6.837 Assignment 6A

Ray Casting

Due 5pm Friday, 22 October 1999

We have finished our tour of the graphics pipeline, concluding with a scan-conversion assignment. There, each pixel was painted by a primitive deduced through incremental scan-conversion of a polygonal primitive (a triangle). Geometry (i.e., shape) was resolved at each pixel through a z-buffer visibility algorithm. Radiometry (i.e., shading) was determined at each pixel through screen-space interpolation of values established at polygon vertices by evaluation of a Phong-like shading model (actually, the diffuse term only).

We next considered **Ray Casting**, in which the assignment of scene primitives is determined at each pixel through an intersection (and sorting) computation in object space. This assignment calls for you to complete the implementation of a ray-casting algorithm by completing code to compute the (object-space) intersection of a ray with a sphere, cone, cylinder, and polygon. (We supply code for parallelepiped intersection, and some of the polygon intersection code.) You are also asked to supply code which generates an eye ray (i.e., a primary ray) for each pixel of a specified viewport and view frustum, in perspective. (We supply example code for the case in which the projection is orthographic, i.e., the eyepoint is at infinity.)

[Part B of this assignment, **Ray Tracing**, will call for the completion of the *radiometric* component of a ray tracer – that is, the recursive weighted integration of radiance at a set of sample points generated by ray casting. Part B will be handed out a week from today.]

We have modified `SceneViewer` by adding two “Ray Cast/Trace” buttons to the left of the viewport, producing a new executable called `ivray`. While viewing any Inventor scene, pressing any of these buttons causes a *callback* to the function `RayTrace()` in `RayTrace.C`. `RayTrace()` is passed a view frustum, a viewport descriptor, and a scene descriptor (actually two lists, one of `Objects` and one of `Lights`). It loops over each pixel of the viewport, calling `ComputeEyeRay()` at each pixel, then *tracing* the resulting ray via a call to `Trace()`. The routine `Trace()` identifies the primitive nearest the eye along the ray via a call to `RayCast()`. This pixel is then “shaded” using an intrinsic object color. (Assignment 6B calls for the implementation of a more sophisticated shading algorithm, as well as the implementation of antialiasing through supersampling.)

Your task is to complete the critical portions of `ComputeEyeRay()`

(in `EyeRay.C`), and `RayCast()` and all indicated object-space ray-primitive `Intersect` routines (in `RayCast.C`). Extensive comments to guide you can be found in the files `EyeRay.[Ch]`, and `RayCast.[Ch]`. **You need modify only the files `EyeRay.C` and `RayCast.C` to complete this assignment.**

The Point: First, you will implement the correct generation of world-space eye rays under a general perspective viewing transformation. Second, you will understand ray-primitive intersection and simple ray-casting optimization techniques based on bounding boxes. Third, if you wish (i.e., for extra credit), you can implement yet more general viewing transformations (Imax, OmniMAX, etc.) or a hierarchical spatial data structure (e.g., an octree) for acceleration of ray-casting, or the `Intersect()` routine for bicubic Bézier patch primitives, or any other ray-casting extension you desire.

Tools: Type

```
attach 6.837
```

You should then

```
cp -r /mit/6.837/ivray/ivray.tar .
tar xvf ivray.tar
```

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

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

```
ivray asst6Atest?.iv
```

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

```
ivray foo.iv
```

where `foo.iv` is any other valid Inventor file. (Keep in mind that `ivray` does not currently handle spline surface primitives.)

Evaluation: We expect your `ivray` to produce a (geometrically) correct ray-cast rendering of each of the four test Inventor files above **(from any view point)**. We will also test it against some more challenging inputs of our own design.

To turn in (details TBA):

- Your modified files `EyeRay.C` and `RayCast.C`; and
- Your stripped executable `ivray`; and
- Any other files you modify for extra credit.