Tutorial 11: Vertex Shaders

Collapse All

Summary

This tutorial will emphasize on the capabilities of the vertex shader. It is meant to show the users the possibilities of allowing manipulation of vertices.

The outcome of this tutorial will be a wave effect imposed on the character model from the previous tutorial. This effect is done entirely by the GPU with no CPU interaction with the data.

Source

(SDK root)\Samples\C++\Direct3D10\Tutorials\Tutorial11

Vertex Shader

While previous tutorials already have shown the use of the vertex shader, this tutorial is meant to emphasize this stage in the pipeline. Vertices contain various types of information such as coordinates, normals, texture coordinates, materials, colors, and even custom data. For example, if you look under the ParticlesGS sample, you will see that each vertex passed into the pipeline will also contain a variable called Timer, which can vary the appearance of the particle over time.

The purpose of the vertex shader is to offload intensive calculations from the CPU, onto the GPU. This frees up the CPU to accomplish other tasks within the application. For example, games often offload the graphics processing to the GPU, while the AI and physics is done on the CPU itself.

This tutorial is meant to show the possibilities of the vertex shader, rather than to teach specific techniques. It is encouraged that you try new effects and experiment with the results. For specific methods to achieve effects, refer to the samples included with the SDK. However, do note that they are more complex in nature and apply much more advanced concepts.

Creating the Wave Effect

The wave effect on the model is created by modulating the x position of each point by a sine wave. The sine wave is controlled by the y position, thus creating a wave along the model. However, to make the wave move and appear animated, the sine wave is also modulated by a time variable.

Finally, there is a variable to control the amount of displacement by the vertex shader, and that is the waviness variable specified. This variable is passed into the shader and controlled by a GUI slider.

```
output.Pos.x += sin( output.Pos.y*0.1f + Time )*Waviness;
```

After this manipulation is done, the vertex shader will still have to prepare the vertices for display. Thus, there is still the transformation of the world, view, and projection matrices. However, we chose to do the world transformation before the wave effect, as it would be easier to determine the effects on the screen, since the x and y axis are apparent.

```
output.Pos = mul( output.Pos, View );
output.Pos = mul( output.Pos, Projection );
```