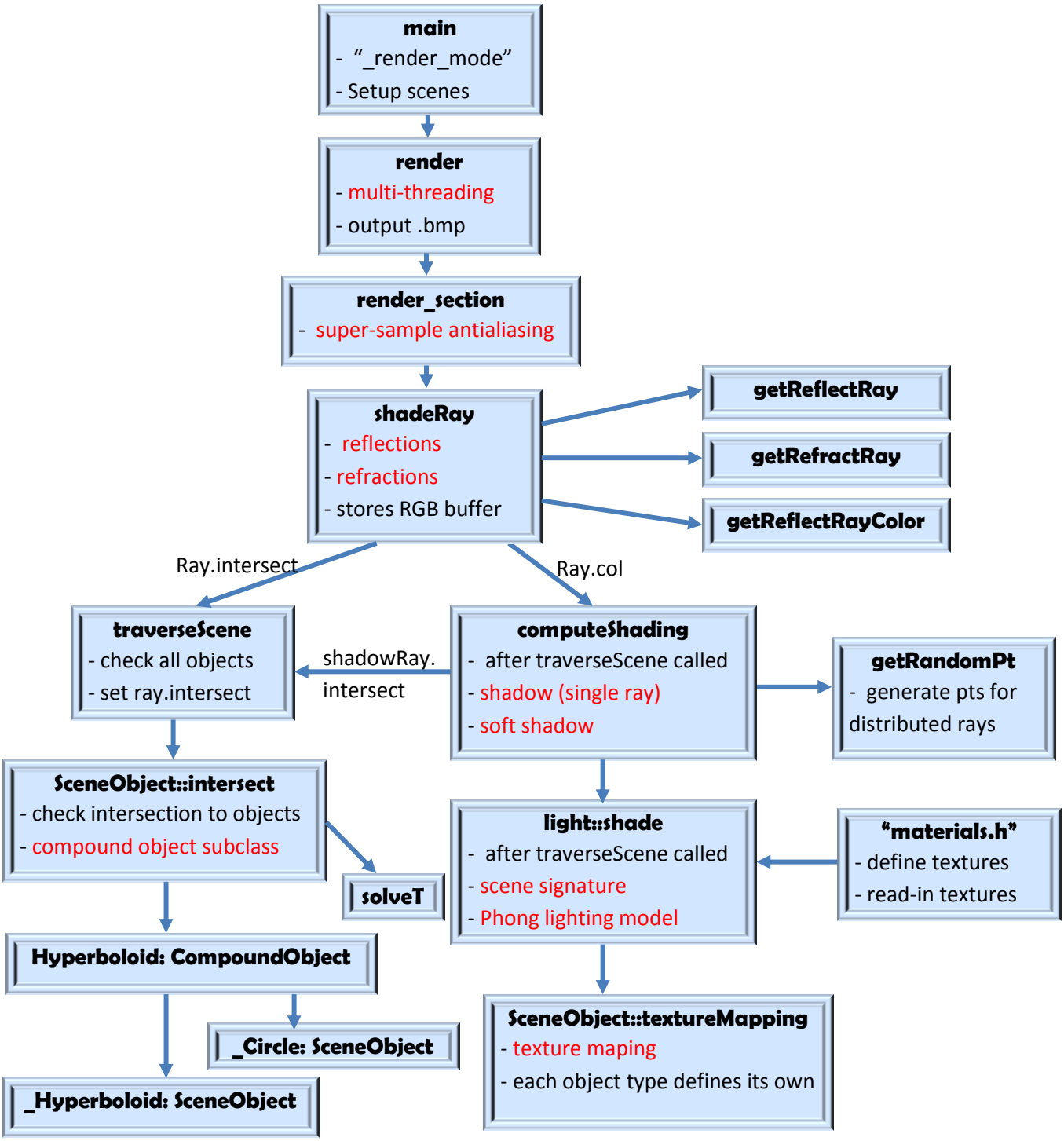
Report Part B

The raytracer program is 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.



Basic Feature List:

1.) Shadows (`MODE_SHADOW`)

Basic shadow is implemented by casting a shadow-ray from the point of ray intersection with an scene object, to the light position. If the shadow-ray intersect any scene object from "traverseScene", including the original object ray intersected with, the intersection point is under shadow and receive no diffuse and specular lighting.

2.) Reflection (`MODE_REFLECTION`)

The maximum recursive level of reflection is controlled by the variable "_reflection_depth". Reflection itself is implemented by casting a reflect-ray from point of ray intersection toward its specular reflection direction, and adding the reflect-ray shaded color to the original ray color. Perfect "mirror" is implemented by adding the reflect-ray color without any attenuation. Other surfaces imposes an attenuation factor which is an exponential function of negative t_value (Beer's Law).

Additional Feature List:

1.) Multi-threading (`MODE_MULTITHREAD`)

Multi-threading divides the whole rendering process into 8 separate sections. Several global and shared variables had to be changed to be thread-safe. The image to be rendered is "cut" into 8 vertical strips and feed to each thread for processing. 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 overhead.

Performance of multithreading the ray tracer program 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`)

Super sampling (stratified) anti-aliasing is implemented at 4x, 16x, 36x, 64x rays per pixel. Each pixel is divided into the specified number of retangular regions, and a randomized coordinate within each region is used to generate the starting ray for tracing.

Noticeable improvement in image quality can be seen going from no anit-aliasing, to SSAA-4x, and then to SSAA-16x. The higher 36x and 64x sampling have little to no difference in noticeable image quality.

3.) Compound object (hyperboloid)

A compound object subclass is created to inherit from SceneObject class. It has an object count and a pointer for an array of SceneObjects. For hyperboloid, it contains an open ended hyperboloid quadratic surface, and two flat circular planes to close off the ends of hyperboloid. When "intersect" function of a compound object is called, it simply calls the "intersect" function of each containing objects. The intersection with the smallest t_value is stored to ray.intersect struct.

The hyperboloid surface is implemented similar to UnitSphere, by solving the t_value of the quadratic equation.

4.) Refraction (`MODE_REFRACTION`)

Refraction is difficult to implement from uni-directional ray tracing (more so when it wasn't planned ahead, as it significantly affect reflection and coloring). An object is marked as refractive by using a material type that has defined refractive index. The current implementation is a simplified model such that a refractive object will still cast shadow as if it's opague. Also light intensity is not attenuated within an refractive path.

Briefly, when a ray hits an object surface, the side (inside or outside) of the intersection is determined by the dot product of ray direction and surface normal. If ray hit outside of surface, a separate reflective and refractive rays will be casted out, assuming both features are enabled. The angle of the refractive ray is calculated using Snell's Law based on the object's refractive index. Similarily when ray hit inside of an object surface, except in this case total internal reflection may occur depending on incident angle.

Water, glass and diamond material types with their refractive index are included in the codes.

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" first calls 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 component 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 soft shadow modes generate 10, 50, and 500 shadow-rays. The randomized directions are within a given radius (preset at 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 multiplied by randomized radius length 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.

Sample Images:

AA_00.bmp – no anti-aliasing
AA_04.bmp, AA_16, AA_36 – anti-aliasing at 4x, 16x, and 36x sampling rate

Hyperboloid01bmp, Hyperboloid02.bmp – compound object hyperboloid

Reflect00.bmp – reflect depth = 4, and note distance-based attenuation of walls edge reflection
Reflect01.bmp – objects and camera in between two panel of glasses, projecting reflection of reflections.
Reflect02.bmp – same as above but with "perfect" glass and 400 levels of recursive reflections.

Refract01_water.bmp – 2 spotlights, a sphere with material type of water, refractive index = 1.33
Refract02_glass.bmp – material glass, refractive index = 1.6
Refract03_diamond.bmp – material diamond, refractive index = 2.24

ShadowHard.bmp – basic one ray shadow
ShadowSoft01Low.bmp – soft shadow at low level, 10 shadow-rays per incident ray
ShadowSoft02Extreme_SSAA16.bmp – soft shadow at extreme 500 shadow-rays each, with 16x SSAA just for kick!

Texture.galaxy.bmp – scene of textures, with basic shadow off 9 tightly packed point-lights.

Zzerror.bmp – cool pic from calculation error…

References and Sources

Lecture notes, course online notes, tutorial notes, and textbook.
Images from Google search.