Raytracer Project - Phase 1 (40 pts)

Some code to get you started can be downloaded from:
https://bitbucket.org/summateaching/raytracing_template.git

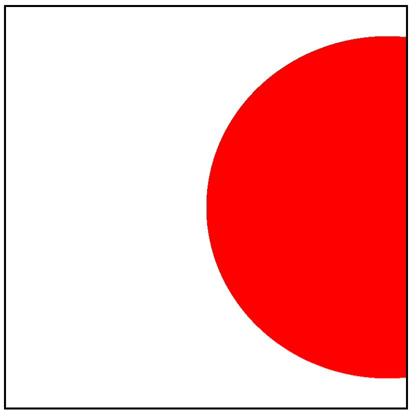
Your first task to to read and fully understand what is happening in this code. Please reference the overview document for more details

For the first part of this project, you are to populate these functions in main.cpp and Object.cpp:

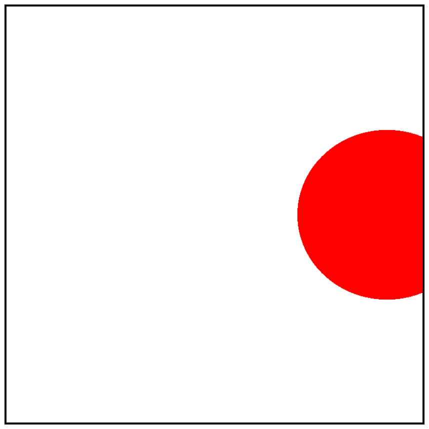
1. **(10 pts)** castRay()
    1.1. This is called per ray and should intersect the ray will all objects in your scene. Initially, (this will change) it should return the color of the closest object hit. Color is located inside of sceneObjects[i]->shadingValues.color for object i. Note that you can use the (not currently used) _ID in IntersectionValues to record and lookup which object created which IntersectionValue.
2. **(15 pts)** intersect()
    2.1. transform ray in world space to object space, call raySphereIntersection() or raySquareIntersection()
    2.2. Populate IntersectionValues result
3. **(5 pts)** raySphereIntersection()
    3.1. Intersect a ray with a sphere
4. **(5 pts)** raySquareIntersection()
    4.1. Intersect a ray with a square

Here are some suggestions on proceeding on getting these functions working.

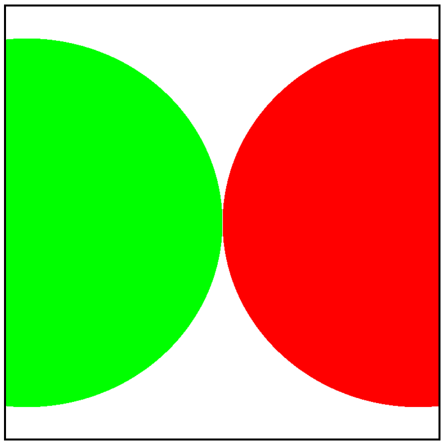
- Frist start with the _SPHERE scene. The modelview for your sphere is the identity (mat4() in the sphere scene initialization in main.cpp). In this case, the world and object coordinates are synced. If this is the case your intersect() function does not need to do anything at first, therefore concentrate on getting castRay() and raySphereIntersection() working. If they are working correctly, you should get a red circle at the center of your image.
- Change the init of the sphere sphere's modelview from mat4() to Translate(1.0, 0.0, 0.0)). Your output should look like:



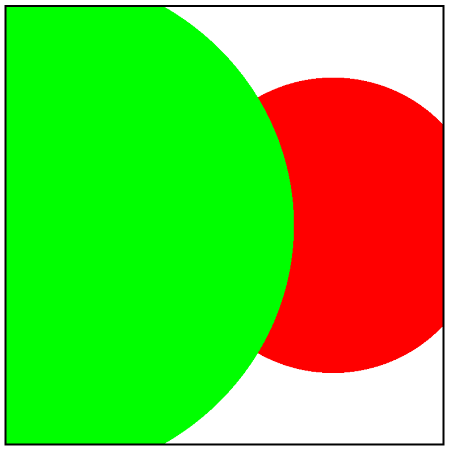
- Change it now to Translate(1.0, 0.0, 0.0)*Scale(0.5, 0.5, 0.5) Your output should look



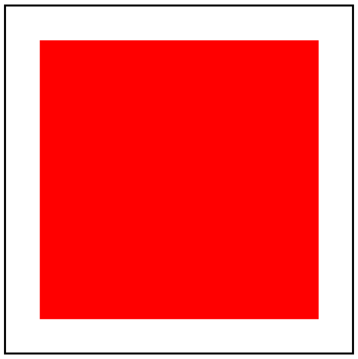
  like:
- Add a second sphere (green color), make sphere one Translate(1.0, 0.0, 0.0) and sphere two Translate(-1.0, 0.0, 0.0). Your output should look like this:



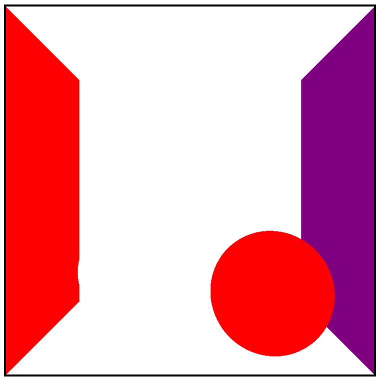
- Move it, rotate it.



- Switch to the _SQUARE scene. Now you have to write code that intersects unit square on the XY-plane with Z=0. That is what Square::raySquareIntersection is doing. (look in your notes for ray-plane intersection and how you would change that to handle a unit square) Now add to Square::intersect() similarly to your sphere. You should now have something like this (if you change the "Unit Square" color to red). Note that red square does not go to the image boundary.



- Move it, rotate it like the sphere and it should operate similarly.
- Now switch to _BOX and TADA a flat shaded cornell box.



A version where i changed things with white color: