A GRAPH DRAWING ALGORITHM FOR THE GAME OF SPROUTS

Wayne Brown and Leemon Baird
Department of Computer Science, The United States Air Force Academy, USAF Academy, CO 80840-6208, USA

Abstract - *A graph drawing algorithm for the Game of Sprouts is presented. The algorithm guarantees that the polylines that connect graph nodes are drawn smoothly and that they maintain reasonable distance from other graph polylines. Vertices of the graph are moved using a combination of repulsive forces and smoothing forces. The repulsive forces come from all other visible graph nodes and visible polyline line segments. The smoothing forces are calculated from neighboring vertices along a polyline. A Sprouts player is not allowed to draw new polylines that cross any existing polyline, and the algorithm prevents edge crossings as the graph is transformed. The distinctive features of this algorithm is the use of smoothing forces instead of traditional spring forces, and the use of line segments as repulsive elements instead of vertices.*

Keywords: Graph drawing, polyline smoothing, Sprouts

1 Introduction

The Game of Sprouts [1], [2] is played by two opponents who take turns connecting two free nodes in a graph with a curved line, called a polyline, that does not cross any existing line in the graph. A free node is any node that has less than three lines connected to it. When a player connects two nodes with a polyline, a new node is created in the middle of the polyline. A player wins when they connect two free nodes and their opponent cannot on the succeeding turn. During the implementation of this game on a Personal Digital Assistant (PDA), an algorithm is needed to spread apart the polylines sketched by a user to allow space for later moves and, at the same time, to keep the polylines smooth and the graph visually pleasing. This paper presents a modified "force directed" graph drawing algorithm that works well for the Game of Sprouts. A traditional "force directed" algorithm produces geometric distributions of a graph's polylines, while our algorithm produces a uniform distribution of the polylines which provides a better visual graph representation for playing the Game of Sprouts.

2 Previous Work

Graph drawing algorithms can be categorized based on the type of drawings they produce (e.g., planar drawings, straight line drawings, grid drawings, etc.) [3], [4]. This paper's algorithm is concerned with polyline graph drawings, where each edge of a graph is represented by a sequence of connected line segments. Two aesthetics of a "nice" polyline graph drawing are typically taken to be a minimal number of edge crossings in conjunction with a minimum number of bends per polyline. To meet these goals, previous researchers typically project polyline vertices onto a grid or curve [5], [6]. These aesthetics do not apply to our work because edge crossings are disallowed during a Sprouts' game-play. Therefore, our algorithm must prevent edge-crossings from happening, but it does not have to remove existing edge-crossings. In addition, in place of a minimal number of polyline bends, our algorithm smoothes polylines without regard for the polylines total length or number of bends.

In relationship to other types of graph drawings, the type most closely related to polyline graph drawings would be straight-line graph drawings, where each graph edge is represented by a single straight line segment. Our algorithm is partially derived from previous work in this area by Eades [7], Kamada and Kawai [8], Fruchterman and Reingold [9], and Davidson and Harel [10]. In these works, a graph drawing is created using an energy function that contains two opposing forces: "magnetic" forces that repulse vertices and "spring" forces that attract vertices. Our algorithm uses only repulsive forces but adds a polyline smoothing algorithm to maintain smooth polylines between nodes. It must be noted that our algorithm is solving a much simpler problem than the previously mentioned work. Their algorithms attempt to solve the general problem of graph drawing where the initial state of the graph is random and the nature of the graph is unknown. Our algorithm attempts to solve a simpler problem because the construction of our graph problems are constrained during game-play and we are not concerned with aesthetic properties such as "total edge length," "uniform edge length," or symmetry.

3 The Graph Drawing Manipulation Algorithm

The Game of Sprouts builds a graph consisting of nodes connected by polylines. A polyline is a sequence of points connected by straight line segments. We collectively call all nodes and points vertices. These terms are shown graphically in Fig. 1.

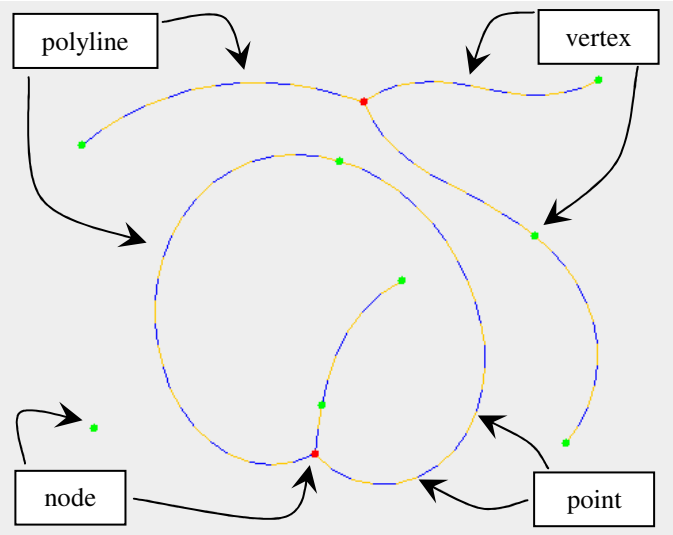

Fig. 1. A screen shot of a Sprouts game. In normal game mode, the polylines would be a single color.

A graph drawing for a Sprouts game is initially composed of three or more unconnected nodes and a single, four-segment polyline that encloses the rectangular game board area. (The polyline that surrounds the game board is invisible to the user.) On each turn of the game, one polyline and one node is added to the graph. As a player sketches out a new polyline, the game software prevents the new polyline from crossing any existing polylines in the graph. Our algorithm manipulates the graph between player turns to accomplish three objectives: 1) to spread the polylines apart to make it easier for a user to draw new polylines on later moves, 2) to smooth the polylines to make the graph less confusing and more visually pleasing, and 3) to guarantee the integrity of the graph - that no polylines intersect. The algorithm attempts to update the graph 30 times per second and moves each vertex a maximum of one pixel per iteration. The basic algorithm is:

```
For each line segment:
   Splitting:
      If a line segment is longer
      than some threshold, split the
      line segment into two separate
      line segments of equal length
      by inserting a new point.
   Joining:
      If two consecutive line
      segments formed by consecutive
      points, v1, v2 and v3, have a
      length between v1 and v3 that
      is below some threshold, and
      there are no other vertices of
      the graph inside the triangle
      formed by v1v2v3, then form a
      single line segment between
      points v1 and v3 and remove the
      two line segments and point v2.
For each vertex in the graph:
   To Spread:
      Assume that every other visible
      node of degree less than two
      and every visible line segment
      exerts a repulsive force on
      this vertex that is
      proportional to one over the
      distance between them cubed and
      calculate a vector that will
      move the vertex towards a
      location where all the forces
      sum to zero.
   To Smooth:
      1) If the vertex is of degree
      two, use a smoothing algorithm
      to calculate a new location for
      the point or node that would
      cause its polyline to become
      smoother.
      2) If the vertex is a node of
      degree greater than two,
      calculate a vector that would
      move the vertex to an average
      of its three connected points.
   Move this vertex in the direction
   of a weighted sum of the unit
   vectors for spreading and
   smoothing.
```

Solving for the repulsive forces on vertices can be done using numerical techniques, (e.g., Runga-Kuta) but this is extremely computationally expensive. Our algorithm uses an iterative approach that calculates only the direction a vertex must move to reach equilibrium and not its exact location for equilibrium.

3.1 Implementation Details

Scale Factors. All force directed methods for graph drawing use scale factors to weight the forces appropriately for a given graph frame. If the graph frame changes size, the weights typically need adjustments as well. In the following descriptions all vertices are assumed to be inside a graph frame with coordinate values in the range 0 to 812, which guarantees that integer overflow will not occur in our smoothing calculations.

The Repulsion Force from a Node. The repulsion force of a node (node) upon another vertex or node (v) is calculated by forming a normalized vector between their locations and then multiplying the vector by 1/d^3, where *d* is the distance between the vertices (Eqs. 1-3). Other

researchers typically use 1/d^2 [9], but using the distance cubed produces good results and avoids a square root calculation.

$$dx = v.x - node.x$$
$$dy = v.y - node.y$$
$$length = (dx^2 + dy^2)^0.5 \quad (1)$$

$$forcex = Kn * (dx / length) * 1/length^3 = Kn * dx / (dx^2 + dy^2)^2 \quad (2)$$
$$forcey = Kn * (dy / length) * 1/length^3 = Kn * dy / (dx^2 + dy^2)^2 \quad (3)$$

A good weight, Kn, for a node repulsion force is 600. This value was determined through experimentation and produced our desired outcomes. The weight has no physical meaning.

The Repulsion Forces from a Line Segment. The repulsion force of a line segment defined by two vertices, v2 and v3, upon a vertex, v1, is calculated by summing all the repulsion vectors for every point along the segment. Let **P** represent any point along the line segment, which is defined in parametric form, as shown in Eq. 4. Therefore, the sum of all repulsion vectors for every point on the line segment is calculated with the integral shown in Eq. 5. Here the force is calculated as 1/d^2 because the solution using 1/d^3 requires an arctan calculation that we wanted to avoid.

$$P = v2 + (v3-v2)*t, \quad 0 <= t <= 1 \quad (4)$$

$$f = \int_0^1 \frac{v_1 - P}{|v_1 P|^3} dt \quad (5)$$

$$a = v1.x - v2.x \qquad b = v3.x - v2.x$$
$$c = v1.y - v2.y \qquad d = v3.y - v2.y$$
$$g = (a^2+c^2)^0.5$$
$$h = (d^2+b^2-2ab-2cd+a^2+c^2)^0.5$$

$$f_x = \int_0^1 \frac{a - bt}{\sqrt{(a-bt)^2 + (c-dt)^2}^3} dt = \frac{-\left(\frac{c-d}{h} - \frac{c}{g}\right)}{a*d - b*c}$$
$$f_y = \int_0^1 \frac{c - dt}{\sqrt{(a-bt)^2 + (c-dt)^2}^3} dt = \frac{\left(\frac{-b+a}{h} - \frac{a}{g}\right)}{a*d - b*c} \quad (6)$$

Equation 6 is a closed form solution to Eq. 5. Since the integral is over dt, the magnitude of the force vector calculated by Eq. 6 is ignored, but the direction of the force vector provides the direction of the combined repulsive forces from the line segment. To calculate the location of this combined force, a ray is formed that starts at v1 and has the opposite direction of the force vector. The intersection of this ray with the line segment provides a location from which the repulsive forces of the line segment are concentrated. The repulsive force of a line segment is then calculated from this point using Eq. 2-3. In addition, the repulsion vector is weighted by the length of the line segment.

The orientation of the repulsion force vector with respect to the vertex is also taken into account when calculating the magnitude of the repulsion force. Eqs. 7-10 show these modifications to the force vector's magnitude.

sin1 = angle between line segment and forceVector (7)

sin2 = angle between direction of polyline and forceVector (8)

(for a node) Repulsion = Kv * LengthLineSegment * forceVector * sin1 (9)

(for a point) Repulsion = Kv * LengthLineSegment * forceVector * max(sin1,sin2) (10)

A good weight, Kv, for a line segment repulsion force could not be found that worked well for all line segments. This was due to the fact that the polyline segments that surround the graph frame are always much longer than the line segments of the graph polylines. Therefore, Kv is 1 for the frame polyline segments and 5 for the graph polyline segments. These values were determined through experimentation and produced our desired outcomes. The weights have no physical meaning. Fig 5 shows a graphical representation of the forces acting on a single vertex.

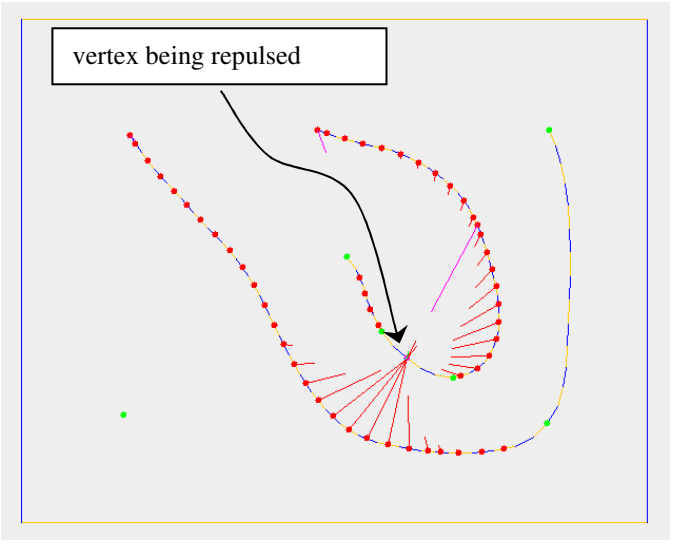

Fig. 5. A visual example of the repulsion forces applied to move a vertex.

Polyline smoothing. The algorithm for smoothing a polyline is described in [11].

Combining the Spreading and Smoothing Forces. The spreading and smoothing algorithms each provide a

vector that indicates which direction a vertex should move to produce an improved graph drawing. Each vector is normalized and a weighted sum of the normalized vectors is used to create a movement vector for a vertex. If the length of either the spreading or smoothing vectors becomes less than a minimum distance, that respective motion vector is ignored. Our experiments found that a 60% spreading force and a 40% smoothing force worked well in most cases. Keeping the spreading forces above 50% prevents polyline crossings. Experimentation also showed that it is desirable to dynamically change the percentage contribution of the forces over time. Initially the spreading forces should be high to separate a graph's polylines, and then smoothing should become the dominate force to improve the visual qualities of the graph. We used a minimum length of one pixel to determine when a force vector should be ignored.

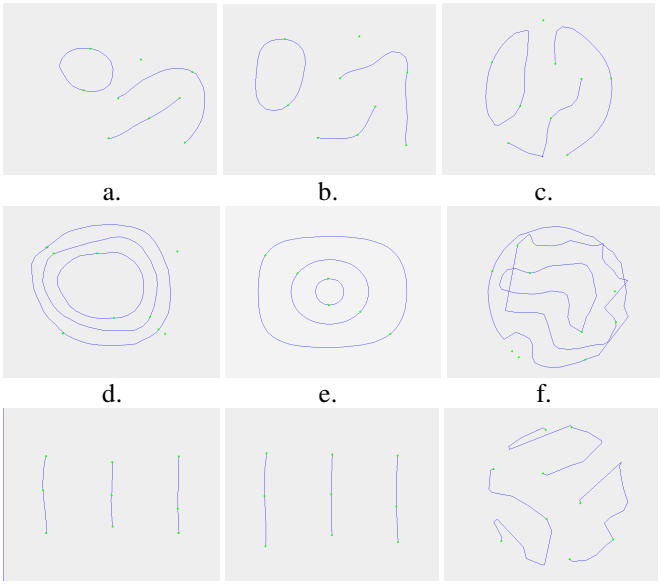

Fig. 6. The left column contains initial graph configurations. The middle column shows our algorithm's modifications. The right column shows a traditional, force-directed algorithm's modifications.

4 Implementation and Results

Our algorithm has a run-time complexity of O(n^3), where *n* is the number of graph vertices. If a graph representation stores topology information, then the number of line segments that must be tested for visibility can be reduced with an associated reduction in the algorithm's run-time complexity. Our algorithm was implemented in Java and tested on an Intel Pentium, 1.86GHz, Gateway laptop computer. For small graphs with approximately 10 vertices, the algorithm could update the graph 30 times per second; for 100 vertices, updates dropped to 10 times per second; and for 300 vertices updates dropped to 1 time per second. The test bed code has not been optimized and a more robust and faster version of the code seems feasible.

A comparison of our algorithm to a traditional, force-directed algorithm is not a straightforward task since most force-directed algorithms have been designed for straight-line graph drawings, not polyline graph drawings. However, a comparison to traditional force-directed algorithms can be made by treating every vertex in our graph as a distinct graph node. Fig. 6 presents some comparisons and demonstrates how our algorithm maintains the integrity of graph polylines and produces good spreading and smoothing of the polylines. (The force-directed algorithm used for the comparisons in Fig. 6 was implemented from equations in [4], page 307-309, and used a constant of 1.0 for the repulsive forces and a constant of 30.0 for the attractive forces. The constants were chosen to make the best possible comparison graphs - in the eyes of the authors.)

5 Conclusions

Our algorithm spreads and smoothes polyline graph drawings in beneficial ways not found in any other published algorithms. It enhances the game play of a computerized implementation of Sprouts and potentially has application to other types of graph drawing problems.

6 References

[1] http://en.wikipedia.org/wiki/Sprouts_game

[2] http://www.geocities.com/chessdp, "World Game of Sprouts Association"

[3] Nishizeki, T., Rahman, M. S., **Planar Graph Drawing**, World Scientific Publishing Co, ISBN 981-256-033-5.

[4] Battista, G., Eades, P., Tamassia, R., Tollis, I. G., **Graph Drawing: Algorithms for the Visualization of Graphs**, Prentice-Hall, Inc., 1998, ISBN 0-13-301615-3.

[5] Bonichon, N., Le Sa¨ec, B., Mosbah, M., *Optimal area algorithm for planar polyline drawings*, in 28th International Workshop, Graph - Theoretic Concepts in Computer Science (WG), volume 2573 of LNCS, pages 35–46. Springer, 2002.

[6] Giacomo E., Didimo, W., Liotta, G., Wismath, S., *Curve-constrained drawings of planar graphs*, Computational Geometry: Theory and Applications, v.30 n.1, p.1-23, January 2005.

[7] Eades, P., *A heuristic for graph drawing*, Congressus Nutnerantiunt, 42, 149–160 (1984).

[8] Kamada, T., Kawai, S., *An algorithm for drawing general undirected graphs*, Information Processing Letters, 31, (l), 7–15 (1989).

[9] Fruchterman, T. M. and Reingold, E. M., *Graph drawing by force-directed placement*, Softw. Pract. Exper. 21, 11 (Nov. 1991), 1129-1164. DOI= http://dx.doi.org/10.1002/ spe.4380211102.

[10] Davidson, R. and Harel, D., *Drawing graphs nicely using simulated annealing*, ACM Trans. Graph. 15, 4 (Oct. 1996), 301-331. DOI= http://doi.acm.org/10.1145/234535.234538.

[11] Brown, W., Baird, L., *A Non-Trigonometric, Area Preserving, Polyline Smoothing Algorithm*, "Consortium for Computing Sciences in Colleges Mid-South Conference 2008", Arkansas Tech University, Russellville, Arkansas, 4 April - 5 April, 2008.