Brack Turner
CS411 AS5 report

Added inverting normal functionality in beginning definitions
Calls on initialize model function to redraw model with new specs.

```
function invertNormals()

 invertNorm = !invertNorm;
 readOBJFile(objName, gl, model, 60, invertNorm)
   .then(() => {
     initializeModel();
     drawScene(gl, gl.program, undefined, buffers, model)
   })
   .catch(console.error);
```

In the initialize Model function we set the initial normals to zero and perform normalization operations on the vertex data read from the object file. Afterwards, we push that data into a new vertex array for the new frames in the scene.

```
//initialize the normals  at zero
model.objDoc.normals = model.objDoc.vertices.map(init => ({x: 0, y: 0, z: 0}));
// then add face normals for each set of vertices
model.objDoc.objects[0].faces.forEach(f =>
  f.vIndices.forEach(vert => {
    model.objDoc.normals[vert].x += f.normal.x;
    model.objDoc.normals[vert].y += f.normal.y;
    model.objDoc.normals[vert].z += f.normal.z;
  })
);

//normalize the vertex normals
model.objDoc.normals = model.objDoc.normals.map(norm => {
  const c = Math.sqrt(Math.pow(norm.x, 2) + Math.pow(norm.y, 2) + Math.pow(norm.z, 2));
  return { x: norm.x / c,
           y: norm.y / c,
           z: norm.z / c
         };
});

let newarray = [];

model.objDoc.objects[0].faces.forEach(f =>
  f.vIndices.forEach(vertindices => {
    const norm = model.objDoc.normals[vertindices];
    newarray.push(norm.x, norm.y, norm.z);
  })
);
model.arrays.normals = new Float32Array(newarray);
```

We then create a bounding box with the centermass initiated as well.

Then we take the newly defined centermass Matrix and perform transformation translations on the centermass object/ array.

```
let centermass = [ 0, 0, 0];
model.arrays.vertices.forEach((con, index) => centermass[index % 3] += con);

centermass = centermass.map(con => - con / model.arrays.vertices.length);
console.log({centermass});

centermassMatrix.setTranslate(centermass[0], centermass[1], centermass[2]);
assignVertexBuffersData( gl, buffers, model);
```

We finally read the object file in the initScene() function so we can upload the object file to the program, and callback the initialize model function to apply to the object. We return an error if the model fails to initialize.

```
// start reading the OBJ file
model = new Object();
var scale=60; // 1
readOBJFile(objName, gl, model, scale, true)
.then(() => {
  initializeModel();
}).catch(console.error);
```

We also move the camera relative to the object instead of the scene using matrix transformations in the scene.

```
mvMatrix.multiply(centermassMatrix);
//do rotation about object
mvMatrix.translate(camZ, 0, 0);
mvMatrix.multiply(curRot);
```

Brack Turner
CS 411
Assignment 5

1.1 Parameters include fixed point (P_f), a rotation angle (θ), and a line or vector to rotate about. By using a 4x4 matrix it helps keep a reference to other transformations using homogeneous coordinates. The advantage is combined rotation and translation operations

1.2
```
var p = new Matrix4();
var sx=2, sy=2, sz=1;
p.setScale(sx,sy,sz);
mvMatrix.scale(sx,sy,sz);
```

1.3 `p.setRotate(45,0,1,0);`
where R(angle, vec(0,1,0))

1.4 $\theta = \arccos\left(\frac{\langle 1,1,0\rangle * \langle 0,1,0\rangle}{\left(\sqrt{1^2+1^2+0^2}\right) * \left(\sqrt{0^2+1^2+0^2}\right)}\right)$

$\theta = 45$ degrees

axis $= \langle 1,1,0\rangle \times \langle 0,1,0\rangle = \langle 0,0,1\rangle$

// aligned

R(theta, axis) = `m.setRotate(45,0,0,1);`

1.5 `R(45, <1,1,0>);`

```
m.setIdentity().rotate(45,0,1,0).rotate(45,0,0,1)
.rotate(1,1,0);
```

1.6
```
var eyeX=0, eyeY=0, eyeZ=5;
var centerX=0, centerY=0, centerZ=0;
```

var $0_{pX}=0$, $0_{pY}=1$, $0_{pZ}=0$;
m.setLookAt(eyeX, eyeY, eyeZ, centerX, centerY, centerZ, 0_{pX}, 0_{pY}, 0_{pZ});

1.7 $p=(1,2,3)$ $v=(1,0,1)$
$v=(-1,-2,-3)$
Rotation $\rightarrow \left[-\frac{1}{\sqrt{14}}, -\frac{2}{\sqrt{14}}, -\frac{3}{\sqrt{14}}\right]$

$$\begin{bmatrix} -1/\sqrt{14} & -3/\sqrt{14} & -3/\sqrt{14} & 0 \\ 2/\sqrt{14} & -1/\sqrt{14} & -3/\sqrt{14} & 0 \\ -3/\sqrt{14} & -2/\sqrt{14} & 1/\sqrt{14} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about 45° on y axis

$$R_y \rightarrow \begin{bmatrix} \frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translate $(1,2,3) = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

$$= \begin{bmatrix} -0.75 & -0.8 & -0.37 & 1 \\ -0.53 & -0.26 & -0.53 & 2 \\ -0.37 & -0.94 & 0.75 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

1.8 $r_1[1,2,3,3]$ $r_2=[2,3,4,4]$
$r_3[3,4,4,5]$ $r_4=[0,0,0,1]$

$$\begin{bmatrix} 1 & 2 & 3 & 3 \\ 2 & 3 & 4 & 4 \\ 3 & 4 & 4 & 5 \\ 0 & 0 & 0 & 1 \end{bmatrix} (m^{-1})^T = \begin{bmatrix} -4 & 4 & -1 & 0 \\ 4 & -5 & 2 & 0 \\ -1 & 2 & -1 & 0 \\ 1 & -2 & 0 & 1 \end{bmatrix}$$

1.8 $(M^{-1})^T v$

$$\begin{bmatrix} -4 & 4 & -1 & 0 \\ 4 & -3 & 2 & 0 \\ -1 & 2 & -1 & 0 \\ 1 & -2 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 2 \\ -1 \\ 1 \end{bmatrix}$$

$[-1, 2, -1, 1] \rightarrow 3D = (-1, 2, -1)$

1.9 orthographic projections are parallel projections where they are perpendicular to the view plane. Perspective projection is when the lens is a finite focal length. The advantage is perspective uses homogeneous coor to not cause scaling.

1.10 It distorts the view volume which supporting API must account for.

1.11 Use orthographic projection matrix to pass on z-coordinate and keeps the 3D point (1,2,3)

1.12 $p = (1,2,3)$ $vec(0,-1,1)$

align vector to z-axis

$$\begin{bmatrix} 1 & 0 & \cot\alpha\cos\phi & 0 \\ 0 & 1 & \cot\alpha\sin\theta & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ -1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

$\cot\alpha\cos\phi = 0$

$\cos\alpha\sin\theta - 1 = 0 \quad \phi = \cos^{-1}0 \quad \phi = 90$

$\cos\alpha\sin(90) - 1 = 0 \quad \alpha = 45°$

1.12 $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \\ 3 \\ 1 \end{bmatrix}$

3D (1, 5, 3)

1.13 $p = (1,2,3)$ focal length $= 10$

$p' = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 10 \\ 20 \\ 30 \\ 3 \end{bmatrix}$

$3D\left(3.\overline{3}, 6.\overline{6}, 10\right)$

1.14 $[x\ y\ z\ w] \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{bmatrix}$

Simple perspective Matrix

$x' = x \cdot m_{00} + y \cdot m_{10} + z \cdot m_{20} + w \cdot m_{30}$

$y' = x \cdot m_{01} \ldots$

$z' = x \cdot m_{02} \ldots$ expand

$w' = x \cdot m_{03} \ldots$

1.15 $x_v = x_{vmin} + \frac{x+1}{2}(x_{vmax} - x_{vmin})$

$y_v = y_{vmin} + \frac{y+1}{2}(y_{vmax} - y_{vmin})$

$z_v = z_{vmin} + \frac{z+1}{2}(z_{vmax} - z_{vmin}$

1.16

1.17

1.18 z-buffer checks whether a point is visible if it is the closest point of intersection along a ray. Painters algorithm uses visual determation with polygons by clipping rear facing ones and putting them in frome buffers
Z-buffer image space. Painters object space.