# Project 3
# CSCI-4967: Three-Dimensional Computer Graphics

**Due: Friday, November 12, 2004, 11:59:59pm**

## 1 Overview

In this project, you are to write a ray tracer. Given an input description of a scene consisting of spheres and triangles, your program must generate a ray traced image of the scene. Your program should consider the reflected and transmitted components of the light, and also compute shadows. The positions and material properties of the objects, and positions and colors of lights will be specified in an input scene file. (The format of the input scene file is described on the project web page.)

Please see the course web page at `www.cs.rpi.edu/~sakella/graphics/` for project updates and additional information.

## 2 Project Tasks

### 2.1 Generating and Representing Rays

You should generate one viewing ray for each pixel such that it originates at the eye and passes through the center of the pixel. This ray will be tested for intersections with the objects in the scene to identify the closest hit object. If no object is hit, the pixel is colored with the background color.

You should assume the eye is at the origin pointing along the negative $Z$ axis. The view plane is parallel to the $XY$ plane and located one unit away along the negative $Z$ axis. The input file will specify the field of view angle (like in `gluPerspective`). The image width and height are specified in terms of the number of view plane pixels in the horizontal and vertical directions.

A ray is represented as having an origin point $P_o$ and a unit vector $u$. The distance along the ray is given by the quantity $s$, so that a point $P$ at a distance $s$ from $P_o$ is given by $P = P_o + su$.

### 2.2 Computing the Pixel Color

The lighting model will consider the ambient, diffuse, and specular components of light. You will use the Phong illumination equation. The light sources are modeled as point light sources whose positions are specified. You may neglect the distance attenuation of the light sources.

The pixel color of an object depends on the ambient light, the ambient material color, light source color, material diffuse color, material specular color, material specular exponent, and how reflective and how transparent the object is. Pixels corresponding to rays that do not hit any objects are given the background color. All colors will be specified as RGB triples with each component in the range 0 to 1.

### 2.3 Input Scene File

The following information is included in the input scene file.
Ambient light color.
Background color.
Default refractive index (refractive index of air). Max recursion depth.
Field of View (degrees).
Point lights (position, color)
Spheres (position, radius)
Triangles (vertices)
Material properties of above objects (spheres, triangles): (ambient color, diffuse color, specular color, shininess (float), reflectance (color), transmittance (color), refractive index (float) )

### 2.4 Software Design

You should create a class to represent objects, which can serve as a common base class for spheres and triangles. This class should include shape and material information. Material information could include ambient color, diffuse color, specular color, shininess specular exponent (float), reflectance (color), transmittance (color), and refractive index (float).

You may also find it useful to create classes to represent points, vectors, and rays. Create math helper functions to perform operations such as computing vector cross products and dot products, normalizing vectors, computing normals to an object surface, compute intersections of a ray with a surface, etc.

### 2.5 Steps

These steps provide a guideline, along with the pseudocode in the class notes.

1. First generate the viewing ray from the eye.

2. Compute the first object surface that the ray hits. You will also need to compute the surface normal at the point of intersection.

3. Initially assign the pixel the ambient illumination, based on the surface that the ray first hits. Next, for each (visible) light source, add the diffuse and specular components of illumination from the light source to the pixel color. If the ray hits no surface, assign the pixel the background color.

4. To compute shadows, check whether the shadow ray from the point of intersection to each light source intersects any opaque objects.

5. Next compute the reflected and transmitted rays, and add their contributions to the pixel color. If an object is perfectly opaque, it will not permit any transmitted rays.

6. Use `glDrawPixels()` to display the ray traced scene on the screen.

7. Code for saving the rendered scene in PPM format is in the file `ppmwriter.h`, available on the project web page.

8. Your `raytrace` program should use command line arguments:
   ```
   raytrace <scenefile> <imagewidth> <imageheight> -window
   raytrace <scenefile> <imagewidth> <imageheight> -ppm <imagefile>.ppm
   ```
   The first command displays the rendered image in a window. The second command writes the output to a PPM format image file.

9. The input scene is described using the format on the project web page, where a simple scene file `simple_scene.txt` is provided. Your program should render this scene using ray tracing in less than a minute on the VCC South machines.

10. Finally, you should create a valid input scene file of your own in the same format, and submit it. The name of this file should be `myscene.txt`. You may add objects to the scene or otherwise enhance it to make it more interesting.

## 3 Grading

Your project will be graded for a total of 100 points as follows:

- 35 points for basic ray-surface intersection with spheres and triangles.
- 20 points for Phong illumination model.
- 20 points for reflection and transmission rays.
- 5 points for shadows computation.
- 5 points for an example scene file (showing enhancements, if any).
- 10 points for code structure, clarity, and documentation.
- 5 points for any special features or creative enhancements.

Describe any enhancements or special features in your README file, and ensure that they do not interfere with the required project functionality.
**Lateness policy:** Please read the lateness policy in the course syllabus. Use your late days carefully, if at all.

## 4 Notes and Suggestions

- Begin by implementing the simplest functions, test them to verify correctness, and then move on to other functions. At each stage, check the rendered image to see if it is being drawn correctly.
- To reduce floating point errors, use `double`s.
- When computing the intersections of a ray originating from a surface with other objects, let the ray begin a small distance $\epsilon$ above the surface (in the direction of the ray) to avoid an intersection with the originating surface. A value of $\epsilon = 0.001$ should be sufficient.
- To help debug your program, you may find it helpful to create debugging functions that print out the pixel values at specified pixels, and perhaps even intermediate computations.
- The direction of the normal to a surface depends on whether the ray hits the object from the inside or the outside.
- While manipulating color values, if any of the RGB components becomes greater than 1.0, then it should be clamped to 1.0.
- Start the project early!!!

## 5 Submission

The code must be submitted no later than **11:59:59 pm on November 12, 2004**. You will follow the same submission procedure as for Project 1. Specific instructions for handing in your code are provided on the course web page. **You are responsible for ensuring that your code can compile and run on the PCs in VCC South to get project credit.** Your source code must be readable and commented. The comments need not be extremely long, just explain clearly the purpose of each block of code.

You must submit a single zip file containing your README file, your `myscene.txt` file, your source code (source and header files), and if necessary, a Makefile to compile it. Your submission should NOT include any object files or executable files. Every file should have your name in a comment line at the top. The README file should contain the following information: your name, email address, instructions on how to compile the code and run it, known bugs or limitations, any extra credit enhancements, and any other relevant information.

Proper submission is entirely **your responsibility**. Contact the TA if you have any doubts whatsoever about your submission. Do **NOT** submit your project via email. Be sure to keep copies of your files and **do not change them after submitting**. After grades are posted, you have exactly one week to resolve all problems. All grades are final two weeks after they are posted.

A project that does not follow the submission guidelines will receive a **10 point deduction**. Please observe the academic integrity guidelines for the course as described in the course syllabus.