Journal of Universal Computer Science, vol. 19, no. 1 (2013), 132-156
submitted: 14/12/11, accepted: 28/12/12, appeared: 1/1/13 © J.UCS

Gyrolayout: A Hyperbolic Level-of-Detail Tree Layout

Dana K. Urribarri Silvia M. Castro Sergio R. Martig
(VyGLab, Universidad Nacional del Sur, Bahía Blanca, Argentina
{dku,smc,srm}@cs.uns.edu.ar)

Abstract: Many large datasets can be represented as hierarchical structures, introducing not only the necessity of specialized tree visualization techniques, but also the requirements of handling large amounts of data and offering the user a useful insight into them. Many two-dimensional techniques have been developed, but 3-dimensional ones, together with navigational interactions, present a promising appropriate tool to deal with large trees.

In this paper we present a hyperbolic tree layout extended to support different level-of-detail techniques and suitable for large tree representation and visualization. This layout permits the visualization of large trees with different level of detail in an enclosed 3-dimensional volume. As a significant part of the layout, we also present a Weighted Spherical Centroidal Voronoi Tessellation, an extension of planar Weighted Centroidal Voronoi Tessellations, in order to find an appropriate distribution of nodes on a spherical surface.

Keywords: Hyperbolic Layout, Einstein Gyrovector Space, Centroidal Voronoi Tessellation, Level of detail, Tree Visualization
Category: I.3.5, I.3.6, H.5.0

1 Introduction

There exist many datasets with a highly variable number of relationships among its elements. In particular, when the relations are structured hierarchically, a tree is a powerful abstraction. A broad set of applications use them to represent the relationships in their datasets. In this context it is necessary to provide visualization techniques that are able to handle large trees; this is an important problem that requires special attention.

In contrast to two-dimensional space, the additional third dimension offers more room for placing nodes. This quality makes 3-dimensional tree layouts suitable for visualizing large trees. However, the two-dimensional nature of the output screen introduces the necessity of suitable interactions which enable the user to navigate through the representation [Kaufmann and Wagner, 2001].

Level-of-detail techniques are based on simplification of complex objects' details or groups of objects. This simplification might be achieved substituting a complex object or a group of objects with a simpler representative new object. The level of simplification depends on how significant the replacing object is in comparison with the objects or group of object to be replaced.

In this paper we present an interactive hyperbolic tree layout which uses a natural mathematical abstraction to deal with hyperbolic geometry and it is

based on the layout presented by T. Munzner in [Munzner, 1997, Munzner, 2000] and Weighted Centroidal Voronoi Tessellations. The layout supports different levels of details (LOD) and can display on the order of 10^5 nodes. Hyperbolic geometry is regulated by gyrovector spaces just as Euclidean geometry is regulated by vector spaces; this permits to have a controlled frame to perform geometric transformations. The main aspect to highlight is how theoretical hyperbolic analysis and discrete tessellation algorithms interplay to accomplish an interactive visualization of large trees.

First, we present the previous work; then we introduce some basic concepts on hyperbolic geometry and the models involved in the development of the layout. Next, we describe the Gyrolayout and detail how to distribute the nodes in the model and the resulting algorithms, including the LOD approach. Afterwards, we introduce the chosen visual mapping and the interactions provided by the prototype implementing our layout. Finally, we present some important details about the implementation of the prototype and a comparison of our layout with the other existing ones. The last section is dedicated to conclusions and future work.

2 Previous Work

A commonly used strategy to gain more room to visualize trees and graphs is to design a 3D visualization instead of a visualization in 2D. Generally, 2D algorithms are adapted to 3D. However, the results are not always advantageous: occlusion between objects and navigation in the 3-dimensional space, for instance, are new problems due to the addition of this extra dimension.

At the moment, there are some 3D tree layouts that are a generalization of a 2D layout, while others were developed directly in 3D. Among the first ones, there is the Spherical layout [Larrea, 2006] which is a generalization of the radial layout. On the other hand, Cone Tree [Robertson et al., 1991] is an example of a layout developed directly in 3D. Spherical representation has been further explored in order to achieve improved tree and graph visualizations ([Dmitrieva and Verbeek, 2009], [Choi et al., 2011], [Schulz et al., 2011], [Brath and MacMurchy, 2012]).

A different strategy to gain even more room, is to lay out the tree in the hyperbolic space. The H3 layout [Munzner, 1997] is a generalization of tree visualizations using projections of hyperbolic geometry in the plane [Lamping et al., 1995]. Walrus [CAIDA, 2005] is an implementation for visualizing trees in 3D hyperbolic space.

Hyperbolic space is suitable for visualization of very large trees: both the available room and trees grow exponentially. In combination with a multiresolution strategy, it permits to accommodate even larger trees.

We have developed a tree layout that combines a multiresolution technique with a representation of the tree in hyperbolic space. The former allows the visualization of very large trees, while the latter enables interactive execution times, achieving together an appropriate layout to visualize large trees.

3 Basic Concepts on hyperbolic geometry

Hyperbolic geometry ([Hestenes et al., 2001]) differs from Euclidean geometry in the negation of its fifth postulate. In particular, in two-dimensional Euclidean space (*Euclidean 2-space* or *Euclidean plane*), given a straight line a and a point P, $P \notin a$, there exists only one line b containing P which is parallel to a. However, in the two-dimensional hyperbolic space (*hyperbolic 2-space* or *hyperbolic plane*) there exist many different lines containing P which are parallel to a.

There are several standard models of hyperbolic geometry: the hyperboloid model, the Poincaré ball and the Klein ball (also known as Beltrami ball), among others. We will focus on the Klein model. The Klein model, as well as the Poincaré model, represents the whole hyperbolic space in an enclosed Euclidean volume; in addition, lines in the Poincaré model correspond to arcs in the Euclidean space, whereas lines in the Klein model are also lines in the Euclidean space. This last characteristic makes the Klein model preferable to the Poincaré model.

Gyrovector spaces [Ungar, 2005] are an adaptation of vector spaces for hyperbolic geometry; in particular, the Klein model algebra of hyperbolic geometry is determined by Einstein gyrovector spaces[1]. As vector spaces are a natural way of dealing with Euclidean space, and gyrovector spaces are an adaptation of vector spaces, they are a natural way of dealing with hyperbolic spaces. In this section we present a brief introduction to gyrovector spaces and its operations, which are necessary to fully understand the algorithms presented in the following sections.

Let $\mathcal{V} = (\mathbb{V}, +, \cdot)$ be a real inner product space (Euclidean space) with the binary operation $+$ and the inner product $\cdot$, and let $\mathbb{V}_s$ be the $s-$ball of $\mathbb{V}$, for any fixed $s > 0$:

$$\mathbb{V}_s = \{\mathbf{v} \in \mathbb{V} : \|\mathbf{v}\| < s\}.$$

The Einstein addition $\oplus$ (*gyroaddition*) is a binary operation in $\mathbb{V}_s$ defined by:

$$\mathbf{u} \oplus \mathbf{v} = \frac{1}{1 + \frac{\mathbf{u}\cdot\mathbf{v}}{s^2}} \left(\mathbf{u} + \frac{1}{\gamma_\mathbf{u}}\mathbf{v} + \frac{1}{s^2}\frac{\gamma_\mathbf{u}}{1+\gamma_\mathbf{u}}(\mathbf{u}\cdot\mathbf{v})\mathbf{u} \right).$$

where

$$\gamma_\mathbf{u} = \frac{1}{\sqrt{1 - \frac{\|\mathbf{u}\|^2}{s^2}}}$$

[1] Möbious gyrovector spaces determine the Poincaré model algebra of hyperbolic geometry.

and the Einstein scalar multiplication $\otimes$ is defined by

$$r \otimes \mathbf{0} = \mathbf{0}$$
$$r \otimes \mathbf{v} = s \tanh\left(r \tanh^{-1} \frac{\|\mathbf{v}\|}{s}\right) \frac{\mathbf{v}}{\|\mathbf{v}\|}$$

where $r \in R$, $\mathbf{v} \in \mathbb{V}_s$ and $\mathbf{v} \neq \mathbf{0}$. In every case, the norm $\|\cdot\|$, the inner product, and the vector-scalar operations are the ones that the ball $\mathbb{V}_s$ inherits from the vector space $\mathcal{V}$. Note that gyroaddition is neither commutative nor associative.

The Einstein gyrovector space $(R_s^3, \oplus, \otimes)$ is equivalent to the *Beltrami ball model* of hyperbolic geometry, which is represented by a ball of radius s embedded in the 3-dimensional Euclidean space R^3. In our layout we assume $s = 1$, however, in what remains of this section, we keep s as a possible variable.

Now, it is necessary to define some important concepts in order to use operations of the Einstein gyrovector space to implement hyperbolic geometry in our tree visualization.

Gyropoint and gyroposition We will call *gyropoint* each element of a gyrovector space to avoid ambiguity between them and regular points in the Euclidean space. A *gyroposition* is a position determined by a gyropoint.

Gyrodifference The *gyrodifference* $\ominus$ is defined as

$$\mathbf{a} \ominus \mathbf{b} = \mathbf{a} \oplus -\mathbf{b} = \mathbf{a} \oplus (-1 \otimes \mathbf{b}).$$

Gyrometric and metric The *metric* of the Einstein gyrovector spaces is defined as

$$h(\mathbf{a}, \mathbf{b}) = \tanh^{-1} \frac{d(\mathbf{a}, \mathbf{b})}{s} = \phi_{\mathbf{b} \ominus \mathbf{a}}$$

where $d(\mathbf{a}, \mathbf{b}) = \| \ominus \mathbf{a} \oplus \mathbf{b}\| = \|\mathbf{b} \ominus \mathbf{a}\|$ is a *gyrometric* of the Einstein gyrovector spaces, and $\phi_{\mathbf{v}} = \tanh^{-1} \frac{\|\mathbf{v}\|}{s}$.

Normalized gyrovector We will consider that a gyrovector $\mathbf{v}$ is *normalized* if the metric $h(\mathbf{0}, \mathbf{v}) = \phi_{\mathbf{v} \ominus \mathbf{0}} = \phi_{\mathbf{v}} = 1$. Therefore, $\|\mathbf{v}\| = s \tanh 1$. Given a gyrovector $\mathbf{u}$, the normalized gyrovector of $\mathbf{u}$ will be

$$\frac{1}{\phi_{\mathbf{u}}} \otimes \mathbf{u} = \frac{s \tanh 1}{\|\mathbf{u}\|} \mathbf{u}.$$

Gyrodirections and angles A *gyrodirection* is a gyrovector $\mathbf{v}$ such as $\|\mathbf{v}\| = 1$. The angle between two gyrovectors is the angle between the two associated gyrodirections. Given two gyrovectors $\mathbf{u}$ and $\mathbf{v}$, let α be the *angle* between them, then $\cos \alpha = \frac{\mathbf{u}}{\|\mathbf{u}\|} \cdot \frac{\mathbf{v}}{\|\mathbf{v}\|}$.

Orthonormal basis An orthonormal basis is a set of 3 normalized gyrovectors $\langle \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3 \rangle$ where the angle between every pair of them is $\pi/2$:

$$\|\mathbf{b}_1\| = \|\mathbf{b}_2\| = \|\mathbf{b}_3\| = s \tanh 1$$

$$\mathbf{b}_1 \cdot \mathbf{b}_2 = \mathbf{b}_1 \cdot \mathbf{b}_3 = \mathbf{b}_2 \cdot \mathbf{b}_3 = \cos \pi/2 = 0$$

Gyrovector The *rooted gyrovector* $\mathbf{pq}$ defined by tail $\mathbf{p}$ and head $\mathbf{q}$ is given by the difference $\mathbf{pq} = \ominus\mathbf{p} \oplus \mathbf{q}$.

Translations and rotations A *translation* T of a gyropoint $\mathbf{p}$ by a gyrovector $\mathbf{t}$ is defined by the equation $\mathbf{p}' = T_{\mathbf{t}}(\mathbf{p}) = \mathbf{p} \oplus \mathbf{t}$. The inverse translation T^{-1} of the translation T is $\mathbf{p} = T_{\mathbf{t}}^{-1}(\mathbf{p}') = \mathbf{p}' \boxplus -\mathbf{t}$, where $\boxplus$ is called coaddition and is defined by the equation

$$\mathbf{u} \boxplus \mathbf{v} = 2 \oplus \frac{\gamma_{\mathbf{u}}\mathbf{u} + \gamma_{\mathbf{v}}\mathbf{v}}{\gamma_{\mathbf{u}} + \gamma_{\mathbf{v}}}.$$

Let $\mathbf{l}$ be a line defined by two distinct gyropoints $\mathbf{p}$ and $\mathbf{q}$. A *rotation* R of a gyropoint $\mathbf{v}$ about the line $\mathbf{l}$ by an angle θ is defined in [Phillips and Gunn, 1992] by the equation

$$R_{\mathbf{l},\theta}(\mathbf{v}) = T_{\mathbf{t}}^{-1} \left(R_{\mathbf{u},\theta}^{euc} \left(T_{\mathbf{t}}(\mathbf{v}) \right) \right),$$

where

$$\mathbf{t} = -\frac{\mathbf{p} \cdot (\mathbf{p} - \mathbf{q})}{(\mathbf{p} - \mathbf{q}) \cdot (\mathbf{p} - \mathbf{q})}\mathbf{q} - \frac{\mathbf{q} \cdot (\mathbf{q} - \mathbf{p})}{(\mathbf{q} - \mathbf{p}) \cdot (\mathbf{q} - \mathbf{p})}\mathbf{p}, \qquad \mathbf{u} = \frac{\mathbf{p} - \mathbf{q}}{\|\mathbf{p} - \mathbf{q}\|}$$

and $R_{\mathbf{u},\theta}^{euc}$ is the regular rotation in Euclidean 3-dimensional space.

Gyroparallelogram Let $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ be three gyropoints. Then, the four points $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$ and $\mathbf{d}$ are the vertices of the *gyroparallelogram* $\mathbf{abdc}$, if $\mathbf{d} = (\mathbf{b} \boxplus \mathbf{c}) \ominus \mathbf{a}$.

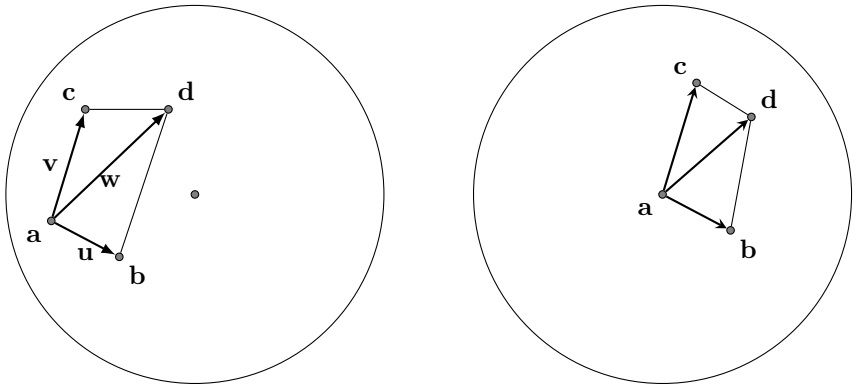
The gyroparallelogram addition law Let $\mathbf{abcd}$ be a gyroparallelogram, then $(\ominus\mathbf{a} \oplus \mathbf{b}) \boxplus (\ominus\mathbf{a} \oplus \mathbf{c}) = (\ominus\mathbf{a} \oplus \mathbf{d})$ (see Figure 1).

4 General layout characteristics

We designed a new layout suitable to accommodate very large trees (in the order of 10^5 nodes) in the hyperbolic 3-dimensional space. This layout allows the visualization of large trees with different levels of details in an enclosed volume embedded in the 3-dimensional Euclidean space. This LOD facilitates the exploration of the dataset by reducing the visual overload.

In the layout we use two different strategies for placing the nodes. At first, the root is placed inside the unit ball (e.g. the center of the sphere), and its children are placed occupying as much as possible of the surrounding available space; to accomplish this task we use a specially designed Centroidal Voronoi Tessellation ([Du et al., 1999]). To place the nodes at a depth greater than or equal to 2 (i.e. the descendants of the children of the root) we based our layout on the placement scheme presented in [Munzner, 1997]: the children of a node n are placed on the surface of a hemisphere centered at that node; the hemisphere is pointing in the outward direction, from the parent node of n to n (see Figure 2).

The algorithm has three major steps:



(a) Let $\mathbf{d}$ be the gyropoint such that $\mathbf{d} = (\mathbf{b} \boxplus \mathbf{c}) \oplus \mathbf{a}$, then **abcd** is a gyroparallelogram. Furthermore, let $\mathbf{u} = \ominus\mathbf{a} \oplus \mathbf{b}$, $\mathbf{v} = \ominus\mathbf{a} \oplus \mathbf{c}$ and $\mathbf{w} = \ominus\mathbf{a} \oplus \mathbf{d}$, then $\mathbf{u} \boxplus \mathbf{v} = \mathbf{w}$.

(b) In the particular case where $\mathbf{a} = \mathbf{0}$, the gyroparallelogram addition law reduces to $\mathbf{d} = \mathbf{b} \boxplus \mathbf{c}$.

Figure 1: The Einstein gyroparallelogram addition law. Figure (a) illustrates the gyroparallelogram addition law in the general case. On the other hand, figure (b) shows the particular case where one of the vertices of the gyroparallelogram is coincident with the origin.

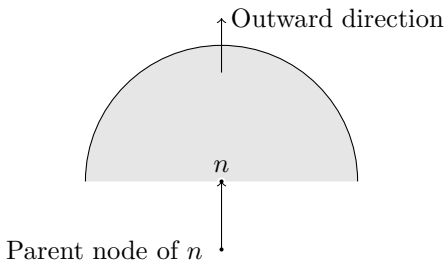


Figure 2: The hemisphere where the children of the node n are placed is centered at n and is pointing in the direction from the parent node of n to n.

A) a bottom-up traversal that calculates the radii of the hemispheres where the children of every node (excepting the root) are to be placed.
B) a weighted centroidal spherical Voronoi Tessellation that places the root and distributes its children in their final location on the sphere surface.
C) a top-down traversal that places the remaining nodes (the descendants of root's children).

In the following three subsections each of these steps is described in detail. The algorithm to generate the Gyrolayout is based on these algorithms, and is presented in the fourth subsection. The last subsection is dedicated to present

the level of detail approach for our layout.

4.1 Calculating the radii of the hemispheres

The *bottom-up traversal* (outlined in Algorithm 1) calculates the radii of the hemispheres needed to place the children of every node. It starts arranging the children of each node in a disk. Then, for each disk, it calculates the radius of the hemisphere where the nodes are to be projected; the hemisphere has the same area as the disk.

Let n be a node and $h_1, h_2, \ldots, h_m$ its m children. Initially, the bottom-up traversal calculates recursively the size of the disk necessary to accommodate the children of each $h_1, h_2, \ldots, h_m$ (step 1.1), that is the size of the disk necessary to accommodate all the *grandchildren* of n. After that, it arranges them into circular rings (steps 1.3–1.7), starting from an internal concentric circle. The nodes must be arranged in such a way that those with larger associated circles occupy the central rings, and as the size of the circles decreases, the nodes are placed in outer rings (see Figure 3). For a leaf node, the associated size (radius) is the radius of the sphere which delimits the volume occupied by the visual element of that leaf. The algorithm knows if there is enough room to place a node in a ring (step 1.5) by keeping track of the distance δ (see Figure 3) and the occupied angle α of each ring:

1: Let $\mathcal{R}$ be a ring with distance δ and occupied angle α.
2: Let n be a node with radius r.
3: There is enough room for n in $\mathcal{R}$ if $2\pi - \alpha \geq \sin^{-1} r/\delta$.

Once all the children are arranged, the algorithm calculates the size (radius) of the whole circle (step 1.8), and finally, calculates the size (radius) of a hemisphere with the same area (step 1.9). Figure 4 shows two views of a hemisphere where only leaves are placed. In the hyperbolic space, the area of a circle with radius r and the surface area of a sphere with radius r are[2], respectively:

area of a circle	$4\pi \sinh^2(\frac{r}{2})$
surface area of a sphere	$4\pi \sinh^2(r)$

Then, the surface area of a hemisphere with radius r is $2\pi \sinh^2(r)$. In this algorithm we need to find the radius R_s of a hemisphere with the same area as a circle with radius R_{out}. Isolating R_s from the following equality

$$4\pi \sinh^2\left(\frac{R_{out}}{2}\right) = 2\pi \sinh^2(R_s)$$

[2] For more details on history and development of these formulae reference to [Bonola, 1912]

Algorithm 1 Distribute the children of node n in rings

▷ *The bottom-up traversal*

Input: A node n.
Output: A distribution for the children of n in rings.

1.1: **for all** child h of n **do**
1.2: Distribute the children of node h in rings. ▷ *Recursive calculation of children's rings*
▷ *Arrangement in circular rings*
1.3: Let i start at 0. ▷ *The interior concentric circle*
1.4: **for all** child h of n **do** ▷ *Nodes retrieved in descending order*
1.5: **if** there is not enough room to put h in ring i **then**
1.6: Increase i by 1. ▷ *Jump to the next outer ring*
1.7: Place h in ring i.
1.8: Let R_{out} be the sum of the width of all necessary rings.
1.9: Let $R_s = \sinh^{-1}\sqrt{\cosh(R_{out}) - 1}$ be the radius of the hemisphere of node n.

and simplifying, we obtain (step 1.9):

$$R_s = \sinh^{-1}\sqrt{2\sinh^2\left(\frac{R_{out}}{2}\right)} = \sinh^{-1}\sqrt{\cosh(R_{out}) - 1}.$$

By the end of this algorithm, we have calculated the radius of the hemisphere necessary to accommodate the children of n and the radii necessary for the children of every descendant of n.

4.2 Placing the root and its children

The root can be placed anywhere inside the sphere; therefore, we can choose the center of the ball as a suitable initial position. After that, the nodes must be distributed in the available space around the root. The chosen strategy was to distribute the children of the root r on the surface of an imaginary unitary ball centered at r, using the Weighted Spherical Centroidal Voronoi Tessellation (WSCVT) presented in [Larrea et al., 2009] and detailed in Appendix A.

The weight of a node indicates how much area of the sphere the node will need to accommodate its children: if the node is a leaf, it will need much less space than any other node with some child. Therefore, the weight of each node must take into account how much area is needed to place its descendants.

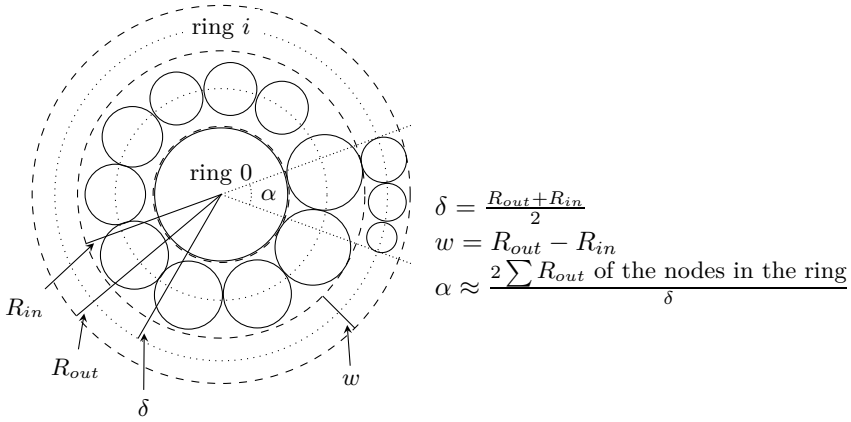


Figure 3: Circular rings of nodes. R_{out} and R_{in} are the outer and inner radii of ring i. δ is the distance from the center of the circle to the middle of the ring, w is the width of the ring, what corresponds to the diameter associated to the "bigger" node placed in the ring, and α represents an approximation to the actual value of the occupied angle inside a ring.

(a) Top view (b) Side view

Figure 4: Two different views of a hemisphere where only leaves are placed.

4.3 Placing the remaining nodes

The *top-down traversal* (Algorithm 2) calculates the final location of the children of n in the hyperbolic space. To accomplish this task the algorithm needs the final gyropositions $\mathbf{p}_n$ and $\mathbf{p}_f$ of the node n and its parent node f, respectively, and the rings associated to n computed by Algorithm 1, previously described. The particular case of calculating the position of the root and its children (these last ones needed by this algorithm) was presented in the previous subsection. This *top-down traversal* goes through every node in each ring (step 2.7), calculating the gyroposition of the nodes. The rings are traversed through from the central one to the outer one (step 2.3).

Algorithm 2 Compute the gyroposition of the children of node n ▷ *The top-down traversal*

Input: A node n
The gyroposition $\mathbf{p}_n$ of n
The normalized gyrovector $\mathbf{d}_n$ pointing in direction of the hemisphere of n

Output: The gyropositions of the children of n

2.1: Let r_n be the radius of the hemisphere containing the children of n.
2.2: Let $\langle \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3 \rangle$ an orthonormal basis where $\mathbf{b}_1 = \mathbf{d}_n$.
2.3: **for all** ring $\mathcal{A}$ where the children of n are placed **do**
2.4: Let $\delta_{\mathcal{A}}$ be the distance from the middle of the ring to the center of the circle that contains it. ▷ *See Figure 3*
2.5: Let ϕ start at 0.
2.6: Define the rotation $R_{\mathbf{b}_3,\beta_{\mathcal{A}}}$, being $\beta_{\mathcal{A}} = \delta_{\mathcal{A}}/r_n$.
2.7: **for all** node m placed in ring $\mathcal{A}$ **do**
2.8: Let $\mathbf{d}_m = \mathbf{b}_1$ be a normalized gyrovector pointing in the direction of the hemisphere of m.
2.9: $R_{\mathbf{b}_3,\beta_{\mathcal{A}}}(\mathbf{d}_m)$. ▷ *Rotate* $\mathbf{d}_m$
2.10: Let α_m be $2\sin^{-1} r_m/\delta_{\mathcal{A}}$.
2.11: Increment ϕ in $\alpha_m/2$.
2.12: Define the rotation $R_{\mathbf{b}_1,\phi}$.
2.13: $R_{\mathbf{b}_1,\phi}(\mathbf{d}_m)$. ▷ *Rotate* $\mathbf{d}_m$
2.14: Increment ϕ in $\alpha_m/2$.
2.15: Set $\mathbf{p}_m = \mathbf{p}_n \oplus (r_n \otimes \mathbf{d}_m)$. ▷ $T_{r_n \otimes \mathbf{d}_m}(p_n)$
2.16: *Compute the gyroposition of the children of node* m $(m, \mathbf{p}_m, \mathbf{d}_m)$.

4.4 The Gyrolayout: Putting all together

In this subsection we describe how the previous algorithms are assembled to obtain the Gyrolayout, and give the details of the three major steps (A, B and C) presented in the introduction of this section (Algorithm 3).

The algorithm takes the tree to be visualized and the position of the root node in the hyperbolic space (a gyroposition) and creates the final visualization of the tree. The first step is to distribute all nodes in rings (step 3.2). If the root has enough children, the algorithm calculates the initial weights for these children (as the radius of the disk where rings centered at that node are placed, step 3.4) and computes the corresponding WSCVT (step 3.5). After that, the algorithm calculates the radius ρ of a sphere with enough area to accommodate the disk of all the root children (step 3.6). Then, for each child i of the root, it projects the gyroposition of i (calculated by the WSCVT) on the surface of radius ρ and calculates the gyrovector pointing in the outward direction (steps 3.8 and 3.9).

Starting with these values, the algorithm calculates the gyroposition of all the descendants of each child node (step 3.10).

If the root has too few children it is not worth computing a distribution that covers all the sphere surface because this distribution is going to be too sparse. The algorithm just assumes some initial direction and places the children of the root also in a hemisphere (steps 3.12 and 3.13).

Algorithm 3 Visualize tree $\mathcal{T}$ ▷ *Putting all together*

Input: A tree $\mathcal{T}$.

A gyroposition $\mathbf{O}$ being the position of the root inside the ball, may be $\{0, 0, 0\}$.

3.1: Let r be the root node of $\mathcal{T}$.

3.2: *Distribute the children of node r in rings.* ▷ *Algorithm 1*

3.3: **if** r has 4 or more children **then**

3.4: Let $W = \{w_1, \ldots, w_n\}$ be the weights of the children of r, where w_i is the weight of child i.

3.5: Let $P = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$ be the result of WSCVT(W). ▷ *Algorithm 6*

3.6: $\rho = \sqrt{\frac{\sum \pi w_i}{4\pi}} = \frac{\sqrt{\sum w_i}}{2}$.

3.7: **for all** child i of r **do**

3.8: Associate with i the gyroposition $\mathbf{g}_i = \rho \otimes \frac{\mathbf{p}_i}{\|\mathbf{p}_i\|}$.

3.9: Let $\mathbf{d}_i = \frac{\ominus \mathbf{O} \oplus \mathbf{g}_i}{\|\ominus \mathbf{O} \oplus \mathbf{g}_i\|}$ be a gyrovector pointing in the direction of the hemisphere of i.

3.10: *Compute the gyroposition of the children of node $i(i, \mathbf{g}_i, \mathbf{d}_i)$.* ▷ *Algorithm 2*

3.11: **else**

3.12: Let $\mathbf{d}$ be some initial direction, for example $\{\tanh 1, 0, 0\}$.

3.13: *Compute the gyroposition of the children of node $r(r, \mathbf{O}, \mathbf{d})$.* ▷ *Algorithm 2*

3.14: Visualize $\mathcal{T}$.

4.5 The Gyrolayout and the level of details

As the position of a node tends to be placed close to the surface of the Klein ball, its visual element will be smaller, because the surface of the ball represents the infinity of the hyperbolic space. Moreover, as a node is deeper in the tree, its location will be farther from the root, and nearer to the surface of the ball. Then, as the node is deeper in the tree, its visual element will be smaller, tending to disappear.

Taking this into account, an appropriate criteria to set a level of detail is how much appreciable are the details of the bunches of nodes: those subtrees for which the visual elements of their nodes are small and crammed can be pruned.

All nodes are represented by a tetrahedron, because of its reduced number of faces; we are prioritizing efficiency over aesthetic-looking. The tetrahedron permits the interaction with the node and the addition of node's properties to the visualization. A pruned subtree is marked with a cone to make evident the existence of hidden details. The cone is a different and significant visual element (based on the coalesced trees in [Carriere and Kazman, 1995]) that represents the root of the pruned subtree.

Algorithm 4 Collapse a subtree rooted at n

Input: A node n.

Output: A cone representing n and the children of n.

4.1: Let $\mathbf{p}_n$ be the gyroposition of n.
4.2: Let $\mathbf{p}_m$ be the gyroposition of the node in the inner ring of n.
4.3: Let $\mathbf{p}_f$ be the gyroposition of the parent of n.
4.4: Let $\mathbf{p}_n$ be the apex of the cone.
4.5: Let $\mathbf{d} = \ominus\mathbf{p}_f \oplus \mathbf{p}_n$ be the direction of the cone.
4.6: Let $h = \|\mathbf{p}_n \ominus \mathbf{p}_m\|$ be the height of the cone.
4.7: Let r_n be the radius of the hemisphere of n.
4.8: **return** A cone with apex in $\mathbf{p}_n$, height h, oriented in the direction of $\mathbf{d}$ and base with radius r_n.

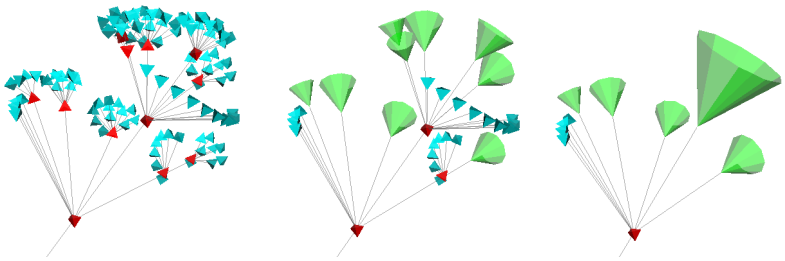
In Algorithm 4 it is outlined how to calculate the size of the cone representing the collapsed subtree.

5 Visualizing a tree

Tree visualization not only implies placing nodes and edges in space, but also deciding which and how attributes in the dataset will be visually represented, and what interactions should be provided. In general, applications for tree visualization differ on some of the following aspects:

- they use different tree layouts to place nodes and edges,
- they use different visual mappings for attributes,
- they provide different sets of interactions.

Our proposal uses the Gyrolayout for placing edges and nodes. Attribute representation and interactions are strongly dependent on the data domain. In



(a) (b) (c)

Figure 5: Different levels of detail. The three figures show the same subtree at different levels of detail. Figure (a) shows it at a full detail. Figure (b) shows it with the nodes which have only leaves as children collapsed. In Figure (c), the nodes which have either leaves or collapsed nodes as children in (b) are collapsed.

the following subsections some key aspects on visual mapping and the provided interactions are discussed.

5.1 Visual Mapping

At the time of establishing the visual mapping of nodes and edges it is necessary to take into account certain factors in order to improve the visualization and obtain a better insight into the structure of the tree.

- the edges of the tree are colored according to the depth of the target node. The color may go from one color, for edges with origin at the root of the tree, to another color, for edges with destination at a node at maximum depth (see Figure 9).
- the nodes of the tree can be colored according to its depth in the tree or according to the semantic value of the node in the dataset.
- the portion of the tree placed into the hemisphere opposite to the viewer is shown less saturated, to give the user some feedback about what is near and what is farther.

5.2 Interactions

Herman et al. [Herman et al., 1998] describe a set of interactions that helps the user in the navigation of large trees in 2D:

1. *Zooming and panning* which are the basic interaction for navigation.
2. *Focus+context* through fish-eye.

3. *Complexity clues* to give the user information to indicate where the interesting subtrees are, for example in which direction the tree grows (indicated by Strahler numbers).
4. *Folding and unfolding* to hide specific subtrees; only the root is left as a representative of the whole subtree.

All these basic interactions are supported in our layout. Taking into account its 3D nature, the interactions are adapted when it is necessary:

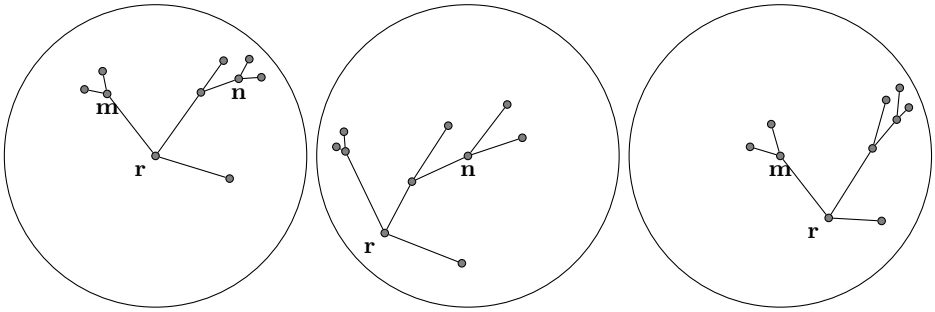
1. The *zooming and panning* in 3D representations correspond to rotation and translation of the camera, which make it possible for the observer to interact with the hyperbolic space without changing the visual representation.
2. The *focus+context* view is intrinsic to our representation in hyperbolic space, where the nodes near the center of the Klein Ball are displayed bigger and more detailed than those lying near the surface of the ball.
3. *Complexity clues* can be added to nodes and made visible by semantic zoom.
4. *Folding and unfolding* of nodes is exactly what the level of detail technique does: it hides some subtrees based on certain criteria.

Besides, an additional set of interactions was designed considering how significant they are for a tree visualization.

- *Semantic zoom on nodes and cones.* Semantic zoom makes it possible to show on demand additional information about the node: when a node is clicked a tooltip is displayed which, for example, shows the absolute path of a file if the tree is a directory hierarchy (see Figure 7b). In case of cones, the tooltip may show the height of the hidden subtree (see Figure 7c).
- *Path highlighting.* When clicking on a node, the path from that node to the root of the tree is highlighted.
- *Translation of the layout in the hyperbolic space.* Besides the translation of the camera, which does not change the hyperbolic representation of the layout, our proposal permits the translation of the layout in the hyperbolic space. This is accomplished translating (see *Translation* in Section 3) all the points from the current position to the original one (the root of the tree at the origin) and then translating the desired node to the center of the ball (see Figure 6). Despite this interaction resulting in a different hyperbolic position for each node and, consequently, altering the representation of the tree, it permits to change the point of interest of the visualization.

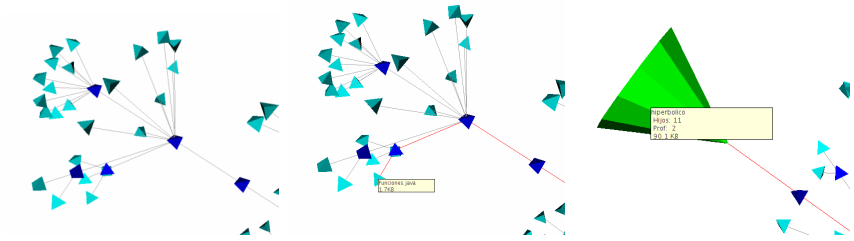
6 Implementation

A prototype application for tree visualization using this Gyrolayout was implemented in Java using Vtk library. The implemented prototype provides the



(a) (b) (c)

Figure 6: An example of the layout translation. Figure (a) shows the original layout of the tree, with the root node placed at the center of the ball. Figure (b) shows the layout after the translation $T_n(\mathbf{p}_i) = \mathbf{p}_i \oplus (-\mathbf{n})$ which places node $\mathbf{n}$ at the center of the ball. Figure (c) shows the layout with node $\mathbf{m}$ at the center of the ball. This last layout results from applying to (b) the concatenation of the inverse translation $\mathbf{p}_i = T_n(\mathbf{p}_i) \boxplus (-\mathbf{r})$ with translation $T_m(\mathbf{p}_i) = \mathbf{p}_i \oplus (-\mathbf{m})$. The former transformation goes back to the original layout of the tree (figure (a)) and the latter one places node $\mathbf{m}$ at the center of the ball.



(a) Portion of a tree at full detail. (b) Tooltip on a node. (c) Tooltip on a cone.

Figure 7: Example of semantic zoom on nodes and cones using tooltips to display additional information.

basic interactions implemented by Vtk: rotation and translation of the camera (see Figure 9), as well as the interactions described in Section 5.2.

In order to deal with very large trees and improve interactivity, some specific characteristics have been considered during the implementation to speed up the rendering process:

- nodes are represented with a simplex (a tetrahedron)
- edges are represented with lines, not tubes.

– nodes are not shown during camera transformations.

To achieve a visualization of the hyperbolic layout, we adapted the class vtkGlyph3D to perform all transformation (i.e. translations according to the gyroparallelogram addition law, cf. Figure 1b–) as gyrotransformations.

7 Comparison with other implementation of hyperbolic tree layouts

This section presents a comparison (Table 1) between our layout and some existing alternatives: H3 [Munzner, 1997] and Walrus [CAIDA, 2005]. The comparison was developed in a Core2Duo @ 2.66GHz processor with 4GB of RAM PC running Debian GNU/Linux Operating System.

While our layout and Walrus were both implemented in Java, our layout uses Vtk for 3D representations and Walrus uses Java3D. H3 was implemented in C++ and OpenGL; we tested with Wine[3] the Windows version of the application. The comparison consists on measuring how much time each application takes to visualize (including preprocessing and rendering time) trees of different amount of nodes. The comparison (table 1) shows that times measured for the Gyrolayout are between those for H3 and Walrus (with simple and double precision[4]). This is remarkable considering that the Gyrolayout not only was implemented using double precision, but also permits important interactions such as folding and unfolding of nodes.

Furthermore, H3 does not permit the rotation of the ball that represents the hyperbolic space or zooming in/out the ball to see more details. Instead, it can translate the tree inside the hyperbolic space; in this way it achieves a more detailed view of the center of the visualization but modifies the representation of the tree making the user to lose context (see Figure 8). In contrast with H3, Gyrolayout can provide more detail using also panning, zooming and rotation of the sphere (see Figure 9).

As for node placement, both Walrus and H3 place the nodes of the tree in such a way that it occupies only one hemisphere (see Figure 10). However, our layout tends to distribute nodes on the whole surface of the sphere (see Figure 9(a)).

8 Conclusion and Future Work

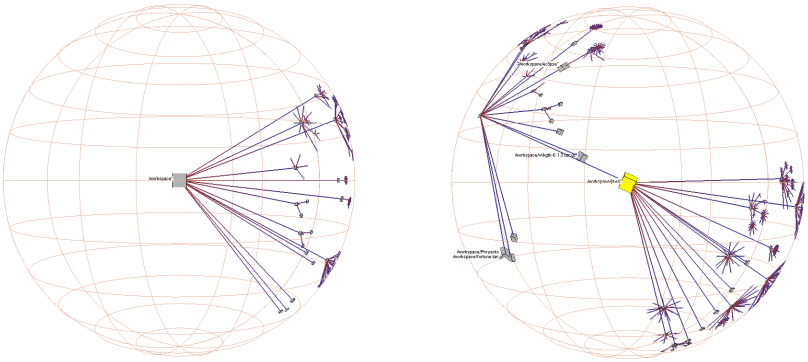
We have developed a tree layout based on Centroidal Voronoi Tessellations and Einstein gyrovector spaces that support different levels of detail (see Figure 11).

[3] `www.winehq.org`

[4] Walrus uses `mpfun` package to handle multiple precisions; this considerably reduces the performance while working with double precision.

Amount of nodes	**Gyrolayout**	**H3**	**Walrus**	
			Simple precision	**Double precision**
2196	00:00.574	instant	00:00.057	00:02.079
4444	00:01.575	a blinking	00:00.150	00:03.627
4994	00:01.790	a blinking	00:00.054	00:03.594
83845	00:15.219	4/5 secs	00:00.555	01:03.578
138543	00:24.996	5/6 secs	00:00.457	01:37.615
193331	00:34.863	8 secs	00:01.722	02:19.130

Table 1: Comparison of our proposal with H3 and Walrus. The comparison was developed taking into account how much time each application takes to visualize (including preprocessing and rendering time) trees of different amount of nodes. Note that there are no times specified for H3, because the application does not provide such information. Besides, Walrus could be tested with simple and double precision, while our layout uses only double precision.



(a) Directory `workspace` visualized with H3.

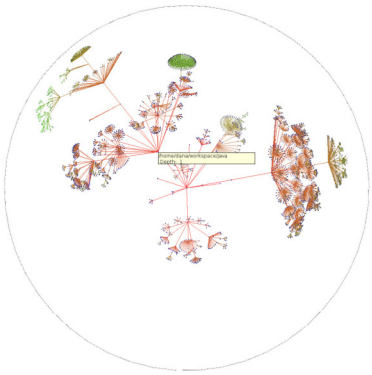
(b) Same directory. The subdirectory `workspace/java` is placed at the center of the sphere.

Figure 8: How to get more details visualizing with H3.

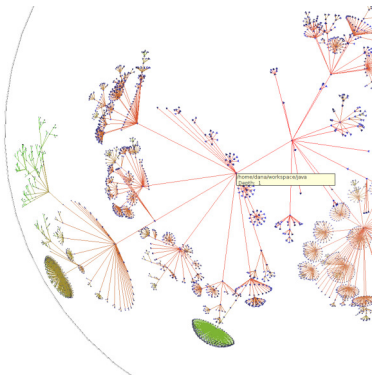
The main strength of our layout is that, due to be placed in the hyperbolic space and to support different levels of detail, is really appropriate to visualize large trees (see figure 12): it represents a whole tree just into the limits of a sphere in the conventional 3-dimensional space. Theoretical hyperbolic analysis and discrete tessellation algorithms interplay in this layout to accomplish an effective visualization of large trees.

As future work we are planing several improvements to the layout in terms of visual complexity and additional interactions. We are also planing to conduct usability tests to validate our layout.

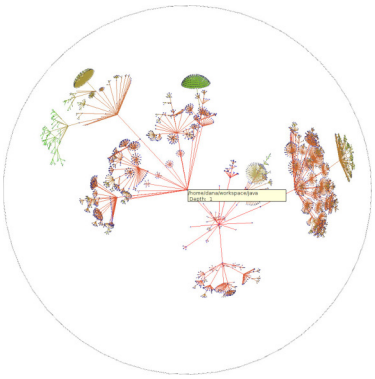
(a) Directory `workspace` visualized with Gyrolayout.



(b) More details on subdirectory `workspace/java` achieved through semantic zoom.



(c) More details on subdirectory `workspace/java` achieved through zooming, panning and rotation.



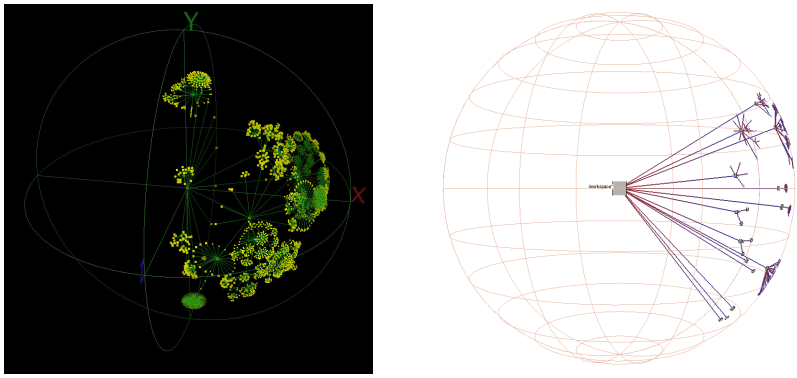
(d) More details on subdirectory `workspace/java` achieved through moving the corresponding node to the center of the sphere.

Figure 9: How to get more details visualizing with Gyrolayout. Directory `workspace` has 4994 nodes and depth 11. The color of the edges goes from red, for edges with origin at the root of the tree, to green, for edges with destination at a node at maximum depth.

Our final goal is to adapt the layout and the level-of-detail approach to support not only large trees, but also large graphs.

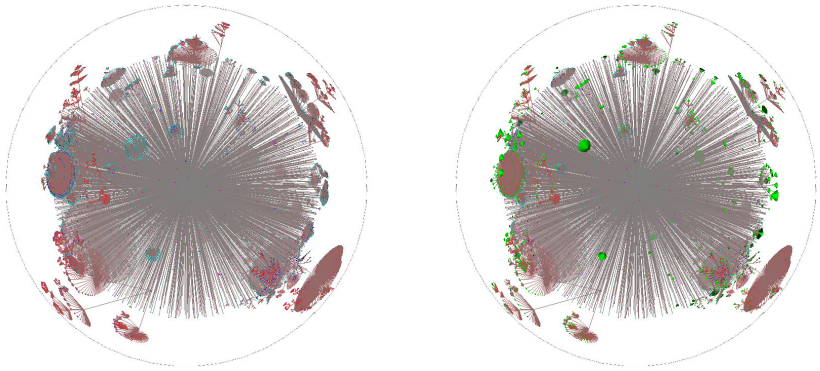
Acknowledgment

The presented work is partially funded by PGI 24/N020, Secretaría General de Ciencia y Tecnología, Universidad Nacional del Sur, Bahía Blanca, Argentina.



(a) Walrus (b) H3

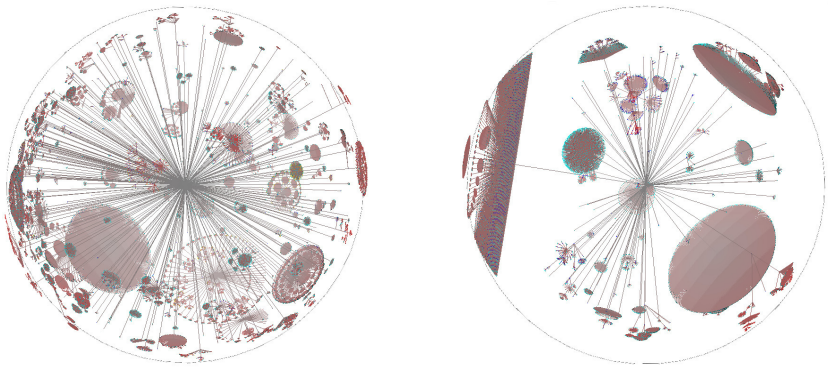
Figure 10: The same dataset from Figure 9 (a directory tree with 4994 nodes and depth 11) visualized with the other two compared application. Both H3 and Walrus place root children in a way that they occupy only a half of the sphere. Gyrolayout places them occupying the whole sphere surface (see Figure 9(a)).



(a) The `/usr/lib` directory shown with all details.

(b) The `/usr/lib` directory with the folders containing only files collapsed.

Figure 11: The `/usr/lib` directory, which has 25,253 files and 2,600 sub-folders, at three different levels of detail. The first level has 2,154 files and 196 folders, and the tree has depth 13. In figure (b) a different color is used for the cones to reinforce the idea of collapsed subtrees.



(a) The `/usr/share` directory which has 127729 files, 10814 subfolders and depth 13.

(b) The `windows` directory which has 68,539 files, 15,305 subfolders and depth 11.

Figure 12: Different trees with different amount of internal nodes and leaves. Figure (a) shows the `/usr/share` directory of a Linux system. Figure (b) shows the `windows` directory of a Windows Vista system.

References

[Balzer and Deussen, 2005] Balzer, M. and Deussen, O. (2005). Voronoi treemaps. In *IEEE Symposium on Information Visualization (InfoVis 2005)*, pages 49–56.

[Bonola, 1912] Bonola, R. (1912). *Non Euclidean-Geometry. A critical and historical study of its development.* Chicago. The Open Court Publishing Company.

[Brath and MacMurchy, 2012] Brath, R. and MacMurchy, P. (2012). Sphere-based information visualization: Challenges and benefits. In Banissi, E., Bertschi, S., Burkhard, R. A., Cvek, U., Eppler, M. J., Forsell, C., Grinstein, G. G., Johansson, J., Kenderdine, S., Marchese, F. T., Maple, C., Trutschl, M., Sarfraz, M., Stuart, L. J., Ursyn, A., and Wyeld, T. G., editors, *IV*, pages 1–6. IEEE Computer Society.

[CAIDA, 2005] CAIDA (2005). Walrus - graph visualization tool. `http://www.caida.org/tools/visualization/walrus/`.

[Carriere and Kazman, 1995] Carriere, J. and Kazman, R. (1995). Interacting with huge hierarchies: Beyond cone trees. In *Proc. IEEE Information Visualization '95*, pages 74–81. IEEE Computer Press.

[Choi et al., 2011] Choi, J., Kwon, O.-h., and Lee, K. (2011). Strata treemaps. In *ACM SIGGRAPH 2011 Posters*, SIGGRAPH '11, pages 87:1–87:1, New York, NY, USA. ACM.

[Dmitrieva and Verbeek, 2009] Dmitrieva, J. and Verbeek, F. J. (2009). Node-link and containment methods in ontology visualization. In Hoekstra, R. and Patel-Schneider, P. F., editors, *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2009), Chantilly, VA, United States, October 23-24, 2009*, volume 529 of *CEUR Workshop Proceedings*. CEUR-WS.org.

[Du et al., 1999] Du, Q., Faber, V., and Gunzburger, M. (1999). Centroidal voronoi tessellations: Applications and algorithms. *SIAM Review*, 41(4):637–676.

[Herman et al., 1998] Herman, I., Delest, M., and Melançon, G. (1998). Tree visualisation and navigation clues for information visualisation.

[Hestenes et al., 2001] Hestenes, D., Li, H., and Rockwood, A. (2001). *A universal model for conformal geometries of Euclidean, spherical and double-hyperbolic spaces*, chapter 4, pages 77–104. Springer-Verlag, London, UK.

[Jenness, 2008] Jenness, J. S. (2008). Calculating areas and centroids on the sphere. Poster presented at Arizona/New Mexico Wildlife Society Meeting. Albuquerque, New Mexico, USA (2008) and 28th Annual ESRI International User Conference. San Diego, California, USA.

[Kaufmann and Wagner, 2001] Kaufmann, M. and Wagner, D., editors (2001). *Drawing graphs: methods and models*. Springer-Verlag, London, UK.

[Lamping et al., 1995] Lamping, J., Rao, R., and Pirolli, P. (1995). A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 401–408. ACM Press/Addison-Wesley Publishing Co.

[Larrea et al., 2009] Larrea, M., Urribarri, D., Castro, S., and Martig, S. (2009). Spherical layout implementation using centroidal voronoi tessellations. *Journal of Computing*, 1(1):81–86. `http://arxiv1.library.cornell.edu/pdf/0912.3974`.

[Larrea, 2006] Larrea, M. L. (2006). Diagramado esférico. Master's thesis, Universidad Nacional del Sur.

[Munzner, 1997] Munzner, T. (1997). H3: Laying out large directed graphs in 3d hyperbolic space. In *IEEE Symposium on Information Visualization*, pages 2–10.

[Munzner, 2000] Munzner, T. (2000). *Interactive Visualization of Large Graphs and Networks*. PhD thesis, Stanford University.

[Phillips and Gunn, 1992] Phillips, M. and Gunn, C. (1992). Visualizing hyperbolic space: unusual uses of 4x4 matrices. In *SI3D '92: Proceedings of the 1992 symposium on Interactive 3D graphics*, pages 209–214, New York, NY, USA. ACM.

[Robertson et al., 1991] Robertson, G. G., Mackinlay, J. D., and Card, S. K. (1991). Cone trees: animated 3d visualizations of hierarchical information. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 189–194. ACM.

[Schulz et al., 2011] Schulz, H.-J., Hadlak, S., and Schumann, H. (2011). The design space of implicit hierarchy visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 17:393–411.

[Ungar, 2005] Ungar, A. A. (2005). *Analytic Hyperbolic Geometry. Mathematical Foundations and Applications*. World Scientific Publishing Co. Pte. Ltd.

A Weighted Spherical Centroidal Voronoi Tessellation

Before introducing the Weighted Spherical Centroidal Voronoi Tessellation used in our layout, we define Delaunay Tessellation, Voronoi Tessellation, Spherical Voronoi Tessellation and Weighted Spherical Voronoi Tessellation.

Delaunay Tessellation

Given a set of points $P = \{p_1, p_2, \ldots, p_n\}$ in R^m, a Delaunay Tessellation is a set of k simplexes $t_1, t_2, \ldots t_k$ with vertices in P such that no point in P is in the interior of the hyper-circumcircle of any simplex t_i.

A Delaunay Triangulation (2-dimensional Delaunay Tessellation) is a set of triangles with vertices in P such that no point in P is in the interior of the circumcircle of any triangle in the triangulation.

Voronoi Tessellation

Given a set of points (generators) $P = \{p_1, p_2, \ldots, p_n\}$ in R^m, a Voronoi Tessellation is a set of n regions $V(p_i)$, where a point $q \in R^m$ lies in region $V(p_i)$ if and only if distance$(p_i, q) <$ distance(p_j, q) for each $p_i, p_j \in P, i \neq j$.

The Voronoi Tessellation and the Delaunay Tessellation are duals: every region of a Voronoi Tessellation (i.e. every generator) corresponds to a vertex of the Delaunay Tessellation and for every two adjacent regions in the Voronoi Tessellation there is an edge between the corresponding vertices in the Delaunay Tessellation.

Spherical Voronoi Tessellation

A Spherical Voronoi Tessellation is a Voronoi Tessellation of the surface of a sphere. In this case the set P is a set of points lying on a surface $\mathcal{S} = \{(x, y, z) \in R^3 : x^2 + y^2 + z^2 = 1\}$, and the regions $V(p_i)$ are the points $q \in \mathcal{S}$ which satisfy distance$(p_i, q) <$ distance(p_j, q) for each $p_i, p_j \in P, i \neq j$.

Weighted Spherical Voronoi Tessellation

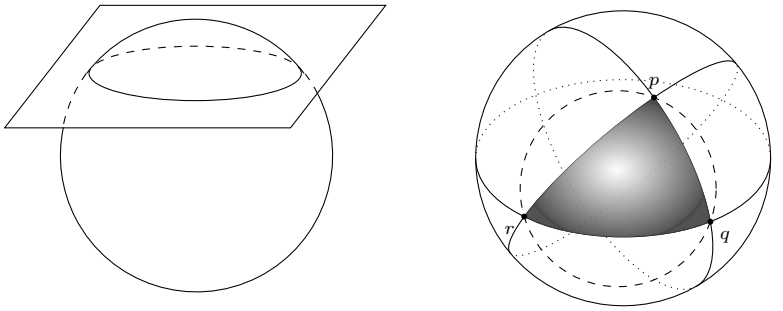
A Weighted Spherical Voronoi Tessellation (WSVT) is a Spherical Voronoi Tessellation where each generator p_i has associated a weight w_i, and the distance between a point q and a generator p_i is the weighted distance w-distance$(a, w_a, x) = |a - x|^2 - w_a$, where $|.|$ is the euclidean distance.

The general idea of the algorithm used to calculate the WSVT of points P on the sphere is outlined in Algorithm 5. Given a set of points P on the sphere surface and the weight W corresponding to each point, the algorithm calculates the Weighted Spherical Voronoi Tessellation of the sphere surface according to P and W.

The Delaunay triangulation of the points on the surface of the sphere is calculated as the convex hull of these points, based on the fact that if $\triangle pqr$ is a face of the convex hull, then there exists a plane P containing p, q, and r which leaves an empty halfspace on one side, and all the remaining points on the other. The intersection between P and the sphere S is a circle (see Figure 13a), which is the circumcircle of the spherical triangle $\spadesuit pqr$ (see Figure 13b). There is no point within this circumcircle (the empty halfspace); all the remaining points are lying outside it (the non-empty halfspace). Thus, $\spadesuit pqr$ is a Delaunay triangle on the surface of the sphere.

The weighted circumcenter of a spherical triangle $\spadesuit abc$ where w_a, w_b and w_c are the weights of a, b and c respectively, is defined as follows. Let c be the circumcenter of $\triangle abc$, c_w be the point coplanar to a, b and c which satisfies

$$\text{w-distance}(a, w_a, x) = \text{w-distance}(b, w_b, x) = \text{w-distance}(c, w_c, x)$$



(a) Plane-sphere intersection

(b) Circumcircle of a spherical triangle

Figure 13: The circumcircle of a spherical triangle. (a) Circle resulting from the intersection between a sphere and a plane. (b) The intersection between the sphere and the plane containing the tree vertexes of a spherical triangle, results in the circumcircle of that spherical triangle.

and n be the normal of the plane which contains $\triangle abc$

$$n = \frac{(b-a) \times (c-a)}{|(b-a) \times (c-a)|}.$$

Then, the weighted circumcenter s_w (see Figure 14) of $\triangle\!\!\!\!\triangle abc$ is the intersection between the unitary ball and the ray of direction n and origin at point c'_w, where c'_w is the following point:

$$c'_w = \frac{(c_w - c)\sqrt{1-(n.c)^2}}{|c_w - c| + 1} + c.$$

Algorithm 5 Weighted Spherical Voronoi Tessellation (WSVT)

Input: A set of points $P = \{p_1, \ldots, p_n\}$.
A set of weights $W = \{w_1, \ldots, w_n\}$ where w_i is the weight of p_i.

Output: The WSVT $\mathcal{V}$ of the spherical surface $\mathcal{S}$ according to P and W.

5.1: Let $\mathcal{H}$ be the convex hull of P in R^3. It represents the Delaunay Triangulation of P on $\mathcal{S}$. Note that $\mathcal{H}$ is not weighted.

5.2: Let $\mathcal{V}$ be the Voronoi Tessellation constructed as the dual graph of $\mathcal{H}$: for each triangle in $\mathcal{H}$ its weighted circumcenter is a vertex in $\mathcal{V}$. If two triangles in $\mathcal{H}$ are neighbors then their weighted circumcenters are linked by an edge in $\mathcal{V}$.

5.3: **return** $\mathcal{V}$.

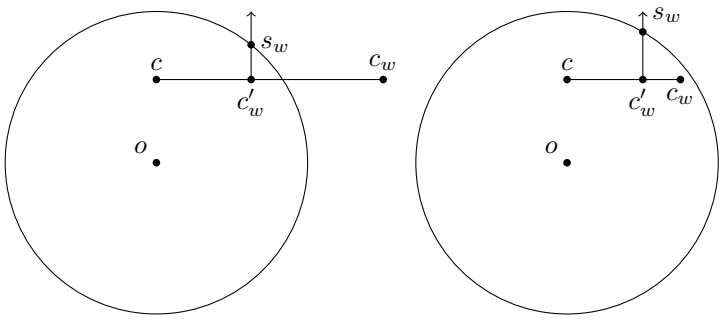


Figure 14: Cross section of the sphere generated by the plane of normal $n\times(c_w-c)$ containing the origin of the sphere. In this cross section it is shown how c_w is mapped into the ball and projected in the surface to reach the spherical weighed cirumcenter.

Weighted Spherical Centroidal Voronoi Tessellation

A Centroidal Voronoi Tessellation (CVT) [Du et al., 1999] is a particular Voronoi Tessellation where the generator of each Voronoi region is the center of mass (centroid) of its own region. A CVT with weighted distance is appropriate to divide a surface into subareas where the size of each one depends on the generator itself and not on the generator's position on the surface. To calculate the Weighted Spherical Centroidal Voronoi Tessellation (WSCVT) it is necessary to introduce the definition of centroid of a spherical triangle and centroid of a spherical polygon. For the next formula we are considering triangles and polygons on a unitary sphere. The centroid of a spherical triangle $\triangle abc$ is $\frac{a+b+c}{|a+b+c|}$. The centroid of a spherical polygon $v_0,\ldots,v_n$ [Jenness, 2008] is

$$\frac{\sum_{i=1}^{n-1}\text{area}(\triangle v_0v_iv_{i+1})\ \ \text{centroid}(\triangle v_0v_iv_{i+1})}{\sum_{i=1}^{n-1}\text{area}(\triangle v_0v_iv_{i+1})},$$

where the area of a triangle $\triangle abc$ is equal to its spherical excess E

$$E=4\tan^{-1}\sqrt{\tan\left(\frac{s}{2}\right)\tan\left(\frac{s-A}{2}\right)\tan\left(\frac{s-B}{2}\right)\tan\left(\frac{s-C}{2}\right)},$$

being A, B and C the side lengths, and s the semiperimeter.

Our WSCVT algorithm is an extension of the one that computes Weighted CVTs in planar surfaces presented in [Balzer and Deussen, 2005]. The general idea of the algorithm is to construct the WSVT of a set of generators and then replace each generator with the centroid of its corresponding Voronoi region, until a desired error or a limited number of iterations has been reached in order

to avoid unnecessary and extensive computation. As the weighted distance is not enough to control the size of each region, it is necessary to adjust the weight of each generator in every iteration. When a region size is bigger than the desired size value, the weight of that generator might be decreased and, analogously, if the region size is smaller, the weight might be increased.

It is important to note that the size values are not areas, but percentages. Making an association between the weights of the generators and the surface of the sphere, the area of a region G_i must represent the same percentage of the entire spherical surface as the weight w_i represents of the overall sum of weights. Then, the desired size value d_i and the actual size value a_i of a region G_i are

$$d_i = \frac{w_i}{\sum w_i} \qquad a_i = \frac{\text{area}(G_i)}{\text{area}(\mathcal{S})}.$$

The algorithm stops when the difference between the desired size value and the actual size value of every region is below a given error ε. The adjusted weight of a generator p_i of current weight w_i, desired size value d_i and actual size value a_i is

$$w_i \left(1 + \frac{a_i - d_i}{d_i}\right) \text{ if this value is greater than } \delta\text{; otherwise, } \delta$$

where δ is a positive value close to 0, for instance 10^{-6}, which avoids points with null weight. Taking into account the weight adjusting and the size value measure, the algorithm to generate a WSCVT is outlined in Algorithm 6.

Algorithm 6 Weighted Spherical Centroidal Voronoi Tessellation (WSCVT)

▷ *Place nodes on the surface of a sphere*

Input: A set of weights $W = \{w_1, \ldots, w_n\}$.

Output: A set of points $P = \{p_1, \ldots, p_n\}$, each point corresponds to the position of a node distributed according W.

6.1: Let P be an initial tentative point distribution on a unitary sphere $\mathcal{S}$. ▷ *Possibly a random distribution*

6.2: Let D be the desired values $\left(d_i = \frac{w_i}{\sum w_i}\right)$

6.3: **while** $\varepsilon_{\max} > \varepsilon$ **do** ▷ *it has not achieve the desired threshold*

6.4: Let $\mathcal{V}$ be the WSVT of P.

6.5: Let $\varepsilon_{\text{actual}}$ be the maximum (or average) difference between desired size values (d_i) and actual size values $\left(\frac{\text{area}(G_i)}{\text{area}(\mathcal{S})}\right)$ of the regions of $\mathcal{V}$.

6.6: **for all** region G_i of $\mathcal{V}$ **do**

6.7: Let $p_i \in P$ be the generator of G_i.

6.8: Replace p_i with the spherical centroid of G_i.

6.9: **return** P.