COMPSCI 715 Part 2

Lecture 2 - GLSL Continued

Terminology

- Code: String of source to be compiled
- Shader Object: A holder for a shader of a particular type (i.e. vertex)
- Program Object: A holder to which shader objects can be attached and linked

Process

1. Create vertex shader object
2. Load code into shader object
3. Compile object
4. Repeat 1-3 for fragment shader
5. Create program object
6. Attach shader objects to program
7. Link program object

API - Load & Compile

- Create a handle

```
vHandle = glCreateShaderObjectARB (GL_VERTEX_SHADER_ARB);
```

- Pass in source

```
const GLchar *code = “...”;
glShaderSourceARB (vHandle, 1, &code, NULL);
```

- Compile

```
glCompileShaderARB (vHandle);
```

API - Link & Verify

- Create program object

```
handle = glCreateProgramObjectARB ();
```

- Attach shader objects

```
glAttachObjectARB (handle, vHandle);
```

- Link

```
glLinkProgramARB (handle);
```

- Verify

```
bool status;
glGetObjectParameterivARB (vHandle,
       GL_COMPILE_STATUS, &status);
glGetObjectParameterivARB (handle,
       GL_LINK_STATUS, &status);
```

API - Debug

- Part of specification is that driver must provide debug info for failed compilation

```
int length;
glGetObjectParameterivARB (handle, GL_INFO_LOG_LENGTH, &length);
char* log = new char[length];
glGetInfoLogARB (handle, length, NULL, log);
```

- Things are not so easy when it is a logic error - demonstrated later

API - Use

- Enable shader

```
glUseProgramObjectARB (handle);
```

- Get location of a variable

```
int location = glGetUniformLocationARB(handle, “name”);
```

- Set a variable

```
glUniform1iARB(location,value);
```

- Disable Shader

```
glUseProgramObjectARB (0);
```

GLSL Examples

- Per-pixel Lighting
- Tangent Space
- Normal Mapping
- Parallax Mapping
- Parallax Occlusion Mapping

Per-Pixel Lighting

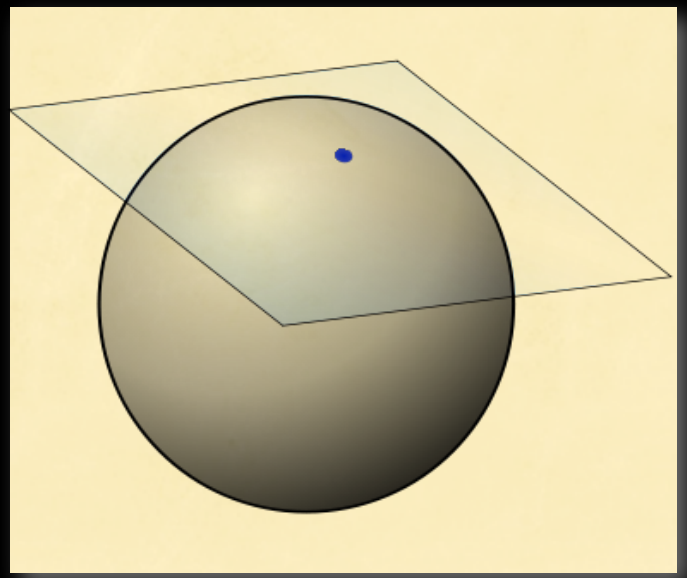
- Implement Phong Illumination model per-pixel
- Same as what we looked at in Lecture 1

Per-pixel lighting - impl

```
// ambient
  vec4 intensity = gl_LightSource[0].ambient;
// diffuse
  intensity += gl_LightSource[0].diffuse*max(0.0,dot(lVec,n));
// specular
  vec3 rVec = reflect(-lVec,n);
  float spec = pow(clamp(dot(rVec,vVec),0.0,1.0),6.0);
  intensity += gl_LightSource[0].specular*spec;
// texture map
  vec4 col = texture2D(colourMap,gl_TexCoord[0].st) * intensity;
  gl_FragColor = vec4(col.rgb,1.0);
```

Tangent Space

- Errors in lighting across curved surfaces
- So create a local coordinate system - called tangent space
- Express view / light dir in this coordinate space



Tangent Space - Impl

- Supply surface tangents at each vertex
- Calculate local system using normal matrix:

```
t = normalize(gl_NormalMatrix * Tangent);
b = normalize(gl_NormalMatrix * Binormal);
n = normalize(gl_NormalMatrix * gl_Normal);
```

- Transform light and view dir into local coord

```
v.x = dot(LightPosition, t);
v.y = dot(LightPosition, b);
v.z = dot(LightPosition, n);
lightDir = normalize(v);

v.x = dot(pos, t);
v.y = dot(pos, b);
v.z = dot(pos, n);
viewDir = normalize(v);
```

Holographic Textures

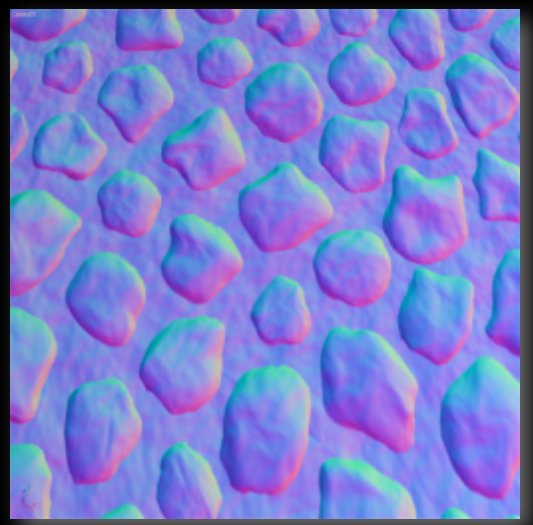
- What they are trying to do:
  - Create the illusion of highly detailed surface geometry
- Why?
  - With modern GPUs bottleneck is now with polygon count, and not pixel-processing

Normal Mapping

- Also called Bump Mapping
- First introduced SIGGRAPH 1996, extended in 1998
- Simplistic and cheap
- Used in Valve's Source engine and on Xbox, Xbox 360 and PS3 games

Normal Mapping

- Use another texture to supply the normal at each point across the surface
- RGB value are values of XYZ normal vector
- Can use alpha channel to change surface's diffuse reflectivity



Normal Mapping - Impl

- Vertex shader same as tangent space
- Get normal from normalMap tex:

```
vec3 norm = vec3(texture2D(normalMap, vec2(gl_TexCoord[0])));
norm = (norm - 0.5) * 2.0;
norm.y = -norm.y;
norm = normalize(norm);
```

- Use this in normal phong lighting

```
intensity += gl_LightSource[0].diffuse*max(0.0,dot(lightVec,norm));
vec3 reflectVec = reflect(-lightVec,norm);
float spec = pow(clamp(dot(reflectVec,viewVec),0.0,1.0),6.0);
intensity += gl_LightSource[0].specular*spec;
```

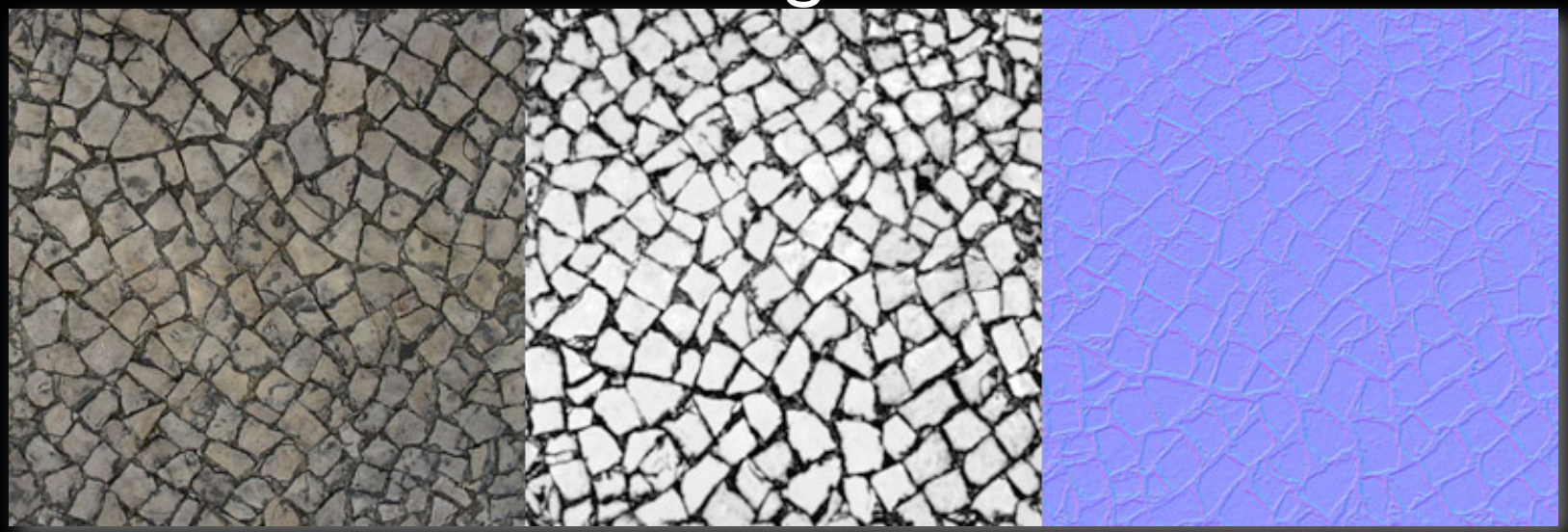
Parallax Mapping

- A simplistic (one step) attempt at true holographic texturing
- Introduced by T. Kaneko in 2001
- Does NOT produce accurate occlusion or silhouettes

Parallax Mapping

- Use either a height texture or alpha channel to provide depth data
- Use height at tex-coord as an offset to ‘true’ tex-coord

Colour Height Normal



Parallax Mapping - Impl

- Start with normal mapping base
- Get initial texture ‘height’ value

```
vec2 texCoord = vec2(gl_TexCoord[0]);
float height = (1.0 - texture2D(colourMap,texCoord).a)
               * offsetScale + offsetBias;
```

- Use this as an offset into final tex-coord

```
texCoord = texCoord - height * viewVec.yx;
```

Parallax Mapping - Impl

- What offset scale and bias?
  - Fully tweakable and material dependant
  - Scale: depth / width
  - Bias: -scale/2

Combining NM and PM

- These methods can be used in a complimentary fashion
- Simply combine/replace the relevant calculations

Problems with PM

- Extreme viewing angles
  - We aren't *actually* creating geometry
  - High frequency information -> artifacts
- No occlusion
- No silhouettes
- No shadows

Little Competition

- Use source provided which compiles a shader and applies it to a textured surface
- Alternatively create your own geometry
- Create some cool effect by next week....

- (I'll try to find some sort of prize for the best / most innovative)

Sources

- Wikipedia - Images and definitions
- Astle, D (ed.) (2006)More OpenGL Game Programming. Thomson Boston, MA
- Kaneko, T et al (2001). Detailed Shape Representation with Parallax Mapping. Proceedings of ICAT 2001