

UNDERSTANDING PIXEL AND VERTEX SHADERS

BY MATTHEW CHRISTIAN (MATT@INSIDEGAMER.ORG)

In this quick article I'm going to describe the difference between a vertex and a pixel shader and what they can be used for in game programming. They're both extremely useful for achieving higher-end graphics and come into play when you start learning about HLSL (High-Level Shading Language) and the programmable pipeline. Let's start with vertex shaders.

Note: It is assumed you have some slight knowledge of programming techniques and or experience. No code is used in this article but some basic information regarding computers may be necessary.

VERTICES, POLYGONS, AND RENDERING POWER

To start out you need to know what a vertex is. Basically a vertex is the point on a triangle, for you math majors out there a normal triangle would then contain 3 vertices. Unless you're using some really messed up system, all your geometry will be made up of triangles. Remember back in the days of the Nintendo64 where Mario had huge square hands with sharp jagged points? That was because the hardware could only process so much, so the developers had to cut down on the amount of polygons used on Mario. Think of a polygon simply as a collection of triangles shaped to resemble an object. Here is a key point when understanding vertices and triangles! The more triangles and vertices a polygon contains the smoother the object is, but requires more processing. Smaller amounts of triangles, means a rougher object that takes less processing power. Notice the Spyro/Ratchet picture on the right. The above is Spyro from Spyro the Dragon which was an earlier game from Insomniac Games (the lower is Ratchet from Ratchet and Clank Future: Tools of Destruction). Spyro was on the PlayStation while Ratchet will be on the PlayStation 3 which, obviously, heavily out performs the original PlayStation. There are three key points in noticing an increase in the amount of vertices and triangles. First, look at each character's shape. Ratchet has many more curved areas and smooth spots while Spyro is quite jagged and 'flat' looking. Spyro probably contains a few hundred vertices/ faces while Ratchet probably contains millions if not more. Next, look at each character's facial features. Ratchet, having many more vertices and points to raise and lower, has much better facial expression. Spyro's face is quite blank or 'robotic' looking.

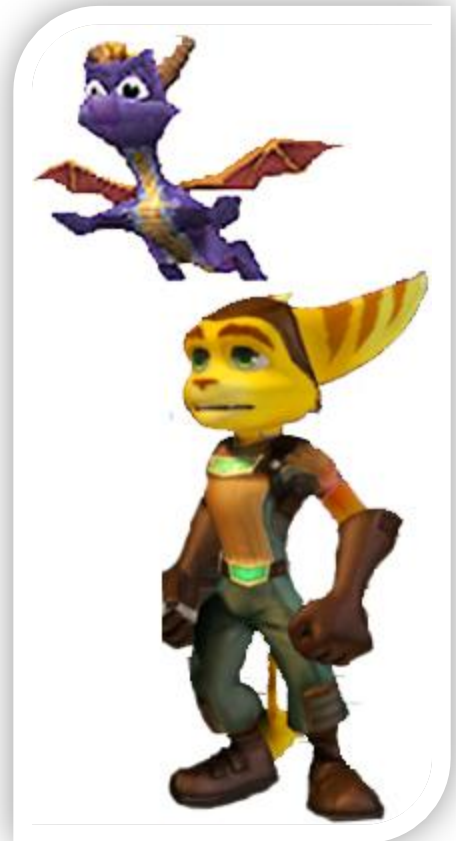


Figure 1 – Spyro and Ratchet
(Images used with permission from
Insomniac Games)

VERTEX SHADERS

Now that you know the basics on vertices, polygons, and rendering, we can move ahead and finally talk about vertex shaders. Vertex shaders are special functions used to manipulate the vertex data by using mathematical operations. Basic vertices are stored with an X, Y, and Z position. Another addition is the color of a vertex. Using vertex shaders you can manipulate the position and colors, changing only their values and not the way their data is stored. Colors and position are very basic but are what's needed to build up to vertex shading examples that can do things like manipulate the vertex position to create more fluid animations. Also included are effects that may not seem to affect the vertex themselves but the way the object is seen. For example, fog, heat waves, and motion blur can all be simulated using vertex shaders. A helpful document provided by nVidia can be found [here](#).

PIXEL SHADERS

Vertex shaders are very handy when it comes to changing the figure of an object but what about how the object looks up close? Enter pixel shaders. A pixel shader goes to each pixel and determines how it should look, thus creating even more detail than what is provided by the vertex shaders. Although they're usually mentioned with vertex shaders, they do quite a different duty. They provide surfaces with the actual feeling they would have in real life. For example, if you had a brick that had small pieces of sand and felt grainy you could apply a pixel shader to replicate that feel. Usually in game programming when we use the word 'texture' we refer to some type of image overlayed on the geometry to give it the same colors and appearances that object would have in real life. In modern day though we would typically refer to the word 'texture' as how something felt, if the cloth was silky smooth or bumpy, if the rock was smooth or had pieces of grainy sand stuck to it. This is what a pixel shader tries to simulate, the real world 'texture' of an object. This includes features such as bump-mapping, a technique used to make flat game textures look as if they have depth; or parallax mapping which does the same thing but adds the element of lighting and shadows on the surface. A document provided by nVidia can be found [here](#).

CONCLUSION

There is a lot to understanding vertex and pixel shaders. This is simply a portion of a primer that's needed to understand shaders, effects, HLSL, and the programmable pipeline. After learning this, it's recommended you learn how the programmable pipeline differs from the fixed, what it is, and then finally get started with some basic HLSL.