VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MODERN METHODS FOR TREE GRAPH STRUCTURES RENDERING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. JIŘÍ ZAJÍC
AUTHOR

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MODERNÍ METODY ZOBRAZOVÁNÍ STROMOVÝCH GRAFŮ

MODERN METHODS FOR TREE GRAPH

STRUCTURES RENDERING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JIŘÍ ZAJÍC

VEDOUCÍ PRÁCE
SUPERVISOR

Prof. Dr. Ing. PAVEL ZEMČÍK

BRNO 2013

Abstrakt

Tento projekt se věnuje problematice zobrazení velkých hierarchických struktur, zejména možnostem vizualizace stromových grafů. Cílem je implementace hyperbolického prohlížeče ve webovém prostředí, který využívá potenciálu neeukleidovské geometrie k promítnutí stromu na hyperbolickou rovinu. Velký důraz je kladen na uživatelsky přívětivou manipulaci se zobrazovaným modelem a snadnou orientaci.

Abstract

This project deals with the graphic portrayal of large hierarchies and explores possibilities of tree graphs visualization. The aim is to implement a hyperbolic browser in web environment. The browser shall exploit the potential of non-euclidean geometry and project the tree onto hyperbolic plane. Great emphasis shall be placed on smooth user interface allowing seamless navigation and orientation.

Klíčová slova

stromová struktura, strom, hyperbolický strom, hyperstrom, hyperbolická rovina, hyperbolický prohlížeč, neeukleidovská geometrie, graf, stromový graf, javascript, web, webové prostředí, canvas, svg, raphaeljs

Keywords

tree structure, data tree, hyperbolic tree, hypertree, hyperbolic plane, hyperbolic browser, non-euclidean geometry, graph, tree graph, javascript, web, web environment, canvas, svg, raphaeljs

Citace

Jiří Zajíc: Modern Methods for Tree Graph
Structures Rendering, diplomová práce, Brno, FIT VUT v Brně, 2013

Modern Methods for Tree Graph Structures Rendering

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením pana Prof. Dr. Ing. Pavla Zemčíka.

....
Jiří Zajíc
9. ledna 2013

Declaration

I declare that I have elaborated my semestral project independently, under the supervision of Prof. Dr. Ing. Pavel Zemčík.

....
Jiří Zajíc
January 9, 2013

Poděkování

Rád bych poděkoval vedoucímu semestrálního projektu panu Prof. Dr. Ing. Pavlu Zemčíkovi za odborné vedení, konzultace a podnětné návrhy k práci. Dále bych chtěl poděkovat Ing. Davidu Mikulkovi za konzultace týkající se neeukleidovské geometrie.

Acknowledgement

I would like to thank my semestral project supervisor Prof. Dr. Ing. Pavel Zemčík for his support, supervision and expert advice. My gratitude belongs also to Ing. David Mikulka for advice regarding non-euclidean geometry.

Contents

List of Figures

Chapter 1

Introduction

Since massive expansion of computer science as a tool to process nearly any kind of information, there has always been a need to visualize the information in the nicest and friendliest way possible. Despite the fact, that nowadays there are numerous known technologies and methods available in order to achieve this very essential goal, it can still be quite a challenge to effectively and efficiently display extremely large portions of data. Especially, when interactive manipulation operations for navigating around are necessary.

Consider the relationship between different pieces of information of hierarchical nature. Useful and versatile way to represent such structure in a graphical form would be a tree. Tree with one root and number of nodes connected to each other by exactly one path. It is a fundamental concept that can be applied in computer science (binary search tree) as well as in many other fields, e.g., biology (evolutionary tree), project management (work breakdown structure), linguistics (phrase structure trees), mathematics (Von Neumann universe). Apparently, a big part of all the information in the world can be hierarchically structured.

It is difficult for user to extract information from large hierarchical structures, as the navigation of the structure is often a burden and content information is hidden inside individual nodes. Also, usually only small part of the monitor display is being used. In 1995, J. Lamping and R. Rao from Xerox Palo Alto Research Center came up with an idea of the Hyperbolic Browser. Browser, that can visualize any tree structure by transforming it into, so called, hypertree. Hypertree uses advantages of hyperbolic plane in non-Euclidean Geometry, providing truly unique displaying and browsing experience.

The idea of Hyperbolic Browser is more than seventeen years old, yet its utilization is seen quite rarely. I think that hyperbolic plane based browser has a great potential. This work is motivated by my interest in human machine interfaces and I wanted to contribute to the topic by revealing unusual non-Euclidean hypertree related features. Its aim is to bring such concept into modern web environment by implementing it in pure JavaScript.

In the following chapter, there is summarized history, current state of the art in information visualization and introduction to the related mathematical terms. The third chapter includes requirements analysis of the web environment based hyperbolic browser and mentions couple of topics for further discussion. The fourth chapter proposes a suitable mathematics model for the implementation, that is realized and described in chapter five. The last chapter discusses results.

Chapter 2

State of the Art in Tree Visualization

Since the beginning of the modern computer era[1], variety of approaches to displaying information in graphical form on a computer has been constantly growing and evolving in response to the needs of computer users.

This chapter provides brief history of graphic portrayal and information quantification. It introduces some interesting and useful facts associated with the thesis topic, followed by summary of the essence of visualization. Due to limited scope of work, this chapter is not able to cover all the related information. It only focuses on facts that are important for the reader to become acquainted with. It gives an overview of the current state of the art in information visualization, divided into several categories.

2.1 History of Visualization

The graphic portrayal of quantitative information has deep roots in histories of thematic cartography, statistical graphics and data visualization. These fields are intertwined with each other and date back to 17th century. As true beginning of data visualization can be considered rise of statistical thinking up through the 19th century, and of course developments in technology in the 20th century [3].

Figure 2.1 shows a graphic from 1644, believed to be the first visual representation of statistical data. At that time, lack of a reliable means to determine longitude at sea hindered navigation and exploration. This 1D line graph, showing several known estimates of longitude between Toledo and Rome, is also a milestone as the earliest-known exemplar of the principle of effect ordering for data display [4].

New Graphic Forms

In the 18th century, abstract graphs and graphs of functions were introduced, together with the early beginnings of statistical theory (measurement error) and systematic collection of empirical data. William Playfair (1759–1823) is widely considered the inventor of most of the graphical forms widely used today:

- Line graph and bar chart (1786)

[1]As the beginning of a modern computer era is considered year 1939, when Hewlett-Packard is founded [8].

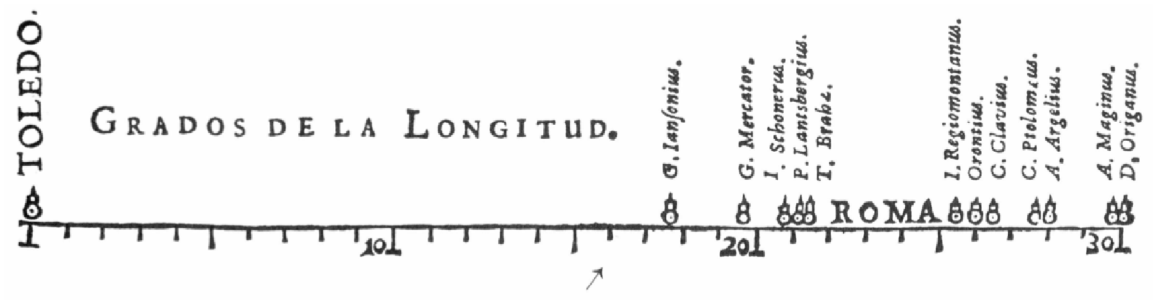


Figure 2.1: Langren's 1644 Graph of the Distance

- Pie chart and circle graph (1801)
- Mixed series graph (1821; shown in Figure 2.2)

Beginnings of Modern Graphics

The early 19th century witnessed explosive growth in statistical graphics and thematic mapping, at a rate which would not be equalled until modern times. In statistical graphics, all of the modern forms of data display were invented: bar and pie charts, histograms, line graphs and time-series plots, contour plots, scatterplots, and so forth.

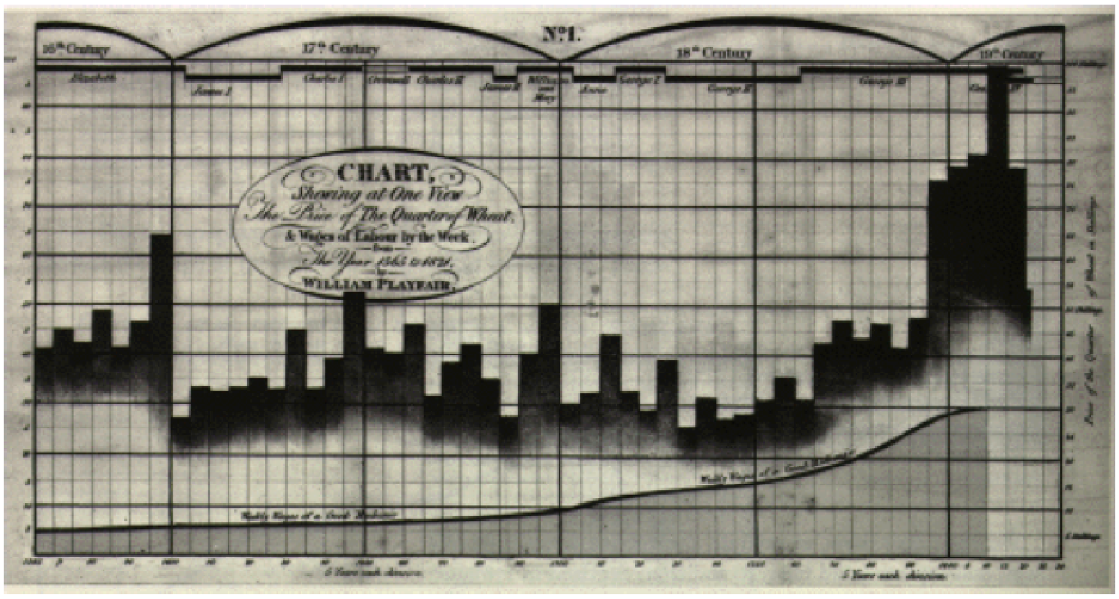


Figure 2.2: Playfair's 1821 Time Series Graph

In the second half of 19th century, statistical theory was initiated by Gauss and Laplace and provided the means to make sense of large bodies of data. For the first time, graphical methods proved crucial in a number of scientific discoveries (e.g. the discovery of atomic number by H. Mosely).

Re-birth of Data Visualization

Finally, computer processing of data had begun, and offered the possibility to construct old and new graphic forms by computer programs. True high-resolution graphics were developed, but would take a while to enter common use.

By the end of this period significant intersections and collaborations would begin: (a) computer science research (software tools, C language, UNIX, etc.) at Bell Laboratories (Becker, 1994) and elsewhere would combine forces with (b) developments in data analysis (EDA, psychometrics, etc.) and (c) display and input technology (pen plotters, graphic terminals, digitizer tablets, the mouse, etc.). These developments would provide new paradigms, languages and software packages for expressing statistical ideas and implementing data graphics. In turn, they would lead to an explosive growth in new visualization methods and techniques.

History Milestones

Contributions to the development and use of graphic forms. Instatistical graphics, inventions of the bar chart, pie chart, line plot (all attributed to Playfair), the scatterplot (attributed to J.F.W. Herschel; see Friendly and Denis (2004)), 3D plots (Luigi Perozzo), boxplot (J. W. Tukey), and mosaic plot (Hartigan & Kleiner) provided new ways of representing statistical data.

Technology and enablement. It is evident that many developments had technological prerequisites, and conversely that new technology allowed new advances that could not have been achieved before. These include advances in rendering regarding computing and video display.

Theory and data on perception of visual displays. Graphic displays are designed to convey information to the human viewer, but how people use and understand this form of communication was not systematically studied until recent times. As well, proposals for graphical standards, and theoretical accounts of graphic elements and graphic forms provided a basis for thinking of and designing visual displays.

Implementation and dissemination. New techniques become available when they are introduced, mainly of implementations of graphical methods in software. But additional steps are needed to make them widely accessible and useable.

2.2 Mathematical Terms

The term visualization is basically any technique for creating images, diagrams, or animations to communicate a message. Before moving on to computer based visualization, it might be helpful to define other terms as they are crucial for understanding following chapters.

Since this work deals, among others, with proposing a suitable model for rendering large tree structures in hyperbolic plane, it is necessary to define terms *tree structure*, *hyperbolic plane*, *rendering*, and some other.

Graph

In mathematics and computer science, graph theory is the study of graphs, which are mathematical structures used to model pairwise relations between objects from a certain

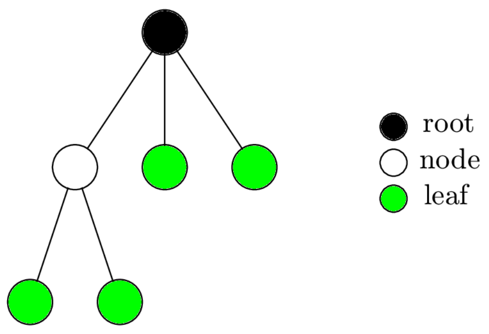


Figure 2.3: Tree Data Structure

collection. A “graph” in this context is a collection of “vertices” (or “nodes”) and a collection of “edges” that connect pairs of vertices. It is either undirected, which means that there is no distinction between the two vertices associated with each edge whatsoever, or its edges may be directed from one vertex to another [12].

Graphs can be used to model many types of relations in many different systems. Also, some practical problems can be represented by graphs. In mathematics, more specifically in graph theory, one of the most important nonlinear structures is a tree. It is an undirected graph in which any two vertices are connected by exactly one simple path. In other words, any connected graph without cycles is a tree.

Tree Structure

Generally speaking, tree structure means a “branching” relationship between nodes, much like that found in the trees of nature [6]. In order to describe a tree, definition of structure is needed first.

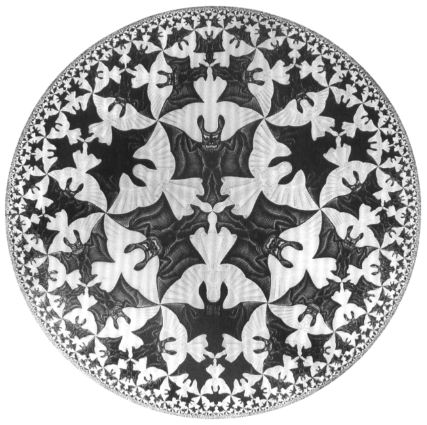
A structure, in computer science and in terms of this paper more accurately a data structure, is particular way of storing and organizing data in a computer, so that it can be used efficiently [11].

Now, tree is a data structure, accessed beginning at the root node. Each node is either a leaf or an internal node. An internal node has one or more child nodes and is called the parent of its child nodes. All children of the same node are siblings. Contrary to a physical tree, the root is usually depicted at the top of the structure, and the leaves are depicted at the bottom. It is a connected, undirected, acyclic graph and it is rooted and ordered unless otherwise specified [1]. Mutual recursive definition would be

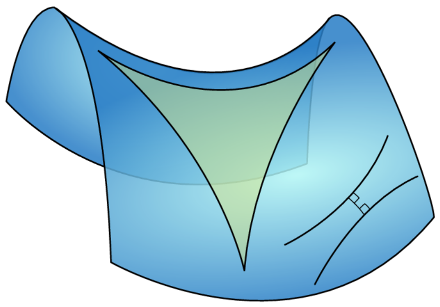
$$f\colon [t[1], t[2], \ldots, t[n]], \tag{2.1}$$
$$t\colon vf, \tag{2.2}$$

where $t$ree is a $f$orest (list of $t$rees), where $t$ree consists of $v$alue and possibly another $f$orest.

Different kinds of data structures are suited to different kinds of applications, and some are highly specialized to specific tasks. The tree structure is perfectly suitable for dealing with model proposal for rendering large hierarchies. Figure 2.3 shows the very simple concept.



(a) Planar Representation of Hyperbolic Plane



(b) Hyperbolic Triangle

Figure 2.4: Hyperbolic Plane Figures

Hyperbolic Plane and Geometry

In mathematics, hyperbolic geometry (also called Lobachevskian geometry or Bolyai-Lobachevskian geometry) is a non-Euclidean geometry, meaning that the parallel postulate of Euclidean geometry is replaced. The parallel postulate in Euclidean geometry is equivalent to the statement that, in two dimensional space, for any given line R and point P not on R, there is exactly one line through P that does not intersect R; i.e., that is parallel to R. In hyperbolic geometry there are at least two distinct lines through P which do not intersect R, so the parallel postulate is broken. Models have been constructed within Euclidean geometry that obey the axioms of hyperbolic geometry, thus proving that the parallel postulate is independent of the other postulates of Euclid (assuming that those other postulates are in fact consistent).

A characteristic property of hyperbolic geometry is that angles of a triangle add to less than a straight angle. In the limit as the vertices go to infinity, there are even ideal hyperbolic triangles in which all three angles are $0°$ [13].

Models of Hyperbolic Plane

There are four models commonly used for hyperbolic geometry: the Klein model, the Poincaré disk model, the Poincaré half-plane model, and the Lorentz model (also called Hyperboloid model). These models define a real hyperbolic space which satisfies the axioms of a hyperbolic geometry. Despite their names, the first three mentioned above were introduced as models of hyperbolic space by Beltrami, not by Poincaré or Klein.

The Klein model, also known as the projective disc model and Beltrami-Klein model, uses the interior of a circle for the hyperbolic plane, and chords of the circle as lines.

- This model has the advantage of simplicity, but the disadvantage that angles in the hyperbolic plane are distorted.

- The distance in this model is the cross-ratio, which was introduced by Arthur Cayley in projective geometry.

A non-Euclidean geometry, also called Lobachevsky-Bolyai-Gauss geometry, having constant sectional curvature -1. This geometry satisfies all of Euclid's postulates except the parallel postulate, which is modified to read: For any infinite straight line L and any point P not on it, there are many other infinitely extending straight lines that pass through P and which do not intersect L.

In hyperbolic geometry, the sum of angles of a triangle is less than 180°, and triangles with the same angles have the same areas. Furthermore, not all triangles have the same angle sum. There are no similar triangles in hyperbolic geometry. The best-known example of a hyperbolic space are spheres in Lorentzian four-space. The Poincaré hyperbolic disk is a hyperbolic two-space. Hyperbolic geometry is well understood in two dimensions, but not in three dimensions.

In Web development jargon and information visualization, a hyperbolic tree (often shortened as hypertree) defines a graph drawing method inspired by hyperbolic geometry. Geometric models of hyperbolic geometry include the Klein-Beltrami model, which consists of an open disk in the Euclidean plane whose open chords correspond to hyperbolic lines. A two-dimensional model is the Poincaré hyperbolic disk. Felix Klein constructed an analytic hyperbolic geometry in 1870 in which a point is represented by a pair of real numbers (x_1, x_2) with

$$x_1^2 + x_2^2 < 1 \tag{2.3}$$

(i.e., points of an open disk in the complex plane) and the distance between two points is given by

$$d(x, X) = a \cosh^{-1}\left[\frac{1 - x_1X_1 - x_2X_2}{\sqrt{1 - x_1^2 - x_2^2}\sqrt{1 - X_1^2 - X_2^2}}\right] \text{ [10].} \tag{2.4}$$

2.3 Known Methods and Existing Software

Thorough research on available tree visualizers divided implementations into two basic categories: Java based applications and applets, and JavaScript libraries and frameworks. Implementations in C/C++ and other languages exist too, but they are less suitable for web environment.

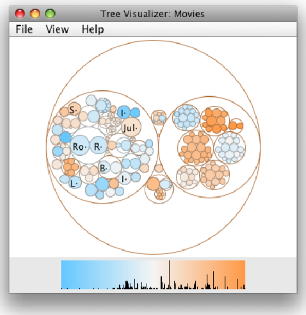
Java

Among many implementations for visualizing tree structures in Java, the most sophisticated seems *Treeviz*. It can be downloaded as a JNLP (Java Network Launching Protocol) file[2]. JNLP extension allows the program to be distributed and managed as a standalone application via the Internet.

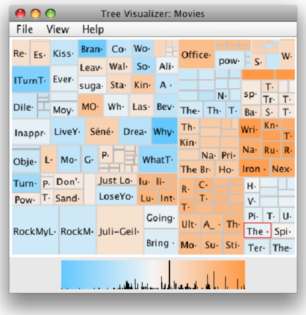
Treeviz has fairly simple, user-friendly interface. It allows to choose from following different ways of model view: Circular Treemap, Rectangular Treemap, Sunburst Tree, Icicle Tree, Sunray Tree, Iceray Tree, and Hyperbolic Tree. Example of each model view can be seen in Figure 2.5, individual images are self-explanatory. Treeviz is intented to display primarily directory structures of a computer's HDD (hard disk drive), but it can also read XML files in a specific format.

There is one interesting fact about licensing of Treeviz software. All its source code is copyrighted by Werner Randelshofer, but the only Hypertree code is licensed under the MIT licence. More on Hypertree licensing and patent can be found in Appendix A.

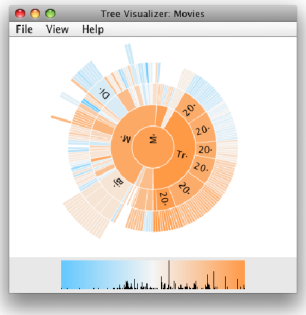
[2]Available at `http://www.randelshofer.ch/treeviz/`.



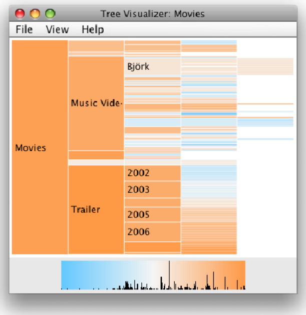
(a) Circular Treemap



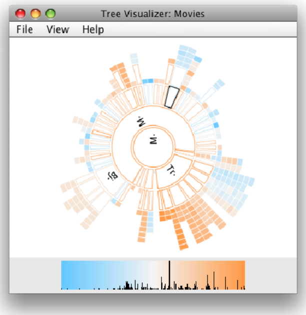
(b) Rectangular Treemap



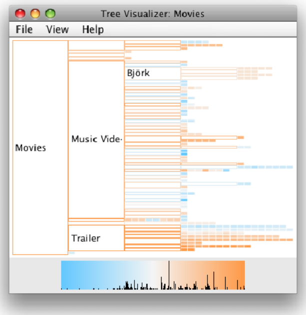
(c) Sunburst Tree



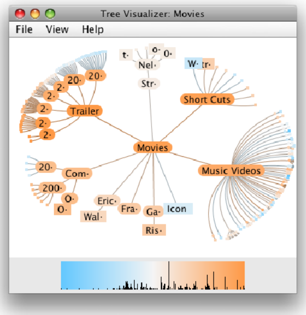
(d) Icicle Tree



(e) Sunray Tree

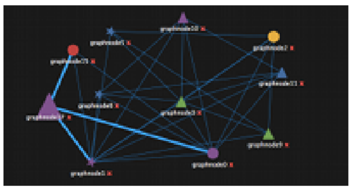


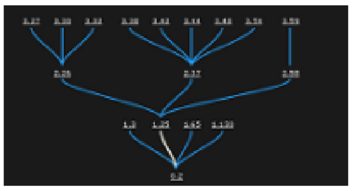
(f) Iceray Tree



(g) Hyperbolic Tree

Figure 2.5: Different Tree View Models
[source: http://www.randelshofer.ch/treeviz/]


(a) ForceDirected


(b) Spacetree

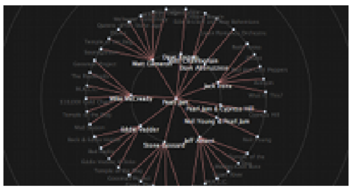

(c) RGraph

Figure 2.6: Different Tree View Models
[source: http://philogb.github.com/jit/demos.html]

JavaScript

Implementations of a hyperbolic browser in JavaScript are less common than in Java. Altough, there are obviously more JavaScript frameworks and libraries for graph visualization in general (e.g. Highcharts JS, ExtJS, GraphUp).

The JavaScript tree visualizer project with the best search engine optimization is obviously a *JavaScript InfoVis Toolkit* (Jit). When "hypertree in javascript" is googled, no other implementations can be found at the first three pages of results[3], but Jit. It can be downloaded [4] as a single-file JavaScript library.

Jit is copyrighted by SenchaLabs, its author is Belmonte, N. G. Besides its capability of drawing area, bar and pie charts, it can visualize trees in the following ways: Sunburst, Icicle, TreeMap, HyperTree, ForceDirected, SpaceTree, RGraph. Last three types are shown in Figure 2.6.

[3]Dated to January 1st, 2013.

[4]Available at: http://philogb.github.com/jit/demos.html.

Chapter 3

Features Evaluation and Requirements Analysis

This chapter evaluates current state of the art. It tries to constructively criticize available software in order to design the best possible requirements for hyperbolic browser runnable in modern web environment[1].

3.1 Evaluation

When speaking of Java powered Treeviz described above, the core implementation of hyperbolic tree view is at very high level. After precomputing the layout, everything seems very smooth. Navigation within the displayed model of more than 1000 nodes does not have impact on user experience at all. What user might find disappointing is the fact, that there are almost no configuration options. Nodes can not be expanded/collapsed, there are no hover or tooltip effects, labeling is not adjustable at all, and reset button that would take you back to the centered root is missing. Unfortunately, when there are too many children nodes, undesirable "filled circle effect"[2] appears.

Jit's implementation of hypertree view model in JavaScript is rather poor. The user interface does not even offer a custom manipulation with the model; only clicking a particular node brings it to the center of the canvas. Latest web technologies offer much more than that.

3.2 Web Environment

Web application development has become very popular during the last two decades. Software development related to web allows users to enhance their experience by using variety of features on a computer as well as on TV, game console, tablet or even on a mobile phone. Such an approach has many advantages, especially when used within company intranet with no need for access to the Internet:

Zero install
Web browser already available within every modern operating system

[1] Modern web environment means browser supporting HTML5 and respecting WWW standards.

[2] Filled circle effect occurs when there are to many edges connecting a parent node with its children. When rendered on a small canvas, they actually fill the circle around the parent.

Zero maintenance
User is not the administrator

Always up-to-date
Immediate “automatic” update when new version is released

Low hardware requirements
Not entirely true, but generally web applications can be run on slower portable devices, unlike standalone applications

Cross platform compatibility
Possible differences only between browsers, not between operating systems

Portability
Since no installation, it can be run from any place in the world[3]

Direct access to the latest information
Immediate update in case of releasing new versions

Surely, the web environment has some disadvantages too, e.g. interfaces are often not as sophisticated, it can take longer to develop such apps as they are more complex, a need for different browsers support, security risks... However, some of these disadvantages do not apply to the company intranet deployment (security risk is lower, standard browser is recommended etc.), and some are simply worth overcoming.

3.3 Requirements Analysis

According to evaluated features in current state of the art applications, and also according to nowadays advantages and/or demands for web applications, I decided to implement hyperbolic browser named **Gr**aphs **In** **C**anvas **H**yperbolic, simply called GrInCH.

Purpose of GrInCH is to enable graphical representation of large tree structures and to allow fast movement in its data model. Application shall be able to load and display a general tree structure (as defined in Chapter 2.2), and present it as graph projected onto hyperbolic plane. This graph shall be visualized and viewable in general modern web browser as part of the website with use of HTML5 technology.

Software requirements include both general as well as interface related requirements:

GrInCH Requirement 001:

(a) GrInCH shall be written in JavaScript and shall be delivered as a single-file JavaScript object.

(b) GrInCH application shall be configurable at least in this extent:

i. Width, height,

ii. XML file upload,

iii. Debug mode on/off.

GrInCH Requirement 002:

(a) GrInCH shall perform smoothly to ensure positive user interface.

[3]Assuming the Internet connection is available.

(b) GrInCH shall handle XMLs with 1 to 500 nodes without any significant impact on user interface smoothness.

(c) GrInCH shall eliminate a “filled circle effect”, when necessary.

GrInCH Requirement 003:

(a) GrInCH shall bring “Save as picture” dialogue when user clicks on SAVE button.

(b) GrInCH shall have means to save user’s current view.

(c) GrInCH shall have means to save default view with the root centered in the middle.

(d) GrInCH shall have means to save the view as both raster and vector.

GrInCH Requirement 004:

(a) When mouse hovers over a node, the node shall be highlighted.

(b) When user clicks an collapse symbol on a node, the node shall hide all its descendants.

(c) When user clicks an expand symbol on a node, the node shall show all its children.

GrInCH Requirement 005:

(a) Font style and font family of GrInCH shall be customizable using CSS.

(b) GrInCH shall provide user interface, that allows to switch from current view to initial view (tree root in the center of canvas).

(c) GrInCH shall provide minimalistic, easy and intuitive design.

GrInCH Requirement 006:

(a) GrInCH shall be able to load initial data in both XML and JSON file format, as defined in section 5.1.

3.4 Other Features

Besides provided requirements, there are also possible features or problems, that can be evaluated and resolved after the implementation of the basic core, such as:

Nodes labeling
Algorithm for positioning the labels of the nodes

AJAX loading
Dynamic loading of nodes when necessary, e.g. after expanding a collapsed node

Anti-aliasing
Ability to minimize the distortion of artifacts

Touch screen optimization
Optimization for touch screen devices, such as tablets, phones, etc.

Mobile phone app
Applications for Google Play[4] and Apple App Store

[4]Formerly known as Android Market

Custom styling
: Ability to apply custom CSS styles

Magnifying glass
: Magnifying glass effect to resolve “filled circle” effect

All necessary information regarding both mandatory and optional features, requirements and web limitations were collected and summarized. Last thing to do, before implementation can be realized, is to go through and analyze the mathematics model of hyperbolic browser.

Chapter 4

Mathematics Model Proposal

This chapter explains in detail the role of hyperbolic plane inside the application, and presents the Poincaré model in conjuction with Klein-Beltrami open disk. It helps the reader to better understand the nature of hyperbolic geometry on which the application is built.

The open unit disk is commonly used as a model for the hyperbolic plane, by introducing a new metric on it, the Poincaré metric. The Poincaré disk is a conformal model of hyperbolic space, i.e. angles measured in the model coincide with angles in hyperbolic space, and consequently the shapes of small figures are preserved[14].

Procedure of transforming tree into hypertree and then to the unit disk is quite complicated. Therefore it is necessary to divide the problem into several simpler sub-problems. Once the structure is loaded, verified and parsed, all the information is stored within the core of the application/library that can now start working with it. First step is to lay out all nodes onto the hyperbolic plane, which is described in detail in section 4.1. The result of this layout is that each node knows its wedge, angle and distance from its parent.

Once this is done, nodes need to be mapped onto the unit disk. This is because it is not simply possible to display the hyperbolic plane in just two dimensions on the screen. Mapping uses the Poincaré model and is described in Section 2.2. Once mapping is done, each node has its x and y coords of the unit disk.

4.1 Layout onto Hyperbolic Plane

When tree is loaded from XML or JSON, the layout is being computed. The mathematical model of hyperbolic geometry is used. Hyperbolic geometry (also called Lobachevskian geometry) is non-Euclidean geometry, meaning that the parallel postulate of Euclidean geometry is redefined. The parallel postulate of Euclidean geometry says that, in two dimensional space, for any given line l and point P not in l, there is one and only one line k through the point P which has no intersection with line l, i.e., lines l and k are parallel.

In hyperbolic geometry the parallel postulate is not fulfilled. Simply put, in Euclidean geometry two parallel lines have still the same distance between themselves. In the hyperbolic geometry distance between these two lines distance grows exponentially, the lines diverge. That means, that if the distance grows, the space between the parallel lines grows too. Having the circle in the hyperbolic plane, its circumference and space grow exponentially with its radius.

There is an abstraction that each node has a given wedge of the circle used in GrInCH.

This wedge is used to lay out the node and its children in it. The node itself is placed at the vertex of its wedge. The angle of the wedge is divided by the number of the node's children, so every child has its own subwedge, inside of it is placed, and its children are placed in it as well. So the node's children are placed along an arc in the node's wedge at an equal distance from the node, and are placed in the middle of theirs subwedges. Because the lines diverge as mentioned above, the subwedge of each node's child can span the same angle of the wedge, as the node's wedge owns and no sibling subwedges can overlap. There are operations like moving some distance and turning through some angle used for computing the children's position relatively to its parent [7].

There is two dimensional space used for layout and visualization, so the complex numbers have to be used for the representation of the point on the plane.

The layout function is called recursively for each node. It starts from the root node and goes through all its children. If the node has n children, its wedge is divided to n equal subwedges for each child. The distance from the node to his child is computed by

$$d = \sqrt{\left(\frac{(1-s^2)\,\sin(a)}{2s}\right)^2 + 1} - \frac{(1-s^2)\,\sin(a)}{2s}, \tag{4.1}$$

where a is an angle between midline and edge of the subwedge, and s is the desired distance between the child and the edge of its subwedge. d is computed distance. All values are scalars; d and s are represented as hyperbolic tangent of the distance in the hyperbolic plane [7].

Given the subwedge for the child and the distance to it, it is possible to compute the child's wedge inside the subwedge, as seen in Figure 4.1. Given the subwedge's vertex p, midline endpoint m and angle of the subwedge a, computed wedge of the child will be calculated using the transformation

$$z_t = Trans(z, \langle P, \theta\rangle) \tag{4.2}$$

$$z_t = \frac{\theta z + P}{1 + \overline{P}\theta z}, \tag{4.3}$$

where the new position z_t is the old position z rotated by angle θ and moved to P.
Vertex calculation:

$$p' = Trans(dm, \langle p, 1\rangle) \tag{4.4}$$

$$p' = \frac{dm + p}{1 + d\overline{m}\overline{p}}. \tag{4.5}$$

Middle endpoint:

$$m' = Trans(Trans(m, \langle p, 1\rangle), \langle -p', 1\rangle) \tag{4.6}$$

$$m' = Trans\left(\frac{m+p}{1+\overline{p}m}, \langle -p', 1\rangle\right) \tag{4.7}$$

$$m' = \frac{\frac{m+p}{1+\overline{p}m} - p'}{1 + \overline{-p'}\frac{m+p}{1+\overline{p}m}}. \tag{4.8}$$

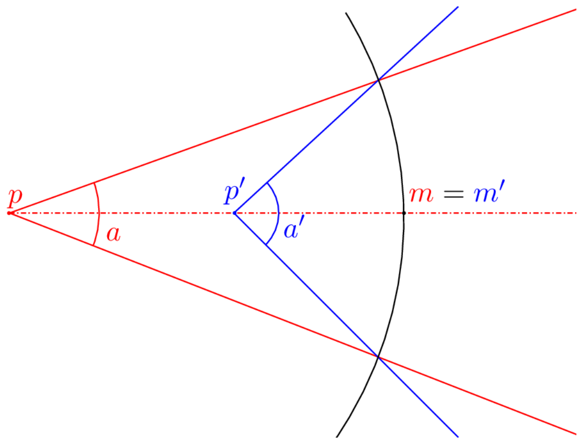


Figure 4.1: Principle of Wedge Computation

Angle of the wedge:

$$a' = \Im\left(\log\left(Trans\left(e^{ia}, \langle -d, 1\rangle\right)\right)\right) \tag{4.9}$$

$$a' = \Im\left(\log\left(Trans\left((\cos a + i\sin a), \langle -d, 1\rangle\right)\right)\right) \tag{4.10}$$

$$a' = \Im\left(\log\left(\frac{(\cos a + i\sin a) - d}{1 + \overline{-d}(\cos a + i\sin a)}\right)\right), \tag{4.11}$$

where $\Im\log(\ldots)$ returns an angle of the wedge converted from the complex number returned from the transformation. The number e^{ia} is the complex number in trigonometric form

$$re^{i\alpha} = r(\cos\alpha + i\sin\alpha), \tag{4.12}$$

and for this case $r = 1$ simply

$$e^{ia} = \cos a + i\sin a. \tag{4.13}$$

4.2 View Projection Initialization

When the layout function is finished, the tree graph is laid out on the hyperbolic plane. Every node stores its own position that is permanent. It is never changed in the future, unless the graph structure (XML or JSON) is modified.

Before the whole graph is rendered on the canvas, the current view projection initialization is needed. There is a request to have the root node in the center of the canvas, so the initial view projection $\langle P, \theta\rangle$ is set to $\langle 0, 1\rangle$.

4.3 Rendering

There is a mapping from the hyperbolic plane to the Euclidean two dimensional plane used for visualizing the graph. There is a mathematical theory of the Poincaré model of mapping from the hyperbolic plane to the unit disk. The nearest neighborhood of the selected point on the hyperbolic plane is in the focus and further neighborhood fades off toward the edge of the unit disk. The special features of the Poincare model is that it preserves angles and the lines in the hyperbolic plane are changed to the arcs on the unit disk that evokes a perception of the hyperbolic plane.

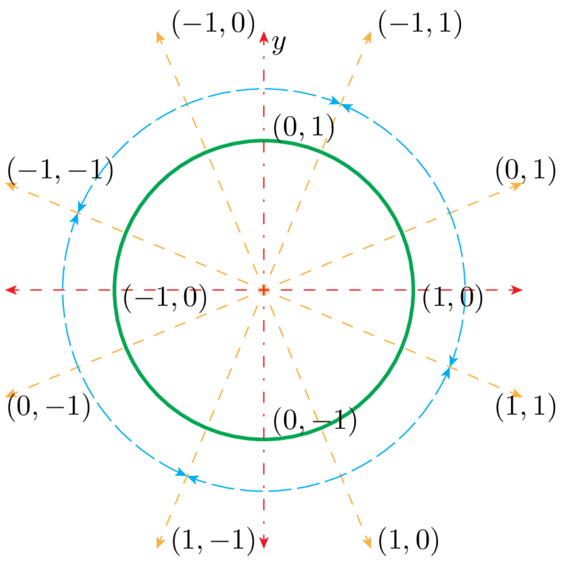


Figure 4.2: Unit Disk Positionig and Rotation

As seen in the picture 4, two main properties of the mapping the hyperbolic plane to the unit disk are that the parts of the image decrease its size and its count grows exponentially with their distance from the center of the unit disk.

There is the recursive node render method call used for rendering the whole graph. The algorithm uses a Depth-First Search (DFS) methodology for traversing the graph. It starts from the root node and explores as far as possible along branch before backtracking [3]. The flow diagram of the rendering is depicted in the picture 5.

At first, node's position in the hyperbolic plane is mapped to the unit disk. The defined transformation under the current view projection is used for it. There are actions like vector rotation, moving, translation and scaling used in mapping algorithm. If we have the node position on the unit disk, the scaling of the position is performed to preserve follow the actual canvas size, because the canvas is not the square, but mostly the rectangle with different width and height, so the unit disk is an ellipse instead of the circle in that cases.

Then the visibility of the node and its label is computed. Visibility of the node and its label is not the same. There are different threshold for their computation. The threshold of the whole node is less than the threshold of the node's label visibility. Visibility of the whole node is significant for rendering the edges of the graph. Edges are thin arcs or lines so more edges can be rendered on the canvas and the distance between two edges can be very small and the user can recognize these edges very well. The situation about nodes' labels is different. Labels contains icon and the node's description, so much more area for its display is needed. So there are three areas of the unit disk. The first area, the biggest of all is the one the nodes' labels and the whole nodes are visible. The whole nodes' are and their labels are not visible in the second area. There are the whole nodes and their labels not visible at all in the third area. This equipment has no effect to the user's perception of the graph structure; the visualization is clearer respectively.

If the current node has no children, the current node is the leaf of the tree and there is nothing to do again. Otherwise there are children who must be rendered; the operations described by next paragraphs will be iterated for each child.

If the number of children is bigger than one hundred; children rendering optimization is needed. The graph visualization may not be providing an easy survey without the optimization. The distance between sibling children may be very small so the user may not be able recognizing the context. The optimization is based on the theory that each child should not be displayed without the context loss. The number of displayed children which are nearer the focus is bigger than the number of displayed children which are further the focus. Every xth child display is skipped relatively to the distance from the focus.

When the current child should be rendered, recursive call is performed to render child's subtree. Then the edge from the child to its parent is rendered.

After the iteration over the node's children was finished, node's area the label should be displayed in is computed. The computation node's label area is very important to prevent the node labels overlapping.

If it is realized the node has some visible children, their rendering must be performed. Their label's area must be recomputed and optimized to prevent an overlapping with parent node and than their icons and labels can be rendered.

4.4 Changing View Projection

Changing the view projection is performed when user interacts the application. When the user clicks on another position to make the focus on this point or drags the mouse to make continuous focusing, view projection must be changed to get requested focus.

To change the view projection, the inverse mapping of the selected point on the canvas, i.e., unit disk must be computed to get corresponding point in the hyperbolic plane. The inverse mapping uses inverse transformation of the current transformation of the actual view projection.

Given the current transformation $\langle P, \theta \rangle$, the inverse transformation can be computed by

$$P' = -\overline{\theta} P \quad \theta' = \overline{\theta}, \tag{4.14}$$

so the point in the hyperbolic plane is computed by

$$z_t = Trans(z, \langle P', \theta' \rangle), \tag{4.15}$$

where z_t is the point on the hyperbolic plane and z is point selected in the unit disk.

The new graph's visualization is achieved by the composition of the current view projection and the new view projection. The composition $\langle P, \theta \rangle$ of the transformations $\langle P_1, \theta_1 \rangle$ and $\langle P_2, \theta_2 \rangle$ is given by

$$P = \frac{\theta_2 P_1 + P_2}{\theta_2 P_1 \overline{P_2} + 1} \quad \theta = \frac{\theta_1 \theta_2 + \theta_1 \overline{P_1} P_2}{\theta_2 P_1 \overline{P_2} + 1} \ . \tag{4.16}$$

Chapter 5

Implementation

After thorough description and explanation of the mathematics model in the previous chapter, the application itself can now be implemented. Among the requirements listed in section 3.3, it might be useful to sum up all the demands; not only from system or software point of view, but also from architectural and functional perspective.

Application shall contain the top bar with heading, logo and control elements, rest of the viewport shall be dedicated to displaying the graph, see the mockup design in Figure 5.1. Application shall allow user free rotation and seamless navigation within the tree. Application shall also be apparent that there were increased focus on human factors. User interface shall be smooth, clean, simple, modern and minimalistic. Touch device ready, and of course, intuitive and easy to use.

The library itself shall be contained in a single file and installation shall be as simple as including the file within a valid HTML5 web page and calling the constructor of the GrInCH class. Application shall support all modern browsers without any compatibility issues or usage restrictions.

5.1 Technologies

The architecture of hyperbolic browser is neither complicated nor confusing, yet it is still worth categorizing the available solutions for each part of the concept. To simply and elegantly achieve all that is stated above, it is quite crucial to carefully choose appropriate languages and technologies for each individual task.

In this section there are covered three fundamental parts of the implementation. Firstly, I discuss and describe possibilites of how to store the data, that GrInCH visualizes. Then I focus on what to use to draw in a browser, and finally the topic of implementing the core is covered.

Data Storage

GrInCH shall be able to read several data formats for different purposes. Because it will be implemented and run in web environment, there are couple of available options right there. I should mention I would always prefer open source solutions, since they are standardized, well documented and thoroughly discussed in many forums all over the world, hence they are also easy to use. And, well, they are free of charge. The first web-related technology to store the data that came to my mind was XML.

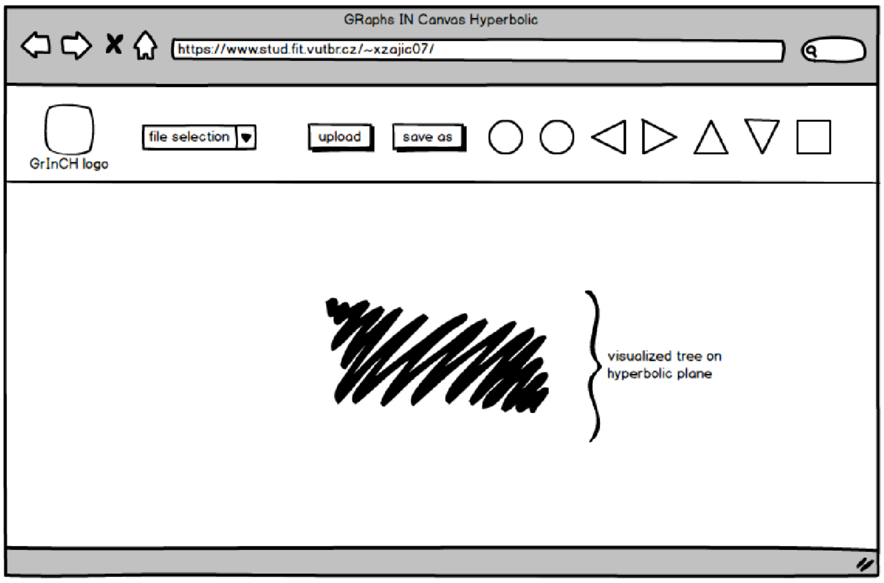

GRaphs IN Canvas Hyperbolic
https://www.stud.fit.vutbr.cz/~xzajic07/
GrInCH logo
file selection
upload
save as
visualized tree on hyperbolic plane

Figure 5.1: GrInCH Mockup Design

XML is a well-supported Internet standard for encoding structured data. It can be easily decoded by practically any programming language and also read or written by humans using standard text editors. Many applications, especially modern standards-compliant web browsers, can deal directly with XML data. Therefore I decided to use it as one of the main formats to store the nodes and edges in.

Supported XML design, structurally corresponding with figure 2.3, including some additional attributes, is shown in code in Listing 5.1. I kept it quite simple, there are, among the root node and the node holding the graph file name, only two basic nodes called simply `node` and `edge` .

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE GraphXML>
<GraphXML>
        <graph filename="vut-structure.xml">
        <node nodeId="0">
            <label>Father</label>
        </node>
        <node nodeId="1" collapsed="true">
            <label>Son A1</label>
        </node>
        <node nodeId="2">
            <label>Son A2</label>
        </node>
        <node nodeId="3">
            <label>Son A3</label>
        </node>
        <node nodeId="4">
```

```
            <label>Grandson B1</label>
        </node>
        <node nodeId="5">
            <label>Grandson B2</label>
        </node>
        <edge source="0" target="1"/>
        <edge source="0" target="2"/>
        <edge source="0" target="3"/>
        <edge source="1" target="4"/>
        <edge source="1" target="5"/>
    </graph>
</GraphXML>
```

Listing 5.1: A Simple Tree Structure Stored in XML

For any web application, it is not possible to read directly from a file located on a user's hard-drive. This is because of client-side browser security restrictions. File must be uploaded to server before handling. Cookies could be used to deal with this issue instead, but not all users have them turned, there are also size limitations for each browser and file would have to be uploaded at least once for the very first time, so this is probably not a very good workaround. Also, the purpose of cookies is to be of benefit to the user, not to application as the main storage point[5]. Furthermore, sometimes it is not desirable to run the application on a server (e.g. demonstration purposes), therefore GrInCH shall also be able to read some other, client-side only format, preferably natively JavaScript compliant.

JSON is a text-based open standard designed for data interchange in a human-readable form and it is directly derived from JavaScript for representing simple data structures and associative arrays, in JavaScript generally called objects. Despite its direct relationship to JavaScript, it is language-independent, with parsers available for many languages[1].

Since XML file has to be uploaded, it makes sense to perform the parsing on the server as well. Here I chose PHP to be in charge of this task. More specifically I used PHP `SimpleXMLElement`[2] class. After calling for new `SimpleXMLElement` , while passing the XML file as the first parameter to the constructor, the contents of the file can be type-casted[3] into an `array` and converted to JSON string using `json_encode` function. The output, that is perfectly human-readable and GrInCH compliant, can be seen in code in Listing 5.2 below.

```
{
  node: [
    {"@attributes": {nodeId: 0}, label: "Father"},
    {"@attributes": {nodeId: 1}, label: "Son A1",
        collapsed: true},
    {"@attributes": {nodeId: 2}, label: "Son A2"},
    {"@attributes": {nodeId: 3}, label: "Son A3"},
    {"@attributes": {nodeId: 4}, label: "Grandson B1"},
    {"@attributes": {nodeId: 5}, label: "Grandson B2"}
  ];
  edge: [
```

[1]JavaScript parser is available e.g. in Java, C#, C++, PHP, Python.
[2]http://php.net/manual/en/class.simplexmlelement.php
[3]Type-casting in PHP: http://php.net/manual/en/language.types.type-juggling.php.

```
    {"@attributes": {source: 0, target: 1}},
    {"@attributes": {source: 0, target: 2}},
    {"@attributes": {source: 0, target: 3}},
    {"@attributes": {source: 1, target: 4}},
    {"@attributes": {source: 1, target: 5}}
  ];
};
```

Listing 5.2: Simple Tree Structure Stored in JSON

GrInCH will most likely need to know asynchronous loading of the nodes as part of optimalization for better performance. The ability to read the data structure from a relational database might be very useful and memory-saving. When reading from XML, the whole file needs to be uploaded and parsed, then the desired data need to be found and finally transformed into GrInCH's internal representation and used. Whilst when reading from an appropiately designed database, only the needed nodes and edges can be loaded with no useless overhead data or already known ballast.

I decided to go with MySQL, as it is the world's most popular open source database software[2]. Simple and suitable ERD in UML representing MySQL dialect could look something like figure 5.2 and could be easily created executing query 5.3.

```
CREATE TABLE `nodes` (
  `id` int(11) NOT NULL,
  `label` varchar(100) NOT NULL,
  `collapsed` smallint(2) NOT NULL DEFAULT 0,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB;

CREATE TABLE `edges` (
  `source` int(11),
  `target` int(11),
  FOREIGN KEY (`source`) REFERENCES `nodes`(`id`),
  FOREIGN KEY (`target`) REFERENCES `nodes`(`id`)
) ENGINE=InnoDB;

INSERT INTO `nodes` (`id`, `label`, `collapsed`) VALUES
(0, 'Father', 0), (1, 'Son A1', 1),
(2, 'Son A2', 0), (3, 'Son A3', 0),
(4, 'Grandson B1', 0), (5, 'Grandson B2', 0);

INSERT INTO `edges` (`source`, `target`) VALUES
(0, 1), (0, 2), (0, 3), (1, 4), (1, 5);
```

Listing 5.3: Simple Tree Structure in MySQL Database

Canvas

The newest markup language for structuring and presenting content for the World Wide Web and a core technology of the Internet, HTML5, brought a new way of drawing in a web browser. The element to draw on is called `canvas` and it is accessed by JavaScript language. It looks like a decent choice for GrInCH.

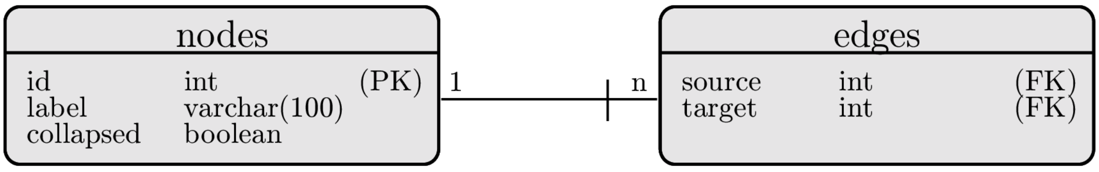


Figure 5.2: Simple ERD for tree storage in MySQL database

Unfortunately, `canvas` is raster-based, i.e., once any shape is drawn, the fact that it was drawn is forgotten. If its position were to be changed, the entire scene would need to be redrawn. This would complicate the implementation as there are nodes with hover effects, click&drag movement and other dynamic features and complex behavior. More on implementation experience of drawing graphs and charts in `canvas` can be found in my bachelor's thesis called Graphs in Web Browser Using JavaScript[15].

Somewhat older technology then `canvas` , that is offered by nowadays browsers to draw shapes, is SVG. Unlike raster-based `canvas` , SVG is vector-based and each drawn shape is remembered as an object in Document Object Model (DOM), which is rendered to a bitmap. This means that when attributes of an SVG object are changed, the browser can automatically re-render the scene. Also, appending standardized events, e.g. *hover* and *onclick*[4], is much more easy and straightforward.

There are two more options worth mentioning when it comes to drawing in browser. One of them are Java applets. Java applet is a small application written in Java, delivered to the user in a form of byte-code. After user's confirmation, browser launches Java applet from a web page and then executes it using Java Virtual Machine (JVM). It is a process technically separated from the web browser, but it appears in a frame as integrated part of the website. The greatest disadvantage lies in lack of support. Lots of new mobile devices, including all Apple products running iOS, do not support Java applets[5].

The other option, the last I am going to mention in this section, is Adobe Flash (formerly called Macromedia Flash). It is a multimedia platform used for creating and displaying vector graphics, animations and games which can be viewed in a browser using Adobe Flash Player plugin. Also, as in the case of Java applets, the lack of support is a huge issue. Again, I would refer to my bachelor thesis[15], Chapter 4, where a detailed description of the above mentioned drawing techniques can be found.

I decided to go with SVG. It is an XML-based vector image format for two-dimensional graphics that has support for interactivity and animation. When speaking of animation, as it is quite important from human factors point of view for GrInCH to provide some animated elements and figures (e.g. expandingcollapsing node with children), there are three means by which SVG animation can be achieved:

Scripting

ECMAScript[6] is a primary means of creating animations and interactive user interfaces within SVG.

[4]More about HTML event attributes can be found at http://www.w3schools.com/tags/ref_eventattributes.asp.

[5]According to Statistical analysis and market research of Internet usage trends (StatOwl.com), Java (all versions) was supported by 66.24% devices during the past 6 months; http://www.statowl.com/java.php.

[6]ECMAScript is the scripting language standardized by Ecma International. One of its well-known dialects is already mentioned JavaScript.

Styling

Stylesheet-driven animation of SVG files from within the DOM using CSS is available in WebKit powered browsers[7] only.

SMIL

Synchronized Multimedia Integration Language (in this context also known as SVG + Time) is a W3C recommended XML markup language to describe multimedia presentations. In order to view a SMIL presentation, client needs to have a SMIL player installed on his/her computer. Only Safari, Opera, Mozilla Firefox and Google Chrome currently support this markup.

I chose scripting from the following reasons:

- Support of scripting, especially on mobiles and tablets, is by far better than other two mentioned means.
- GrInCH shall be implemented in JavaScript, therefore the compatibility and uniformity.
- I have already had plenty of experience with JavaScript from my bachelor thesis[15] and from working as a Software Design Engineer in Honeywell on user interface of a web based cloud application in ExtJS[8].

I should also mention that SVG is not supported by Internet Explorer until version 9. Old versions use VML instead. It is also an XML-based file format for two-dimensional vector graphics. However, its syntax is different.

Core

Data storage and drawing issues were discussed and resolved and it is now time to take a look at the GrInCH application itself. It shall be written primarily in JavaScript and it shall be HTML5 compliant. PHP shall handle the server operations upload and parsing, if needed, and SVG shall be used for drawing the graph.

During the development, several minor issues arose and sometimes, it was necessary to adjust pre-approved requirements or supply with additional auxiliary technologies. Most of issues were resolved by adding some extra functionalities and/or using some additional libraries.

First obvious topic to discuss was, whether to draw SVG nodes using JavaScript *manually*. From my point of view, that would be a silly solution, since there is quite a lot of available JavaScript frameworks that are supposed to simplify one's work with vector graphics on the web. Such libraries appear and disappear in todays fast evolving web environment very often and therefore I did not spend too much time choosing the one.

Two most referenced and used libraries, in my opionion, that was based on several Google search queries and couple of articles found on the Internet, were RaphaëlJS[9] by Dmitry Baranovskiy, and jQuery SVG[10] by Keith Wood. While RaphaëlJS is a standalone

[7]WebKit core powers Google Chrome and Safari; Mozilla Firefox is powered by Gecko and Internet Explorer by Trident.

[8]Sencha ExtJS is a JavaScript front-end framework; http://www.sencha.com/products/extjs.

[9]Raphaël - JavaScript Library: http://raphaeljs.com/.

[10]jQuery SVG: http://keith-wood.name/svg.html.

library, jQuery SVG, as the name itself suggests, is a jQuery[11] plugin. Both are licenced under free software MIT License and both provide some sort of wrapper for creating and editing SVG nodes within the DOM of HTML page. I simply chose RaphaëlJS because of its rich variety of examples and quite good documentation.

When implementing XML upload, I used jQuery Form Plugin. This is because standard form in HTML requires page refresh after submitting. From user's point of view, I find this quite annoying. Luckily, this can be easily worked around using AJAX. AJAX is a group of interrelated web development techniques to asynchronously both send and retrieve data from a server. Concept is as follows:

- Create a simple HTML form for file upload.
- Set the target to an iframe which is on the actual page, but not visible. This way, when the form is submitted, only the hidden iframe will be refreshed.
- Call a JavaScript function on form submit to display the loading animation.
- Have an event handler registered for the iframe's load event to parse the response and to hide the animation.
- Parse the response from iframe.

AJAX shall also handle queries to database when necessary. This feature is however not implemented, because latter testing indicated it does not affect the performance dramatically. More on that in section 5.3.

User can save the current view projection of the visualized tree in GrInCH onto harddrive. Two extensions are availble: vector *.svg* and raster graphics *.png*. Saving SVG is quite straightforward - the root SVG DOM node is serialized to string using `XMLSerializer`[12] and sent to browser with MIME media type `image/svg+xml`. Desktop browser automatically offers to download the created file.

Dealing with saving PNG was a little bit more tricky. Root SVG DOM node is serialized to string similarly, like when saving SVG. Then, element `canvas` (discussed in section 5.1) is created on the fly using JavaScript `document.createElement` funcion. Serialized SVG is then imported onto `canvas` using Javascript SVG parser and renderer[13]. After that, `canvas` method `toDataURL` finally provides stringifed version of PNG image which is sent to browser with MIME `image/png`.

Last third party software I used during the implementation was Raphael SVG Import[14]. It is not needed to run the application, but it was useful when converting SVG to RaphaëlJS source code when developing and working with the design.

5.2 Library Modules

Even though GrInCH application is in its final form distributed as a single file library[15], during the implementation it was necessary to divide it into several functional units. This

[11] jQuery is a JavaScript library designed to simplify the client-side scripting of HTML.

[12] Supported by Internet Explorer since version 9.

[13] canvg.js: Javascript SVG parser and renderer on Canvas by Gabe Lerner, MIT Licensed, http://code.google.com/p/canvg/.

[14] raphael-svg-import.js: Raphael SVG Import 0.0.4, MIT Licensed, https://github.com/wout/raphael-svg-import.

[15] With dependency to jQuery and RaphaëlJS.

helped to achieve and maintain clear and transparent working environment during the whole time.

math.js

Hyperbolic browser works with hyperbolic geometry and unit disk coordinates. Firstly, I implemented object for working with complex numbers called `Complex`. Besides getting and setting the real and imaginary value of the number, it can also return absolute and inverted value, conjugated number and square root of a complex number. Additional operations are:

`Complex.add(Complex c)`	addition of `c`
`Complex.conjugate()`	complex conjugate
`Complex.divide(Complex c)`	division by `c`
`Complex.distance(Complex c)`	distance to `c`
`Complex.multiply(Complex c)`	multiplication by `c`

Vectors of complex numbers were needed as well, object `Vector` only stores its complex base and direction. `Projection` presents object for inverting, multiplying and projecting projections:

`Projection.invert()`	inversion
`Projection.multiply(Projection p)`	multiplication by `p`
`Projection.project(Vector v)`	projection to `v`

JavaScript `Math` object only defines basic mathematical functions, that can be found in Appendix B). More complicated mathematical functions were implemented as methods of `HyperMath` object using the pre-existing built-in functions:

`HyperMath.sinh(x)`	hyperbolic sine of `x`
`HyperMath.cosh(x)`	hyperbolic cosine of `x`
`HyperMath.asinh(x)`	hyperbolic arc sine of `x`
`HyperMath.acosh(x)`	hyperbolic arc cosine of `x`

hyperbolic.js

Poincaré model of hyperbolic geometry is quite complicated. Luckily, it was greatly explained by Dr. David C. Royster, one of the teachers at University of Kentucky, during spring lectures in 2004[9]. After going through his *Chapter 9 – Poincaré Models of the Hyperbolic Plane*, I was able to create `PoincareModel` object with the following methods:

`PoincareModel.rotate(Complex point, angle)`	rotation about `point` by `angle`
`PoincareModel.translate(Complex c)`	translation by `c`
`PoincareModel.distance(Complex c)`	distance to `c`

Algorithm 1 Layout onto Hyperbolic Plane

```
1:  function LAYOUT($parent$)                         ▷ post-oder DFS
2:      $children$ ← children of parent
3:      for all $children$ do
4:          LAYOUT($child$)                           ▷ recursive call
5:          if $child$ is last sibling then
6:              SCALE($child$)                        ▷ scaling
7:          end if
8:      end for
9:      COMPUTE($parent$)                             ▷ computing
10: end function
```

Algorithm 2 Projection Onto Unit Disk

```
1: function PROJECT($parent$)                         ▷ recursive function
2:     for all $children$ do
3:         COMPUTE($parent$)                          ▷ computing
4:         PROJECT($child$)                           ▷ recursive call
5:     end for
6: end function
```

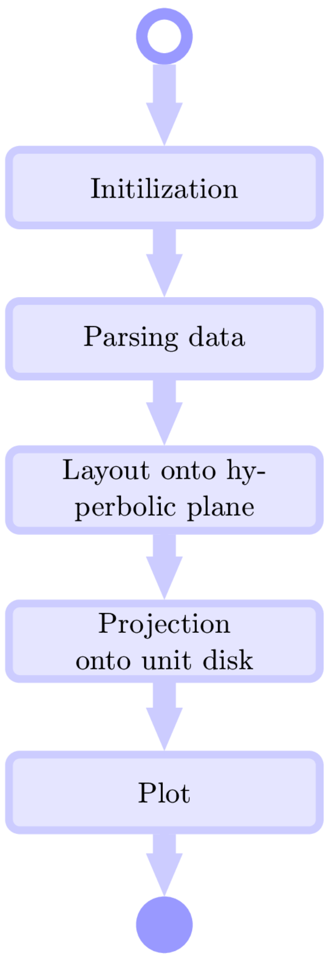


Figure 5.3: Flow of GrInCH Application

model.js

For more readable and self-explanatory code I designed a model, that consists of `Node` and `NodeList` objects. This is internal representation of the tree inside GrInCH. Probably the most useful is `NodeList`'s method `eachDFSpostCustom`, that accepts parameters function and callbackFunction and is able to recursively walk through the whole tree acting as post-order depth-first search algorithm. It also applies callback function for every node that is last sibling at its level, as can be seen in Figure 5.3 represented by algorithm 1.

grinch.js

Last but not least comes the core, the brain of the browser. It controls flow of the whole application. During initialization, GrInCH creates two SVG canvases. One for the top bar, containing heading, logo and controls, because it is only redrawn when new file is uploaded or when window resized; the second one for displaying the tree is redrawn every time user interacts with the graph.

The core also adjusts the size of the canvas for the screen dimensions, sets, parses and stores the data into internal representation, computes the layout of the nodes on the hyperbolic plane as well as projects them onto unit disk. It also takes care of drawing with help of integrated library RaphaëlJS.

In this part of the program, all actions and events are handeled. Library recognizes, whether it is being run on a desktop computer, or on some sort of touch device, and adjusts the size and behavior of the graph accordingly.

5.3 Results

Application is functional and finished and it was minified into single file *grinch-min.js*. Following section describes installation and usage. It also covers pros and cons of the solution(s) I chose, performance and compatibility issues. It includes several screenshots from different devices for better imagination of the reader.

Installation

Installation was already briefly mentioned in Chapter 5. GrInCH application is a single file library and it takes up the whole body of a web page. The process of getting it to work is so easy:

1. Create minimal valid HTML5 file[16] (see Listing C.1 in Appendix C).
2. Include jQuery, RaphaëlJS and GrInCH libraries (see Listing C.2 in Appendix C).
3. Optionally, add a hidden form into the page `body` tag to allow file uploads.
4. Optionally, add library for conversion from SVG to canvas to allow saving graph as PNG.

[16] The minimal valid HTML5 structure according to W3C Makup Validation Service

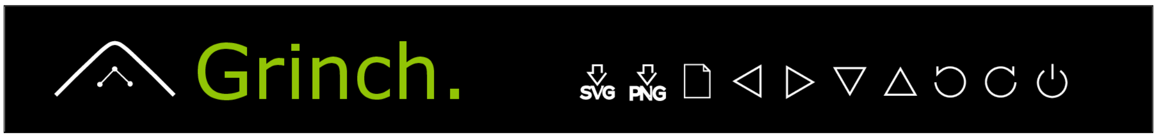


Figure 5.4: GrInCH Bar

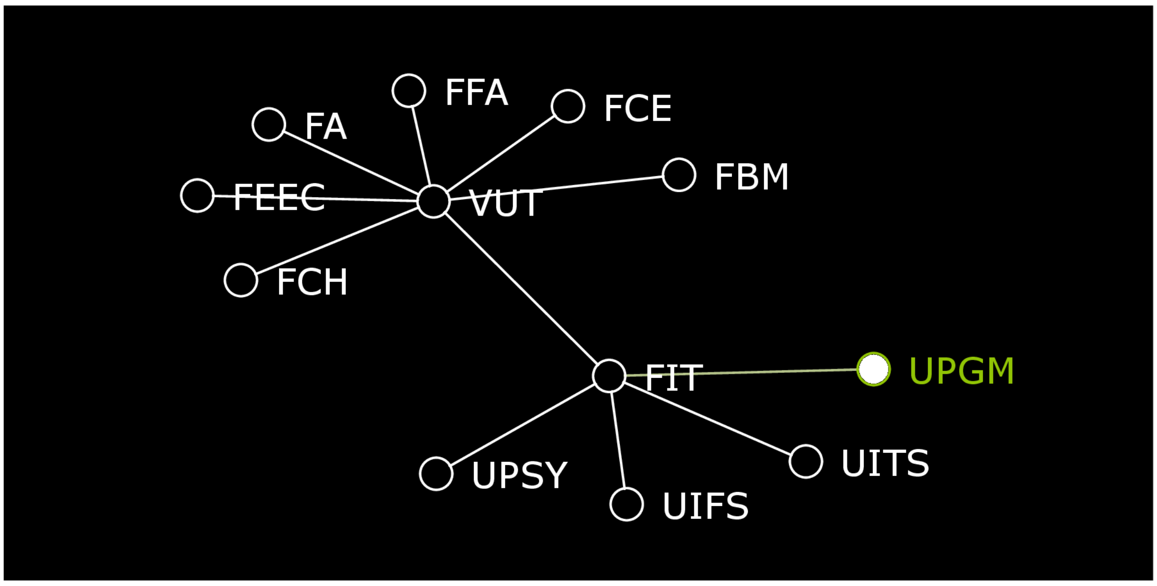


Figure 5.5: GrInCH Canvas

Usage

Using GrInCH shall be quite easy, natural and user friendly. The top bar, see figure 5.4, takes 200 px off of screen height and contains basic control elements. Clicking on SVG (or PNG) icon downloads the current tree view as SVG (or PNG[17]). The four arrows move the graph view in the four directions, rotating arrows rotate it clockwise and counter-clockwise and finally reset button resets the view into initial projection by performing a smooth animation.

The rest of the screen is reserved for displaying the tree. Design is simple and minimalistic, see figure 5.5. Black circles with a white border are either leaf nodes, or expanded regular nodes. Completely white circles are collapsed nodes. By clicking on a collapsed node, additional data are downloaded (if needed) and node is expanded (if contains any children). Clicking on the same node again causes him to collapse (except the root node, that can not be collapsed). By clicking & dragging, either on one of the displayed nodes or on the black background, user can move the tree into desired location. Rotation using multi-touch gesture is supported only on touch device.

Performance

GrInCH was developed on Google Chrome 26 and tested under Safari 6. It also runs under Mozilla Firefox 20 and Internet Explorer 9. It is optimized for touch devices and was tested

[17]When downloading PNG, *canvg.js* or any other library for importing SVG onto HTML `canvas` must be present.

in Google Chrome and Safari under iOS and Google Chrome under Android.

The best performance is experienced under Chrome and Safari. They are both powered by WebKit rendering core. Most buggy and unstable is under Internet Explorer, that uses core Trident. It runs fine in Firefox using core Gecko, but the experience is may be slower and/or disjointed. GrInCH is extremely demanding in several areas:

- JavaScript computation of hyperbolic plane coordinates.
- Adding lots of new nodes into DOM.
- Rendering quite complicated SVG.

There are several benchmark suits to measure the performance of the JavaScript engine in a browser. I used *Dromaeo*[18]., because it aggregates variety of tests from other developer suites, i.e. *SunSpider* or *V8*, and thus offers very complex testing with detailed information. The testing was aimed specifically to tasks that GrInCH performs the most:

DOM Core Tests

DOM Attributes Setting and getting DOM node attributes. Tests: dom, attributes.

DOM Modification Creating and injecting DOM nodes into a document. Tests: dom, modify.

DOM Query Querying DOM elements in a document. Tests: dom, query.

DOM Traversal Traversing a DOM structure. Tests: dom, traverse.

Dromaeo/SunSpider/V8 JavaScript Tests

Rotating 3D Cube Rotating the individual pixels of a cube. No rendering done. Tests: object, array, property, math.

Trigonometric Calculation Calculate values from hyperbolic and trigonometric functions. Tests: math, looping.

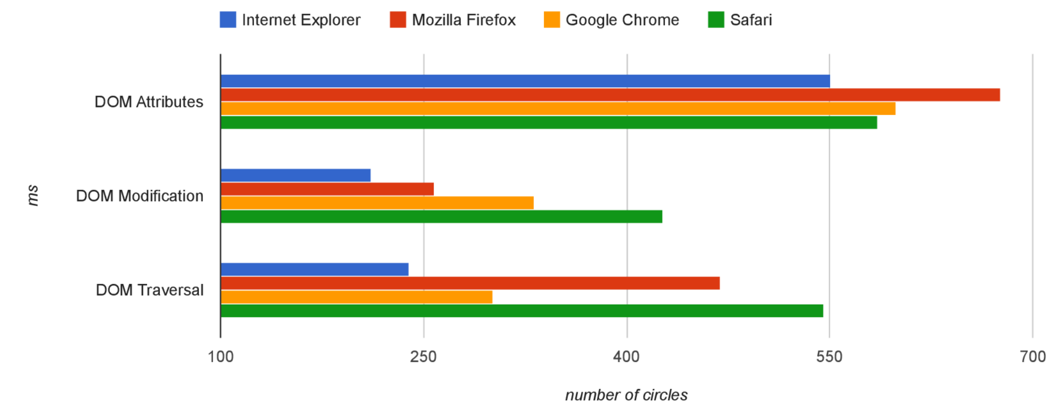
Richards Benchmarks A series of benchmarks to test the quality of system implementation languages. Tests: functions, object.

According to these carefully selected tests, visualized in charts 5.6 are Google Chrome and Safari the fastest. Mozilla Firefox takes second place and the worst optimization of the above listed JavaScript operations has Internet Explorer. This generally means, that WekBit core is best optimized for GrInCH, Gecko is not too bad and Trident is the worst to run.

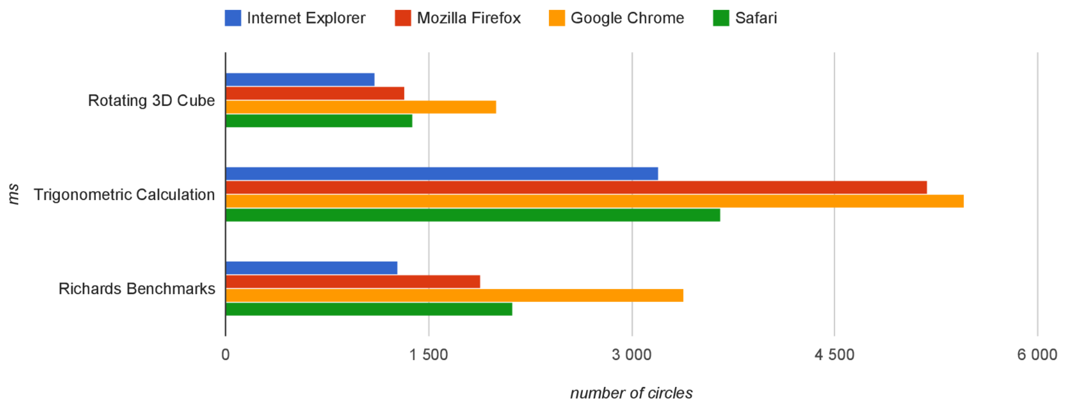
Besides DOM core tests and JavaScript optimization, it would me more than appropriate to find out, what is the optimization of SVG rendering in different browsers. I let SVG draw from 50 to 5000 circles on canvas size 2048 × 1536 px. In Figure 5.7 there are also results of HTML `canvas` to have something to compare the results with.

To sum up, every SVG element is appended to the Document Object Model (DOM) and can be manipulated using a combination of JavaScript and CSS. Moreover, one can attach an event handlers to a SVG element or update its properties based on another document event. `Canvas` , on the other hand, is a simple graphics API. It draws pixels and nothing

[18]Dromaeo: JavaScript Performance Testing Suite by John Resig; http://dromaeo.com/



(a) DOM Core Tests



(b) JavaScript Tests

Figure 5.6: DOM and JavaScript Benchmark Results

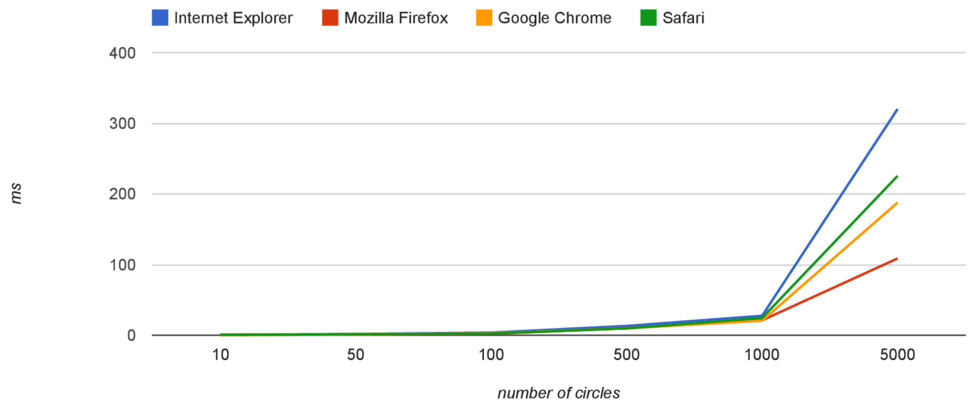
more. There's no way to alter existing drawings or react to events. If an update of canvas image is needed, the scene needs to be redrawn.

As a result, even though SVG showed to be overall slower than `canvas` mostly because of working with DOM, it was still better choice for hyperbolic browser. One can assume that the overhead of redrawing the scene manually would take much longer to both implement and then perform, than leaving it to a little slower SVG. Furthermore, it is not unexpected that SVG optimization in browsers will get better as more developers start using it.

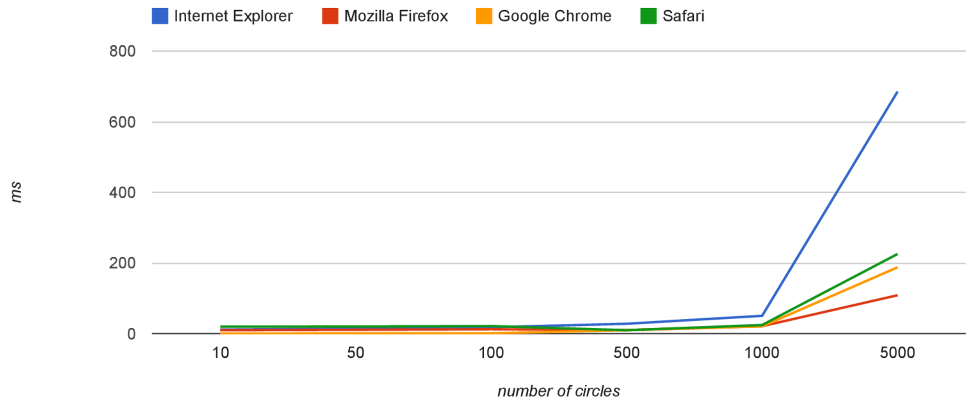
Use in Practice

Parts of the code, measured results and the idea of hyperbolic browser found use in Honeywell Technology Solutions in Brno. Condition Based Maintenance team is seriously considering usage of such concept within their own applications; negotiations with customers are taking place.

Also, article about GrInCH was accepted into the collection of *Electrical Engineering, Information and Communication Technologies* (EEICT) conference in 2013 held by Faculty of Information Technology and Faculty of Electrical Engineering and Communication



(a) SVG



(b) `canvas`

Figure 5.7: SVG vs. HTML `canvas`

at Brno University Technology. It placed among the nine best projects in Graphics and Multimedia category.

Chapter 6

Conclusion and Future Work

The aim of this thesis was to get acquainted with methods of visualization of tree structures and existing software, and to propose and implement hyperbolic browser suitable for web environment. This work is based on the article *Hyperbolic browser: A focus + context technique for visualizing large hierarchies* by J. Lamping and R. Rao published in Journal of Visual Languages and Computing in 1996.

Analysis of requirements for efficient visualization of large tree structures in hyperbolic plane and evaluation of features considered for the task resulted in a suitable mathematical model for computing coordinates of nodes in hyperbolic plane, distributed as a single-file script called GrInCH (Graphs in Canvas Hyperbolic). It is a pure JavaScript library that transforms any modern web browser into hyperbolic tree visualizer using SVG technology. The browsing experience is truly modern, uncommon and quite eye-catching. GrInCH was also optimized for touch devices and it is easily usable on any mobile device, assuming it runs browser with HTML5 support.

The key feature of presented hyperbolic browser is the fact, that it offers seamless navigation while browsing large tree hierarchies, thus avoids loosing orientation when exploring trees with enormous width and depth.

GrInCH experience is the best when run in browsers powered by WebKit rendering core, accelerated directly by GPU. Optimization for slower devices, touch devices and Internet Explorer were the most tricky parts. Understanding hyperbolic geometry rules and laws was also quite a challenge.

Testing on real data confirmed that concept of hyperbolic browser is hiding great potential and numerous opportunities of how to enhance its functionality and improve user experience. GrInCH could be used for educational purposes, for example to search for one specific node (representing e.g. some biological species) within very deep tree structure (e.g. evolutionary tree of life) and then automatically navigate to that node providing clear instructive animation. Honeywell is considering using hyperbolic browser to navigate among parts of airplane engine when it needs to be dismantled.

Literature

[1] P. E. Black and 1999 Algorithms & Theory of Computation Handbook. “tree” — dictionary of algorithms and data structures, crc press llc. http://www.nist.gov/dads/HTML/tree.html, 14 August 2008. [Online; accessed 5-Jan-2013].

[2] Oracle Corporation. Mysql :: About mysql. http://www.mysql.com/about/, 2013. [Online; accessed 8-May-2013].

[3] M. Friendly. Milestones in the history of data visualization: A case study in statistical historiography. In C. Weihs and W. Gaul, editors, *Classification: The Ubiquitous Challenge*. Springer, New York, 2005.

[4] M. Friendly and E. Kwan. Effect ordering for data displays. In *Computational Statistics & Data Analysis*. Elsevier B.V., 2002.

[5] M. Haverbeke. *Eloquent JavaScript: A Modern Introduction to Programming*. No Starch Press, 2011.

[6] D. Knuth. *The art of computer programming*. Addison-Wesley, Upper Saddle River, NJ, 2005.

[7] J. Lamping and R. Rao. Hyperbolic browser: A focus + context technique for visualizing large hierarchies. *Journal of Visual Languages and Computing*, (7):33–35, 1996.

[8] Computer History Museum. Timeline of computer history. http://www.computerhistory.org/timeline/, 2006. [Online; accessed 5-Jan-2013].

[9] D. C. Royster. Class worksheets and lecture notes. http://www.ms.uky.edu/~droyster/courses/spring04/, 2013. [Online; accessed 12-May-2013].

[10] E. W. Weisstein. Hyperbolic geometry — mathworld–a wolfram web resource, 2012. [Online; accessed 30-December-2012].

[11] Wikipedia. Data structure — Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Data_structure, 2012. [Online; accessed 30-December-2012].

[12] Wikipedia. Graph theory — Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Graph_theory, 2012. [Online; accessed 30-December-2012].

[13] Wikipedia. Hyperbolic geometry — Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Hyperbolic_geometry, 2012. [Online; accessed 30-December-2012].

[14] Wikipedia. Unit disk — Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Unit_disk, 2013. [Online; accessed 12-May-2013].

[15] J. Zajic. *Graphs in Web Browser Using JavaScript.* Bachelor Thesis, BUT FIT, Brno, Czech Republic, 2011.

Nomenclature

$\Im$	Imaginary part of a number
GrInCH	**Gr**aphs **In** **C**anvas **H**yperbolic (name of the implemented application)
AJAX	Asynchronous JavaScript and XML
CSS	Cascading Style Sheet
DFS	Depth-First Search
DOM	Document Object Model
EEICT	Electrical Engineering, Information and Communication Technologies
ERD	Entity Relationship Diagram
GPU	Graphics Processing Unit
HDD	Hard disk drive
HTML5	HyperText Markup Language
iOS	Apple Operating System for Mobile Devices
Jit	JavaScript InfoVis Toolkit
JNLP	Java Network Launching Protocol
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
MIME	Multipurpose Internet Mail Extensions
MVS	Markup Validation Service
MySQL	My Structured Query Language
PHP	PHP: Hypertext Preprocessor
PNG	Portable Network Graphics
SMIL	Synchronized Multimedia Integration Language
SVG	Scalable Vector Graphics

UML	Unified Modelling Language
VML	Vector Markup Language
W3C	World Wide Web Consortium
WWW	World Wide Web
XML	Extensible Markup Language

Appendix A

Description of Related Art

The hyperbolic browser idea discussed and studied in this project is based on a paper entitled "A Focus+Context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies" by Lamping, J. and Rao, R., published in April 3rd, 1995 in Journal of Visual Languages and Computing. In the Acknowledgments section of that paper, the authors claim that Xerox Corporation is seeking patent protection for the described technology.

There are several patents issued by U.S. Patent and Trademark Office, related to the hyperbolic browser described in Rao's and Lamping's paper. After seeking "Tree visualization system and method based upon a compressed half-plane model of hyperbolic geometry" under patent No. 6901555 at United States Patent and Trademark Office Website[1], the following can be read:

> A focus+context technique for visualizing large hierarchies is described in U.S. Pat. No. 5,590,250, entitled "Layout of Node-link Structures in Space with Negative Curvature," in U.S. Pat. No. 6,108,698, entitled "Node-Link Data Defining a Graph and a Tree Within the Graph," and in U.S. Pat. No. 5,619,632, entitled "Displaying Node-link Structure with Region of Greater Spacings and Peripheral Branches." Related prior art patent applications include: Local Relative Layout of Node-Link Structures in Space with Negative Curvature. John Lamping, Ramana Rao, Tichomir Tenev. EP Publication No. 0977155, 2 Feb. 2000. Mapping a Node-Link Structure to a Rendering Space Beginning from and Node. Ramana Rao, John Lamping, Tichomir Tenev. EP Publication No. 0977153, 2 Feb. 2000. Controlling Which Part of Data Defining a Node-Link Structure is in Memory. Tichomir Tenev, John Lamping, Ramana Rao. EP Publication No. 0977131, 2 Feb. 2000.

This semestral project, including the implementation of the hyperbolic browser, is intended for educational purposes only.

[1]Searching for patent numbers can be accessed at http://patft.uspto.gov/netahtml/PTO/srchnum.htm

Appendix B

JavaScript Mathematical Functions

`Math.abs(x)`	the absolute value of `x`
`Math.acos(x)`	arc cosine of `x`
`Math.asin(x)`	arc sine of `x`
`Math.atan(x)`	arc tangent of `x`
`Math.atan2(x, y)`	arc tangent of `x` `y`
`Math.ceil(x)`	integer closest to `x` and not less than `x`
`Math.cos(x)`	cosine of `x`
`Math.exp(x)`	exponent of `x` (`Math.E` to the power `x`)
`Math.floor(x)`	integer closest to `x` , not greater than `x`
`Math.log(x)`	log of `x` base e
`Math.max(x, y)`	the maximum of `x` and `y`
`Math.min(x, y)`	the minimum of `x` and `y`
`Math.pow(x, y)`	`x` to the power `y`
`Math.random()`	pseudorandom number from 0 to 1
`Math.round(x)`	integer closest to `x`
`Math.sin(x)`	sine of `x`
`Math.sqrt(x)`	square root of `x`
`Math.tan(x)`	tangent of `x`

Appendix C

GrInCH Installation

```html
<!doctype html>
<html>
        <head>
        <meta charset=utf-8>
        <title>GrInCH</title>
        </head>
        <body>
                <!-- content -->
        </body>
</html>
```

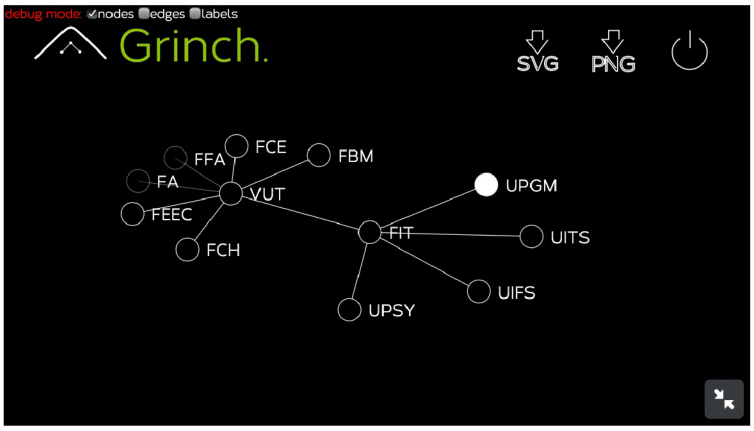
Listing C.1: Minimal valid HTML5 file

```html
<!doctype html>
<html>
<head>
    <meta charset=utf-8>
    <script type="text/javascript"
        src="http://code.jquery.com/jquery-latest.min.js">
    </script>
    <script type="text/javascript"
        src="http://github.com/DmitryBaranovskiy/raphael-min.js">
    </script>
    <script type="text/javascript"
        src="js/grinch.js">
    </script>
<title>GrInCH</title>
</head>
<body>
    <script type="text/javascript">
    <!--
        $(document).ready(function() {
            var grinch = new Grinch();
            var rawData = '...', // Tree Structure Stored in JSON
                data = JSON.parse(rawData);
            grinch.setData(data);
        });
    //-->
    </script>
</body>
</html>
```

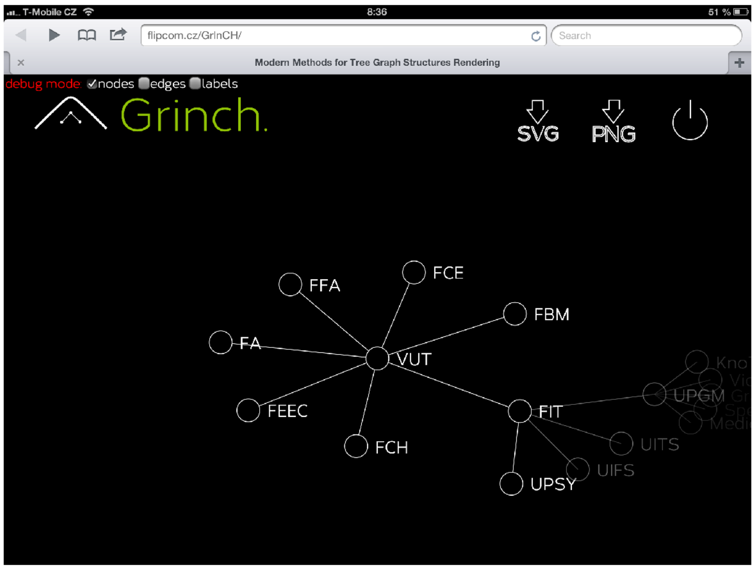
Listing C.2: GrInCH Minimal Installation

Appendix D

Screenshots



(a) Screenshot of GrInCH on iPhone 5



(b) Screenshot of GrInCH on iPad

Figure D.1: Screenshots

Appendix E

DVD Contents

Attached DVD contains all the source code, including benchmark results from testing, in the following directory structure:

/benchmarks.xlsx
Results of benchmark testing

/install.txt
The installation manual

/thesis.pdf
Master's thesis

/video1.mov
Demonstration video #1

/video2.mov
Demonstration video #2

/tex/*
Source codes of the thesis

/www/
GrInCH application

- **index.html**
  HTML5 web page with included GrInCH
- **js/***
  JavaScript libraries
- **source/***
  Testing XML files