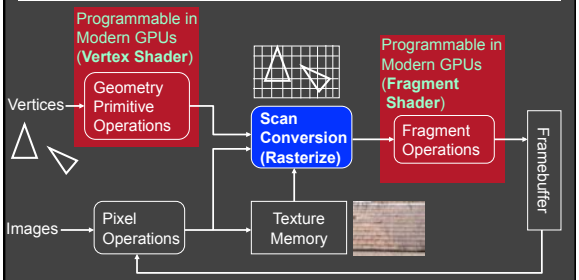
## Real-Time High Quality Rendering

CSE 291 [Fall 2015], Lecture 4

Brief Intro to Programmable Shaders

http://www.cs.ucsd.edu/~ravir

## OpenGL Rendering Pipeline




Traditional Approach: Fixed function pipeline (state machine)
New Development (2003-): Programmable pipeline

## Simplified OpenGL Pipeline

- User specifies vertices (vertex buffer object)
- For each vertex in parallel
  - OpenGL calls user-specified vertex shader: Transform vertex (ModelView, Projection), other ops
- For each primitive, OpenGL rasterizes
  - Generates a *fragment* for each pixel the fragment covers
- For each fragment in parallel
  - OpenGL calls user-specified fragment shader: Shading and lighting calculations
  - OpenGL handles z-buffer depth test unless overwritten
- Modern OpenGL is "lite" basically just a rasterizer
  - "Real" action in user-defined vertex, fragment shaders

## Shading Languages

- Vertex / Fragment shading described by small program
- Written in language similar to C but with restrictions
- Long history. Cook's paper on Shade Trees, Renderman for offline rendering
- Stanford Real-Time Shading Language, work at SGI
- Cg from NVIDIA, HLSL
- GLSL directly compatible with OpenGL 2.0 (So, you can just read the OpenGL Red Book to get started)

## Shader Setup

- Initializing (shader itself discussed later)

1. Create shader (Vertex and Fragment)
2. Compile shader
3. Attach shader to program
4. Link program
5. Use program

- Shader source is just sequence of strings
- Similar steps to compile a normal program

## Shader Initialization Code

```
GLuint initshaders (GLenum type, const char *filename) {
  // Using GLSL shaders, OpenGL book, page 679
  GLuint shader = glCreateShader(type) ;
  GLint compiled ;
  string str = textFileRead (filename) ;
  GLchar * cstr = new GLchar[str.size()+1] ;
  const GLchar * cstr2 = cstr ; // Weirdness to get a const char
  strcpy(cstr,str.c_str()) ;
  glShaderSource (shader, 1, &cstr2, NULL) ;
  glCompileShader (shader) ;
  glGetShaderiv (shader, GL_COMPILE_STATUS, &compiled) ;
  if (!compiled) {
    shadererrors (shader) ;
    throw 3 ;
  }
  return shader ;
}
```

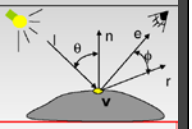
## Linking Shader Program

```
GLuint initprogram (GLuint vertexshader, GLuint fragmentshader)
{
  GLuint program = glCreateProgram() ;
  GLint linked ;
  glAttachShader(program, vertexshader) ;
  glAttachShader(program, fragmentshader) ;
  glLinkProgram(program) ;
  glGetProgramiv(program, GL_LINK_STATUS, &linked) ;
  if (linked) glUseProgram(program) ;
  else {
    programerrors(program) ;
    throw 4 ;
  }
  return program ;
}
```

## Phong Shader: Vertex

**This Shader Does**

- Gives eye space location for v
- Transform Surface Normal
- Transform Vertex Location




```
varying vec3 N;
varying vec3 v;

void main(void)
{
  v = vec3(gl_ModelViewMatrix * gl_Vertex);
  N = normalize(gl_NormalMatrix * gl_Normal);

  gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```

Created For Use Within Frag Shader

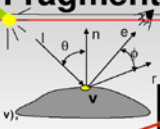
(Update OpenGL Built-in Variable for Vertex Position)

Cliff Lindsay web.cs.wpi.edu/~rich/courses/imgd4000-d09/lectures/gpu.pdf

## Phong Shader: Fragment

```
varying vec3 N;
varying vec3 v;

void main (void)
{
  // we are in Eye Coordinates, so EyePos is (0,0,0)
  vec3 L = normalize(gl_LightSource[0].position.xyz - v);
  vec3 E = normalize(-v);
  vec3 R = normalize(-reflect(L,N));

  //calculate Ambient Term:
  vec4 Iamb = gl_FrontLightProduct[0].ambient;

  //calculate Diffuse Term:
  vec4 Idiff = gl_FrontLightProduct[0].diffuse * max(dot(N,L), 0.0);

  // calculate Specular Term:
  vec4 Ispec = gl_FrontLightProduct[0].specular
          * pow(max(dot(R,E),0.0), gl_FrontMaterial.shininess);

  // write Total Color:
  gl_FragColor = gl_FrontLightModelProduct.sceneColor + Iamb + Idiff + Ispec;
}
```

Passed in From VS




Cliff Lindsay web.cs.wpi.edu/~rich/courses/imgd4000-d09/lectures/gpu.pdf

## Fragment Shader Compute Lighting

```
vec4 ComputeLight (const in vec3 direction, const in vec4
   lightcolor, const in vec3 normal, const in vec3 halfvec, const
   in vec4 mydiffuse, const in vec4 myspecular, const in float
   myshininess) {

     float nDotL = dot(normal, direction)  ;
     vec4 lambert = mydiffuse * lightcolor * max (nDotL, 0.0) ;

     float nDotH = dot(normal, halfvec) ;
     vec4 phong = myspecular * lightcolor * pow (max(nDotH, 0.0),
   myshininess) ;

     vec4 retval = lambert + phong ;
     return retval ;
}
```