Meshes and Primitives

CS GY-6533 / UY-4533

Quaternions

Problems with rotation

$$\begin{bmatrix} w \\ \hat{\mathbf{c}} \end{bmatrix},$$

$$\begin{bmatrix} \cos(\frac{\theta}{2}) \\ \sin(\frac{\theta}{2})\hat{\mathbf{k}} \end{bmatrix}.$$

$$\begin{bmatrix} -\cos(\frac{\theta}{2}) \\ -\sin(\frac{\theta}{2})\hat{\mathbf{k}} \end{bmatrix}.$$

Rotating by θ around axis k.

$$\begin{bmatrix} 1 \\ \hat{\mathbf{0}} \end{bmatrix}, \begin{bmatrix} -1 \\ \hat{\mathbf{0}} \end{bmatrix}$$

Identity quaternion

$$\begin{bmatrix} 0 \\ \hat{\mathbf{k}} \end{bmatrix}, \begin{bmatrix} 0 \\ -\hat{\mathbf{k}} \end{bmatrix}$$

180 degree rotation around axis k

Multiplication by scalar

$$\alpha \begin{bmatrix} w \\ \hat{\mathbf{c}} \end{bmatrix} = \begin{bmatrix} \alpha w \\ \alpha \hat{\mathbf{c}} \end{bmatrix}.$$

Multiplication of quaternions

$$\begin{bmatrix} w_1 \\ \hat{\mathbf{c}}_1 \end{bmatrix} \begin{bmatrix} w_2 \\ \hat{\mathbf{c}}_2 \end{bmatrix} = \begin{bmatrix} (w_1 w_2 - \hat{\mathbf{c}}_1 \cdot \hat{\mathbf{c}}_2) \\ (w_1 \hat{\mathbf{c}}_2 + w_2 \hat{\mathbf{c}}_1 + \hat{\mathbf{c}}_1 \times \hat{\mathbf{c}}_2) \end{bmatrix},$$

Quaternion inverse

$$\begin{bmatrix} \cos(\frac{\theta}{2}) \\ \sin(\frac{\theta}{2})\hat{\mathbf{k}} \end{bmatrix}^{-1} = \begin{bmatrix} \cos(\frac{\theta}{2}) \\ -\sin(\frac{\theta}{2})\hat{\mathbf{k}} \end{bmatrix}.$$

Quaternion to matrix

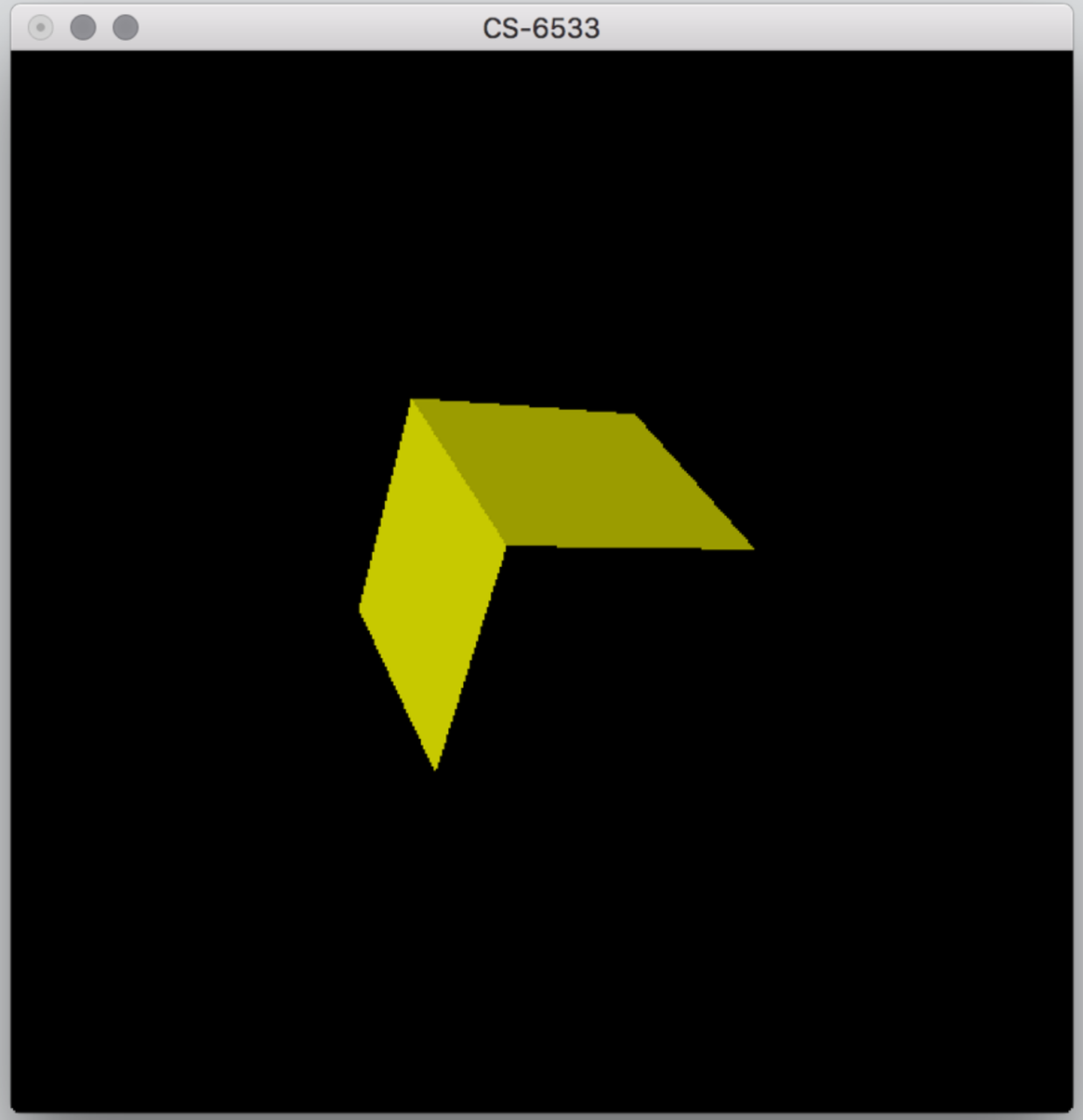
$$M = \begin{vmatrix} 1 - 2y^2 - 2z^2 & 2xy + 2wz & 2xz - 2wy \\ 2xy - 2wz & 1 - 2x^2 - 2z^2 & 2yz + 2wx \\ 2xz + 2wy & 2yz - 2wx & 1 - 2x^2 - 2y^2 \end{vmatrix}$$

Using quaternions in your code.

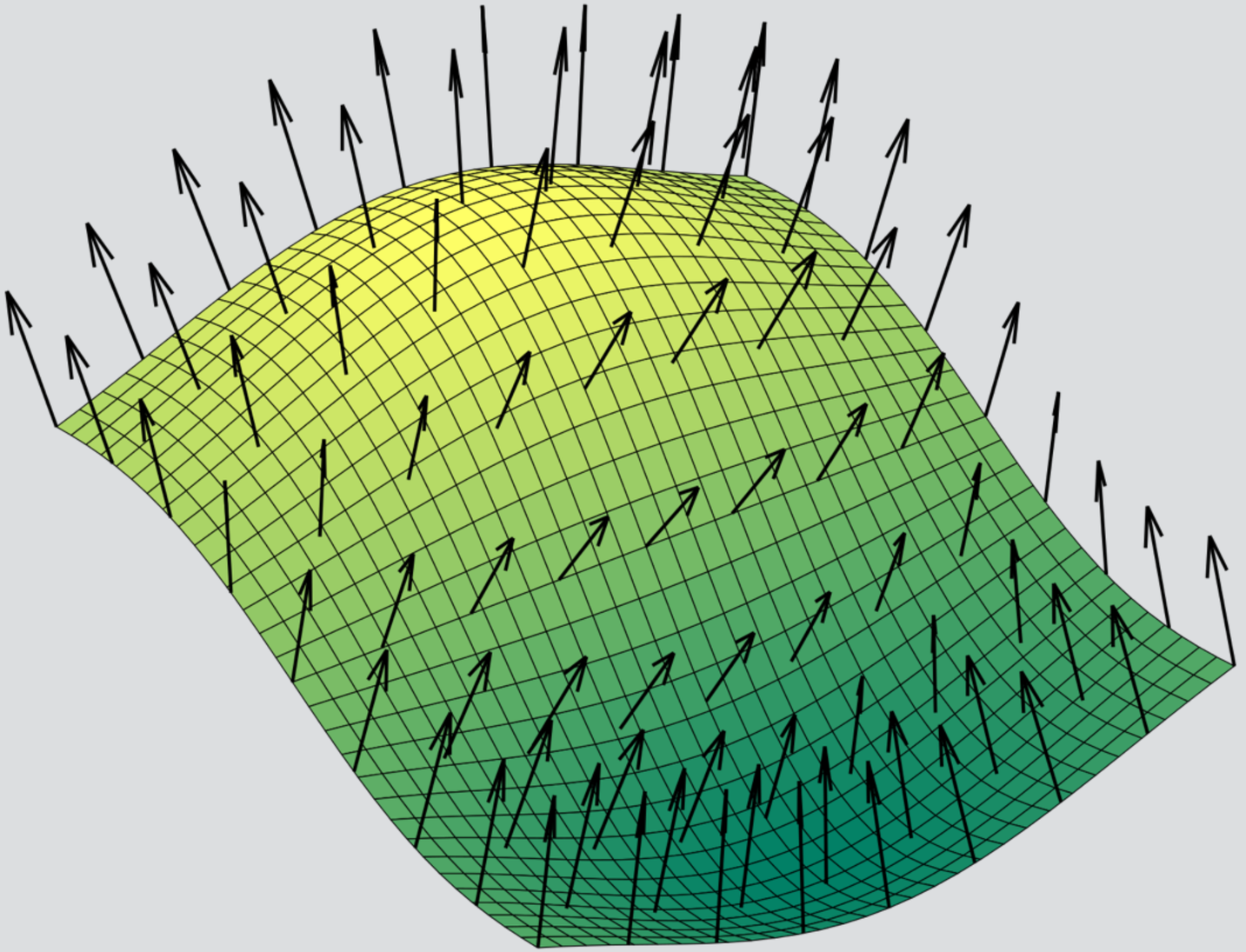
```
#include “quat.h"



Quat rotation = Quat::makeXRotation(45.0f);
Matrix4 rotationMatrix = quatToMatrix(rotation);



Quat q1 = Quat::makeYRotation(70.0f);
Quat q2 = Quat::makeZRotation(20.0f);
Quat combined = q1 * q2;
Matrix4 rotationMatrix = quatToMatrix(combined);
```

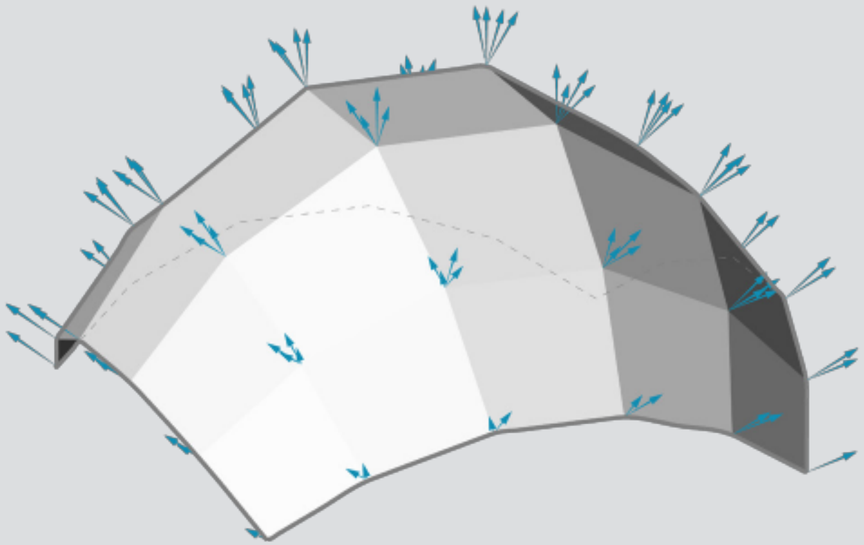
```
struct Transform {

    Quat rotation;
    Cvec3 scale;
    Cvec3 position;

    Transform() : scale(1.0f, 1.0f, 1.0f) {
    }

    Matrix4 createMatrix();
};
```

Using a basic lighting shader

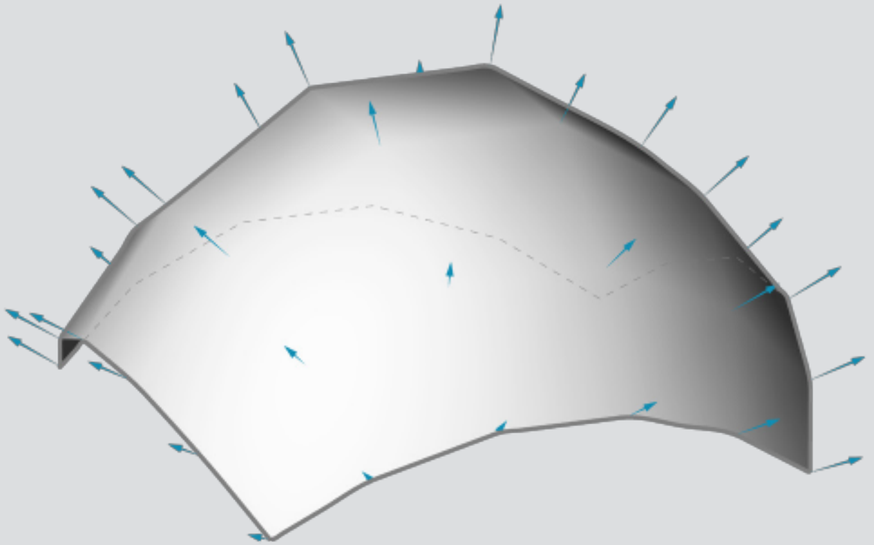

CS-6533

Normals





1



2

```
GLfloat cubeNormals[] = {
        -1.0f, 0.0f, 0.0f,
        -1.0f, 0.0f, 0.0f,
        -1.0f, 0.0f, 0.0f,
         0.0f, 0.0f,-1.0f,
         0.0f, 0.0f,-1.0f,
         0.0f, 0.0f,-1.0f,
        0.0f,-1.0f,0.0f,
        0.0f,-1.0f,0.0f,
        0.0f,-1.0f,0.0f,
        0.0f, 0.0f,-1.0f,
        0.0f, 0.0f,-1.0f,
        0.0f, 0.0f,-1.0f,
        -1.0f, 0.0f,0.0f,
        -1.0f, 0.0f,0.0f,
        -1.0f, 0.0f,0.0f,
        0.0f,-1.0f, 0.0f,
        0.0f,-1.0f, 0.0f,
        0.0f,-1.0f,0.0f,
        0.0f, 0.0f, 1.0f,
        0.0f,0.0f, 1.0f,
        0.0f,0.0f, 1.0f,
        1.0f, 0.0f, 0.0f,
        1.0f,0.0f,0.0f,
        1.0f, 0.0f,0.0f,
        1.0f,0.0f,0.0f,
        1.0f, 0.0f, 0.0f,
        1.0f,0.0f, 0.0f,
        0.0f, 1.0f, 0.0f,
        0.0f, 1.0f,0.0f,
        0.0f, 1.0f,0.0f,
        0.0f, 1.0f, 0.0f,
        0.0f, 1.0f, 0.0f,
        0.0f, 1.0f, 0.0f,
        0.0f, 0.0f, 1.0f,
        0.0f, 0.0f, 1.0f,
        0.0f,0.0f, 1.0f
    };
```

A simple lighting shader

Vertex program

```
attribute vec4 position;
attribute vec4 normal;

uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;
uniform mat4 normalMatrix;

varying vec4 varyingNormal;

void main() {
  varyingNormal = normalMatrix * normal;
  gl_Position = projectionMatrix * modelViewMatrix * position;
}
```

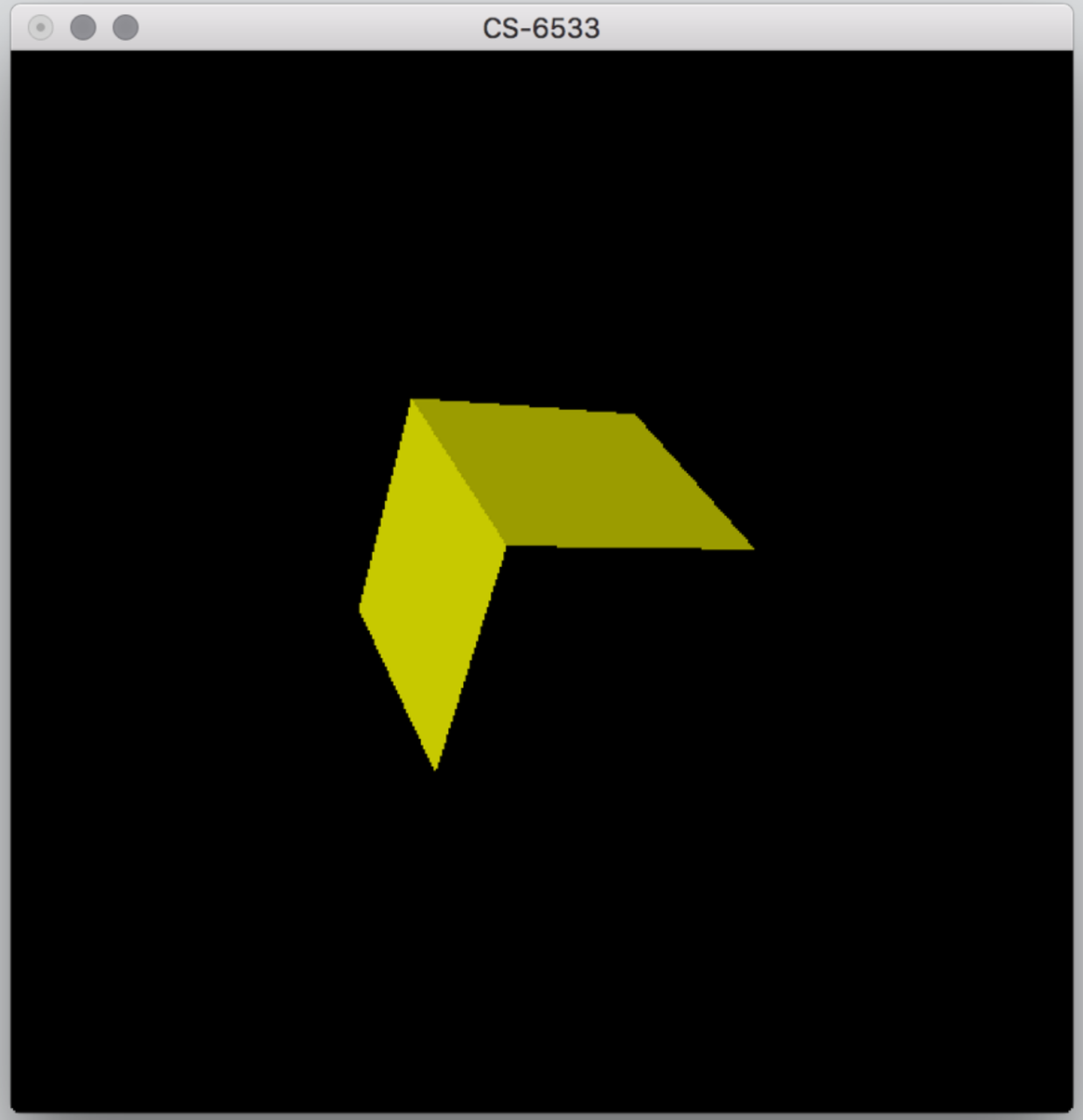
Fragment program

```
varying vec4 varyingNormal;

uniform vec3 uColor;

void main() {
   float diffuse = max(0.0, dot(varyingNormal, vec4(-0.5773, 0.5773, 0.5773, 0.0)));
   vec3 intensity = uColor * diffuse;
   gl_FragColor = vec4(intensity.xyz, 1.0);
}
```

Normal matrix

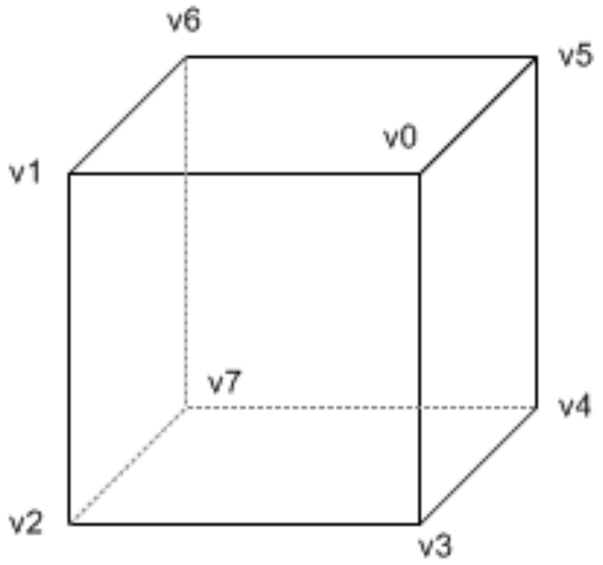
The normal matrix is the transpose of the inverse of the modelview matrix.

```
Matrix4 normalMatrix = transpose(inv(modelViewMatrix));
```


CS-6533

Beyond the cube

Vertex structures and indexed drawing


v6
v5
v0
v1
v7
v4
v2
v3

```
struct VertexPN {

    Cvec3f p;
    Cvec3f n;

    VertexPN() {}
    VertexPN(float x, float y, float z, float nx, float ny, float nz) : p(x,y,z), n(nx, ny, nz) {}
};
```

```
  std::vector<VertexPN> vtx;
  std::vector<unsigned short> idx;

// fill our arrays

  glGenBuffers(1, &vertexBO);
  glBindBuffer(GL_ARRAY_BUFFER, vertexBO);
  glBufferData(GL_ARRAY_BUFFER, sizeof(VertexPN) * vtx.size(), vtx.data(), GL_STATIC_DRAW);

  glGenBuffers(1, &indexBO);
  glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, indexBO);
  glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(unsigned short) * idx.size(), idx.data(), GL_STATIC_DRAW);
```

```
glBindBuffer(GL_ARRAY_BUFFER, vertexBO);
glVertexAttribPointer(positionAttribute, 3, GL_FLOAT, GL_FALSE, sizeof(VertexPN), (void*)offsetof(VertexPN, p));
glEnableVertexAttribArray(positionAttribute);

glVertexAttribPointer(normalAttribute, 3, GL_FLOAT, GL_FALSE, sizeof(VertexPN), (void*)offsetof(VertexPN, n));
glEnableVertexAttribArray(normalAttribute);

glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, indexBO);
glDrawElements(GL_TRIANGLES, numIndices, GL_UNSIGNED_SHORT, 0);
```

geometrymaker.h

Cube

```
void getCubeVbIbLen(int& vbLen, int& ibLen);


template<typename VtxOutIter, typename IdxOutIter>
void makeCube(float size, VtxOutIter vtxIter, IdxOutIter idxIter);
```

Plane

```
void getPlaneVbIbLen(int& vbLen, int& ibLen);

template<typename VtxOutIter, typename IdxOutIter>
void makePlane(float size, VtxOutIter vtxIter, IdxOutIter idxIter);
```

Sphere

```
void getSphereVbIbLen(int slices, int stacks, int& vbLen, int& ibLen);

template<typename VtxOutIter, typename IdxOutIter>
void makeSphere(float radius, int slices, int stacks, VtxOutIter vtxIter, IdxOutIter idxIter)
```

VtxOutIter must be an iterator of a type that has a copy assignment operator to the GenericVertex struct (see geometrymaker.h for GenericVertex fields), IdxOutIter must be an iterator of an integer type.

```
struct VertexPN {
    Cvec3f p, n;

    VertexPN() {}
    VertexPN(float x, float y, float z, float nx, float ny, float nz) : p(x,y,z), n(nx, ny, nz) {}

    VertexPN& operator = (const GenericVertex& v) {
        p = v.pos;
        n = v.normal;
        return *this;
    }
};
```

```
int ibLen, vbLen;
getCubeVbIbLen(vbLen, ibLen);

std::vector<VertexPN> vtx(vbLen);
std::vector<unsigned short> idx(ibLen);

makeCube(2, vtx.begin(), idx.begin());
```

Tying it all together

```
struct Transform {
    Quat rotation;
    Cvec3 scale;
    Cvec3 position;

    Transform() : scale(1.0f, 1.0f, 1.0f) {
    }

    Matrix4 createMatrix();
};
```

```
 struct Geometry {
     GLuint vertexBO;
     GLuint indexBO;
     int numIndices;

     void Draw(GLuint positionAttribute, GLuint normalAttribute) {
         // bind buffer objects and draw
     }
 };
```

```
struct Entity {
    Transform transform;
    Geometry geometry;

    void Draw(Matrix4 &eyeInverse, GLuint positionAttribute, GLuint normalAttribute,
              GLuint modelviewMatrixUniformLocation, GLuint normalMatrixUniformLocation) {

        // create modelview matrix
        // create normal matrix
        // set the model view and normal matrices to the uniforms locations

        geometry.Draw(positionAttribute, normalAttribute);
    }

};
```

Assignment 2

- Render a simple 3D scene using primitives.
- At least 3 objects must be in a hierarchy (transforming relative to another object).
- Use the simple lighting shader for your objects and set their color using the uColor uniform.