Computer Graphics In Games

Kristopher Maddeaux
Thanks For Listening!
When I first arrived into my program I wanted to be a game designer. I never programmed before and I had no desire to. But after being exposed to programming concepts, such as computer graphics, I became passionate about programming.

Today I am going to give a very high level explanation about how computer graphics is used in game development. Hopefully everybody can learn something useful from my presentation.

So how are the graphics rendered in a 3D game?

To answer this, we must simplify the problem. In terms of 3D graphics, a game is made up of many 3D models. Each model is made up of many polygons, or triangles. Each triangle is made up three vertices.

So let’s say we have a 3D model of a cube.

If we divide the faces of the cube into triangles, which greatly speeds up the processing and rendering of the model on the GPU, the cube has 12 triangles. Since each triangle has three vertices, the cube has 36 vertices.

To render the cube onto the screen, the cube’s vertices are sent to the GPU’s programmable Graphics Pipeline. This pipeline is composed of the following steps:

1. From the 3D scene, the vertices are first sent to the vertex processor. Within the vertex processor, the location and color of the vertex in the 3D world is computed. The results are sent to the primitive assembly.
2. Within the primitive assembly, vertices are grouped into geometric objects, such as a triangle. The results are then sent to the clipper.
3. Within the clipper, objects that are not within the camera’s viewing volume are clipped. This is done to ensure that objects that will not show up onto the screen are not sent further down the pipeline. All the geometric objects that are not clipped are sent to the rasterizer.
4. The rasterizer computes a fragment that has a color and depth attribute for each corresponding pixel on the screen. These fragments are then sent to the fragment shader.
5. Within the fragment processor the final color of each fragment is computed. The results are then sent to the back buffer which are then sent to the screen to be displayed.

But how do we send these vertices to the GPU’s programmable Graphics Pipeline?

In order to communicate between the CPU (where the game program is running) and the GPU, we must use an API such as OpenGL or DirectX.

By using any of the API’s, we can send the vertices of our cube model to the GPU. But now we need to create code that can execute on the GPU at each stage so that the cube can be rendered and displayed onto the screen properly with a desired effect.

To do this we use shaders. Shaders are programs that can execute on the GPU. You can create shader programs to execute on each stage of the pipeline to achieve a desired effect.

For example, if we had a plane, with many vertices, how can we create a wave effect to animate the plan to look like a body of water like a lake? To do this we can use the sine to calculate the location of the vertices over time in the vertex shader to mimic a sine wave which will result in the plane looking like a body of water.

With a very flexible system, developers can easily make creative and innovative programs. Once mastering this process, developers can create programs to process non-graphical data. This is possible because the input and output of the graphics pipeline is essentially an image with each pixel having 4 channels. These four channels are just data and therefore can represent anything.

An example of this is storing all of the necessary data to compute the lighting in the scene after the scene has been rendered. By doing this, it greatly speeds up the rendering process.

Another example is that last summer I worked as a research assistant for Dr. Bill Kapralos in which I developed a framework on mobile devices which can process audio data on the GPU.