6.837 Assignment 3B

Anti-Aliased Recursive Ray Tracing

Due 5pm Friday, 25 October 1996

Assignment 3A called for the implementation of an interactive rendering engine based on **ray casting**. Visibility at each eye ray was determined by propagating eye rays through pixel centers. The closest scene object encountered was then "shaded" at the relevant pixel. The shader in Assignment 3A, however, made no reference to, or use of, scene light sources or any multi-bounce paths from light source to eye.

Assignment 3B challenges you to complete the **radiometric** component of a rendering engine based on recursive **ray tracing**. You will do so by extending the function `Shade()` to aggregate radiance due to ambient and local illumination (with and without shadow rays), and due to reflection and transmission effects. To do so, `Shade()` will have to invoke `Trace()` – thereby completing the **co-recursive** structure of `ivray`. You will also implement simple screen-space anti-aliasing by supersampling.

Your first task is to complete the four critical portions of `Shade()` (in `Shade.C`) to achieve recursive ray tracing. Specifically, you must implement four actions corresponding to various values of `interface::shademode`, each of which currently produces an error message:

`case SHADE_AMBIENT`: Aggregate radiance due to ambient illumination (invoke `ivray -s A`).

`case SHADE_LOCAL`: This mode simply aggregates radiance due to direct illumination by point light sources, without regard to whether the optical path to those sources is obstructed. This is essentially what is done by the graphics hardware (as discussed in class); consequently, when `ivray -s L` is invoked and the ray trace button pressed, one should observe a reasonably close match to whatever the hardware produces.

`case SHADE_DIRECT`: Similar, except that in this mode shadow rays must be spawned to determine whether each light source illuminates the surface fragment (invoke `ivray -s D`). Note that you must handle three kinds of Inventor light sources: local points, infinite points, and directional spotlights!

`case SHADE_RECURSIVE`: Full-blown recursive shading (invoke `ivray -s R`). Aggregate radiance due to reflection (determined by spawning reflection rays) and refraction (determined by spawing refraction rays).

Note that clever use of the C-language `switch` primitive, and ordering of the four shading cases above, yields a shader that handles all four modes without any code duplication.

Your second task is to extend function `RayTrace()` in `RayTrace.C` to implement regular and jittered supersampling (using invocations `ivray -p N foo.iv` and `ivray -j N foo.iv`, respectively).

Your third task is to modify the functions `RayTraceStart()` and `RayTraceFinish()` in `RayTrace.C` in order to output statistics about your ray tracer, e.g. total number of rays cast, average number of rays cast per pixel, average recursion depth, etc.

Extensive guidance for these tasks can be found in the files `README`, `Shade.C`, and `RayTrace.C`. **You need modify only the two files `Shade.C` and `RayTrace.C` to complete this assignment.**

The Point: First, you will implement the recursive spherical integration of irradiance required by ray tracing. Second, you will understand anti-aliasing by (both regular and pseudo-random) multisampling. Third, you will learn about the behavior of your ray tracer. Fourth, if you wish (i.e. for extra credit), you can implement one or more generalizations to ray tracing: a hierarchical spatial data structure (e.g., an octree) for acceleration of ray-casting; motion blur by sampling over time; soft shadows by sampling over area light sources; dispersion by sampling over wavelength, etc., or any other ray tracing extension you desire.

Tools: As usual,

```
cd /mit/imagery4/6.837
```

and

```
cp -r ps3B students/uid
```

to copy the original `ps3B` code into your working area. Finally,

```
cd students/uid/ps3B ; make
```

to produce the executable `ivray`, which you can run as

```
ivray ps3Btest?.iv
```

where `?` is `1, 2, 3,` or `4`, or

```
ivray ../../../ivexamples/*.iv
```

where `*.iv` comprise many other examples. (Keep in mind that `ivray` does not currently handle spline surface primitives.)

Evaluation: We expect your `ivray` to produce a (geometrically and radiometrically) correct ray-cast rendering of each of the four test Inventor files above **(from any view point)**, when run with any of the flags `-s A`, `-s L`, `-s D`, and `-s R`. We may also test it against some more challenging inputs of our own design.

To turn in (details TBA by email):

Your modified files `Shade.C` and `RayTrace.C`, and

Your stripped executable `ivray`.