Chapter 1

Computer Graphics at a glance

1.1 Overview

In the last few decades, computer graphics has evolved from a niche application of mostly academic interest to a common tool for image generation. A large number of images that are produced today do not emanate from a camera, but are generated in part or in their entirety by software in a computer.

While becoming more common, computer graphics has also become a more complex subject. In its early days, computer graphics used to be almost impossible to do, and in order to get any results at all you had to resort to simple and crude methods that were just barely good enough. To put it bluntly, you had to cheat. Over time, the cheats have become better as the processing power of computers has increased tremendously, and computer graphics can now reasonably be called a proper field of applied mathematics rather than a collection of software hacks. The hacks and cheats are still being used for interactive content like games, and they can still be good enough when the requirements on image quality are not too high, but even real time rendering methods are rapidly getting better and more advanced.

As a result of this improvement, it requires quite a lot of time and effort to get to the forefront of today's research and development in computer graphics, even if you restrict yourself to a narrow sub-topic. When taking the first steps in the field, it's easy to be confused, even intimidated, and it's useful to have a broad overview of the many aspects of the complex subject, a "map", so to speak. A conceptual map of computer graphics in the broad sense is presented in Figure 1.1. The figure is explained in detail below.

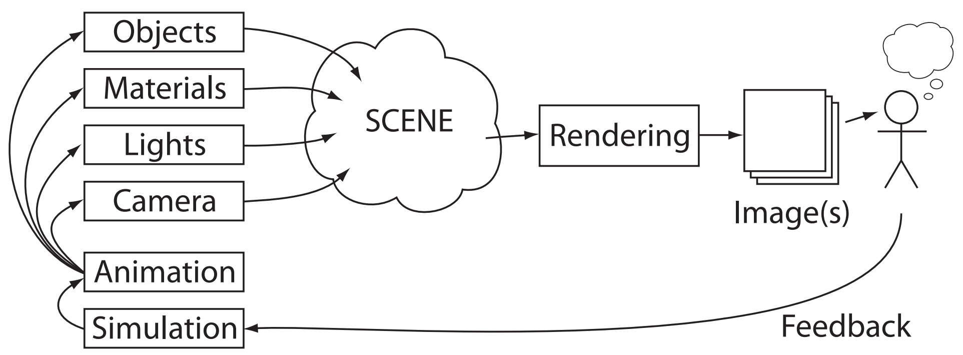

Figure 1.1: The technical components of computer graphics

1.1.1 Viewing

Starting at the far right, we have a human *observer* in the system. This might seem obvious, but it's important to keep in mind that the purpose of computer graphics is to create an impression in a human mind by presenting an image to a person. The primary concern, in fact the only concern, is what people *see*. The fact that a human viewer is an inherent part of the problem has a tremendous impact on the solutions.

1.1.2 Rendering

Second from right, we have the output of the process, the *image*. The process of computing a computer graphics image is called *rendering*. In technical terms, a film or video sequence is nothing but a sequence of still images presented in rapid sucession, and rendering of animated content is simply a matter of rendering still images with small incremental differences between frames. Some information can sometimes be saved between frames to speed things up, but broadly speaking, each frame of a sequence is rendered more or less separately.

Off-line versus real time

Animated content can be rendered to images and assembled off-line to a video sequence for later viewing, like for an animated movie or a special effects sequence, or the rendering can be performed in real time while the images are being watched, like in a computer game or a virtual reality application. Real time rendering places heavy demands on rendering speed. When rendering and displaying 30 or even 60 frames per second, you have only a few milliseconds to render an image with around one million pixels. That's only a few nanoseconds per pixel – a very small amount of time even for a computer. In off-line rendering, on the other hand, you can spend a lot more time on each image. A few seconds rather than a few milliseconds per frame is not a problem, and it's not uncommon to spend minutes or even hours on each frame if image quality is important. Five hours is a million times longer than 1/60 second, and, of course, being able to spend a million times more work on each image makes it possible to perform more computations, and computations of an entirely different kind, during rendering. Some frames can also be allowed to require more work than others. This not an option for real time rendering, where any extra delays would cause stutter in the frame rate.

This roughly million-fold difference in the required speed between real time rendering and off-line rendering is a huge gap which is not easily bridged. For the foreseeable future, real time rendering will continue to be constrained by the available resources and forced to use simpler rendering methods with less accurate simulations of optics and physics, and it will remain important to know what can be done, to know what is important for the situation at hand, and to spend the limited resources wisely.

1.1.3 Scene

The rendering process takes as its input a description of a virtual *scene.* The scene must contain geometric *objects*, surface properties of those objects in the form of *materials*, some description of virtual *light sources* to illuminate the scene, and some representation of a virtual *camera* to determine the vantage point of the rendered image. A virtual scene is generally a very large and complex data structure which requires careful design to be useful and efficient.

Objects

There are several different ways of representing objects in computer graphics. Solid objects can often be represented only by their *surface*, for example as a *mesh structure* with *vertices*, *polygons* connecting those vertices and surface *normals* to represent the surface orientation. Today, the polygon mesh is the invariably most common representation of objects in 3D graphics, both for real time and off-line rendering. It's a simple representation, but still flexible and general enough to be useful in many different circumstances. The polygons are mostly triangles, because a triangle is guaranteed to be planar, and any other polygon may be composed from two or more triangles. 3D modelling software often presents a mesh of quadrilaterals to the user for greater ease of use, but each quadrilateral is internally composed of two triangles.

Another common and useful object representation is a *parametric surface*, often represented as a set of *control points* and interpolating functions to determine where the surface is between those points. A different strategy for object representation is *implicit surfaces*, where a matematical expression in 3D coordinates determines what is the inside and the outside of an object. The implicit representation makes it possible to also represent a more complex *volume*, either as an enclosed surface with an interior density function or as a continuously varying volumetric function which is defined for every point in space. Volumetric information can also be represented as sampled data on a regular grid, which works a lot like pixel grids for 2D digital images, only with small rectilinear 3D *voxels* instead of 2D pixels. Such voxel volumes can represent sampled real world data or computed data from simulations.

Materials

Surfaces have a position and an orientation, which are both represented by the geometrical model, but they also have surface properties which determine their appearance. Real world surfaces are typically not a single color all over, but have a pattern of some sort. In computer graphics, this is modelled by a *texture map* on the surface, which can either be a mathematical function evaluated by a short program, so-called *procedural texturing*, or a sampled digital image represented as pixels, often referred to as a *bitmap texture.* Texture mapping requires some sort of texture coordinates to determine the position, orientation and size of the pattern on the surface.

Texture mapping can be used to vary any property of a surface, not only its color. Properties that are often mapped include transparency ("alpha") and glossiness. A texture an also be used to locally adjust the surface normal from its true direction to make a surface appear rough, despite being a flat polygon. This is called *bump mapping* or *normal mapping.* A texture can also be used to simulate reflections from shiny surfaces, so-called *reflection mapping.*

Another important property of a surface is its reflectance variation with the angle of incidence for the illumination and with the angle of view. The surface reflectance as a function of these two angles is called the Bidirectional Reflection Distribution Function, or *BRDF* for short. Modelling a BRDF to closely match that of a real surface is an ongoing effort in computer graphics research, and there are quite a few different reflection models in common use to model different kinds of surfaces.

Particularly complicated to simulate, but also quite important for many scenes, are materials that exhibit *subsurface scattering.* This means that the surface is not completely opaque, but that light penetrates some distance into the object before being reflected. This is a significant contributing factor for the appearance of many real world objects, like skin, milky or murky liquids, clouds and translucent plastic. Thin or small objects with subsurface scattering also allow some of the light to pass through to the opposite side of the object, making their appearance still harder to simulate.

Light sources

Most applications of computer graphics are making at least some sort of visual simulation of a real world situation. A real world image is created by light that is reflected from surfaces, and that light emanates from *light sources.* Virtual light sources come in many different flavours, but they are all borrowing at least some of their properties from their real world counterparts. At least one rudimentary light source is required in a scene, because otherwise there is no light and no image, but a good lighting design, in computer graphics as well as in the real world, commonly makes use of several light sources for dramatic effect and to highlight important features.

Virtual light sources can be given properties that would be impossible to achieve in reality, like a lack of decay with distance, a lack of shadows, a sudden drop-off after a certain distance, or the ability to illuminate only certain objects. Furthermore, computer graphics light sources are just mathematical constructs and have no physical shape, so they can be placed anywhere in a scene without obstructing the view or creating unpleasant glare in the camera. These unworldly properties are highly useful in some situations, but other times you need to go to great lengths to simulate the imperfections of real light sources to create a good looking image.

Good lighting design is crucial to achieving a good result in computer graphics, just like in reality. Even very ordinary everyday environments in the world around us have complex lighting, and mimicking that complexity with computer graphics lighting is a current effort in research and development. In many computer graphics applications, the illumination is still rather bland and disinteresting compared to reality, much because of the fact that it is still difficult during rendering to simulate how light bounces off various surfaces and creates *indirect illumination.*

The rendering problem of simulating the interactions between light and objects is called *global illumination.* In a real scene, indirect light comes from literally everywhere, and to compute how it gets where in a reasonably accurate manner is still taxing for modern computers. The physics and optics behind it has been known for centuries, and the math is not terribly complicated, but there is simply too much of it to compute in reasonable time. Even a weak indoor light source emits somewhere in the order of 10^{29} photons during a brief shutter time of 1 millisecond, and simulating them all with a software algorithm is downright impossible. Fortunately, there are short-cuts and cheats that can be employed, and with advanced rendering software used with enough care, most effects in global illumination can now be simulated reasonably well in off-line rendering. There are exceptions, however, and there is still research to be done.

Camera

The concept of a *camera* in computer graphics has been carried over from film and photography, but the roots are older than that. Human eyes create images of the outside world in much the same manner as a camera, and it comes natural to us to interpret such images as depictions of reality, or even as a substitute for reality. Computer graphics "cameras" are really just a projection from 3D to 2D, but for the storytelling it is often useful to treat the point of view as an object in the scene, either a camera or a viewer, that can be moved and rotated.

In computer graphics, the most common projection by far is a *central perspective projection*, where rays are assumed to converge in a centre of projection inside the camera. This is the projection created by a traditional camera lens, or at least the kind of projection a lens tries to accomplish, but it's not the only choice. Furthermore, a real world camera exhibits several imperfections like bad focus, lens distortions, colour aberrations, depth of field, motion blur and glare. Even though these effects can all be said to reduce the image quality, they sometimes need to be simulated to get the image you want.

1.1.4 Animation

A lot of the computer graphics content we experience today is *animated.* You could say that animation is simply the trick of changing some software parameters over time, like position and orientation of the camera and the objects in the scene, and rendering a sequence of images. While that would be technically true, it doesn't even begin to capture the array of new problems you encounter when trying to create and render animated graphics. You need to provide a way to describe motion in the scene, and you need to give users reasonably simple tools for creating and editing motion data. You also need to make your renderer aware of the motion if you want to simulate motion blur.

There are many ways to perform animation, and we won't try to make a list here, but the art of animation is still one of the most time consuming processes in the business. Some animations can be generated more or less automatically or with good assistance from advanced tools, but quite a lot of the animation work for computer graphics is still being performed with lots of manual labour.

Literally *anything* in a scene can be changed over time: the position and orientation of objects, their shape, colour and visibility, the illumination, the camera properties, anything. Knowing what to change, how to change it, and timing it right requires great care and skill, and a lot of practice.

1.1.5 Simulation

Animations can be created and edited manually, but some kinds of motion are better created by solving equations in a *simulation*. This is true for mechanical motion like objects falling, bouncing and colliding, cloth and hair draping over objects or blowing in the wind, and fluid and turbulent effects like water, smoke, fire and explosions. There are lots of software tools for simulating such effects in computer graphics. These tools have a lot in common with simulation software for engineering applications, even though computer graphics tools mostly take a more lax approach and compute solutions that look OK but are not very accurate.

Simulations can assist even in manual animation, by generating secondary motion for attachments like hair, fur and clothes, or the suspension of a car driving on a bumpy road. More complex assistance could involve adjusting the posture of an animated character to maintain balance when some limbs are moved manually, or simply to provide hints to whether something is physically plausible.

Mechanical simulation together with *collision detection* plays an important role for interactive content like games. It's a reasonable requirement that a virtual world should act and react like a real environment, within reasonable limits, and one would imagine that simple Newtonian mechanics, with forces acting on rigid objects, should be easy to compute. In fact, there's a lot more to it. The equations are deceptively simple, but the accuracy required in solving them with a computer turns out to be a problem. Collisions in the physical world are impulse events, with large forces of very short duration acting on objects, and accurate mathematical models of those contact forces are so-called "stiff equations", which are notoriously hard to solve by numerical simulation. Real time solutions often take the approach of making the world squishy, with "contact" between objects being represented by a gradually progressing repelling force acting at some small distance rather than a sudden, hard impact. This can greatly improve the accuracy and stability of the solution. Simulations for off-line rendering can be made less squishy and more accurate, but the exact equations are still stiff, and there is always a trade-off to be made between accuracy and speed. To save time, even a result that is slightly wrong can be acceptable, because it might not be visible to a human observer, and things that look slightly wrong can often be fixed with some manual tweaks.

1.1.6 Interaction

What has been described above has concerned off-line rendering as well as real time rendering. There are some differences, but the goals and the methods are similar for both situations. One thing that is unique to real time rendering, however, is *interaction*. The very purpose of rendering content in real time is that the viewer should be allowed an influence over what is shown. The interaction can be as simple as allowing the viewer to move the camera in the world, and that in itself adds a very strong sense of presence and believability. We are inquisitive beings who like to explore our environment on our own terms and in our own pace, whether that environment is real or virtual. A simple interactive walk-through of a virtual scene can be a lot more engaging than a pre-rendered video sequence, even if the pre-rendered images are of higher quality. Games and similar interactive experiences usually add some kind of physical interaction with the environment and objects, add some simulated inhabitants to the virtual world and create tasks for the user to solve. Even though the "artificial intelligence" (AI) for simulated characters in game worlds is still very far from deserving the name "intelligence", it is clear that games engage people a lot, and computer games taking place in virtual 3-D worlds is a very big business.

For interactive content, it is important to maintain a high frame rate for the rendering, at least 30 frames per second (FPS), preferably 60 FPS if the motions are rapid. One important reason is of course the same as for pre-rendered content: that you want to trick the human visual system into believing that the motion is continuous and not a sequence of still images. However, another important reason which is unique to interactive rendering is that you need a low *latency*: the delay from when the user signals that they want to take a particular action to when the action happens on the display must not be too long. For direct user interactions where immediate response is expected, a latency of as little as one tenth of a second becomes not only noticeable, but clearly objectionable and annoying. For camera motion, latency can even create physical discomfort in the form of motion sickness. This is why computer gamers spend lots of money on their graphics cards to get a high frame rate for their games: a high and steady frame rate makes for a lower latency and makes the game more believable, better looking and more fun.