer systems that are designed for graphics applications, including video game machines, incorporate these devices in their design. In the case of more general-purpose computer systems, this technology can be added in the form of a **graphics card,** or **graphics adapter,** which is attached to the computer's bus as a specialized controller (see Chapter 2). Such hardware substantially reduces the time required to perform the rendering process.

Figure 10.12 A sphere as it might appear when rendered using bump mapping



Rendering-pipeline hardware also reduces the complexity of graphics application software. Essentially, all the software needs to do is provide the graphics hardware with the scene graph. The hardware then performs the pipeline steps and places the results in the frame buffer. Thus, from the software's perspective, the entire rendering pipeline is reduced to a single step using the hardware as an abstract tool.

As an example, let us again consider an interactive video game. To initialize the game, the game software transfers the scene graph to the graphics hardware. The hardware then renders the scene and places the image in the frame buffer from where it is automatically displayed on the monitor screen. As the game progresses, the game software merely updates the scene graph in the graphics hardware to reflect the changing game situation, and the hardware repeatedly renders the scene, each time placing the updated image in the frame buffer.

We should note, however, that the capabilities and communication properties of different graphics hardware vary substantially. Thus, if an application such as a video game were developed for a specific graphics platform, it would have to be modified if transferred to another environment. To reduce this dependence on the specifics of graphics systems, standard software interfaces have been developed to play an intermediary role between graphics hardware and application software. These interfaces consist of software routines that convert standardized commands into the specific instructions required by a particular graphics hardware system. Examples include **OpenGL** (short for **Open Graphics Library**), which is a nonproprietary system developed by Silicon Graphics and widely used in the video game industry, and **Direct3D,** which was developed by Microsoft for use in Microsoft Windows environments.

In closing we should note that, with all the advantages associated with the rendering pipeline, there are disadvantages as well—the most significant of which is the fact that the pipeline implements only a **local lighting model,** meaning that the pipeline renders each object independently of other objects. That is, under a local lighting model, each object is rendered in relation to the light sources as though it were the only object in the scene. The result is that light interactions between objects, such as shadows and reflections, are not captured. This is in contrast to a **global lighting model** in which interactions among objects are considered. We will discuss two techniques for implementing a global lighting model in the next section. For now we merely note that these techniques lie outside the real-time capabilities of current technology.

This does not mean, however, that systems using rendering-pipeline hardware are not able to produce some global lighting effects. Indeed, clever techniques have been developed to overcome some of the restrictions imposed by a local lighting model. In particular, the appearance of **drop shadows,** which are shadows cast on the ground, can be simulated within the context of a local lighting model by making a copy of the polygonal mesh of the object casting the shadow, squashing the duplicate mesh flat, placing it on the ground, and coloring it dark. In other words, the shadow is modeled as though it were another object, which can then be rendered by traditional rendering-pipeline hardware to produce the illusion of a shadow. Such techniques are popular in both "consumer level" applications such as interactive video games and "professional level" applications such as flight simulators.

Questions & Exercises

1. Summarize the distinction between specular light, diffuse light, and ambient light.
2. Define the terms *clipping* and *scan conversion*.
3. Gouraud shading and Phong shading can be summarized as follows: Gouraud shading uses the orientation of an object's surface along the boundaries of a patch to determine the appearance of the surface along the boundaries and then interpolates these appearances over the interior of the patch to determine the appearance of the particular points in question. Phong shading interpolates the boundary orientations to compute the orientations of points interior to the patch and then uses those orientations to determine the appearance of the particular points in question. How would the appearance of an object possibly differ?
4. What is the significance of the rendering pipeline?
5. Describe how reflections in a mirror might be simulated using a local lighting model.

10.5 Dealing with Global Lighting

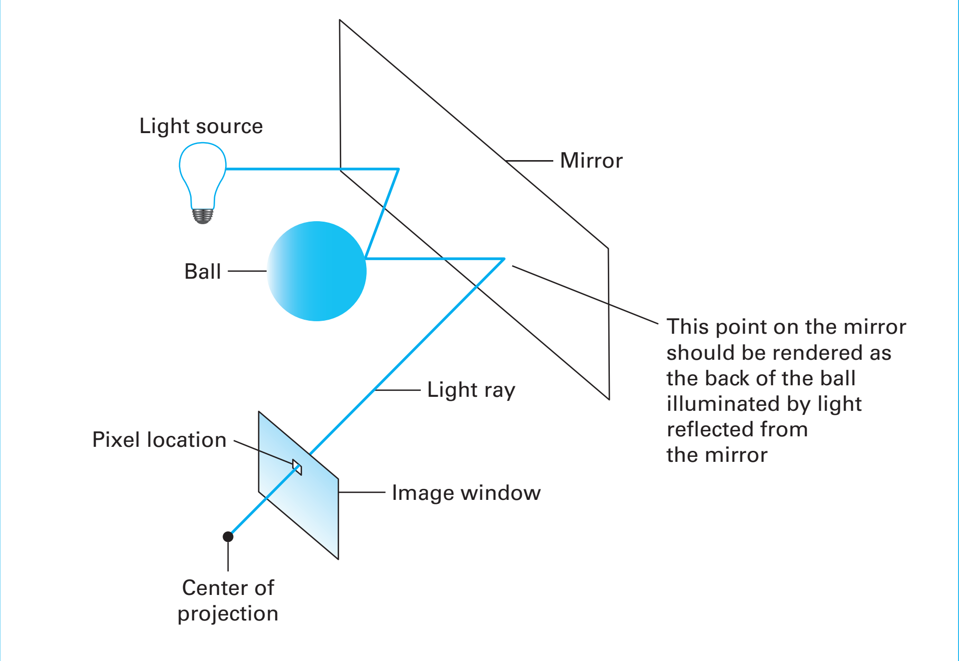
Researchers are currently investigating two alternatives to the rendering pipeline, both of which implement a global lighting model and thus provide the potential of overcoming the local lighting model restrictions inherent in the traditional pipeline. One of these alternatives is ray tracing, the other is radiosity. Both are meticulous, time-consuming processes, as we are about to see.

Ray Tracing

Ray tracing is essentially the process of following a ray of light backward to find its source. The process starts by selecting a pixel to be rendered, identifying the straight line passing through that pixel and the center of projection, and then tracing the light ray that strikes the image window along that line. This tracing process involves following the line into the scene until it contacts an object. If that object is a light source, the ray tracing process terminates and the pixel is rendered as a point on the light source. Otherwise, the properties of the object's surface are evaluated to determine the direction of the incoming ray of light that was reflected to produce the ray the process is backtracking. The process then follows that incoming ray backward to find its source, at which point yet another ray may be identified and traced.

An example of ray tracing is depicted in Figure 10.13 where we see a ray traced backward through the image window to the surface of a mirror. From there the ray is traced to a shiny ball, from there back to the mirror, and from the mirror to the light source. Based on the information gained from this tracing process, the pixel in the image should appear as a point on the ball illuminated by the light source reflected in the mirror.

Figure 10.13 Ray tracing


Light source
Mirror
Ball
This point on the mirror should be rendered as the back of the ball illuminated by light reflected from the mirror
Light ray
Pixel location
Image window
Center of projection

One drawback to ray tracing is that it traces only specular reflections. Thus, all the objects rendered by this method tend to have a shiny appearance. To counter this effect, a variation of ray tracing, called **distributed ray tracing,** can be applied. The difference is that rather than tracing a single ray backward from a point of reflection, distributed ray tracing traces multiple rays from that point, each extending in a slightly different direction.

Another variation to basic ray tracing is applicable when transparent objects are involved. In this case, two effects must be considered each time a ray is traced back to a surface. One is reflection, the other is refraction. In this case, the task of tracing the original ray splits into two tasks: tracing the reflection backwards and tracing the refraction backwards.

Ray tracing is normally implemented recursively, with each activation being asked to trace a ray to its source. The first activation may trace its ray to a shiny opaque surface. At that point it would recognize that its ray is the reflection of an incoming ray, compute the direction of that incoming ray, and call another activation to trace that incoming ray. This second activation would perform a similar task to find the source of its ray—a process that may result in calling still other activations.

A variety of conditions can be used to terminate recursive ray tracing. The ray being traced may reach a light source, the ray being traced may exit the scene without striking an object, or the number of activations may reach a predetermined limit. Still another termination condition can be based on the absorption properties of the surfaces encountered. If a surface is highly absorptive, such as a dark matte surface, then any incoming ray will have little effect on the surface's appearance and ray tracing can cease. Accumulative absorption can have a similar effect. That is, ray tracing can terminate after visiting several moderately absorptive surfaces.

Being based on a global lighting model, ray tracing avoids many of the restrictions inherent in the traditional rendering pipeline. For example, the problems of hidden-surface removal and detecting shadows are naturally solved in the ray tracing process. Unfortunately, ray tracing has the major drawback of being time consuming. As each reflection is traced back to its source, the number of computations required grows immensely—a problem that is compounded when allowing for refractions or applying distributed ray tracing. Hence, ray tracing is not implemented in consumer-level real-time systems such as interactive video games but instead may be found in professional-level applications that are not real-time sensitive such as the graphics software used by motion picture studios.

Radiosity

Another alternative to the traditional rendering pipeline is **radiosity.** Whereas ray tracing takes a point-by-point approach by tracing individual rays of light, radiosity takes a more regional approach by considering the total light energy radiated between pairs of planar patches. This radiated light energy is essentially diffuse light. The light energy that is radiated from an object is either generated by the object (as in the case of a light source) or reflected off the object. The appearance of each object is then determined by considering the light energy it receives from other objects.

The degree to which light radiated from one object affects the appearance of another is determined by parameters called **form factors.** A unique form factor is associated with each pair of patches in the scene to be rendered. These form factors take into account the geometric relationships between the patches involved such as separation distance and relative orientations. To determine the appearance of a patch in the scene, the amount of light energy received from all the other patches in the scene is computed using the appropriate form factor for each computation. The results are combined to produce a single color and intensity for each patch. These values are then interpolated among adjacent patches using techniques similar to Gouraud shading to obtain a smoothly contoured appearing surface rather than a faceted one.

Because there are numerous patches to be considered, radiosity is very computationally intense. In turn, like ray tracing, its application falls outside the capabilities of real-time graphics systems currently available in the consumer market. Another problem with radiosity is that because it deals with units consisting of entire patches rather than individual points, it fails to capture the details of specular light, meaning that all the surfaces rendered by radiosity tend to have a dull appearance.

Radiosity does, however, have its merits. One is that determining the appearance of objects using radiosity is independent of the camera. Thus, once the radiosity computations for a scene have been performed, the rendering of the scene can be completed quickly for various camera positions. Another is that radiosity captures many of the subtle characteristics of light such as **color bleeding** where the color of one object affects the hue of other objects around it. In turn, radiosity has its niches. One is in graphics software used in architectural design settings. Indeed, the light within a proposed building consists mainly of diffuse and ambient light so that specular effects are not significant, and the fact that new camera positions can be handled efficiently means that an architect can quickly view different rooms from different perspectives.

Questions & Exercises

1. Why does ray tracing follow rays backward from the image window to the light source rather than forward from the light source to the image window?
2. What is the difference between straightforward ray tracing and distributed ray tracing?
3. What are two disadvantages of radiosity?
4. In what ways are ray tracing and radiosity similar? In what ways are they different?

10.6 Animation

We turn now to the subject of computer animation, which is the use of computer technology to produce and display images that exhibit motion.

Animation Basics

We begin by introducing some basic animation concepts.

Frames Animation is achieved by displaying a sequence of images, called **frames,** in rapid succession. These frames capture the appearance of a changing scene at regular time intervals, and thus their sequential presentation creates the illusion of observing the scene continuously over time. The standard rate of display in the motion picture industry is twenty-four frames per second. The standard in broadcast video is sixty frames per second (although because every other video frame is designed to be interwoven with the preceding frame to produce a complete detailed image, video can also be classified as a thirty-frames-per- second system).

Kineographs

A kineograph is a book of animation frames that simulate motion when the pages are flipped rapidly. You can make your own kineograph out of this text (assuming that you have not already filled the margins with doodles). Simply place a dot in the margin of the first page and then, based on the position of the impression made on the third page, place another dot on the third page in a slightly different position than the dot on the first. Repeat this process on each consecutive odd page until you reach the end of the book. Now flip through the pages and watch the dot dance around. Presto! You have made your own kineograph and perhaps taken the first step toward a career in animation! As an experiment in kinematics, try drawing a stick figure instead of a simple dot and make the stick figure appear to walk. Now, experiment with dynamics by producing the image of a drop of water hitting the ground.

Frames can be produced by traditional photography or generated artificially by means of computer graphics. Moreover, the two techniques can be combined. For example, 2D graphics software is often used to modify images obtained via photography to remove the appearance of support wires, to superimpose images, and to create the illusion of **morphing,** which is the process of one object appearing to change into another.

A closer look at morphing provides interesting insights into the animation process. Constructing a morphing effect begins by identifying a pair of key frames that bracket the morphing sequence. One is the last image before the morph is to occur; the other is the first image after the morph has occurred. (In traditional motion picture production, this requires "filming" two sequences of action: one leading up to the occurrence of the morph, the other proceeding after the morph.) Features such as points and lines, called **control points,** in the frame preceding the morph are then associated with similar features in the postmorph frame, and the morph is constructed by applying mathematical techniques that incrementally distort one image into the other while using the control points as guides. By recording the images produced during this distortion process, a short sequence of artificially generated images is obtained that fills the gap between the original key frames and creates the morphing illusion.

The Storyboard A typical animation project begins with the creation of a **storyboard,** which is a sequence of two-dimensional images that tell the complete story in the form of sketches of scenes at key points in the presentation. The ultimate role of the storyboard depends on whether the animation project is implemented using 2D or 3D techniques. In a project using 2D graphics, the storyboard typically evolves into the final set of frames in much the same way that it did back in the Disney studios of the 1920s. In those days, artists, called master animators, would refine the storyboard into detailed frames, called **key frames,** that established the appearance of the characters and scenery at regular intervals of the animation. Assistant animators would then draw additional frames that would fill the gaps between the key frames so that the animation would appear continuous and smooth. This fill-in-the-gap process was called **in-betweening.**

The major distinction between this process and that used today is that animators now use image processing and 2D graphics software to draw key frames

Blurring

In the field of traditional photography, much effort has been expended toward the goal of producing sharp images of fast-moving objects. In the field of animation, the opposite problem arises. If each frame in a sequence depicting a moving object portrays the object as a sharp image, then the motion may appear jerky. However, sharp images are a natural byproduct of creating frames as individual images of stationary objects in a scene graph. Thus, animators often artificially distort the image of a moving object in a computer-generated frame. One technique, called **supersampling,** is to produce multiple images in which the moving object is only slightly displaced and then to overlay these images to produce a single frame. Another technique is to alter the shape of the moving object so that it appears elongated along the direction of motion.

and much of the in-betweening process has been automated so that the role of assistant animators has essentially disappeared.

3D Animation Most video game animation and full-feature animated productions are now created using 3D graphics. In these cases the project still begins with the creation of a storyboard consisting of two-dimensional images. However, rather than evolving into the final product as with 2D graphics projects, the storyboard is used as a guide in the construction of a three-dimensional virtual world. This virtual world is then repeatedly "photographed" as the objects within it are moved according to the script or the progression of the video game.

Perhaps we should pause here and clarify what it means for an object to move within a computer-generated scene. Keep in mind that the "object" is actually a collection of data stored in the scene graph. Among the data in this collection are values that indicate the location and orientation of the object. Thus, "moving" an object is accomplished merely by changing these values. Once these changes have been made, the new values will be used in the rendering process. Consequently, the object will appear to have moved in the final two-dimensional image.

Dynamics and Kinematics

The degree to which motion within a 3D graphics scene is automated or controlled by a human animator varies among applications. The goal, of course, is to automate the entire process. To this end, much research has been directed toward finding ways to identify and simulate the motion of naturally occurring phenomena. Two areas of mechanics have proven particularly useful in this regard.

One is **dynamics,** which deals with describing the motion of an object by applying the laws of physics to determine the effects of the forces acting on the object. For example, in addition to a location, an object in a scene might be assigned a direction of motion, speed, and mass. These items could then be used to determine the effects that gravity or collisions with other objects would have on the object, which would allow software to calculate the proper location of the object for the next frame.

As an example, consider the task of constructing an animated sequence depicting water sloshing in a container. We could use a particle system in the scene graph to represent the water, where each particle represents a small unit of water. (Think of the water consisting of large "molecules" the size of marbles.) Then, we could apply the laws of physics to compute the effects of gravity on the particles as well as the interaction among the particles themselves as the container rocks side to side. This would allow us to compute the location of each particle at regular time intervals, and by using the location of the outer particles as vertices of a polygonal mesh, we could obtain a mesh depicting the surface of the water. Our animation could then be obtained by repeatedly "photographing" this mesh as the simulation progresses.

The other branch of mechanics used to simulate motion is **kinematics,** which deals with describing an object's motion in terms of how parts of an object move with respect to each other. Applications of kinematics are prominent when animating articulated figures, where it is necessary to move appendages such as arms and legs. These motions are more easily modeled by simulating joint movement patterns than by computing the effects of the individual forces exerted by muscles and gravity. Thus, whereas dynamics might be the technique of choice when determining the path of a bouncing ball, the motion of an animated character's arm would be determined by applying kinematics to compute the proper shoulder, elbow, and wrist rotations. In turn, much research in animating living characters focuses on issues of anatomy and how joint and appendage structure influence motion.

A typical method of applying kinematics is to begin by representing a character by a stick figure that simulates the skeletal structure of the character being portrayed. Each section of the figure is then covered with a polygonal mesh representing the character's surface surrounding that section, and rules are established to determine how adjacent meshes should connect with each other. The figure can then be manipulated (either by software or a human animator) by repositioning the joints in the skeletal structure in the same manner that one manipulates a string puppet. The points at which the "strings" attach to the model are called **avars,** short for "articulation variables" (or more recently "animation variables").

Much research in the application of kinematics has been directed toward developing algorithms for automatically computing sequences of appendage positions that mimic natural occurring motion. Along these lines, algorithms are now available that generate realistic walking sequences.

However, much of the animation based on kinematics is still produced by directing a character through a preset sequence of joint-appendage positions. These positions may be established by the creativity of an animator or obtained by **motion capture,** which involves recording the positions of a living model as the model performs the desired action. More precisely, after applying reflective tape to strategic points on a human's body, the human can be photographed from multiple angles while throwing a baseball. Then, by observing the locations of the tape in the various photographs, the precise orientations of the human's arms and legs can be identified as the throwing action progresses, and these orientations can then be transferred to a character in an animation.

The Animation Process

The ultimate goal of research in animation is to automate the entire animation process. One imagines software that, given the proper parameters, would automatically produce the desired animated sequence. Progress in this direction is demonstrated by the fact that the motion picture industry now produces images of crowds, battle scenes, and stampeding animals by means of individual virtual "robots" that automatically move within a scene graph, each performing its allotted task.

An interesting case in point occurred when filming the fantasy armies of Orcs and humans for the *Lord of the Rings* trilogy. Each onscreen warrior was modeled as a distinct "intelligent" object with its own physical characteristics and a randomly assigned personality that gave it tendencies to attack or flee. In test simulations for the battle of Helms Deep in the second film, the Orcs had their tendency to flee set too high, and they simply ran away when confronted with the human warriors. (This was perhaps the first case of virtual extras considering a job too dangerous.)

Of course, much animation today is still created by human animators. However, rather than drawing two-dimensional frames by hand as in the 1920s, these animators use software to manipulate three-dimensional virtual objects in a scene graph in a manner reminiscent of controlling string puppets, as explained in our earlier discussion of kinematics. In this way, an animator is able to create a series of virtual scenes that are "photographed" to produce the animation. In some cases this technique is used to produce only the scenes for key frames, and then software is used to produce the in-between frames by automatically rendering the scene as the software applies dynamics and kinematics to move the objects in the scene graph from one key-frame scene position to the next.

As research in computer graphics progresses and technology continues to improve, more of the animation process will certainly become automated. Whether the role of human animators as well as human actors and physical sets will someday become obsolete remains to be seen, but many believe that day is not too far in the future. Indeed, 3D graphics has the potential of affecting the motion picture industry far more than the transition from silent movies to "talkies."

Questions & Exercises

1. Images seen by a human tend to linger in the human's perception for approximately 200 milliseconds. Based on this approximation, how many images per second must be presented to the human to create animation? How does this approximation compare to the number of frames per second used in the motion picture industry?
2. What is a storyboard?
3. What is in-betweening?
4. Define the terms *kinematics* and *dynamics.*

Chapter Review Problems

1. Which of the following are applications of 2D graphics and which are applications of 3D graphics?
   a. Designing the layout of magazine pages
   b. Drawing an image using Microsoft Paint
   c. Producing images from a virtual world for a video game
2. In the context of 3D graphics, what corresponds to each of the following items from traditional photography? Explain your answers.
   a. Film
   b. Rectangle in viewfinder
   c. Scene being photographed
3. When using a perspective projection, under what conditions will a sphere in the scene not produce a circle on the projection plane?
4. When using a perspective projection, can the image of a straight line segment ever be a curved line segment on the projection plane? Justify your answer.
5. Suppose one end of an eight-foot straight pole is four feet from the center of projection. Moreover, suppose that a straight line from the center of projection to one end of the pole intersects the projection plane at a point that is one foot from the center of projection. If the pole is parallel to the projection plane, how long is the image of the pole in the projection plane?
6. Explain the concepts of kinematics and dynamics as used in 3D animation.
7. How is blurring in animation different from image blurring?

8. What is a significant difference between applying 3D graphics to produce a motion picture and applying 3D graphics to produce the animation for an interactive video game? Explain your answer.

9. Identify some properties of an object that might be incorporated in a model of that object for use in a 3D graphics scene. Identify some properties that would probably not be represented in the model. Explain your answer.

10. Identify some physical properties of an object that are not captured by a model containing only a polygonal mesh. (Thus, a polygonal mesh alone does not constitute a complete model of the object.) Explain how one of those properties could be added to the object's model.

11. A triangle is displayed with approximate shading in Figure 10.9(b). Suggest a way to improve the accuracy of the shape.

12. Each collection that follows represents the vertices (using the traditional rectangular coordinate system) of a patch in a polygonal mesh. Describe the shape of the mesh.

```
Patch 1: (0, 0, 0) (0, 2, 0)
         (2, 2, 0) (2, 0, 0)
Patch 2: (0, 0, 0) (1, 1, 1)
         (2, 0, 0)
Patch 3: (2, 0, 0) (1, 1, 1)
         (2, 2, 0)
Patch 4: (2, 2, 0) (1, 1, 1)
         (0, 2, 0)
Patch 5: (0, 2, 0) (1, 1, 1)
         (0, 0, 0)
```

13. Each collection that follows represents the vertices (using the traditional rectangular coordinate system) of a patch in a polygonal mesh. Describe the shape of the mesh.

```
Patch 1: (0, 0, 0) (0, 4, 0)
         (2, 4, 0) (2, 0, 0)
Patch 2: (0, 0, 0) (0, 4, 0)
         (1, 4, 1) (1, 0, 1)
Patch 3: (2, 0, 0) (1, 0, 1)
         (1, 4, 1) (2, 4, 0)
Patch 4: (0, 0, 0) (1, 0, 1)
         (2, 0, 0)
Patch 5: (2, 4, 0) (1, 4, 1)
         (0, 4, 0)
```

14. Design a polygonal mesh representing a rectangular solid. Use the traditional rectangular coordinate system to encode the vertices and draw a sketch representing your solution.

15. Using no more than eight triangular patches, design a polygonal mesh to approximate the shape of a sphere with radius one. (With only eight patches, your mesh will be a very rough approximation of a sphere, but the goal is for you to display an understanding of what a polygonal mesh is rather than to produce a precise representation of a sphere.) Represent the vertices of your patches using the traditional rectangular coordinate system and draw a sketch of your mesh.

16. Why would the following four points not be the vertices of a planar patch?

```
(0, 0, 0) (1, 0, 0)
(0, 1, 0) (0, 0, 1)
```

17. Suppose the points (1, 0, 0), (1, 1, 1), and (1, 0, 2) are the vertices of a planar patch. Which of the following line segments is/are normal to the surface of the patch?
   a. The line segment from (1, 0, 0) to (1, 1, 0)
   b. The line segment from (1, 1, 1) to (2, 1, 1)
   c. The line segment from (1, 0, 2) to (0, 0, 2)
   d. The line segment from (1, 0, 0) to (1, 1, 1)

18. What is meant by flat shading?

19. What procedures can be adopted to obtain polygonal meshes for the following cases?
   a. A mob at a rock concert
   b. A building
   c. A sphere
   d. A painted canvas

20. Explain the following terms with the help of a labelled diagram:
   a. Projection plane
   b. Center of projection
   c. View volume
   d. Image window

21. Explain the terms hidden-surface removal, back face elimination, and z-buffer in brief.

22. What is meant by aliasing? Suggest one of the possible ways to overcome the problem of aliasing.

23. Suppose the surface of the planar patch with vertices (0, 0, 0), (0, 2, 0), (2, 2, 0), and (2, 0, 0) is smooth and shiny. If a light ray originates at the point (0, 0, 1) and strikes the surface at (1, 1, 0), through which of the following points will the reflected ray pass?
a. (0, 0, 1)
b. (1, 1, 1)
c. (2, 2, 1)
d. (3, 3, 1)

24. Suppose a buoy supports a light ten feet above the surface of still water. At what point on the water's surface will an observer see the reflection of the light if the observer is fifteen feet from the buoy and five feet above the water's surface?

25. If a fish is swimming below the surface of still water and an observer is viewing the fish from above the water, where will the fish appear to be from the observer's position?
a. Above and toward the background of its true position
b. At the true position
c. Below and toward the foreground of its true position

26. Suppose the points (1, 0, 0), (1, 1, 1), and (1, 0, 2) are the vertices of a planar patch and the vertices are listed in a counter-clockwise order as seen from outside the object. In each case that follows, indicate whether a ray of light originating at the given point would strike the surface of the patch from outside or inside the object.
a. (0, 0, 0)
b. (2, 0, 0)
c. (2, 1, 1)
d. (3, 2, 1)

27. In what way is the process of hidden-surface removal more complex than back face elimination? Explain your answer.

28. How is movement produced on screen in animation?

29. In our discussion of hidden-surface removal, we described the procedure for solving the "foreground/background" problem with the aid of a z-buffer. Express that procedure using the pseudocode introduced in Chapter 5.

30. Suppose the surface of an object is covered by alternating orange and blue vertical stripes, each of which is one centimeter wide. If the object is positioned in a scene so that the pixel positions are associated with points on the object spaced at two-centimeter intervals, what would be the possible appearances of the object in the final image? Explain your answer.

31. Although texture mapping and bump mapping are means of associating "texture" with a surface, they are considerably different techniques. Write a short paragraph comparing the two.

32. Give an example scenario where the painter's algorithm fails to solve the foreground/background problem.

33. You have two images of resolution 1024 × 1024 and 640 × 480. Which image would be more detailed?

34. In what way does the hardware in a computer designed for interactive video games differ from that of a general-purpose PC?

35. What are the various advantages and disadvantages of radiosity?

36. What is the distinction between anisotropic surfaces and isotropic surfaces?

37. What advantage does ray tracing have over the traditional rendering pipeline? What disadvantage does it have?

38. What advantage does distributed ray tracing have over traditional ray tracing? What disadvantage does it have?

39. What is the significance of graphics software interfaces in graphics processing? Explain your answer.

40. Write a pseudocode to draw a triangle with vertices at (0, 0), (2, 2) and (2, 0). Use the brute force approach to choose every pixel to be printed on screen.

41. How many frames would be required to produce a ninety-minute animated production to be shown in a motion picture theater?

42. What is the distinction between Bezier curves and Bezier surfaces? Explain your answer.

43. Explain how the use of a z-buffer could assist when creating an animation sequence depicting a single object moving within a scene.

44. Dynamics and kinematics have proven to be useful in simulating motion. Give a brief distinction between the two.

Social Issues

The following questions are intended as a guide to the ethical/social/legal issues associated with the field of computing. The goal is not merely to answer these questions. You should also consider why you answered as you did and whether your justifications are consistent from one question to the next.

1. Suppose computer animation reaches the point that real actors are no longer needed in the motion picture and television industries. What would the consequences be? What would be the ripple effect of no longer having "movie stars"?
2. With the development of digital cameras and related software, the ability to alter or fabricate photographs has been placed within the capabilities of the general public. What changes will this bring to society? What ethical and legal issues could arise?
3. To what extent should photographs be owned? Suppose a person places his or her photograph on a website and someone else downloads that photograph, alters it so that the subject is in a compromising situation, and circulates the altered version. What recourse should the subject of the photograph have?
4. To what extent is a programmer who helps develop a violent video game responsible for any consequences of that game? Should children's access to video games be restricted? If so, how and by whom? What about other groups in society, such as convicted criminals?

Additional Reading

Angel, E. and D. Shreiner. *Interactive Computer Graphics, A Top-Down Approach with Shader-Based OpenGL,* 6th ed. Boston, MA: Addison-Wesley, 2011.

Bowman, D. A., E. Kruijff, J. J. LaViola, Jr., and I. Poupyrev. *3D User Interfaces Theory and Practice.* Boston, MA: Addison-Wesley, 2005.

Hill, Jr., F. L., and S. Kelley. *Computer Graphics Using OpenGL.* 3rd ed. Upper Saddle River, NJ: Prentice-Hall, 2007.

McConnell, J. J. *Computer Graphics, Theory into Practice.* Sudbury, MA: Jones and Bartlett, 2006.

Parent, R. *Computer Animation, Algorithms and Techniques,* 3rd ed. San Francisco, CA: Morgan Kaufmann, 2012.