Scott Schumacher

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* glGenVertexArrays(GLsizei *n*, GLuint \**arrays*);
  + r 17-18
  + **what occurs**: returns *n* unused names for use as vertex-array objects in the array *arrays.* The names returned are marked as used for the purposes of allocating additional buffer objects, and initialized with values representing the default state of the collection ofunitialized vertex arrays.
  + **when used:** in the init function

// Create a vertex array object for each model to be displayed

glGenVertexArrays(numModels, vao); // SSM

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* glBindVertexArray(Gluint *array*);
  + r 17-18
  + **what occurs:** 
    - when using the value array that is other than zero and was returned from glGenVertexArrays(), a new vertex-array object is created and assigned that name.
    - when binding to a previously created vertex-array object, that vertex array object becomes active, which additionally affects the vertex array state stored in the object.
    - when binding to an array value of zero, OpenGL stops using application-allocated vertex-array objects and returns to the default state for vertex arrays.
  + **when used:**
    - in the init function, for the first reason listed above
    - subsequently, when making a different object active (as we used in program 4)

// loop- bind vao for each file

for (int m = 0; m < numModels; m++)

{

glBindVertexArray(vao[m]);

instModels[m].load(program); //program returned by InitShader.cpp

}

// binding an already bound vao makes it the active object

glBindVertexArray(vao[modelChoice]);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* GLuint Loc = glGetUniformLocation(GLuint *program*, const char\* *name*);
  + r 47, 569, 744, 784
  + **what occurs:** returns the index of the unifiorm variable **name** associated with the shader program. All that’s generated is the index for a name; the object with that name may not even exist.
    - name is a null-terminated character string with no spaces
    - a value of -1 is returned if name doesn’t correspond to a uniform variable in the active shader program, or if a reserved shader variable name (gl\_ prefix) is specified
    - **there’s more**
  + **when used:**
  + GLuint Loc

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* GLuint vPosition = glGetAttribLocation(GLuint *program*, GLchar \**name*);
  + r 129, **739**, **784**
  + **what occurs:**
  + **when used:** in load

GLuint vPosition = glGetAttribLocation(program, "vPosition");

glEnableVertexAttribArray(vPosition);

glVertexAttribPointer(vPosition, 4, GL\_FLOAT, GL\_FALSE, 0,

BUFFER\_OFFSET(0)); // offset 0: vertex position data in first half of buffer

GLuint vColor = glGetAttribLocation(program, "vColor");

glEnableVertexAttribArray(vColor);

glVertexAttribPointer(vColor, 4, GL\_FLOAT, GL\_FALSE, 0,

BUFFER\_OFFSET(size)); // offset size: color data in second half of buffer

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* glUniformMatrix4fv(ModelViewLoc, 1, GL\_TRUE, model\_view);
  + r 554, **openGL.org**
  + what are the params, etc.
    - ModelViewLoc: GLint *location*
    - 1: GLsizei *count*
    - GL\_TRUE: GLboolean *transpose*
    - model\_view: constGLfloat *\*value*
  + **what occurs:**
  + **when used:**
    - in reshape, with projection

projection = Perspective(fovy, aspectRatio, zNear, zFar); // SSZ

glUniformMatrix4fv(ProjectionLoc, 1, GL\_TRUE, projection); // SSZ

* + - in display, with model\_view:

glUniformMatrix4fv(modelViewLoc, 1, GL\_TRUE, model\_view);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* glEnableVertexAttribArray(vPosition);
  + r **27-8**, 112-6, 129-30
  + **what occurs:** the array designated by the variable *index,* in this case vPosition, is enabled
  + **when used:** in load

GLuint vPosition = glGetAttribLocation(program, "vPosition");

glEnableVertexAttribArray(vPosition);

glVertexAttribPointer(vPosition, 4, GL\_FLOAT, GL\_FALSE, 0, BUFFER\_OFFSET(0));

// offset 0: vertex position data in first half of buffer

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* glVertexAttribPointer(vPosition, 4, GL\_FLOAT, GL\_FALSE, 0,BUFFER\_OFFSET(0));
  + r **26-7**, 30, 93, 108-13, 129, 130, 149
  + Specifies where the data values for shader attribute location *index* can be found.
  + (GLuint *index*, GLint *size*, GLenum *type*, GLboolean *normalized*, GLsizei *stride*, const GLvoid *\*pointer*)
    - index: location of vshader input variable
    - size: number of values for each vertex in the array
    - type:
    - normalized: are values constrained to (-1,1), that is, normalized?
    - stride: byte offset between data members; 0 means tightly packed
    - \*pointer: the location within the buffer where the data starts; buffer offset (0) means the data begins at the first byte (address zero) of the buffer.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* glDrawArrays(GL\_TRIANGLES, start, length);
  + r 11, **30**, 115-19, 128, 135, 487, 518, 539-40, 628, 639
  + glDrawArrays(GLenum *mode*, GLint *first*, GLsizei *count*);
  + what is start and length?
    - start- the first element from the bound vertex array
    - length- the number of vertices to be drawn
  + Is length the number of bytes or vertices or triangles?
    - Vertices
  + **what occurs:** sends vertex data to the openGL pipeline..30
  + **when used:** in draw

void Object::draw()

{

glDrawArrays(GL\_TRIANGLES, 0, numIndices);

}

https://robot.bolink.org/ebooks/Interactive%20Computer%20Graphics%20-%20A%20Top-Down%20Approach%206e%20By%20Edward%20Angel%20and%20Dave%20Shreiner%20(Pearson,%202012)%20BBS.pdf

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* LookAt();
  + A 26, **212-14**, 247-9
  + what are the params
    - eye, vec4 point- the location of the camera / eye
    - at, vec4 point- usually the center of the
    - up, vec4 vector- direction up; usally parallel to positive y-axis
  + what is returned
    - 4x4 matrix-

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* RotateX(theta);
  + what is the parameter: theta[Xaxis]—the angle of rotation
  + what is returned:

// SSX rotation matrix for x axis

mat4 RotX(GLfloat theta)

{

return mat4(1.0, 0.0, 0.0, 0.0,

0.0, cos(theta), sin(theta), 0.0,

0.0, -sin(theta), cos(theta), 0.0,

0.0, 0.0, 0.0, 1.0);

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);
  + 8, **28**-9, 147, 190
  + glClear(GLbitfield mask);
  + **what occurs:** clears the current buffers to their current clearing values; argument mask is a bitwise OR combination of two of the following three:
    - GL\_CLEAR\_BUFFER\_BIT
    - GL\_DEPTH\_BUFFER\_BIT
    - GL\_STENCIL\_BUFFER\_BIT
  + **when used:** in display, just before the call to draw

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

// send model\_view to the shaders as a uniform variable

glUniformMatrix4fv(modelViewLoc, 1, GL\_TRUE, model\_view);

instModels[modelChoice].draw(); // call to draw

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* Ortho() ;
  + what are the params
  + what is returned
  + what does this return value actually do?

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* Perspective();
  + what are the params
  + what is returned
  + what does this return value actually do?

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* glDrawArrays(GL\_TRIANGLES, start, length);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* glGenBuffers(1, &buffer);
  + **19**-20, 92-4, 356, 877-8
  + glGenBuffers(GLsizei *n*, GLuint *\*buffers*);
  + **what occurs:** returns *n* currently unused names for buffer objects in the array *buffers*. The names returned in buffers do not have to be a contiguous set of integers. The names ar marked as used for the purposes of allocating additional buffer objects, but only acquire a valid state when they’ve been bound. Zero is reserved and never returned by glGenBuffers().
  + **when used:** in Object::init

glGenBuffers(1, &buffer);

glBindBuffer(GL\_ARRAY\_BUFFER, buffer);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* glBindBuffer(GL\_ARRAY\_BUFFER, buffer);
  + 18-20, 63, 69, 94, 242, 891
  + glBindBuffer(GLenum *target*, GLuint *buffer*);
  + target must be set to one of these eight types:
    - GL\_ARRAY\_BUFFER
    - GL\_ELEMENT\_ARRAY\_BUFFER
    - GL\_PIXEL\_PACK\_BUFFER
    - GL\_PIXEL\_UNPACK\_BUFFER
    - GL\_COPY\_READ\_BUFFER
    - GL\_COPY\_WRITE\_BUFFER
    - GL\_TRANSFORM\_FEEDBACK\_BUFFER
    - GL\_UNIFORM\_BUFFER
  + *buffer* specifies the buffer object to be bound to.
  + **what occurs:** one of three things…
    - when using an buffer of an unsigned int other than zero for the first time, a new buffer object is created and assigned that name
    - when binding to a previously created buffer object, that buffer object becomes the active buffer object
    - when binding to a buffer value of zero, OpenGL stops using buffer objects for that *target*.
  + **when used:** in Object::init

glGenBuffers(1, &buffer);

glBindBuffer(GL\_ARRAY\_BUFFER, buffer);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* glBufferData(GL\_ARRAY\_BUFFER, Size+Size,  
  NULL, GL\_STATIC\_DRAW);glBufferSubData(GL\_ARRAY\_BUFFER, 0, Size, &vertexList[0]);
  + 11, **21-2**, 63, 69, 95, 97, 99-103, 108, 242-3, 357, 602
  + glBufferData(GLenum *target*, GLsizeiptr *size*, constGLvoid *\*data*, GLenum *usage*);
    - *size* is the amount of data required; generally the number of elements multiplied by their size in bytes, such as

int size = numIndices \* 16; // each vertex requires 16 bytes, and is listed repeatedly

* + - *data* is either a pointer to client memory that is used to initialize the buffer object or NULL. If a valid pointer is passed, size units of storeage are copied from the client to the server. If NULL is passed, *size* units of storage are reserved for use but are left uninitialized.
    - *usage* provides a hint as to how the data will be read and written after allocation.
      * GL\_STATIC\_DRAW
      * GL\_STATIC\_READ
      * GL\_STATIC\_COPY
      * GL\_STREAM\_DRAW
      * GL\_STREAM\_READ
      * GL\_STREAM\_COPY
      * GL\_DYNAMIC\_DRAW
      * GL\_ DYNAMIC \_READ
      * GL\_ DYNAMIC \_COPY
  + **what occurs:** Allocates *size* storage units (usually bytes ) of OpenGL server memory for storing data or indices. Any previous data associated with the currently bound object will be deleted.
  + *target* may be:
    - GL\_ARRAY\_BUFFER for vertex attribute data
    - GL\_ELEMENT\_ARRAY\_BUFFER for index data
    - GL\_PIXEL\_UNPACK\_BUFFER for pixel data being passed into OpenGL
    - GL\_PIXEL\_PACK\_BUFFER for pixel data being retrieved from OpenGL
    - For data being copied between buffers:
      * GL\_COPY\_READ\_BUFFER or
      * GL\_COPY\_WRITE\_BUFFER
    - GL\_TEXTURE\_BUFFER for texture data stored as a texture buffer
    - GL\_TRANSFORM\_FEEDBACK\_BUFFER fro results from executing a transform feedback shader
    - GL\_UNIFORM\_BUFFER for uniform variable values.
  + **when used:** in Object::load

//put color and vertex indices in same buffer

glBufferData(GL\_ARRAY\_BUFFER, size + size, NULL, GL\_STATIC\_DRAW);

glBufferSubData(GL\_ARRAY\_BUFFER, 0, size, &vertices[0]);

glBufferSubData(GL\_ARRAY\_BUFFER, size, size, &vertices[0]);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* glBufferSubData(GL\_ARRAY\_BUFFER, Size, Size, &normalList[0]);
  + 97-101, 103, 242, 602
  + glBufferSubData(GLenum *target*, GLintptr *offset*, GLsizeitptr *size*, constGLvoid *\*data*);
  + what occurs: replaces a subset of a buffer object’s data store with new data. The section of the buffer object bound to *target* starting at *offset* bytes is updated with the *size* bytes of data addressed by *data*.
  + where used: in Object::load. Below, the two calls to glBufferSubData address the first and second halves of GL\_ARRAY\_BUFFER; the same data goes in both halves, although the second half is used for vertex colors, which in this case are based on the vertex data.

//put color and vertex indices in same buffer

glBufferData(GL\_ARRAY\_BUFFER, size + size, NULL, GL\_STATIC\_DRAW);

glBufferSubData(GL\_ARRAY\_BUFFER, 0, size, &vertices[0]);

glBufferSubData(GL\_ARRAY\_BUFFER, size, size, &vertices[0]);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* Know the meaning of the code in the following vertex shader.
  + **How are the in variables fed.**  From opengl.org: User-defined inputs for vertex shaders are called vertex attributes. They are passed via [vertex arrays](https://www.opengl.org/wiki/Vertex_Specification) to the vertex shader, usually from data stored in [Buffer Objects](https://www.opengl.org/wiki/Buffer_Object)
  + **How are the uniform variable set.**  Uniform variables are set in the client program using, in the case of Projection, glUniformMatrix4fv(ProjectionLoc, 1, GL\_TRUE, projection);
  + **What do the commands do?**

#version 420 **version 4.2**

**//The in variables are fed via matrices from buffer objects; sent from client program with**

**//commands such as glVertexAttribPointer(etc);**

in vec4 vPosition;  
in vec4 vNormal;  
out vec4 color;

**// uniform variables are set in the client program with commands such as glUniformMatrix4fv(etc);**

**// they are uniform throughout the drawing of a mesh.**

uniform vec3 LAmbient, LDiffuse, LSpecular; **// three components of light intensity**  
uniform vec3 MAmbient, MDiffuse, MSpecular; **//three components of material reflectivity**

uniform mat4 ModelView; **//transformation matrices**uniform mat4 Projection;  
uniform vec4 LightPosition; **// location of the light source**  
uniform float Shininess; **// shininess coefficient; exponent for specular intensity calculation**

void main()  
{  
// Transform vertex position into eye coordinates  
vec3 pos = (ModelView \* vPosition).xyz; **// in eye coordinates, the origin is at the center of the camera lens**

**// point – point subtraction results in a vector of unit length pointing toward the light position**

vec3 L = normalize( LightPosition.xyz – pos );

**// vector of unit length pointing toward the camera position**

vec3 E = normalize( -pos );

**// the halfway vector is a unit length vector in the direction of L+E**

vec3 H = normalize( L + E );

**// Transform vertex normal into eye coordinates and normalize**  
vec3 N = normalize( ModelView\*vNormal).xyz;

// Compute terms in the illumination equation

**// the ambient light intensity times the material’s ambient reflectivity coefficient**

vec3 ambient = LAmbient\*MAmbient;

**// the magnitude of L dot N is the cosine of the angle between them**

float Kd = max( dot(L, N), 0.0 );

**// this is equivalent to the first term in the Phong light intensity equation**

vec3 diffuse = Kd\*LDiffuse\*MDiffuse;

**// this is equivalent to the [v dot r raised to the alpha power] term in the Phong equation**

float Ks = pow( max(dot(N, H), 0.0), Shininess );

**// the above term multiplied by the product of the specular intensity and the specular reflectivity**

vec3 specular = Ks \* LSpecular\*MSpecular;

**//if the magnitude of L dot N is negative, the specular component is zero. This can be visualized by imagining viewing a smooth lake surface at high noon; the sun is directly (90 degrees) overhead and your viewing angle is horizontal (0 degrees); the dot product would be zero, meaning that you wouldn’t see a specular reflection (glare spot) on the water**

if( dot(L, N) < 0.0 )

{  
specular = vec3(0.0, 0.0, 0.0);  
}

**//applies projection to eye coordinates**

gl\_Position = Projection \* ModelView \* vPosition;

**// color out is the sum of the ambient, diffuse and specular components calculated above, reconstituted as a vec4 point**

color = vec4(ambient + diffuse + specular,1.0);

}