CGI & 3-D Computer Graphics project

This project is about producing computer-generated images by solving the equations of light transport that arise from ray optics, a process known as physically-based ray-tracing. A typical one-semester project should cover up to about the end of section 3 of this document, i.e. coding and validation of a simple ray-tracing program, and associated write-up in a mathematical report. It is important not to copy material from this summary, but instead use references, both those provided and ones you have found yourself. The references in the 'Further Reading' section of pbr-book.org Chapter 1 provide a good overall history of this field, and the technical report 'Physically Based Lighting at Pixar' shows its relevance to modern animated movies.

In this project it is important to show (a) how the images generated are the solution of an equation (the Light Transport Equation), and (b) demonstrate that your code correctly solves this equation.

1 Light transport

General references on light transport include pbr-book.org (sections 1.2, 5.4, 5.5, 5.6), Veach (1998) chapter 2, Cohen & Wallace (1993) chapter 2, Kajiya (1986).

1.1 Radiometry

We generate images by modelling light emitted by and reflected from a set of objects $\mathcal{M} \subset \mathbb{R}^3$, which consists of two-dimensional surfaces (planes, spheres, etc.) embedded in three-dimensional space. A convenient way of describing the brightness this of light is the *radiance* L (pbr-book.org section 5.4, Veach section 3.4, Cohen & Wallace section 2.4). Specifically, the radiance of light at a particular point in space $\mathbf{x} \in \mathbb{R}^3$, travelling in a particular direction $\boldsymbol{\omega} \in \mathcal{S}^2$[1] and of wavelength λ is denoted $L(\mathbf{x}, \boldsymbol{\omega}, \lambda) \in \mathbb{R}^+$. The dependence of radiance on wavelength λ defines the colour of the light; this dependence is usually simplified by defining radiance L as vector with (non-negative real) components, $L = (L^r, L^g, L^b)$ that represent the brightness of red, green and blue light respectively[2].

1.2 Reflectance equation

If the point $\mathbf{x}$ lies on one of the surfaces in our scene ($\mathbf{x} \in \mathcal{M}$), and the unit normal to that surface at $\mathbf{x}$ is $\mathbf{n}$, then the radiance of the light reflected from the surface at $\mathbf{x}$ in direction $\boldsymbol{\omega}_o$, denoted $L_r(\mathbf{x}, \boldsymbol{\omega}_o)$, is given in terms of the incident radiance coming in to $\mathbf{x}$ from direction $\boldsymbol{\omega}_i$, denoted $L_i(\mathbf{x}, \boldsymbol{\omega}_i)$, by the reflectance equation (Cohen & Wallace equation 2.26, Veach equation 3.12)

$$L_r(\mathbf{x}, \boldsymbol{\omega}_o) = \int_{\mathcal{H}^2} f(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) L_i(\mathbf{x}, \boldsymbol{\omega}_i)(\boldsymbol{\omega}_i \cdot \mathbf{n})\, d\boldsymbol{\omega}_i. \tag{1}$$

The integral is over all possible incoming light directions $\boldsymbol{\omega}_i$ in the hemisphere $\mathcal{H}^2 = \{\boldsymbol{\omega} \in \mathcal{S}^2 : \boldsymbol{\omega} \cdot \mathbf{n} > 0\}$[3], and describes how the light reflected from a point in the scene is a combination of all of the light incident to this point.

[1] $\mathcal{S}^2 = \{\boldsymbol{\omega} \in \mathbb{R}^3 : |\boldsymbol{\omega}| = 1\}$ is the unit sphere, which can be thought of as the set of all possible directions in three-dimensional space.

[2] Since computer monitors can display only these three independent colour values, and since the dependence on wavelength of the path that light takes around a scene is often negligible (with the notable exception of wavelength-dependent refraction), this approximation is usually acceptable. More sophisticated ways of describing the dependence of L on wavelength are possible. See e.g. pbr-book.org chapter 5, Veach section 3.4.4

[3] The vectors $\boldsymbol{\omega}_o$, $\boldsymbol{\omega}_i$ and $\mathbf{n}$ all lie in the same hemisphere $\mathcal{H}^2$ that defines the side of the surface from which the outgoing ray travels. Thus $\boldsymbol{\omega}_o$ points in the direction that reflected light is *travelling to*, $\boldsymbol{\omega}_i$ points in the direction that incident light has *come from*, with $\mathbf{n} \cdot \boldsymbol{\omega}_o > 0$ and $\mathbf{n} \cdot \boldsymbol{\omega}_i > 0$. See Veach figure 3.1 (page 85).

1.3 Bidirectional Reflectance Distribution Function

The function $f(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i)$ in (1) is the Bidirectional Reflectance Distribution Function (BRDF) (Cohen & Wallace sections 2.5.1 and 2.5.4, Veach section 3.6.2) and describes how the surface reflects light, for example whether it is shiny or matte[4]. Energy conservation (Veach (3.14), Cohen & Wallace section 2.5.4) is an important property of any physically realistic BRDF. You should implement the Lambertian BRDF (e.g. Cohen & Wallace section 2.5.4) in your code.

1.4 Light Transport Equation

The total light outgoing in direction $\boldsymbol{\omega}_o$ from a point $\mathbf{x}$ on a surface, denoted $L_o(\mathbf{x}, \boldsymbol{\omega}_o)$, is the sum of the reflected light $L_r(\mathbf{x}, \boldsymbol{\omega}_o)$ given by (1), and light that is directly emitted by this surface $L_e(\mathbf{x}, \boldsymbol{\omega}_o)$ (which is specified as part of the scene),

$$L_o(\mathbf{x}, \boldsymbol{\omega}_o) = L_e(\mathbf{x}, \boldsymbol{\omega}_o) + \int_{\mathcal{H}^2} f(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) L_i(\mathbf{x}, \boldsymbol{\omega}_i)(\boldsymbol{\omega}_i \cdot \mathbf{n})\, d\boldsymbol{\omega}_i. \tag{2}$$

We can write L_i as a function of L_o (at a different point in space) by exploiting the fact that radiance is conserved along a ray of light[5]. Suppose we look from point $\mathbf{x}$ in the direction of the incoming light $\boldsymbol{\omega}_i$, and define $\mathbf{x}_{\mathcal{M}}(\mathbf{x}, \boldsymbol{\omega}_i)$ as the first point where this ray intersects a surface in the scene $\mathcal{M}$,

$$\mathbf{x}_{\mathcal{M}}(\mathbf{x}, \boldsymbol{\omega}_i) := \mathbf{x} + \mu\, \boldsymbol{\omega}_i, \qquad \text{with} \qquad \mu = \min_{\substack{\mu' > 0 \\ \mathbf{x} + \mu' \boldsymbol{\omega}_i \in \mathcal{M}}} \mu'. \tag{3}$$

Since radiance is conserved along this ray, the radiance incoming to point $\mathbf{x}$ from direction $\boldsymbol{\omega}_i$ is the same as the radiance outgoing from point $\mathbf{x}_{\mathcal{M}}(\mathbf{x}, \boldsymbol{\omega}_i)$ in direction $-\boldsymbol{\omega}_i$,

$$L_i(\mathbf{x}, \boldsymbol{\omega}_i) = L_o(\mathbf{x}_{\mathcal{M}}(\mathbf{x}, \boldsymbol{\omega}_i), -\boldsymbol{\omega}_i). \tag{4}$$

Substituting (4) into (2) gives the Light Transport Equation (LTE) with one unknown L_o,

$$L_o(\mathbf{x}, \boldsymbol{\omega}_o) = L_e(\mathbf{x}, \boldsymbol{\omega}_o) + \int_{\mathcal{H}^2} f(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i) L_o(\mathbf{x}_{\mathcal{M}}(\mathbf{x}, \boldsymbol{\omega}_i), -\boldsymbol{\omega}_i)(\boldsymbol{\omega}_i \cdot \mathbf{n})\, d\boldsymbol{\omega}_i. \tag{5}$$

(See e.g. Veach section 3.7.2, but also the introduction to chapter 3 and sections 3.5, 3.6, Cohen & Wallace equation 2.48). The LTE is linear in L_o, which gives it some nice mathematical properties (Veach chapter 4, particularly sections 4.3, 4.4).

1.5 The measurement equation

The colour of each pixel that our code generates is given by the measurement equation, Veach equation 3.18. We can simplify Veach's rather general integral expression by setting the responsivity W_e to be a Dirac delta function of both $\mathbf{x}$ and $\boldsymbol{\omega}$, which corresponds to a simple pinhole camera. The value of a pixel with coordinates (x, y) on the screen is then given by

$$I_{xy} = L_i(\mathbf{c}, \mathbf{d}_{xy}), \tag{6}$$

where $\mathbf{c} \in \mathbb{R}^3$ is the position of the camera and $\mathbf{d}_{xy} \in \mathcal{S}^2$ is the unit direction vector pointing in the direction that corresponds to pixel (x, y)[6].

[4] Like the radiance, the BRDF is in general function of wavelength λ, but is usually represented in terms of red, green and blue components $f = (f^r, f^g, f^b)$ (each, potentially different functions of $\mathbf{x}$, $\boldsymbol{\omega}_o$ and $\boldsymbol{\omega}_i$), which define the colour of the surface. The product fL_i in the integrand of (1) is then a componentwise product, $fL_i = (f^r L_i^r, f^g L_i^g, f^b L_i^b)$.

[5] Make sure you explain clearly why radiance is conserved (in a vacuum). See pbr-book.org chapter 15 for situations in which radiance is not conserved (mist, fog etc.).

[6] The 'perspective projection' provides a way of generating $\mathbf{d}_{xy}$. For example, for an $n \times n$ pixel image generated by a camera at $\mathbf{c}$, pointing at point $\mathbf{p}$, with $\mathbf{u}$ defining the upward direction, try $\mathbf{d} = (\mathbf{p} - \mathbf{c})\hat{}$, $\mathbf{v} = (\mathbf{d} \times \mathbf{u})\hat{}$, $\mathbf{w} = \mathbf{d} \times \mathbf{v}$, $\mathbf{d}_{xy} = (\mathbf{d} + \alpha_x \mathbf{v} + \alpha_y \mathbf{w})\hat{}$, where $\alpha_x = (2x/n - 1)\tan(\theta/2)$, $\alpha_x = (2y/n - 1)\tan(\theta/2)$, θ is the field of view angle, and $(\mathbf{x})\hat{} := \mathbf{x}/|\mathbf{x}|$. See e.g. pbr-book.org section 6.2.

2 Numerical solution of the LTE

Algebraic solutions of the LTE (5) are possible only in a few special cases, so usually we must solve it numerically. This requires converting the LTE (5) into an explicit form (with L_o only on the left hand side), and finding a numerical approximation to the integral over $\boldsymbol{\omega}_i$.

2.1 Monte-Carlo integration

The integral in (5) is usually approximated with Monte-Carlo integration[7]. This provides an estimate for an integral over some set Ω, by taking the mean of the integrand evaluated at randomly chosen points $\mathbf{x}_i$,weighted by p, the probability distribution of the random variable $\mathbf{x}_i$,

$$I = \int_\Omega g(\mathbf{x})\, d\mathbf{x} \approx \frac{1}{N} \sum_{i=1}^{N} \frac{g(\mathbf{x}_i)}{p(\mathbf{x}_i)}. \tag{7}$$

The variance of the estimate (7) determines the amount of noise in the image; this can be decreased by increasing the number of samples N, which increases the computational cost. Much current research in computer graphics is about finding smart ways of reducing the variance while keeping the number of samples N small, often by choosing p carefully (importance sampling). These topics are discussed in pbr-book.org chapter 13 (especially sections 13.1, 13.2, 13.7, 13.8, 13.10) and Veach chapter 2 (especially sections 2.4, 2.4.1, 2.4.3, 2.5.2).

We also evaluate the integral in the measurement equation (Veach equation 3.18) with Monte-Carlo, so that (6) is replaced by

$$I_{xy} \approx \frac{1}{N} \sum_{i=1}^{N} L_o(\mathbf{x}_\mathcal{M}(\mathbf{c}, \mathbf{d}_{xy}), -\mathbf{d}_{xy}). \tag{8}$$

The radiance L_o in (8) is evaluated using the LTE (5), again using Monte-Carlo integration, but this time with only $N = 1$ sample[8].

2.2 Evaluating the integrand of the LTE

Importantly, the integrand on the right hand side of the LTE (5) includes the unknown L_o. We can substitute (5) into itself, where each substitution corresponds to the light bouncing off a surface, The Monte-Carlo approximation of this when $N = 1$ is

$$L_o(\mathbf{x}_0, \boldsymbol{\omega}_0) \approx L_e(\mathbf{x}_0, \boldsymbol{\omega}_0) + \frac{f(\mathbf{x}_0, \boldsymbol{\omega}_0, \boldsymbol{\omega}_1)}{p_0(\boldsymbol{\omega}_1)} \left(L_e(\mathbf{x}_1, -\boldsymbol{\omega}_1) + \frac{f(\mathbf{x}_1, -\boldsymbol{\omega}_1, \boldsymbol{\omega}_2)}{p_1(\boldsymbol{\omega}_2)} \left(L_e(\mathbf{x}_2, -\boldsymbol{\omega}_2) + \cdots \right) \right), \tag{9}$$

where $\mathbf{x}_{i+1} = \mathbf{x}_\mathcal{M}(\mathbf{x}_i, \boldsymbol{\omega}_i)$, $\mathcal{H}_i^2$ is the hemisphere defined by the normal at $\mathbf{x}_i$, and $\boldsymbol{\omega}_i$ has probability distribution p_i over $\mathcal{H}_i^2$. This recursive process is called path tracing[9].

2.3 Terminating the path

To evaluate (9) on a computer, at some point the recursion must terminate, otherwise we will never finish evaluating the sum. This can be done by simply truncating after some number of bounces (perhaps 5 to 50, depending on the scene), but this leads to a *biased estimate* (see any statistics textbook) of the solution to the LTE (5). The Russian Roulette method (Veach section 2.7.2, pbr-book.org section 13.7, Arvo & Kirk 1990) gives a method of evaluating this infinite sum in an unbiased way.

[7] Kajiya (1986) and Cohen & Wallace describe some other approximation methods, now mostly superceded

[8] see 'Infinite-dimensional solution' in section 4 of Kajiya (1986), and Arvo & Kirk (1990) for alternatives to $N = 1$ sample in the LTE.

[9] See e.g. Kajiya (1986) or pbr-book.org section 14.5, although pbr-book.org and other recent sources often use 'path tracing' only when the integral is rewritten an integral over paths, not solid angles. See also the related ideas of Recursive Ray Tracing and Distributed Ray Tracing (Cook 1984). The process substituting the LTE into itself is called 'reverse ray tracing', because the light is traced from the camera to light sources. Light can instead be traced from light source to camera ('forward ray tracing') or elements of these two methods combined in bidirectional path tracing (pbr-book.org section 16.3, Veach 3.7.3, 3.7.4).

2.4 Choosing and sampling p

The simplest choice p for Monte-Carlo estimation of the integral in the LTE (5) is uniform sampling[10], $p(\boldsymbol{\omega}_i) = 1/(2\pi)$ (this satisfies $\int_{\mathcal{H}^2} p(\boldsymbol{\omega}_i)\, d\boldsymbol{\omega}_i = 1$ as required). To evaluate the Monte-Carlo integral we also need an algorithm to generate points on the hemisphere $\mathcal{H}^2$ with this uniform distribution. See prb-book.org section 13.6.1[11].

2.5 Evaluating $\mathbf{x}_{\mathcal{M}}$

We evaluate $\mathbf{x}_{\mathcal{M}}(\mathbf{x}, \boldsymbol{\omega})$ by calculating the intersection points (if any) of the ray with starting point $\mathbf{x}$ and direction $\boldsymbol{\omega}$ with each object in the scene in turn, and choosing the intersection point closest to $\mathbf{x}$.

A simple starting point is to construct our scene $\mathcal{M}$ out of a union of spheres. The intersection of the ray and sphere i in the scene with centre $\mathbf{C}_i$ and radius R_i occurs when

$$|(\mathbf{x} + \mu\boldsymbol{\omega}) - \mathbf{C}_i| = R_i, \tag{10}$$

which simplifies to a quadratic for μ that can be solved straightforwardly to find the real positive roots (if any). In accordance with (3), the value of $\mathbf{x}_{\mathcal{M}}(\mathbf{x}, \boldsymbol{\omega})$ corresponds to the smallest positive root μ across all spheres i in the scene.

2.6 Validating code

For some simple scenes there are exact algebraic solutions to the light transport equation – see pbr-book.org section 14.4.2. It is very important to validate your code by comparing its numerical results against this or other algebraic solutions to the LTE. Some things to demonstrate include:

- how your Monte-Carlo estimate of the LTE solution using uniform sampling converges to the correct algebraic solution as the number of samples (and the number of bounces, if not using Russian Roulette) gets very large,
- how the variance in your estimate decreases with N at the rate expected from Monte-Carlo methods,
- how truncating after a finite number of bounces leads to a biased estimate, whereas Russian Roulette leads to an unbiased estimate,
- the effect of using cosine-weighted sampling (if implemented) on the pbr-book.org section 14.4.2 example.

You may well need to test your implementations of the $\mathbf{x}_{\mathcal{M}}$ function, your random generation of $\boldsymbol{\omega}_i$, etc. as part of the debugging process. Test also that all four cases when the camera and/or light source are inside or outside of non-emitting sphere are rendered correctly.

2.7 Example scenes

Once you have validated your numerical solution of the LTE, you can generate some test images, e.g. showing the effects of changing N, the number of bounces (if not using Russian Roulette), scenes which are particularly rapid or slow to convege with N, and so on. You could recreate the famous Cornell Box test image, which can be approximated with five (large) spheres[12]. There is plenty of scope for creativity here, but make sure your test images serve a purpose in illustrating an argument made in the text.

[10] Another distribution to try is cosine-weighted sampling, $p(\boldsymbol{\omega}_i) = (\boldsymbol{\omega}_i \cdot \mathbf{n})/\pi$ (pbr-book.org section 13.6.3). In the Monte-Carlo estimate, the dot product in p cancels with that in the LTE – a simple form of importance sampling.

[11] or alternatively choose a point $\mathbf{x}$ uniformly on the surface of a sphere (e.g. using an algorithm from `http://mathworld.wolfram.com/SpherePointPicking.html`) and flip the sign of $\mathbf{x}$ if $\mathbf{x} \cdot \mathbf{n} < 0$.

[12] Generally this looks realistic when the BRDF values f^r, f^g, f^b are between 0.05 and 0.95 (times $1/\pi$), as real surfaces are not perfect reflectors or absorbers.

3 Code

Ray-tracing is computationally expensive – typical test images will take several minutes to render in a fast compiled language (e.g. C++, C or Fortran). Completing this project in MATLAB and Python is possible, but codes are likely to be several times slower than in C++. In the `code/` directory I provide two helper C++ classes for this project: `Vec3` representing three-vector ($\in \mathbb{R}^3$), and `Image` representing an can be created and saved as a `.png` file. Example code for both of these classes is provided in `header_tests.cpp`.

3.1 Code structure

A program implementing the algorithms is likely to need the following parts:

- Code to loop over each pixel in the image and evaluate its value using (8).
- A function to evaluate L_o using path tracing (9). A recursive function is a natural way of doing this, although a loop can be used instead.
- Function(s) to generate random vectors $\boldsymbol{\omega} \in \mathcal{H}^2$ according to some distribution p, as in §2.4.
- A function to calculate $\mathbf{x}_{\mathcal{M}}(\mathbf{x}, \boldsymbol{\omega})$, as per the algorithm in §2.5. You may find it useful to return from this function a variable indicating which object in the scene has been hit, as well as the intersection position $\mathbf{x}$; this makes it easier to then look up the required normal vector, BRDF value etc. evaluated at $\mathbf{x}$. Make sure that you exclude the solution $\mathbf{x}_{\mathcal{M}}(\mathbf{x}, \boldsymbol{\omega}) = \mathbf{x}$ if $\mathbf{x} \in \mathcal{M}$.
- A way of storing the scene $\mathcal{M}$, such as an array (/vector/list) of spheres, each with a centre $\in \mathbb{R}^3$, radius, emitted light intensity L_e and BRDF f.

4 Extensions and topics for two-semester projects

Following the construction of a basic ray-tracer, there are many ways in which it can be extended. Some examples are given below; stars give a rough indication of difficulty, from *=easier to ***=harder. Most of the following topics are covered in pbr-book.org and references therein (look in the 'Further Reading' sections at the end of each chapter), and on the web.

- Reduction of variance (Veach sections 2.5 to 2.8)
  - Direct lighting: sampling over light sources and multiple importance sampling (**). (**Recommended for a two-semester project**, Veach chapter 9, pbr-book.org sections 14.2, 14.3)
  - Quasi-Monte-Carlo and stratified sampling (**) (Veach section 2.6, pbr-book.org chapter 7). This is particularly effective when applied to direct lighting and/or depth-of-field sampling.
- More flexible description of the scene $\mathcal{M}$:
  - More general shapes (cylinders, cuboids(*), general isosurfaces(***), pbr-book.org chapter 3)
  - Spatially-dependent BRDF f and emitted radiance L_e functions (**), bump mapping (**)
  - Non-Lambertian BRDFs: perfect mirrors(*), transparent objects with refraction(**), specular and microfacet BRDFs(***). (Cohen & Wallace section 2.5, pbr-book.org chapter 8)
  - Transforms of objects (rotation, scaling etc.) (**) (pbr-book.org section 2.7)
  - Acceleration structures for object intersection (***) (pbr-book.org chapter 4)
- Sampling the measurement equation
  - Anti-aliasing (*) (pbr-book.org section 7.8)
  - Depth-of-field and/or motion blur (**) (pbr-book.org section 6.2.3)