Assignment P3 : RAY TRACING

16 March 1995

Computer Graphics 2

Due 13 April 1995

For this assignment you will extend your ray caster from assignment P2 with features to increase realism: recursive ray tracing with specular reflection and refracted transmission, shadows, distance-attenuated lights, and antialiasing.

Shading

The shading formula you should use is:

$$\mathbf{I} = \mathbf{C}_{mat}\left[\mathbf{I}_a + \sum_i s_i \mathbf{I}_i k_{dr} \max\{(N \cdot L_i), 0\}/r_i^2\right] + \sum_i s_i \mathbf{I}_i k_{sr} \max\{(N \cdot H_i), 0\}^e / r_i^2 + k_{sr}\mathbf{I}_r + k_{st}\mathbf{I}_t$$

This is the same as in the previous assignment except for:

s_i is the shadow coefficient: 0 if light i is shadowed
and 1 if light i is unshadowed

r_i is the distance of light i from the point being shaded
(this term included for $1/r^2$ attenuation of light)

$\mathbf{I}_r$ is the radiance of light in the specularly reflected direction

$\mathbf{I}_t$ is the radiance of light in the specularly transmitted direction

See Foley for explanations of these terms (in particular, see sections 16.1.4, 16.5.2, and 16.12). The formulas are otherwise the same as before. To prevent infinite recursion on $\mathbf{I}_r$ and $\mathbf{I}_t$, only trace such rays to a maximum ray depth (user-specified, at run time).

Some details. Regarding shadows: note that we're not asking you to do anything fancy to simulate partial shadowing of light through transparent objects. When testing for shadows, you can treat each object as opaque. Regarding transmitted rays, when you find the refracted ray direction, you need to find the ratios of the indices of refraction of the material and the material you're entering. When you're leaving air and entering a material (such as glass) with index of refraction "ir", then this ratio would be 1/ir, but when leaving the material, entering air, the ratio would be the reciprocal of that. You'll need to do a test to determine if you're entering or leaving a solid. You can do this by comparing the incident ray direction and the normal vector, assuming your normal vectors point out. Regarding normal vectors: after doing the test above, but before shading, flip the normal vector if it and the direction D of the incident ray aren't on opposite sides of the surface (flip if $N \cdot D > 0$).

Antialiasing

To reduce aliasing, implement something at least as good as the following: Make one pass over the image, generating one ray per pixel, saving RGB radiances as a vector of 3 floats. During this pass also compute the average of the maximum channel value of all pixels: avgmax = $\sum_{x,y} \max\{r_{x,y}, g_{x,y}, b_{x,y}\}/(\#\text{ pixels})$. Because of distance-attenuation, this will not necessarily be near 1.

Then go back on a second pass, at each pixel testing the color there against the color of the four neighbors (above, below, left, and right) and supersample the center pixel 16 times (in a 4×4 grid) if any of the four neighbor's colors differs from the center color above some threshold. A good way to implement the threshold test is to compute $\Delta r^2 + \Delta g^2 + \Delta b^2$ where Δr, Δg, and Δb are the differences between red, green, and blue components, respectively, and if this sum of squares is greater than $(\text{avgmax} \times \text{thresh})^2$ then the colors are considered "too different". The parameter "thresh" should be some fraction between 0 and 1; perhaps .1 or so (bigger makes it faster, smaller makes the picture better).

You're free to use stochastic sampling (page 788) or recursive adaptive sampling if you're interested (page 714).

On the second pass, you can write out pixels. If you multiply them by 128/avgmax then that should guarantee a reasonable range of pixel values (not too dark, not too bright), since it maps the average radiance to pixel value 128. In spite of this, some of your pixel values might overflow (> 255), so you should peg at 255 to prevent this. The calculation and use of "avgmax" effectively implements automatic exposure control, much like automatic exposure control on a smart camera.

Test Scenes

We've set up three test scenes:

table.sc A simple test of shadows and specular reflection rays.

glass.sc A test of specular transmission rays. See description in *classdir*/`pub/p3/README` for a more complete description.

chex.sc A test of antialiasing.

What to Submit

Everything you submit should be in *yourdirectory*/`p3`. Submit the following:

1. Source code to your `raytrace` program, (called "raytrace", please).
2. A working executable of your `raytrace` program that we will be able to run (executable on either a DEC, SGI, or Sun4) with instructions in your README file on how we can run your program to reproduce "table.ppm".
3. PPM picture files for `table.sc`, `glass.sc`, and `chex.sc`. Call them `table.ppm`, etc. Do them all with antialiasing. Set your antialiasing threshold so the pictures look good, but don't take excessive amounts of time (we'll leave that judgement up to you). Use a maximum ray depth of at least 5 on all pictures.
4. A scene of your own creation: scene file and ppm file. Please be creative.
5. A text file `README` where you summarize briefly what you've done, what each of the files in your directory is, and a description of the new scene you created. For each picture, tell us: machine type computed on, maximum ray depth, threshold for antialiasing, compute time (seconds of CPU time), # objects in scene, # pixels, # samples (rays from eye including antialiasing rays), average # rays per pixel (including shadow rays and rays at all depths in ray tree).

Extra credit will be given if you implement data structures for optimization (grid, octree, or hierarchical bounding volumes), or recursive adaptive antialiasing as described in page 714 of the book.