Visualizing and Animating the Winged-Edge Data Structure [⋆]

Bryan Neperud John Lowther Ching-Kuang Shene [*]

Department of Computer Science, Michigan Technological University, Houghton, MI 49931, USA

Abstract

The winged- and half- edge data structures are commonly used representations for polyhedron models. Due to the complexity, students in an introductory to computer graphics course usually have difficulty in handling these data structures and developing applications. This paper describes the authors' effort in the development of a visualization and animation tool for teaching and learning these data structures. This tool also includes a simple pseudo code-like language for algorithm design. Instructors may employ this tool for presentation and demonstration purposes. Students may use the simple language to develop and experiment with new algorithms before their actual implementation. The visualization and animation system may be used to explore and understand the relationship among mesh elements and algorithm execution.

Key words: Winged-Edge Data Structure, Half-Edge Data Structure, Algorithm Visualization and Animation, Computer Graphics Education
PACS:

1. Motivation

No matter how sophisticated and capable a rendering system is, a scene must include geometric objects in order to generate images. As a result, the way of representing geometric objects for efficient rendering and manipulation is an extremely important topic in computer graphics. While there are many different polyhedron model representations, the winged- and half- edge data structures are the most popular and commonly used ones. The winged-edge data structure was proposed by Baumgart more than 30 years ago [1,2], originally for computer vision. It has been discussed and analyzed extensively [12,26,28] and has become a popular topic in computer graphics [9,10,19,24,25], especially for modeling practice [3,8,11,18]. Its variation, the half-edge data structure, is frequently used in edge-based mesh representation, design and modeling. Since winged- and half- edge data structures provide a more compact and efficient representation than the conventional data structures being taught in a typical computer graphics course, and since modeling is an important skill, it is worth to present this topic to some depth in computer graphics and related (*e.g.*, geometric modeling or computer-aided design) courses.

A third year elective course *Elementary Geometric Objects and Processing* [21] was created to introduce design and modeling skills to computer science students nearly ten years ago. The authors' experience in teaching this course showed that it is usually difficult for students to design algorithms for the winged-edge data structure due to the complexity of the edge table (Section 7). Many students used the

[⋆] A preliminary version of this paper by the first author was published in [20] and received a Best Graduate Student Paper for MICS 2005 award.
[*] Corresponding author
Email address: shene@mtu.edu (Ching-Kuang Shene).
URL: http://www.cs.mtu.edu/∼shene (Ching-Kuang Shene).

winged-edge data structure tables in a straightforward way, defeating the original purpose completely. Presenting a data structure that records complex adjacency relationships is also a challenging task to instructors. Experience also showed that class discussion alone may not be the best way, and that a visualization and animation tool may be needed to reveal the nature of the data structure. While there are programming tools supporting the winged- and half- edge data structures (*e.g.*, CGAL [4] and OpenMesh [5]), the authors could not find reasonable pedagogical visualization and animation tools for the data structures *and* their related algorithms. Although algorithm animation systems are available, they are too general to be used for 3-dimensional geometric visualization and animation, and the use of C/C++ and Java and additional instrumentation may be too tedious for students to use. Consequently, after teaching the topic for many years, the authors decided to develop a visualization and animation tool for the winged- and half- edge data structures.

The major goal is to develop a tool that can be used to visualize the tabular and mesh structure clearly, and permits instructors and students to design and animate various basic winged- and half-edge related algorithms. This paper presents the authors' effort in addressing the above mentioned teaching and learning challenges. To simplify presentation, only the winged-edge data structure will be discussed even though this tool supports the half-edge data structure equally well. In the following, Section 2 reviews the winged-edge data structure; Section 3 emphasizes the importance and efficiency of the data structure; Section 4 briefly discusses this system; Section 5 presents the visualization component; Section 6 has the details of a simple language for algorithm design, visualization and animation components, and a complete example; Section 7 presents the authors' long term experience in teaching this topic and their findings; and, finally, Section 8 has the conclusions.

2. The Winged-Edge Data Structure

This paper is only concerned with orientable 2-manifold meshes without boundary. Thus, each edge has exactly two incident faces, and faces are oriented in a *compatible* way so that each edge receives opposite directions from its incident faces. Vertices of a face are conceptually oriented clockwise. Some insist that this orientation has to be counter-clockwise in order to generate correct normal vectors for rendering. This is incorrect since the winged-edge data structure does not store vertices in any particular order, and the order of vertices of each face has to be generated with a traversal algorithm which can be clockwise or counter-clockwise.

In Fig. 1, edge a has incident faces 1 and 2, and the traversal of each incident face under the predefined orientation induces a predecessor edge and a successor edge. The predecessor and successor edges of edge a with respect to face 1 are edges b and d, respectively, and the predecessor and successor edges of edge a with respect to face 2 are edges e and c, respectively. Since each face induces a different direction to the common edge, a direction must be chosen in order to specify the "left" face and the "right" face. For example, in Fig. 1, if the chosen direction is from vertex X to vertex Y, the left face is face 1 and the right face is face 2. Otherwise, the left and right faces are 2 and 1, respectively.

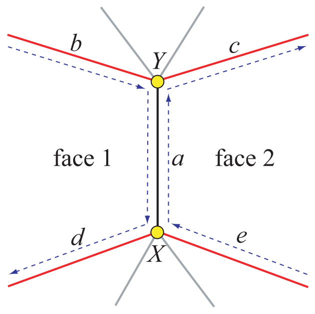


Fig. 1. The Winged-Edge Data Structure

Once the orientation of an edge is chosen, one can assemble nine pieces of information from an edge: the edge name, the start and end vertices, the left and right faces (*i.e.*, the wings), the predecessor and successor edges when traversing the left face, and the predecessor and successor when traversing the right face. In Fig. 1, if the start and end vertices of edge a are X and Y, respectively, the predecessor and successor edges when traversing the left face are b and d, respectively; and the predecessor and successor edges when traversing the right face are e and c, respectively (Table 1).

The winged-edge data structure of a mesh consists of *three* tables: vertex table, edge table, and face table. There is one entry for each edge in an edge table, and each edge entry consists of the nine pieces of information in Table 1; however, if the edges of a mesh are numbered, one may use the table index

Table 1
Table Entry for Edge a

Edge	Vertex		Face		Left Traversal		Right Traversal	
Name	Start	End	Left	Right	Pred.	Succ.	Pred.	Succ.
a	X	Y	1	2	b	d	e	c

for the edge name. Each entry in the vertex table contains the coordinates of a vertex and a pointer to an incident edge in the edge table. Each entry in the face table only contains a pointer to an incident edge in the edge table. For example, the vertex table entry of vertex X in Fig. 1 may have a pointer to edge a, d, e or any other incident edge. Similarly, the incident edge of face 1 may be edge b, a, d or any edge of face 1. Fig. 2 is a tetrahedron of four vertices, six edges, and four faces. A possible winged-edge data structure of this tetrahedron is shown in Table 2.

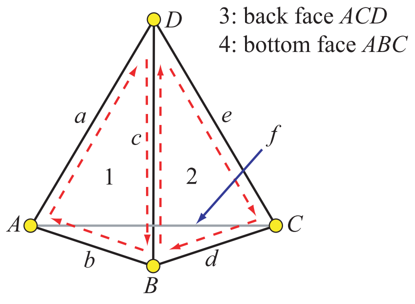


Fig. 2. A Tetrahedron

Table 2
Tables of a Tetrahedron in Fig. 2

Edge	Vertex		Face		Left		Right	
Name	*S.*	*E.*	*L.*	*R.*	*P.*	*S.*	*P.*	*S.*
a	A	D	3	1	e	f	b	c
b	A	B	1	4	c	a	f	d
c	B	D	1	2	a	b	d	e
d	B	C	2	4	e	c	b	f
e	C	D	2	3	c	d	f	a
f	A	C	4	3	d	b	a	e

(a) Edge Table

Vertex	*Edge*
A	a
B	b
C	d
D	e

(b) Vertex Table

Face	*Edge*
1	a
2	c
3	a
4	b

(c) Face Table

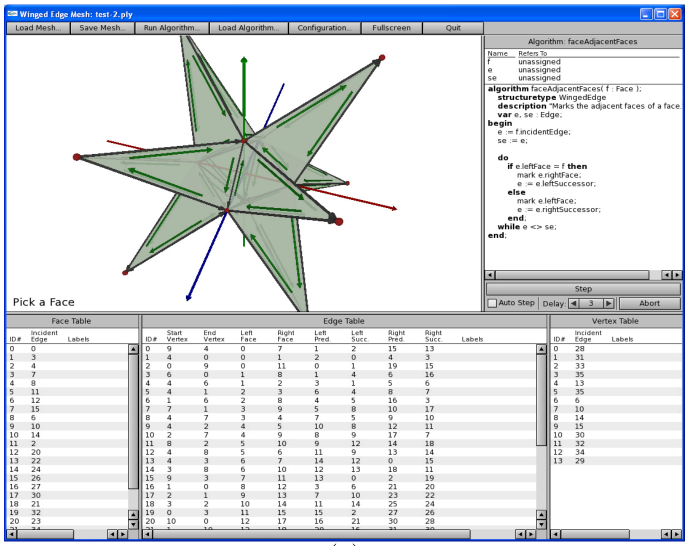
3. Why the Winged-Edge Data Structure

The advantages of the winged-edge data structure may not be initially obvious due to its complex edge table structure. However, the complexity of the edge table structure does help answer many topological inquiries easily and efficiently, such as finding all incident edges (or faces) of a vertex. These topological inquiries can be very time-consuming to perform for a conventional data structure. A conventional data structure usually has (1) a vertex table that stores the coordinates of each vertex, (2) an edge table which stores the two incident vertices of each edge, and (3) each entry of a face table stores the vertices (or edges) of that face with a linked-list. To find all incident edges of a vertex, one must scan every edge table entry and report those that are incident to the given vertex. This requires $2E$ comparisons, where E is the number of edges. Similarly, one must scan every linked list in the face table to find the incident faces of a given vertex or edge. Since each edge appears exactly twice in the face table linked lists, one must follow $2E$ links to complete the scan.

Many topological inquires can easily be answered with the winged-edge data structure. Since each entry in an edge table stores all incident elements of that edge, answering topological inquires would only require a local scan around the given element. For example, to find all incident edges of a vertex, one may simply retrieve an incident edge from the vertex table, and follow the predecessor edge or successor edge to the next edge. Therefore, if an application requires many topological inquiries, the winged-edge data structure is more efficient than a conventional one. Additionally, the winged-edge data structure may be used efficiently to implement Euler operators in solid modeling [10,18]. Complexity analysis can be found in [12,28].

4. System Overview

This system has one main window that contains three areas (Fig. 3(a)). The main window has menu items for loading a mesh (Load Mesh), saving a mesh (Save Mesh), loading and running various algorithms



Algorithm: faceAdjacentFaces

Name	Refers To
f	unassigned
e	unassigned
se	unassigned

```
algorithm faceAdjacentFaces( f : Face );
   structuretype WingedEdge
   description "Marks the adjacent faces of a face.
   var e, se : Edge;
begin
   e := f.incidentEdge;
   se := e;

   do
      if e.leftFace = f then
         mark e.rightFace;
         e := e.leftSuccessor;
      else
         mark e.leftFace;
         e := e.rightSuccessor;
      end;
   while e <> se;
end;
```

Face Table

ID#	Incident Edge	Labels
0	0	
1	3	
2	4	
3	7	
4	8	
5	11	
6	12	
7	15	
8	6	
9	10	
10	14	
11	2	
12	20	
13	22	
14	24	
15	26	
16	27	
17	30	
18	21	
19	32	
20	23	

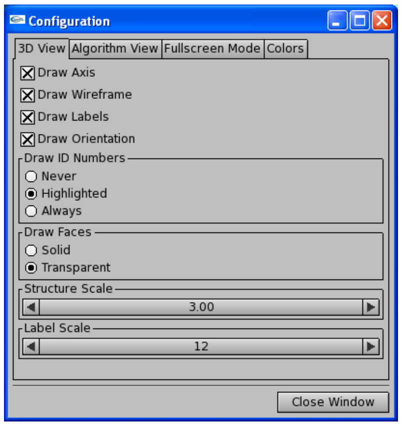
Edge Table

ID#	Start Vertex	End Vertex	Left Face	Right Face	Left Pred.	Left Succ.	Right Pred.	Right Succ.	Labels
0	9	4	0	7	1	2	15	13	
1	4	0	0	1	2	0	4	3	
2	0	9	0	11	0	1	19	15	
3	6	0	1	8	1	4	6	16	
4	4	6	1	2	3	1	5	6	
5	4	1	2	3	6	4	8	7	
6	1	6	2	8	4	5	16	3	
7	7	1	3	9	5	8	10	17	
8	4	7	3	4	7	5	9	10	
9	4	2	4	5	10	8	12	11	
10	2	7	4	9	8	9	17	7	
11	8	2	5	10	9	12	14	18	
12	4	8	5	6	11	9	13	14	
13	4	3	6	7	14	12	0	15	
14	3	8	6	10	12	13	18	11	
15	9	3	7	11	13	0	2	19	
16	1	0	8	12	3	6	21	20	
17	2	1	9	13	7	10	23	22	
18	3	2	10	14	11	14	25	24	
19	0	3	11	15	15	2	27	26	
20	10	0	12	17	16	21	30	28	

Vertex Table

ID#	Incident Edge	Labels
0	28	
1	31	
2	33	
3	35	
4	13	
5	35	
6	6	
7	10	
8	14	
9	15	
10	30	
11	32	
12	34	
13	29	

(a)



[✓ Draw Axis]
[✓ Draw Wireframe]
[✓ Draw Labels]
[✓ Draw Orientation]

Draw ID Numbers: Never [✓ Highlighted] Always

Draw Faces: Solid [✓ Transparent]

Structure Scale: 3.00

Label Scale: 12

(b)

Fig. 3. System Screen and the Configuration Window

provided by the instructor or designed by students (Load Algorithm and Run Algorithm), changing the system configuration (Configuration), switching between window mode and full screen mode for presentation purpose (Fullscreen), and exiting the system (Quit). The large display area, the Mesh View, shows the loaded mesh; and the bottom area, in this order, has the face table, edge table and vertex table. If an algorithm is being animated, the Algorithm View area on the right edge shows the animated algorithm. The Algorithm View area is further divided into three sections: the top one shows all variables and their current values, the middle one has the animated algorithm with keywords and comments shown in boldface and italic, respectively; and the bottom one is for animation control.

Menu items Load Mesh, Save Mesh, Load Algorithm and Run Algorithm will bring up a window asking the user to pick or input a filename and/or an algorithm to use. Currently, supported formats include the widely used `.obj` and `.ply` formats, and the simple tabular winged-edge `.wtb` format in Table 2. More formats will be added in the future. A user may choose an input format, load a mesh, examine its winged-edge tables, and save the mesh to another format. Since this system converts its input to the winged-edge table format to be used internally for visualization and algorithm animation, it can also be used as a simple file format converter.

The Fullscreen menu item brings the system display to full screen for presentation purpose and classroom use, and an Escape Fullscreen button that appears at the upper-left corner on the screen returns to windows view. The Configuration menu item brings up a small window, the Configuration window (Fig. 3(b)), allowing the user to set various options. This window has four tabs, 3D View, Algorithm View, Fullscreen Mode, and Colors. The 3D View tab permits the user to set various display options: coordinate axes, the wireframe of the mesh, the ID/labels of vertices, orientation of each face, solid or transparent faces, and scaling factors of the mesh structure and labels. The Algorithm View tab permits the user to set font size to be used in the Algorithm View area. The Fullscreen Mode tab allows the user to set what is shown in full screen mode, which may include the algorithm, tables and menu buttons. This is very useful if the system is being used for presentation and demonstration. Finally, the Colors tab provides the user a chance to modify the color scheme used in the system. In this way, the user would be able to adjust the colors for his/her preference.

5. Visualizing the Data Structure

Since the main goal of this system is the visualization of the winged-edge data structure, the design merit is allowing the user to see every piece of information in the winged-edge data structure of a mesh.

On the Mesh View, vertices and oriented edges of the loaded mesh are shown as small spheres and

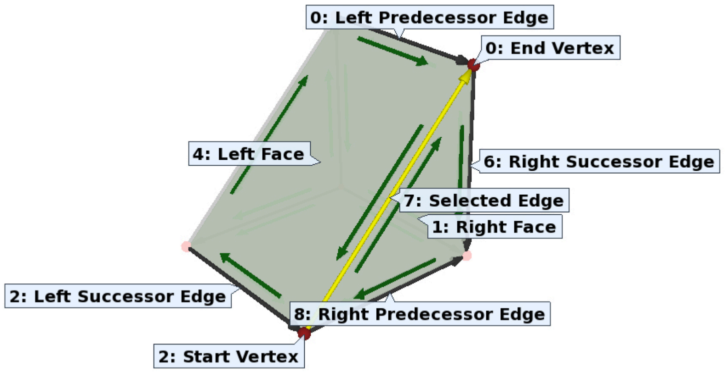


Fig. 4. Mesh View

solid arrows, respectively. The displayed mesh can be rotated by dragging with the left mouse button, while the middle and right buttons are for translation and zooming. The user may click on a vertex, an edge, or a face to display its related information in winged-edge tables. For example, if the user clicks on an edge, all information in Fig. 1 will be displayed with the selected edge marked in a different color. Fig. 4 shows the display of selecting edge 7. The Mesh View displays all nine pieces of information of edge 7. Items are shown in different colors and marked with labels. Each label has two components: the element number (*i.e.*, an index to the corresponding table) and its description such as "Selected Edge", "Left Predecessor", "Start Vertex" and "Right Face".

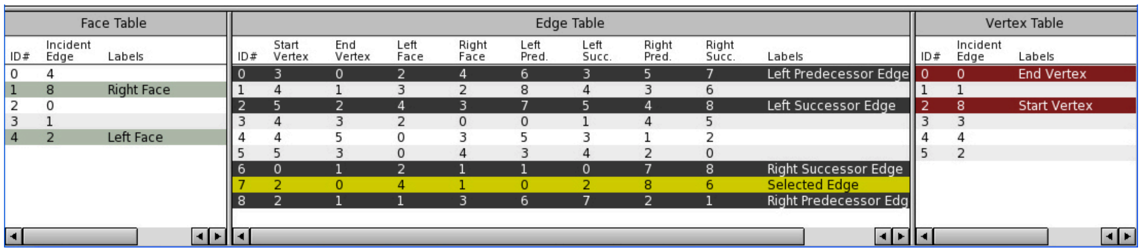
The bottom of the main window shows all three tables with elements highlighted in the same color as used in the Mesh View. The user may also select an element by clicking on the corresponding row in one of the three tables. The edge table displays the selected edge and the predecessor and successor edges of the left and right faces using a different color. The vertex table highlights the start and end vertices of the selected edge, while the face table highlights the left and right faces. Thus, the relationship of each selected element with other elements is clearly revealed, and it is easy for the user to verify and visualize the organization of the winged-edge data structure of the loaded mesh. Fig. 5 shows the tables after edge 7 is selected.

6. Animating Algorithms

The most important component of this system is the visualization and animation of the winged-edge data structure and its algorithms. The user may use a simple pseudo code-like language to design algorithms, which will be loaded, parsed, and animated in step-wise or continuous mode (Section 6.1). Currently, this system provides the following basic algorithms: (1) given a vertex find its incident edges and adjacent faces, (2) given a face, find its incident vertices, edges and faces, and (3) given a vertex, find its link and star. As an algorithm is being animated, the statement being executed is highlighted and the variables and their values are also shown in various colors (Section 6.2). Additionally, mesh elements (*e.g.*, vertices, edges and faces), operators (*e.g.*, comparisons and selections), and values of variables are shown on the Mesh View, and the lower-left corner of the Mesh View (*i.e.*, the message area) displays a description of each animated step. In this way, the user will be able to follow the execution of an algorithm easily (Section 6.3). Combining data structure visualization, simple pseudo-code algorithm design, and algorithm animation, instructors and students will have an environment for teaching, practicing, and learning the basics of the winged-edge data structure.

6.1. *A Simple High Level Pseudo Code-Like Language*

This system uses a simple high level, pseudo code-like language for algorithm design. This simple language provides sufficient power for a user to easily use and look up any element in a winged-edge structure, and hence hides the un-necessary implementation details from the users. A complete set of syntax rules in EBNF form is in Appendix A. Fig. 6 is an example that finds all incident edges of a given vertex. In an algorithm, the user may use types `Vertex`, `Edge` and `Face`, executable statements `if-then-else`, `while-do`, and `do-while`, and as-



Face Table

ID#	Incident Edge	Labels
0	4	
1	8	Right Face
2	0	
3	1	
4	2	Left Face

Edge Table

ID#	Start Vertex	End Vertex	Left Face	Right Face	Left Pred.	Left Succ.	Right Pred.	Right Succ.	Labels
0	3	0	2	4	6	3	5	7	Left Predecessor Edge
1	4	1	3	2	8	4	3	6	
2	5	2	4	3	7	5	4	8	Left Successor Edge
3	4	3	2	0	0	1	4	5	
4	4	5	0	3	5	3	1	2	
5	5	3	0	4	3	4	2	0	
6	0	1	2	1	1	0	7	8	Right Successor Edge
7	2	0	4	1	0	2	8	6	Selected Edge
8	2	1	1	3	6	7	2	1	Right Predecessor Edg

Vertex Table

ID#	Incident Edge	Labels
0	0	End Vertex
1	1	
2	8	Start Vertex
3	3	
4	4	
5	2	

Fig. 5. Tables Content Showing the Selected Edge

signment `:=`, equal `=`, not equal `<>` and "look-up" operators. Look-up operators are keywords that can be appended to a variable. For example, if `e` is an edge (Fig. 6), `e.startVertex` is the start vertex of edge `e`, and `e.leftSuccessor` is the successor edge of `e` when traversing its left face. For vertex `v`, `v.incidentEdge` is the incident edge of `v` in the vertex table. Similarly, `f.incidentEdge` is the incident edge of face `f` in the face table. There are two "action" operators, `highlight` and `mark`. The former highlights the elements on the Mesh View, and the latter specifies algorithm output.

```
 1 algorithm vertexAdjacentEdge(v : Vertex);
 2   structuretype WingedEdge
 3   description "Mark the adjacent edges of a vertex."
 4   var e, se : Edge;
 5 begin
 6   e := v.incidentEdge;
 7   se := e;
 8   do
 9      mark e;
10      if e.startVertex = v then
11         e := e.leftSuccessor;
12      else
13         e := e.rightSuccessor;
14      end;
15   while e <> se;
16 end;
```

Fig. 6. Find All Incident Edges of a Given Vertex

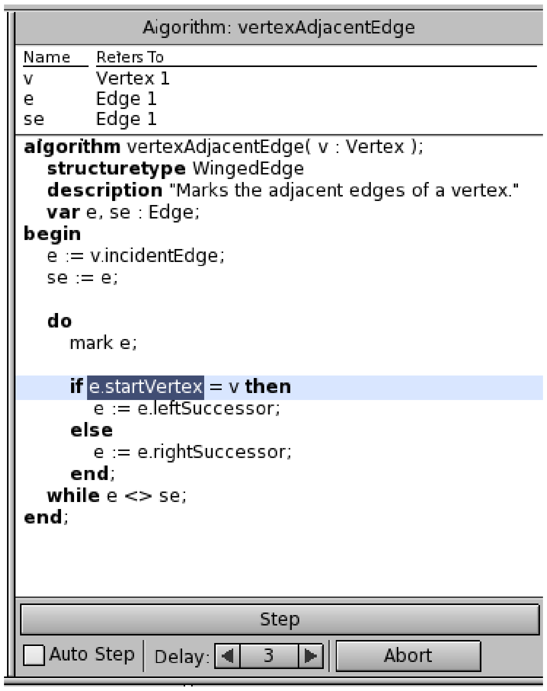
Algorithm `vertexAdjacentEdge()` in Fig. 6 takes a vertex `v` as its input and reports all of its incident edges. Line 2 declares that the underlying data structure is `WingedEdge` with keyword `structuretype`, and line 3 specifies a description string which is used by the system and the user to identify this algorithm (*e.g.*, picking an algorithm to run). As in C++, a comment starts with a "//" and extends to the end of the same line.

This algorithm uses two `Edge` variables `e` and `se`. Line 6 retrieves an incident edge of `v` and stores it to `e`, and line 7 saves this edge to `se`. The `do-while` loop uses `e` as a working variable and loops around `v` to find other incident edges. Edge `e` is marked (line 9) because it is an incident edge of `v`. The `if-then-else` statement moves `e` to the next edge. To do so, `e`'s start vertex `e.startVertex` is retrieved and compared with `v`. If they are equal, `e` is the successor edge of its left face (line 11). Otherwise, `e` is the successor edge of its right face (line 13). The `while` condition (line 15) tests if edge `e` returns to the initial edge `se`. If it does, the algorithm has found all incident edges of `v`. Otherwise, the algorithm loops back and uses `e` to find the next incident edge. Hence, edge `e` moves around vertex `v` in counter clock-wise order. One can easily modify the `if-then-else` for a clock-wise traversal. This shows a fact stated in Section 2 that the orientation of each face does not matter. What matters most is the algorithm used to generate mesh elements.

User designed algorithms are saved in a text file. When the system starts, the default algorithm source code is read from a file, 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 area. A user may load additional algorithms into the system at any time with the Load Algorithm menu button, and use the Run Algorithm menu button to select and run an algorithm. Therefore, an instructor may design new algorithms, load them, and use the animation feature for classroom presentation. On the other hand, students may use this feature to practice algorithm design and debugging. For example, a student may use the system to verify a concept or an algorithm before actual implementation takes place.

6.2. *The Algorithm View*

An executing algorithm waits for the user to select elements for the algorithm's parameters. Then, the Algorithm View area highlights the syntactic element being executed and the line that contains the element. For example, in Fig. 7, which is the algorithm in Fig. 6, shows that the syntactical element being executed is `e.startVertex`, retrieving the start vertex of edge `e`. If execution continues, the next element to be highlighted will be `e.startVertex = v`, because the retrieved vertex will be compared with the given vertex `v`. Note that the line containing the executing element is highlighted differently. In this way, the user will be able to know exactly which element in an algorithm is being executed.


Algorithm: vertexAdjacentEdge

Name	Refers To
v	Vertex 1
e	Edge 1
se	Edge 1

```
algorithm vertexAdjacentEdge( v : Vertex );
   structuretype WingedEdge
   description "Marks the adjacent edges of a vertex."
   var e, se : Edge;
begin
   e := v.incidentEdge;
   se := e;

   do
      mark e;

      if e.startVertex = v then
         e := e.leftSuccessor;
      else
         e := e.rightSuccessor;
      end;
   while e <> se;
end;
```

Fig. 7. The Algorithm View Area

Before animation starts, elements of the mesh are shown in light color so that they will not have a significant impact on visualization. As animation continues, involved elements will be shown in different colors with the same color pattern used to highlight the rows of these elements (Fig. 8). The default color scheme is shown in Table 3; however, this color scheme can be modified with the Colors tab under the Configuration menu button. Additionally, if an element becomes the value of some variables, the `labels` field (*i.e.*, the last column) in its table entry shows the variable names. For example, the edge table in Fig. 8 shows that edge 8 is currently the value of variables `e` and `se`, and vertex 10 is the value of variable `v`. Hence, the user can accurately and easily keep track of the execution of the algorithm and the value of each variable.

The algorithm shown in the Algorithm View area can be animated in a step-wise mode or a continuous

Table 3
Default Color Scheme

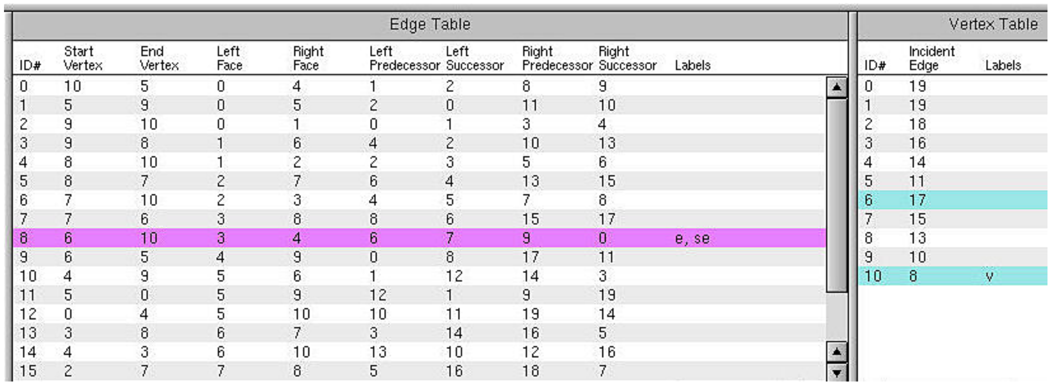
Color	*Meaning*
Gray	Base color
Blue	Algorithm output
Green	Elements in a look-up operator
Yellow	Highlighted elements
Purple	Elements with variables referring to
Light Blue	Elements in a comparison

mode with adjustable delay. In the step-wise mode, the user may use the Step button or space bar to step through the algorithm. In the continuous mode, the user may set a time delay between steps to execute the animation without user intervention. The user may switch between modes or abort the animation at any time.

6.3. *Algorithm Animation – An Example*

This section presents an example of applying the algorithm in Fig. 6 to a tetrahedron (Fig. 2). After loading the tetrahedron file and starting the algorithm, the user clicks on vertex 1 to find its incident edges. Thus, vertex 1 becomes the value of variable `v` (*i.e.*, the formal argument of the algorithm), which is indicated by the label "1: v" (Fig. 9(a)). Although the orientation of each face is not shown, a user may turn on this feature. Then, the algorithm executes line 6, and retrieves an incident edge of `v` from the vertex table. As a result, this incident edge (*i.e.*, edge 1) is shown in green (Fig. 9(b)). This edge is assigned to edge variable `e`, making the label of edge 1 "1: e" (Fig. 9(c)), and edge variable `e` now has value 1. Line 7 saves the value of `e` to `se`, making edge 1 the value of variables `e` and `se` (Fig. 9(d)).

Next, the execution reaches the `do-while` loop. Edge `e` is marked in blue (line 9) because it is an incident edge of `v` (Fig. 9(e)). The `if` statement compares the start vertex of `e` with `v`. To do so, `e.startVertex` is retrieved (*i.e.*, vertex 1) and `e` is shown in green (Fig. 9(f)) because a look-up operator is applied to `e`. Then, this vertex is compared with `v` (Fig. 9(g)). There should be two solid arrows pointing to vertex 1, one for vertex `v` and the other for vertex `e.startVertex`. They are blocked by the label of vertex 1; however, the user may rotate or zoom the mesh to reveal the arrows. Since `e.startVertex` and `v` are equal, both being vertex 1, the left successor of `e` is assigned to `e`. In this case, the left successor is edge 4, and `e` becomes edge 4,



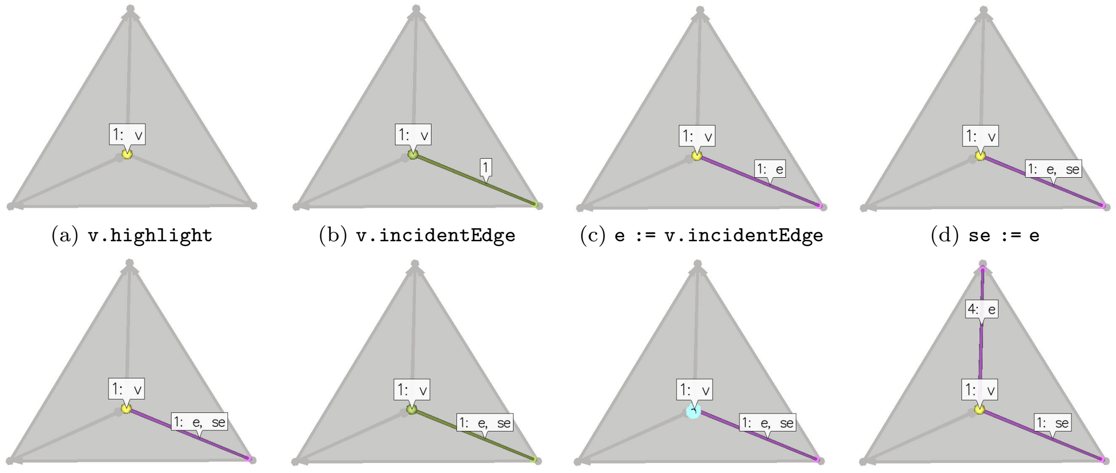
Edge Table

ID#	Start Vertex	End Vertex	Left Face	Right Face	Left Predecessor	Left Successor	Right Predecessor	Right Successor	Labels
0	10	5	0	4	1	2	8	9	
1	5	9	0	5	2	0	11	10	
2	9	10	0	1	0	1	3	4	
3	9	8	1	6	4	2	10	13	
4	8	10	1	2	2	3	5	6	
5	8	7	2	7	6	4	13	15	
6	7	10	2	3	4	5	7	8	
7	7	6	3	8	8	6	15	17	
8	6	10	3	4	6	7	9	0	e, se
9	6	5	4	9	0	8	17	11	
10	4	9	5	6	1	12	14	3	
11	5	0	5	9	12	1	9	19	
12	0	4	5	10	10	11	19	14	
13	3	8	6	7	3	14	16	5	
14	4	3	6	10	13	10	12	16	
15	2	7	7	8	5	16	18	7	

Vertex Table

ID#	Incident Edge	Labels
0	19	
1	19	
2	18	
3	16	
4	14	
5	11	
6	17	
7	15	
8	13	
9	10	
10	8	v

Fig. 8. The Edge and Vertex Tables



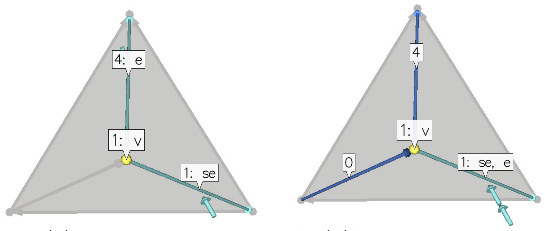
(a) `v.highlight` (b) `v.incidentEdge` (c) `e := v.incidentEdge` (d) `se := e`

(e) `e.mark` (f) `e.startVertex` (g) `if e.startVertex=v` (h) `e:=e.leftSuccessor`

Fig. 9. Operator Animation – I

overwriting the original edge 1, as shown in Fig. 9(h).

Then, the algorithm executes `while e <> se`, and two light blue solid arrows point to edge 1 and edge 4 (Fig. 10(a)) showing the edges under comparison. Since `e` and `se` are not equal, the execution loops back and repeats the steps shown in Fig. 9. The loop body will execute two more times. When edge `e` returns to edge `se`, `e` and `se` are equal, both being edge 1 as shown by the two light blue solid arrows in Fig. 10(b), and the execution of the `do-while` loop completes. This brings the algorithm execution to an end.



(a) `while e <> se` (b) The Last `while`

Fig. 10. Operator Animation – II

7. Experience Discussion

7.1. *Course Background Information*

While the content in a typical computer graphics course varies, many textbooks focus more on rendering with just a brief survey on modeling/design skills. Wolfe's survey also showed similar findings [27]. Even though Wolfe's work was published in 1999 and many popular textbooks have been updated and expanded since, the basic theme did not change much. Some educators consider programming in a graphics API being *the* skill that a student must know for his/her career. This is not entirely true, because knowing an API, which is similar to knowing a programming language, is just the beginning and students must acquire other vital skills to have successful professional careers.

Under the support of National Science Foundation, the authors addressed this issue by including the following four important components in a more balanced way: rendering, modeling/design, animation and postprocessing [14,15], and created a junior elective course *Elementary Geometric Objects and Processing* [21] in which the most fundamental and important topics in modeling and design are presented. Topics covered include boundary representations, constructive solid geometry, continuity issues, Bézier, B-Spline and NURBS curves and surfaces, curve and surface interpolation and approximation, subdivision surfaces, mesh representations, mesh simplification and compression, and multiresolution modeling. A web-based tutorial was developed to disseminate this material, which has become a popular reference on the web for educators, students, researchers and professionals [22]. Details can be found in [7,6,17,13,16,23,29–31].

7.2. *Findings*

In our course (mentioned in Section 7.1), a pretest is conducted in the first class meeting and a post-test and attitudinal survey are performed in the last. See [16] for some preliminary results. Through these tests and survey, the authors hope to determine what the most difficult topics are and how the problems can be addressed properly. The most difficult topics according to the students, surprisingly enough, are the winged-edge data structure, continuity issues for curves and surfaces, and curve and surface interpolation and approximation. It is understandable that the latter two may be difficult due to the unavoidable mathematical content. But, why would the winged-edge data structure be difficult? Surveys in recent years revealed that the difficulty is not in the data structure itself. Instead, the most difficult part, as virtually all students agreed upon, is how to use this data structure to develop applications (*i.e.*, the algorithmic component). The second exercise in this course is to design a program that reads in a mesh in the winged-edge tabular format and displays it in a window. Other problems also appeared in quizzes, midterm and final exams.

Carefully examining student programs showed that some students completely ignored the adjacency relations recorded in the tables, and their programs searched the tables to answer topological inquires. As a result, the performance of their programs cannot be efficient. Initially, this inefficiency was not obvious for simple meshes; however, when students were asked to implement other mesh applications that require many topological inquires (*e.g.*, subdivision surfaces), sooner or later, they discovered that the speed difference may be many times slower for larger meshes and can be prohibitively high for complex meshes. Moreover, experience also showed that students struggled more in the winged-edge data structure than they did in other topics that many educators considered to be difficult (*e.g.*, knot insertion and de Boor's algorithm in B-Spline curves and surfaces). The main evidence is that the successful rate of the winged-edge data structure exercises was lower than those of B-Spline based exercises. The former had an average successful rate around 80% while the latter was usually higher than 90%.

7.3. *Overcoming the Problems*

Initially, the authors believed that extensive class discussion would be sufficient to explain the concept, merit, and use of the winged-edge data structure; however, it became obvious very soon that this was not the case for two major reasons. **First**, class discussion, no matter how sophisticated is, cannot capture the 3-dimensional sense of a mesh, because students cannot "rotate" a drawing to "see" the details of the "other side". The high probability of making minor mistakes here and there (*e.g.*, incorrect interpretation of left and right faces of an edge on the "other side" of the board) could easily produce incorrect and totally useless tables. Consequently, it is not easy to successfully and correctly construct a winged-edge data structure for a moderately complex model. Even a cube can be very challenging for board work. **Second**, algorithms that use the winged-edge data structure representation are usually not very straightforward and require some additional thinking and understanding. This means only presenting the data structure is insufficient for students to use it properly and successfully. A discussion of the merit of the data structure and algorithms is required.

In recent years, the subdivision techniques have proven to be a promising modeling tool, and the use of meshes increases steadily. Thus, computer graphics educators face a grand challenge: how can we help students learn mesh based modeling and design skills? To address this issue, a number of mesh based tools such as subdivision techniques, mesh simplification, mesh compression, multiresolution modeling, and mesh reconstruction are under construction. All of these tools start with a mesh representation. While there are many data structures for mesh representation, the winged- and half-edge data structures are certainly the most basic and commonly used ones. Therefore, it is reasonable to start with a visualization and animation tool for these two data structures.

This tool was used in classroom presentations and given to students for their own practice. However, a controlled tests of the effectiveness of this tool was not performed for a major reason. The authors felt that dividing the class into two groups, one of which would be taught with the tool and the other without, is not fair to the students, especially that the value of visualization and animation tools have been widely considered effective and justified. The authors' experience and student feedback indicated that this tool is useful in understanding the merit and details of the data structure, and can help the students develop their own programs. More importantly, once they understand the key concept, students can indeed use the data structure successfully, and appreciate the speed gain when handling large meshes. Since this tool includes a set of basic algorithms and can overcome the difficulties mentioned earlier, it can also be used as a platform for further study. For example, an instructor may ask the students to design some simple algorithms such as (1) given a tetrahedron, find the "opposite" face of a vertex, (2) given a cube, find the "opposite" face of a face, (3) given a quad mesh, find the two "opposite" edges of an edge, (4) given an octahedron, find the "opposite" vertex of a selected vertex, and (5) determine if a given vertex is *extraordinary* (*i.e.*, valency not being 4). In summary, the authors believe this tool is worth being used by instructors for classroom presentation and demonstration, and students to visualize and animate the basic data structure and algorithms. This and other tools being developed will be part of the authors' MeshMentor distribution.

8. Conclusions

The above presented a visualization and animation tool for the winged- and half- edge data structures. Previously, board work and diagrams were used in classroom presentation. After many years' experience and input from other educators and students, the authors decided to convert their materials to a visualization/animation tool. This tool can help users visualize the winged-edge data structure with a graphical display and the traditional table format with extensive labeling. With the simple pseudo code-like language, instructors may use this tool for presentation and demonstration purposes with the supplied basic algorithms or other new algorithms. The animation component is capable of animating an algorithm written in the simple language and showing all activities during its execution. This makes algorithm presentation, learning, design and evaluation much easier.

Since mesh related techniques have become more important and widely used, it is time for computer graphics educators to rethink course content and add mesh related topics to their syllabi. Because winged- and half- edge data structures are basic, they are certainly good candidates to be considered. The authors hope their work may serve as the starting point for emphasizing the modeling component in computer graphics courses. After all, there would be no image if their are no geometric objects in a scene. In the near future, this tool will be extended in a number of directions. For example, the simple language may be extended to allow integer and float types, array, and calls among algorithms, and permit modifications (*i.e.*, adding and removing elements) to be applied to the existing data structures. With these new features, a user may use this tool to design more complex algorithms such as mesh compression and triangle strip generation for more efficient rendering of triangular meshes. An algorithm library may be possible. The visualization and animation components may also be extended to support these new features. Interested readers may find more about this work and future developments at `www.cs.mtu.edu/~shene/NSF-2`.

Appendix A. EBNF Definition of Our Simple Language

The following is a list of syntax rules in the EBNF form, where non-terminals are in italic and terminals (*i.e.*, reserved words) and operators are in courier font. The non-terminal *expr* is defined as usual; however, only comparison operators `=` and `<>` are supported currently. In a future version with the `Integer` and `Float` types, all arithmetic operators will be supported fully.

```
file := algorithm_declaration ;
        [ algorithm_declaration ; ]*
algorithm_declaration :=
      algorithm identifier ( parameter_list ) ;
        structuretype meshtype
        [ description string ]
        [ var decls_list ; ]
      begin
        [ statement_list ]
      end
parameter_list :=
      identifier_list : type
        [ ; identifier_list : type ]*
decls_list := identifier_list : type
        [ ; identifier_list : type ]*
identifier_list := identifier [ , identifier ]*
statement_list := statement ; [ statement ; ]*
statement := assignment | while_statement
    | do_while_statement | do_statement | if_statement
    | break | continue | action_statement
while_statement := while expr do statement_list end
do_while_statement := do statement_list while expr
do_statement := do statement_list end
if_statement :=
        if expr then statement_list
          [ else if expr then statement_list ]*
          [ else statement_list ]
        end
action_statement := action_type  element
action_type := mark | highlight
element := identifier | element . pointer
pointer := startVertex | endVertex
    | leftFace | rightFace | leftPredecessor
    | rightPredecessor | leftSuccessor
    | rightSuccessor | vertex | face
    | incidentEdge
assignment := identifier := expr
mesh_type := WingedEdge | HalfEdge
type := Vertex | Edge | Face
identifier := letter [ letter | digit ]*
string := " character* "
```

Acknowledgment

This work was supported by the National Science Foundation under grants DUE-9653244, DUE-9952621 and DUE-0127401, and by a Michigan Research Excellence Fund 1998-1999. An IBM Eclipse Innovation Award 2003 also contributed to the development of this tool. Additionally, the third author appreciates the hospitality of Dr. Horng-jinh Chang, the former President of Tamkang University, Taipei, Taiwan, and Dr. Chuan-Jen Chyan, Chair of Department of Mathematics, for a short-term visit in the summer of 2004 during which the development of this work and other related research activities were carried out.

References

[1] B. G. Baumgart, Winged-Edge Polyhedron Representation, Tech. rep., Stanford University, Computer Science Department, technical Report SRAN-CS-320 (1972).

[2] B. G. Baumgart, A Polyhedron Representation for Computer Vision, in: National Computer Conference, AFIPS Conference Proceedings, 1975.

[3] S. Bischoff, L. Kobbelt, Teaching Meshes, Subdivision and Multiresolution Techniques, Computer-Aided Design 36 (2004) 1483–1500.

[4] CGAL, Computational Geometry Algorithms Library, available at `www.cgal.org` (2005).

[5] Computer Graphics Group, OpenMesh, available at `www.openmesh.org` (2005).

[6] J. Fisher, J. Lowther, C.-K. Shene, Curve and Surface Interpolation and Approximation: Knowledge Unit and Software Tool, in: ACM 9th ITiCSE 2004 Conference, 2004.

[7] J. Fisher, J. Lowther, C.-K. Shene, If You Know B-Splines Well, You Also Know NURBS!, in: ACM 35th Annual SIGCSE Technical Symposium, 2004.

[8] L. D. Floriani, L. Kobbelt, E. Puppo, A Survey on Data Structures for Level-of-Details Models, in: N. A. Dodgson, M. S. Floater, M. A. Sabin (eds.), Advances in Multiresolution for Geometric Modeling, Springer-Verlag, 2005, pp. 49–74.

[9] J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes, Computer Graphics: Principles and Practice, 2nd ed., Addison-Wesley, 1990.

[10] A. S. Glassner, Maintaining Winged-Edge Models, in: J. Arvo (ed.), Graphics Gems II, Academic Press, 1991, pp. 191–201.

[11] C. M. Hoffmann, Geometric & Solid Modeling: An Introduction, Morgan Kaufmann, 1989.

[12] L. Kettner, Designing a Data Structure for Polyhedral Surfaces, in: Proceedings of the Fourteenth Annual Symposium on Computational Geometry, ACM, 1998.

[13] J. L. Lowther, C.-K. Shene, Geometric Computing in the Undergraduate Computer Science Curricula, The Journal of Computing in Small Colleges 13 (2) (1997) 50–41.

[14] J. L. Lowther, C.-K. Shene, Rendering + Modeling + Animation + Postprocessing = Computer Graphics, The Journal of Computing in Small Colleges 16 (1) (2000) 20–28.

[15] J. L. Lowther, C.-K. Shene, Rendering + Modeling + Animation + Postprocessing = Computer Graphics, Computer Graphics 34 (4) (2000) pp. 15–18, reprinted version of [14].

[16] J. L. Lowther, C.-K. Shene, Computing with Geometry as an Undergraduate Course: A Three-Year Experience, in: ACM 32nd Annual SIGCSE Technical Symposium, 2001.

[17] J. L. Lowther, C.-K. Shene, Teaching B-splines Is Not Difficult!, in: ACM 34th Annual SIGCSE Technical Symposium, 2003.

[18] M. Mäntylä, An Introduction to Solid Modeling, Computer Science Press, 1988.

[19] J. J. McConnell, Computer Graphics: Theory into Practice, Jones and Bartlett, 2006.

[20] B. Neperud, Visualizing Mesh Data Structures and Algorithms, in: Proceedings of MICS 2005, 2005, available online at `http://www.micsymposium.org/mics_2005/index.htm`.

[21] C.-K. Shene, CS3621 Computer Graphics: Elementary Geometric Objects and Processing, department of Computer Science, Michigan Technological University, available at `www.cs.mtu.edu/~shene/COURSES/cs3621/` (1997).

[22] C.-K. Shene, CS3621 Introduction to Computing with Geometry Notes, department of Computer Science, Michigan Technological University, available at `www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/` (1997).

[23] C.-K. Shene, Raytracing as a Tool for Learning Computer Graphics, in: ASEE/IEEE 32nd Frontiers in Education, vol. III, 2002.

[24] P. Shirley, Fundamentals of Computer Graphics, A K Peters, 2002.

[25] M. Slater, A. Steed, Y. Chrysanthou, Computer Graphics and Virtual Environments: from Realism to Real-Time, Pearson Education, 2002.

[26] K. Weiler, Edge-Based Data Structures for Solid Modeling in Curved-Surface Environments, IEEE Computer Graphics and Applications 5 (1) (1985) 21–40.

[27] R. Wolfe, A Syllabus Survey: Examining the State of Current Practice in Introductory Computer Graphics Courses, Computer Graphics 33 (1) (1999) 32–33.

[28] T. Woo, A Combinatorial Analysis of Boundary Data Structures, IEEE Computer Graphics and Applications 5 (1) (1985) 19–27.

[29] Y. Zhao, J. L. Lowther, C.-K. Shene, A Tool for Teaching Curve Design, in: ACM 29th SIGCSE Technical Symposium, 1998.

[30] Y. Zhao, Y. Zhou, J. L. Lowther, C.-K. Shene, Cross-Sectional Design: A Tool for Computer Graphics and Computer-Aided Design Courses, in: ASEE/IEEE 29th Frontiers in Education, vol. II, 1999.

[31] Y. Zhou, Y. Zhao, J. L. Lowther, C.-K. Shene, Teaching Surface Design Made Easy, in: ACM 30th SIGCSE Technical Symposium, 1999.