Programming Assignment 4

COMP 575/770 Spring 2016

Due: April 25, 2016

Instructions

- Please work on the assignment on your own. It is okay to discuss the assignment with other students, but please write your own code independently. If you use code from the Internet or any other source, please acknowledge the source.
- You are free to use any programming language you are comfortable with, but try not to use anything too obscure. C/C++, C#, Objective-C, Java, and Python are common choices.
- You will need to use a fairly small amount of OpenGL and GLUT to get your image on-screen.
- Submit your source code via email along with a README file containing compilation instructions, additional required libraries (if any), and a short description of what the different parts of your program do.
  Email Ids: Tanmay (tanmay@cs.unc.edu), Michael (mkcolema@cs.unc.edu)
- This programming assignment consists of two parts. Only Part 1 is required for COMP 575 students. COMP 770 students must do both Part 1 and Part 2. COMP 570 students who solve Part 2 will receive extra credit.

Part 1: Recursive Ray Tracing
In this part, you will add mirror-like reflections to the scene from Programming Assignment 1. Refer to the problem specification in Programming Assignment 1 for details on view properties, lighting and material properties, and scene geometry. In addition, we will be using the following model to add reflections to the scene:

$$L = (1 - \alpha)(L_a + L_d + L_s) + \alpha L_m \tag{1}$$

where L_a, L_d, and L_s are the ambient, diffuse, and specular shading terms, L_m is the shading value along a ray reflected (using the laws of reflection) at the ray's hit point, L is the total shading value, and α is a weight. Since L_m will in turn be computed using a similar shading expression, the result is a recursive reflection.

For this problem, use a maximum recursion depth of 2 (i.e., don't reflect a ray more than 2 times). For the spheres S_1 and S_2, $\alpha = 0$. For S_3, $\alpha = 0.8$. For the plane P, $\alpha = 0.5$. A reference rendering is shown in Figure 1.

Note that antialiasing is not required for this assignment.

Part 2: kD Trees
In this part, you will use ray tracing to render a triangle mesh for a cathedral. The mesh contains around 80,000 triangles. The mesh and code to load the mesh is also provided. Use the following parameters for your rendering:

- Use an image plane defined by $l = -0.1$, $r = 0.1$, $b = -0.1$, $t = 0.1$, and $d = 0.1$.
- Use a camera defined by $\mathbf{e} = (0, -10, 0)$, $\mathbf{u} = (0, 0, 1)$, $\mathbf{v} = (0, 1, 0)$, and $\mathbf{w} = (-1, 0, 0)$.
- Use a single point light source at $(0, 0, 0)$, with no falloff.

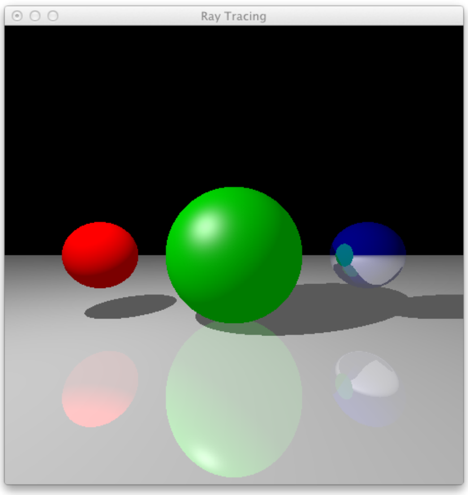
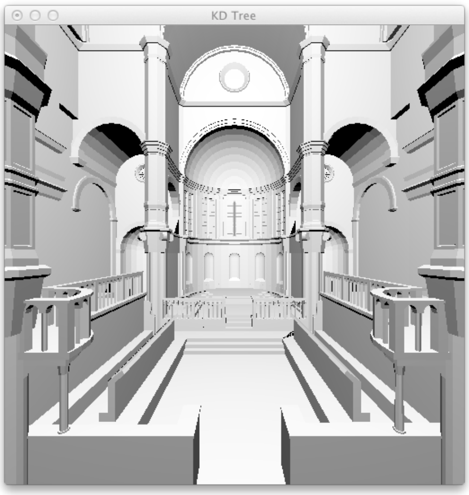
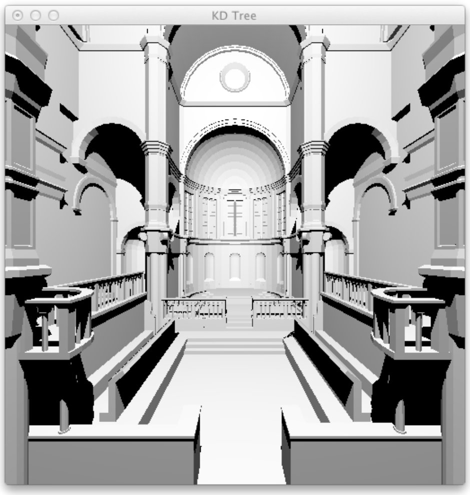


Figure 1: Recursive ray tracing for mirror-like reflections.



(a) Without shadows.



(b) With shadows.

Figure 2: Reference images for the Sibenik cathedral.

- Assume all triangles have the same material, with $k_d = (1, 1, 1)$, $k_a = k_s = (0, 0, 0)$, and $p = 0$.

Since this is a complex model, you will have to use a kD tree to accelerate the intersection calculations. We have provided a KD-tree for this model. Or you can write the code to build a KD-tree based on any known heuristic to compute a KD-tree. It will be good to analyze the performance benefit or speedup (in terms of overall ray tracing) based on the KD-tree.

Render the image with and without shadow rays. You do not have to perform recursive ray tracing or antialiasing. Reference images are shown in Figure 2.