A FORCE FEEDBACK PROGRAMMING PRIMER

For PC Gaming Peripherals Supporting I-Force 2.0 and Direct-X 5.0

Louis Rosenberg, Ph.D.

A Force Feedback Programming Primer

For Gaming Peripherals Supporting **DirectX 5** and **I-FORCE 2.0**

by
Louis B. Rosenberg Ph.D.

Endorsed by these Manufacturers of Force Feedback Hardware:

Logitech Inc.	CH Products
ThrustMaster	SC&T International
ACT Labs	Interactive IO
Advanced Gravis	InterAct Accessories
Nuby Manufacturing	Immersion Corporation

Immersion Corporation
the Force Feedback Company

IMMERSION CORPORATION, San Jose, California 95131 Copyright © 1997

Rosenberg, Louis B.

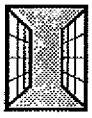
A Force Feedback Programming Primer. For PC Gaming Peripherals Supporting I-Force 2.0 and DirectX 5

Editorial/production supervisor: Tim Lacey
Cover design: Cathy Ricke
Cover art contributions: Bruce Schena

Special Thanks to Adam Braun, Dean Chang, and everyone else on the I-FORCE development team who provided technical assistance without which this book would have been impossible.

Special thanks to I-FORCE hardware manufacturers, who contributed to Section 1.5 of this book.

DirectX is a trademark of Microsoft Corporation. I-Force and I-Force Studio are trademarks of Immersion Corporation. Product and company names in section 1.5 and throughout this book are trademarked by their respective holders.



Copyright © 1997
Immersion Corporation
San Jose, CA 95131

First Printing, April 1997
Second Printing (revised), June 1997

All rights reserved. No part of this book may be reproduced, in any form or by any means, without permission in writing from the publisher.

Printed in the United States of America

APPLE INC.
EXHIBIT 1017 - PAGE 3

PREFACE

Like graphics and sound, Force Feedback is a creative medium that will let you add engaging perceptual content to your gaming environments. As is true of any creative medium, the potential of force feedback is essentially unlimited but the effectiveness of particular force feedback implementations is ultimately in the hands of the individual developer. Our goal in writing this book is to expose developers to the exciting potential of force feedback technologies, to provide guidelines on how to use "feel" most effectively within your gaming applications, and to encourage gaming innovators to invent creative uses of "feel" that have never before been imagined. We sincerely hope the information and insights provided in this text will encourage developers to use force feedback technologies to their fullest potential.

This book is structured as an introduction to the hardware and software issues of force feedback technology. The text is presented as a reference document written for professionals involved in all aspects of game development from conceptual design to low-level coding. For the game designer, this book provides an overview of force feedback device capabilities, giving you an understanding of how "feel" can enhance your current gaming titles. For the programmer, this book describes the methods used to add feel sensations to your applications. We hope the information provided in this text will clarify, simplify, and facilitate the force feedback development process.

While many of the concepts introduced in this text are applicable across all platforms, the specific code examples used in this text assume that you are programming force feedback hardware compatible with the **DirectX 5** specification defined by Microsoft and Immersion Corporation. Some of the advanced features described in this text are enabled through the "**I-FORCE Studio**" toolset from Immersion Corporation. The I-FORCE Studio toolset enhances and facilitates DirectX force feedback development and is described in detail in the last chapter of this book. For the latest updates on force feedback technology and a list of titles currently supporting this new technology, please visit the web site **www.force-feedback.com.**

APPLE INC.
EXHIBIT 1017 - PAGE 4

CONTENTS

1. INTRODUCTION **1**

1.1 OVERVIEW OF FORCE FEEDBACK TECHNOLOGY 2
1.2 THE SCIENCE OF FORCE FEEDBACK. 6
1.2.a Bi-Directional Interaction *9*
1.2.b So, What is a Feel Sensation Anyway? *11*
1.2.c Reflexes, A Technique for Efficient Force Feedback Programming *16*
1.2.d Intelligent Disturbance Filtering (IDF) *18*
1.3 SUPPORTING FORCE FEEDBACK - THE EMERGING STANDARDS 20
1.4 THE DIRECTX API AND THE I-FORCE STUDIO TOOLSET. 21
1.5 WHAT TO EXPECT FROM MANUFACTURERS 23
1.5.a Logitech Inc. *24*
1.5.b CH Products *25*
1.5.c ThrustMaster *26*
1.5.d Immersion Corporation *27*
1.5.e ACT Labs *28*
1.5.f SC&T International *29*
1.5.g Nuby Manufacturing *30*
1.5.h Interact *31*
1.5.i Interactive I/O *32*

2. THE BASICS OF FORCE FEEDBACK **33**

2.1 OVERVIEW 34
2.2 OVERVIEW OF FORCE FEEDBACK HARDWARE ARCHITECTURE 35
2.3 OVERVIEW OF FORCE FEEDBACK SENSATIONS 43

APPLE INC.
EXHIBIT 1017 - PAGE 6

2.3.a Spatial Conditions *43*
2.3.b Temporal Waves *44*
2.3.c Dynamic Sensations *45*
2.4 FORCE STREAMING 47

3. UNDERSTANDING SPATIAL CONDITIONS 49

3.1 OVERVIEW 50
3.2 SPRING 52
3.3 DAMPER 57
3.4 INERTIA 59
3.5 FRICTION 60
3.6 TEXTURE 61
3.7 WALL 63
3.8 BARRIER 68

4. UNDERSTANDING TEMPORAL WAVES 71

4.1 OVERVIEW 72
4.2 DEFINING FORCE SIGNALS: CONSTANT & PERIODIC 73
4.2.a Impulse Wave Shaping of Force Signals *75*
4.2.b Summary of Periodic Force Signal Generation *81*
4.2.c Three types of Periodic Waves *83*
4.3 DEFINING FORCE PROFILES 86
4.3.a Custom Force Profiles *86*

5. UNDERSTANDING DYNAMIC SENSATIONS 89

5.1 OVERVIEW 90
5.2 THE BASIC DYNAMIC SENSATIONS 93
5.3 DYNAMIC RECOIL - "ULTRA-REALISTIC WEAPON SIMULATION" 96
5.4 DYNAMIC IMPACT - "ULTRA-REALISTIC COLLISION SIMULATION" 98
5.5 DYNAMIC LIQUID - "A LIQUID SENSATION THAT JIGGLES" 100
5.6 DYNAMIC INERTIA - "ADJUST THE WEIGHT OF YOUR INTERFACE" 102
5.7 DYNAMICCENTERDRIFT - "SPRING ORIGIN FOLLOWS USER OVER TIME" 103
5.8 DYNAMIC SLING - "LETS YOU FEEL A BALL-ON-A-STRING" 104
5.9 DYNAMIC PADDLE -"LETS YOU FEEL A BALL-PADDLE INTERACTION" 107
5.10 DYNAMIC CONTROL LAW 111

6. THE PROGRAMMING MODEL FOR SIMULATED FEEL 113

6.1 OVERVIEW 114

APPLE INC.
EXHIBIT 1017 - PAGE 7

6.2 THE CONCEPTUAL MODEL SHARED BY DIRECTX AND I-FORCE 116
6.3 THE EFFECT STRUCTURE 119
6.4 THE TYPE DEFINITIONS 123
6.4.a Condition Type Definitions 123
6.4.b Wave Type Definitions 124
6.4.c Dynamic Type Definitions 124
6.5 DIRECTION CONVENTIONS 125
6.5.a 1D Devices 125
6.5.b 2D Devices 125
6.5.c 3D Devices 125
6.5.d nD Devices 126
6.6 TYPE SPECIFIC PARAMETER STRUCTURE 127
6.6.a Condition Struct 127
6.6.b Periodic Struct 128
6.6.c Constant Struct 129
6.6.d Custom Struct 130
6.6.e Ramp Struct 130
6.6.f Dynamic Struct 131
6.6.g Barrier Struct 133
6.6.h Wall Struct 135
6.6.i Advanced Periodic Struct 137
6.6.j Texture Struct 138

7. IMPLEMENTATION OF FORCE FEEDBACK USING DIRECTINPUT 139

7.1 OVERVIEW 140
7.2 ENUMERATING DIRECTINPUT FORCE FEEDBACK JOYSTICKS 141
7.3 CREATING THE DIRECTINPUT FORCE FEEDBACK DEVICE 142
7.4 GETTING DIRECTINPUT FORCE FEEDBACK DEVICE CAPABILITIES 143
7.5 GENERATING DIRECTINPUT FORCE FEEDBACK DEVICE EFFECTS EXAMPLE: SPRING AND TEXTURE 144

8. THE I-FORCE 2.0 WRAPPER FUNCTIONS 147

8.1 OVERVIEW 148
8.2 CONDITION FUNCTION WRAPPERS 149
8.3 WAVE FUNCTION WRAPPERS 153
8.4 DYNAMIC FUNCTION WRAPPERS 155
8.5 SAMPLE CODE FOR FUNCTION WRAPPERS. 158

APPLE INC.
EXHIBIT 1017 - PAGE 8

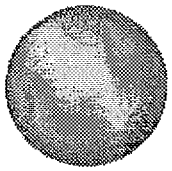
9. THE I-FORCE STUDIO TOOLSET FOR DIRECTX 171

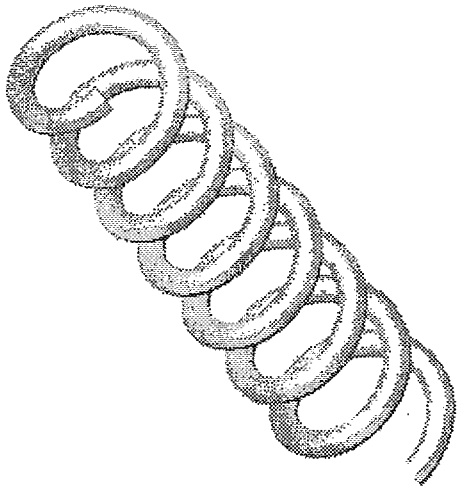
9.1 OVERVIEW 172
9.2 INTRODUCTION 173
9.3 THE DEVELOPMENT ENVIRONMENT 174
9.4 THE SENSATION DESIGN PROCESS 177
9.5 THE SIMPLIFIED PROGRAMMING PROCESS 181

APPLE INC.
EXHIBIT 1017 - PAGE 9

1. Introduction

You are Captain Flack, the latest space warrior to battle evil creatures in the dark tunnels of Griok. In the distance you see a platoon of enemy soldiers. Using your force feedback joystick, you cautiously retreat into the shadows – abruptly you ***feel*** the solid surface of a stone wall at your rear. You can withdraw no further. Instead, you strafe along the wall, ***feeling*** the texture of the rock surface as you cover ground. Suddenly you ***feel*** a snap as the wall gives way behind you – a secret passage. You enter. ***Splash***, you slip into an underground pool. As you wade through the waste-deep slime, the ***feeling*** of undulating muck makes it difficult to walk in a straight line. But you are a fast study and quickly learn that by absorbing the undulations with your wrist, you can move as efficiently in water as on land.





APPLE INC.
EXHIBIT 1017 - PAGE 10

1.1 Overview of Force Feedback Technology

Physical sensations added to gaming software greatly enhance the realism of simulated environments, providing players with more engaging, more intuitive, and more entertaining experiences. High fidelity feel, like advanced graphics and sound, adds dramatically to the perceptual richness of simulated interactions by presenting users with compelling information through natural and intuitive channels. But while graphics and sound are passive media presented to static *observers*, force feedback is a inherently interactive media that is engaged by active *participants*. In other words, when using force feedback hardware technology, the player pushes on the game and the game pushes back.

The implications of such bi-directional interactions are nothing less than profound – for when a player makes a physical gesture and the computer fights back with genuine physical forces, the basic premise of computer gaming is launched to an entirely new level of realism. *Computer Gaming World* magazine, after reviewing the world's first consumer force feedback joystick, the **Force-FX Joystick** developed by CH Products and Immersion Corporation, commented:

> *"A force feedback joystick does more for the feeling of 'being there' than any VR helmet."*
> Computer Gaming World, 11/96

The above observation is highly insightful, for it summarizes a universal reaction players report when first experiencing force feedback technology – namely that force feedback makes even the most basic simulated interactions seem intensely real. A ball bouncing on a paddle, a boat floating through water, a sword clanking into armor, a missile thrusting off the deck of your ship; no matter how simple or sophisticated the graphics, an appropriate feel sensation can make the gaming event seem startlingly genuine.

Many users are actually surprised by the profound improvement that subtle physical sensations can have upon the realism of gaming interactions. Many first time users even inquire as to how such simple physical cues can so substantially enhance the perceived realism of a simulated event. The answer has nothing to do with software, it has only to

APPLE INC.
EXHIBIT 1017 - PAGE 11

do with the human perceptual system – the human organism has evolved to rely heavily on feel sensations as a primary means of instilling the surrounding world with the sensory impression of concrete substance and physical realism. Although we often take it for granted, people depend greatly upon *feel* as a critical modality for interacting with and understanding the physical world. As a result, adding accurate feel sensations to simulated worlds greatly facilitates a user's *suspension of disbelief* and induces users immerse themselves within the simulated experience, transforming game players from passive observers to interactive participants.

Of course today's computer users have been cultured to expect and accept simulated environments that are rich in graphics and sound but devoid of all physical content. These cultural expectations, though deeply ingrained by decades of "feel-less" computing, will rapidly change as users learn that just because an environment is computer generated, it need not deprive them of their intuitive physical senses. With a number of force feedback products already on the market and many more scheduled for release over the next year, consumers will quickly start demanding gaming experiences that are instilled with physical realism. Feel will soon become a basic requirement of quality gaming rather than the advanced enhancement it is today.

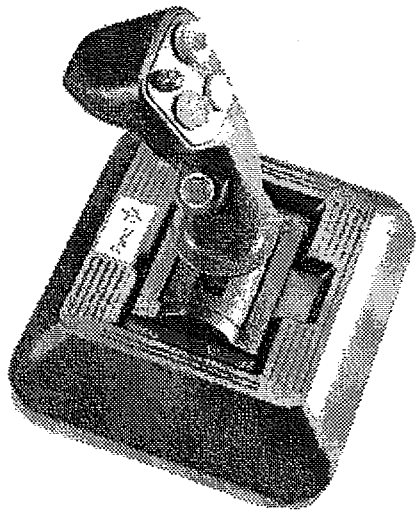


Figure 1-1 The Currently Shipping Force-FX Joystick

In many ways, the addition of Force Feedback technology to gaming controllers is much like the transition from monochrome displays to color displays made by computer users a

APPLE INC.
EXHIBIT 1017 - PAGE 12

generation ago – once a user makes the shift, there is simply no turning back. For most users, the first experience with force feedback is a truly memorable event, a revelation of sorts – suddenly a new channel of information has been added to the world of computing and users do not want to turn it off. *Game Pro Magazine* and *Computer Life Magazine* gave the following assessments after their first experience with a joystick enabled with the I-FORCE force feedback technology from Immersion Corporation:

"Once you've felt the force, it may be hard to go back to a static stick"
Computer Life Magazine 9/96

"Once you've tried an I-FORCE joystick with a game, playing without it isn't nearly as fun!"
Game Pro Magazine 8/96

It is not only users who are excited by the potential of force feedback. Game developers are captivated not only by the feel, but by the creative potential of this largely unexplored new medium. By adding the physical modality to the toolbox of perceptual effects that game developers have at their disposal, Force Feedback will breathe new life into common video game paradigms. In addition, this new physical modality for human-computer interaction is sure to inspire entirely new scenarios for video game experiences which have never been considered because the feel sensations were simply unavailable. Game paradigms will be developed where feel is so critical to play, users can not perform the task without leveraging their physical dexterity and manual intuition – it is not hard to imagine feel becoming essential to skilled play – after all, in the real world you can not throw a ball, swing a sword, or even walk across a room without depending heavily on your sense of feel.

APPLE INC.
EXHIBIT 1017 - PAGE 13

Reporting on their first experience with force feedback technology, gaming experts at *Next Generation Magazine* wrote:

> *"More than any other advanced controller technology, Force-Feedback promises to open up whole new ways of experiencing a video or computer game"*
> Next Generation Magazine 5/96

> *"Force feedback joysticks enable players to go out and physically wrestle with an opponent for the ball...Basically, the simplest game in the world becomes very interesting when you add force feedback... This could be the understatement of the year"*
> Next Generation Magazine 5/96

Of course there will remain skeptics who view force feedback as a novel embellishment rather than a fundamental milestone in interactive computing. Such misconceptions will be born from either a lack of experience with quality Force Feedback technology or from confusion between Force Feedback and gimmicky technologies where simple buzzing is touted as "feel". Force Feedback, when done right, is not just a "whack" or a "buzz" corresponding with a game event, true Force Feedback is the generation and presentation of complex and subtle feel sensations that accurately represent physical phenomenon through carefully simulated forces. In other words, Force Feedback is an "information technology" that provides a rich and perceptually important channel of information that informs, engages, and interacts with users as they experience simulated environments.

APPLE INC.
EXHIBIT 1017 - PAGE 14

1.2 The Science of Force Feedback.

Force Feedback, also known as *haptic feedback* or *force reflection*, refers to the technique of adding "feel" sensations to computer software by imparting real physical forces upon the user's hand. These forces are imposed by actuators, usually motors, incorporated in the interface hardware. The interface hardware may be a joystick, steering wheel, flight yoke, or other standard peripheral. When equipped with force feedback technology, the interface device can impart the simulated "feel" of jolting blasts, rigid surfaces, viscous liquids, compliant springs, jarring vibrations, grating textures, heavy masses, gusting winds, rumbling engines, impacting asteroids, and just about any other physical phenomenon that can be represented mathematically.

The basic premise behind Force Feedback technology is quite simple – as the user manipulates the interface hardware, the actuators apply computer modulated forces that either resist or assist the manipulations. These forces are generated based on mathematical models appropriate for the desired sensations. For example, when simulating the feel of a rigid wall with a force feedback joystick, motors within the joystick apply forces to the handle to replicate the feel of encountering a wall. In this case, the mathematical model driving the forces is as follows – as the user moves the joystick to penetrate the wall, the motors apply a force that resists the penetration. The harder the user pushes, the harder the motors push back. The end result is a sensation that feels quite compelling because it truly represents a physical encounter with an obstacle – a simulated obstacle – but an obstacle none the less. Other sensations follow more complex mathematical models, but the paradigm is basically the same.

The magic behind generating compelling force feedback sensations is of course in the mathematical models that control the actuators. These models may be very simple, modulating force based on a predefined function of time (the resulting sensations are known as "*Temporal Waves*"). These models may be more complex, modulating forces based on user manipulations (the resulting sensations are known *as Spatial Conditions*"). These models may even modulate force based on both time and user manipulations (the resulting sensations are known as "*Dynamic Sensations*"). Together, these three *classes* of feel sensation make up the basic conceptual foundation for force feedback

APPLE INC.
EXHIBIT 1017 - PAGE 15

programming. These types of feel sensation are listed for your reference below and are described in detail throughout the rest of this text:

Sensation Class	Sensation Overview	Details
Spatial "**Conditions**"	Feel sensations that are functions of *motion*	Chapter 3
Temporal "**Waves**"	Feel Sensations that are a function of *time*	Chapter 4
"**Dynamic**" Sensations	Feel sensations based on *dynamic interactions*	Chapter 5

To simplify the process of generating feel sensations, the manufacturers of force feedback hardware have taken care of all the mathematics required in the generation of the above types of feel sensations. Most of the sophisticated computations are handled by dedicated hardware on board the peripheral device. For example, I-FORCE is a computation engine from Immersion Corporation that has been licensed to many major manufacturers of gaming peripheral devices for use in their force feedback products. Such a computation engine enables a wide variety of complex feel sensation to be produced efficiently in hardware with minimal programming overhead.

With manufacturers handling the mathematical complexities of force feedback in dedicated hardware, game programmers can be provided with an easy to use high-level API that abstracts the problem of feel programming to a perceptual rather than mathematical level. API calls allow programmers to easily define and initiate feel sensations using intuitive function calls with descriptive physical names such as "Wall", "Vibration" or "Liquid". These functions are highly parameterized so that programmers can customize the feel of each basic sensation type with great flexibility. Programmers are thereby spared the burden of actually controlling force as a mathematical function of time or motion. Of course there are methods by which advanced programmers can delve into the mathematics and create their own sensations at the lowest level, but for the most part programmers can generate very sophisticated sensations that are carefully tuned to desired gaming events by using the high-level API calls provided by manufacturers.

APPLE INC.
EXHIBIT 1017 - PAGE 16

To make force feedback programming easier, the core API for the I-FORCE hardware processor has been added to **DirectX 5** from Microsoft. The **I-FORCE Studio** toolset has also been developed by Immersion Corporation, the inventors of consumer force feedback technology, to further support force feedback development within DirectX. I-FORCE Studio is a development package for use with DirectX that greatly facilitates the force feedback programming process and simplifies the design of feel sensations. An added feature of I-FORCE Studio is that it enables force feedback in operating systems other than those supported by DirectX (e.g., **DOS** and later perhaps consoles).

Whether programming for Windows, DOS, or another platform, the DirectX API and/or the I-FORCE Studio toolset greatly facilitate the generation of the three primary classes of sensations: **Temporal Waves**, **Spatial Conditions**, and **Dynamic Sensation**. These three sensation types are described in great detail throughout this text. In addition, this text will also describe a technique for modulating forces directly, allowing you to create sensations from the most basic building blocks, discrete forces. This technique is called **Force Streaming** and is best implemented on rapid communication channels such as USB. See section 2.4 for details on Force Streaming. Finally, at the end of this book, Chapter 9 will introduce the I-FORCE Studio toolset and describe how force feedback programming is accelerated by using these simple graphical tools to create DirectX compatible code.

APPLE INC.
EXHIBIT 1017 - PAGE 17

1.2.a Bi-Directional Interaction

To fully understand force feedback programming, it is helpful to first explore how force feedback hardware devices differ from traditional gaming peripherals. Typical human interface devices are *input-only*; they track a user's physical manipulations but provide no manual feedback representing the results of those manipulations. As a result, information flow for traditional joysticks, steering wheels, and other gaming peripherals is in only one direction, from the peripheral to the host computer. Force Feedback human interface devices are *input-output devices*. They not only track a user's physical manipulations (input), they also provide realistic physical sensations coordinated with game play (output). Therefore the host computer running the gaming application needs to communicate quickly with the force feedback device bi-directionally. Tracking information is sent from the peripheral to the host for use in controlling game play. Force Feedback information is sent from the host to the peripheral, to coordinate feel sensations with gaming events.

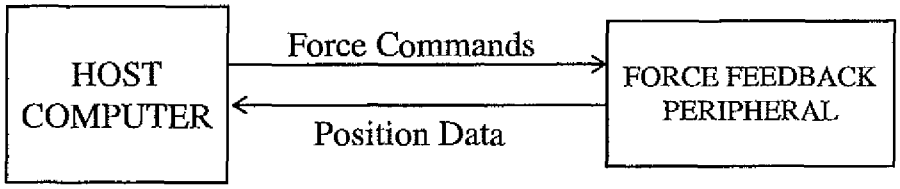


Figure 1-2 Force Feedback Information Flow

Because force feedback devices require bi-directional communication to coordinate feel sensations with gaming play, communication speed has a substantial effect upon force feedback performance. The faster the communication link, the better the coordination between visual and physical events.

Ideally, force feedback devices use efficient processing techniques to minimize the amount of information that needs to be communicated between the host and the peripheral.

APPLE INC.
EXHIBIT 1017 - PAGE 18

For example, efficient force feedback devices use sophisticated local microprocessors to produce force feedback sensations locally in response to concise high-level commands sent from the host. The I-FORCE processor is a popular example of such a force feedback computation engine that has been licensed by many major manufacturers as listed in Section 1.5 of this book. Beyond reducing the amount of information that needs to pass between the host and the peripheral, such local processing hardware has the added benefit of performing force feedback computations in parallel with host execution of gaming software, thereby reducing the processing burden of the host. This distributed processing architecture is shown in Figure 1-3 and will be described in great detail in Section 2.2.

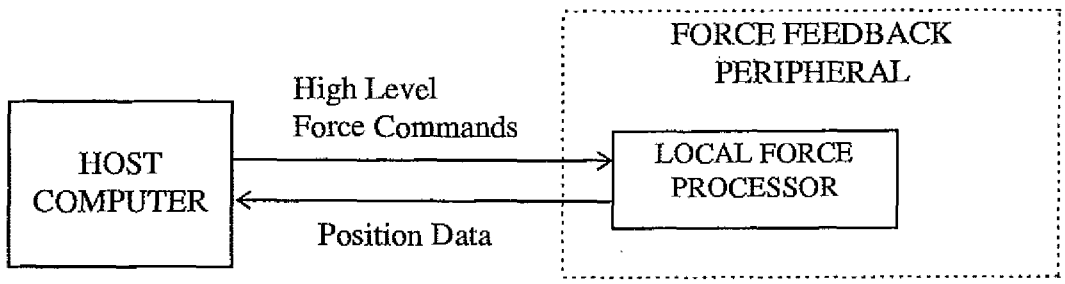


Figure 1-3 Distributed Processing Architecture

By reducing the amount of information transfer required to convey high fidelity feel sensations, local processing is a powerful means of enhancing force feedback performance. Another method of increasing performance is simply increasing communication speed between the host and the peripheral. At the present time, the most ideal communication medium for force feedback gaming peripherals is the Universal Serial Bus or USB. The USB provides a high speed bi-directional communication channel which can service multiple peripherals simultaneously.

APPLE INC.
EXHIBIT 1017 - PAGE 19

1.2.b So, What is a "Feel Sensation" Anyway?

The key components of a typical force feedback device are the **actuators** (usually motors) and the **sensors** (usually encoders or potentiometers). A force feedback sensation is generated by controlling the current through the motors, thereby modulating forces imposed on the user's hand. These forces can be controlled as a function of time, controlled as function of sensor readings, or controlled as a function of both time and sensor readings. To clarify these three distinct cases, lets take a look at three basic yet important sensations – the Jolt, the Spring, and the Dynamic Impact :

Example 1: The Jolt Sensation

The Jolt is a basic example of a "Temporal Wave" meaning that a jolt is a force modulated as a function of time. Like all temporal Waves (described in detail in Chapter 4), a Jolt is a pre-defined force profile that is stored and "played back" over a time period. In the simplest case, a Jolt is defined by parameters such as a *magnitude*, *direction*, and *duration*. So, you might describe a jolt as a force with a magnitude of 50%, at an orientation of 45 degrees, for a duration of 50 milliseconds. By sending this information to the force feedback peripheral device, an appropriate Jolt sensation is executed. For example, your software might issue this call when your simulated Star Fighter is hit by an enemy blast. Depending upon the intensity and direction of the blast, you can choose the appropriate parameters. The effect will be a sensation that can be drawn as a force felt as a function of time.

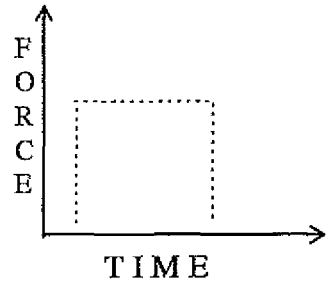


Figure 1-4 A Simple Jolt

APPLE INC.
EXHIBIT 1017 - PAGE 20

Of course Temporal Waves can be more complicated than a simple Jolt of constant magnitude. Forces played over time can have complex predefined profiles. For example, vibrations can be composed of *periodic* signals such as square waves, sine waves, or triangle wave. Custom *profiles*, on the other hand, are a different type of temporal Wave that are not simple mathematical functions, but rather are defined as a prescribed sequence of discrete sampled data. Regardless of how the Wave is define, all Temporal Waves have one basic thing in common – they are predefined and played back over time. In other words, they are "canned" sensations where force varies over time in a pre-planned manner. Such effects have certain important applications, but because they are canned rather than "interactive", they are typically the least interesting class of force sensations and should be used sparingly.

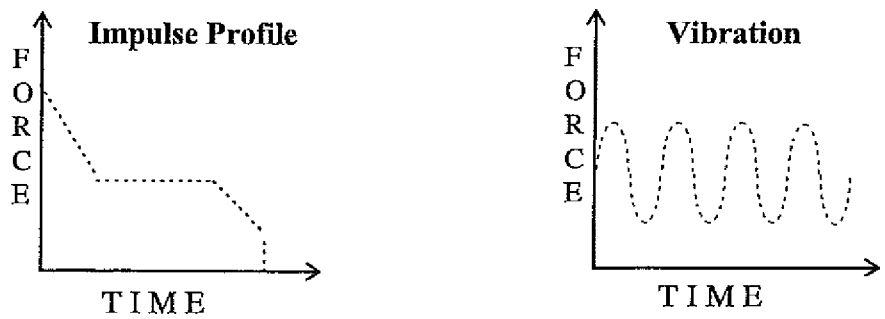


Figure 1-5 Force Profiles for Predefined Temporal Waves

Example 2: The Spring Sensation

The Spring is a basic example of a "Spatial Condition" meaning that the spring is a force sensation generated by modulating the current through the motor as a function of spatial motion of the interface device. In the case of a joystick, force is modulated as a function of displacement of the joystick handle. A you might imagine, a spring sensation is one where the force increases as you move the

APPLE INC.
EXHIBIT 1017 - PAGE 21

joystick in a given direction, creating the illusion that a spring is being compressed in that direction.

It is important to understand that the force produced by a *spring* Condition is NOT predefined and played over time as is the force profile of the simple is a jolt effect described above – for a spring condition, all that is predefined is the relationship between force and displacement. Once the relationship is defined the actual sensation depends upon how the user moves the joystick. If the user holds the joystick in one place, the force will not change. If the user moves the joystick through the range of the simulated spring, the forces will change. For a basic spring, the Force versus Displacement profile looks something like this:

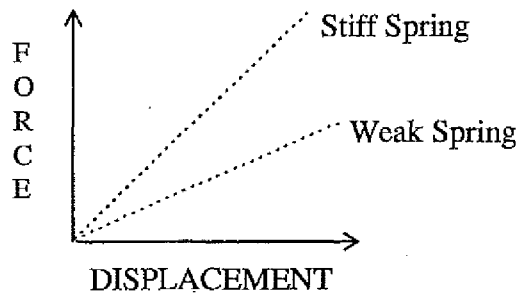


Figure 1-6 Force Displacement Profiles

As shown in the diagram above, the feel of the spring sensation is defined by the particular relationship between displacement of the joystick and increasing force. If force increases steeply with displacement, the simulated spring will feel *stiff*. If the force increases gradually with displacement, the simulated spring will feel *loose*. The slope of this relationship is commonly called the "stiffness" of the simulated spring sensation. To implement a Spring sensation, a programmer would simply call a "spring" command and would define basic parameters such as the stiffness and the orientation.

APPLE INC.
EXHIBIT 1017 - PAGE 22

Because spring sensations, like all Conditions (described in detail in Chapter 3), create force sensations that are functions of user motion, they are much more "interactive" than canned Waves which are merely played back over time. Therefore Conditions are more compelling implementations of force feedback than Waves. Conditions can be as simple as springs, but can also be more complex sensations such as dampers, textures, friction, or even hard walls. For example, a damper sensation can provide a viscous feel as if you were pulling your joystick through a vat of thick honey. A texture sensation can provide a feel as if you were pulling your joystick across a rough sand-paper surface. A wall sensation can provide a feel as if your joystick banged into a simulated hard stop. All of these sensations share one thing in common, the FEEL is based on the motion of the interface device – how the honey is stirred by the user, how the texture is rubbed by the user, how the wall is impacted by the user, and how the spring is compressed by the user. Because the feel of such sensations are dependent upon how the user moves the interface device in space, they are classified as "Spatial Conditions".

Example 3: The Dynamic Impact Sensation

The Dynamic Impact is an example of a "Dynamic Sensation" meaning that the force is not a canned function of time (like Waves) or a simple function of motion (like Conditions) but rather is based on a sophisticated dynamic simulation involving both *time* and *motion*. Dynamic Sensations are best thought of as interactive events like the jarring collision of two bodies hitting and bouncing apart, the undulating sensation of a mass moving through a pool of water, or the reverberating recoil of a weapon after firing. Such Dynamic sensations will be described in detail in Chapter 5.

A Dynamic Impact sensation is a Sensation that can be conceptualized as the physical representation of an object propelling itself into the handle of your joystick and then bouncing off with the interactive feel of a ball bouncing off a rubber surface. This is not just a pre-defined jolt, it is a truly interactive exchange of energy between the joystick and the user. The user's actions *during*

APPLE INC.
EXHIBIT 1017 - PAGE 23

the impact event actually affects the feel of the sensation. If the user stiffens his or her wrist, the sensation will feel very different than if the user cushions the blow with a loose arm. In other words, while a canned Jolt Wave is just a predefined sequence of forces played over time that always feels the same regardless how the user reacts during the play period, Dynamic Impact (like all Dynamic Sensations) is an interactive dynamic simulation where the user's motion during the event changes the feel. This is because the feel of the Dynamic is adjusted in real-time to account for user motion thereby creating a compelling interactive feel based on physically intuitive model.

In a sense, Dynamic force feedback interactions make video gaming much like physical sporting where dexterity and finesse are central to play. Just as the feel of a ball bouncing off a tennis racquet depends greatly upon how you react with the racquet *during* the sporting interaction, the feel of a Dynamic Sensation depends upon how you react with the joystick *during* the gaming interaction. The results are extremely realistic sensations that allows for fine interactive control. For example, good players of a space game using Dynamic Impacts will learn how to minimize the disturbing effect of asteroids as they collide with their ship by reacting appropriately to absorb the impact energy in their palm during the collision. Such advantages of Dynamics over both Conditions and Waves will be described in detail in Section 2.3.

APPLE INC.
EXHIBIT 1017 - PAGE 24

1.2.c Reflexes, A Technique for Efficient Force Feedback Programming

In the human body, very rapid physical responses such as a *knee jerk*, are made possible through the mechanism of *spinal chord reflexes*. The reflex concept in humans is quite simple. Rather than waste the processing time to bring the stimulus (the strike of a mallet on the knee) from the leg all the way to the brain so the brain can process the stimulus and then send a knee jerk command all the way back to the leg muscles – the spinal chord processes the input *locally* and responds directly without intervention of the brain. In other words, the reflex is a *local* reaction to an input that allows for faster output response.

In force feedback devices, very often you want physical output to happen in response to an input such as a button press. For example, you are playing a first person action game and when you pull a trigger, your character fires a revolver. You want the user to feel a *recoil* sensation every time this trigger is pulled. One way to do this is to have your host computer monitor the button press (input) associated with the trigger, then command the joystick to perform the *recoil* sensation (output). This method works, but it uses up communication bandwidth in both directions, especially if the user pulls the trigger many times in rapid succession. In the worst case, the trigger will be pulled too fast for the communication channel and some recoil sensations will be missed. And even if no sensations are missed, if the user pulls the trigger many times, the host is forced to perform many operations quickly, burdening the host and the communication pathway for no good reason. Like the brain of a human, the host computer can benefit greatly by having a means of off-loading simple relations between local events (button presses) and reactions to those events (force sensations).

The solution is the force reflex. The notion is simple, a local microprocessor on board the force feedback device can be configured to execute any sensation in response to any local event. For example, a vibration sensation of a specific frequency and duration might be associated with the press of hat switch #2. A jolt sensation of a particular magnitude might be associated with the press of trigger #1. Ideally your force feedback hardware allows you to set up a great many reflexes that are all resident at once. This greatly reduces the communication throughput between the host and the joystick and it reduces the processing burden of the host computer.

APPLE INC.
EXHIBIT 1017 - PAGE 25

The Reflex concept works very well for games where a user is going to select a weapon and use it for an extended period until either the user runs out of ammunition or selects another weapon. When a weapon is selected, the host just sets up a reflex on the force feedback device that creates the appropriate sensation when the particular button is pressed. Then during the extended period of play that follows, the host does not have to worry about that particular sensation – it will be generated appropriately by the joystick regardless of how quickly the button might be pressed in rapid succession. If the user runs out of ammunition or changes the weapon, the reflex is updated by the host. If 5 weapons are active at once, all assigned to different buttons, then 5 reflexes are resident on the peripheral device. For example, the local processor might have a representation in local memory that associates buttons with sensations that can be described as follows:

Button	Reflex Sensation Description
Button #4	Produce a **Sine-Wave** Vibration at **100Hz** with **20%** Magnitude in the left-right direction in order to simulate the engine-hum associated with engaging the Afterburner Rockets.
Hat Switch #2	Produce a **30** millisecond **Jolt** at **75%** magnitude to simulate the feel of launching your Patriot Missile.
Trigger #1	Produce a **Square-Wave** Vibration at **10 Hz** with **70%** Magnitude to simulate the feel of firing your forward machine-guns.

Setting up reflexes is actually quite simple and is supported by the DirectX 5 programming API using the Trigger Button described in Section 6.3. If the particular hardware being used with your game does not support reflexes in the local processor on board the device, the host can emulate the feature in the driver. Of course this removes all advantage of the reflex process because the burden is passed back to the host and communication is required between the host and the device for every button press.

APPLE INC.
EXHIBIT 1017 - PAGE 26

Therefore you will notice substantially better performance with hardware that supports "button reflex processing" as compared to hardware devices that do not.

1.2.d Intelligent Disturbance Filtering (IDF)

An important thing to understand about force feedback sensations is that because actual forces are imposed on the user's hand, sensations have the potential to disrupt a user's manual actions. In most cases this is not a problem because the sensation is representative of the gaming event so the force is complementary rather than destructive to game play. In some case, a programmer may want to impose a sensation but may NOT want the sensation to affect a user's ability to manipulate the interface device. To address this need, many force feedback hardware devices will soon support a feature called Intelligent Disturbance Filtering or IDF. This is best conveyed through example:

Imagine that you are piloting a jet-ski across choppy water. Every time your craft hits the crest of a wave, you feel a jolt and your joystick is pitched back. In this case, the jolt is a valuable physical disturbance because the pitching of the joystick, while making play more difficult, is a realistic gaming experience – in real life, the impact of the waves makes piloting the craft more difficult. So, in this game scenario there is probably no need to use Intelligent Disturbance Filtering.

Now Imagine that you are using a force feedback joystick within a first person shooting game, moving your player through a space station while shooting robots. As you run, you feel the vibration of your gravity jet-pack. While this sensation ads great realism to the feel of the game, it might be disruptive to your ability to aim your laser gun and fire because the joystick is vibrating. Hence the dilemma – as a programmer you want to provide vibration to convey a realistic sensation, but you do not want to disrupt the users ability to steer or aim. The solution is to engage an Intelligent Disturbance Filter. When engaged, the peripheral device will report data that is filtered to reduce the disturbing effect of a given force feedback sensation.

For example, a very common disturbance is a *periodic disturbance* such as a square wave, sine wave, triangle wave, or any other repeating force profile. Such a force signal will provide a vibration at a given frequency. To minimize the affect of a periodic force

APPLE INC.
EXHIBIT 1017 - PAGE 27

disturbance, the IDF activated on board the force feedback device can filter the data before it is reported to the host computer. This provides the game with a less jittery signal despite the compelling sensation.

Another common disturbance is an *impulse disturbance* which is a short pulse of force that causes a jolt sensation. This might be a recoil from a gun or the impact from an asteroid. Such a short, intense force signal often causes the user manipulated object, a joystick for example, to be jarred in a given direction. Most of the time, programmers want this disturbance to affect game play because it adds a realistic dimension to the interactions. In some cases, however, programmers want the player to *feel* the jarring disturbance but do not want the jolt to affect the aiming of weapons or other play activities. For such cases, the IDF lets the peripheral report stable data in spite of the rapid impulse.

Ideally this filtering process is performed by a local microprocessor on board the interface device. The filtering happens prior to the local microprocessor reporting data back to the host. Therefore the filtering process happens independent of the host and is invisible to the game. This works well because the force feedback device is performing the high speed force feedback sensations and can therefore filter the data reported back to the host based on how the force feedback sensations are being generated. The host computer receives clean data and the software running on the host does not need to know anything about the filtering process. This architecture is ideal because the IDF filter can be an option that is *chosen by the player*. If the player wants to reduce the difficulty of a force feedback game he or she can choose to engage the filter and enhance control of the device when disturbances hit.

APPLE INC.
EXHIBIT 1017 - PAGE 28

1.3 Supporting Force Feedback - The Emerging Standards

The industry of consumer Force Feedback was launched in 1995 when Immersion Corporation began promoting **I-FORCE** technology to hardware makers and game developers alike. I-FORCE is a hardware technology that turns basic joysticks, steering wheels, and flight yokes into sophisticated force feedback systems. Immersion has licensed the **I-FORCE** core to a half dozen manufacturers of gaming hardware. This core includes a sophisticated local processor that is optimized to perform the computations associated with generating force feedback sensations. The first shipping product with I-FORCE support was the Force-FX joystick from CH Products. Many more products from additional manufacturers will soon follow.

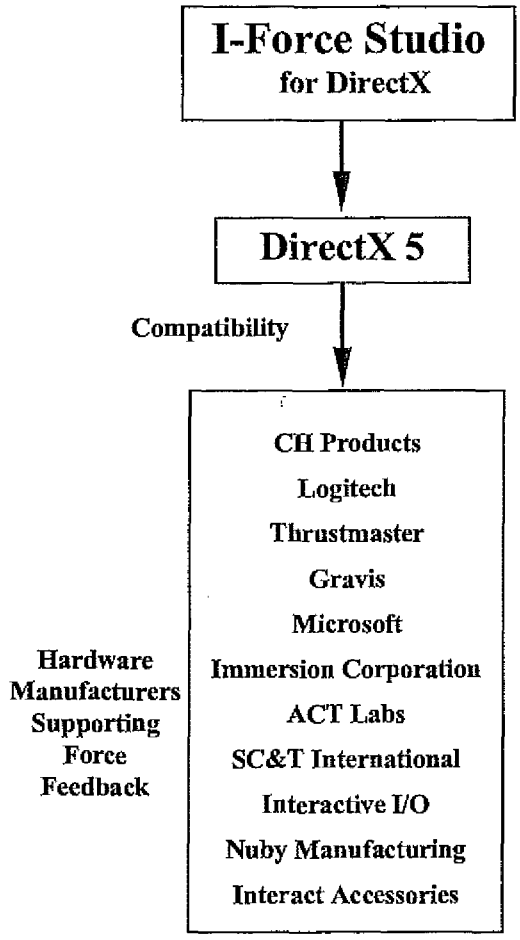
In 1996, Immersion Corporation launched their own **I-FORCE Programming API** to allow game developers to support the many force feedback products scheduled to incorporate the I-FORCE hardware technology. Since then, Immersion Corporation has been working closely with Microsoft to incorporate and extend the functionality of the I-FORCE programming API into Microsoft's **DirectX** standard. Through an extensive collaborative effort, DirectX 5 supports force feedback hardware products enabled by Immersion's I-FORCE technology, letting Windows applications access the power of the I-FORCE processing core.

To facilitate understanding of force feedback technology and promote efficient use of feel sensations within gaming applications, Immersion has recently announced the **I-FORCE Studio** toolset. This is a set of graphical tools for use with DirectX 5 that assists developers in crafting effective feel sensations. Because these tools are so valuable in promoting the effective use of force feedback within gaming applications, we have devoted a full chapter of this book to I-FORCE Studio – Chapter 9.

APPLE INC.
EXHIBIT 1017 - PAGE 29

1.4 The DirectX API and the I-FORCE Studio Toolset.

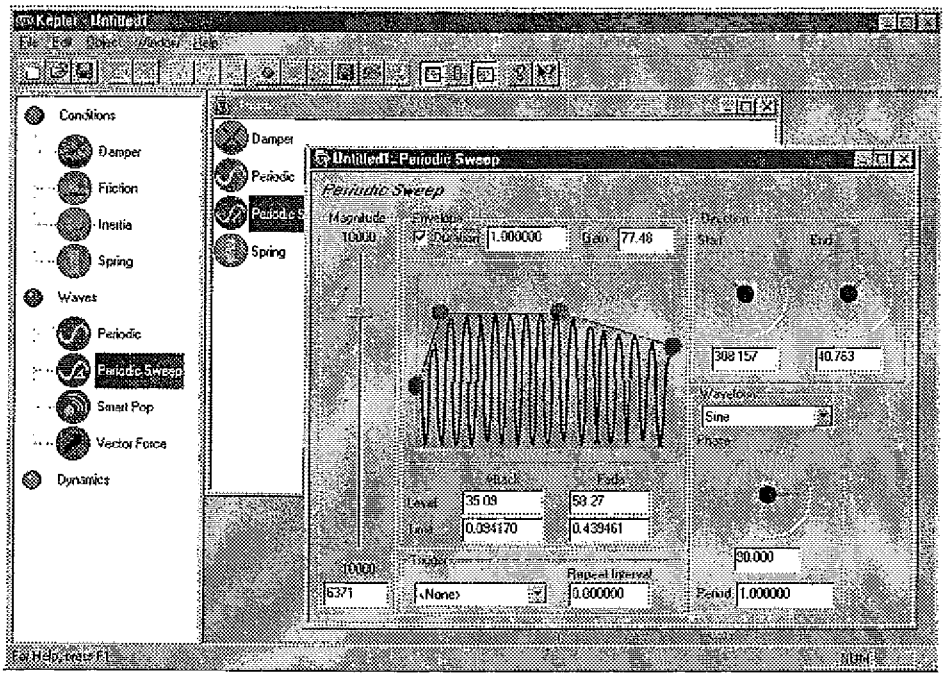
The most important thing to do when planning your force feedback development is make sure that your integration efforts will be fast, easy, fun, *and* will support the largest number of hardware products possible. With so many hardware companies introducing force feedback products in the near future (see Section 1.5), you may be confused about how to best support them all. At the present time, there are only two paths that lead you to support all of the upcoming hardware products in a single effort: either you can code using the DirectX 5 API directly, or you use **I-FORCE Studio**, the graphical force feedback toolkit from Immersion that automatically creates and executes standard DirectX effects for you. The following figure shows the basic architecture for programming force feedback using DirectX directly or using the I-FORCE Studio tools. An overview of both DirectX and I-Force Studio follows on the next page.



APPLE INC.
EXHIBIT 1017 - PAGE 30

DirectX 5: Immersion Corporation and Microsoft worked together to spec out and add force feedback to the newest version of DirectX (DX5). DirectInput now provides a standard set of force feedback features that will work on any force feedback device that supports DirectX. This includes ALL of the hardware products from the 10 manufacturers that are using the I-FORCE technology from Immersion in upcoming products. Unfortunately, direct coding through DirectX is not the best way to rapidly "design" feel sensations because *design* is an iterative process that requires assigning parameters, feeling sensations, and modifying parameters until you get the sensation just right. Thus, programming through DirectX 5 alone is a slow and difficult way to design sensations.

I-FORCE Studio At the request of many hardware and software makers, Immersion Corporation has developed a graphical programming environment for use with DirectX that assists in designing, testing, and coding force feedback sensations. The intent of this toolkit is to greatly simplify the effort required to generate DirectX compatible force feedback code. I-FORCE Studio, shown below, lets you design any DirectX compliant force feedback sensation using intuitive graphical tools. The tools are fully animated so that what you feel is what you see: feel a spring, see a spring – feel a damper, see a damper – feel a wave, see a wave. It makes feel sensation design fast and fun.



APPLE INC.
EXHIBIT 1017 - PAGE 31

1.5 What to Expect from Manufacturers

At the time this book went to press, there were almost a dozen major hardware manufacturers who announced their plans to produce a force feedback gaming peripherals in the near future. Of those companies, almost all have formally endorsed the I-FORCE processing core from Immersion Corporation and have conveyed their support for the I-FORCE force feedback architecture presented in this text. In fact, those companies who have already announced their intent to produce I-FORCE compatible force feedback hardware include:

- Logitech Inc.
- CH Products
- ThrustMaster
- ACT Labs
- Advanced Gravis
- InterAct Multimedia
- Interactive IO
- SC&T International
- Immersion Corporation
- Nuby Manufacturing

APPLE INC.
EXHIBIT 1017 - PAGE 32

1.5.a Logitech Inc.

Logitech

Logitech Inc., and Immersion Corporation are co-developing a next-generation Logitech WingMan Joystick that will feature realistic force feedback. The product will incorporate Logitech's award-winning industrial design with Immersion's advanced I-FORCE 2.0 force feedback technology, and will include features responsible for the continuing success of current models in the retail channel, including rubberized buttons, a heavier base for greater stability and control, and a rugged, sculpted grip for realistic feel and comfort over long periods of time.

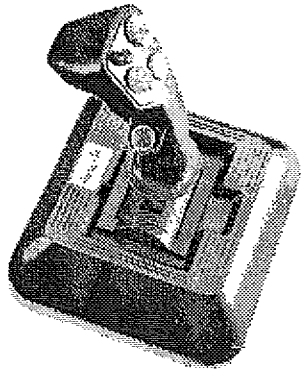
The Logitech-Immersion relationship reinforces Logitech's position as the leading manufacturer and innovator of control devices for PC-based games. Founded in 1981, Logitech designs, manufactures, and markets products that make human-to-computer communication more intuitive and natural. Retail and OEM product offerings include pointing devices, scanners, gaming hardware, and digital video cameras. The company has been a pioneer in numerous gaming hardware innovations, including the popular WingMan Warrior.

For more information or to obtain pre-release force feedback hardware, contact:

CONTACT INFORMATION	
Contact Name	Bob Wudeck
Company Name and Address	Logitech Inc. 6505 Kaiser Dr. Fremont, CA 94555
Telephone	(510) 713-4739
Fax	(510) 792-8901
E-mail	robert_wudeck@logitech.com
World Wide Web	http://www.logitech.com

APPLE INC.
EXHIBIT 1017 - PAGE 33

1.5.b CH Products



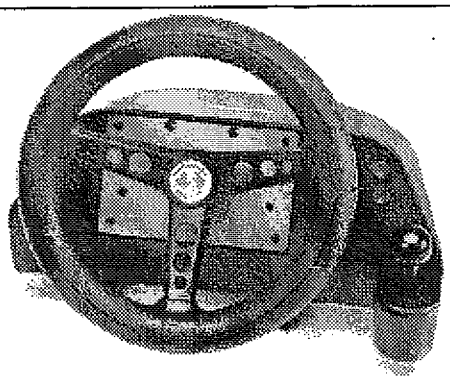
CH Products' Force FX is guaranteed to knock your socks off with six built-in effects you can experience through the stick in many variations of magnitude, duration, direction and repetitiveness. Each variation is determined through your favorite I-Force supported game. The six integrated styles of force have full, programmable feel parameters to allow thousands of distinct sensations. The Force FX features two 4-way, five fire buttons, a trigger and trim controls for a total of 14 functions to play with. If you're gonna play with the FORCE FX, you'd better sit down and fasten your seatbelt. It's a bumpy ride!

For more information or to obtain a developer kit, contact:

CONTACT INFORMATION	
Contact Name	Jennifer Barbaruolo
Company Name and Address	CH Products 970 Park Center Dr. Vista, CA 92083
Telephone	(619) 598-2518
Fax	(619) 598-2524
E-mail	jennifer@chproducts.com
World Wide Web	http://www.chproducts.com

APPLE INC.
EXHIBIT 1017 - PAGE 34

1.5.c ThrustMaster



ThrustMaster, Inc. and Immersion Corporation are working together to bring the excitement of I-FORCE force feedback to its Formula T2 driving system. ThrustMaster's Formula T2 driving control system has already captured the market place due to its quality, durability, and realistic experience. ThrustMaster's mission is to continue as the technology leader in entertainment based input devices.

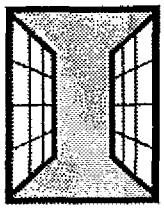
ThrustMaster will be working closely with software developers and publishers to enable the force feedback driving wheel to simulate authentic forces such as skids, collisions, terrain, and other driving conditions within driving software titles.

ThrustMaster, Inc. designs, develops, manufactures and markets input devices that take users of computer entertainment software to new heights bringing realism and true functionality to their gaming experience. For more information or to obtain a developer kit, contact:

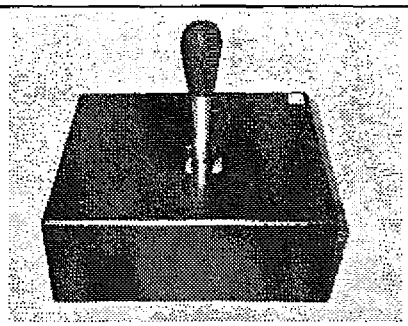
CONTACT INFORMATION	
Contact Name	Alan Smith
Company Name and Address	Thrustmaster 7175 NW Evergreen Pkwy #400 Hillsboro, OR 97124
Telephone	(503) 615-3200
Fax	(503) 615-3300
E-mail	alan@thrustmaster.com
World Wide Web	http://www.thrustmaster.com

APPLE INC.
EXHIBIT 1017 - PAGE 35

1.5.d Immersion Corporation



The Impulse Stick Robust Joystick for Arcade and Location Based Entertainment



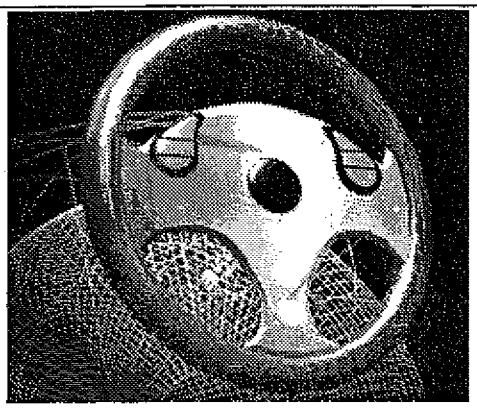
The **Impulse Stick** from Immersion is the worlds first force feedback joystick designed specifically to meet the needs of Arcade and Location Based Entertainment applications. The Impulse Stick is a robust two axis joystick product with analog tracking and force feedback features intended for harsh-use environments. The currently shipping Impulse-Stick product uses the **I-FORCE 1.5** processing core to enable high-performance force feedback functionality through standard serial port interfaces. The Impulse Stick is compatible with PC platforms running Windows-95 or DOS operating systems. The Impulse Stick is also compatible with SGI and other platforms used for location based entertainment systems.

Hardware developers who wish to purchase the Impulse-Stick module from Immersion Corporation as an OEM component for incorporation into Arcade and LBE system should contact Immersion Corporation directly at the number below. Software Developers who want to support the Impulse Stick, should use the I-FORCE 1.5 API.

CONTACT INFORMATION	
Contact Name	Mike Levin
Company Name and Address	Immersion Corporation 2158 Paragon Drive San Jose, CA 95131
Telephone	(408) 467-1900
Fax	(408) 467-1901
E-mail	levin@immerse.com
World Wide Web	http://www.immerse.com

APPLE INC.
EXHIBIT 1017 - PAGE 36

1.5.e ACT Labs



ACT Labs, developers of cutting-edge game peripherals for PC and console gaming systems, has partnered with Immersion Corporation to produce a high-fidelity force feedback driving wheel for the PC platform.

The new wheel will combine ACT Labs' HYPER programmable game control with Immersion's I-Force technology. The wheel will also allow force feedback responses to be programmed into buttons on the wheel. Programmable force feedback can be used in games with or without force feedback support.

Founded in 1994, ACT is committed to bringing gamers the latest in advanced technology. ACT's multi-platform product line consists of joysticks, gamepads, arcade sticks, and light guns for both the video game console and PC markets.

CONTACT INFORMATION	
Contact Name	Erwin Rosales
Company Name and Address	ACT Labs 120-13571 Commerce Parkway Richmond, BC, CANADA V6V 2R2
Telephone	(604) 278-3650
Fax	(604) 278-3612
E-mail	erwin@actlab.com

APPLE INC.
EXHIBIT 1017 - PAGE 37

1.5.f SC&T International

SC&T²

PLATINUM SOUND

The Ultimate PER4MER Steering Wheel is a revolutionary product and has been designed and built from the ground up by an industry leading design studio. The Ultimate PER4MER provides high fidelity force feedback, creating a much more realistic and enjoyable experience. This product is suitable for DOS and Windows based games that support the I-Force API from Immersion Corporation.

To be released with this product is an ergonomically designed industrial strength set of foot pedals. These will incorporate a unique pivotal pedal assembly that will allow the pedal to be comfortably depressed under the foot.

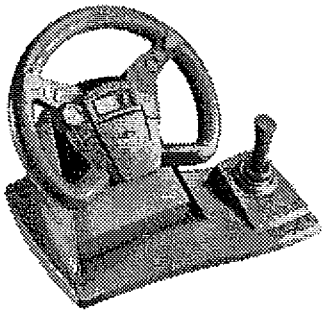
The Ultimate PER4MER Wheel is the creation of SC & T International Inc, a leading manufacturer of racing wheel products, and will be marketed under the company registered trademark, Platinum Sound®.

CONTACT INFORMATION	
Contact Name	Klaus Muerzl
Company Name and Address	SC&T International, Inc. 15695 North 83 Way Scottsdale, AZ 85260
Telephone	(602) 368-9490, x398
Fax	(602) 607-6801
E-mail	kmuerzl@platinumsound.com
World Wide Web	http://www.platinumsound.com

APPLE INC.
EXHIBIT 1017 - PAGE 38

1.5.g Nuby Manufacturing

NUBY



Nuby Manufacturing has recently announced its plans to add Immersion's I-FORCE technology to its current line of joystick and steering wheel products. Nuby makes high quality gaming peripherals for PC, Nintendo 64, and Sony PlayStation platforms as shown in the photographs above. For more information about force feedback product plans, contact Immersion Corporation or Nuby directly at the address below.

CONTACT INFORMATION	
Contact Name	Ed Hames
Company Name and Address	Nuby Holdings Corp. 35 Main Street, P.O. Box 3080 Peterborough, NH 03458
Telephone	(603) 924-2260
Fax	(603) 924-2261
E-mail	Edhames@aol.com

APPLE INC.
EXHIBIT 1017 - PAGE 39

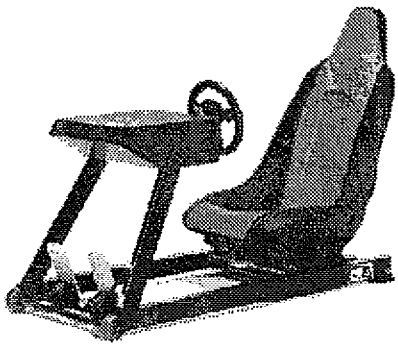
1.5.h Interact

InterAct Multimedia Products develops, manufactures, and markets fun and exciting controllers and joysticks that make gaming more fun, more exciting, and more realistic. Rather than developing for a small niche of gamers, InterAct's products are noted for innovations that appeal to a wide audience of gamers, including the development of a line of products based on I-FORCE force-feedback technology.

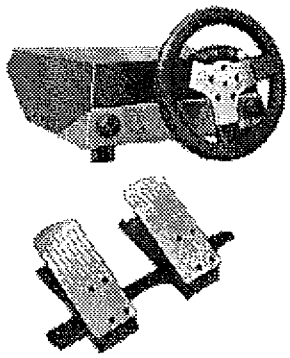
CONTACT INFORMATION	
Contact Name	Michael Rothman, VP of Marketing
Company Name and Address	InterAct Multimedia Products 9611 Pulaski Park Dr. Suite 309 Baltimore, MD 21120
Telephone	(410) 238-2424 x211
Fax	(410) 238-1427
E-mail	mrothman@gameshark.com
Web	http://www.interact-acc.com

APPLE INC.
EXHIBIT 1017 - PAGE 40

1.5.i Interactive I/O



Virtual Vehicle JDi



Virtual Vehicle TDi

At the heart of Interactive I/O's product line is the Virtual Vehicle JD. Both realistic and rugged, Virtual Vehicle JD has been chosen by Papyrus and NASCAR as the control units for their new NASCAR Racing II simulation and on-line kiosk found at the Myrtle Beach NASCAR Café. The Virtual Vehicle JDi incorporates I-FORCE versions 1.5 and 2.0 into a totally self-contained force feedback driving unit. The JDi produces stronger forces than the mass produced consumer sticks and wheels. This makes the JDi the platform of choice among discerning developers who demand the most realistic force feedback environment.

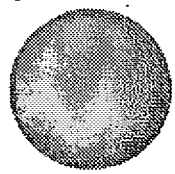
The Virtual Vehicles TD and TDi are based on the same award winning technology as the Virtual Vehicle JD and JDi. This New tabletop line offers a compact solution for those looking for a smaller home or more flexible Location Based Entertainment / commercial product. The Virtual Vehicle TD and TDi offer a compact footprint, easily attached to a tabletop or desk. The sturdy steel cover is perfect for mounting and offers easy installation into a commercial display or cockpit.

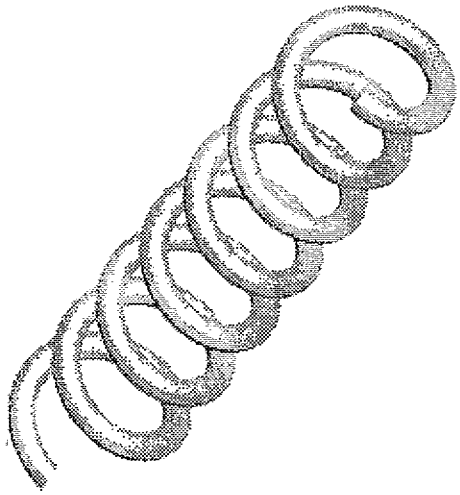
CONTACT INFORMATION	
Contact Name	Francis J. De Giorgio
Company Name and Address	Interactive I/O, Inc. Orange, CA
Telephone & Fax	(714) 921-0994 (tel), (714) 921-2393 (fax)
E-mail	francisd@interactiveio.com
World Wide Web	http://www.interactiveio.com

APPLE INC.
EXHIBIT 1017 - PAGE 41

2. The Basics of Force Feedback

Imagine landing a simulated F-14 fighter plane on the deck of an aircraft carrier in rough seas. As you make your approach, you can *feel* the wind buffeting your plane in all directions. Suddenly a strong *gust* hits you from the side, jarring your stick and rattling your nerves. You recover, and continue your approach. You must fight the driving gusts to keep your nose pointed. As your wheels touch down, you *feel* the sharp thud of contact, then the impulse of rapid deceleration. You catch the arrestor cable and your stick *snaps* forward with a final jerk. You are safe on the deck.





APPLE INC.
EXHIBIT 1017 - PAGE 42

2.1 Overview

This chapter provides a high level overview of force feedback technology. Half of this chapter is devoted to the hardware side of force feedback, providing insights into the components and architecture that make force feedback peripherals work. The second half of this chapter is devoted to force feedback sensations, providing a high level conceptual model for sensation generation. You will be introduced to the key concepts of Conditions, Waves, and Dynamics, the three classes of force feedback sensations that are fundamental to quality force feedback devices. Later chapters will explore all of these concepts in much greater detail.

This chapter is divided up as follows:

A. Force Feedback **Hardware** Overview
- i. Bi-Directional Communication
- ii. Lag Time / Update Rate Requirements
- iii. Distributed Processing Architecture
- iv. Local Memory, Safety Features, Bandwidth

B. Force Feedback **Sensation** Overview
- i. Spatial Conditions
- ii. Temporal Waves
- iii. Dynamic Sensations

APPLE INC.
EXHIBIT 1017 - PAGE 43

2.2 Overview of Force Feedback Hardware Architecture

A force feedback computer peripheral performs two basic functions when connected to a host computer; a) it tracks a user's manual manipulation, feeding sensory data to the host computer representative of those manipulations, and b) it provides physical feedback to the user in response to commands from the host computer. In other words, a force feedback computer peripheral is an input/output device, feeding input data (sensor readings) to the host and displaying output data to the user (physical forces). Because of this unique input/output functionality, a force feedback device requires a bi-directional communication link with rapid information flow on both directions.

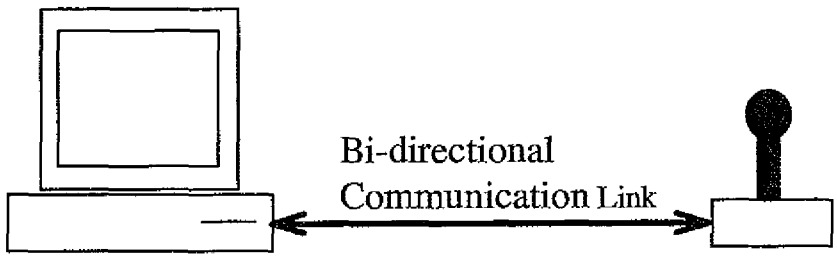


Figure 2-1

The most important thing to understand about force feedback is that the two information pathways, the sensory input and the force output, are tightly coupled. In other words, the force sensations that are commanded by the host (the output) is highly dependent upon how the user is manipulating the interface (the input). In addition, the manual gestures made by the user (the input) are affected by the force sensations commanded by the host (the output). To make this clear, consider the following example:

> A player is using a force feedback steering wheel to drive a race car around a track. When the user pulls the wheel hard and causes the car to make a sharp turn, the host computer updates the simulation based on the sensor input from the wheel. The host determines that the car lost traction around the turn and skidded into the outer wall of the track. At the moment of impact, the host commands the force feedback peripheral

APPLE INC.
EXHIBIT 1017 - PAGE 44

to provide a force feedback sensation representative of the interaction. This might be a sharp jolt. Thus the physical output from the computer is obviously dependent upon the physical input from the user. The reverse is also true, for when the interface device hits the user with the sharp jolt, his or her hand is jarred away from the wall, causing the wheel to move and causing the input to change. In other words, *output is dependent upon input* and *input is dependent upon output.*

This tight coupling between input and output is the very feature that allows force feedback to be the compelling interactive medium that it is. This coupling between input and output is also the most complex aspect of force feedback technology that programmers need to deal with. The first thing to understand about the coupling is that time delay between input and output is very bad. Using the car example above, it is obvious that even a short delay between visually seeing the car hit the wall and physically feeling the car hit the wall will be disconcerting for the user. But such delays can be more than just disconcerting, they can cause stability problems during interactive play. Imagine driving the car into the wall, recovering control of the car through skilled driving, and then being hit by a late jolt. You then try to recover from the jolt and end up over-steering and hitting the wall again. You recover from the wall collision, but another late jolt forces you to repeat the cycle. Clearly this is not going to enhance your driving performance.

To avoid such problems, force feedback devices require rapid communication between the host and the peripheral. The communication rate must be fast enough to ensure that the lag between visual and physical events are not perceptible by the user. As a rule of thumb, time-lags below 25 milliseconds between the onset of visual and physical events will not be detectable by the human perceptual system. This means that standard serial port connections are fast enough to convey force information without a time lag problem. However, this only solves the coordination problem for simple canned Waves like the Jolt sensation described in the car-wall collision presented above. Such simple effects are easy to coordinate because they require only that the onset (and completion) of the force events coordinate with the onset (and completion) of the visual event. But what about complex sensations that require **real-time** coupling between manual input and force output *during* the entire sensation rather than just at the start and end of the sensation? Consider the classic example of the **Spring** that was introduced in Chapter 1.

APPLE INC.
EXHIBIT 1017 - PAGE 45

The **Spring** is a force sensation where the force output is a function of displacement of the peripheral device. Because displacement is continually changing during play, the force must be continually updated at a very fast rate. For example, as you move a force feedback joystick away from center position, a simulated spring sensation provides a restoring force that increases linearly with that displacement. As you bring the joystick back towards center, the restoring force decreases linearly. The steeper the curve that relates force and displacement, the "stiffer" the simulated spring will feel. This is depicted with the simple graphical relation between force as displacement indicative of a spring sensation:

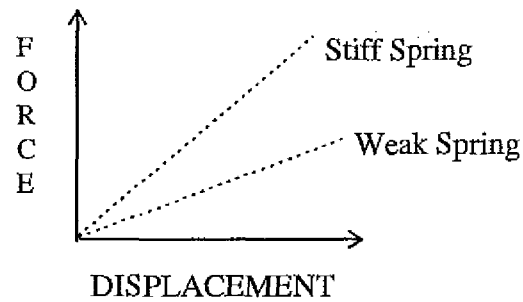


Figure 2-2 Force Displacement Profiles

Now, consider that a spring sensation is simulated by rapidly reading displacement sensors (input) and commanding forces (output) based on those sensor readings. The quality of such a sensation will be greatly effected by delays between input and output. For example, if you were compressing a simulated spring by moving a force feedback joystick and it took 2 seconds to read sensors, compute the force output based on the spring model, and then command forces back to the peripheral device, you would move the joystick to a given position and have to wait 2 seconds for it to feel like the spring was compressed to that location. This 2 second delay would greatly corrupt the feel sensation - especially if you were moving the joystick back and forth within the spring and you only felt force changes every 2 seconds and they were 2 seconds late. As you might expect, this will not feel like a smooth spring being compressed, it would feel jerky and erratic.

APPLE INC.
EXHIBIT 1017 - PAGE 46

Ideally, to make a spring sensation with high quality interactive feel you want to read sensors and control the motors very quickly. The faster you can read sensors and update the forces, the smoother the spring sensation. This "loop" of reading sensors and updating forces is called the "**Update Rate**" of the force feedback sensation. It is typically reported as how many times per second the loop can be completed. Sensations such as springs and dampers that are highly interactive require update rates on the order of 1000Hz. In other words, delays between reading sensors and controlling motors that are as small as 2 or 3 milliseconds are enough to greatly corrupt the feel of physically interactive sensations like the Spring.

So, the perceptual requirements of quality feel sensations can be summarized as two simple rules of thumb.

i. The lag time between the onset of a visual event and the onset of an associated force event should be less than 25 milliseconds or else the user will notice a disconcerting perceptual delay.

ii. The update rate in a real-time control loop where force is continually updated based on user motion (sensor readings) of a peripheral device should be on the order of 1000 Hz for high quality feel simulations and should never fall below 500 Hz – if so, stability distortions will greatly corrupt the feel.

As a programmer, the above update rate requirement is probably a concern – clearly you do not want to have your software read sensors and update forces 1000 times per second, especially not across a slow communication channel. This would divert substantial processing from your game and stress the communication channel to its limit. Fortunately, the manufacturers of most force feedback hardware devices have adopted a technology that solves this problem for you. The solution is achieved by having a **local microprocessor** on board the force feedback device that performs the high speed computations in parallel with host execution of the gaming software. The following Figure 2-3 is a high level depiction of this "distributed processor architecture".

APPLE INC.
EXHIBIT 1017 - PAGE 47

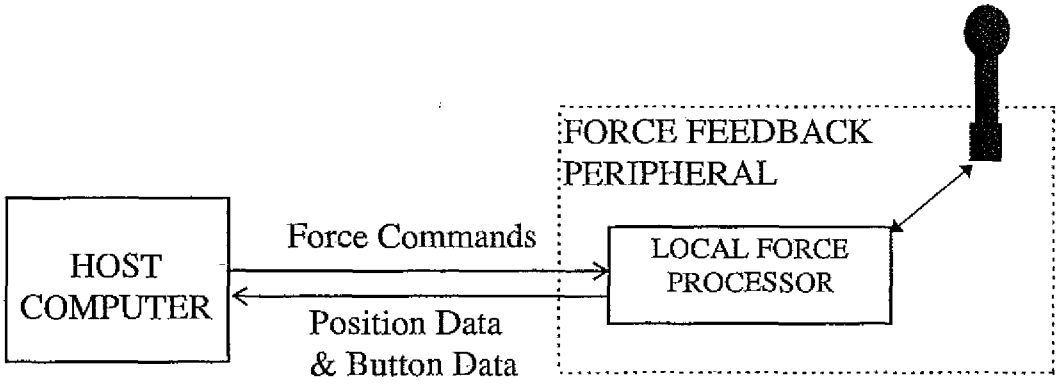


Figure 2-3 Distributed Processing Architecture

Figure 2-3 depicts the basic architecture of most high quality force feedback computer peripherals. The local processor contains sophisticated firmware that makes the device intelligent enough to generate force feedback sensations based on high level commands from the host computer. For example, the I-FORCE processing core that is incorporated in peripheral devices from many manufacturers has the ability to create the spring sensation described above based on a simple high-level command from the host that includes basic parameters such as the "stiffness" of the spring and the "offset" location of the spring. It also provides for advanced parameters such as the "saturation" of the spring and the "dead-band" of the spring. All of these parameters will be described in detail later in this text, demonstrating how such a simple high level command allows programmers great flexibility in defining feel sensations.

APPLE INC.
EXHIBIT 1017 - PAGE 48

Referring to the detailed architectural overview diagram in Figure 2-4, we see that a force feedback device, using a local microprocessor to assist in force feedback sensation generation, allows for two "control loops" that function in parallel. One control loop is called the "host control loop" wherein the host computer reads sensors, updates the gaming application, and commands high level force sensations back the peripheral. If this loop runs too slow, users will notice a delay between what they see on the screen and what they feel through the peripheral. To maintain coordination between visual and physical events, this loop should run on the order of 100Hz.

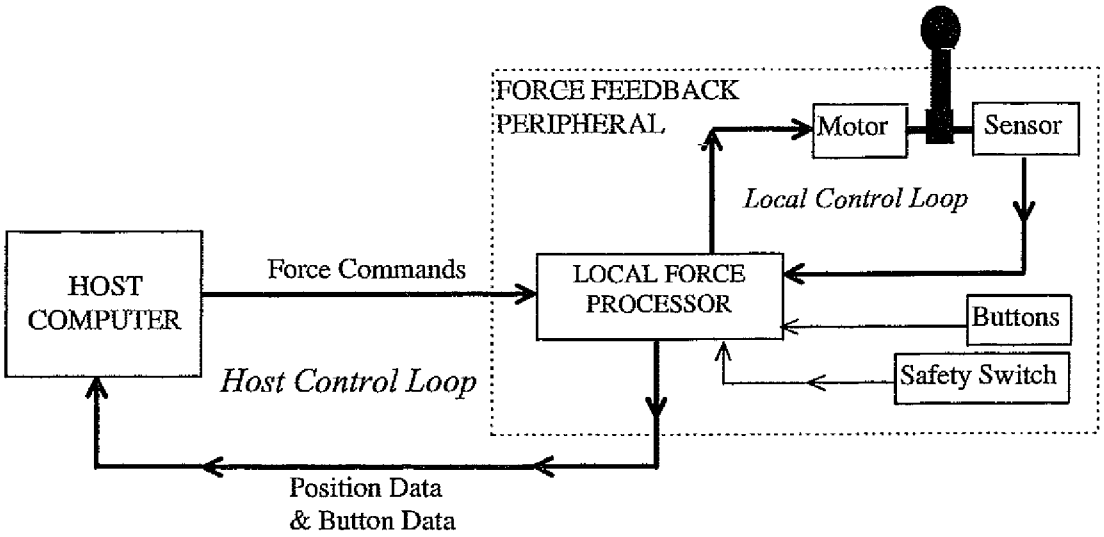


Figure 2-4 Control Loops in the Distributed Processing Architecture

The other control loop is called the "local control loop" wherein the local force processor reads sensors, computes force sensations, and controls the motors. To generate "high fidelity" force sensations, this loop should run on the order of 1000hz. The Host Control Loop and the Local Control Loop are coordinated through high level commands issued from the host to the local processor. These commands allow for a great diversity of sensations, from simple jolts to complex interactive sensation. In the following Section we will introduce the broad *classes* of sensations known as **Conditions**, **Waves**, and **Dynamics** and describe the type of commands and parameters that programmers can use to coordinate such sensations.

APPLE INC.
EXHIBIT 1017 - PAGE 49

Local Triggers: Also shown in Figure 2-4, the local processor also functions to monitor the buttons, hat switches, throttles, and other controls local to the peripheral. Because the local processor has access to the state of all these controls, the processor can use their state to "trigger" local force feedback sensations. The technique of *local triggering* is what enables the "reflex" concept introduced in Chapter 1. The actual method of using local controls such as buttons and hat switches to automatically trigger sensations on board the force feedback device is described in Chapter 6. As you will find when you review that Chapter, a simple *Trigger Button* parameter can be set when defining a feel sensation to establish the button reflex relationship.

Local Safety Features: Many force feedback peripheral devices include a safety switch, sometimes called a "dead-man" switch, that reports if a user is properly engaging the device with his or her hand. If a user is not holding the device properly, the hardware will not produce forces. This prevents the device from moving around when not being attended. Programmers should note that high quality force feedback devices will include a feature known as "safety fade-in" wherein forces are faded-in rather than abruptly returned when the safety feature is engaged. The advantages are obvious, when a user grabs a device and engages the safety feature, forces should be restored, but if they are restored abruptly the joystick or wheel might jump out of the users hand before he or she firmly grasped it. For example, the local microprocessor within most shipping I-FORCE class devices performs an intelligent safety fade-in feature

Local Memory Limitations: Many force feedback hardware products have local memory within the product that is accessible by the Local Force Processor. This memory is used for storing the parameters associated with commands sent from the host computer, storing button assignments for Reflexes, storing data for digitized force profiles, and for storing run-time variables used in generation of sensations. Because force feedback hardware devices are typically low-cost products, memory is usually limited. As a result, the functionality of a force feedback device is limited by memory resources. For example, most force feedback hardware devices are limited in how many sensations can be generated simultaneously. This means that you should not attempt to store large numbers of effects locally at the same time. If you try to command the force feedback device to create too many force feedback sensations simultaneously, the device will report an creation error when asked to load an effect that exceeds available memory. The implication of this is that effects should only be created when they are needed and should be destroyed when the are no longer useful. If an application keeps many extraneous

APPLE INC.
EXHIBIT 1017 - PAGE 50

(unused) effects that it has created, these effects will be using device memory that could be used for other effects.

Local Bandwidth Limitations: As described previously, many force feedback hardware peripherals provide enhanced performance by using a local microprocessor to off-load processing burden from the host computer. While this technique is very effective in providing an easy to use force feedback architecture that does not burden the host, the processors employed by force feedback products are low-cost and therefore limited in processing power. As a result, force feedback devices are limited in how many sensations can be generated simultaneously based on the available processing resources of the local processor. Different hardware products have different limitations.

APPLE INC.
EXHIBIT 1017 - PAGE 51

2.3 Overview of Force Feedback Sensations

Because force feedback devices can produce such a wide variety of feel sensations, each with its own unique parameters, constraints, and implementation issues, it is very helpful to subdivide the overall spectrum of sensations into manageable subsets. On the highest level, it is very helpful to define the following three unique and distinct classes of feel sensations **Spatial Conditions, Temporal Waves,** and **Dynamic Sensations.** These will more generally be referred to as:

- **Conditions** - Forces that are a function of user motion
- **Waves** - Forces that are a predefined profile played back over time
- **Dynamics** - Forces that are based on an interactive dynamic model of motion and time

2.3.a Spatial Conditions

"Conditions" describe the basic physical properties of an interface device based on spatial motion of the interface. For example, a joystick device has basic properties such as Stiffness, Damping, Inertia, and Friction in the joystick handle. These elementary *conditions* define the underlying feel of handle motion during general manipulation. Conditions can be used to make a joystick feel "loose", "stiff", "heavy", "light", "sticky", "scratchy", "slippery", etc. Conditions can also be barriers or obstructions that restrict spatial manipulation of the stick. These are usually called Walls and they can be "hard" or "soft" barriers. Conditions can also be Textures, and may feel "rough" or "smooth".

Programmers use conditions to *tune* the general feel of the device based upon game parameters. For example, when flying an F-16 fighter, the joystick handle might be made to feel very *stiff* and *heavy*. When flying an old Spitfire, the joystick handle might be made to feel *loose* and *light*. When the craft is damaged by an enemy fire, the joystick handle might be made to feel *sticky* with a *scratchy* texture.

APPLE INC.
EXHIBIT 1017 - PAGE 52

Overall, it should be understood that conditions are typically *not* associated with discrete sudden events during game play, but rather are background conditions of game play, hence the name "conditions". A condition is usually an environmental feel that is set-up and experienced over an extended period. For example, when your aircraft lifts off the runway, a set of conditions would be engaged that represent the feel of that particular plane. Such conditions are felt by the user for an extended period, but will be updated as gaming events change. For example, when the wing-flaps are raised, the stick might be made to feel more stiff. Another good example is a driving game – depending upon what car is being driven, a set of conditions will define the stiffness of the wheel, the range of motion of the wheel, even the damping and friction in the wheel. When the car is air-borne off a jump, these conditions may change to simulate the feel of the free tires. When driving in mud, the conditions may change again. Thus conditions, while typically used over extended periods of time, can be highly tuned to changing game events.

2.3.b Temporal Waves

"Waves" are feel sensations that are closely correlated with discrete temporal events during game play. For example, a shuttle-craft is blasted by an alien laser and the user feels a physical blast that is synchronized with graphics and sound that also represent the event. The *jolt* will likely have a predefined duration and possibly have other parameters that describe the physical feel of the event. The blast may even be represented by a complex *force profile* that defines a feel unique to the weapon that was fired. While discrete, Waves can have a substantial duration – for example, a small motor boat is caught in the wake of a huge tanker, the bobbing sensation may be an Wave that lasts over an extended period and may vary over time.

Waves are best thought of as time related sensations such as *vibrations* and *jolts* that are "overlaid" on top of the background conditions described above. For example, in the driving example given above, a given car may have Conditions that define the "feel" of the wheel such as the stiffness, damping, and friction in the wheel. But, when the car hits a pot hole or bumps another car, the discrete jolt that is overlaid on top of the background sensations are Waves. In addition, the subtle motor-hum, felt as the engine is revved, is another Wave overlaid on top of the background conditions.

APPLE INC.
EXHIBIT 1017 - PAGE 53

The biggest drawback of Waves is that they are pre-defined sensations that are simply played back over time. While conditions are highly interactive sensations that respond to user motion, Waves are canned force profiles that execute the same, regardless of how the user responds. Nevertheless, effects can be very effective when well coordinated with gaming events. The key is to use canned Waves only where appropriate and to use Conditions and Dynamics to provide the interactive richness.

2.3.c Dynamic Sensations

While **Conditions** and **Waves** have been part of force feedback programming since the launch of the first commercial force feedback API in 1995, **Dynamics** is a new innovation just recently made possible by advances in the hardware processors employed within force feedback devices. Most force feedback devices that will launch in 1997 will support embedded Dynamics. For example, all hardware products that support the latest generation of I-FORCE engine, currently I-FORCE 2.0, include what is called a *Distributed Dynamic Processing Engine*. This local software engine allows complex Dynamic Sensations to be executed at high speeds in parallel with host executing of gaming events. High level commands allow the host to coordinate the feel and execution of the Dynamic Sensations with gaming interactions.

Transforming Games into Sports: Imagine the feel of a ball impacting a racquet, compressing the strings, and then bouncing off with a snap. It is a very compelling force feedback sensation. It is also a sensation that provides critical real time information that allows players of racquet sports to impart subtle control over the ball. It is also a great example of a "dynamic" sensation that can not be represented by a simple "canned" profile that is predefined and played back over time. This is simply because how the user interacts with the ball *during* the event greatly changes the feel. The user could cushion the ball to a halt by absorbing energy in the wrist, could whip the ball sharply off the paddle by tightening the wrist with a snap, or could sling the ball off to the side with a flick of the arm. Although the entire event may only last 500 milliseconds, the subtle feel of the continuously changing forces during the interaction is very important to making a realistic sensation.

APPLE INC.
EXHIBIT 1017 - PAGE 54

With the new level of realism provided by force feedback with Dynamic Sensations, feel becomes not just a embellishment but an integral part of the gaming experience – it lets users take advantage of their inherent sensory-motor skill and dexterity to optimize their control. The implications are profound, for video games can finally take on a level of physicality reminiscent of real sports. Dynamic Sensations will be described in detail in Chapter 5 and are most easily understood by grabbing hold of a force feedback device that supports Dynamics and feeling it.

APPLE INC.
EXHIBIT 1017 - PAGE 55

2.4 Force Streaming

While the makers of force feedback hardware provide programmers with a wide variety of predefined feel sensations such as the Conditions, Waves, and Dynamics introduced in the previous sections, some programmers will inevitably want to create force feedback sensations from scratch by manipulating forces directly. In other words, rather than using the mathematical relationships between *force and motion* or *force and time* provided by hardware makers, some programmers may want to modulate force directly from the host. This can be achieved using the **Vector Force** or **Constant Force** commands described in section 8.3.

For example, the Vector Force function allows a programmer to command a force of a given magnitude and direction upon the force feedback hardware. By continually updating the magnitude and direction from the host in real time based on sensor readings reported from the hardware, the host can create it's own complex force sensations. This technique is often called "closing the loop from the host" since the force feedback control loop is not being performed by the local microprocessor on board the hardware but rather being performed by the host.

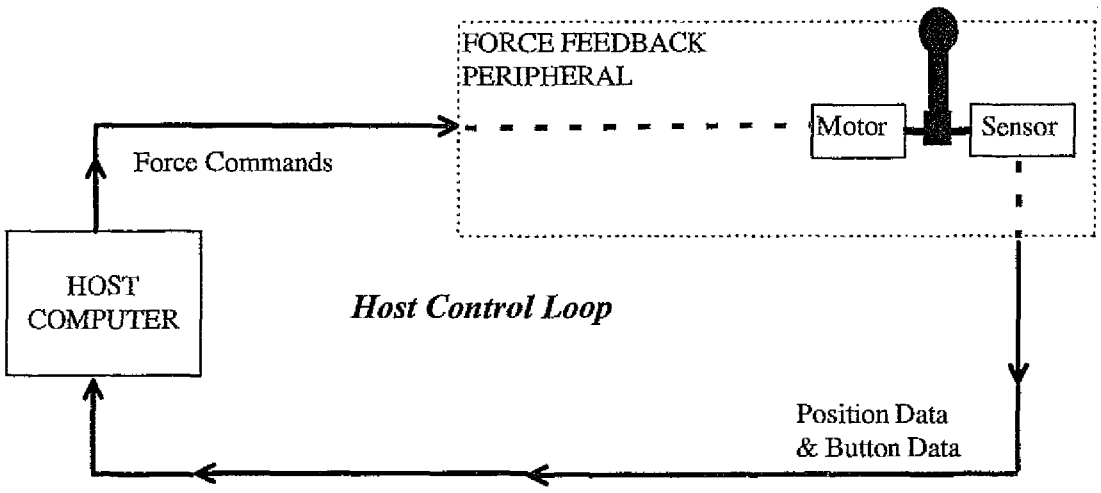


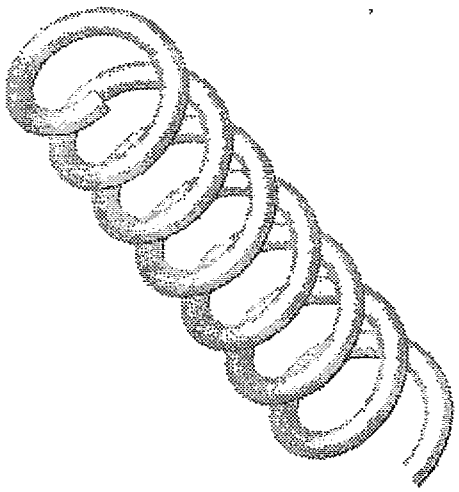
Figure 2-5

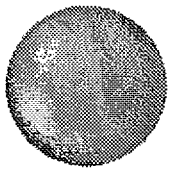
APPLE INC.
EXHIBIT 1017 - PAGE 56

As you might have guessed, there are two draw-backs of this approach. First, the host needs to read the sensors and control the forces very rapidly (1000 times per second or as close to that rate as you can achieve). Therefore, closing the loop from the host is a computational burden that can slow game play. Second, the communication bus between the host and the hardware must work very hard to read sensors and command forces at such a rapid rate. This can be a bottleneck that limits fidelity of the feel sensation and further burdens the host.

While the first problem mentioned above is only solved through faster host processors and more efficient operating systems, the second problem is solved on force feedback devices that employ high speed communication means such the Universal Serial Bus. Using the USB, the many force feedback hardware devices can transmit force data at very fast rates. For example, devices that use the I-FORCE processing core and use the USB employ a method called **Force Streaming** where data is conveyed in a highly efficient and rapid manner. Nevertheless, we do not recommend closing the loop on the host unless it is absolutely necessary and you know you can handle the additional host processing overhead. For more information on Force Streaming, refer to your I-FORCE 2.0 device developer documentation included with all USB I-FORCE units.

APPLE INC.
EXHIBIT 1017 - PAGE 57





3. Understanding Spatial Conditions

You are 10 laps away from winning a stock-car qualifier in Indianapolis. As you scream into the most treacherous turn, force feedback steering wheel fights you with a stiff centrifugal force that helps you judge your speed and maintain your bearing. As you fly into the straight-away, the wheel suddenly lightens up, the hot rubber of your tires are slipping on the track surface. Being the skilled driver that you are, you can judge by the feel of the wheel that it is time for a pit stop. You pull-in and your crew outfits your car with a new set of tires. As you race out of the pit, you know you made the correct decision – your car now feels tight and maneuverable.

APPLE INC.
EXHIBIT 1017 - PAGE 58

3.1 Overview

As described in Chapter 2, Conditions are best thought of as the "background physical properties" of the force feedback interface device. Conditions define the interactive feel of the controller as it is moved throughout the workspace. For example, Conditions include the *stiffness* felt as a joystick is moved away from the center of the workspace, the *damping* felt when a steering wheel is spun quickly, or the *texture* felt when a flight-yoke is pulled back into a certain region. Conditions are very compelling physical sensations because their feel is so dependent upon user interaction. This is because Conditions generate forces as a function of the spatial motion of the interface device as caused by the user.

Some Conditions create force sensations that are a function of interface position, some create force sensations that are a function of interface velocity, and some create force sensations that are a function of interface acceleration. At the present time, there are a standard set of condition types supported by most force feedback hardware. These include **Springs, Dampers, Inertia, Friction, Texture, Walls,** and **Barriers.** These types can be defined briefly as follows:

Spring:	A restoring force that feels like stretching or compressing a spring
Damper:	A drag resistance that feels like moving through a viscous liquid
Inertia:	An inertial sensation that feels like moving a heavy mass
Friction:	A simple rubbing resistance that encumbers free motion.
Texture:	A spatially varying resistance that feels like dragging over a grating
Wall:	A one-sided obstruction that feels like an impenetrable hard stop.
Barrier:	A two-sided obstruction that feels like a penetrable surface.

Generating Conditions of the above categories simply involve specifying the condition type and defining the unique physical properties associated with that type. For example, the most basic **type** of Condition is a *Spring* and the fundamental physical property is the

APPLE INC.
EXHIBIT 1017 - PAGE 59

stiffness. The resulting sensation will be a restoring force upon the interface that resists motion away from the spring origin. The stiffness defines how quickly the force increases with motion. Additional parameters can further customize the feel of the spring by adjusting the location of the spring origin, by assigning which axis or axes the spring is applied to, by limiting the maximum force output of the spring sensation, etc. All of these physical properties are the **parameters** for the stiffness sensation. By defining these parameters, a wide variety of feels can be created. By combining multiple springs, even more diverse sensations can be defined. By combining spring sensations with other conditions such as textures and friction, the diversity grows further.

At the highest level, generating a Condition can be conceptualized as assigning the following definitions:

Type:	Depending upon hardware capabilities, choose one of the basic condition types such as Spring, Damper, Inertia, Friction, Texture, Wall, Barrier, etc...
Axis:	The axis or axes effected by the condition being defined: X, Y, XY, XZ, etc...
Direction:	For multi-axis conditions, you can define a direction. The direction conditions are described in Section 6.5.
Parameters:	These define the physical properties associated with the given condition type. The parameter set depends upon the type of condition chosen. For Spring, this would include *stiffness*, for Damper, this would include *damping*, etc...

To fully understand the power and potential of Conditions, we will describe below each of the condition types and the parameters associated with them. Once you understand this basic conceptual framework, you can use the programming model defined in Chapter 6 to actually implement these sensations.

APPLE INC.
EXHIBIT 1017 - PAGE 60

3.2 Spring

Spring is a Condition that defines a physical property best described as the stiffness of the device. Lets think first in terms of a single axis of a peripheral device, such as the x-axis of a joystick or the wheel-spin axis of a steering wheel. A joystick axis with a high stiffness will feel as if a strong spring resists displacement of the handle. A steering wheel axis with a low stiffness will feel loose, as if a weak spring resists displacement of the wheel.

One way to understand the spring sensation is to think of the mathematical relationship between force and displacement. A spring is generally modeled using *Hooke s Law* where resistance force (F) is simply proportional to the displacement (d). The proportionality (k) is generally called the stiffness or "spring constant".

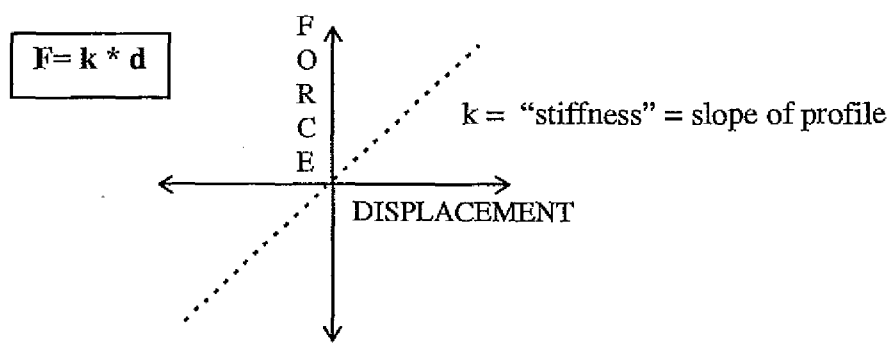


Figure 3-1 Hooke s Law: Spring Forces

A number of parameters can be used to fine tune the feel of the spring sensation. Most importantly, different values of stiffness (k) can be defined for positive and negative displacements of a device axis. For example, the spring can be defined with a +k and -k such that it is much harder to push the joystick forward than it is to pull it back. This might be used to simulate the feel of a tug boat pulling a heavy load.

APPLE INC.
EXHIBIT 1017 - PAGE 61

In addition, other parameters can also be used such as the location of the center of the simulated spring and the maximum-minimum allowable force values. All relevant parameters for a general Stiffness condition are described below and shown graphically:

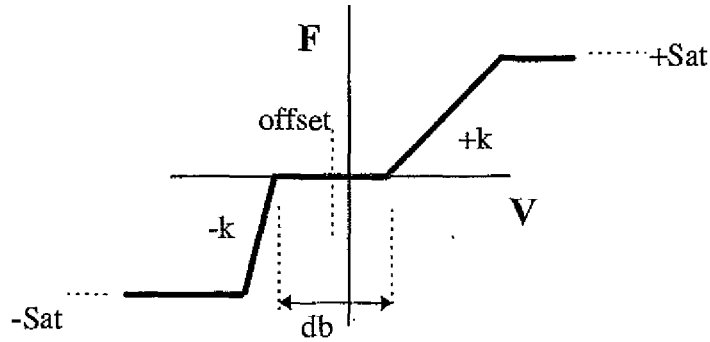


Figure 3-2 Generalized Spring Profile

+k: The spring constant (k) on the positive side of the simulated neutral position

-k: The spring constant (k) on the negative side of the simulated center

offset: The percent distance of the simulated spring "center" from the axis' zero origin

db: deadband - The range around the "center" where the spring condition is not active

+Sat: Positive Saturation - The maximum positive force output

-Sat: Negative Saturation - The maximum negative force output

Another way to conceptualize the Spring Condition is to think of the physical metaphor rather than the mathematical relationship between force and displacement. The following diagram, shows a physical representation of a single axis of a force feedback peripheral device. This figurative representation shows the handle of the object free to move in either the positive or negative direction along the given axis and encounter either a positive or negative stiffness.

APPLE INC.
EXHIBIT 1017 - PAGE 62

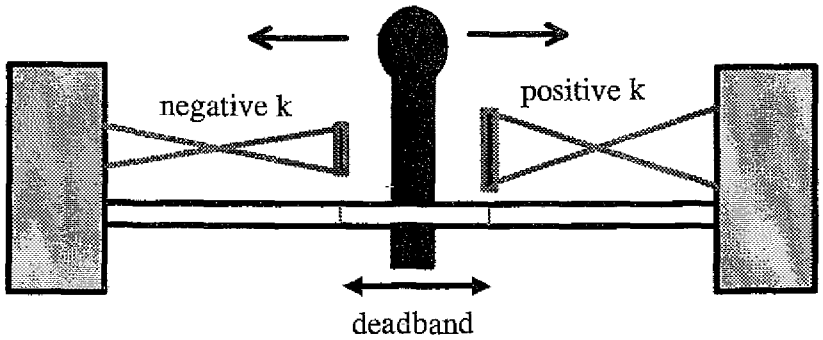


Figure 3-3 Visualization of a Single Joystick Axis with Springs

Along each axis, there is a spring as defined by a positive stiffness parameter (k) and a negative stiffness parameter (k). Graphically we have represented a big stiffness as a larger spring. The origin of this condition is shown at the center, but could be **offset** as defined by an offset parameter. A **deadband** region is shown graphically as the gap between the handle and the spring. As you could imagine, if the handle moved to the left, it would have to move some distance before encountering the spring and compressing it. If there was no gap (no deadband), the spring would immediately start compressing as soon as the handle was moved.

To make the physical metaphor more clear, Figure 3-4 shows what happens as the user moves the interface device in the positive direction along the axis, encounters the spring stiffness in the positive direction and compresses the spring.

APPLE INC.
EXHIBIT 1017 - PAGE 63

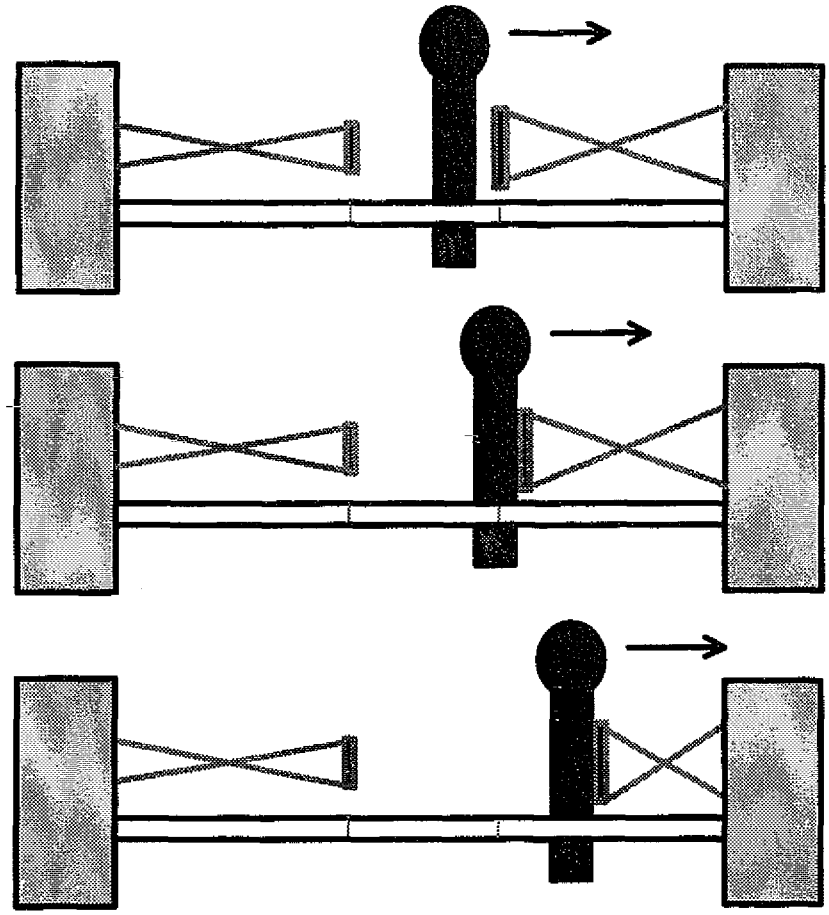


Figure 3-4 Visualization of a Spring in one Axis

The above diagram demonstrates that no spring resistance is felt when the user moves the interface within the deadband region. Once the positive spring stiffness is encountered, the resistance force increases linearly with compression of the spring (as is true of a real spring). If the programmer had defined a saturation value for the positive direction, the force output would cease increasing with compression once the saturation limit in the positive direction was exceeded. This simple physical metaphor makes the spring Condition clear.

APPLE INC.
EXHIBIT 1017 - PAGE 64

Two Types of Spring Sensations:

Axis Spring: In some cases, it is desire that the Spring Condition be associated with a single degree of freedom of the force feedback device such as the X axis of a joystick, the rotation axis of a steering wheel, or the Y axis of a flight yoke. For these situations where the spring is oriented along a primary axis of the hardware device, the spring can be thought of as an "axis spring".

Angle Spring: For cases where the Spring Condition is associated with multiple degrees of freedom of the force feedback device, such as both the X and Y axes of a joystick, the Spring Condition must be assigned an *angle* to dictate how it is oriented within the multi degree of freedom space. For example, a spring can be assigned a 45 degree angle as shown in Figure 3-5 below with respect to the X-Y workspace of a standard joystick interface. This angle parameter is very useful for generating interesting off-axis sensations.

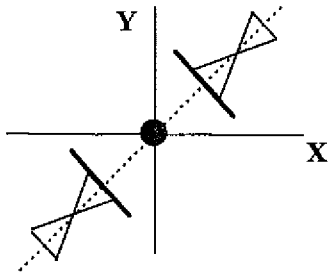


Figure 3-5 Angle Spring

The easiest way to create both Axis Spring and Angle Spring sensations is to use the I-FORCE Studio toolset as described in Chapter 9. These tools provide a simple graphical environment where you can manipulate spring parameters and feel the results in real-time.

One last thing to keep in mind is that multiple spring Conditions can be present on a single axis of a force feedback peripheral device. In such a case, the resisting force imposed by the springs will simply sum. Of course if the springs are defined with different offsets, deadband regions, and saturation values, the summation of the multiple springs can give a very interesting sensation where the resistance either increases or drops at different locations in the range of travel of the interface device.

APPLE INC.
EXHIBIT 1017 - PAGE 65

3.3 Damper

Damper is a Condition that defines a physical property that is best described as the sensation of moving an object through a viscous fluid. A device axis with high "damping" will feel as though it is moving through a thick fluid such as honey. A device axis with low "damping" will feel as though it is moving through a thin medium like air. When heavy damping is applied to an interface like a joystick or steering wheel, the handle will feel sluggish or unresponsive because the damping resists quick user motions.

Like the Spring Condition, the Damping Condition is a force sensation that is computed as a mathematical function of motion. While the computed force level for a Spring sensation increases with *displacement* of the interface device, the computed force level for a damper sensation increases with *velocity* of the interface device. In other words, the faster the handle of a joystick moves through a simulated damping, the stronger the resistance felt by the user. This makes intuitive physical sense – for example, if you stir thick honey slowly, the resistance you feel is low. But if you try to stir thick honey quickly, the resistance you feel is high. This is because viscous fluids resist disturbance based on the velocity of the disturbance.

One way to understand the Damper sensation is to think of the mathematical relationship between force and velocity. A damper is generally modeled as *Viscosity* wherein the resistance force (***F***) is simply proportional to the velocity (***v***). The proportionality (***b***) is generally called the stiffness or "damping constant".

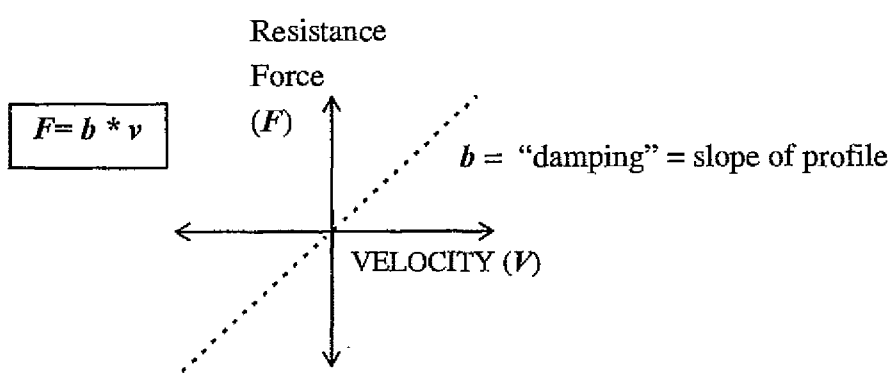


Figure 3-6 Damping: Resistance versus Velocity

APPLE INC.
EXHIBIT 1017 - PAGE 66

In some cases, damping is represented with slightly different mathematical relations between force and velocity. For example, sometimes damping is represented as a function of the velocity squared. Regardless of how damping is represented, the fundamental aspect of all damping sensations is that the resistance felt by the user increases substantially as with user velocity.

A number of parameters can be used to fine tune the feel of the damping sensation. Most importantly, different values of damping (***b***) can be defined for positive and negative velocities along of a device axis. For example, the damper can be defined with ***a+b*** and ***-b*** such that it is much harder to quickly push the joystick forward than it is to pull it back.

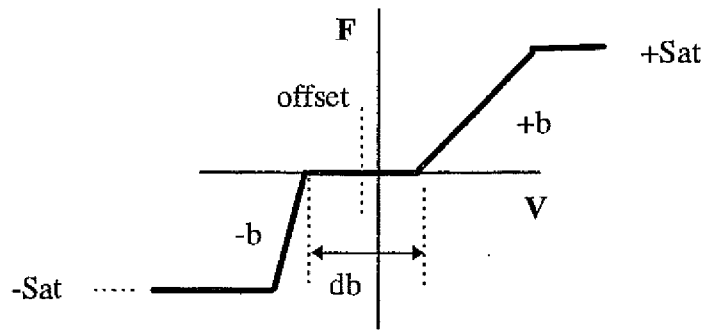


Figure 3-7 Generalized Damping Profile

Other basic parameters include *offset*, *deadband*, and *saturation* values. All of these parameters are shown above in Figure 3-7 and are described below.

+b: The damping coefficient (***b***) used for positive velocities

-b: The damping coefficient (***b***) used for negative velocities

offset: The velocity value for which damping force is zero.

db: Deadband range centered around the "zero" where the physical damping is not active

+Sat: Positive Saturation = The maximum positive force output from damping

-Sat: Negative Saturation = The maximum negative force output from damping

APPLE INC.
EXHIBIT 1017 - PAGE 67

3.4 Inertia

Inertia is a Condition that defines a physical property best described as the feel of moving a heavy mass. A device axis with a high "Inertia" will feel *heavy* and a device axis with low Inertia will feel *light*. High Inertia should feel like your joystick or steering wheel is filled with sand and you are now carrying the weight as you move the interface around. This can be a very compelling sensation that simulates the feel of piloting a large craft or dragging a heavy weight.

One way to understand the Inertia sensation is to think of the mathematical relationship between force and acceleration. It is generally modeled as an inertial force using the very well known Newton's Law represented as **F = m a**. wherein the resistance force (F) is simply proportional to the acceleration (a). The proportionality constant (m) is simply the mass. So, to make an interface device feel heavy, just define a high mass. To make an interface device feel light, define a small mass. Inertia is a very simple Condition to use.

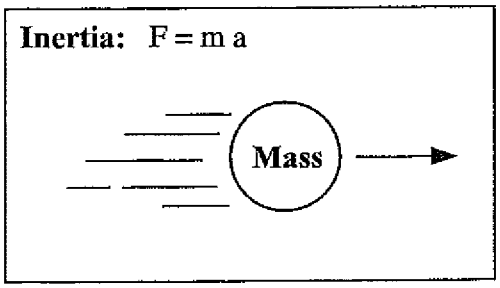


Figure 3-8 Inertia

APPLE INC.
EXHIBIT 1017 - PAGE 68

3.5 Friction

Friction is a Condition that defines a physical resistance sensation representing the feel of sliding an object against a uniform frictional surface. Like damping, friction force always resists the direction of motion. Unlike damping, friction force has a *constant* magnitude regardless of the velocity. In other words, friction is a force sensation where in the direction of the force is dependent upon the *sign* of the velocity, but the magnitude of the force is constant. This is represented below in the following simple diagram, Figure 3-9.

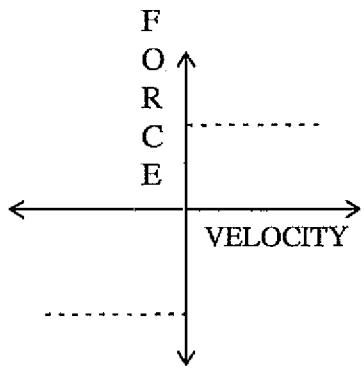


Figure 3-9 Force vs. Velocity for Friction Condition

The magnitude of the frictional resistance is defined by a parameter known as the **Friction Coefficient**. Of course a number of parameters can be used to fine tune the feel of the friction sensation. Most importantly, different values of friction coefficient (f) can be defined for positive and negative velocities along a device axis. For example, the friction can be defined with a +f and -f such that it is much harder to turn a steering wheel left than it is to turn it right.

It is important to note that there is a fundamental problem with the pure mathematical friction sensation shown in the diagram above – namely that when you change direction you encounter a sudden sharp force. This situation is usually not desired. Therefore, most force feedback devices employ simulated friction that have a smooth transition where force is proportional to velocity when velocity is very small.

APPLE INC.
EXHIBIT 1017 - PAGE 69

3.6 Texture

Texture is a spatial Condition which give the user a feel similar to dragging an object over a rough surface like a metal grating. Textures are similar to vibration Waves in that they result in a periodic force signal felt by the user. The big difference between a texture and a vibration is that a texture creates a periodic force signal that varies based on the *spatial motion* of the interface whereas vibrations are periodic forces that are simple functions of time. In other words, vibrations are canned routines "played" over time while textures create interactive spatial environments wherein the feel is highly dependent upon user motion. For this reason, textures are very compelling force feedback sensations.

A number of simple parameters can be used to tune the feel of a texture sensation. These parameters include the *Roughness*, *Spacing*, and *Density*. By adjusting each of these values, programmers can define a variety of texture sensations with great flexibility. The meaning of each parameter is described concisely below.

<u>Roughness</u> Roughness defines the intensity of the texture sensation. Since this is an interactive effect, the strength of the sensation also depends on how the user moves the interface through the texture environment. You can think of Roughness (R) as the grit of sandpaper, the feel depends both on the grit and how you rub your hand over it. It should be noted that high quality force feedback devices allow you to define a positive roughness (+R) and a negative roughness (-R) such that the intensity of the texture depends upon which direction the interface device is moving.

<u>Spacing</u> Spacing defines the center to center spacing between the "bumps" in the texture. The smaller the spacing the finer the texture.

<u>Density</u> Density defines the width of the "bumps" in the texture with a range of 1 to 100 where 50 means 50% of the center to center spacing of the bumps. A small density means the bumps are small with respect to the empty space between them. A big

APPLE INC.
EXHIBIT 1017 - PAGE 70

density means the bumps are large with respect to the empty space between them. The implications of this parameter are best understood by simply feeling a force feedback device.

The following diagram, Figure 3-10, is useful in helping to convey the implications of a Texture Condition. As you can see, a Texture gives a periodic spatial grating that is felt as the user moves the interface device through a region. A different intensity can be felt in the positive and negative directions.

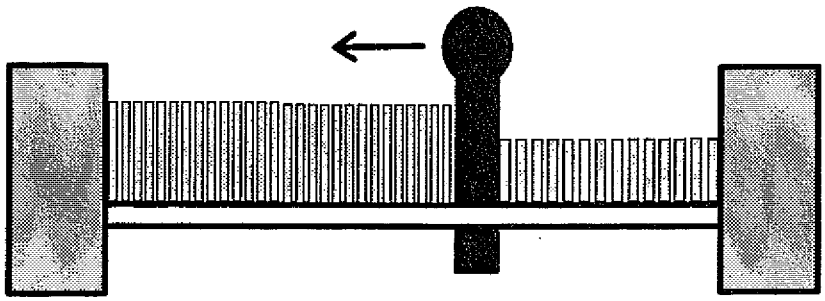


Figure 3-10 Visualization of Texture Condition

APPLE INC.
EXHIBIT 1017 PAGE 71

3.7 Wall

Wall is a Condition that creates the feel of encountering a hard linear surface within the range of travel of the force feedback interface device. For example, if you are using a force feedback joystick and you move the joystick into a simulated Wall, you will feel a hard obstacle when the stick is moved perpendicular to the wall (pushed into the wall) and you will feel a smooth surface when the stick is moved parallel to wall (rubbed along the wall). The wall will stop providing resistance forces if the stick is backed away from the wall. The wall will re-apply forces if the stick re-encounters the wall. In principle the wall sensation is very simple and very effective.

Mathematically, a Wall is represented as a force that increases rapidly with displacement upon encountering the defined location of the wall boundary. Looking at the diagram below, you can imagine a joystick handle starting at rest in the center of its workspace. The joystick moves to the right. No force is felt until the wall location is encountered (where the dotted line meets the horizontal axis on the figure). Once the joystick crosses the location of the wall, the force increases very rapidly with displacement, creating a force that pushes back, resisting the joystick from penetrating the wall.

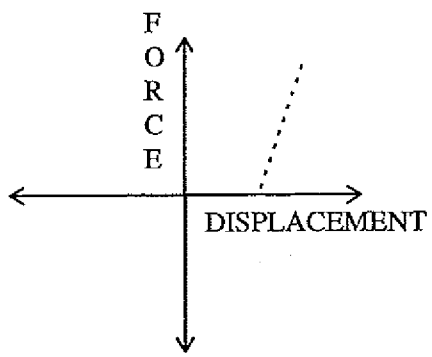


Figure 3-11 Force and Displacement for a Simulated Wall

The dotted line in Figure 3-11 represents the relation between force and displacement. It is a steep line, representing a force that increases quickly with penetration into the simulated wall. The steeper the force increases with displacement into the wall, the "harder" the wall will feel.

APPLE INC.
EXHIBIT 1017 - PAGE 72

An even simpler way to conceptualize a Wall sensation is to think of it as a physical-stop placed in the path of travel of the interface device. The physical-stop could be hard like metal or soft like rubber, but it is still a physical-stop. Figure 3-12 below depicts this conceptual representation of the hard-stop:

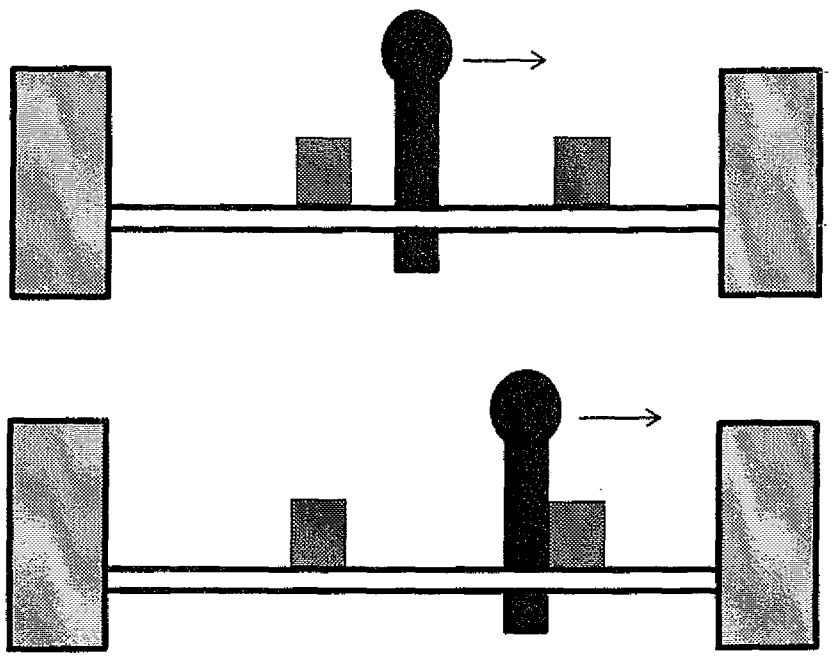


Figure 3-12 Visual Conceptualization of a Hard Stop

Of course, programmers do not have to worry about the mathematical relationship between force and displacement in order to define a Wall sensation. The Wall sensation has been abstracted to simple parameters that can be rapidly defined by programmers. As you would expect, a Wall is defined in terms of its *hardness*, its location, and its orientation. In fact, there are a number of central parameters that make the Wall sensation very diverse and extremely powerful. Each parameter is described below:

<u>Hardness</u> — Hardness is the basic physical parameter for a Wall Condition. It describes how rigid or compliant the surface of the wall feels when pushed against. It is very important to understand that a force feedback device CAN NOT simulate

APPLE INC.
EXHIBIT 1017 - PAGE 73

the feel of a truly "rigid" wall because the force output capability of the hardware is limited. Most force feedback joysticks made for consumer applications produce, at maximum, less than 1 pound of force. This means when a user pushes a force feedback joystick into a simulated wall, the joystick can only resist penetration with a limited resistance force.

Sense — Sense is a binary parameter defined as either positive or negative. A wall with positive sense will resist crossing of the wall when moving in a positive direction along an axis. A wall with negative sense will resist crossing of the wall when moving in a negative direction. Another way to look at this: Sense defines which way the wall is facing.

Offset — Offset is defined in one of two ways, either it is the distance away from the origin of the peripheral device's range of motion (i.e. the center of the joystick space), or Offset is defined as a distance from the current location of the peripheral (i.e. the location of the joystick at the instant that the command is called). In the first instance, Offset is an absolute value and in the second instance the Offset is a relative value. An *Offset-Flag* is used to indicate which way you want to define your wall location.

Surface_Friction — Surface Friction is a parameter that governs surface damping felt by the user when *rubbing* along the length of the Wall surface. If the Surface_Friction is low, the wall will feel like a smooth surface, like glass or ice. If the Surface_Friction is high, the wall will feel less smooth, as through it was coated in rubber. Note – high quality force feedback devices employ a method called "pressure mode" when generating Wall sensations that greatly enhances the realism of the Surface Friction feel. Pressure mode modulates the intensity of the rubbing resistance based on how hard the user is pushing against the wall. In other words, the resistance felt by the user while rubbing along a surface is dependent upon the pressure exerted by the user against the surface. As a

APPLE INC.
EXHIBIT 1017 - PAGE 74

result, when a user pushes hard against the wall, friction along the wall surface will feel stronger than when the user pushes light against the wall. This makes for a very realistic wall sensation, especially when used in combination with Clipping (as described below).

Surface_Texture — Surface Texture is a parameter that governs the texture felt when rubbing along the length of the Wall surface. If Surface_Texture is low, the feel of rubbing along the wall will feel very uniform. If the Surface_Texture is high, the feel of rubbing along the wall will feel bumpy, like dragging over a rough concrete. Surface_Texture and Surface_Friction together can define a wide range of sensations from a smooth glass to a bumpy stone. Quality force feedback devices use "pressure mode", as described above, to enhance the realism of the rubbing texture sensation. Pressure mode automatically modulates the intensity of the texture sensation based on how hard the user is pushing against the wall surface.

Clipping — Clipping is a binary parameter that can be *on* or *off*. When clipping is ON, the peripheral device creates a unique visual illusion that greatly enhances the impression of wall hardness. Before describing the illusion, lets motivate the need: As described previously, a simulated wall can never be fully rigid, for it will always have some compliance due to the fact that the motors are limited in force output capability. This means that a user using a force feedback joystick and pushing against a simulated wall, *will* penetrate the wall by some distance. If there is no visual representation of this penetration, it is not noticeable to the user. But, if a graphical object drawn by the host computer is following joystick location and is visually shown to penetrate a graphical wall, the penetration becomes very noticeable. Therefore, it is useful to have a means of creating a visual illusion where the graphical object hits a wall and *stops* even though the joystick actually *penetrates* the simulated wall.

APPLE INC.
EXHIBIT 1017 - PAGE 75

To achieve this illusion we need to break the mapping between the graphical display and joystick position. This is called *Clipping* and it works as follows: When a peripheral device penetrates a simulated wall, the peripheral will normally report position data that reflects the penetration. But, if Clipping is turned ON, the data reported from the device will *not* reflect the penetration into the wall, it will reflect the location of the joystick as if the wall was impenetrable. In other words, Clipping causes the peripheral to send false data to the host – data that makes the wall seem rigid even though it is not. This is very useful for graphical simulations where walls are used as real boundaries that objects can not move beyond. The best way to understand this, is to try it. Note, at the present time most force feedback hardware that support Clipping only support this feature for horizontal and vertical walls – not walls at arbitrary angles.

APPLE INC.
EXHIBIT 1017 - PAGE 76

3.8 Barrier

Barrier is a Condition, much like a Wall, that creates the feel of encountering a hard surface within the range of travel of the force feedback interface device. Unlike a Wall, a Barrier is a simulated obstacle that can be *penetrated*. For example, if you are using a force feedback joystick and you move the joystick into a simulated Barrier, you will feel a hard obstacle when the stick is moved perpendicular to the Barrier (pushed into the Barrier). If you push into the Barrier with enough force, you will penetrate and thereby "pop" to the other side. Because you can be on either side of a Barrier, it does not require the Sense parameter used by the Wall sensation. However, unlike the Wall, the Barrier requires two hardness parameters – a *positive hardness* and a *negative hardness* which define the feel of the barrier depending upon which direction you are crossing it. Also, the Barrier sensation requires a *thickness* parameter that defines how difficult or easy it is to penetrate.

Mathematically, a Barrier is represented as a force that increases sharply with displacement at the location of the Barrier when crossed from a given direction. If the user pushes against the Barrier with enough force to cause the peripheral to penetrate by half the *thickness* (t) of the Barrier, the force profile will flip direction thereby "popping" the user to the other side.

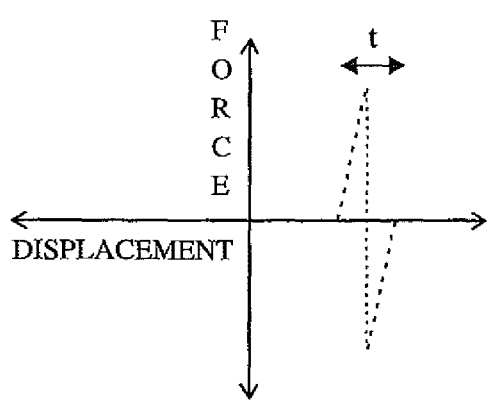


Figure 3-13 Force vs. Displacement for a Barrier Condition

APPLE INC.
EXHIBIT 1017 - PAGE 77

Looking at Figure 3-13, imagine a joystick starting at rest in the center of its workspace and then moving to the right. No force is felt until the Barrier location is encountered (where the dotted line meets the horizontal axis on the figure). The force resists penetration until the joystick penetrates the barrier by half the thickness distance (t). Then, the force flips and the user pops to the other side. If the user now approaches from the right side, the force will again resist penetration until the user "pops" back to the left side of the Barrier.

The angled dotted lines in Figure 3-13 represents the relation between force and displacement when the barrier is penetrated from a given direction. A steep line represents a force that increases quickly with penetration. The steeper the force increases with displacement into the Barrier, the "harder" the Barrier will feel when penetrated from that direction.

Like the Wall sensation, The Barrier sensation has been abstracted to simple parameters that can be rapidly defined by programmers. Below is a summary:

Positive Hardness	Positive Hardness describes how rigid or compliant the surface of the Barrier feels when pressed against from the positive direction.
Negative Hardness	Negative Hardness describes how rigid or compliant the surface of the Barrier feels when pressed against from the positive direction.
Thickness	Thickness lets the programmer control how easy or difficult it is to penetrate a given Barrier. In essence, Thickness defines how deep into the barrier the user needs to push before "popping through" to the other side. As described above, the user needs to push into the barrier by *half* the Thickness in order to pop to the other side. Because positive and negative hardness define how much force is required to penetrate to a given depth, Thickness and hardness together define the feel of the penetration.
Offset	Offset is defined in one of two ways, either it is the distance away from the origin of the peripheral device's

APPLE INC.
EXHIBIT 1017 - PAGE 78

range of motion (i.e. the center of the joystick space), or Offset is defined as a distance from the current location of the peripheral (i.e. the location of the joystick at the instant that the command is called). In the first instance, Offset is an absolute value and in the second instance the Offset is a relative value. An *Offset-Flag* is used to indicate which way you want to define your wall location.

Surface_Friction

Surface Friction is a parameter that governs surface damping when rubbing along the length of the Barrier surface. If the Surface_Friction is low, the Barrier will feel like a smooth surface, like glass or ice. If the Surface_Friction is high, the Barrier will not feel as smooth, as through it was coated in rubber.

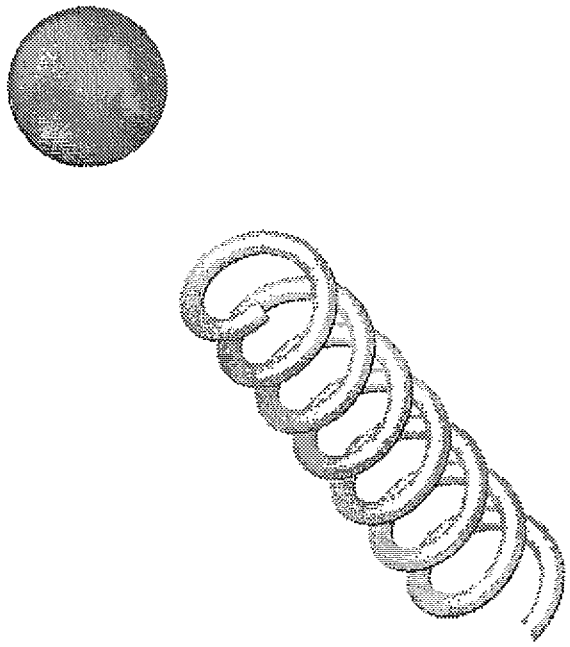
Surface_Texture

Surface Texture is a parameter that governs the texture felt when rubbing along the length of the Barrier surface. If Surface_Texture is low, the feel of rubbing along the Barrier will feel very uniform. If the Surface_Texture is high, the feel of rubbing along the Barrier will feel bumpy, like dragging over a rough concrete.

APPLE INC.
EXHIBIT 1017 - PAGE 79

4. Understanding Temporal Waves

You are the pilot of a Celestial Star Cruiser. The enemy is closing in from behind. You flick a switch on your force feedback joystick and feel the comforting ***rumble*** of your hyper-thrusters as they warm up. Seconds tick away, the enemy gains ground, and the rumble in your stick steadily rises to a mild, high frequency ***hum***. Finally, the feel is just right – your thrusters are tuned and ready for activation. You pull the trigger and ***twack***, your ship lurches forward. You can feel the pressure on your hand steadily rising as your ship gains speed. But then it happens – a spastic ***jerk*** to the left, a sickly ***jolt*** to right, then an agonizing ***pop*** as your engine fails. All is still and quiet. The next thing you feel is the familiar shock wave caused by an exploding enemy torpedo off your starboard bow. They are closing in.



4.1 Overview

Whereas Conditions are described as spatial feel sensations that are based on the motion of the interface device, Waves are best conceptualized as temporal feel sensations that are predefined functions of time. In other words, Waves are force sensations that are defined and then "played back" over time when called. Temporal Waves fall into two basic categories: a) Force Signals and b) Force Profiles

> **Force Signals** are Wave effects that are defined based on a mathematical relationship between force and time using standard wave-form conventions. For example, a Force Signal might be defined as a force that varies with time based on a *sine-wave* of a given *frequency, magnitude,* and *duration*. As will be described throughout this Chapter, Force Signals can have either a constant or a periodic source. The resulting sensation can be as simple as a constant force or as complex as a modulated sine wave, square wave, triangle wave, or saw-tooth wave. Force Signals are useful for generating canned feel sensations such as vibrations and jolts.
>
> **Force Profiles** are Wave effects that are defined based on a stream of digitized data. This is basically just a list of force samples that are stored and played back over time. Because Force Profiles are so free-form, they are often called Custom Effects. Force Profiles are useful in generating canned feel sensations that can not be easily represented as a constant or periodic Force Signal.

The primary advantage of Force Signals over Force Profiles is that a complex sensation can be defined based on simple parameters such as *Sine-Wave, 50 Hz, 50% Magnitude*. As a result, Force Signals are very efficient to represent and work with. On the other hand, Force Profiles allow for more general force sensations than Force Signals. Unfortunately, complex Force Profiles require that a significant amount of data to be stored and transferred to the peripheral device. This has implications on processing burden, communication bandwidth, and local memory limitations of the force feedback hardware. As a rule of thumb when creating canned Wave effects, you should use constant or periodic Force Signals unless the sensation you need to achieve absolutely requires the you design a custom Force Profile from scratch.

APPLE INC.
EXHIBIT 1017 - PAGE 81

4.2 **Defining Force Signals:** *Constant & Periodic*

Many feel effects can be created by having actuators generate time-varying force signals upon a force feedback device. The form and shape of the resulting wave has great influence upon the "feel" experienced by the user. This is analogous to the audio modality in which the "sound" perceived by a user is directly influenced by the form and shape of a generated audio signal. Diverse force feedback effects can be generated by implementing various forms of force waves. For example, a force signal can be a simple *constant force*, or can be a square-wave, a sine-wave, a saw-tooth, or other common *periodic* wave-form. Wave Parameters such as the *frequency*, *duration*, and *magnitude* are used to modulate the feel of a given sensation.

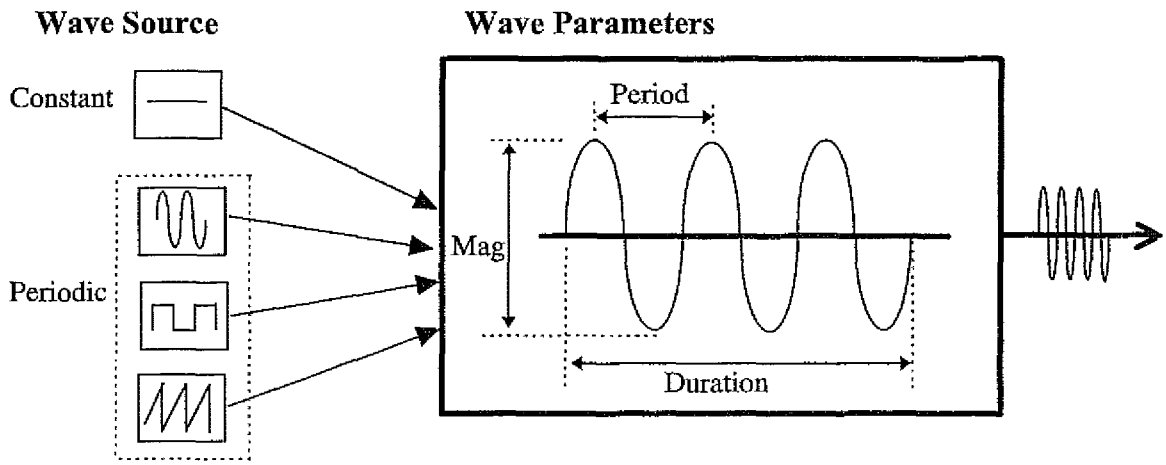
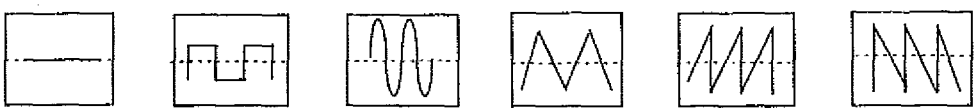


Figure 4-1 Force Signals: Sources and Parameters

Source: At the heart of every force signal is a signal source. This is basically the mathematical foundation for the force versus time relationship. The simplest Source is just a constant force wherein the force remains the same over time. More complex signal sources involve basic periodic wave forms such as the sine-wave, triangle-wave, square-wave and saw-tooth waves (both up and down). The choices for Source supported by most force feedback hardware devices are shown below in Figure 4-2.

APPLE INC.
EXHIBIT 1017 - PAGE 82



Constant Square Sine Triangle Saw-Up Saw-Down

Figure 4-2 Wave Sources

The basic parameters shown in the diagram above define the basic structure of the Force Signal. These parameters include:

Magnitude: Once a signal source is chosen, the Magnitude simply scales the source based on a percentage of full capability of the given hardware device. Magnitude can be 0 to 100%. As will be described in the next section, it is best not to set the Magnitude near 100% because it consumes your dynamic range.

Period: Periodic signal sources such as square, sine, triangle, and saw-tooth require a period to define how the wave should be generated over time. The period is simply the time it takes for a single cycle of the wave form to be played and is defined in microseconds. As is true of all periodic signals, period is defined as 1/frequency of the resulting form.

Duration: The duration parameter simply defines how long the periodic signal should be played. It is defined in microseconds.

Offset: While periodic wave forms usually oscillate about zero force, the offset parameter allows the center of the wave-form to be shifted in the magnitude domain. The result of an offset is a force bias felt throughout the duration of the force signal.

Phase: Periodic wave forms such as Sine and Triangle are such that forces always start at zero and follow a mathematical profile to a peak. Phase is a parameter that allows programmers to have such wave forms start at some value other than zero.

Note, there is one additional Wave Source that is a special case of the Triangle periodic form called the **Ramp**. The Ramp is just half a cycle of a triangle wave form representing either a linearly increasing force or linearly decreasing force with time. While ramp

APPLE INC.
EXHIBIT 1017 - PAGE 83

sensations can be generated using the triangle form by setting parameters appropriately, the Ramp is often treated as it's own signal type. Therefore we will list the Ramp, shown in Figure 4-3, for the sake of completeness:

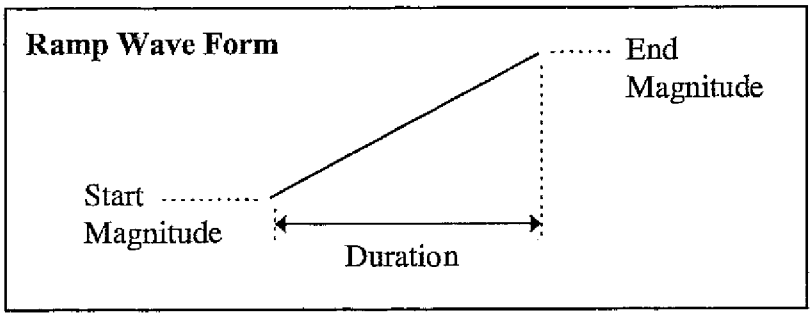


Figure 4-3 Ramp Wave Form

4.2.a "Impulse" Wave Shaping of Force Signals

Once we have defined the basic form, frequency, duration, and magnitude of a feel waveform, the sensation can be greatly influenced by adjusting the shape of the signal. While flexibility in wave shaping is desirable to achieve a wide variety of feel sensations, complete generality is inefficient because humans are only sensitive to certain perceptual features within a "feel" signal. Therefore the most efficient method is to manipulate *only* those variables that have significant impact upon perception of the resulting sensation. This approach helps reduce the complexity of force feedback hardware and minimizes the computational burden of representing diverse feel sensations.

The Audio Analogy:

Like feel signals, sound signals can take many shapes and forms. When artificially synthesizing audio signals, it is often valuable to extract primary perceptual qualities that define the sound of a given signal. Psychophysical research has demonstrated that the sound quality of an audio signal can be represented by fundamental perceptual features known as attack, sustain, and decay that dictate the global shape, or *envelope*, of an audio wave form. Attack, for example, represents the onset of a sound sensation resulting from the pluck of a string or the blow of a pipe.

APPLE INC.
EXHIBIT 1017 - PAGE 84

While the overall concept of the **signal envelope** can be applied to the Force Signals to allow for efficient modulation of the key perceptual qualities of feel sensation, the parameters of the standard audio envelope (attack, sustain, and decay) have little meaning for the physical senses. This is because human perceptual abilities in the physical modalities are substantially different than those in the audio realm. Audio signals, for example, are perceived in the frequency domain while force sensations are perceived in the magnitude domain. This fundamentally changes how an envelope should be defined and manipulated to extract the primary perceptual qualities of the signal. Fortunately, there are physical parameters that are much more meaningful ways to shape a physical signal as compared to the audio attack, sustain, and decay model. These include two primary parameters known as **Impulse Level** and **Settle Time**. These also include two secondary parameters called *Fade Level*, and *Fade Time*. As will be described in the sections that follow, Impulse Level and Settle Time are essential envelope parameters for creating efficient force feedback Waves. These parameters follow what is called the **Impulse Model** of Wave generation. As will also be discussed, Fade Level and Fade Time are secondary parameters that, while not critical to efficient sensation generation, do extend the flexibility and control of Wave sensations.

The Impulse Model:

Like audio signals, Force Signals can be decomposed into primary perceptual features that make the representation simpler and more efficient. The key notion force feedback programmers must understand is that the human physical senses are most sensitive to changes in force intensity (called "transients" or "impulses") and least sensitive to constant forces or slowly changing forces. In fact, a force that abruptly changes from 0% to 40% magnitude, might feel significantly more intense than a force that slowly changes from 0% to 100% magnitude. This is because it is not the magnitude that effects the intensity of a feel sensation, it is the transition -- the change. Therefore, when shaping a Force Signal, the most important parameters to adjust are those that allow the programmer to accentuate the sharp transitions in force level. Psychophysical research in the area of feel perception has revealed that force sensations can be decomposed into two perceptually distinct features referred to here as *Impulse* and *Steady State*. Impulse is defined by two parameters, a *Impulse Level* and a *Settle Time*.

APPLE INC.
EXHIBIT 1017 - PAGE 85

Impulse Level is a variable (defined as a percentage of full) that describes how much initial "kick" or "punch" should be delivered upon initiation of the Wave to accentuate the transient. The effect of Impulse Level can be described as effecting the "crispness" of a physical sensation. An effect with a high Impulse Level will feel abrupt and intense. As is shown in the diagram below, Impulse Level is usually defined as some value significantly higher than the steady state force level of the Wave sensation.

Settle Time is a variable (defined as a duration) that describes how quickly the Force Signal will settle from the Impulse Level to the steady state magnitude. Together these two parameters define an effective envelope for force signals that represents the primary perceptual qualities of feel sensations – the transient. Typically the Settle Time is very short as compared to the Duration of the entire Wave. While the Duration of an Wave can be on the order of hundreds of milliseconds or even seconds, Settle Time is usually on the order of tens of milliseconds.

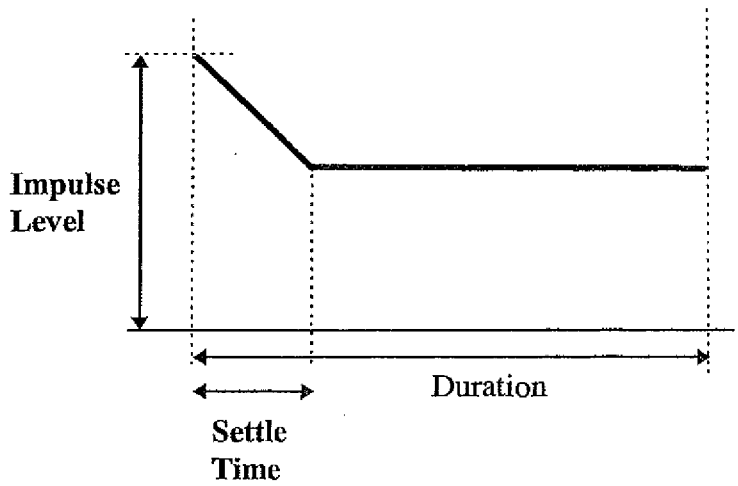


Figure 4-4 Impulse Parameters

Example: Let's say you want to create the feel of a car hitting the side railing of a race track while making a wide turn. Perceptually this is represented as a sharp *impulse* upon impact wherein the impulse quickly *settles* to a lower steady state value that maintains the sensation of continued physical contact with the railing. The Impulse part of the signal is

APPLE INC.
EXHIBIT 1017 - PAGE 86

what conveys the intensity of the collision sensation. The steady state part of the signal need only be a mild, non-intrusive resistive force that simply maintains the physical illusion that the railing is a concrete entity that resists penetration. The steady state force level need not end until the driver recovers the turn and pulls the car away from the wall.

Making the Most of Limited Dynamic Range: The great advantage of the Impulse Wave Shape used in this example is that the high force is only applied for very short period. Most of the Duration of the Wave sensation is filled with much smaller forces. In other words, the Impulse Wave Shape encourages the efficient use of force resources -- applying the high forces when needed, during the sharp transients, and using low forces when appropriate, during steady state periods. This is critical to efficient force feedback programming because it helps manage the limited force resources of the hardware. The last thing a programmer wants to do is saturate the capabilities of the hardware by keeping steady state forces near the maximum capability of the peripheral device because if you do that, you have no available *range* left to provide the perceptually critical sharp transients. In other words, Impulse Wave Shaping lets you make the most of the limited dynamic range of force feedback hardware devices.

Consider the following two principles. They represent important concepts that every programmer should keep in mind when designing force feedback sensations:

- Humans are more sensitive to *force transients* than steady state forces.

  In other words, to make a sensation such as a blast or crash feel intense, the objective should be to impart a strong impulse upon a user (an abrupt *change* in force), not a strong steady force. An impulse is the differential between the background force level and a sudden transient.

- Force Feedback devices are limited in force output (limited in *Dynamic Range*).

  In other words, the maximum force level a device can impart in a given direction is generally on the order of one pound. Therefore it is easy to impart a sensation that uses up most of the available force, leaving no room to impart the strong transients that are so important to compelling sensations. Safety limitations and power

APPLE INC.
EXHIBIT 1017 - PAGE 87

constraints prevent manufacturers from building hardware with greater dynamic range, so programmers must be able to efficiently use the available range of forces.

Impulse Wave Forming helps reconcile the above two points. By designing effects as strong impacts with subdued steady state values, programmers will maintain a low background force level from which strong transients can be applied. This institutes an efficient management of hardware resources, allows multiple effects to be superimposed without reaching dynamic range limitations of the hardware.

Example:

Problem: In a flight simulator game, the airplane suddenly gets hit by a strong cross-wind. As the programmer, you want the sensation of the wind to be intense so you apply a strong constant force on the joystick that pushes to the left. Suddenly, an enemy fighter emerges from the clouds and starts pummeling the craft with gunfire from the right. As programmer, you want the sensation of the gun blasts to be an intense stream of Jolt sensations, also to the left. Unfortunately, the joystick motors are near their force limit due to the cross-wind sensation so the gun blasts feel like weak taps that are not even noticed by the user.

Solution: Using the Impulse Wave Forming method, the sudden cross wind is represented by a sharp Impulse of force to the left that quickly settles to a moderate side force. Because the user is most sensitive to the transient and not the magnitude of the constant force, the resulting sensation is still convincing as an intense wind blast. Now the enemy fighter emerges and there is sufficient dynamic range left to provide a convincing stream of jolts above the background force level. Clearly, use of Impulse Wave Shaping helps you make the most of your force resources.

The General Wave Effect Envelope

While the parameters of **Impulse Level** and **Settle Time** are enough to define the primary envelope features for feel based sensations, there are two secondary parameters that extend the flexibility of the Impulse Wave Shaping paradigm. **Fade Level** and **Fade**

APPLE INC.
EXHIBIT 1017 - PAGE 88

Time are analogous to Impulse Level and Settle Time but are applied at the trailing end rather than at the onset of a Force Signal. Fade Level defines the final force level to be felt and Fade Time defines how long it takes for the steady state value to decay to that level. Because the human sensory system is not sensitive to slowly changing forces, using fade parameters to create force sensations that slowly decay is usually not effective.

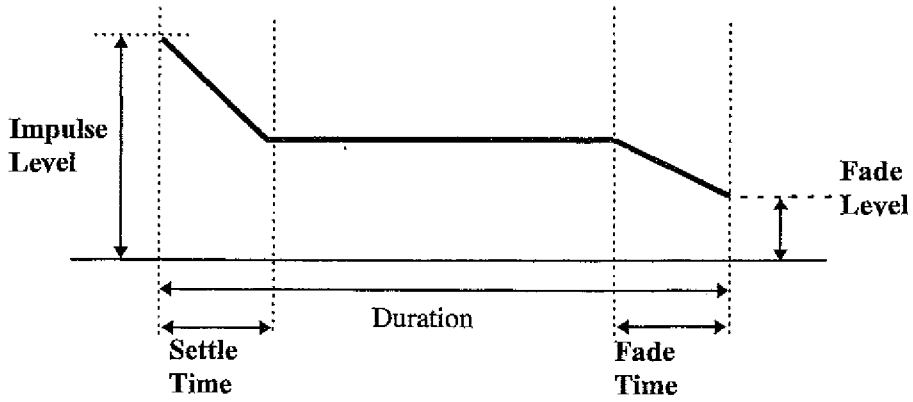


Figure 4-5 Force Signal Parameters

By using the four parameters defined above as Impulse Level, Settle Time, Fade Level and Fade Time, programmers can generate a wide variety of envelope shapes. Figure 4-6 shows some of the shapes that can be achieved with this basic parameter set.

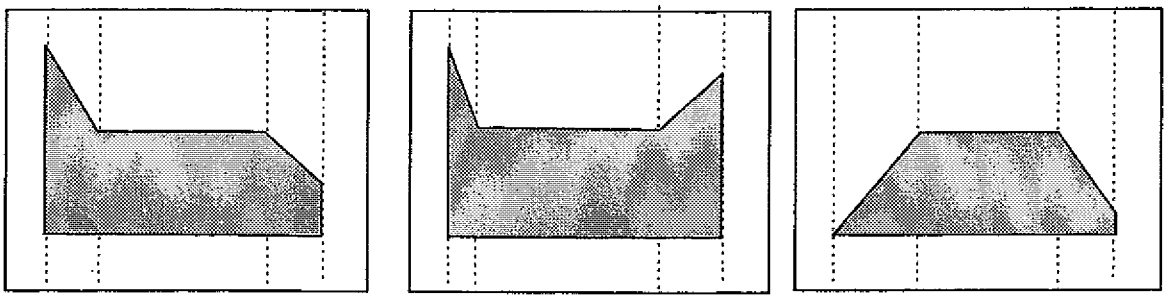


Figure 4-6 Example s of Force Signal Wave Shapes

APPLE INC.
EXHIBIT 1017 - PAGE 89

4.2.b Summary of *Periodic* Force Signal Generation

All Force Signal type Waves are based upon a force **Source**. The source is the basic force signal from which the Wave is created. The Source may be as simple as a constant force value or may be as complex as a sine wave or square wave. The typical Source values supported by hardware are:

- Constant Force
- Square Wave
- Sine Wave
- Triangle Wave
- Saw-Tooth (Up or Down)

All Force Signal type Waves have **Control Parameters** that regulate the feel of the force Source. Control parameters include some or all of the following values:

- Magnitude
- Frequency
- Duration
- Offset
- Phase

All Force Signal type Waves may also have **Impulse Parameters** that shape the quality of the feel signal described by the parameters above. Impulse Parameters used to define the feel envelope are:

- Impulse Level
- Settle Time
- Fade Level
- Fade Time

APPLE INC.
EXHIBIT 1017 - PAGE 90

All Waves also have **Application Parameters** that describe how the resulting signal is applied to the given device. Application Parameters may be a Direction in vector space or may be an Axis Mask that defines one or more device axes for signal application.

- Axes
- Direction

All Waves may also have **Trigger Parameters** that indicate when to execute the defined effect. Many effects may simply trigger (execute) upon being called by the host. Other effects may trigger automatically when a given button is pressed on the device. This automatic execution of an Wave is called a *reflex* and is helpful in reducing the communication burden for effects that occur very frequently such as "gun recoil" sensations. The parameter Trigger Button defines what button should trigger a given effect. The parameter Trigger Repeat Interval defines the time the delay between multiple executions of a given effect when a button is held down for an extended period.

- Trigger Button
- Trigger Repeat Interval

APPLE INC.
EXHIBIT 1017 - PAGE 91

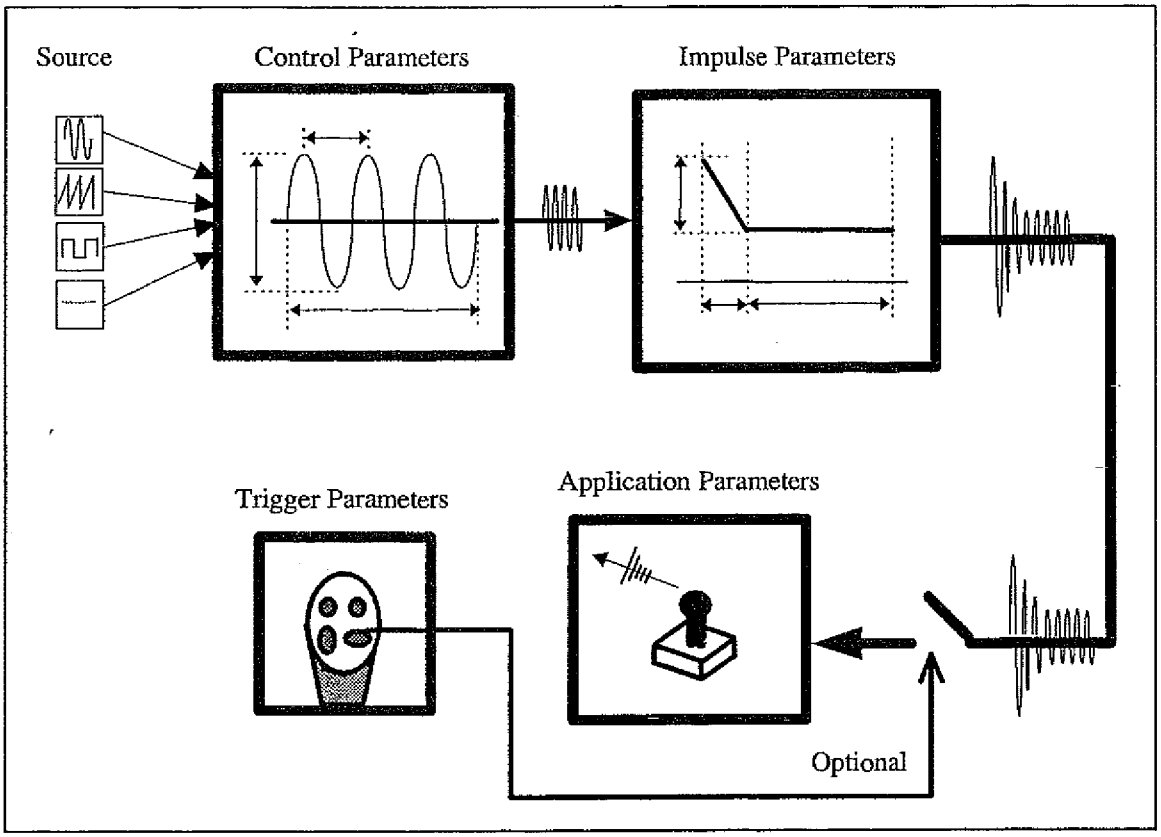


Figure 4-7 Diagram of Wave Generation Process

The above figure shows the entire wave generation process at the conceptual level. A signal source is chosen based on the desired feel, then control parameters are then defined to tune the feel of the basic signal, Impulse parameters are then applied to fit the signal to a desired envelope, and finally application parameters are assigned which indicate the axis or orientation that the signal will be applied to. If the signal is to trigger from a button press, Trigger Parameters are also assigned. While this process may seem complex, it has been reduced to a simple programming paradigm within DirectX that is introduced in Chapter 6.

4.2.c Three types of Periodic Waves

Thus far we have presented a process of picking the *type* of wave source, defining basic effect parameters such as Magnitude, Direction, and Duration, and then applying an envelope to the resulting form. While this process allows for a wide variety of feel

APPLE INC.
EXHIBIT 1017 - PAGE 92

sensations, high quality force feedback devices employ some advanced notions that allow for even more compelling Wave sensations. The three Basic and Advanced Periodic notions that are important to introduce here are **Axis Wave**, **Angle Wave**, and **Angle Sweep Wave.**.

Axis Wave: In many situations, it is desirable to have the Wave sensation associated with a single degree of freedom of the force feedback device such as the X axis of a joystick, the rotation axis of a steering wheel, or the Y axis of a flight yoke. For example, when an airplane stalls in a flight simulation, you may want to have a subtle vibration on the pitch axis (Y axis) of the Flight Stick. For such cases where the Wave is oriented along a primary axis of the hardware device, the signal can be thought of as an "axis wave".

Angle Wave: For cases where the Wave sensation is associated with multiple degrees of freedom of the force feedback device, such as both the X and Y axes of a joystick, the Wave sensation must be assigned an *angle* to dictate how it is oriented within the multiple degree of freedom space. For example, in a flight simulation your plane might be bombarded by machine-gun fire from an enemy fighter that is attacking from a 30 degree angle off your right wing. In this case you might want to have a square wave represent the gun fire pounding your craft such that the force is oriented at the appropriate 30 degree angle. In such cases, the *angle* parameter associated with the Wave sensation is very useful.

Angle Sweep Wave: The basic periodic wave described above (with a single *angle* parameter) can create a vibration or other time varying signal at a particular orientation. While you can repeatedly call such an angle wave, updating the angle parameter, to very the orientation over time, sometime it is desirable to have that angle change smoothly in real time. For this reason, the makers of force feedback hardware have invented the Angle Sweep concept where a periodic wave is assigned a Start Angle and an End Angle such that the direction of the force will sweep

APPLE INC.
EXHIBIT 1017 - PAGE 93

between these angles over the duration of the sensation. Because this sweep is controlled by the local processing hardware on board the force feedback device, the programmer need not continually update the angle parameter. This is useful in creating very complex Wave sensations.

One way to create all three types of Wave sensations described above (Axis Waves, Angle Waves, and Angle Sweep Waves) is to use the DirectX structs described in Chapter 6. This is a well defined process where simple parameters are assigned to describe the desired wave form. Alternatively, you can use the graphical **I-FORCE Studio** toolset described in Chapter 9. Using these tools you can construct your wave form in a *graphical wave editor* and when the sensation is complete, the Toolset will generate the appropriate DirectX code automatically.

APPLE INC.
EXHIBIT 1017 - PAGE 94

4.3 Defining Force Profiles

As described in the previous sections, Temporal Waves are a class of feel sensations defined as forces that vary as pre-defined functions of time. Many interesting Wave sensations can be created by using periodic signals to define the relationship between force and time. And while periodic signals provide an efficient means of representing Wave sensations, there are situations where these simple combinations of sine waves, square waves, or other periodic sources will *not* be capable of defining the canned force-time profile you desire. For such cases, manufacturers of force feedback hardware devices have provided a versatile means for programmers to define custom force profiles of any shape.

4.3.a Custom Force Profiles

Custom Force Profiles (or *Custom Forces*, as they are called under DirectX), are simply a sequence of force values stored in array that define through discrete samples how force should vary over time. A custom force profile is best thought of as a digitized signal composed of (n) **Samples** wherein each sample is played for a defined **Sample Period.** The samples can be stored to represent the forces to be played over a single hardware axis, or the samples can be stored as forces to be played over multiple hardware axes – the number of axes represented in an array of sampled data is represented by a variable that indicates how many **Channels** are included in the structure. How a custom force profile of 10 samples might look follows in Figure 4-8.

APPLE INC.
EXHIBIT 1017 - PAGE 95

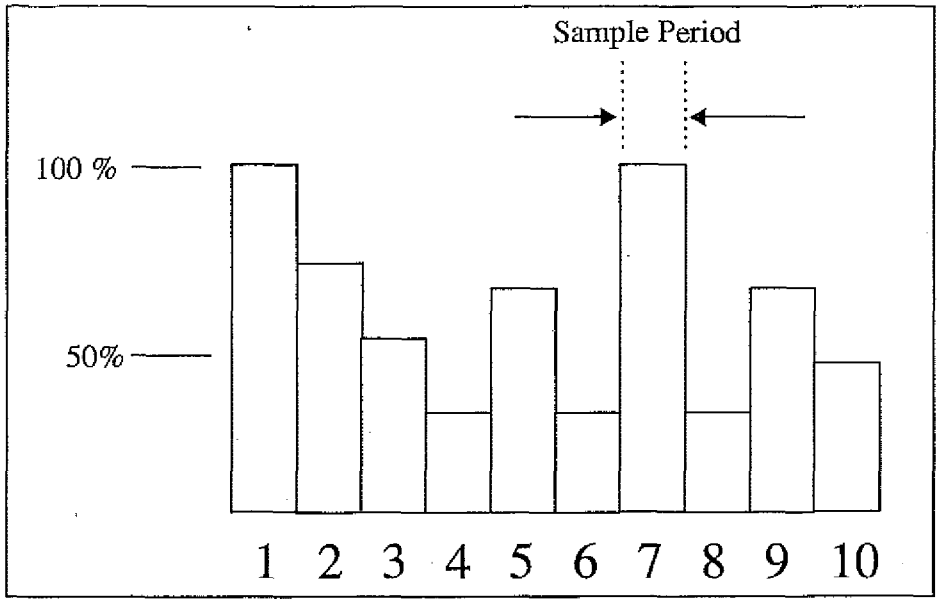


Figure 4-8 Custom Force Profile Example

As represented above, a force profile composed of an array of force values, each value in the representing the force to be played for a sample period. While the forces shown above are all positive, it should be noted that the values stored in a profile array can be positive or negative. Also, it should also be noted that a profile array can contain multiple channels, meaning multiple sets of data each of which is assigned to a different axes of the force feedback device. This allows a force profile to be multidimensional is desired.

Ideally, the force feedback device that generates the force profile sensation has an intelligent local microprocessor architecture as described in Section 2.2 of this text. Such a local processor greatly facilitates the generation of custom profiles because the force array can be downloaded to the peripheral device and played upon demand. For example, the patented I-FORCE architecture enables force profiles to downloaded from the host and stored in memory local to the peripheral device. These force profiles can then be triggered by host commands or be triggered by a button reflexes (reflexes are described in detail in Section 1.2.c). The button reflex is an ideal implementation for custom profiles that define the unique feel of a particular weapon-fire event. For example, a compelling custom profile that defines the feel of your gyro-blaster weapon can be downloaded to the local microprocessor along with a trigger assignment that establishes a button reflex. Then, every time the given button is pressed, the particular force profile is executed. This

APPLE INC.
EXHIBIT 1017 - PAGE 96

allows for clean coordination between gaming events and feel sensations with minimal programming overhead.

Designing the FEEL of a Custom Profile

Because it is not obvious what a given force profile will feel like when generated through force feedback hardware, it is difficult to design a custom force profile without engaging in an iterative process of defining values and feeling sensations. Ideally, force profiles are created using an interactive force editing tool that speeds the iterative design process. Such a tool can allow force sensations to be constructed rapidly using a graphical editor and played repeatedly until the feel is just right. Among other things, the **I-FORCE Studio** toolset described in Chapter 9 of this text includes such an interactive editing tool. The *Profile Editor* component of the I-FORCE Studio toolset greatly facilitates the use of the custom force feature and is highly recommended.

APPLE INC.
EXHIBIT 1017 - PAGE 97

5. Understanding Dynamic Sensations

Suddenly your Alterian Cargo Vessel has drifted into an uncharted asteroid belt. At first you feel the subtle ***rattle*** of small stones bouncing of your hull. Then you see a massive boulder screaming in from above. You engage your force field and brace yourself for impact. ***Whack*** – the large stone compresses your energy field like a rubber-band stretched to its limit, but thankfully the boulder is repelled – you can feel the compression and release as the giant rock flies clear. Then another stone hits with an even deeper compression and more violent release. ***Thud***. Then another. It is becoming increasingly difficult to steer while repelling stones from every direction. Three more collisions hit in rapid succession. If you don't learn to cushion the blows by pitching your ship away from the impact, your field will be penetrated by the next large boulder.

5.1 Overview

The industry of consumer force feedback was born in 1995 with the launch of the first force feedback programming interface, **I-FORCE** from Immersion Corporation. Version 1.1 of the I-FORCE 1.1 supported the basic notions of Conditions and Waves but did not enable Dynamic Sensations, or simply "Dynamics". While Conditions and Waves do provide programmers with a compelling ability to embellish gaming environments with feel sensations, **Dynamics** is a fundamental leap forward in force feedback technology that enables a level of realism and interaction never before provided.

While Dynamics is a relatively new innovation, most force feedback devices that will launch in 1997 will support embedded Dynamics capabilities. Dynamics is made possible by recent advances in the hardware processors employed within force feedback devices. For example, all hardware products that support the latest generation of I-FORCE engine, currently referenced as **I-FORCE 2.0**, will include what is called a *Distributed Dynamic Processing Engine*. This local software engine allows complex Dynamic Sensations to be executed at high speeds in parallel with host executing of gaming events. The DDP engine unleashes an new level of power and sophistication in force feedback programming – consider the following:

- The feel of a ball impacting a racquet, compressing the strings, and then bouncing off with a final snap.
- The feel of wading through a thick pool of swamp-muck that jiggles as you struggle, making it difficult to move straight.
- The feel of swinging a boulder on the end of a cable – the faster you spin, the harder it tugs until you let go, letting it fly.
- The feel of an asteroid as it dives into the force field surrounding your spaceship and is deflected by your quick maneuvering skills.

All of the above sensations have one important thing in common – how the user reacts <u>during</u> the sensation event has a significant effect on the feel of the event. This is because the above example all involve real-time physical interactions based on <u>user motion</u> *and* a <u>physical system</u> wherein user motion *during* the interaction effects the behavior of the

APPLE INC.
EXHIBIT 1017 - PAGE 99

physical system. For example, if the user wades through swamp-muck in a violent manner that stirs up undulations in the fluid, the user's rash motions will increase the difficulty of travel because the undulations in the fluid will worsen as the user struggles. But, if the user wades through the swamp-muck in a dexterous manner that absorbs the undulations in the fluid, the user will have an easier time passing through the muck. This example, like all interactions with physical systems, demonstrates that how the user influences the system during the event will affect how the event feels. Whether is be a ball bouncing off a paddle or a bolder swinging on a string, when feel is influenced by user actions, the realism of the gaming environment is propelled to a new level.

The importance of real-time dynamic interactions can not be underestimated. After feeling a demonstration of a simple ball-paddle dynamic interaction developed by Immersion Corporation for I-FORCE 2.0, **Next Generation Magazine** reported:

> *The technology makes such a difference that even the test demo is mesmerizing. . . All you do is bounce the ball on the rubber band and the feel of the bounce is communicated through the joystick. The experience is so unique that you could spend hours doing nothing but bouncing the ball it sounds stupid, but its true.*
>
> **Next Generation Magazine,** March 1997

While **Conditions** and **Waves** are force feedback entities that are incapable of supporting real-time physical interactions based on user motion *and* a physical system, **Dynamics** is a new class of force feedback entity developed specifically to meet this need. Dynamics are force feedback sensations generated based on real-time dynamic simulations of genuine physical systems. For example, to generate the feel sensation of a mass impacting a compliant object (i.e. an asteroid impacting the force field of a space ship), a real-time dynamic simulation of the physical system must be used that takes into account how user moves during the interaction. The result – a feel sensation that is so real, it makes physical dexterity part of game play.

Dynamic Sensations lets users take advantage of their inherent sensory-motor skill and physical dexterity when interacting with software. Consider the ball bouncing off a racquet. Beyond being a very compelling force feedback sensation, it is a sensation that

APPLE INC.
EXHIBIT 1017 - PAGE 100

provides critical real time information that allows players of racquet sports to impart subtle control over the ball. The user could cushion the ball to a halt by absorbing energy in the wrist, could whip the ball sharply off the paddle by tightening the wrist with a snap, or could sling the ball off to the side with a flick of the arm. Although the entire event may only last 500 milliseconds, the subtle feel of the continuously changing forces during the interaction is very important to making a realistic sensation. It is an clear example of a "dynamic" sensation that can not be represented by a simple "canned" profile that is predefined and played back over time. This is because how the user interacts with the ball *during* the event greatly changes the feel.

The Distributed Dynamic Processing Engine: As you might imagine, the dynamic simulation of the physical system used to generate the feel of a ball bouncing off a paddle needs to run very quickly in order to provide a sensation that is not corrupted by lag. In other words, the software that runs the dynamic simulation needs to read sensor position, update location of the paddle, compute the dynamic interaction between the paddle and the ball, compute the resultant forces to be felt by the user, and send those forces to the motors of the interface device every few milliseconds. If the host computer was to perform the dynamic simulation, this process would also include the steps of reading sensor data from the interface device across the communication bus and sending force command back to the interface device across the communication bus. This would be a computational burden for the host and a bottleneck for the communication bus. Luckily, the designers of force feedback hardware peripherals have solved this problem by allowing the force feedback device to perform the dynamic simulation *locally* using the DDP engine. As a programmer, all you have to do is set up the dynamic simulation by sending physical parameters to the interface device – everything else is handled for you in parallel with gaming execution.

APPLE INC.
EXHIBIT 1017 - PAGE 101

5.2 The Basic Dynamic Sensations

At the present time, there are a handful of primary dynamic sensations supported by force feedback hardware. Each dynamic sensation sets up a compelling physical sensation within the local processing engine of the force feedback device. Simple parameters defined by the programmer can tune the dynamic sensation to mesh with gaming events. The basic types of Dynamic Sensation are listed below and described in detail in the following sections:

DYNAMIC SENSATION	SENSATION OVERVIEW
Dynamic Recoil	The interactive feel of firing a weapon that kicks back and reverberates
Dynamic Impact	The interactive feel of a collision with a physical body
Dynamic Liquid	The interactive feel of moving through a jiggling / oscillating liquid
Dynamic Inertia	The interactive feel of dragging a mass
Dynamic Center Drift	The interactive feel of spring centering that adapts to game play
Dynamic Control Law	Lets you program the dynamic engine directly.

The DDP engine performs dynamic simulations of physical events such as a reverberating recoils, jarring collisions, and undulating liquids based on simple physical parameters defined by the programmer. These simulations are performed locally within I-FORCE enabled hardware products using embedded I-FORCE microelectronics. This distributed processing architecture greatly reduces the computational burden on the host and ensures that the complex dynamic feel sensations can be generated with minimal impact upon game speed.

Dynamics with Interim Reporting:

For discrete dynamic events such as *impacts* and *recoils*, or for dynamic environments such as dynamic *liquid* and dynamic *inertia*, the above distributed architecture works fine.

APPLE INC.
EXHIBIT 1017 - PAGE 102

The host just sends parameters to the DDP engine and the hardware executes the complex sensation in parallel to what ever is happening in the host application. But, say you wanted to do a more complex feel simulation that was coordinated in real time with dynamic graphical events shown on the screen. For example, say you wanted to simulate the feel of a ball interacting with a compliant paddle, the subtle dynamics of the interaction being coordinated with a screen image of the ball compressing and bouncing off the paddle. Or say you wanted to simulate the feel of a ball being swung around on the end of a string while simultaneously displaying a graphical image of the ball that coordinated with the dynamic feel. Clearly, to create such compelling coordination *during* a dynamic simulation performed by the DDP engine, the DDP engine needs a means of updating the host *during* the feel event. We call this updating feature **interim reporting**.

Interim Reporting is a feature of the I-FORCE 2.1 Processing Core that allows the DDP engine to report to the host the location of the simulated mass used by the dynamic feel sensation at discrete time intervals during the sensation. While the DDP engine might be performing the dynamic simulation with an internal update rate as high as 1000 Hz required for good feel, the host does not need such rapid update for graphical display. Instead, the DDP engine reports data back to the host at 60hz, sufficient for visualization of the event.

Interim Reporting enables the following two additional Dynamic Sensations:

DYNAMIC SENSATION	SENSATION OVERVIEW
Dynamic Sling	The interactive feel of a mass being swung on the end of a cable.
Dynamic Paddle	The interactive feel of a ball/puck bouncing off a paddle.

In addition to reporting interim mass location values during a dynamic simulation, it is also valuable for the DDP engine to perform **Final Value Reporting**, reporting the final position and final velocity of the simulated mass at the end of the interaction. The purpose for this is easy to see in the Ball-on-Paddle interaction example described below.

APPLE INC.
EXHIBIT 1017 - PAGE 103

Example:

Ball-on-Paddle The DDP engine of the I-Force 2.1 Processing Core makes it easy to achieve this complex interactive event described above where the graphical display of a ball compressing and bouncing off a paddle is coordinated with the dynamic feel simulation of a ball-paddle interaction. When the simulated ball first impacts the simulated paddle, the host configures the dynamic event by telling the DDP engine the physics of the event. Parameters such as the mass of the ball, the incoming velocity of the ball, the compliance of the paddle, and the inherent damping of the paddle are conveyed. The DDP engine then simulates the event, taking control over the simulation. In real time, the ball compresses the paddle while the user simultaneously moves the paddle in response. Based on the momentum of the paddle, the compliance and damping of the paddle, and the motions made by the user – the DDP engine computes the simulated location of the ball in real time. Based on that simulated location and the stretch of the paddle, the DDP engine creates the appropriate and realistic feel sensation. At regular time intervals during this complex interaction, the DDP engine reports ball location and paddle location back to the host. These interim reports are used by the host to update the graphics and create a visual display that corresponds with the complex feel. When the ball leaves the paddle, the event is over and the DDP engine returns control of the simulation to the host by reporting the final velocity of the ball as it leaves the paddle.

The result is a well coordinated graphical and physical simulation where the high speed computations required for feel simulation are done local to the joystick and therefore do not slow down the host application. At the present time, I-FORCE 2.1 products running under DirectX 5 support two sensations that are enhanced by interim reporting: **Dynamic_Sling** and **Dynamic_Paddle.**

APPLE INC.
EXHIBIT 1017 - PAGE 104

5.3 Dynamic Recoil - Ultra-Realistic Weapon Simulation

Recoil is a very common sensation used in gaming applications wherein a weapon is fired and the user is given a feel sensation of the "kick-back". The simplest recoil sensation is just a canned jolt, a force of a given magnitude played over time. A slightly more complex recoil sensation might be force profile defined as scripted force variation over time. A common recoil may also include a canned vibration profile to simulate the feel of weapon resonating after firing. In all such cases, the canned effects may be complex to create but they still lack realism of feel because the jolts, profiles, or vibrations are merely pre-defined scripts played over time, regardless of user interaction. In other words, such canned routines do not vary dynamically based on how the user resists motion during the interaction. For example, if the user tenses his grip during a kick, the canned recoil will not feel any different than if the user cushions the blow with a loose grip. In the real world, a tense grip would result in a more abrupt recoil sensation with a high frequency resonance. A loose grip would cause a less jarring recoil with a slow resonance that quickly fades away. Simply put, how the user reacts to the blast, through grip and hand motion, should greatly influence the recoil response of the simulated weapon. Therefore what is needed is an entirely new method for defining recoil sensations that is not just a scripted profile but actually accounts for the fact that in the real world, the feel of the blast depends greatly upon how the user responds manually in real-time. What is needed is Dynamic Recoil.

Dynamic Recoil is a revolutionary method of defining the feel of a weapon fire that takes advantage of the *distributed dynamic processing engine* on-board all I-FORCE 2.0 products. Rather than scripting a canned force profile (which is time consuming and gives less than compelling results), the programmer simply defines the dynamic properties of the weapon and lets the joystick's DDP engine generate the detailed force profile in real-time based on how the user reacts. The result is a very complex and rich sensation that is easy to define and that changes appropriately based on user actions. This makes the weapon fire an interactive physical event rather than a scripted whack. Good players will learn how to cushion the blow, thereby limiting the effect that the recoil may have on play. The following parameters are used in Dynamic Recoil:

APPLE INC.
EXHIBIT 1017 - PAGE 105

Blast Direction	Blast Direction simply defines the direction that the weapon is being fired.
Blast Intensity	Blast Intensity defines the overall strength of the blast.
Dynamic Mass	Dynamic Mass defines the simulated physical mass of the weapon used by the DDP engine. A weapon defined with large dynamic mass will recoil as if it where heavy, having a strong kick that is difficult to counter. A weapon defined with a small mass will recoil as if it were light and easy to wield, being very controllable during kick-back.
Blast Resonance	Blast resonance defines how the simulated mass will resonate after the shock of the initial blast. A high resonance parameter will result in a violent shaking after the initial blast. A low resonance parameter will result in a light tremble. Remember these are physical simulation parameters, not canned routines over time, so how the user grabs the stick, reacts to the blast, and cushions the blow will greatly influence the result.
Blast Decay	Blast Decay defines how quickly the simulated mass will decay back to rest after the initial blast. A high decay parameter will result in one or two oscillations. A low decay parameter may result in a lengthy reverberation. Again, this is a physical simulation parameter, not a canned routine over time, so how the user grabs the stick, reacts to the blast, and cushions the blow, will greatly influence the resulting feel.

APPLE INC.
EXHIBIT 1017 - PAGE 106

5.4 Dynamic Impact - Ultra-Realistic Collision Simulation

Impact is a common sensation used in gaming applications to simulate the feel of being hit by an incoming object or slamming into an external obstruction. The incoming object might be an asteroid careening into your space ship, a laser blast impinging on your robot-walker, or an opponent's Indy-car bumping you around a tight turn. An external obstruction might be the railing of a race track, the walls of a dungeon, or the back bumper of the opponents car.

The simplest Impact sensation is just a jolt imposed in a given direction for a given duration. A more sophisticated Impact might be a force profile scripted over a predefined time period. These canned effects will convey that a collision event has occurred, but the scripted nature of the event will cause the sensation to feel somewhat artificial. Because such canned routines do not vary based on how the user reacts *during* the collision, the scripted whack lacks realism. For example, if a user of a first-person action game runs head first into a wall, a canned Impact could produce simple whack to represent the collision. The canned Impact could even give a complex scripted profile that simulates the feel of colliding with the wall and bouncing off. But, the collision is pre-planned and just played back over time – how the user is holding the stick during the collision event will *not* change the feel. For example, if the user is holding the stick rigidly during the collision, the impact will not feel any different than if the user was holding the stick loosely. Or if the user reacts to the collision by quickly pulling back, it will not feel any different than if the user was caught off guard and hit the wall so unexpectedly he could not react at all. What is needed is a new method for defining impact sensations that is not just a scripted profile but actually accounts for the fact that real collisions depend greatly upon how the interacting objects respond in real-time *during* the event. In other words, what is needed is Dynamic Impact.

Dynamic Impact is a new method of defining and executing force sensations associated with collision events. Dynamic Impact allows for very complex collision sensations with minimal programming burden because it uses the advanced features enabled by the *distributed dynamic processing engine* used by I-FORCE 2.0 products. Rather than merely scripting force profile, the programmer simply defines the physical properties of the collision and lets the joystick's DDP engine generate the forces in real-time based on

APPLE INC.
EXHIBIT 1017 - PAGE 107

how the user reacts during the collision. The result is a complex sensation that is easy to define and that changes appropriately based on user actions. This makes the collision an interactive physical event rather than a scripted jolt. By allowing such interaction, force feedback becomes part of game play, not just an overlay. Good players will learn how to minimize the disturbing effect of a collision by reacting appropriately to absorb the energy in their palm. The following parameters are used to define the physics of the impact.

Impact Direction	Impact Direction simply defines the direction from which the impact came
Impact Intensity	Impact Intensity defines the overall magnitude of the impact.
Dynamic Mass	Dynamic Mass defines the simulated physical mass of the object controlled by the user. For example, in a driving game it would represent the mass of your car, in an action game, the mass of your character. A object defined with large dynamic mass will impact forcefully (with substantial inertia behind it) whereas an object defined with a small mass will not impact with much momentum.
Elasticity	Elasticity defines how the object under your control (ship, car, character) will respond to the collision. An elastic object will compress and absorb much of the impact – this will create a much softer feel that an inelastic object which will collide with a crisp crack. Of course how the user grabs the stick, reacts to the collision, and cushions to impact will greatly influence the feel.
Collision Absorption	Collision Absorption defines how quickly the collision disturbance will be absorbed and dissipated after the initial impact. A high absorption parameter will result in the disturbance decaying after one or two oscillations. A low absorption may result in a lengthy reverberation. Again, this is a physical simulation parameter, not a canned routine over time, so how the user grabs the stick, reacts during the collision, and cushions the impact, will greatly influence the resulting feel.

APPLE INC.
EXHIBIT 1017 - PAGE 108

5.5 Dynamic Liquid - *A Liquid Sensation that Jiggles*

Liquid is a sensation often used in gaming environments where a character is submerged in water, mud, or other viscous mediums. Typically liquid is simulated as simple damping – this creates the sensation of "drag" but not the dynamic jiggles and oscillations of water. As a result, static damping falls short as a realistic liquid simulator. What is needed is a dynamic liquid simulation wherein the users motions will cause disturbances in the liquid and excite realistic jiggles and undulations. In other words, what is needed is Dynamic Liquid.

Dynamic Liquid is an astounding technique for defining and executing force sensations representing motion within a liquid environment. Dynamic Liquid allows for very complex undulations and jiggle disturbances with minimal programming burden because it uses the advanced features enabled by the *Distributed Dynamic Processing Engine* used by I-FORCE 2.0 products. Rather than merely defining simple damping and scripting predefined vibrations, the programmer simply defines the physical properties of the liquid medium and lets the joystick's DDP engine generate the forces in real-time based on how the user disturbs the simulated liquid. The result is a complex sensation that is easy to define and is highly dependent upon user interactions. This makes the liquids seem like interactive physical environments rather than a scripted sensations. Good players will learn how to minimize the disturbing undulations caused when moving through a dynamic liquid by controlling their abrupt motions. The following three parameters define the physics of the liquid environment:

APPLE INC.
EXHIBIT 1017 - PAGE 109

Density	Density defines the thickness of the simulated liquid as felt by the user – the greater the density, the larger the disturbance generated when moving through the dynamic liquid. A high density will feel like sloshing around in a thick liquid while a low density might feel like mildly disturbing a thin gas.
Settle	Settle defines how quickly the undulations induced in the liquid will settle down after a disturbance. A liquid with a high Settle parameter will oscillate mildly. A liquid with a low settle parameter will oscillate for a long time after the disturbance. Again, this is a physical simulation parameter, not a canned routine over time, so how the user hold the stick and resists the repeated undulations will influence how quickly or slowly the sensation settles.
Viscosity	Viscosity represents the resistance to motion felt object under your control (ship, car, character) as it moves through the liquid. High viscosity will feel like mud. Low viscosity will feel like water.

APPLE INC.
EXHIBIT 1017 - PAGE 110

5.6 Dynamic Inertia - *Adjust the Weight of Your Interface*

Simulated **weight** is an interesting sensation that has many useful applications within gaming environments, however there are few sensations that adequately emulate the feel of heavy or light joystick or wheel. For example, it is desirable to make a joystick or wheel feel "heavy" when controlling a big vehicle like a tank or a bomber and feel "light" when controlling a small vehicle like a hang-glider or scooter.

Dynamic_Inertia is a new command that enables accurate weight simulation by using the distributed dynamic processing engine of I-FORCE 2.0 products. To use dynamic inertia, you simply define the dynamic parameters Inertia and Play as follows:

Inertia	Inertia defines the felt weight of the joystick, wheel, or other gaming device.
Play	Play defines a gap of free motion between the stick and the mass. It feels like the mass is not attached to the stick very well. This works very well for wheels since in real steering systems, there is play caused by gear backlash.

APPLE INC.
EXHIBIT 1017 - PAGE 111

5.7 DynamicCenterDrift - *Spring Origin Follows User Over Time*

DynamicCenterDrift is an interesting command made possible by the distributed dynamic processing engine of I-FORCE 2.0 products. When using DynamicCenterDrift, the joystick or wheel will be pulled to an origin by simulated springs of some stiffness. This is very similar to static spring return commands used in older versions of I-FORCE. The new enabled feature is that if the user stays in a given location for a period of time, the origin will slowly drift to that new location. The sensation is very interesting and is best understood by simply trying it. To define a DynamicCenterDrift, you simply define the two *dynamic parameters* **Stiffness**, and **Drift_Resistance**.

Stiffness	Stiffness is the strength of the simulated restoring spring.
Drift_Resistance	Drift Resistance defines how quickly or slowly the origin will drift to follow the current location of the interface.

APPLE INC.
EXHIBIT 1017 - PAGE 112

5.8 Dynamic Sling - *Lets you feel a ball-on-a-string*

Dynamic Sling is a unique sensation that simulates the feel of swinging a ball on the end of a string. As with a real ball, the Dynamic Sling sensation is such that the faster you swing the ball, the stronger the force you feel. The force pulls radially along the length of the string, simulating the centripetal force pulling on the swinging mass. The string can be rigid like a steel cable or compliant like rubber bungie chord.

The best way to understand Dynamic Sling is to think of the interface device, such as the joystick, controlling the location of a "bobbin" to which one end of the simulated string is attached. The other end of the string is attached to a mass. Since a joystick is a two dimensional interface, the motion of the mass and the bobbin is restricted to a plane. This is represented graphically in Figure 5-1.

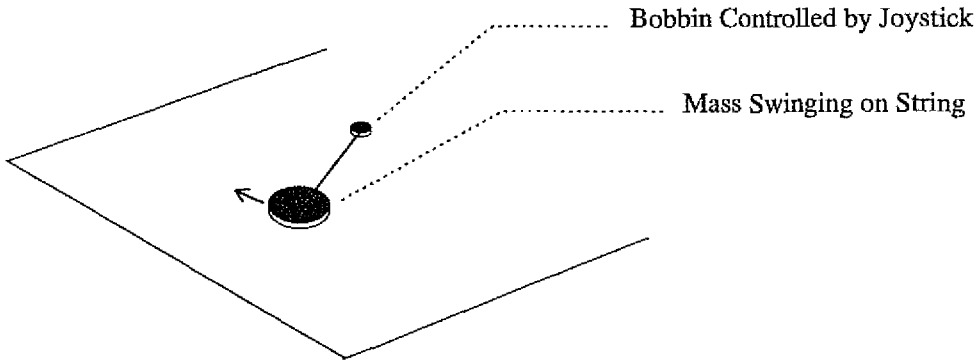


Figure 5-1 Dynamic Sling: Bobbin and Mass

Imagine that the joystick interface controls the motion of the bobbin. As the user moves the bobbin in the plane, the mass will move in the plane. If the user moves the bobbin in a circular motion, the mass will swing around and around as it would in the real world. A certain amount of physical coordination is required to get the mass spinning. This physical coordination is highly dependent upon the *feel* of the mass pulling on the string. Clearly this is a simulated interaction where user motion in real-time and mass feel in real time are both critical to performance. In other words, the algorithm that updates the

APPLE INC.
EXHIBIT 1017 - PAGE 113

dynamic simulation of the mass-string interaction must read user position and update the forces felt by the user very quickly (on the order of 1000 times per second). While the host computer could perform this closed-loop simulation, the communication interface between the host and the remote peripheral device is a bottleneck to rapid control. Also, if the host performed this rapid control loop, it would slow down other critical tasks the host needs to perform such as updating the graphics and controlling game play.

Fortunately, the manufacturers of force feedback hardware have solved these problems by giving the Distributed Dynamic Processing engine the ability to perform this closed loop simulation locally. In other words, the processor on board the force feedback peripheral device can track user motion, update the dynamic simulation of the "mass-on-a-string" based on how the bobbin would move based on joystick motion, and then update forces applied to the user. The DDP engine can do this at rates as fast as 1000 times per second, without putting any burden on the host processor or on the communication interface.

There are two important methods of coordinate the Dynamic Sling sensation generated locally by the DDP engine with software events happening on the host computer. The first method is to customize the feel of the ball-string sensation using parameters. The host can update the DDP engine by sending basic parameters that define the physics of the interaction. The parameters include *Ball_Mass, String_Length, String_Compliance*, and *Ball_Damping*. These four parameters allow for very diverse feel sensation.

The second method for coordinating the Dynamic Sling sensation generated locally by the I-Force 2.1 DDP engine with software events happening on the host is to take advantage of the Interim Reporting feature described previously in this section. As defined, Interim Reporting allows the local processor on board the peripheral device to provide the host with intermittent updates about the physical simulation. In this case, the processor reports the location of the simulated mass with respect to the joystick location.

Why should the peripheral device report the locations of a simulated mass in addition to reporting the locations of joystick itself? Lets say you are designing the next first person fighting game and you want your character to be able to wield a mace. As you know, a mace is a heavy metal ball with spikes that swings on the end of a chain. Your animators have developed a killer graphical display of swinging mace. You have created a compelling feel to go with the mace, by setting up your force feedback joystick with the

APPLE INC.
EXHIBIT 1017 - PAGE 114

appropriate Dynamic Sling parameters – you define the Ball_Mass, String_Length, String_Compliance, and Ball_Damping to make it feel just right. NOW you need to coordinate the killer animation displayed by the host with the awesome feel performed by the peripheral device. How do you do this? With interim reporting – just ask the peripheral device to report to the host the location of the simulated mass with respect to the joystick. The host can now use this data to display the mass in a location that corresponds with the feel. The end result – a thoroughly compelling graphical and physical representation that brings realism to a level never before possible.

Ball_Mass	Bass Mass defines the simulated physical mass of the ball on the end of the string. The heavier the mass, the stronger the pull for a given spin velocity. Also, the stronger the mass, the more energy you need to put in to get the ball spinning.
String_Length	String Mass defines the simulated distance between the center of the bobbin and the center of the mass. The longer the string, the longer it takes for the mass to make a complete revolution about the bobbin.
String_Compliance	String Compliance defines the compliance of the simulated cable that connects the bobbin and the mass. A compliant string will give the mass a bouncy feel as it spins. A stiff string will give the mass a smooth, rigid feel as it spins.
Ball_Damping	Ball Damping is best conceptualized as the "wind-resistance" on the ball as it revolves around the bobbin. If the Ball_Damping is low, the ball will have little resistance to motion. This means that if the user stops moving the bobbin, the ball will continue to spin for a long time until its motion dies out. If Ball_Damping is high, the opposite is true – the ball will come to rest very quickly when the user stops moving. This parameter is best understood by imagining what it would feel like to spin a ball on the end of a string in air versus in water. In water, the ball would come to rest very quickly if you stop spinning. In air, the ball would make a few revolutions before coming to rest.

APPLE INC.
EXHIBIT 1017 - PAGE 115

5.9 Dynamic Paddle - *Lets you feel a ball-paddle interaction*

Dynamic Paddle is a compelling sensation that simulates the feel of hitting a ball (or other projectile like a puck, asteroid, cannon-ball, etc...) with a paddle (or other compliant object like a rubber-band, force-field, pillow, etc...). As with a real ball-paddle interaction, the Dynamic Paddle sensation is such that the faster the ball hits the paddle, the harder the impact. Also, how the user reacts during the ball-paddle interaction, greatly changes the feel. For example, if the user tries to cushion the blow and slow-down the ball, the feel is very different than if the user tries to resist the blow and accelerate the ball.

The best way to understand Dynamic Paddle is to think of the interface device, such as the joystick, controlling the location of a "paddle" which has just been hit by a "ball". The weight of the ball, the velocity of the ball, the direction of motion of the ball, and the stiffness of the paddle, all affect the feel of the interaction. All of these physical factors can be defined with basic parameters such as the *Ball_Mass, Initial_Velocity, Paddle_Compliance, Paddle_Damping*, and *Gravity*. Also, how the user moves during the interaction greatly effects the feel. It is also important to understand that how the user moves during the interaction also effects how the ball bounces off the paddle – a cushioned blow will eject a slow moving ball while a stiff blow will eject a fast moving ball.

To clarify this point, lets examine the ball-paddle interaction in more detail. The basic ball-paddle interaction can be represented graphically in the figure below. Of course this physical interaction can be abstracted beyond the basic notion of a ball compressing and rebounding off a paddle – the physical sensation can represent any "elastic collision" where a mass impacts a flexible surface, compresses the surface, and then rebounds off the surface. It could be an asteroid bouncing off a force-field, an acrobat jumping on a trampoline, or a race-car car bouncing off a... Regardless of the metaphor used, the physical interaction can be conceptualized as having two distinct stages – a *compression* stage and a *rebound* stage. These are shown in Figure 5-2:

APPLE INC.
EXHIBIT 1017 - PAGE 116

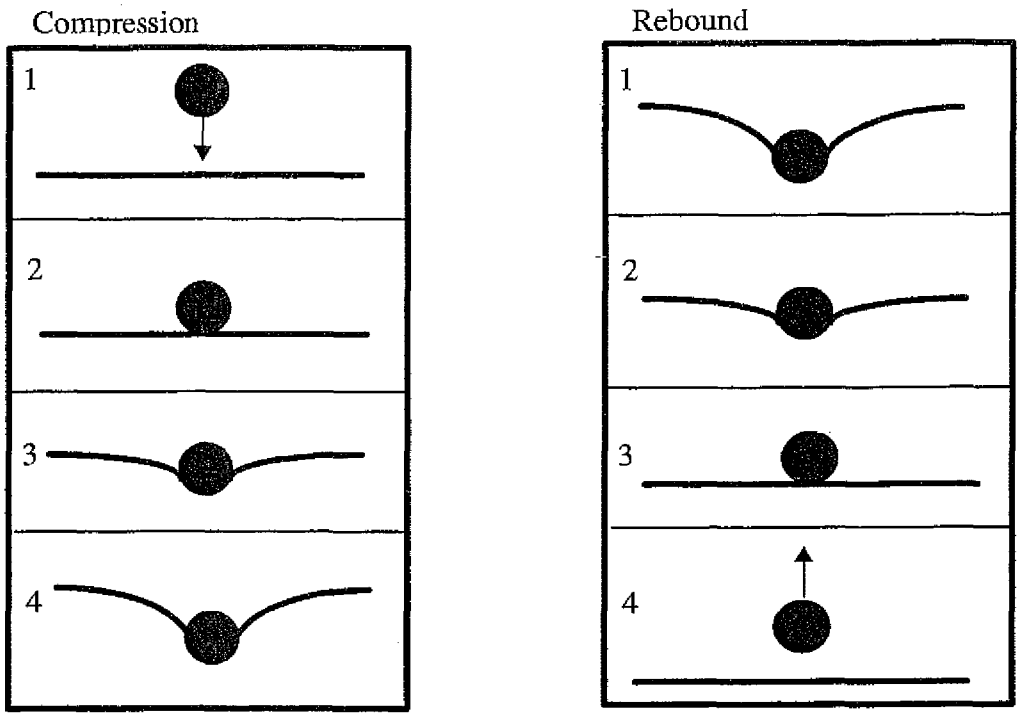


Figure 5-2 Stages of Paddle Interaction

Because the user is controlling the location of the paddle during both the compression and rebound stages, the user's motions effects the physical behavior of the overall interaction. If the user pulls the paddle back during the compression stage with skilled timing, the impact will be cushioned and the rebound will be dulled. If the user pumps the paddle forward during the rebound stage with appropriate timing, the paddle will pump energy into the ball and it will leave the paddle with extra energy behind it.

Interim Reporting - Because the user's reaction during the interaction affects the outcome of the event – it is very useful to use the Interim Reporting feature of the I-Force 2.1 Processing Core described earlier in this chapter. This interim reporting feature serves two function: a) to provide a real-time update of ball location with respect to the paddle (joystick) during the interaction and b) to report the final velocity of the ball when it leaves the paddle so that the host can update the software with an appropriate final state. Let's explore this feature further using the following example.

APPLE INC.
EXHIBIT 1017 - PAGE 117

Example:

You are designing a game where simulated asteroids (balls) are going to impact your space ship's force field (paddle). The asteroids compress the force field and bounce off. Now, lets say you make your game so that skilled players can deflect the asteroids towards their opponents if they cushion the blow just right. Now, lets say that your animator has done a great job of representing the graphical impact of the asteroid hitting the force field and bouncing off. At the same time, you create a compelling feel by defining the basic parameters such as Ball_Mass, Initial Velocity, and Paddle_Compliance to represent the feel of an asteroid hitting the space ship. All that is left is the coordination – at the moment that the asteroid hits the space ship, you command the DDP engine of the force feedback device to produce a Dynamic_Paddle sensation with appropriate parameters. Those parameters reflect the initial velocity of the asteroid, so that the sensation correlates well with game play. Now the DDP engine computes the high-fidelity dynamic simulation. With interim reporting, the position of the asteroid with respect to the joystick is reported to the host. This value lets you update the animation to show how deeply the force field is stretched during the collision.

Now, imagine that the user reacts to the blow and deflects the asteroid using a subtle wrist snap so that it should continue forth and pummel an opponent ship. With interim reporting, the final velocity of the asteroid is reported back to the host so the game can maintain coordination and have the asteroid fly off in the right direction. This creates a gaming interaction based so deeply on physical skill and dexterity that it is more like an interactive physical sport than a passive visual game.

To use the Dynamic_Paddle feature, programmers need only define the following simple parameters.

Ball_Mass	Defines the simulated physical mass of the ball impacting the paddle. The heavier the mass, the stronger the impact.
Initial_Velocity	Defines the physical velocity of the ball at the moment of impact. This initial velocity has both a magnitude and direction, both of which greatly affect the feel.

APPLE INC.
EXHIBIT 1017 - PAGE 118

Paddle_Compliance	Defines the springiness of the paddle surface. A compliant paddle will stretch and snap back with a large displacement as if it were a loose rubber band. A stiff paddle will not stretch very far as if it were the tight strings of a tennis racquet.
Paddle_Damping	Defines how much momentum is lost within the paddle as the ball bounces off. A paddle with low Paddle_Damping will repel a projectile with the same velocity as it had upon impact. A paddle with high Paddle_Damping will absorb much of the momentum during the interaction so that the projectile bounces off much slower than when it impacted.
Gravity	Defines a physical bias upon the projectile that represents the acceleration of gravity. This parameter has both a magnitude and a direction. It is useful for gaming scenarios where the mass is a ball that is being bounced in a gravitational field (a ball bouncing into the air off a tennis racquet) but is not useful for scenarios where the ball is not effected by a directional gravitational pull (like the asteroid bouncing off a space ship).

APPLE INC.
EXHIBIT 1017 - PAGE 119

5.10 Dynamic Control Law

Dynamic Control Law lets the advanced programmer control the Distributed Dynamic Processing engine directly by defining parameters at the lowest level. This is for the ambitious programmer who wants to go beyond the previously defined dynamic sensations and achieve the most general sensations possible within the DDP hardware limitations.

The best way to understand the Dynamic_Control_Law is to think of the physical simulation that is being performed by the DDP engine. This simulation includes a dynamic mass that is connected to the peripheral device (Joystick handle, Steering Wheel, etc...) by a simulated spring and a simulated damper. The graphical representation in Figure 5-3 depicts this simple physical model. The Dynamic Mass is the block marked M and the spring and damper are shown as the two graphical symbols connecting the mass to the joystick. It should be noted that the following diagram shows one dimension, Dynamic Control Law can be extended to two dimensions for 2D interface devices and three dimensions for 3D interface devices.

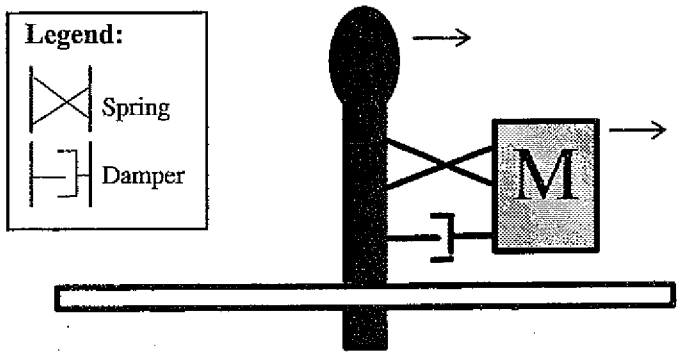


Figure 5-3 Physical Model of the Dynamic Control Law

As you might have guessed, when the joystick moves, the simulated mass moves because the spring and the damper link the two systems physically. Depending upon how the mass, the spring, and the damper parameters are defined, the mass might jiggle, jerk, or sluggishly lag behind the joystick. Also, there are initial conditions that can be defined to help tune the feel sensation. These initial conditions include the initial position of the

APPLE INC.
EXHIBIT 1017 - PAGE 120

mass with respect to the joystick (this defines an initial stretch of the spring) as well as an initial velocity of the mass. Finally, there is an ambient damping parameter that defines the simulated medium that the mass is moving in. A high ambient damping implies the mass is in a thick fluid while a low ambient damping implies the mass is in air.

Below is a brief description of each parameter used by the Dynamic_Control_Law sensation:

Dynamic_Mass:	The mass of a dynamic body.
Dynamic_Stiffness:	The stiffness of a spring between the joystick and the dynamic body
Dynamic_Damping:	A damping resistance on the joystick based on the relative velocity between the dynamic body and the joystick
Ambient_Damping:	A damping resistance on the dynamic body with respect to a fixed frame
Initial Velocity:	The initial velocity of the dynamic body upon onset of the sensation
Initial Position:	The initial position of the dynamic body with respect to the joystick handle defined upon onset of the sensation.
Deadband:	The slop or play that is artificially induced between the dynamic body and joystick.
Gravity:	Defines a gravitational force acting on the Dynamic Mass. Unlike in the real world, you can have gravity in any direction (+y, -y, +x, or -x).

APPLE INC.
EXHIBIT 1017 - PAGE 121

6. The Programming Model for Simulated FEEL

6.1 Overview

This Chapter introduces the programming foundation for developing feel sensations within Windows and DOS environments. The software structure presented is based on the **DirectX 5** force feedback interface that was designed through a collaborative effort of Microsoft Corporation and Immersion Corporation. While the published DirectX 5 specification inherently includes sample structures for basic force feedback features, the true power of the DirectX force feedback specification is its extensibility. In many ways, the DirectX 5 force feedback specification is simply a framework within which hardware makers can define the features and functionality of their products. DirectX 5 allows manufacturers of force feedback hardware to *define* force feedback functionality using DirectX upon initialization of a hardware device. This is achieved through a polling process where the hardware reports its capabilities to the host allowing DirectX to *enumerate* the supported features.

The fact that every manufacturer of force feedback hardware can provide a unique and distinct force feedback feature-set under DirectX is potentially a problem for programmers. *How do you know what basic feature-set should be used to so that you support the functionality of the greatest number of devices?* Fortunately for programmers, there is a hardware-standard called I-FORCE that has been adopted by most makers of gaming peripherals to solve this very problem. I-FORCE is the processing core that has been endorsed by major makers of force feedback products including Logitech Inc., CH Products, Advanced Gravis, InterAct, SC&T International, ACT Labs, Interactive IO, Nuby Manufacturing and others. As listed in Section 1.5, each of these makers will ship product (or is currently shipping product) that uses the licensed I-FORCE processing core. Currently shipping devices use the I-FORCE 1.01 core while future products will support the I-FORCE 2 core.

Because Immersion Corporation, the inventor of I-FORCE, was a collaborator in the definition of the DirectX 5 specification, DirectX is capable of supporting all the features of the advanced I-FORCE 2.0 processing core. The basic features are explicitly supported within the published DirectX specification. The advanced features are enabled through the *I-FORCE Extensions to DirectX*. These I-FORCE extensions are enabled through the enumeration process that automatically occurs upon connecting to any I-FORCE device.

APPLE INC.
EXHIBIT 1017 - PAGE 123

This seamless integration of DirectX software and advanced I-FORCE hardware allows programmers to continually take advantage of the latest force feedback features being offered by hardware makers.

The following pages will describe the DirectX foundation and describe how to program for force feedback devices.

APPLE INC.
EXHIBIT 1017 - PAGE 124

6.2 The Conceptual Model Shared by DirectX and I-FORCE

When programming force feedback devices, the fundamental goal is to design and implement feel sensations, referred to as "Effects". As you might expect, to design a force feedback sensation under DirectX and I-FORCE you use a simple command **CreateEffect** that creates and initializes an instance of the desired feel sensation. To inquire what types of sensations are supported by a particular force feedback hardware device, you poll the device with **EnumEffects**. To cause a force feedback device to generate a sensation defined by a given effect, you would use **StartEffect**. Of course there are many other functions that allow you to find the status of an effect, terminate an effect, but overall the process is simple and clear.

The Effects that force feedback devices are capable of producing are decomposed into three high level classes. As described though out this book, the *classes* include **Conditions**, **Waves**, and **Dynamics**. Each of these classes of sensations can be further decomposed into sensation *types*. For example, Springs, Dampers, and Friction are all *types* of Condition Effects. Sine-Wave, Square-Wave, and Triangle-Wave are all types of Wave Effects. Dynamic-Recoil, Dynamic-Liquid, and Dynamic-Collision are all types of Dynamic Effects.

When defining a force feedback sensation, the first step in the process is to choose the type. Once a type is chosen, DirectX will automatically know what parameters the force feedback device requires to instantiate that type. For example, if you were to create an effect of type *Spring*, DirectX would know that the parameters required to define the spring sensation include the *positive stiffness*, *negative stiffness*, *spring offset*, *positive saturation*, *negative saturation*, and *deadband*. Such parameters are all stored in a DirectX structure called the **Type Specific Parameter Structure.**

For efficiency, each class of force feedback sensation has it's own type specific parameter structure that is used as the template for most of the sensation types within that class. For example, there is a **Condition Struct** that is used for most Condition Effects. There is a **Periodic Struct** that is used for most Wave Effects. And there is a **Dynamic Struct** that is used for most Dynamic Effects. Of course there are some types of sensations in each class that have parameter requirements that are unique. For example, Ramp is a type of a

APPLE INC.
EXHIBIT 1017 - PAGE 125

Wave that is not periodic and therefore has its own struct called the **Ramp Struct**. Similarly, Texture is a Condition that has unique parameters and therefore has its own struct called the **Texture Struct**. As will be described in great detail in the following section, there are also basic type specific parameter structures such as the Constant Struct, Wall Struct, Barrier Struct, Circle Struct, Advanced Periodic Struct, and Custom Struct.

As a whole, the Effect Structure allows a force feedback sensation of any type to be defined under DirectX. Depending upon the effect *type*, the appropriate parameter structures are used. The complete architecture can be represented as follows:

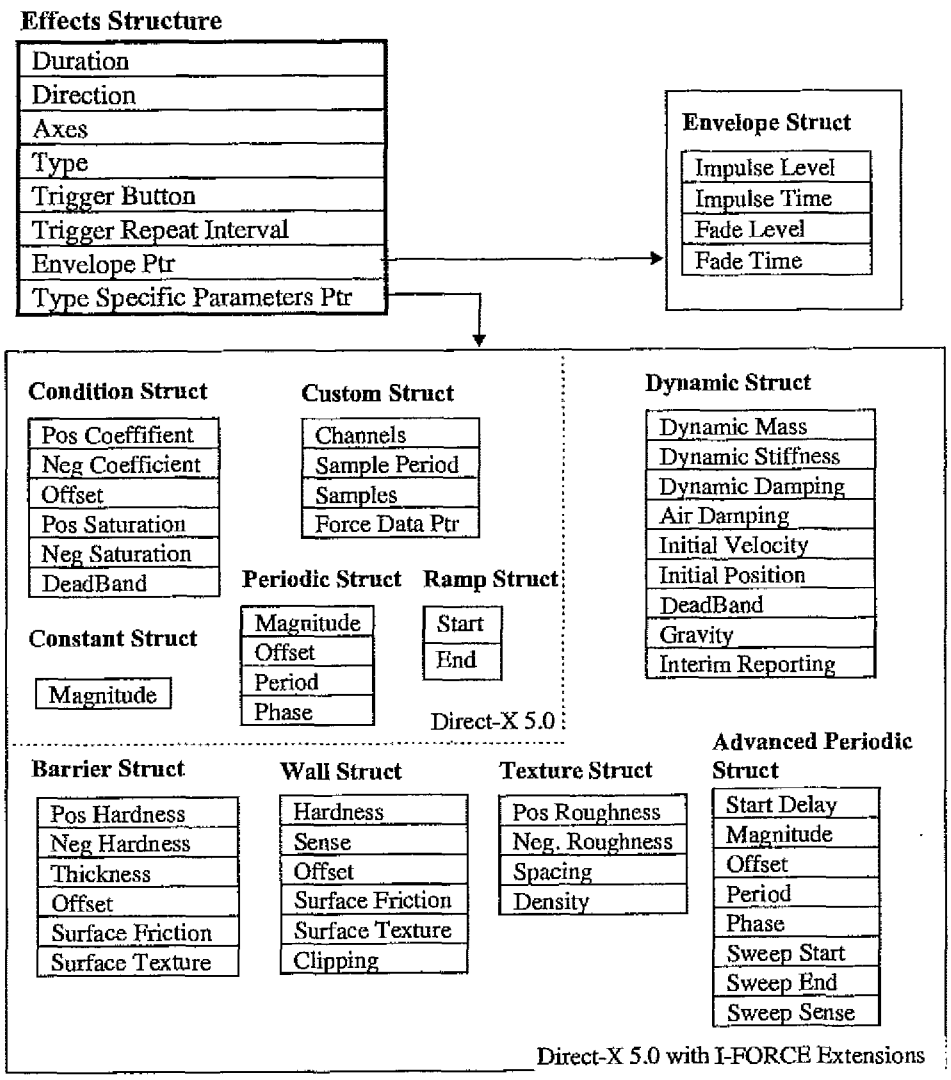


Figure 6-1 Complete Architecture

APPLE INC.
EXHIBIT 1017 - PAGE 126

How Does DirectX know what parameters are needed for a particular Effect type? Simple, every type of force feedback sensation whether it be a texture or a spring has a globally unique ID, or GUID associated with it. The hardware driver that links the force feedback peripheral to DirectX knows what parameters are required based on the GUID definition for each and every sensation.

The following pages will describe each of the DirectX structures in greater detail.

APPLE INC.
EXHIBIT 1017 - PAGE 127

6.3 The Effect Structure

As described above, the primary aspect of defining an Effect is to specify the *type*. Once a type is defined, there are a number of type specific parameters that need to be define. In addition to the parameters that are unique to the given sensation type, there are number of universal parameters that need to be defined for all Effects. These universal parameters include Duration, Direction, Axes, Trigger Button, and Trigger Repeat Interval. All of these parameters, along with the type definition are stored within the **Effect Structure**. In addition, if the effect is of the Wave class, there is an additional set of Envelope Parameters need to be defined. These Envelope parameters are stored within an Envelope Structure as shown below:

Effects Structure

Duration	0 to 65 million
Direction	ptr to direction info
Axes	ptr to axis infor
Trigger Button	id or offset
Trigger Repeat Interval	0 to 65 million
Envelope Ptr	ptr to envelope
Type Specific Parameters Ptr	ptr to effect type

Duration: Duration is a parameter that simply defines how long the given sensation will be active for. Duration can be any value between 1 and 65 million where the units are microseconds. Therefore the maximum duration for a given effect is about 65 seconds. Anything longer than 65 seconds would essentially be an infinite, or endless Effect. This can be achieved using the #DEFINE value INFINITE.

Axes: Axes is a parameter that defines which axis or axes are influenced by the given effect. For example, a spring effect could be applied to a single axis of a joystick or could be applied to both axes of a

APPLE INC.
EXHIBIT 1017 - PAGE 128

joystick. In the general case, it could be applied to n axes of an N dimensional interface.

Direction: Direction is a parameter that defines the direction the given effect is to be applied. Of course this depends upon which axes are active. Direction conventions can be Cartesian, Polar, or Spherical and will vary from one device to another. The case of a single axis device is easily defined. In the case of a multiple axis device, the situation is complicated as the axes in the device may or may not be directly related to each other. For a complete description of Direction Conventions, see section 6.5, later in this chapter.

Trigger Button: Trigger button is a parameter that is directly related to the Button Reflex concept introduced in section 1.2c of this text. As described previously, there are often times when you want a given effect to be generated (triggered) in response to a local event such as the press of a button on a joystick. The Trigger value indicates which button should trigger the effect being defined.

Trigger Repeat Interval:

Trigger Repeat Interval is a parameter that is also related to the Button Reflex concept introduced in section 1.2c of this text. As defined by the Trigger Button parameter above, a given effect can trigger in response to a locally detected button press. If the player holds that button down for a given period, it is often desirable to have the effect trigger again and again. The Trigger Repeat Interval defines how long the user must hold the button down for the effect to repeat. This value is very useful in coordinating recoil sensations with automatic weapons that fire repeatedly at a given interval. The Trigger Repeat Interval can be any value between 1 and 65 million where each increment is one microsecond.

Gain: Gain is a parameter that is a global scaling factor for the magnitude of the given effect. This is not a very widely used parameter.

APPLE INC.
EXHIBIT 1017 - PAGE 129

The Envelope Structure is used to shape Wave effects as described in Section 4.2 entitled Impulse Wave Shaping. The Envelope structure takes four parameters that represent the Impulse Level, Impulse Time, Fade Level and Fade Time. To understand the usage and implications of these parameters on a Wave effect, please refer to Section 4.2.a of this text.

Envelope Struct

Impulse Level	0 to 10000
Settle Time	0 to 65 million
Fade Level	0 to 10000
Fade Time	0 to 65 million

Impulse Level

Impulse Level is a variable (defined as a percentage of full) that describes how much "kick" should be delivered upon initial execution of the Wave as shown in the diagram below. Delivering sensations with a sharp initial Impulse is an important way to accentuate the sensation. This is because the human perceptual system is more sensitive to sharp transients in force than it is to static magnitude of force. The effect of Impulse Level can be described as effecting the "crispness" of a physical sensation. An effect with a high Impulse Level will feel abrupt and intense. An effect with a low Impulse Level will feel subtle and subdued.

Settle Time

Settle Time is a variable (defined as a duration in microseconds) that describes how quickly the Force Signal will settle from the Impulse Level to the steady state magnitude. Typically the Settle Time is very short as compared to the Duration of the entire Wave. While the Duration of an Wave can be on the order of hundreds of milliseconds or even seconds, Settle Time is usually on the order of tens of milliseconds.

Fade Level

Fade Level is similar to Impulse Level, but it defines the termination level rather than the onset level of a wave sensation. Like an Impulse, a Fade is most effective when it dictates a very sharp transition in force rather than a slowly changing force. For this reason, Fade Level is often set to zero.

APPLE INC.
EXHIBIT 1017 - PAGE 130

Fade Time Fade Time is similar to Settle Time in that it defines how quickly the Force Signal will settle from the steady state magnitude to the Fade Level. This variable is only useful in the rare situation when you want a force sensation to slowly decay.

Note: The *Steady State Magnitude* level shown in the diagram below is not defined by a parameter in the Envelope Struct. Instead it is defined by a *Magnitude* Parameter that is part of the Periodic Struct as shown in Figure (The Programming Model for Simulated FEEL) and described in Section 6.5.b. This is because the Magnitude is a general parameter for all Periodic Waves, regardless of whether or not an envelope is applied.

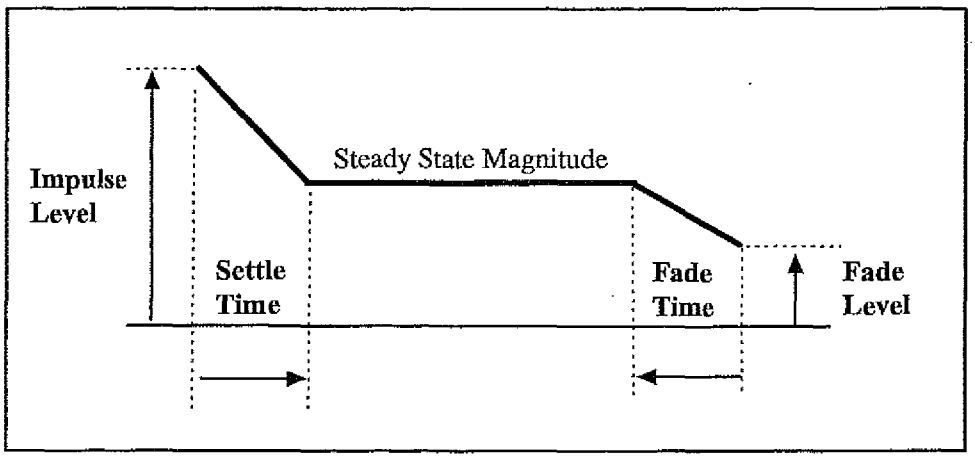


Figure 6-2 Envelope Parameters

Type Specific Structs: Having defined the **Effect Struct** and the **Envelope Struct**, the only part of the force feedback data structure left to define are the **Type Specific Structs.** As described previously, the overall Effect Struct includes a *Type Specific Parameter Pointer* that points at additional Parameter Structs specific for the type of sensation selected. The following sections of this text describe each of these sensation types in detail.

APPLE INC.
EXHIBIT 1017 - PAGE 131

6.4 The Type Definitions

As described in the previous section, the key step in building a force feedback sensation is defining the *type* of the Effect. Every type has a unique GUID that represents the given sensation. All force feedback devices will report which upon initialization, what effect types it is capable of producing. While there is no standard set of sensation types, all force feedback peripheral devices that employ the I-FORCE 2.0 hardware core can support the following master set of sensations:

6.4.a Condition Type Definitions

Type	Type Specific Parameter Struct	Envelope Support
Spring	Condition Struct	NO
Damper	Condition Struct	NO
Inertia	Condition Struct	NO
Friction	Condition Struct	NO
Angle Spring	Condition Struct	NO
Axis Barrier	Barrier Struct	NO
Angle Barrier	Barrier Struct	NO
Axis Wall	Wall Struct	NO
Angle Wall	Wall Struct	NO
Texture	Texture Struct	NO

APPLE INC.
EXHIBIT 1017 - PAGE 132

6.4.b Wave Type Definitions

Type	Type Specific Parameter Struct	Envelope Support
Vector Force	Constant Struct	YES
Smart Pop	Constant Struct	NO
Sine Wave	Periodic Struct	YES
Square Wave	Periodic Struct	YES
Triangle Wave	Periodic Struct	YES
Saw-Up Wave	Periodic Struct	YES
Saw-Down Wave	Periodic Struct	YES
Sweep Sine Wave	Advanced Periodic Struct	YES
Sweep Square Wave	Advanced Periodic Struct	YES
Sweep Triangle Wave	Advanced Periodic Struct	YES
Sweep Saw-Up	Advanced Periodic Struct	YES
Sweep Saw-Down	Advanced Periodic Struct	YES
Force Profile	Custom Struct	YES
Ramp	Ramp Struct	YES

6.4.c Dynamic Type Definitions

Type	Type Specific Parameter Struct	Envelope Support
Dynamic Recoil	Dynamic Struct	NO
Dynamic Impact	Dynamic Struct	NO
Dynamic Liquid	Dynamic Struct	NO
Dynamic Inertia	Dynamic Struct	NO
Dynamic Center Drift	Dynamic Struct	NO
Dynamic Control Law	Dynamic Struct	NO
Dynamic Sling	Dynamic Struct	NO
Dynamic Paddle	Dynamic Struct	NO

APPLE INC.
EXHIBIT 1017 - PAGE 133

6.5 Direction Conventions

Direction conventions can be Cartesian, Polar, or Spherical and will vary from one device to another. The case of a single axis device is easily defined. In the case of a multiple axis device, the situation is complicated as the axes in the device may or may not be directly related to each other. For example, in a joystick, the X and Y axes are related by the motion of the stick itself. However, in a steering wheel device with a pedal attachment, the motion of wheel is completely independent of that of the pedals. In considering multiple axes devices, the concept of direction should be applied to axes that are directly related to one another.

6.5.a 1D Devices

A single axis device will have the positive direction defined as a force output that would oppose displacement in a positive direction on the axis.

6.5.b 2D Devices

A two axis device can be represented as the X and Y axes of a joystick. If the direction is defined as Polar, then the direction is the angle the force is COMING FROM based on the stick's perspective. Directions are also defined based on the compass angles. For example, a force coming from behind the stick on the right side would have a direction of 135°.

6.5.c 3D Devices

A three axis device can be represented by using the definition of a spherical coordinate system. Two angles are used in this definition. The first describes the rotation in the XY plane from the x axis. This is angle starts at zero when the force is coming from the

APPLE INC.
EXHIBIT 1017 - PAGE 134

positive X axis direction and increases as the force rotates to be from the positive Y axis. The second angle describes the rotation of the force away from the positive Z axis. For a force coming purely from the positive Z direction, this angle has a value of 0 degrees. For a force purely in the XY plane, this angle has a value of 90 degrees.

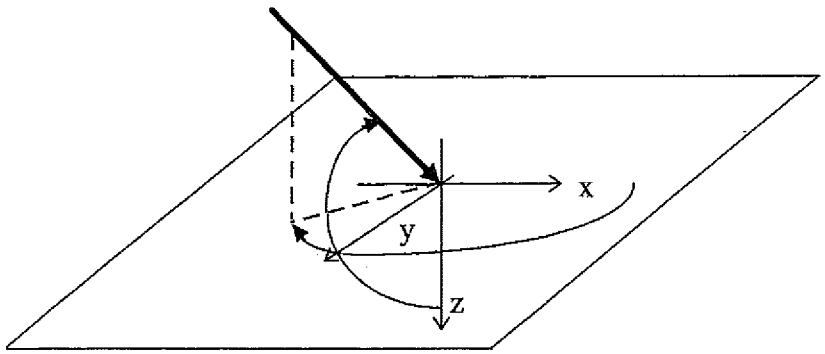


Figure 6-3

6.5.d nD Devices

A device with more than 3 axes will apply additional forces as rotational moments only. The specification of these moments is defined as the rotational components around the X, Y, and Z axes.

APPLE INC.
EXHIBIT 1017 - PAGE 135

6.6 Type Specific Parameter Structure

6.6.a Condition Struct

This struct is used for Condition effect types that include Spring, Damper, Inertia, Friction, and Angle Spring. The parameter block is as shown below.

Condition Struct

Pos Coeffifient	-10,000 to 10,000
Neg Coefficient	-10,000 to 10,000
Offset	-10,000 to 10,000
Pos Saturation	0 to 10,000
Neg Saturation	0 to 10,000
DeadBand	0 to 10,000

Pos. Coefficient / Neg. Coefficient:

These values correspond with the physical constant for the particular condition. For example, for Springs these values represent the Stiffness for positive displacement and the Stiffness for negative displacement. For Dampers these values represent the Damping for positive velocity and the Damping for negative velocity. For Inertia it represents the Mass for positive accelerations and the mass for negative accelerations (Note, most hardware devices only allow a positive coefficient for Inertia). For Friction these values represent the Friction Coefficient for positive and negative motions. See Chapter 3 for a more detailed description of the above physical properties.

Offset: This tells the peripheral device where to make the switch between Positive and Negative Coefficients. This is most clear by looking at the diagrams in Chapter 3. For example, in a Spring Condition, the Offset is the location (defined as an offset from 0) where the spring flips from stretching to compressing. In a Damper

APPLE INC.
EXHIBIT 1017 - PAGE 136

Condition, the Offset is the velocity (defined as an offset from 0) where the velocity swaps from positive to negative damping. Offset is usually not used for Inertia or Friction.

Pos. Saturation / Neg. Saturation:

For all Condition types, these values simply define a maximum positive force and a maximum negative force that can be produced by the given Effect. This is useful for limiting how much influence the effect might have upon the user. For example, in a Spring Condition these values represent how much force can be imposed upon the peripheral during positive and negative displacements respectively.

Deadband: Defines the size of a region around the Offset point where the force output from the given Condition is zero. This is best understood by referring to Chapter 3.

6.6.b Periodic Struct

This struct is used for Wave effect types that include Sine Wave, Square Wave, Triangle Wave, Saw-Up Wave, and Saw-Down Wave. Please refer to Chapter 4 for a complete description of these Wave effect types.

Periodic Struct

Magnitude	0 to 10,000
Offset	-10,000 to 10,000
Period	0 to 65 million
Phase	0 to 35999

Magnitude: Defines the magnitude of the periodic signal. It should be noted that if an envelope is applied to the Wave effect, the Magnitude is the "steady state" level of the sensation – the envelope parameters define the starting and ending levels. The value is defined as a percentage of maximum force capability of the given device.

APPLE INC.
EXHIBIT 1017 - PAGE 137

Offset: The baseline level for the periodic signal. In other words, if an Offset is defined, the periodic signal will oscillate about the Offset value rather than about the zero force level. The value is defined as a percentage of maximum force capability of the given device.

Period: The duration of a single cycle of the given periodic function. This is defined graphically in Chapter 4. The value is defined in microseconds.

Phase: Determines where in the wave-form the periodic sensation begins. This values is defined in hundredths of degrees.

6.6.c Constant Struct

This struct is used to produce constant forces. Since these forces are generally produced as a function of time, the Constant Struct is considered part of the Wave class of sensations.

Constant Struct

Magnitude	-10,000 to 10,000

Magnitude: Defines the magnitude of the constant signal. It should be noted that if an envelope is applied to the constant signal, the Magnitude is the "steady state" level of the sensation – the envelope parameters define the starting and ending levels. The value is defined as a percentage of maximum force capability of the given device.

APPLE INC.
EXHIBIT 1017 - PAGE 138

6.6.d Custom Struct

This struct is used to produce a canned Wave sensation that is defined as sequence of discrete force values. The data that represents the digitized force profile is stored in a *force array*. The array may contain data for a single axis (single channel) or may contain force values that are to be applied to multiple axes (channels). Each element of the array is a force sample that is to be applied to the device for a duration known as the sample period.

Custom Struct

Channels
Sample Period
Samples
Force Data Ptr

Channels: The number of channels or axes represented in the force array

Sample Period: The duration of each sample in microseconds

Samples: The number of samples stored in the force array. It must be an integral multiple of the Channels

Force Data Ptr: A pointer to the array of force values.

6.6.e Ramp Struct

This struct is used to Wave effect representing a linearly ramping force value. The ramp is defined as a starting and ending value. The duration of the ramp, like all effect durations is defined in the Effect Structure.

Ramp Struct

Start	-10000 to 10000
End	-10000 to 10000

Start: The starting value for the ramping wave form

End: The ending value for the ramping wave form

APPLE INC.
EXHIBIT 1017 - PAGE 139

6.6.f Dynamic Struct

This struct is used for Dynamic effect types that include Dynamic Recoil, Dynamic Impact, Dynamic Liquid, Dynamic Inertia, Dynamic Center Drift, Dynamic Control Law, Dynamic Sling, Dynamic Paddle. Review Chapter 5 for a complete description of these Dynamic effect types.

Dynamic Struct

Dynamic Mass	1 to 10000
Dyn Pos Stiffness	1 to 10000
Dyn Neg Stiffness	1 to 10000
Dynamic Damping	0 to 10000
Air Damping	0 to 10000
Initial Velocity	-10,000 to 10000
Initial Position	-10,000 to 10000
DeadBand	0 to 10000
Gravity	-10,000 to 10000
Interim Reporting	*flags*

Dynamic Mass: The simulated mass attached to the peripheral device handle.

Dynamic Stiffness: The stiffness of a simulated spring between the simulated mass and the device handle

Dynamic Damping: The damping of a simulated damper between the simulated mass and the device handle.

Air Damping: The damping between the simulated mass and the surrounding environment

Initial Velocity: The starting velocity of the simulated mass upon inception of the Dynamic effect.

Initial Position. The starting position of the simulated mass with respect to the handle of the peripheral device upon inception of the Dynamic Effect. This position essentially defines a pre-stretch of the simulated spring that connects the mass to the handle.

APPLE INC.
EXHIBIT 1017 - PAGE 140

Dead-Band: Defines the size of the "slop" that exists between the simulated mass and the peripheral

Gravity: Defines a gravitational force acting on the Dynamic Mass. Unlike in the real world, you can have gravity in any direction (+y, -y, +x, or -x).

Interim Reporting: Defines whether intermediate position and/or velocity data of the Dynamic Mass should be reported to the host application for synchronizing graphics. This property will be supported in the Advanced Dynamic struct.

APPLE INC.
EXHIBIT 1017 - PAGE 141

6.6.g Barrier Struct

This struct is used for Barrier Sensations described in detail in Section 3.8. A Barrier creates the feel of encountering a hard surface that can be penetrated. If you push into the Barrier with enough force, you will "pop" through to the other side. The feel of the penetration is governed by hardness and thickness parameters as described below.

Barrier Struct

Pos Hardness	-10000 to 10000
Neg Hardness	-10000 to 10000
Thickness	1 to 10000
Offset	-10000 to 10000
Angle	0 to 35999
Surface Friction	*0 to 10000*
Surface Texture	*0 to 10000*

Pos. Hardness / Neg. Hardness:

Defines how rigid or compliant the surface of the barrier feels when pressed against from the positive or negative direction respectively.

Thickness: Governs how easy or difficult it is to penetrate a given Barrier. A thick Barrier will be hard to penetrate and a thin barrier will be easy to penetrate. Of course the hardness parameters also affect penetration.

Offset: Defines the location of the barrier with respect to the origin. This is usually a distance along a single axis or (for multiple axis effects) it is a radial distance along a radii whose orientation defined by the Direction parameter of the Effects Struct.

Angle: Defines the compass angle of the barrier in hundredths of a degree.

***Surface Friction*:** Defines the rubbing resistance felt when pushed up against the barrier surface and moving along the length of the barrier. This property will be supported in the Advanced Barrier struct.

APPLE INC.
EXHIBIT 1017 - PAGE 142

Surface Texture: Defines the rubbing texture felt when pushed up against the barrier surface and moving along the length of the barrier. This property will be supported in the Advanced Barrier struct.

APPLE INC.
EXHIBIT 1017 - PAGE 143

6.6.h Wall Struct

This struct is used for Wall Sensations described in detail in Section 3.7. A Wall creates the feel of encountering a hard surface that can not be penetrated. The feel of pushing against the Wall as well as rubbing along the Wall is governed by the parameters described below.

Wall Struct

Hardness	-10000 to 10000
Sense	flags
Offset	-10000 to 10000
Angle	0 to 35999
Surface Friction	*0 to 10000*
Surface Texture	*0 to 10000*
Clipping	*flags*

Hardness: Defines how rigid or compliant the surface of the wall feels when pushed against

Sense: Simply defines which way the wall is facing (i.e., which side of the Wall you are on).

Offset: Defines the location of the Wall with respect to the origin. This is usually a distance along a single axis or (for multiple axis effects) it is a radial distance along a radii whose orientation defined by the Direction parameter of the Effects Struct.

Angle: Defines the compass angle of the wall in hundredths of a degree.

Surface Friction: Defines the rubbing resistance felt when pushed up against the wall surface and moving along the length of the wall. This property will be supported in the Advanced Wall struct.

Surface Texture: Defines the rubbing texture felt when pushed up against the wall surface and moving along the length of the wall. This property will be supported in the Advanced Wall struct.

APPLE INC.
EXHIBIT 1017 - PAGE 144

Clipping: This is a binary parameter that can be ON or OFF as described in detail in Section 3.7. When ON, this parameter prevents the hardware peripheral from reporting position data that represents penetration of the wall, even though the wall is not truly rigid. In other words, the hardware devices enhances the illusion of rigidity by "clipping" the position data when the wall is penetrated. This property will be supported in the Advanced Wall struct.

APPLE INC.
EXHIBIT 1017 - PAGE 145

6.6.i Advanced Periodic Struct

This struct is used for Advanced Periodic Wave Sensations described in detail in Chapter 4. This struct is very similar to the standard Periodic Struct. In fact, the Magnitude, Offset, Period, and Phase parameters are identical. However, the Advanced Periodic Struct include 4 additional parameters that greatly increase the flexibility of Wave sensations as described below:

Advanced Periodic Struct

Start Delay	0 to 65 million
Magnitude	-10000 to 10000
Offset	-10000 to 10000
Period	0 to 65 million
Phase	0 to 35999
Sweep Start Angle	0 to 35999
Sweep End Angle	0 to 35999
Sweep Sense	flags

Start Delay: This parameter that allows the peripheral device to wait some period before starting the given periodic function of force versus time. This allows multiple Waves to be triggered at the same time but allows them actually start playing forces at staggered intervals. This parameter can therefore be used to create very interesting sensations composed of multiple *sequenced* periodic waves.

Sweep Start /Sweep End: While standard periodic waves can only be generated along a given direction, Advanced Periodic can play the periodic form such that the direction sweeps between a *sweep start angle* and a *sweep end angle*. This feature can only be used on multiple-degree of freedom hardware devices such as joysticks. The sweep occurs at a linear rate over the duration of the Wave sensation. Sweep is described in more detail in Section 4.2c

Sweep Sense: This simply defines which way the sweep occurs between the Start and End angles. This can be either *clockwise* or counter *clockwise*.

APPLE INC.
EXHIBIT 1017 - PAGE 146

6.6.j Texture Struct

This struct is used for Texture Sensations described in detail in Section 3.6. A Texture is a spatial condition that gives the user a feel similar to dragging an object over a rough surface like a metal grating. The Texture sensation is defined by four simple parameters as defined below.

Texture Struct

Pos. Roughness	1 to 10000
Neg. Roughness	1 to 10000
Spacing	1 to 10000
Density	1 to 10000

Pos. Roughness / Neg. Roughness: Roughness defines the intensity of the texture sensation. You can think of Roughness as the grit of sandpaper, the feel depends both on the grit and how you rub your hand over it. Quality force feedback devices allow you to define a positive roughness and a negative roughness such that the intensity of the texture depends upon which direction the interface device is moving.

Spacing: Spacing defines the center to center spacing between the "bumps" in the texture. The smaller the spacing the finer the texture.

Density : Density defines the width of the "bumps" in the texture with a range of 1 to 10000 where 5000 means 50% of the center to center spacing of the bumps. A small density means the bumps are small with respect to the empty space between them.

APPLE INC.
EXHIBIT 1017 - PAGE 147

7. Implementation of Force Feedback using DirectInput

7.1 Overview

This chapter describes the programming details for designing force feedback sensations using DirectInput from Microsoft. As you will find in the sections below, the process is clear and methodical, but is somewhat involved. As a means of accelerating and simplifying this detailed process, Immersion Corporation has developed **I-FORCE Studio,** a toolset for DirectX. Using the I-FORCE Studio toolset (as described in detail in Chapter 9), programmers can define force feedback sensations using an intuitive graphical environment – once these sensations are designed, the I-FORCE tools automatically generate the DirectInput code for you.

While the I-FORCE Studio toolset frees you from the effort of creating the force feedback code described in this chapter, it is important that you review the process described below so you understand how to use and modify the automatically generated code. The following sections cover the basics of DirectInput specifically related to force feedback. It does not cover topics such as setting up DirectInput and handling keyboard and mouse input. Those topics are covered in Microsoft's Direct X 5.0 SDK. For your convenience, many of the examples below are similar to those provided within the Direct X 5.0 SDK but have been specifically tailored to force feedback implementation.

APPLE INC.
EXHIBIT 1017 - PAGE 149

7.2 Enumerating DirectInput Force Feedback Joysticks

Enumerating input devices is a required step in DirectInput. You can choose to enumerate only force feedback joysticks that are attached to the system. The following example using **IDirectInput::EnumDevices** does just that.

```
// already initialized earlier
// LPDIRECTINPUT  lpdi;

HRESULT hr;

hr = lpdi->lpVtbl->EnumDevices(lpdi, DIDEVTYPE_JOYSTICK,
       InitJoystickCallback, lpdi, DIEDFL_ATTACHEDONLY | DIEDFL_FORCEFEEDBACK);
```

A pointer to the DirectInput interface lpdi is required.

DIDEVTYPE_JOYSTICK is a constant that specifies that the device must be a joystick.

InitJoystickCallback is the address of your callback function that initializes a joystick each time one is found.

The fourth parameter can be any 32-bit value that you want to make available to the callback function. It can be the pointer to the DirectInput interface or a globally unique identifiers (GUIDs) that will be used to create device objects.

The flags DIEDFL_ATTACHEDONLY | DIEDFL_FORCEFEEDBACK will enumerate only devices that are physically attached to the computer *and* that support force feedback.

APPLE INC.
EXHIBIT 1017 - PAGE 150

7.3 Creating the DirectInput Force Feedback Device

To access force feedback in DirectInput, you must create **IDirectInputDevice2** objects rather than **IDirectInputDevice** objects. The following function is a C or C++ wrapper for **CreateDevice** that attempts to obtain the **IDirectInputDevice2** interface.

```
HRESULT IDICreateDevice2(LPDIRECTINPUT lpdi,
                                    REFGUID rguid,
                                    LPDIRECTINPUTDEVICE2 lpIF2Device2)
{
    LPDIRECTINPUTDEVICE lpDIDevice;
    HRESULT hres;

    // create a temporary standard DI device
    hr = lpdi->lpVtbl->CreateDevice(lpdi, rguid, &lpDIDevice, NULL);

    if (SUCCEEDED(hr)) {
      // create a DI device2 that supports force feedback
       hr = lpDIDevice->lpVtbl->QueryInterface(lpDIDevice,
                        &IID_IDirectInputDevice2,&lpIF2Device2);
      // release temporary standard DI device
       lpDIDevice->lpVtbl->Release(lpDIDevice);

    } else {
       *lpIF2Device2= 0;
    }
    return hr;
}
```

The function declares a temporary local pointer to a DirectInput device, *lpDIDevice*. Once the temporary device is created, **IDirectInputDevice::QueryInterface** is used to get a pointer *lpIF2Device2* to a **IDirectInputDevice2** device which supports force feedback. The temporary **IDirectInputDevice** pointer can then be released.

The REFGUID *rguid* identifies the instance of the **IDirectInputDevice2** interface to create and should be returned by **IDirectInput::EnumDevices**.

APPLE INC.
EXHIBIT 1017 - PAGE 151

7.4 Getting DirectInput Force Feedback Device Capabilities

Use **IDirectInputDevice::GetCapabilities** to determine if the device supports force feedback, and if so, whether it supports force feedback capabilities such as force saturation and positive and negative coefficients. This method returns the data in a **DIDEVCAPS** structure.

Here's an example that checks whether the joystick supports force feedback and is attached:

```
// LPDIRECTINPUTDEVICE2 lpIF2Device; // initialized previously

DIDEVCAPS  DIIF2DeviceCaps;
HRESULT    hr;
BOOLEAN    HasIF2;

DIIF2DeviceCaps.dwSize = sizeof(DIDEVCAPS);
hr = lpIF2Device->lpVtbl->GetCapabilities(lpIF2Device, &DIIF2DevCaps);
HasIF2 = ((DIIF2DevCaps.dwFlags & DIDC_ATTACHED)
        && (DIIF2DevCaps.dwFlags & DIDC_FORCEFEEDBACK));
```


APPLE INC.
EXHIBIT 1017 - PAGE 152

7.5 Generating DirectInput Force Feedback Device Effects
Example: Spring and Texture

To create an I-FORCE 2 effect, you must use the **IDirectInputDevice2::CreateEffect** member. You can use **IDirectInputDevice2::EnumEffects** to check if the desired effect is supported by the device. Once the effect is created, you must use the to **IDirectInputEffect::Download** and **IDirectInputEffect::Start** members to download and start the effect on the joystick.

Below is an example that creates and starts a **Spring** effect on the X-axis of the joystick and a **Texture** on the Y-axis:

```
// LPDIRECTINPUTDEVICE2 lpIF2Device; // initialized previously

LONG rglDirections[2];
DWORD rgdwAxes[2];
DIEFFECT dieffect;
DICONDITION dicondSpring;
DITEXTURE ditexture;
LPDIRECTINPUTEFFECT lpdieffStiffSpring;
LPDIRECTINPUTEFFECT lpdieffWashboard;
HRESULT hr;

dieffect.dwSize = sizeof(DIEFFECT);
dieffect.dwSamplePeriod = 0; // use default sample period
dieffect.dwTriggerButton = DIEB_NOTRIGGER;
dieffect.dwTriggerRepeatInterval = 0;
dieffect.rgdwAxes = rgdwAxes;
dieffect.rglDirection = rglDirections;
dieffect.dwGain = 10000; // maximum gain

dieffect.dwFlags = DIEFF_POLAR;
rgdwAxis[0] = DIJOFS_X;  // X axis
rgdwAxes[1] = DIJOFS_Y;  // Y axis
rglDirections[0] = 0; // specified in hundredths of degrees (not used by
conditions)
rglDirections[1] = 0; // must be 0 for polar coords
```


APPLE INC.
EXHIBIT 1017 - PAGE 153

```
dieffect.dwDuration = INFINITE;
dieffect.cAxes = 1;    // only play one axis (rgdwAxis[0] = X axis)
dieffect.lpEnvelope = NULL;
dieffect.cbTypeSpecificParams = sizeof(DICONDITION);
dieffect.lpvTypeSpecificParams = &dicondSpring;

// enumerate to see if Spring is supported
hr = lpIF2Device->lpVtbl-
>EnumEffects(lpIF2Device,EnumEffectCallback,&GUID_Spring,
                                          DIEFT_CONDITION)
```

EnumEffectCallback is the address of your callback function that handles the force feedback effects each time one is found The third parameter can be any 32-bit value that you want to make available to the callback function. It can be the globally unique identifiers (GUIDs) of the desired effect. objects.

```
// if Spring supported, fill in Spring condition structure and create effect
if (SUCCEEDED(hr)) {
   // fill in the spring condition parameters
   dicondSpring.lOffset = 0;           // centered at 0
   dicondSpring.lPositiveCoefficient = 1000;
   dicondSpring.lNegativeCoefficient = 1000;
   dicondSpring.dwPositiveSaturation = 10000;
             // maximum saturation level
   dicondSpring.dwNegativeSaturation = 10000;
             // maximum saturation level
   dicondSpring.lDeadBand = 0;                  // no deadband
   hr = lpIF2Device->lpVtbl->CreateEffect(lpIF2Device,
       &GUID_Spring, &dieffect, &lpdieffSpring, NULL);
}

// if Spring created, download and start it
if (SUCCEEDED(hr)) {
   hr = lpdieffSpring->lpVtbl->Download(lpdieffSpring);
   if (SUCCEEDED(hr)) {
      hr = lpdieffSpring->->lpVtbl->Start(lpdieffSpring,INFINITE,0);
             // play until stopped
   }
}
```


APPLE INC.
EXHIBIT 1017 - PAGE 154

```
dieffect.dwFlags = DIEFF_POLAR;
rgdwAxes[0] = DIJOFS_X;  // X axis
rgdwAxes[1] = DIJOFS_Y;  // Y axis
rglDirections[0] = 0;
      // 0 degrees = Y axis (specified in hundredths of degrees)
rglDirections[1] = 0; // must be 0 for polar coords
dieffect.dwDuration = INFINITE;
dieffect.cAxes = 2;
dieffect.lpEnvelope = NULL;
dieffect.cbTypeSpecificParams = sizeof(DITEXTURE);
dieffect.lpvTypeSpecificParams = &ditexture;

// enumerate to see if Texture is supported
hr = lpIF2Device->lpVtbl->EnumEffects(lpIF2Device,
      EnumEffectCallback,&GUID_Texture,DIEFT_ALL)

// if Texture supported, fill in Texture structure and create effect
if (SUCCEEDED(hr)) {
   // fill in the spring condition parameters
   ditexture.dwPosRoughness = 5000; // moderate roughness
   ditexture.dwNegRoughness = 5000; // moderate roughness
   ditexture.dwSpacing = 2000;   // relatively sparse bumps
   ditexture.dwDensity = 7500;   // relatively wide bumps
   hr = lpIF2Device->lpVtbl->CreateEffect(lpIF2Device,
      &GUID_Texture, &dieffect, &lpdieffWashboard, NULL);
}

// if Texture created, download and start it
if (SUCCEEDED(hr)) {
   hr = lpdieffWashboard->lpVtbl->Download(lpdieffWashboard);
   if (SUCCEEDED(hr)) {
      hr = lpdieffWashboard->->lpVtbl->Start(lpdieffWashboard,
            INFINITE,0); // play until stopped
   }
}
```


APPLE INC.
EXHIBIT 1017 - PAGE 155

8. The I-FORCE 2.0 Wrapper Functions

8.1 Overview

To simplify the process of using DirectX to create force feedback sensations, the creators of the I-FORCE processing core provide a set of wrapper functions that abstract the DirectX structures to a physically intuitive set of parameterized function calls. Examples of these wrapper functions are available from the I-FORCE 2.0 SDK via download from WWW.FORCE-FEEDBACK.COM.

While these wrapper functions greatly simplify programming force feedback in DirectX, the graphical tools described in Chapter 9 provide an even easier method of designing effects and implementing them in your program.

Following is a suggested list of I-FORCE 2.0 wrapper functions and their parameters and ranges. Section 8.5 gives detailed examples of what goes into the wrapper functions and can be copied and pasted to easily create your own DirectX force effects. Some force feedback devices may not support all the functions listed.

APPLE INC.
EXHIBIT 1017 - PAGE 157

8.2 CONDITION FUNCTION WRAPPERS

XSpring (KPos, KNeg, Offset, PosSat, NegSat, DeadBand)
Defines a spring condition along the X axis.

YSpring (KPos, KNeg, Offset, PosSat, NegSat, DeadBand)
Defines a spring condition along the Y axis.

Spring (XKPos, XKNeg, XOffset, XPosSat, XNegSat, XDeadBand, YKPos, YKNeg, YOffset, YPosSat, YNegSat, YDeadBand)
Defines two springs - Springs along both X and Y axes.

AngleSpring (KPos, KNeg, Offset, PosSat, NegSat, DeadBand, Angle)
Defines a spring at a particular angle.

CenterSpring (K, Saturation, DeadBand)
Defines a spring symmetrical in X and Y and returning to center.

SpringatCurrentLocation (K, Saturation, Deadband)
Defines a spring symmetrical in X and Y and returning to location of device when command is called.

Parameter	Range	Comment
KPos	-10000 to 10000	stiffness in positive direction
KNeg	-10000 to 10000	stiffness in negative direction
K	-10000 to 10000	stiffness in both directions
Offset	-10000 to 10000	center point offset of spring
PosSat	0 to 10000	maximum positive force
NegSat	0 to 10000	maximum negative force
Angle	0 to 35999	angle of effect
Saturation	0 to 10000	maximum force
DeadBand	0 to 10000	range around center where spring is inactive

APPLE INC.
EXHIBIT 1017 - PAGE 158

XDamper (BPos, BNeg, Offset, PosSat, NegSat, DeadBand

Defines a damping along the positive and negative directions of the X axis

YDamper (BPos, BNeg, Offset, PosSat, NegSat, DeadBand)

Defines a damping along the positive and negative directions of the Y axis

Damper (XBPos, XBNeg, XOffset, XPosSat, XNegSat, XDeadBand,YBPos, YBNeg, YOffset, YPosSat, YNegSat, YDeadBand)

Defines a general damping along the positive and negative directions of both the X and Y axes.

Parameter	Range	Comment
BPos	-10000 to 10000	damping in positive direction
BNeg	-10000 to 10000	damping in negative direction
Offset	-10000 to 10000	velocity at which damping force is zero
PosSat	0 to 10000	maximum positive force
NegSat	0 to 10000	maximum negative force
DeadBand	0 to 10000	range around zero where damping is inactive

XInertia (XMass, Saturation)

Defines an inertia felt along the X axis

YInertia (YMass, Saturation)

Defines an inertia felt along the Y axis

Inertia (Mass, Saturation)

Defines an inertia felt symmetrically along both the X and Y axes

Parameter	Range	Comment
Mass	1 to 10000	mass (identical in both directions)
Saturation	1 to 10000	maximum inertial force

APPLE INC.
EXHIBIT 1017 - PAGE 159

Friction (F)

Defines a symmetrical frictional resistance along both X and Y axes

Xfriction (FPos, FNeg)

Defines a frictional resistance along the positive and negative directions of the X axis

Yfriction (FPos, FNeg)

Defines a frictional resistance along the positive and negative directions of the Y axis

Parameter	Range	Comment
F	0 to 10000	friction force in both directions
FPos	0 to 10000	friction force in positive direction
FNeg	0 to 10000	friction force in negative direction

Texture (Roughness, Spacing, Density, Angle)

Defines a symmetrical texture felt along a defined orientation

XTexture (Roughness, Spacing, Density)

Defines a texture felt only along the X axis in both pos and neg directions

XTexture (PosRoughness, NegRoughness,Spacing, Density)

Defines an X-axis texture with different roughness in each direction

YTexture (PosRoughness, NegRoughness, Spacing, Density)

Defines an Y-axis texture with different roughness in each direction

Parameter	Range	Comment
Roughness	1 to 10000	roughness of bumps
PosRoughness	1 to 10000	roughness of bumps in pos direction
NegRoughness	1 to 10000	roughness of bumps in neg direction
Spacing	1 to 10000	spacing of bumps
Density	1 to 10000	width of bumps

APPLE INC.
EXHIBIT 1017 - PAGE 160

Wall (Hardness, Sense, Offset, Angle)

Defines a wall at a given orientation within the plane defined by the X and Y axes

HorizontalWall (Hardness,Sense,Offset,SurfaceFriction,SurfaceTexture,Clipping)

Defines a horizontal wall

VerticalWall (Hardness, Sense, Offset, SurfaceFriction, SurfaceTexture, Clipping)

Defines a vertical wall

Parameter	Range	Comment
Hardness	-10000 to 10000	hardness of wall
Sense	flags	wall on left or right of location
Offset	-10000 to 10000	location of wall
Angle	0 to 35999	angle of wall
SurfaceFriction	0 to 10000	surface friction on wall
SurfaceTexture	0 to 10000	surface texture on wall
Clipping	flags	flag to clip position

Barrier (PosHardness, NegHardness, Thickness, Offset, Angle)

Defines a barrier at a given orientation within the plane defined by the X and Y axes

Horizontal Barrier(PosHardness,NegHardness,Thickness,Offset,SurfaceFriction, SurfaceTexture)

Defines a horizontal barrier (as felt from either direction)

VerticalBarrier(PosHardness,NegHardness,Thickness,Offset,SurfaceFriction, SurfaceTexture)

Defines a vertical barrier (as felt from either direction)

Parameter	Range	Comment
PosHardness	-10000 to 10000	hardness of barrier in positive direction
NegHardness	-10000 to 10000	hardness of barrier in negative direction
Thickness	1 to 10000	thickness of barrier
Offset	-10000 to 10000	location of center of barrier
Angle	0 to 35999	angle of barrier
SurfaceFriction	0 to 10000	surface friction on barrier
SurfaceTexture	0 to 10000	surface texture on barrier

APPLE INC.
EXHIBIT 1017 - PAGE 161

8.3 WAVE FUNCTION WRAPPERS

VectorForce (Magnitude, Duration, Direction)
Create a force of indefinite duration of a defined magnitude and direction

ConstantForce (Magnitude, Duration, Direction)
Create a force of a defined duration, magnitude, and direction

ConstantForceTrigger (Magnitude, Duration, Direction, Trigger, TriggerRepeatInterval)
Create a force that triggers from a button of a defined duration, magnitude, and direction

Envelope Constant (Magnitude, Duration, Direction, ImpulseLevel, SettleTime, FadeLevel, FadeTime, Trigger, TriggerRepeatInterval)
Create a general constant force and apply an envelope to it. Note this will create a force profile that follows the envelope shape exactly.

Ramp (Start, End, Duration, Direction, Trigger, TriggerRepeatInterval)
Create a ramping force at a given direction and trigger from a button.

RampEnvelope (Start, End, Duration, Direction, ImpulseLevel, SettleTime, FadeLevel, FadeTime, Trigger, TriggerRepeatInterval)
Create a ramping force and apply an envelope to it. Note, this will create a very unique force profile.

Periodic (WaveForm, Magnitude, Offset, Phase, Period, Duration, Angle, Trigger, TriggerRepeatInterval)
Create a general periodic waveform

EnvelopePeriodic (WaveForm, Magnitude, Offset, Phase, Period, Duration, Angle, ImpulseLevel, SettleTime, FadeLevel, FadeTime, Trigger, TriggerRepeatInterval)
Create a general periodic wave form with an envelope applied

DelayedPeriodic (Pause, WaveForm, Magnitude, Offset, Phase, Period, Duration, Angle, Trigger, TriggerRepeatInterval)
Create a general periodic wave form that starts after a timed pause. This is useful for sequencing wave sensations that have a staggered onset.

APPLE INC.
EXHIBIT 1017 - PAGE 162

SweepPeriodic (StartAngle, StopAngle, Sense,WaveForm, Magnitude, Offset, Period, Phase, Duration)

Create a general periodic wave form whose direction will sweep from a StartAngle to a StopAngle.

Parameter	Range	Comment
Trigger	flags	button trigger
TriggerRepeatInterval	1 to 65 million	delay, in microseconds, before restarting the effect when triggered by a button press
Magnitude	1 to 10000	magnitude of wave
Offset	0 to 10000	offset of baseline of wave
Phase	0 to 35999	phase of start of wave
Period	1 to 65 million	period of the wave in microseconds
Duration	1 to 65 million	duration of wave (can be INFINITE)
Direction	0 to 35999	direction of wave on device
ImpulseLevel	0 to 10000	impulse kick
SettleTime	1 to 65 million	settle time, in microseconds, to reach the sustain level
FadeLevel	0 to 10000	fade level
FadeTime	1 to 65 million	fade time, in microseconds, to reach the fade level
Start	0 to 10000	magnitude of start of Ramp
End	0 to 10000	magnitude of end of Ramp
StartAngle	0 to 35999	start angle of wave
StopAngle	0 to 35999	stop angle of wave
Sense	flags	direction of sweep

APPLE INC.
EXHIBIT 1017 - PAGE 163

8.4 DYNAMIC FUNCTION WRAPPERS

DynamicRecoil(BlastDirection,BlastIntensity,DynamicMass,BlastResonance, BlastDecay)
Creates a dynamic recoil sensation as described on page 96.

DynamicImpact(ImpactDirection,ImpactIntensity,DynamicMass,Elasticity, CollisionAbsorption)
Creates a dynamic impact sensation as described on page 98.

DynamicLiquid(Density,Settle,Viscosity)
Creates a dynamic liquid sensation as described on page 100.

DynamicInertia(Inertia,Play)
Creates a dynamic inertia sensation as described on page 102.

DynamicCenterDrift(Stiffness,DriftResistance)
Creates a dynamic center drift sensation as described on page 103.

DynamicSling(BallMass,StringLength,StringCompliance,BallDamping, InterimReporting)
Creates a dynamic sling sensation as described on page 104.

DynamicPaddle(BallMass,InitialVelocity,PaddleCompliance,PaddleDamping, Gravity,InterimReporting)
Creates a dynamic paddle sensation as described on page 107.

DynamicControlLaw(Mass, KPos, KNeg, B, VelInit, PosInit, HalfCycles, Length, Drag, Gravity, InterimReporting)
Creates a dynamic control law sensation as described on page 111.

These are the parameters for the DynamicControlLaw function, which simulates dynamic behavior on a single axis. All other Dynamic functions are based on the DynamicControlLaw on one or both axes. The mapping between Dynamic function parameters and the parameters of the DynamicControlLaw follow the table. For Dynamic functions with a direction, the intensity is decomposed into two single axis initial velocities.

APPLE INC.
EXHIBIT 1017 - PAGE 164

Parameter	API Range	Comment
Mass	1 to 10000	mass of object tethered to joystick
KPos	1 to 10000	tether stiffness in positive direction
KNeg	1 to 10000	tether stiffness in negative direction
B	0 to 10000	tether damping
VelInit	-10000 to 10000	initial velocity of object
PosInit	-10000 to 10000	initial position of object
HalfCycles	0 to 10000	end effect after HalfCycles oscillations (0 = indefinite)
Length	0 to 10000	tether length (amount of free play between object and joystick)
Drag	0 to 10000	drag resistance of environment
Gravity	-10000 to 10000	gravity acting on object (negative gravity allowed!)
InterimReporting	flags	report object position and velocity back to application
Angle	0 to 35999	angle or direction of initial velocity; decomposed into 2 single axis effects

Below are the mappings between Dynamic Function parameters and the parameters of the Dynamic Conditions.

DynamicRecoil

BlastDirection: blast angle
BlastIntensity: VelInit
DynamicMass: Mass
BlastResonance: KPos (= KNeg)
BlastDecay: B

APPLE INC.
EXHIBIT 1017 - PAGE 165

DynamicImpact

ImpactDirection: impact angle
ImpactIntensity: VelInit
DynamicMass: Mass
Elasticity: KPos (= KNeg)
CollisionAbsorption: B

DynamicLiquid

Density: Mass
Settle: KPos (= KNeg)
Viscosity: B

DynamicInertia

Inertia: Mass
Play: Length

DynamicCenterDrift

Stiffness: KPos (= KNeg)
DriftResistance: Drag

DynamicSling

BallMass: Mass
StringCompliance: KPos (= KNeg)
StringLength: Length
BallDamping: Drag

DynamicPaddle

BallMass: Mass
PaddleCompliance: KPos (= KNeg)
PaddleDamping: B
InitialVelocity: VelInit

APPLE INC.
EXHIBIT 1017 - PAGE 166

8.5 Sample Code for Function Wrappers.

The following is sample code for how to create wrapper functions that simplify the creation and bookkeeping of force feedback effects. Once these wrapper functions are included, your program need only make minimal high level calls to these wrapper functions. Note that these wrapper functions still require some basic structures to have been declared and made accessible to the wrapper functions (e.g. an **IDirectInputDevice2** pointer, a **DIEffect** structure). The first section of code below declares and fills the required structures for subsequent wrapper function calls. The sample wrapper functions themselves follow and are shown in bold face.

Required Declarations For All Wrapper Functions

```
// LPDIRECTINPUTDEVICE2 lpIF2Device; // initialized previously
LONG rglDirections[2];
DWORD rgdwAxes[2];
DIENVELOPE dienvelope;
DIEFFECT dieffect;
DICONDITION dicondition;
DICONDITION rgdicondition[2];
DIDYNAMIC didynamic[2];
DIPERIODIC diperiodic;
DITEXTURE ditexture;
DIWALL diwall;
DIBARRIER dibarrier;
LPDIRECTINPUTEFFECT lpdieffect;
HRESULT hr;

dieffect.dwSize = sizeof(DIEFFECT);
dieffect.dwSamplePeriod = 0; // use default sample period
dieffect.rgdwAxes = rgdwAxes;
dieffect.rglDirection = rglDirections;
dieffect.dwGain = 10000; // maximum gain
dieffect.dwFlags = DIEFF_POLAR;
```


APPLE INC.
EXHIBIT 1017 - PAGE 167

DynamicImpact (ImpactDirection, ImpactIntensity, DynamicMass, Elasticity, CollisionAbsorption)

```
dieffect.dwTriggerButton = DIEB_NOTRIGGER;
dieffect.dwTriggerRepeatInterval = 0;
dieffect.dwDuration = INFINITE;
dieffect.lpEnvelope = NULL;
rgdwAxes[0] = DIJOFS_X;
rgdwAxes[1] = DIJOFS_Y;
dieffect.cAxes = 2;
dieffect.cbTypeSpecificParams = sizeof(DIDYNAMIC);
dieffect.lpvTypeSpecificParams = didynamic;
hr = lpIF2Device->lpVtbl->EnumEffects(lpIF2Device,
    EnumEffectCallback,&DIFORCE_DYNAMIC,DIEFT_ALL)
if (SUCCEEDED(hr)) {
   didynamic[0].dwMass = DynamicMass;
   didynamic[0].dwKPos = Elasticitiy;
   didynamic[0].dwKNeg = Elasticitiy;
   didynamic[0].dwB = CollisionAbsorption;
   didynamic[0].lVelInit = ImpactIntensity*cos(ImpactDirection);
   didynamic[0].dwHalfCycles = 0;  // infinite duration
   didynamic[0].dwLength = 0;
   didynamic[0].dwDrag = 0;
   didynamic[0].dwGravity = 0;
   didynamic[1].dwMass = DynamicMass;
   didynamic[1].dwKPos = Elasticitiy;
   didynamic[1].dwKNeg = Elasticitiy;
   didynamic[1].dwB = CollisionAbsorption;
   didynamic[1].lVelInit = ImpactIntensity*sin(ImpactDirection);
   didynamic[1].dwHalfCycles = 0;  // infinite duration
   didynamic[1].dwLength = 0;
   didynamic[1].dwDrag = 0;
   didynamic[1].dwGravity = 0;
   hr = lpIF2Device->lpVtbl->CreateEffect(lpIF2Device,
   &DIFORCE_DYNAMIC, &dieffect, &lpdieffecct, NULL);
}
if (SUCCEEDED(hr)) {
   hr = lpdieffect->lpVtbl->Download(lpdieffect);
   if (SUCCEEDED(hr)) {
      hr = lpdieffect->->lpVtbl->Start(lpdieffect,INFINITE,0);
            // play until stopped
   }
}
```


APPLE INC.
EXHIBIT 1017 - PAGE 168

Texture (Roughness, Spacing, Density, Angle)

```
dieffect.dwTriggerButton = DIEB_NOTRIGGER;
dieffect.dwTriggerRepeatInterval = 0;
dieffect.dwDuration = INFINITE;
dieffect.lpEnvelope = NULL;
rgdwAxes[0] = DIJOFS_X;
rgdwAxes[1] = DIJOFS_Y;
rglDirections[0] = Angle*100; // specified in hundredths of degrees
rglDirections[1] = 0; // must be 0 for polar coords
dieffect.cAxes = 2;
dieffect.cbTypeSpecificParams = sizeof(DITEXTURE);
dieffect.lpvTypeSpecificParams = ditexture;
hr = lpIF2Device->lpVtbl->EnumEffects(lpIF2Device,
    EnumEffectCallback,&DIFORCE_TEXTURE,DIEFT_ ALL)
if (SUCCEEDED(hr)) {
   ditexture.dwPosRoughness = Roughness;
   ditexture.dwNegRoughness = Roughness;
   ditexture.dwSpacing = Spacing;
   ditexture.dwDensity = Density;
   hr = lpIF2Device->lpVtbl->CreateEffect(lpIF2Device,
        &DIFORCE_TEXTURE, &dieffect, &lpdieffecct, NULL);
}
if (SUCCEEDED(hr)) {
   hr = lpdieffect->lpVtbl->Download(lpdieffect);
   if (SUCCEEDED(hr)) {
      hr = lpdieffect->->lpVtbl->Start(lpdieffect,INFINITE,0);
            // play until stopped
   }
}
```


APPLE INC.
EXHIBIT 1017 - PAGE 169

Wall (Hardness, Sense, Offset, Angle)

```
dieffect.dwTriggerButton = DIEB_NOTRIGGER;
dieffect.dwTriggerRepeatInterval = 0;
dieffect.dwDuration = INFINITE;
dieffect.lpEnvelope = NULL;
rgdwAxes[0] = DIJOFS_X;
rgdwAxes[1] = DIJOFS_Y;
rglDirections[0] = Angle*100; // specified in hundredths of degrees
rglDirections[1] = 0; // must be 0 for polar coords
dieffect.cAxes = 2;
dieffect.cbTypeSpecificParams = sizeof(DIWALL);
dieffect.lpvTypeSpecificParams = diwall;
hr = lpIF2Device->lpVtbl->EnumEffects(lpIF2Device,
    EnumEffectCallback,&DIFORCE_ANGLEWALL,DIEFT_ALL)
if (SUCCEEDED(hr)) {
   diwall.dwK = Hardness;
   diwall.dwSense = Sense;
   diwall.lOffset = Offset;
   hr = lpIF2Device->lpVtbl->CreateEffect(lpIF2Device,
        &DIFORCE_ANGLEWALL, &dieffect, &lpdieffecct, NULL);
}
if (SUCCEEDED(hr)) {
   hr = lpdieffect->lpVtbl->Download(lpdieffect);
   if (SUCCEEDED(hr)) {
      hr = lpdieffect->->lpVtbl->Start(lpdieffect,INFINITE,0);
            // play until stopped
   }
}
```

Barrier (PosHardness, NegHardness, Thickness, Offset, Angle)

```
dieffect.dwTriggerButton = DIEB_NOTRIGGER;
dieffect.dwTriggerRepeatInterval = 0;
dieffect.dwDuration = INFINITE;
dieffect.lpEnvelope = NULL;
rgdwAxes[0] = DIJOFS_X;
rgdwAxes[1] = DIJOFS_Y;
rglDirections[0] = Angle*100; // specified in hundredths of degrees
rglDirections[1] = 0; // must be 0 for polar coords
dieffect.cAxes = 2;
dieffect.cbTypeSpecificParams = sizeof(DIBARRIER);
dieffect.lpvTypeSpecificParams = dibarrier;
```


APPLE INC.
EXHIBIT 1017 - PAGE 170

```
hr = lpIF2Device->lpVtbl->EnumEffects(lpIF2Device,
    EnumEffectCallback,&DIFORCE_ANGLEBARRIER,DIEFT_ALL)
if (SUCCEEDED(hr)) {
   dibarrier.dwKPos = PosHardness;
   dibarrier.dwKNegs = NegHardness;
   dibarrier.dwThickness = Thickness;
   dibarrier.lOffset = Offset;
   hr = lpIF2Device->lpVtbl->CreateEffect(lpIF2Device,
        &DIFORCE_ANGLEBARRIER, &dieffect, &lpdieffecct, NULL);
}
if (SUCCEEDED(hr)) {
   hr = lpdieffect->lpVtbl->Download(lpdieffect);
   if (SUCCEEDED(hr)) {
      hr = lpdieffect->->lpVtbl->Start(lpdieffect,INFINITE,0);
            // play until stopped
   }
}
```

SineWave (Magnitude, Offset, Phase, Period, Duration, Angle, Trigger, TriggerRepeatInterval)

```
dieffect.dwTriggerButton = Trigger;
dieffect.dwTriggerRepeatInterval = TriggerRepeatInterval;
dieffect.dwDuration = Duration;
dieffect.lpEnvelope = NULL;
rgdwAxes[0] = DIJOFS_X;
rgdwAxes[1] = DIJOFS_Y;
rglDirections[0] = Angle*100; // specified in hundredths of degrees
rglDirections[1] = 0; // must be 0 for polar coords
dieffect.cAxes = 2;
dieffect.cbTypeSpecificParams = sizeof(DIPERIODIC);
dieffect.lpvTypeSpecificParams = diperiodic;
hr = lpIF2Device->lpVtbl->EnumEffects(lpIF2Device,
    EnumEffectCallback,&GUID_Sine,DIEFT_PERIODIC)
if (SUCCEEDED(hr)) {
   diperiodic.dwMagnitude = Magnitude;
   diperiodic.lOffset = Offset;
   diperiodic.dwPhase = Phase;
   diperiodic.dwPeriod = Period;
   hr = lpIF2Device->lpVtbl->CreateEffect(lpIF2Device,
        &GUID_Sine, &dieffect,
        &lpdieffecct, NULL);
```


APPLE INC.
EXHIBIT 1017 - PAGE 171

```
}
if (SUCCEEDED(hr)) {
   hr = lpdieffect->lpVtbl->Download(lpdieffect);
   if (SUCCEEDED(hr)) {
      hr = lpdieffect->->lpVtbl->Start(lpdieffect,INFINITE,0);
            // play until stopped
   }
}
```

ImpulseSquareWave (Magnitude, Offset, Phase, Period, Duration, Angle, ImpulseLevel, SettleTime, FadeLevel, FadeTime)

```
dienvelope.dwsize = sizeof(DIENVELOPE);
dienvelope.dwAttackLevel = ImpulseLevel;
dienvelope.dwAttackTime = SettleTime;
dienvelope.dwFadeLevel = FadeLevel;
dienvelope.dwFadeTime = FadeTime;
dieffect.dwTriggerButton = DIEB_NOTRIGGER;
dieffect.dwTriggerRepeatInterval = 0;
dieffect.dwDuration = Duration;
dieffect.lpEnvelope = &dienvelope;
rgdwAxes[0] = DIJOFS_X;
rgdwAxes[1] = DIJOFS_Y;
rglDirections[0] = Angle*100; // specified in hundredths of degrees
rglDirections[1] = 0; // must be 0 for polar coords
dieffect.cAxes = 2;
dieffect.cbTypeSpecificParams = sizeof(DIPERIODIC);
dieffect.lpvTypeSpecificParams = diperiodic;
hr = lpIF2Device->lpVtbl->EnumEffects(lpIF2Device,
    EnumEffectCallback,&GUID_Square,DIEFT_PERIODIC)
if (SUCCEEDED(hr)) {
   diperiodic.dwMagnitude = Magnitude;
   diperiodic.lOffset = Offset;
   diperiodic.dwPhase = Phase;
   diperiodic.dwPeriod = Period;
   hr = lpIF2Device->lpVtbl->CreateEffect(lpIF2Device,
        &GUID_Square, &dieffect, &lpdieffecct, NULL);
}
```


APPLE INC.
EXHIBIT 1017 - PAGE 172

```
if (SUCCEEDED(hr)) {
   hr = lpdieffect->lpVtbl->Download(lpdieffect);
   if (SUCCEEDED(hr)) {
      hr = lpdieffect->->lpVtbl->Start(lpdieffect,INFINITE,0);
            // play until stopped
   }
}
```

XSpring (KPos, KNeg, Offset, PosSat, NegSat, DeadBand)

```
dieffect.dwTriggerButton = DIEB_NOTRIGGER;
dieffect.dwTriggerRepeatInterval = 0;
dieffect.dwDuration = INFINITE;
dieffect.lpEnvelope = NULL;
rgdwAxes[0] = DIJOFS_X;
dieffect.cAxes = 1;
dieffect.cbTypeSpecificParams = sizeof(DICONDITION);
dieffect.lpvTypeSpecificParams = &dicondition;
hr = lpIF2Device->lpVtbl->EnumEffects(lpIF2Device,
    EnumEffectCallback,&GUID_Spring,DIEFT_CONDITION)
if (SUCCEEDED(hr)) {
   dicondition.lOffset = Offset;
   dicondition.lPositiveCoefficient = KPos;
   dicondition.lNegativeCoefficient = KNeg;
   dicondition.dwPositiveSaturation = PosSat;
   dicondition.dwNegativeSaturation = NegSat;
   dicondition.lDeadBand = DeadBand;
   hr = lpIF2Device->lpVtbl->CreateEffect(lpIF2Device,
        &GUID_Spring, &dieffect, &lpdieffecct, NULL);
}
if (SUCCEEDED(hr)) {
   hr = lpdieffect->lpVtbl->Download(lpdieffect);
   if (SUCCEEDED(hr)) {
      hr = lpdieffect->->lpVtbl->Start(lpdieffect,INFINITE,0);
            // play until stopped
   }
}
```

YSpring (KPos, KNeg, Offset, PosSat, NegSat, DeadBand)

```
Same as XSpring with rgdwAxes[0] = DIJOFS_Y;
```


APPLE INC.
EXHIBIT 1017 - PAGE 173

Spring (XKPos, XKNeg, XOffset, XPosSat, XNegSat, XDeadBand, YKPos, YKNeg, YOffset, YPosSat, YNegSat, YDeadBand)

```
dieffect.dwTriggerButton = DIEB_NOTRIGGER;
dieffect.dwTriggerRepeatInterval = 0;
dieffect.dwDuration = INFINITE;
dieffect.lpEnvelope = NULL;
rgdwAxes[0] = DIJOFS_X;
rgdwAxes[1] = DIJOFS_Y;
dieffect.cAxes = 2;
dieffect.cbTypeSpecificParams = 2*sizeof(DICONDITION);
dieffect.lpvTypeSpecificParams = rgdicondition;
hr = lpIF2Device->lpVtbl->EnumEffects(lpIF2Device,
    EnumEffectCallback,&GUID_Spring,DIEFT_CONDITION)
if (SUCCEEDED(hr)) {
   rgdicondition[0].lOffset = XOffset;
   rgdicondition[0].lPositiveCoefficient = XKPos;
   rgdicondition[0].lNegativeCoefficient = XKNeg;
   rgdicondition[0].dwPositiveSaturation = XPosSat;
   rgdicondition[0].dwNegativeSaturation = XNegSat;
   rgdicondition[0].lDeadBand = XDeadBand;
   rgdicondition[1].lOffset = YOffset;
   rgdicondition[1].lPositiveCoefficient = YKPos;
   rgdicondition[1].lNegativeCoefficient = YKNeg;
   rgdicondition[1].dwPositiveSaturation = YPosSat;
   rgdicondition[1].dwNegativeSaturation = YNegSat;
   rgdicondition[1].lDeadBand = YDeadBand;
   hr = lpIF2Device->lpVtbl->CreateEffect(lpIF2Device,
        &GUID_Spring, &dieffect, &lpdieffecct, NULL);
}
if (SUCCEEDED(hr)) {
   hr = lpdieffect->lpVtbl->Download(lpdieffect);
   if (SUCCEEDED(hr)) {
      hr = lpdieffect->->lpVtbl->Start(lpdieffect,INFINITE,0);
            // play until stopped
   }
}
```


APPLE INC.
EXHIBIT 1017 - PAGE 174

CenterSpring (Stiffness, Saturation, DeadBand)

```
dieffect.dwTriggerButton = DIEB_NOTRIGGER;
dieffect.dwTriggerRepeatInterval = 0;
dieffect.dwDuration = INFINITE;
dieffect.lpEnvelope = NULL;
rgdwAxes[0] = DIJOFS_X;
rgdwAxes[1] = DIJOFS_Y;
dieffect.cAxes = 2;
dieffect.cbTypeSpecificParams = 2*sizeof(DICONDITION);
dieffect.lpvTypeSpecificParams = rgdicondition;
hr = lpIF2Device->lpVtbl->EnumEffects(lpIF2Device,
   EnumEffectCallback,&GUID_Spring,DIEFT_CONDITION)
if (SUCCEEDED(hr)) {
   rgdicondition[0].lOffset = 0;
   rgdicondition[0].lPositiveCoefficient = Stiffness;
   rgdicondition[0].lNegativeCoefficient = Stiffness;
   rgdicondition[0].dwPositiveSaturation = Saturation;
   rgdicondition[0].dwNegativeSaturation = Saturation;
   rgdicondition[0].lDeadBand = DeadBand;
   rgdicondition[1].lOffset = 0;
   rgdicondition[1].lPositiveCoefficient = Stiffness;
   rgdicondition[1].lNegativeCoefficient = Stiffness;
   rgdicondition[1].dwPositiveSaturation = Saturation;
   rgdicondition[1].dwNegativeSaturation = Saturation;
   rgdicondition[1].lDeadBand = DeadBand;
   hr = lpIF2Device->lpVtbl->CreateEffect(lpIF2Device,
        &GUID_Spring, &dieffect, &lpdieffecct, NULL);
}
if (SUCCEEDED(hr)) {
   hr = lpdieffect->lpVtbl->Download(lpdieffect);
   if (SUCCEEDED(hr)) {
      hr = lpdieffect->->lpVtbl->Start(lpdieffect,INFINITE,0);
            // play until stopped
   }
}
```


APPLE INC.
EXHIBIT 1017 - PAGE 175

XDamper (BPos, BNeg, Offset, PosSat, NegSat, DeadBand)

```
dieffect.dwTriggerButton = DIEB_NOTRIGGER;
dieffect.dwTriggerRepeatInterval = 0;
dieffect.dwDuration = INFINITE;
dieffect.lpEnvelope = NULL;
rgdwAxes[0] = DIJOFS_X;
dieffect.cAxes = 1;
dieffect.cbTypeSpecificParams = sizeof(DICONDITION);
dieffect.lpvTypeSpecificParams = &dicondition;
hr = lpIF2Device->lpVtbl->EnumEffects(lpIF2Device,
    EnumEffectCallback,&GUID_Damper,DIEFT_CONDITION)
if (SUCCEEDED(hr)) {
   dicondition.lOffset = Offset;
   dicondition.lPositiveCoefficient = BPos;
   dicondition.lNegativeCoefficient = BNeg;
   dicondition.dwPositiveSaturation = PosSat;
   dicondition.dwNegativeSaturation = NegSat;
   dicondition.lDeadBand = DeadBand;
   hr = lpIF2Device->lpVtbl->CreateEffect(lpIF2Device,
        &GUID_Damper, &dieffect, &lpdieffecct, NULL);
}
if (SUCCEEDED(hr)) {
   hr = lpdieffect->lpVtbl->Download(lpdieffect);
   if (SUCCEEDED(hr)) {
      hr = lpdieffect->->lpVtbl->Start(lpdieffect,INFINITE,0);
            // play until stopped
   }
}
```

YDamper (BPos, BNeg, Offset, PosSat, NegSat, DeadBand)

Same as XDamper with rgdwAxes[0] = DIJOFS_Y;

APPLE INC.
EXHIBIT 1017 - PAGE 176

Damper (XBPos, XBNeg, XOffset, XPosSat, XNegSat, XDeadBand, YBPos, YBNeg, YOffset, YPosSat, YNegSat, YDeadBand)

```
dieffect.dwTriggerButton = DIEB_NOTRIGGER;
dieffect.dwTriggerRepeatInterval = 0;
dieffect.dwDuration = INFINITE;
dieffect.lpEnvelope = NULL;
rgdwAxes[0] = DIJOFS_X;
rgdwAxes[1] = DIJOFS_Y;
dieffect.cAxes = 2;
dieffect.cbTypeSpecificParams = 2*sizeof(DICONDITION);
dieffect.lpvTypeSpecificParams = rgdicondition;
hr = lpIF2Device->lpVtbl->EnumEffects(lpIF2Device,
    EnumEffectCallback,&GUID_Damper,DIEFT_CONDITION)
if (SUCCEEDED(hr)) {
   rgdicondition[0].lOffset = XOffset;
   rgdicondition[0].lPositiveCoefficient = XBPos;
   rgdicondition[0].lNegativeCoefficient = XBNeg;
   rgdicondition[0].dwPositiveSaturation = XPosSat;
   rgdicondition[0].dwNegativeSaturation = XNegSat;
   rgdicondition[0].lDeadBand = XDeadBand;
   rgdicondition[1].lOffset = YOffset;
   rgdicondition[1].lPositiveCoefficient = YBPos;
   rgdicondition[1].lNegativeCoefficient = YBNeg;
   rgdicondition[1].dwPositiveSaturation = YPosSat;
   rgdicondition[1].dwNegativeSaturation = YNegSat;
   rgdicondition[1].lDeadBand = YDeadBand;
   hr = lpIF2Device->lpVtbl->CreateEffect(lpIF2Device,
        &GUID_Damper, &dieffect, &lpdieffecct, NULL);
}
if (SUCCEEDED(hr)) {
   hr = lpdieffect->lpVtbl->Download(lpdieffect);
   if (SUCCEEDED(hr)) {
      hr = lpdieffect->->lpVtbl->Start(lpdieffect,INFINITE,0);
            // play until stopped
   }
}
```


APPLE INC.
EXHIBIT 1017 - PAGE 177

XInertia (Mass, Saturation)

```
dieffect.dwTriggerButton = DIEB_NOTRIGGER;
dieffect.dwTriggerRepeatInterval = 0;
dieffect.dwDuration = INFINITE;
dieffect.lpEnvelope = NULL;
rgdwAxes[0] = DIJOFS_X;
dieffect.cAxes = 1;
dieffect.cbTypeSpecificParams = sizeof(DICONDITION);
dieffect.lpvTypeSpecificParams = &dicondition;
hr = lpIF2Device->lpVtbl->EnumEffects(lpIF2Device,
    EnumEffectCallback,&GUID_Inertia,DIEFT_CONDITION)
if (SUCCEEDED(hr)) {
   dicondition.lOffset = 0;
   dicondition.lPositiveCoefficient = Mass;
   dicondition.lNegativeCoefficient = Mass;
   dicondition.dwPositiveSaturation = Saturation;
   dicondition.dwNegativeSaturation = Saturation;
   dicondition.lDeadBand = 0;
   hr = lpIF2Device->lpVtbl->CreateEffect(lpIF2Device,
        &GUID_Inertia, &dieffect, &lpdieffecct, NULL);
}
if (SUCCEEDED(hr)) {
   hr = lpdieffect->lpVtbl->Download(lpdieffect);
   if (SUCCEEDED(hr)) {
      hr = lpdieffect->->lpVtbl->Start(lpdieffect,INFINITE,0);
            // play until stopped
   }
}
```

The example I-FORCE 2.0 wrapper functions shown above should provide guidelines for writing or modifying your own wrapper functions for other I-FORCE 2.0 effects.

APPLE INC.
EXHIBIT 1017 - PAGE 178

9. The I-FORCE Studio Toolset for DirectX

9.1 Overview

The I-FORCE Studio Toolset is an interactive graphical environment that enables programmers to design feel sensations rapidly and efficiently. The **I-FORCE Studio** toolset allows Conditions, Waves, and Dynamics to be defined through intuitive graphical metaphors that convey the physical meaning of each parameter involved. As the parameters are manipulated, sensations can be felt in real-time, allowing for an iterative design process that fine-tunes the feel to your exact need. Once the appropriate sensation is achieved, the I-FORCE Studio toolset saves the parameters as a resource which is automatically loaded and executed by your application through DirectX. In other words, the I-FORCE Studio toolset takes care of the entire force feedback development process for the programmer, from design to implementation. With these tools, force feedback programming becomes a fast, simple, and fun process.

APPLE INC.
EXHIBIT 1017 - PAGE 181

9.2 Introduction

The challenge of programming for force feedback is not the act of coding, it is the act of *designing* feel sensations that appropriately match your gaming events. Designing *feels* requires a creative and interactive process where parameters are defined, experienced, and modified until the sensations are just right. For Conditions, this interactive process might involve setting the *stiffness* of springs, sizing the *deadband*, manipulating the *offset*, and tuning the *saturation* values. For Waves, this might involve picking the wave source (sine, square, triangle...), setting the *magnitude*, *frequency*, and *duration*, of the signal and then tuning the *envelope* parameters. For a Dynamic, this might involve setting the *dynamic mass*, and then tuning the *resonance* and *decay* parameters. With so many parameters to choose from, there needs to be a fast, simple, and interactive means for sensation design. To solve this need, Immersion Corporation has developed the **I-FORCE Studio** toolset, a graphical environment for rapidly setting physical parameters, feeling sensations, and then saving them as resources which will automatically call DirectX in your application.

The I-FORCE Studio toolset has provides the following features and benefits that make it the ideal environment for all your force feedback programming needs:

- Allows for **interactive real-time sensation design** of Conditions, Waves, and Dynamics where parameters can be defined and experienced through a rapid iterative process.
- Provides **intuitive graphical metaphors** that will enhance your understanding of the physical parameters related to each sensation type, thereby speeding the iterative design process
- Provides **valuable file-management tools** so feel sensations to be saved, copied, modified, and combined – thereby letting you establish your own library of favorite feel sensations.
- Once sensations are defined, the I-FORCE Tools **store the parameters as Resources**. By linking the I-FORCE Tools DLL into your application, the Resources are automatically loaded and executed through **DirectX** in your application. In other words, the tools take care of most of sensation generation process from design to coding.

APPLE INC.
EXHIBIT 1017 - PAGE 182

9.3 The Development Environment

As shown in Figure 9-1, the I-FORCE Studio toolset development environment has three primary work areas. The column on the left is the **Sensation Pallet**. The column on the right is the **Button Trigger Pallet**. The Window in the center is the **Design Space**. Sensations are created in the Design Space and can be saved and loaded into that space using all standard file handling features.

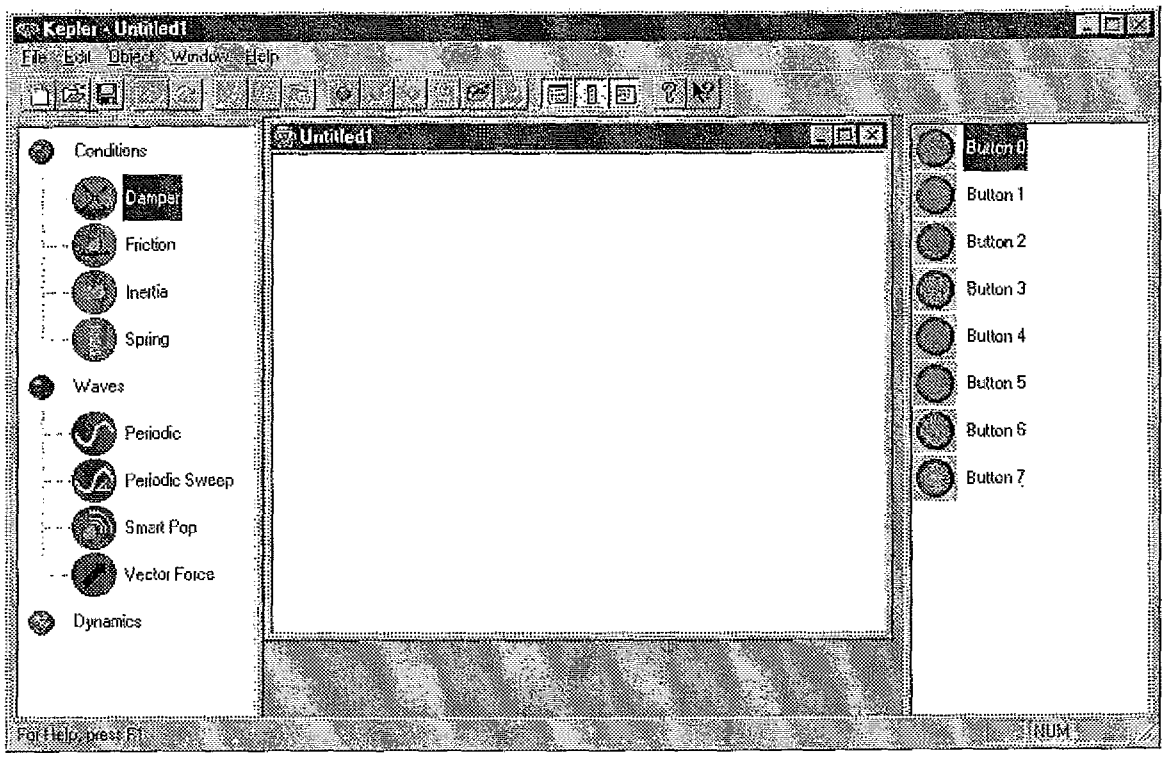


Figure 9-1

To Create a new Effect, a sensation type is chosen from the Sensation Pallet. This Pallet is shown in an expandable tree format. In the figure above, you can see that the root of the three includes the three classes of force feedback sensations, Conditions, Waves, and Dynamics. Users can also define their own headings, for example "My Favorites" as a place to store sensations with desirable preset parameters.

APPLE INC.
EXHIBIT 1017 - PAGE 183

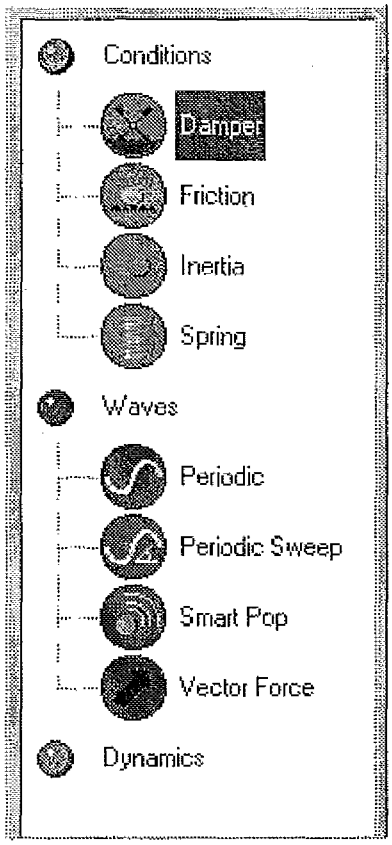


Figure 9-2

In Figure 9-2 , you can see that Conditions and Waves are shown in expanded view while Dynamics is shown compressed. When in expanded view, the programmer will get a listing of all the sensation *types* that are supported by the hardware being programmed for. For example, when programming for a hardware supporting the latest I-FORCE 2.0 processing core, you will see a very complete list of sensation types. If programming for an older I-FORCE 1.x processing core, some sensation types will not be listed. The I-FORCE Studio toolset can determine exactly what sensations are supported by a given device connected to the PC by using the *effect enumeration* process.

Once a sensation type is chosen from the Sensation Pallet, the sensation type is added to the Design Space (the icon for that sensation will now be shown within the Design Space window). That icon can now be opened in order to set the parameters for the given sensation type using graphical development tools (as described in the next section). Once

APPLE INC.
EXHIBIT 1017 - PAGE 184

the parameters are set for the given effect, the sensation can be saved as a resource file. Using this process, you can create a diverse library of feel sensations as resource files. Also, Immersion Corporation provides their own library of sample resource files with interesting pre-defined sensations that you can work from.

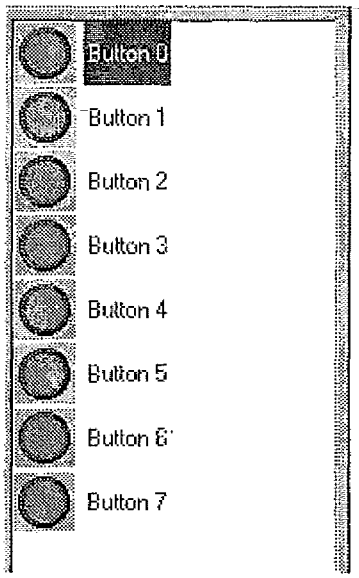


Figure 9-3

As shown in Figure 9-3, the Right side of the development environment is the Trigger Button Pallet. This is used for testing sensations that are going to be defined as button reflexes. For example, say you want a sensation to be a combination of a Square Wave and a Sine Wave that triggers when Button #2 is pulled. You would create the Square Wave by choosing the Periodic type from the Sensation Pallet and define the parameters appropriate for that wave. Then you would Create a Sine Wave by choosing another Periodic type from the Sensation Pallet and again define the parameters for that wave. At this point you will have two Periodic Icons in the Design Space window. To test the trigger, you can just drag and drop these icons into the Button 2 icon. Now, when you press Button 2 on your joystick, you will feel the reflex sensation. The process is fast, simple, and versatile. When you get the sensation to be exactly as desired, you can save it as a resource file which will be automatically loaded and executed through **DirectX** in your application.

APPLE INC.
EXHIBIT 1017 - PAGE 185

9.4 The Sensation Design Process

When a sensation type is selected, the icon expands into a graphical environment for setting and testing the physical parameters associated with the given sensation. For example, a Spring sensation type is selected from the Condition pallet. Within the Spring Window, you can set all the parameters associated with the spring sensation. For example, you can set *the positive stiffness*, *negative stiffness*, *positive saturation*, *negative saturation*, *offset*, and *deadband*. You can also choose the *axis* or *direction* of the spring. As parameters are set, they are shown in an intuitive graphical format.

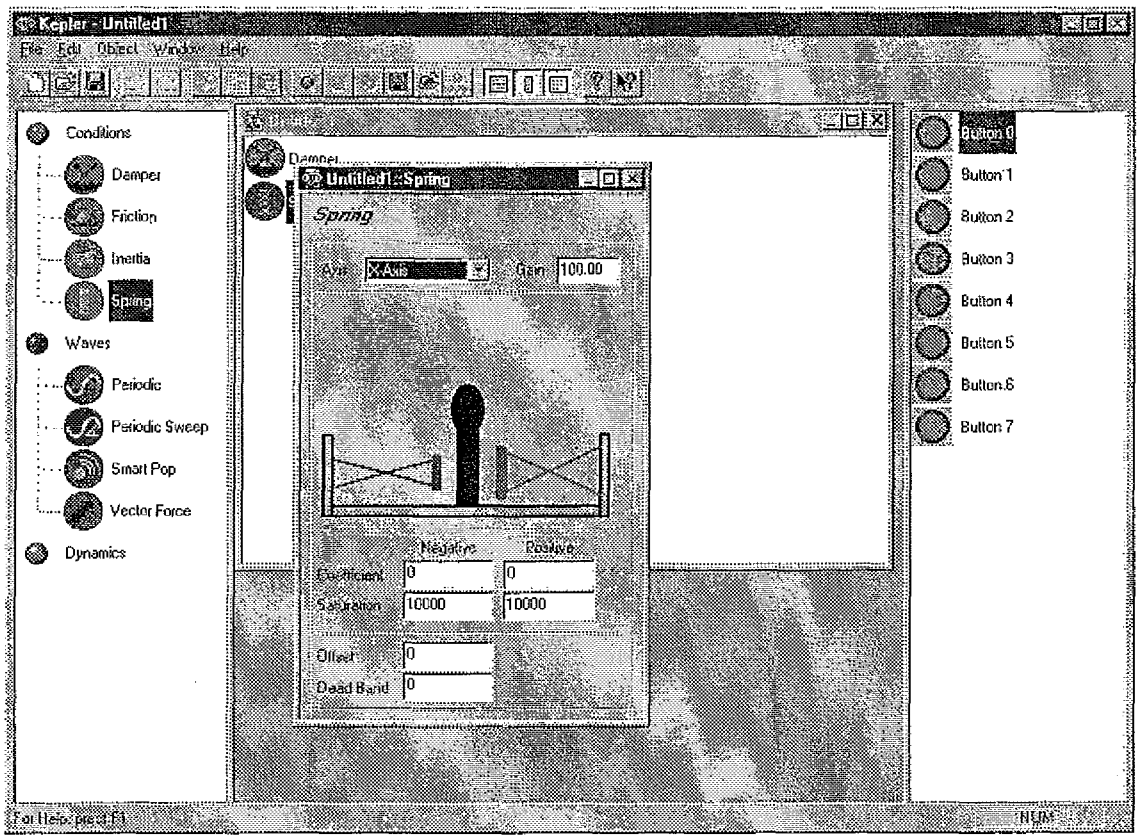


Figure 9-4

Another Example is the Periodic Sensation. As shown in Figure 9-5, the Periodic Window allow allows a programmer to choose from among the set of standard *waveform* signal

APPLE INC.
EXHIBIT 1017 - PAGE 186

sources. In this case the programmer chose square-wave. In addition, the programmer can set the *Magnitude*, *Period*, *Duration*, and *Direction* of the signal. The user can also activate an envelope, but in this case the envelope option is not selected. Also, the user can assign a *button trigger* and a *trigger repeat rate*, but in the shown example below the option is not being used.

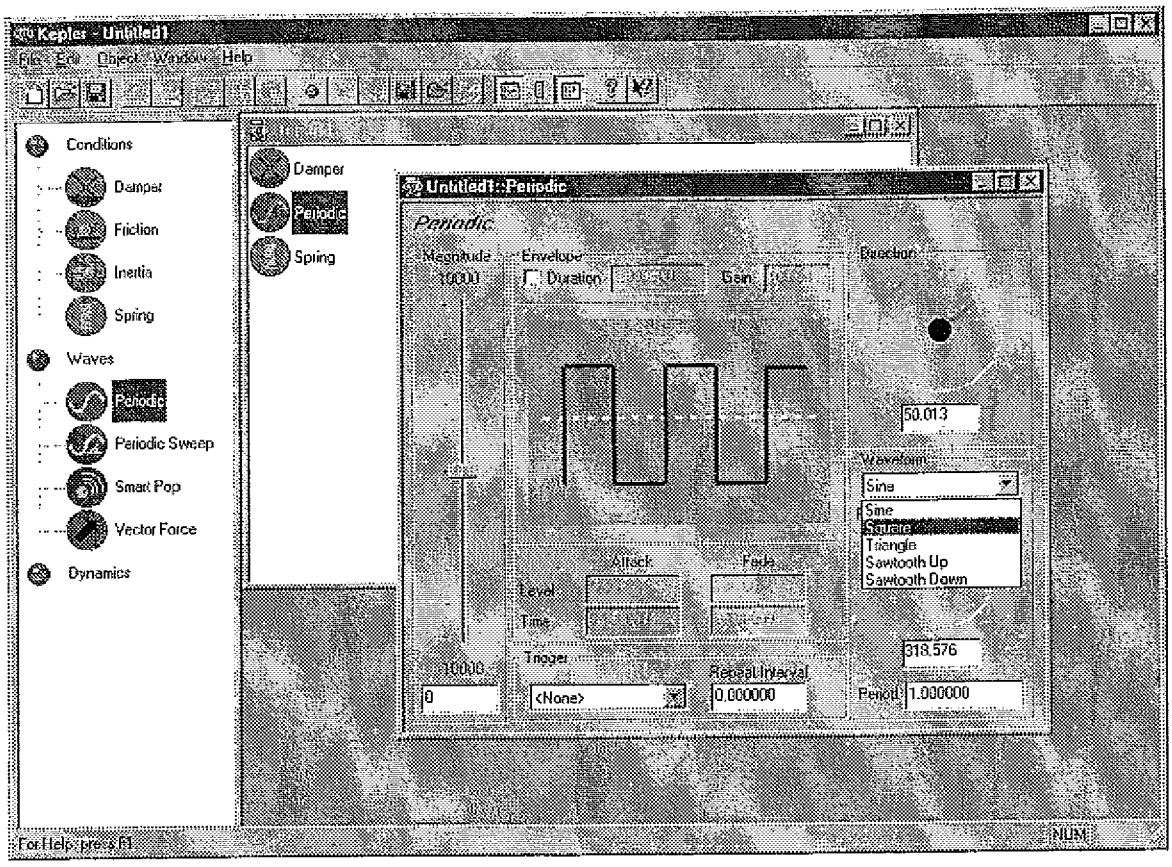


Figure 9-5

For our final example of the speed and ease by which sensations can be generated using the I-FORCE Studio toolset, we will show how a programmer might design an Advanced Periodic sensation, one of the more complex effect types presented in this text. In Figure 9-6, the programmer has chosen an Periodic Sweep sensation which is very similar to a standard Periodic but where the direction sweeps between a start and end orientation. As you can see, two dials are used to by the programmer to define the starting and ending orientations for the Periodic Sweep. In this example, the user has chosen a *sine wave* as

APPLE INC.
EXHIBIT 1017 - PAGE 187

the signal source. Also, the user can assign the *Magnitude*, *Period*, *Duration*, and *Phase* of the signal. Also, in this example the user has activated the *Envelope* feature and has created an Impulse Wave Shape using Attack and Fade parameters. These parameters can me entered as numbers or by dragging the graphical outline of the waveform with a cursor.

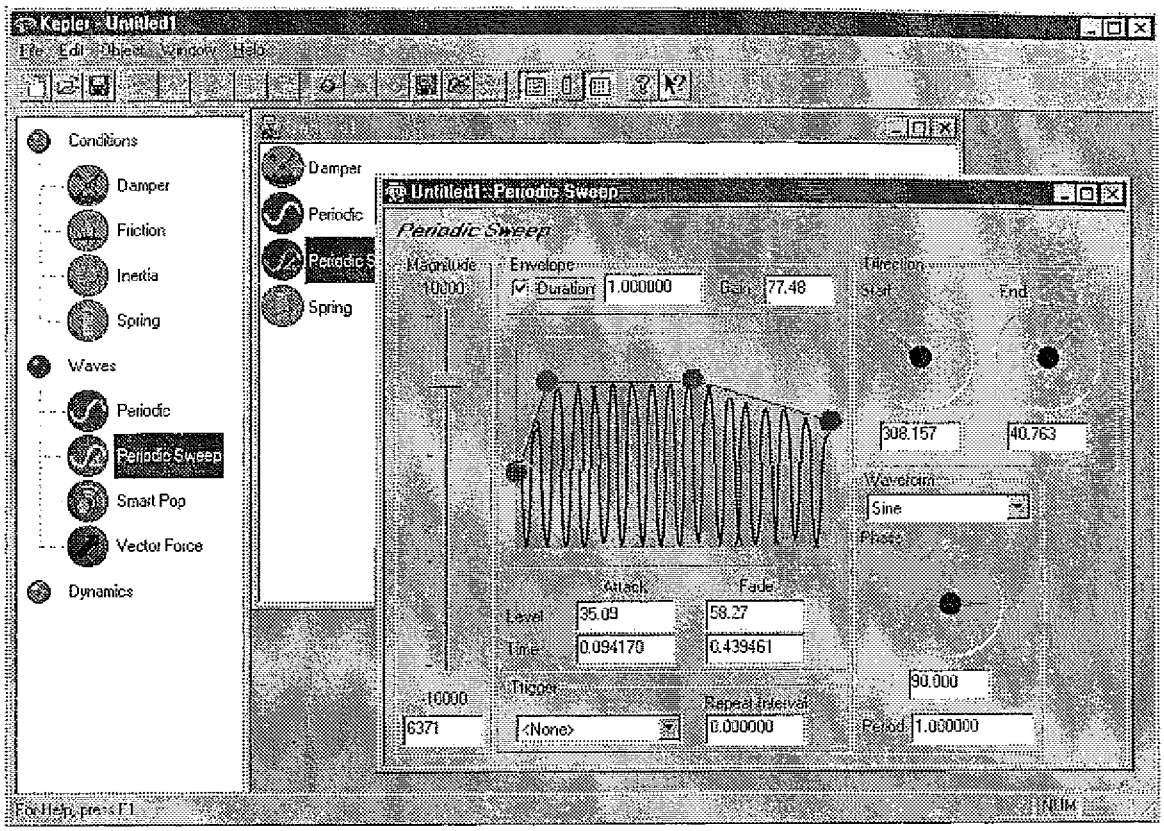


Figure 9-6

Figure 9-7 and Figure 9-8 are two additional screenshots of the I-FORCE Tools which show some of the other effects you can create.

APPLE INC.
EXHIBIT 1017 - PAGE 188

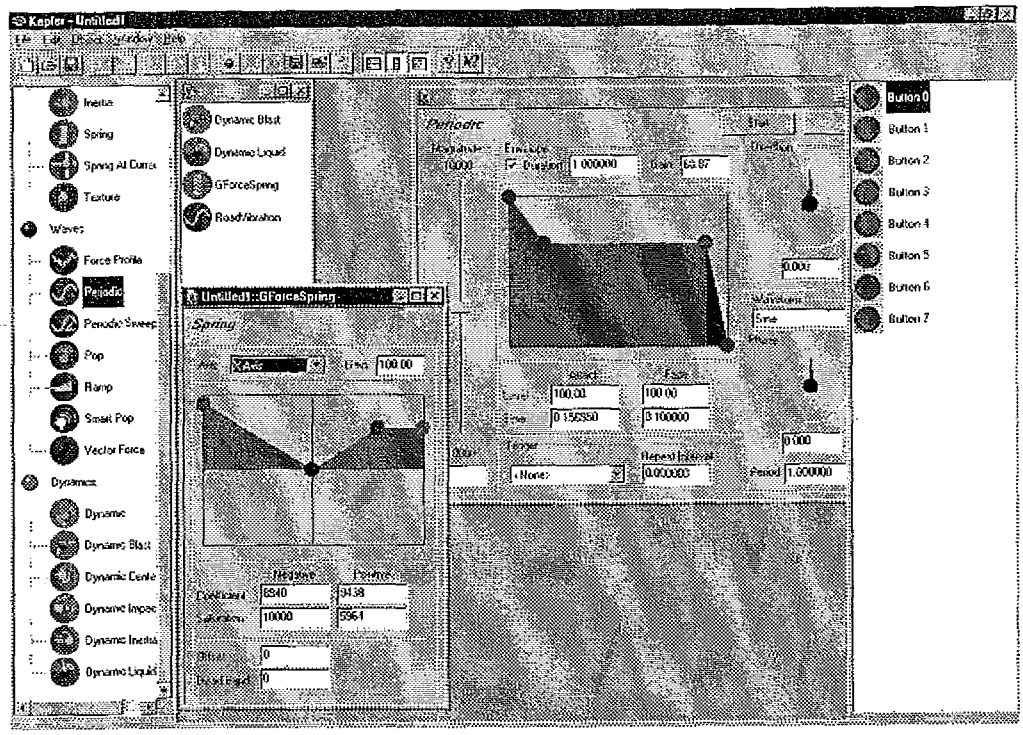


Figure 9-7

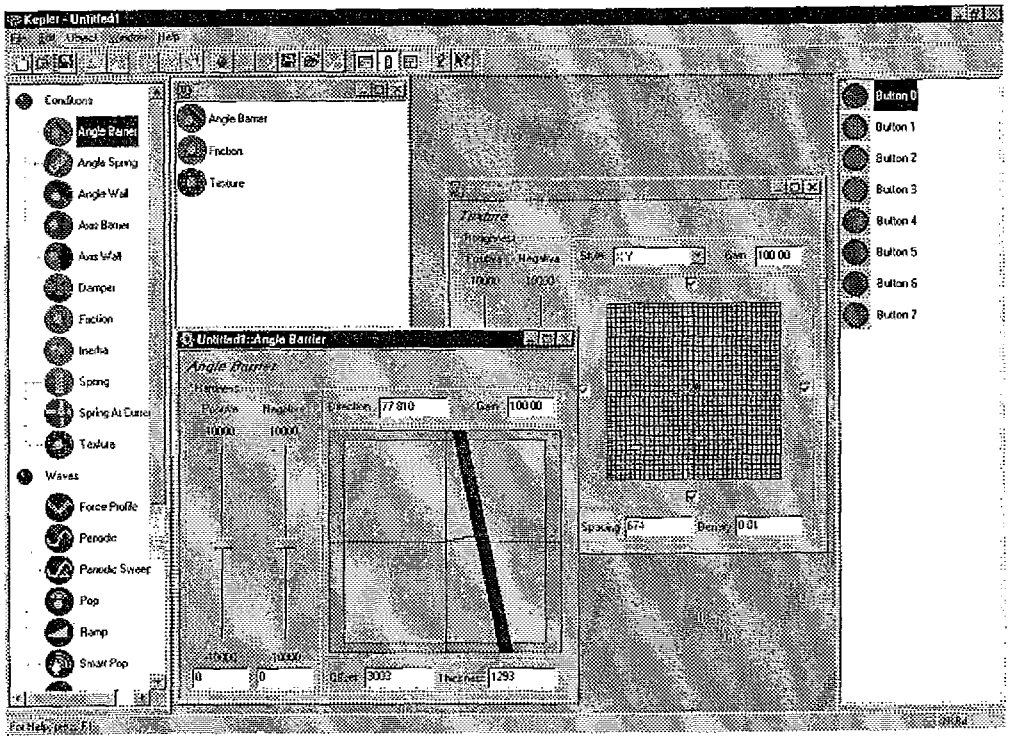


Figure 9-8

APPLE INC.
EXHIBIT 1017 - PAGE 189

9.5 The Simplified Programming Process

Once you have designed your sensations using the graphical tools as described above, you can save all the Effect definitions as a Resource file and link the I-FORCE Tools DLL to your application. One function call to the DLL at the start of your program will load the resource at the start of your application and automatically instantiate all the necessary DirectX structs, and fill them with the appropriate values, offsets, and masks. A second call to the DLL will actually execute your specified effect. You need not worry about creating a DirectInput force feedback device, enumerating and acquiring the device, choosing polar or Cartesian coordinates, determining axis masks, looking up parameter ranges, or any other of the tedious but necessary tasks required by DirectX force feedback programming. Instead, two simple calls to the I-FORCE Tools DLL takes care of all that for you.

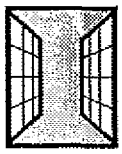
Should you want to alter some parameters of one of your designed effects directly in your application, the DLL will provide you with a pointer to the DirectX effect structs. You can then directly change whatever parameters you wish. For example, if you design a cool DynamicImpact from the North direction, you may wish to use this effect again later in your application but want it to come from the East. You can directly change the rglDirection member of the DIEFFECT struct and then execute it again.

In summary, the I-FORCE Studio toolset provides developers with a powerful and intuitive tool for creating and testing force effects. Once designed, the force effects can be easily loaded and executed in your application with two simple functions from the I-FORCE Tools DLL, or they can directly modified from within your application.

APPLE INC.
EXHIBIT 1017 - PAGE 190

Immersion Corporation is the developer of **I-FORCE** force feedback technology. This includes the I-FORCE processing core and hardware architecture licensed by most makers of quality force feedback products. Immersion Corporation is also the developer of the **I-FORCE Studio** toolset, the graphical environment for rapid feel sensation design. For technical support on I-FORCE Hardware or I-FORCE Software, contact Immersion Corporation at:

Immersion Corporation
2158 Paragon Drive
San Jose, CA 95131



Toll-free (800) 893-1160
Telephone (408) 467-1900
FAX (408) 467-1901

General e-mail info@immerse.com
I-Force e-mail force@immerse.com

World Wide Web: http://www.immerse.com
http://www.force-feedback.com

FTP: ftp://ftp.immerse.com/pub/users/immerse

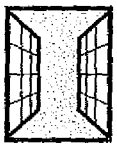
Thank you for your interest in I-Force and Force Feedback.

Embrace the Resistance....

Force Feedback is the latest technological innovation to take the computer entertainment world by storm. Simply put, force feedback adds compelling FEEL sensations to gaming environments, allowing players to encounter *realistic physical resistance* when piloting a ship, blasting a monster, getting checked into the boards, or taking down a linebacker. Force Feedback is achieved by adding *motors* to computer peripherals such as joysticks, steering wheels, or flight yokes. Under the guidance of a skilled programmer, these motors bring the peripherals to life, letting them push back in the player's hand, thereby adding an interactive physical realism to gaming that has never before been possible.

This book is an introduction to the hardware and software issues of force feedback technology. The text is structured as a reference document written for professionals involved in all aspects of computer game development from conceptual design to hard-core coding. The text will expose you to the limitless potential of force feedback, giving you guidelines on how to use "feel" effectively within your gaming applications, and encouraging you to invent your own creative implementations of "feel" never before imagined.

For the latest updates on force feedback gaming technologies from hardware makers and software developers, visit the web site WWW.FORCE-FEEDBACK.COM



Immersion Corporation
The Force Feedback Company