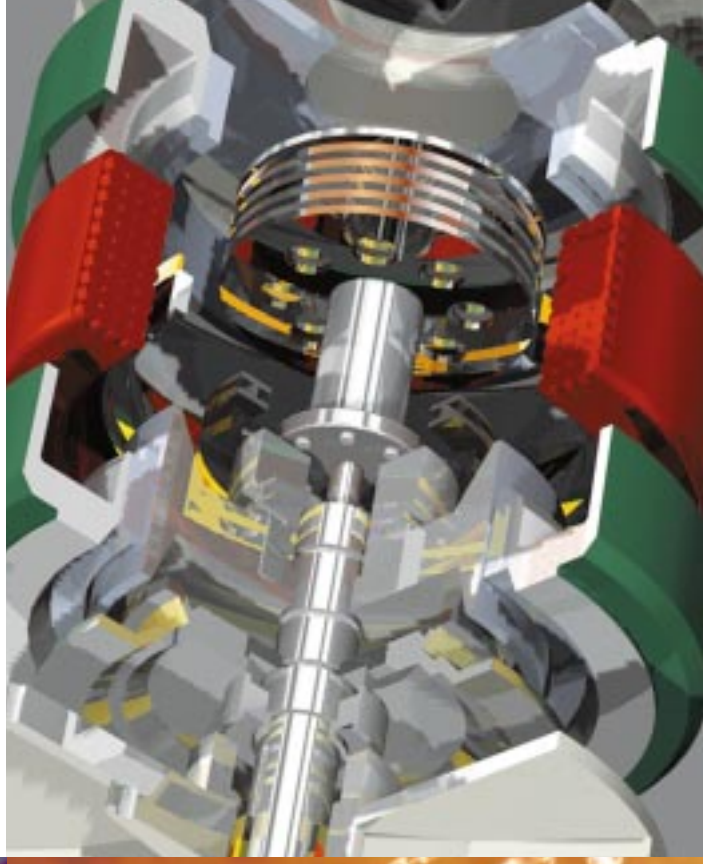
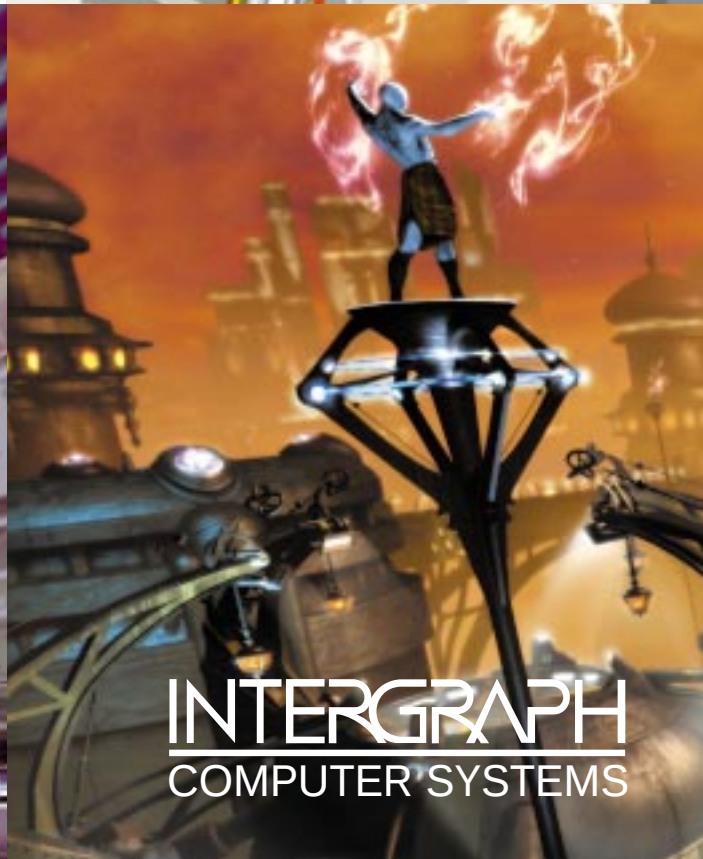
GRAPHICS SUPERCOMPUTING ON WINDOWS NT





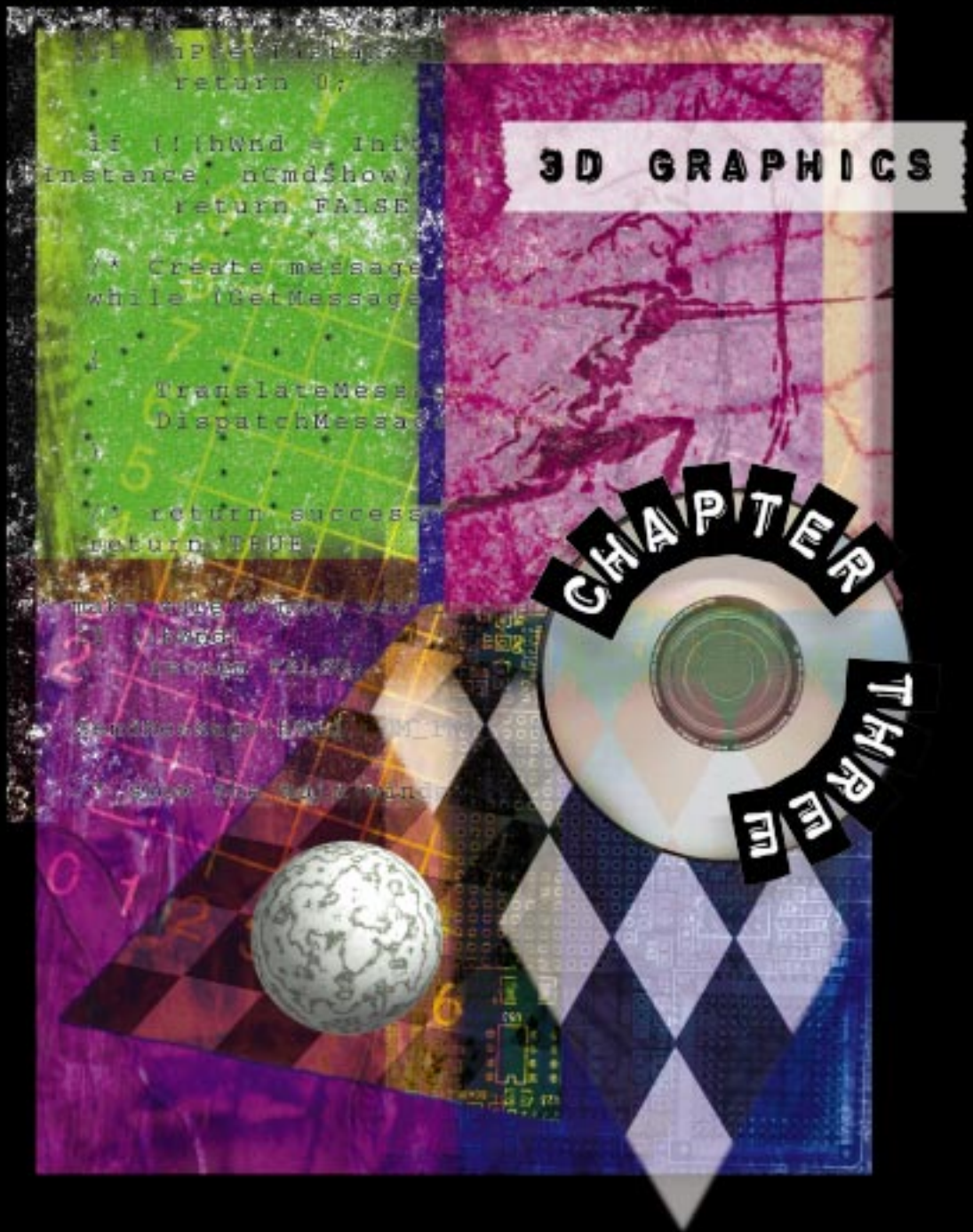
INTERGRAPH
COMPUTER SYSTEMS

GRAPHICS SUPERCOMPUTING ON WINDOWS NT

Creator image courtesy of Blur Studio. This image was created by Sam Gephardt with art direction by Tim and Jennifer Miller. The image was created on an Intergraph TDZ-610 quad-200 MHz processor workstation with RealiZm graphics using Kinetix® 3D Studio MAX® R2 and Adobe Photoshop® 4.0.

Virtual Reality Mars Rover Simulator image courtesy CG^2 Inc. This simulation, which can be experienced at U.S. Space Camp, was created using CG^2's VTree™ SDK, the cost-effective OpenGL API for developement of real-time visualization and simulation applications, and runs on Intergraph's TDZ workstations with RealiZm graphics.

Electric brake image courtesy of Intergraph Corporation. This image was created by R. Trent Pope on an Intergraph TDZ-425 workstation with RealiZm graphics, modeled with Solid Edge™ and rendered with TriSpectives™.


3D GRAPHICS
CHAPTER THREE

What are 3D Graphics and Who Needs Them?

The "créme de la créme" of computer graphics applications and graphics subsystems are those that can be used to create photorealistic 3D *computer-generated images (CGI)* and animations. Each element in a 3D image is considered to have `x`, `y`, and `z` (horizontal, vertical, and depth) coordinates.

3D computer graphics applications offer a number of significant advantages over their 2D counterparts. The 3D applications generate images with shading, shadows, reflections, and material textures that are far more technically correct (and hence more realistic) than we can "hand-draw" with a 2D package. Furthermore, once we've used a 3D application to create an image, we can easily make changes to the lighting, size, and material properties of objects, and our point of view, and then redisplay the resulting image. Unlike a 2D representation, a 3D application allows us to *"stroll around to the back"* of the 3D image to see what's there.

The stunningly realistic images produced by 3D graphics applications are employed for many diverse purposes, including animations and special effects for films and commercials, flight and vehicle simulations, architectural walk-throughs of proposed buildings to check aesthetic and ergonomic considerations, interactive mechanical design, scientific visualizations, and virtual reality environments.

The 3D Graphics Workflow

Although there are a number of entry points into the 3D graphics process, we typically start by using our 3D application to create a set of models that are squirreled away in a database for subsequent use (Figure 3-1).

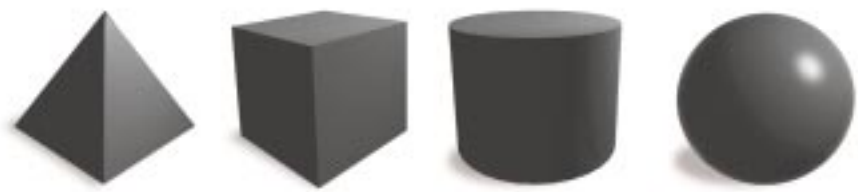


Figure 3-1. First we create some models.

The models shown here are obviously simple to the extreme. In the real world, we might create models that are fantastically sophisticated, such as a model of an elephant that is difficult to distinguish from the real thing. At some stage, a process known as *tessellation*

is used to automatically convert each model into a collection of small polygons (usually triangles), but this is beyond the scope of our discussions here (see *tessellation* in the glossary).

Note that when creating these models, one would typically assign them default material properties such as color, shininess, and so forth. We haven't assigned any such properties here, however, because we wish to differentiate the various stages in the process. When all of the models have been created, the next step is to start building a 3D world (Figure 3-2).

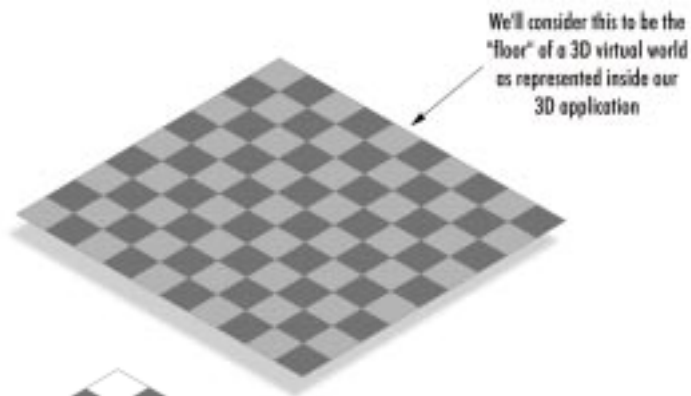


Figure 3-2.
Our 3D world is initially dark and gloomy because we haven't applied any lighting at this stage.

Figure 3-3.
Ambient lighting affects all of the objects in the 3D world identically.

Applying ambient light

In the real world, the 3D application would typically initialize with some form of default lighting, but we've chosen to show Figure 3-2 as being dark and gloomy to illustrate the various steps involved. In this case, our first step is to apply a form of lighting called *ambient lighting*, which affects all of the objects in the 3D world identically (see *lighting and material properties* in the glossary) (Figure 3-3).

Placing objects

At this stage, there are a number of things we might decide to do, including placing objects, lights, and a camera in our 3D world and specifying any attributes to be associated with these little rascals. For the purpose of this discussion, however, we'll assume that our next task is to retrieve some of the objects we created earlier from their sojourn in the model database and place them in our 3D world (Figure 3-4).

Figure 3-4:
We can place multiple copies of each model/object.

Note that we can happily place multiple copies of each object, such as the two balls shown here. Also note that once an object has been placed, it can be scaled independently of the other objects. In Figure 3-4, for example, we could easily make one of the balls twice the size of the other if we wished to do so.

Applying material properties

Remember that our objects would typically have been assigned default material properties (such as intrinsic color and shininess) when they were created, but we've chosen to represent them as relatively nondescript entities here to better emphasize the various steps in the process. Speaking of which, our next action would be to apply such properties to these objects (Figure 3-5).

Note that the reason we qualified the object's color as "intrinsic color" above is that an object's *final color*, the color it appears to be, depends on the lighting conditions. If no lighting is specified in our 3D application, for example, then all of the objects will (not surprisingly) appear to be black. Alternatively, if we were to decide to use a red light source, for example, then this would also affect the appearance of the objects.

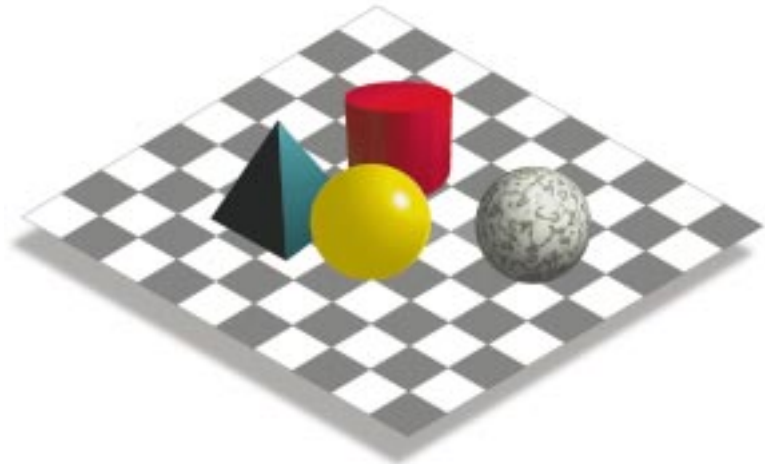


Figure 3-5.
Each object has its own material properties that define its intrinsic color, shininess, and so forth.

In addition to an object's intrinsic color, we can also specify its shininess, such as the matt surface we've applied to our pyramid, for example, and the glossy surface we've associated with our yellow ball. Note that although the yellow ball in Figure 3-5 is shiny, it doesn't cast any reflections of the other objects in the image. This is because we're currently considering a shaded image, and calculating effects such as reflections is much more complex (see *shading*, *rendering*, and *reflections* in the glossary).

Rather than simply associating an object with a color, we can also apply a pattern to it or give it the illusion of texture (such as the right-hand ball in Figure 3-5) by means of an image called a *texture map*. For the purpose of this discussion, we can consider a texture map to be a flat 2D image that we "drape over" (or "paint onto") our 3D object, but, of course, there's a little more to this than meets the eye (see *2D textures* and *3D textures* in the glossary).

Applying more realistic lighting models

Once we've applied material properties to our objects, the next step might be to supplement the ambient lighting with something more realistic, such as one or more positional light sources (Figure 3-6).

In this case, we've chosen to apply just one light source and for that source to present predominantly white light. Even so, this greatly improves the realism of the image, and we can add multiple such light sources, each of which can have its own spectral and other characteristics (see *lighting and material properties* in the glossary).

One important point is that applying the directional light source in Figure 3.6 doesn't cast any shadows! Why not? Well, what we've been referring to as our "3D world" is really just a mathematical representation in the computer's memory. We're currently considering a shaded image, and calculating effects like shadows requires a more complex form of rendering (see *shading* and *rendering* in the glossary).

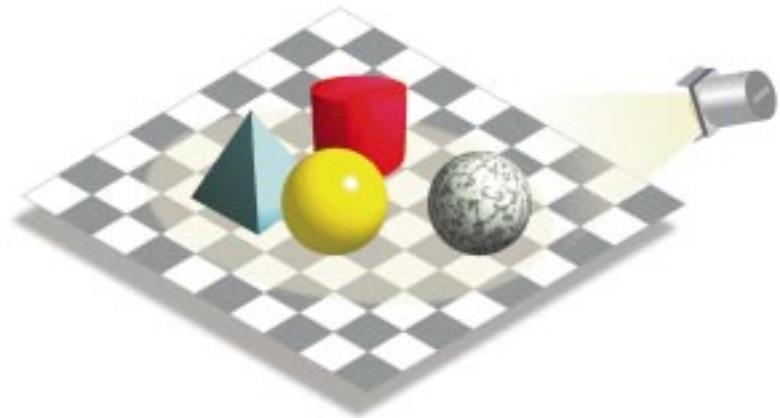


Figure 3-6.
Placing additional light sources improves the realism of the image.

Positioning a camera

Once we're happy with our lighting arrangements, the next step is to position a camera in order to define the view that we're going to see on the computer screen (Figure 3.7).

THE WORD *FRUSTUM* COMES FROM THE LATIN MEANING "PIECE" OR "BIT." IN MATHEMATICAL TERMS, A FRUSTUM IS THE PART OF A SOLID CONE OR PYRAMID THAT REMAINS AFTER WE'VE CUT ITS TOP OFF USING A PLANE THAT'S PARALLEL TO ITS BASE.

The tip (lens) of the camera (in our virtual world) is known as the *viewpoint*, and the totality of what can be "seen" by the camera is referred to as the *viewing frustum*. We also talk about an imaginary surface known as the *viewplane*, which represents the virtual location of the computer's screen. To further increase our delectation and delight, the width and height of the viewing frustum can be varied, as can the position of the viewplane along the viewing frustum's main axis, but for the sake of what little sanity we have left, we'll try not to dwell on this here. What we need is a cunning diversion that will allow us to slip gracefully into the next topic – GOOD GRIEF! Did you see what just flew past your window!

We're ready to rock-and-render

And so we are finally ready to perform a task known as rendering, which will generate the "all-singing-all-dancing" image on the computer's screen (Figure 3-8).

There are various rendering techniques, each of which requires different amounts of resources from the CPU and/or graphics subsystem, and each of which returns a different

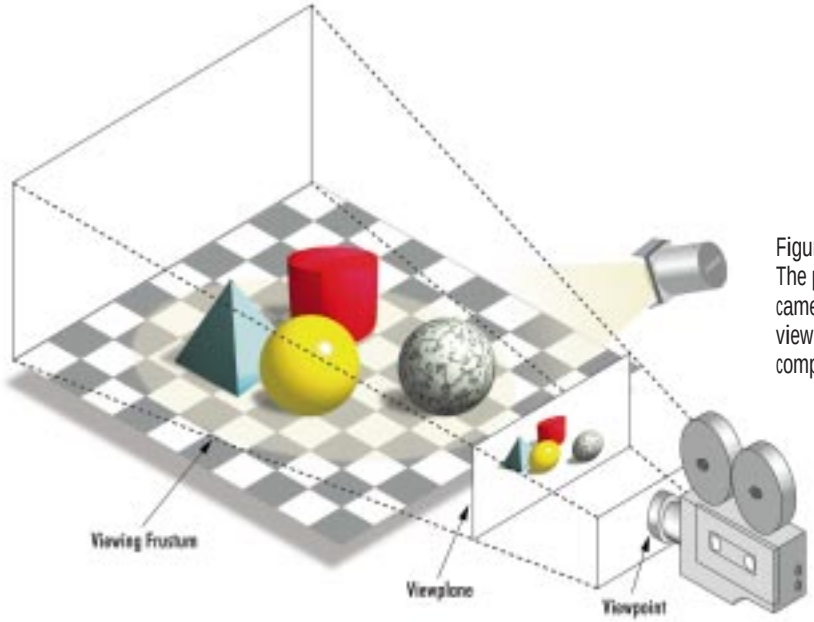


Figure 3-7.
The position of the camera defines the view we'll see on the computer screen.

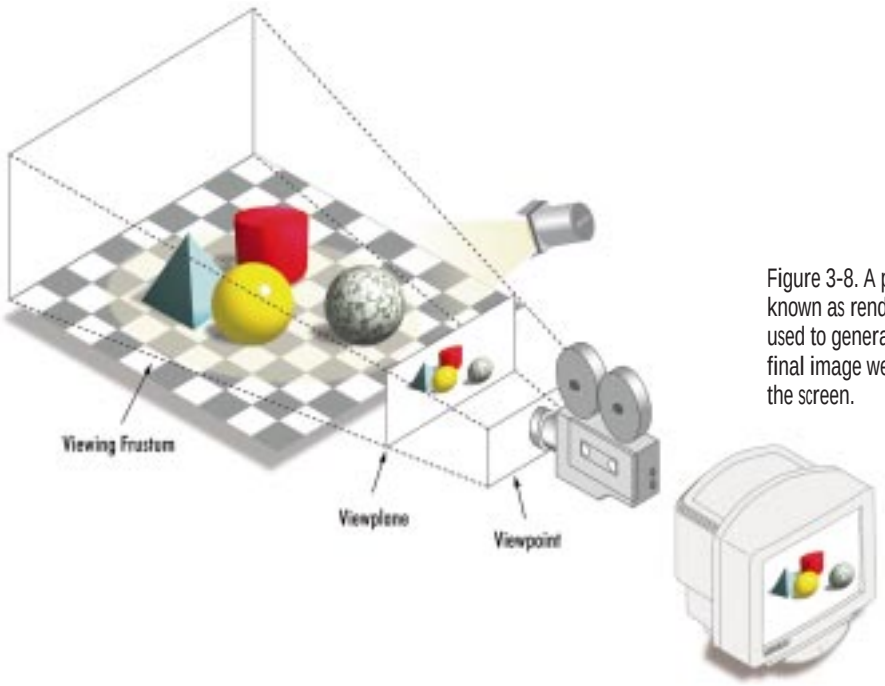


Figure 3-8. A process known as rendering is used to generate the final image we see on the screen.

level of realism. At the low end, we might display something called a *wireframe*, which essentially means that we display the objects as a collection of lines. Significantly more realistic results can be obtained using some type of shading algorithm, of which there are a variety of flavors, including (in increasing order of sophistication) *flat shading*, *Gouraud shading*, and *Phong shading*.

Algorithms such as Gouraud and Phong shading are extremely powerful and return highly sophisticated images. But there are yet more refined rendering algorithms available such as *ray tracing* and *radiosity rendering*, both of which require extreme amounts of processing, but both of which return much more photorealistic images (including shadows and reflections).

The advantage of rendering techniques such as Gouraud shading is that they take full advantage of hardware accelerated versions of OpenGL, which is the industry standard *Application Programming Interface* (API) for 3D graphics. This makes Gouraud shading ideal for interactive graphics applications where users need real-time performance. By comparison, rendering techniques like ray tracing and radiosity currently have to be performed in software. This means that ray tracing is somewhat time-consuming, even on today's high-performance graphics workstations. One solution is to use Intergraph's RenderGL® API, which extends the capabilities of OpenGL.

RenderGL is multithreaded to take advantage of today's multiprocessor Windows NT systems, and it also takes full advantage of Intergraph's high-performance 3D graphics accelerators. Images rendered using RenderGL approach the sophistication of ray traced images (including shadows), but they render up to *60 times faster* than traditional ray traced rendering techniques!

The 3D graphics pipeline

The term *3D graphics pipeline* refers to the steps involved in rendering a 3D image to the screen. This includes performing *geometry calculations*, *triangle setup*, and *rasterization* (Figure 3-9). Each of these steps may be realized using either software or hardware implementations. However, dedicated hardware solutions are far superior in performance to their software counterparts.

Graphics subsystems dedicated to the rendering of 3D images are far more sophisticated than their 2D counterparts. In addition to the frame buffer, such subsystems may contain *geometry engines*, *triangle setup engines*, *texturing engines*, and *Z buffers*. Furthermore, the rendering process and the graphics subsystems can add effects like *fogging* and make use of such tools as *overlay planes* and *stencil planes*.

The nitty-gritty details of the rendering process are really delightfully cool and interesting (honest), but if we plunged

headfirst into them here we probably wouldn't surface for air for quite some time, so we'll defer these discussions for the nonce. Suffice it to say that all of these terms are defined in excruciating detail in the glossary portion of this guide (see *3D graphics pipeline*, *shading*, and *rendering* in the glossary).

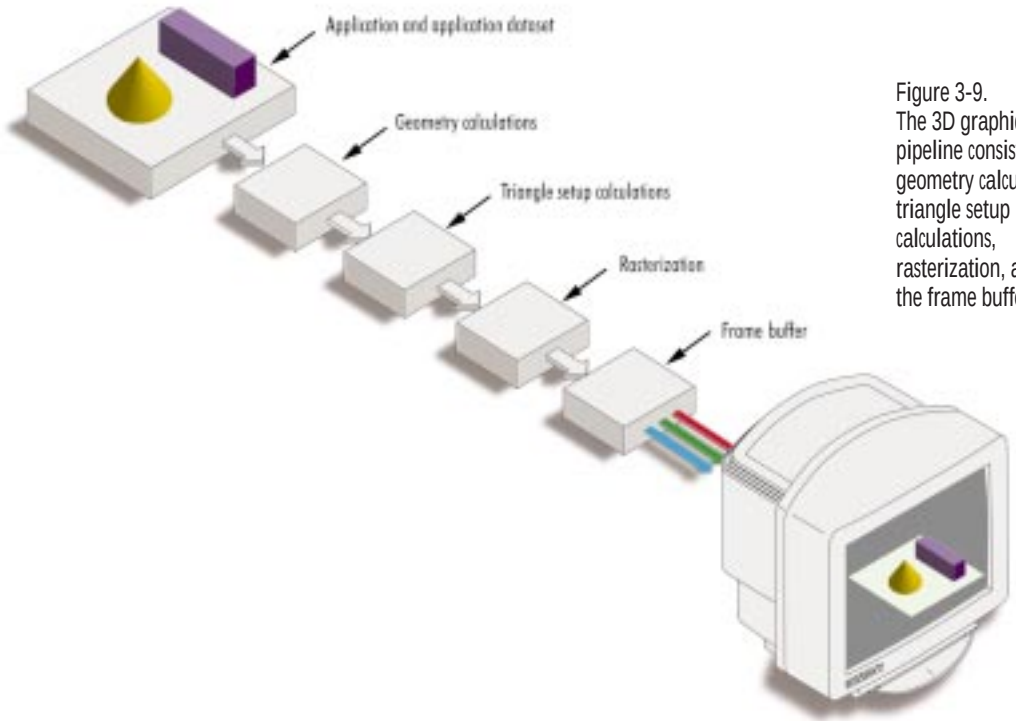


Figure 3-9.
The 3D graphics pipeline consists of geometry calculations, triangle setup calculations, rasterization, and the frame buffer.

Static versus dynamic 3D images

The process described above actually results in only a single (static) image. Many applications, however, require an entire sequence of such images. In such cases, the positions and orientations of the objects can be varied on a frame-by-frame basis, as can the positions, orientations, and spectral components of any light sources and the location and orientation of the camera.

Yet more cunning effects can be achieved by actually deforming or otherwise modifying the objects themselves on a frame-by-frame basis. When we drop a ball in the real world,

for example, it "squashes up" when it hits the floor and then expands and rebounds. We can emulate all of this in our 3D world.

In some cases, the individual images forming the sequence can be rendered in a leisurely manner, and the results can then be composited together to form a video or film. But many applications require the frames to be rendered on the fly so as to allow us to interact with the 3D world in real time, and this real-time rendering is where the really high-performance computers and graphics subsystems are required.

Summary

3D computer-generated graphics are most commonly associated with entertainment, engineering, and scientific applications. This form of representation, however, is having an ever-increasing impact on our lives, from choosing and buying a house to helping preserve our heritage.

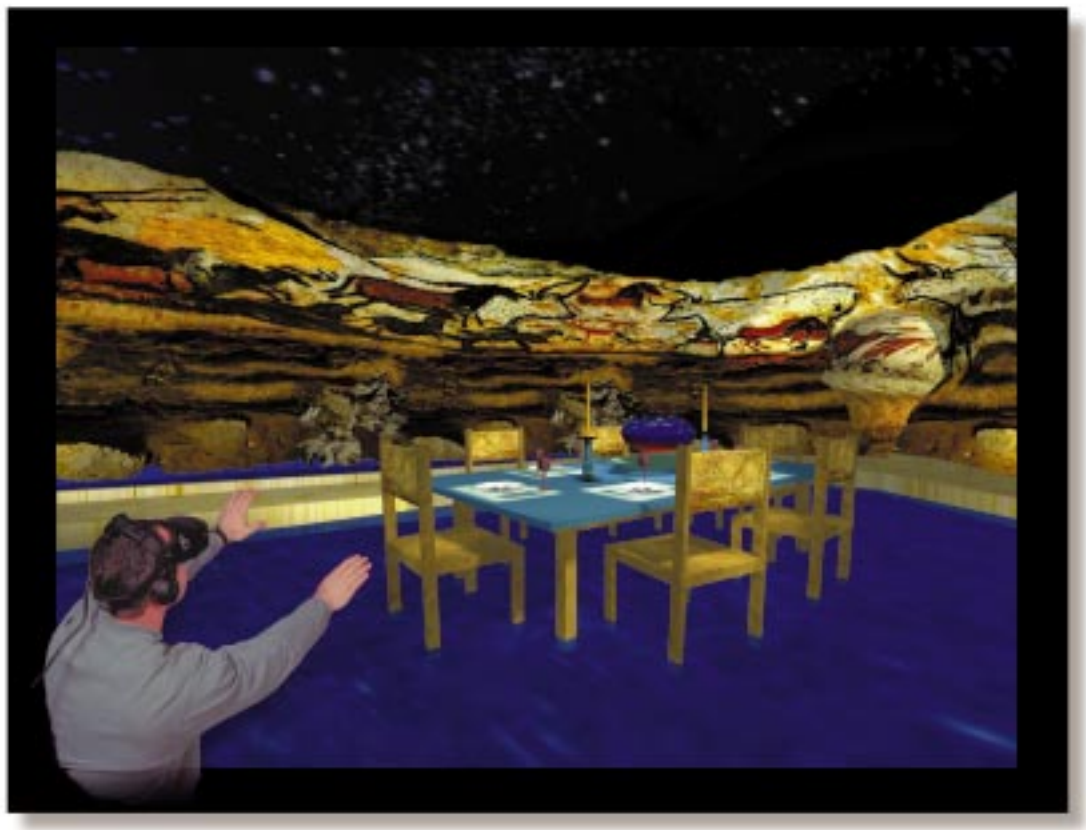
Preserving our heritage? You may recall the prehistoric painting introduced in Chapter 1. Many would say that the finest prehistoric paintings in the world are to be found in the cave of Lascaux in France. Unfortunately, this cave was closed to the public in 1963 because of wear and tear caused by visitors, including the detrimental effect of the water vapor expelled in their breath (the caves form an extensive three-dimensional maze, so running through them while holding one's breath is not considered to be a viable option).

In 1990, Benjamin Britton, assistant professor of fine art at the University of Cincinnati, began to create a 3D model of the cave using high-resolution photographs. The final exhibit, which was completed in 1995, allows visitors to perform virtual reality walkthroughs of the cave (Figure 3-10).

The high-resolution photographs of the real cave are used to create extremely detailed texture maps, and these texture maps are subsequently applied to a 3D model describing the contours of the cave's walls. This exhibit, which is currently appearing at museums around the world, was developed using Intergraph's high-performance 3D graphics workstations, which are numbered among the very few workstations in the world capable of performing this level of simulation. Similar unique projects include Intel's virtual model of Stonehenge and a virtual model of Shakespeare's original theater created by an Intergraph team in the UK.

The usability of any interactive 3D graphics application is directly affected by the speed and capability of the graphics subsystem. Intergraph Computer Systems is the leading vendor of high-performance Intel and Windows NT-based 3D graphics workstations. Intergraph's 3D graphics subsystems are designed for use with the most sophisticated 3D graphics applications to generate the highest quality images.

Figure 3.10.
Visitors can "walk through" this virtual reality cave and examine the world's finest prehistoric paintings without damaging the originals.



Virtual cave image provided courtesy of LASCAUX Virtual Reality Art Installation.
Head mount display provided courtesy of Virtual Research, Santa Clara, CA.

Since time immemorial, humans have employed pictures to quickly and efficiently convey both conceptual and emotional information. Today, we are creating some of the most fantastic 2D and 3D images imaginable using computer graphics workstations.

This book introduces graphics supercomputing on Windows NT and explains computer graphics technology in an interesting and informative way.

For information, call:

U.S.	1-800-763-0242
U.S. Federal	1-888-671-5339
Asia-Pacific	61-2-9929-2888
Europe	31-23-5666576
Middle East	971-4-367555
Other areas	1-205-730-5441

www.intergraph.com/ics

Printed in USA. 3/98 MC970941

INTERGRAPH
COMPUTER SYSTEMS

$49.95

DDDS350A0