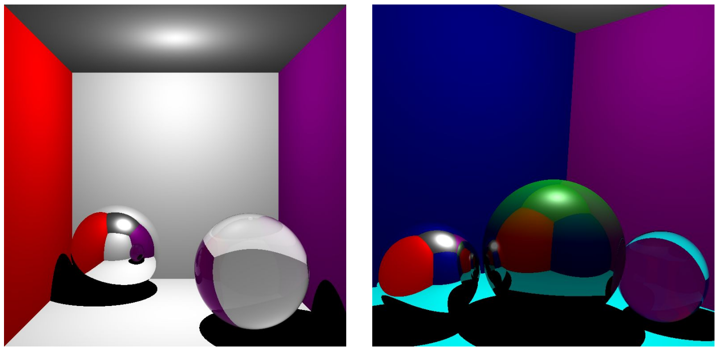
Raytracing Final Project Overview (100 pts Undergrad, 120 pts Grad)



Project: Create a Whitted-style raytracer.
Turn in: Code + 3-4 raytraced images with varying scenes and/or views

Template Code: All of the functions that I used to construct my raytracer are included although the pertinent code has been removed (look for TODO comments in the code). This code base is similar to your HW3, but you'll need to adjust more than that assignment so I'll document some important aspects.

- **ObjMesh** class is your mesh loader from HW3 but now has a subdivision sphere as a potential mesh
- **Object** is a super class for all the scene's objects. It contains:
  - The mesh needed for OpenGL rendering
  - Attributes used for local shading
  - Matrices like C (modelview), C^{-1}, $(C^*)^{-1}$, and $(C^{-1})^T$. These are computed by the setModelView function
- **Sphere** and **Square** are child classes of **Object**.
  - Each contain an intersect function that you'll need to complete these functions.
  - Each contain a more general ray/sphere and ray/square intersection routines that are used by the intersect function. You'll need to complete these functions.
- I tried to keep most of the important code main.cpp:
  - *rayTraceReceptor* and *write_image* are functions that create the output png and can be ignored
  - *initGL* and *drawObject* are handing the OpenGL window and can be ignored.
  - *findRay* and *rayTrace* are provided for you. *findRay* provides the ray start and direction for a pixel with result is returned as a length 2 vector (ray_start, ray_dir). *rayTrace* calls *findRay* for every pixel in your raytraced image, calls *castRay*, for each and outputs a png with the results.
  - As you can see, *castRay* is the recursive function that is the major chunk of this assignment. It's input is the ray start, the ray direction, the last object hit by the ray (think about why this might be important), and the depth of the recursion.

- *shadowFeeler* sends a shadow ray to any point lights from a point. It also takes as input the object casting the shadow ray (think about why this might be useful).
- I have also included a *castRayDebug* function that I used to cast a single ray in order to look at the output it produces. It might be useful for you as well, to create something similar. (no code is provided)

Template Scenes: I have provided 3 example scenes a sphere, a cube, and a cornell box: 6 sides, two spheres (one mirrored and one glass-like), and a single point like at the center of the ceiling. ***You will be graded on the cornell box scene.***

How the Program Works: Your program gives you an OpenGL window of your scene. You are given a flat fragment shader as default, please swap that out for your solution to assignment 3. ***To render an OpenGL view via your raytracer hit the 'r'.*** '1', '2', '3' switch your scene.

Grad Portion: In addition to the project milestones, grads must also include:

- (10 pts) Soft shadows via an area light
- (10 pts) A raytraced triangle mesh (pick your favorite from Assignment 3, excluding the cube)