GLSL Introduction

Fu-Chung Huang

Thanks for materials from many other people
Shader Languages

- Currently 3 major shader languages
  - Cg (Nvidia)
  - HLSL (Microsoft)
    - Derived from Cg
  - GLSL (OpenGL)
- Main influences are
  - C language
  - pre-existing Shader languages developed in university and industry

Source: [http://http.developer.nvidia.com/CgTutorial/cg_tutorial.chapter01.html](http://http.developer.nvidia.com/CgTutorial/cg_tutorial.chapter01.html) (Modified with information on HLSL and GLSL)
Fixed Functionality

Vertex Shader

Frag. Shader
Shader Initialization
Qualifiers in pipeline

Attribute

- `glColor()`
- `gINormal()`
- `gITexCoord()`
- `gIVertex()`

Vertex Shader

- `uniform`:
  - Lightings, Xforms, etc.
- `attribute`:
  - (x,y,z)

Rasterizer

- `varying`:
  - Interpolated Normals, TexCoords, Colors, 3D position, etc.

Fragment Shader

- `varying`:
  - Color & depth

Buffer Op...
Really Complicated Pipeline

App. Memory

- Geometry
  - Per-Vertex Operations
  - Primitive Assembly
  - Clip Project Viewport Cull
  - (Geometry)
  - Rasterize
  - (Pixels)

Pixel Unpack

Pixel Transfer

Fragment Processing

Texture Memory

Frame Buffer

Per Fragment Operations

Frame Buffer Operations

Read Control

- Pixel Groups
- Vertices
- Fragments
- Textures
Simplified Data Flow

- **Attribute**
  - Color
  - normal
  - Texture coord
  - 3D position

- **Uniform**
  - Lightings, Xforms, etc.

- **Vertex Shader**
  - varying

- **Rasterizer**
  - Interpolate everything
  - varying

- **Fragment Shader**
  - Color & depth
  - To the screen
Vertex Shader

- Vertex Xform
- Normal Xform
- Text Coord
- Per-vertex lighting
Vertex Shader

Built-in attribute variables
- gl_Color
- gl_Normal
- gl_Vertex
- gl_MultiTexCoord0...
- etc...

User-defined attribute variables
- Velocity
- Elevation
- Tangent
- etc.

Built-in uniform variables
- gl_ModelViewMatrix
- gl_FrontMaterial
- gl_LightSource[0...]
- gl_Fog
- etc.

User-defined uniform variables
- EyePos
- LightPosition
- etc.

User-defined varying variables
- Normal
- RefractionIndex
- Density
- etc.

Built-in varying variables
- gl_FrontColor
- gl_BackColor
- gl_FogFragCoord
- gl_TexCoord[0...]
- etc.

Texture Maps

Special output variables
- gl_Position
- gl_PointSize
- gl_ClipVertex
Fragment (pixel) Shader

- Interpolated
- Texture access
- Applications
  - Texture
  - Fog
  - Color sum
**Fragment Shader**

### Built-in varying variables
- gl_Color
- gl_SecondaryColor
- gl_TexCoord[0...]
- gl_FogFragCoord etc...

### Special input variables
- gl_FragCoord
- gl_FrontFacing

### User-defined attribute variables
- Normal
- RefractionIndex
- Density etc...

### User-defined uniform variables
- EyePos, LightPosition, etc...

### Built-in uniform variables
- gl_ModelViewMatrix, gl_FrontMaterial, gl_LightSource[0...], gl_Fog, etc...

### Special output variables
- gl_FragColor
- gl_FracDepth

---

**Texture Maps**

**Fragment processor**

GLSL Language Definition

- **Data Type Description**
  - `int` Integer
  - `float` Floating-point
  - `bool` Boolean (*true* or *false*).
  - `vec2` Vector with two floats.
  - `vec3` Vector with three floats.
  - `vec4` Vector with four floats.
  - `mat2` 2x2 floating-point matrix.
  - `mat3` 3x3 floating-point matrix.
  - `mat4` 4x4 floating-point matrix.
Vector

• Vector is like a class
• You can use following to access
  – .r .g .b .a
  – .x .y .z .w
  – .s .t .p .q
• Example:
  
  ```cpp
  vec4 color;
  color.rgb = vec3(1.0, 1.0, 0.0);  color.a = 0.5
  color = vec4(1.0, 1.0, 0.0, 0.5);
  color.xy = vec2(1.0, 1.0);
  color.zw = vec2(0.0, 0.5);
  ```
GLSL Variable Qualifiers

- Qualifiers give a special meaning to the variable. In GLSL the following qualifiers are available:
  - `const` - the declaration is of a compile time constant
  - `uniform` – (used both in vertex/fragment shaders, read-only in both) global variables that may change per primitive (may not be set inside glBegin,/glEnd)
  - `varying` - used for interpolated data between a vertex shader and a fragment shader. Available for writing in the vertex shader, and read-only in a fragment shader.
  - `attribute` – (only used in vertex shaders, and read-only in shader) global variables that may change per vertex, that are passed from the OpenGL application to vertex shaders.
Vertex Shader Code Example

```glsl
varying vec3 normal, lightDir, eyeDir; //output

void main()
{
    // Calculate position for lighting
    vec3 vVertex = vec3(gl_ModelViewMatrix * gl_Vertex);

    normal = gl_NormalMatrix * gl_Normal;   //or use ModelViewInverseTranspose

    lightDir = vec3(gl_LightSource[0].position.xyz - vVertex);
    eyeDir  = -vVertex;

    // Calculate position for real projection(camera)
    gl_Position = projection_matrix * modelview_matrix * vec4(gl_Vertex, 1.0);
}
```
Fragment Shader Code Example

```glsl
varying vec3 normal, lightDir, eyeDir;

void main (void)
{
    vec4 final_color = (gl_LightSource[0].ambient * gl_FrontMaterial.ambient); //first ambient term

    vec3 N = normalize(normal); //remember to normalize every direction vector
    vec3 L = normalize(lightDir);

    float lambertTerm = dot(N, L); //cosine term in the diffuse component

    if(lambertTerm > 0.0)
    {
        final_color += gl_LightSource[0].diffuse * gl_FrontMaterial.diffuse * lambertTerm;

        //Finally specular term
        vec3 E = normalize(eyeDir);
        vec3 R = reflect(-L, N);
        float specular = pow( max(dot(R, E), 0.0), gl_FrontMaterial.shininess );
        final_color += gl_LightSource[0].specular * gl_FrontMaterial.specular * specular;
    }

    gl_FragColor = final_color;
}
```
Vertex vs. Fragment Shader

Smooth Shading

per vertex lighting

Phong Shading

per fragment lighting
Result

OpenGL Gouraud Shading

GLSL Phong Shading
GLSL Statements

• Control Flow Statements: pretty much the same as in C.
• HIGHLY HARDWARE DEPENDENT!!

```cpp
if (bool expression)
    ...
else
    ...

for (initialization; bool expression; loop expression)
    ...

while (bool expression)
    ...

do
    ...
while (bool expression)
```

Note: only “if” are available on most current hardware
Fragment Shader Applications

smooth shading

environment mapping

bump mapping
Bump Mapping

- Perturb normal for each fragment
- Store perturbation as textures