The Addition Of The Haptic Modality
To The Virtual Reality Modeling Language

by

Evan F. Wies

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Electrical Engineering and Computer Science
and Master of Engineering in Electrical Engineering and Computer Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 29, 1997

Copyright 1997 Evan F. Wies. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce and
distribute publicly paper and electronic copies of this thesis
and to grant others the right to do so.

Author

Department of Electrical Engineering and Computer Science
August 29, 1997

Certified by

Nathaniel Durlach
Thesis Supervisor

Accepted by

Arthur C. Smith
Chairman, Department Committee on Graduate Theses

August 29, 1997
The Addition Of The Haptic Modality
To The Virtual Reality Modeling Language
by
Evan F. Wies

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Electrical Engineering and Computer Science
and Master of Engineering in Electrical Engineering and Computer Science

Abstract

The creation of virtual environments (VE's) that incorporate sight, sound, and touch is a difficult task. In particular, the greatest challenge is presented by the haptic (touch) modality. This thesis introduces a computer haptics system which facilitates the development of these multi-modal virtual environments.

The computer haptics system serves the needs of two groups of people: multi-modal application developers and haptics technology researchers. This system incorporates the industry-standard Virtual Reality Modeling Language version 2.0 (VRML). VRML is extended to accommodate the haptic modality, allowing the rapid-prototyping and development of multi-modal applications. With VRML, programmers can develop multi-modal applications in minutes rather than weeks.

The computer haptics system has a modular design. Different aspects of the system can be interchanged to suit the needs of the user. Previously, haptics researchers had to build an entire system from the ground up. With the system introduced in this thesis, each module can be investigated separately, allowing haptics researchers to focus on their interests. In addition, this system offers haptics researchers a venue for distributing their technology to end-users.

This document describes how the haptic modality is added to VRML and details the design of the modular computer haptics system.

Thesis Supervisor: Nathaniel I. Durlach
Title: Senior Research Scientist
M.I.T. Research Laboratory of Electronics
Acknowledgments

It is impossible to express in prose my appreciation for the help and guidance that many people have given to me.

I must thank my thesis advisor Nat Durlach for keeping a watchful eye, for always giving me a reassuring smile, and for teaching me a management style that emphasizes integrity.

Then there is my "boss", my colleague, and friend Tom Wiegand. He gave me my first UROP (undergraduate research job) and led me down a whimsical path of interesting research. Tom and Jonathon Pfautz always entertained me with their "intellectuals-raised-in-backwoods-Pennsylvania" humor, which I continue to have nightmares about.

Kenneth Salisbury and Mandayan Srinivasan gave me both financial and intellectual support. They gave me the opportunity to work at three labs at once, which definitely broadened my horizons.

I have had great discussions and collaborations with my "haptics homeys" — Walt Aviles, Cagatay Basdogan, David Brock, Alexandra Hou, Craig Latimer, and Trudy Liu. Thank you for sharing your outlooks and ideas with me.

My academic advisor, G.W. Pratt, and the noted savant, Anne Hunter, helped me throughout my entire MIT education. Their dedication to their students is inspiring.
My life has been greatly enriched by all my friends, teammates, players, teachers, and coaches. I hold you all dear to my heart and will remember you always.

My parents, Stephen and Judith Wies, and my brother, Darren Wies, have always been there for me. Thank you does no justice to your love and service.

Finally, thanks to my best friend, Ann Torres. I do not know what I would do without you.

This thesis was supported by a contract with NAWC/TSD and ONR on the Development of Haptic Interfaces (#N61339-96-K-0002).

Here is a puzzle. The answer describes my entire MIT experience. Attempts may be submitted to evanw@alum.mit.edu.

1. e4   e5
2. Nf3   Nc6
3. Bc4   Bc5
4. b4
## Contents

1 Introduction .......................................................................................................................... 6

2 Background ........................................................................................................................... 8

   2.1 The Virtual Reality Modeling Language ...................................................................... 8

   2.2 The Wide World Of Haptics ......................................................................................... 11

      2.2.1 Machine Haptics ................................................................................................. 11

3 Representing Haptics In VRML ......................................................................................... 15

   3.1 Specifying Haptic Properties ...................................................................................... 15

   3.2 Representing the Haptic Interface Device ................................................................... 20

   3.3 Communicating Haptic Interactions .......................................................................... 22

4 System Architecture ........................................................................................................... 24

   4.1 Overview ...................................................................................................................... 24

   4.2 VRML-Haptics API ..................................................................................................... 29

      4.2.1 A Brief History of Haptics API’s ....................................................................... 30

      4.2.2 Technical Overview ............................................................................................. 32

   4.3 Haptic Renderer ........................................................................................................... 35

   4.4 Haptics Client ............................................................................................................... 37

   4.5 Architecture Extensions .............................................................................................. 38

5 Conclusion ............................................................................................................................ 39

6 References ........................................................................................................................... 40

A VRML-Haptics Prototypes .................................................................................................. 43
1 Introduction

The creation of virtual environments (VE's) that incorporate sight, sound, and touch is a difficult task. In particular, the greatest challenge is presented by the haptic (touch) modality. The simplest of these multi-modal VE's can take one hundred person-hours to develop. The problem lies in the fact that no cohesive computer haptics system currently exists. The solution lies in the creation of a systems integration framework and rapid-prototyping tools.

This solution is similar to previous, successful efforts in the computer graphics community. Many years ago the computer graphics community was facing the same challenges that the haptics community is now facing. To remedy their problems, OpenGL was created as a common software platform for three-dimensional graphics programming [12]. Now, new graphics hardware devices and software products merely need to support the OpenGL standard in order to be widely accepted. The advent of this standardization aided the boom of the graphics industry over the last decade. More recently, the Virtual Reality Modeling Language (VRML) was created as a vehicle for the rapid-prototyping and Internet distribution of three-dimensional (3D) graphical VE's [13]. VRML brought 3D graphics technology to the mass market.

By creating a common platform for haptics development, similar productivity enhancements can be achieved in the haptics domain. Virtual environment application developers, haptic rendering developers, and haptic
interface device manufacturers will all directly benefit from this platform. Application developers will have a common development platform with a wide array of tools and support; furthermore, this platform can be implemented on a wide variety of computer types. Renderer developers and device developers will benefit in similar ways. Since there is a common object representation and common hardware interface, each group will be able to focus on their specialty. In addition, it will not be necessary to develop "home-grown" tools, since there will exist a standard platform upon which to test, improve, and apply their technology. This will greatly decrease the length of development cycles. This common platform will prove to be a boon to the haptics industry because applications and technology can be created and improved with greater speed and efficiency.

This document describes a modular computer haptics system which can serve as a basis for the consolidation of the haptics community. This system uses the Virtual Reality Modeling Language version 2.0 (VRML 2.0, or simply VRML) to allow the rapid prototyping and development of multi-modal virtual environments. The next chapter describes VRML and its significance to the Virtual Reality community. Chapter 3 details how haptics is incorporated into VRML 2.0. Chapter 4 describes the computer haptics system framework upon which the VRML-Haptics integration is implemented.
2 Background

The following sections provide some background and history about VRML and haptics. Subsequent chapters delve into further technical detail.

2.1 The Virtual Reality Modeling Language

In 1994, Gavin Bell, Anthony Parisi, and Mark Pesce sought a way to share three-dimensional (3D) information over the suddenly popular World-Wide Web. To achieve this goal, they created the Virtual Reality Modeling Language (VRML), a description language for three-dimensional objects and worlds.

Version 1.0 of VRML was based on Inventor, Silicon Graphics’ 3D modeling and rendering package. By using an existing graphics framework, they were able to easily create VRML browsers, software which enables the viewing of these three-dimensional worlds, particularly over the Internet. Unfortunately, these worlds were static — one could only navigate through the environment, they could not interact with it.

VRML quickly became a widely accepted standard. This was due to its simplicity, its price (browsers were free), the public’s fascination with 3D graphics, and the explosive growth of the Internet. This acceptance combined with VRML 1.0’s shortcomings translated into a clear need to improve VRML 1.0. This need was filled through the help of ‘Industry’¹, who came onto the scene.

¹ ‘Industry’ refers to the entities who could foresee the benefits and profitability of a better VE description language.
with VRML’s popularity. The VRML Architecture Group (VAG), an ISOaffiliated standards body responsible for maintaining and improving the VRML specification, created VRML 2.0 through a Request For Proposal process. The victorious proposal was submitted by Silicon Graphics. This proposal used a framework similar to VRML 1.0, but it allowed object behaviors to be specified (i.e. was dynamic), was designed to be extensible, and included support for sound. The VRML 2.0 specification was finalized by the VAG in August 1996.

In 1997, a nonprofit corporation called the VRML Consortium was created with the mission of fostering and evangelizing VRML as the open standard for 3D multimedia and shared virtual worlds on the Internet [23]. The VAG was dissolved and its responsibilities were assumed by the VRML Consortium’s VRML Review Board (VRB). In April 1997, the VRB released the next iteration of the VRML standard, VRML97, which makes small technical revisions to VRML 2.0.

Despite the improvements to VRML 2.0, it still has several shortcomings. Currently, it does not inherently allow environments to be shared, a critical feature for many VE applications. Also, physically-based modeling in VRML requires inordinate amounts of behavior specification and programming; a better mechanism must be incorporated. These issues and many others are being investigated by the VRB and other interested parties. Future versions of VRML will include more powerful features.

The advent of VRML 2.0 will be significant to the VE industry and research communities. First of all, it has a scalable design: desktop PC users and
high-end 3D graphics workstation users can both use it (although a user’s experience will be of higher quality using the latter). Secondly, VRML is a widely accepted, open standard. Instead of using proprietary VE systems, developers can use VRML and thus offer their systems to a wider audience. Finally, VRML is extremely flexible. One can use it for anything ranging from rapid-prototyping, to corporate presentations, to the development of highly complex and detailed VE systems.

Theoretically, the popularity and extensibility of VRML make it a prime platform upon which to add haptics. The unfortunate reality is that the VRML platform suffers from serious performance problems. Even on workstation-class computers with graphics acceleration, most VRML worlds suffer from immersion-disrupting latencies. Fortunately, these performance problems are not inherent to VRML itself, but rather to VRML browser implementations. High quality multi-modal interaction in VRML will be hard to achieve unless VRML browser software improves. A proprietary VE application tool such as Sense8’s WorldToolKit and Division’s dVise would be much better suited than VRML for serious virtual environment simulation. The situation will improve as computers become faster and VRML software becomes more efficient. Despite these shortcomings, VRML will suffice as a platform for the rapid-prototyping of multi-modal virtual environments.
2.2 The Wide World Of Haptics

Haptic refers to manual sensing and manipulation. The incorporation of haptics into human-computer interfaces allows the simulation of tasks that are usually performed using hands in the real world [18]. For example, a medical simulation that includes haptics would allow a doctor to cut with a virtual scalpel and apply a virtual suture. This task would be difficult to perform properly and realistically without force-feedback.

The world of haptics can be separated into two broad categories: machine haptics and human haptics [11]. Machine haptics refers to the design of haptic interface devices and software architectures for their control. Human haptics involves psychophysical and physiological research on human perception and performance. These categories are highly interrelated. Research in human haptics establishes design constraints for machine haptics. Conversely, research in machine haptics fuels new investigations in the area of human haptics.

This thesis focuses specifically on software architectures for machine haptics. The following subsection offers some background in machine haptics technology.

2.2.1 Machine Haptics

There are two primary areas in machine haptics: haptic interface devices and haptic rendering software.

The haptic interface device is the hardware that performs the magic in
“touching” virtual objects. Haptic interface devices are unique in that they are both an input device and an output device. These devices read the user’s position and send it to the host computer. These devices also stimulate the user’s cutaneous, proprioceptive, and kinesthetic senses; generally this stimulation is either force-based or tactile. When the user’s position and the stimulation are properly correlated, the user will perceive the existence of a virtual object.

Haptic renderers take virtual environment models and allow users to haptically interact with those models. Rendering is the process of converting an abstract computer model into a form which a human can understand. For instance, a graphic renderer will take an abstract representation of an object and create a bitmap for display on a computer monitor (to allow a human to see that representation). Similarly, haptic renderers take models and device information and then “displays” those models on the haptic interface device. If the device employed force-feedback, the display would be a force profile corresponding to the contours of the model. Haptic rendering is a hot research topic [2, 5, 9, 21, 22, 27].

The haptics research community is in its infancy. A particular group may investigate all aspects of haptics simultaneously — psychophysics, devices, representations, rendering methods, and applications. Since most laboratories perform research that entirely overlaps, efforts are often duplicated. Of the twenty-five papers published in the Proceedings of the First PHANToM Users Group Workshop in 1996, eleven described similar proprietary client-server
architectures for computer haptics [15]. In general, these architectures were a means to an end; the researcher created the architecture to build a specific multi-modal application. If a framework for these means already existed, their redundant work would be unnecessary.

Prior to 1993, the redundant work revolved around the design of haptic interface devices rather than client-server architectures. This situation was greatly improved with the advent of a decent commercial haptics device, SensAble Technologies’ PHANToM [17]. The PHANToM relieved a great burden from groups whose core competency was not devices, freeing them to focus on their strengths. In the past, the Massachusetts Institute of Technology (MIT) Laboratory for Human and Machine Haptics (TouchLab) created devices to explore the psychophysics of touch. The arrival of the PHANToM gave the TouchLab the freedom to focus mainly on their interests of psychophysics and rendering techniques. Although they may still delve into the realm of device creation, the PHANToM gives them a well-supported and popular platform upon which to do their work. This phenomena seems to be occurring in many labs. The PHANToM also opened up the field of haptics to those in other disciplines. For instance, the holography researchers at the MIT Media Laboratory have created a virtual environment using haptics in conjunction with real-time holograms.

This thesis project seeks to make a similar contribution in the realm of multi-modal application development. This work builds upon our first-hand
experience creating multi-modal applications and on similar work by colleagues [25, 10, 27]. During their pursuits, many of the issues faced in this project were addressed. However, their aim was application-specific and focused on low-level representations such as springs. Instead, this project caters to a general-purpose development platform, VRML, and deals with high-level representations such as complex polyhedra and Euclidean primitives.
3 Representing Haptics In VRML

In the current VRML specification (version 2.0), no provisions are made to incorporate haptics in a virtual environment. However, VRML has mechanisms which allow it to be extended by third parties\(^2\). It is through these extension mechanisms that haptics can be incorporated into VRML.

To successfully add haptics to VRML, three fundamental problems must be addressed. First, a method for specifying the haptic properties (compliance, texture, etc.) of VE objects needs to exist. Next, a virtual representation of the haptic interface device must be placed in the VRML world. Finally, interactions between the haptic interface device and VE objects must be communicated. Once these three problems are addressed, any VRML world can incorporate the haptic modality.

The following sections address each of these problems by drawing upon the parallels between haptics and graphics. Since the VRML specification caters to three-dimensional computer graphics, simple solutions to these haptics problems can be found by taking advantage of their computer graphics analogues.

3.1 Specifying Haptic Properties

VRML worlds are specified with nodes — pieces of information which

\(^2\) Sometimes these improvements are incorporated into the official VRML specification.
describe the world. A node has a name and parameters which distinguish it from other nodes. These parameters are called fields and can be nodes themselves. The following example describes a shiny cylinder:

```plaintext
Shape {
    geometry Cylinder {
        height 2.0
        radius 1.5
    }
    appearance Appearance {
        material Material {
            shininess 0.8
        }
    }
}
```

This example contains four different node types: Shape, Appearance, Material, and Geometry. The Shape node describes a virtual object and thus contains the parameters of appearance and geometry. The Appearance node (which fills the appearance parameter) specifies the visual properties of the shape by defining the Material and graphical Texture nodes. The fields of the Material node, such as shininess, determine the way light reflects off an object to create color. When fields are not specified, they become set to a default value; the shape in the example has no texture, so the Texture node was omitted.

The geometry parameter describes the physical geometry of the object; VRML includes many types of Geometry nodes ranging from Euclidean primitives (Cylinder, Sphere, Cone, and Box) to complex polyhedra (IndexedFaceSet).

Currently, VRML does not include any nodes which describe the haptic properties of objects. To be able to include the haptic modality in a VRML world, these haptic properties must be specified. VRML 2.0 has a mechanism called
prototyping which allows the creation of new node types, called *prototypes*.

Using this mechanism, a node describing haptic properties can be added to VRML. Thus, a shape can now be described by its geometry, its appearance, and its *feel*.

Here is the prototype of the new haptic property node, named **Feel**:

```
PROTO Feel [  
exposedField MFNode surfaceModel  
  field SFNode proxyGeometry  
}  
  # Implementation omitted  
}
```

The significance of these haptic properties will be discussed shortly, but first there will be an explanation of the VRML prototyping syntax. The **PROTO** keyword declares that **Feel** is a new node type. The **exposedField** keyword means that the proceeding parameter can be changed by other nodes while the **field** keyword means that the proceeding parameter cannot be changed by other nodes. The **SFNode** keyword means that the proceeding parameter is a type of node while the **MFNode** keyword means that the proceeding parameter is an array of nodes. A prototype is always followed by its implementation in VRML, which usually includes some Java or JavaScript code; in this document, the prototype implementations are omitted for clarity. Since **surfaceModel** and **proxyGeometry** are new nodes themselves, they each must have a prototype declared. A complete list of node prototypes for authoring multi-modal VRML worlds is in Appendix A.

The **Feel** node allows a content developer to assign haptic properties to
objects. The proxyGeometry parameter specifies a geometry to use for the haptic rendering of the object; if omitted, the graphical geometry will be used. The surfaceModel parameter accepts an array of nodes which specify models of the object’s surface. These could include compliance models, texture maps, vibration models, and heat models. Currently, only spring-damper compliance models are supported. As haptics research continues and better representations for these models are found, then other models can be incorporated into VRML.

To represent a spring-damper compliance model, the SDCompliance prototype was created:

```proto
PROTO SDCompliance [  
  exposedField SFFloat SpringK # Spring Constant  
  exposedField SFFloat DampingK # Damping Constant  
  exposedField SFFloat StaticF # Static Friction Const  
  exposedField SFFloat DynamicF # Dynamic Friction Const  
] {  
  # No implementation needed  
}
```

The four properties of the SDCompliance node are a minimal, yet useful set of parameters used in force-feedback rendering of surfaces. This set was proposed by David Brock in his “Proposal for Multi-Modal Sensory Extensions to VRML” [3].

While the Feel node allows one to specify haptic properties, the MultiModalShape node is needed to associate those properties with an object. The MultiModalShape node is identical to the built-in Shape node, but has two additional parameters:
The `feel` parameter allows the user to attach a `Feel` node to the object. The `mass` parameter allows a user to feel that object’s weight; `mass` is not incorporated into the `Feel` node because it is a property which is not specific to the haptic modality and might be used for other purposes. The following example demonstrates how easy it is to describe multi-modal objects in VRML using the `Feel` and `MultiModalShape` nodes:

```
MultiModalShape {
  geometry Cylinder {
    height 2.0
    radius 1.5
  }
  appearance Appearance {
    material Material {
    }
  }
  feel Feel {
    surfaceModel SDCompliance {
    }
    }
}
```

This is the cylinder from the first example, but this time it is multi-modal. It has a `Feel` node which specifies the default spring-damper compliance model for haptic rendering.

The set of haptics representations is also needs methods of grouping, positioning, and orienting multi-modal VRML objects. VRML provides a node called `Transform` which allows the grouping, translation, and rotation of objects [24]. Because of implementation restrictions, a special type of `Transform` node
is required to manipulate MultiModalShapes. Called MultiModalTransform, this node is identical to the Transform node, except that it has extra internal functionality. The impact of this functionality is transparent to the end user.

3.2 Representing the Haptic Interface Device

Now that objects have haptic properties, the next task is to actually place a virtual representation of a haptic interface device into the VRML world. Without this, the user would not be able to see where the device is in the virtual environment, nor could the device actually interact with any objects in the virtual environment. A virtual representation of the haptic interface device can be created through the prototyping mechanism. There are four primary characteristics of the haptic interface that need to be specified in the prototype:

- It must have a unique identity to allow multiple devices.
- It must have a shape (which gives it a geometry and appearance).
- It must have a position and orientation in the virtual environment.
- It must be able to interact with the virtual environment.

The following is a prototype that fulfills these needs:

```protodef
PROTO HapticDevice {  
    field SFString hdName      # Unique identity
    field SFNode hdShape      # It’s virtual shape
    exposedField SFVec3f hdPosition     # It’s position
    exposedField SFRotation hdOrientation   # It’s orientation
    }  
    # Implementation omitted
}
```

This prototype sufficiently describes a haptic device in a virtual environment.
environment. hdName gives it a unique identity; hdShape gives it a geometry and appearance (and possibly feel); hdPosition and hdOrientation reflect its movement in the environment. The final criterion is satisfied by the HDTouchSensor, a VRML prototype which registers the HapticDevice interactions with the virtual environment and is described in the next section.

There are a few new VRML keywords in this prototype. The SFString keyword declares a parameter to be a string. The SFVec3f keyword declares the proceeding parameter to be a three-dimensional vector. The SFRotation keyword declares the proceeding parameter to be a rotation field. To instantiate the HapticDevice, one must only declare its name and shape (i.e., its geometry, appearance, and possibly feel). The VRML browser and the prototype’s implementation will take care of the rest.

The basic idea behind the implementation of the HapticDevice prototype is as follows. A program external to the VRML browser (called the Haptic Renderer, which is detailed Chapter 4) tracks the haptic interface device and updates the Using the Feel and MultiModalShape nodes and hdOrientation fields of HapticDevice. Internal to HapticDevice is a Transform node which translates and rotates the HapticDevice’s shape. By updating the Transform’s fields, the HapticDevice undergoes a spatial transformation, reflecting the physical haptic interface device’s state.

By inserting a HapticDevice node in a VRML world, a virtual representation of the physical haptic interface device is created. This allows the user to see the device in the virtual world. The next section describes how to
make the virtual device affect the environment.

3.3 Communicating Haptic Interactions

The final challenge is to make the haptic interface device interact with the virtual environment. There needs to be a way to change the environment depending on what the user does with the HapticDevice. This is achieved using a special node called a HDTouchSensor in conjunction with VRML’s message passing mechanisms.

To allow objects to communicate with each other, VRML uses a message passing mechanism [24]. Objects can generate events, messages containing a data value, which are passed to other objects via routes, explicitly defined connections between objects. Routes link an eventOut to an eventIn; these can be thought of as object outputs and inputs; an exposedField is implicitly both an eventIn and eventOut.

An example of using VRML’s message passing is to connect a Switch to a Light (both are fictitious VRML nodes). By routing a Switch’s “enabled” exposedField to a Light’s “on” exposedField, an event will be passed from Switch to Light whenever “enabled” changes. Using this route, it is possible to turn the light on and off by enabling and disabling the Switch.

A TouchSensor is a VRML node which generates events when a user interacts with a geometry using the mouse pointer (the virtual representation of a mouse). To function, a TouchSensor must be grouped (using the Group or
Transform nodes) with some geometry; the events generated reflect the mouse clicking and moving over the geometry. This allows a VRML world builder to program worlds that respond to the mouse’s interaction with this geometry. For example, the Switch described above can be grouped with a TouchSensor. If we route this TouchSensor’s isActive eventOut to the Switch’s “enabled” exposedField, the Switch will become enabled whenever it is clicked. Since the Switch’s “enabled” field is routed to the Light’s “on” field, clicking the Switch effectively turns on the Light. By properly routing events, the VRML world can become dynamically influenced by the user through the TouchSensor. The VRML specification fully describes a TouchSensor and its capabilities [24].

The HDTouchSensor is a new VRML prototype that allows the HapticDevice to dynamically change the VRML world. The HDTouchSensor encapsulates much of the functionality of a TouchSensor (HDTouchSensor stands for Haptic Device Touchsensor). When the virtual haptic device comes into contact with geometry that is grouped with a HDTouchSensor, events are generated which can be passed to other VRML objects. Those objects can then interpret and act upon those events. Beyond normal TouchSensor functionality, the HDTouchSensor can also generate events which track the device’s orientation and the amount of pressure being applied to an object. Grouping together a HDTouchSensor and a MultiModalShape in a MultiModalTransform enables fully dynamic multimodal interaction within a VRML world. The VRML prototype for the HDTouchSensor is given in Appendix A.
4 System Architecture

The system architecture described in this chapter seeks to partition the computer haptics system into independent, modular parts. Each partition can be investigated separately, allowing specialists to focus on their interests. They will not have the burden of developing every component and can benefit from advancements made by their colleagues. This system architecture will increase research efficiency and allow synergistic collaboration between laboratories, accelerating the growth of the haptics community.

4.1 Overview

The computer haptics system architecture separates the functionality into four distinct layers. Each layer can be implemented independently. Optimally, any two implementations of the same layer could be interchanged, so that the system can be modular. The crucial design rule when maintaining modularity is that each layer need only understand how to interface with the layer above and below it [20]. These interfaces are unimportant when the system is implemented as one monolithic unit. However, to attain the goal of modularity, each layer must remain a “black box” with a specified interface. This section addresses these issues and describes each layer in detail. This overall system architecture is

3 A “black box” is a unit with a specified functionality and interface, but whose contents are unknown. The internal definition of a black box is not available to be used or played with in any fashion. When the interface of the black box is adhered to, the black box is guaranteed to function properly no matter how it is implemented.
illustrated in Figure 4.1.

At the lowest layer of the system is the physical Haptic Interface Device. The device gathers information about the user's position and offers it to the Haptic Renderer (Figure 4.2). It then receives "commands" from the haptic renderer and stimulates the user's sense of touch. The device is generic — it may be a glove with vibrotactile stimulators, a point-interaction force-feedback arm, a force-feedback joystick, or some other interface device. To the overall system, the type of device is unimportant; the only critical factor is that the haptic renderer one level above it knows how to deal with it. This frees device researchers to build their equipment without having to worry about rendering technology.
Above the Haptic Interface Device is the Haptic Renderer. The Haptic Renderer accepts virtual environment models and state information through the VRML-Haptics Application Programmer’s Interface (API) and retrieves user position from the Haptic Interface Device (Figure 4.3). It processes this information and then issues corresponding commands to the Haptic Interface Device. Upper layers can also request information through the VRML-Haptics API; for example, an application can ask the renderer to tell it what objects are being ‘touched’. A particular Haptic Renderer must be tied with a particular type of device. For instance, a point-interaction force-feedback device must interface with a point-interaction force-feedback Haptic Renderer; the commands that this renderer issues would be force commands. Similarly, a vibrotactile display would require its own type of Haptic Renderer. However, the same Haptic Renderer can be used for the same type of device. A particular point-interaction force-feedback Haptic Renderer could work with all point-interaction force-feedback devices (providing that the devices have the same interface).
The layer above the Haptic Renderer is the Haptics Client. The Haptics Client takes models and state information from the application and gives it to the Haptic Renderer (Figure 4.4). As with all the layers, the Haptics Client and the Haptic Renderer communicate with each other through a specified interface. In this case, the interface is the VRML-Haptics API. The power of the layered architecture lies in the fact that the Haptics Client does not need to know what type of Haptic Interface Device the Haptic Renderer conforms to. The VRML-Haptics API is an interface only between the Haptics Client and the Haptic Renderer. The type of device is insulated from the Haptics Client, allowing the end user to select any Haptic Renderer/Haptic Interface Device pair that suits their particular need.
The topmost layer is the application itself, which manages all aspects of the virtual environment and uses the Haptics Client as its proxy to the rest of the haptics system (Figure 4.5). In this system, the application is a VRML browser since the goal of the project is to use VRML as a rapid-prototyping tool. However, the this system framework could be applied to any application.

When this framework is first established and utilized, the interfaces between these layers must be home-grown. For instance, there is no standard way for a point-interaction force-feedback arm to communicate with a haptic
Currently all haptic renderers are tied to a specific device, such as the PHANToM. However, the hardware interface for a PHANToM is published. If another device were built with an identical interface, it would work with a PHANToM-specific haptic renderer. Early commercial devices may use this technique to provide compatibility with existing software. Eventually, this interface will become obsolete and another will replace it; at this time, the haptics industry may be at a point where this interface will be created by a standards committee, just as graphics interfaces are.

The only interface fully described in this chapter is the VRML-Haptics API which lies between the Haptics Client and the Haptic Renderer. The development of the other interfaces are left to those who are more intimate with their layer (such as devices manufacturers and multi-modal application developers). A good example of an interface between a physical haptic device and a haptic renderer is the device driver used in University of North Carolina (UNC) Chapel Hill's Armlib [10]. The following sections describe the upper-level, more software-oriented layers of this computer haptics system in technical detail.

4.2 VRML-Haptics API

The VRML-Haptics Application Programmer's Interface (API) is an attempt to establish a standard way for applications to interface with haptic renderers. As its name suggests, the VRML-Haptics API is modeled after the
semantics of VRML. This is significant because VRML is a popular intermediate representation used by many 3D application developers. The parallelism between the VRML-Haptics API and VRML not only simplifies the integration of a haptic device with a VRML browser, but it also offers integration with a wide variety of other applications.

4.2.1 A Brief History of Haptics API's

Many haptics API's preceded the VRML-Haptics API. Both CyberNet Systems and Immersion Corporation offer programming libraries for their two degree-of-freedom force-feedback devices [4, 7]. These libraries focus on low-level force-effects (vibrations and force vectors) rather than high-level shapes. The use of these libraries is generally restricted to video game sensations, such as joystick jolts elicited when a machine gun is fired. Anything more sophisticated would require the programmer to “roll their own” haptic renderer, which is truly a non-trivial task.

The first high-level haptics programming tool was the MIT MAGIC Toolkit, which combined graphics and haptics in its own proprietary system. This toolkit offered several primitives to place in an environment. The graphics were two-dimensional while the haptics were three-dimensional. It was used as a platform for some interesting multi-modal psychophysical experiments [6].

The haptics programming tools which proceeded the MAGIC Toolkit decided not to take an integrated multi-modal approach. Instead, these tools
focused solely on haptic rendering, but modeled themselves after graphics packages. Many programmers already utilized these graphics packages in their applications, so incorporating a similar haptics package would facilitate multi-modal development. The first commercial high-level haptics API was SensAble Technologies' GHOST which was modeled after OpenInventor [1]. The most recent high-level haptics programming API is the Stanford-Interval HL library, modeled after OpenGL [14].

The advantage of the VRML-Haptics API over GHOST and HL is its simplicity. The disadvantage of the VRML-Haptics API is its limited functionality. A primary reason for the creation of VRML was to simplify graphics programming, opening it up to a wider audience. VRML world builders only have to focus on their content; browsers would take care of the nuts-and-bolts graphics operations. Similarly, the VRML-Haptics API allows programmers to focus on their multi-modal content, not on the specifics of rendering. The price of this simplicity is that programmers cannot use a specific renderer's more sophisticated features. For example, GHOST allows you to modify polyhedral geometry once it is created; this is impossible in the VRML-Haptics API (as well as in VRML itself).

The VRML-Haptics API is not tied to any particular rendering technology. GHOST and HL are both programming front-ends to point-interaction force-feedback renderers for a specific force-feedback device, the PHANToM. The VRML-Haptics API only understands geometries and their 'feel'. The simplicity of the VRML-Haptics API makes it easy to "wrap it around" other API's. For
example, the haptic renderer in this project was implemented using GHOST; this is discussed thoroughly in the next subsection. Another haptic renderer can be written utilizing a vibrotactile rendering engine. The system operator has the freedom to select the haptic renderer which best fits their need.

4.2.2 Technical Overview

The VRML-Haptics API is a thin layer between two Java programs — a Haptics Client and a Haptics Renderer. The client and server operate together using Java’s Remote Method Invocation (RMI)\(^4\) [8]. RMI allows two Java objects to communicate. The communication protocols are abstracted from the programmer, allowing them to focus on the design of the client and the server responsibilities. The client and server may be on separate computers or on the same computer. This gives great flexibility to the user of the VRML-Haptics system since it inherently enables distributed, multi-modal virtual environment interaction. The VRML browser (with Haptics Client) can be on a Silicon Graphics machine and the haptic renderer can be placed on a personal computer (PC); the underlying system will make them work together properly.

Java was chosen as the implementation language because almost every VRML 2.0 browser, World-Wide Web (WWW) browser, and computer system supports Java. In addition, wonderful packages such as RMI are available for

\(^4\) Originally, the Common Object Request Broker Architecture (CORBA) was going to be used for client-server interactions [15]. CORBA allows any two objects, each written in any programming language, to interoperate. RMI only allows two Java objects to interoperate. Unfortunately, CORBA is not ready-for-prime-time in terms of performance and availability, whereas RMI is fast and everywhere.
Java. The performance problems inherent in a Java implementation is a non-issue in this case because the VRML-Haptics API is extremely lightweight. It simply passes and massages data. The actual work is eventually performed by native code (non-Java entities such as the haptic renderer and the VRML browser), so the overhead of VRML-Haptics API Java-based mechanisms are low.

An important issue to be addressed in the foundation of a haptics API is where to place the VE's dynamics (the control of the movements of objects). Applications without haptics maintain all aspects their VE's dynamics; when a box moves in the environment, it is the application that decided it should move. The situation becomes more complicated when haptics is introduced. In a multi-modal VE, users want to interact with objects. When a user pushes against a virtual box, it should move. The fundamental question that arises is "Should the haptic renderer or the application move the box?".

A haptic renderer could calculate the physics of the situation, render it properly, and inform the application that the box has moved. This is good because the haptic renderer has an intimate relationship with the user; it is giving them stimulation and reading their position. The renderer will supply highly realistic experience. However, this scenario is bad because the application is losing its autonomy of the environment. The haptic renderer has no information about the environment besides what it is told by the application. But, the application handles multiple entities affecting the box, such as its programmed
behavior and other non-haptic effects. Thus, there may be a conflict between the haptic renderer and the application.

On the other hand, the haptic renderer could simply inform the application that the user is applying a force to the box and let the application handle the rest. The application would take this information and all other factors into consideration when updating the box’s position. If the box moves, the application will inform the haptic renderer. This is good because the application remains in total control of the environment. However, this is bad because there is potential for a large delay between when the user pushes the box and when the application decides the box should move. Humans are sensitive to such delays and the result would be a poor user experience.

The VRML-Haptics API takes the latter approach of leaving all of the dynamics to the application. There are two key factors which influenced this decision. First, having all the dynamics computed centrally simplifies the implementation of both the application and the haptic renderer. If the haptic renderer dealt with haptic interaction dynamics and the application dealt with all the other dynamics, then enormous coordination between them would be required; this complication would violate the KISS design principle. The simplicity of the design will result in a more robust system. The second reason for choosing this approach is that over time, computers become faster and implementations become streamlined. This speedup will reduce the experience-impairing delay. So, the downsides of this approach will be eventually be

\(^5\) Keep It Simple, Stupid.
As described above, the VRML-Haptics API is an interface between the Haptic Render and the Haptics Client. The following two sections describe the functionality and implementation of these layers. The documentation for the VRML-Haptics API can be found in the "VRML-Haptics Application Programmer’s Interface Reference" [26].

### 4.3 Haptic Renderer

The Haptic Renderer interfaces a multi-modal application’s Haptics Client to a haptic device. Its actual embodiment is as an server program which accepts VRML-Haptics API commands as RMI transactions. These commands are issued by the Haptics Client and either query the Haptic Renderer for haptic interactions or give it models to display.

Many haptics researchers have created haptic rendering engines⁶. None of these were designed specifically to be a Haptic Renderer, nor must they. Most haptic rendering engines contain more sophisticated functionality than a Haptic Renderer and were written for platforms other than VRML. Nevertheless, it is generally simple to build a Haptic Renderer from a haptic rendering engine; this is due to the fact that the VRML-Haptics API is lightweight and that Java

---

⁶ Haptic Renderer is a proper name for a layer in the computer haptics system. A haptic rendering engine is not a proper name, but merely computer code. The distinction is that the Haptic Renderer adheres to an explicit interface — the VRML-Haptics API.
programming is robust and easy.

This project’s Haptic Renderer implementation used GHOST as its underlying haptic rendering engine. Since the Haptic Renderer uses RMI, it must be written in Java. Although GHOST is a C++ library, its functions can be used within Java programs via the Java Native Interface (JNI) [19]. Since both GHOST and VRML are modeled after OpenInventor, the VRML-Haptics API fits well into the GHOST framework. When VRML-Haptics API commands are issued to the Haptic Renderer, these commands are received through RMI and massaged by Java code; data is then passed through JNI to GHOST which actually performs the rendering tasks (Figure 4.6). This paradigm can be used to implement the entire functionality of a Haptic Renderer using GHOST.

![Figure 4.6: Haptic Renderer implemented with GHOST using Java technologies.](image)
4.4 Haptics Client

The Haptics Client is the liaison between an application (typically a VRML browser) and a Haptic Renderer. Its actual embodiment is a Java object which executes VRML-Haptics API commands as RMI transactions. These commands are processed by the Haptic Renderer, which is a RMI server. The application passes its models and information requests to the Haptics Client.

The Haptics Client is simply a Java object which abstracts the intricacies of RMI and performs application-specific data manipulation. For a VRML browser application, the Haptics Client is the basis for implementation of the VRML prototypes presented in Chapter 3. For example, the HapticDevice queries the Haptics Client for the position and orientation of the physical Haptic Interface Device. MultiModalShape and MultiModalTransform pass their nodes and fields to the Haptics Client; from this, the Haptics Client builds VRML-Haptics data structures and passes them to the Haptic Renderer.

The Haptics Client is closely tied to the application. Whereas the Haptic Renderer may be on another computer and the Haptic Interface Device is an external piece of hardware, the Haptics Client is software that is local to the application. Usually, this software would either be in the form of a Java object instantiated in the application or in the form of a dynamically-linked library called by the application.
4.5 Architecture Extensions

As suggested previously, some haptic rendering engines have more sophisticated functionality than the Haptic Renderer requires. The VRML-Haptics API was intentionally simplified to give a wide variety of haptic rendering engines access to the VRML platform. Sometimes a user might not care about the versatility of the VRML-Haptics API and would want to use the full power of a haptic rendering engine in a VRML scene.

For example, GHOST gives the ability to interact with knobs and sliders; this is a useful feature to VRML content developers. Using the concepts presented in Sections 3.2, 4.4, and 4.5, it would be straightforward to create multi-modal VRML prototypes for these multi-modal “widgets”. The procedure would be as follows:

1) Create VRML prototypes for the widgets.
2) Add widget-specific functions to the VRML-Haptics API.
3) Implement these functions in the Haptic Renderer and Haptics Client.
4) Program the VRML prototypes using the extended VRML-Haptics API.

This approach could be applied to many different features of haptics rendering engines.

It must be noted that a multi-modal widget such as a knob or slider could actually be implemented in VRML without the aid of GHOST’s features. It would be much more difficult to implement, but would maintain the open nature of the computer haptics system. Unfortunately, there may exist some renderer features which cannot be emulated through pure VRML-Haptics API programming.
5 Conclusion

The work presented in this thesis extends VRML to include the haptic modality. These extensions and their underlying system provide a platform for the rapid-prototyping of multi-modal virtual environments. The availability of such a platform will allow non-technical users to easily create multi-modal virtual environments and interfaces. The design of this platform allows haptics researchers to investigate individual modules of the system. Not only does this allow researchers to focus on their interests, but it allows the results of their labor to be easily "plugged" into an end-user's system.

Although this project focuses on incorporating haptics into VRML, the frameworks and concepts discussed can be applied to other problem domains. For example, the haptics system framework can facilitate the incorporation of haptics into a computer-aided design (CAD) application. These principles can eliminate technological barriers for those unfamiliar with haptics, inviting many more researchers and companies into the haptics community. The availability of an open, modular computer haptics platform will increase the productivity of haptics researchers and will promote the mass-market proliferation of haptic interfaces.
6 References


A VRML-Haptics Prototypes

The following are a basic set of VRML prototypes that allow the incorporation of haptics into VRML. The implementation of these prototypes is omitted.

# Node: Feel
# Specifies haptic properties
# Surface model is an array that only contains a SDCompliance.
# It could theoretically accept other models such as texture, vibration and heat, but they need VRML prototypes and renderer support.
PROTO Feel [ 
    exposedField MFNode surfaceModel # Surface Models
    field SFNode proxyGeometry # Haptic-specific geometry
] { 
    # Implementation omitted
}

# Node: SDCompliance
# Specifies the constants for a spring-damper surface compliance model
PROTO SDCompliance [ 
    exposedField SFFloat SpringK # Spring Constant
    exposedField SFFloat DampingK # Damping Constant
    exposedField SFFloat StaticF # Static Friction Const
    exposedField SFFloat DynamicF # Dynamic Friction Const
] { 
    # No implementation needed
}

# Node: MultiModalShape
# Multi-modal representation of objects
# Supersedes the VRML-standard Shape node, adding a slot for feel and dynamics
PROTO MultiModalShape [ 
    exposedField SFNode geometry # actual shape
    exposedField SFNode appearance # its look
    exposedField SFNode feel # its feel
    exposedField SFNode dynamics
] { 
    # Implementation omitted
}
# Node: MultiModalTransform
# A Transform node that is specialized to contain MultiModalShapes.
# It can group any nodes together since it is effectively a Transform.
PROTO MultiModalTransform [ 
  eventIn MFNode addChildren #
  eventIn MFNode removeChildren #
  exposedField SFVec3f center #
  exposedField MFNode children # This is all
  exposedField SFRotation rotation # the same as
  exposedField SFVec3f scale # the ordinary
  exposedField SFRotation scaleOrientation # Transform node
  field SFVec3f bboxCenter #
  field SFVec3f bboxSize #
] { 
  # Implementation omitted
}

# Node: HapticDevice
# Virtual representation of the physical haptic interface device
PROTO HapticDevice [ 
  field SFString hdName # Unique identifier
  field SFNode hdShape # It’s virtual shape
  exposedField SFVec3f hdPosition # It’s position
  exposedField SFRotation hdOrientation # It’s orientation
] { 
  # Implementation omitted
}

# Node: HDTouchSensor
# Sensor that is triggered by HapticDevice
# Its functionality is the same as a TouchSensor, 
# but also tracks the device’s orientation and registers forces.
# A HDTouchSensor responds to the haptics device named in hdName
# DP = Device Position
# DO = Device Orientation
PROTO HDTouchSensor [ 
  exposedField SFString hdName # Who should I watch?
  exposedField SFBool enabled # Is it on?
  eventOut SFVec3f hitNormal_changed # Surface Normal at DP
  eventOut SFVec3f hitPoint_changed # DP in Local Coords
  eventOut SFVec2f hitTexCoord_changed # DP in Texture Coords
  eventOut SFRotation hitOrient_changed # DO in Locals Coords
  eventOut SFFloat pressure # How hard it’s pressing
  eventOut SFBool isActive # Is it dragging?
  eventOut SFBool isOver # Is it touching?
  eventOut SFTime touchTime # Time of hit
] { 
  # Implementation omitted
}