Extracted from:

The Ray Tracer Challenge

A Test-Driven Guide to Your First 3D Renderer

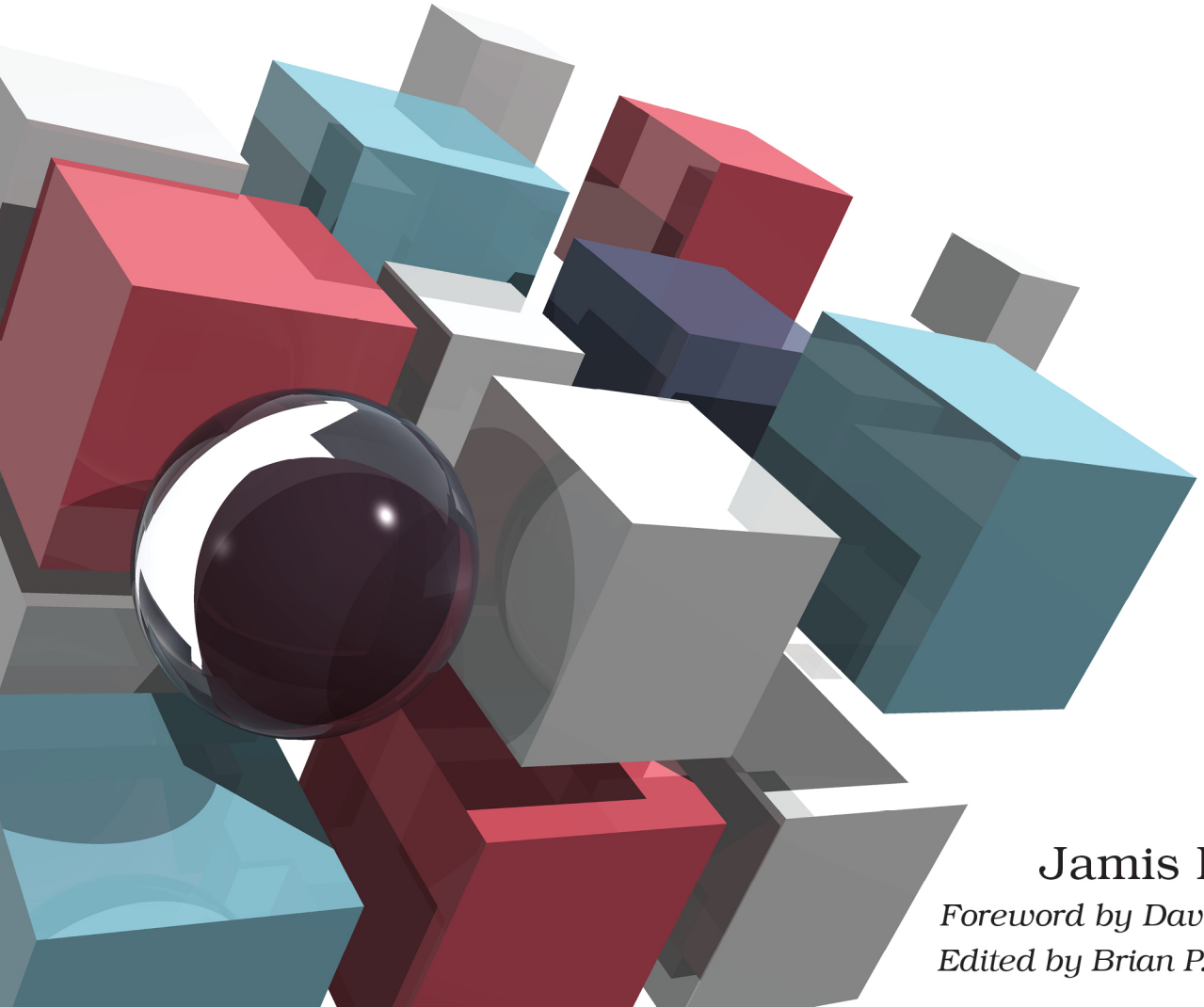
The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

The Ray Tracer Challenge

A Test-Driven Guide to Your First 3D Renderer



Jamis Buck

Foreword by David Buck

Edited by Brian P. Hogan

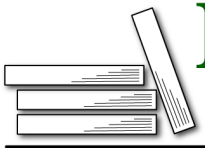
The Ray Tracer Challenge

A Test-Driven Guide to Your First 3D Renderer

Jamis Buck

The Pragmatic Bookshelf

Raleigh, North Carolina



Pragmatic Bookshelf

Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Managing Editor: Susan Conant
Development Editor: Brian P. Hogan
Copy Editor: L. Sakhi MacMillan
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics

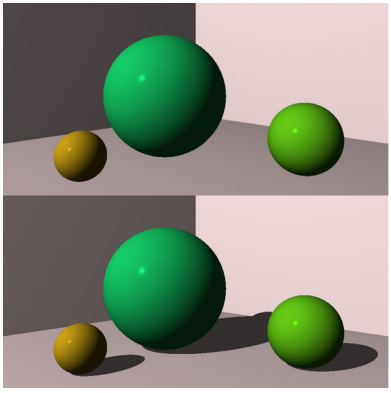
For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-271-8
Book version: P1.0—February 2019

Your ray tracer is really starting to come together. Just look at it! You've got spheres, realistic shading, a powerful camera, and a world that supports scenes with many objects.

It's a pity those objects don't cast shadows, though. Shadows add a delightful dash of realism to a scene. Check out the following figure which shows the same scene both with and without shadows:



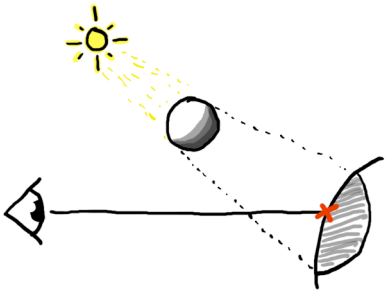
Your brain uses those shadows as cues for depth perception. Without shadows, the image looks artificial and shallow, and that will never do.

Thus, the time has come to add shadows, and the best part is that you've already written most of the infrastructure to support this. The first step is to adjust your `lighting()` function to handle the case where a point is in shadow. Then you'll implement a new method for determining whether a point is in shadow or not, and last you'll tie those pieces together so your ray tracer actually renders the shadows.

Let's dig into it!

Lighting in Shadows

Given some point, you can know that it lies in shadow if there is another object sitting between it and the light source, as shown in the following figure.



The light source is unable to contribute anything to that point. Take a moment and recall how your lighting() function works, from *The Phong Reflection Model*, on page ?. The diffuse component relies on the vector to the light source, and the specular component depends on the reflection vector. Since both components have a dependency on the light source, the lighting() function should ignore them when the point is in shadow and use only the ambient component.

Add the following test to the others you wrote for the lighting() function. It's identical to the one titled "Lighting with the eye between the light and the surface" on page ?, where the specular and diffuse components were both at their maximum values, but this time you're going to pass a new argument to the lighting() function indicating that the point is in shadow. It should cause the diffuse and specular components to be ignored, resulting in the ambient value alone contributing to the lighting.

(Recall that the m and position variables being passed to the lighting() function are defined in the "Background" block on page ?.)

features/materials.feature

```
Scenario: Lighting with the surface in shadow
  Given eyev ← vector(0, 0, -1)
    And normalv ← vector(0, 0, -1)
    And light ← point_light(point(0, 0, -10), color(1, 1, 1))
    And in_shadow ← true
  When result ← lighting(m, light, position, eyev, normalv, in_shadow)
  Then result = color(0.1, 0.1, 0.1)
```

You may need to fix your other tests to accommodate the addition of that new parameter. Go ahead and address that, and then make this new test pass as well by making your lighting() function ignore the specular and diffuse components when in_shadow is true.

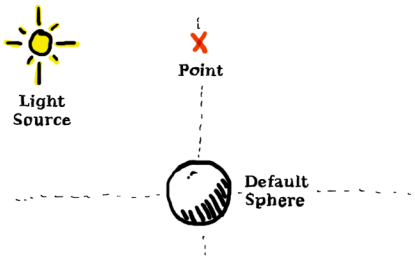
Once things are all passing again, let's teach your ray tracer how to tell when a point is in shadow.

Testing for Shadows

A ray tracer computes shadows by casting a ray, called a *shadow ray*, from each point of intersection toward the light source. If something intersects that shadow ray between the point and the light source, then the point is considered to be in shadow. You're going to write a new function, is_shadowed(world, point), which will do just this.

Implement the following four tests, which demonstrate four different scenarios. Each assumes the existence of the default world that was defined in *Building a World*, on page ?.

In the first test, the world is set up like the following figure.

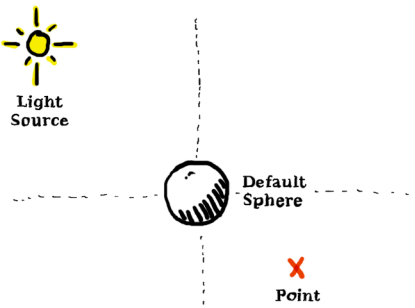


Nothing at all lies along the line connecting the point and the light source, and the point should therefore not be in shadow.

features/world.feature

```
Scenario: There is no shadow when nothing is collinear with point and light
  Given w ← default_world()
    And p ← point(0, 10, 0)
   Then is_shadowed(w, p) is false
```

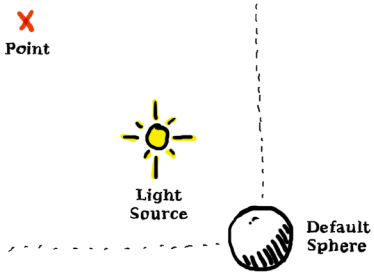
In the second test, the point is placed on the far side of the default world's spheres, putting them between it and the light source, like this:



The point should be in the shadow cast by the spheres.

features/world.feature
```
Scenario: The shadow when an object is between the point and the light
  Given w ← default_world()
    And p ← point(10, -10, 10)
   Then is_shadowed(w, p) is true
```

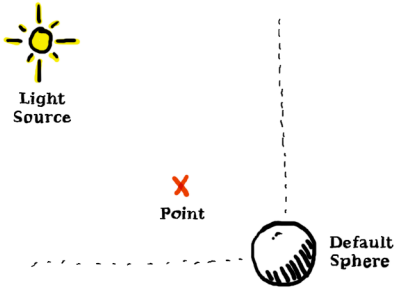
The next test positions the point so the light lies between it and the spheres.



Once again, the point should not be in shadow, because nothing lies between the point and the light.

features/world.feature
```
Scenario: There is no shadow when an object is behind the light
  Given w ← default_world()
    And p ← point(-20, 20, -20)
   Then is_shadowed(w, p) is false
```

The last test is similar, but it positions the point to lie between the light and the spheres, like this:



And again, even in this configuration nothing lies between the light and the point, so the point is still not shadowed.

features/world.feature
```
Scenario: There is no shadow when an object is behind the point
  Given w ← default_world()
    And p ← point(-2, 2, -2)
   Then is_shadowed(w, p) is false
```

The algorithm for is_shadowed() goes like this:

1. Measure the distance from point to the light source by subtracting point from the light position, and taking the magnitude of the resulting vector. Call this distance.
2. Create a ray from point toward the light source by normalizing the vector from step 1.
3. Intersect the world with that ray.
4. Check to see if there was a hit, and if so, whether t is less than distance. If so, the hit lies between the point and the light source, and the point is in shadow.

In pseudocode it might look like this:

```
function is_shadowed(world, point)
  v ← world.light.position - point
  distance ← magnitude(v)
  direction ← normalize(v)

  r ← ray(point, direction)
  intersections ← intersect_world(world, r)

  h ← hit(intersections)
  if h is present and h.t < distance
    return true
  else
    return false
  end if
end function
```

Recall from *Identifying Hits*, on page ?, that the hit() function returns the intersection with the lowest *nonnegative* t value. Thus, the hit's t will never be negative, so you don't need to worry about checking for intersections that occur behind the point.

Implement that function, make those tests pass, and then move on. Just one more thing needs changing to actually render those shadows!