Ray Tracing Documentation

First, there is definitions of point3d struct with three float fields representing x, y, and z coordinates and ray struct that contains two pointers to Point3d objects, representing the endpoints of a ray.

Then some helper functions performing vector math on Point3d objects, including subtracting, calculating the dot product and cross product of two vectors, and normalizing a vector.

The main functions:

1. rotation_transpose: It computes the transpose of the rotation matrix that would align the vector from p2 to p3 with the positive z-axis.
It does this by first determining the normal n of the vector from p2 to p3, then determining the vector u that is orthogonal to both n and the vector from p2 to p3, and lastly determining the vector v that is orthogonal to both n and u. The rotation matrix that results is then placed in a matrix.

2. ray_construction: With an image pixel position and camera settings, this function generates a ray in 3D space.
Pixel location (i,j), focal length, picture boundaries (xmin, xmax, ymin, ymax), a camera transformation matrix Mcw, and a viewpoint in world coordinates are all input parameters (vrp).
The result is a Ray object with the origin point (ray->a) and a normalized direction vector (ray->b).
The function first uses the camera transformation matrix and focal length to map the pixel position to a 3D point in camera coordinates, and then uses the viewpoint to convert that point to world coordinates.
The difference between the 3D point and the perspective is used to construct the direction vector, which is then normalized.
The Ray object stores the resultant origin and direction.

3. ray_sphere_intersection: This function computes the point at which a ray intersects a sphere. It accepts a Ray and an SPHERE as input, as well as pointers to a Point3d to store the sphere's normal at the intersection, a Point3d to record the intersection location, and a float pointer to save the sphere's material attribute. It returns the distance from the ray's origin to the junction point, or 0 if no intersection exists. The function first computes the intersection equation's quadratic coefficients A, B, and C, and then determines if the discriminant D is positive.
If D is positive, the function computes the intersection point, the normal of the sphere at the intersection point, and the material property, and then returns the distance from the ray's origin to the junction point. The function returns 0 if D is not positive.

4. find_project_plane: This function accepts an array of length 3 representing the surface normal, a 2D array of size 4x3 representing the four vertices of a quadrilateral in 3D space, and a 2D array of size 4x2 containing the vertices' 2D coordinates projected onto a plane perpendicular to the surface normal.

5. point_in_polygon_test: The point-in-polygon test determines if a given point is within or outside of a polygon defined by a collection of vertices. It accepts as input the number of polygon vertices (number of vert), arrays of the vertices' x and y coordinates (vert x and vert y), and the x and y coordinates containing the test point (test x and test y). The function then runs around the polygon's edges, checking if the test point is to the left or right of the edge. If the test point is to the left of an odd number of edges, the polygon is within, and the method

returns 1. If the test point is to the left of an even number of edges, it is outside the polygon and yields 0.

6. ray_polygon_intersection: This function computes the intersection of a ray with a four-vertex polygon. The function accepts a Ray and a POLY4 structure as input (which contains the coordinates of the four vertices and the polygon's normal vector) and returns the distance between the ray origin and the intersection point, or 0 if no intersection exists. The function also provides the polygon's surface normal, intersection point, and diffusion coefficient. It initially computes D and the point of intersection of the ray with the polygon. The intersection point is then projected onto a plane specified by the polygon's normal vector, and the projected point is checked to see if it is within the polygon. If so, the function returns the distance between the origin of the ray and the intersection point. If not, it returns 0.

7. ray_object_intersection: In a 3D environment, this function computes the intersection point of a given ray and either a sphere or a polygon object.
It yields the distance between the ray's origin and the intersection point, as well as the object's normal at the intersection point and the intersection pointolated color at that point.
It initially uses two distinct helper functions to check for an intersection with a sphere and a polygon object.
If neither intersection happens, the method returns 0.0, indicating that there is no intersection.
If just one junction occurs, the method returns the distance to the intersection and sets the normal and intersection point to the values provided by the appropriate helper function.
If both intersections happen, the method returns the intersection with the shortest distance based on which intersection happened first.
The function additionally assigns the nearest intersection's normal and intersection point to the output parameters n and intersection point, as well as the intersection pointolated color to kd.

8. shading: The light source position lrp, the normal vector n, the intersection point p, the diffuse coefficient kd, and the light intensity ip are all inputs into this function. The shading value is calculated by taking the dot product of the normalized light vector and the normal vector and multiplying it by the diffuse coefficient and the light intensity. If the cosine is negative, the shading value is set to zero, indicating that the surface is not lighted since it is facing away from the light. After that, the shading value is returned as an unsigned char.

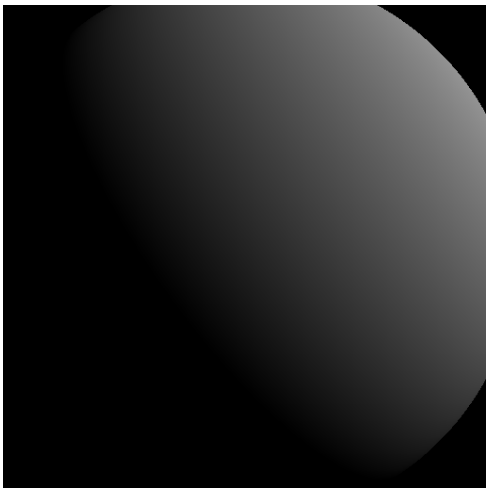
8. ray_tracing: The ray_tracing function accepts a Ray, two shapes, an SPHERE and a POLY4, as well as a light location lrp and intensity.
The function initially calculates the ray's intersection point P, normal vector n, and intersection point with the first object it intersects (either the sphere or the polygon). If the intersection is valid (i.e., P is greater than zero), the function computes and returns the shading color C at the intersection point using the shading function. The method returns 0 if the intersection is not valid.

9. Main: the main function that loads the scene from mymodel.h and does ray tracing on each pixel in the picture plane.
It creates rays for each pixel and initializes transformation matrices. The ray tracing function is then used to identify the color at the junction point of each ray with the scene. The findings are saved in an image buffer before being written to a binary file.

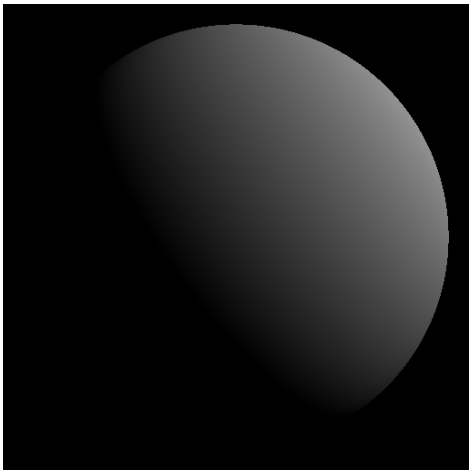
The code is tested using the default parameters in the model.h and then some parameters were changed to create different scenes. The final included model.h is same as the file provided for the assignment.

Results:

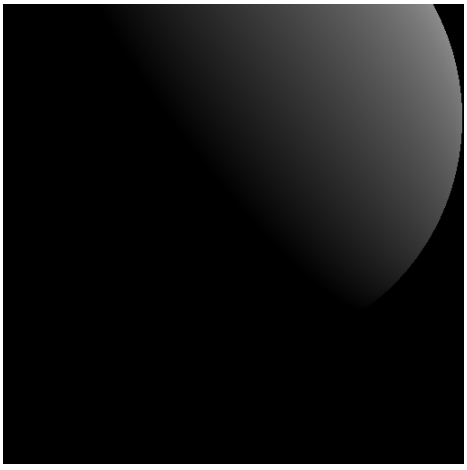
1. With default values in the provided model.h file (corresponding to ray_default.raw).



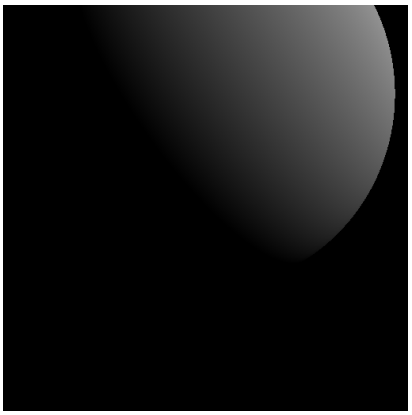
2. With changing the focal length simulating lens to 40mm. (Corresponding to ray_1.raw)



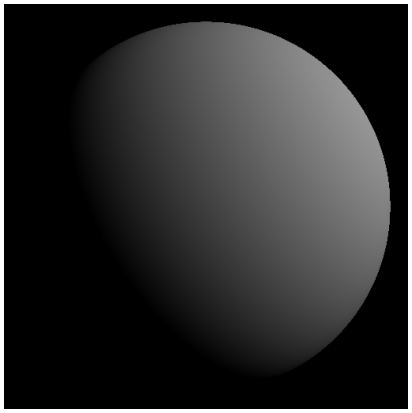
3. With changing VRP to {1.0, 1.5, 3.5} with 40mm focal lens. (Corresponding to ray_2.raw)



4. With changing VPN to {0.0, -1.0, -1.5} with 40mm focal lens and other default variables. (Corresponding to ray_3.raw)



5. With changing LRP to {-10.0, 10.0, 5.0} with 40mm focal lens and other default variables. (Corresponding to ray_4.raw). We can see that it is brighter than 2.



6. With moving the polygon 1 unit to the right by adding 1 to the x-coordinates of all its vertices, with 40mm focal lens, VPN[0] = 1.0 and other default variables. (Corresponding to ray_5.raw).