CENG415 Applications of Computer Graphics – Assignment 3

It is time to implement a ray tracer. Until now, you send a ray and didn't trace it. Now, you will trace the reflected and refracted rays. We also need new material properties.

What to do:

- Phong Material: Implement an abstract Material class and PhongMaterial class that extends Material.
- Objects will have material instead of color. Also hit will store material instead of color.
- Implement ray tracing.
    - Send shadow rays to light sources
    - Trace reflected rays for reflective objects
    - Trace refracted rays for transparent objects
- Store multiple lights. For that implement an abstract Light class and a DirectionalLight class.
- Provide a readme file. Explain how long it took to do the homework, where you had difficulty, and how to run your program.

New classes to write:

Light: This is an abstract class for all types of light.

Attributes:

color: color of the light, a vector of 3 values.

DirectionalLight: A class to store attributes of a directional light source. This is a child of Light class.

Attributes:

direction: light direction, a vector of 3 values

Material: Another abstract class for all material types

Attributes:

diffuseColor: a vector of 3 values indicating diffuse color
reflectiveColor: a vector of 3 values indicating reflective color
transparentColor: a vector of 3 values indicating transparent color
indexOfRefraction: refraction index of the object. Effective for transparent objects

Functions:

shade(ray, hit, light): abstract method for shading calculation. Returns an rgb color value.

PhongMaterial: This is a subclass of Material.

Attributes:

specularColor: a vector of 3 values indicating specular color
exponent: a floating value for specular exponent.

Functions:

shade(ray, hit, light): This function takes, a ray, hit, and light properties to calculate shading from a single light source.

Updated classes:

Hit:

Hit will store a material instead of color.
material: material of the intersected object.

Object3D:

Instead of color, objects will have material

Explanations

Multiple Lights: You will add contribution of each light to compute final lighting. So you need to call shade function of the material for each light source. Each of them will return an rgb value and you will combine all of them.

Shade: Material class has shade function. For Phong Material, use the calculation given in course slides.
Phong model has three components: Ambient, Diffuse, and Specular. You used ambient and diffuse components in your previous assignment. Now you need to add specular component too.

$$P_{color} = Color_{ambient} + Color_{diffuse} + Color_{specular}$$
$$Color_{ambient} = ambientColor \times diffuseColor_{object}$$
$$Color_{diffuse} = \max(dir_{light}.N, 0) \times (diffuseColor_{object} \times color_{light})$$
$$Color_{specular} = \max(dir_{view}.R, 0) \times (specularColor_{object} \times color_{light})$$

Here R is direction of Reflection vector. R can be calculated as follows (check slides too):

$$R = N \times 2 \times (N.L) - L$$

Ray Tracing

To trace rays you may send three more rays at each intersection. For that purpose, first write a traceRay function. You can generate a RayTracer class or implement it directly in your main method. In any case to trace ray you need to access lights and scene geometry.

Given a ray, traceRay method computes the color seen from the origin along the direction. This computation can be recursive because of reflection and refraction. We therefore need a stopping criterion to prevent infinite recursion. traceRay takes as additional parameters the current number of bounces (recursion depth) and a ray weight that indicates how much the contribution will be dimmed in the

final image. You need to determine maximum recursion depth and the minimum ray weight. Note that the weight is a scalar that corresponds to the magnitude of the color vector.

Function:

traceRay(ray, tmin, bounces, weight, indexOfRefraction, hit)

First, implement traceRay so that it computes the closest intersection with the scene and performs shading. For ambient lighting, you should use the *diffuseColor* of the *Material* pointer stored by *Hit* and multiply it by the *ambient* light given in scene description. Add the contributions of the light sources in the scene by iterating over the light sources and call the *shade* method of the Material.

Shadows

Next, you need to implement cast shadows. Modify the code described above to test for each light whether the line segment joining the intersection point and the light source intersects an object. If there is an intersection, then discard the contribution of that light source. Recall that you must set *tmin* as a small positive value (e.g., 0.0001) to avoid self-shadowing. Note that in this naive version, semi-transparent objects still cast opaque shadows.

Mirror reflection

You will then implement mirror reflection. This involves the use of secondary rays in the direction symmetric with respect to the normal. If the material is reflective (reflectiveColor > (0,0,0)), then you must create a new ray with origin the current intersection point, and direction the mirror direction. For this, we suggest you write a function:

Function:

mirrorDirection(normal, incoming): returns mirror direction of incoming direction.

Trace the secondary ray with a recursive call to *traceRay* using modified values for the recursion depth and ray weight. The ray weight is simply multiplied by the magnitude of the *reflectiveColor*. Make sure that *traceRay* checks these stopping conditions. Add the reflected contribution to the color computed for the current ray. Don't forget to take into account the reflection coefficient of the material.

Refraction

Finally, you will implement the computation of transmitted rays. For semitransparent objects, you will trace a new ray in the transmitted direction. We suggest you implement a function transmittedDirection that given an incident vector, a normal and the indices of refraction, returns the transmitted direction.

Function:

transmittedDirection(normal, incoming, index_i, index_t): returns transmitted direction. Check course slides to see how to compute transmitted direction.

Be careful about the direction of the vectors and the ratio of indices. We make the simplifying assumption that our transparent objects exist in a vacuum, with no intersecting or nested refracting materials. This allows us to determine the incident and transmitted index of refraction simply by looking at the dot product between the normal and the incoming ray. Indeed, because we now consider transparent objects, we might hit the surface of a primitive from either side, depending on whether we were inside or outside the object. Note that the dot product of the normal and ray direction is negative when we are outside the object, and positive when we are inside. You will use this to detect whether the new index of refraction is 1 or the index of the hit object. (The index of refraction for the material surrounding the ray origin is passed as an argument to traceRay.) If you are inside the object, you must flip the normal before computing the transmitted direction. It is not necessary to flip the normal when computing the direction of the reflected ray (you'll get the same answer). For shading, use the original normal.

Recap

To summarize, the color returned by traceRay is the sum of the ambient contribution, the shading contribution of the visible light sources, the mirror reflection and the transmitted contribution.

Input files & sample outputs

Check input files to understand how to parse lights and materials. Be careful, this time, object descriptions have material indices instead of color.