CG Programming II (VGP 352)

Agenda:

- Discuss last week's assignment.
  - Vent :)
  - Discuss the problems with that approach to per-pixel lighting.
- GLSL Intro.
  - What is it?
  - How is it used?

Sign up for the mailing list NOW!

http://lists.paranormal-entertainment.com/mailman/listinfo/aipd-vgp35x

Per-pixel Lighting without GLSL

- What are the problems with this technique?

Per-pixel Lighting without GLSL

- What are the problems with this technique?
  - Slow
    - Lots of work to do on the CPU.
    - New data per-frame -> uploads and pipeline stalls.
  - Difficult to implement
    - 'Nuff said.
  - Inflexible
    - Very difficult to implement an “shininess” values.
    - Requires multiple passes for diffuse, etc.

Root Causes of Those Problems

- Duplicate a lot of work OpenGL already does.
  - Re-transform data.
- Don't have access to the data that we really want.
  - Transformed light position.
  - Should just interpolate the transformed vertex normals.

Programmable GPUs Solve This

- Vertex stage is programmable.
  - Perform arbitrary operations on per-vertex inputs.
  - Pass arbitrary data to the fragment pipeline.
  - However, must also perform the usual vertex transformations.
- Fragment stage is programmable.
  - Perform arbitrary operations on inputs.
  - Must generate desired output color.
  - Can also modify fragment's Z value.

Dependent Texturing

- Can use arbitrary values to sample textures.
  - Interpolated texture coordinates (like fixed function).
  - Coordinates calculated by fragment program.
  - Value read from another texture.
    - Example: sample a noise texture, use that value to sample another texture.

What is GLSL?

- See 3dlabs GLSL overview.

http://developer.3dlabs.com/documents/presentations/GLSLOverview2005.zip

- Use GLSL Quick ref. for later.

http://www.mew.cx/glsl_quickref.pdf

Using GLSL

- There are a lot of steps, but it's not too scary.
  1. Create shader objects.
  2. Associate source code with shared objects.
     - To this point, it's *just like textures.*
  3. Compile objects.
  4. Attach objects to a program.
  5. Link program.
  6. Use the linked program!
- There's actually a *bit* more to it than this.

Basic Vertex Shader

```
varying vec3 normal;

void main()
{
    gl_Position = gl_ModelViewProjectionMatrix
       * gl_Vertex;
    normal = gl_NormalMatrix * gl_Normal;
}
```

Basic Vertex Shader (cont.)

- Shader text is stored as string inside program or read from disk file.

```
vert_shader =
  glCreateShaderObjectARB(GL_VERTEX_SHADER_ARB);

glShaderSourceARB(vert_shader, 1,
  &vert_shader_text, NULL);

glCompileShaderARB(vert_shader);
```

Basic Fragment Shader

```
uniform vec3 lightPos;
uniform vec4 diffuse;
uniform vec4 specular;
varying vec3 normal;

void main ()
{
    float dotProd = max(
        dot(lightPos, normalize(normal)), 0.0);
    gl_FragColor = diffuse * dotProd
        + specular * pow (dotProd, 20.0);
}
```

Basic Fragment Shader (cont.)

```
frag_shader =
  glCreateShaderObjectARB(GL_FRAGMENT_SHADER_ARB)
  ;

glShaderSourceARB(frag_shader, 1,
  &frag_shader_text, NULL);

glCompileShaderARB(frag_shader);
```

Putting Them Together

- Link the shaders as a program...

```
program = glCreateProgramObjectARB();
glAttachObjectARB(program, fragShader);
glAttachObjectARB(program, vertShader);
glLinkProgramARB(program);
glUseProgramObjectARB(program);
```

- And the uniforms...

```
uDiffuse = glGetUniformLocationARB(program,
  "diffuse");
glUniform4fvARB(uDiffuse, 1, diffuse);
```

Questions?

Legal Statement

- This work represents the view of the authors and does not necessarily represent the view of IBM or the Art Institute of Portland.
- OpenGL is a trademark of Silicon Graphics, Inc. in the United States, other countries, or both.
- Khronos and OpenGL ES are trademarks of the Khronos Group.
- Other company, product, and service names may be trademarks or service marks of others.