Contents

Chapter 1

Introduction

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

1.1 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.

1.1.1 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.

1.2 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.

1.2.1 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.

1.2.2 Ray-Object Intersections

Each time the camera generates a ray, the first task is to calculate which object, if any, that ray intersects first and where the intersection occurs. This will be the visible point for this ray, and we will want to simulate the interaction of light with the object at this point. To do this, we will test the ray for intersection against all object, and select the one that the ray intersects first. Given a ray r, we write:

$$r(t) = o + t\mathbf{d},$$

where o is the ray's origin, $\mathbb{d}$ is the ray's direction, and t is a parameter whose range is $(0, \infty)$. We obtain a point along the ray, by specifying a value for t.

It is easy to find the intersection between the ray r and a surface defined by an implicit function $F(x, y, z) = 0$. We substitute the ray equation into the implicit equation and solve for t.

The reset of the algorithm requires information of the material for the surface, at the point of intersection, and some geometric information such as the normal to the surface at the point of intersection $\mathbf{n}$.

Most scenes are constructed of many objects, and the brute force method of testing the intersection for all objects, quickly becomes slow. So a better method is to construct an *acceleration structure* that quickly rejects whole groups of objects that the ray will not intersect with.

1.2.3 Light Distribution

The ray-object intersection gives us point to be shaded, and some information about the local geometry. Out goal is to determine how much light is leaving this point in the direction of the camera, to do this, we must know how much light is *arriving* at this point.

Using light sources, we will determine the *differential irradiance* on a surface, from a given light source. This will be of the form

$$dE = \frac{\Phi \cos\theta}{4\pi r^2},$$

where Φ is the power associated with the light source, r is the distance of the point from the light source, and θ is the angle between the surface normal and the ray from the light source.

Since illumination is linear, scenes with multiple light sources are easily handled, by summing the contribution of each light source individually.

1.2.4 Visibility

The description of light distribution is lacking in the component of *shadows*. Each light source will only contribute to the point being shaded if the path between the point and the light source is unobstructed.

In a ray-tracer this is easily done using a *shadow ray*, which is a ray from the point in the direction of the light source. If the ray intersects objects with $t < r$, where r is the distance to the light source, there we can conclude that there is an obstruction.

1.2.5 Surface Scattering

We now have the location and the incident lighting. Now we must determine the amount of the incident light is *scattered* off of the surface, and back along the ray that was initially sent from the camera to this point.

Each object in the scene provides a *material*, which describes its appearance properties at each point on the surface. This description is given by the *bidirectional reflectance distribution function* (BRDF). This tells up how much energy is reflected from an incoming direction ω_i to an outgoing direction ω_o. We will write the BRDF at p as $f_r(p, \omega_o, \omega_i)$. Now computing the amount of light L scattered back towards the camera will follow this process:

```
for each light:
  if light is not blocked:
    incident_light = light.L(point)
    ammount_reflected = surface.BRDF(hit_point, camera_vector, light_vector)
    L += amount_reflected * incident_light
```

In this case L represents the *radiance*, a unit for measuring light that we will use much more.

The concept of BRDF can be generalized to transmitted light(BTDF). A function that describes general scattering of light is called a *bidirectional scattering distribution function* (BSDF). More complex yet is the *bidirectional subsurface scattering reflectance distribution function* (BSSRDF), which models light that exits a surface at a different point than it enters.

1.2.6 Indirect Light Transport

In the original paper on ray tracing the *recursive* nature was emphasized. It is key to be able to reflect off of a mirror surface, and recursively invoke the ray-tracing routine to find the light arriving at the intersected point.

In general the amount of light that reaches the camera from a point on an object is the sum of the light emitted by the object, and the amount of reflected light. The formalization of this is the *light transport equation*, which says that the outgoing radiance $L_o(p, \omega_o)$ from a point p in direction ω_o is the emitted

radiance at that point in that direction, $L_e(p, \omega_o)$, plus the incident radiance from all directions on the sphere $\mathcal{S}^2$ around p scaled by the BSDF $f(p, \omega_o, \omega_i)$ and a cosine term:

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{\mathcal{S}^2} f(p, \omega_o, \omega_i) L_i(p, \omega_i) |\cos \theta_i| \, d\omega_i$$

Whitted's algorithm simplifies this integral by ignoring all directions with the exception of the direction of light sources, and that of perfect reflection and refraction. Thus it turns this integral into a sum over a small number of directions. This method can be extended to capture more effects, by sampling many recursive rays near the perfect reflection.

We can always recursively trace a ray when we hit an object. This will provide very realistic effects. It also means that each image location is associated with a tree of rays, each ray in the tree can have a *weight* associated with it; this allows the modeling for not perfectly reflective surfaces.

1.2.7 Ray Propagation

So far we have assumed that the rays are traveling in a vacuum, but in most cases, there will be some presence of a *participating media*. A participating medium can do one of two things to a ray.

First a medium can *extinguish* light, either by absorbing it or by scattering it in a different direction. We can model this by computing the *transmittance* T between the ray origin and the intersection point.

Second a medium can add light along the ray. This can happen if the medium emits light, or if the medium scatters light from other directions back along the ray. We can find this quantity by evaluating the *volume light transport equation*.