CS248 REVIEW SESSION

SHADERS AND FRAMEBUFFER OBJECTS

02/17/12

WHAT IS A SHADER?

- It's a small script-like program that gets compiled to run on your GPU.

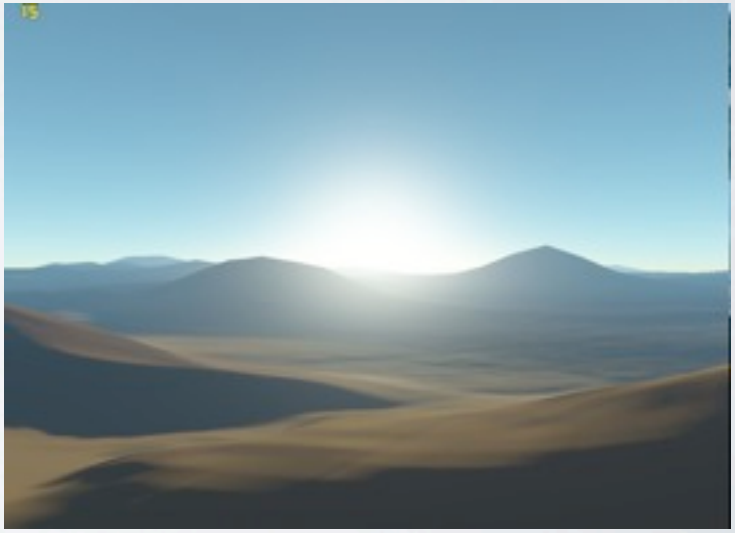
```
uniform vec3 v3CameraPos;        // The camera's current position
uniform vec3 v3LightPos;         // The direction vector to the light source
uniform float fOuterRadius;      // The outer (atmosphere) radius
uniform float fOuterRadius2;     // fOuterRadius^2
uniform float fInnerRadius;

uniform vec3 v3InvWavelength;    // 1 / pow(wavelength, 4) for the red, green, and blue
channels

uniform float fScale;            // 1 / (fOuterRadius - fInnerRadius)
uniform float fScaleDepth;       // The scale depth (i.e. the altitude at which the
atmosphere's average density is found)
uniform float fScaleOverScaleDepth; // fScale / fScaleDepth

uniform float fKr4PI;            // Kr * 4 * PI
uniform float fKm4PI;            // Km * 4 * PI

uniform float fSamples;
uniform float nSamples;

varying vec3 v3Direction;
```

WHY SHADERS?

- Programs used to use the fixed-function pipeline (basically, what you implemented for assignment 2)
- But this is very limiting. You are stuck with whatever made it into the OpenGL API at the last committee meeting.
- Shaders let you rewrite part of the pipeline.

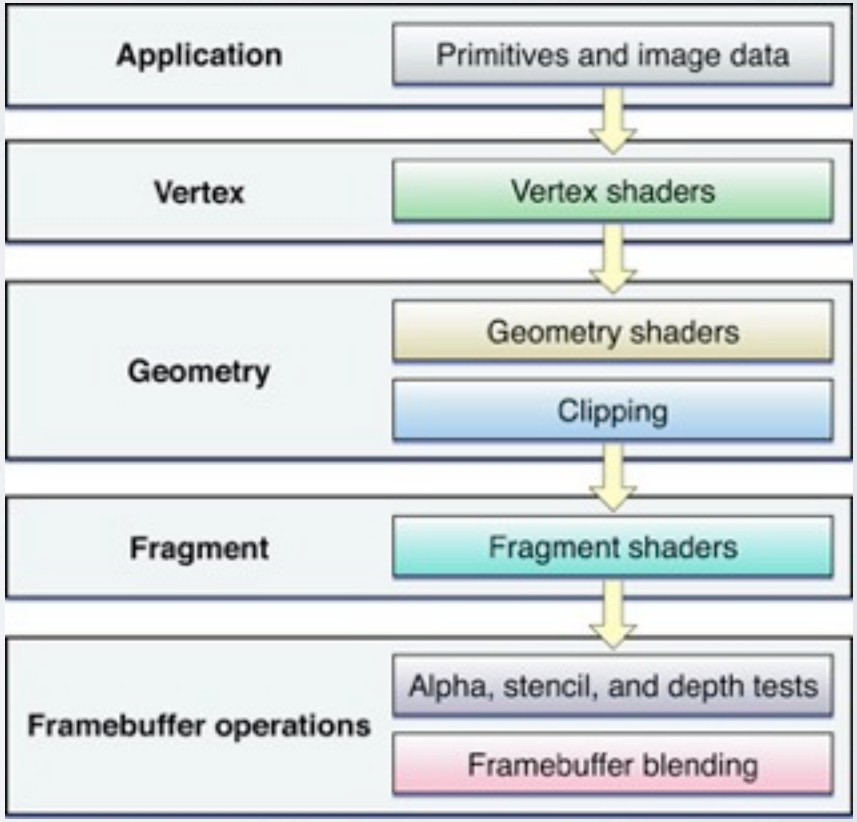
WHAT CAN WE DO WITH SHADERS?

- Shadow mapping
- Normal mapping
- Atmospheric scattering
- And much more...



HOW SHADERS WORK

- Vertex shader: Transforms vertices
- Geometry shader: Adds or filters extra vertices
- Fragment shader: Computes the final color of the pixel



developer.apple.com

CREATING AND COMPILING A SHADER

- glCreateShader
- glShaderSource
- glCompileShader
- glCreateProgram
- glAttachShader
- glLinkProgram

http://www.lighthouse3d.com/opengl/glsl/index.php?ogloverview

OPENGL SHADER LANGUAGE (GLSL)

- Looks very much like C, but more powerful
- Simple vertex shader:

```
void main() {
  gl_Position = gl_ModelViewMatrix
    * gl_ProjectionMatrix
    * gl_Vertex;
}
```

PRIMITIVE TYPES

- Matrices: mat4, mat3
- Vectors: vec2, vec3, vec4
- Scalars: float, int
- Texture samplers: sampler2D, samplerCube

USEFUL BUILT-IN OPERATIONS

- normalize(vec)
- length(vec)
- reflect(vec, vec)
- pow, max, sin, cos, etc.
- dot(vec, vec)
- Transforms: mat * vec
- Overloaded operators: +, -, *
- Swizzling: vec.xyz = vec.zyx
- tranpose(mat)

VERTEX SHADER

- **Input** comes from the client program, through "uniform variables" and "attributes"
  - uniform: Constant for all vertices
  - attribute: Some part of a vertex (normal, position, tangent, color, etc.)
- **Output** goes to the rasterizer, through "varying variables"
  - varying: Automatically interpolated values
  - gl_Position: special output, must be written to tell OpenGL the vertex position

SIMPLE VERTEX SHADER

```
attribute vec3 positionIn;

attribute vec3 normalIn;



varying vec3 normal;



void main() {

   gl_Postion = gl_ModelViewMatrix
      * gl_ProjectionMatrix
      * vec4(positionIn, 1)

   normal = gl_NormalMatrix
      * normalIn;

}
```

FRAGMENT SHADER

- **Input** comes from the rasterizer (post-interpolation) through **varying** variables

- **Output** goes to the framebuffer and alpha-blending through "gl_FragColor"

SIMPLE FRAGMENT SHADER

```
varying vec3 normal;



void main() {
    vec3 N = normalize(normal);
    vec3 L = normalize(gl_LightSource[0].xyz);

    gl_FragColor = dot(N, L);

}
```

BUILT-IN VARIABLES (VERTEX SHADER)

- gl_Vertex: Vertex position from glVertex*()
- gl_Normal: Normal from glNormal*()
- gl_ModelViewMatrix
- gl_ProjectionMatrix
- gl_NormalMatrix
- gl_LightSource[i]: Light from glLight*()
- gl_Material: Material from glMaterial*()
- gl_Position: For writing the vertex position
- More

BUILT IN VARIABLES (FRAGMENT SHADER)

- gl_LightSource[i]: Light from glLight*()
- gl_Material: Material from glMaterial*()
- gl_FragDepth: Depth of the fragment
- gl_FragColor: Write the final fragment color
- More

BUILT-INS VS. ATTRIBUTES AND UNIFORMS

- Your choice. You don't have to use the built-in values.

- You may want to use your own variable names, or you may have special shader inputs.

MORE SHADER API CALLS

- GLint glGetAttributeLocation(shader, string)
  - Get a handle to an attribute variable
- GLint glGetUniformLocation(shader, string)
  - Get a handle to a uniform variable
- glUniform*(handle, ...)
  - Set a uniform variable
- glEnableVertexAttribArray(handle)
  - Enable an attribute (e.g., the tangent vector)
- glVertexAttribPointer(handle, ...)
  - Pass a pointer to an array of data that to use for the attribute. One data element per vertex

SHADER DEMO

WHY TEXTURES ?

- Shadows
- Reflections
- Ambient occlusion
- Color bleeding
- Depth of field
- Motion blur
- Global Illumination...

Need **global information** !

FRAMEBUFFER OBJECTS

- Control where the rendered pixels go.
- Don't always want them to go to the screen.
- Examples:
  - Shadow mapping: depth goes into a texture
  - Mirrors: entire scene goes into a texture
  - Environment mapping: scene render 6x goes into 6 textures
  - Deferred rendering: stages go to textures

FRAMEBUFFER OVERVIEW

- The framebuffer is considered an "object"
- You can attach different layers to it
- Including:
  - Color render buffer (or texture)
  - Depth render buffer (or texture)
  - Stencil render buffer (or texture)

DEPTH RENDER TARGET

- Step 1: Create framebuffer
  - glGenFramebuffers
- Step 2: Create a blank texture
  - glGenTextures
  - glTexImage2D
- Step 3: Attach texture
  - glFramebufferTexture2d
- Step 4: Disable color buffer
  - glDrawBuffer(GL_NONE)
- Step 5: Check & render
  - glCheckFramebufferStatus
  - glBindFramebuffer

COLOR RENDER TARGET

- Step 1: Create framebuffer
- Step 2: Create a blank texture
- Step 3: Attach texture
- Step 4: Create a depth buffer (!)
  - glGenRenderbuffers
  - glRenderbufferStorage
- Step 5: Attach depth buffer
  - glFramebufferRenderbuffer
- Step 6: Check & render
  - glCheckFramebufferStatus
  - glBindFramebuffer

IS THAT REALLY IT?

- Many more details in the demo.

- Setting up the framebuffer is quite complicated

- Suggestion: Wrap ALL of your OpenGL calls with this macro

```
#define GL_CHECK(x) {\
  (x);\
  GLenum error = glGetError();\
  if (GL_NO_ERROR != error) {\
    printf("%s", gluErrorString(error));\
  }\
}
```

- For example:

```
GL_CHECK(glFramebufferRenderbuffer(...));
```

EXAMPLE :
DEFERRED RENDERING IN KILLZONE 2

SLIDES FROM MICHAL VALIENT, SENIOR PROGRAMMER, GUERILLA

http://www.guerrilla-games.com/publications/dr_kz2_rsx_dev07.pdf

THANK YOU !

GOOD LUCK WITH ASSIGNMENT 3