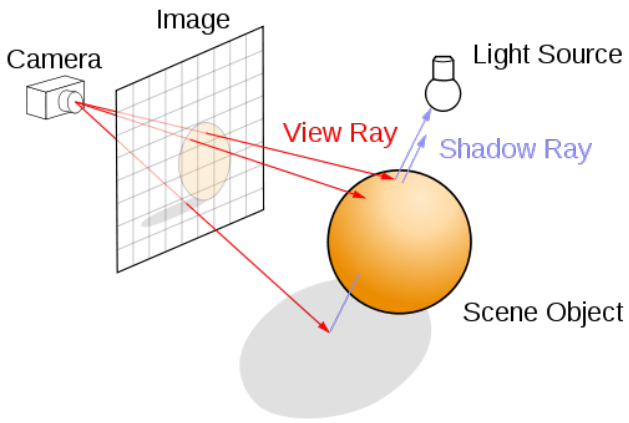

Figure 1: Diagram illustrating ray casting [1].

Project 2
Due on Wednesday, Mar 16

Introduction

Ray-casting is a basic rendering technique. Suppose a viewer at coordinates $(0, 0, -w)$ is looking through the plane $z = 0$ at some surface. We pretend that the screen lives on the plane $z = 0$, and the pixel at (x, y) is colored based on where the ray from the viewer through $(x, y, 0)$ first intersects some surface.

In this project, we will use ray casting to render an *implicit* surface defined by an equation $f(x, y, z) = 0$. This means that for each pixel position (x, y) in the image, we must find the smallest $z > 0$ that satisfies the equation

$$g(z; x, y) = f(x(1 + z/w), y(1 + z/w), z) = 0.$$

If $g(z; x, y)$ is zero, then we color the screen point (x, y) according to the illumination of the point $(x(1+z/w), y(1+z/w), z)$ on the zero set of f. We provide code to compute the shading of each pixel based on a simple Phong shading model with a single light source. Your goal is to efficiently solve the implicit equation $g(z; x, y) = 0$ for every pixel position (x, y) in an image.

Defining implicit surfaces

The chief input is a handle for a MATLAB function with input arguments `x`, `y`, `z` that are parallel coordinate arrays and output arguments `f`, `f_x`, `f_y`, `f_z` that are the corresponding function values and components of the gradient. For example, we use the following MATLAB function to define a sphere of radius one centered at $(0, 0, 2)$:

```
function [f, f_x, f_y, f_z] = spherel(x,y,z)

  f = x.^2+y.^2+(z-2).^2 - 1;
  if nargout > 1
      f_x = 2*x;
      f_y = 2*y;
      f_z = 2*(z-2);
  end
```

Basic task

You will be given a template for a function `raycast` that is only missing the definition of a helper function `ray_intersections`. Your goal is to provide a reasonably efficient, vectorized implementation of `ray_intersections`. I recommend the following strategy:

1. Evaluate f for all points along several planes between $z = 0$ and $z = z_{\max}$ in order to estimate where each ray first intersects the surface (if indeed it intersects the surface at all).

2. Using the information obtained by sampling f, take a few steps of Newton iteration (in parallel for all points in the image).

3. Use bisection or Brent's method to locate those points where Newton did not provide sufficient accuracy.

You may use a different approach, but you should make sure that you use a superlinearly convergent algorithm most of the time. You may not use the built-in MATLAB function `fzero` in your final version of the code (though it may be useful for intermediate versions).

Submission

You should submit the completed file `raycast.m`, as well as a write-up that includes at least the following:

- Basic tests to demonstrate that your routine to compute intersections between the ray and surface converges superlinearly. You may use the intersection with the sphere as a model problem.
- One test image of your choosing (other than the sphere).
- A discussion of which parts of the code take the most time according to MATLAB's profiler. Use `help profile` to learn more.
- A description of shortcomings of your root finding code. Though we do not expect your code to be able to render arbitrary surfaces (though it should handle "nice" smooth surfaces), we do expect you to understand what types of surfaces that might cause problems. Think of an audience who might use this code. Are there artifacts they might see because of the root finding strategy? Is there anything they can do to the problem formulation to fix them?

  Note: Unless you feel a compelling need to do so from reasons of your own, you do not need to say anything about the lack of shadows and other features due to non-recursiveness in the raycasting, nor do you need to comment on the quality of the shading model.

The write-up is an important part of this project. Please spend the time necessary to do a good job on it!

References

[1] Wikipedia: Ray casting. Work found at http://en.wikipedia.org/wiki/File:Ray_trace_diagram.svg / CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0/)