Mike Petrus
CS 180

Assignment 6

1. *Implement ray_sphere_intersect() to return the closest hit of a sphere (if it exists) and render the pixel color of the sphere at that point*

We have all the parts of the Ray-Sphere intersection: (**o** + *t***d** - **c**) - R^2 = 0, so solve for the equation: $at^2 + bt + c = 0$
a = d · d, where **d** is the normalized *ray_direction*
b = 2 (**o** - **c**) **· d**, where **o** is the *ray_origin* and **c** is the *sphere.position* (location of center)
c = (**o** - **c**) **·** (**o** - **c**) - **R^2**, where R is the *sphere_raidus*

We can first calculate the the *discriminant* of the quadratic equation. If it is less than 0 then we have no real solutions so return **false**.

Otherwise, calculate the two solutions for **t**, with **t1** being the smaller (closer) solution. If **t1** is within the range of *t_near* and *t_far* then return {**t1, true**} to find our intersection point/color. If not, check the same conditions for **t2**. If neither solution is within *t_near* and *t_far* then we have no valid hit on the screen and return **false**.

2. *Implement ray_triangle_intersect() to return the closest hit of a triangle mesh (if it exists) and render the pixel color of the triangle at that point*

To find the intersection most efficiently we use the **Möller Trumbore Algorithm**.

We can find $\mathbf{E_1 = P_1 - P_0}$ and $\mathbf{E_2 = P_2 - P_0}$ by using the vertices of our triangle where v_0, v_1, v_2 correspond to the points P_0, P_1, P_2

Next find
$\mathbf{S = O - P_1}$, where **O** is the *ray_origin*
$\mathbf{S_1 = D \times E_2}$, where **D** is the *ray_direction*
$\mathbf{S_2 = S \times E_1}$

$$\begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = \frac{1}{\vec{\mathbf{S}}_1 \bullet \vec{\mathbf{E}}_1} \begin{bmatrix} \vec{\mathbf{S}}_2 \bullet \vec{\mathbf{E}}_2 \\ \vec{\mathbf{S}}_1 \bullet \vec{\mathbf{S}} \\ \vec{\mathbf{S}}_2 \bullet \vec{\mathbf{D}} \end{bmatrix}$$

Then find the matrix value of **t** using the formula:

Finally, the check that **t** is inside the bounds of the triangle, using the barycentric coordinates $\mathbf{b_1}$ and $\mathbf{b_2}$ with the final coordinate = ($\mathbf{1 - b_1 - b_2}$). If all 3 coordinates are **>= 0**, then **t** lies inside (or on the edge) of the triangle and we return {**true, t**}
Otherwise, return **false.**