Assignment 1d: Transparency and Mirror Reflections Due Date: Mon Nov. 4, 2019

In this assignment, you are asked to extend the raycasting program that you wrote for assignment 1c to compute the color at a ray/surface intersection point using *recursive ray tracing* with an extended version of the Phong illumination equation that additionally considers:

1) intensity that is specularly reflected to the ray/surface intersection point from other portions of the scene; and
2) intensity from other parts of the scene that is transmitted through the intersected surface, when that surface is transparent.

For 5% extra credit, you may extend your program to enable depth of field effects through the use of distributed ray tracing.

Detailed instructions:

- In order to extend the Phong model to represent transparent and mirror-like surfaces, you will need to know two additional parameters: the material's *opacity* α, and its *index of refraction* η. You will need to extend your scene input file to allow the specification of these addition material properties. Please use the following syntax for this:

mtlcolor $Od_r\ Od_g\ Od_b\ Os_r\ Os_g\ Os_b$ `ka kd ks n` $\alpha\ \eta$

- To keep things simple, your program may assume that all viewing rays originate from a point located in air (η `= 1`). [Note that if you wanted to create a scene in which, for example, the camera is under water, you would need to change that assumption.]
- In order to determine the intensity of the illumination that can be seen via mirror reflection at the ray/surface intersection point, you will need to recursively trace a ray from the ray/surface intersection point out into the scene in the *direction of reflection* and incorporate an appropriate portion of the color that this ray returns into the color that is computed at the ray/surface intersection point.
- In order to determine the intensity of the illumination that can be seen through a transparent surface at a ray/surface intersection point, you will need to recursively trace a ray from the ray/surface intersection point through the surface in the refracted *direction of transmission* and incorporate an appropriate portion of the color that this ray returns into the color that is computed at the ray/surface intersection point.

- Your program should use Schlick's approximation of the Fresnel reflectance to determine both the reflectivity of an opaque surface and the relative contributions of the intensities returned by the reflected and transmitted rays when a ray/surface intersection occurs with a transparent object. Be sure that your program correctly recognizes and handles situations of total internal reflection.
- Because the direction of a refracted ray depends on the index of refraction of the medium on *each side* of a surface, you will need to keep track of whether a ray is entering or exiting a solid object, and use an appropriately directed surface normal for the situation, as well as appropriately-defined angles of incidence, reflection, and transmission. For example, when a ray hits the inside of a sphere, you should use the negative direction of the outward-pointing surface normal; likewise, if a ray hits the backside of a triangle you should use the plane normal defined by the vertices in clockwise, rather than counterclockwise, order.
- If you want to model something like a hollow sphere (e.g. air bubble inside glass) you will need to keep track of the indices of refraction of all of the materials that a ray passes through as it travels around in your scene. You can use a stack to do this. Don't worry about trying to handle intersecting spheres robustly. That's actually a modeling problem, as real objects typically don't interpenetrate. The outer shell formed by multiple intersecting objects can be defined using Constructive Solid Geometry (CSG), but a full-fledged CSG implementation is outside the scope of what I want you to consider for this assignment. To keep your scene behaving intuitively, you are advised to model all surfaces or objects as solids (i.e. double-sided walls, with finite thickness) when rendering with transparency.
- You will need to take steps to enforce termination of the recursion. The simplest approach is to keep track of the recursive depth of each ray and stop when that depth exceeds a specified limit. An alternative approach would be to terminate recursion when the incremental change to the final pixel color would be smaller than a pre-defined tolerance.
- Because transparent surfaces allow light to pass through them, images tend to look more realistic when the shadows cast by transparent objects merely *reduce* the diffuse and specular components of the intensity at a point in shadow rather than completely *eliminating* them. You can approximate this effect by considering the opacities of *all* of the objects that your shadow ray intersects on its way to the light source and compositing their light-blocking effects to achieve a shadow flag whose values may fall

somewhere between 0 and 1. Clearly such an approach will not yield results that are physically correct, both because it does not properly consider light attenuation as a function of the density and volume of the intervening material and also because it ignores the effects of refraction, but it is beyond the scope of this assignment to attempt a more robust solution.

- To achieve depth of field effects: rather than tracing a single ray from the eye through an appropriate point in the viewing window to determine the color at a given pixel, you can trace *multiple* rays, each using a slightly jittered offset from the eye position, and then average the colors that are returned. Note that with this approach, the rays will be focused at the viewing window; objects closer or farther than your defined 'viewing distance' will appear out of focus. This is the only situation in which the distance between the eye and the viewing window will make a difference to the output of your raytracer. In order to achieve different depth-of-field effects, you will need to be able to vary this distance. You can do this by adding another keyword and user-defined parameter to your input file format:

  `viewdist` *`d`*

What you should turn in

- all of your source code, with detailed comments and appropriate instructions on how to compile your program
- one or more input files containing scene descriptions that are handled by your program; please be sure to define these files to show off all of the capabilities of your program
- one or more output images showing the results of running your program using your provided scene description files as input