6.837 F02 Assignment 4

Polygon Scan Conversion

Our tour of the classical graphics pipeline is nearly complete. Today, we consider 3D *scan-conversion* algorithms, which operate by considering each raster of the viewport in turn, and processing (and depth-ordering) *all* primitives which might contribute pixels to that raster. This assignment challenges you to implement a scan conversion algorithm for overlapping 3D triangles. In contrast to assignment 3, we will now fill the triangles, resolve their visibility, and compute and interpolate color information. You are encouraged to build upon the code that you have written for assignment 3.

As in assignment 3, we have modified the Inventor `SceneViewer` by adding several buttons to the left of the viewport, to produce a new program called `ivscan`. While viewing an Inventor scene, pressing any of these buttons causes a series of C++ *callbacks* to occur. One callback per polygon edge is made to `EdgeRec::Init()`. One callback per frame is made to `RenderScene()`. We have supplied incomplete versions of both routines. The assignment challenges you to complete the critical inner operations of both, producing a semi-interactive polygon scan converter.

`EdgeRec::Init()` takes as input a single polygon `Edge` (with 2D screen-space vertices, eye-space z values, lighting information, etc., as defined in `Edge.h`). It is called exactly once per edge for each triangle that survives clipping (which we perform in `ScanWrap.C`). You must fill in `EdgeRec::Init()` so that it correctly initializes each field of the `EdgeRec`.**You must now initialize the additional fields that were not necessary for assignment 3.**

`RenderScene()` is passed an `Environment` variable, and a function `setPixel()`. The `Environment` data structure contains a viewport descriptor, a blank framebuffer, and a pointer to an `EdgeRecTable`. (The `EdgeRecTable` is one concrete implementation – an array of linked lists of `EdgeRecs` – of the abstract "Event Bucket" data structure described in class.) `RenderScene()` is called exactly once for each rendered scene (i.e., each time one of the scan conversion buttons in `ivscan` is pressed). Finally, you can use `Raster.setPixel(x, color)` to write a color value to a specified pixel of the current raster, and `Raster.write(y)` to write the completed raster into the framebuffer.

Your job is to fill in `RenderScene()` so that it produces, through a series of calls to `RenderScanLine()` [and thus to `setPixel()`], a correct scan-converted image of the scene. Here, "correct" means three things: first, that at each pixel the correct triangle contributes its color at the pixel; second, that the color at the vertices is correctly computed using the diffuse **and specular** parts of the Phong model; and third, that the computed color at the pixels inside the triangle is the correct Gouraud

interpolation of the colors at the vertices.

Extensive comments to guide you through both tasks can be found in the files `README`, `EdgeRec.C`, `EdgeRecList.C`, `Scan.C` and `ScanWrap.C`. **You need modify only these four `C` files to complete this assignment.**

The Point: You will reuse the correct initialization of the `EdgeRec` and AEL (Active Edge List) data structures from assignment 3. In addition, you will first initialize the additional fields required for handling color and visibility. This includes the calculation of color for each vertex, given the normal, material property and set of light sources. Only the diffuse and ambient terms are provided. You must add the specular component of the Phong model using the half vector and the exponent. Second, you will implement the incremental maintenance along polygon edges of depth, and color in order to achieve correct scan conversion, visibility determination, and Gouraud interpolation of shading, respectively. Third, you will understand the (integer-centered) screen-space coordinate model, and implement a sampling rule that renders polygon meshes without dropouts or overdraws. Fourth, if you wish (i.e., for extra credit) you may also implement a more powerful scan converter: per-pixel Phong calculation,; per-pixel normal interpolation (a.k.a. Phong interpolation); real clipping; better interactivity; texture mapping, etc. Note that the normal field in `EdgeRec.h` needs to be maintained only if you choose to implement per-pixel Phong evaluation for extra credit. See the `README` file for details.

Tools: Same tools as in assignment 3. A "reference executable" can be found at `/mit/6.837/F02/asst4/irix,linux/ivscan`.

Evaluation: We expect your `ivscan` to correctly scan-convert all of the included test Inventor files, **from any view point**. We will also test it against several more challenging Inventor objects. We will pay particular attention to the sampling of the polygon boundaries and to the use of consistent shadow rules (i.e. for connected polygons, the pixels on the common edges should be drawn exactly once).

To turn in (details online):

- Your modified `EdgeRec.C`, `EdgeRecList.C`, `Scan.C`, and ScanWrap.C;
- **Any other `.C` or `.h` files you modify**;
- Your **stripped** executable `ivscan`.

We will test your source for compilation using the assignment `Makefile`. Thus you **must** include any files that you have modified (we will supply unmodified files from the assignment source).

This assignment is due 5pm Friday, 11 October 2002.