CMSC 435 / 634 Introduction to Computer Graphics

Project Assignment 4: Raytracing

Goals of this project:
Understand how images are created in computer graphics (animations). Once this project is completed, you will understand how reflection, refraction, and texture are generated in graphics.

This is also the first project that the instructor won't provide any supporting code to give you maximum flexibility to start engineering your own computer graphics program.

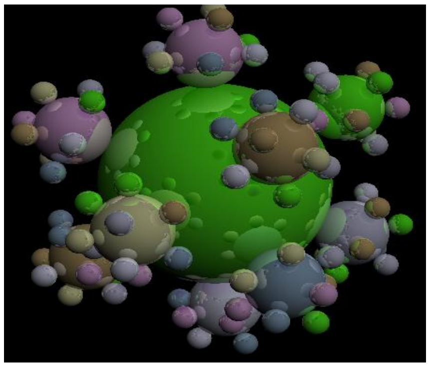
The Assignment
Ray tracing is a simple and powerful algorithm for rendering images. Within the accuracy of the scene and shading models and with enough computing time, the images produced by a ray tracer can be physically accurate and can appear indistinguishable from real images. The ray tracer we do in this class will not be powerful enough to produce physically accurate ones and if you are really interested, go on to take CMSC 635, where you will learn how to build modern physically accurate images.

In this assignment, your ray tracer will have support for:

- Spheres
- Diffuse shading and ambient shading
- Arbitrary orthogonal cameras

Your output should be:

- An image in PPM format



http://www.cs.unc.edu/~rademach/xroads-RT/RTarticle.html

Input File Format
The input file for your ray tracer is in XML. An XML file contains sequences of nested elements that are delimited by HTML-like angle-bracket tags. For instance, the XML code:

```
<scene>
        <camera>
        </camera>
        <surface type=Sphere>
                <center> 0, 1, 1 </center>
        </surface>
</scene>
```

contains four elements. One is a *scene* element that contains two others, called camera and surface. The *surface* element has an attribute named *type* that has the value *Sphere*. It also contains a center element that contains the text "0, 1, 1", which in this context would be interpreted as the 3D point (0, 1, 1).

An input file for the ray tracer always contains one *scene* element, which is allowed to contains tags of the following types:

- *Surface*: This element describes a geometry object. It must have an attribute *type* with value *Sphere* (we won't use any other geometries). For Sphere, *center* containing a 3D point, and *radius* containing a real number.
- *Camera*: This element describes the camera. It is described by the following elements:
  - *viewPoint*, a 3D point that specifies the center of projection.
  - *viewDir*, a 3D vector that specifies the direction toward which the camera is looking. Its magnitude is not used.
  - *viewUp*, a 3D vector that is used to determine the orientation of the iamge.
  - *projNormal*, a 3D vector that specifies the normal to the projection plane. Its magnitude is not used, and negating its direction has no effect. By default it is equal to the view direction.
  - *projDistance*, a real number *d* giving the distance from the center of the image rectangle to the center of the projection.
  - *viewWidth and viewHeight,* two real numbers that give the dimensions of viewing window on the image plane.

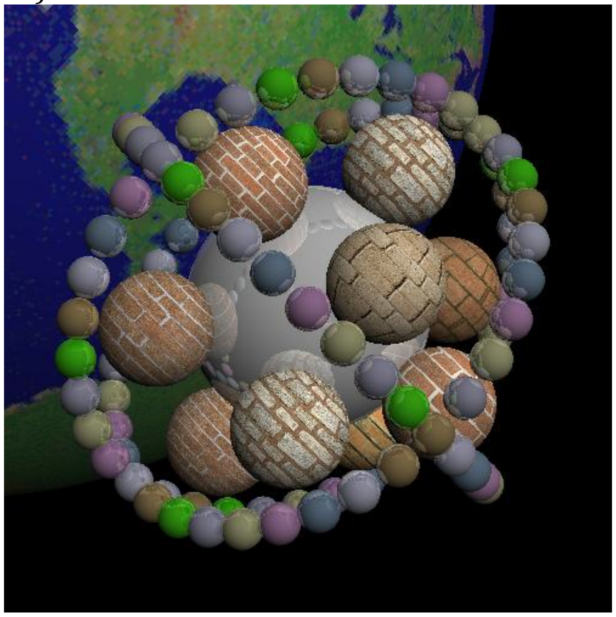
  The camera's view is determined by the center of projection (the viewpoint) and a view window of size *viewWidth* and *viewHeight*. The window's center is positioned along the view direction at a distance d from the viewpoint. It is oriented in space so that it is perpendicular to the image plane normal and its top and bottom edges are perpendicular to the up vector.
- *Image:* This element is just a pair of integers that specify the size of the output image in pixels.
- *Light:* This element describes a light. It contains the 3D point position and the RGB color *color*.
- *Shader:* This element describes how a surface should be shaded. It must have an attribute type with value Lambertian. The Lambertian shader uses the RGB color *diffuseColor*. A shader can appear inside a surface element, in which case it applies to that surface.

Your ray tracer framework:
Your ray tracer may contain a parser (the instructor strongly encourages you to write a lib to do this), a vector calculator lib (using C++ will make your life easier since its vector class handles addition, multiplication, and dot produce already), and your ray tracer.

Start **now**! Or you will probably not finish. Really, I promise you will not be able to do it in the last two days.

Extra credits

- For 15 points, render your scene with texture (define a *texture* tag in your input file.)



http://www.cs.unc.edu/~rademach/xroads-RT/RTarticle.html

- For 15 points, render your scene with refraction effects (define a *refraction* tag in your input file.)
- For 15 points, generate some interesting geometries or material or texture.

What to turn in

Source code only by email to TA. Please do not include any .o files. Please include:

- A README with your handin containing basic information about your design decisions and any known bugs or extra credit;
- How to compile and run your code as if you are telling a colleague that is to continue the development.

Note: Please comment on your code. The better Alisa understands your code, the higher your grade is likely to be.