Visualizing Mesh Data Structures and Algorithms

Bryan Neperud
Department of Computer Science
Michigan Technological University
Houghton, MI 49931
bmneperu@mtu.edu

Abstract

The winged-edge and half-edge data structures are widely used in computer graphics to represent meshes, but they are complex structures that are difficult to learn. Visualization is useful for dealing with the difficulty, but current visualization methods have problems. Drawing pictures by hand is tedious and error prone, and there are no suitable software systems available. This paper describes a new software system for visualizing the winged-edge and half-edge data structures and simple algorithms using them. The system has two components. The first component displays a mesh while exposing the data structure storing that mesh. The second component works with the first to display and trace through simple algorithms that operate on the displayed mesh. By using this system, users are able to focus on learning the data structures rather than dealing with the difficulties of other visualization methods.

1 Introduction

The winged-edge [1] and half-edge [2] data structures are widely used in graphics software to efficiently store and manipulate meshes. The open source system OpenMesh [3] and the Computation Geometry Algorithms Library (CGAL) [4], for instance, use the half-edge data structure. Graphics programmers who work on mesh or related modeling systems must understand these data structures. Unfortunately, the winged-edge and half-edge data structures can be difficult to learn as they are based on complex relationships among adjacent mesh elements that can be hard for beginners to keep track of. Visualization is useful for dealing with this difficulty.

Visualization can be done by hand drawing pictures or using visualization software. Drawing pictures by hand has many problems, though. Simply having to draw the picture is itself a distraction from the goal of learning and understanding the data structure. More importantly, drawing pictures by hand is tedious and error prone. The drawing process itself is simple enough, mostly a matter of plotting out the dots that represent the vertices, then connecting the dots with edges, but anything more than the simplest of meshes have many vertices and edges and require a long time to draw. Drawing only a portion of the mesh can save some time, but then it is hard to see how that small area is related to the rest of the mesh. Errors can easily occur in the drawing process. Vertices all look the same, and two vertices that end up near each other on paper can easily be mixed up. Properly labeling everything would prevent this, but someone trying to speed up the process would likely skip that. Eventually the wrong two dots get connected, and when the drawer finally notices something is wrong, it may take a long time to find the error. Errors are especially bad when tracing through algorithms. Unfamiliarity with the data structure makes it hard to tell if the algorithm is written wrong, the mesh is drawn wrong, or an error was made in tracing the algorithm. The boredom and frustration that result from all these difficulties could cause someone to simply give up trying to learn the structures. Drawings on paper are also limited by being static; once drawn, the only way to do something like rotating the mesh to see the other side is to redraw the picture. Visualizing these data structures on a computer could avoid some of the problems with drawing by hand, but it too has a major problem as there is currently no software aimed at visualizing mesh data structures. There are certainly many systems that make use of meshes, ranging from simple assignments for graphics classes to more complex systems such as OpenMesh and CGAL. However, even though the systems may use winged-edge or half-edge structures internally, the user never sees any hint of that underlying data structure. Even a system such as OpenMesh is focused on using the data structure to build meshes rather than on teaching users how the data structure works. Lacking suitable software, the user is stuck drawing pictures by hand.

This work addresses the need for a suitable method of visualization that can help users understand the data structures and basic algorithms by providing a software system that handles all the tedious details of visualizing mesh data structures and basic operations, allowing the user to focus on learning to use them. The system allows the user to interact with a mesh while exposing the internal data structure storing that mesh through a set of tables that list the relationships each element has with adjacent elements. The system

also provides the ability to trace through simple mesh algorithms. This algorithm visualization component keeps track of and displays the state of an algorithm, including the values of variables and current step in the algorithm, while using labeling and highlighting in the mesh image and data structure tables to further expose the algorithm's interactions with the data structure.

The remainder of this paper will discuss the system in more detail and briefly explore possible uses of the system. Section 2 briefly reviews the winged-edge and half-edge data structures. Section 3 gives an overview of the system. The details of how the system exposes the data structure of a mesh are described in Section 4. Section 5 describes how the algorithm visualization system works.

2 Mesh Data Structures

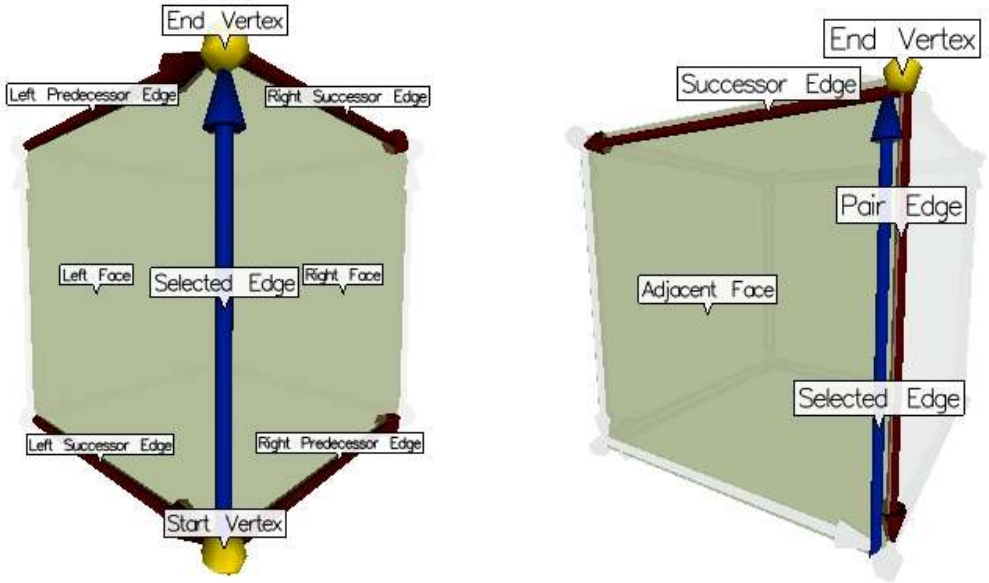
A mesh is composed of vertices, edges, and faces. Vertices are points in space. Each edge is a one-dimensional line connecting two vertices. Faces are two-dimensional surfaces bounded by three or more edges. There are many data structures for storing a mesh, but the more efficient ones are edge-based data structures such as the winged-edge and half-edge data structures. These data structures are edge-based because nearly all of the information about the arrangement of the vertices, edges, and faces is stored with the edges. In both data structures, the vertices and faces store a reference to an *incident edge*, which is an edge next to the vertex or face. Additionally, each vertex stores its position in space. The rest of the information about how parts of the mesh are related is stored with the edges. What exactly is stored depends on the data structure.

2.1 The Winged-Edge Structure

In the winged-edge data structure, the edges store references to eight adjacent parts of the mesh. There are two references to vertices, one on each end of the edge, called the *start vertex* and the *end vertex*. There are also two references to faces, one on each side of the edge. Looking at an edge with the start vertex on the bottom and the end vertex on the top, as in Figure 1(a), the face on the left is called the *left face*, while the face on the right is called the *right face*. Lastly, there are four references to adjacent edges which are determined by traversing edges of the left and right faces in the clockwise direction when looking at the outside of the mesh. When traversing the edges of the left face in Figure 1 (a) in the clockwise direction, the edge preceding the edge labeled *selected edge* is called the *left predecessor edge*, while the edge after the selected edge is called the *left successor edge*. Similarly, when traversing the edges of the right face, the edge preceding the selected edge is called the *right predecessor* while the edge after the selected edge is the *right successor*. These four adjacent edges form the "wings" of the selected edge, which is why it is called the "winged-edge" data structure.

2.2 The Half-Edge Structure

The half-edge data structure differs from the winged-edge structure by dividing each edge of the mesh into a pair of *half-edges*. Each half-edge stores four references as shown in Figure 1(b). There is one reference to the other half-edge in the pair, called the *pair edge*. Another reference stores a vertex, called the *end vertex*. The end vertices of the two half-edges in a pair are at the opposite ends of the whole edge. There is a reference to the face, called the *adjacent face*. Lastly, there is a reference to one other edge, called the *successor edge*. This is the edge after the current edge when traversing the edges of the adjacent face in the counter-clockwise direction.



(a) Winged-edge structure *(b) Half-edge structure*

Figure 1: The edges in both data structures store references to adjacent elements.

2.3 Mesh Elements

In this paper, the faces, edges, and vertices that make up the mesh will be generically referred to as *mesh elements*, or simply *elements*. In the system, every mesh element has a few common attributes. First, each element has an identification number that distinguishes that element from other elements of the same type. In a mesh with *n* faces and *m* edges, for example, the faces are numbered 0 through *n-1* and the edges are numbered 0 through *m-1*. Second, each element may have zero or more labels associated with it. These labels could be descriptions, variable names, or something else. If there is more than one label for an element, those labels will all be displayed as a list of labels. Lastly, elements can be highlighted in various colors. Highlighting is used to distinguish some element or set of elements from all the other elements, such when as showing which elements are involved in some operation in an algorithm.

3 System Overview

When the system is running, a number of windows may appear as shown in Figure 2. At the upper left corner of the screen is the **System Control** window. Below that is the **Mesh Control** window. To the right of the control windows is the 3D view, and to the right of that is the algorithm view. At the bottom are the three windows of the table view. Additionally, and not visible in Figure 2, there is a **Load File** dialog, a **Save File** dialog, and a **Colors** dialog. The **System Control** window, which remains visible as long as the system is running, contains a `Load Mesh` button, which brings up the **Load File** dialog that lets the user load meshes for viewing, and an `Adjust Colors` button, which brings up the **Colors** dialog for adjusting some of the colors used by the system. The remaining windows are associated with a loaded mesh and may be duplicated if more than one mesh is loaded. The windows for a particular mesh can be identified by the mesh file name in the window title. The 3D view displays an interactive image of the mesh. The three windows of the table view display tables describing the data structure that stores the mesh shown in the 3D view. The **Mesh Control** window includes controls for adjusting the image in the 3D view and for controlling the algorithm visualization system. At the bottom of the **Mesh Control** window is a `Save Mesh` button that brings up the **Save File** dialog, which lets the user save a mesh. The algorithm view, only visible when tracing an algorithm, displays an algorithm's source code and the values of variables in the algorithm.

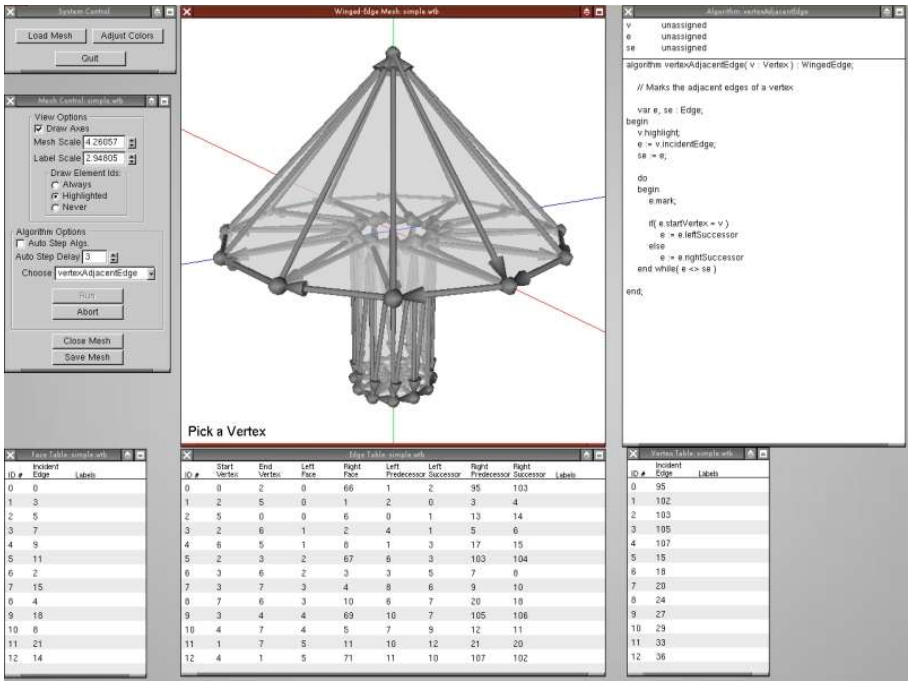


Figure 2: The default layout of the windows. The large center window is the 3D view. Below that are the three windows of the table view. To the left are two control windows. The algorithm view appears to the right of the 3D view.

The system can be divided into two major components. The first component deals with simply visualizing the data structure using the 3D and table views. The second component is the algorithm visualization system, which uses the 3D and table views and adds the algorithm view to provide a complete picture of how an algorithm works with the data structure.

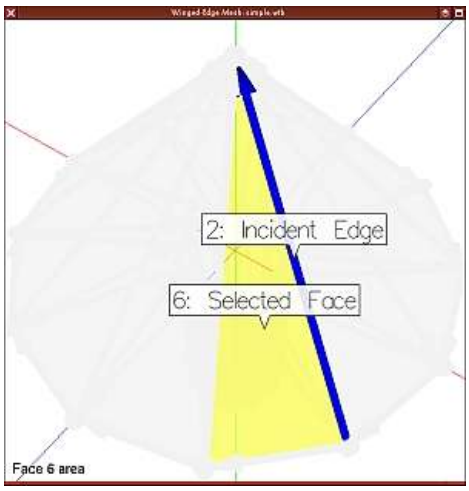
4 Visualizing the Data Structure

The 3D view and the table view work together to fully expose the data structure that stores a mesh by displaying the same structure in different ways. The 3D view displays elements of the data structure as an image of the mesh, while the table view displays the references that make up the data structure as a set of tables.

4.1 The 3D View

The 3D view displays the data structure as an interactive wireframe rendering of the mesh. The image can be rotated by dragging with the left mouse button, and zoomed by dragging up and down with the right mouse button. Mesh elements can be selected by left clicking. The most notable feature of the 3D view is the edges that make up the wireframe. They are drawn as arrows that point to the end vertex of the edge, which provides some hint of the underlying data structure. Vertices are simply drawn as spheres. Faces are drawn mostly transparent so the entire mesh structure is visible. Element identification numbers and labels may be drawn in a box just above the midpoint of an element, as shown in Figure 3(a). Highlighting, also shown in Figure 3(a), is displayed by drawing the element in its highlight color. Figure 3(a) also shows how unhighlighted elements are faded so they are less distracting. Messages, such as descriptions of what is being displayed or prompts for input, are displayed along the bottom of the window.

Some adjustments to how the 3D view is drawn can be made through the controls in the `View Options` section of the **Mesh Control** window, shown in Figure 3(b). The `Draw Axes` option simply turns the axes on and off. The `Mesh Scale` option scales the structure, which means making the edges and vertices thicker or thinner, while the `Label Scale` option adjusts the size of the labels, making them bigger or smaller. These two options are meant to deal with the varying density of mesh structures. On a dense mesh with many elements in a small area, for example, labels drawn at the default size end up overlapping each other. Shrinking the labels reduces the overlapping, making them easier to read. The last option, `Draw Element Ids`, decides when the identification numbers of elements are drawn in the 3D view. The default option of `Highlighted` means the numbers are only displayed for highlighted elements, as in Figure 3(a). The `Always` option means the numbers will be displayed on all elements. The `Never` option means the numbers are never displayed.



(a) Other features of the 3D view

(b) The Mesh Control window

Figure 3: (a) In the 3D view, labels are drawn just above the midpoint of an element. Messages are displayed along the bottom of the window. (b) The Mesh Control window provides some options for changing how the 3D view is drawn.

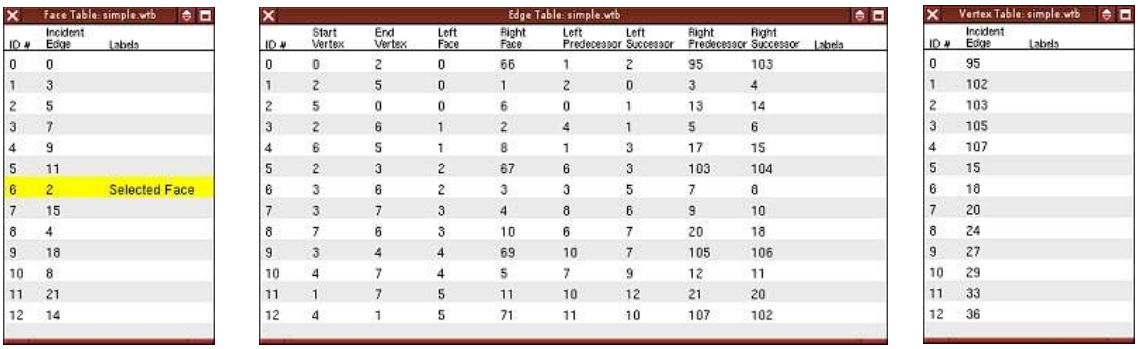
The mesh shown in Figure 2 is stored and displayed as a winged-edge data structure. Meshes stored using the half-edge structure are drawn slightly different. The key point with a half-edge structure is that each physical edge is stored as a pair of half-edges. Figure 1(b) shows how this is reflected in the 3D view by drawing pairs of edges. Each edge still points to its end vertex. The label boxes for half-edges are shifted from the midpoint towards the end vertex so the labels for both edges of a pair are visible.

4.2 The Table View

The table view is a set of three tables that list all the relationships that make up the data structure that stores the mesh displayed in the 3D view. This is the same data structure that is used internally by the system. There is one table for each type of element. The leftmost table (Figure 4(a)) is the face table, listing all the faces of the mesh. The center table (Figure 4(b)) is the edge table, and the rightmost table (Figure 4(c)) is the vertex table. Relationships are described by listing the identification numbers of elements. The first column in every table is the identification number of an element, while the last column lists any labels associated with that element. The middle columns list the references each element has to other elements. In the face and vertex tables, the single middle column lists the identification number of the incident edge. The middle columns of the edge table depend on the data structure type. For a winged-edge structure, the columns list the identification numbers of the start vertex, end vertex, left face, right face, left predecessor edge, left successor edge, right predecessor edge, and right successor edge. For a half-edge structure, the columns list the end vertex, adjacent face, pair edge, and successor edge. The columns are labeled at the top of the table. Figure 4(b), for example, is the edge table for a winged-edge structure. The entry for Edge 5 shows that the start vertex for Edge 5 is Vertex 2, the end vertex is Vertex 3, the left face is Face 2, the right face is Face 67, the left predecessor edge is Edge 6, the left successor edge is

Edge 3, the right predecessor edge is Edge 103, the right successor edge is Edge 104, and there are no labels for Edge 5.

In the tables, highlighted elements are drawn by shading the background of an element's row in the table, as in Figure 4(a). Usually a table will be too long to fit in its window, so a table can be scrolled up and down by dragging up and down with the left mouse button. When scrolling the column labels remain stationary. An element can be selected by left clicking on its row in a table.



Face Table: simple.wtb

ID #	Incident Edge	Labels
0	0	
1	3	
2	5	
3	7	
4	9	
5	11	
6	2	Selected Face
7	15	
8	4	
9	18	
10	8	
11	21	
12	14	

Edge Table: simple.wtb

ID #	Start Vertex	End Vertex	Left Face	Right Face	Left Predecessor	Left Successor	Right Predecessor	Right Successor	Labels
0	0	2	0	66	1	2	95	103	
1	2	5	0	1	2	0	3	4	
2	5	0	0	6	0	1	13	14	
3	2	6	1	2	4	1	5	6	
4	6	5	1	8	1	3	17	15	
5	2	3	2	67	6	3	103	104	
6	3	6	2	3	3	5	7	8	
7	3	7	3	4	8	6	9	10	
8	7	6	3	10	6	7	20	18	
9	3	4	4	69	10	7	105	106	
10	4	7	4	5	7	9	12	11	
11	1	7	5	11	10	12	21	20	
12	4	1	5	71	11	10	107	102	

Vertex Table: simple.wtb

ID #	Incident Edge	Labels
0	95	
1	102	
2	103	
3	105	
4	107	
5	15	
6	18	
7	20	
8	24	
9	27	
10	29	
11	33	
12	36	

(a) Face Table *(b) Edge Table* *(c) Vertex Table*

Figure 4: The table view describes the relationships between mesh elements that make up the data structure.

4.3 The 3D and Table Views Together

Since the 3D and table views work together to expose the data structure, they are designed so that anything that is done in one view can be done in the other. Picking an element of the mesh, for example, can be done through either view by clicking on the element in the 3D view or by clicking on the element's entry in the table view. When not tracing an algorithm, as when a mesh is first loaded, selecting an element of the mesh through either view will highlight that element and the elements related to it in the data structure. The highlighting is done using colors and labels. Figure 3(a), for example, shows how a face and its incident edge are shown in the 3D view. This feature lets users explore which adjacent elements are significant in the data structure as well as how those adjacent elements are related. Only one element can be picked at a time. When a new element is picked any existing highlighting or labels are removed. Left clicking on the background in the 3D view clears any selection.

4.4 Loading and Storing Meshes

Meshes to be displayed are stored in files and loaded by the user. Three different file formats can be used. The first two are table formats, one for winged-edge structures and one for half-edge structures, while the third is the PLY format. Filenames for the winged-edge table files end in *.wtb*, while names for half-edge table files end in *.htb*, and names

for PLY files end in *.ply*. The table file formats store geometry information followed by tables describing the data structure as in the table view, while the PLY format typically stores geometry information followed by lists of vertices for each face. The table formats are useful for users implementing simple programs since the data structure can simply be read in, which is much easier than building the structure from the lists of vertices for each face provided by the PLY format. A disadvantage of the table format is that writing the files by hand tends to be extremely tedious. To help with this, the program provides a limited ability to convert PLY files to table files, which is done by first loading the PLY file and then saving the mesh to the table format. Table files can be converted to PLY files using a similar method. The PLY format is widely used for storing meshes, and so there are many models available in this format. Not all models are suitable for this system, though. The model should not be too large as drawing all the elements of a large mesh is extremely slow on currently available hardware, and it cannot have any holes, loose edges, or rogue vertices, which cannot be represented by the data structures without some modifications to the structure.

The **Load File** and **Save File** dialogs are used to load and save mesh files. The **Load File** dialog appears when the program first starts and can be accessed later through via the `Load Mesh` button in the **System Control** window. The dialog includes a text field for the file name and lets the user choose the file type and data structure type. The default choice for the file type is `Automatic`, which means that the system will try to figure out the file type and structure type from the filename extension. With winged-edge and half-edge table files, there is no choice for the structure type; a winged-edge file must be loaded as a winged-edge structure, and a half-edge file as a half-edge structure. With a PLY file, however, the structure type is up to the user. The **Save File** dialog lets users save a mesh to a different file, possibly in a different format. The dialog is accessed through the `Save Mesh` button at the bottom of a mesh's **Mesh Control** window. The user provides a file name and picks the file format.

5 Visualizing Algorithms

The algorithm visualization component of the system uses the algorithm view to display the state of an algorithm while tracing through it. A number of simple algorithms are included with the system, and in a future version of the system it should be possible to let users write and load their own algorithms. An algorithm is visualized by displaying in the 3D and table views each of the operations that make up the algorithm in sequence.

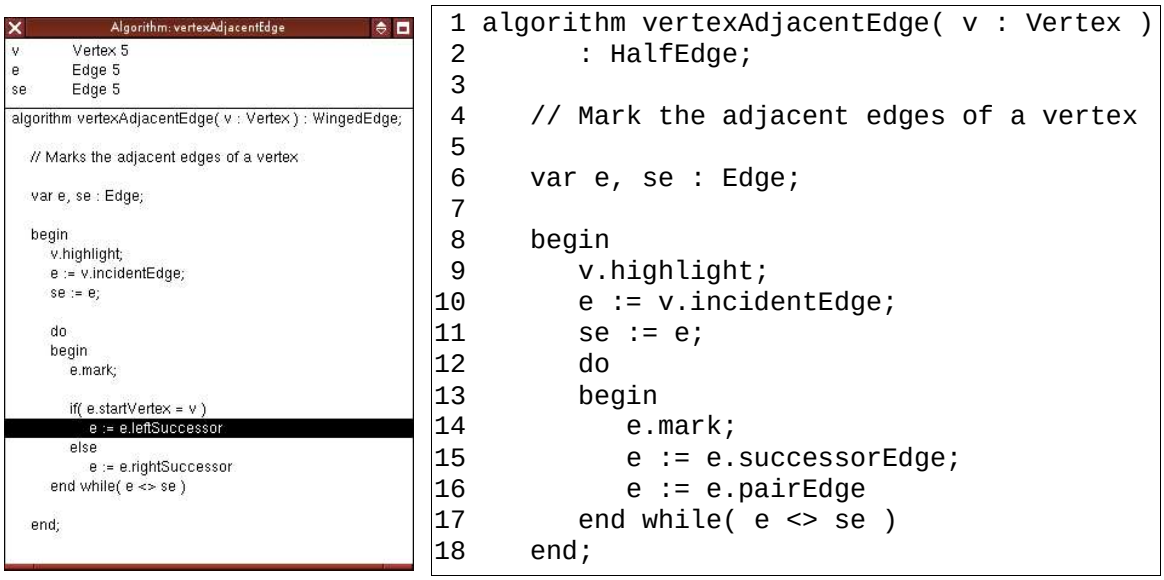
5.1 The Algorithm View

The algorithm view, shown in Figure 5(a), displays the state of the algorithm, including the values of variables and the current step in the algorithm. The upper portion of the algorithm view displays a list of the variables in the algorithm with their current values. The values are listed by element type and number, such as *Edge 13* or *Vertex 3*. Below

the list of variables is the algorithm's source code. When the system is tracing through an algorithm, the source code line of the current operation is highlighted. If the code is too long to fit in the window, it scrolls up and down as needed to keep the highlighted line visible. The source code can also be scrolled manually by dragging up and down with the left mouse button.

5.2 Algorithms

The system includes a number of simple algorithms for each data structure type. Examples include finding all the vertices adjacent to a face and finding all the edges incident to a vertex. The algorithms are kept simple because the goal is to show how algorithms work with the data structure, not how some complex algorithm like mesh subdivision works. These simple algorithms are instead ones that are used in implementing more complex algorithms. Finding all the vertices adjacent to a face, for instance, is needed when rendering the faces of a mesh. Finding all the edges incident to a vertex is useful in mesh subdivision algorithms. All the algorithms available for a mesh are listed in a combobox in the `Algorithms` section of the **Mesh Control** window.



(a) The Algorithm view *(b) Example code*

Figure 5: (a) The algorithm view displays the values of variables in the algorithm and the algorithm source code with the current line highlighted. The algorithm in this image finds the adjacent edges of a vertex in a winged-edge structure. (b) The same algorithm for a half-edge structure. Algorithms are defined using a pseudo-code-like language.

The included algorithms are coded using a simple pseudo-code-like language and stored in a text file that is loaded when the system starts. By defining the algorithms in a separate file outside of the system's source code, it is easy to update or add algorithms to the system. When the system starts, the algorithm source code is read in, parsed, and compiled into a linear representation that is similar to assembly code. This linear form is

used internally to execute the algorithm while the source code is displayed in the algorithm view. Eventually this mechanism could be extended to allow users to write and load their own algorithms. Using a pseudo-code-like language avoids any assumptions about what languages the user knows and avoids any details of a particular implementation of the data structure. However, despite looking like pseudo-code, it is a well-defined language. Figure 5(b) shows an example of an algorithm written in this language that finds the edges adjacent to a vertex in a half-edge structure. The algorithm starts by declaring the name, *vertexAdjacentEdge*, followed by the parameters. Parameters and variables can have one of three data types: *Face*, *Edge*, and *Vertex*. The data types correspond to the types of mesh elements. In Figure 5(b), there is a single parameter named *v* of type *Vertex*. Next is the mesh structure type, in this case *HalfEdge*, indicating that this algorithm is for a half-edge data structure. Algorithms for a winged-edge data structure use the *WingedEdge* keyword instead. Comments, such as on line 4 in Figure 5(b), are indicated by a double slash and continue to the end of the line. The variables in the algorithm are declared as on line 6, then the body of the algorithm is enclosed by a *begin* and *end* pair. The two basic control structures are if-then-else statements and do-while loops. The *vertexAdjacentEdge* algorithm here uses a do-while loop. Assignments are done using the *:=* operator, as on lines 10, 11, 15, and 16. The data structure is accessed by putting the name of the reference after the variable name, as in *v.incidentEdge*. Two variables can be compared using == (equal) and <> (not equal) to see if they refer to the same element. The *highlight* operation is used to highlight input elements, and the *mark* operation is used to indicate the output of the algorithm.

5.3 Displaying Operations

Each step in an algorithm consists of one or more individual operations. The line *e := v.incidentEdge*, for example, consists of a lookup in the data structure to get the incident edge of *v* and an assignment that sets *e* to refer to that retrieved edge. The three basic operations are *data structure lookups*, *assignments*, and *comparisons*. A fourth operation, *marking*, is used to simulate output from an algorithm. Each operation is displayed in the 3D and table views using highlighting and labeling to identify the elements involved in the operation. A different color is used for highlighting each type of operation. A line of text along the bottom of the 3D view describes the operation being displayed.

The *data structure lookup* operation means referring to the data structure to retrieve an element adjacent to a base element. In *v.incidentEdge*, *v* is the base element, while *incidentEdge* is the name of the adjacent element being retrieved. As shown in Figure 6, the operation is displayed by highlighting both elements in some color, and additionally highlighting the column of the adjacent element in the base element's entry in the table view in a different color. The highlights disappear when the next operation is displayed.

An *assignment* means setting some variable in the algorithm to refer to some mesh element. In *e := v.incidentEdge*, for instance, the variable *e* is being set to refer to the incident edge of *v*. Assignments are displayed by highlighting the element and adding the variable's name to the element's label, as in Figure 7. When a variable is reassigned (set

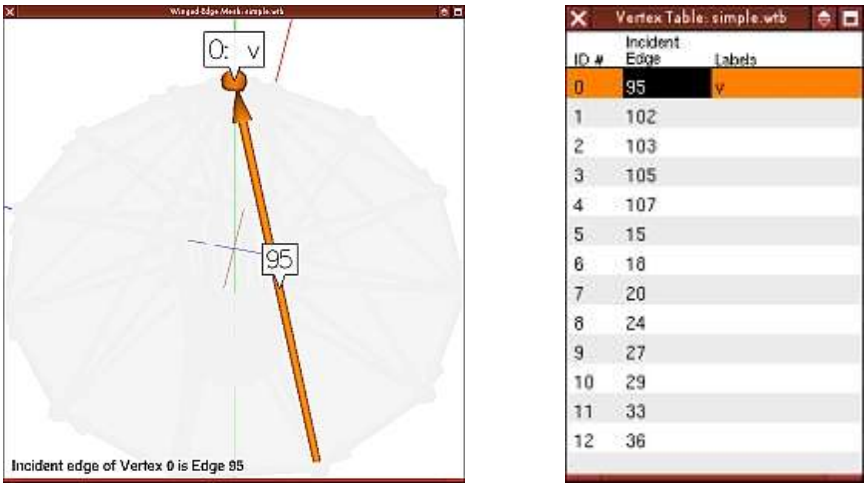


Figure 6: Data structure lookups are displayed by highlighting both the base and the adjacent element, and by highlighting the column in the table view.

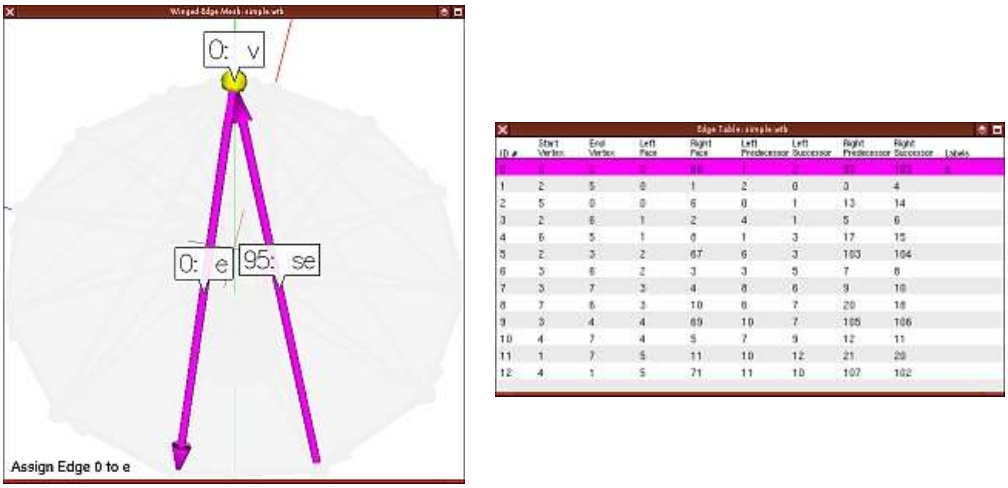
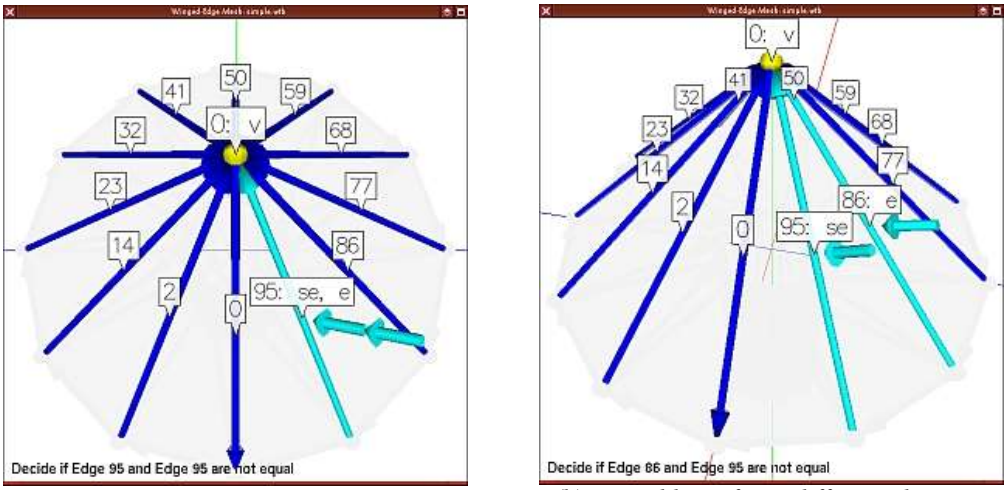


Figure 7: Assignments are displayed by highlighting the element the variable refers to and adding the variable's name to the element's label.

to refer to some other element), the variable's name is removed from the old element's label and added to the new element's label. If multiple variables refer to the same element, all the variables' names will be listed in the element's label. An element will remain highlighted in the assignment color as long as some variable refers to it.

Comparisons are made between two variables of the same data type to decide if they refer to the same element. In the *vertexAdjacentEdge* algorithm, for example, *e* <> *se* decides if *e* and *se* refer to the same edge to control a loop. To display the operation, the element that each variable refers to is highlighted, and in the 3D view an arrow points at each element. The arrows are intended to avoid confusion if both variables refer to the same element since only one element would be highlighted, and the user might be wondering where the second element is. If both variables refer to the same element, only that one element will be highlighted, but there will be two arrows pointing at it, as in Figure 8(a). Alternately, if the variables refer to different elements, two elements will be highlighted

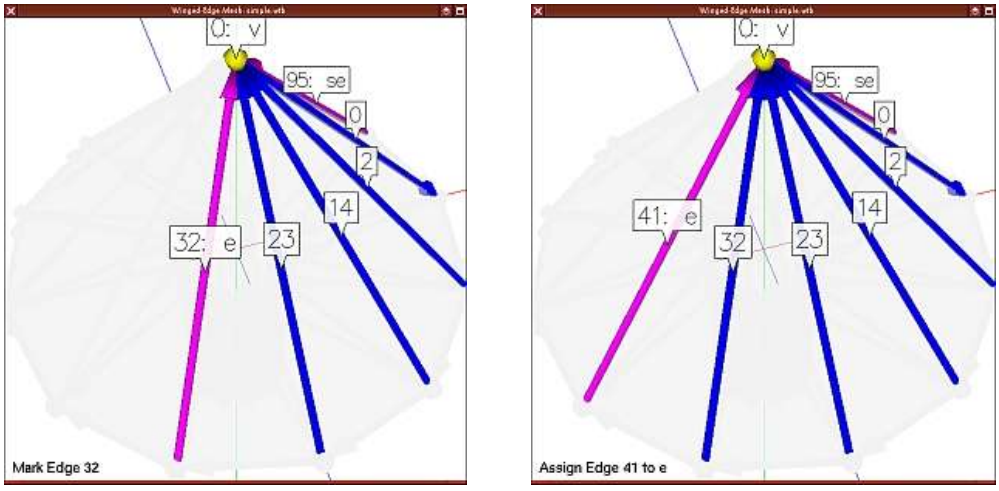


(a) Variables refer to the same element *(b) Variables refer to different elements*

Figure 8: For comparisons, the element each variable refers to is highlighted. In the 3D view, an arrow points at each element.

with one arrow pointing at each, as in Figure 8(b). The highlights and arrows for this operation disappear when the next operation is displayed.

Marking is used to simulate output from an algorithm. Algorithms usually produce some form of output such as a list of edges, but how that output is returned depends on the implementation of the algorithm. In real code, arrays, function calls, or pointers might be used, but here those details are avoided by simply highlighting output elements in a particular color, which is referred to as *marking*. In an algorithm's source code, marking appears as something like *e.mark*, where *e* is a variable referring to the element that should be marked. Marking highlights have a lower priority than the highlighting used for other operations, so the marking may not be immediately visible, as shown in Figure 9 (a), where Edge 32 has just been marked. As Figure 9(b) shows, though, once an element



(a) Marking hidden by other highlighting *(b) Marking becomes visible*

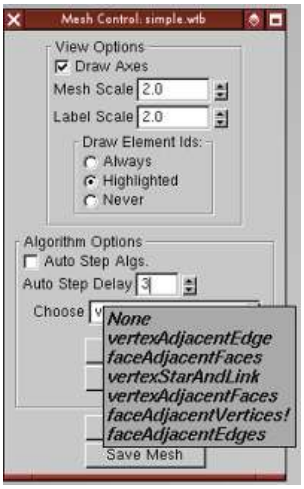
Figure 9: (a) Marking highlights the output of an algorithm. Other highlights have a higher priority, so marking may not be immediately visible, as is the case with Edge 77. (b) Once the other highlights are gone, the marking becomes visible.

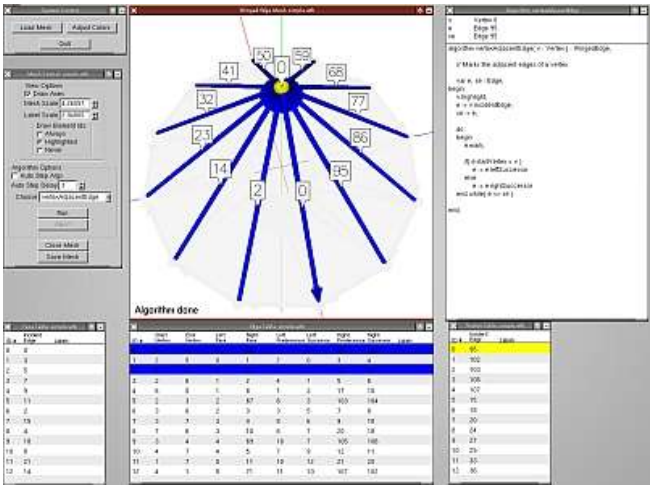
has no variables referring to it and it is not involved in some other operation, the marking will be visible. Once an element is marked, it remains marked until after the algorithm ends when the user starts some other visualization.

5.4 Animating an Algorithm

An algorithm is run by selecting it from the list of algorithms, clicking the `Run` button, selecting the input, then stepping through operations until the algorithm terminates. The list of available algorithms is in a combobox in the `Algorithm` section of the **Mesh Control** window (Figure 10(a)). When the user selects an algorithm and clicks the `Run` button, located just below the algorithm list, the algorithm view appears and the user is prompted to pick the input elements for the algorithm, as in Figure 2. Input elements can be picked through the 3D or table view, and only elements of the needed type can be picked. If, for instance, the algorithm is asking for a face, only faces are pickable. Once the input has been selected, the system starts stepping through the algorithm one operation at a time. Figures 6, 7, 8, and 9 show how operations are displayed at various steps in the *vertexAdjacentEdge* algorithm, which marks all edges incident to a vertex. The whole process can be aborted at any time by clicking the `Abort` button in the **Mesh Control** window. Aborting clears any highlighting and labels from the mesh and the algorithm view disappears. If the algorithm is allowed to run to completion, a message along the bottom of the 3D view informs the user that the algorithm is done, any labels on the mesh are removed, and all highlighting except for marking is removed, as in Figure 10(b). The algorithm view remains visible and output elements remain marked until the user picks an element or starts a new algorithm.

The system has two modes for stepping through operations: *manual* and *automatic*. The user picks the mode through the `Auto Step Algs` option in the **Mesh Control** window. In *manual* mode, the user presses the spacebar to advance to the next operation.





(a) Selecting an algorithm *(b) The vertexAdjacentEdge algorithm is done*

Figure 10: (a) Algorithms are selected through the combobox in the Mesh Control window (b) After the algorithm is done, the algorithm view and output remain visible.

In *automatic* mode, the system runs on a timer, advancing every few seconds. The delay between steps in automatic mode is adjustable through the `Auto Step Delay` option in the **Mesh Control** window.

6 Conclusion

Visualization is a useful tool for learning to use the winged-edge and half-edge data structures, but methods like drawing pictures on paper are distracting or even frustrating. This work provides a system that handles the tedious and error-prone parts of visualizing the structures and tracing basic algorithms, letting user focus on learning the structure. This system is intended for anyone with a need to learn the concepts and basic algorithms of the winged-edge or half-edge data structures. Students in graphics and geometric modeling classes, for example, might use the system as a reference while working on mesh-related assignments, while their instructors might use it to demonstrate how the data structure stores larger meshes than can be drawn on the board and how basic algorithms work with the data structure. Since the winged-edge and half-edge data structures are being used in systems such as OpenMesh, professionals might also use this system to gain a basic understanding of the data structures so they can make better use of other mesh-related systems. The system described in this paper is intended to make computer graphics programming more accessible by removing some of the difficulty in learning the winged-edge and half-edge data structures.

Acknowledgments

This project is supported by the National Science Foundation under grant DUE-0127401 and by an IBM Eclipse Innovation Award 2003.

References

[1] Bruce G. Baumgart, *Winged edge polyhedron representation*, Stanford University, Stanford, CA, 1972.

[2] Kevin Weiler, *Edge-based Data Structures for Solid Modeling in Curved-Surface Environments*, IEEE Computer Graphics and Applications, January, 1985.

[3] Computer Graphics Group, RWTH Aachen, *OpenMesh*, Retrieved March 2, 2005, from http://www.openmesh.org/.

[4] *The CGAL Home Page*, Retrieved March 4, 2005, from http://www.cgal.org/home.html.