**Report Part B**

The raytracer program extended from part A starter codes. All newly implemented features and their location in the code blocks are shown in red in the diagram below. A global variable “\_render\_mode” is used to enable/disable most features. For example “\_render\_mode = (mode)(MODE\_FULLPHONG | MODE\_SHADOW)” will enable full phong lighting model and shadow.

getRandomPt

- generate pts for distributed rays

“materials.h”

- define textures

- read-in textures

computeShading

- after traverseScene called

- shadow (single ray)

- soft shadow

light::shade

- after traverseScene called

- scene signature

- phong lighting model

traverseScene

- check all objects

- set ray.intersect

SceneObject::intersect

- check intersection to objects

- compound object subclass

main

- “\_render\_mode”

- Setup scenes

render

- multi-threading

- output .bmp

render\_section

- super-sample antialiasing

shadeRay

- reflections

- refractions

- stores RGB buffer

getReflectRay

getRefractRay

getReflectRayColor

SceneObject::textureMapping

- texture maping

- each object type defines its own

Hyperboloid: CompoundObject

\_Circle: SceneObject

\_Hyperboloid: SceneObject

Ray.col

Ray.intersect

shadowRay. intersect

**Basic Feature List:**

1.) Shadows (MODE\_SHADOW)

2) Reflection (MODE\_REFLECTION)

**Additional Feature List:**

1.) Multi-threading (MODE\_MULTITHREAD)

Multi-threading divides the whole rendering process into 8 separate sections, each handled by an individual thread. Several global and shared variables implementation had to be changed to be thread-safe. The RBG buffer arrays are shared as each thread is accessing different section (index) of the arrays and will not interfere with each other. The write-out of buffers to bmp and the read-in of bmp (for textures) are not multi-threaded, as it would be difficult to implement and probably with little performance improvement limited to disk access.

Performance of multithreading on a 8-cores AMD-FX8320 is about 4x to 5x the speed of single thread, depending on the complexity of the rendered scene.

2.) Anti-aliasing (MODE\_SSAA4, MODE\_SSAA16, MODE\_SSAA36, MODE\_SSAA64)

3.) Compound object (hyperboloid)

4.) Refraction (MODE\_REFRACTION)

5.) Texture mapping

Texture map are defined to a specific material struct. Material struct has a field for the filename of its associated bmp texture, and during initialization the RGB data is read and stored.

Each scene object type must define its own “textureMapping” function. For example, UnitSphere maps it coordinate by interpolating an intersection’s vertical and horizontal (with respect to +x axis) angles to the height and width of the texture, respectively.

“LightSource::shade” will first call each object’s mapping function to get a base color if the object is indicated to be textured, and then do the regular Phong lighting. The texture base color is then multiplied to the Phong lighting as final ray color.

6.) Soft shadows (MODE\_SOFTSHADOW\_LOW, MODE\_SOFTSHADOW\_HIGH, MODE\_SOFTSHADOW\_EXTREME)

Soft shadow is implemented by shooting numerous shadow-rays with randomized directions near the intersection-to-light direction, from each ray’s intersection. The 3 modes generate 10, 50, and 500 shadow-rays. The randomized directions are within a given radius (0.3) of the point light position, on a plane that is always perpendicular to the intersection-to-light vector. Coordinates are generated using sine and cosine of randomized radian angles times randomized radius within given limits. The percentage of shadow-rays that is not obstructed is passed to LightSource::shade for coloring the original ray.

To save computation time, if the first 15 shadow-rays is not obstructed, assumes the intersection is not in shadow and the code stops further shadow-rays checking. Similarily, it also stops if the first 40 checks are all in shadow.