Algorithm 7 Ray

Introduction to Computer Graphics, Fall 2019

Solution Key

Instructions: Complete this assignment by yourself without any help from anyone or anything except a current CS123 TA, the lecture notes, official textbook, and the professor. Hand in the assignment using `cs1230_handin ray_algo` no later than 11:59pm on the due date. Late hand-ins are not accepted under any circumstances.

This assignment is worth **10%** of your Ray grade.

1. The high-level view of our ray tracer is exactly the same as for intersect, except for a few additions. Below is the high-level pseudocode for intersect.

```
Algorithm 1 RAY-TRACE(Scene, Canvas)
for $point \in Canvas$ do
  cast a ray to find the nearest object
  if ray intersects an object then
    for each light do
      cast a ray to the light and evaluate the lighting equation
      $Canvas[pt] = Canvas[pt] +$ ambient color + diffuse color.
    end for
  else
    $Canvas[pt]$ = background color
  end if
end for
```

[1 point] 1.1: What needs to be changed / added to make this a full implementation of the Phong lighting model?

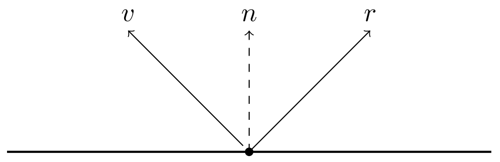
[1 point] 1.2: What needs to be changed / added to make this a full-fledged raytracer?

(Note: Just specify what changes need to be made – no pseudocode please.)

Solution: (a) 1 point: Line 6 should be changed to add specular contributions.

(b) 1 point: Line 6 should be changed to add reflection. Shadows should also be mentioned somewhere. Textures may be mentioned, but are not required (we are discussing only the modeling of light).

2. **[2 points]** Given vector $\vec{v}$ from the surface to a light and the surface normal $\vec{n}$, find the equation for the vector $\vec{r}$ which is the reflection of an **incoming** light ray about $\vec{n}$. Write your equation in terms of vector operations. How do you compute the color contributed by the reflected ray? Give a brief description.



Solution:

$$\vec{r} = 2\vec{n}\,(\vec{n}\cdot\vec{v}) - \vec{v}$$

To compute the color contributed by the reflected ray, trace the reflected ray (hint: use recursion). For the point with the smallest non-negative t value, compute the color using the lighting model. The contribution is that color multiplied by the global specular coefficient and the object's reflection color.

3. **[1 point]** In words, how does attenuation affect the light shown in a pixel?

Solution: When light hits an object, it bounces in a new direction. Since our scenes model the real world, light is traveling through air, which will similarly interfere with some light rays before they reach their destination. The longer the distance traveled, the fewer light rays you'll see. This means that faraway lights will have less of an effect on the illumination of an object than close lights. This effect is also a product of the inverse square law.

4. **[1 point]** When would an object (or portions of an object) not be affected by a light source? Provide at least 2 examples. You can list more for extra credit.

Solution: (a) when there is an object intersecting the vector from the point of intersection on this object to the light source in question

(b) when the surface is not facing the light (the surface normal and the light vector form an angle of at least 90 degrees)

(c) when the surface does not reflect the color of the light (i.e. if the light is purely blue and the surface is purely green)

5. **[2 points]** Recall that we can think of texture mapping in two steps. First, mapping from the object to the unit square, and second, mapping from the unit square to the texture map. Let u and v be the x and y values in the unit square that a particular point on an object gets mapped to in the first step. Note

that u and v are calculated differently depending on the object. From here, how do you find the coordinates (s, t) to look up in a texture map in terms of u, v, j, k, w and h, where j and k are the number of repetitions in the x and y directions, respectively, w is the texture width, and h is the texture height? (You may assume that both the *u,v* and *s,t* coordinate systems are oriented with (0,0) in the same corner.)

Solution:

$$\begin{aligned} s &= [u * j - floor\,(j * u)] * w \\ t &= [v * k - floor\,(k * v)] * h \end{aligned}$$

Here's an alternative answer that's faster and (possibly) easier to understand.

$$\begin{aligned} s &= ((int)\,(u * j * w)) \mod w \\ t &= ((int)\,(v * k * h)) \mod h \end{aligned}$$

6. **[1 point]** Given the ambient, diffuse, and specular components of a surface point's color, what's the equation for the pixel's color, adding in the color of the surface's texture map at that point? (**Hint:** You'll also need the color of the texture map and the *blend* value in the lighting equation but you shouldn't need any other parameters.)

   Solution:

$$final\,color = a\,color + (blend) * (t\,color) + (1 - blend) * (d\,color) + s\,color$$

7. **[1 point]** What is the purpose of the specular exponent in the Phong lighting model?

   Solution: The exponent determines how quickly specular reflectance falls off. The larger the exponent, the shinier the surface.

8. **[Extra credit]** Compare and contrast path tracing and ray tracing: what are the advantages and disadvantages to each? What are alternative global illumination methods? (Note that ray tracing does not cover global illumination.)

   Solution: Raytracing renders the scene from one specific view point and only cares about light that could enter the eye. It is good for getting specular highlights and specular reflection, but it needs an ambient hack to simulate diffuse reflection.

   Path tracing is an extension to ray tracing that stochastically covers global illumination (GI). This means that diffuse interreflection is rendered physically (well, up to the physicality of the scene and ignoring wave optics). A scene, when rendered until convergence with a path tracer, should look realistic, with the kinds of soft gradients and color bleeding you find in real life. Why "until convergence"?

   As mentioned above, path tracing is *stochastic,* meaning the final image is the average of many random samples. It will take a long time to get a good-looking result without noise. Compare that to ray tracing: fast rendering of diffuse light, with a few big effects simulated (refraction, reflection). If you're looking for speed and can afford to sacrifice or fake GI, ray tracing is the way to go.

   Some 3D artists prefer to avoid GI. Because GI renders the scene physically, there are lots of subtle connections between objects, lights, materials and so forth in a scene. Using plain old rasterization or ray tracing can give an artist more control over a scene's appearance. Of course, plenty of people regularly use GI as well.

   Here are some common ways to achieve GI (or an approximation thereof). Photon mapping is an extension of ray tracing that simulates diffuse interreflection by precomputing photons from lights. Some variants of path tracing include: bidirectional path tracing, where you trace from both the camera and the object and Metropolis Light Transport, which involves mutating light paths to take advantage of how similar local light paths can be. Finally, there are non-ray tracing solutions, such as radiosity, which is a matrix-based solution to global illumination, and spherical harmonics, which involves approximating lighting on a projected sphere. The latter two techniques are useful for interactive rendering.

   Naturally, it's impossible to give a complete overview of this topic here. You don't need to know about this in any detail. For more information on GI, see the textbook.