VISPAK: A PACKAGE OF VISIBILITY ALGORITHMS WRITTEN IN LEDA *

L. JACKSON, H. PINTO, S. K. WISMATH†

Abstract. VisPak is a software package of visibility routines for interactively displaying and manipulating the visibility graph of sets of objects in the plane. Various visibility algorithms have been implemented in C++ using LEDA as a base; binary and source code are available. VisPak should be a useful tool for researchers in the field of visibility algorithms and computational geometry.

1991 Mathematics subject classification. Primary 68U, secondary 05

Keywords. computational geometry, visibility graph, LEDA

1. Introduction and Definitions. Visibility research focuses on the combinatorial and algorithmic issues related to the visibilities of objects in the plane. Although usually considered a sub-field of computational geometry, visibility problems are frequently motivated by robotics, graphics, VLSI design and combinatorial mathematics.

VisPak is a set of tools written at the University of Lethbridge and includes several of the common operations required by researchers in the area of visibility. It is written in C++ and uses the LEDA (Library of Efficient Data types and Algorithms) package available from the Max-Planck Institute at Saarbrucken []. VisPak is available free of charge for research and teaching purposes with the same constraints as LEDA. Both binary and source code versions of VisPak can be obtained from the University of Lethbridge and were written for SPARCstations running the Solaris operating system. The source code version will require that LEDA is installed on the host system, but the binary version should run directly. We make no guarantee of correctness, assume no responsibility for its maintenance, and the package is distributed on an "as is" basis.

Typically, researchers in visibility restrict themselves to a set O of objects (ex. points, line segments, polygons, etc.) in the plane and use a natural definition of visibility, namely, two objects o_1, o_2 are **visible** if and only if it is possible to draw a line segment between o_1 and o_2 that intersects no other object of O. There are several subtle issues involved in the definition of visibility and various flavours have been proposed, for example: strong, weak, epsilon, one directional, two directional, weighted, etc.; the non-specialist is referred to the following survey works: [4], [5], and [6].

The **visibility graph** of a set O of objects has one node for each object and an edge between a pair of nodes if and only if the corresponding objects are visible. There is a substantial literature on visibility graphs, in particular on their construction, reconstruction, and characterization, and again a variety of small differences between definitions. See for example [5], [8], [6].

* THIS WORK WAS SUPPORTED BY N.S.E.R.C.

† Department of Mathematics and Computer Science, University of Lethbridge, 4401 University Drive, Lethbridge, AB, Canada, T1K 3M4; E-mail: wismath@cs.uleth.ca

VisPak allows the user to input a variety of objects (line segments, polygons, rectangles) and construct the associated visibility graph. The primary purpose here is as a tool for viewing the visibility graphs of objects and to be able to interactively change the set of objects and associated visibility graph. In general, the algorithms are not the most efficient known, but rather simple, practical solutions were implemented. Currently, five visibility graphs are provided; details of each are contained in the following sections.

A second goal of this project was ti implement a particular algorithm for efficiently computing the visibility polygons of the endpoints of a set of line segments. The algorithm uses a technique called Depth First Spiralling which has been proposed as a practical method of traversing the endpoints of line segments, given their (endpoint) visibility information [2].

2. Implementation of visibility algorithms in LEDA. Currently, the VisPak package contains implementations of six different visibility algorithms, five implementing the construction of visibility graphs of various objects, and one constructing visibility polygons of line segment endpoints. The visibility graph construction implementations include: Vertical Line Segments (with one line of sight); Rectangles (with two lines of sight); Vertices of a General Simple Polygon (interior visibility); Vertices of an Orthogonal Simple Polygon (interior visibility); Endpoints of Disjoint Line Segments. Also, the visibility polygons of the endpoints of a set of line segments has been implemented. A short description of each program follows.

2.1. Visibility Graph of Vertical Line Segments. The input to this program is a set of vertical line segments. The resulting visibility graph has a node for each line segment in the set, and an edge joining a pair of nodes, n_i and n_j, if there exists a *horizontal* line of visibility between the corresponding segments, s_i and s_j. The program takes point input from the mouse, using the first (left most) mouse button. The first point of every pair of inputs gives the (x, y) coordinates of one endpoint and the x coordinate of the second endpoint. The y coordinate of the second endpoint is the distance between the two input points. This ensures all segments are vertical.

- *Algorithm*: This is an implementation of a sweep line algorithm, set up to serve as a demonstration package. First the y coordinates of the endpoints of the segments are sorted. Next a sweep function begins at $y = -\infty$ and stops at the y coordinate of each of the endpoints and creates a list (*history*) of all segments visible to that endpoint. Finally a sweep line is moved from $-\infty$ up to a user chosen location. As the line passes through each endpoint, nodes corresponding to visible segments from the *history* are connected on the visibility graph and these segments highlighted. The program also has a function that reverses the sweep line, for demonstration purposes. It backs up the connections already made on the graph, so the sweep line output can be redisplayed.
- *Data Structures*: There are two main data structures used. Upon input, the segments are stored in a priority queue, with each item consisting of the segment and the y coordinate of this point. The priority queue is chosen because

of its *findmin()* function. Each segment is entered into the priority queue twice, once for each endpoint. During the sweep function a sorted sequence, *history*, is created. Each entry in *history* consists of the y coordinate of the corresponding endpoint, and a sorted sequence of the segments that are visible to that endpoint.

- *Analysis*: The sorting of the $2n = O(n)$ endpoints by y coordinate takes $O(n) * O(\log n)$ time using the insert function of the sorted sequence. The sweep function consists of a loop that is executed $O(n)$ times, with each iteration requiring $O(\log n)$ time, thus needing $O(n \log n)$ time. Finally the function that updates the graph and moves the sweep line requires a total of $O(n)$ iterations to create the entire graph, with each using $O(\log n)$ time. Therefore, the total time used is $O(n \log n)$.

2.2. Visibility Graph of Rectangles. The input to this program is a set of non-intersecting rectangles, whose edges are oriented parallel to the x and y axes. The resulting visibility graph has a node for each rectangle in the set, and an edge joining a pair of nodes, n_i and n_j, if there exists a horizontal or vertical line of visibility between corresponding rectangles, r_i and r_j. The user inputs, (using the mouse) two opposite corners of each rectangle. The program checks to ensure the new rectangle does not overlap any existing rectangle, then draws the rectangle on the window.

- *Algorithm*: The visibility graph is created using two sequential scan lines, one in each dimension.
- *Data Structures*: This program uses 5 main data structures. A list of polygons, is used to store all the rectangles. Two sorted sequences store the horizontal and vertical segments of the polygons. Another two sorted sequences indicate which segments are visible to each endpoint of each segment, one sequence for each dimension.
- *Analysis*: Since the visibilities in the two dimensions are determined sequentially, the $O(n \log n)$ time required for each are added, giving an $O(n \log n)$ total running time.

2.3. Visibility Graph of Vertices of a General Simple Polygon. The visibility graph created has a node for each vertex of a general simple polygon, and an edge for each pair of vertices that can see each other in the interior.

- *Algorithm*: The visibility check has been implemented using a brute force technique where every pair of vertices is checked for intersection with all sides of the polygon. A further check then determines whether the visibility is internal to the polygon.
- *Data Structures*: The visibility graph is stored as a parameterized graph. The node information is the point at which the node lies and is used to draw the graph. The edge information is an integer used to distinguish polygon sides or visibility edges via different colours. A list of segments keeps track of the polygon sides, and a list of points is used to determine which pairs of vertices can see each other.

- *Analysis*: This approach checks $O(n^2)$ pairs of endpoints against all $O(n)$ segments. Therefore the program runs in $O(n^3)$ time.

2.4. Visibility Graph of Vertices of an Orthogonal Simple Polygon. This program is very similar to the previous one, except that the polygon is restricted so that all edges are parallel to either the x or y coordinate axes. This restriction makes it necessary to tailor a separate procedure which orients each input polygon side horizontally or vertically, whichever is closer. The user is responsible for terminating input by returning to the start point.

- *Algorithm*: As in the previous case, this program uses a brute force approach.
- *Data Structures*: These are exactly the same as the previous program.
- *Analysis*: The program uses $O(n^3)$ time.
- *Problems encountered*: Every side must be orthogonal to the previous side. Consecutive vertical or horizontal sides will crash the program. Due to their complexity, edit options have not yet been added to this program.

2.5. Endpoint Visibility Graph of Disjoint Line Segments. This program creates a visibility graph that has a node for each endpoint of a set of disjoint line segments, and an edge joining a pair of nodes if they are visible to each other.

- *Algorithm*: The visibility check has been implemented with a brute force technique where every pair of endpoints is checked against all segments for intersections.
- *Data Structures*: The visibility graph is stored as a parameterized graph. Since this graph is also used in the following program, the node and edge structures contain more detail than is required at this stage. In addition, two lists are used, one contains the segments and the other tracks the endpoints.
- *Analysis*: There are $O(n^2)$ pairs of endpoints, each requiring $O(n)$ time to check against all segments. Therefore the program runs in $O(n^3)$ time.

2.6. Visibility Polygons of Endpoints of Line Segments. This program determines the vertices of a polygon that are visible from any of the endpoints of a set of non intersecting line segments. It relies on the depth first spiralling technique of Keil and Wismath as described in [2].

- *Algorithm*: An endpoint visibility graph is created first, using the previous Endpoint Visibility Graph of Disjoint Line Segments algorithm. Next the *Depth First Spiralling* algorithm is run to compute all the visibility polygons.
- *Data Structures*: This visibility graph is stored as a parameterized graph. Node information contains the point at which the node lies; the segment on which the point lies; and a boolean variable *visited* required to implement the depth-first spiral. Each edge e stores its stab point; the segment this point lies on; and pointers to preceding and succeeding edges of $e+\pi$. (See algorithm for details.) Other structures include an edge_map, *edge_ord*, used in Transition to sort the graph edges in anti-clockwise order; an edge_array, *Rev*, to permit constant time access to the reverse of any given edge; and a node_map, *known*, described in the algorithm and implemented as node-indexed pointers to edges. The

list, *VP_list*, stores the vertices of the visibility polygon for one node, and the segments on which these vertices lie. It is updated during, and completed by the end of the spiral procedure. The node_map, *VPoly*, is simply a node-indexed list of *VP_list*. Finally, the node_map, *Remove*, is a structure used specifically to keep track of and later remove from the visibility polygon of any given node, the point on the other end of the same segment. It is assumed that visibility is unhindered by this endpoint.

- *Analysis*: This implementation uses $O(n^3)$ preprocessing to create the endpoint visibility graph and $O(E)$ time to determine all the visibility polygons. (E is the number of edges in the visibility graph: $E \leq (n^2 - n)/2$.) Thus the overall running time is $O(n^3)$. Notice that the literature does contain algorithms that reduce the $O(n^3)$ running time of the endpoint visibility graph construction, but it has not yet been implemented by this group.
- *Problems Encountered*:
  - *Robustness (collinearity)*: This algorithm is based on the assumption that collinear points do not occur. If collinear points are detected, the user must edit appropriately. This is the easiest way around the problem. However, the implementation has not been entirely successful in detecting all collinear points. (To illustrate this, try running *vpoly* with input file *bbb*). The user is therefore warned that output could be incorrect or the program will fall into an infinite loop, if such an error goes through undetected.
  - *Pre-loading information*: In an effort to meet output sensitive time limits a lot of information is pre-loaded on the parameterized graph or other data structures. For example, edge_array *Rev* is needed for easy access to reverse edges; and *opp_pred/opp_succ* edges are vital for efficiently implementing the Stabs procedure.
  - *Inefficiency of the endpoint routine*: When running *vpoly* with more than 1000 segments, the brute force implementation of the endpoint visibility graph becomes a severe limitation. Further development of this package will implement a more efficient endpoint routine.

2.7. Editor. All of the programs contain a version of a graph editor that use LEDA's file input/output. Future releases will streamline the editting features. Each program in this package uses an individually tailored version of the editor. Editting options vary slightly from one program to another.

- *Data Structures*: In every program that edits segments, a sorted sequence, *S_structure*, is used to enable constant time access to any segment when either of its endpoints is selected. All changes are made to this sequence before being interpreted back into program-specific structures.The list, *prev_edit*, is used to keep track of and reinstate the last edit(s).
- *Problems encountered*: The *save* option appends to (but does not overwrite) the named file if existing. The simplest possibile data structures (i.e, points, segments) are used when LEDA performs a save operation. This could create

problems when loading previously saved sets of polygons.

3. Distribution.

4. Distribution. VisPak is available free of charge for research and teaching purposes. Binary versions of the programs can be obtained via anonymous ftp (ftp.uleth.ca in directory pub/mathematics) or on the web (URL http://www.cs.uleth.ca/../wismath/vis.html). The current versions of the programs are distributed on an "as is" basis for testing purposes and should run on any SPARC workstation running the Solaris operating system. As yet there are no plans for porting to other platforms, however the source code can also be obtained by contacting the authors. For more detailed information on using anonymous ftp, or the world wide web to get the programs, and to report bugs or suggest improvements, please e-mail the authors.

REFERENCES

[1] S. Ghosh, D. Mount, An Output Sensitive Algorithm for Computing Visibility Graphs, FOCS 1987, 11-19.

[2] M. Keil, S. K. Wismath, Depth first spiralling on the endpoints of a set of line segments in output sensitive time, submitted to International J. of Computational Geometry and Applications, Sept. 1995. Also available as technical report UL-CS 95-2.

[3] K. Mehlhorn, S. Naeher, LEDA: A platform for combinatorial and geometric computing, *Communications of the ACM*, 38, No 1, Jan. 1995, 96-102.

[4] J. O'Rourke, *Art Gallery Theorems and Algorithms*, Oxford University Press (1987).

[5] J. O'Rourke, *Computational Geometry Column 18*, SIGACT News Vol. 24, No 1 (1993), pp. 20-25.

[6] T. Shermer, Recent Results in Art Galleries, *Proceedings IEEE*, Vol. 20, No. 9, Sept. 1992, pp. 1384-1399.

[7] E. Welzl, Constructing the Visibility Graph for n-Line Segments in O(n^2) Time, *Information Processing Letters* 20 (1985), 167-171.

[8] S. K. Wismath, Computing the full visibility graph of a set of line segments, *Information Processing Letters* 42 (1992), 257-261.

[9] S. K. Wismath, Characterizing Bar Line-of-Sight Graphs, *Proc. 1st ACM Symposium on Computational Geometry*, Baltimore (1985), 147-152.

User's Manual

A. Installation. The VisPak package is available in two forms, as binaries, or as source code. Table 1 lists the programs and their associated files. When using binaries, refer to the executable for start up names. In order to compile the source code, the system must have LEDA-3.2 installed, and the appropriate paths set. The Code and Header file columns of Table 1 indicate which VisPak files are needed for each program.

Although the software is distributed on an *as is* basis, bug reprots and suggestions for imporvement may be e-mailed to the VisPak priject, in care of wismath@cs.uleth.ca

B. Edit Functions. Each of the six programs has edit features implemented. This section describes the general edit functions. Note, however that not all of the features are available in all of the programs. Those features that are not available for specific programs are indicated with the descriptions of each individual program.

PROGRAM	Executable	Code File	Header File
Visibility Graph of Vertical Line Segments	sweep	sweep.c sweep_edit.c sweep_edit2.c	sweep.h
Visibility Graph of Rectangle - Part A	rect	rectsweep.c rect_func.c rect_edit1.c rect_edit2.c	rect.h
Visibility Graph of Rectangle - Part B	rect2	rectsweep2.c rect_func.c rect_edit1.c rect_edit2.c	rect2.h *
Visibility Graph of Vertices of a General Simple Polygon	ordpoly	ordpoly.c opoly_func.c opoly_edit1.c opoly_edit2.c	opoly.h
Visibility Graph of Vertices of an Orthogonal Polygon	orthpoly	orthpoly.c opoly_func.c opoly_edit1.c opoly_edit2.c	opoly.h
Endpoint Visibility Graph of Disjoint Line Segments	endpt2	endpt2.c endpt2_driver.c	endpt2.h
Depth First Spiral (separate)	spiral	endpt2.c spiral.c spiral_driver.c	endpt2.h
Keil and Wismath Algorithm for Visibility Polygons	vpoly	driver.c endpt2.c stabs.c vpoly.c vispoly.c edit_func.c loadsave.c	endpt2.h

TABLE 1
Table of Program Files

The editor features are available in most programs by selecting the *options* button from the initial panel. When this button has been pressed, the following options are available:

- *load* a set of objects from a file.
- create a *random* set of non-intersecting objects of user input size.
- *save* a set of objects to a file.
- *clear* the current set of objects from both windows and all relevent data structures.
- *edit* the current set of objects in window W. The available edit options are:
  - *add* new objects to the window
  - *delete* objects from the window
  - *drag* an object to a different location in the window
  - *re-orient* an existing object by moving one of its endpoints
  - *undo* either the last edit or the last 5 edits
  - *return* from the edit options
- *return* from the options menu.

C. Running the Programs. Each individual program can be started in one of two ways:

- Clicking on the appropriate icon in the filemanager.
- Typing the executable filename in a shell tool.

The recommended method of starting any program is from the shell tool, since this would allow messages to be reported to the shell.

C.1. Visibility pairs of Line Segments. This section describes how to run the Visibility pairs of Line Segments program. Its executable filename is *sweep*.

Once invoked, the program will bring up two windows, V on the left and W on the right, and a panel, entitled *Horizontal Sweep Visibility* below W. The window W is for drawing the input line segments, while the window V is for output of the corresponding visibility graph[1]. The *Horizontal Sweep* Panel tells the program which function is to be performed next.

Normal program usage is via the following mouse operations:

- Press the *draw* button on the panel in order to begin drawing the input line segments. Then go to the W window and begin specifying the start and end-points of the line segments by moving the mouse to the necessary position and pressing the first mouse button. Note that the window does display the current mouse coordinates, and that the program will force the line segments to be horizontal. Press the third mouse (right most) button to indicate that all segments have been input. If any incorrect segment has been input, the *options* button can be used later to make changes.
- Press the *sweep* button to run the sweep line demonstration and to create the output visibility graph. Initially, the sweep line is assumed to be at the bottom

[1] A horizontal visibility definition is assumed.

of the screen. Click on a location in window W where the sweep line is to be moved to, and the visibility graph will be created up to that point. Clicking on a location that is above the current sweep line location will add more edges to the visibility graph, while a location below the current sweep line location will subtract edges from the graph. To create the complete visibility graph, choose a sweep line location that is above the entire set of line segments.

- The *options* button allows access to the edit features, as described above. Note that creating a random set of line segments is not available, since it has not yet been implemented, nor is the re-orient feature, since it is not appropriate for this application.
- Press the *quit* option to exit the program.

C.2. Visibility pairs of Rectangles. This section describes how to run the Visibility pairs of Rectangles program. Its executable filename is *rect*.

Once invoked, the program will bring up two windows, *Visibility Graph* on the left and *Set of Polygons* on the right, and a panel, entitled *Visibility Graph of Polygons* below. The *Set of Polygons* window is for drawing the input rectangles, while the *Visibility Graph* window is for output of the corresponding visibility graph[2]. The panel below controls which program function is to be performed next.

Normal program operation is in the following order:

- Press the *draw* button on the panel in order to begin drawing the input rectangles. Then go to the *Set of Polygons* window and begin specifying the diagonal corners of the rectangles by moving the mouse to the desired position and pressing the first mouse button. The program will force the rectangles to be properly oriented and ensures that none overlap. Press the third mouse button to indicate that all rectangles have been input.
- Press the *sweep* button to run a vertical sweep. The output visibility graph, displayed in the *Visibility Graph* window, shows the edges added as solid red lines. A *continue* button will be displayed and must be depressed to run the second (horizontal) sweep. Dotted blue lines are used to display the edges added in this sweep. This program does not have the backing up feature of the previous demonstration sweep package.
- The *options* button allows access to the edit features, as described above. Note that creating a random set of rectangles is not available, nor is the re-orient feature, since it is not appropriate for this application.
- Press the *quit* option to exit the program.

C.2.1. Visibility Pairs of Rectangles, version 2. This section describes a variation of the Visibility Pairs of rectangles program. Its executable filename is *rect2*. The only difference here is that three visibility graphs are displayed. Window V contains the *Complete Visibility Graph*; window U contains the *Vertical Sweep Visibility*; and window T contains the horizontal sweep visibility.

[2] Horizontal and Vertical visibilities only.

C.3. Visibility Graph of Vertices of a General Simple Polygon. This section describes how to run the Visibility Graph of Vertices of a General Simple Polygon program. Its executable filename is *ordpoly*.

Once invoked, the program will bring up two windows, *V:Visibility Polygon Graph* on the left and *W:Polygons* on the right, and a panel, entitled *Polygon Visibility* below *W*. The window *W* is for drawing an input polygon, while the window *V* is for output of the corresponding visibility graph[3]. The *Polygon Visibility* Panel tells the program which function is to be performed next.

Normal program operation is in the following order:

- Press the *draw* button on the panel to begin drawing the input polygon. Then go to the *W* window and begin specifying the vertices of the polygon by moving the mouse to the desired positions and pressing the first mouse button. Be sure to maintain simplicity of the input polygon and specify the bertices in a clockwise order. Press the middle mouse button to indicate that all vertices have been input. As soon as the polygon is completed, window *V* will display the visibility graph of the polygon.
- The *edit* button allows the user to edit the polygon as described in the edit section above. Note that the random feature has not been implemented, nor is it possible to add, or delete the vertices of the polygon, once it is created. The re-orient option does not apply to this application.
- Press the *quit* option to exit the program.

C.4. Visibility Graph of Vertices of an Orthogonal Simple Polygon. This section describes how to run the Visibility Graph of Vertices of an Orthogonal Simple Polygon program. Its executable filename is *orthpoly*.

Once invoked, the program will bring up two windows, *Orthogonal Polygon Visibility Graph* on the left and *W:Polygons* on the right, and a panel, *entitled Options* below *W*. The window *W* is for drawing an input polygon, while the *Orthogonal Polygon Visibility Graph* window is for output of the corresponding visibility graph[4]. The *Options Panel* tells the program which function is to be performed next.

Normal program operation is in the following order:

- Press the *draw* button on the panel in order to begin drawing the input polygon. Then go to the *W* window and begin specifying the vertices of the polygon by moving the mouse to the desired positions and pressing the first mouse button. The program will orient edges parallel to either the Horizontal or Vertical axis. Be sure to maintain simplicity of the input polygon. When all vertices are input, move the mouse pointer back to the first vertex, and use the first button to click on that point again. As soon as the polygon is completed, the other window will display the visibility graph of the polygon.
- The *edit* button allows the user to load or save a polygon to, or from, a file, and to clear the window. However, no polygon editting features are available.

[3] Interior, arbitrary directional visibilities.
[4] Interior, arbitrary directional visibilities.

- Press the *quit* option to exit the program.

C.5. Endpoint Visibility Graph of Disjoint Line Segments. This section describes how to run the Endpoint Visibility Graph of Disjoint Line Segments program. Its executable filename is *endpt2*.

Once invoked, the program will bring up two windows, *Endpoint Visibility Graph* on the left and *Segments* on the right, and a panel, entitled *Endpoint Visibility Graph* below the right window. The *Segments* window is for drawing of the input segments, while the *Endpoint Visibility Graph* window is for output of the corresponding visibility graph[5]. The *Endpoint Visibility Graph* panel tells the program which function is to be performed next.

Normal program operation is in the following order:

- Press the *draw* button on the panel in order to begin drawing the input segments. Then go to the *Segments* window and begin specifying the endpoints of the set of segments by moving the mouse to the desired positions and pressing the first mouse button. Press the third mouse button to indicate that all vertices have been input. As soon as the segments are completely input, the *Endpoint Visibility Graph* window will display the visibility graph of the endpoints of the segments.
- The *options* button gives access to all edit features as described in the earlier edit section.
- Press the *quit* option to exit the program.

C.6. Visibility Polygons of Endpoints of a Set of Line Segments. This section describes how to run the Visibility Polygons of Endpoints of a Set of Line Segments program. Its executable filename is *vpoly*.

Once invoked, the program will bring up two windows, *V:Endpoint Visibility Graph* on the left and *W:Segments* on the right, and a panel, entitled *Endpoint Visibility Graph* below the right window. The window *W* is for drawing of the input segments, while the window *V* is for output of the corresponding endpoint visibility graph and the visibility polygons. The *Endpoint Visibility Graph* panel tells the program which function is to be performed next.

Normal program operation is in the following order:

- Press the *draw* button on the panel in order to begin drawing the input segments. Then go to the *Segments* window and begin specifying the endpoints of the set of segments by moving the mouse to the desired positions and pressing the first mouse button. Press the third mouse button to indicate that all vertices have been input. As soon as the segments are completed, the *Endpoint Visibility Graph* window will display the visibility graph of the endpoints of the segments.
- Press the *spiral* button on the panel to invoke the spiral procedure. This procedure, as described in Keil and Wismath [2], spirals around all the vertices

[5] Arbitrary directional visibilities.

of the polygon, and computes the visibility polygon of each vertex. The user has the option of advancing one vertex at a time, or of allowing the program to run through all vertices without stopping. Pre-condition: If collinear points are detected, they are displayed in window W and the user is given the chance to edit appropriately. *Warning: Detecting and handling collinearity is as yet a problem (see Problems Encountered in section 2.6).*

- Press the *display* button to view the visibility polygons. Use the mouse to click on nodes of the endpoint visibility graph in window V to indicate which polygon is to be displayed. The window will revert to displaying the end-point visibility graph if the cursor position does not correspond to a recognized node. Return to the main menu by pressing the right-mouse-button. Pre-condition: Run *spiral* first.
- The *options* button gives access to all edit features as described in the edit section above.
- Press the *quit* option to exit the program.