Blending Textures For Terrain

Alex Vlachos
ATI Research

Artists historically have had problems texturing terrain. Hiding texture seams and repetitive patterns is a difficult problem most games suffer from. This shader shows how to programatically interpolate between two textures based on their normal. The interpolation provides varying combinations of base textures when rendered as a combination of two textures. In the example below, a brown rocky texture is biased towards polygons that are generally facing towards the sky. A strange blue texture is given to anything that is on an incline. For hills that are in between, a blended combination of the two textures is used.

Excerpted from ShaderX: Vertex and Pixel Shader Tips and Tricks
Ideally, the angle of the normal from the ground plane should be used as the interpolation value to blend between the two textures. However, to simplify the math, an approximation can be implemented that simply uses the z component of each vertex normal. In the example shown here, \( z^3 \) is used to get a sharper cross fade between the two terrain textures. When using just \( z \), the blending is less harsh, which provides a slightly different look. Experimenting with different powers of \( z \) will be necessary to get the desired look given the surrounding scene.

The effect is achieved by computing the blend value in the vertex shader. The pixel shader will, at every pixel, use the interpolated blend value to blend between the two textures. In pre-shader hardware, these values can be pre-computed and stored in the vertex. However, more complex schemes can be implemented in a vertex shader if the desired effect would be to blend between 4 or 6 textures based on which direction the hill was facing. This technique could be used to simulate snow falling from a given direction sticking to the side of a mountain. Over time, the intensity of the blend value could be increased to show the accumulation of snow.

A further enhancement could be to compute the blend value per-pixel instead of per-vertex. This could provide some very interesting results when used with a bump map. When computing the per-vertex values, the blend only varies linearly over the face of a polygon. A bump map could provide more interesting variation within polygons.

**Vertex Shader**

```
vs.1.1
m4x4 oPos, v0, c0  //Transform vertex by transformation matrix
mov oT0, v7        //Texture coordinates for texture 0
mov oT1, v8        //Texture coordinates for texture 1
mul r0, v3.z, v3.z //z^2 into temp register
mul oT2, r0, v3.z  //z^3 into third set of texture coordinates
```

**Pixel Shader**

```
ps.1.4
texld r0, t0      //Base texture 0
texld r1, t1      //Base texture 1
texcrd r2.rgb, t2 //Interpolation value
lrp r0, r2.r, r1, r0 //Blend between the two textures
//Lighting computations go here
```

Excerpted from *ShaderX: Vertex and Pixel Shader Tips and Tricks*