ArrangePak, OrthoPak, and VisPak 2.0 *

M. Closson[†] H. Everett[‡] S. Gartshore[§] S. K. Wismath[¶]

Abstract

ArrangePak, VisPak 2.0 and OrthoPak are three packages of software for researchers working on arrangements of lines, visibility graphs of line segments and polygons, and orthogonal 3-d graph drawing. All three packages are written in C++, and use the LEDA library extensively.

1 Introduction

1.1 ArrangePak

ArrangePak is a collection of tools designed for researchers studying arrangements of lines and pseudo-lines. In the main window, an arrangement of lines can be manipulated with the mouse. A wiring diagram of the arrangment can be generated and a simple duality transform is provided. A particularly useful option allows the user to create a pseudo-line arrangement topologically equivalent to the line arrangement but in which small faces of the arrangement are enlarged, making the topology more visually apparent. The algorithm for producing this improved arrangement was developed by Everett and Wismath and is described in section 2.2.

1.2 OrthoPak

OrthoPak is a collection of routines for displaying graphs orthogonally in 3 dimensions. There are two categories of such implementations:

*The authors gratefully thank N.S.E.R.C. for financial assistance.

†Department of Mathematics and Computer Science, University of Lethbridge, Lethbridge, Canada.

‡Département d'Informatique, Université du Québec à Montréal, C. p. 8888, Succ. Centre-Ville, Montréal, Québec, H3P 3C8; everett@info.uqam.ca

§Department of Mathematics and Computer Science, University of Lethbridge, Lethbridge, Canada.

¶Department of Mathematics and Computer Science, University of Lethbridge, Lethbridge, Canada.

- arbitrary degree graphs - in which vertices are represented by orthogonal *boxes* and edges are routed, crossing-free, along integer grid lines. One, two, and three bend layouts are provided that implement the results of a paper by Biedl, Shermer, Whitesides, and Wismath [1].

- max degree 6 graphs - in which vertices are represented by (integer grid) points and are again routed crossing-free along grid lines. The construction implemented ensures a maximum of 7 bends per edge and a volume of $O(n^2)$. Refer to the paper by Closson, Gartshore, and Wismath[3] for details.

For each construction, the output is a VRML v2.0 world which can be viewed and examined with a VRML browser supporting VRML 2.0 syntax. Colouring is preserved in the VRML file so that edges and nodes can easily be identified.

1.3 VisPak 2.0

VisPak is a package of visibility routines for displaying the visibility graph of various sets of objects in the plane. It includes implementations of algorithms for computing the visibility polygons of the endpoints of line segments and currently contains ArrangePak as a separate menu item.

2 Detailed description of the Packages

In this section we provide a more detailed description of the three packages.

2.1 ArrangePak

This program generates an arrangement of n lines bounded by a circle. The lines, by definition, must cross each other and all intersection points are constrained to lie inside the bounding circle. Moving the lines, by dragging an endpoint about the circumference of the circle is provided. A special option allows the manipulation of lines while preserving the current topology.

Once invoked, this program will bring up two windows, *Pseudo Lines* on the left and *Lines* on the right, and a panel, entitled *Arrangement of Lines*, below the *Lines* window. The window *Lines* is used to modify the arrangement of lines. The *Pseudo Lines* window is used to display and interact with the set of pseudo-lines which is initially based upon the given arrangement of lines.

The following are the main functions:

- The *Lines* bar is used to select the number of lines to generate when the graph gets redrawn.

- The *Protect Mode* toggle is used to make it impossible to remove any existing faces or create new ones (i.e. it preserves the current topology).

- *Draw Random* creates a new arrangement with the selected number of lines.

- *Drag Points* allows you to move the endpoints about the circumference of the circle. The right mouse button must then be clicked to return to the menu. This function, however, doesn't allow you to move any endpoint past another, and if the *Protect Mode* is selected, it will not allow you to remove or create any new faces.

- *Induced Path* creates a pop-up window displaying a simple inducing n-gram which uses each line exacly once. This implementation is based on a result by Bose, Everett, and Wismath [2] - refer to the paper for details.

- *Pseudo-Lines* creates a pseudo-line representation of the current arrangement graph. The dual graph is created from the arrangement, and a new graph created from this dual. The nature of this lastly created graph is such that each point within a dual face is moved to the center of that dual's face. This operation can then be iterated to create a graph which tends to spread small faces of the arrangement.

  The Pseudo-line window has the following functions:

  - *Iterate* causes the most recently created graph to become the basis for the creation of a dual and an altered pseudo-line arrangement.
  - *PS Output* is used to output the current window to a PostScript file.
  - *Load/Save* is used to save and/or load the current graphs to and from files.
  - *Help* gives a limited help file, useful for quick reference.
  - *Return* returns you to the controls for the main section of this program.
  - Within this screen you may click the mouse buttons to toggle on and off the graphs as follows.
    - First (Left-most) - Main Arrangement Graph
    - Second (Middle) - Dual of Main Graph
    - Third (Right-most) - Modification of Main Graph Based Upon The Dual (a new pseudo-line arrangement).
  - Implementation

- * *Algorithm*: The set of pseudo-lines is generated by computing the set of intersection points of the set of lines. Next, the set of intersection points are linearly searched to create the line segments for the resulting graph.
  * *Data Structures*: The arrangement of lines is stored as a list of Line class. This Line class stores the radian values for the endpoints as opposed to points. The pseudo-lines are stored as a parameterized graph, as are its dual and the graph generated from the dual.
  * *Analysis*: There are $O(n^2)$ intersection points, thus generation of the pseudo-lines takes $O(n^2)$ time. From the pseudo-lines, the dual is created by creating a midpoint for each face. Next edges are added to join these midpoints by searching a list of Face-Edge relations.
  * *Problems Encountered*: There is a slight problem with the generation of the dual graph. In some setups, generally when the number of lines exceeds 25, the created graph is non-planar. This is potentially due to the nature of the storage of the points. They are stored as a pair of double precision floating point numbers. In intersection and correspondence calculations, therefore, an approximate point is selected. Another side effect is that three lines may appear to cross at the same point but the points of intersection are not equal.

- *Dual Transform* creates a point-line dual transform of the current arrangement of lines. Each line is of the form $y = mx + b$ and is mapped to a point $(-m, b)$.

- *Wiring Diagram* will create a pop-up window containing the wiring diagram of the current arrangement of lines. A left to right sweep is used to compute the crossings for this diagram.

- *Load/Save* is used to save and load arrangements of lines. It also contains an option to output the lines to a PostScript file.

- *Help* contains a limited description of the section as well as a quick reference for the commands.

- *Quit* returns to the main menu.

2.2 VisPak 2.0

This updated release of VisPak is an extention and enhancement of the original VisPak [5]. The major enhancements in this version are twofold. First it now includes the addition of a section to compute the full visibility graph of a set of line segments [6]. VisPak now also contains a section to manipulate arrangements of lines. This section is also separately available as *ArrangePak* and has been described in the previous section. Also included in this release is a online help section and the ability to output to a postscript file. All major functions now have a help file associated with them for quick reference.

2.2.1 Full Visibility Graph

This program takes as input a set of disjoint line segments. The resulting visibility graph represents each line by a vertex placed at its midpoint and draws an edge between two vertices if their associated line segments are anywhere visible. The program takes input from the mouse, using the left button to indicate the endpoint of line segments, and the right mouse button to exit.

- *Algorithm*[6]: This full visibility graph is determined by first adding all edges of the *end-point visibility graph*. Next the visibility polygons are created by utilizing the *Topological Algorithm* discussed in the original VisPak documentation.

  Then, for each endpoint in the graph, the following rotational sweep algorithm is carried out:

  $i := 1$

  $a := \max_j\{\text{angle}(v_1 q v_j) < \pi\}$

  $b := \min_j\{\text{angle}(v_1 q v_j) > \pi\}$

  WHILE (angle($v_1 q v_1$) $< \pi$) DO

      report *segment*(v_i) visible to *segment*(v_b)

      $i := i + 1$

      WHILE (angle($v_i q v_b$) $< \pi$) DO

          $a := b + 1 (\text{mod } m)$

          $b := b + 2 (\text{mod } m)$

          report *segment*(v_i) visible to *segment*(v_b)

  Refer to the paper by Wismath [6] for a more detailed account of the algorithm.

- *Data Structures*: The data structures used to store the visibility polygons in the *Spiral Algorithm* and *Topological Algorithm* sections are again used.

2.2.2 Program Features

This section describes how to run the Full Visibility portion of VisPak.

Once invoked the program will bring up two windows, *Visibility Graph* on the left side, which will display the endpoint and full visibility graphs, and the *Segments* window which is used to input the line segments. The *Full Visibility Graph* menu is used to perform operations in the program.

The functions of the buttons are:

- *Draw* allows the input of line segments. They are entered by clicking the left mouse button, and clicking the right mouse button when finished entering lines. The *Visibility Graph* window will show the endpoint visibility of the set of segments.
- *Display* will calculate the visibility polygons for the given set of segments according to the *Topological Algorithm* discussed in the original VisPak manual.
- *Output PS* will pop-up a box to output the full visibility graph to a postscript file.
- *Options* is used to save and load the graph, to edit the graph, and to output the set of line segments and their endpoint visibility graph to a PostScript file.
- *Help* provides a command quick reference as well as a brief description of the function of this section of the program.
- *Quit* returns to the main menu.

2.3 OrthoPak

OrthoPak was designed as a tool to help researchers develop and implement algorithms to layout graphs orthogonally in 3D space. All of the algorithms currently supported were developed by the authors. The principle aim is to allow the writers to visualize, analyse and optimize these algorithms. OrthoPak borrows heavily from LEDA, its GraphEditor and many of the underlying data structures were taken from this package.

2.3.1 Program Functions

When OrthoPak is run, the user is faced with the LEDA GraphEditor. This will allow for the input, loading, or generation of a graph. The additional graph generators, accessable in the *Graph* menu under *Create*, are:

- *Generate a Binary Tree* which prompts for a number of nodes and will then generate a binary tree containing that number of nodes.
- *Generate Random Max Degree 6* which prompts for both a number of nodes and a number of edges to create. This then generates a graph with the specified parameters in a random fashion.

These, and any other, graphs can then be saved or output. The output functions added to this program, under the *3D Ortho Layouts* menu, are:

- *Save Wismath Graph* which will output the current graph, with maximum degree 6, to a VRML world. This construction represents work in progress and is tentatively referred to as the "spiralling columns" algorithm. A separate technical report will describe it in further detail [3].
- *Convert Graph To 1 Bend* will output the graph to a VRML world using the one-bend construction detailed in the paper by Biedl, Shermer, Whitesides and Wismath [1]
- *Convert Graph To 2 Bend* will output the graph to a VRML world using the two-bend algorithm of [1].
- *Convert Graph To 3 Bend* will output the graph to a VRML world using the three-bend algorithm of [1]. This construction is asymptotically optimal in terms of volume for complete graphs.
- *Max Deg. 6 Tree, 0 Bends* generates a tree representation of the current graph and outputs it to a VRML world using the *Closson, Gartshore 0-Bend Max. Degree 6 Tree Generation Algorithm* to be described separately.

3 Distribution

ArrangePak, OrthoPak and VisPak 2.0 are available free of charge for teaching and research purposes. Binary versions are available for Sparc workstations running Solaris 2.x and for x86 machines running Linux. Source code is also available for porting to other platforms. Please e-mail us if you use the software or discover bugs. Downloading can be performed over the web at:

`http://www.cs.uleth.ca/~wismath/packages`

4 Acknowledgments

Several researchers have tested this software and made suggestions for its improvement. Special thanks go to Mark Keil, and Jit Bose.

References

[1] T. Biedl, T. Shermer, S. Whitesides, S. K. Wismath, *Orthogonal 3-D Graph Drawing*, Submitted to *J. of Graph Theory and Algorithms.*

[2] P. Bose, H. Everett, S. Wismath; Properties of Arrangement Graphs, submitted to *J. of Graph Theory and Algorithms.*

[3] M. Closson, S. Gartshore, S. Wismath, *Fully Dynamic 3-D Orthogonal Graph Drawing*, U. of Lethbridge technical report, to appear.

[4] G. Di Battista, P. Eades, R. Tamassia, I. Tollis, *Graph Drawing*, Prentice Hall Publishers, 1999.

[5] L. Jackson, H. Pinto, S. Wismath, *VisPak: A Package of Visibility Algorithms Written in Leda*, U. of Lethbridge technical report, CS-95-1, 1995.

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