Terrain Shader
Terrain Shader
Terrain Shader

• Why a Terrain?
  – Useful for outdoor levels
    • Ground to walk on
    • Visual boundary of environment
  – More efficient than assed
  – Easier to model
  – Multi-texturing and special lighting
Terrain Shader

• Why a Terrain?
  – Useful for outdoor levels
    • Ground to walk on
    • Visual boundary of environment
  – More efficient than assed
  – Easier to model
  – Multi-texturing and special lighting
  – Also: good example of vertex shader use
Terrain Shader

• How does it work?
  – Geometry is a regular triangle mesh
    • Fast to render
    • Created in source code
    • Can be chopped down
Terrain Shader

- Height provided by heightfield
- A grayscale texture
- Can be floating point
- Normal gray image usually enough
Terrain Shader

- Vertex shader:
  - Init shader
  - Grab texture coords
  - [...]
Terrain Shader

• Vertex shader:
  – [...]  
  – Get height from heightmap
  – Compute vertex position
  – [...]
Terrain Shader

- Vertex shader:
  - [...]
Terrain Shader

- Vertex shader:
  - [...]  
  - Compute normals  
  - [...]  

```
// 6) compute normal
float2 texCoordOffsetVec = xTexCoordDelta * 2.0f;
float north_h = tex2Dlod(HeightTextureSampler, float4(globalTexCoords + float2(0.0f, xTexCoordDelta.y), 0.0f, 1.0f)).x;
float south_h = tex2Dlod(HeightTextureSampler, float4(globalTexCoords + float2(0.0f, -xTexCoordDelta.y), 0.0f, 1.0f)).x;
float east_h = tex2Dlod(HeightTextureSampler, float4(globalTexCoords + float2(xTexCoordDelta.x, 0.0f), 0.0f, 1.0f)).x;
float west_h = tex2Dlod(HeightTextureSampler, float4(globalTexCoords + float2(-xTexCoordDelta.x, 0.0f), 0.0f, 1.0f)).x;
float two_texDelta = xTexCoordWorldDelta + xTexCoordWorldDelta;
float divisor = 1.0f / two_texDelta;

float3 south_minus_north = (0.0f, (south_h - north_h)*xHeightFieldWorldHeight, -(two_texDelta));
float3 east_minus_west = (-two_texDelta), (east_h - west_h)*xHeightFieldWorldHeight, 0.0f);
Output.Normal = float4(normalize(cross((south_minus_north)*divisor, (east_minus_west)*divisor)), 1.0f);
Output.Normal.x = -Output.Normal.x;
```

Fast accurate normal calculation for heightfield lighting on a non-isometric grid [H. Zhao, 2006]
Terrain Shader

- Vertex shader:
  - [...]  
  - Call standard vertex code

```cpp
return CommonVertexShader(Output);
```
Terrain Shader

- Vertex shader:
  - [...]  
  - Call standard vertex code

```cpp
return CommonVertexShader(Output);
```
• **Vertex shader:**

```cpp
CommonPS_input CommonVertexShader( CommonVS_input input )
{
    CommonPS_input Output = (CommonPS_input)0;

    // compute ViewProjection matrix and WorldViewProjection matrix
    float4x4 ViewProjection = mul(xViewMatrix, xProjectionMatrix);
    float4x4 LocalTransform = mul(xMeshBoneMatrix, xObjectWorldMatrix);
    float4x4 WorldProjection = mul(LocalTransform, ViewProjection);

    // calculate World position and Normal
    float4 Position = mul( input.Position, WorldProjection );
    float3 Normal = normalize(mul|normalize(input.Normal), (float3x3)LocalTransform)).xyz;

    // evaluate lighting equation and combine with vertex color
    Output.Position = Position;

    // add horizontal image shift
    Output.Position.x += xImageShift * Output.Position.w;

    Output.Normal = Normal;
    Output.TexCoords = input.TexCoords;
    Output.Color = input.Color;
    Output.LightingFactors = evaluateLightingEquation(Output.Position, Output.Normal);

    return Output;
}
```
Terrain Shader

\[ \text{Terrain} + \text{Shader} = \text{Result} \]
Terrain Shader

A bit colorless
Terrain Shader

- One could slap a single texture onto the surface
- Good enough for far away terrain
- When close looks horrible
Terrain Shader

- Instead:
- Use color coded texture
- One texture kind per color
- Allows smooth blended multi texturing
Terrain Shader
Terrain Shader

- Pixel shader:
  - Init shader
  - Grab channel texture
  - [...]
Terrain Shader

- Pixel shader:
  - […]
  - Grab stretch factors (global variable)
  - […]

```c
float4 xf = xLocalTexStretchFactors;
float yf = xGlobalHeightToWidthRelation;
```
Terrain Shader

- Pixel shader:
  - […]
  - Grab stretch factors (global variable)
  - […]

xf = 1  xf = 2  xf = 4
Terrain Shader

- Pixel shader:
  - [...]
  - Grab all texel values
  - Channel colors together
  - [...]
Terrain Shader

- **Pixel shader:**
  - [...]  
  - Grab all texel values  
  - Channel colors together  
  - [...]

\[ (0,1,0,0) \quad \rightarrow \quad (1,0,0,0) \]

\[ (0.5,0.5,0,0) \]
Terrain Shader

- Pixel shader:
  - [...] 
  - Finally add some light

```cpp
Output.Color = color * input.LightingFactors.x;
Output.Color += color * input.LightingFactors.y;
Output.Color += 0.33 * (1,1,1)*xSpecular * input.LightingFactors.z;
```
Terrain Shader

![Diagram of Terrain Shader process]

- [Image of terrain with grid]
- [Image of texture map]
- [Image of height map]
- [Image of color map]

Result:

- [Image of final terrain texture]

**ETH GAME PROGRAMMING LABORATORY**
Terrain Shading with *environment lights*

- Similar to previous environment lighting example
  - Write simple (~40 lines) ray-marcher in pixel shader
  - Use some pre-derived equations (or look-up tables)
  - Compute smooth shadows (and global-illumination)

See below for details, or contact derek@disneyresearch.com

http://www.dgp.toronto.edu/~derek/abstracts.2008.html#SN08
http://www.dgp.toronto.edu/~derek/abstracts.2009.html#NS09