Week 4

The complex index of refraction (introduced in Week 2) provides a way of accurately capturing the appearance of metals (conductors) and transparent objects such as glass (dielectrics). The Fresnel equations 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.

Ray Tracing

The main objective in the ray tracing part is to show how 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 need to be traced. The trade-off is an increase of high-frequency noise in the image. Therefore one should always consider splitting rays at the first (or first few) bounces before switching to Russian roulette.

- Most real-world material parameters are available as a spectrum (per wavelength). To translate a spectrum into RGB, we need the CIE color matching functions discussed in Week 1. An XML file format and some code has been developed to do this translation for you. The file format is called MPML (Material Properties Markup language) and the loader is implemented in the `load_mpml` and `Medium` files of the framework. Take a look at the file `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 literature. The format is simple, but still undocumented. Try to figure out how it works. Pick a metal from `media.mpml` (feel free to add a new one) and apply it to the Stanford bunny model (`bunny.obj`). (In the framework, names of the wavefront OBJ materials will be matched to names in `media.mpml`. This means that you can load the optical properties for your materials by ensuring that these 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. (In the `pathtrace` project of the framework, implement the function `fresnel_R` in `fresnel.h` which computes the Fresnel reflectance for complex indices of refraction. And implement the `shade` function in `Metal.cpp`. The framework uses the metal shader for wavefront OBJ materials set to `illum 11`.)
- Render the (metal) bunny inside the Cornell box (`CornellBox.obj`) and save the resulting image.
- Load the elephant model (`justElephant.obj`). Make sure the elephant has only perfectly transparent glass material (use `illum 4` for all three wavefront OBJ materials). Render the elephant three times with maximum trace depth 1 (reflection only), 2, and 10 respectively. Save the three images. Replace the maximum trace depth by an implementation of Russian roulette and splitting using Fresnel reflectance. Your implementation works when the elephant disappears. Load the elephant together with the (metal) bunny into the Cornell box and render the scene. Save the resulting image. (In the framework, revise the `shade` and `split_shade` functions in `Transparent.cpp`.)

Real-Time Shading

Russian roulette does not make much sense in real-time shading, but we can upload the complex index of refraction to a shader and use the same RGB Fresnel reflectance computation to render the appearance of metals. Keep in mind, however, that shader languages do not have a complex number data type. The

Global Illumination Compendium [Dutré 2003, §58, see reference below] provides a convenient formula for computing Fresnel reflectance with complex refractive indices when a complex number data type is not available.

- Implement a real-time metal shader. The RGB complex index of refraction for the metal is uploaded as three vectors each with two floating point values. (In the `realtime` project of the course framework, implement the fragment shader in `Metal.cpp`.)
- Load the Stanford bunny (`bunny.obj`) and the Cornell box (`CornellBox.obj`) into your real-time program. Make sure that the bunny still has a metal as its material. Render the scene and save the resulting image.
- For comparison to the ray traced image, load the elephant (`justElephant.obj`) into the Cornell box with the bunny and render it using your single refraction shader for transparent objects (implemented in Week 2). Save the resulting image.

Week 4 Deliverables

Two Cornell box images with a metal bunny (one ray traced, one real-time shaded). An updated `media.mpml` if you added a new metal to it. Three glass elephant images with maximum trace depth of 1, 2, and 10 respectively. Two Cornell box images with the same metal bunny and a glass elephant (one ray traced, one real-time shaded). Discuss the differences between off-line and real-time rendering. Both differences in implementation difficulty, output quality, and generality. Include frame rates for the real-time results. Answer the following questions:

Why is Russian roulette an unbiased method?

Using Russian roulette increases noise. Why?

Will the shaders that use Russian roulette instead of a maximum trace depth work in all cases?

Reading Material

The curriculum for Week 4 is

P Sections 15–15.1. *Russian Roulette and Splitting*.

You should also review **P**: Section 9.2, which was part of the curriculum for Week 2.

Alternative literature available online or uploaded to CampusNet:

- Philip Dutré. *Global Illumination Compendium*. Lecture Notes, Katholieke Universiteit Leuven, September 2003. http://www.cs.kuleuven.ac.be/~phil/GI/
  (See Section VII: Optics.)
- James Arvo and David Kirk. Particle Transport and Image Synthesis. *Computer Graphics (Proceedings of ACM SIGGRAPH 90)*, Vol. 24, No. 4, August 1990.

JRF 2010