Lecture 8: Polygon Rendering

We have now covered most of the basic tools required to put together a simple graphics application, and so we will consider the example of a video game based on polygon rendering. The basic outline is expressed by the data flow diagram shown in diagram 8.1. To create the impression of a smooth movement it is necessary to generate at least ten frames each second, and even this rate is inadequate for fast moving objects. Television produces 25 frames per second.

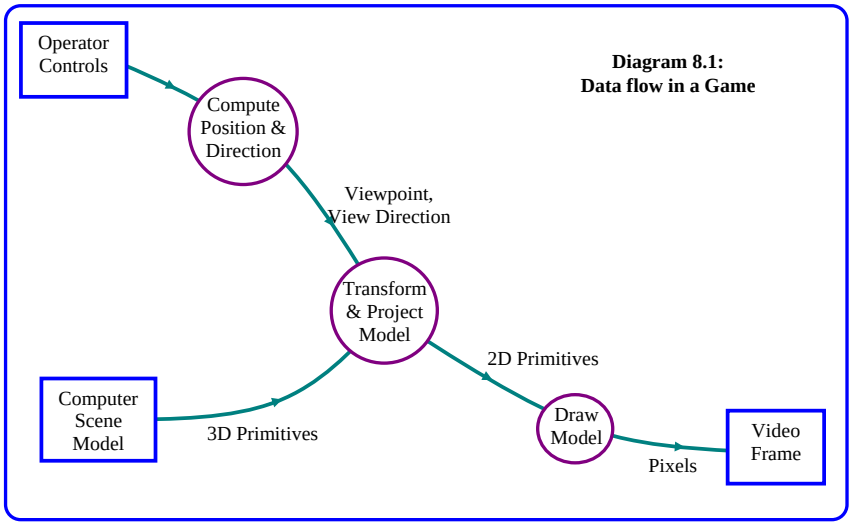


Diagram 8.1: Data flow in a Game

Polygon maps

For most applications a graphics model of some sorts can be constructed out of a set of polygons, and this is the normal practice in computer animation. It is easy to see that simple rectangles are all that is required to build a crude representation of, for example, a race track, since the tarmac, hard shoulders, fences and grass verges are all flat and regular. Buildings tend to be rectanguloid structures, and complex details, are usually not shown. Usually, the surrounding terrain is viewed from a distance. Flat areas can be represented by rectangles coloured green for grass or brown for ploughed fields. For rougher terrain, triangulation is an appropriate modelling method. A set of grid points is selected, for example a regular array in x and z with variable height y, and joined to form triangles as illustrated by Diagram 8.2.

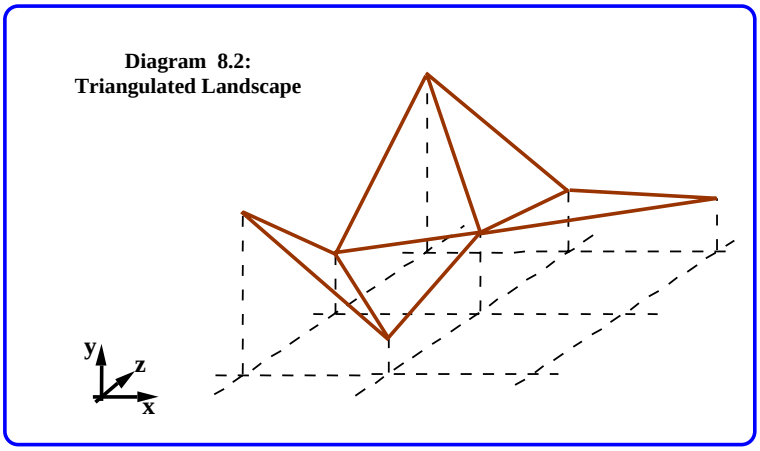


Diagram 8.2: Triangulated Landscape

Fractal Landscapes

Specification of a landscape is a time consuming problem. However, since much of the detail need not be exact, it is possible to start with an approximate triangulated surface, based on a coarse grid of points, outlining the main peaks and valleys of the scene, and refine it automatically. This refinement can be done by a number of processes, of which a typical one is shown in Diagram 8.3. For line joining two points in the original triangulation the mid point is displaced in a random direction by a random amount. The new mid points are joined up as

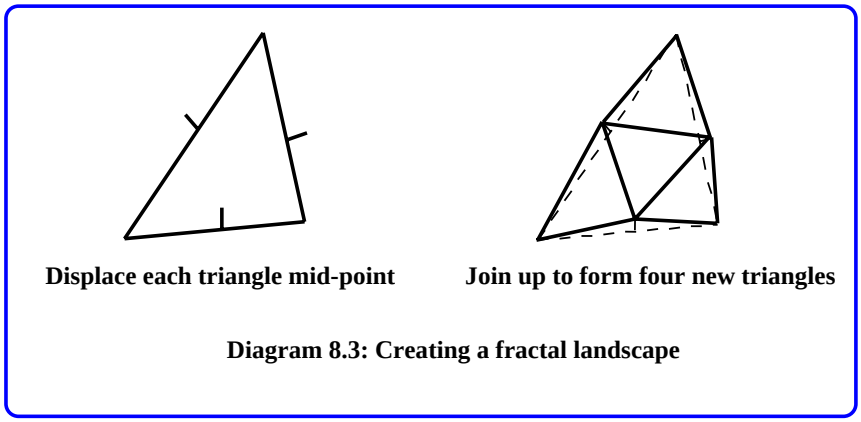
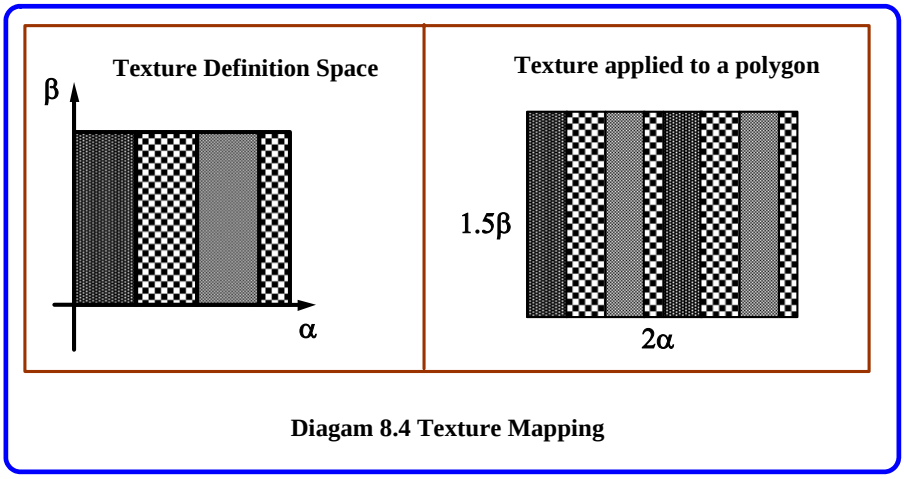


Diagram 8.3: Creating a fractal landscape

shown to form four new triangles. These triangles are then processed similarly until the required level of detail is achieved. This is one example of a fractal, resulting from a chaotic function. These are iterative functions which neither converge or diverge.

Texture

One simple and highly effective way of increasing the realism of any polygon based scene is to use a texture when filling in the polygons. Texturing is essentially a mapping between the texture space, and the polygon. In the simplest case, we are mapping one rectangular space into another as shown in Diagram 8.4. In general however, we need to map the texture on a general quadrilateral, and this is done by bi-linear interpolation.



Diagam 8.4 Texture Mapping

Gouraud Shading

Another way of achieving improved realism in animation systems is to use shading instead of (or sometimes as well as) texture. This means that the pixel intensities change across the polygon, though their colour remains constant. The most common technique is called Gouraud shading. We will not go into great depth as to how it works, but in essence it uses linear interpolation of shade values set at the vertices of a polygon. Consider the triangle shown in Diagram 8.5. The desired intensities (I1, I2, I3) at the three vertices can be set by the game designer, or calculated using the known position of the light sources and physical laws, in particular Lambert's cosine law. Any edge pixel can be calculated by linear interpolation, as shown for pixel *Ie*. Once the intensities at each edge pixel have been found, the centre pixels intensities can be found by processing a series of scan lines as shown in stage 2 of diagram 8.5.

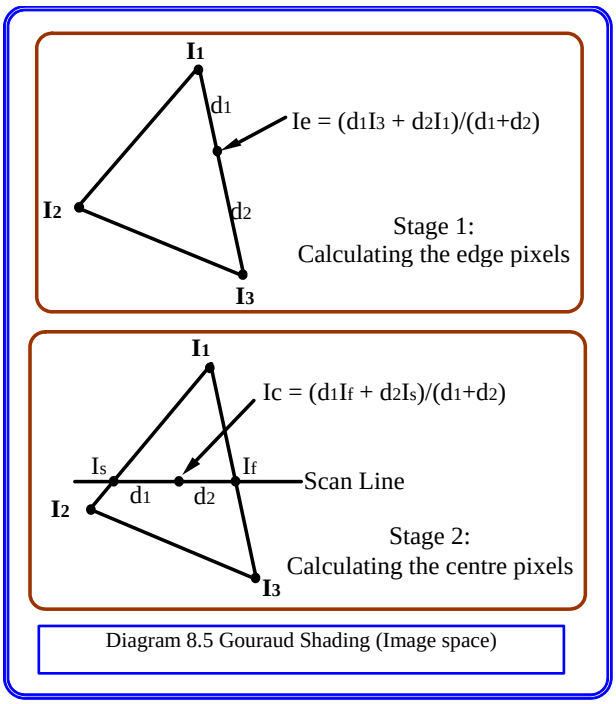


Diagram 8.5 Gouraud Shading (Image space)

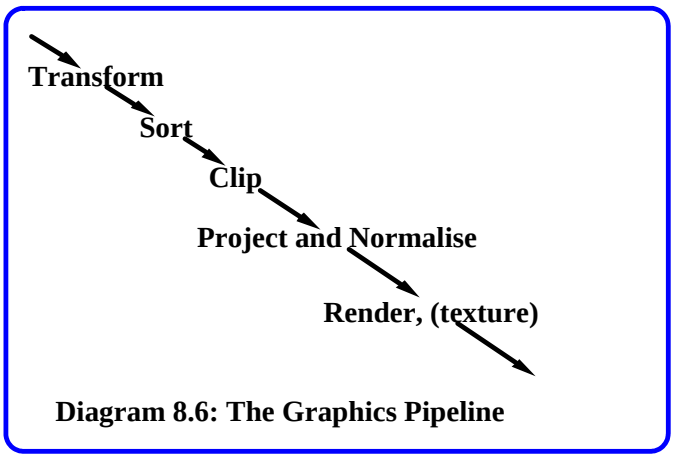


Diagram 8.6: The Graphics Pipeline

Computation pipeline

Animations must produce a succession of images that change sufficiently fast to give apparent smooth motion. As we noted above, the minimum requirement is about 10-15 frames per second. During the frame time it is necessary to read all the controls, compute

where the viewpoint has moved to, and its new direction, transform the polygons, clip them to the viewing volume, project them and render them with the appropriate texture. This has traditionally been achieved with pipeline processing as is shown in principal in diagram 8.6.

Limitations of Polygon Systems

There are many limitations of polygon systems, particularly in relation to natural scenes. Because they have linear boundaries, they cannot represent smooth curves very easily. The eye is very sensitive to straight line approximations to curves, and to achieve realism on smooth contours, many polygons are needed. Frequently natural objects cannot be easily decomposed into polygons (or patches). Consider for example a tree. For simulators which can be used for low flying (eg helicopters) it is not possible to represent trees with textures, since it is necessary to fly past them. Generally speaking, the realism in these cases is poor. Amorphous objects, such as fog, clouds, smoke and fire cause problems. Often flames and smoke can be included by using a series of polygons to overlay the scene, rather like running a short video in front of the scene. In many cases special purpose tricks are needed for these effects.