Analysis of Flights of Fantasy

Jed Margolin

{Company's} Mr. {Person's} Prior Art References

> Flights of Fantasy, Christopher Lampton (The Waite Group 1993). Discusses computer graphics algorithms for the purpose of generating perspective scenery for flight simulators. Includes source code examples.

Flights of Fantasy – Programming 3-D Video Games in C++ by Christopher Lampton (**ISBN:** 1878739182). *{Ref. 7a}* Mr. {Person's} characterization of the book does not do the book justice. This is an excellent book that teaches practical computer graphics including 3D graphics. Only a small part of it is about generating perspective scenery. Scenery is taught in **Chapter 13: Fractal Mountains and Other Types of Scenery (**Pages 439 – 455). That's 17 pages out of a 556 page book. From page 440:

> So far, though, we've populated our world with very few objects-a cube here, a pyramid there. How do we build up something that looks like the scenery in the real world?
>
> **Landscaping the Imagination**
>
> Building scenery for a flight simulator is a tedious task, but the results can be worthwhile. So far, we've given you no tools in this book for this task other than ASCII text files that can hold object descriptions and that can be edited with an ASCII text editor. Ambitious readers, however, may want to write their own scenery- and object-design utility that uses the mouse, the joystick, or the keyboard to drag polygons around and to build buildings, mountains, and other pieces of the landscape. You should be aware, though, that such programs are difficult to write, with much of the difficulty lying in the user interface itself.
>
> Without such a utility, you have <u>two alternatives for developing scenery</u> for your flight simulators: <u>edit the ASCII object descriptors by hand</u> or <u>write custom programs for generating specific types of scenery</u>. In this chapter, we'll give you some tips about both methods.

Lampton then teaches the use of fractals for generating perspective scenery. If {Company} generated fractal terrain instead of using the DTED the FAA would not have certified it. And the system would also have been useless for synthetic vision.

The Lampton Flight Simulator is a Simulator. It does not contain the elements needed to fulfill FAA's definition of synthetic vision. As previously discussed, according to FAA:

> Synthetic vision means a computer-generated image of the external scene topography from the perspective of the flight deck that is derived from aircraft attitude, high-precision navigation solution, and database of terrain, obstacles and relevant cultural features.

The Lampton Flight Simulator does not use the DTED and is not used with a real, physical aircraft.

However, there is information relevant to the DTED. In **Chapter Eight Polygon-Fill Graphics** it teaches the same method of storing polygons as is explained in my article **Polygon Databases - Is a Digital Terrain Elevation Database (DTED) really a Polygon Database?** *{Ref. 7b}*

Lampton teaches the use of a Point List (**vertex type *vertex; // Array of vertices in object**) and a Face List (**polygon_type *polygon; // List of polygons in object**) because (page 269):

> It would be wasteful to store two separate lists of vertices, so we've defined one of these lists (the one in the polygon structure) as a list of pointers that point at the vertex descriptors in the list maintained by the object structure. This concept is illustrated in Figure 8-4.

Figure 8-4 makes it clearer than the program snippet. The following is from Lampton CHAPTER EIGHT POLYGON-FILL GRAPHICS (page 269):

The Object-Type Structure

Now we need a structure that will describe an object made up of polygons, similar to the *shape_type* structure that we used in Chapter 7 for storing wireframe shapes. Here's the structure definition:

```
struct          object_type {
     int        number_of_vertices;           //  Number of vertices in object
     int        number_of_polygons;           // Number of polygons in object
     int        x,y,z;                        // World coordinates of
                                              //    object's Local origin
     polygon_type    *polygon;                // List of polygons in object
     vertex type   *vertex;                   // Array of vertices in object
     int        backface_removal;             // Do we want backface removal?
};
```

You'll notice that much of this structure is redundant. We've seen some of these same fields in the *polygon_type* structure. Why do we need a list of vertices in both the polygon structure and the object structure? The reason is that sometimes we'll want to treat a *polygon* as a list of vertices; at other times we'll want to treat an object as a list of vertices. It would be wasteful to store two separate lists of vertices, so we've defined one of these lists (the one in the polygon structure) as a list of pointers that point at the vertex descriptors in the list maintained by the object structure. This concept is illustrated in Figure 8-4.

The backface_removal field might look mysterious to you. We'll discuss this field momentarily.

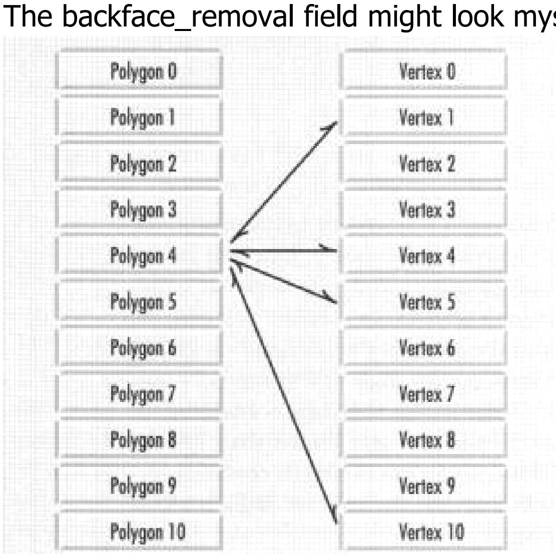

Figure 8-4
The pointers in the polygon list, pointing to the vertices in the vertex list

----

Lampton also discusses the basics of Polygon Smoothing (page 535):

CHAPTER SIXTEEN THREE-DIMENSIONAL FUTURE

Polygon Smoothing

No amount of light sourcing can turn a polygon-fill object into a completely realistic representation of an actual object. The problem is that real objects, unlike polygon-fill objects, have curves. With some exceptions (such as crystalline structures), nature isn't made of polygons. There are techniques, however, that can turn a surface made out of polygons into a smoothly curved surface, at least on the computer screen. The most popular of these techniques are *Gouraud* and *Phong* shading.

Gouraud shading is the simpler of the two, though it produces slightly less realistic results than Phong shading. Instead of filling polygons with a solid color based on the angle of that surface relative to a light source, Gouraud shading interpolates the color of each pixel on each scan line inside the polygon, based not only on the light-sourced color of the polygon but on the colors of adjacent polygons as well and the distance of the pixel from the edges of the polygon. Pixels near the center of the polygon are given the color that all of the pixels in the polygon would be given using the simple light sourcing techniques described earlier, but pixels toward the edge of the polygon have colors closer to those of adjacent polygons. Phong shading uses much the same technique, though the calculations used to determine the colors of the pixels are a bit more complex.

Compared to ordinary polygon-fill graphics, Gouraud and Phong graphics require a great many computations and are therefore quite slow. At present, it's not likely that a straightforward implementation of either technique could be used in a real-time flight simulator. However, at least one popular three-dimensional game - *Links*, the golf simulation from Access Software - uses

533

FLIGHTS Of FANTASY

rendering techniques that look suspiciously close to either Phong or Gouraud shading. This is made possible by the fact that *Links* is not a real-time simulation. The player watches each scene drawn on the display over a period of seconds. Such slow rendering would be unacceptable in a flight simulator, where the out-the-window view is constantly changing and at least a dozen frames a second are required to make the animation seem smooth. But in *Links*, the view of the golf course only changes when the ball is hit and therefore remains static for long periods of time. And the high quality of the graphics is well worth the amount of time required to produce them.

The results produced by these polygon-smoothing techniques are a dramatic improvement over the polygon-fill techniques we've used in this book. Will it be possible in the future to use them in a flight simulator? Almost certainly. In fact, simplified versions of these techniques have already found their way into several recent flight simulators.

The following is what is in the Lampton book. The first is the Table of Contents. The second is a more detailed list of the contents.

Flights of Fantasy – Programming 3-D Video Games in C++, Christopher Lampton, The Waite Group, 1993

Table of Contents

Contents

References

Ref. 7a - **Flights of Fantasy – Programming 3-D Video Games in C++** by Christopher Lampton (**ISBN:** 1878739182).

Ref. 7b - **Polygon Databases - Is a Digital Terrain Elevation Database (DTED) really a Polygon Database?**, Jed Margolin, http://www.jmargolin.com/patents2/pilotrefs/PolygonDatabases2.pdf

---