

CHAPTER 1 – INTRODUCTION

1.1 POINTS, EDGES, AND SURFACES

The purpose of computer graphics is the manipulation and visual display of existing or imagined objects or data with physical dimensions. The display output of “traditional” computer graphics, ignoring virtual reality environments that provide the illusion of 3 dimensions, is the 2D representation on a display device. Many geometric and topological concepts are involved through algorithms and data structures that deal with points, lines, surfaces and volumes – 0-, 1-, 2-, and 3-dimensional entities.

Consequently, data structures and related algorithms in computer graphics are oriented around points (or vertices), lines (or edges), surfaces, and volumes. Lines, surfaces, and volumes must be represented eventually by points that are involved in forming the boundary of the higher dimensional entity. However, although points are viewed geometrically as having 0-dimension, they are represented with 3 coordinates since they must exist in 3-dimensional space in order to contribute to the concept of boundary of higher dimensional objects. In practice, 4 coordinates are used to represent a point with conventional homogeneous coordinates.

It is difficult to classify problems according to their dimension since the question must be asked, “Which part of the problem is being classified?” For example, the representation of solid objects, inherently 3 dimensional, may use a collection of volume elements or a collection of 2D surfaces which bound the solid. In turn, volume elements must be bound by surfaces, surfaces must be bound by edges, and edges must be bound by points. Eventually, points must be represented

somewhere in the relevant set of data structures. Typically, elements with dimension between points and that of the relevant object are represented implicitly if not explicitly.

It is true, however, that different approaches to the same problem may use points or edges or surfaces as the focus of an algorithm which deals with edges or surfaces or volumes. The “focus” of an algorithm can be considered as the controlling factor in loops or recursion and is often used in the complexity measure of the algorithm. For example, graph algorithms may measure complexity in terms of the number of vertices or the number of edges in the graph.

As represented in the title of this research, “Point Based Approaches in Graphics,” two areas are investigated with point-based algorithms and data structures that are traditionally viewed as edge oriented.

Solids, surfaces, and edges each consist of an infinite number of points. The difference between these items of different dimension is actually the dimensionality of the connectivity relationships between the infinite number of points composing each. It is sometimes not clear what type of connectivity to consider. For example, Arthur Loeb [1] approaches polyhedra with a type of duality that on the one hand stresses the concept of a polyhedron as a set of connected items of different dimensionalities but on the other hand as a set of rigorously defined points. These approaches give either a connectivity or a symmetry point of view.

In like manner, many problems of computer graphics and related areas can be viewed from multiple perspectives. The choice of viewpoint will effect the data structures and algorithms used in the problem solution.

The remainder of this section will consist of a brief survey of some classical problems of computer graphics and the dimensionality of typical solutions.

An example application that can be approached in a number of ways is that of solid modeling. One approach to organizing solid modeling techniques includes decomposition models, constructive models, and boundary models[2]. Decomposition and constructive models are described as viewing solids as point sets with decomposition approaches discretizing the set and constructive approaches constructing the set from simpler sets. The boundary model represents a solid indirectly by forming representations of the bounding surfaces.

These boundary models can be vertex-based, edge-based, or polygon-based. A vertex-based model defines bounding faces of the surface in terms of vertices with the actual coordinates stored in a separate vertex data structure. Edge-based models are useful if curved surfaces are involved and represents a face boundary as a closed sequence of edges (loop) with vertices represented only as end points of edges. A model that needs only to represent planar faces may use a polyhedral model where the faces are stored as polygons with each polygon represented as a sequence of coordinates. A solid is represented by a collection of faces.

The data structures of the edge-based models are perhaps the most interesting. Since an edge is the “middle” layer in the point to edge to surface hierarchy, data structures often represent each explicitly. The doubly-connected-edge-list is an example of such a data structure [3]. A slightly more elaborate structure that also includes loop information in edge nodes is the winged-edge data structure [4].

Perhaps the most obvious example of a class of problems that can be solved using multiple approaches based on varying dimensionality are graph algorithms. Graphs are a source of

problems to examine in discussing the usefulness of points and edges in algorithm development since a graph consists of only points and the edges between them and is not concerned with faces or any higher dimensional structure.

As expected there are multiple data structures that can be used to represent graphs and these data structures affect the algorithms used and their efficiency [5]. The standard adjacency matrix approach represents edges by the presence of a non-zero entry in a matrix indexed by points. This data structure represents edges but is indexed by points. The other common approach to graph representation, the adjacency-list uses a list of neighbor points to represent the edges incident on a given point. Here, edges are represented by the presence of a point in a list where the individual lists are indexed by the originating point. Both data structures index edges by points. Space considerations are also important. The adjacency matrix approach requires $O(V^2)$ space while the adjacency-list structure uses $O(V + E)$ where V is the number of vertices and E the number of edges in the graph.

Graph searching problems are important in a number of contexts including connectivity and checking for cycles. Since the vertices are the objects being searched it is natural for algorithms to be driven by vertices. However, there must be a path used to get to a vertex so that the edges must play a key role in any search algorithm. In some applications, such as maze traversal, the paths (edges) are the important part of the result.

Depth first search using the adjacency matrix approach moves from a given vertex by examining the matrix to determine if there is an edge to each of the remaining vertices. Thus, the algorithm has complexity proportional to V^2 . If the adjacency-list structure is used, edges from the given vertex are considered and the complexity is proportional to $V + E$. The choice of data structure, and therefore algorithm, is based on the density of the graph.

An additional example of differing approaches driven by points or edges is the minimum spanning tree problem. The priority first search approach modifies a search algorithm to consider points in an order determined by the length of an edge from the currently constructed tree to vertices not yet included. The vertices of the graph are divided into three sets: vertices contained in the current tree, fringe vertices which are waiting to be examined, and unseen vertices whose existence has not been detected by the algorithm. The basic algorithm is to move one vertex from the fringe list to the tree and to move any newly discovered vertices to the fringe list. Note that the algorithm seems to be driven by the vertices of the graph since they are the objects of consideration at each stage and all vertices must be included in the final minimum spanning tree. However, the priority queue used to choose the next direction to “search”, while consisting of a collection of vertices, is ordered based on the length of edges. This algorithm has complexity for sparse graphs proportional to $((E + V) \log V)$.

Kruskal’s algorithm is an alternative approach which seeks to add edges to an existing tree by finding the shortest edge that does not introduce a cycle. The algorithm starts with a forest of trees, each consisting of a single vertex, and combines trees based on edges between them until one tree remains. An important data structure is again the priority queue but the entries are edges rather than vertices. The complexity of this algorithm is $(E \log E)$.

Other similar examples, such as shortest path, could be discussed where either points or edges are used as the driving force behind the algorithm. In the graph algorithm examples the choice of algorithm is guided by the number of edges relative to the number of vertices. In the algorithms considered in this research, this consideration is not sufficient. Surface reconstruction problems begin with a set of points but have no knowledge of the edges or the number of edges needed to construct the surface. Polygon filling is, by nature, a much different type of problem where the

polygon can be viewed as a graph consisting of a single cycle and where the number of edges and vertices are in one-to-one correspondence.

1.2 POLYGON FILLING

A polygon consists of an infinite number of points. The problem of polygon filling is one of mapping the points inside a polygon to pixels on a display. Each pixel represents a range of points. Perhaps the problem is more accurately stated as mapping a polygon to a display and then mapping each pixel inside the mapped polygon to a set of points so that the pixel best represents the points mapped to that portion of the display.

Each of the edges that bound the polygon consists of an infinite number of points that are also mapped to the display and to pixels. The edges are bound by a pair of points each of which is mapped to a pixel. The connectivity relationships are known and include the polygon edges, the points bounding the edges, pixels on scan lines, etc.

The problem, then, is one of mapping a polygon to a set of pixels with the needed connectivity relationships known. Traditional approaches use a mapping algorithm that is driven by polygon edges. This research presents two approaches that are driven by the points that are the vertices of the polygon.

1.3 SURFACE RECONSTRUCTION

The general problem of surface reconstruction begins with a finite number of points sampled from the infinite number of points on a surface. The initial steps in the problem solution are to

choose a subset of the sampled points and to establish relationships between these points to represent the surface. A subset of the points is chosen so that the resulting mesh or surface is not too dense to be useful.

The problem is, therefore, one of establishing connectivity among points to represent a surface. Edges are often used as an intermediate step or to represent the surface as a collection of polygons. Many algorithms for surface reconstruction begin with a large set of edges and work toward a simplified surface. The approach described in this research begins with points and uses edges only to represent the reconstructed surface.