Physically Based Rendering

From Theory to Implementation

Contents

Introduction

Physically based rendering techniques attempt to simulate reality and utilize physics to model the interactions with light.

- Literate Programming
- Photorealistic Rendering and the Ray-Tracing Algorithm
- pbrt: System Overview
- Parallelization of pbrt
- How to Proceed through This Book
- Using and Understanding the Code
- A Brief History of Physically Based Rendering

Literate Programming

Literate programming is a system where the documentation and the code are written in a single document, then a tool extracts and formats the documentation, and a different tool extracts and compiles the code.

Each function can be deconstructed into fragments of the form `<Fragment Name>`. Then these fragments can be referenced later in book.

```
<Function Defintions> ==
  void InitGlobals() {
    <Initalize Global Variables>
  }
```

Then later in the documentation, when the variables are defined, we can then write

```
<Initalize Global Variables> == size = 13;
```

Then when another variable is defined, we are able to append that variable into the fragment like so

```
<Initalize Global Variables> += value = true;
```

Most of the code in the book is decomposed in this way, to produce more readable documentation.

Indexing and Cross-Referencing

Indices in the page margins give page numbers where the functions, variables and methods are defined. Induces at the end of the book collect all of these identifiers so that it is possible to find definitions by name.

Photorealistic Rendering and the Ray-Tracing Algorithm

Ray-tracing is the basis of photorealistic rendering, it follows the path of a ray of light as it interacts with the objects of the scene. Each ray-tracer must simulate at least the following properties.

- *Cameras:* A camera determine how and from where the scene is being viewed, many rendering systems generate rays starting at the camera.
- *Ray-object intersections:* It is necessary to determine when a ray intersects and object, and useful to also find the surface normal or its material. Most implementations have a method for testing multiple intersections at once, and finding the nearest.
- *Light sources:* Ray-tracers must model the distribution of light throughout the scene, including the locations of the lights themselves.
- *Visibility:* We must know wherever there is an uninterrupted path between a point and a light source. This is relatively easy for ray-tracers.
- *Surface scattering:* Each object must provide a description of its appearance, including how light interacts with the object's surface, and how it scatters light. This is usually parametrized, so that many different appearances can be modeled.
- *Indirect light transport:* Light can arrive at a surface after bouncing off of, or going though other surfaces, thus it is usually necessary to trace additional rays originating form the surface to capture this.
- *Ray propagation:* Rays of light will propagate differently between a vacuum, glass, smoke, fog and other mediums. We need to be able to model these appropriately.

Cameras

As with physical camera technology, we will begin by simulating a pinhole camera. For our case, we will place the film plane in front of the pinhole. In this case, we will refer to the pinhole as the *eye*. We will call the area that will be imaged by the eye that is in front of the film plane, as the viewing volume.

Now the process of determining the color at each point on the image begins. In a pinhole camera, the only light that effects the film, is the ray that travels in through the pinhole and hits the film. In our model of a camera, we will use the eye as the origin for a ray, and the vector from the eye to the film plane as the direction. Each of these rays will then correspond to the light for a single point in the image.

More complex models of a camera can be constructed, using lenses to simulate more modern cameras.