OVERVIEW

- Announcements
- Texture Mapping
- GLSL Shader Language
ANNOUNCEMENTS

- Project 2 is Due Today.
- Project 3 also goes out Today.
OVERVIEW

- Announcements
- Texture Mapping
- GLSL Shader Language
Texture Mapping
What is a Texture?

- A texture is just a bitmap image
- 2D array – texture[height][width][4]
- Pixels of the texture are called texels
- Texture coordinates are in 2D, in the range [0,1]
TEXTURE MAPPING

(0,0,0) -- (10,0,0) -- (12,5,0) -- (2,5,0) -- (0,0,0)

(0,0) -- (1,0) -- (1,1) -- (0,1)
TEXTURE MAPPING IN OPENGL: RENDERING

glEnable(GL_TEXTURE_2D);
glBegin(GL_QUADS);
    glTexCoord2f(0.0, 0.0); glVertex3f(0.0, 0.0, 0.0);
    glTexCoord2f(1.0, 0.0); glVertex3f(10.0, 0.0, 0.0);
    glTexCoord2f(1.0, 1.0); glVertex3f(12.0, 5.0, 0.0);
    glTexCoord2f(0.0, 1.0); glVertex3f(2.0, 5.0, 0.0);
glEnd();
glDisable(GL_TEXTURE_2D);
TEXTURE MAPPING IN OpenGL

/* Specifies the texture to be used */
void glTexImage2D(GLenum target, GLint level, 
    GLint internalFormat, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const GLvoid *texture);
TEXTURE MAPPING IN OPENGL: INITIALIZE

/* loads image */
size_t brick_width = 32, brick_height = 32;
unsigned char *brick_texture = load_image(...);

/* Generates and binds a texture object */
GLuint texture_id;
glGenTexture(1, &texture_id);
glBindTexture(GL_TEXTURE_2D, texture_id);

/* Defines the texture */
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA,
            brick_width, brick_height, 0, GL_RGBA,
            GL_UNSIGNED_BYTE, brick_texture);
**Texture Object**

- A texture object stores texture data in OpenGL.
- Can easily switch between different texture images without reloading the images, by binding different texture objects.
- Big performance gain

```c
/* For Texture Object */
void glGenTextures(GLsizei n, GLuint *textureNames);
void glBindTexture(GLenum target, GLuint textureName);
```
**Color Blending**

- Final pixel color = f(texture color, object color)
- GL_REPLACE – use texture color
- GL_BLEND – linear combination of 2 colors

- Eg. `glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);`
INTERPOLATING COLOR

- For a given texture coordinate \((s, t)\), look into the texture image to get color.

- What if the \((s, t)\) does not correspond to a pixel in texture image?
**INTERPOLATING COLOR**

- For a given texture coordinate \((s, t)\), look into the texture image to get color.

- What if the \((s, t)\) does not correspond to a pixel in texture image?

  - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);`
  - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);`
USEFUL RESOURCES: TEXTURE MAPPING

- OpenGL Programming Guide, Version 1.1
  http://www.glprogramming.com/red/chapter09.html
OVERVIEW

- Announcements
- Texture Mapping
- GLSL Shader Language
**Motivation for Programmable Shaders**

- The GPU is basically a bunch of small processors.

- Back before shaders, the portion of the graphics pipeline that handled lighting and texturing was hardcoded into what we call the Fixed-Functionality Pipeline.
  - Must use Blinn-Phong shading, model view etc.

- But now, we can write our own shaders to change the way the pipeline works

- OpenGL Fixed-Functionality is now implemented on shaders
**Shader**

- A shader is a program that basically rewrites a portion of the graphics pipeline.
- Used in different places of graphics pipeline – vertex shader, fragment shader etc.
- Can be written in different languages
  - OpenGL’s GLSL
  - Microsoft HLSL
  - Nvidia’s Cg
OpenGL Pipeline
OpenGL Pipeline (Simplified)

Vertex Data

- Vertex Operations
- Rasterization (Interpolation)
- Fragment Operations

Framebuffer
OpenGL Pipeline (Simplified)

Vertex Data

- Vertex Shader
- Rasterization (Interpolation)
- Fragment Shader

Framebuffer
OpenGL Pipeline (Simplified)

Vertex Data

- GL_MODELVIEW
- Lighting calculation

Vertex Shader ➔ Rasterization (Interpolation) ➔ Fragment Shader ➔ Framebuffer
OpenGL Pipeline (Simplified)

Vertex Data

- Interpolate into fragments

Vertex Shader → Rasterization (Interpolation) → Fragment Shader

- GL_MODELVIEW
- Lighting calculation

Framebuffer
OpenGL Pipeline (Simplified)

Vertex Data
- GL_MODELVIEW
- Lighting calculation

- Interpolate into fragments
- Texturing
- Fog calculation
- Blending etc

Framebuffer
PROJECTS

- Project 1 & Project 2 – use fixed functionality in OpenGL

- Project 3 – write your own shaders
SHADERS

- **Vertex Shader**
  - Operates on vertex data (normal, position, tex_coord)
  - One vertex at a time

- **Fragment Shader**
  - A fragment is the smallest unit being shaded
  - One fragment at a time
SHADER PROGRAMMING

- Syntax looks like C language
- int, bool, float etc
- vec2, vec3, mat3, mat4 etc
**Variable Types**

- **const**
- **attribute**
  - Written in OpenGL application, passed to vertex shader.
  - Read-only in vertex shader. Only accessible in vertex shader.
- **varying**
  - written in vertex shader, interpolated and read-only in fragment shader
- **uniform**
  - written in OpenGL application, passed to vertex and fragment shaders.
  - Read-only in both shaders
/*Sample vertex shader*/
attribute float shift; /*Attribute values passed from OpenGL program*/
varying vec3 norm;    /*Varying values written in vertex shader
                      and read only in fragment shader*/

void main(void){
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex * shift;
    norm = gl_Normal;
}
SAMPLE CODE: VERTEX SHADER

/*Sample vertex shader*/
attribute float shift; /*Attribute values passed from OpenGL program*/
varying vec3 norm;      /*Varying values written in vertex shader
                        and read only in fragment shader*/

void main(void){
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex * shift;
    norm = gl_Normal;
}
SAMPLE CODE: VERTEX SHADER

/*Sample vertex shader*/
attribute float shift; /*Attribute values passed from OpenGL program*/
varying vec3 norm;    /*Varying values written in vertex shader
                        and read only in fragment shader*/

void main(void)
{
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex * shift;
    norm = gl_Normal;
}
/*Sample vertex shader*/
attribute float shift; /*Attribute values passed from OpenGL program*/
varying vec3 norm; /*Varying values written in vertex shader and read only in fragment shader*/
void main(void){
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex * shift;
    norm = gl_Normal;
}

/*Sample fragment shader*/
varying vec3 norm;
uniform vec3 color; /*Uniform values passed from OpenGL program and read only in fragment shader*/
const vec3 black = vec3(0.0, 0.0, 0.0);

void main(void){
    if(length(norm) >= 1)
        glFragColor = vec4(color, 1.0);
    else
        glFragColor = vec4(black, 1.0);
}

/*Note: This is just a demo, and did nothing useful*/
SAMPLE CODE: VERTEX AND FRAGMENT SHADERS

```cpp
/*Sample vertex shader*/
attribute float shift; /*Attribute values passed from OpenGL program*/
varying vec3 norm;    /*Varying values written in vertex shader
                     and read only in fragment shader*/

void main(void){
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex * shift;
    norm = gl_Normal;
}

/*Sample fragment shader*/
varying vec3 norm;
uniform vec3 color;    /*Uniform values passed from OpenGL program and
                       read only in fragment shader*/
const vec3 black = vec3(0.0, 0.0, 0.0);

void main(void){
    if(length(norm) >= 1)
        glFragColor = vec4(color, 1.0);
    else
        glFragColor = vec4(black, 1.0);
}

/*Note: This is just a demo, and did nothing useful*/
```
/*Sample vertex shader*/
attribute float shift; /*Attribute values passed from OpenGL program*/
varying vec3 norm;    /*Varying values written in vertex shader
                     and read only in fragment shader*/
void main(void){
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex * shift;
    norm = gl_Normal;
}

/*Sample fragment shader*/
varying vec3 norm;
uniform vec3 color; /*Uniform values passed from OpenGL program and
                    read only in fragment shader*/
const vec3 black = vec3(0.0, 0.0, 0.0);

void main(void){
    if(length(norm) >= 1)
        glFragColor = vec4(color, 1.0);
    else
        glFragColor = vec4(black, 1.0);
}

/*Note: This is just a demo, and did nothing useful*/
/*OpenGL Program*/

void somethingOpenGL() {
    /*Creates an empty program object and returns its handle*/
    GLhandleARB shader = glCreateProgramObjectARB();

    /*Create shader - a helper function in P3*/
    create_shader(shader, "shaders/vert.glsl", "shaders/frag.glsl");

    /*Get locations for the uniform variables in shader program*/
    int color_loc = glGetUniformLocationARB(shader, "color");

    /*Bind the shader with the vertex shaders and fragment shaders written*/
    glUseProgramObjectARB(shader);

    /*Pass in the uniform values to the shader*/
    glUniform3fARB(color_loc, 1.0, 1.0, 1.0);

    /*Unbind the shader*/
    glUseProgramObjectARB(0);
}
BUILT-IN VARIABLES & FUNCTIONS

- Built-in variables:
  - Vertex Shader – gl_Normal, gl_Vertex, gl_Color etc
  - Fragment Shader – gl_Color, gl_FragCoord etc

- Built-in functions:
  - Trigonometry – sin, cos, atan, radians etc
  - Geometry – length(vec), normalize(vec) etc

- Refer to OpenGL Shading Language (GLSL) Quick Reference Guide for a complete list (will be on website)
OpenGL Pipeline (Multi-pass)

Vertex Data

- Vertex Shader
- Rasterization (Interpolation)
- Fragment Shader

Framebuffer

- glReadPixels / glFramebufferTexture2D
- Rasterization (Interpolation)
- Fragment Shader

Framebuffer

Framebuffer
**OpenGL Pipeline (Multi-pass)**

**Vertex Data**

- Vertex Shader
- Rasterization (Interpolation)
- Fragment Shader

- OpenGL Fixed Functionality

- `glReadPixels`
- `glFramebufferTexture2D`

**Framebuffers**
MULTI-PASS RENDERING

- There are multiple ways to copy data in Framebuffer to textures.

  - `glReadPixels(GLint x, GLint y, GLsizei width, GLsizei height, GLenum format, GLenum type, GLvoid *data)`

  - Use framebuffer objects
**FrameBuffer Objects**

- Used to capture images that would normally be drawn to the screen.
- Faster and more efficient compared to `glReadPixels`
SAMPLE CODE: FRAMEBUFFER

GLuint fbo;
GLuint colorTexID, depthTexID;
/* Bind colorTexID and depthTexID to texture objects */

/* Create Framebuffer object */
glGenFramebuffers(1, &fbo);

/* Bind the framebuffer object to the textures */
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT,
                         GL_TEXTURE_RECTANGLE_ARB, colorTexID, 0);
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_DEPTH_ATTACHMENT_EXT,
                         GL_TEXTURE_RECTANGLE_ARB, depthTexID, 0);

/* Render scene using fixed functionality or shaders */
render();
/* Unbind the fbo */
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
USEFUL RESOURCES: GLSL

- GLSL Tutorials: http://www.lighthouse3d.com/opengl/gls/
  http://www.opengl.org/sdk/docs/tutorials/TyphoonLabs/