Programming Assignment 4

COMP 575/770 Fall 2014

Due: November 12, 2014

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.
- 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 575 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, and can be downloaded from the course web page. Code to load the mesh is also available from the course web page. 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.
- Assume all triangles have the same material, with $k_d = (1, 1, 1)$, $k_a = k_s = (0, 0, 0)$, and $p = 0$.

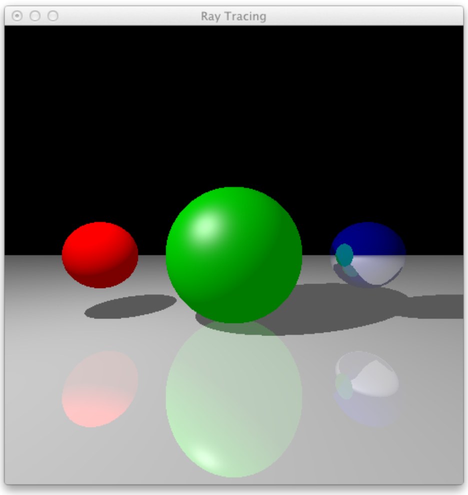


Figure 1: Recursive ray tracing for mirror-like reflections.

Since this is a complex model, you will have to build a kD tree and use it to accelerate the intersection calculations. You may use any heuristic to construct the kD tree. One option for determining the splitting plane for a given node is to compute the bounding box of the node and split its longest dimension halfway through. If either of the children of a node contain all the triangles of the node, mark the node as a leaf instead.

[**Update**] The file available at `http://gamma.cs.unc.edu/graphicscourse/kdtree.simple` describes a kd-tree of the sibenik model and can be used in place of implementing the full kd-tree construction. Each line of this file fully describes a single node of the tree and has the following format:

Inner Nodes: *inner*{ min_x min_y min_z Max_x Max_y Max_z ; *left right axis split* }

- $min_{\{x,y,z\}}$:= the minimal coordinates of the node's bounding box
- $Max_{\{x,y,z\}}$:= the maximal coordinates of the node's bounding box
- *left right* := the node indices of the left and right child of this node
- *axis* := the coordinate $(x = 0, y = 1, z = 2)$ axis that this node's splitting plane splits
- *split* := the position of the node's splitting plane along the axis of the split

Leaf Nodes: *leaf*{ min_x min_y min_z Max_x Max_y Max_z ; t_0 t_1 t_2 ... }

- $min_{\{x,y,z\}}$:= the minimal coordinates of the node's bounding box
- $Max_{\{x,y,z\}}$:= the maximal coordinates of the node's bounding box
- $t_0 t_1 t_2$ ... := the sequence of triangle indices contained in this region

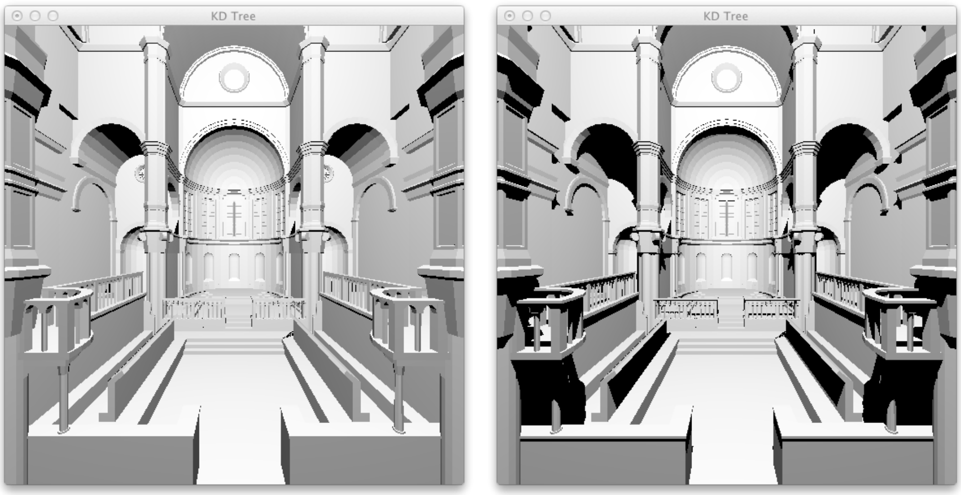
Examples :

```
leaf{ -19.81 -11.59 -5.38 -19.51 -11.28 -5.14 ; 27097  27098  27167  27168 }

inner{ -20.14 -15.30 -8.49 0.00 4.48 8.49 ; 3 4 0 -10.070550 }
```

Root node and indexing: Both the triangle and node indices start at 0. The nodes are indexed according to the order in which they appear in the file with the first line being the root node with index 0 (Ex: If the nodes were read from the file and constructed in an array in-order their offset from the first node in the array would equal their index). The triangle indices follow the order they appear in the .obj file as well.

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.



(a) Without shadows.   (b) With shadows.

Figure 2: Reference images for the Sibenik cathedral.