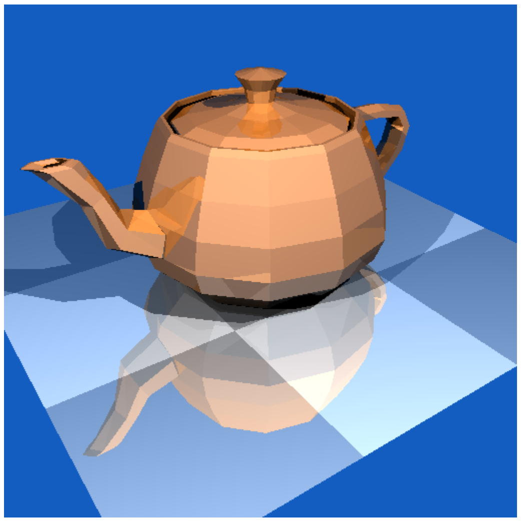
CMSC 435/634: Introduction to Computer Graphics

Assignment 2
Ray Tracing II
Due October 2, 2018 @ 11:59 PM

Before you start

This assignment builds on the previous one. If yours worked, I strongly recommend using your own code as a starting point for this assignment. If you were not able to get the first part of your ray tracer working, I will provide access to my program in `~adamb/public/hide` (after the late submission deadline for assignment 1). You could either use this code to figure out what was wrong with your assignment, or use it as a base for this one. I have also placed an executable for my program in `~adamb/public/shade`.

The Assignment



For this assignment, you will add mirror reflection and diffuse and Blinn-Phong specular lighting with shadows to your ray tracer. I have provided an example `teapot.nff` (spd3.14/teapot -s 3) that you can use for testing (In addition to the other spd models). Note that the teapot is made up of polygonal patches (abbreviation pp), so your program will need to support this input. You can simply ignore the normal information.

You will need to use additional parameters from the '`f`' lines in the nff file. The full set of '`f`' parameters is

```
f r g b Kd Ks e Kt ir
```

Where `r,g,b` is the base surface color, `Kd` is a coefficient to scale the overall diffuse contribution, `Ks` is the coefficient for both reflection rays and a Blinn-Phong glossy specular component with exponent `e`, `Kt` is the coefficient for refracted rays, and `ir` is the index of refraction.

In assignment 1, you needed to track the color at the closest intersection. For this assignment, you will also need to track the surface normal, intersection location, and additional surface parameters. Once you find that intersection, you will need to compute the surface shading. Given unit normal, `N`; unit vector from the surface intersection point toward the light, `L`; unit vector halfway between the light and ray, `H`; and light intensity of `1/sqrt(numLights)`, sum the contribution from each light. For each light, cast a shadow ray from the surface to the light. If that light is not shadowed, the diffuse and specular contribution from that one light would be computed as:

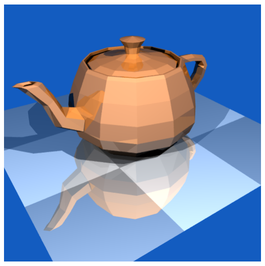
```
diffuse = max(0, dot(N,L))
specular = pow(max(0, dot(N,H)), e)
localColor.r += (Kd*r*diffuse + Ks*specular) * lightIntensity
localColor.g += (Kd*g*diffuse + Ks*specular) * lightIntensity
localColor.b += (Kd*b*diffuse + Ks*specular) * lightIntensity
```

Since the color of each reflection ray is the total color of whatever it hits, including its own diffuse, specular and reflection, compute the color in a recursive process:

```
Color trace(ray) {
  ...
  totalColor = localColor;
  if (Ks > 0 && !recursionLimit)
    totalColor += Ks*trace(reflectionRay)
  return totalColor
}
```

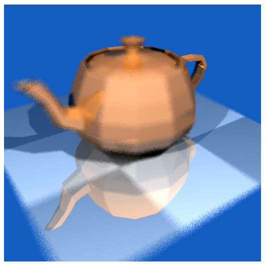
Stop the recursion if the number of bounces would be greater than 5. The easiest way to do this is to add data to each ray to track the "depth" of the ray---the number of bounces.

Extra Credit



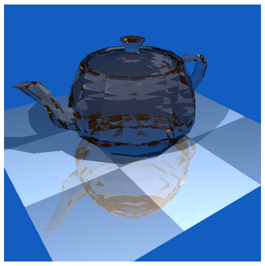
For up to 20 points of extra credit, implement stratified sampling ("jittering") for antialiasing by shooting multiple, jittered, rays per pixel. You should allow the user to specify the square root of the number of rays as a command line option (see getopt). The number of samples will be the square of the parameter (i.e. a parameter of 3 results in 9 samples), arranged in a jittered 3x3 grid). At right is the output of my program with:

```
./shade -j -s 3 teapot-3.nff stratified.ppm
```

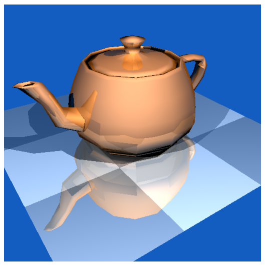


For up to 25 points of extra credit, implement depth of field by shooting multiple rays per pixel. You should allow the user to specify the square root of the number of rays and aperture size as command line options. The focal plane should pass through the 'at' point specified in the nff file. Note that if implementing depth-of-field and stratified sampling bothe the lens and the image should be sampled for each ray. At right is the output of my program with:

```
./shade -a 0.5 -s 3 teapot-3.nff dof.ppm
```



For up to 20 points of extra credit, implement refraction too. If your ray tracer supports concave polygons, the gears nff files include refaction. I have also provided an edited tetra version of `teapot-3.nff` called `refract.nff` that has been edited to use for refraction testing.



For up to 20 points of extra credit, implement phong/smooth shading with the "-p" command line option. At right is the output of my program with:

```
./shade -p teapot-3.nff phong.ppm
```

What to turn in

Turn in this assignment electronically by pushing your source code to your `assn2` GIT directory by 11:59 PM on the day of the deadline. We will be looking for multiple checkins documenting your development process.

As always, double check that you have submitted **everything** we need to build and run your submission, but **no** generated files (.o's, executables, or images). Be sure to include a Makefile that will build your project when we run '`make`', and a `readme.txt` file telling us about your assignment. Do not forget to tell us what (if any) help did you receive from books, web sites or people other than the instructor and TA.

(This page: www.csee.umbc.edu/~adamb/435/proj2.html)