CS770/870 Spring 2017 Miscellaneous OpenGL Items

Most aimed at P1 assignment

Enabling the Depth Buffer

- Why do we need it?
  - Otherwise, apparent depth depends on order drawn
- One-time necessary initialization items

```
glEnable(GL_DEPTH_TEST);
```

- Done once per frame

```
glClear( GL_COLOR_BUFFER_BIT |
         GL_DEPTH_BUFFER_BIT )
```

- Other functions that still work

```
glDepthFunc(), glDepthMask(), glDepthRange()…
```

Shading Items

- Apply at start of rendering
  - Transform (possibly its own class)
  - Material properties (probably its own class)
- Apply potentially per-vertex
  - `glShadeModel(GL_SMOOTH)` *or* `glShadeModel(GL_FLAT)`
  - Need normal at each vertex

Old Camera Functions Still Live

- Both JOML and unhmat.h support old functions
- Setting up the camera
  - Orthographic or perspective projection?
    - `ortho(left, right, bottom, top, near, far)`
    - `perspective(angleY, aspect, near, far)`
    - `etc.`
  - Eye Coord System
    -
```
lookAt( eyeX, eyeY, eyeZ,
        targetX, targetY, targetZ,
        upX, upY, upZ)
```

C++ note: the original *unhmat.h* capitalized all its method names.
A new version is available on web that uses "standard" lower case naming.

GLSL has NO lights

- Programmer must do it all;
  - don't need to follow traditional model
  - but most come close; it's well known and useful.
- Let's look at part of a shader from
  - Bailey/Cunningham, Graphics Shaders, *CRC Press*

Shader Program Uniform Variables for one light

```
// position in eye coordinates
uniform float uLightX, uLightY, uLightZ;
// ambient, diffuse, specular of this light
uniform float uKa, uKd, uKs;
// light color for ambient and diffuse
uniform vec4  uColor;
// light color for specular
uniform vec4  uSpecularColor;
// shininess
uniform float uShininess;
```

Shader variables for vertex

```
out vec3 vN;   // surface normal vector
out vec3 vL;   // vector from point to light
out vec3 vE;   // vector from point to eye
out vec3 colV; // composite color for vertex

uniform  mat4 MV;  // model view matrix
uniform  mat4 P;   // Projection matrix

in  vec4 pos; // vertex location
in  vec4 normal; // vertex normal

<and more>
```


Some Shader Lighting Code

```
// need vertex and light in same coord system
vec4 eyeVertexPos = uModelViewMatrix * aVertex;
vec3 eyeLightPos = vec3( uLightX, uLightY, uLightZ );
// need unit normals
vNs = normalize( uNormalMatrix * aNormal );
// vector from vertex pos to light
vLs = eyeLightPos - eyeVertexPos.xyz;
// vector from vertex pos to eye (origin)
vEs = vec3( 0., 0., 0. ) - eyeVertexPos.xyz;

// make these unit normals
vec3 Normal = normalize( vNs ); // already done?
vec3 Light = normalize( vLs );
vec3 Eye = normalize( vEs );
```

Now some color

```
// need vertex and light in same coord system
vec4 ambient = uKa * uColor; // scalar * vector

// 0 if light can't see point
float d = max( dot( Normal, Light ), 0. );
// need unit normals
vec4 diffuse = uKd * d * uColor;

float s = 0.;
// only do it if light can see point
if ( dot( Normal, Light ) ) > 0. )
{
    // do specular calculation
}
colV = ambient.rgb + diffuse.rgb + specular.rgb;
gl_Position = uProjection * uMV * pos;
```


BoxDemo vertex shader

```
vec3 lightedColor( vec3 oCol, vec3 vNorm )
{
    vec3 lDir = normalize( vec3( 2, 3, 4 ));
    vec3 lCol = normalize( vec3( 1, 1, 1 ));
    vec3 vNorm = vec3( normalize( vNorm ));
    float cosLN = dot( lightDir, vNorm )
    vec3 color = ka * oCol + kd*oCol * lCol*cosLN;
    return color;
}
//main
vec3 color3 = vec3( uColor.r, uColor.g, uColor.b );
vec4 vPos4 = vec4( vPosition, 1 );
gl_Position = projXview * uModel * vPos4;
color = vec4( lightedColor( color3, vNormal ), 1 );
```

GLSL Buffer Management from *BoxDemo*

- Creating/using buffer for coordinate data
  - CPU creates/fills buffer

```
this.coordVBO = glGenBuffers(); // get id
// "bind" this buffer: subsequent buffer msgs go to it
glBindBuffer( GL_ARRAY_BUFFER, this.coordVBO );
// send data in the CPU FloatBuffer, coordBuf, to GPU
glBufferData( GL_ARRAY_BUFFER, coordBuff, GL_STATIC_DRAW );
```

  - CPU links bound buffer with metadata, VAO

```
vaoPos = glGetAttribLocation( shader, "vPos" );
glEnableVertexAttribArray( vaoPos );
glVertexAttribPointer(vaoPos,coordSize,GL_FLOAT,false,0,0L);
```

Shader variable

floats/coord

stride

offset

- Also has normals, you need to add color

Face Colors

- P1 includes face-coloring of objects
  - every face is assigned a color
    - use setColor(i, …);
    - the first n colors are used for n faces
      - 6 faces for a box, 4 for a pyramid
  - every face has 2 triangles and 6 vertex coords
  - need to have a color associated with each vertex in coord buffer
  - distinguish between *box vertex* (there are 8) and *coordinate vertex* (there are 36: 12 triangles)

Face Colors (cont)

- Tasks
  - for each coordinate vertex (36)
    - identify which face *that* vertex belongs to
    - store *that* face's color in the position in the color buffer that corresponds to the coordinate vertex position in its buffer

A Sample Program Instance

Might create classes with relationships such as this