WEB-BASED DRAWING SOFTWARE FOR GRAPHS IN 3D AND TWO LAYOUT ALGORITHMS

FARSHAD BARAHIMI
Bachelor of Science, Shahid Bahonar University of Kerman, 2010

A Thesis
Submitted to the School of Graduate Studies
of the University of Lethbridge
in Partial Fulfillment of the
Requirements for the Degree

MASTER OF SCIENCE

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

WEB-BASED DRAWING SOFTWARE FOR GRAPHS IN 3D AND TWO LAYOUT ALGORITHMS

FARSHAD BARAHIMI

Date of Defense: April 20, 2015

Dr. Stephen Wismath Supervisor	Professor	Ph.D.
Dr. Robert Benkoczi Thesis Examination Committee Member	Assistant Professor	Ph.D.
Dr. Joy Morris Thesis Examination Committee Member	Associate Professor	Ph.D.
Dr. Howard Cheng Chair, Thesis Examination Committee	Associate Professor	Ph.D.

Dedication

Dedicated to my family whose value for me can not be expressed in words.

Abstract

A new web-based software system for visualization and manipulation of graphs in 3D, named We3Graph is presented with a focus on accessibility, customizability for applications of graph drawing, usability and extendibility. The software system allows multiple users to work on the same graph at the same time and is accessible through web browsers. The software can be extended using plugins written in any programming language and custom render engines written in the Javascript language. Also two new algorithms are proposed to answer the following question, previously raised in [53]:

> Given a graph G with n vertices, $V = \{v_1, v_2, \ldots, v_n\}$, and given a set of n distinct points $P = \{p_1, p_2, \ldots, p_n\}$ each with integer coordinates in three dimensions, can G be drawn crossing-free on P with v_i at p_i and with a number of bends polynomial in n and in a volume polynomial in n and the dimension of P?

Acknowledgments

I want to express my gratitude to Dr. Stephen Wismath for being a kind, experienced and professional supervisor whose support helped me a lot and I learned a lot from him. Also I want to express my special thanks to Dr. Howard Cheng, my coach in ACM ICPC programming contests. Also I want to express my gratitude to my great committee members, Dr. Robert Benkoczi and Dr. Joy Morris. I'm also grateful of my great friends for helping me during my master's studies and life.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

Graph visualization has many applications in software engineering, chemistry, bioinformatics, social sciences, information visualization, VLSI circuit design, etc. The academic efforts on graph drawing are usually separated into two categories. The first category is to determine algorithms to lay out graphs with respect to some criteria such as area, volume, number of crossings, symmetry, etc. The second category is to develop software systems that allow visualization and manipulation of graphs or implementation of layout algorithms.

Drawings of graphs are typically in two dimensions (2D) or three dimensions (3D). While 2D graph drawing has been studied more, 3D graph drawing also looks promising as 3D technology is advancing and some applications of graph drawing have a 3D nature (such as protein structures).

A new web-based 3D graph visualization software package named We3Graph is developed as part of this dissertation with a focus on accessibility, customizability for applications of graph drawing, usability and extendibility. Also two new algorithms are proposed to answer a previously raised question in the 3D graph drawing literature.

1.2 Structure of this thesis

The rest of this chapter presents a high level system description and motivations of We3Graph as well as some background information. Chapter 2 reviews previous efforts on theoretical aspects of 3D graph drawing and also presents the new proposed algorithms. Chapter 3 reviews previous efforts on graph drawing software. Chapter 4 talks about the software architecture and implementation of We3Graph. Chapter 5 concludes and discusses

possible paths for future work.

1.3 Applications of graph drawing

First we give a brief overview of the applications of graph drawing. Graphs as a general model are used in many disciplines. Graph drawing adds geometrical properties to the graphs making it a more structured model. The 3D structure of chemistry molecules can be modelled as graphs in 3D space, in particular the 3D structure of some important biological molecules such as proteins can be modelled as graphs in 3D. Graph drawing is also used to visualize some important biological networks such as protein-protein interaction networks. Different types of diagrams can be modelled as graphs by adding shape properties to vertices and adding bends to edges. The UML diagrams used in software engineering are a good example. Using graphs to visualize computer networks is another application of graph drawing. Social sciences use graphs to model social networks. Graph drawing is also used in VLSI circuit design. Different types of maps such as routes between cities can be modelled as graphs. The abstract model of graphs has also different types such as directed or undirected graphs and graphs with multiple edges between two vertices. In some applications 2D drawings are used in practice but researchers are also studying 3D alternatives to 2D drawings [32, 51, 48, 63].

1.4 Web-based visualization approach

With the advent of WebGL in 2011, it is possible to build a web-based visualization tool for graphs in 3D. To be more precise, the graphs can be visualized and manipulated in a web browser and the World Wide Web can be used as a medium to communicate between different users and different pieces of the software.

In this dissertation we refer to traditional software systems as software systems that do not use web browsers for their interface and instead rely on a native user interface provided by the operating system. Some of the traditional software systems have a more offline na-

ture and depend solely on the computing and storage power of their host machine, while others have a more online nature and use mediums such as the World Wide Web communication protocols to access information from other resources and use computing and storage power of servers instead of the host machine.

Web-based software systems rely solely on the web browser for their user interface which increases the accessibility of the software. These software systems do not have any installation process and are also accessible through web browsers which are available on almost all platforms and devices such as computers, tablets, cell phones, windows, linux, mac, android, iOS, etc. We3Graph uses web browsers for its user interface to benefit from the above advantages but it is not designed for cell phones due to the restriction of their small display size. We3Graph also uses the World Wide Web communication protocols to communicate with the server. Graphs can be stored on the server, accessed from any location and shared among different users. Given the communication power of the World Wide Web, We3Graph provides a collaborative environment where multiple users can work on the same graph at the same time.

Although web-based software packages have the above benefits they suffer from lower performance and a less powerful user interface compared to traditional software, but the gap is closing as web technology is advancing with solutions like faster Javascript engines, HTML 5 and rich user interface libraries such as jQuery UI. Although in the first generation of web-based software systems, user interaction was very basic and static and it was not comparable to traditional software systems, in the new generation of web-based software systems, the user interaction is more mature and dynamic and is close but not equal to traditional software systems.

In the rest of this section, we briefly describe a few tools which can help the development of web-based visualization software, and touch on how We3Graph makes use of them.

1.4.1 WebGL standard

WebGL is a hardware accelerated 3D graphics standard for web browsers based on OpenGL ES. WebGL uses an HTML5 canvas element for rendering, and Javascript as the programming interface. Though the first ideas of WebGL came in 2006 by Vladimir Vukićević, Khronos Group released the first official version of WebGL in 2011. WebGL is currently supported in the latest desktop and mobile versions of Google Chrome, Mozilla Firefox, Microsoft Internet Explorer and Apple Safari. We3Graph uses WebGL indirectly through Three.js library described bellow.

1.4.2 HTML 5 standard(s)

HTML 5 is the newest version of HTML (Hypertext Markup Language) and sometimes is used as an umbrella term referring to the set of standards for web browsers that are suitable for the next generation of powerful web applications, even though those standards have separate official specifications. HTML 5 supports multimedia with the new audio and video elements, 2D graphics with the new canvas element and elements for scalable vector graphics (SVG) and 3D graphics with WebGL. It also provides new semantic tags such as header, article, section, footer and new types for form inputs such as email and URL. Other notable standards are Web Storage, Indexed Database API, File API, Web Workers, WebSockets and GeoLocation. HTML 5 combined with Javascript and server side code can be an alternative solution to traditional software. We3Graph use the WebGL standard as well as canvas element for textures used in custom render engines of WeGraph.

1.4.3 Three.js Javascript 3D graphics API

While WebGL provides a low level API for 3D web graphics, Three.js is a Javascript library that provides a higher level API around the WebGL API to make writing code for 3D web graphics easier. We3Graph uses Three.js instead of programming WebGL directly.

1.5 Extendibility with web services

Web Services allow exchange of data between a server application and a client application with a behaviour similar to the client application calling a function on the server application. We3Graph uses a web service API as core communication between different components of the system. Using web services allows plugins to be written in any programming language and run on any computer. Chapter 4 discusses the software architecture and implementation issues of We3Graph in more detail.

1.6 User interaction

This section provides an overview of the We3Graph user interaction and typical workflow of the program. In addition to typical mouse, keyboard and 2D display interaction methods, We3Graph provides other interaction methods such as stereoscopic 3D using a side by side method, Leap Motion hand tracking device and the Oculus Rift virtual reality headset. Also the We3Graph user interface is tuned for touch screens.

1.6.1 Stereoscopic 3D and the Oculus Rift

In stereoscopic 3D, two different images are sent to each eye which will help the brain in depth perception. There are different ways to produce 3D stereoscopic effect. Among them we can mention anaglyph method, alternate frame sequencing method, polarization method and side by side 3D method. In the side by side 3D method, the rendered frame is divided into two sections, one for the left eye, and one for the right eye. The displaying device such as a 3D projector or a 3D TV combined with the proper 3D glasses send appropriate images to each eye. Previous research [64, 65] has shown that stereoscopic and motion cues improve the performance of the user for tracing paths in graphs. The Oclulus Rift virtual reality headset provides both stereoscopic and motion cues. The head mounted display of the Oculus Rift, combines stereoscopic 3D with head tracking, providing an immersive user experience. While low resolution is a drawback of the Oculus Rift headset, the newer

version (DK2) has a higher resolution than the older version (DK1). We3Graph provides support for the OculusRift to allow the user benefit from the advantages stereoscopic display coupled with motion cues of head tracking and also provides a different user experience with immersive virtual reality.

1.6.2 Leap Motion

Leap Motion is a small usb device that tracks the hands and fingers. It can provide information such as finger tip position and direction, or palm normal vector. The reported model of hands is not always correct and the reliability of the reported model varies in different hand gestures. We3Graph uses Leap Motion in combination with a keyboard to provide a more reliable user interface. We3Graph allows the user to use the palm normal vector to manipulate the camera and the index finger to point to the monitor by emulating mouse movement. While the natural behaviour of pointing with the index finger improves the user experience compared to the mouse, the lack of accuracy downgrades the user effectiveness compared to the mouse. Also the fatigue caused by holding the hand high, makes the Leap Motion more appropriate for short interactions than long interactions. The spatial mapping between the camera orientation and palm normal improves the user interaction for navigating the graph.

1.6.3 Typical workflow

To use the web based interface, the user logs in with a username and password as depicted in figure 1.1. After logging in, the start panel of the software is shown which provides an interface for different features of the system as depicted in figures 1.2 and 1.3. The graphs are organized in folders. To create a graph the user should specify a folder, a graph name and a render engine to be used for that graph. Different render engines are suitable for different applications of graph drawing and different graph types. To start viewing or manipulating a graph, the user can select a folder and then select a graph in that folder and click the start button. After clicking the start button the application tries to load the graph

and shows the main interface to view and edit the graph as depicted in figures 1.4 and 1.5. The user can manipulate the graph and camera using the buttons on the special rectangular toolbar of the software, mouse, keyboard or touch (in touch screens). With the help of plugins the user is also able to use the Leap Motion and/or Oculus Rift to interact with the software. Clicking the button with the pin icon on the top right of the toolbar will bring the accordion menu as depicted in the right side of figure 1.5. The accordion menu helps manipulate the current selection and also helps change custom properties used in different applications of graph drawing.

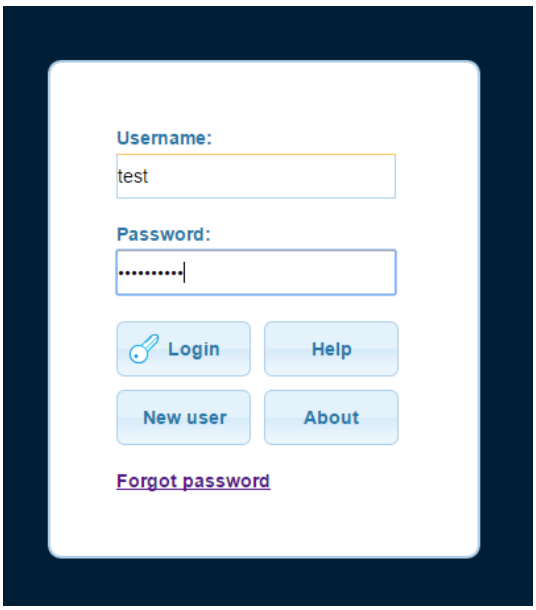

Figure 1.1: Login interface of We3Graph

1.6.4 Server installation and administration

The software should be installed once on the server by running a prepared sql script, copying some files and adjusting two configuration files.

Administration such as managing folders, groups, permissions and memberships is done using the administration panel of the software that is accessible to the admin user as shown in figure 1.6.

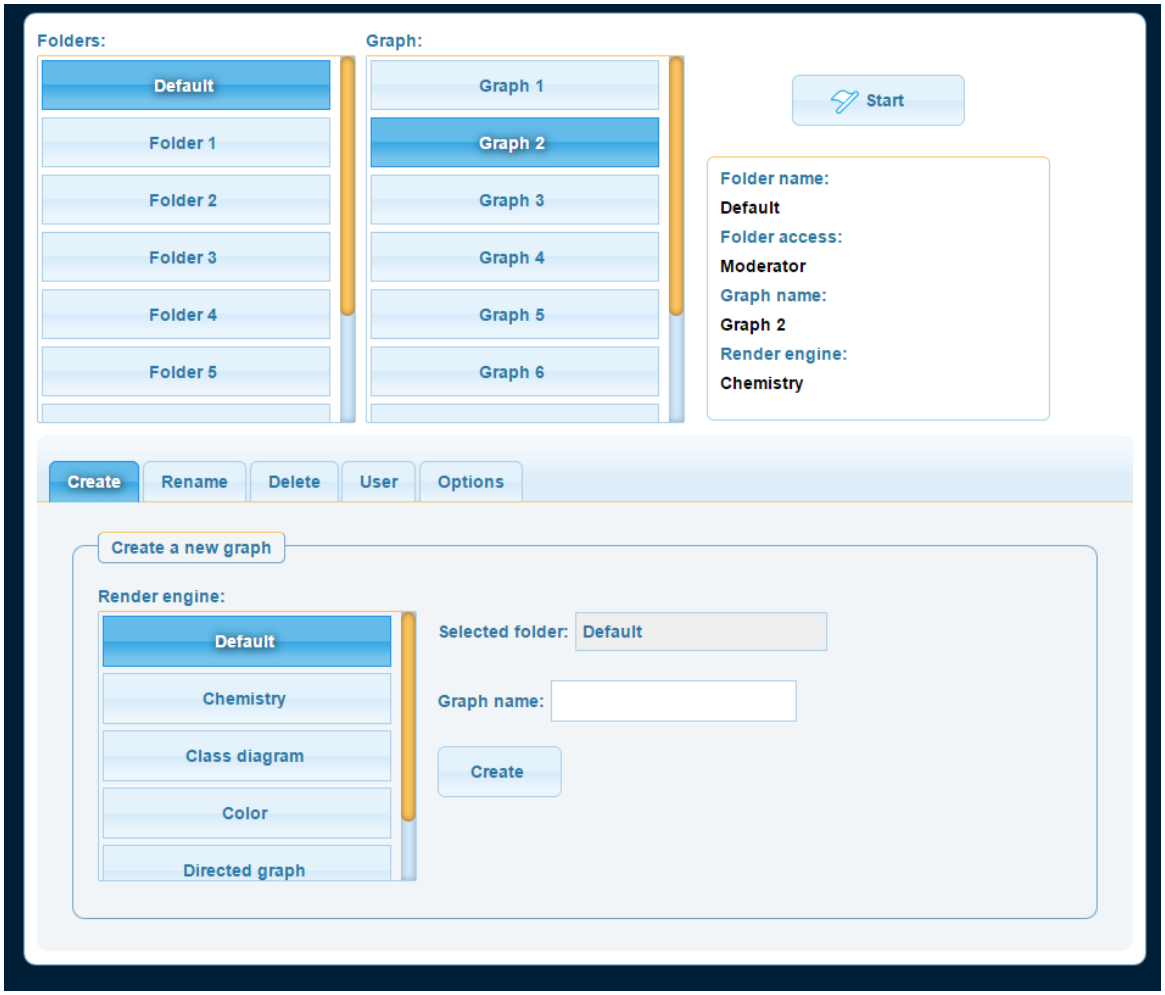

Figure 1.2: Start panel interface of We3Graph. Create tab on the bottom.

1.7 Application-specific customizability

We3Graph also pays particular attention to applications of graph drawing, trying to make it easier to customize We3Graph for a specific application. Different render engines, for different applications of graph drawing or different types of graphs, can be built upon the default render engine of We3Graph. Custom render engines combined with custom properties that can be attached to the vertices or edges, help customize We3Graph for different applications of graph drawing. We3Graph has sample render engines such as the ones for chemistry molecules or 3D class diagrams, while other engines can be added as Javascript extensions. Figure 1.7 shows the 3D structure of a methyl hydrogen sulfate molecule in We3Graph. Figure 1.8 shows a 3D class diagram in We3Graph.

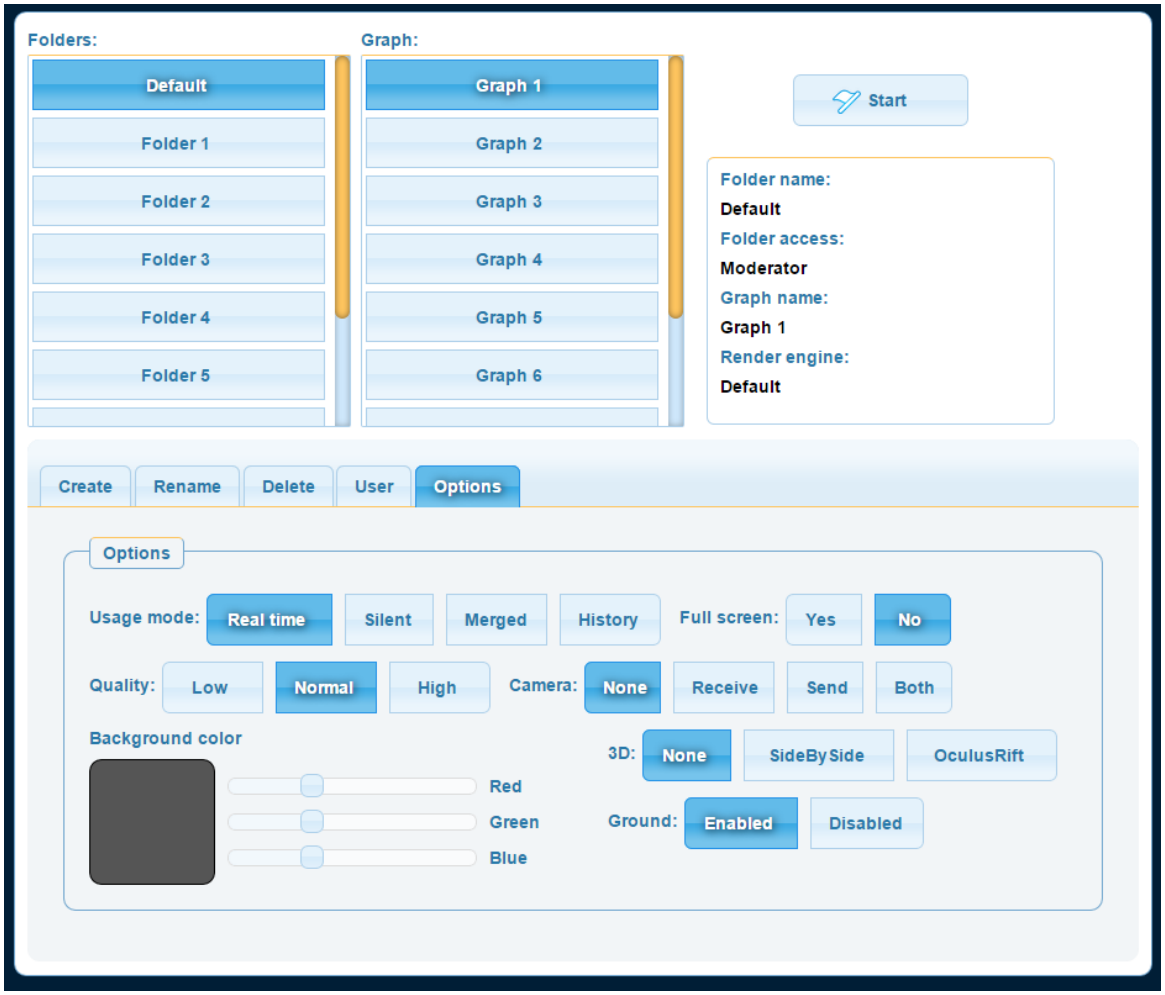

Figure 1.3: Start panel interface of We3Graph. Options tab on the bottom.

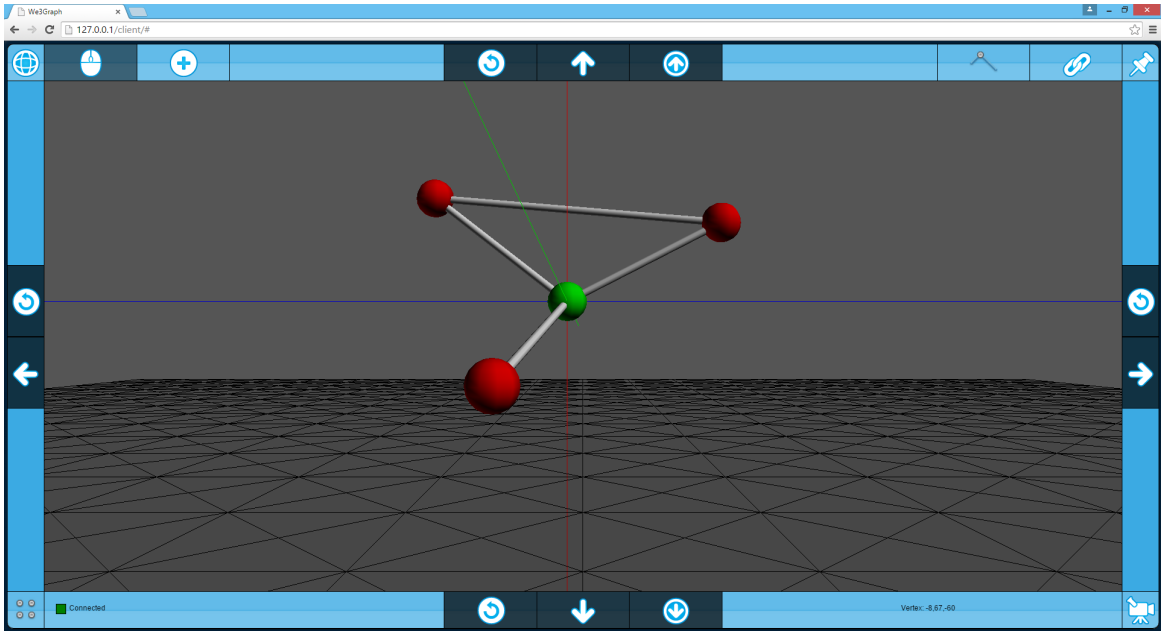

Figure 1.4: Graph viewing and editing interface of We3Graph in a Chrome browser.

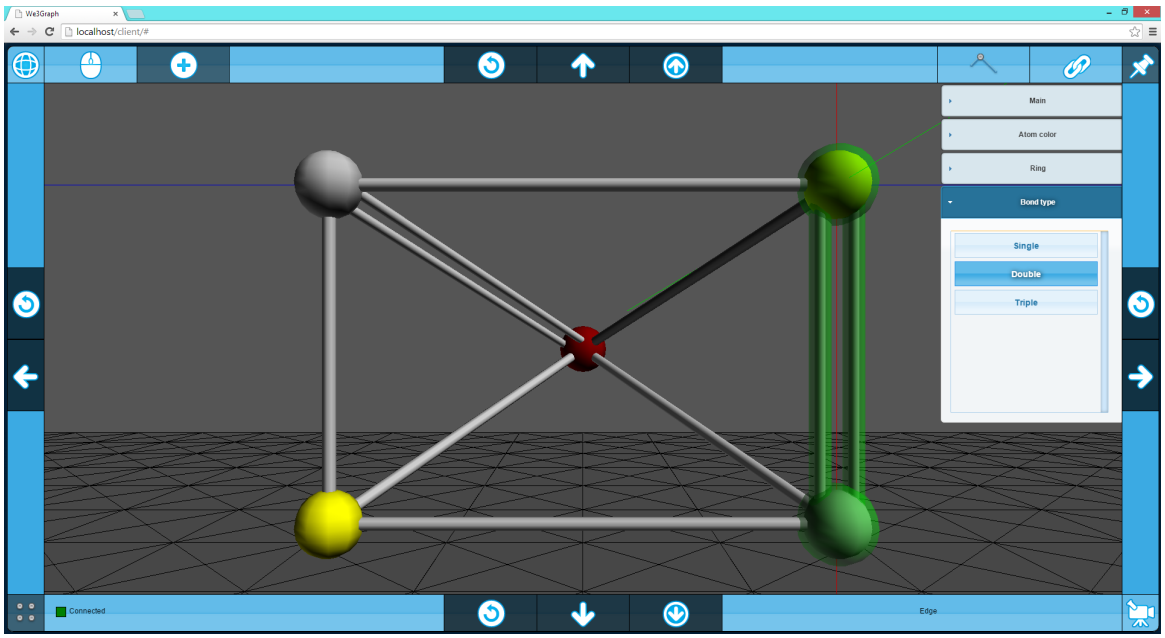

Figure 1.5: Graph viewing and editing interface of We3Graph in a Chrome browser. Accordion menu on the right.

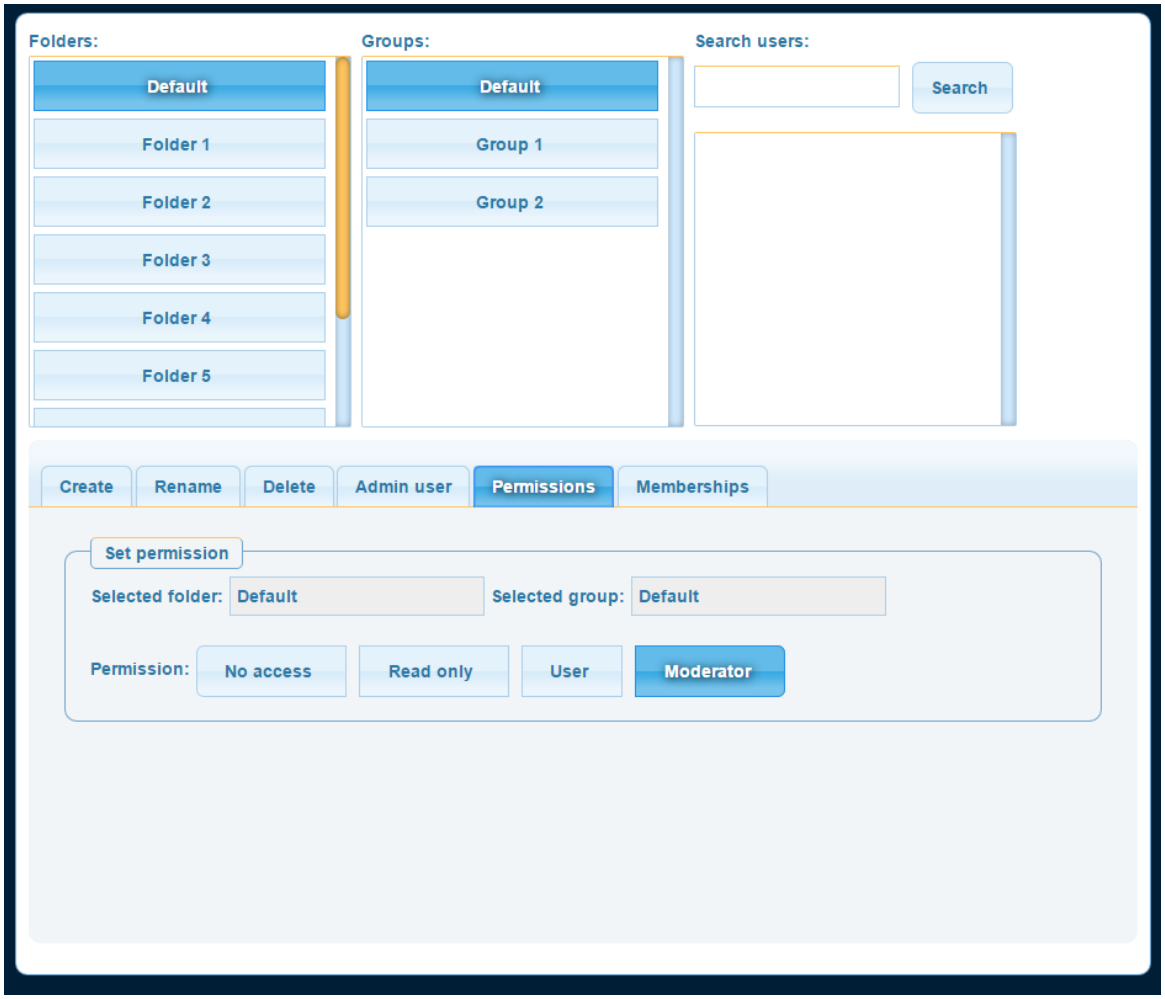

Figure 1.6: Administration interface of We3Graph.

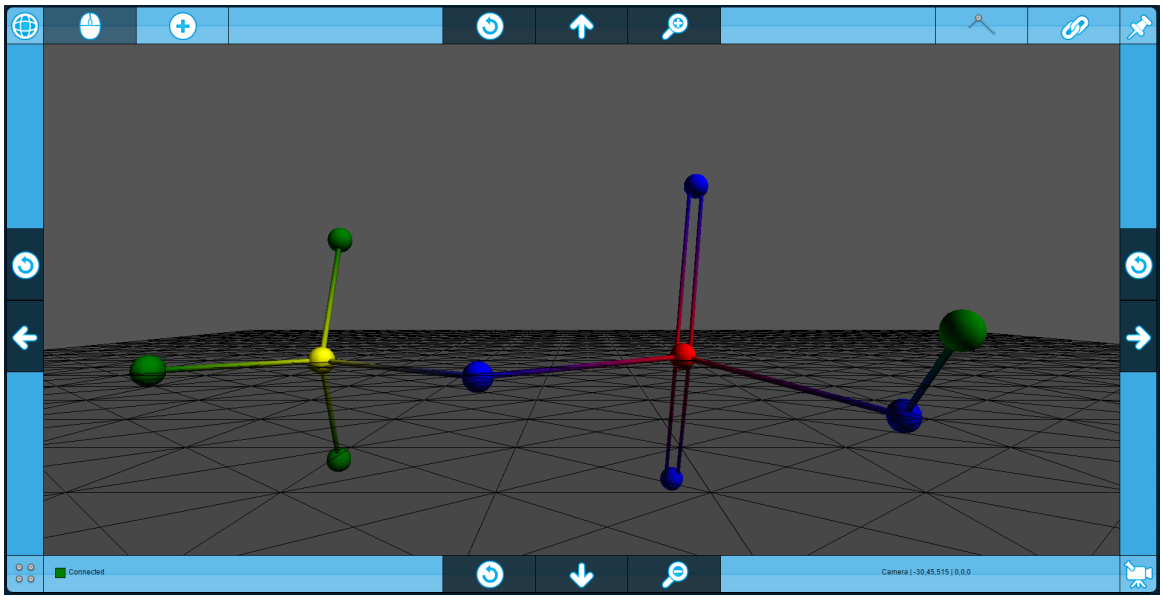

Figure 1.7: The 3D structure of a methyl hydrogen sulfate molecule shown in We3Graph

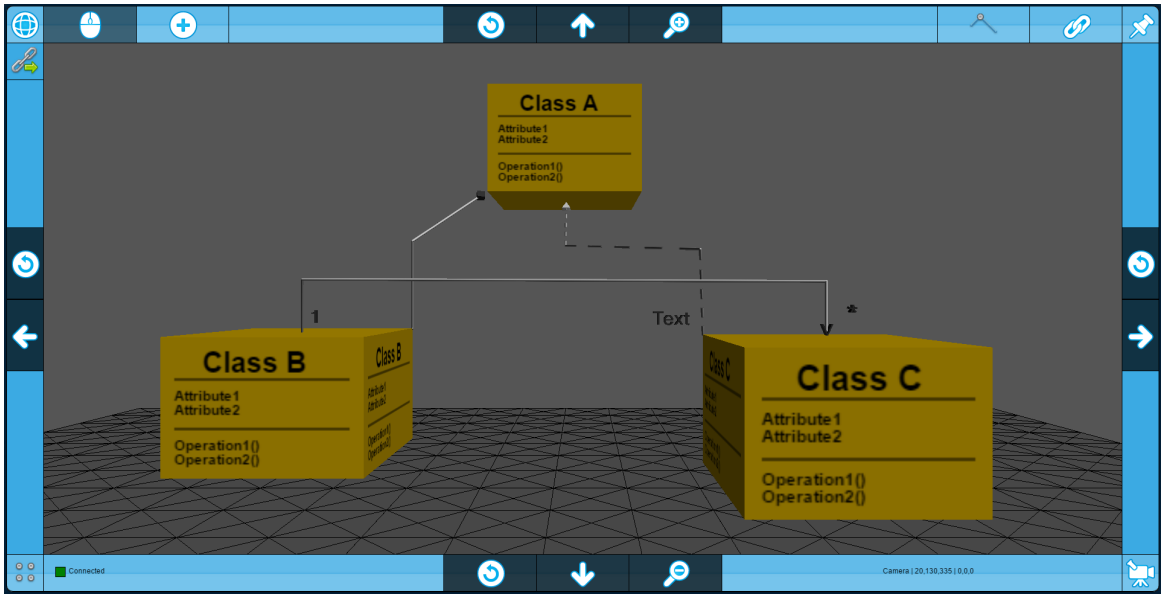

Figure 1.8: A 3D class diagram shown in We3Graph

Chapter 2

Theoretical Graph Drawing

2.1 Overview

A *drawing* of a graph, is a mapping of each vertex to a point in 2D or 3D Euclidean space and each edge to a simple curve between the mapped points of its endpoints. Most graph drawing algorithms try to find a drawing for graphs so that some aesthetic criteria are improved or some constraints are satisfied. Different constraints that can be put on the drawing, result in different types of drawings. Also different classes of graphs may require special treatment in the drawing algorithms.

In a *planar drawing* the graph is drawn on the plane such that there is no edge crossing. The lack of edge crossings in planar drawings increases the readability of the drawing. A graph that has a planar drawing is called a *planar graph*. All planar graphs are 4-colorable and the number of edges can be considered linear in terms of the number of vertices as opposed to general graphs where the number of edges can be quadratic in terms of the number of vertices. The *planar embedding* of a planar graph can be specified by defining the circular ordering of the edges incident on a vertex, that is two drawings with the same circular ordering of edges around a vertex are considered the same embedding. There are many linear time algorithms for planarity testing and computing a planar embedding of a planar graph such as [47, 52, 17]. A planar drawing of a graph partitions the plane into *faces*. Among the faces of a planar drawing, there is exactly one an unbounded face and it is called the *external face*.

Two important subclasses of planar graphs are trees and outerplanar graphs. *Trees* are connected acyclic graphs and not only can they model many concepts but also they have special combinatorial properties that can be exploited in designing and analysing algo-

rithms. An *outerplanar graph* is a graph which has a planar embedding such that all vertices are on the external face. Such an embedding is called an *outerplanar embedding*.

An *r-colorable graph* or an *r-partite graph*, is a graph where vertices can be colored with r colors such that any two adjacent vertices have different colors. If in an *r-partite graph*, any two vertices with the same color are adjacent it is called a *complete r-partite graph*. If the number of vertices of each color in a complete r-partite graph, are the same or differ by one, it is called a *balanced complete r-partite graph*.

Bends are special points on the curve of an edge, that can be used to define multi segment edge curves. The points between two consecutive bends or vertices form the individual segments of the edge. In a *grid drawing* each vertex or bend is placed at an integer grid point. A drawing where each edge is parallel to an axis of the Cartesian coordinate system, is considered an *orthogonal drawing*. Edges can be drawn as straight line segments without any bends and such a drawing is called a *straight line drawing*. A straight line drawing with no edge crossings is also called a *Fary drawing*. In a *polyline drawing*, edges are split into segments by some bend points so that each segment is a straight line. Directed graphs can have a special type of drawing that is called an *upward drawing* where all of the edges flow in the same direction. When vertices are put on layers and edges are drawn between layers flowing in the same direction, the resulting drawing is called a *hierarchical drawing*. In a *radial drawing* concentric circles are used as layers.

2.1.1 Aesthetic criteria

Some of the most important aesthetic criteria that are used to evaluate the outcome of layout algorithms are:

- Minimizing the area or volume of the drawing.
- Minimizing the number of edge crossings.
- Maximizing the symmetry.

- Minimizing the number of bends per edge.
- Distributing vertices evenly.
- Maximizing the minimum angle between two edges incident on the same vertex.
- Minimizing the edge length variance.
- Maximizing the number of edges that flow in the same direction for directed graphs.

2.2 3D graph drawing algorithms

In this section we review some of the main results and ideas in 3D graph drawing. Robert Cohen, et al. [23] showed that it is possible to have a 3D Fary grid drawing of any graph with n vertices such that the volume does not exceed $n \times 2n \times 2n$. They compute a set of points in 3D without considering any particular graph, such that if vertices of any graph are placed on these points, the straight-line edges of the graph will not produce any crossing. This universal set of points has a property that any four distinct points are not coplanar. Any edge crossing requires four coplanar vertices and thus no crossing can exist. They use the moment curve idea of [28] to compute the universal set of points but they reduce the volume using modular arithmetic as described in Algorithm 2.1. Although they proved that their $O(n^3)$ result is asymptomatically optimal for complete graphs, there are other papers that show other classes of graphs can be drawn in a lower volume and some of them are reviewed in this section.

Algorithm 2.1 3D Fary grid drawing of general graphs (Algorithm1 in Robert Cohen, et al. [23])

Input: A graph G with n vertices
Output: A 3-D drawing of G
1: Choose a prime p with $n < p <= 2n$
2: for i= 1 to n do place v_i at point $p_i = (i, i^2 \, mod \, p, i^3 \, mod \, p)$

Robert Cohen, et al. [23] also showed how to convert any 2D orthogonal grid drawing to a 3D Fary grid drawing by rolling the 2D paper using a serpentine rollup as depicted in

figure 2.1 and proving that it doesn't create any edge crossing by considering three cases for horizontal and vertical line segment pairs.

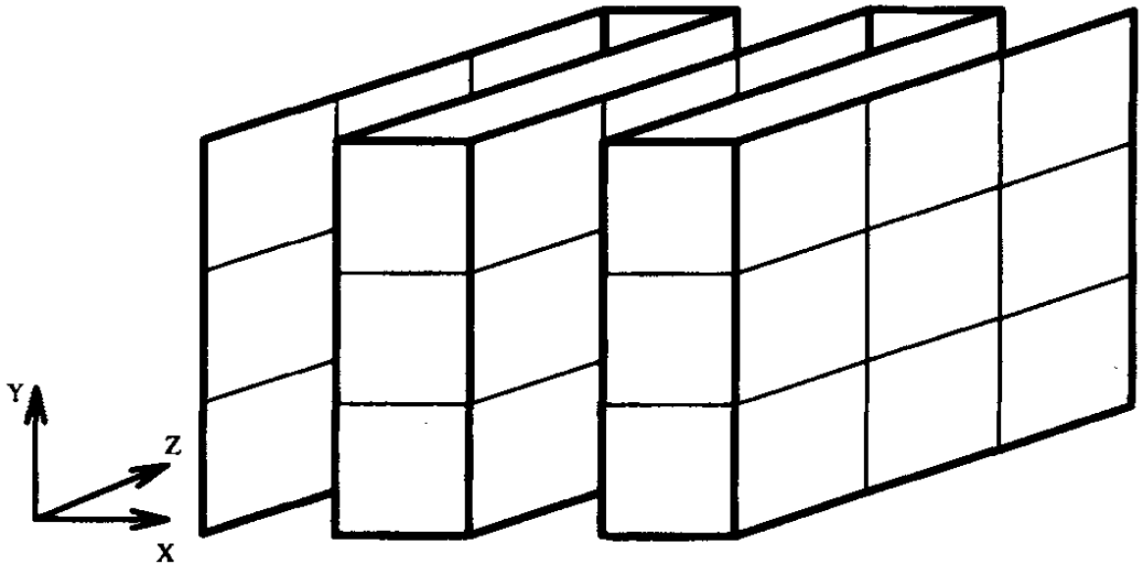

Figure 2.1: Serpentine rollup to convert 2D orthogonal grid drawings to 3D Fary drawings (Fig. 1 in Robert Cohen, et al. [23])

Tiziana Calamoneri and Andrea Sterbini [21] showed that it is possible to draw every 4-colorable graph on integer coordinates and with no crossing in an $O(n^2)$ volume by using four carefully skew lines as depicted in figure 2.2, and forbidding some positions to avoid crossings when two edges have endpoints on three different skew lines. They also showed that $\Omega(n^{\frac{3}{2}})$ is a lower bound for the volume of 3D Fary grid drawings of complete bipartite graphs.

János Pach, et al. [57] showed that for any constant r, every r-colorable graph can be drawn crossing-free on interger coordinates in $O(n^2)$ volume. Their result is based on a lemma that every balanced complete r-partite graph where n is divisible by r can be drawn with a volume not exceeding $r \times 4n \times 4rn$ using a universal set of points. To apply this lemma to r-colorable graphs, they proved that every r-colorable graph is a subgraph of a balanced complete $(2r-1)$-partite graph with less than $2n+2r$ vertices. They also showed that their result is asymptotically tight by showing that a balanced complete 2-partite graph with n vertices requires $\Omega(n^2)$ volume.

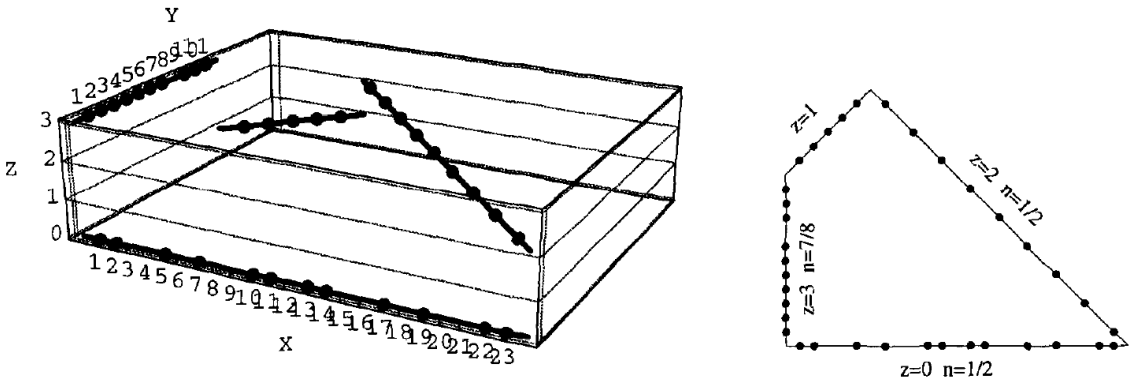

Figure 2.2: Drawing 4-colorable graphs in an $O(n^2)$ volume. The position of vertices on the left and the projection on XY plane on the right (Fig. 5 in Tiziana Calamoneri and Andrea Sterbini [21])

Prosenjit Bose, et al. [16] showed that the maximum number of non-crossing edges that can be contained in an $X \times Y \times Z$ volume is exactly $(2X-1)(2Y-1)(2Z-1) - XYZ$ and as a result, $\frac{m+n}{8}$ is a lower bound for the volume of a 3D Fary grid drawing of a graph with n vertices and m edges. The main idea behind their work is that no two edges share a midpoint and the midpoint of every edge is in a point set P defined by all points (x,y,z) such that $2x, 2y$ and $2z$ are integers.

Stefan Felsner, et al. [40] showed that it is possible to have a 3D Fary grid drawing of any outerplanar graph with n vertices on a 3D prism in an $O(n)$ volume. They used a BFS traversal of the outerplanar graph guided by the circular ordering of edges around each vertex to generate a 2D k-track drawing of G ($k \leq n$) that maintains the given outerplanar embedding as depicted in figure 2.3. A *2D k-track drawing* is a 2D grid drawing where the endpoints of each edge are either in the same track or adjacent tracks and a track is a horizontal line of a 2D grid. In the next step they wrap the 2D k-track drawing around a 3D prism to obtain a 3D drawing. In the generation of the 2D k-track drawing they increase the x coordinate as they progress in the BFS so that the wrapping process does not create any overlap.

Vida Dujmović and David Wood [31] showed that it is possible to have a 3D crossing-free grid drawing of every graph with n vertices and m edges in a $O(n+m \log q)$ volume

and with $O(\log q)$ bends per edge, where q is the queue number of the graph. The queue number of a graph is the minimum k such that there exists a linear ordering of vertices and a partition of the edges into k partitions and edges within each partition are not nested. The problem of computing the queue number of a graph is NP-Complete [46]. Di Battista, et al. [30] showed that the queue number of every planar graph is $O(\log^2 n)$ and based on the result, Vida Dujmović and David Wood [31] implied that every planar graph can be drawn crossing-free on integer coordinates in an $O(n \log\log n)$ volume and with $O(\log\log n)$ bends per edge. Di Battista, et al. [30] also showed that it is possible to have a 3D Fary grid drawing of any planar graph in an $O(n \log^8 n)$ volume.

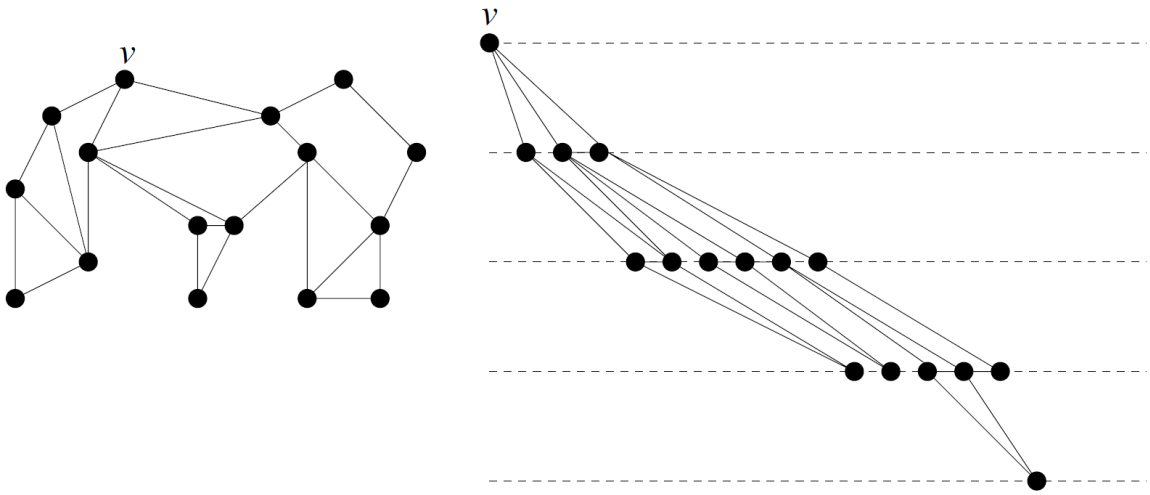

Figure 2.3: 2D k-track drawing of an outerplanar graph (Figure 12 in Stefan Felsner, et al. [40])

2.3 Point set embedding

The class of point set embedding problems studies the layout of graphs when a set of fixed points are given for the location of vertices. If the mapping between the vertices and points is specified then it is called *with mapping* otherwise it is called *without mapping*. In the with mapping variant of the problem the layout is determined only by identifying the position of the bends, where as in the without mapping variant of problem identifying the mapping between the vertices and the given point set, is also required to determine the

layout.

Here is a formulation of the two dimensional point set embedding problem (2DPSE) by Henk Meijer and Stephen Wismath [53]:

> Given a planar graph G with n vertices, $V = \{v_1, v_2, \ldots, v_n\}$, and given a set of n distinct points $P = \{p_1, p_2, \ldots, p_n\}$ each with integer coordinates in the plane, can G be drawn crossing-free on P with v_i at p_i and with a number of bends polynomial in n and in an area polynomial in n and the dimension of P?

Sergio Cabello [20] considered a version of the problem where bends are not allowed and proved that it is NP-Hard to determine whether a planar graph has a straight-line crossing-free drawing on a predefined set of points when the mapping between the vertices and the points is not specified. János Pach and Rephael Wenger [58] proved that is possible to draw any planar graph crossing-free on a predefined set of points with $O(n^2)$ bends per edge where the mapping between vertices and points is fixed. Michael Kaufmann and Roland Wiese [50] proved that it is possible to have a crossing-free drawing of every planar graph, with at most two bends per edge where each vertex can be positioned at any point of a set of predefined positions but the area of the drawing has an at least exponential growth.

In 3D similar issues can be considered. Henk Meijer and Stephen Wismath [53] formulated the three dimensional point set embedding problem (3DPSE) as follows:

> Given a graph G with n vertices, $V = \{v_1, v_2, \ldots, v_n\}$, and given a set of n distinct points $P = \{p_1, p_2, \ldots, p_n\}$ each with integer coordinates in three dimensions, can G be drawn crossing-free on P with v_i at p_i and with a number of bends polynomial in n and in a volume polynomial in n and the dimension of P?

Notice that G does not have to be planar, in contrast to the 2DPSE. In [53], this general problem is considered as an open problem and instead solutions to modified versions of the problem are given. Two new algorithms described in section 2.4 of this dissertation answer the general problem by constructing such a drawing.

The first modification to 3DPSE that is considered in [53] is to remove the polynomial volume constraint from the problem definition. They prove that K_n can be drawn crossing-free on any predefined set of integer points in 3D with at most 3 bends per edge but the volume is unbounded. The proof is done by incrementally adding edges to the graph. For each endpoint of each edge, a visible bend point outside the bounding box of the current drawing is found and the endpoint is connected to that bend. The bends found for each edge can be connected by finding a third visible bend point and connecting both to it. The idea of finding visible bend points is used in the proposed algorithms in section 2.4 but the visible bend points are found in a bounded volume.

The second modification to 3DPSE that is considered in [53] is to restrict P to the XY plane and the problem is called $3DPSE_p$. They proved that a graph with n vertices and m edges can be drawn crossing-free in 3D with vertices on a predefined set of integer points in a $W \times H$ rectangular area of the XY plane using $O(\log m)$ bends per edge and within a bounding box of $\max(W,m) \times (H+3) \times (2+\log m)$. To create such a drawing, they first introduce a method to draw a perfect matching of two sets of m points in 2D on $O(\log m)$ tracks with $O(\log m)$ bends per edge and no X-Crossings. An X-Crossing happens when there are two edges (u,v) and (w,z) such that u and w are on the same track and, v and z are on the same track and u appears before w in their track but v appears after z in their track. This track layout can be converted to 3D without any edge crossings in a box of volume of $m \times 3 \times (1+\log m)$ and with $O(\log m)$ bends per edge. — this technique is also used in the first proposed algorithm of this dissertation described in section 2.4.2. To draw an arbitrary graph in 3D, two lines are considered and for each edge two bend points are added, one one the first line and one on the second line. In the first line the order of the bends representing edges is lexicographic meaning that edges of the vertex v_i appear before the edges of the vertex v_{i+1}. For the second line the order of the bends representing edges is opposite of the first line meaning that edges of the vertex v_i appear after the edges of the vertex v_{i+1}. The two corresponding bends of each edge on these two lines are connected on $O(\log n)$ tracks

with $O(\log n)$ bends using the perfect matching technique. Next another line is added for vertices of the graph and vertices are connected to the corresponding bend of their incident edges without creating any crossings.The track layout and 3D drawing of K_6 are depicted in figures 2.4 and 2.5. For the $3DPSE_p$ problem, without loss of generality it is assumed that the vertices are ordered by X coordinate and then by Y coordinate in case of a tie. The vertices are put in the $Z = 0$ plane. The first line for the matching is placed at the $Z = -1$ plane and the second line of the matching is put on the $Z = 1 + \log m$ plane.

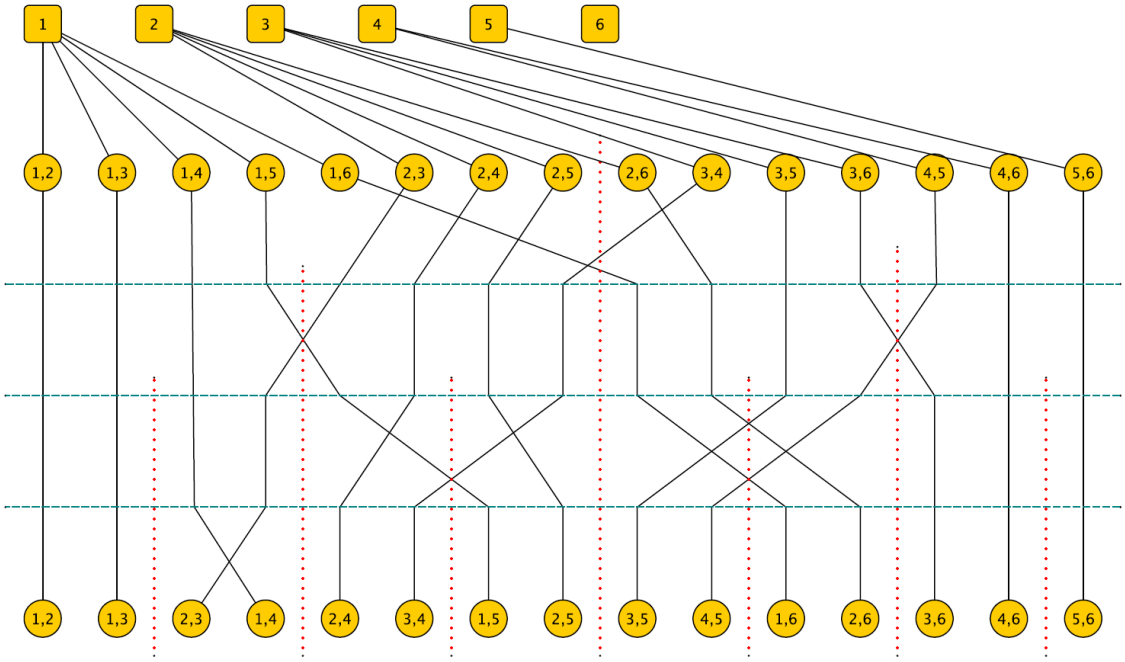

Figure 2.4: Track layout of the matching technique for K_6 (Figure 4 in Henk Meijer and Stephen Wismath [53])

2.4 The proposed algorithms

In this section two new algorithms are proposed to answer the 3D point set embedding problem (3DPSE). In section 2.4.2 a polynomial volume algorithm is given using $O(\log n)$ bends per edge. In section 2.4.3 a polynomial volume algorithm is given using 2 bends per edge but with a higher volume. First, here is a precise restatement of the 3DPSE problem.

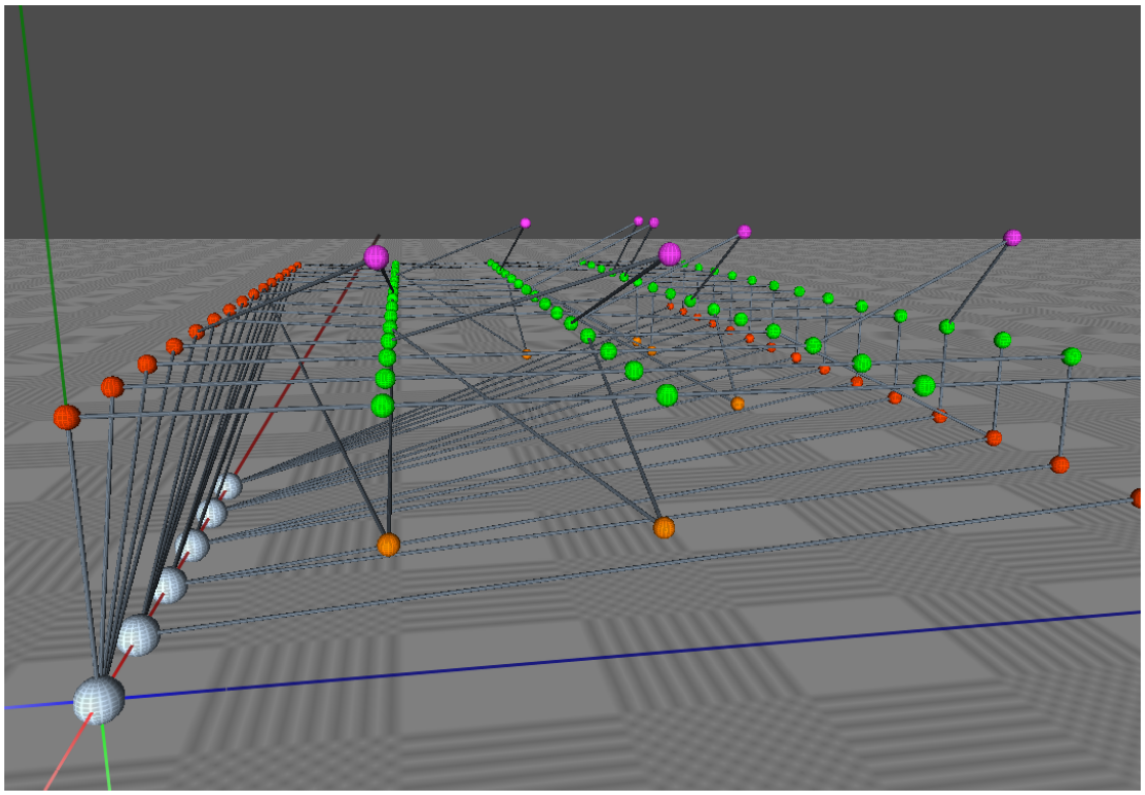

Figure 2.5: 3D drawing of K_6 with the matching technique.(Figure 7 in Henk Meijer and Stephen Wismath [53])

2.4.1 Problem Definition

Given a graph G with n vertices, $V = \{v_1, v_2, \ldots, v_n\}$, m edges, $E = \{e_1, e_2, \ldots, e_m\}$ and a given set of n distinct points $P = \{p_1, p_2, \ldots, p_n\}$ each with integer coordinates in three dimensions, can G be drawn crossing-free on P with v_i at p_i and with a number of bends polynomial in n and in a volume polynomial in n and the dimension of P such that each bend has three dimensional integer coordinates?

Without loss of generality, suppose the bounding box of P is from $(1,1,1)$ to (w,l,h).

2.4.2 The algorithm with a logarithmic number of bends per edge

In this section an algorithm is given which will produce a drawing of size $O(m+n+w) \times O(m+n+l) \times O(\log n + h)$, with at most $O(\log n)$ bends per edge.

2.4.3 General idea

The algorithm has three phases and the general ideas are outlined below while details follow later:

- Phase one: Consider two rectangles R_A and R_B perpendicular to the XY plane. R_A is one unit in front of the bounding box of the points in the direction of the Z axis and R_B is one unit from the back of the bounding box of the points in the direction of the Z axis. For each edge find two visible integer bend points, one in R_A and one in R_B. Connect the first vertex of the edge to the bend point in R_A and connect the second vertex of the edge to the bend point in R_B.

- Phase two: Consider two lines L_A and L_B parallel to the Y axis. L_A is at least one unit in front of R_A in the direction of the Z axis and two units to the left of R_A in the direction of the X axis. L_B is one unit from the back of R_B in the direction of the Z axis and two units to the left of R_A in the direction of the X axis. Connect each bend point in R_A to a corresponding integer bend point in L_A and connect each bend point in R_B to a corresponding integer bend point in L_A.

- Phase three: Each edge has two corresponding bend points in L_A and L_B. If the corresponding bend points of each edge in L_A and $L_B$are connected then they form a matching. This matching can be drawn crossing free using the perfect matching technique of [53] in a bounding box of $3 \times m \times (1 + \log m)$.

Figure 2.6 shows a conceptual picture of R_A, R_B, L_A, L_B and the bounding box of the points.

2.4.4 Phase one

Let $k = \max(n,m)$. Let P_A denote the plane $z = h+1$ and R_A denote the rectangle going from $(1,1,h+1)$ to $(2k,2k,h+1)$ in the plane P_A. Let P_B denote the plane $z = 0$ and R_B denote the rectangle going from $(1,1,0)$ to $(2k,2k,0)$ in the plane P_B. A point s is visible

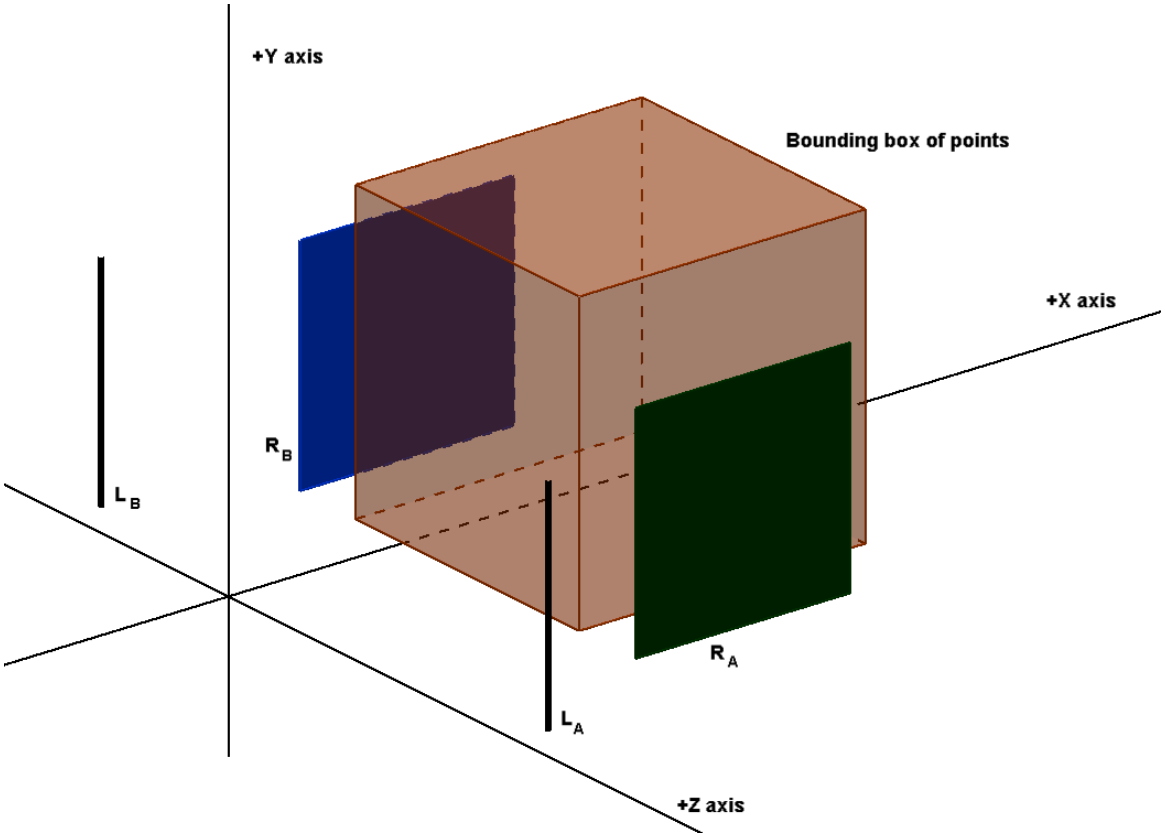

Figure 2.6: The conceptual picture of R_A, R_B, L_A, L_B and the bounding box of the points.

from point t if the line segment connecting s to t does not intersect any vertex of G or any line segment that is previously drawn. The edges are considered one by one in m steps. At the i^{th} step $(1 \leq i \leq m)$, the i^{th} edge e_i, connecting vertices u_i and w_i is considered. Now a visible integer bend point a_i from u_i is found in R_A and a line segment α_i is drawn between u_i and a_i. Next a visible integer bend point b_i from w_i is found in R_B and a line segment β_i is drawn between w_i and b_i. At the end of this phase each edge has one corresponding bend point in R_A and one corresponding bend point in R_B.

To prove that there is always a visible integer bend point from u_i in R_A, or from w_i in R_B, at the i^{th} step of this phase of the algorithm, consider that there are only two ways that an integer bend point in R_A or R_B becomes invisible from u_i or w_i:

1. A previously drawn line segment is between R_A and u_i, or R_B and w_i. The previously drawn line segment can be any of α_j or β_j for $1 \leq j < i$, or α_i for w_i. There are at most

$2k-1$ such line segments and each line segment can make at most $2k$ integer points of R_A or R_B invisible. So this case will make at most $(2k-1)2k$ integer points of R_A or R_B invisible. To prove that each such line segment such as e connecting vertices or bend points q and t, will make at most $2k$ integer points in R_A or R_B invisible, consider the plane P_{uqt} containing u_i, q and t. If the plane P_{uqt} intersects with the plane P_A or P_B the intersection will be a line. This line can contain at most $2k$ integer points of R_A or R_B. If u_i, q, and t are collinear, at most one integer point of R_A or R_B is made invisible.

2. A vertex is between R_A and u_i, or R_B and w_i: This can be any vertex other than u_i or w_i. Each such vertex can make at most one integer point of R_A or R_B invisible. There are at most $k-1$ such vertices. So this case can make at most $k-1$ integer points of R_A or R_B invisible.

Subtracting the maximum number of invisible points of both cases from the number of integer points of R_A or R_B, leaves at least $k+1$ visible points as shown in equation 2.1.

$$4k^2-(2k-1)2k-(k-1)=k+1 \tag{2.1}$$

2.4.5 Phase two

Let $\lambda = \max(h+2, \log m)$. Let L_A denote the line segment going from $(-1,1,\lambda)$ to $(-1,m,\lambda)$ and let L_B denote the line segment going from $(-1,1,-1)$ to $(-1,m,-1)$.

For each bend point a_i in R_A, find a corresponding integer bend point in L_A called $\overline{a}_i$ and draw a line segment between a_i and $\overline{a}_i$. To find such corresponding bend points, consider the integer bend points of L_A in the order of increasing Y coordinate and consider a_i integer bend points in the order of X coordinate and in case of a tie in the order of Y coordinate, and match them one by one. This ordering will avoid any crossings.

Similarly, for each bend point b_i in R_B, find a corresponding integer bend point in L_B called $\overline{b}_i$ and draw a line segment between b_i and $\overline{b}_i$. To find such corresponding bend

points, consider the integer bend points of L_B in the order of increasing Y coordinate and consider b_i integer bend points in the order of X coordinate and in case of a tie in the order of Y coordinate, and match them one by one. This ordering will avoid any crossings. At the end of this phase each edge has four corresponding bend points, one in R_A, one in L_A, one in L_R and one in L_B.

2.4.6 Phase three

Each edge e_i has a corresponding bend point $\overline{a}_i$ in L_A and a corresponding bend point $\overline{b}_i$ in L_B. If each $\overline{a}_i$ is connected directly to each $\overline{b}_i$ they form a perfect matching but it may introduce crossings. To avoid crossings the perfect matching technique of [53] is used to draw this perfect matching in 3D. Such a 3D perfect matching drawing can be drawn in a bounding box of $3 \times m \times (1 + \log m)$ using the [-2,0] range of X coordinates and at most $O(\log n)$ bends per edge. Also it is notable that this phase does not use any bend point on the two lines $X = 0, Z = \lambda$ and $X = 0, Z = -1$, otherwise it may introduce crossings with the line segments of the previous phase. This phase will add at most $O(\log n)$ bends per edge, at most $O(\log n)$ to the dimension of drawing in the Z direction, and at most three units to the dimension of drawing in X direction.

2.4.7 Summary

Each phase of algorithm does not have any crossing inside it. The three phases use different partitions of space which will avoid crossings between the three phases.

To find the visible points, for each vertex v, the algorithm maintains a set of integer points in R_A or R_B that are visible from v. The set is implemented using a balanced binary search tree. After adding each line segment at each step of the algorithm, for each vertex v, the algorithm removes the integer points blocked by that line segment from the set of visible points of v. The algorithm has $O(m \cdot n \cdot k \cdot \log n)$ time complexity and $O(nk^2)$ memory complexity. The algorithm is summarized in Theorem 2.1 and Algorithm 2.2. Also figures 2.7 and 2.8 show the drawing of K_5 on a given point set using the proposed algorithm in

We3Graph.

Theorem 2.1. *Given a graph G with m edges, and n vertices, $V = \{v_1, v_2, \ldots, v_n\}$, and a given set of n distinct points $P = \{p_1, p_2, \ldots, p_n\}$ each with integer coordinates in three dimensions, G can be drawn crossing-free on P with v_i at p_i and with at most $O(\log n)$ bends per edge and in a $O(m+n+w) \times O(m+n+l) \times O(\log n + h)$ volume such that each bend has three dimensional integer coordinates. The drawing can be produced in $O(m \cdot n \cdot k \cdot \log n)$ time and $O(nk^2)$ memory.*

Algorithm 2.2 The algorithm with logarithmic number of bends per edge

R_A denotes the rectangle going from $(1,1,h+1)$ to $(2k,2k,h+1)$ in the plane $z=h+1$

R_B denotes the rectangle going from $(1,1,0)$ to $(2k,2k,0)$ in the plane $z=0$

1: $\lambda = \max(h+2, \log m)$

2: Let S_v be the set of all visible integer points from the vertex v, in R_A.

3: Let $\hat{S}_v$ be the set of all visible integer points from the vertex v, in R_B.

4: **for all** vertex v in V **do**

5: **for all** vertex v_2 in V - v **do**

6: Remove the point in S_v or the point in $\hat{S}_v$ that is blocked by v_2 from v (if it exists).

7: **for all** edge $e_i = (u_i, w_i)$ in E **do**

8: Let a_i be a point in S_u

9: Draw a line segment α_i from u_i to a_i.

10: **for all** vertex v in V **do**

11: Remove every point in S_v and $\hat{S}_v$ that is blocked by α_i from v.

12: Let b_i be a point in $\hat{S}_w$

13: Draw a line segment β_i from w_i to b_i.

14: **for all** vertex v in V **do**

15: Remove every point in S_v and $\hat{S}_v$ that is blocked by β_i from v.

16: counter=1

17: **for all** α_i ordered by x coordinate and in case of a tie by y coordinate **do**

18: Draw a line segment between α_i and the point $\overline{a}_i = (-1, counter, \lambda)$.

19: counter++

20: counter=1

21: **for all** β_i ordered by x coordinate and in case of a tie by y coordinate **do**

22: Draw a line segment between β_i and the bend point $\overline{b}_i = (-1, counter, -1)$.

23: counter++

24: Use the technique of [53] for drawing a perfect matching in 3D to connect each $\overline{a}_i$ to $\overline{b}_i$.

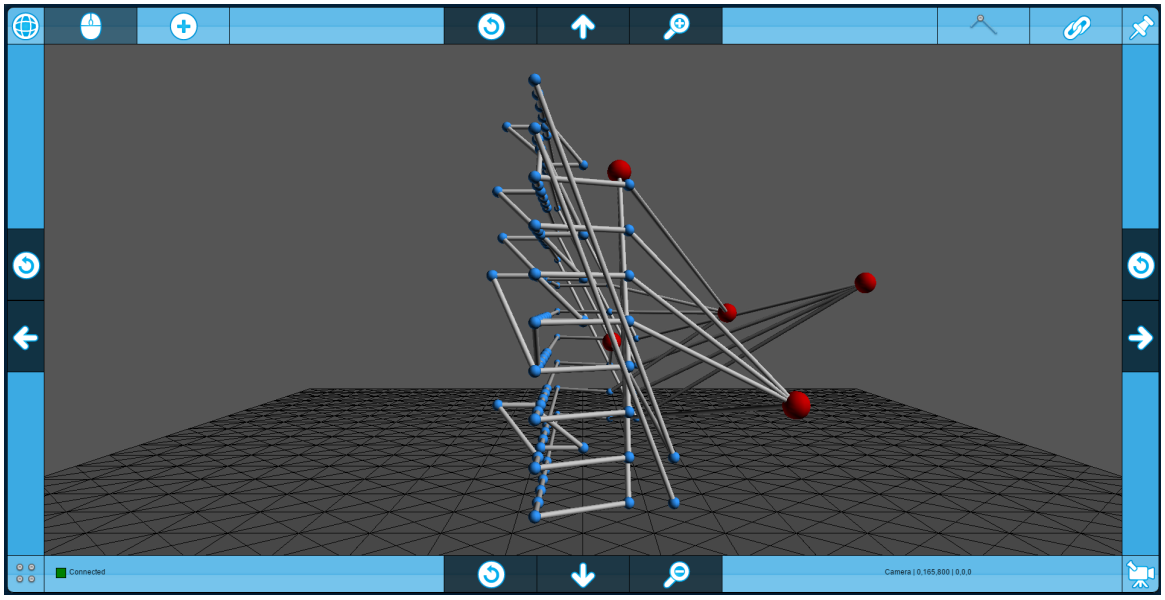

Figure 2.7: 3D drawing of K_5 on a given point set using the first proposed algorithm. Y axis upward and camera looking toward the negative side of Z axis direction.

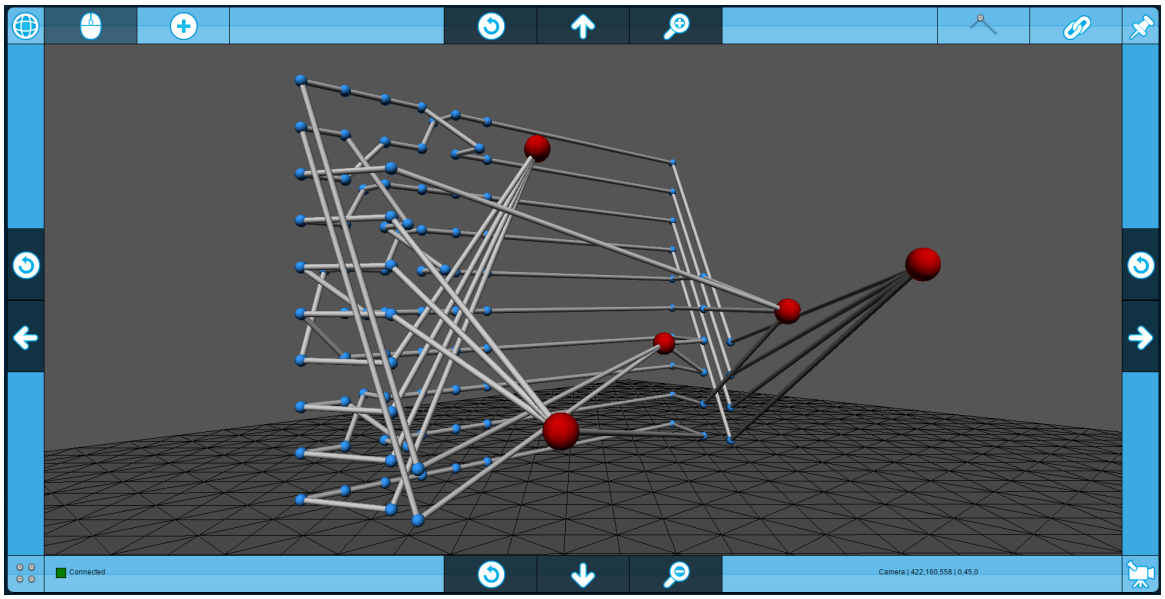

Figure 2.8: 3D drawing of K_5 on a given point set using the first proposed algorithm. Y axis upward and camera looking toward the (-1,-1,0) direction.

2.4.8 The algorithm with two bends per edge

In this section an algorithm is given which will produce a drawing of size

$O(m+n+w) \times O(m+n+l) \times O(m+n+h)$, with at most 2 bends per edge. The algorithm

considers rectangles parallel to the XY plane in front of the bounding box of the points in the direction of the Z axis and for each edge, finds two visible integer bend points in one of the rectangles and connects them directly. Here is a detailed explanation of the algorithm.

Let $k = \max(n,m)$. For any particular integer ϕ, let P_ϕ denote the plane $z = \phi$ and R_ϕ denote the rectangle going from $(1,1,\phi)$ to $(3k,3k,\phi)$ in the plane P_ϕ. A point s is visible from point t if the line segment connecting s to t does not intersect any vertex of G or any line segment that is previously drawn. At the i^{th} step $(1 \leq i \leq m)$, the i^{th} edge e_i, connecting vertices u_i and w_i is considered. Now a visible integer bend point a_i from u_i is found in R_{h+i} and a line segment α_i is drawn between u_i and a_i. Next a visible integer bend point b_i from w_i is found in R_{h+i} and a line segment β_i is drawn between w_i and b_i. Then a line segment γ_i is drawn between a_i and b_i.

Figure 2.9 shows a conceptual picture of R_{h+1}, R_{h+i} and the bounding box of the points.

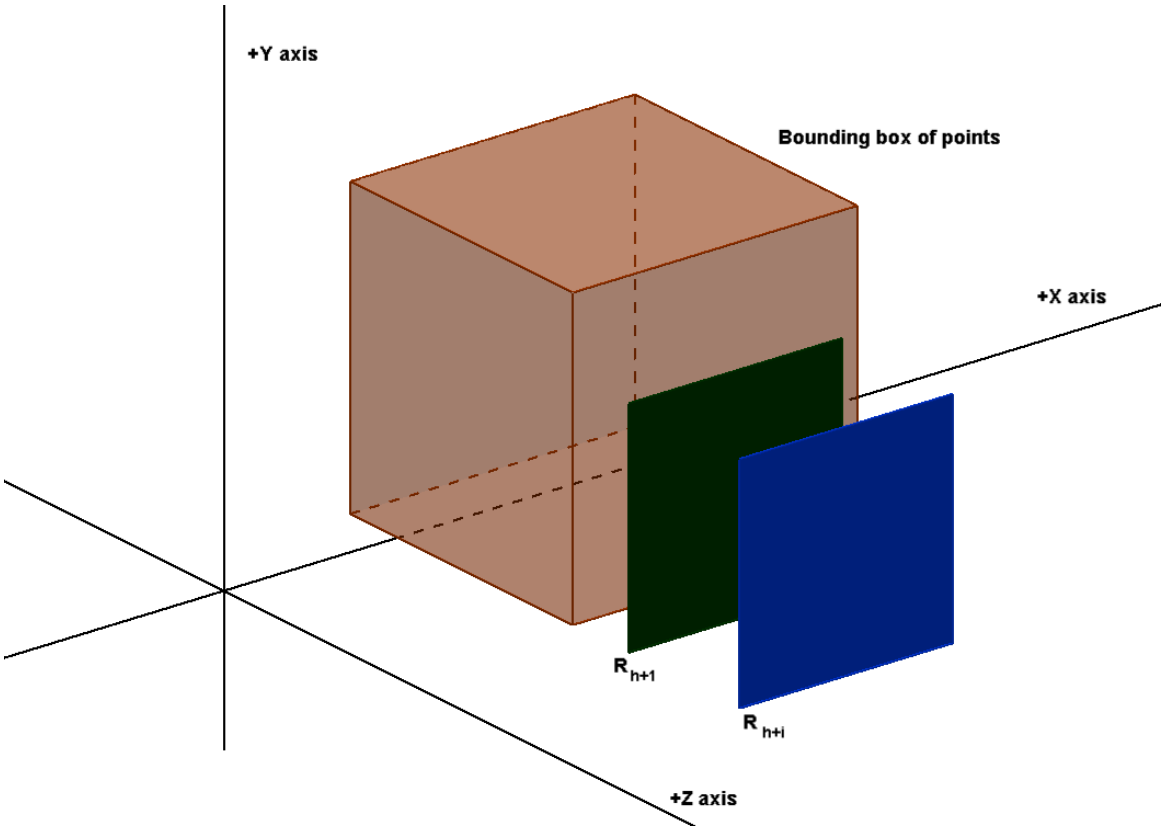

Figure 2.9: The conceptual picture of R_{h+1}, R_{h+i} and the bounding box of the points.

To prove that there is always a visible integer bend point from u_i or w_i in R_{h+i} at the i^{th} step of the algorithm, consider that there are only two ways that an integer bend point in R_{h+i} becomes invisible from u_i or w_i:

1. A previously drawn line segment is between R_{h+i} and, u_i or w_i. The previously drawn line segment can be any of α_j, β_j or γ_j for $1 \leq j < i$, or α_i for w_i. There are at most $3k-2$ such line segments and each line segment can make at most $3k$ integer points of R_{h+i} invisible. So this case will make at most $(3k-2)3k$ integer points of R_{h+i} invisible. To prove that each such line segment such as e connecting vertices or bend points q and t, will make at most $3k$ integer points in R_{h+i} invisible, consider the plane P_{uqt} containing u_i, q and t. If the plane P_{uqt} intersects with the plane P_{h+i} the intersection will be a line. This line can contain at most $3k$ integer points of R_{h+i}.If u_i, q, and t are collinear, at most one integer point of R_{h+i} is made invisible.

2. A vertex is between R_{h+i} and, u_i or w_i: This can be any vertex other than u_i or w_i. Each such vertex can make at most one integer point of R_{h+i} invisible. There are at most $k-1$ such vertices. So this case can make at most $k-1$ integer points of R_{h+i} invisible.

Subtracting the maximum number of invisible points of both cases from the total number of integer points of R_{h+i}, leaves at least $5k+1$ visible points as shown in equation 2.2.

$$9k^2 - (3k-2)3k - (k-1) = 5k+1 \tag{2.2}$$

To find the visible points, at the i^{th} step, the algorithm considers every point in R_{h+i}, every previously drawn line segment and every vertex other than u_i or w_i. The algorithm has $O(mk^3)$ time complexity and $O(k)$ memory complexity. The algorithm is summarized in Theorem 2.2 and Algorithm 2.3. Also figures 2.10 and 2.11 show the drawing of K_5 on a given point set using the proposed algorithm in We3Graph.

Theorem 2.2. *Given a graph G with m edges, and n vertices, $V = \{v_1, v_2, \ldots, v_n\}$, and a given set of n distinct points $P = \{p_1, p_2, \ldots, p_n\}$ each with integer coordinates in three dimensions, G can be drawn crossing-free on P with v_i at p_i and with at most two bends per edge and in a $O(m+n+w) \times O(m+n+l) \times O(m+n+h)$ volume such that each bend has three dimensional integer coordinates. The drawing can be produced in $O(m(m+n)^3)$ time and $O(m+n)$ memory.*

Algorithm 2.3 The algorithm with two bends per edge

R_ϕ denotes the rectangle going from $(1,1,\phi)$ to $(3k,3k,\phi)$ in the plane $z = \phi$

1: **for all** edge $e_i = (u_i, w_i)$ in E **do**

2: Select a visible integer bend point a_i from u_i in R_{h+i} by considering every point in R_{h+i}, every previously drawn line segment and every vertex other than u_i.

3: Draw a line segment α_i from u_i to a_i.

4: Select a visible integer bend point b_i from w_i in R_{h+i} by considering every point in R_{h+i}, every previously drawn line segment and every vertex other than w_i.

5: Draw a line segment β_i from w_i to b_i.

6: Draw a line segment γ_i from a_i to b_i.

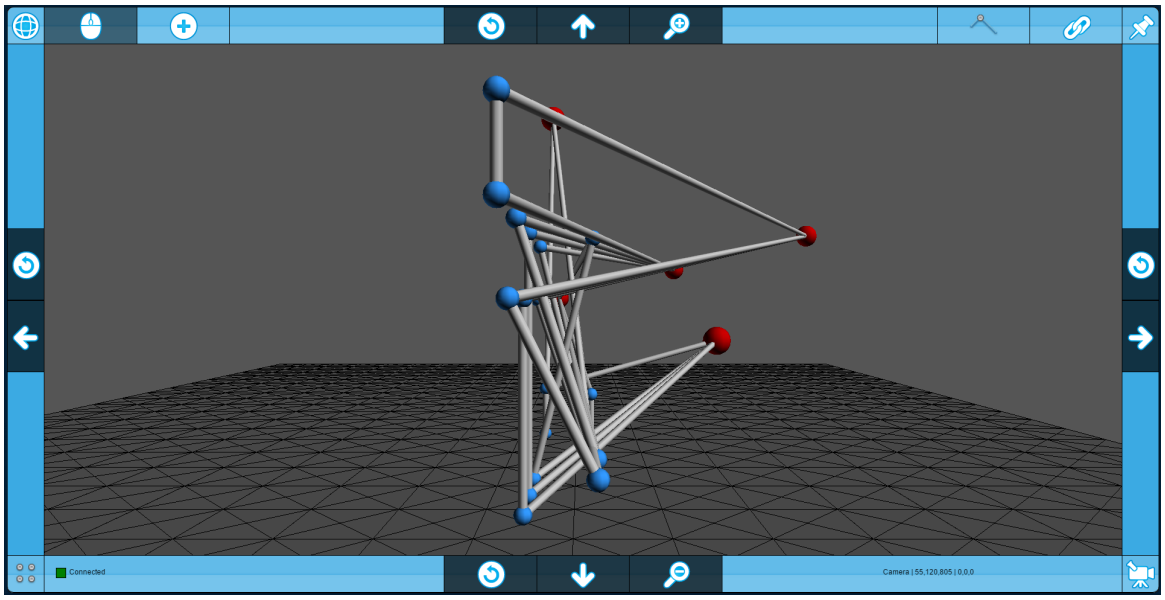

Figure 2.10: 3D drawing of K_5 on a given point set using the second proposed algorithm. Y axis upward and camera looking toward the negative side of Z axis direction.

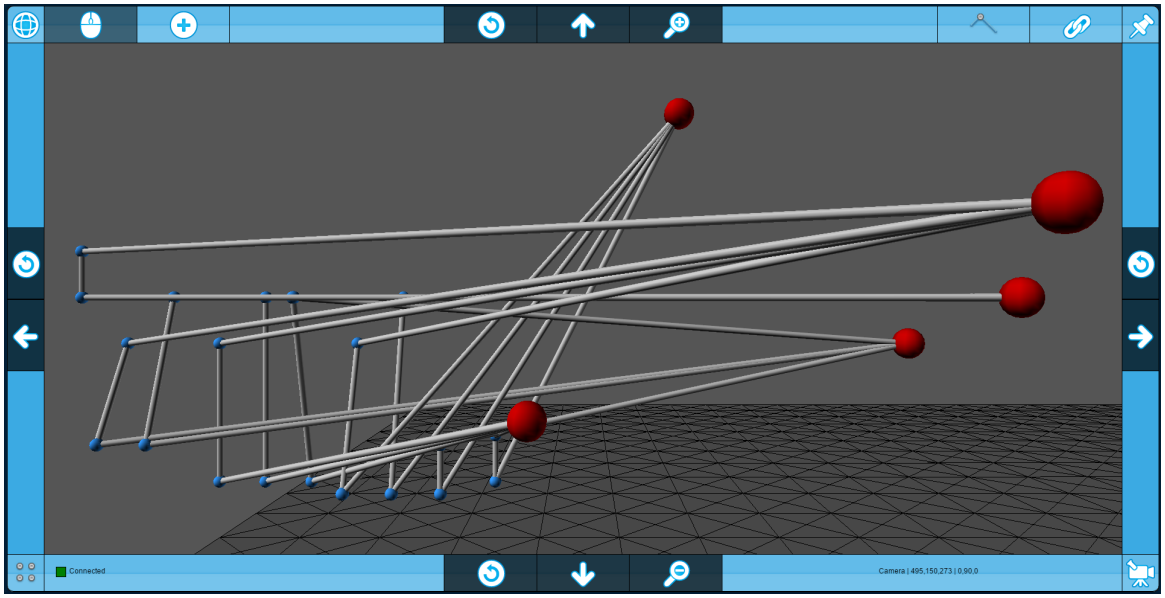

Figure 2.11: 3D drawing of K_5 on a given point set using the second proposed algorithm. Y axis upward and camera looking toward the negative side of X axis direction.

Chapter 3

Previous 2D and 3D Graph Drawing Software

This chapter reviews some of the previous efforts on graph drawing software with particular emphasis on 3D applications. These software packages provide visualization, manipulation or algorithms for graphs as an abstract model or as a model used in an application of graph drawing.

There are some software packages that are solely implementations of algorithms and do not provide any user interface. Among them we can mention AGD (Algorithms for Graph Drawing) [45, 49],GDS (Graph Drawing Server) [18, 19], GDToolkit (Graph Drawing Toolkit) [29] and OGDF(Open Graph Drawing Framework) [22, 1].

There are some software packages that are designed for 2D graph drawing such as Graphvis [37, 38], Pajek [14], PIGALE [27] and Tulip [26, 12]. Tulip can also handle 3D graph drawing but the user interface is more tuned for 2D graph drawing and most algorithm plugins bundled with it are for 2D graph drawing.

There are some software systems such as GIOTTO3D [44], Gluskap [35], H3Viewer [56, 55], WilmaScope [34, 33] that are primarily designed for 3D graph drawing. Joel Bennett and Stephen Wismath [15] extended Gluskap to provide 3D printing for graphs. Farshad Barahimi and Stephen Wismath [13] extended Gluskap to provide a virtual reality environment for graphs using the Oculus Rift. H3Viewer uses the 3D hyperbolic space for layout and visualization of graphs. GIOTTO3D is aimed at layout and visualization of hierarchical structures in 3D.

There are some software packages that are especially designed for biology applications. Among them we can name BioJAKE [60],bioWeb3D[59],Cytoscape [61, 62] and Web 3DNA [66]. Cytoscape was originally designed for Biology but later changed to a

general platform for visualizing and analysis of networks. Some software packages such as ChemDraw [24], Chemlab [2], PyMol [3] and GLMol [4] target chemistry applications. Biology and chemistry sometimes overlap in the field of BioChemistry.

General diagramming software is another category of software packages related to graph drawing which address the need to draw different diagrams to represent information or processes. Among them we can mention the open source Dia [5] project, the web based diagramming software of Gliffy [6], Microsoft Visio [7] from the Microsoft office family of products and yED. Flowcharts, concept maps, mind maps, organizational charts are some of the diagram types that software packages in this category can draw. Sometimes these software packages can draw more specialized diagram types such as network diagrams, UML (Unified Modelling Language) diagrams, entity-relationship diagrams, data flow diagrams and PERT (Program Evaluation and Review Technique) charts. While the yEd editor gives the end user an ability to draw diagrams, the yFiles library from the same company provides programmers with a diagramming component to be used in their programs as well as some layout algorithms.

There is another group of software packages that are designed primarily for software engineering applications such as Enterprise Architect [8], MagicDraw [9], Rational Rose [10] and Visual Paradigm[11]. UML (Unified Modelling Language) diagrams play an important role in these software packages. BPMN(Business Process Model and Notation) diagram is another type of diagram that is used in these software packages. Generating source code from the diagram and reverse engineering the source code to diagram is a notable feature of software packages in this category.

3.1 Open Graph Drawing Framework(OGDF)

The Open Graph Drawing Framework [22, 1] is an open-source C++ software library for graph data structures and algorithms. The framework provides different layout algorithms such as:

- Orthogonal layout algorithms.
- Straight-line and polyline planar layout algorithms.
- Planarization layout algorithms.
- Layered layout algorithms.
- Energy-based and force-directed layout algorithms.
- Tree layout algorithms.
- Circular layout algorithms.

The framework is used in some other software packages such as Tulip [26, 12] as well as more than 20 publications.

3.2 Tulip

Tulip [26, 12] is a data visualization software package written in C++ which uses graphs as the primary model of data. Also Tulip allows properties to be attached to vertices or edges which helps different visualization views of Tulip to visualize the graph accordingly. In addition to the Node Link Diagram View which provides a typical visualization of a graph, Tulip has also other types of views such as:

- Spreadsheet view: Allows the user to view and manipulate properties of vertices and edges of a graph in a spreadsheet.
- Adjacency matrix view: Displays the adjacency matrix of a graph.
- Geographic view: Allows different map modes to be used to visualize geographically related graphs.
- Histogram view: Displays properties of vertices or edges of a graph using histograms.

- Parallel coordinates view: Shows the relationship between two or more properties of vertices or edges of a graph.

Figure 3.1 shows the user interface of Tulip, displaying a 5 vertex planar graph in the Node Link Diagram view. The algorithms panel on the left side of the image, allows the user to apply different algorithms on the graph.

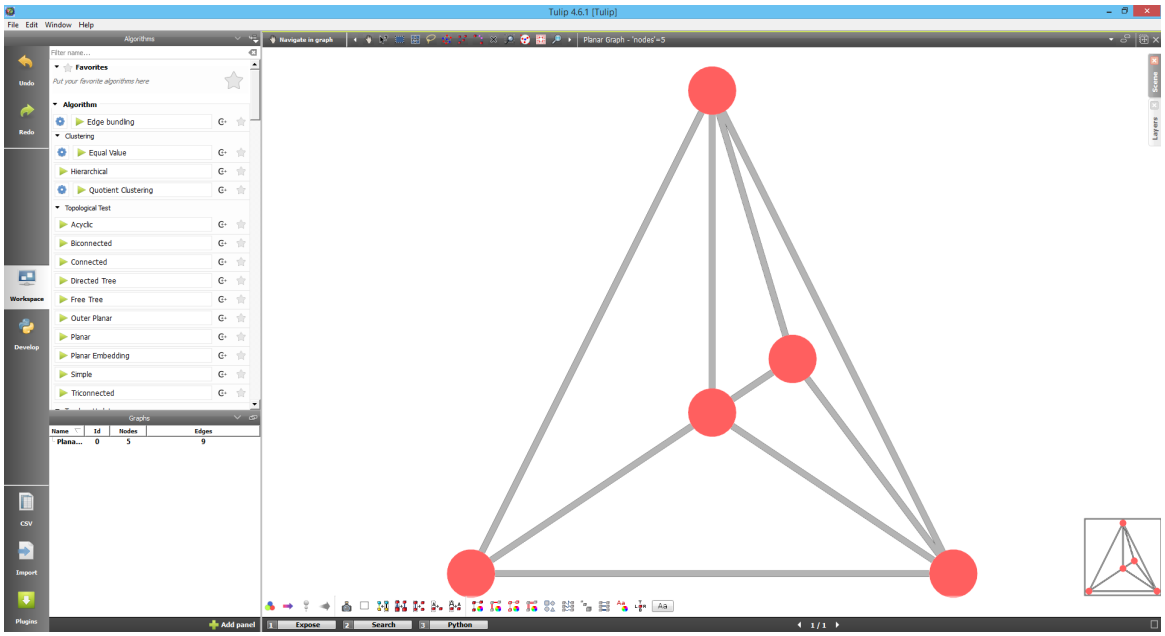

Figure 3.1: The user interface of Tulip, displaying a 5 vertex planar graph in the Node Link Diagram view.

Although Tulip can handle 3D graph drawing, the user interface is more tuned for 2D graph drawing and most of the algorithm plugins bundled with it are for 2D graph drawing.

The plugins can be written in C++ or python. The software package is bundled with several built in plugins for layout algorithms, topological testing algorithms, etc. Some of the built in plugins are from the Open Graph Drawing Framework (OGDF).

Tulip can import graphs from different file formats such as Tulip TLP format, Tulip JSON format, GML, graphVis format(.dot), pajek format(.net) and an adjacency matrix file format. The graphs can be exported to Tulip TLP format, Tulip JSON format, SVG format and GML format. The user can also start using some predefined graphs such as Complete

graphs, Complete Trees, Grids and Random graphs. Tulip has been used in more than 80 publications.

3.3 Cytoscape

Cytoscape is a network analysis and visualization platform originally designed for biological research. The open source platform can be used to visualize molecular interaction networks and is capable of attaching other information such as annotations and gene expression profiles to the elements of these networks. Cytoscape is written in Java, and plugins called apps can be written in Java. Cytoscape provides an app store where apps can be downloaded or new apps can be submitted. Also an app named cyREST provides a REST web service which can be used by other programming languages to interact with Cytoscape. Some of the layout algorithms of Cytoscape are from yFiles while there are other layout algorithms that do not exist in yFiles such as an edge-weighted force directed algorithm for similarity analysis in biology called BioLayout [39] based on the general force directed algorithm of Fruchterman and Reingold [43]. Cytoscape also provides a javascript library called Cytoscape.js which helps web applications to visualize Cytoscape networks. Cytoscape supports file formats such as SIF, XGMML, BioPax, PSI-MI, GraphML, KGML, SBML, OBO and Gene association. Cytoscape can connect to biological databases and retrieve information from them. Cytoscape has been used in more than 60 publications. Figure 3.2 shows the user interface of Cytoscape, displaying the E. coli interactome.

3.4 Gluskap

Gluskap is a general 3D graph drawing software package written in python that allows the visualization and manipulation of graphs in 3D. The software has evolved for over 10 years and is explicitly designed for 3D. Joel Bennett and Stephen Wismath [15] extended Gluskap to provide 3D printing of graphs. Farshad Barahimi and Stephen Wismath [13] extended Gluskap to provide a virtual reality environment for viewing graphs using the

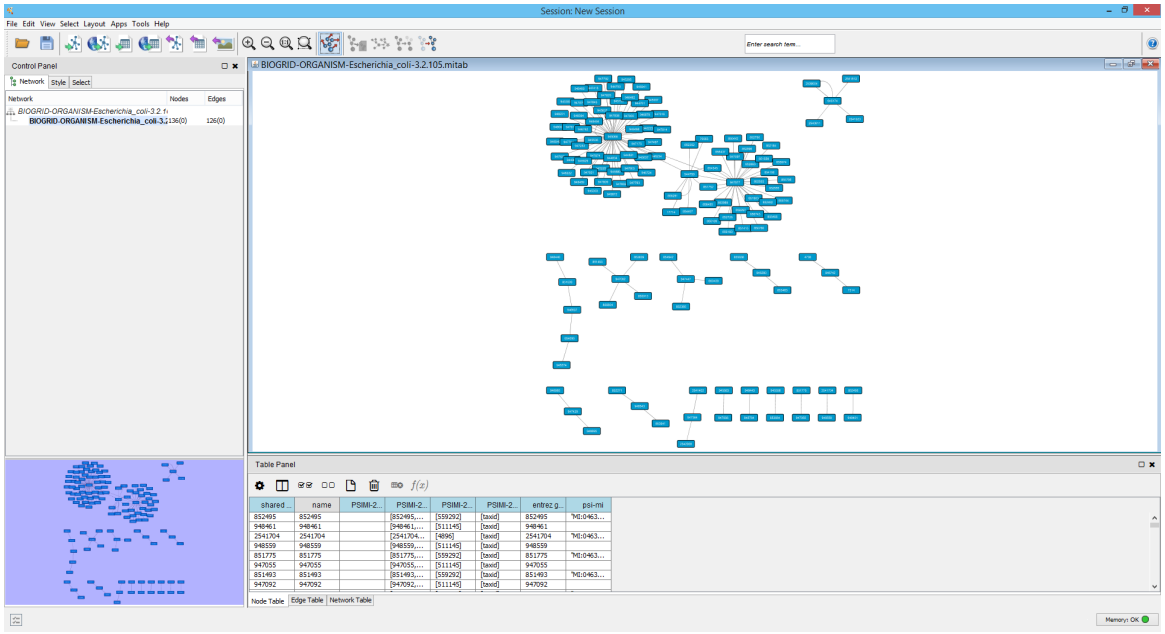

Figure 3.2: The user interface of Cytoscape, displaying the E. coli interactome.

Oculus Rift. Gluskap also supports stereoscopic visualization of graphs using a side by side 3D method or interleaving method. Gluskap plugins can be written in python. The layout plugins bundled with Gluskap are related to 3D representation of graphs rather than 2D representation of the graphs such as the layout algorithm of [23] for drawing graphs in 3D based on the moment curve idea of [28], the layout algorithm of [54] for drawing graphs in 3D using one bend, the layout algorithm of [36] for drawing graphs in 3D using two bends and the layout of [53] for drawing graphs on a given point set with a logarithmic number of bends. The internal Gluskap file format is mg2 but it also supports importing and exporting from Tulip TLP format, Graphvis dot format, GraphML format, GML format, Trivial Graph Format and CSV format. Gluskap is also capable of exporting the current view of the 3D scene to POV-Ray which can be used to render the graph using the POV-Ray software. Figure 3.3 shows the user interface of Gluskap, displaying a 3D drawing of the Peterson graph.

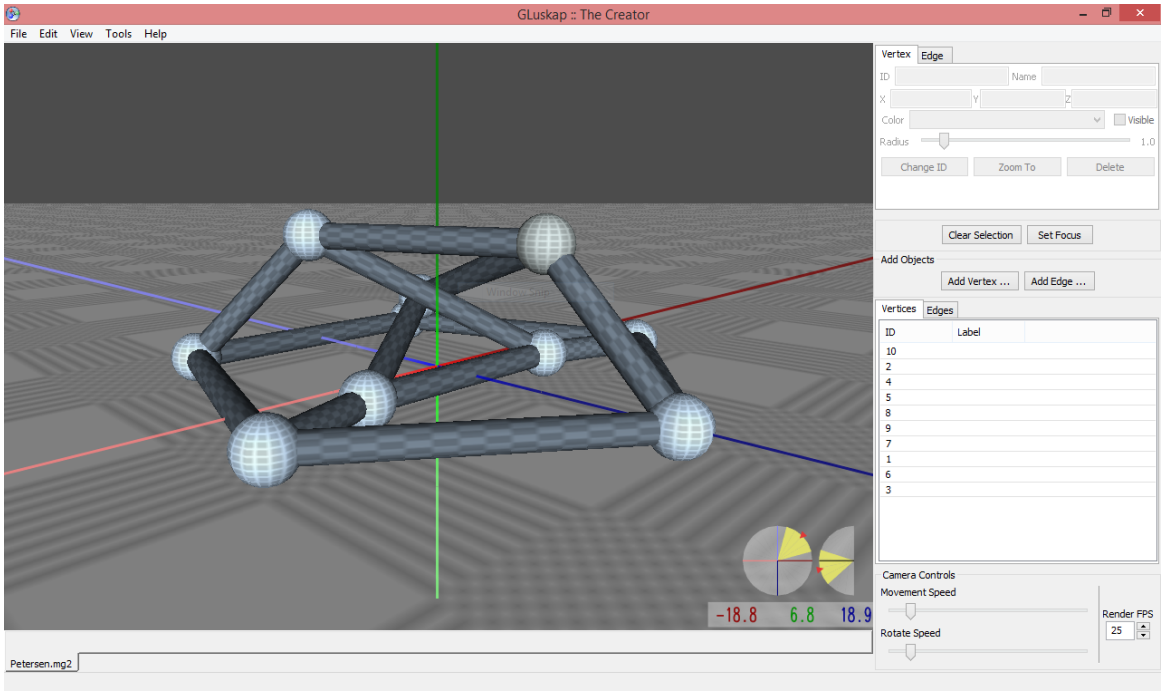

Figure 3.3: The user interface of Gluskap, displaying a 3D drawing of the Peterson graph.

3.5 GLmol

GLMol is a web based visualizer for 3D molecules based on WebGL and written with Javascript and THREE.js. GLMol provides the user with the ability to view molecules in a web browser but does not provide editing capabilities for the molecule. GLMol can read from PDB, SDF/Mol and XYZ file formats. It is also capable of reading PDB files from RCSB PDB server and SDF files from the NCBI PubChem server. It can show the molecules using different representations such as line, ball and stick, stick, sphere, star, ribbon, strand, etc. NDKMol is a version of GLMol for android written using C++ and android NDK. Figure 3.4 shows the user interface of GLMol in a Chrome web browser, displaying the 3D structure of the Porin protein.

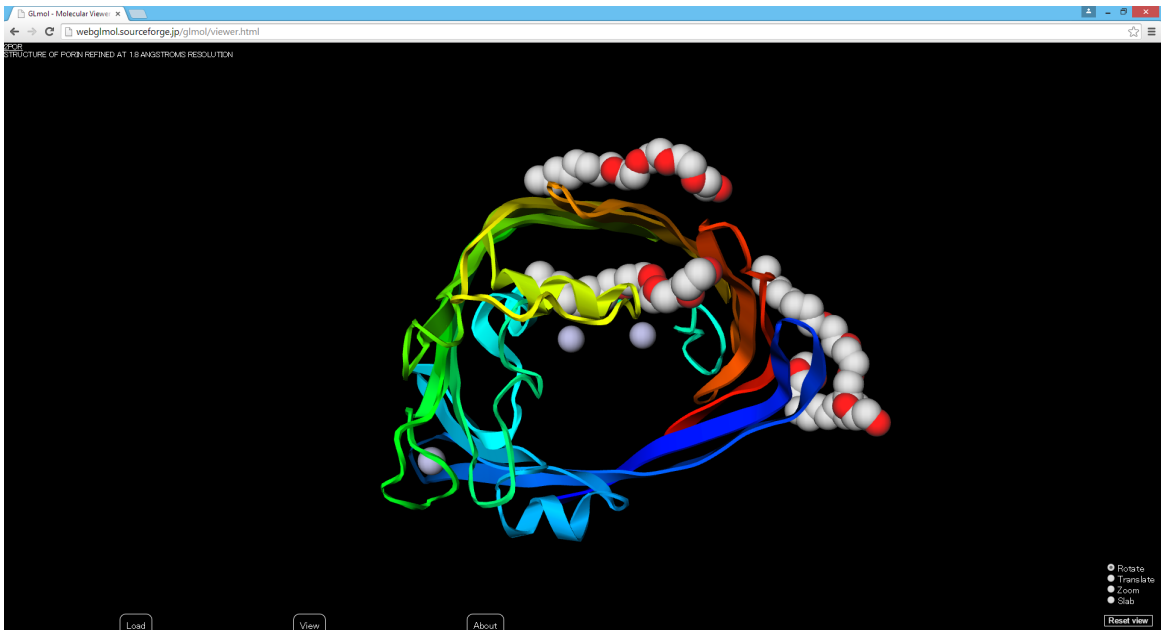

Figure 3.4: The user interface of GLMol in a Chrome web browser, displaying the 3D structure of the Porin protein.

Chapter 4

Software Architecture and Implementation of We3Graph

4.1 Architecture

The architecture of We3Graph as illustrated in figure 4.1 builds upon ideas from Representational State Transfer (REST) web services and Model View Controller (MVC) architecture.

The web service plays a central role in connecting all pieces together and making the application accessible on various devices and operating systems and extendible using various programming languages. Web service benefits are expressed beautifully by Robert Daigneau [25]:

> Web services make it relatively easy to reuse and share common logic with such diverse clients as mobile, desktop and web applications. The broad reach of web services is possible because they rely only on open standards that are ubiquitous, interoperable across different computing platforms, and independent of the underling execution technologies.

In terms of MVC the model is the graph stored either in the database or as an object in the user interface or plugin, the controllers can be user interface and plugins that modify the model using the web service. The views are the web pages opened by different users on different devices. For the communication between the server, user interfaces and plugins, a predefined set of commands is used. Those commands hold information about changes to the graph instead of the state of the graph. This way we can have a history of changes to the graph in addition to the state of the graph. As well, communication resources will be minimized. The list of commands and their description is provided in table 4.1:

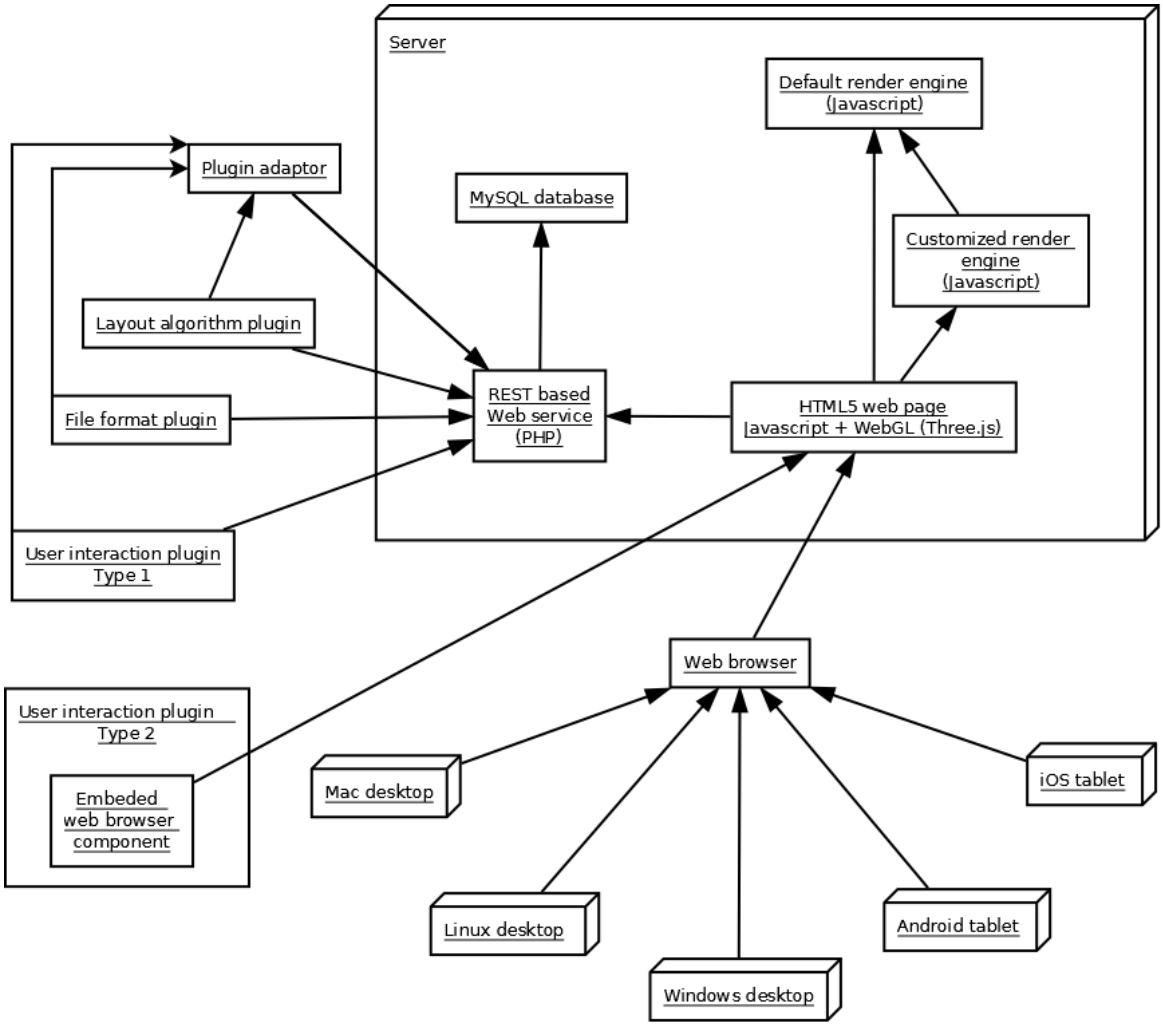

Figure 4.1: Architecture

The plugins can be written in any language as they only need to interact with the web service. To further facilitate this process, plugin adaptor components for C#,C++ and Java can be used to hide the web service interaction from the plugin and provide an object oriented model of the graph as we usually expect.

The plugins can be also written by embedding a web browser component in a native application and sending Javascript commands to the web browser component. This type of plugin is particularly useful in user interaction plugins as the changes can be applied to the user interface before sending them to the server thus reducing the latency that might be critical to user interaction plugins.

Table 4.1: The list of commands used to apply changes to the graph.

Name	Description
InsertVertex	Adds a vertex to the graph.
InsertEdge	Adds an edge to the graph.
BreakEdgeLine	Adds a new bend to break a line segment of an edge into two line segments.
RemoveVertex	Removes a vertex.
RemoveEdge	Removes an edge.
RemoveBend	Remove a bend.
MoveVertex	Moves a vertex to a new position.
ChangeVertexScale	Changes the scale of a vertex.
ChangeVertexRotation	Changes the orientation of a vertex.
MoveBend	Moves a bend to a new position.
ChangeCameraPosition	Move the camera to a new position.
ChangeCameraRotation	Changes the orientation of camera.
SetVertexProperty	Adds, Updates or removes a vertex property.
SetEdgeProperty	Adds, Updates or removes an edge property.
CustomCommand	Used for any command that is needed but not listed above.

Customized render engines for applications of graph drawing can be written in Javascript by inheriting a class from the default engine and only changing the required behaviour. Though Javascript is not a classical object oriented language, object oriented concepts and inheritance can be simulated using object prototypes.

We3Graph uses WebGL as open standard which is accessible on many devices decreasing the need to have different user interfaces on different devices, though it is possible for someone to develop a user interface for We3Graph that does not use WebGL and uses other technologies such as OpenGL and DirectX.

4.1.1 REST

We3Graph borrows many characteristics and benefits of a REST design for web services but it ignores or modifies some of the REST elements or constraints when needed so it can be called a REST based web service instead of the common phrase of a RESTful web service. While REST was first developed in 1994 [41], the term REST was introduced in

year 2000, in the Ph.D. dissertation of Roy Fielding [42]. Here is a description of REST from the Ph.D. dissertation:

> The Representational State Transfer (REST) style is an abstraction of the architectural elements within a distributed hypermedia system. REST ignores the details of component implementation and protocol syntax in order to focus on the roles of components, the constraints upon their interaction with other components, and their interpretation of significant data elements. It encompasses the fundamental constraints upon components, connectors, and data that define the basis of the Web architecture, and thus the essence of its behavior as a network-based application. [42]

While the modern web is one instance of a REST style architecture [41], the architectural style of REST can be also applied to web services. When a web service uses the architectural style defined by REST, it is called a RESTful web service.

From architectural elements of REST, We3Graph uses resources identified by URLS, Javascript Object Notation (JSON) as representation, proper MIME type (application/JSON) for representation metadata and no resource metadata or control data.

From architectural constraints of REST We3Graph implements the uniform interface constraint and the client-server constraint completely, slightly degrades the stateless constraint, practically makes the 'Cache' constraint irrelevant by marking every request as non-cachable, and does not do anything special for the 'Layered system' constraint other than using web which is a layered architecture and ignores the optional 'Code-On-Demand' constraint completely.

For the uniform interface constraint We3Graph uses standard HTTP methods – GET, POST, PUT, DELETE– as actions on resources identified by URLs and proper HTTP status code is returned along with the JSON encoded response.

Being stateless is one of the constraints of REST:

> Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client. This constraint induces the properties of visibility, reliability, and scalability[42].

We3Graph tries to keep the request as stateless as possible with the exception of authentication and authorization information. Having complete authentication and authorization in each request will increase the latency which is critical to this application as described in section 4.3.

We3Graph marks every request as non-cachable due to the nature of the application.

From REST connectors, We3Graph uses client and server and also SSL as a tunnel and DNS as resolver and avoids cache.

4.2 Users and security

In We3Graph each user can be a member of many groups and each folder can be accessible in different ways to different groups using permissions defined by the administrator. Furthermore each graph belongs to a folder but there is no folder hierarchy for simplicity. Also each user can have multiple clients, that is every plugin or browser page of a user is considered a different client but the same user.

Passwords are hashed using the php password_hash function and stored in database. The php password_hash function uses a random salt in combination with hashing which helps protect against lookup table and rainbow table attacks. The php password_hash function has a cost option which can be used to slow down the hashing process to make brute force attacks harder. The default cost(10) of the function is used though it can be adjusted based on the server hardware. All non-local REST requests use the HTTPS (SSL/TLS) protocol for secure end to end transfer of data to prevent wire tapping and man-in-the-middle attacks. Also a minimum length of 10 characters per password is enforced.

4.3 Component interaction example

Here is an example of the interactions between different components of the system after an action by the user such as when a user removes a vertex. The user A decides to remove a vertex by pressing the delete key on the keyboard when the vertex is selected.

The Javascript code running on the browser first changes its internal model of the graph and then tries to send the remove vertex command to the server asynchronously to avoid interruption in the user interface. The rendering loop of the user A which runs regularly reflects the change of the model in the rendered image and the user A sees that the vertex is removed. When the remove command reaches the server, the server first checks if the user has the authentication and authorization to remove the vertex and then adds the remove vertex command to its database. The Javascript code on the User B browser has a regular loop which asks the server for new commands. Since the new command is in database, next time the User B asks for new commands, after the server checks that the user has authentication and authorization to receive the commands, the server will retrn the remove vertex command to the User B browser. The Javascript code on the User B browser changes its internal model of the graph. The rendering loop of the user B reflects the change of the model in the rendered image and the user B sees that the vertex is removed.

4.4 Latency challenge

In We3Graph it is important to render each frame in a reasonable time frame after a change is made otherwise the user experience is significantly affected. If latency goes above some threshold the application may become useless. To be more precise, latency is the time between a change by a user and the time that this change is displayed either in the same user or other users user interface. The importance of latency depends on the type of change. Latency plays a very important role in changes involving user interaction especially when we are using devices such as the LeapMotion and the OculusRift because those devices have their own latency which will be added to We3Graph's latency. There are 4 types of latencies as described in table 4.2 and depicted in figure 4.2:

From a change to render there are several processes which may cause latency as listed below and described individually in the next subsections:

- Database query

Table 4.2: Latency types

Type	Description
1	The latency between rendering and a change made by the same user using the normal web based user interface.
2	The latency between rendering and a change made by another user using the normal web based user interface or another user's plugin.
3	The latency between rendering and a change made by the same user's plugin not using the embedded web browser approach.
4	The latency between rendering and a change made by the same user's plugin using embedded web browser approach.

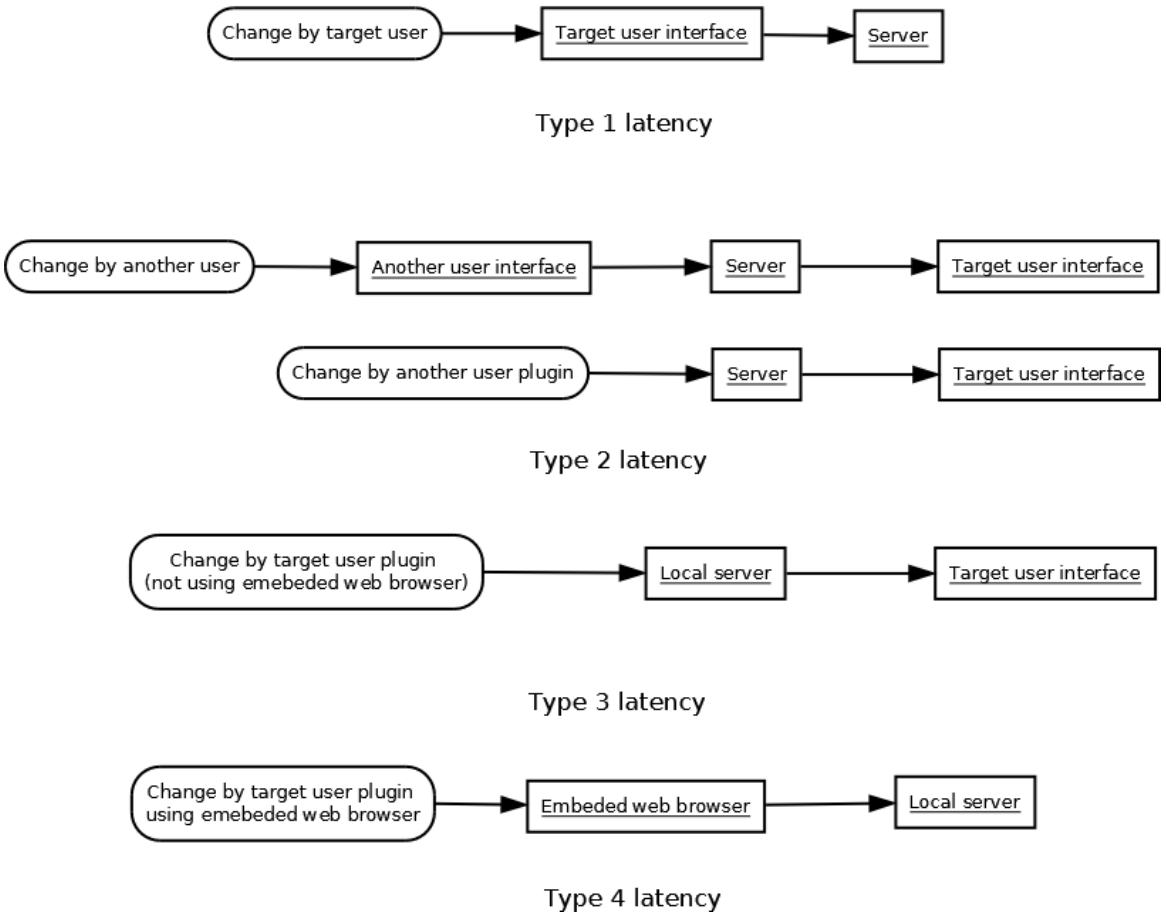

Figure 4.2: Latency types

- Graphical rendering
- Network latency
- Authentication and authorization latency

4.4.1 Database query latency

For type 2 and 3 latencies, each frame needs to query the database to see if there are new commands. To address latency in this part, proper database design and indexes were considered. Also some tables are stored using the MyISAM storage engine while other tables use InnoDB to increase query performance. Also special care was made in the authentication and authorization process to allow faster authentication and authorization on each frame. As the user changes the graph, the number of commands in the database will increase but the effect of many commands such as move commands are overridden by new commands. We3Graph marks these commands as ineffective and with a proper index speeds up the query. Also the ineffective commands can be deleted later so the commands table becomes more compact. The database schema is shown in figure 4.3.

4.4.1.1 MyISAM vs InnoDB storage engines

The MySQL relational database management system (RDMS), has different storage engines. The two most popular ones are MyISAM and InnoDB. Different tables in a database can use different storage engines. In We3Graph, some of the tables use the MyISAM storage engine and some use the InnoDB storage engine.

MyISAM was the default storage engine for MySQL prior to version 5.5 when the default engine switched to InnoDB. MyISAM can be useful in situations where there are many read operations but few write operations. This property was especially useful for the who-tokens, graph-access-tokens and clients table of the We3Graph database. Each client on each frame has to read who-tokens, graph-access-tokens and clients table for the authentication and authorization process. On the other hand each client needs a write operation to those tables only once. This leads to a significant amount of read operations and a much smaller number of write operations.

On the negative side, MyISAM does not support ACID transactions and lacks foreign keys. This leads to weaker data integrity, so care should be taken when using the My-

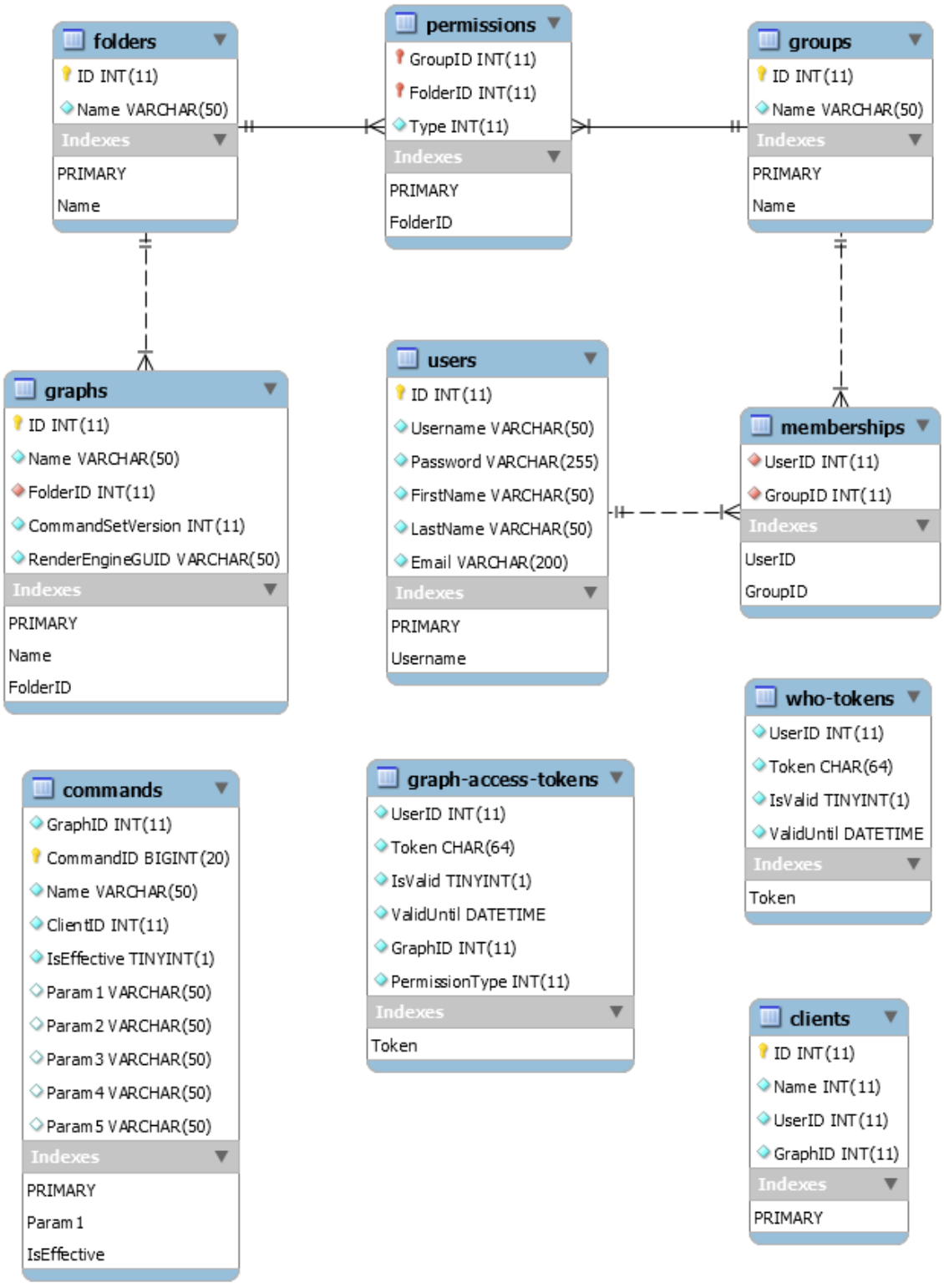

Figure 4.3: Database schema

ISAM engine. InnoDB supports ACID transactions and foreign keys. Except who-tokens, graph-access-tokens and clients tables which required special treatment due to the significant amount of read operations, other tables in We3Graph use InnoDB to benefit from better data integrity and also higher performance when there are many write operations.

4.4.1.2 Indexes

Proper index design is critical to query performance which will lead to lower database latency. To check for new commands, three queries are done on the database. The first query is on the who-tokens table to see if the client is who it claims to be. An index on the Token field of the who-tokens table is defined to facilitate this query. The next query is on graph-access-tokens to see what permission does the client have on this graph. Similar to the who-tokens table, an index on the Token field of the graph-access-tokens table is defined to facilitate this query. The third query is on the commands table to see if they are new commands after the previous command received. To make this query faster, a primary key index is defined on the CommandID field and an index is defined on the IsEffective field for the commands that are overridden by newer commands. To store a new command in addition to the first two queries above, three new queries are needed. The first query is to check if the client number is valid; a primary index on Client ID helps this query. The second query is to store the command in the database. The GraphID field in the commands table would be normally defined as foreign but it is not declared as foreign key to avoid another index in this important table to allow faster write operations. The third query is to mark previous commands as ineffective depending on the command name. An index on the Param1 parameter helps this query as for move commands param1 helps identify which commands to mark as ineffective.

4.4.2 Graphical rendering latency

As the size of the graph increases, graphical rendering on the GPU can be slowed down until becoming useless. This can affect all types of latencies mentioned above. Another

approach that was considered to address this latency was cloud rendering but was ignored after evaluation of positive and negative points. In cloud rendering the 3D scene is rendered in the cloud server and the output of rendering is transferred to the client as a video stream. The client just needs to be able to show the video stream.

On the positive side cloud rendering offers rendering large graphs on devices with low graphical power.

On the negative side, any user interaction that needs a change in the visualization of the model such as a mouse move which may require changing the color of the hovered vertex or bend should be sent to the server before getting visualized. Without cloud rendering it is possible to avoid network latency and, authentication and authorization latency before applying the changes. These latencies become more important for input/output devices such as the LeapMotion and the OculusRift, as these devices themselves have latencies, which will be added to these latencies, and providing a quick visual feedback is very important to the user experience with these devices.

Another limitation of cloud rendering is network speed and resources on both client and server. The client needs to have a high speed network and also the server needs to handle significantly more network traffic. Also the graphical processing power of the server should expand as the number of users expands but expanding server power without cloud rendering is much cheaper and is a much more mature technology. One might propose having both cloud and non-cloud rendering, but the downfall is that the extendibility for customized rendering engines will be significantly degraded as it takes 2 times more time to add a customized rendering engine for a specific application of graph drawing. Also the redundancy of having two different sets of codes for the same custom render engine can cause maintenance issues and may cause different users having different experiences with the software.

In the case that the user does not intend to change the graph, the merged mode can be used which merges geometries of all vertices, bends and edges in order to gain performance.

This feature does not exist for some render engines such as the class diagram engine, which use multiple textures for different faces of the same mesh.

4.4.3 Network latency

Network communication can cause type 2 latency. Also, communication with the local server can cause type 3 latency which might be lower than type 2. To reduce network latency We3Graph only transfers changes to a graph instead of the complete graph state. Only changes made since the last sent or received command will be sent or received over the network.

4.4.4 Authentication and authorization latency

Authentication and authorization can cause type 2 and 3 latencies. Good authentication needs to be slow in order to prevent brute force attacks so if we want to authenticate each user from scratch, this will increase our latency. To reduce this latency we authenticate each client just once and produce a secure who-token which will be used in each frame or request to authenticate the user. Also for authorization a secure graph-access-token will be produced just the first time that the user decides to interact with a special graph and will be used on every frame or request to authorize the user. The tokens are stored in the database and will become invalid after 30 days. The tokens are generated using SHA-256 on a 256 bit CSPRNG generated string (openssl_random_pseudo_bytes or mcrypt_create_iv function of php).

4.5 Coding style

Table 4.3, shows the coding style that is used to code We3Graph in different languages.

4.6 Concurrency and race conditions

When multiple processes access the same resource there is always the chance of a race condition. As in We3Graph multiple users access the same graph model, there is the chance

Table 4.3: Coding style

Element	Style	Example
Class names	pascal case	FirstNext
Public methods	pascal case	FirstNext
Private or protected methods	camel case	firstNext
Internal methods	camel case preceded by double underlines	__firstNext
Local variables	camel case	firstNext
Class member variables	camel case followed by underline	firstNext_
Constants	all letters uppercase and words separated by underline	FIRST_NEXT
Global variables	all letters uppercase and words separated by underline	FIRST_NEXT
Function parameters	camel case	firstNext
Namespaces	pascal case	FirstNext
Packages	pascal case	FirstNext
MySQL table names	all letters lowercase and words separated by dash	first-next
MySQL table fields	pascal case	FirstNext
REST resource names	all letters lowercase and words separated by dash	first-next
File and folder names (Javascript and PHP)	all letters lowercase and words separated by dash	first-next
File and folder names (C#, C++ and Java)	pascal case	FirstNext
Block brackets	brackets start at the next line	
Documentation	JSDoc, PHPDoc, Javadoc, Microsoft XML Documentation for C# and C++	
Abbreviation in names	all letters follow the first letter	getURL for private function or url for local variable
String literals	Use single literals when possible	
Line length	at most 80 but this rule is not strict and can be avoided if it makes code more readable	

of a race condition, such as when two users move the same vertex to two different locations. In We3Graph race conditions on move, scale and rotation commands are considered normal

behaviour and no special action is taken to prevent them. If a race condition is on an action on a vertex,edge or bend that is removed by another user the action will be ignored. For vertex or edge insertion commands each user attaches its client ID to the ID it generates for vertices or edges to avoid having duplicate IDs for vertices or edges. There are still conditions that may lead to unintended or inconsistent behaviour such as when two users try to add the same edge at the same time or when two users try to add a bend to the same edge at the same time.

Chapter 5

Conclusion and Future Work

A new web-based software system for drawing graphs in 3D was presented in this dissertation which has the following features that to the best of the author's knowledge does not exist in previous 3D graph drawing software although some exist in 2D graph drawing software:

- Collaboration: Multiple users can work on the same graph at the same time.
- Sharing with access control: Graphs can be shared among different users and can have different access controls.
- Leap Motion controller and a user interface tuned for touch screens.
- Customizable render engines for different applications of graph drawing: We3Graph introduces the idea that customized render engines can help adapt a general graph drawing software for a specific application of graph drawing. We3Graph also provides examples of this idea such as the custom render engines for class diagrams and chemistry
- REST based API for plugin programming: plugins can be written in any programming language. Also We3Graph introduces the idea of plugin adaptors for hiding REST web service interaction from the developer and provides examples in C#,C++ and Java.
- Detailed 3D class diagram manipulation: Although class diagrams have been visualized in 3D before, We3Graph allows the user to interactively manipulate classes, edge bend points, member of classes, edge labels, edge types and edge arrow symbols.

- Web-based manipulation: Although there has been some effort to use WebGL to visualize graphs in 3D such as GLMol, We3Graph offers manipulation of graphs in 3D.

Here is a list of items that can be considered for future work:

- Extend the render engines for different applications of graph drawing.
- Extend the layout algorithms or incorporate previously written layout algorithms.
- Extend file format plugins (We3Graph has only basic support for MG2 file format). Also it would be nice if the file format plugins can be called from within the web interface.
- Although We3Graph offers a good 3D view, two dimensional views of the same graph such as front,back,left,right,top or bottom views can help the user in perceiving and manipulating the graph.
- Although We3Graph uses the Leap Motion controller, it does not use the structure of the graph or camera to improve the user experience. We3Graph might be able to combine the structure of the graph and camera with Leap Motion data, to guess what the user intends to do.

Also two new algorithms were presented to answer a previously raised question in the 3D graph drawing literature. Although the algorithms run in polynomial time, they can be considered in the slow range of polynomial algorithms, limiting their practical usage especially for dense graphs. Improving performance of these algorithms is an area which can be investigated.

We3Graph is publicly available at http://www.we3graph.com. Also a simple user's manual is provided.

Bibliography

[1] http://www.ogdf.net/doku.php.

[2] http://chemlab.github.io/chemlab/.

[3] http://www.pymol.org/.

[4] http://webglmol.sourceforge.jp/index-en.html.

[5] http://sourceforge.net/projects/dia-installer/.

[6] http://www.gliffy.com/.

[7] http://products.office.com/en-us/visio/.

[8] http://www.sparxsystems.com/products/ea/.

[9] http://www.nomagic.com/products/magicdraw.html.

[10] http://www-03.ibm.com/software/products/en/ratirosefami/.

[11] http://www.visual-paradigm.com/.

[12] David Auber. Tulip — a huge graph visualization framework. In Michael Jünger and Petra Mutzel, editors, *Graph Drawing Software*, Mathematics and Visualization, pages 105–126. Springer Berlin Heidelberg, 2004.

[13] Farshad Barahimi and Stephen Wismath. 3d graph visualization with the oculus rift (poster). In *Graph Drawing 2014, Würzburg*, volume 8871 of *Lecture Notes in Computer Science*, pages 519–520. Springer-Verlag, 2014.

[14] Vladimir Batagelj and Andrej Mrvar. Pajek— analysis and visualization of large networks. In Petra Mutzel, Michael Jünger, and Sebastian Leipert, editors, *Graph Drawing*, volume 2265 of *Lecture Notes in Computer Science*, pages 477–478. Springer Berlin Heidelberg, 2002.

[15] Joel Bennett and Stephen Wismath. 3d graph printing in gluskap (poster). In *Graph Drawing 2013, Bordeaux*, volume 8242 of *Lecture Notes in Computer Science*, pages 514–515. Springer-Verlag, 2013.

[16] Prosenjit Bose, Jurek Czyzowicz, Pat Morin, and David R. Wood. The maximum number of edges in a three-dimensional grid-drawing. *Journal of Graph Algorithms and Applications*, 8(1):21–26, 2004.

[17] John M. Boyer and Wendy J. Myrvold. On the cutting edge: Simplified o(n) planarity by edge addition. *Journal of Graph Algorithms and Applications*, 8(3):241–273, 2004.

[18] Stina Bridgeman and Roberto Tamassia. The graph drawing server. In Petra Mutzel, Michael Jünger, and Sebastian Leipert, editors, *Graph Drawing*, volume 2265 of *Lecture Notes in Computer Science*, pages 448–450. Springer Berlin Heidelberg, 2002.

[19] Stina Bridgeman and Roberto Tamassia. Gds — a graph drawing server on the internet. In Michael Jünger and Petra Mutzel, editors, *Graph Drawing Software*, Mathematics and Visualization, pages 193–213. Springer Berlin Heidelberg, 2004.

[20] Sergio Cabello. Planar embeddability of the vertices of a graph using a fixed point set is np-hard. *Journal of Graph Algorithms and Applications*, 10(2):353–363, 2006.

[21] Tiziana Calamoneri and Andrea Sterbini. 3d straight-line grid drawing of 4-colorable graphs. *Information Processing Letters*, 63(2):97 – 102, 1997.

[22] Marcus Chimani, Carsten Gutwenger, Michael Jünger, Gunnar W. Klau, Karsten Klein, and Petra Mutzel. The Open Graph Drawing Framework (OGDF). In Roberto Tamassia, editor, *Handbook of Graph Drawing and Visualization*. Chapman&Hall/CRC, 2013.

[23] R.F. Cohen, P. Eades, Tao Lin, and F. Ruskey. Three-dimensional graph drawing. *Algorithmica*, 17(2):199–208, 1997.

[24] Kimberley R. Cousins. Computer review of chemdraw ultra 12.0. *Journal of the American Chemical Society*, 133(21):8388–8388, 2011. PMID: 21561109.

[25] Robert Daigneau. *Service Design Patterns: fundamental design solutions for SOAP/WSDL and restful Web Services*. Addison-Wesley, 2012.

[26] Auber David. Tulip. In Petra Mutzel, Michael Jünger, and Sebastian Leipert, editors, *Graph Drawing*, volume 2265 of *Lecture Notes in Computer Science*, pages 435–437. Springer Berlin Heidelberg, 2002.

[27] Hubert de Fraysseix and Patrice Ossona de Mendez. PIGALE. In Roberto Tamassia, editor, *Handbook of Graph Drawing and Visualization*. Chapman&Hall/CRC, 2013.

[28] G. Di Battista. private communication.

[29] G. Di Battista and W. Didimo. GDToolkit. In Roberto Tamassia, editor, *Handbook of Graph Drawing and Visualization*. Chapman&Hall/CRC, 2013.

[30] G. Di Battista, F. Frati, and J. Pach. On the queue number of planar graphs. *SIAM Journal on Computing*, 42(6):2243–2285, 2013.

[31] Vida Dujmović and David Wood. Stacks, queues and tracks: Layouts of graph subdivisions. *Discrete Mathematics & Theoretical Computer Science*, 7(1), 2006.

[32] Tim Dwyer. Three dimensional uml using force directed layout. In *Proceedings of the 2001 Asia-Pacific Symposium on Information Visualisation - Volume 9*, APVis '01, pages 77–85, Darlinghurst, Australia, Australia, 2001. Australian Computer Society, Inc.

[33] Tim Dwyer. Extending the wilmascope 3d graph visualisation system: Software demonstration. In *Proceedings of the 2005 Asia-Pacific Symposium on Information Visualisation - Volume 45*, APVis '05, pages 39–45, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.

[34] Tim Dwyer and Peter Eckersley. Wilmascope — a 3d graph visualization system. In Michael Jünger and Petra Mutzel, editors, *Graph Drawing Software*, Mathematics and Visualization, pages 55–75. Springer Berlin Heidelberg, 2004.

[35] Breanne Dyck, Jill Joevenazzo, Elspeth Nickle, Jon Wilsdon, and Stephen Wismath. Gluskap: Visualization and manipulation of graph drawings in 3-dimensions. In Giuseppe Liotta, editor, *Graph Drawing*, volume 2912 of *Lecture Notes in Computer Science*, pages 496–497. Springer Berlin Heidelberg, 2004.

[36] Breanne Dyck, Jill Joevenazzo, Elspeth Nickle, Jon Wilsdon, and Stephen K Wismath. Drawing kn in three dimensions with two bends per edge. Technical report, Tech. Rep. TR-CS-01-04, Department of Mathematics and Computer Science, University of Lethbridge, 2004.

[37] John Ellson, Emden Gansner, Lefteris Koutsofios, StephenC. North, and Gordon Woodhull. Graphviz— open source graph drawing tools. In Petra Mutzel, Michael Jünger, and Sebastian Leipert, editors, *Graph Drawing*, volume 2265 of *Lecture Notes in Computer Science*, pages 483–484. Springer Berlin Heidelberg, 2002.

[38] John Ellson, EmdenR. Gansner, Eleftherios Koutsofios, StephenC. North, and Gordon Woodhull. Graphviz and dynagraph — static and dynamic graph drawing tools. In Michael Jünger and Petra Mutzel, editors, *Graph Drawing Software*, Mathematics and Visualization, pages 127–148. Springer Berlin Heidelberg, 2004.

[39] Anton J. Enright and Christos A. Ouzounis. Biolayout—an automatic graph layout algorithm for similarity visualization. *Bioinformatics*, 17(9):853–854, 2001.

[40] Stefan Felsner, Giuseppe Liotta, and Stephen Wismath. Straight-line drawings on restricted integer grids in two and three dimensions. *Journal of Graph Algorithms and Applications*, 7(4):363–398, 2003.

[41] Roy T Fielding and Richard N Taylor. Principled design of the modern web architecture. *ACM Transactions on Internet Technology (TOIT)*, 2(2):115–150, 2002.

[42] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

[43] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, 1991.

[44] Ashim Garg and Roberto Tamassia. Giotto3d: A system for visualizing hierarchical structures in 3d. In Stephen North, editor, *Graph Drawing*, volume 1190 of *Lecture Notes in Computer Science*, pages 193–200. Springer Berlin Heidelberg, 1997.

[45] Carsten Gutwenger, Michael Jünger, GunnarW. Klau, Sebastian Leipert, Petra Mutzel, and René Weiskircher. Agd: A library of algorithms for graph drawing. In Petra Mutzel, Michael Jünger, and Sebastian Leipert, editors, *Graph Drawing*, volume 2265 of *Lecture Notes in Computer Science*, pages 473–474. Springer Berlin Heidelberg, 2002.

[46] L. Heath and A. Rosenberg. Laying out graphs using queues. *SIAM Journal on Computing*, 21(5):927–958, 1992.

[47] John Hopcroft and Robert Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, October 1974.

[48] Pourang Irani and Colin Ware. Diagramming information structures using 3d perceptual primitives. *ACM Trans. Comput.-Hum. Interact.*, 10(1):1–19, March 2003.

[49] Michael Jünger, GunnarW. Klau, Petra Mutzel, and René Weiskircher. Agd — a library of algorithms for graph drawing. In Michael Jünger and Petra Mutzel, editors, *Graph Drawing Software*, Mathematics and Visualization, pages 149–172. Springer Berlin Heidelberg, 2004.

[50] Michael Kaufmann and Roland Wiese. Embedding vertices at points: Few bends suffice for planar graphs. *Journal of Graph Algorithms and Applications*, 6(1):115–129, 2002.

[51] Paul McIntosh, Margaret Hamilton, and Ron van Schyndel. X3d-uml: Enabling advanced uml visualisation through x3d. In *Proceedings of the Tenth International Conference on 3D Web Technology*, Web3D '05, pages 135–142, New York, NY, USA, 2005. ACM.

[52] K. Mehlhorn and P. Mutzel. On the embedding phase of the hopcroft and tarjan planarity testing algorithm. *Algorithmica*, 16(2):233–242, 1996.

[53] Henk Meijer and Stephen Wismath. Point-set embedding in three dimensions. In *24th Canadian Conference on Computational Geometry*, pages 223–228, 2012.

[54] Pat Morin and David R. Wood. Three-dimensional 1-bend graph drawings. *Journal of Graph Algorithms and Applications*, 8(3):357–366, 2004.

[55] T. Munzner. Exploring large graphs in 3d hyperbolic space. *Computer Graphics and Applications, IEEE*, 18(4):18–23, Jul 1998.

[56] Tamara Munzner. Drawing large graphs with h3viewer and site manager. In SueH. Whitesides, editor, *Graph Drawing*, volume 1547 of *Lecture Notes in Computer Science*, pages 384–393. Springer Berlin Heidelberg, 1998.

[57] János Pach, Torsten Thiele, and Géza Tóth. Three-dimensional grid drawings of graphs. In Giuseppe Di Battista, editor, *Graph Drawing*, volume 1353 of *Lecture Notes in Computer Science*, pages 47–51. Springer Berlin Heidelberg, 1997.

[58] János Pach and Rephael Wenger. Embedding planar graphs at fixed vertex locations. *Graphs and Combinatorics*, 17(4):717–728, 2001.

[59] Jean-Baptiste Pettit and John Marioni. bioweb3d: an online webgl 3d data visualisation tool. *BMC Bioinformatics*, 14(1):185, 2013.

[60] Wayne Salamonsen, Kevin Yee, Chuen Mok, and S. Subbiah. Biojake: a tool for the creation, visualization and manipulation of metabolic pathways. In *Proc. the Pacific Symposium on Biocomputing '99, 392–400*, pages 392–400, 1999.

[61] Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S Baliga, Jonathan T Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome research*, 13(11):2498–2504, 2003.

[62] Michael E Smoot, Keiichiro Ono, Johannes Ruscheinski, Peng-Liang Wang, and Trey Ideker. Cytoscape 2.8: new features for data integration and network visualization. *Bioinformatics*, 27(3):431–432, 2011.

[63] Uwe Thaden and Friedrich Steimann. Animated uml as a 3d-illustration for teaching oop. In *Proceedings of the 7th Workshop on Pedagogies and Tools for Learning Object-Oriented Concepts in 17th European Conference on Object-Oriented Programming (Darmstadt, Germany)*. Citeseer, 2003.

[64] Colin Ware and Glenn Franck. Evaluating stereo and motion cues for visualizing information nets in three dimensions. *ACM Trans. Graph.*, 15(2):121–140, April 1996.

[65] Colin Ware and Peter Mitchell. Visualizing graphs in three dimensions. *ACM Trans. Appl. Percept.*, 5(1):2:1–2:15, January 2008.

[66] Guohui Zheng, Xiang-Jun Lu, and Wilma K. Olson. Web 3dna—a web server for the analysis, reconstruction, and visualization of three-dimensional nucleic-acid structures. *Nucleic Acids Research*, 37(suppl 2):W240–W246, 2009.