Kinematic Modeling with Constraints

by

Sameer Raheja

B.S., Mechanical Engineering (1996)
Massachusetts Institute of Technology

Submitted to the Department of Civil and Environmental Engineering in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

January 1998

© Massachusetts Institute of Technology 1998. All rights reserved.

Author............................................

Department of Civil and Environmental Engineering

January 15, 1998

Certified by...........

John R. Williams
Associate Professor
Thesis Supervisor

Accepted by ..................

Joseph Sussman
Chairman, Department Committee on Graduate Students
Kinematic Modeling with Constraints

by

Sameer Raheja

Submitted to the Department of Civil and Environmental Engineering
on January 15, 1998, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

A system for modeling kinematics using constraints is presented. The use of constraints in kinematic motion is described and compared to other methods of simulation, along with a simulation written in Java with a VRML interface. The user may constrain the elements in any manner and then simulate the motion of the system. The system motion is resolved by constraints on each of the elements. Constraints offer the advantage of simplicity and computational speed, and are suitable for a finite number of elements. Kinematic simulation allows designers to build and test systems computationally, alleviating the need for physical prototyping.

Thesis Supervisor: John R. Williams
Title: Associate Professor
Acknowledgments

I would like to thank the many people who have supported me through my time as a graduate student at MIT. First, Professor John Williams, who has enthusiastically supported me from the beginning. I could not have done it without him.

Jen-Diann Chiou, who passed on much of his knowledge to me and kept our computers running, I will miss our late evening discussions. Petros Komodromos, who was always willing to lend a hand. Abel Sanchez, who took over many of my responsibilities as I got more involved in this thesis. Joan McCusker, who keeps things in order for IESL.

Satayan Mahajan, Geeta Sankappanavar, and all my friends who helped me enjoy MIT.

Rajen, who is always willing to listen. And finally, my parents, without whom I could never have accomplished as much as I have.
Contents

1 Introduction ........................................ 9
  1.1 Kinematics ...................................... 9
  1.2 Kinematic simulation ............................ 10
  1.3 The Program Code ............................... 11
  1.4 What follows .................................. 11

2 Background ....................................... 13
  2.1 Kinematic Modeling ............................. 13
     2.1.1 Lower kinematic pairs ..................... 14
     2.1.2 Animating the model ...................... 18
  2.2 VRML as a Graphical Tool ..................... 19
     2.2.1 A brief history of VRML .................. 19
     2.2.2 Anatomy of VRML .......................... 20
     2.2.3 Use of VRML ............................... 21

3 System Design .................................... 24
  3.1 Constraints .................................... 24
     3.1.1 Penalty functions ......................... 25
     3.1.2 Force constraints ......................... 26
     3.1.3 Lookup tables ............................. 27
     3.1.4 Path of travel constraints ................ 28
  3.2 Code ............................................ 28
     3.2.1 Interconnection between VRML, Java, and Netscape 28
List of Figures

2-1 Four bar linkage, example of a kinematic chain. This is an example of a system a designer might try to simulate. ........................................ 15
2-2 Cylindrical joint ............................................. 16
2-3 Prismatic joint .................................................. 16
2-4 Helical joint ..................................................... 17
2-5 Revolute joint ................................................... 17
2-6 Spherical joint .................................................. 18
2-7 This diagram shows the structure of a VRML browser (plugin). .... 22

3-1 The normal force on the box constrains it to move to move only along the plane at angle $\theta$. ......................................................... 26
3-2 The forces $F_1$ and $F_2$ always act in an equal and opposite direction in this dumbbell. ......................................................... 27
3-3 Diagram of the different parts of a VRML and Java world. The different boxes may send messages to each other, except for the LiveConnect/ActiveX component, which acts as a pass-through for messages between Java and the VRML browser. ........................................... 30
3-4 Class hierarchy for kinematic simulation. .............................. 31

4-1 The applet user interface for the VRML browser. ...................... 34
4-2 This figure shows an example of the simulation. The engine element is in the middle, connected to a cylindrical element. ...................... 35

A-1 This figure shows the class relationship for Vrml.Objects. ............ 41
A-2 The Control class (which extends Applet) and the Driver classes methods are shown.
List of Tables

3.1 The degrees of freedom for each element are given. The degrees of freedom in a place are given by \( x \), and the angular degrees of freedom given by \( \theta \), \( \rho \), and \( \phi \) [19].
Chapter 1

Introduction

1.1 Kinematics

The study of motion of mechanical systems has often been the focus of important studies over the last 400 years. Galileo, Newton, Euler, Ampère, and Bernoulli are among those who focused on mechanical systems for a significant part of their work. The motion of a body has two quantitative parts, its geometry and its force component. Euler noted this when in 1775 he wrote:

The investigation of the motion of a rigid body may be conveniently separated into two parts, the one geometrical, the other mechanical. In the first part, the transference of the body from a given position to any other position must be investigated without respect to the causes of motion, and must be represented by analytical formulae, which will define the position of each point of the body. This investigation will therefore be referable solely to geometry, or rather to stereotomoy.\textsuperscript{1}

During the early 1800s, the first books dedicated to kinematics were published by professors at École Polytechnique. In 1834, Ampère formalized the study of mechanics of motion by coining the term \textit{cinématique} from the Greek word for motion, which later became kinematik in German. He wrote:

\textsuperscript{1}From Euler's "Novi commentarii Academiae Petrop.,” vol. XX, 1775.
There exist certain considerations which if sufficiently developed would constitute a complete science, but which had hitherto been neglected, or have formed only the subject of memoirs or special essays. This science ought to include all that can be said with respect to motion in its different kinds, independently of the forces by which it is produced . . . It should treat in the first place of spaces passed over, and of times employed in different motions, and of the determination of velocities according to the different relations which may exist between those spaces and times[5].

Thus, kinematics is defined as the study of motion, ignoring the physical forces acting on the objects that compose the mechanical system.

1.2 Kinematic simulation

In any system that contains moving parts, a designer wants to know the geometry of motion of those parts. Using kinematics, the designer can solve this problem. Computers can be a useful tool in modeling the kinematics of a system. Using computers, a kinematic simulation can be created to model the geometric motion of moving parts. Unfortunately, computers have not yet become the ubiquitous tool for modeling kinematics or forces that they have promised to be because of the complexity involved in creating simulations. However, step by step, these problems are being overcome.

One way to overcome the complexity is to break down the problem of simulation into simple parts to be solved individually. In this thesis, the problem of kinematic simulation of lower kinematic pairs will be addressed. Lower kinematic pairs are pairs of mechanical elements that contact each other only on a surface. There are a finite number of them that are most often used in creating a kinematic model. They are important in developing any sort of mechanical system that involves joints, and they are easily adaptable to create many different mechanical systems.

There are a variety of ways to model simulated mechanical systems, for example, using penalty functions or mathematically solving motion at infinitesimally small time

2 From Ampère’s “Essai sur la philosophie des sciences”, 1834.
steps. This report will focus on using constraints to model the kinematic motion of a system. Constraints offer the advantage of simplicity and minimal computation, and are suited for a finite number of elements.

In order to model the kinematic simulation with a computer, a user will be presented with an interface that will allow them to place lower kinematic pairs in a "scene" or "world" and then connect the elements. In addition to the lower kinematic pairs, there is a ground element and a driving "engine" element. The user may connect the elements in any manner they wish and then simulate the motion of the system. The system motion will be resolved by constraints on each of the elements.

The hope is that the kinematic simulation will help designers build and test their system quickly on a computer screen before spending time and material building real prototypes.

1.3 The Program Code

The program for modeling kinematic pairs was written in Java, which is an object oriented language. The interface to the scene is written in VRML, which provides a set of 42 primitive nodes from which scenes or worlds can be created and manipulated. The advantage of VRML is that it is available as a plugin to web browsers, making the program available across different operating systems. In addition, each element in the VRML scene can be represented as an object in Java, making manipulation of the objects easy from a coding point of view.

The code uses an application programming interface (API) to the VRML browser to communicate with the browser. The API for VRML is known as the External Authoring Interface, or EAI.

1.4 What follows

The following chapter gives some more background on kinematics and discusses what VRML is and how it is used in the simulation program. Chapter 3 discusses different
methods of constraining systems, and also goes into more detail on the design of the code for the simulation. Following that is a brief explanation of the program and some screen shots, followed by a discussion and conclusion in the last chapter. The appendices give more details on the code and its functions along with a listing of the code.
Chapter 2

Background

When an engineer is designing a mechanical system, he is given a set of design parameters. The design parameters are one set of constraints that usually describe the size, weight, cost, and other factors necessary for the machine to perform well. Mechanical systems that consist of moving parts are kinematically constrained. They must follow a certain path of motion in order to efficiently deliver energy from one part of a system to another. They may also be spatially constrained by the environment in which they will work. For example, a robot in a factory may be constrained to move along a certain path so that it does not injure people nearby. As the designer is creating the system, one of the challenges is to prototype the two dimensional drawing as a three dimensional object so he can simulate the resulting system. This is often done by building the part out of wood or foam in a machine shop, or perhaps even with cardboard. Reducing the amount of time spent on prototyping and increasing the accuracy of the prototype is one way to reduce costs for the engineer. This can be done by computer simulations.

2.1 Kinematic Modeling

One part of the simulation problem is kinematic modeling. On the face of it, it is a relatively simple problem. Kinematics is the study of the motions of mechanical systems. The kinematics ignores the forces, and therefore the accelerations and masses
of the system are irrelevant. The only question is whether the system produces the desired motion.

For example, in the design of a water sprinkler, the motion of the system must allow the sprinkler system to move in a periodic motion so that the water is distributed evenly. When working on the kinematics, the forces on the system can be ignored, we are just concerned with motion. The motion in question might be for the endpoints of a system like the four bar linkage shown in figure 2-1. Another example is a piston engine. A piston is generally driven by a cam that must produce a straight line motion to get the maximum efficiency out of the engine. The path of travel will drive the design process.

One way to help designers is to automate this process of kinematic simulation through the use of computers. The system described here shows how a simulation can be used to model kinematic systems in an intuitive, graphical way. The goal of this simulation is to build a framework system that can be easily expanded to add other features and allow designers to more accurately simulate complex three dimensional systems before time and money are invested in prototyping and then design. This software focuses on design using mechanical joints known as lower kinematic pairs.

2.1.1 Lower kinematic pairs

A kinematic pair is defined as two bodies whose motion relative to each other is constrained, for example a sphere sitting on an infinite plane is constrained to always lie above the plane. A lower kinematic pair is two bodies that contact each other on a surface, for example a revolute joint or a helical joint.

A higher kinematic pair is two mechanical elements that make contact on a point or a curve, or two bodies whose motion is constrained by contact. An example of this would be a sphere on a plane, or a cylinder or cone lying on a plane in such a manner that it could roll. A set of gears is also a higher kinematic pair because the teeth contact each other on more that one surface.

Bodies made of of kinematic pairs linked together form a kinematic chain. Figure 2-1 show a four bar linkage, which is an example of a kinematic chain.
Figure 2-1: Four bar linkage, example of a kinematic chain. This is an example of a system a designer might try to simulate.

*Lower kinematic pairs* are elements that contact each other on a surface. The most common examples of lower kinematic pairs are

**Cylindrical joint** A bearing on a circular rod. The bearing may slide along the length of the rod or may rotate relative to the rod. 2 degrees of freedom (figure 2-2).

**Prismatic joint** An slider on a prismatic rod that can slide along the rod, but not rotate relative to it (unlike the cylindrical joint). 1 degree of freedom (figure 2-3).

**Helical joint** Similar to the prismatic joint, except the rate of motion is limited by the slope of the threads. 1 degree of freedom (figure 2-4).

**Revolute joint** A joint with a rod and bearing as a pivot. Essentially the same as a cylindrical joint without the ability to slide along the rod. 1 degree of freedom (figure 2-5).

**Spherical joint** A ball in a socket. The ball may rotate relative to the socket. 3 degrees of freedom (figure 2-6).

The kinematic simulation is designed to allow engineers the opportunity to test their designs. It will give them an indication of the path of travel of their system and allow them to rearrange the design without great effort. It is also a way for people learning about engineering to use a hands on tool to get an intuitive feel for kinematics. The software is, however, limited in that it cannot provide any indication
Figure 2-2: Cylindrical joint

Figure 2-3: Prismatic joint
Figure 2-4: Helical joint

Figure 2-5: Revolute joint
of the forces or acceleration on the bodies besides that which can be gleaned from viewing the kinematic simulation.

2.1.2 Animating the model

In order to animate the kinematic simulation, some modeling tool will be required. There do exist some modeling tools that will numerically solve the motion of objects. Some of them are scripts written for numerical analysis tools such as Mathematica or Matlab. Other tools are built into drawing programs like ProEngineer or SolidWorks.

This thesis is focused on using pre-built mechanisms and making the tool as easy to use as possible. For this reason, the actual animation is modeled by constraints that are modeled as an object in the program code. These constraints can be easily removed and replaced by another method of resolving motion if need be. However, solving the motion and creating the animation are two separate parts. The motion constraints are described in section 3.1 and the animation with VRML is discussed below.
2.2 VRML as a Graphical Tool

2.2.1 A brief history of VRML

VRML stands for Virtual Reality Modeling Language. The concept of a three-dimensional (3-D) world came out of two ideas at the first international conference on the World Wide Web in 1994. The first was a presentation by Mark Pesce and Tony Parisi on their program called Labyrinth that allowed users to integrate large scale virtual worlds. The second was a desire to make a 3-D file format similar to HTML in that it would be text based and platform independent.

At the end of the conference a mailing list was set up to come up with ways to create this file format. After much debate on the mailing list, it was decided to use a language based on Silicon Graphics’ Open Inventor file format. The resulting language was called VRML 1.0, and was released on May 26, 1995. A revision to version 1.0 called VRML 1.0c was later distributed.

The first version of VRML is a static scene description language. It allows a developer to create a three dimensional scene that a user could view, rotate, and examine. The description language allowed the user to create nodes that described each object in the world. Each node had its own set of properties that defined shape, size, color and dimension. In addition, there were other nodes for lighting, level of detail, and WWW links. The nodes could be reused, modified, and transformed. The node structure made VRML somewhat of an object oriented description language.

One advantage of VRML’s open format is that it is easy to write converters to and from VRML. This is useful for the developer that needs to draw things in a simple, portable and convertible format. There are many tools to convert existing drawings from Open Inventor, 3D Studio and ProEngineer to VRML 2.0 and also various “lint”s for VRML.

A feature that VRML 1.0 that was lacking was it allowed little to no interaction with the objects in the world. It was a good first step, but more needed to be done. The next version of VRML needed to give the creators of worlds a way for their users to interact with the objects in the world. It brought this capability to the developer
in two ways. First, the developer can add interactive behavior directly to objects in the worlds (nodes). If a user clicks on an object, it can move in some predetermined fashion. Objects may translate or deform over time. Developers can also add real time behavior to the objects using a node called a Script Node. This node allows the developer to send messages to other nodes in the scene. Currently there is support for Script Nodes in Java and Javascript.

The second way the developer can add behavior is using an external application interface (EAI) [12]. The EAI is an application programming interface (API) to the VRML world that may be implemented in any language. There is a Java version of the EAI that some VRML browsers support. The EAI allows a developer to let the user change properties of objects in a scene. In effect, the EAI has a handle on the scene, and can add objects, remove objects, or change objects in the scene. It is similar to the Script Node, but adds some more features because the developer can add a Java graphical user interface (applet) to a web page which plays a part in manipulating a VRML scene.

The current version of VRML in wide use that supports these new behaviors is the April 4, 1997 VRML 2.0 ISO-DIS (draft for international specification). This is the version used in this thesis.

2.2.2 Anatomy of VRML

Figure 2-7 shows the structure of the VRML browser with the external interface [11]. The external authoring interface is an extension to the VRML plugin. The EAI code connects to the browser through a proprietary browser connection, however, the connection is normally taken care of by the EAI, so it does not matter to the developer which browser the user chooses.

The bottom of figure 2-7 show the different kinds of VRML nodes, the basic building blocks of a VRML scene. The script node also allows a developer to control a world through code. The node is a formal part of VRML, and provides a compact way to manipulate objects. The proto node is another way to extend VRML. It allows a developers to come up with their own type of node or object that can be
used just like any other node in the VRML script. This is useful when there is an oddly shaped object that may need to be used over and over in the scene. Finally, the standard node provides primitives with which the world may be created. There are forty standard nodes in all, not including the script and proto nodes.

2.2.3 Use of VRML

VRML is still in it’s early stages of development. It has much potential, but it has not become as widely accepted as HTML. Part of the reason is that it is computationally intensive. Rendering three dimensional objects quickly can be slow even on the fastest machines, depending on the number of polygons involved. Games would do well on VRML except that it is much easier to tune them for performance when a custom (proprietary) rendering engine is used. Also, people have struggled to come up with a “killer app” for VRML that will make it ubiquitous. People have begun to add three dimensional interfaces to databases; perhaps VRML will fill a role for people trying to access data over the web.

One application that has recently become popular is using VRML for banner ads on web pages. The advantages of this are that people can interact with the adds if they contain script nodes. They are also arguably flashier than animated GIFs. And they are definitely more compact. The problem here is that the add takes a significant amount of time to render on the end user's machine, making the process appear slower and thus more frustrating than animated GIFs.

Other popular applications include modeling chemical elements, building models of buildings and cities, and trying to build animated characters in VRML. More information on VRML can be found at [1], [13], and [17]. Information on how VRML is being used on the web can be found at [3], [4] and [8].

One disadvantage of VRML is that it has a limited number of predefined primitive shapes. This results in somewhat inflated file sizes and requires a developer to create his own primitives for shapes which do not exist, which is inefficient. It’s EAI is bulky next to VRML, since you must keep almost parallel representations of the VRML world in Java. Future versions of VRML may include inherent multiuser capabilities.
Figure 2-7: This diagram shows the structure of a VRML browser (plugin).
and the use of the Java external interface as part of the specification. There is a proposal for a compressed VRML binary file format to allow an even smaller file size.

VRML itself is only a specification for the language. A third party must write a VRML viewer that reads a description file and produces the scene. The VRML viewer can be written as a stand-alone program or as a plugin for a web browser. The browser plugin is a natural extension of the idea that VRML is the 3-D version of HTML. In this thesis, I will be assuming that the user is using a VRML 2.0 compatible viewer that supports the Java EAI (see [7], [9] for more information on browsers).
Chapter 3

System Design

The following chapter will briefly outline a number of ways to constrain elements in a system and discuss how it is done in this program. Then, the design of the code using VRML and Java will be presented.

3.1 Constraints

One of the challenges of modeling a real system in a computer simulation is that the position of each element in the system must be known at every time step. This is an advantage and disadvantage of computer systems. One of the problems with doing software simulation is the time step. A computer cannot work in the limit as $\Delta t$ approaches zero, the time step will always be measurable. This can bring about instabilities in the simulation if the time step is too large, but can cause the program to run painfully slowly if the time step is too small. However, the user can visualize with a computer every motion or force acting within the system. In contrast, a human trying to model a system will usually find the end result with great accuracy without seeing the time steps in between.

To model such a system on a computer, it is impossible to enter all the possible permutations of forces and elements in a system. Ideally, the computer should know just as much as it needs about modeling systems, and no more. There are various ways to limit the amount of information needed depending on the system to be modeled.
Four such methods are described below. They are: penalty functions, which constrain a system based on the forces and motions that exist; force constraints, which remove forces based on the geometry of a system; lookup tables, which limit motion based on predefined rules for different connections; and path of travel constraints, which limit motion based on predefined rules for different elements. The following sections describe each of these types of constraint in more detail.

### 3.1.1 Penalty functions

A penalty function is an equation that takes into account forces and motions on each part of a system and tries to satisfy the system. The penalty function effectively tries to minimize the penalty equation. The penalty function is given by

$$F(a) = f(a) + \sum_{ij} k_{ij} g_i(a) g_j(a).$$  \hspace{1cm} (3.1)

where $a$ is a vector of the free parameters in the system, $g(a)$ is an equality constraint, $f(a)$ is a chi-square distribution, and $k$ is a positive constant large enough (on the order of $10^6$--$10^8$) so every term is more important than $f(a)$. The function can also be used for equality and inequality constraints where $g(a) > 0$, except the equation only comes into play when the inequality constraints are violated [2].

For example, if a beam is attached to $n$ springs and is driven by an impulse, a penalty function of such a system would require that the displacement of the beam must line up with the displacement of each of the springs (the equality constraint) and the penalty constant $k$ is the spring constant for each spring.

The penalty function may not minimize the constraints at every time step, but as long as the constant $k$ is large enough and the forces are small relative to $k$, then the constraints are usually satisfied. Solving a constraint equation is computationally intensive, but creating the constraints is not terribly difficult. It has been said that the “dependence on contact coherence is perhaps the greatest limitation of the analytic methods” [14].
Another method of modeling a system in a computer simulation is to use a constraint force. The idea of a constraint force is to replace physical objects in a system with the forces that they must impart [6]. For example, if a vehicle is moving on an inclined surface, the vehicle is constrained by the normal force acting on it to move at the same angle as the surface, as shown in figure 3-1. Another example is a cart constrained to move along a track. The cart will always have a force acting on it normal to the track, no matter what direction the track takes.

In the same way, objects can be constrained by their acceleration and relative distance. Yet another example is the forces on a dumbbell. A dumbbell always has two masses a fixed distance apart from each other. The force on the connecting rod (normal to the masses) must be the same for both masses.

One disadvantage of this type of constraint is that each body must be modeled as a point mass. As point masses, it becomes easy to use Newton’s equations to solve for the motion and accelerations. However, there are many exceptions to this case. For example, the ball joint show in figure 2-6 is constrained to move within its socket. There is a normal force acting on every point on the surface of the ball, and there is no way to constrain it as a point mass; the surface must be considered as a three-dimensional surface.
3.1.3 Lookup tables

Another way to constrain objects is to set rules for connections between different types of objects. In other words, the constraints on elements in a mechanical system are defined by their interconnection. The interconnection for different objects is found in a lookup table or database. A set of commonly used joints can be added to the database and resolved at runtime. The resolved connections may have force or connection constraints attached to them. This information allows the program to drive the motion of the system. The lookup table effectively generates the correct "messages" that different elements receive to move in a physically correct fashion [10].

This is a useful way to group elements in a system. There is a large amount of complexity involved in building lookup tables (or matrices) and examining the elements to determine how they can be grouped for different elements in a table. There are many questions to be answered, like, should a four or six bar linkage be viewed as single entities in the lookup table, or are they a collection of separate two bar elements linked together? One way to answer these questions is to have tables of tables, or n-dimensional matrices with different properties in each dimension. These questions make the lookup table a somewhat complicated to implement.

Figure 3-2: The forces $F_1$ and $F_2$ always act in an equal and opposite direction in this dumbbell.
<table>
<thead>
<tr>
<th>Element</th>
<th>Degrees of Freedom</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cylindrical</td>
<td>$x, \phi$</td>
</tr>
<tr>
<td>Prismatic</td>
<td>$x$</td>
</tr>
<tr>
<td>Helical</td>
<td>$x$</td>
</tr>
<tr>
<td>Revolute</td>
<td>$\phi$</td>
</tr>
<tr>
<td>Spherical</td>
<td>$\theta, \rho, \phi$</td>
</tr>
</tbody>
</table>

Table 3.1: The degrees of freedom for each element are given. The degrees of freedom in a place are given by $x$, and the angular degrees of freedom given by $\theta$, $\rho$, and $\phi$ [19].

### 3.1.4 Path of travel constraints

A computer does not have knowledge of physics, so the physics of every problem must be built into software. By narrowing the problem down to kinematic simulation, much of the physics can be taken out of the picture while still getting a good approximation of motion.

One way to remove the physics is determine the motion (ie., the path of travel) that will result from a given set of forces on a set of objects. Six coordinates are needed to specify the location of a body in 3-space. Limiting the bodies to lower kinematic pairs, the number of degrees of freedom is limited to at most 3. Assuming the degrees of freedom of any object are $x$, $y$, and $z$ in space and can be rotated at $\theta$, $\rho$, and $\phi$, the list of degrees of freedom for the lower kinematic pairs is given in table 3.1.

This is similar to a lookup table, but in this case, the knowledge about the forces and motion is built into each element before it is even instantiated in the scene.

### 3.2 Code

#### 3.2.1 Interconnection between VRML, Java, and Netscape

The user interface for the simulations consists of two parts: the VRML plugin in the web browser and the Java interface in the browser. The anatomy of the browser was shown in figure 2-7. The two parts communicate with each other through either Netscape Navigator’s LiveConnect (also known as Javascript) or through Microsoft
Internet Explorer's ActiveX.

Figure 3-3 shows how the different components pass messages to each other. The web browser provides the framework for the connections. The user may add or remove elements from the world with the Java GUI. This GUI sends a message to a class in the Java code, which then passes the message on to the connection. The connection then sends the message to the plugin. The process may also be initiated by the Java code, which may modify the GUI or send messages on its own to the VRML plugin. Or, a users' mouse events can be captured by the VRML browser and sent to the Java code to be processed. However, all VRML events have to be registered in Java beforehand. The LiveConnect/ActiveX component is built into the browser, and only serves to manage messages. It does not initiate any messages by itself.

3.2.2 Class hierarchy

The kinematic simulation code was designed with the Java object oriented programming language. The code attempts to take advantage of the object oriented nature of the language by structuring the relationship between objects in a natural way. Figure 3-4 shows the class hierarchy of the code. In the Java programming language, every object inherits from the Object parent class.

The Control class is the class that creates the user interface. The Control class provides a drop down list of objects for the user to choose from, and the ability to add, remove, connect, or animate the objects. To create an object in the browser, an object of type Vrml_Objects is instantiated. The subclass LKPElement creates a lower kinematic pair element, while the parent class provides methods for accessing and manipulating the object. Unfortunately, the subclassing is not as clean as it should be; all VRML elements including the engine and ground are LKPElements.

The Driver element is instantiated when the user clicks on the Start button in the user interface (Control class). The Driver class manages the motion of the objects on the screen with constraints. The Driver class is independent of the joints that appear on the screen, but it is linked to the Engine class.

All of the Java code may be considered a Java package, even though it has not
Figure 3-3: Diagram of the different parts of a VRML and Java world. The different boxes may send messages to each other, except for the LiveConnect/ActiveX component, which acts as a pass-through for messages between Java and the VRML browser.
Figure 3-4: Class hierarchy for kinematic simulation.
been compiled that way. However, it would be trivial to do so. More information on
the methods that comprise the Java code can be found in Appendices A and B.
Chapter 4

The Kinematic Modeler

The user interface is shown in figure 4-1. The user first selects an element from the menu options. Then, the element may be added to the scene by choosing the Add element button. Once the element is in the scene, it may be removed by clicking on it and then clicking on the Remove element button. To join two elements together, click on the two elements to be connected and then click on Connect. To animate the simulation, click on the Start button. The engine will run for $2\pi$ radians (one revolution) then stop. The Disconnect and Stop buttons are not functional at this time.

A snapshot of the kinematic simulation program is shown in figure 4-2. The diagram shows an engine element (in the center) connected by a connection object to a cylindrical joint. The cylindrical joint is connected to a helical joint, and that in turn is connected to a spherical joint. The ball in the lower right hand corner of the window is intended as a visual anchor, so that if the user spins the simulation around they know where they are positioned relative to the starting point. The positioning of the elements in this example is similar to the four bar linkage shown in figure 2-1. The first and last elements (the engine and the spherical joint) are essentially constrained as ground elements, since their centers do not move anywhere. The engine drives the system, and the sliders on the bars show a constrained path of travel for these joints.

The user may add as many elements to the scene as they wish. There can only be one engine in the scene, and each element can be connected to only one other
element. The user may remove elements and their connectors, then add new ones to the scene to create a new simulation.
Figure 4-2: This figure shows an example of the simulation. The engine element is in the middle, connected to a cylindrical element.
5.1 The VRML/Java Simulation

The kinematic simulation is a good step towards creating a system where engineers can model motion. It models a fixed set of joints and displays their motion, allowing the user to connect different joints together.

The advantage of the class hierarchy of the system is that joints can be easily added, and the method of motion (constraints, forces, etc.) can be easily removed and replaced by another object. However, the system does need some work in modifying the class structure to make it more accurately reflect the relationships between objects.

5.2 Recommendations

There are a number of ways the kinematic simulation can be improved. Some are given below.

5.2.1 Using constraints

Using constraints is a useful way to constrain mechanical joints since for each joint, the range of motion is known. The forces on each joint do not need to be calculated,
since the constraints are used to constrain motion, not forces. There are a number of different ways to simulate the motion, as mentioned in section 3.1. Using penalty functions has the advantage of being very general, but the disadvantage of being difficult to code. One way to expand the system might be to consider the forces on the joints in addition to the constraints. This would allow other disparate elements to be added to the system, as long as each object knew the forces that acted on it. Or, lookup table scheme could be implemented where objects whose constraints determined motion had their constraints imposed, and forces for other objects calculated to determine motion. This is assuming all constrained elements are frictionless.

5.2.2 Modifying the program

Some of the features of the program can be improved. The program runs as a single thread. If it were multithreaded, the Stop button could be used to stop the simulation at any point. A useful tool for viewing the animation would be allowing different objects to leave a graphical trail. Or, one addition might be a text entry field where the user can specify how many time steps the program is to run for.

Another way to improve the simulation is to use the Java Native Interface (JNI) to get C or C++ code to perform the calculations for the constraints or forces. Although Java is fast, it still uses a bytecode interpreter. The JNI allows Java code to link to C or C++ routines. This is based on the assumption that the non-Java code will be faster than the Java Virtual Machine (JVM), but on an Intel based Windows machine, the JVM is almost as fast as native C code. The just in time (JIT) compilers for Java does speed Java up to close to native performance.

Yet another way to speed up the program is to recode all of it using a faster graphics tool than VRML. One possibility is to use Open Inventor, which is similar to Java/VRML in that it represents objects in a scene as C++ objects. The disadvantage of this is that the program is more difficult to port over a wide number of platforms. Open Inventor is available for PCs and SGI's.
5.2.3 New elements

Adding new elements to the system would enhance its capabilities. Two elements that would be useful to add are spring and damping elements. Springs would be useful because joints could be connected by springs of different stiffnesses. Adding dampers would simulate friction in the simulation, and combinations of springs and dampers would produce interesting forces within the system. If springs and dampers were added, having the ability to produce graphs of distance/velocity/acceleration versus time for any element would be useful.

5.3 Future work

The VRML kinematic simulation does it's job well, but is somewhat constrained by the limited number of elements. The lower kinematic pairs were chosen for their ubiquity and usefulness, but it would be beneficial for the functionality if more elements like springs and dampers were added. It may also be interesting to wrap the code up as a Java package that can be used as an API for other kinematic simulations.
Appendix A

Explanation of methods

This appendix discusses some of the methods used in the code. The class diagram is the same as the one discussed in figure 3-4 in chapter 3, with the methods added.

A.1 Vrml_Objects Class

Figure A-1 shows the methods used by Vrml_Objects and it’s subclasses. The subclasses mostly contain only constructor methods, leaving the method implementation to the parent. The two exceptions are the Connector class and the Engine class, shown in figure A-2.

All the Vrml_Objects are registered in the Vrml_Objects constructor and have a unique integer index. The Vrml_Objects are stored in a java.util.Vector class. The Vector class provides the functionality of a linked list for elements of type Object. The callback function is a virtual function of EventOutObserver, which is implemented by this class. This function returns mouse events for registered objects. The get_index function returns an instance’s index.

Most of the methods in the Vrml_Objects class relate to positioning the object either angularly or in the x-z plane. The position is set either by the user when the objects is dragged in the browser or by the Driver when the object is animated. The get and set translation and rotation functions set these positions.

The VRML joints are made up of two different parts, the stationary part and
the moving part. The moving part is actually a sub-node of the stationary node in VRML. The reason they are separated is so the Java code has a pointer to the Translate{} node of each part. The transArray variable is the stationary portion, and the transArray2 is the animated portion or the object. The Vrml_Object provides methods to access both of these variables.

One important part of the code is the ability to connect different elements together. The add_arm function is called by Connector when a connection arm needs to be created. The creation of the arm is slightly different if the arm is coming from an Engine. The arms, or connectors, are also kept in an array as the joints are. The set and get functions add and retrieve Connector pointers from the array. In addition, a Vrml_Objects element can find out which other elements is is connected to.

The Connector class creates the connection arm object. The arm must translate and rotate in a different fashion than the Vrml_Objects. It must also know how to calculate the distance and angle between different elements so it can be instantiated at the correct orientation. The calculate_quadrant function returns the quadrant of the second connecting joint relative to the first so when the angle between them is calculated using trigonometric functions, an appropriate fraction of $\pi$ can be added to correct for the quadrant.

The Engine class actually "run"s the animation. It produces the coordinates for the displacement of all the objects in the scene that the Driver class processes. It also implements ActionListener so it knows which buttons are pressed in the Control class (see below for more information on the Control and Driver classes). Before it runs, it creates a list of elements in the scene so it can displace each element in sequential order.

The other classes which only have constructor methods are joint objects (except for Ground), all of whose methods are inherited from Vrml_Objects.
Vrml_Objects

Vrml_Objects
get_index
callback
set_element_translation
set_element_rotation
get_element_translation
get_transArray
get_transArray2
get_geomArray
get_element_type
add_arm
set_connector
get_connector
get_connector_array
set_connected_to
get_connected_to

LKPElement

LKPElement

Prismatic

Revolute

Helical

Cylindrical

Connector

Prismatic

Revolute

Helical

Cylindrical

Ground

Spherical

Engine

Engine

run
actionPerformed
get_engine_radius
create_list_of_elements

translate
rotate
make_engine_connector
make_connector
get_connector_info
calculate_distance
calcualte_angle
get_angle
calculate_quadrant

Figure A-1: This figure shows the class relationship for Vrml_Objects.
A.2 Control and Driver Classes

Figure A-2 shows the methods used by the Driver and Control classes. The Control class inherits from the Applet class and implements ActionListener. Since it is an applet, it has an `init` and `start` method. The `init` method performs most of the work handling both the VRML browser and the applet interface, since neither needs to be redone each time the page is loaded. The `actionPerformed` method handles events from the applet, and is a virtual function from ActionListener. The method `makeCurrent` is called by Vrml_Objects and is used to make the object that is clicked on the current object in the scene. The `getTranslationChanged` keeps track of the position of the object if it is dragged in the VRML browser. The `getBrowser`, `getStartEvent`, and `getStopEvent` do exactly what they say they do. The latter methods are implemented for the benefit of ActionListener. The last method, `get_scene_elements` is used to get a list (Vector) of all the objects in the browser.

The Driver class has a constructor and an animate function that is called by the Engine's `run` method. The animate function implements the constraints imposed by the Engine class.
Appendix B

Code listing

The following pages list the code for the kinematic simulation. Each of the sections contains a class, and the final section has the html code used in creating the web page. There is one class that is left out due to its length, called wrl.java that contains a set of string constants used to create the actual joints. Appendix A contains more detail on the class diagram and function definitions.
B.1 Connector.java

import java.lang.*;
import java.lang.Math.*;
import java.util.*;
import Constraint;

public class Connector extends VrmlObjects {

    // to hold the vrml objects
    private VrmlObjects driving_element, driven_element;

    // so I know who I am connected from, and who I am connected to
    private int element_index[] = new int[2];
    private Constraint[] element_constraint = new Constraint[2];

    // used in calculating distances and angles for connector
    private float x1, x2, y1, y2, z1, z2, x;
    // the string created once the correct length and rotation are found
    private String vrmlconnector;

    // these translations are the offsets from the center, so they are basically position
    private float[] translation1 = new float[3];
    private float[] translation2 = new float[3];

    // angle of connector, and quadrant to which the connector connects relative to the
    // driving element
    private float angle;
    private int quadrant;

    Connector(VrmlObjects driving, VrmlObjects driven) {
        // note, this does not increment object_index, because the VrmlObjects() constructor
        // does not increment object_index
        System.out.println("In Connector constructor");
        driving_element = driving;
        driven_element = driven;

        element_index[0] = driving_element.get_index();
        element_index[1] = driven_element.get_index();

        // so I know how long to make myself
        translation1 = driving_element.get_element_translation();
        translation2 = driven_element.get_element_translation();
        x = calculate_distance(translation1, translation2);

        // so I know what angle I should be rotated at
angle = calculate_angle(translation1, translation2);
// adjust the angle that atan returns to take into account quadrant
quadrant = calculate_quadrant(translation1, translation2);

switch(quadrant) {
    case 1:
        angle += (float) Math.PI;
        break;
    case 2:
        break;
    case 3:
        break;
    case 4:
        angle += (float) Math.PI;
        break;
    default:
        System.out.println("Problem figuring out what quadrant the joint is in");
        break;
} // end switch

if (driving_element.get_element_type() == ENGINE) {
    vrml_connector = make_engine_connector(x, angle);
} else {
    vrml_connector = make_connector(x, angle);
}

driving_element.add_arm(vrml_connector);

driving_element.set_connector(this);
driven_element.set_connector(this);

System.out.println("angle is: "+ angle + " x is: " + x + " Quadrant is: " + quadrant);
System.out.println("Driving Element: " + driving_element.get_index() +
    " , Driven Element: " + driven_element.get_index());

} // end constructor

public void translate(float displacement[]) {

    switch(driving_element.get_element_type()) {
    case CYLINDRICAL_JOINT:
    case HELICAL_JOINT:
    case PRISMATIC_JOINT:
        displacement[1] = 0.0f;
        driven_element.set_element_translation(displacement);
        break;
    case SPHERICAL_JOINT:
    case GROUND:
        displacement[0] = displacement[1] = 0.0f;
        break;
    }
case ENGINE:
    driving.element.set_element_translation(displacement) ;
    break ;
default:
    System.out.println("Could not figure out which element to drive!") ;
    System.out.println("Driving element is: " + driving.element.get_element_type()) ;
    break ;
}

switch(driven_element.get_element_type()) {
    case CYLINDRICAL_JOINT:
    case HELICAL_JOINT:
    case PRISMATIC_JOINT:
        displacement[1] = 0.0f ;
        driven.element.set_element_translation(displacement) ;
        break ;
    case SPHERICAL_JOINT:
    case GROUND:
        displacement[0] = displacement[1] = 0.0f ;
        break ;
    default:
        System.out.println("Could not figure out which element is driven") ;
        System.out.println("Driven element is: " + driving.element.get_element_type()) ;
        break ;
}
} // end translate()

public void rotate(float displacement[])
{
}

private String make_engine_connector(float x, float angle) {
    x-=0.3 ; // to account for the radius of the engine

    String arm ;
    arm =
        "Group { " +
        "children Transform {" +
        "rotation 0 0 1 1.57" +
        "children Transform { " +
        "rotation 0 0 1 " + angle + " " + // this rotates the arm
        "children Transform { " +
        "translation 0 " + (x/2 + 0.3) + " 0 " + // this moves the arm for the engine
        "children Shape { " +
        "appearance Appearance { " +
        "material Material { " +
        "diffuseColor 0 1 0 " +

private String make_connector(float x, float angle) {
    // need to add two spheres here, to the end of the connector for looks
    String arm = "Group {
        " +
        "children Transform {" +
        "rotation 0 0 1 1.57" +
        "children Transform {" +
        "rotation 0 0 1" + angle + " " + // this rotates the arm
        "children Transform {" +
        "translation 0" + (x/2 + 0.3) + " 0" + // this moves the arm for the engine
        "children Shape {" +
        "appearance Appearance {" +
        "material Material {" +
        "diffuseColor 0 1 0" +
        "}" +
        "}" +
        "geometry Cylinder {" +
        "height " + x + // this sets the length of the arm
        "radius 0.05" +
        "}" +
        "}" +
        "}" +
        "}";
    
    return arm;
}
} // end make.connector

public String get_connector_info() {


int i, j;
i = driving_element.get_index();
j = driven_element.get_index();

String words = "Driving Element: " + i + ", Driven Element: " + j;
return(words);
}

public float calculate_distance(float[] a, float[] b) {
    x1 = a[0];
x2 = b[0];
y1 = a[1];
y2 = b[1];

    return((float) Math.sqrt(Math.pow((x1-x2),2)+Math.pow((y1-y2),2)));
}

public float calculate_angle(float[] a, float[] b) {
    x1 = a[0];
x2 = b[0];
y1 = a[1];
y2 = b[1];

    return((float) Math.atan((y2-y1)/(x2-x1)));
}

public float get_angle() {
    return angle;
}

// a is the driving element, the engine, b is the element whose quadrant
// relative to a this function will return
public int calculate_quadrant(float[] a, float[] b) {
    x1 = a[0];
x2 = b[0];
y1 = a[1];
y2 = b[1];

    if((x2 ≥ x1) && (y2 ≥ y1)) { return 1; }
    if((x2 < x1) && (y2 ≥ y1)) { return 2; }
    if((x2 < x1) && (y2 ≤ y1)) { return 3; }
    if((x2 ≥ x1) && (y2 < y1)) { return 4; }
    else return -1;
}

public int get_quadrant() {
    return quadrant;
}
public Vrml_Objects get_driving_element() {
    return driving_element;
}

public Vrml_Objects get_driven_element() {
    return driven_element;
}

} // end Connector class
B.2 Control.java

// Code for controlling the Java/VRML simulation
// Sameer Raheja 9/2/97

import java.lang.*;
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.util.Vector;
import vrml.external.field.*;
import vrml.external.Browser;
import vrml.external.Node;
import vrml.external.exception.*;

// needed for netscape's livescript/javascript handle between
// VRML plugin and applet if JSObject is used
// import netscape.javascript.JSObject;
// import netscape.javascript.JSException;

import VrmlObjects;
import wrl;

//+++**********************************************************************
//@*****+************************************************************************
//public class Control extends Applet implements ActionListener {

private Browser browser; // handle on the VRML plugin
private Node root; // node to hold the default root.wrl file

private Choice element_choice; // a dropdown list
// to hold the list of all the elements added to the scene
private Vector scene_elements = new Vector();
// huh? i don't use this
private Vector element_position = new Vector();

private VrmlObjects curElement = null; // to hold the most recently clicked element
// don't think i need the following
// private Vrml.Objects temp = null; // used in removing elements

private EventInMFNode addChildren = null; // event to add objects
private EventInMFNode removeChildren = null; // event to remove objects

private int clicked_element_array[] = {-1,-1}; // holds the last two element clicked

private boolean engine_suspended = false;
private Vector engine = new Vector(); // to hold the engines?
private Vector connector = new Vector(); // to hold the connector objects
// static Connector connector;

private Button connect_button, disconnect_button, start_button, stop_button;
private Button remove_button, add_button;

//**************************************************************
//**************************************************************
public void init() {
    // try getting the CosmoPlayer plugin using JavaScript
    try {
        // the newer versions of CosmoPlayer prefer you to use the getBrowser() function,
        // so the call is browser independent
        // static public Browser getBrowser(Applet pApplet, String frameName, int index);
        // static public Browser getBrowser(Applet pApplet);
        browser = browser.getBrowser(this);
        // Here is a Netscape Javascript/Livescript analagous method, works with older CP
        // JSObject win = JSObject.getWindow(this);
        // JSObject doc = (JSObject) win.getMember(document);
        // JSObject embeds = (JSObject) doc.getMember(embeds);
        // browser = (Browser) embeds.getSlot(0);
    } // end try
    catch (Exception e) {
        System.out.println("Exception getting browser: "+ e);
    } // end catch

    // try getting the root node in the browser
    try {
        root = browser.getNode("ROOT");
    } // end try
    catch (InvalidNodeException e) {
        System.out.println("Problem getting ROOT node:" + e);
    } // end catch

    // Create a dropdown list to choose which object you want to add
    element_choice = new Choice();
    Panel p_bottom = new Panel();
    Panel p_top = new Panel();

    p_bottom.setLayout(new FlowLayout());
    p_top.setLayout(new FlowLayout());

    connect_button = new Button("Connect");
    disconnect_button = new Button("Disconnect");
    start_button = new Button("Start");
    stop_button = new Button("Stop");
    remove_button = new Button("Remove element");
add_button = new Button("Add element");

connect_button.addActionListener(this);
disconnect_button.addActionListener(this);
start_button.addActionListener(this);
stop_button.addActionListener(this);
remove_button.addActionListener(this);
add_button.addActionListener(this);

p_bottom.add(connect_button);
p_bottom.add(disconnect_button);
p_bottom.add(start_button);
p_bottom.add(stop_button);

element_choice.addItem("Cylindrical Joint");
element_choice.addItem("Helical Joint");
element_choice.addItem("Prismatic Joint");
element_choice.addItem("Revolute Joint");
element_choice.addItem("Spherical Joint");
element_choice.addItem("Ground");
element_choice.addItem("Engine");

p_top.add(add_button);
p_top.add(remove_button);
p_top.add(element_choice);

add("North", p_top);
add("South", p_bottom);

} // end init

public void start() {
    System.out.println("****Starting!");
} // end start

public boolean action(Event evt, Object arg) {
    public void actionPerformed(ActionEvent evt) {
        VrmlObjects newElement = null;
        try {
            if (evt.getActionCommand().equals("Add element")) {
                System.out.println("In add element!");
            }
        } catch (Exception e) {
            System.out.println("Error in add element!");
        }
    }

    try {
        if (evt.getActionCommand().equals("Add element")) {
            System.out.println("In add element!");
        }
    } catch (Exception e) {
        System.out.println("Error in add element!");
    }
}

52
System.out.println(element_choice.getSelectedItem());

// Create a new element
if (element_choice.getSelectedItem() == “Cylindrical Joint”)
    newElement = new Cylindrical(this);
if (element_choice.getSelectedItem() == “Helical Joint”)
    newElement = new Helical(this);
if (element_choice.getSelectedItem() == “Prismatic Joint”)
    newElement = new Prismatic(this);
if (element_choice.getSelectedItem() == “Revolute Joint”)
    newElement = new Revolute(this);
if (element_choice.getSelectedItem() == “Spherical Joint”)
    newElement = new Spherical(this);
if (element_choice.getSelectedItem() == “Ground”)
    newElement = new Ground(this);
if (element_choice.getSelectedItem() == “Engine”)
    newElement = new Engine(this);

// just so java does not lose pointer to object
scene_elements.addElement(newElement);
// now add the new element to the scene
try {
    addChildren = (EventInMFNode) root.getEventln(“addChildren”);
    addChildren.setValue(newElement.get_transArray());
}
catch (InvalidEventlnException e) {
    System.out.println(“Problem adding child: ” + e);
}

if (evt.getActionCommand().equals(“Remove element”)) {
    System.out.println(“Removing Element “ + curElement.get_index());
    try {
        if(scene_elements.contains(curElement)) {
            System.out.println((curElement.getgeomArray()).getType());
            removeChildren = (EventInMFNode) root.getEventln(“removeChildren”);
            removeChildren.setValue(curElement.get_transArray());
            scene_elements.removeElement(curElement);
            curElement = null;
        }
    } else {
        System.out.println(“Invalid element selected”);
    }

} catch (NullPointerException e) {
    System.out.println(“Null pointer exception: ” + e);
}
catch (InvalidEventlnException e) {
    System.out.println("Problem removing child, index " + curElement.get_index() + ": e") ;
}
} // end if

if (evt.getActionCommand().equals("Start")) {
    System.out.println("Starting engine") ;
    // the Engine has an ActionListener for this button...
    // if (connector != null) {
    //     connector.get_connector_info() ; // this is not returning anything!
    // }
    // else { System.out.println(Connector is null) ; }
} // end if

if (evt.getActionCommand().equals("Stop")) {
    // seems like i can't get the thread to stop or suspend
    System.out.println("Stopping engine") ;
    // temporary
    // System.out.println(connector.get_connector_info()) ;
} // end if

if (evt.getActionCommand().equals("Connect")) {
    int a0, a1 ; // integers to hold the element index we are connected from and to
    if((clicked_element_array[0] != -1) && (clicked_element_array[1] != -1)) {
        connector.addElement(new Connector(
            (Vrml_Objects) scene_elements.elementAt(clicked_element_array[0]),
            (Vrml_Objects) scene_elements.elementAt(clicked_element_array[1]))) ;

        a0 = ((Vrml_Objects) scene_elements.elementAt(clicked_element_array[0])).getindex() ;
        a1 = ((Vrml_Objects) scene_elements.elementAt(clicked_element_array[1])).getindex() ;

        ((Vrml_Objects) scene_elements.elementAt(clicked_element_array[0])).set_connected_to(a1) ;
        ((Vrml_Objects) scene_elements.elementAt(clicked_element_array[1])).set_connected_to(a0) ;

        ((Connector) connector.lastElement()).get_connector_info() ;
    } else { System.out.println("You need to click on two elements") ;}
} // end if

if (evt.getActionCommand().equals("Disconnect")) {
    System.out.println((curElement.get_geomArray()).getType()) ;
    // removeChildren = (EventInMFNode) transform2.getEventIn(removeChildren) ;
    
    } // end try
    catch (NullPointerException e) { System.out.println("In Control: " + e) ; }
    catch (Exception e) { System.out.println("Exception creating or removing element: " + e);
          
    } // return true;
} // end actionPerformed

//************************************************************************
//************************************************************************
public void getTranslationChanged() {
    float[] val = new float[3];
    val = curElement.get_element_translation() ;
    System.out.println("Translation value: "+ val[0] + " " + val[1] + " " + val[2]);
}

//************************************************************************
//************************************************************************
public void makeCurrent(Vrml_Objects which) {
    curElement = which ;
    clicked_element_array[1] = clicked_element_array[0] ;
    clicked_element_array[0] = curElement.get_index() ;
    System.out.println("Object Index = " + clicked_element_array[0] ) ;
    System.out.println("Object Last touched = " + clicked_element_array[1] ) ;
    getTranslationChanged() ;
} // end makeCurrent() 

//************************************************************************
//************************************************************************
public Browser getBrowser() {
    return browser;
} // end getBrowser()

//************************************************************************
//************************************************************************
// public Control() {
//     super();
//     System.out.println("In Control constructor");
// }
public Button getStartEvent() {
    return start_button;
}

public Button getStopEvent() {
    return stop_button;
}

public Vector get_scene_elements() {
    return scene_elements;
}

} // end class
import java.lang.*;
import vrml.external.field.*;
import vrml.external.Browser;
import vrml.external.Node;
import vrml.external.exception.*;

public class Cylindrical extends LKPElement {

    Cylindrical(Control myParent) throws IllegalArgumentException {
        this.super(myParent);

        element_type = CYLINDRICAL_JOINT;

        System.out.println("In Cylindrical constructor, type = " + element_type);

        try {
            geomArray = browser.createVrmlFromString(wrl.cylindrical);
            geomArray2 = browser.createVrmlFromString(wrl.cylindrical2);

            transform = transArray[0];
            transform2 = transArray2[0];
            material = matArray[0];

            EventInSFNode nodeln = (EventInSFNode) shapeArray[0].getEventIn("appearance");
            nodeln.setValue(appArray[0]);
            nodeln = (EventInSFNode) appArray[0].getEventIn("material");
            nodeln.setValue(material);

            EventInMFNode nodesln = (EventInMFNode) transform.getEventIn("addChild");
            nodesln.setValue(geomArray);
            nodesln.setValue(transform2);

            EventInMFNode nodesln2 = (EventInMFNode) transform2.getEventIn("addChild");
            nodesln2.setValue(geomArray2);

            nodesln.setValue(shapeArray);
            nodesln.setValue(sensArray);
            nodesln.setValue(planeArray);

            set_rotation = (EventInSFRotation) transform.getEventIn("rotation");
            set_scale = (EventInSFVec3f) transform.getEventIn("scale");
            set_translation = (EventInSFVec3f) transform.getEventIn("translation");

            rotation_changed = (EventOutSFRotation) transform.getEventOut("rotation");
        }
    }
}
scale_changed = (EventOutSFVec3f) transform.getEventOut("scale");
translation_changed = (EventOutSFVec3f) transform.getEventOut("translation");
touchTime_changed = (EventOutSFTime) sensArray[0].getEventOut("touchTime");
touchTime_changed.advise(this, null);

browser.addRoute(planeArray[0], "translation_changed",
        transform, "set_translation");

set_rotation2 = (EventInSFRotation) transform2.getEventIn("rotation");
set_scale2 = (EventInSFVec3f) transform2.getEventIn("scale");
set_translation2 = (EventInSFVec3f) transform2.getEventIn("translation");

rotation_changed2 = (EventOutSFRotation) transform2.getEventOut("rotation");
scale_changed2 = (EventOutSFVec3f) transform2.getEventOut("scale");
translation_changed2 = (EventOutSFVec3f) transform2.getEventOut("translation");

System.out.println("inserted element successfully!");
} // end try

catch(InvalidVrmlException e) { System.out.println("Error adding LKPElement: " + e); }
catch(InvalidEventInException e) { System.out.println("Error adding LKPElement: " + e); }
catch(InvalidEventOutException e) { System.out.println("Error adding LKPElement: " + e); }
catch(InvalidRouteException e) { System.out.println("Error adding LKPElement: " + e); }
catch(InvalidNodeException e) { System.out.println("Error adding LKPElement: " + e); }
catch(NullPointerException e) { System.out.println("Error adding LKPElement: " + e); }

} // end constructor

}
B.4 Driver.java

```java
import java.lang.*;
import java.util.Vector;
import VrmlObjects;
//import Connector;

public class Driver {

    private double PI = java.lang.Math.PI);
    private final double engine_radius = 0.3; 

    private Vrml_Objects driving_element, driven_element;
    private Connector connector ;
    private Vector connections;

    Driver(Connector connection_element) {
        connector = connection_element;
    }

    public void animate(float[] displacement) {
        int i, j;

        // translate the current connector
        connector.translate(displacement);
        // get the element that the connector is driving
        driven_element = connector.get_driven_element();
        // get the Vector that says how many other elements the driven element is connected to
        // connections = driven_element.get_connected_toO();
        connections = driven_element.get_connector_array();
        // get the size of that vector
        // need to initialize i somehow
        i = connections.size();

        // translate whatever the driven_element is driving
        for(j=1; j<i; j++) {
            Connector temp ;
            temp = (Connector) connections.elementAt(j);
            temp.translate(displacement);
        }
    }
}
```
B.5 Engine.java

import java.lang.*
import java.awt.event.*;
import java.util.Vector;
import vrml.external.field.*;
import vrml.external.Browser;
import vrml.external.Node;
import vrml.external.exception.*;
import Driver;

public class Engine extends LKPElement implements ActionListener {
    private double PI = java.lang.Math.PI;
    private double theta;
    private double x, y;
    private float displacement[] = new float[3];
    private float offset[] = new float[3];
    private final double engine_radius = 0.3; // engine radius is 0.3 from wrl.java file

    private Driver driver;
    private Control parent;

    public Engine(Control myParent) throws IllegalArgumentException {
        this.super(myParent);
        // parent = myParent;


        myParent.getStartEvent().addActionListener(this);
        myParent.getStopEvent().addActionListener(this);

        element_type = ENGINE;

        System.out.println("In Engine constructor, type = " + element_type);

        try {
            geomArray = browser.createVrmlFromString(wrl.engine);
            geomArray2 = browser.createVrmlFromString(wrl.engine2);

            transform = transArray[0];
            transform2 = transArray2[0];
            material = matArray[0];

            nodeln = (EventInSFNode) shapeArray[0].getEventIn("appearance");
            nodeln.setValue(appArray[0]);
        }
    }
}
nodeIn = (EventInSFNode) appArray[0].getEventIn("material")
nodeIn.setValue(material);

nodeIn = (EventInMFNode) transform.getEventIn("addChild");
nodeIn.setValue(geomArray);
nodeIn.setValue(transArray2);

nodeIn2 = (EventInMFNode) transform2.getEventIn("addChild");
nodeIn2.setValue(geomArray2);

nodeIn.setValue(shapeArray);
nodeIn.setValue(sensArray);
nodeIn.setValue(planeArray);

set_rotation = (EventInSFRotation) transform.getEventIn("rotation");
set_scale = (EventInSFVec3f) transform.getEventIn("scale");
set_translation = (EventInSFVec3f) transform.getEventIn("translation");

rotation_changed = (EventOutSFRotation) transform.getEventOut("rotation");
scale_changed = (EventOutSFVec3f) transform.getEventOut("scale");
translation_changed = (EventOutSFVec3f) transform.getEventOut("translation");

touchTime_changed = (EventOutSFTime) sensArray[0].getEventOut("touchTime");
touchTime_changed.advise(this, null);

browser.addRoute(planeArray[0], "translation_changed",
transform, "set_translation");

set_rotation2 = (EventInSFRotation) transform2.getEventIn("rotation");
set_scale2 = (EventInSFVec3f) transform2.getEventIn("scale");
set_translation2 = (EventInSFVec3f) transform2.getEventIn("translation");

rotation_changed2 = (EventOutSFRotation) transform2.getEventOut("rotation");
scale_changed2 = (EventOutSFVec3f) transform2.getEventOut("scale");
translation_changed2 = (EventOutSFVec3f) transform2.getEventOut("translation");

System.out.println("inserted element successfully!");

} // end constructor

catch(InvalidOsrmlException e) { System.out.println("Error adding LKPElement: " + e); }
catch(InvalidEventInException e) { System.out.println("Error adding LKPElement: " + e); }
catch(InvalidEventOutException e) { System.out.println("Error adding LKPElement: " + e); }
catch(InvalidRouteException e) { System.out.println("Error adding LKPElement: " + e); }
catch(InvalidNodeException e) { System.out.println("Error adding LKPElement: " + e); }
catch(NullPointerException e) { System.out.println("Error adding LKPElement: " + e); }
} // end constructor
public void run() {

    int list_of_elements[];
    float angle;
    float quad_correction[] = new float[3]; // to correct motion based on quadrant
    float[] position = new float[3]; // to correct position base
    float delta = 0.1f;

    list_of_elements = create_list_of_elements();
    // this will not work.. if the connector becomes a vector
    connector = this.get_connector();

    System.out.println("In run");
    try {
        angle = connector.get_angle();
    }
    catch(NullPointerException e){System.out.println("Could not get_angle()" + e);angle=0; }

    position = this.get_element_translation();
    System.out.println("X: ", position[0], " Y: ", position[1], " Z: ", position[2]);

    // if (vrml_engine == null) { System.out.println(Huh? engine is null); }

    switch(connector.get_quadrant()) {
    case 1:
        angle -= Math.PI;
        quad_correction[0] -= engine_radius*Math.cos(angle);
        quad_correction[1] -= engine_radius*Math.sin(angle);
        break;
    case 2:
        angle += Math.PI;
        quad_correction[0] -= engine_radius*Math.cos(angle);
        quad_correction[1] -= engine_radius*Math.sin(angle);
        break;
    case 3:
        angle += Math.PI;
        quad_correction[0] -= engine_radius*Math.cos(angle);
        quad_correction[1] -= engine_radius*Math.sin(angle);
        break;
    case 4:
        angle += Math.PI;
        quad_correction[0] -= engine_radius*Math.cos(angle);
        quad_correction[1] -= engine_radius*Math.sin(angle);
        break;
    default:
    }
for ( x=0, y=0, theta=angle; theta<=angle+2*PI; theta+=delta) {
    x = engine_radius * Math.cos(theta);
    y = engine_radius * Math.sin(theta);

    // need to offset y this because it is
    // offset in the vrml node (see wrl.java)
    displacement[0] = (float) x + offset[0] + quad.correction[0];
    displacement[1] = (float) y + offset[1] + quad.correction[1];
}

System.out.println("Theta: "+ theta + ", X: "+ x + ", Y: "+ y);
try {
    // vrmlengine.set_element_translation(displacement);
    if (connector == null) {
        System.out.println("doh, connector is null");
        break;
    } else {
        driver = new Driver(connector);
        driver.animate(displacement);
    }
} catch (NullPointerException e) {
    System.out.println("Engine could not set translation: "+ e);
}
System.out.println("done running");
}

public void actionPerformed(ActionEvent evt) {
    if (evt.getActionCommand().equals("Start")) {
        System.out.println("got start in engine");
        this.run();
        // this.start();
    }
    if (evt.getActionCommand().equals("Stop")) {
        // this.stop();
    }
} // end actionPerformed

public double get_engine_radius() {
    return engine_radius;
}

public int[] create_list_of_elements() {
    int i;
}
int[] list = null;
Vector elements;

elements = this.get_connected_to();
for(i=0; i<elements.size(); i++) {
    System.out.println("this is connected to: " + ((Integer)elements.elementAt(i)).intValue());
}

return list;
}
B.6 Ground.java

```java
import java.lang.*;
import vrml.external.field.*;
import vrml.external.Browser;
import vrml.external.Node;
import vrml.external.exception.*;

public class Ground extends LKPElement {

    Ground(Control myParent) throws IllegalArgumentException {
        super(myParent);
        element_type = GROUND;
        System.out.println("In Ground constructor, type = " + element_type);
        try {
            geomArray = browser.createVrmlFromstring(wrl.ground);
            transform = transArray[0];
            material = matArray[0];
            EventInSFNode nodeln = (EventInSFNode) shapeArray[0].getEventIn("appearance");
            nodeln.setValue(appArray[0]);
            nodeln = (EventInSFNode) appArray[0].getEventIn("material");
            nodeln.setValue(material);
            EventInMFNode nodesln = (EventInMFNode) transform.getEventIn("addChildren");
            nodesln.setValue(geomArray);
            nodesln.setValue(shapeArray);
            nodesln.setValue(sensArray);
            nodesln.setValue(planeArray);
            set_rotation = (EventInSFRotation) transform.getEventIn("rotation");
            set_scale = (EventInSFVec3f) transform.getEventIn("scale");
            set_translation = (EventInSFVec3f) transform.getEventIn("translation");
            rotation_changed = (EventOutSFRotation) transform.getEventOut("rotation");
            scale_changed = (EventOutSFVec3f) transform.getEventOut("scale");
            translation_changed = (EventOutSFVec3f) transform.getEventOut("translation");
            touchTime_changed = (EventOutSFTime) sensArray[0].getEventOut("touchTime");
            touchTime_changed.advise(this, null);
            browser.addRoute(planeArray[0], "translation_changed",
```
transform, "set_translation";

    //
    System.out.println("inserted element successfully!");
} // end try

catch(InvalidVrmlException e) { System.out.println("Error adding LKPElement: " + e); }
catch(InvalidEventInException e) { System.out.println("Error adding LKPElement: " + e); }
catch(InvalidEventOutException e) { System.out.println("Error adding LKPElement: " + e); }
catch(InvalidRouteException e) { System.out.println("Error adding LKPElement: " + e); }
catch(InvalidNodeException e) { System.out.println("Error adding LKPElement: " + e); }
catch(NullPointerException e) { System.out.println("Error adding LKPElement: " + e); }

} // end constructor
}
public class Helical extends LKPElement {

    Helical(Control myParent) throws IllegalArgumentException {
        this.super(myParent);
        element_type = HELICAL_JOINT;

        System.out.println("In Helical constructor, type = " + element_type);

        try {

            geomArray = browser.createVrmlFromWrl(wrl.helical);
            geomArray2 = browser.createVrmlFromWrl(wrl.helical2);

            transform = transArray[0];
            transform2 = transArray2[0];
            material = matArray[0];

            EventInSFNode nodeln = (EventInSFNode) shapeArray[0].getEventIn("appearance");
            nodeln.setValue(appArray[0]);
            nodeln = (EventInSFNode) appArray[0].getEventIn("material");
            nodeln.setValue(material);

            EventInMFNode nodesln = (EventInMFNode) transform.getEventIn("addChildren");
            nodesln.setValue(geomArray);
            nodesln.setValue(transArray2);

            EventInMFNode nodesln2 = (EventInMFNode) transform2.getEventIn("addChildren");
            nodesln2.setValue(geomArray2);

            nodesln.setValue(shapeArray);
            nodesln.setValue(sensArray);
            nodesln.setValue(planeArray);

            set_rotation = (EventInSFRotation) transform.getEventIn("rotation");
            set_scale = (EventInSFVec3f) transform.getEventIn("scale");
            set_translation = (EventInSFVec3f) transform.getEventIn("translation");

            rotation_changed = (EventOutSFRotation) transform.getEventOut("rotation");
            scale_changed = (EventOutSFVec3f) transform.getEventOut("scale");
        }
    }
}
translation_changed = (EventOutSFVec3f) transform.getEventOut("translation");

touchTime_changed = (EventOutSFTime) sensArray[0].getEventOut("touchTime");
touchTime_changed.advise(this, null);

browser.addRoute(planeArray[0], "translation_changed",
                   transform, "set_translation")

    /////
    set_rotation2 = (EventInSFRotation) transform2.getEventIn("rotation");
    set_scale2 = (EventInSFVec3f) transform2.getEventIn("scale");
    set_translation2 = (EventInSFVec3f) transform2.getEventIn("translation");

    rotation_changed2 = (EventOutSFRotation) transform2.getEventOut("rotation");
    scale_changed2 = (EventOutSFVec3f) transform2.getEventOut("scale");
    translation_changed2 = (EventOutSFVec3f) transform2.getEventOut("translation");

    System.out.println("inserted element successfully!");
} // end try

catch(InvalidVrmlException e) { System.out.println("Error adding LKPElement: " + e); }
catch(InvalidEventInException e) { System.out.println("Error adding LKPElement: " + e); }
catch(InvalidEventOutException e) { System.out.println("Error adding LKPElement: " + e); }
catch(InvalidRouteException e) { System.out.println("Error adding LKPElement: " + e); }
catch(NullPointerException e) { System.out.println("Error adding LKPElement: " + e); }

} // end constructor
B.8 LKPElement.java

// Code for the objects that we’ll be inserting into the vrml world
// Sameer Raheja 9/20/97

import java.lang.*;
import java.util.Vector;
import java.applet.*;

import vrml.external.field.*;
import vrml.external.Browser;
import vrml.external.Node;
import vrml.external.exception.*;

import Engine;
import wrl;

public class LKPElement extends VrmlObjects {

    protected Connector connector = null;

    LKPElement(Control myParent) {
        this.super(myParent);
    }

} // end LKPElement class
public class Prismatic extends LKPElement {

    Prismatic(Control myParent) throws IllegalArgumentException {
        this.super(myParent);

        element_type = PRISMATIC_JOINT;

        System.out.println("In Prismatic constructor, type = " + element_type);

        try {

            geomArray = browser.createVrmlFromString(wrl.prismatic);
            geomArray2 = browser.createVrmlFromString(wrl.prismatic2);

            transform = transArray[0];
            transform2 = transArray2[0];
            material = matArray[0];

            EventInSFNode nodeln = (EventInSFNode) shapeArray[0].getEventIn("appearance");
            nodeln.setValue(appArray[0]);
            nodeln = (EventInSFNode) appArray[0].getEventIn("material");
            nodeln.setValue(material);

            EventInMFNode nodesIn = (EventInMFNode) transform.getEventIn("addChild");
            nodesIn.setValue(geomArray);
            nodesIn.setValue(transArray2);

            EventInMFNode nodesIn2 = (EventInMFNode) transform2.getEventIn("addChild");
            nodesIn2.setValue(geomArray2);

            nodesIn.setValue(shapeArray);
            nodesIn.setValue(sensArray);
            nodesIn.setValue(planeArray);

            set_rotation = (EventInSFRotation) transform.getEventIn("rotation");
            set_scale = (EventInSFVec3f) transform.getEventIn("scale");
            set_translation = (EventInSFVec3f) transform.getEventIn("translation");

            rotation_changed = (EventOutSFRotation) transform.getEventOut("rotation");
            scale_changed = (EventOutSFVec3f) transform.getEventOut("scale");
        }
    }
translation_changed = (EventOutSFVec3f) transform.getEventOut("translation");

touchTime_changed = (EventOutSFTime) sensArray[0].getEventOut("touchTime");
touchTime_changed.advise(this, null);

browser.addRoute(planeArray[0], "translation_changed",
    transform, "set_translation") ;

///
set_rotation2 = (EventInSFRotation) transform2.getEventIn("rotation");
set_scale2 = (EventInSFVec3f) transform2.getEventIn("scale");
set_translation2 = (EventInSFVec3f) transform2.getEventIn("translation");

rotation_changed2 = (EventOutSFRotation) transform2.getEventOut("rotation");
scale_changed2 = (EventOutSFVec3f) transform2.getEventOut("scale");
translation_changed2 = (EventOutSFVec3f) transform2.getEventOut("translation");

///
System.out.println("inserted element successfully!");
}
} // end constructor

catch(InvalidVrmlException e) { System.out.println("Error adding LKPElement: " + e) ; }
catch(InvalidEventInException e) { System.out.println("Error adding LKPElement: " + e) ; }
catch(InvalidEventOutException e) { System.out.println("Error adding LKPElement: " + e) ; }
catch(InvalidRouteException e) { System.out.println("Error adding LKPElement: " + e) ; }
catch(NullPointerException e) { System.out.println("Error adding LKPElement: " + e) ; }
} // end constructor
}
B.10 Revolute.java

import java.lang.*;
import vrml.external.field.*;
import vrml.external.Browser;
import vrml.external.Node;
import vrml.external.exception.*;

public class Revolute extends LKPElement {

    Revolute(Control myParent) throws IllegalArgumentException {
        this.super(myParent);
        element.type = REVOLUTE_JOINT;
        System.out.println("In Revolute constructor, type = " + element.type);

        try {

            geomArray = browser.createVrmlFromString(wrl.revolute);
            geomArray2 = browser.createVrmlFromString(wrl.revolute2);

            transform = transArray[0];
            transform2 = transArray2[0];
            material = matArray[0];

            EventInSFNode nodeln = (EventInSFNode) shapeArray[0].getEventIn("appearance");
            nodeln.setValue(appArray[0]);
            nodeln = (EventInSFNode) appArray[0].getEventIn("material");
            nodeln.setValue(material);

            EventInMFNode nodesIn = (EventInMFNode) transform.getEventIn("addChildren");
            nodesIn.setValue(geomArray);
            nodesIn.setValue(transArray2);

            EventInMFNode nodesIn2 = (EventInMFNode) transform2.getEventIn("addChildren");
            nodesIn2.setValue(geomArray2);

            nodesIn.setValue(shapeArray);
            nodesIn.setValue(sensArray);
            nodesIn.setValue(planeArray);

            set_rotation = (EventInSFRotation) transform.getEventIn("rotation");
            set_scale = (EventInSFVec3f) transform.getEventIn("scale");
            set_translation = (EventInSFVec3f) transform.getEventIn("translation");

            rotation_changed = (EventOutSFRotation) transform.getEventOut("rotation");
            scale_changed = (EventOutSFVec3f) transform.getEventOut("scale");
        }
    }
}
translation_changed = (EventOutSFVec3f) transform.getEventOut("translation");

touchTime_changed = (EventOutSFTime) sensArray[0].getEventOut("touchTime");
touchTime_changed.advise(this, null);

browser.addRoute(planeArray[0], "translation_changed",
                   transform, "set_translation") ;

///
set_rotation2 = (EventInSFRotation) transform2.getEventIn("rotation");
set_scale2 = (EventInSFVec3f) transform2.getEventIn("scale");
set_translation2 = (EventInSFVec3f) transform2.getEventIn("translation");

rotation_changed2 = (EventOutSFRotation) transform2.getEventOut("rotation");
scale_changed2 = (EventOutSFVec3f) transform2.getEventOut("scale");
translation_changed2 = (EventOutSFVec3f) transform2.getEventOut("translation");

///
System.out.println("inserted element successfully!");
} // end try

catch(InvalidVrmlException e) { System.out.println("Error adding LKPElement: " + e) ; }
catch(InvalidEventInException e) { System.out.println("Error adding LKPElement: " + e) ; }
catch(InvalidEventOutException e) { System.out.println("Error adding LKPElement: " + e) ; }
catch(InvalidRouteException e) { System.out.println("Error adding LKPElement: " + e) ; }
catch(InvalidNodeException e) { System.out.println("Error adding LKPElement: " + e) ; }
catch(NullPointerException e) { System.out.println("Error adding LKPElement: " + e) ; }

} // end constructor
public class Spherical extends LKPElement {

Spherical(Control myParent) throws IllegalArgumentException {
    this.super(myParent);
    element_type = SPHERICAL_JOINT;

    System.out.println("In Spherical constructor, type = " + element_type);

    try {
        geomArray = browser.createVrmlFromString(wrl.spherical);
        geomArray2 = browser.createVrmlFromString(wrl.spherical2);

        transform = transArray[0];
        transform2 = transArray2[0];
        material = matArray[0];

        EventInSFNode nodeln = (EventInSFNode) shapeArray[0].getEventIn("appearance");
        nodeln.setValue(appArray[0]);
        nodeln = (EventInSFNode) appArray[0].getEventIn("material");
        nodeln.setValue(material);

        EventInMFNode nodesln = (EventInMFNode) transform.getEventIn("addChildren");
        nodesln.setValue(geomArray);
        nodesln.setValue(transArray2);

        EventInMFNode nodesln2 = (EventInMFNode) transform2.getEventIn("addChildren");
        nodesln2.setValue(geomArray2);

        nodesln.setValue(shapeArray);
        nodesln.setValue(sensArray);
        nodesln.setValue(planeArray);

        set_rotation = (EventInSFIRotation) transform.getEventIn("rotation");
        set_scale = (EventInSFVec3f) transform.getEventIn("scale");
        set_translation = (EventInSFVec3f) transform.getEventIn("translation");

        rotation_changed = (EventOutSFIRotation) transform.getEventOut("rotation");
        scale_changed = (EventOutSFVec3f) transform.getEventOut("scale");
`translation_changed = (EventOutSFVec3f) transform.getEventOut("translation");`

`touchTime_changed = (EventOutSFTime) sensArray[0].getEventOut("touchTime");`

`touchTimechanged.advise(this, null);`

`browser.addRoute(planeArray[0], "translation_changed", transform, "set_translation")`;

```java
set_rotation2 = (EventInSFRotation) transform2.getEventIn("rotation");
set_scale2 = (EventInSFVec3f) transform2.getEventIn("scale");
set_translation2 = (EventInSFVec3f) transform2.getEventIn("translation");
```

```java
rotation_changed2 = (EventOutSFRotation) transform2.getEventOut("rotation");
scale_changed2 = (EventOutSFVec3f) transform2.getEventOut("scale");
translation_changed2 = (EventOutSFVec3f) transform2.getEventOut("translation");
```

```java
// System.out.println("inserted element successfully!");
} // end try
```

```java
catch(InvalidVrmlException e) { System.out.println("Error adding LKPElement: " + e); }
catch(InvalidEventInException e) { System.out.println("Error adding LKPElement: " + e); }
catch(InvalidEventOutException e) { System.out.println("Error adding LKPElement: " + e); }
catch(InvalidRouteException e) { System.out.println("Error adding LKPElement: " + e); }
catch(NullPointerException e) { System.out.println("Error adding LKPElement: " + e); }
```

```
} // end constructor
```
B.12  Vrml_Objects.java

import java.lang.*;
import java.util.Vector;
import vrml.external.field.*;
import vrml.external.Browser;
import vrml.external.Node;
import vrml.external.exception.*;
import Control;

public class Vrml_Objects implements EventOutObserver {
    private Node root;
    private static int object_index = -1;
    private int index;
    private Vector connected_to = new Vector(2,1);

    protected Browser browser;
    protected Control parent;

    protected Node[] transArray = null;
    protected Node[] transArray2 = null;
    protected Node[] geomArray = null;
    protected Node[] geomArray2 = null;
    protected Node[] shapeArray = null;
    protected Node[] shapeArray2 = null;
    protected Node[] sensArray = null;
    protected Node[] planeArray = null;
    protected Node[] matArray = null;
    protected Node[] appArray = null;

    protected EventInSFNode nodeIn = null;
    protected EventInMFNode nodesIn = null;
    protected EventInMFNode nodesIn2 = null;

    protected Node transform;
    protected Node transform2;

    protected Node material;

    protected int element_type;

    // this is bad...
    private Connector connector;
    private Vector connector_array = new Vector();

/ EventIns to modify
protected EventInSFRotation set_rotation;
protected EventInSFVec3f set_scale;
protected EventInSFVec3f set_translation;

protected EventInSFRotation set_rotation2;
protected EventInSFVec3f set_scale2;
protected EventInSFVec3f set_translation2;

// EventOuts to watch for
protected EventOutSFRotation rotation_changed;
protected EventOutSFVec3f scale_changed;
protected EventOutSFVec3f translation_changed;
protected EventOutSFTime touchTime_changed;

protected EventOutSFRotation rotation_changed2;
protected EventOutSFVec3f scale_changed2;
protected EventOutSFVec3f translation_changed2;
protected EventOutSFTime touchTime_changed2;

// set up constants for each joint type
public final static int GROUND = 0;
public final static int CYLINDRICAL_JOINT = 1;
public final static int HELICAL_JOINT = 2;
public final static int PRISMATIC_JOINT = 3;
public final static int REVOLUTE_JOINT = 4;
public final static int SPHERICAL_JOINT = 5;
public final static int ENGINE = 6;

VrmlObjects() { // index = ++object_index; }

VrmlObjects(Control myParent) {
    index = ++object_index;
    System.out.println("here! index=");
    parent = myParent;
    browser = parent.getBrowser();
    try {
        transArray = browser.createVrmlFromString("Transform {}”);
        transArray2 = browser.createVrmlFromString("Transform {}”);
        shapeArray = browser.createVrmlFromString("Shape {}”);
        shapeArray2 = browser.createVrmlFromString("Shape {}”);
        sensArray = browser.createVrmlFromString("TouchSensor {}”);
        planeArray = browser.createVrmlFromString("PlaneSensor {}”);
        matArray = browser.createVrmlFromString("Material {}”);
        appArray = browser.createVrmlFromString("Appearance {}”);
catch(InvalidVrmlException e) { System.out.println("Error adding Vrml_Objects: "+ e); }
catch(InvalidEventInException e) { System.out.println("Error adding Vrml_Objects: "+ e); }
catch(InvalidEventOutException e) { System.out.println("Error adding Vrml_Objects: "+ e); }
catch(InvalidRouteException e) { System.out.println("Error adding Vrml_Objects: "+ e); }
catch(InvalidNodeException e) { System.out.println("Error adding Vrml_Objects: "+ e); }
catch(NullPointerException e) { System.out.println("Error adding Vrml_Objects: "+ e); }
} // end constructor

public int get_index() {
    return index;
}

// callback() for EventOutObserver implementation
public void callback(EventOut event, double time, Object userData) {
    System.out.println("Object Type = "+ element.type);
    parent.makeCurrent(this);
    browser = parent.getBrowser();
} // end callback

// translates the geom2 part of the object
public void set_element_translation(float displacement[]) {
    try {
        set_translation2.setValue(displacement);
    }
    catch (InvalidEventInException e) {
        System.out.println("Error in set_element_translation");
    }
} // end set_element_translation

public void set_element_rotation(float rotation[]) {
    try{
        set_rotation2.setValue(rotation);
    }
    catch (InvalidEventInException e) {
        System.out.println("Error in set_element_rotation");
    }
} // end set_element_rotation

public float[] get_element_translation() {
    float[] val = new float[3];
    val = translation_changed.getValue();
    return val;
}

public Node[] get_transArray() {
}
public void add_arm(String arm) {
// check to see if there is a second element already part of this node
System.out.println("trying to remove old joint");
try {
EventInMFNode removeChildren=(EventInMFNode)transform2.getEventIn("removeChildren");
EventInMFNode addChildren = (EventInMFNode) transform2.getEventIn("addChildren");

if (transArray2 != null) {
    System.out.println(geomArray2[0].getType());
    if (this.get_element_type() == ENGINE) {
        removeChildren.setValue(geomArray2);
    }
} else { System.out.println("transArray2 was null"); }

// geomArray2 = browser.createVrmlFromString(Shape {});
System.out.println(arm);
geomArray2 = browser.createVrmlFromString(arm);
addChildren.setValue(geomArray2);
}

catch(InvalidVrmlException e) {
    System.out.println("Error adding geom2 in add_arm(): " + e); }
catch(InvalidEventInException e) {
    System.out.println("Error adding geom2 in add_arm(): " + e); }
catch(InvalidEventOutException e) {
    System.out.println("Error adding geom2 in add_arm(): " + e); }
catch(InvalidRouteException e) {
    System.out.println("Error adding geom2 in add_arm(): " + e); }
catch(InvalidNodeException e) {
    System.out.println("Error adding in add_arm(): " + e); }
} // end try
catch(NullPointerException e) {
    System.out.println("Error adding in add_arm(): " + e); }
System.out.println("created arm");
} // end add_arm()

public void set_connector(Connector new_connector) {
    connector = new_connector;
    connector_array.addElement(new_connector);
    System.out.println("Angle in set_connector() is: " + connector.get_angle());
}

public Connector get_connector() {
    return connector;
}

public Vector get_connector_array() {
    return connector_array;
}

public void set_connected_to(int x) {
    // ugh, had to use the integer object for the array
    connected_to.addElement(new Integer(x));
}

public Vector get_connected_to() {
    return connected_to;
}

} // end Vrml_Objects class
B.13 simulate.html

<html>
<head>
<title>
Kinematic Simulation
</title>
</head>

<body>
<center>
<embed src="root.wrl"
border="0" height="500" width="500"
SGI_IMAGEQUALITY="Best Quality">
<br>
<applet code="Control.java"
height="100" width="400"
mayscript >
</applet>
</center>
</body>

</html>
Bibliography


