Worksheet 2

Local illumination, like the result from the first worksheet, is rendered more quickly and in equal quality using rasterization techniques. However, the moment we start looking at global illumination effects (shadows, reflections, refractions, etc.), ray tracing has a number of advantages. Rasterization techniques for global illumination effects work well for some special cases only,[1] whereas ray tracing is simpler to implement and works well in general.

Learning Objectives

- Implement ray tracing.
- Render hard shadows by tracing shadow rays to a point light.
- Render reflections and refractions by tracing rays recursively.
- Compute shading of surfaces using the Phong illumination model.

Ray Tracing

Continuing the exercises of the first week, we will in the following extent our ray tracer such that it also supports shadows and specular surfaces.

- Render hard shadows by tracing shadow rays to point lights. Modify the function `sample` in the file `PointLight.cpp` such that it traces a shadow ray from the considered surface point to the position of the point light. Return false if the ray hits something. Store the render result.
- Implement mirror reflection and render the sphere in the default scene as a mirror ball. Do this by implementing the function `trace_reflected` in the file `RayTracer.cpp`. The sphere in the default scene has glossy material. Let the function `shade` in the file `Glossy.cpp` return perfect mirror reflection by calling `Mirror::shade` and render the default scene. Store the result.
- Implement refraction and render the sphere in the default scene as a glass ball. Do this by implementing the first of the overloaded `trace_refracted` functions in the file `RayTracer.cpp`. Note that there is a helper function called `get_ior_out`. Make a comment that explains how it works. Use `Transparent::shade` to return 90% refraction and 10% reflection from the glossy shader in order to approximate glass. Render the default scene and store the result.

Phong Reflection

You may have noticed that the point light in the default scene does not reflect in the glass sphere. The problem is that a point has no physical extent, and, thus, the reflected rays cannot hit the light source. The Phong illumination model provides a way of imitating the reflection of light sources in the surfaces of glossy (specular and/or diffuse) objects. To include this effect, we will add Phong reflection to the ray tracer.

- Implement the Phong illumination model. Do this by implementing the function `shade` in the file `Phong.cpp`. Let your glossy shader return the result from the `Phong::shade` function to examine the Phong model on its own. Store the render result.
- Combine the Phong illumination model with the glass shader to complete the glossy shader. Examine the glass shader function `Transparent::shade` in the file `Transparent.cpp`. Copy the glass shader to the function `shade` in the file `Glossy.cpp`. Modify it such that it also includes a local Phong component. Store the render result.

[1] As an example, shadow mapping works well if you do not zoom in too closely, since the map has a limited resolution.

Worksheet 2 Deliverables

Renderings of the default scene (e.g. shading of diffuse objects, mirror ball, glass ball, Phong ball, and glossy ball). Include relevant code snippets. Explain the helper function `get_ior_out` and your final glossy shader (`Glossy::shade`). Please insert all this into your lab journal.

Reading Material

The curriculum for Worksheet 2 is (7 pages)

- **B** Sections 4.5.3-4.5.4 and 5.2.2: *Shadows*, *Mirror Reflection*, and *Specular Reflection*.
- **B** Section 14.3. *Smooth Dielectrics*.

JERF 2022