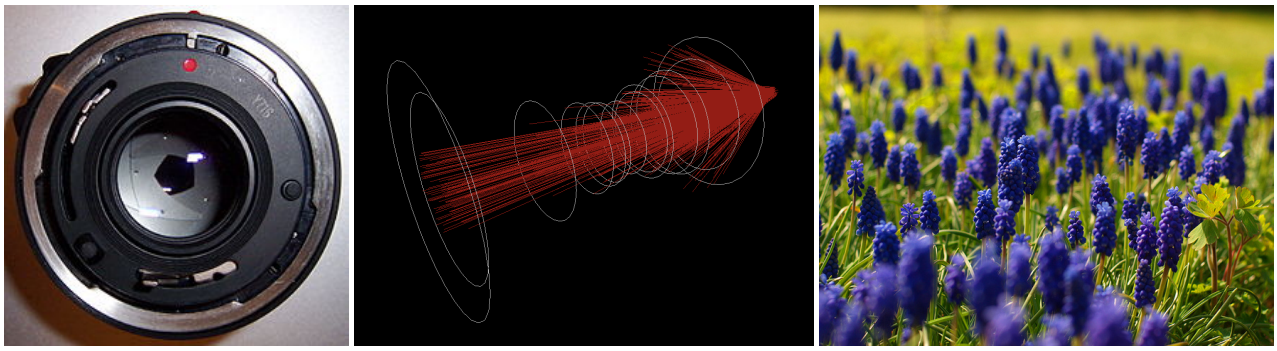
CS148 Assignment 2

Realistic Camera Simulation using PBRT



Goals: *Learn the basic optics required to simulate realistic lenses as part of a modification to the PBRT raytracer; visualize ray paths through camera elements using* `vdb` *; generate images using simulated lenses.*

1 Background

Many rendering systems mimic a pinhole camera, generating images in which the entire scene is in sharp focus. Real cameras, on the other hand, contain multi-lens assemblies with finite apertures and exhibit a variety of imaging characteristics such as limited depth of field, field distortion, vignetting and spatially varying exposure. Simulating or approximating these effects is critical to the production of synthetic images with realistic appeal.

In this assignment, you'll help fill in the important methods of an extension to `pbrt`, providing a more physically plausible camera model that accurately recreates the look and feel of real camera lenses. You will be provided with most of the code necessary, as well as specifications for realistic wide-angle, double gauss, fisheye, and telephoto lenses. The result will be a camera plug-in that simulates the traversal of light through lens assemblies and onto the film plane of a virtual camera. With this, you will be able to render virtual scenes with a variety of lenses.

Before getting started, you should review *A Realistic Camera Model for Computer Graphics* By Kolb, Mitchell, and Hanrahan (part of the course readings.) You may also find it helpful to read Chapter 6 (Camera Models) of *Physically Based Rendering: from Theory to Implementation*, which is available in electronic format from the Stanford Library[1].

2 Getting Started

Installing PBRT

You will begin by downloading and installing `OpenEXR` and `pbrt`. Before getting started, please carefully read the instructions for doing so on the project's github page:

`https://github.com/mmp/pbrt-v2`

If you run into problems, the staff for CS348B has written an excellent compilation of instructions for installing PBRT here:

`http://candela.stanford.edu/cs348b-14/doku.php?id=build_instructions`

Once you have done this, run a few of the given examples. For instance, from the root `pbrt-v2/` directory, you might run:

```
./src/bin/pbrt scenes/killeroo-simple.pbrt
./src/bin/exrtotiff killeroo-simple.exr killeroo.tiff
```

The first line generates the high dynamic range `.exr` image file, and the second line uses a built-in utility from `pbrt` to convert the `.exr` to a `.tiff` image.[2]

[1] http://searchworks.stanford.edu/view/8669621

[2] Note: you may not find this conversion necessary; lots of modern image viewers, such as OSX's Preview app, can display `.exr` images directly.

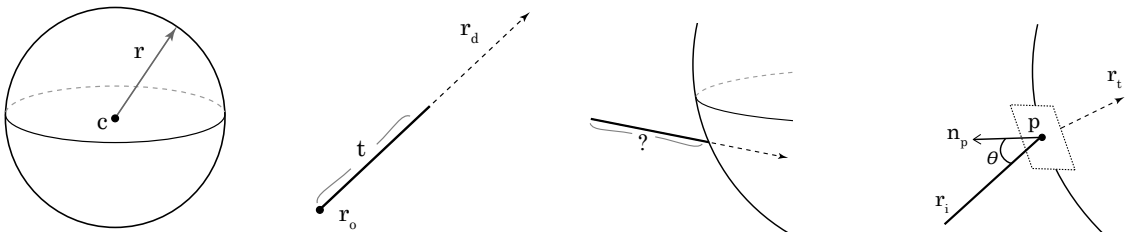


Figure 1: *(a) A point* $\mathbf{x}$ *lies on the surface of a sphere with center* $\mathbf{c}$ *and radius* r *iff* $(\mathbf{x}_x - \mathbf{c}_x)^2 + (\mathbf{x}_y - \mathbf{c}_y)^2 + (\mathbf{x}_z - \mathbf{c}_z)^2 = r^2$*. (b) A ray is parameterized as* $r(t) = r_o + t\, r_d$*. (c) When ray and sphere intersect, this point is identified by the value of* t *at which the intersection occurs. (d) The angle of incidence w.r.t. the surface normal at intersection point* $\mathbf{p}$ *determines the trajectory of the transmitted ray* $\mathbf{r}_t$*.*

Starter code

Next, you will need to download the starter code from the course webpage[3]. You will notice that the subdirectories within the `hw2_starter` directory correspond to those inside the main `pbrt` directory. For each, drag and drop the files from within the starter code package into their corresponding directory within your local installation of `pbrt`. Recompile and make sure that the example `.pbrt` scenes still render correctly.

3 Derivations

In a real camera, photons pass from the scene, through each lens element, and onto a film plane. In ray-tracing, this process is reversed as simulated *rays* are transmitted through each lens element. During this process, they refract as predicted by Snell's law, and are then cast into the scene. There, the rays interact with scene geometry, contributing to the calculation of that pixel's color.

In this section, we will review how to find the intersections between camera lenses and rays in order to complete the implementation provided in the starter code.

Ray/Sphere interactions

The lenses you will be simulating are comprised of elements with spherical curvature. Recall that a point $\mathbf{x}$ lies on the surface of a sphere iff

$$(\mathbf{x}_x - \mathbf{c}_x)^2 + (\mathbf{x}_y - \mathbf{c}_y)^2 + (\mathbf{x}_z - \mathbf{c}_z)^2 = r^2 \tag{1}$$

$$\Leftrightarrow (\mathbf{x} - \mathbf{c}) \cdot (\mathbf{x} - \mathbf{c}) = r^2, \tag{2}$$

where the sphere is centered at a point $\mathbf{c}$ and has radius r.

> **Question 1**: Given a sphere $S_{\mathbf{c},r}$ with radius r centered at point $\mathbf{c}$, write down an expression for the unit normal at a point $\mathbf{p}$ lying on the surface (see Fig. 1d).

A ray is a semi-infinite line often abstracted as follows:

$$\mathbf{r}(t) = \mathbf{r}_o + t\, \mathbf{r}_d \qquad 0 \leq t \leq \infty. \tag{3}$$

Here, $\mathbf{r}_o$ is the point of origin, $\mathbf{r}_d$ is the vector defining the direction of the ray, and t is the parameter that, when varied, specifies the points in space through which the ray passes (see Fig. 1b-c). When a ray intersects a lens element, we identify the intersection point with the parameter t specifying that point along the ray.

> **Question 2**: Write down an equation for the intersection of a ray $r(t)$ and sphere $S_{\mathbf{c},r}$.

> **Question 3**: Your answer for the previous question should be quadratic in t, and therefore has two roots. Explain the physical significance when the these roots are (a) both imaginary; (b) one real and one imaginary; (c) both real.

> **Question 4**: In the next section, you will use this equation to calculate the intersection point between a ray and a spherical lens element. Which root should be returned as the intersection point between the ray and the lens? Why?

The following code is located in `core/pbrt.h`, and may be helpful to you when computing ray/sphere intersections. It returns the real roots of a quadratic equation $ax^2 + bx + c = 0$:

[3] https://www.stanford.edu/class/cs148/assets/hws/hw2_starter.zip

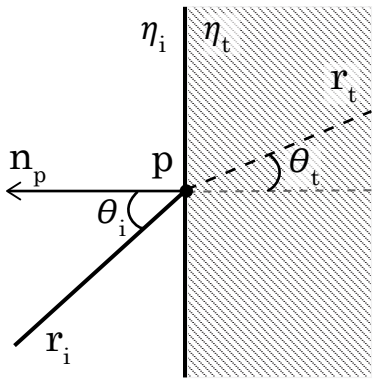
```
inline bool Quadratic(float A, float B, float C, float *t0, float *t1){
    // Find quadratic discriminant
    float discrim = B * B - 4.f * A * C;
    if (discrim < 0.) return false;
    float rootDiscrim = sqrtf(discrim);

    // Compute quadratic _t_ values
    float q;
    if (B < 0) q = -.5f * (B - rootDiscrim);
    else       q = -.5f * (B + rootDiscrim);

    *t0 = q / A;
    *t1 = C / q;
    if (*t0 > *t1) swap(*t0, *t1);
    return true;
```

Question 5: The roots of a quadratic equation are generally given as $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$. Why is this formulation problematic, and why is the alternate method above used instead?

Snell's Law and Refraction

Snell's law expresses the change in angle that a beam of light experiences when being transmitted through an interface into a material with a different index of refraction, η (Fig. 2a):

$$\frac{\eta_i}{\eta_t} = \frac{sin(\theta_t)}{sin(\theta_i)} \tag{4}$$



(a)

(b)

Figure 2: *(a) Snell's law relates incident and transmitted directions $\{\theta_i, \theta_t\}$ with refractive index $\{\eta_i, \eta_t\}$. (b) These fishes have something to tell you about angles of incidence.*

Question 6: When using Snell's law to calculate refracted rays, you will have to check that $sin(\theta_t)$ does not exceed 1. What would this indicate? How does the image in Fig 2b illustrate this concept?

Question 7: A normalized ray $\mathbf{r}_i$ intersects a surface at point $\mathbf{p}$ with an angle θ_i to the unit normal $\mathbf{n_p}$, as shown in Figure 2. Use Snell's law to derive an expression for the (normalized) transmitted ray, $\mathbf{r}_t$, in terms of η_i, η_t, θ_i, $\mathbf{r}_i$, and $\mathbf{n_p}$. Show your work.[a] *Hint: Separate $\mathbf{r}_t$ into two components: one parallel to the interface, and another along the normal.*

[a] You may find it easier to write this out by hand. Feel free turn in a scanned or carefully photographed image of your derivation.

4 Implementing Ray/Lens Interactions

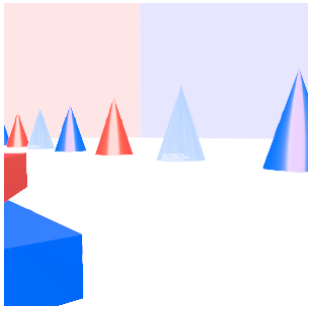
In the starter code, you will find a directory called `cameras/`. There, you will find files `realistic.h` and `realistic.cpp` containing most of the code necessary to implement your realistic camera lens. You will also find simple example scenes (`*.pbrt`) for testing out your lens code, as well as several different lens specification files (`*.dat`).

Using your answers from Part 3, complete the following functions found in `realistic.cpp`:

```
static bool IntersectSphericalElement(float radius, float center,
                                      const Ray &ray, float *t, Normal *n);

static bool Refract(const Normal &n, const Vector &wi, float eta_i,
                    float eta_t, Vector *wt);

bool RealisticCamera::TraceThroughLenses(Ray *ray,
                                         float filmDistance);
```

Instructions and hints can be found in the source code. We suggest taking these steps:

- Take a tour of `realistic.h` and `realistic.cpp`. Start with the function `RealisticCamera *CreateRealisticCamera` which calls the realistic camera constructor `RealisticCamera(...)`. Using your knowledge from the reading, you should be able to follow the comments and understand what each function does.
- Recompile and run your code with a perspective camera, checking that `pbrt` is working as expected. For instance, you might run:
  ```
  ./src/bin/pbrt perspective.pbrt
  ./src/bin/exrtotiff perspective.exr perspective.tiff
  ```
- Use `vdb` to visualize the lens elements and ray trajectories as you complete the missing methods. You may choose to do something similar to the visualization shown in Figure 4a. Keep in mind that the number of rays going through the camera is vast – trying to visualize all of them at once will be extremely inefficient!
- Use your lens simulation to recreate the images in Figure 3. These are generated by ray-tracing `{fisheye, telephoto, wide, dgauss}.pbrt`. Low-fidelity versions can be made by changing the number of samples requested per pixel; see the comments in the corresponding `*.pbrt` files for details.



perspective.tiff

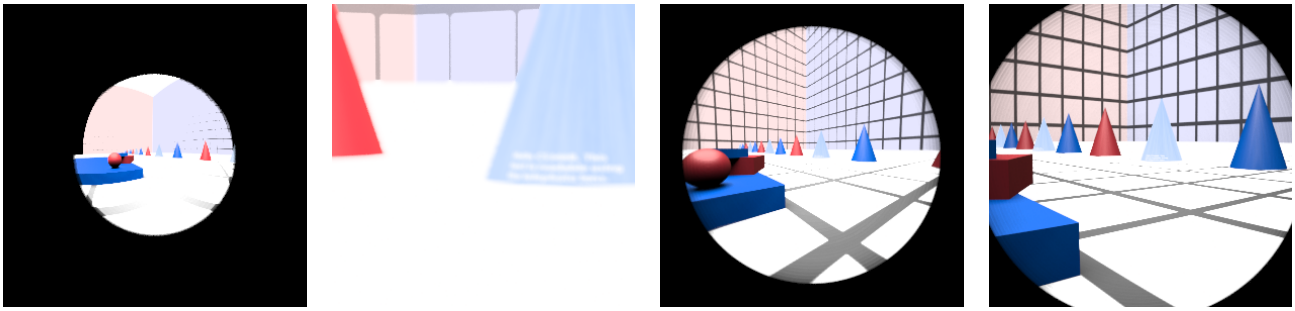


Figure 3: *Target images for each lens on the simple scene. Left to right: fisheye, telephoto, wide-angle, double gauss.*

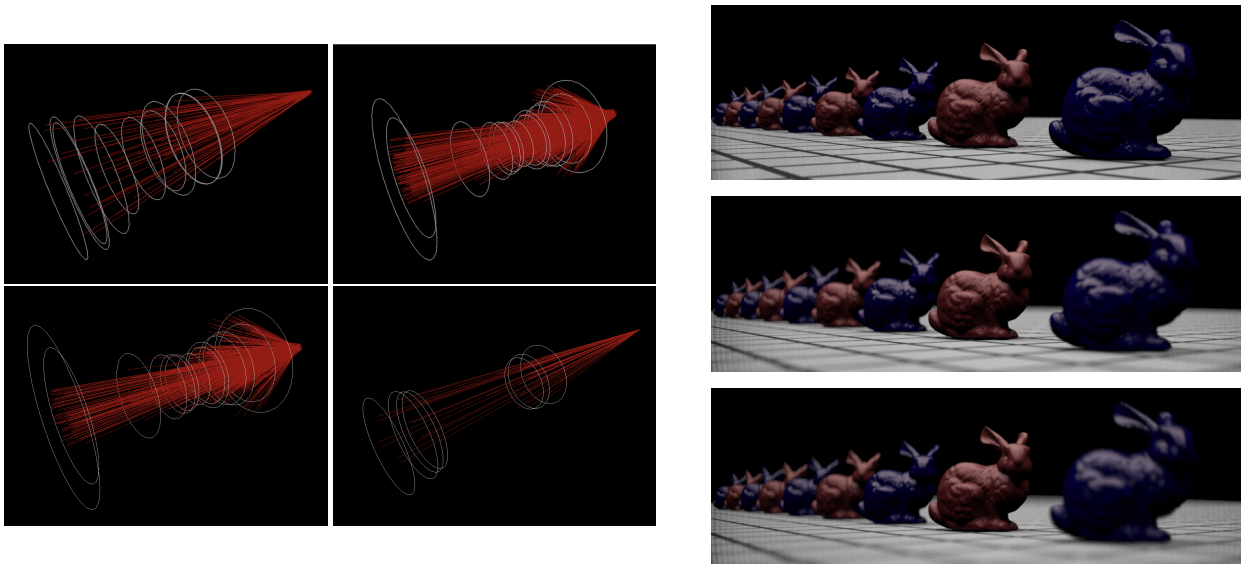
5 Image Analysis

Question 8: Figure 4a) shows each of the 4 realistic lenses provided in the starter code. Identify each.

Question 9: Figure 4b) shows images rendered using the following camera specification:

```
Camera "realistic"
"string specfile" "dgauss.50mm.dat"
"float focusdistance" 0.75
"float filmdistance" 36.77
"float aperture_diameter" 30
"float filmdiag" 70
"float shutteropen" 0
"float shutterclose" 0
```

Only one value was changed for each of the three images. Which one, and how did it vary?



(a) (b)

Figure 4: *(a) A sample visualization of ray-lens interaction. (b) Your Stanford bunnies, hard at work for science.*

6 What to turn in

To submit your assignment, please bundle the following into a `.zip` file:

1. A PDF or .rtf file containing:
   - Your responses for Questions 1 - 9. If you prefer to write your derivations by hand, please scan or neatly photograph your work.
   - `vdb` screenshots for each lens, showing ray paths through the lens elements.[4]
   - Any "accidental art"– weird images, mistakes, &c.– demonstrating challenges you faced. Include a brief explanation of what went wrong and how you fixed it.
2. Your completed version of `realistic.cpp`. Please leave your name in a comment at the top of the file.
3. Your high-fidelity renderings of the test scenes: `{wide, fisheye, dgauss, telephoto}.pbrt`.
4. *Bonus credit:* Any additional `pbrt` scene rendered using one of the realistic lenses. Please include your image, all files needed to reproduce it, and a sentence or two describing the scene. Many example scenes can be found at `http://www.pbrt.org/scenes.php`.
5. A text file README containing anything else we should know: names of classmates you worked with, help received, references, etc.

Your `.zip` file can be submitted using the form on the students' section of the course webpage. Happy lensing!

[4]Note: if you had trouble getting `vdb` operational in HW1, you are not required to use it here. Just leave your grader a short explanation.