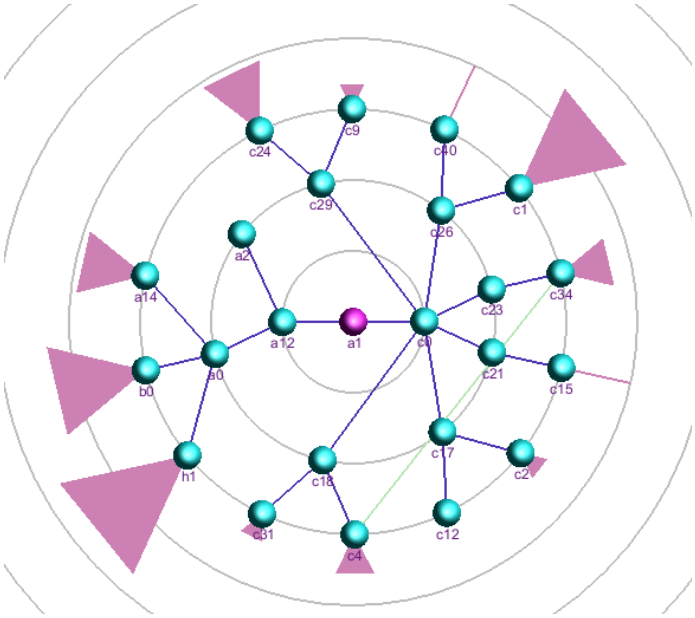
Scaling Up Radial Graph Layouts



Cody Robson

University of British Columbia

Abstract

The well-known radial graph layout technique has plenty of advantages for graph visualization and graph exploration, but is quite limited in the size of graphs it can display effectively because the layout is inherently global and allows far away or out of view nodes to distort the focus region. The primary contribution of this project is to introduce sub tree clustering to allow the layout algorithm to run locally and provide a visually pleasing focus region regardless of the overall graph size. This solution builds upon an existing radial graph visualization technique published in Yee et. al.'s *Animated Exploration of Graphs with Radial Layout* published in Proc. InfoVis 2001. In addition to clustering, other common information visualization techniques are added to Yee et. al.'s solution to further aid with graph navigation and visualization as the graph size increases.

Keywords: Graph exploration, radial layout.

1 Introduction

The focus of this project is to provide extensions that aid in the scaling of the radial graph layout algorithm. Radial graph layouts emphasize shortest path distances from a central focus node to all other nodes in the graph by placing nodes on layout rings emanating from the center of the graph. First, the layout algorithm performs a breadth first search from a primary or *focus* node to all other nodes in the graph (which must be a single connected component). Using this tree, it first places the focus node, now the root of the tree, in the center of the canvas, at the *zero ring*. It then partitions the radians of the first ring, some determined radius away from the center, amongst the root's children based on the span, or number of leaf nodes, in each of their sub trees. The algorithm then recursively partitions each of the children node's slice of the next layout ring amongst its children. This encoding allows a user to determine path distance from any node to the focus just by observing which ring a given node is placed on. Furthermore, sub trees will ideally be spread out as well as possible since their allocation space is proportional to their span. Additionally, sub trees and their edges are guaranteed to not intersect one another because each child node isn't allowed out of its allocated range of radians.

This type of layout is ideal for data for which the shortest path between two nodes is the most important, and other paths are relevant only for context. Datasets for which this is advantagous are things like network data, where connections would be routed through the shortest path, and alternative paths are meaningful only in the event of a log jam or network outage. In order to easily judge distances between two non-focus nodes the user must select one of them to be the new focus node and the layout algorithm must be rerun.

As will be described in detail later, this layout can scale well under certain circumstances but gives very poor layouts for focus areas in tree-like or sparsely connected sub regions of large graphs. Ideally, this layout technique should work on a local neighbourhood around the focus node and not be negatively effected by large number of nodes far away from the focus region. This is the primary intuition behind sub tree clustering, a

*cjrobson@cs.ubc.ca

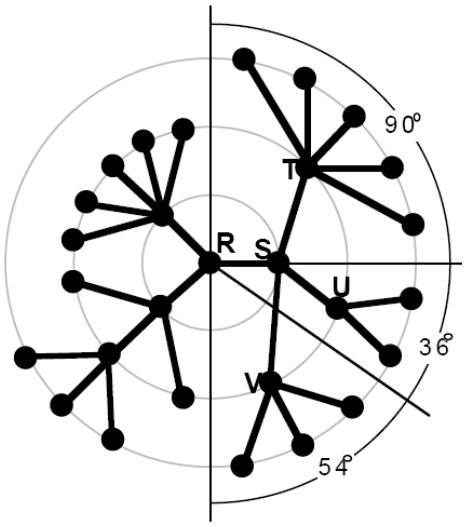


Figure 1: A radial placement diagram from [10]. Notice sub tree S is given half of the available canvas, which it divides up amongst its children nodes based on the span of their sub trees.

technique that hopes to allow radial graph layout to scale beyond small well-connected graphs.

2 Previous Work

Radial graph layout is only one of many tree-based layout algorithms. Some tree placement algorithms utilize hyperbolic geometry in order to handle large tree or graph sizes [5][7]. These types of methods do not recompute the layout when the user changes their focus, and instead allows the user to pan in hyperbolic space. NicheWorks [10] is a large scale graph layout technique that was derived from a radial graph layout, and gives excellent details about the calculation of the original radial graph layout properties. The final implementation is not a direct application of radial graph layout but more of an evolution.

Choosing a layout algorithm is very much dependant on a perticular dataset, and if my project was specific to a dataset I would have most likely chosen more than one different layout techniques to compare and contrast with my data. However, I have decided to take a technique-driven approach and start with a recent radial graph layout solution and extend it to deal with larger datasets.

A recent notable radial graph layout solution is Yee et. al.'s *Animated Exploration of Graphs with Radial Layout* [12] which promotes the use of animated focus node transitions, using interpolation of polar node coordinates and traditional animation techniques. They also apply some useful layout constraints to allow for minimal graph layout changes between focus transitions. As mentioned in their paper, their solution requires a node aggregation technique of some kind if it is to do a good job of visualizing large graphs. My solution intends to provide a possible node aggregation technique, which I will refer to as sub tree clustering, to allow radial graph layouts to apply to arbitrary sized graphs while maintaining the visually pleasing radial layout in the rings around the focus region.

3 Description of Problem

The radial graph layout utilizes the entire graph when determining even the first of the root node's children's available space. For very well connected graphs or mesh-like data, the sub trees of the root's first few generations of children will likely be quite balanced, leading to a very pleasing graph layout. However, in the event that much of the rest of the graph is only accessible through one of the the root's children, that child's allocated space is going to comprise of the majority of the available space,

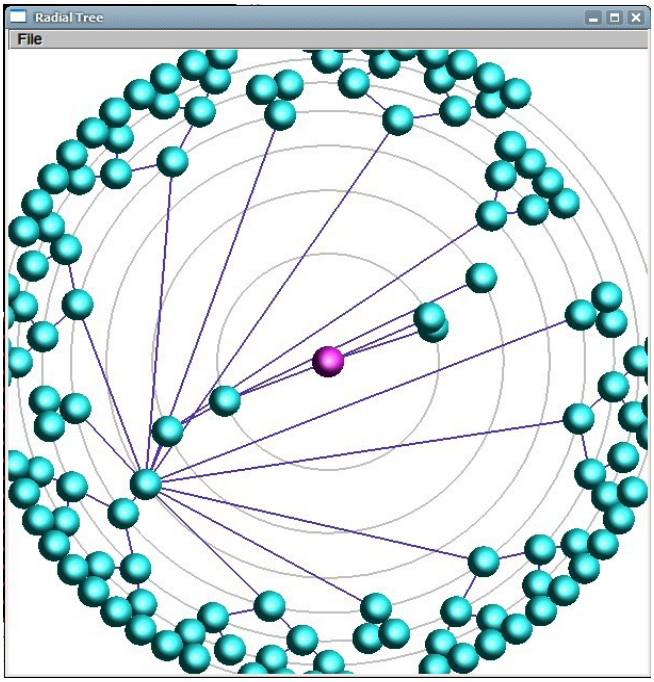


Figure 2: Traditional radial graph layout of a tree near a leaf node. Notice all the nodes on the first three rings, the focus region, are constrained to a straight line, giving the user a poor layout for the region that their focus node is implying they care most about.

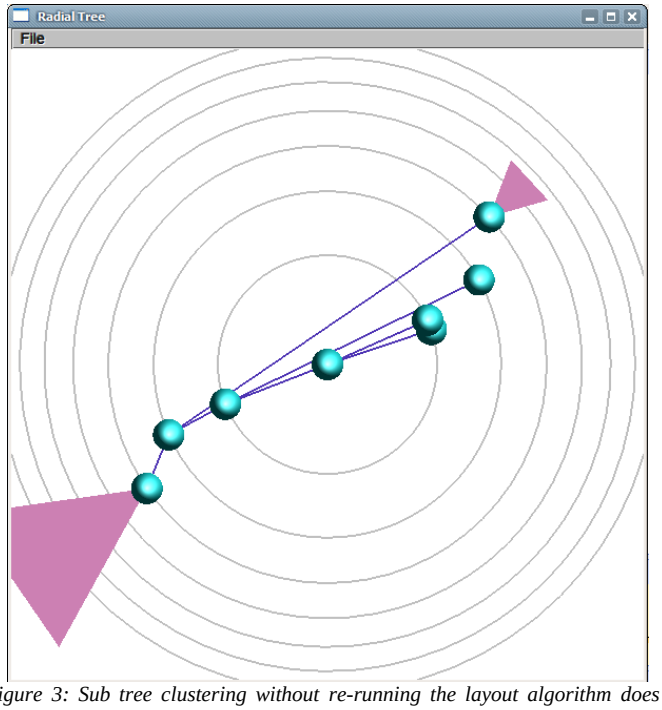


Figure 3: Sub tree clustering without re-running the layout algorithm does little to prove the poor placement of the first few generations of children nodes.

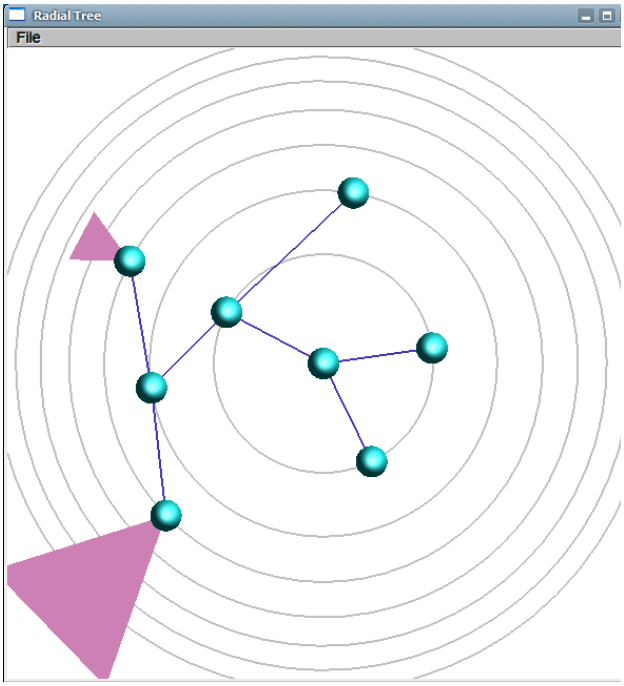


Figure 4: Once the layout algorithm is re-run with knowledge of the clustering level, the placement of nodes along the first few rings is much more visually pleasing. This is a simplistic example, a typical user would most likely view more than 3 rings before invoking sub tree clustering.

smashing the rest of the root's children into a very small sliver of radians in the first few rings and allowing children of the aforementioned bottleneck node to sweep across the rest of the graph and consume the majority of canvas space, regardless if they are very far away from the root via traversal (figure 2).

This phenomena is best visualized in a small tree, which is not itself a contrived example because many generic graphs will have subregions which are trees. By having the radial graph layout algorithm run on tree data and traversing one's way to a leaf node or node a few jumps away from a leaf node, this unpleasant, unbalanced partition effect becomes apparent (figure 2). Because the root's children that lead to the tree leaves will have a very small span, they are allocated a minuscule radial slice in comparison to the root's bottleneck child, which will lead to the entire rest of the graph. In addition to trees, a sub graph with one edge to another sub graph will invoke this same unbalanced partitioning. Because the breadth first search involved in the layout builds a tree out of the data, only one sub tree connected to the focus node will be able to claim the entire sub graph because it was the node with the shortest distance to that single isthmus-like edge.

The solution must in some way allow for the layout of the focus region, or the first few rings around the centre, to be laid out independently from the rest of the graph, most of which will not even be visible to the user (figure 3).

Beyond poor layouts, as graph size increases, a single-jump transition from one node to another becomes harder and harder for a user to visually grasp. Animation techniques utilized in Yee et. al.'s radial graph visualization work very well for nodes that are in the user's field of view, or even just beyond it. Once the new focus node becomes far outside the viewing window, however, layout constraints and slow-in/slow-out animation will not be enough support for the user to mentally parse the rapid flow of possibly thousands of nodes across the screen at one time. As proposed in [12], a large transition must be broken up into smaller user-friendly chunks that are easily to follow in succession.

4 Description of Solution

My program builds upon Yee et. al.'s program description, including the features their solution implemented to best support focus transitions as well my own extensions to aid with graph size scaling. The key feature of their solution that became the baseline for my program was the slow-in/slow-out polar coordinate interpolation of nodes between layouts. The general premise is that linear interpolation of Cartesian coordinates caused quite a bit of crossover among nodes during layout transitions. By interpolating polar coordinates, nodes and sub trees will glide along the layout rings and be less likely to cross paths with each other. Furthermore, traditional animation techniques like slow-in/ slow-out movements provide the user with useful motion cues, so that they are well situated to mentally process moving nodes as they begin to move faster. Their paper was quite convincing that these properties best allowed the user to keep track of their current location in the graph during a focus transition, to the point that I felt these features should be enabled permanently.

The two layout constraints talked about in [12] were also implemented. The first constraint enforces that a focus node's edge to its previous parent node is maintained in the new layout. This will reduce unnecessary and arbitrary rotations. The second constraint forces nodes to maintain the order of their children nodes. For tree graphs, this is not very hard to enforce, but in radial layouts, non-tree neighbour nodes become child nodes and vice versa. Keeping this constraint reduces the amount of child node travel from one layout to another. Children will still be forced to travel large distances when the shortest path from the focus to the child node becomes rerouted through another parent node.

Among the features of Yee et. al.'s implementation I decided to omit from my solution were dynamic node addition and varying node sizes. I felt these features were meant to compliment their file-sharing network data and had no implication on the radial layout technique itself.

Yee et. al.'s own proposed extensions proved to be the most beneficial of the new features added in my solution. First and foremost, node aggregation or, as I call it, sub tree clustering removes the single biggest obstacle in allowing radial graph layouts to scale up to graphs of hundreds or thousands of nodes. Since the graph layout is a tree, nodes at a given *clustering ring* can represent the entire sub tree of which that node is the root. More importantly, the radial node placement algorithm can take this into account and not factor in clustered sub trees when partitioning the available space (figures 2 and 3 illustrate the layout algorithm ignoring or considering clustering). This is the all-important migration from a global layout to a local layout needed to produce good results near the focus node as graph size increases.

The second proposed extension in [12] is intermediate node transition steps or a *transition series* for large focus transitions. This is a much needed feature, as it becomes increasingly harder to follow the transitions as the distance between the old and new focus nodes increases. Additionally, if the new focus node is currently omitted as part of a sub tree cluster, a single, large transition to it will be nearly impossible for the user to track. This transition series can either be specified by jump size or total number of jumps from beginning to end.

I included a few other visualization features to my solution to aid with graph size scaling and the core graph visualization in general. The inclusion of clustering made the need for fade-in fade-out support quite apparent, as nodes, edges, clusters, and labels materialize and vanish without warning. By linearly fading these elements as they cluster and uncluster, the user is better able to track which nodes became members of which clusters during a focus transition. Also, by making the amount of transparency a factor of the node's distance to the origin, the fade-in and fade-out rates coincide with the slow-in/slow-out animation, hopefully providing a similar visual cue.

Focus+context, the idea of introducing distortion to a visualization technique in order to emphasize a focus region and include surrounding data mainly to provide context for that focus region, is a very important principal of many Information Visualization solutions and is very easy to include in a radial layout. Normally, the distance between each layout ring is constant, but by instead making the ring's radii grow logarithmically, more space is given between the first few rings and rings around the peripheral become closer together. Being able to visualize the local neighbourhood around the focus is a large component of this layout technique, and this improves that aspect at the cost of pushing further away nodes closer together. However, this disadvantage is marginalized by the inclusion of clustering, which would omit the rendering of these far away nodes to begin with (figure 10).

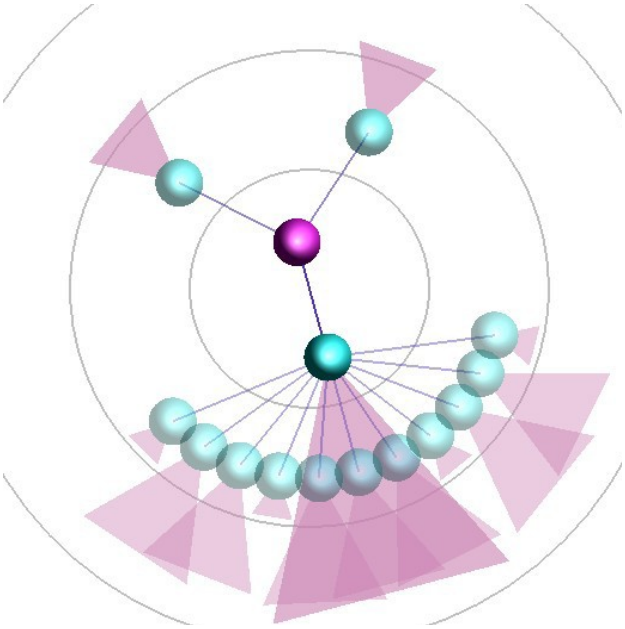


Figure 6: Nodes and clusters fade in and fade out as nodes interpolate between the cluster ring (in this case, the first ring) and the first ring beyond it.

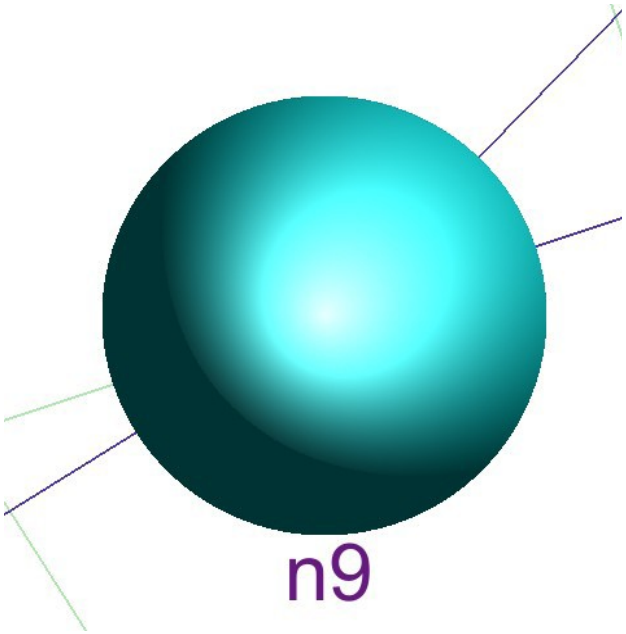


Figure 5: Aesthetics of node rendering. The imposter sphere will have a crisp circular edge regardless of zoom level. The anti-aliased FTGL text label also appears smooth.

Many InfoVis papers deal with the process of tying zoom to a pan movement. One such example that provides an excellent mathematical framing of this notion is [11]. The idea is that a system would have to pan less when it is zoomed out, in addition to allowing the user to see more of the dataset move slower and thus better keep track of their position in the dataset. Since this application does no true panning per se, one could characterize a change in focus as closely related to a pan, as the user is moving across the dataset, just not in a traditional moving-camera/static-world situation. Because of this a zoom-on-transition feature was added that zooms out an amount relative to the distance between the current and new focus nodes. The calculated zoom out amount is spread across the transition series so that each step slowly zooms out during the first half of the transition and zooms back in during the later half to return the user to their preferred zoom level.

The rest of the features are not visualization techniques but rather included to make the final result as visually pleasing as possible (figure 5). Node drawing has two available modes. One draws nodes as low-cost no-frills coloured squares with black outlines for programmable shader-impaired systems. The other, intended for modern hardware, uses an impostor sphere billboard shader to draw the nodes as circles with per-pixel Phong-esque shading. This allows the nodes to appear as perfect circles at all scales and still be rendered with only two triangles. The shader requires the lowest subset of shader abilities, Shader Model 1.0, so any card supporting the OpenGL Shading Language (GLSL) should be capable of its execution.

The other aesthetic feature is the inclusion of anti-aliased text labels for each of the nodes. These labels are placed just below each node, and a collision detection process runs a axis-aligned bounding-box check on each label to find intersections and hide labels of lesser priority (i.e. further away from the center) so that the user never has to decipher overlapping characters. Because the user is allowed to rotate the graph at any time, this bounding

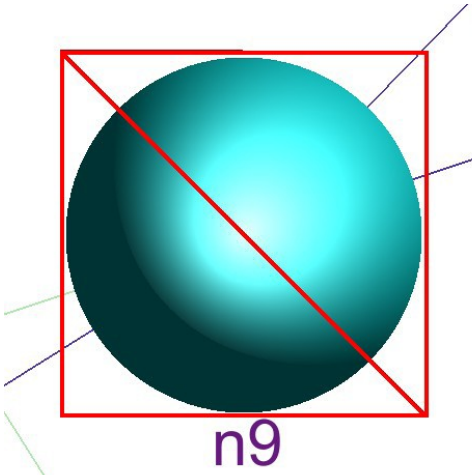


Figure 7: Red outlines illustrate the actual rendered geometry as opposed to the perceived impostor sphere.

box calculation has to be run many times and therefore it is best to utilize multithreading so that a decent frame rate can be maintained.

5 Third Party Software

I chose to do all the rendering in OpenGL[4] and its associated shading language, GLSL, because of its speed and flexibility. The imposter sphere shader is compatible with Shader Model 1.0 or newer video cards. I had done previous work with the FLTK [3] windowing tool kit and decided to use it again because of its light weight, C++ implementation, and easy to use wysiwyg editor. Both supported file formats are XML based (GraphML and the InfoVis 2003 Contest tree data files), and parsed with the IrrXML [2] library, the stand-alone XML-reader associated with the open-source Irrlicht game engine [1]. There exist a plethora of font rendering libraries for OpenGL, and I chose FTGL [6] for its support for nice-looking anti-aliased TrueType fonts and OpenGL display lists. All other aspects of the program were implemented from scratch and are described in the following section.

6 Implementation Details

When a node is selected to become the focus (both by the user and for initial layout), a breadth-first search (BFS) determines the new tree layout. Once the parent-child relationships are determined, the span of each node is computed and starting at the root, the radians of the circle are divided amongst the children nodes based on the span of their sub trees. Each node keeps track of their previous, current, and new ring value and orientation theta so that it can interpolate the two positions during the slow-in, slow-out animation function. The parent-child relationships must be determined with care. Each node keeps a list of its neighbours and the subset of neighbours deemed its children. In order to enforce the neighbour-ordering constraint, the order of these children must be consistent when children nodes transition to neighbour nodes and neighbour nodes become children, an effect of a new focus finding a new shortest path to a given node based on the initial BFS. This is done by examining the direction of the edges connecting the neighbouring nodes to the current node and ordering them counter-clockwise starting at the current node's parent edge (or former parent edge in the case of the parent-less focus node).

The animation function is called at each frame, and it is responsible for updating the positions of all the nodes as well as the current zoom level. It uses a quadratic velocity function multiplied by the time since the last frame to keep the transition times independent of frame rate. In general, transitions about a second in length seemed to be a good trade off between providing the user adequate motion queues and waiting too long for animations to complete. Since the movement of each node is independent of all other nodes, this process could be threaded in the event the number of nodes becomes exceedingly large. However, I found the single-threaded implementation adequate for graphs in the hundreds of nodes.

The zooming is done in the animation function and is treated exactly the same as a node interpolating two positions. The previous, current, and intended zoom levels are all tracked so that the user can always modify the intended zoom level and the zooming speed should smoothly change to accommodate the larger or smaller zoom distance. Zoom level is specified either by a user interface slider or by the mouse wheel.

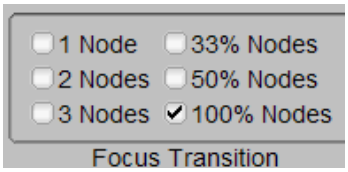


Figure 8: Available transition modes on the user interface.

If a transition path other than 100% is activated from the UI, the path of transitional nodes from the current focus to the new focus is saved as the transition series. When the animation function completes without updating any node's position, the layout algorithm is called for the next node on the transition series. If the *zoom on transitions* option is selected from the UI, an amount of unzooming is determined based on the current ring position of the new focus (the last node in the transition series). Each node of the transition series carries with it a new zoom level value, which starts at the current zoom level and increases to the predetermined unzooming amount for the centre nodes and decreases back down to the current zoom level for the last node.

Figure 9: The normal map of a sphere.

The nodes themselves have two rendering modes. The first is a simple coloured square with a black outline, done in two rendering passes. A custom shader could be written to do this in a single pass, but this mode was included to support shader-

incompatible environments. The second mode draws an impostor sphere with Phong-esque shading. The details of this approach were taken from QuteMol [9], a molecule rendering suite. The process involves sending four vertices to the vertex shader at the centre of the sphere. Each vertex's texture coordinates specify a corner of a billboard, and once each vertex is transformed into normalized device coordinates they are translated into their respective corners of the screen-aligned square containing the impostor sphere. In the fragment shader, pixels that fall outside the radius of the imposter sphere are discarded, giving the imposter a pixel-perfect circular boundary at all zoom levels (figure 7). Normals for the sphere are easily calculated because they can be solved given a pixel's distance to the impostor's centre (figure 9).

Unlike QuteMol, the impostor rendered in this program is actually a billboard (i.e. the depth of an actual sphere is not computed). This has a few implications. First, sphere-sphere intersections will appear as overlapping circles as opposed to intersecting three-dimensional shapes. I felt this was appropriate as the nodes are meant as data points and their actual geometry is arbitrary. Second, the fragment shader becomes much shorter, allowing for faster rendering. Third, the specular highlight component of the Phong shading model is a bit inaccurate, as the half-angle calculation requires the depth value at each pixel, hence I refer to the lighting as Phong-esque. I felt the increase in speed outweighs the disadvantages to using billboards for this type of 2D application, as the nodes themselves are simply meant to look nice at all scales and the subtleties of their geometry are quite insignificant.

The lines representing tree and non-tree edges as well as the layout rings are drawn with simple line strips. Their colours were selected to imply the tree edges (dark blue) are the most important, followed by the non-tree edges (faded green), and lastly the layout rings (light grey) which could have been omitted altogether but were included to make the focus+context slider's effects apparent. The rings have two spacing modes, one linear and one logarithmic, deemed the focus+context mode, as it allows for much more space between the first few rings and diminishing space between distant rings. The slider on the UI linearly interpolates between these settings.

Nodes beyond the UI-specified sub tree clustering ring are not drawn. Instead a line or triangle is placed at the node on the

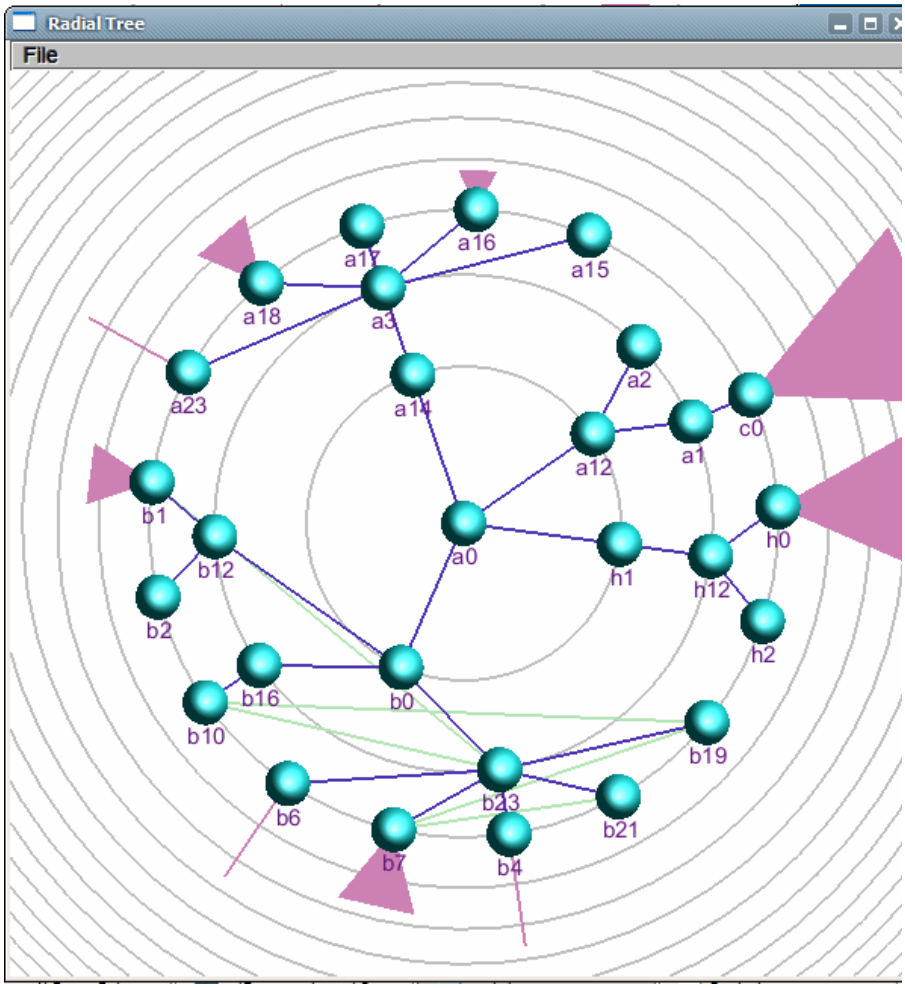


Figure 10: Results of radial graph layout with sub tree clustering. Tree edges are drawn in blue, non-tree edges in green, and the layout rings in grey. Clustering triangles are proportional to the sub trees they represent, or are drawn as lines for simple node chains. The focus+context setting places the layout rings logarithmically away from the centre, but nodes become clustered before getting too close together.

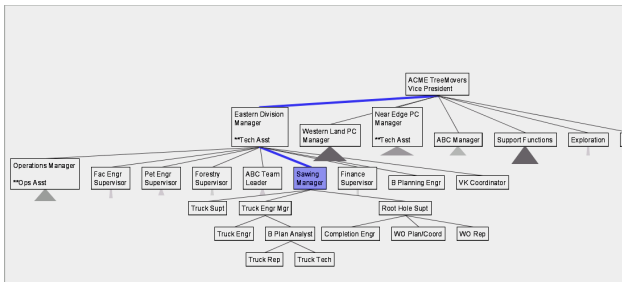


Figure 11: Screen shot found in [8] of the SpaceTree system. The triangular previews were the primary inspiration for the sub tree clustering triangles.

clustering ring and is meant to inform the user how large the sub tree being hidden is relative to other hidden sub trees. If the sub tree is a simple chain of nodes, a line is drawn instead. The size of the triangle scales logarithmically with the span of the sub tree, and transitions will very commonly combine two or more clustered sub trees. A linear relationship between the span of the sub tree and the triangle scale would result in very, very large triangles that would be most certainly be partially occluded by the window border and make visual comparisons impossible. The style of these sub tree clusters was inspired by the triangular previews of the SpaceTree [8] system (figure 11).

Despite the orthographic projection of the scene, the depth and ordering of each object had to be chosen with care for proper alpha blending. Clusters, nodes, and lines fade linearly between the clustering ring and the first ring beyond it. Because the velocity of anything transitioning between those two rings will slow-in slow-out, the fading effect inherits this property automatically.

7 Results

The largest improvement to the radial graph layout technique is the inclusion of sub tree clustering. Enabling the layout algorithm to consider only nodes in a local neighbourhood at the focus, the resulting graphs are no longer constrained to tiny slices of the layout rings. I believe this removes the hurdle that unbalanced tree-like sections imposed on the graph that made radial layouts incapable of scaling to graphs of larger than a few dozen nodes. Because the current layout is now local, the size of the underlying graph becomes only a problem of computational graph-traversal cost and not one of human perception.

While the the implementation of Yee et. al.'s solution without further extensions could possibly scale to large graphs, they would have to be well connected not contain any trees of more than a dozen or so nodes or sub graphs connected to each other by only a few edges. Those scenarios produce unbalanced trees during the layout process making the focus region's layout consist of very small angles between edges and edge lengths that span nearly the diameter of one or more layout rings. Taking clustering into account during the layout process relieves this focused tree distortion and allows the graph size to be independent of the local tree layout algorithm at the focus.

Going along with clustering, the intermediate node transitioning allows for transitions of more than a few nodes at time without the user becoming lost. For some data sets, the user may wish to follow the path from one node to another via a third node. User-specified paths would be an easy extension for an application that supports a focus transition series. The rest of the additional features aid the users ability to visualize graph exploration, the central focus of Yee et. al.'s solution. The domain best suited for this technique remains unchanged, being best for data which the shortest path from one node to another is the most important path and useful for the user to be able to visualize, leaving non-shortest paths drawn only for the sake of context.

8 Future Work

I believe the biggest strength of this system is that it opens the door to allow radial layout of arbitrary-sized graphs, simply by making the layout algorithm local instead of global. Additionally, navigation from one focus node to another is greatly improved when the user is able to digest smaller steps as opposed to one giant shift that may involve many nodes and sub trees changing parents and moving great distances.

Despite the gains in dealing with trees or sparsely-connected sub graphs that sub tree clustering enabled, it is still, in my opinion, the biggest weakness of this approach. The user would most likely want to view many rings at once, leaving the clustering ring many jumps away, resulting in the layout algorithm considering enough nodes to stumble into the same problems I hoped to avoid by using clustering in the first place. In the case of sparsely connected sub graphs, this is much less pronounced, but in the case of trees, even a tree of only a couple dozen nodes, becomes enough to really distort the graph structure in the first few layout rings. A balance between a balanced tree structure in the focus region and the ability to view nodes many rings away must be set by the user specific to their data, I just worry for some data sets this results in a focus region of too few rings to get enough context for the region around the focus node or enough rings for context but at the same time too many rings to be considered by the layout algorithm to properly balance the focus region tree.

As far as lessons learned, I have mixed feelings about implementing the rendering and graph systems of this solution from scratch. One one hand, reinventing the wheel can be fun at times and the ability to tailor everything to my problem and avoid sifting through mountains of documentation (or dare I mention the horrors of finding bug work-arounds by searching community message boards!) can be quite nice. On the other, time I spent reimplementing existing graph code could have been spent adding more features. Additionally, support for lots of small features and file formats would most likely come for free had I started with a publicly available graphing system.

If given more time to work on this project the first feature I would have added would be to in some way highlight the path a multi-jump transition is taking to better cue the user for changes in direction. There are cases when jumping even a handful of nodes at a time was simply too many and it became a lot of effort to keep up with the animation sequence. Beyond that I would have liked to try to branch out from traditional radial placement a bit, perhaps trying to incorporate other graphing techniques like curved edges or hyperbolic geometry that might have meshed well with this particular problem domain.

References

[1] Gebhardt, Nikolaus et al. *Irrlicht: Lightning Fast Realtime 3D Engine*. http://irrlicht.sourceforge.net/

[2] Gebhardt, Nikolaus et al. *IrrXML: High Speed Simple XML Parser*. www.ambiera.com/irrxml

[3] FLTK: Fast Light Toolkit. www.fltk.org

[4] Khronos Group, OpenGL. www.opengl.org

[5] John Lamping, Ramana Rao. *The Hyperbolic Browser: A Focus +Context Technique for Visualizing Large Hierarchies.* Proc SIGCHI 1995.

[6] Henry Maddocks. *FTGL.* http://ftgl.wiki.sourceforge.net

[7] Tamara Munzner. *H3: Laying Out Large Directed Graphs in 3D Hyprebolic Space.* Proc InfoVis 1997.

[8] Catherine Plaisant, Jesse Grosjean, Ben B. Bederson. *SpaceTree: Supporting Exploration in Large Node Link Tree, Design Evolution and Empirical Evaluation.* Proc InfoVis 2002.

[9] Marco Tarini, Paolo Cignoni, Claudio Montani. *Ambient Occlusion and Edge Cueing to Enhance Real Time Molecular Visualization.* IEEE Transactions on Visualization and Computer Graphics Vol 12 No 5.

[10] Wills, *NicheWorks – Interactive Visualization of Very Large Graphs.* Journal of Computational and Graphical Statistics, Vol 8 No 2.

[11] Jack J. van Wijk, Wim A. A. Nuij. *Smooth and Efficient Zooming and Panning.* Proc InfoVis 2003.

[12] Ka-Ping Yee, Danyel Fisher, Rachna Dhamija, Marti Hearst. *Animated Exploration of Graphs with Radial Layout.* Proc InfoVis 2001.