Chapter 5: A Primer in 3D Graphics

Many of the readers of this book have enjoyed three-dimensional graphics—in films like *Independence Day* or *The Lost World*, or in video games like *DOOM* or *Super Mario World*—but chances are, you don't have any idea what powers the creative magic behind them. If so, this primer is written for you; it will help you to understand the basic components of three-dimensional graphics, and how we use a computer to create realistic images.

Tinker Toys: How the Computer Works with 3D

If you played with Tinker Toys as a child—or perhaps as an adult, playing with your children—you already have an intuitive understanding of how the computer conceptualizes 3D graphics. In the Tinker Toy set, you have spokes and hubs; the hubs are joint points, places where spokes are joined together. The spokes and wheels together create frames, outlines of structures like buildings, rocket ships, boats, and so forth.

In the computer-generated world of 3D graphics, the computer uses *points*—which are analogous to the Tinker Toy hubs—and links these points together to create the frameworks of objects. The collection of points used to create an object is called a *point cloud* because, when you remove the framework that connects the points together, it looks like a cloud of points, floating in space. Almost all computer-generated objects begin as point clouds.

In the next step toward realism, the computer links the points together, creating a framework for the object. Now the object begins to look real, but hollow—very much like the superstructure of a building before the walls and floors have been added. This framework is also known as a *wireframe*. Each section of the framework – where the lines between points create a closed surface – is known as a *face*. For example, a cube is composed of eight points and six faces, while a pyramid is composed of five points and five faces.

After the frame has been created, a skin—called a *surface*—is applied to each face of the object. A surface can have many qualities; like the coat of paint on an automobile, it can be flat, shiny, or reflective, any color of the rainbow, and might even have racing stripes or a decal. The surface makes the object appear solid—even though it's hollow all the way through—and, with a creative surface design, the object can look impressively realistic.

In the real-world, objects are visible because they either emit or reflect light—you can see your car in the driveway because sunlight reflects off it and toward your eyes. In order to create a realistic view of an object, the computer must light the object using a computer-generated light source. Once the object has been lit, it is shaded. Each face of an object reflects light differently – it depends upon the lighting and surface qualities of the face - and the computer must calculate the shading for every face of an object.

Now the computer's created an object that looks very much as it would in the real world; the degree of realism is generally proportional to the amount of computing done to create a view of the object. Each dinosaur in *The Lost World* took thousands of hours to create, render, and animate; using tools on the CD-ROM, however, you can be building 3D worlds in just a few minutes. Your own creations won't likely fool anyone into thinking that they're "real" from the way they look – but you'll soon be learning how to make them *behave* realistically – something that took countless hours for Steven Spielberg's crew is something that you can do in just a few.

Any understanding of 3D graphics must begin with place—where things are.

A Place for Everything

Everything in the world has a *position*—the place where it is. Think of your living room: in one corner is a sofa, and opposite it—in most homes—sits the television set. These positions are important because they define relationships, associations, and possibilities. You are about a foot away from this book as you read it; any less and the page would be blurry, too much more and the print would be too small.

In addition to position, the physical world has three dimensions, or *axes of translation*, which we call width, height, and depth. Width is the quality of an object as it spreads from side to side; height is its characteristic as it spans above and below us; and depth is its characteristic as it comes toward or recedes away from us.

The computer doesn't understand width or what that might imply; its realm is one of pure mathematics, of symbols rather than words. Instead of "breadth", the computer uses "x". For "height", the computer uses "y", and for "depth", "z". To the computer, x, y, and z are as meaningful as width, height, and depth are for us. They are often represented as axes coming together to form one corner of a cube.

Yaw, Pitch, and Roll

Let's go back to your living room again. You've placed the television set on the other side of the room from the sofa. Did you place it so that the picture tube points toward the sofa—or perhaps so that the picture shines against the wall? In either case, the television is in the same location, but its *orientation* has changed. In the real world, every object has a position *and* an orientation, and these are independent attributes. Think of a traffic light hanging in the middle of an intersection—its location is not particularly important, but its orientation certainly is.

Orientation is represented—again—using three values, because we can move an object through three *axes of rotation*. These rotations are called pitch (which relates to looping), yaw (which relates to spinning), and roll (which relates to cartwheeling); each of these corresponds to orientation in x, y, and z, respectively.

Take a child's top as an example; it might wobble a bit, but basically it spins around and around. This spinning is called yawing. If you sat in a swivel chair and spun around, you'd be changing your orientation in *yaw*.

A roller coaster—especially the newer models—always lifts you up an incredible ramp, until it dips you down into a screaming terror. This lifting up and dropping down is called pitching. If you've been on one of the newer coasters that do a full loop-de-loop, you've been through the entire circle of *pitch*.

Finally, consider doing a cartwheel; you put your hands out to your side, and roll your body. In the end, you'll finish in the same position from which you began and, if you've done it well, you'll neither spin nor bounce—that is, yaw or pitch—as you make the movement. This change in orientation is called *roll*.

Yaw, pitch, and roll are circular motions; if you yaw far enough—spin in an entire circle—you'll end up back where you started. Because these motions have that circular quality, they're measured in degrees. A complete circle has 360 degrees, so a loop-de-loop on a roller coaster puts you through 360 degrees of pitch.

We often collect the three axes of translation – that is, x, y, and z of location – together with the three axes of rotation – that is, the pitch, yaw, and roll of orientation - and call them *six degrees of freedom*. That means that in the 3D computer world, six different values are required to completely specify the location and orientation of an object, both where it is *and* where it’s pointing.

Polygons and Normals

As we mentioned in the introduction to this chapter, the computer creates an object out of a cloud of points; these points are then connected together into a frame, and this frame is then covered with a surface. The surface is composed of faces, objects which are also known as *polygons*.

A polygon is the basic building block of a computer-generated 3D world. It can be thought of as a sheet—a sheet that is perfectly flat and infinitely thin. You already know how to identify some polygons; for example, a polygon composed of three points (from the point cloud) would be a triangle, four points would create a quadrilateral, like a cube or rectangle, five points would make a pentagon, six a hexagon, and so forth. A polygon can be very complex, with hundreds or thousands of points, but it must have some area inside of it, so a line (two points) is not a polygon.

Computers particularly like triangles; it's very easy for a computer to "color in" or *shade* the interior of a triangular area, but it is often very difficult for a computer to fill an irregularly shaped polygon. Most 3D authoring tools will enable you to create polygons with many sides, but they'll later break them down into three-sided polygons for the computer's convenience.

What's rather strange about polygons is that they have *only* one visible side. While this may seem impossible, remember that polygons are mathematical constructions—a one-sided object makes sense mathematically, even though it doesn't make much sense in the real world. It's as if polygons have an "outside" without having an inside.

Now the question is: which side of a polygon is the "outside," that is, the visible side? This is determined by the *normal* of the polygon. A normal is a fancy way of saying a "right angle." The normal of a polygon is a line that passes through the surface of the polygon, at a right angle to it.

The normal can be thought of as a light ray that passes through the polygon's surface on the way to your eye. A polygon has its "outside" on the side where the normal points *away* from the polygon's surface. A polygon's normal determines its visible side.

Lights

First the computer takes points in space, connects them to build a framework, and then wraps that framework in a surface of polygons. Now the object must be illuminated with a computer-generated light source, to create the real-world effects of reflectivity, shininess, and shadow. As in the real world, nothing is visible unless it is lit; because world creators often forget to put light sources into their worlds, 3D applications often provide an "ambient" light source, which provides uniform lighting to an object if no other light source is supplied.

Well-designed lighting can create a mood or effect that makes any scene more realistic. Think about it—if mystery and horror films were brightly lit, they'd be less tense, less brooding. A bright scene makes people feel happy and contented, while a darker scene implies danger and foreboding. Light is mood; the same is true in computer-generated environments.

Lights come in several varieties, as they do in the real world. A bare lamp can illuminate an entire room, whereas a spot light can highlight specific features of the environment. The Sun gives a brilliant illumination, while the Moon casts a pale glow. Each of these have an analog in the world of 3D graphics.

Point Lights

Point lights are light sources that radiate out equally in all directions from a single point. An incandescent light bulb, without a shade, is a good example of a point light. Point lights can be near, like a lamp, or very far away, like the Sun.

Directional Lights

A directional light is very much like a point light. All of the light rays that come from a directional light are parallel to each other. That means the light behaves as if it were very

far away, which is a useful characteristic if you want to re-create sunlight; the Sun is so far away from us that it appears as if all of its rays are parallel to us here on Earth.

Spot Lights

It's often quite useful to have track lighting in 3D environments; that is, light sources that can be pointed at particular objects in the computer-generated world. These are called *spot lights*, and they have both a location (given in x, y, and z coordinates) and an orientation (given in pitch, yaw, and roll degrees). In addition, spot lights also have a focus, called an *umbra*. The umbra determines the width of the beam cast by the spot light, that is, how quickly it widens as it passes through. A laser beam is an extreme example of this kind of spot light; it remains perfectly straight as it radiates outward from its point source. Stage lights, used in the theater, are often adjustable spot lights.

Rendering

We've created our object from points, frames, and polygons, and illuminated it. Suppose, however, that I just wanted to look at the frame of the object; it would be see-through, and would look much like our Tinker Toy framework. This type of drawing, or *rendering*, as it's called in computer graphics, is called *wireframe* rendering. The computer connects the dots in the point cloud with lines or wires, and displays on these wires.

The computer can render a wireframe object very quickly, so wireframe rendering is often used on low-power computers to show 3D models without painfully slowing down things.

Conversely, if you want to see the faces of an object, you would use solid rendering. In solid rendering, the polygons are "colored in" (technically known as "polygon filling"). This creates a rendered view of an object that appears solid.

Shading

Solid objects are composed of polygon faces; these faces have normals which determine how light is reflected from these faces. The computer calculates how much light is shining onto any polygon face from any of the light sources in the 3D scene; this is known as *shading*. There are several types of calculations that can be used to determine the shading of an object; some of these are simple and fast, others are complicated and very slow.

We've all seen video games that look cartoonish, and we've seen incredible computer graphics in films that almost appear alive. The biggest difference between *DOOM* at the low end and *The Lost World* at the high end is the shading model used in each. *DOOM* is fast because it uses an uncomplicated shading model; it creates 3D, but at the expense of realism.

Flat Shading

Flat shading is the simplest type of shading. Using the normal—that is, the light ray that projects outward from the polygon's face—the computer calculates a single value for the angle between the polygon face's normal and the light source, and uses that value to shade the entire polygon face.

To get some understanding of how this might work, consider a light source that is directly in front of a polygon face that is facing it head-on. In this case, the polygon face will be quite bright—it's reflecting light coming directly from the source. Flat shading, while fast, does make objects look rather artificial.

If, on the other hand, the light source is at an angle, so that the light strikes the faces obliquely—like sunlight on North America during wintertime—the polygon faces reflect less light, and are therefore less brilliant.

Gouraud Shading

Each progressive level of shading uses more of the computer's mathematical ability to smooth the shading model. The next level after flat shading is called *Gouraud* shading. Gouraud shading uses the normal of the polygon's face plus the normals of the faces of all of the polygon's neighbors—and averages the shading from one polygon face to the next. There's a lot more math going on—all of the normals and the averaging—so Gouraud shading takes about 10 times more computer processing power than flat shading. Gouraud shading makes objects look much more realistic.

Phong Shading

Even with Gouraud shading, complex objects can still show banding—that is, lines where the math used to do the shading falls short. While Gouraud shading is very common on computers, it still looks somewhat artificial.

Phong shading was invented to overcome some of the deficiencies of Gouraud shading. Using even more mathematical calculations, the computer uses the normal of the polygon face, and calculates more normals for every corner of the polygon's face (which can be very many if the polygon has an irregular shape), and averages these together with the normals (both center and corners) from all of the adjoining polygons. This again is about 10 times more processing-intensive than Gouraud shading, but it produces results that are pretty spectacular.

Ray Tracing

Ray tracing, used in TV commercials, Hollywood films, and photo-realistic still images, is by far the most mathematically intensive form of shading. The computer literally calculates the path of every ray of light through the scene—traces the paths—and from this, generates a "realistic" view of the 3D world. Depending on the complexity of the world, this can take anywhere from a few minutes to a few days, even on the most powerful computers in the world.

Texture Mapping

The real world is rich and detailed. Computers have trouble maintaining that complexity (unless they are very big and very expensive)—so we cheat a lot. One way we cheat is by using something called *texture maps*. Texture maps are a kind of "wallpaper" that is applied to polygon faces. Often, an entire object is "wrapped" with a single texture map. A texture map is an image, literally a picture.

As an example, consider a model of Earth. We could, if we chose, model every feature of the planet—every mountain range, every sea floor rift, and so forth. It would be a complex model, with billions if not trillions of polygons. It's unlikely we could ever get that onto any computer we have today. But what if, instead, we took a sphere, and then "wrapped" a detailed texture map of the Earth's surface onto it. It would not have depth, but it would have detail. And it would be visible on almost any computer.

Texture maps are a great way to cheat the complex. Using them, you can create scenes that are rich with images—billboards in cyberspace, wood grain on a floor, or wallpaper in a room—without creating a complicated environment.

Conclusion

While those are the basics of three-dimensional graphics, we've left a great deal of ground uncovered. Many of these topics—ray tracing, texture mapping, lighting—take an entire book to cover in detail. You now have a basis for understanding how 3D graphics work, but the best way to learn is by experimentation. Everything covered here - except ray tracing - is possible in VRML, the only limits are your creativity and your craft. You may not get *Toy Story* – but you could easily create something as engaging. Remember – visual fidelity isn't the key to success; richness, emotion and engagement are. But that's the subject of our next chapter…