VISUALIZING BIOCHEMICAL NETWORKS WITH NETVIEW

A Thesis presented to the Faculty of the Graduate School
University of Missouri-Columbia

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

---

by
ARCHANA CHIKKABEL

Dr. Toni Kazic, Thesis Supervisor

FALL, 2005

ACKNOWLEDGMENTS

I extend my sincere gratitude and appreciation to people who made this thesis possible. I would like to thank Dr. Mary Polacco and Dr. Chi-Ren Shyu for serving on my thesis committee.

I am highly indebted to Dr. Toni Kazic, my advisor, not only because of the tremendous amount I learned from her but also because of her care and the unbelievable support she provided.

I would also like to extend my gratitude to Nandini Basu, Arpit Ghoting, Geetha Kutikkad, Avanthi Mummaneni, Amith Gosukonda, Gaurav Sanghi, and Raman Seth for their help and support.

This work is supported by a grant to T. K. (GM 56529) from the U. S. National Institutes of Health.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

abbreviation	*full name*
VRML	Virtual Reality Modelling Language
END	Enzyme Nomenclature Database
BND	Biochemical Names Database
UM-BBD	University of Minnesota Biocatalysis and Biodegradation Database

ABSTRACT

Networks are modelled as graphs and are represented visually using graph-drawing algorithms. Visualization of such big networks helps biologists understand them.

In this paper, we present a Java3d-based visualization toolkit called NetView that is designed for the interactive three-dimensional visualization of biochemical networks. The network is first laid out in three dimensions using an algorithm and then rendered onto the screen for visualization and interaction. The system allows the examination of the network and querying the underlying database by each of its components. NetView offers basic navigational features such translation, scaling, and rotation of the network. It also offers features such as animation, traversal of the network in fly, walk, and orbit modes, and storage of snapshots of the network from various angles.

1 Introduction

Reactions among molecules are often represented by a type of mathematical network called biochemical networks [10,19]. Biochemical networks constitute very large, complex systems with very diverse components, rank among the most complex systems we know, and contain some of the best-characterized molecular components. The largest network of experimentally characterized reactions is a union of the enzymatic reactions in two databases, END and UM-BBD [7, 18].

These large networks offer the best means to understand the relationships between network structure and their biochemical and dynamical functions. By structure, we mean both the topological arrangement of the network's components (its architecture) and its qualitative and quantitative properties [19]. By function, we mean both the dynamical and biochemical aspects of function. A better understanding of network structure-function relationships would increase our ability to predictably alter them for specific applications, such as reengineering metabolisms, improving the nutritional yield of crops, predicting drug metabolism, or understanding the effects of a given mutation on a pre-cancerous cell. Whilst characterizing a network's architecture by graph theoretic measures can yield significant insight [19], direct visualization of large networks is an essential step in understanding them since it lets users iteratively explore, query, and interpret them.

One can visualize graphs in either two or three dimensions. Both pose problems to navigation and layout techniques. A large graph makes a normally good two-dimensional layout algorithm completely unusable: the layout is very dense despite algorithms that minimize the crossing of edges; and the high density and resulting occlusion of nodes and edges makes interaction with and navigation of the graphs very difficult. Finally, since rotations are possible only within the plane of the image, two-dimensional layouts preclude the viewer from managing occlusions by looking

"behind" them [13, 23, 25]. In a three-dimensional layout, the extra dimension literally gives more space, easing the problem of displaying large graphs. The edges are less likely to intersect and the extra space reduces occlusions. A dynamic three-dimensional environment lets the user change the view by moving around in space, and apparent edge intersections can be visually minimized with motion parallax shading or stereoscopic disparities to distinguish edges that lie at different depths.

To interactively visualize large networks of biochemical reactions such as those shown in Table 1, we have developed NetView, a Java application that works both locally and over the Internet. It can be used to visualize, navigate, interact with, and query biochemical networks stored as graphs in a database.

1.1 Related Work

Information visualization has gradually emerged over the past fifteen years as a distinct field with its own research agenda. Many visualization tools have been developed for commercial and research purposes [12]. In this section we discuss a few tools that are closely related to NetView with respect to design, layout, and visualization.

1.1.1 Layout

There are two kinds of visualization: those whose geometry is known [17, 27, 29] and those whose geometry is unknown [14]. Biochemical networks have no inherent geometry, so the first step is to assign three-dimensional coordinates to the network's components – that is, to ***lay out*** the graph [9]. The basic graph layout problem can be put simply: given a set of nodes with a set of edges, calculate the positions of the nodes and the edges that connect them. The challenge is to efficiently and predictably produce geometries that display large networks but still remain readable. The main problems are to minimize the number of edges that intersect, so that the

visualization is less compact; to produce the same layout for a graph repeatedly, so that users can more quickly learn to navigate around a network; and to produce the layout quickly. These problems become increasingly difficult as the size of the network and its connectivity increase.

Eades proposed a force-directed layout technique, modelling the nodes as physical bodies and edges as springs [5]. Using Hooke's law, which describes the forces between the bodies, he was able to produce layouts for undirected graphs. Spring layouts have been used successfully to produce well-balanced layout for graphs [11]. But in general, force-directed methods can be rather slow. Each iteration involves a visit to all pairs of nodes in the graph, and the quality of the layout depends on the number of full iterations. Moreover, the layouts are not reproducible: the same graph can produce different layouts each time the algorithm is run, and nearly identical graphs can produce radically different layouts. InterViewer3 is known to generate clear and aesthetically pleasing three dimensional drawings of large-scale networks, and is an order of magnitude faster than other force directed algorithms [11]. Unlike other force directed algorithms, InterViewer3 does not compute the force between every pair of nodes.

The cone tree is one of the best known three-dimensional graph layout techniques in information visualization [24]. A root node is placed at the apex of a cone with its children placed evenly along its base. Multiple cones are grouped into layers, with cones in the lower layers connected to a node in a cone in a layer above. The main problem with this kind of tree layout technique in Euclidean space is that as the number of nodes grow exponentially, the circumference of the circle or the area of the sphere grow only polynomially.

In the original implementation, each layer has cones of the same height, and the cone base diameters for each level are progressively reduced so that the bottom layer

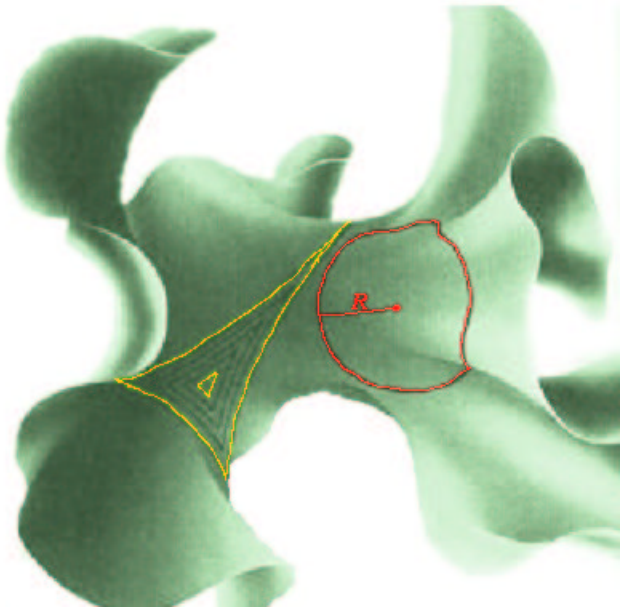


Figure 1: There is literally more room in hyperbolic space than in Euclidean space, as shown in this illustrated embedding of the hyperbolic plane into 3D Euclidean space. The local saddle-like structure leads to an angle sum deficit in each triangle (< 180). This allows many regular tessellations with triangles, shown as a set of triangles on the left. Exponential growth of the circumference and area is experienced if a "circle" with radius R is drawn in the wrinkling structure, as shown on the right. The drawing is from reference 30.

fits into the width of a box containing the full cone tree. To avoid collisions among nodes and crossings of edges, layouts in Euclidean space allocate less volume to nodes further away from the root node. The user can pick any node and rotate the cone tree so that the chosen node is brought to the front.

Munzner proposed laying out large directed graphs as node-edge diagrams in three-dimensional hyperbolic space [21,22]. In hyperbolic space, the circumference and area increase exponentially instead of geometrically (see Figure 1). In her H3 algorithm, she adapted the cone tree layout algorithm to three-dimensional hyperbolic space. In contrast to cone tree layouts, H3 allocates the same volume to every node, no matter how deep in the tree, to avoid collision among the nodes and edge crossings. A

spanning tree is calculated over the graph, and this tree is laid out in hyperbolic space. Successive levels of the tree are placed at increasing radial distance from the centered root node, such that a level's volume is evenly divided among all the nodes of that level. Those nodes and edges not in the spanning tree do not influence the layout, and are connected to the tree at their level after the tree has been positioned. By laying out a spanning tree, rather than the entire graph, H3 can efficiently handle thousands of nodes (up to at least 20,000); and since the layout does not depend on interactions among the nodes, a given graph will always produce the same layout [20, 22]. Since hyperbolic space is an infinite space, in order to inspect the result one needs to project hyperbolic space into a finite volume of Euclidean space. H3 lays out a graph's spanning tree in hyperbolic space, then adds the rest of the nodes and edges to the layout before projecting back into the Euclidean space [20]. The input graph is organized so that nodes of the same path distance to an arbitrary root node are grouped together.

1.1.2 Visualization

A number of tools for visualizing networks have been developed, though nearly all of these are limited to two-dimensional layouts [2, 8, 28]. In most cases, the geometry of the object is fixed, though TouchGraph and Graphlet both permit the user to interactively change the layout [3, 26]. Three dimensional viewers include HypViewer, the companion to Munzner's H3 algorithm; Walrus, a Java3d tool based upon Munzner's hyperbolic layout and navigation techniques; UBViz, a software tool for exploring metabolic pathways in three dimensional space; and several pathway viewers [9, 14, 16].

HypViewer uses ***focus+context*** distortion. In this type of navigation, interacting with the view means changing the position of the center of the Euclidean circle used

to "map" the hyperbolic view onto Euclidean space. This approach allows the user to examine fine details of a small area whilst maintaining a glimpse of the whole graph as a frame of reference. The problem is that since the apparent position of the nodes is always changing, it is very hard for the user to become familiar with the layout. As the network becomes more complicated, this problem is likely to increase. Hence the focus+context technique is not adopted in NetView, and we provided other ways to interact with and navigate around the network.

Unlike UBViz, NetView offers dynamic querying capabilities.

1.2 Design Goals for NetView

The purpose of NetView is to retrieve a biochemical network from a database, lay it out into a three-dimensional model, render that model in three dimensions, visualize and interact with the model, and query the database from which the network is retrieved — all in real time using the same tool. NetView is a standalone application that has been integrated with The *Agora*, an electronic infrastructure for sharing curatorial functions, data, and queries among independent databases. The *Agora* provides the necessary databases and query processing capabilities for the visualization.

Although the motivating data are biochemical reactions, we did not want to build a special purpose tool with limited functions. Therefore our design specifications included goals to make NetView more generally useful. These are enumerated below.

1. ***Have a user friendly interface.*** The main aim here is to make model manipulations as simple and as intuitive as possible whilst reducing the effort and time required on the part of the user to obtain a desired orientation.

2. ***Allow the user to dynamically change the orientation*** (see Item 1) of the model by manipulating the image. The main advantage of three-dimensional

models is that they can be viewed from any arbitrary orientation to gain a better understanding of the shape and spatial relationships.

3. ***Allow the user to navigate through the network.*** In this case, the camera flies through the virtual world. This allows a better appreciation of spatial relationships between the nodes.

4. ***Support animating transitions between different model orientations.*** Viewers have a much easier time retaining their mental model of a network if changes to its structure or its position are shown as smooth transitions as a function of time.

5. ***Allow the user to pick each and every node for further investigation.*** Picking ability was required in order to display the attributes attached to each component and to query The *Agora* for further information about compounds and reactions.

6. ***Allow the user to issue queries to the database*** to retrieve the appropriate network.

7. ***Support bookmarking viewpoints***, *i. e.*, the user can save a particular model orientation and state so as to come back to it later.

8. ***Support extension for more visualizations and functions.*** This implies that the program should be modular, object-oriented, and opensource.

9. ***The system should be integrated with data sources*** so that it can offer efficient data access.

10. ***The tool must be domain independent*** so that we can visualize data from other domains.

11. ***The tool should be deployable across many different platforms*** with minimum effort.

12. ***The tool should support visualization in a browser.*** This enables users working in a standard web browser to visualize and manipulate the biochemical network models.

2 Materials and Methods

2.1 Equipment and Benchmarking

Database services, layout, and VRML file generation were performed on a four-processor Sun E450 with 1 GB of memory running Solaris 8.0. To mimic conditions most users are likely to encounter, rendering and visualization were performed on a Dell Inspiron 600m laptop running Windows XP with 1.4 GHz CPU and 512 MB memory. We benchmarked the layout and VRML file generation and the rendering in two separate steps, since the first occurs on the server and the second on the client.

2.2 Technological Infrastructure

Java, Java3d, C++, and VRML2.0 (Virtual Reality Modelling Language) have been used in our implementation. Java is platform neutral, widely used for distributed graphics applications, robust, and supports applications deployed over heterogeneous network environments. The graphical user interface is built using Java swing classes. The Java3d API provides a set of object-oriented interfaces for applications that require high performance, interactive three-dimensional graphics. It also enables integration with other modules that are not related to the three-dimensional rendering. The Java3d API takes advantage of existing hardware accelerators *via* its use of low level APIs such as OpenGL and Direct3D. This layering allows applications to run on any platform with a Java virtual machine and an OpenGL or a Direct3D implementation. Java3d is based on the scene graph programming model, and VRML is an hierarchical scene graph description language that defines the geometry and behavior of a 3D scene or "world" (see Section 2.4 for a brief description of this visualization paradigm). VRML content can be included in the Java application by using run-time loaders such as Xj3d [15]. The VRML specification is now an Inter-

	END 2.0	UM-BBD	total
total molecules	7578	845	8423
enzymes	3564	427	3971
small molecules	3999	418	4417
reactive conjunctions	3579	779	4358
non-catalytic edges	13,389	1167	14,556

Table 1: Biochemical networks used in this study. END is derived from the *Enzyme Nomenclature*; UM-BBD is the University of Minnesota Biocatalysis/Biodegradation Database. Some molecules are present in both databases. The networks are bipartite in nodes, with one type of node used for molecules and the other for their joining together in reactions ("reactive conjunctions"). The type of a molecule's interaction in a reaction (catalytic or noncatalytic) is denoted by the type of edge between the molecule's node and the reactive conjunction node. Each enzyme is represented only once in these networks.

national Standards Organization (ISO) Specification. VRML and Java3d are openly available and portable to most platforms on the Internet. The library files for the H3 layout algorithm are open source and were implemented in the C++ programming language. END and UM-BBD are databases of enzymatic reactions represented as biochemical networks [7, 18]. Table 1 shows the distribution of nodes, edges, and the size of the largest connected component for these databases.

2.3 Layout Technique

We used a layout algorithm based on Munzner's H3 layout technique for drawing large graphs as node-edge diagrams in 3D hyperbolic space because of speed and reproductibility. H3 only lays out nodes, and depends on HypViewer to supply the connecting edges as lines. Since we didn't use HypViewer for display and lines would have been too faint to distinguish the complex networks, we implemented an algorithm that computes the three dimensional coordinates of the edge that connects each pair of nodes. The algorithm computes the length of the standard cylinder between each pair of connected nodes and translates and rotates the cylinder to that position for

each edge in the network. This allowed us to vary the thickness and the color of the cylinders to denote the different types of edges in the network and to improve the appearance of the visualization.

2.4 Scene Graph Basics

The 3D graphics model can become very complicated, often involving thousands or even millions of polygons that must be collectively transformed, rotated, and altered upon interaction. Therefore, an important modeling issue is how to manage this complexity effectively. One of the principal techniques is to organize the graphical model hierarchically using scene graph modelling.

The VRML file contains the complete description of the 3D scene, listing all the objects that appear within that world. This information is stored in node objects that form a directed acyclic graph. (These objects are distinct from the nodes of the network that is to be rendered.) One node is the root of the scene (not the network), and the other nodes are accessible following edges from the root node. The leaf node object contains the elements that Java3d uses in rendering. The path from the root of the scene graph to a specific scene leaf node is the leaf node's scene graph path, which specifies the state information of its leaf. State information includes the location, orientation, and size of a visual object. The nodes intermediate between root and leaf on the scene graph path store information about properties common to many descendant nodes, such as geometric transformations. There is only one path from the root node to each leaf node. Consequently, the visual attributes (for NetView these are shape, color, and lighting) of each visual object depend only on its scene graph path. The Java3d renderer takes advantage of this fact and renders the leaves in the order it determines to be most efficient.

The virtual universe is a three-dimensional space with an associated set of objects.

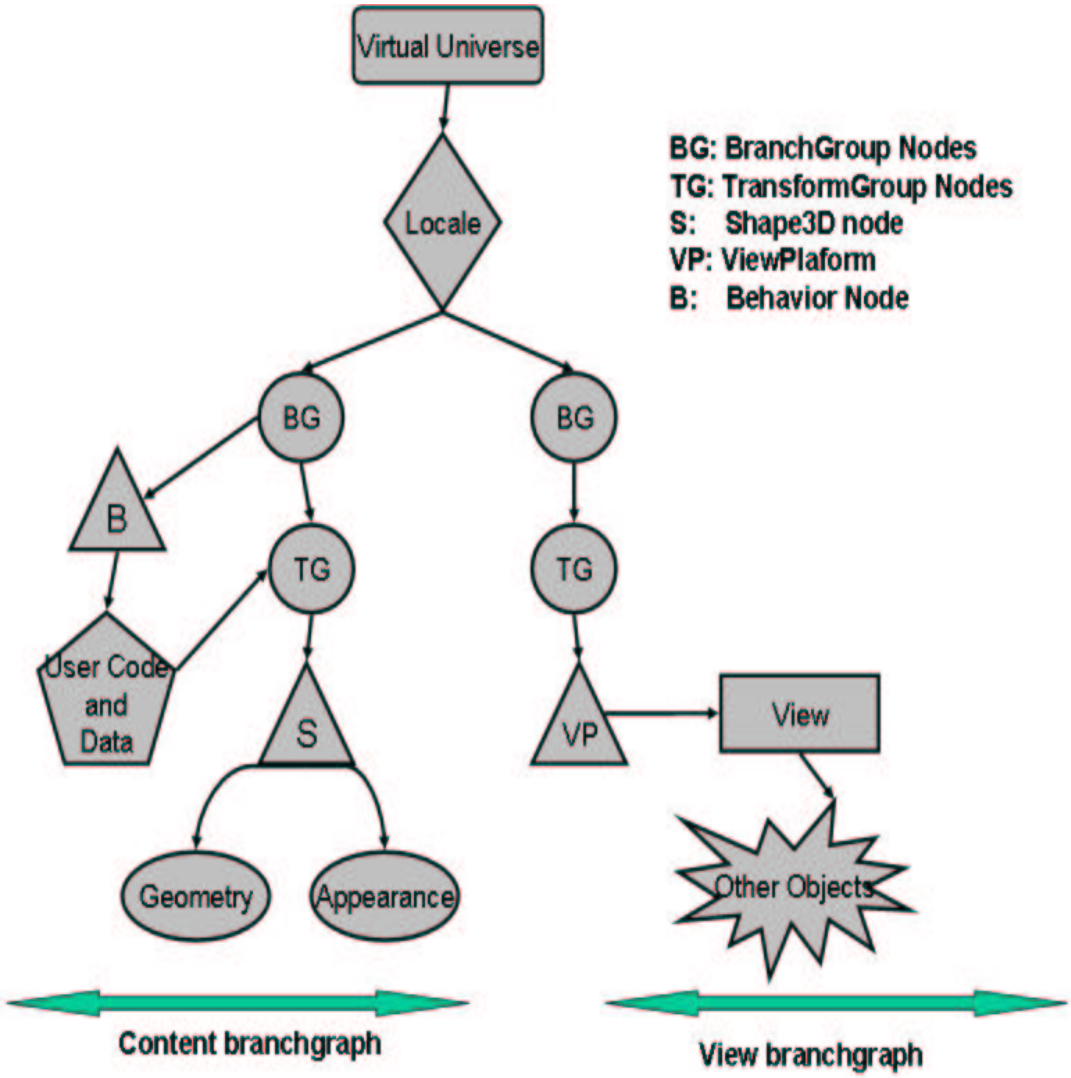


Figure 2: Scene graph implementation. A scene graph provides a high level description of a 3D virtual world, including all the information required to render it. The content branch on the left contains only content-related leaf nodes such as shapes, lights, and behaviors. The View branch graph on the right contains viewing information such as the user's position and orientation in the scene and the specification of the display device used by the viewer.

All Java3d scene graphs must connect to the virtual universe object to be displayed. The virtual universe (or "world") has a universal coordinate system and all other nodes follow that coordinate system. To support large virtual universes, Java3d introduced the concept of locales that have associated high resolution coordinates that serve as the next level of representation. The locale object acts as a container for a collection of subgraphs of the scene graph. The coordinates of all objects attached to a particular locale are all relative to the location of that locale's high resolution coordinates.

As shown in Figure 2, there are two subgraphs rooted at the branchgroup objects attached to the locale. They are the content branch graph (left) and the view branch graph (right). The content branch graph specifies the contents of the virtual universe — geometry, appearance, location, and lights. The objects in the content branch graph can be created either by code to specify geometry or by using loader classes. The view branch graph specifies the viewing parameters such as the viewing location and direction. Together, the two branches specify much of the work the renderer has to do. A TransformGroup (T) is often used in the creation of scene graphics, which holds the object's geometry and its transformation. The scene graph can also contain behavior and interpolator nodes for animation, interaction, and other features. The behavior node object is closely tied to the user code, which tells the TransformGroup how to transform the geometry depending on what the user code wants to do with the geometry (rotate, translate, animate, *etc.*).

In addition to the modelling aspect of the scene graph, it is also used by the Java3D runtime system to organize the processing as the scene is rendered. The content branch is processed first followed by the viewing transformation and then the projection transformation.

3 Results

3.1 System Overview

Figure 3 shows the basic design of NetView. NetView relies on the client-server model to allow for the integration of the visualization with the data sources, distribution of services, interoperability over multiple computational resources, and multiple clients using the services. The client process uses a graphical user interface (GUI) to display and manipulate the visualization and to issue queries to The *Agora*. Several methods for users to control the display are provided. The server integrates multiple data sources using The *Agora*, retrieves the data, lays out the data in three dimensions, transforms them into VRML for use by the visualization clients, and caches and pipelines the cached data back to the client. On the client side, Xj3d reads the VRML file and returns a content scene graph which can be attached to the virtual universe for rendering. The rendered network is changed only in response to a user interaction rather than laying it out each time when the user interacts. This approach is usually faster, since layout is more computationally intensive than rendering, and it allows the user to become familiar with the results of a particular layout algorithm. The graphical information of the network can be stored as a VRML file for later visualization.

3.2 Using NetView

NetView allows biologists to interactively browse the biochemical networks. When it is compiled and run, a graphical user interface is opened (see Figure 4).

Design of the user interface has been critical, as the target audience is people with biology backgrounds who do not have much experience with complicated three-dimensional manipulation controls. Providing good navigational tools is not an easy

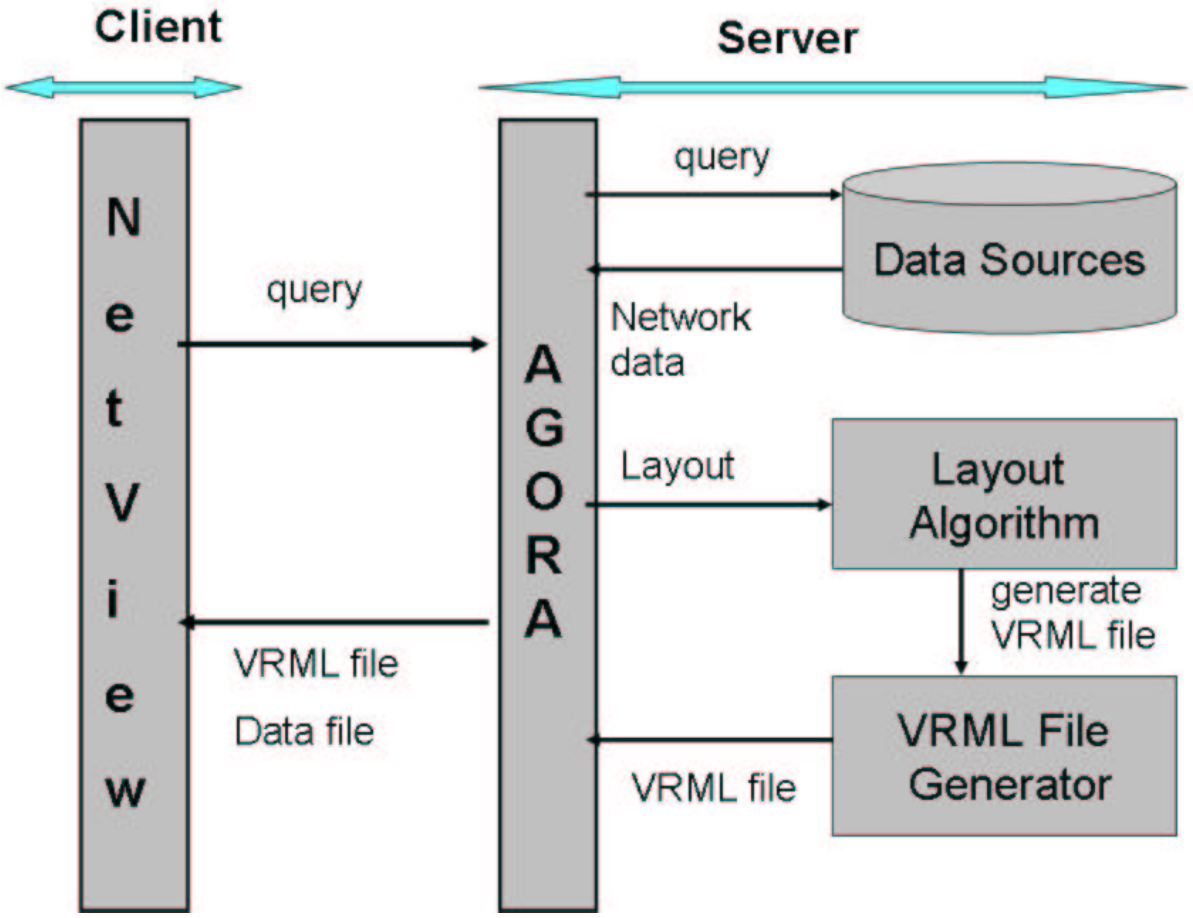


Figure 3: System architecture. NetView issues a query to the participating databases through The *Agora*. The *Agora* retrieves the resulting biochemical network data and passes them to the layout algorithm. This computes the three-dimensional coordinates of the nodes of the network. The orientation of the edges connecting each pair of nodes are computed and a VRML file is generated. The generated VRML file is streamed back to NetView for rendering and display.

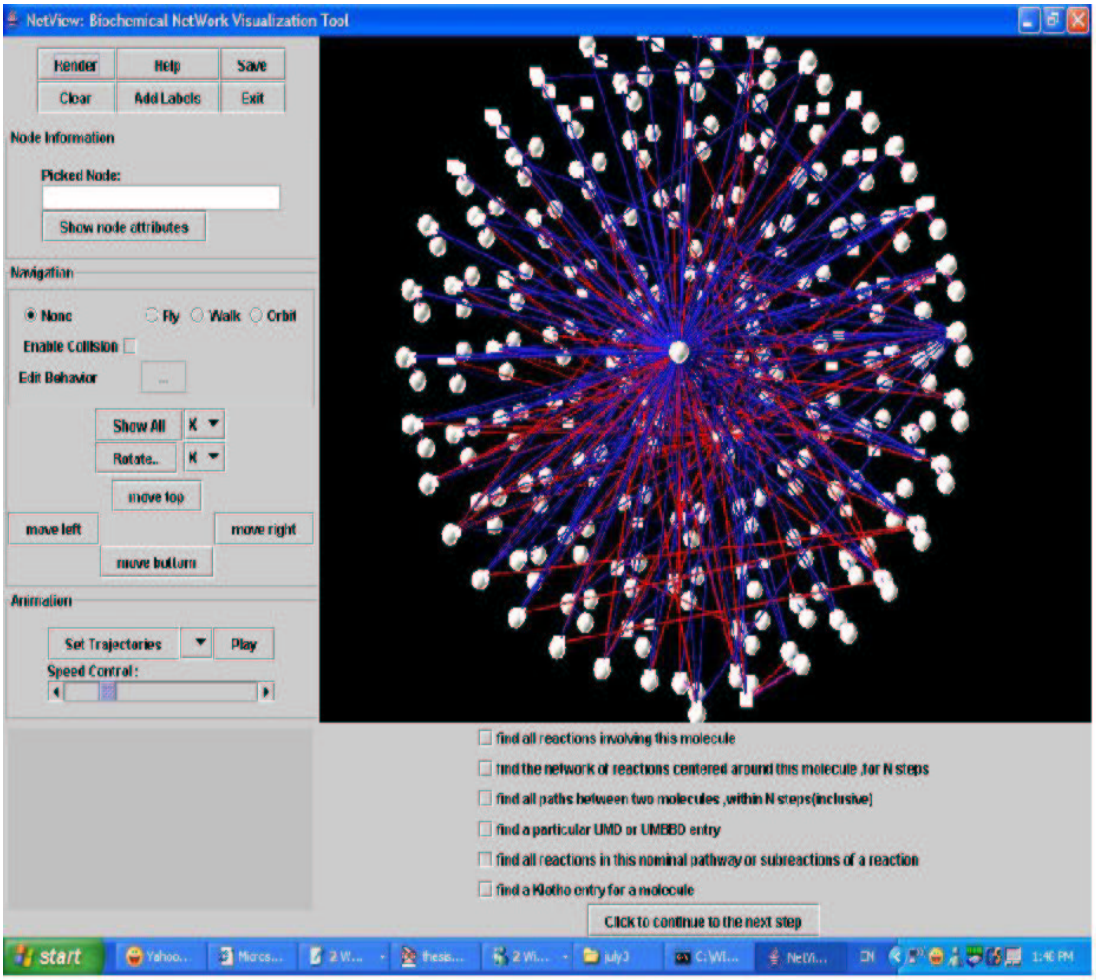


Figure 4: Screen shot of NetView. The main components of NetView are a canvas (large right panel), where the three dimensional model is rendered for visualization; an interaction panel which has different options to interact with the rendered biochemical network (left panel); and a query panel to query the network components (bottom panel). A network centered around pyruvate and expanding one reaction step from it was computed from all the data sources. The network rendered has 375 nodes and 1011 edges.

design task, and ease of manipulation goes a long way toward making the user comfortable in dealing with three-dimensional simulations. The source code of NetView and related software are in the public domain and available [4].

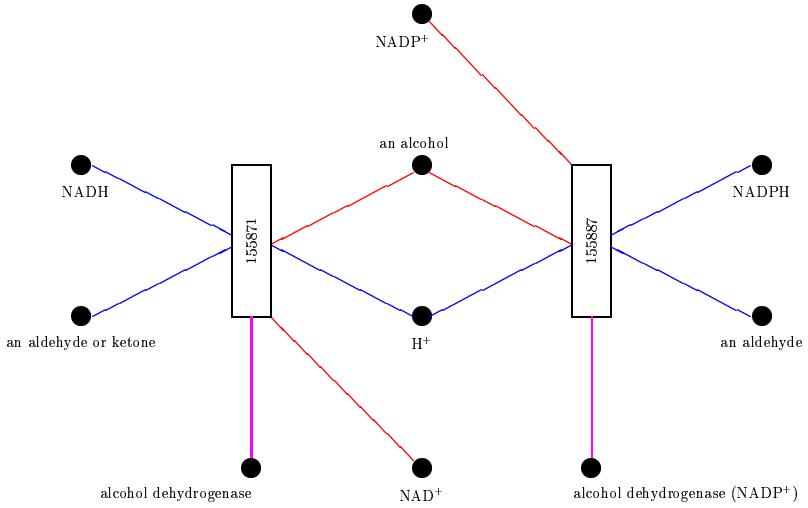
The network has two types of nodes, one for molecules, macromolecular complexes, elements, and ions ("compound nodes"); and the other for the reactive conjunction of the compound nodes. Figure 5 shows a tiny network as reaction equations, rendered in two dimensions, and rendered in three dimensions with NetView. Compound nodes are rendered as filled circles (spheres in NetView) and reactive conjunction nodes as rectangles (rectangular rhomboids in NetView). The network has three types of edges: for reactants appearing on the sinitralateral (nominal left hand) and dextralateral (nominal right hand) sides of the formal reaction equations; and for molecules appearing as classical catalysts in the reaction. In NetView the edges are rendered as cylinders. Sinistralateral, dextralateral, and catalytic edges are colored red, blue, and magenta respectively. Each graphical primitive has a set of attributes attached to it and uses END's distinct accession number for identification. Attributes associated with compounds are name, concentration, state, compartment, and accession number. The reactive conjunctions have EC number, accession number and the source of the database. The edges carry information about their stoichiometry, accession number and the type. More node and edge attributes can be displayed as they are added to the database.

3.3 Implementation Of Navigational Behaviors in NetView

Navigation and interaction are essential for any visualization tool. No layout algorithm alone can overcome the problems large graphs pose for applications. As *per* the specifications, interaction behaviors such as navigation, animation, and picking were created and incorporated into the virtual world. Behaviors provide the means

155871 (EC 1.1.1.1) $\quad$ an alcohol + $NAD^+ \underset{\text{dehydrogenase}}{\overset{\text{alcohol}}{\rightleftharpoons}}$ an aldehyde or ketone + NADH + H^+

155887 (EC 1.1.1.2) $\quad$ an alcohol + $NADP^+ \underset{\text{dehydrogenase } (NADP^+)}{\overset{\text{alcohol}}{\rightleftharpoons}}$ an aldehyde + NADPH + H^+



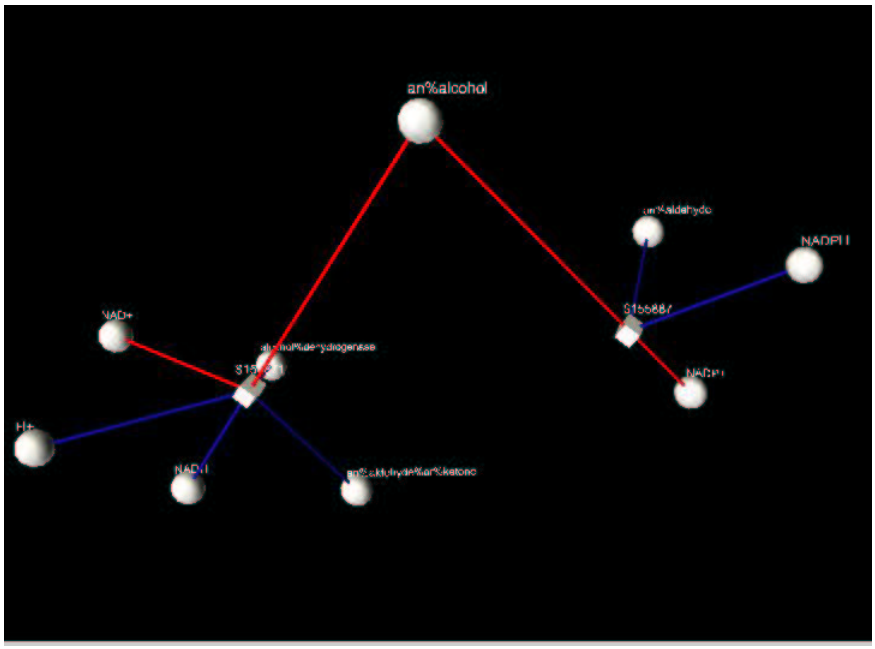


Figure 5: The top panel shows the network as reaction equations, the middle panel the network visualized in two dimensions, and the bottom panel the network visualized with NetView. Circles and rectangles in the two dimensional visualization are translated into spheres and rectangular rhomboids in NetView. For the purpose of illustration, the three dimensional network was manually generated.

Behavior	*Mouse Action*	*Action in Response to Mouse Action*
MouseRotate	Left click and drag	Rotate the object in place
MouseZoom	Middle click and drag	Zoom the object in and out.
MouseTranslate	Right click and drag	Translate the object.

Table 2: Mouse manipulation behaviors. Left mouse click and drag results in rotation of the object. The middle mouse click and drag results in zooming in and out of the object. Right mouse click and drag results in translation.

for animating objects, processing keyboard and mouse inputs, reacting to movement, and enabling and processing pick events.

The default behaviors when the model is loaded are the mouse translate, mouse zoom, and mouse rotate. As shown in Table 2, the user can change the position and orientation of the object in space by mouse click and mouse drag motions. These behaviors are automatically enabled when the fly, walk, and orbit modes are disabled.

Interactive navigation is a behavior that consists of changing either the viewpoint or the position of an object in a scene [23, 26]. Once the system is set up, all the navigation is self-contained. Internally, events are received and the code calculates the new view platform and automatically renders the changed scene on the screen. Table 3 explains the different modes of navigation.

The user interface for moving about relies on click and drag actions with the mouse. The distance between the mouse cursor and the center of the canvas controls the speed of the motion. The further away from the center of the canvas the mouse event is, the faster the speed of motion. The user can change the velocity and angle constants, which lets him or her control the speed of motion. When interacting with the world, implied feedback like collision detection is also incorporated.

Collision detection stops the user from passing through an opaque object in the scene. In the virtual environment, the virtual objects are created and placed in the world. A user may wish to move these objects or alter the scene. The simple action of

Behavior	*Mouse Button*	*Action*
Fly	Left	Clicking up and down results in zoom in/zoom out motion. Clicking left and right yaws the view around the y axis.
	Middle	Clicking up and down pitch's the view around the x axis. Clicking left and right rolls the view around the z axis.
	Right	Clicking up and down moves the view up and down the y axis. Clicking left and right translates the view along x axis.
Walk	Left	Clicking up and down results in zoom in and zoom out motion. Clicking left and right yaws the view around the y axis.
	Right	Clicking up and down moves the view up and down the y axis. Clicking left and right translates the view along the x axis.
Orbit	Right	Clicking up and down rolls the view up and down. Clicking left and right rolls the view left and right.
	Middle	Clicking up and down moves the view closer and farther.

Table 3: Navigational Behaviors. Navigation is when the imagery changes in response to user action. NetView offers three modes of navigation: fly, walk, and examine. Fly mode implements collision detection but has no terrain following. In walk mode, the user to bound to the terrain and has collision detection. Orbit mode does not implement collision detection.

touching or picking an object involves contact with the geometry. To achieve realism for such basic motions, collision detection must be implemented. Collision detection is usually coupled with an appropriate response to the collision. The user needs to know where he or she is in the virtual universe at present, have the movement routine calculate where he or she is going to be next, and then have it calculate the new scene before the movement routine actually commits those changes. A set of six rays (vectors describing the start end of each ray) are cast in each direction to determine if the view platform is in collision with the scene geometry. If there is a collision, the view platform does not change; otherwise, the code updates the view platform and renders the changed scene on the screen.

Picking is a behavior that allows the manipulation of the picked node in the virtual world. NetView allows users to pick on each and every node to display the attributes attached to the picked node and to explore their relationships. Picking an object is a combination of two simple actions: handling a mouse click or movement and mapping two-dimensional screen coordinates into the three-dimensional world. The first part of picking an object is to get the mouse position. The x and y coordinates of the mouse are cast onto the scene as a three-dimensional ray from the user's position. The origin of the ray is set to the position of the user's eyeball in the world space, which is typically the view position; and the direction is set such that the ray projects through the position of the mouse cursor in the view's display plane. Once the pick ray is established, we need to find with what this ray intersects in the scene and return the nearest intersecting object.

In order to create an animation, each "state" of the 3D scene needs to be rendered to create a frame. The playback of these individual renderings at a certain rate creates the illusion of animation. Storing the three-dimensional position of every object in every frame may require a lot of data and the transform matrix is recalculated

when ever the animation is run. One way to reduce the work is to generate key frames less frequently and to allow the program to generate the frames in-between by interpolation [6]. The user can bookmark interesting viewpoints (key frames) in the model and give them names. Animation is performed by interpolating between these key frames using the interpolation algorithm. Interpolators use a clock to time how fast an animation runs; in Java the clock is called an Alpha object. The Alpha object produces a value between zero and one, inclusive, depending on the system time and the parameters of the Alpha object. Key frames are created by storing their rotational, translational, and scale components of the transformation matrix. A ***knot*** is placed between every two frames of the animation and is used to anchor the interpolation between those frames. The knot values map to the Alpha object in Java3d. The interpolator then varies the rotational, translational, and scale components of its target TransformGroup by interpolating among the series of predefined knot/position, knot/orientation and knot/scale pairs, using the value generated by Alpha object.

3.4 Incorporation of Queries

Dynamic querying capabilites are incorporated so that the relationships among any molecules in the displayed biochemical network can be explored. These queries retrieve reactions, networks, and paths from END based on the type of query issued. The information required for a query is collected and the query is initiated by writing the data to The *Agora*'s URL over the network. On the server end a cgi-bin script validates the incoming data stream, queries The *Agora*, retrieves the biochemical network data, and passes them to the layout module. The layout module computes the three-dimensional coordinates and passes the data to a VRML generator; it generates the VRML file and a text file containing the attributes of each of the nodes and

sends the output files back to the NetView client for rendering on the canvas. The queries implemented so far include: find all reactions involving a particular molecule; find the network of reactions centered around a molecule for N steps; find the paths between two molecules within N steps; find all reactions involving some coreactants; find all reactions in a pathway or subreactions of a reaction; find the *Klotho* entry for a molecule.

One of the other useful features implemented in NetView is optional labeling of the compounds and molecules using billboard three-dimensional text. The labels always face the viewer no matter to what orientation the model is rotated.

3.5 Experiments

We have visualized several networks using NetView. The data shown in Table 4 are for related networks of increasing size and complexity. Most of the time needed to return a scene to the user is consumed by the H3 layout algorithm. Rendering very large networks is quite fast once the layout is computed.

set number	*compound*	*data source*	*nodes*	*edges*	*layout time(secs)*	*rendering time(secs)*
1	acetoin	all	10	12	0.48	1
2	glycerol	all	60	73	2.90	2
3	formate	all	130	148	6.06	2
	formate	UM-BBD	11	12	0.23	1
4	pyruvate	all	375	1011	25.58	5
	pyruvate	END	14	13	0.48	1
	pyruvate	UM-BBD	6	5	0.22	1
5	L-arginine	END	14	14	0.49	1

Table 4: Bench marks for retrieval, layout, and rendering. For each network, the central compound, database, and number of nodes and edges are shown. All networks were extended for one reaction step around the central compound. Layout and rendering times are in seconds. Layout was performed on a four-processor Sun E450 with 1 GB of memory running Solaris 8.0 and rendering and visualization were performed on a Dell Inspiron 600m laptop running Windows XP with 1.4 GHz CPU and 512 MB memory.

4 Discussion

We have developed NetView, an advanced biochemical network visualization tool. It offers biologists a cross-platform visualization solution with many interactive features. We have successfully visualized biochemical networks with 375 nodes (274 compounds and 101 reactive conjunctions) and 1011 edges so far. We have implemented and tested techniques for interactively exploring these networks using virtual reality concepts. The architecture of NetView can easily be extended to include more advanced features as well as adapted for other applications.

Java3d is currently an alpha technology and many of its features are still being developed. However, the succesful development of NetView does illustrate the power and flexibility of the current Java3d API.

Effective layout of large networks is still an open problem. H3 produces a more cluttered layout than we would like. Though it is a good layout algorithm in terms of its space and time complexity, back-projection of the hyperbolic geometry into Eu-

clidean space introduces distortion into the model. As a result, edges in the first few layers of the layout are disproportionately longer than the rest. The ***focus+context*** technique minimizes this distortion but only by introducing additional ones. Moreover, H3 wastes much of the volume of the virtual world since it always projects the minimal spanning tree over a hemisphere, rather than the entire sphere. We plan to test other layout algorithms, especially the one used in InterViewer3 [11]. It is an optimized version of a force-directed layout algorithm and has successfully produced well balanced layouts faster of networks with thousands of nodes and edges in tens of seconds. However, it is unclear how reproducible the layout will prove to be from one calculation to another.

There are several enhancements we think users will find valuable. One is to let the user choose layout algorithms. It would also be very useful to select a rectangular area on the screen and maximize it so that the user can have a better look at the nodes deeper in the graph, or to save it separately for further analysis. Subnetworks could be colored or highlighted to indicate topological, structural, or functional similarities, much as different types of amino acids can be highlighted in protein structures. NetView offers many possibilities to display the results of numerical simulations of the dynamics of the network, such as expanding and shrinking the size of the compound and reactive conjunction nodes in proportion to changes in reactant concentrations and reaction rates and fluxes as a function of time. Finally, we wish to make NetView collaborative so that geographically distributed clients can view the same biochemical network interactively at the same time and share ideas.

5 Appendix: User's Guide

This User's Guide describes how to run and use NetView to visualize biochemical networks. The guide documents the user interfaces for displaying, interacting and navigating the biochemical network.

5.1 User Interface Components

The user interface consists of a canvas and a control form. The network loaded is rendered on to the screen. The controls on the form are used to open, save, navigate and interact with the network, control trajectory playback, display and clear the label of each of the components in the network, query The *Agora*, access help, and quit the program.

5.2 Loading a Biochemical Network

There are two ways to view a network on NetView. NetView allows one to open files locally by clicking on the "Render" button from the main control panel. This opens a File Dialog box; one selects the appropriate VRML file to be loaded onto the screen. The other method is to issue a query to the database. The query panel at the bottom has several queries for The *Agora*. The resulting data are rendered in VRML and text format, and are automatically loaded onto the screen. Once the biochemical network is loaded, the user may choose to label all the nodes by clicking the "label" button.

5.3 Using the Mouse in the Graphics Window

Once the network is loaded, a mouse is used to interact with the network, either on the display window or on the control panel forms.

5.3.1 Mouse Controls

There are three basic mouse modes: Rotation, translation, and zooming.

- The left mouse button is used for rotation.
- The middle mouse button is used for translation in the XY plane. As the mouse is moved up and down the view is moved to the right and left, respectively.
- The right mouse button is used for translation along the z axis, zooming in and out as one moves the mouse to the right and left, respectively.

5.3.2 Navigation

There are three modes of navigation: fly, orbit, and walk. All controls are provided by the mouse. All these behaviors are represented by radio buttons in the interface.

Fly This behavior allows the user to move to any point with any orientation in 3D space. Each mouse button generates a different type of motion when the button is pressed. The distance of the mouse cursor to the center of the canvas controls the speed of motion.

- Left Mouse Button – Moving the mouse up and down moves the view forward(zoom in) and backward(zoom out). Moving the mouse left and right yaws the view around the y axis.
- Middle Mouse Button – Moving the mouse up and down pitches the view around the x axis. Moving the mouse left and right rolls the view around the z axis.
- Right Mouse Button – Moving the mouse up and down moves the view up and down the y axis. Moving the mouse left and right translates the view along the x axis.

Walk This behavior will allow the user to move to any point in 3 space whilst following the terrain.

- Left Mouse Button – Moving the mouse up and down moves the view forward(zoom in) and backward(zoom out). Moving the mouse left and right yaws the view around the y axis.
- Right Mouse Button – Moving the mouse up and down moves the view up and down the y axis. Moving the mouse left and right translates the view along the x axis.

Orbit This behavior lets the user to move the view around a point of interest when the mouse is dragged with the right mouse button pressed.

None Choosing this behavior disables the fly, walk, and orbit behaviors.

5.3.3 Animation Controls

NetView facilitates the creation of animations. The desired frames are stored and then played back.

- Move the network to the desired viewpoint and click the set trajectory button. A dialog box opens: give an arbitrary name to the view point.
- Store any number of such viewpoints as above.
- Click the animate button, which plays back these viewpoints.
- The speed of the animation is controlled using the slider bar.
- The stored viewpoints can be deleted individually.

5.3.4 Query Controls

- Select the particular query and click the "OK" button.
- Either enter the name of the compound in the text area or click the compound in the network (the name pops up in the compound text area).
- Enter all the other relevant information and click the query button.
- The query results are piped back to the application and loaded onto the screen.
- The user can store the retrieved network on the local disk by clicking the "Save" button.

5.3.5 Picking Controls

- Click the node you want to study. The name of the node appears on the text box in the control panel.
- Click the "Node Information" button. The attributes of the selected node are displayed in a table.

6 Appendix: PseudoCode

6.1 Behaviors

Picking procedure

- Set the target transformGroup on which the bahavior acts.
- Upon awakening by the left mouse button down event:
  - Update a switch node to draw a ray from the center of the screen.

  - Change that ray's TransformGroup node so that the ray points in the direction of the current mouse position.
  - Declare interest in the mouse-move or the left-mouse-button-up event.

- Upon awakening from the mouse move event:
  - Change that ray's TransformGroup node so that the ray points in the direction of the current mouse position and declares its interest in the mouse-move or left-mouse-button-up event.

- Upon awakening from the left-mouse-button-up event:
  - Change that ray's TransformGroup node so that the ray points in the direction of the current mouse postion.
  - Intersect the ray with all the objects in the universe to find the first object that the ray intersects.
  - Return the user data of the nearest object that the ray intersects with.
  - Declare interest in the left-mouse-button down events.

- To determine the direction of the ray from the center of the screen:
  - Establish where the mouse is in 3D space and the user is in 2D space.
  - Convert both the eye and the mouse position into real virtual world coordinates.
  - eye position - mouse position = direction of pick ray.

Animation procedure

- Create the target object with the ALLOW_TRANSFORM_WRITE capability set.

- Create the Alpha Object and specify time parameters .
- Create the interpolator object and have it reference the Alpha and TransformGroup objects.
- Set the scheduling region for the behavior.
- Make the behavior the child of TransformGroup.

Fly Behavior procedure

- Set the target TransformGroup on which the behavior acts.
- For every mouse event, the new view platform is calculated and the changed scene is rendered on the screen.
- Compute the yaw angle and the translation along the z axis and integrate it with the target transform for every left mouse click.
- Compute the pitch and the roll angles and integrate with the target transform for every middle mouse click.
- Compute the translation along the x and y axis and integrate with the target transform for every right mouse click.

Walk Behavior procedure

- Set the target TransformGroup on which the behavior acts.
- Compute the yaw angle and translation along the z-axis for the left mouse event.
- Compute the translation along the x and y axis for the right mouse event.
- Intergrate the two translations with the target transform to render the scene.

Collision Detection procedure

- For every mouse event registered, compute the new target transform based on the behavior set.
- Cast a set of six rays (vectors) in each direction to determine if the view platform is in collision with the scene geometry.
- If there is collision, the view platform will not change.
- Else, the new computed view platform is rendered on the screen.

6.2 Edge Layout Algorithm

- Extract the three dimensional coordinates of compounds and the conjunctive nodes of the network computed by the H3 layout algorithm.
- Compute the three dimentional coordinates for all the edges that connect the nodes and place cylinders with appropriate thickness.
  - The length of the cylinder is the distance between the two nodes.
  - The midpoint between the two nodes is the translation point.
  - Rotate the cylinder from its initial orientation along the y axis to the vector from the parent node to the child node.
  - Set the color and the thickness of the cylinder.
- Generate a VRML file representing the 3D information of the nodes and the edges which are represented as cylinders.

7 Appendix: Geometric Transformations

7.1 Vectors and Points

The standard way of representing points and vectors in computer graphics is by using homogeneous coordinates. These coordinates represent three-dimensional points and vectors as subspaces of a four-dimensional vector. The fourth coordinate, usually called w, is set to the value 1 for points and the value 0 for vectors. Thus, points form an affine 3D subspace of the 4D homogeneous space, whilst vectors form a vector subspace.

7.2 Homogeneous coordinates

The real power of homogeneous coordinates comes when transforming coordinate systems, which is done frequently in computer graphics. Transformation is a way of moving a group of points (or vectors) that describe one or more geometric objects within a frame to new positions. These transformations usually involve both a translation and rotation, as well as scaling and shearing, and are called affine transformations. Most of the transformations that one needs in computer graphics are affine. Rotation and translation are affine rigid body transformations in the sense that no combination of translations and rotations may alter the shape of an object, only its location and orientation. In contrast, scaling is an affine non-rigid body transformation. By using 4×4 homogeneous transform matrices, it is possible to represent all of these operations in a single matrix. A point in the local coordinate system can be transformed into a global one by post-multiplying it with the local to global transformation matrix:

$$p_{\text{global}} = pM_{\text{transform}}. \tag{1}$$

For an affine transformation, this can be expressed as:

$$p_{\text{global}} = \begin{bmatrix} p_1 & p_2 & p_3 & 1 \end{bmatrix} \begin{bmatrix} M_{11} & M_{12} & M_{13} & 0 \\ M_{21} & M_{22} & M_{23} & 0 \\ M_{31} & M_{32} & M_{33} & 0 \\ t_1 & t_2 & t_3 & 1 \end{bmatrix} \tag{2}$$

where t_1, t_2, t_3 are the translation components of the affine transformation. The 3×3 submatrix $M_{i,j}$ represents the concatenation of a scaling, shear, and rotation operation. It is a pure rotation for that subset of affine transformations known as rigid body transformations.

7.3 Transformation Hierarchies

3D graphic models can become very complicated, often involving thousands of polygons. To manage such complex systems effecively, they are organized into hierarchies. The most important use of such hierarchical representations is to organize transformations between coordinate systems. This type of transformational hierarchy can be described as an n-ary tree of nodes with a homogeneous transform matrix at each node. The matrix represents the node's local coordinate transformation with respect to its parent, that is, multiplying a point by this matrix will transform it from local coordinates into the coordinate system of its parent. Every node can therefore store its geometric information in its own local coordinate system, which does not change even if the node is moved to a different location in the hierarchy. The global coordinates of the geometric information can then be calculated by walking up the hierarchy to the root node, concatenating each local matrix to form the final local-to-global transformation matrix. Multiplying by this matrix will then transform a local coordinate point into a global coordinate one. One way to visualize this process

is to draw a graph with nodes representing the coordinates and the edges representing the transformations. Moving along an edge is equivalent to multiplying by that edge's transformation matrix and moving backwards along the edge is equivalent to multiplying it by the inverse transformation. For interactive 3D graphic systems, this allows one to change transformations at any point in the hierarchy and also to be able to reposition all of a node's desendents at the same time.

7.4 Rigid Body Motion

An animated sequence can be generated from a 3D model by rendering an image, making a small movement in the 3D model, rendering the next image, and so on. Since the position and orientation of three-dimensional objects are usually specified using 4×4 homogeneous transform matrices, the simplest way to animate them would be to make the components of the matrix a function of time or of some user input.

The position of the rigid body can easily be controlled by controlling the elements of the transform matrix, as the translational components of the matrix are independent of the other components and also of each other. However, for rotational components it's not quite so simple. The nine upper-left components of the transformation matrix are interdependent, as they contain the scaling and shear components, which are non-rigid body transformations. Therefore, it is necessary to parametrize rotations in a more fundamental way so that they are independent of each other. The most traditional way to do this is by use of Euler angles. Euler proved that any orientation in space could be represented by a single rotation about an arbitrary axis and this rotation is equivalent to three successive rotations about the x, y and z axes. An arbitrary rotation in space is therefore parametrized by these three angles.

This parameterization of the rotation space has at least two problems. The first, known as gimbal lock, is caused by x and the z axes being in the same order after

rotating the y axis $\pm 90°$. In this situation, a degree of freedom is lost and not all rotations can be represented. The second problem concerns interpolation, which is an important property for motion control. Since the three Euler angles are not independent parameters in rotation space, there can be more than one way to represent a single orientation and more than one way to interpolate between successive orientations. Interpolating motions between two successive orientations by simply linearly interpolating their Euler angles, as though they were Cartesian coordinates, will generally result in a contorted motion and not a smooth rotation about a single axis one would expect from Euler's theorem. Further more, this motion will depend on the orientation of the coordinate system.

The first practical solution to this problem was offered by Ken Shoemake, who represented rotation using unit quaternions [6]. A quaternion is an extension of a complex number. It has one real dimension and three imaginary dimensions. Since quaternions form a four dimensional space, unit quaternions lie on a three-dimensional surface of a unit hypersphere that can represent the space of possible 3D rotations. It can be shown that multiplication of unit quaternions corresponds exactly to rotations in three dimensional space. Unlike rotation matrices, the four quaternions components can be interpolated, resulting in a smooth interpolation from one orientation to the next.

Hence, for rigid body motion, the three Cartesian coordinates for the position, and the four quaternion coordinates for orientation, form the appropriate parameter set. Once we have the set of parameters with which to control the rigid body motion, it becomes important to specify exactly how these parameters change over time. The transformation between any two key frames can easily be obtained by linear or spline interpolation. In linear interpolation, the difference between the adjacent key parameters is divided by the number of intervening frames to determine the parameter

increment for each frame. A linear interpolation algorithm produces undesirable effects such as a lack of smoothness in motion, discontinuities in the speed of motion, and distortions in rotations. Instead, high level interpolations such as cubic or spline interpolation are used. A good method is the Kochanek Bartels spline interpolation because it allows the curve to be controlled at each point by three paremeters: tension, continuity, and bias. The tension parameter controls how sharply the curve bends at a key point p_i. Continuity controls the continuity of the spline at p_i. Bias controls the direction of the curve as it passes through p_i. A time value is added to each control point to control the motion. The method is valid for interpolation between scalar values like angles and vector values like positions.

BIBLIOGRAPHY

1. 1997. *Proceedings of the 1997 IEEE Symposium on Information Visualization.* IEEE Computer Society Press.

2. Bourne, P. E., Gribskov, M., Johnson, G., Moreland, J., and H. Weissig, 1998. A prototype Molecular Interactive Collaborative Environment (MICE). In Altman, R. B., Dunker, A. K., Hunter, L., and T. E. Klein, eds., *Pacific Symposium on Biocomputing '98.* World Scientific Publishing Co., Singapore.

3. BRAINSYS Informatic Systems, 1999. *Graphlet.* University of Passau, Passau, Germany, `http://www.brainsys.de./products./Graphlet./graphlet.html`.

4. Chikkabel, A. and T. Kazic, 2004–present. *NetView.* University of Missouri, Columbia, `https://www.the-agora.org/netview/`.

5. Eades, P. and M. L. Huang, 2000. Navigating clustered graphs using force directed methods. In *Force Directed Methods*, volume Vol 4 no.3, pages 157–181. `http://-citeseer.ist.psu.edu/eades00nagigating.html`.

6. Eberly, D., 1999. *Kochanek Bartel's Cubic Splines (KCB).* BK&CD, `http://-www.magic-software.com/Documentation/KBSplines`.

7. Ellis, L. B. M., Hershberger, C. D., and L. P. Wackett, 2000. The University of Minnesota Biocatalysis/Biodegradation Database: microorganisms, genomics, and prediction. *Nucleic Acids Res.* **28**:377–379.

8. Ellson, J. and S. North, 2004-present. *GraphViz.* AT&T Research, `http://-www.graphviz.org/About.php`.

9. Enright, A. J. and C. A. Ouzounis, 2001. Bio(L)ayout – an automatic graph layout algorithm for similarity visualization. *Bioinfo* **Volume 17**:853–854.

10. Gomez, S. M., Noble, W. S., and A. Rzhetsky, 2003. Learning to predict protein-protein interactions from protein sequences. In *Georgia Tech International Conference on Bioinformatics*, volume 19, pages 1875–1881.

11. Han, K. and B.-H. Ju, 2003. A fast layout algorithm for protein interaction networks. *Bioinformatics* **Vol 19 no.15**:1882–1888.

12. Herman, I., Melancon, G., and M. S. Marshall, 2000. *Graph Visualization and Navigation in Information Visualization: a Survey.* Center for Mathematics and Computer Sciences, Amsterdam.

13. Hsu, J.-W., Huang, T.-W., Chang, Y.-L., and C.-Y. Kao, 1999. Interaction viewer: A visualization tool for protein-protein interaction networks. In *Genome Informatics Workshop, 1999.* `http://www.jsbi.org/journal/GIW04/GIW04P134.pdf`.

14. Hyun, Y., 1999. *Walrus.* Caida.org, `http://www.caida.org/tools/Visualization/walrus`.

15. Java3D and V. TaskGroup, 1998-present. *Xj3d Loader Interface.* Java3D and VRML TaskGroup, `http://www.web3d.org/TaskGroups/source/xj3d.html`.

16. Li, S. and H.-H. Chou, 2005. UBViz:a software tool for exploring metabolic pathways in 3-D space. *Biotechniques* **38,Number 4**:540–542.

17. Moreland, J., 1999. *Molecular Interactive Collaborative Environment.* SDSC and NAPACI, `http://mice.sdsc.edu`.

18. Mummaneni, A., Boyce, S., Bugrim, A., McDonald, A., Slomczynski, J., Fabrizio, F., Sulaman, W., Akunuri, B., Wise, W. B., Tipton, K., and T. Kazic, 2000–present. END*: Enzyme Nomenclature Database*. University of Missouri, Columbia, `http://www.biocheminfo.org/end`.

19. Mummaneni, A. and T. Kazic, 2002. The architectural dynamics of cellular biochemistry and molecular biology. In Callaos, N., ed., *Proceedings of the Sixth World Multiconference on Systemics, Cybernetics and Informatics (SCI 2002)*. Also at `http://www.biocheminfo.org/repository/sci2002.ps`.

20. Munzner, T., 1997. H3: laying out large directed graphs in 3D hyperbolic space. In graph layout and drawing (but not rendering) [1], pages 2–10. Also at `http://-graphics.stanford.edu/papers/h3`.

21. Munzner, T., 1998. Drawing large graphs with H3Viewer and Site Manager (system demonstration). In *GD '98: Symposium on Graph Drawing*, number 1547 in *Lecture Notes in Computer Science*, pages 384–393. Springer Verlag, Berlin. Also at `http://graphics.stanford.edu/papers/h3draw`.

22. Munzner, T. and P. Burchard, 1997. Visualizing the structure of the World Wide Web in 3D hyperbolic space. In *Proceedings of the 1997 IEEE Symposium on Information Visualization* [1], pages 33–39.

23. Parker, G., Franck, G., and C. Ware, 1998. *Visualization of large nested graphs in 3D: navigation and interaction*. J. Vis. Lang. Comp., `http://www.ccom.unh.-edu/vislab/PDFs/visualnav.p`.

24. Robertson, G. G., MacKinlay, J. D., and S. K. Card, 1991. Cone trees: animated 3D visualization of hierarchical information. *ACM conference on Human Factors in Computing Systems* pages 189–194.

25. Salamonsen, W., 1999. *BioJake: Tool for creation, visualization and manipulation of metabolic pathways.* Pac Symp Biocomput, `http://www.smi.stanford.edu/-projects/helix/psb99/Salmonsen.pdf`.

26. Shapiro, A., 2003–present. *Touch Graph.* SourceForge.net, `http://-touchgraph.sourceforge.net`.

27. Suhnel, J., 1997. Virtual Reality Modeling for Structural Biology. *Bioinformatics* **Volume 12**:189–198.

28. Theoretical Biophysics Group, 2000– present. *VMD: Visual Molecular Dynamics.* Beckman institute, University of illinois at Urbana-Champaign, `http://www.-ks.uiue.edu/Research/vmd/`.

29. Vollhardt, H. and J. Brickmann, 1998. *Virtual Reality Modeling Language in Chemistry.* Chemistry in Britain, `http://www.pc.chemie.tudarmstadt.de/-research/vrml/psb95/index_right.shtml`.

30. Walter, J., Ontrup, J., Wessling, D., and H. Ritter, 2003. Interactive visualization and navigation using the hyperbolic space. In *IEEE Int. Conf. Data Mining (ICDM'03)*, pages 355–362. `http://www.techfak.uni.bielefeld.de/-walter/h2vis/`.