Ju Young (Amy) Lee

CSCI 480 Spring 2013
Design Document Template

Purpose/Overview
The purpose of the assignment is to build a ray tracer. The ray tracer will be able to handle opaque surfaces with lighting and shadows. This requires the understanding of Phong shading, global illumination, and sphere-triangle intersections using barycentric coordinates. The user can specify which files to read which has information of the light sources, types of objects, and its corresponding color, position, diffusion, and other light properties. Upon completion, jpeg images are stored to display the ray trace.

Requirements
This is taken directly from the assignment 3 document.

- Triangle intersection (20 points)
- Sphere intersection (20 points)
- Triangle Phong shading (15 points)
- Sphere Phong shading (15 points)
- Shadows rays (15 points)
- Still images (15 points)

The levels

- Uniformly send out rays from the camera location. Since the camera does not have to move, you can assume that its location is (0,0,0). You should use backwards ray tracing where rays are sent from the camera, one ray per pixel. The final images should be 640x480, but for debugging you should use smaller resolutions with faster rendering times.
- Write the intersection code.
- Implement the illumination equations.
- Create still images

Instructions
The required source, header and resources (image files) are all included in the folder assign3, along with the visual studios (2012) project called assign3.

The initial text file chosen in the command argument is "test2scene.txt". This is the simple triangle, sphere ray trace example.

There are three other text files for the texture mapping. They are "test1file.txt" and "spheresfile.txt"

The images are saved in the same folder. They are named the same as its text file.

Classes
No additional classes were created.
The important source and headers, however, are:

Assign3.cpp : the main program that creates and displays the ray trace.

pic.h: creates a struct that contains the width, height of the pixel and the actual data in an array, and the bpp (the rgb channel).

Global Variables
These are new global variables that were created.

There are two defined variables (*fov, PI)*. *fov* is the field of view which is set to 60. *PI* is the mathematical pi.

There is one enum variable (*PLANEYTYPE*). This helps with the 2d barycentric coordinates. The types are XY, YZ, XZ for each 2d planes.

There is a struct (*Point)*. It has three varibles (*x, y, z)* that are doubles.

There are three Point variables (*lightP, interP, sumColor)*. *lightP* is the position of the shadow ray vector. *interP* is the position of intersections which get updated whenever we find a closer intersection from the view ray. *sumColor* contains the colors of the x, y, z coordinate for the pixels.

All other global variables had been defined in the starter code.

GUI

No GUI in this assignment.

Functions
New functions that were added.

void sphereIntersect (int x , int y, double x0, double y0, double z0, double *viewRay, double &dist)	part of the rayTracer function. This passes the camera ray, the pixel location (x,y) and the origin of the ray (x0, y0, z0), and updated distance (dist) to figure out which object has the closest intersection. This is specific to the intersection regarding spheres and the camera ray.
void triangleIntersect(int x , int y, double x0, double y0, double z0, double *viewRay, double &dist);	similarly to sphereIntersect, all the parameters are the same. This computes the intersection between the camera ray and triangles (planes) All the methods of computation is found from the lecture slide.
Void shadowSphereIntersect (int x, int y, double x0, double y0, double z0,double *viewRay, double &dist, int i, double lightMag);	This is part of the shadowrayTracer function. This computes the intersction of spheres to shadow rays. Only difference is, it also passes in the integer *i*, a counter of *num_lights.* Also, *lightMag*, which is the magnitude (length) of the shadow ray.
void shadowTriangleIntersect (int x, int y, double x0, double y0, double z0, double *viewRay, double &dist,int i, double lightMag);	Same as *shadowSphereIntersect*(). Deals with the intersection between triangles (planes) and the shadow ray.
double calculateArea(Vertex V1, Vertex V2, Vertex V3,PLANETYPE rt_planetype);	Arithmetic function that calculates the area of a plane. Based on the planetype given, it computes the correct alpha, beta, and gamma numerators. Area(ABC) = (1/2) |(B – A) x (C – A)|
void normalizeVector(double &X, double &Y, double &Z, double &norm);	Arithmetic function that normalizes vectors and finds the magnitude.

.,