Worksheet 3

The subject of this third worksheet is primarily reflection and refraction/transmission. The fact that light is electromagnetic waves is surprisingly important in this context. The complex index of refraction provides a way of accurately capturing the appearance of metals (conductors) and transparent objects such as glass (dielectrics). Fresnel's equations for reflection tell us how much light is reflected and how much it is refracted into the material. The imaginary part of the refractive index also tells us how much light is absorbed once inside the material. In the following, we will use these properties of the complex index of refraction to accurately render metals and transparent objects. So, the Fresnel equations and the recursive nature of reflection and transmission is what you will be working with in this set of exercises.

Learning Objectives

- Implement recursive ray tracing.
- Explain the trade-off in splitting versus Russian roulette.
- Apply Russian roulette to do unbiased rendering.
- Use the complex index of refraction and the Fresnel equations to simulate the appearance of dielectrics and conductors in their pure form.

Ray Tracing

After introducing Lambertian (perfectly diffuse) reflection in Worksheet 1, your ray tracer now needs to include the ability to render perfectly specular objects.

- Load the Cornell box (`CornellBox.obj`) and two spheres (`CornellLeftSphere.obj` and `CornellRightSphere.obj`) into your ray tracer.
  (Course frameworks: the materials of the two spheres are mirror and glass, respectively, but they will appear black as long as shaders for mirror objects and transparent objects have not been implemented.)
- Implement a shader for mirror objects. Use the number of reflections (the trace depth) to stop the recursive ray tracing. (CPU framework: implement the `shade` function in `Mirror.cpp`. GPU framework: implement `__closesthit__mirror` in `shaders.cu`.)
- Change the material of the right sphere such that both spheres are perfect mirrors. Render an image and save the result.
- Change the material of the right sphere back to glass and implement a shader for transparent objects. Again, stop the recursion after a specific trace depth. Render the scene and save the result.
  (CPU framework: implement the `split_shade` function in `Transparent.cpp` using the `splits` variable as maximum depth and the `trace_refracted` function with a scalar output argument R to retrieve the Fresnel reflectance in you shader. GPU framework: implement `__closesthit__transparent` in `shaders.cu`.)

Russian Roulette

We can use Russian roulette to eliminate the error (bias) that eventually appears when you stop tracing rays at a certain trace depth (after a pre-determined number of light bounces). Russian roulette also prevents a combinatorial explosion in the number of rays that needs to be traced. The trade-off is an increase of high-frequency noise in the image. Thus, in some cases, splitting can be an advantage at the first (or first few) bounces before switching to Russian roulette.

- Pick a metal and apply it to the Stanford bunny. Most real-world material parameters are available as a spectrum (per wavelength). Convert a spectrum to RGB by using the CIE colour matching functions.
  (Course frameworks: an XML file format and some code has been developed to do the conversion to

RGB for you. The file format is called MPML (Material Properties Markup Language) and the loader is implemented in the files `load_mpml` and `Medium`. Take a look at the file `models/media.mpml` which is distributed with the framework. The idea is to build a database of material properties where you can add new materials whenever you find measured or computed properties in the literature. The format is simple. Try to figure out how it works. Pick a metal from `media.mpml`, feel free to add a new one, and apply it to a model by giving it a Wavefront MTL material with the name of the medium in `media.mpml`. Additional optical properties are loaded from `media.mpml` when the names match.)

- The imaginary part of the refractive index for metals is very large. This means that everything that refracts into a metal is almost immediately absorbed. Consequently, we can use an RGB Fresnel reflectance computation with the complex index of refraction to capture the appearance of metals. Implement a metal shader. Use Russian roulette to stop the recursion after a random number of light bounces. Use splitting for the first light bounce to reduce noise. (The frameworks use the metal shader for Wavefront MTL materials set to `illum 11`. CPU framework: implement the `shade` function in `Metal.cpp`. GPU framework: implement `__closesthit__metal` in `shaders.cu`.)
- Render the (metal) bunny in the environment from the previous worksheet.
- Load the elephant model (`justElephant.obj`) and set a flat background colour. Make sure the elephant has only perfectly transparent glass material (use `illum 4` for all three wavefront OBJ materials). Implement Russian roulette and splitting using Fresnel reflectance. Render the elephant with maximum trace depth 1 (reflection only), 2, 10, and 200. Your implementation works if the elephant disappears. Render the elephant in the environment from the previous worksheet. (Revise the shader functions for transparent materials.)

Worksheet 3 Deliverables

Cornell box images (two mirror balls, one mirror ball and one glass ball). Glass elephant on flatly coloured background with maximum trace depth of 1, 2, 10, and 200. Metal bunny in environment and glass elephant in environment. Any new materials that you might have added to `media.mpml`. Include relevant code. Please copy everything into your lab journal.

Reading Material

The curriculum for Worksheet 3 is

- Frisvad, J. R. Electromagnetic Radiation. In *Light, Matter, and Geometry: The Cornerstones of Appearance Modelling*, Chapter 4. PhD thesis, Technical University of Denmark, May 2008.

P Section 8.2. *Specular Reflection and Transmission*.

P Sections 13.7. *Russian Roulette and Splitting*.

Alternative literature:

- Dutré, P. *Global Illumination Compendium*. Lecture Notes, Katholieke Universiteit Leuven, September 2003. https://people.cs.kuleuven.be/~philip.dutre/GI/
(See Section VII: Optics.)
- Arvo, J., and Kirk, D. Particle transport and image synthesis. *Computer Graphics (Proceedings of ACM SIGGRAPH 90) 24*(4), pp. 63-66, August 1990. https://doi.org/10.1145/97879.97886