Powercursor
a toolkit for designing active cursor behaviours

Hendrix, K.J.J.

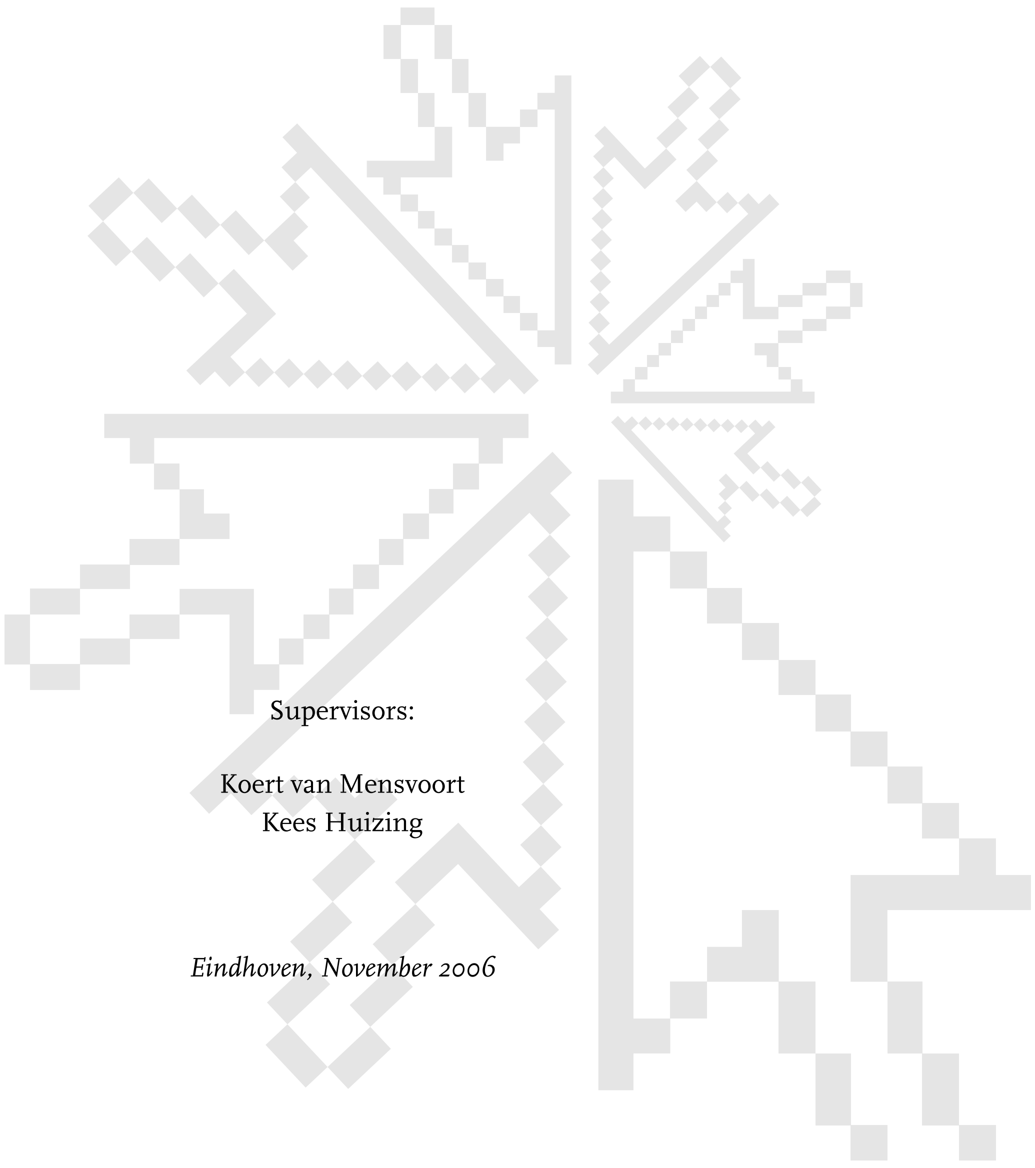
Award date:
2006

TECHNISCHE UNIVERSITEIT EINDHOVEN
Department of Mathematics and Computer Science

PowerCursor:

a Toolkit for Designing Active Cursor Behaviours

K.J.J. Hendrix



Supervisors:

Koert van Mensvoort
Kees Huizing

Eindhoven, November 2006

PowerCursor:

a Toolkit for Designing Active Cursor Behaviours

K.J.J. Hendrix

Afstudeerverslag ter afronding van de studie Technische Informatica aan de Technische Universiteit Eindhoven, 2006.

Afstudeerdocent: Dr. C. Huizing, faculteit Wiskunde & Informatica, TU/e.

Afstudeerbegeleider: Ir. Mfa K. van Mensvoort, faculteit Industrial Design, TU/e.

Abstract

Force feedback can add tactile sensations to computer interfaces, but it requires special haptic equipment which is not part of the standard desktop computer setup. It has been proposed that the effect of force feedback can also be visually simulated using cursor displacements. This document describes PowerCursor, a toolkit for prototyping new mouse behaviours using cursor displacements. PowerCursor has been implemented in Macromedia Flash MX 2004.

The toolkit contains different objects such as hills, gutters and rough surfaces. These objects are made 'tactile' using tiny displacements on a mock cursor, visually simulating force feedback. Several tactile objects are available as ready-made components that can be imported into any Flash document; it is also possible to create tactile objects using existing graphics. The toolkit has been built to integrate into the Flash design environment, and to allow non-programmers to easily add the toolkit's complex functionality to their own designs.

Abstract 4

1 INTRODUCTION 7

2 BACKGROUND 8

2.1 Computers and human senses 8

2.2 Simulated haptic feedback 8

2.3 Enriching mouse interaction 10

2.3.1 Material expression 10
2.3.2 Assisted navigation 10
2.3.3 Mixed initiative 11

3 THE POWERCURSOR TOOLKIT 12

3.1 Terminology 12

3.2 Platform 12

3.2.1 Macromedia Flash MX 2004 12
3.2.2 ActionScript 2.0 13
3.2.3 Version 2 Components versus MovieClips 13

3.3 Design considerations 14

3.3.1 Accessible for interface designers 14
3.3.2 Accommodating for non-programmers 15
3.3.3 Flexible generic component design 16

3.4 Operation 17

3.4.1 Engine and Cursors 17
3.4.2 Objects and Modifiers 20
3.4.3 Dimmers and Walls 20

3.5 Architecture 21

3.5.1 Classes 21
3.5.2 Events 24
3.5.3 Runtime sharing 25
3.5.4 Extension to the system cursor 25

3.6 Graphics 26

3.6.1 Requirements 26
3.6.2 Implementation 26
3.6.3 Shading 27

3.7 Performance 28

4 CONCLUSION 29

4.1 Discussion ... 29
4.2 Suggestions for further work ... 30

5 References ... 31

Appendix A: Components ... 32

A.1 General components ... 32
A.1.1 PC_Engine ... 32
A.1.2 PC_Watcher ... 32
A.2 Cursors ... 33
A.2.1 Cursors in general ... 33
A.2.2 PC_ArrowCursor ... 33
A.2.3 PC_BallCursor, PC_FreeBallCursor ... 33
A.2.4 PC_CursorMaker ... 34
A.2.5 PC_ForceSensor ... 34
A.3 Ready-made Objects ... 34
A.3.1 PC_RoundReadyMadeObject ... 34
A.3.2 PC_Dimmer ... 36
A.3.3 PC_Gutter, PC_PushGutter ... 37
A.3.4 PC_Rough ... 38
A.3.5 PC_Slick ... 38
A.3.6 PC_Slope ... 39
A.3.7 PC_Wall ... 39
A.3.8 PC_Wedge ... 40
A.4 Modifiers ... 40
A.4.1 PC_ButtonMaker ... 40
A.4.2 PC_DimmerMaker ... 41
A.4.3 PC_DragMaker ... 42
A.4.4 PC_EventDispatcher ... 42
A.4.5 PC_HillMaker ... 42
A.4.6 PC_RoughMaker ... 42
A.4.7 PC_SlickMaker ... 43
A.4.8 PC_SlopeMaker ... 44
A.4.9 PC_Synchronizer ... 44
A.4.10 PC_WallMaker ... 44
A.5 Events ... 44

1 Introduction

The document describes PowerCursor, a toolkit for prototyping new cursor behaviours that use active cursor displacements. Using these displacements, the tactile sensation of hills, holes, roughness, and other shapes and surfaces can be visually simulated. The design and implementation of the PowerCursor toolkit is the graduation project of the author.

The mouse plays an important role in human-computer interaction. The majority of all graphical user interfaces on desktop computers involve a mouse-controlled cursor. As such, the mouse and its cursor are used for a wealth of different actions and manipulations, including navigating, browsing, scrolling, selecting and sorting. Because the mouse and cursor are so frequently used in operating a desktop computer, a small improvement in the cursor's behaviour may imply a leap forward in improving computer human-interaction. We propose using cursor displacements to add a new dimension to mouse interaction, as well as convey a sense of touch through visually simulated force feedback.

The idea of using cursors displacements to simulate haptic feedback was proposed several years ago by K. v. Mensvoort [1,2]. Although the concept has been verified, ; further exploration of the possibilities of this new mouse interaction concept is required. For this reason, this prototyping toolkit has been built. It will allow interface designers to experiment with cursor displacements in their designs, hopefully leading to a broad variety of ways and styles in which cursor displacements can be applied.

Chapter 2 introduces some background information on human senses in computer interaction, and describes how we aim to enrich this interaction. In chapter 3 the design and implementation process of the PowerCursor toolkit is detailed, focusing on the decisions and considerations involved in that process. Chapter 4 concludes this report with a discussion of the project and recommendations for further work. Finally, appendix A details all the components in the PowerCursor toolkit.

2 Background

2.1 Computers and human senses

From a sensory point of view, computers are very limited machines. Almost all information users receive from a computer is conveyed via 2-dimensional visual information, occasionally accompanied by a short beep or chord from the speakers. The items on a computer's desktop do not have a texture, smell, weight, or taste, because the computer has no way to convey these sensory experiences to the user. Similarly, the cursor controlled by the mouse has no apparent mass, grip, or any other physical property.

Although the insubstantiality of the virtual environment of computers has certain advantages, communicating over a larger part of the human sensory bandwidth may improve the interaction between the user and his or her machine in a number of ways. We aim to enrich this interaction using (simulated) haptic feedback. The ability to touch, feel, prod or squeeze things can relay information on (virtual) material properties in an very immediate, natural and intuitive manner, and (simulated) haptic feedback may well be able to communicate other information in a more efficient or appropriate manner as well.

2.2 Simulated haptic feedback

As mentioned above, one possible way to enrich computer interaction is to have the computer interact on more senses than just sight and hearing – in other words, interact with the user through more media than just its screen and speakers. One method of doing this is by applying force to input devices, a technique commonly known as 'force-feedback'. With non-force-feedback input devices, the user is the only entity controlling the position of such a device. A force-feedback unit gives the computer (some) control over its movements as well. Such devices can give haptic feedback to the user who is holding the input device. This is illustrated for the mouse in Figure 1 and Figure 2.

$$\begin{tikzcd}
\textit{user} \arrow[r] & \textit{mouse position} \arrow[r] & \textit{cursor position} \arrow[r] & \textit{computer}
\end{tikzcd}$$

Figure 1: Normal mouse communication (one-way)

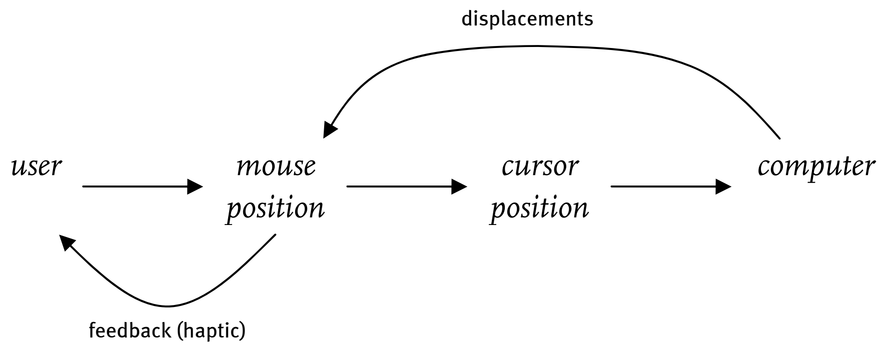


Figure 2: Force-Feedback mouse feedback loop

Force-feedback technology has been applied in various input devices, including the mouse, joystick, and steering wheel. Unfortunately, virtually no desktop computers feature a force-feedback mouse. This scarcity might be caused by their relatively high price tag or by the lack of software applications for these devices. And the software is scarce because hardly anybody has a force-feedback mouse at home. Due to this vicious circle, the force-feedback mouse never caught on and is virtually unknown to the public today. Other force-feedback input devices such as joysticks and steering wheels have a more obvious application in the entertainment industry and have moved into the mainstream of computer input devices.

To offer richer mouse interaction on standard desktop PCs without resorting to special force-feedback devices, haptic feedback may be visually simulated [1]. In such a simulation, the physical displacement of the input device is replaced by small displacements of the mouse cursor on the screen (see Figure 3). The resulting visual feedback is very similar to actual force-feedback on the mouse, although it lacks the actual haptic dimension. While these 2-dimensional cursor displacements can only be seen and not be felt, they can still offer an experience similar to haptic feedback. Indeed, for some tasks, visually simulated force feedback works better than haptic force feedback [2].

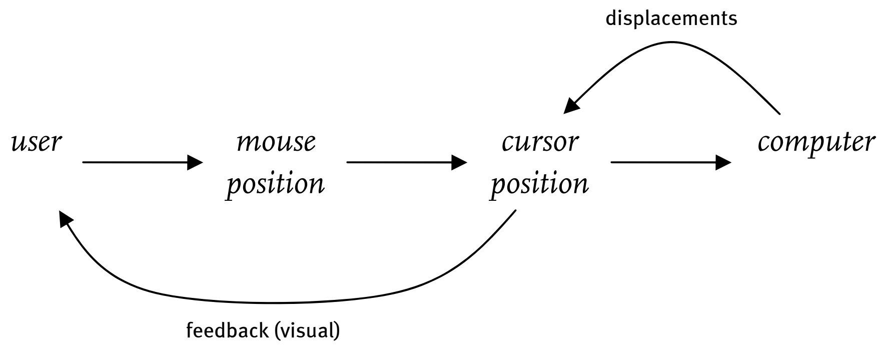


Figure 3: PowerCursor mouse feedback loop

2.3 Enriching mouse interaction

The idea of using cursor displacements was born, as described in the previous section, as a visual simulation of force-feedback technology. However, cursor displacements have more applications; several ways of using them to improve mouse interaction are described below.

2.3.1 Material expression

Using cursor displacements as visually simulated force feedback, the mouse experience can be enriched by adding haptic properties to interface elements. Cursor displacements that help or hinder mouse movement can suggest a slanted or textured desktop surface. Many common interface elements – buttons, scrollbars and the like – are already shaped or shaded to suggest tactile forms. Using cursor displacements, these shapes could actually be experienced.

For example, it is common for buttons in a user interface to have a protruding 'two-and-a-half dimensional' (2½D) shape, which makes the button stand out from its surrounding surface and gives it a 'pushable' appearance. Tiny cursor displacements around the edges of such a button might make it hard for the cursor to 'climb onto' the button and easy to 'slide off'. In this manner, its 2½D shape might be 'felt'. Many other shapes and textures could be realised in a similar fashion: draggable objects might have a rough texture, radio buttons might feel like small holes, and empty desktop space might be slick and smooth.

2.3.2 Assisted navigation

Apart from adding shape and texture to interface objects, cursor displacements could also be used to aid the user in her mouse movements. Based on the user's mouse actions, the computer may deduce the user's intentions, and assist as necessary by nudging the mouse cursor toward or away from interface objects.

For example, when the user is dragging a scrollbar to scroll through a document, it is reasonable to assume that while the user keeps on pressing the mouse button, he or she wants to keep scrolling. However, if the cursor is (accidentally) moved too far from the scrollbar, it loses its hold and the drag is released. Tiny cursor displacements towards the scrollbar could prevent such accidental drifting. When navigating though a pull-down menu, similar displacements could help users remain within the menu, select items and navigate into submenus [3].

The cursor displacements used to assist navigation can also be translated to haptic sensations. Drawing the cursor towards a scrollbar might suggest a gutter-like shape around the bar, while pushing the cursor away from a 'dangerous' button might be experienced as a hill-shape around the button.

2.3.3 Mixed initiative

Cursor displacements need not only be used in direct reaction to users' actions. Apart from enhancing the user's ability to move the mouse, giving the computer some initiative in moving the cursor can be another way of assisting the user in her mouse interactions. A system that combines the user's direct manipulation with computer-initiated service is called a mixed-initiative user interface [4]. PowerCursor offers some possibilities for creating such an interface.

By moving the cursor, the computer could lead the user to the next point where it wants his or her input, or suggest particular options by moving the cursor over them. In situations where the user has to make a choice (such as Yes/No/Cancel dialog boxes), cursor displacements could steer the cursor toward the recommended option. Such 'hints' or 'suggestions' could be programmed into interface designs beforehand, but they might also be derived from user's previous actions or preferences: the most used option, the most recently used option, or the safest option could all be valid suggestions. Perhaps recommending 'the option most used by people like you in situations like this' might one day be possible.

As another example, so-called 'wizards' are often used to guide users through a complex process such as configuring a network connection or installing a game. In such wizards, the user is often sequentially presented with a number of choices or input fields. If there is only one choice or input field on screen, a PowerCursor application could guide the cursor towards it, and in the case of more input options the cursor could be guided to a suggested or most likely option. By remembering previous uses of the same wizard (or similar ones), a PowerCursor system could help the user by guiding the mouse towards the choice he or she always makes, such as only installing network printers or always installing games in the same directory.

Computer-initiated mouse movement might be especially useful to assist users when they try to do things that require some other action elsewhere first; for instance, trying to log in without entering a name or password, or trying to continue an installation before entering a necessary serial number. In such cases, the computer could point the cursor to the required input field, signalling the user that it requires some action there before continuing.

Other options for new interactions are discussed in chapter 4.

3 The PowerCursor Toolkit

This chapter deals with the design and creation of the PowerCursor toolkit, and the decisions and considerations that were made during the process. For a manual on how to use the PowerCursor toolkit, see Appendix B (p. **Error! Bookmark not defined.**).

3.1 Terminology

The PowerCursor toolkit consists of the following parts. First, there is the *engine*, which keeps track of and manages communication between other parts. Then there are *cursors*, artefacts that can be displaced by the engine. The engine calculates these displacements based on mouse movements and additional 'forces' sent by *objects*. Objects are shapes on the screen that can affect cursors when they touch them, by exerting 2-dimensional forces on these cursors. The objects come in two flavours: *ready-made objects* such as a hill or gutter, which already have a shape, and *modifiers* which cover no area themselves but turn an existing non-PowerCursor shape into a cursor-affecting one.

All of these items are created in Flash. In Flash, self-contained units of graphics and/or animation are called *symbols*. Symbols come in three flavours: Those with their own timeline (MovieClips), those without (Graphics), and the special-purpose Buttons. All PowerCursor parts have been implemented as Flash symbols, and more specifically, MovieClips. The canvas of a Flash document on which these symbols can be placed is called the *stage*.

> To separate the workings of the final PowerCursor toolkit from the various interesting developments it went through during its evolution, the information on the development process has been separated into grey text boxes such as this one.

3.2 Platform

3.2.1 Macromedia Flash MX 2004

The PowerCursor toolkit has been implemented in Macromedia Flash MX 2004 (Flash version 7). This platform was chosen for several reasons.

First, Flash provides a rich graphical design environment in conjunction with a programming environment, allowing users to easily create graphics on screen and add behaviour to them as well. This is what we need to create the toolkit. Flash MX 2004 features an improved programming language called ActionScript 2.0 (see paragraph 3.2.2) which even allows for object-oriented programming of the graphical symbols. Second, Flash is already widely used in the (interface) design community. This is a major advantage, because it means we will not have to develop some interface design application for our PowerCursor components and testers nor end users will have to learn to use such an application. Finally, Flash MX 2004 has some features that simplifiy the distribution of our toolkit (see section 3.2.3).

Although using Flash as the platform for the PowerCursor toolkit has many advantages, there are also some inherent drawbacks.

One such limitation is posed by Flash's security restrictions. Flash uses an extensive sandbox security system to limit the transfer of information that might pose a risk to security or privacy. A Flash movie always executes inside a sandbox, and access to information outside the sandbox is severely limited. A Flash movie cannot access the user's operating system, hard disk, screen resolution, or webcam (unless, in some cases, explicit permission is granted by the local user). This poses some limitations to our toolkit, particularly the fact that Flash cannot change the system cursor's position. Flash does offer easy ways to hide the real mouse cursor and replace it with a custom one, making this drawback a little less disadvantageous.

Another drawback of Flash is that even though it supports object-oriented programming, it is not as powerful or convenient as non-scripting object-oriented programming languages such as C++ or Java. Flash has evolved into a sometimes awkward combination of programming language and graphical editor, and anyone who programs Flash's symbols in an object-oriented manner is bound to encounter some difficulties. Compilation is slow, variables that refer to symbols have to be declared in both the code and the authoring environment, code execution is tied to Flash's symbols and timelines, and its syntax is very tolerant which makes syntax checking and debugging facilities are rather poor.

3.2.2 ActionScript 2.0

"ActionScript" is the scripting language of Macromedia Flash. Since its introduction several years ago, the ActionScript language has grown with every new release of Flash, adding new keywords, objects and methods. In Flash MX 2004, new concepts were introduced to implement object-oriented programming in a more standard way. This fundamental addition called for a new version of ActionScript itself: ActionScript 2.0. Although this new language still supports all the elements of the original ActionScript language, its new features enable authors to write scripts that more closely adhere to standards used in other object-oriented languages, such as Java. [5a]

The architecture of the PowerCursor classes in ActionScript can be found in section 3.5.1.

3.2.3 Version 2 Components versus MovieClips

Apart from the new ActionScript 2.0 scripting language, Flash MX 2004 also features a new architecture for creating components: the Macromedia Component Architecture (MCA) version 2, which is very different from Flash MX's version 1 [5b]. Components created in MCA version 2 are usually referred to as 'version 2 components'.

Version 2 components differ greatly from version 1. To name a few differences, they are built with ActionScript 2, have an object-oriented hierarchy, are skinnable, and can be packaged into a single SWC file. Especially this last feature is convenient for the PowerCursor toolkit, because it allows for both easy distribution and concealable code. SWC files (which use the `.swc` file name extension) are archives that contain all the ActionScript, SWF files, and other optional files needed to use a component. Although the content of an SWC file is meant to remain hidden, SWC files use the ZIP file format (MIME type `application/zip`), and can be readily unpacked using any ZIP archiver such as WinZip.

> Implementing our PowerCursor objects as version 2 components in Flash seemed a good idea at first. Hence, this path was followed for several months, but as numerous drawbacks of components emerged along the way, it was decided to use common MovieClips for the toolkit instead.

Components have several drawbacks as well. It is not possible to create a rotatable, scalable component. This is caused by a bug in the interpretation of height and width in the UIObject/UIComponent class. When Flash components are rotated, they may be stretched out of proportion (even out of their bounding box!) in the authoring environment. Hence, components may have a very different appearance in the authoring environment than when published. Because we want the use of our cursor displacement object to be as intuitive and as WYSIWYG as possible, these are undesirable properties of components.

MovieClips cannot be packaged and distributed the way components can. Distributing the PowerCursor as a Flash document containing MovieClips forces us to make PowerCursor open-source, as the code must accompany the Flash document in separate plain-text files. However, using runtime sharing (see section 3.5.3) the toolkit can be distributed through a single SWF file, keeping its source files (and code) exclusive to the developers. MovieClips do not have the aforementioned problem with rotation and scaling, but can never be redrawn within the authoring environment.

In summary, although Flash components have superior distribution possibilities, their behaviour when used in Flash is not very well suited for PowerCursor objects. For this reason, we chose to implement the PowerCursor toolkit's objects as MovieClips.

3.3 Design considerations

This section describes the main issues that were considered in the design of the PowerCursor toolkit. Most technical matters are discussed in section 3.5.

3.3.1 Accessible for interface designers

The goal of the PowerCursor toolkit is to allow interface designers to create new cursor behaviours into their designs. Naturally, we aim to make the toolkit itself as usable as possible for this target audience, and several core aspects of the toolkit are geared towards this purpose.

The aforementioned choice of implementing the PowerCursor toolkit in Flash is a great step forward in making a toolkit suitable for interface designers. Macromedia Flash is a widely used tool in the interface and interaction design world, and any such designer is almost certainly familiar with the application and its possibilities. Also, any interface design that has previously been created in Flash can easily be enhanced with PowerCursor functionality. Flash's layering system allows for new, transparent layers to be created on top of the initial design, in which all the PowerCursor objects can be placed. In this way, the original interface design is left untouched and separated from the new features introduced by the toolkit.

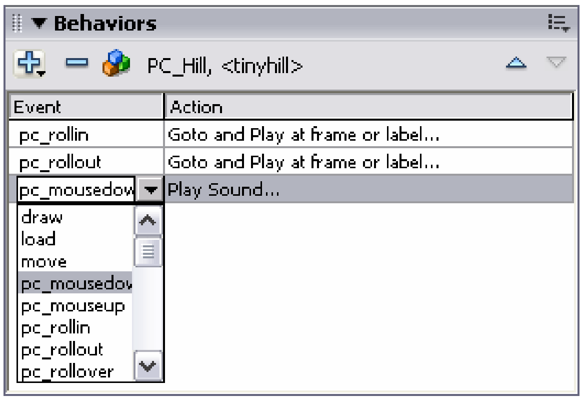
To further accommodate interface designers, PowerCursor has been built to work with interfaces of any size, shape, appearance, and purpose. All ready-made objects can be freely scaled and rotated, and their effect on the cursor can be modified using parameters. Moreover, the special modifiers allow designers to create custom-shaped PowerCursor objects, tailored to their own interface. All PowerCursor objects can be freely combined into more complex arrangements. And although PowerCursor objects have a default appearance, they function completely independent of these visuals. Objects can be shaded, coloured and made semi- or completely transparent without any loss in functionality (for more information on this, see section 3.6.). All of this gives designers more freedom in creating their work.

3.3.2 Accommodating for non-programmers

The amount of new cursor behaviours that might be implemented using Flash and the PowerCursor toolkit is virtually limitless, especially since ActionScript can be used to add custom and dynamic behaviour to Flash symbols. Interface designers might not always be very skilled at Flash ActionScript programming, though.

Although non-programmers will not be able to create behaviours as complex as designers who do use ActionScript, we still want to accommodate for non-programmers by making as many of PowerCursor's potential designs and behaviours as possible accessible without coding. For this reason, the PowerCursor toolkit was designed with non-programmers in mind – to make it possible to create relatively complex behaviours with no, or very little, need for programming skills.

Fortunately, Flash MX 2004 already includes several features to make encoding behaviour into graphical objects relatively easy. The most important feature is the so-called Behaviors panel. Normally, to add behaviour to a symbol, one has to write ActionScript code in a special panel while the right symbol is selected. However, using the behaviours panel, it is possible to have this code generated and attached to the symbol by the application itself, without the need for actual programming.



Event	Action
pc_rollin	Goto and Play at frame or label...
pc_rollout	Goto and Play at frame or label...
pc_mousedow	Play Sound...

Figure 4: The Behaviors panel.

The Behaviors panel allows Flash users to trigger any of a list of predefined actions on any of a list of predefined events, for every individual Flash symbol. For example, the action "Stop playing" could be triggered on the event "Mouse click" on a particular symbol, to stop the symbol's animation whenever it is clicked. To make the toolkit as powerful and easy-to-use as possible for non-programmers, the PowerCursor toolkit has been designed to be fully compatible with the Behaviors panel. For more information on how this was implemented, see section 3.5.2.

Apart from codeless event handling through the Behaviors panel, there are other features in the toolkit designed to assist non-programmers. For instance, many of the variables used in the objects' and cursors' source code are accessible in the Flash authoring environment as parameters. Although these parameters are not more accessible than the code (both are available at the click of a button), they are safer and more familiar for someone who has never or rarely worked with ActionScipt code.

3.3.3 Flexible generic component design

In the design of the various objects in the PowerCursor toolkit, it would have been relatively easy to create objects for specific purposes, such as a square hole the size of a desktop icon, or an object that exactly fits Windows' default minimize-maximize-close buttons. Instead, we chose to make PowerCursor objects very generic, to allow for a broad range of new cursor behaviours to be implemented by them. Their shapes are generic (round or rectangular), rotatable and scalable, their graphics can be modified independent of their functionality, and the forces they apply to cursors are configurable.

Moreover, the modifier objects are designed to expand existing symbols of *any* shape. This makes it even easier for designers to add PowerCursor functionality to their designs: PowerCursor functionality can be added to any shape, logo, or other graphic in Flash. Such graphics can be made into a hole, or a wall, or a slick area by simply adding a modifier to that graphic.

It is also possible to combine PowerCursor objects with other Flash symbols, including other PowerCursor objects, into new compound symbols. And by adding a special Dimmer object (see section 3.4.3) to other objects, specific sections of objects can be deactivated. This makes it possible to produce new shapes such as gutters cut in half or quarter-hills. Through this mechanism, one can also combine parts of objects with each other, which could for example result in a circular area which is half hill and half hole. Additionally, Dimmers can be used to diminish the effect of the mouse movements.

3.4 Operation

3.4.1 Engine and Cursors

At its name implies, the engine is the heart of the PowerCursor toolkit. Without it, none of the other parts will operate properly. One of the few requirements for using the toolkit is that the Engine always be placed in the 'root' of the document. This way, the engine can always be found by all objects and cursors. Because the engine needs to send displacements to the cursors, all cursors report themselves to the engine immediately when they are created. The engine is the only entity that has a list of all the cursors, and is therefore always the link between objects and cursors. It acts as a mediator between objects and cursors, both providing the cursors' information to the objects and propagating the object's forces to the cursors (see Figure 6). However, it does not simply pass on these forces but compiles them into a single displacement per frame.

Objects such as hills and gutters request every cursor's position continuously (e.g., at every frame) through the engine. They then compare the cursors' coordinates with their own, testing whether any cursors are touching the object's area. If so, the object will calculate a (2-dimensional) force for each of those cursors, and sends these forces to the engine. Every frame, the engine may thus receive a multitude of forces from different objects and targeted at different cursors. It is the engine's task to combine these forces with each other (which can be done in several ways) and with other data. This process is illustrated in Figure 7. Taking into account the forces as well as the cursor's current speed, grip, inertia, and mouse movements, the engine then calculates a (2-dimensional) displacement vector for every cursor and sends it to the cursors involved.

In the initial design of the PowerCursor toolkit, the engine maintained many records containing all information on all the cursors. At this point, the cursors were no more than puppets, completely controlled by the engine. When we wanted to make more interesting cursors (such as the CursorMaker and the ForceSensor) it was necessary to give the cursor class more independence. The cursor classes now have numerous variables, as well as properties through which other symbols can access them. The cursor-records in the engine were replaced by just the addresses of the cursors, through which the engine can request information from the cursors themselves when needed.

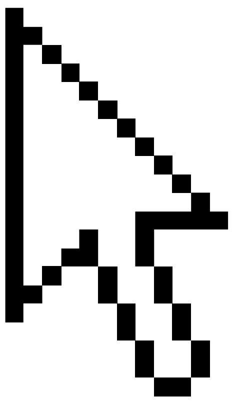


Figure 5: The Arrowcursor.

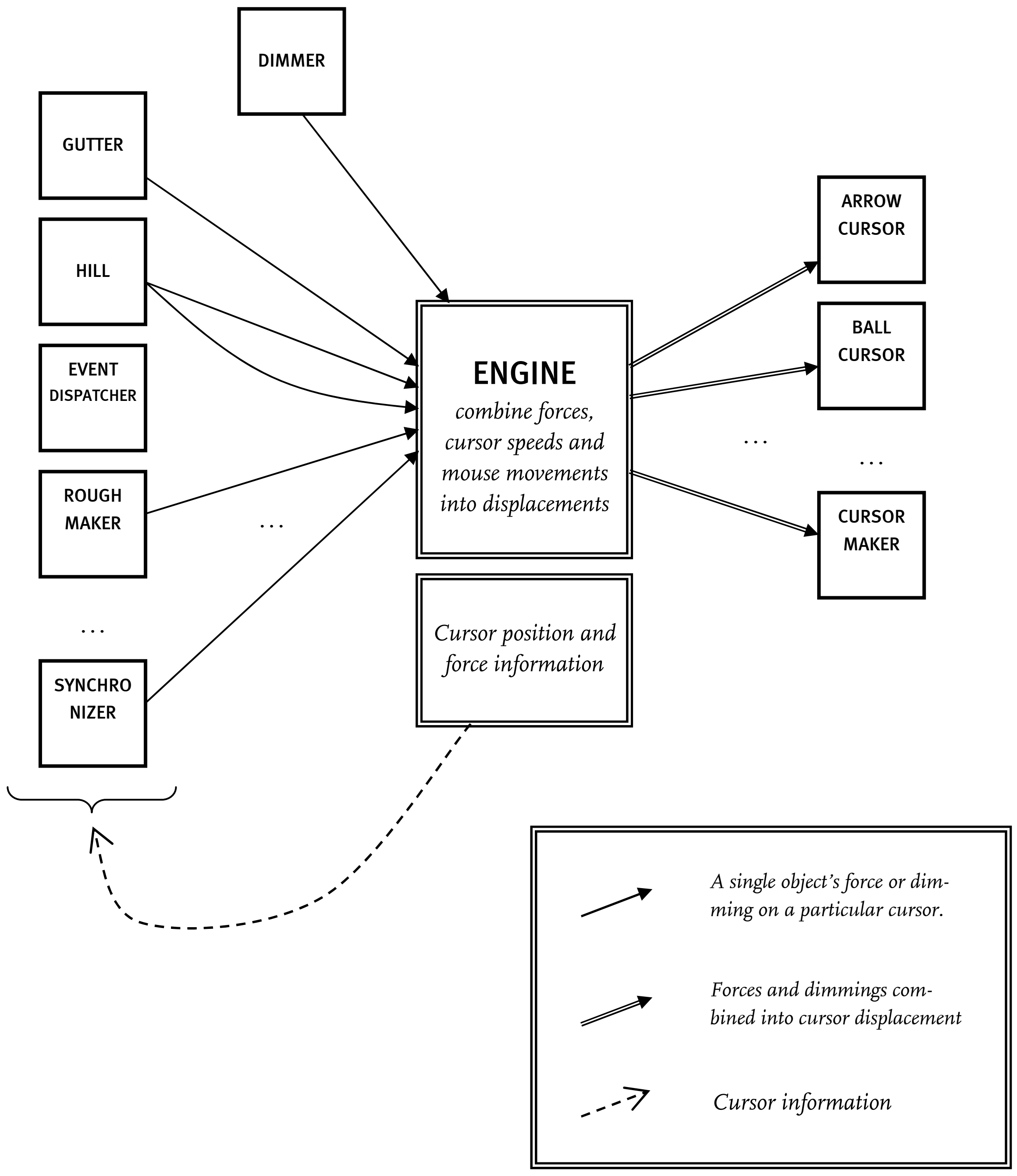

DIMMER
GUTTER
HILL
EVENT DISPATCHER
ROUGH MAKER
...
SYNCHRO NIZER
...
ENGINE
combine forces, cursor speeds and mouse movements into displacements
Cursor position and force information
ARROW CURSOR
BALL CURSOR
...
...
CURSOR MAKER
A single object's force or dimming on a particular cursor.
Forces and dimmings combined into cursor displacement
Cursor information

Figure 6: Calculation of cursor displacements through the different PowerCursor objects.

Cursors passively wait until they receive displacements from the engine. They typically apply that displacement to themselves, although some special cases (the ForceSensor) might not. There are several types of cursors: The ArrowCursor is the default cursor (see Figure 5), the BallCursors are round and can bounce against each other, the CursorMaker can be used to create cursors with custom graphics and the ForceSensor visualizes displacements instead of moving. Note that the PowerCursor toolkit does not force cursors to the top layer of all graphical objects, and therefore cursors may be hidden behind other symbols if they are not arranged into the top layer.

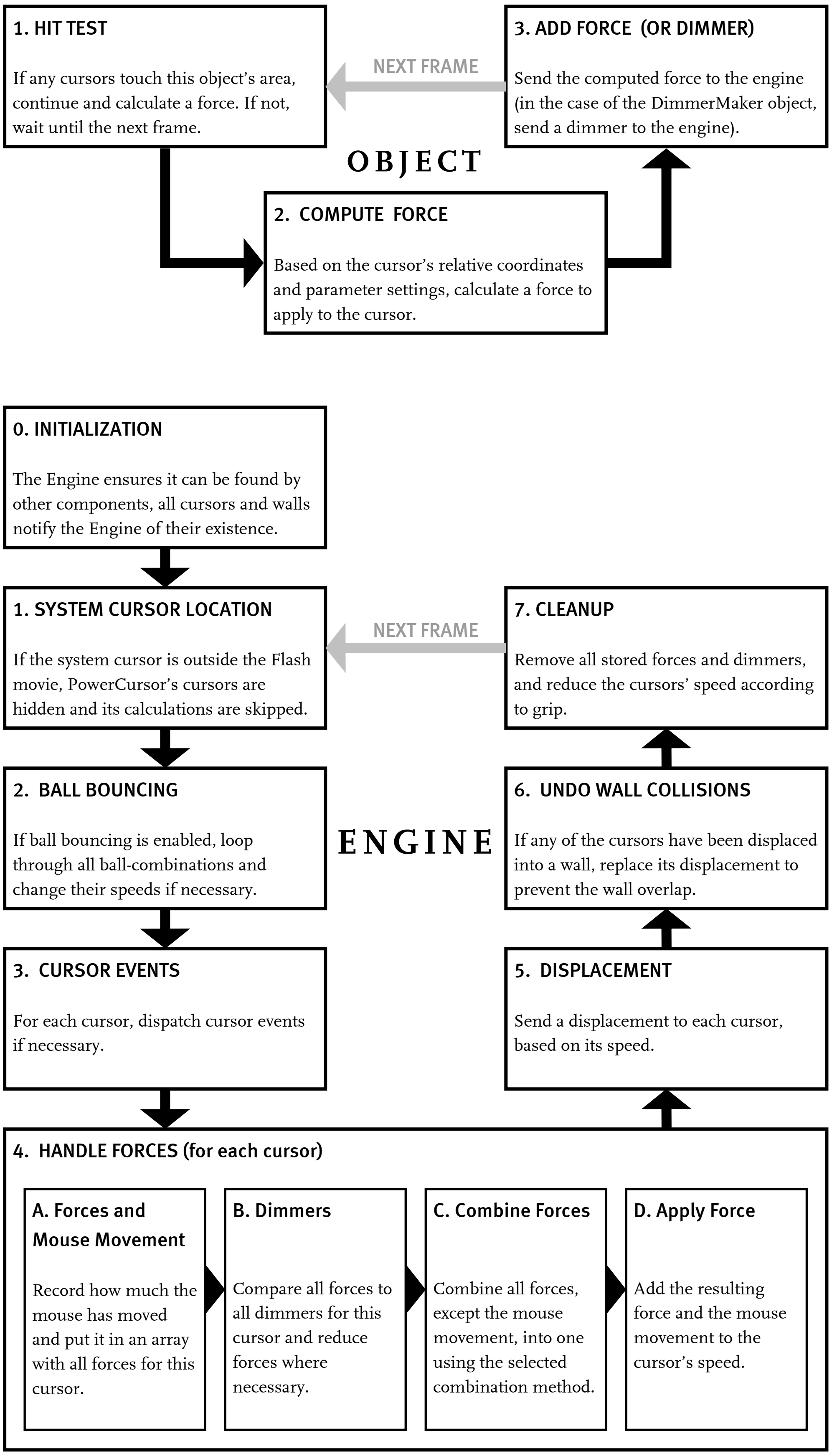

1. HIT TEST
If any cursors touch this object's area, continue and calculate a force. If not, wait until the next frame.
NEXT FRAME
3. ADD FORCE (OR DIMMER)
Send the computed force to the engine (in the case of the DimmerMaker object, send a dimmer to the engine).
OBJECT
2. COMPUTE FORCE
Based on the cursor's relative coordinates and parameter settings, calculate a force to apply to the cursor.
0. INITIALIZATION
The Engine ensures it can be found by other components, all cursors and walls notify the Engine of their existence.
1. SYSTEM CURSOR LOCATION
If the system cursor is outside the Flash movie, PowerCursor's cursors are hidden and its calculations are skipped.
NEXT FRAME
7. CLEANUP
Remove all stored forces and dimmers, and reduce the cursors' speed according to grip.
2. BALL BOUNCING
If ball bouncing is enabled, loop through all ball-combinations and change their speeds if necessary.
ENGINE
6. UNDO WALL COLLISIONS
If any of the cursors have been displaced into a wall, replace its displacement to prevent the wall overlap.
3. CURSOR EVENTS
For each cursor, dispatch cursor events if necessary.
5. DISPLACEMENT
Send a displacement to each cursor, based on its speed.
4. HANDLE FORCES (for each cursor)
A. Forces and Mouse Movement
Record how much the mouse has moved and put it in an array with all forces for this cursor.
B. Dimmers
Compare all forces to all dimmers for this cursor and reduce forces where necessary.
C. Combine Forces
Combine all forces, except the mouse movement, into one using the selected combination method.
D. Apply Force
Add the resulting force and the mouse movement to the cursor's speed.

Figure 7: The objects' and engine's activities per frame.

3.4.2 Objects and Modifiers

The PowerCursor toolkit includes many objects, divided into ready-made objects and modifiers. As their name implies, ready-made objects work right out of the box, and they can be added to a Flash document simply by dragging them onto the stage. Ready-made objects have some default graphics, and are perfect for quick prototyping or a first exploration of the toolkit. Modifiers do not have such standard graphics attached to them. As mentioned in the previous section, all objects check at every frame whether any cursors are touching them. Ready-made objects compare the cursors' coordinates against their own standard graphics.

Modifiers, on the other hand, compare the cursors' positions to their parent symbol. Thus, to use a modifier, the modifier itself should be included in a new symbol, together with another symbol or shape it should modify. In this way, modifiers can add PowerCursor functionality to any symbol or shape in Flash. For example, it is possible to draw any custom shape in Flash using the Brush tool and subsequently make that surface sticky or rough or impassable by adding the appropriate modifier to it.

Although the new parent symbol defines the object's shape and also broadcasts it events, the modifier's parameters are still maintained within the modifier symbol itself.

The Watcher class does not belong to the PC_Object class, as it has nothing in common with the other objects – it does not compare its own position with cursors or exert forces. Instead, it simply retrieves information about a single cursor from the engine and displays it on screen. It shows the cursor's position and speed, which objects try to affect the cursor, and the current forces acting on it. It is included in the toolkit for educational and debugging purposes.

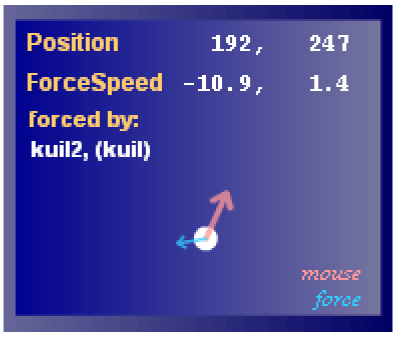

Position 192, 247
ForceSpeed -10.9, 1.4
forced by:
kuil2, (kuil)

Figure 8: The Watcher object.

3.4.3 Dimmers and Walls

Dimmers and walls are two special kinds of objects. Walls can be created using either the PC_Wall object or the PC_WallMaker modifier, and dimmers can be constructed using the PC_Dimmer or the PC_DimmerMaker class.

Dimmers are objects that, instead of exerting forces on cursors, weaken or deactivate the forces applied to cursors. Dimmers apply their dimming effect to cursors that touch them, in the same way that other objects apply their forces to cursors that touch them. Two different types of dimmers can be constructed: mouse dimmers, which reduce the effect of mouse movements on cursors that touch them, and force dimmers, which come in a local and global flavor. Global force dimmers weaken all forces on the cursors they affect, while local force dimmers only diminish forces from objects that have the same parent as the dimmer itself. When the engine handles these forces, it compares all forces on the cursor to all dimmers, and applies the dimming effect when appropriate by scaling the forces' strength.

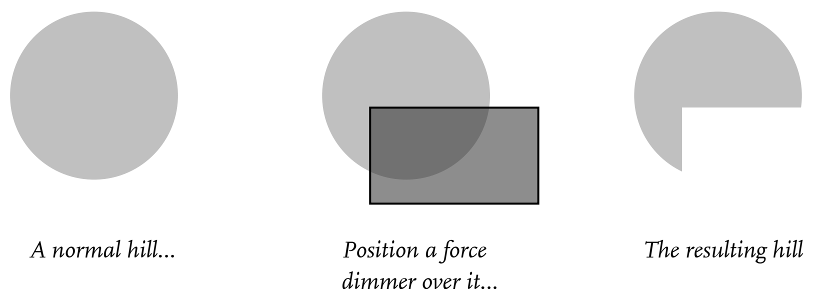


Figure 9: Dimmers can deactivate other objects.

Because of their force-nullifying ability, dimmers can be used to re-shape objects. Hills are by default round, but sometimes a half- or quarter-circle hill may be required. Such shapes can be 'cut out' of normal hills using dimmers. Of course, these new shapes can also be combined into new objects. This makes dimmers an important tool for advanced designers creating their own custom PowerCursor objects.

The Wall is other type of special object. While other objects act on any cursor that touches it, the wall is an object that cannot be touched at all. Whenever a displacement would put a cursor on a wall, that displacement it reconsidered by the engine until it finds a suitable displacement that prevents the overlap. In its search for an acceptable displacement, the engine considers several 2-dimensional factors of the original displacement vector. This allows cursors to 'slide' alongside a wall even if they are forced into it at an angle.

Although the wall's implementation makes it impossible to move a cursor onto a wall area, it is possible to 'jump' over a wall by moving the mouse briskly. This can cause the cursor to move from one side of the wall object to the other without touching the positions in between, hence, jumping over the wall.

3.5 Architecture

3.5.1 Classes

ActionScript 2.0 provides new language elements to implement object-oriented programming standards. This new version of the ActionScript scripting language supports classes, inheritance, interfaces, strong typing, and the event model. In older versions of Flash, all Actionscript code was embedded in the Flash `.fla` file itself, but the new classes should be stored in separate `.as` files (similar to Java's `.class` files). These ActionScript classes can be implicitly or explicitly imported into other class files.

We chose to use these new features of ActionScript and have implemented PowerCursor in an object-oriented way. In Figure 10, the classes on the PowerCursor toolkit and their inheritance are illustrated. For simplicity's sake, the "PC_" class name prefixes are omitted in this diagram.

Most of the PowerCursor toolkit's functionality is coded inside three basic classes: The Object, the Engine, and the Cursor. As illustrated in the diagram, these are all subclasses of the standard Flash class mx.core.UIObject, a direct subclass of the MovieClip.

The Object is a superclass of all ready-made and modifier objects, and contains their common functionality: comparing its area with the cursors (hit detection) and event dispatching. The function to actually exert a force on a cursor is virtual – it is implemented in the individual objects' classes, as every object displaces cursors in a different way, of course. The Object's direct subclasses, the Ready-made Object and the Modifier, consist only of a handful lines of code to differentiate between these two types of Objects. The difference is that Ready-made Objects apply the hit-testing and event dispatching functions of their parent class to themselves, whereas Modifiers use those functions on their parent. the Modifier class has eight children which are all specific modifiers. The Ready-made Object class is inherited by five specific object classes as well as the Round Ready-made Object class, which is a parent to three more specific objects.

The Engine class is the largest class in the toolkit. It has no PowerCursor super- or subclasses; in fact the toolkit is designed to have only one Engine present in any PowerCursor design. The Engine takes care of general tasks that concern more than one component, such as frame rate handling, bouncing balls, and system cursor tracking. More importantly, it computes cursor displacements by combining cursor properties, objects' forces, and mouse movements. The Engine class also uses the Force and Dimmer classes to store its data. The Force class is a record for the forces that objects send to the engine, and the Dimmer class stores dimmers, effects which weaken or cancel out forces on a particular area.

Since PowerCursor cursors are rather passive, the Cursor is a rather simple class. It notifies the engine of its existence when it is created, after which it just waits for displacements to arrive. Whenever the engine calls the cursor's displacement function, it re-positions itself accordingly (or, in case of the ForceSensor, displays some other behaviour based on this displacement).

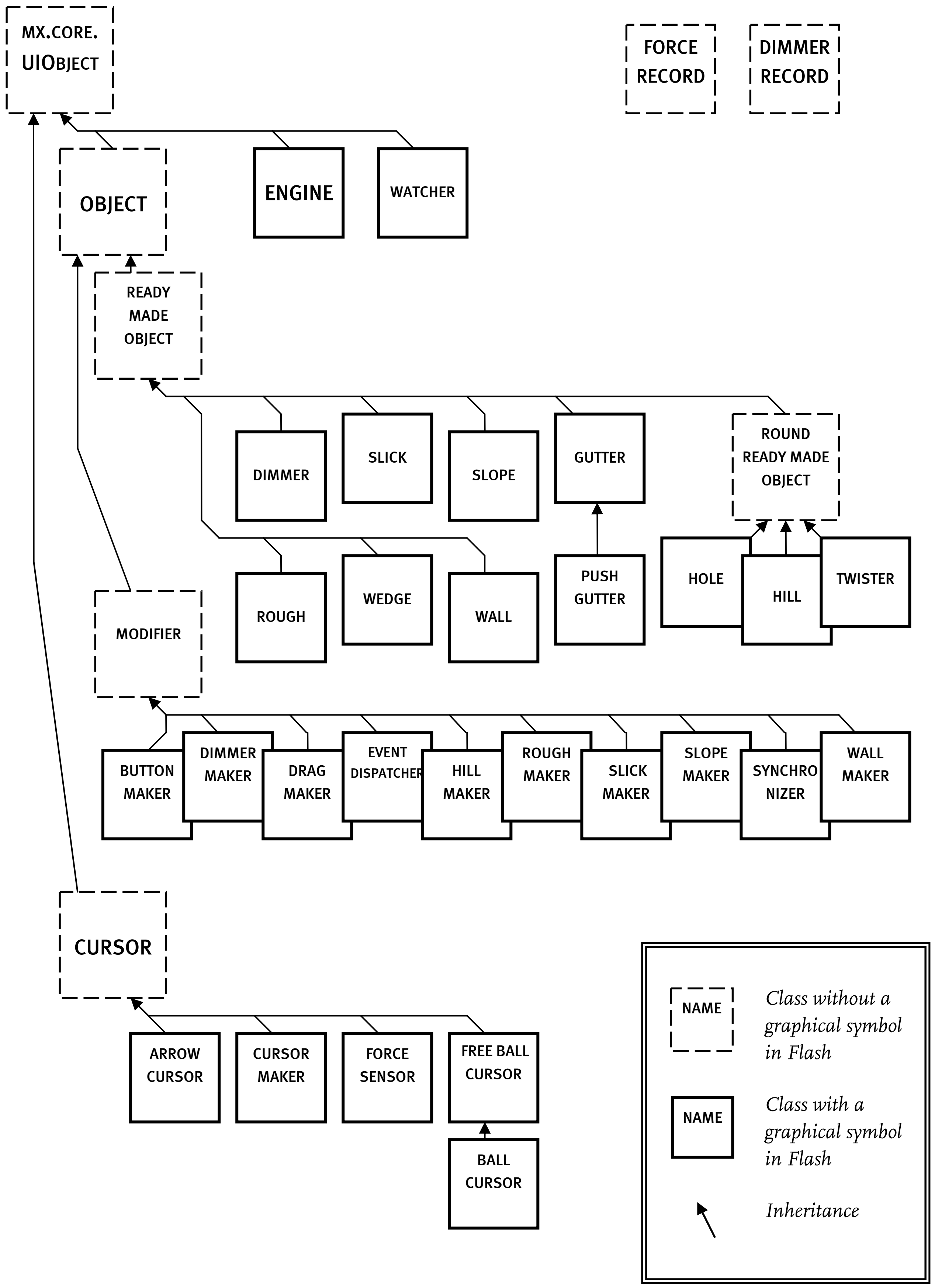


Figure 10. Class inheritance in the PowerCursor toolkit.

3.5.2 Events

Many programming languages feature events, and ActionScript 2.0 is no exception. Events are generated when a certain incident occurs, such as a key being pressed or a mouse being moved. Every time an event is triggered, code that is specifically connected to that event (a so-called event handler) is executed. Flash has some standard built-in events, such as `keyPress`, `mouseDown`, and `rollover`.

Moreover, Flash MX 2004 features a new interface panel with which numerous standard actions can be tied to such events. This so-called 'behaviours panel' allows Flash users to use the power of events without the need to manually code event handlers. For example, though a few mouse clicks in the behaviours panel, a designer could link the action "Go to frame 7" to the event "Mouse roll over". This would cause the selected MovieClip to jump to frame #7 on its timeline whenever the mouse rolls over it.

The built-in mouse events that Flash uses in the Behaviours panel, such as `mouseDown` and `rollover`, are related to the system cursor. As the PowerCursor system uses mock cursors instead of the system cursor, the built-in events do not work properly on PowerCursor cursors and objects. We do still want to offer Flash designers the convenience of using events and behaviours, especially without coding. Therefore, several events were recreated in PowerCursor versions; specifically `pc_rollin`, `pc_rollover`, `pc_rollout`, `pc_mousedown`, `pc_mouseup`, and `pc_plateau`.

These events use the same mechanism as the exertion of forces, namely, comparing the cursors' positions with the object's area. Each object keeps track of the cursors that touch it, and whenever this set changes, `pc_rollin` or `pc_rollout` events are generated. Whenever the set is not empty, `pc_rollover` events are generated, and mouse clicks will trigger `pc_mouseup`s and `pc_mousedown`s. A subset of all objects, namely the hill, hole, twister and (push)gutter, also generate the `pc_plateau` event. This event is spawned when a cursor hits the flat top or bottom (which is called a plateau) of one of these objects

All PowerCursor events are generated consistently across the toolkit; every ready-made object and every modifier in the PowerCursor toolkit works with them. Because the dispatching of events and all related administration is implemented in the PC_Object parent class, all ready-mades and modifiers also send out the same events with the same data at the same incidents. The only difference is that the ready-made objects create the events themselves, while the modifiers dispatch the events from the symbols they modify. So whether a hill comes directly out of the PowerCursor toolkit or was hand-drawn and modified into a hill, they both dispatch the same events.

3.5.3 Runtime sharing

Flash features the possibility to import symbols from external Flash documents at runtime, aptly named 'runtime sharing'. A normal Flash file contains a library which stores the symbols it uses. When using 'runtime-sharable' symbols from another Flash file, they are stored only as a reference to the symbol in the source file. Whenever the source file is updated and published, those updates will show in all files using the runtime shared symbols.

All PowerCursor symbols are configured to be runtime sharable. Presuming that users will actually make use of it (as opposed to edit the source PowerCursor file itself), this offers some convenient maintenance benefits. As the shared PowerCursor files are left untouched, no work is destroyed when they are updated. Better yet, as the symbols are imported from the shared PowerCursor files every time a Flash document that uses them is shown, there is no need to update those documents in any way. As long as the new PowerCursor files have the same filenames as the replaced ones, all Flash files using them will be updated without the need to change them.

Because of the runtime sharing mechanism, PowerCursor can even be deployed through the web. It is possible to create a Flash document that uses PowerCursor object from a shared file on the web, as the references to that file are stored as an URL. Using the PowerCursor toolkit directly from the web has the advantages that one cannot accidentally disrupt the shared source files and they can be updated from the server side (although this might also be considered a disadvantage). Of course, the drawback of this method is that an internet connection to the shared file is required for the Flash document to work at all.

3.5.4 Extension to the system cursor

The PowerCursor toolkit applies its cursor displacements to its own mock cursors, not the system cursor. As a security policy, Flash movies are denied access to the system cursor. Nevertheless, the possibility of controlling the actual system cursor from within a Flash movie has been investigated.

Using a workaround, it is technically possible to extend the PowerCursor toolkit in such a way that displacements can be applied to the system cursor. The Flash class `XMLSocket` lets the computer running the Flash movie player communicate with another computer through a socket (despite its name, it can communicate any string of text, not just XML). Because this other computer should be identified by an IP address or domain name, it could be the same system the player is running on ('localhost'). Hence, a Flash movie can communicate with another application on the same computer.

To make a Flash movie displace the system cursor, an application is required that can both connect to a socket and control the system cursor. The Flash movie should send displacements to a local socket in some format, and the mentioned application should be able to receive those displacements and apply them to the system cursor. With the application running, a Flash movie can thus apply displacements to the system cursor by sending them to the communication socket. It may be hard, though, to synchronize the system cursor's displacements with the PowerCursor objects in Flash, as Flash movies do not have information on where or how large on the screen they are displayed.

As an experiment, a small application as described above was built in C++. We found that the displacements on a PowerCursor cursor in Flash could be applied to the system cursor using the method described above. However, we did not want to force prospective PowerCursor users to install an application before seeing any results, and desired the products from our toolkit to be demonstrable over the web. Hence, controlling the system cursor from within a Flash movie was explored no further.

3.6 Graphics

3.6.1 Requirements

When PowerCursor objects are used to augment existing interface designs, their graphics should interfere with the graphics of these designs as little as possible. For first-time PowerCursor users, on the other hand, the objects should possess a clear visual representation of their functionality at authoring time, and provide some default visuals for those design stages in which the objects should still be seen.

To accommodate these different needs, the PowerCursor's visuals should be flexible. There should be a default appearance for every ready-made object, but this should be customizable and, more importantly, it should be possible to remove it entirely. Modifiers should be invisible when a Flash file is published. The default appearance of any objects should convey information about its operation, especially its visually simulated haptic shape; a PowerCursor Hole should look like an actual hollow in the desktop surface as much as possible.

3.6.2 Implementation

The PowerCursor objects have been designed in such a way that all their functionality operates independently of their graphics. Whether an object is made visible or not, shaded or not, or opaque or not, never affects its operation.

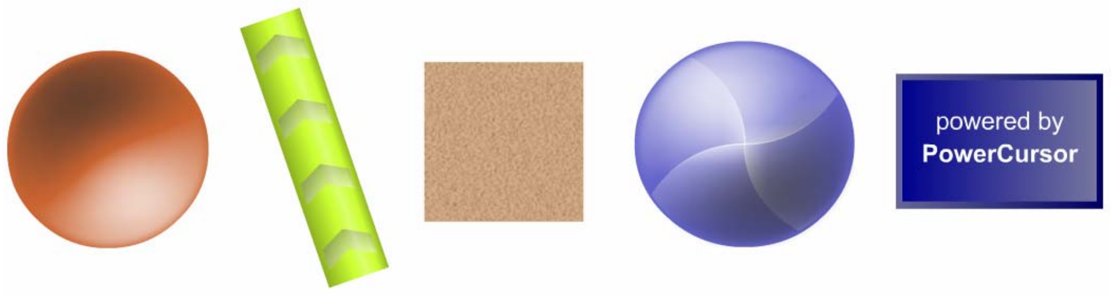


Figure 11: The standard graphics for the Hole, PushGutter, Rough, Twister and Engine.

PowerCursor's ready-made objects have a default appearance with a simple base shape, either round or rectangular (see Figure 11). These shapes are filled, shaded, and boldly coloured by default, but by setting various parameters, these looks can be altered. The base colour can be changed, the base shape can be filled, outlined, or hidden, and shading can also be turned on or off. Unfortunately, any changes in these parameters are not directly visible in the authoring environment (as this is technologically impossible in Flash MX 2004); the changes only show themselves when the Flash movie is published and viewed.

> Initially, even more of the objects properties could be set using parameters, such as rotation, scale, and alpha transparency. This was necessary because Flash Components are not properly transformable in the authoring environment. When the various parts of the PowerCursor toolkit were converted from Components to MovieClips, many parameters were abandoned because Flash's built-in tools for transforming *do* work properly on MovieClips. If we assume that Flash-proficient interface designers would rather use Flash's built-in tools than PowerCursor-specific parameters, this was an improvement in usability.

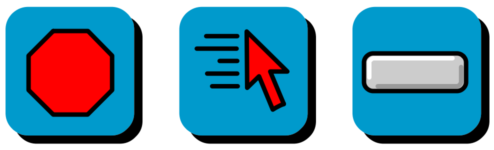


Figure 12: Some of the modifier icons: Wallmaker, Slickmaker, and Buttonmaker.

PowerCursor's modifier objects do not need graphics as elaborate as the ready-made objects. Although visible in the authoring environment, they are made invisible whenever the file is published. During editing time, they are shown as icons on a square blue background. The icon gives an indication of the modifier's purpose, as can be seen in Figure 12.

3.6.3 Shading

To make hills, holes and other shapes as realistic as possible, they can be shaded. The shading used on the PowerCursor Hill and Hole can be seen in Figure 13. This shading is designed to fit the 2.5-D interface shading that is common in GUIs.

However, the realism of the object's shadows is severely constrained by technical limitations. Realistic shading would have to change when the object is altered at authoring time; for example, when the shaded object is rotated, or when a hill or gutter's profile (V-shaped, U-shaped) is changed. But Flash MovieClips, such as the PowerCursor objects, cannot be dynamically redrawn in the Flash authoring environment. Realistic shading that updates on rotation and parameter changes is therefore unachievable.

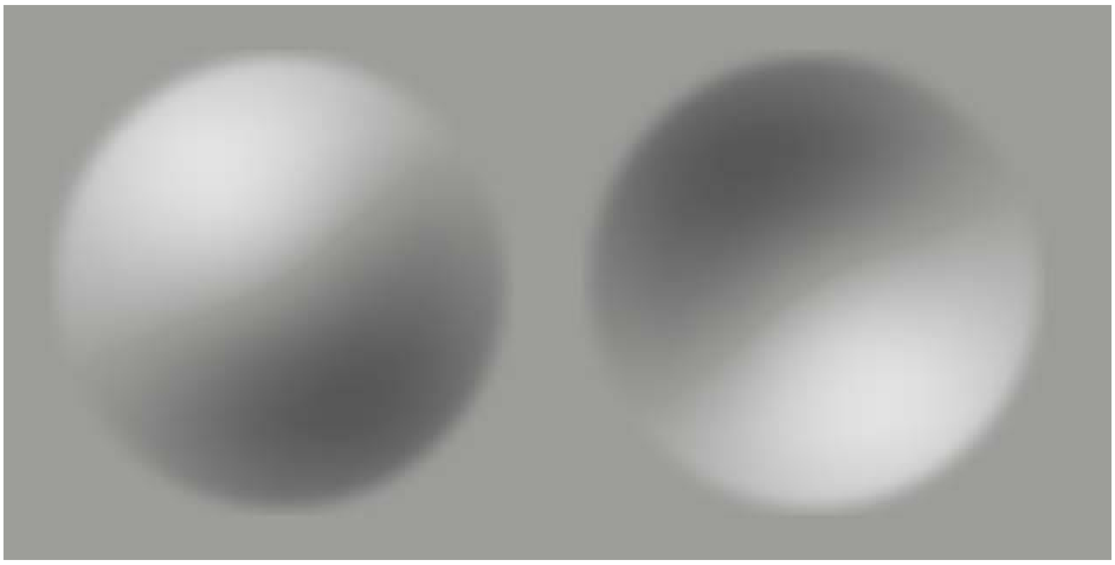


Figure 13: The shading of a PC_Hill (left) and a PC_Hole against a neutral gray background.

As realistic shading is unfeasible, shaded objects just maintain the same shading regardless of rotation or parameters. Because this static shading also rotates along with the object, a hill rotated 180° actually looks like a hole. Of course, there is little reason to rotate a hill or hole as these objects are round. Furthermore, because the standard visualization and shading of objects are meant as temporary graphics for easy prototyping, the fact that they lack realistic shading is not a big problem. Any designer using the PowerCursor object can easily hide the standard visual and place the objects over his own graphics designs.

3.7 Performance

Performance was not a prime concern in the design of the toolkit. PowerCursor is a set of Flash symbols meant for prototyping, and as such it is more geared towards exploration of new cursor behaviours than to providing the best way to implement them. There was no real performance constraint other than the requirement that the toolkit should work on a modern desktop PC which also runs Flash. This has been achieved, and using the PowerCursor toolkit in Flash yields adequate results on a normal desktop PC (Pentium IV 2.0 GHz, 256 MB RAM). Because the toolkit's calculations increase linearly (and in some small cases quadratic) with the addition of more cursors or objects, PowerCursor designs may suffer performance difficulties if large quantities of PowerCursor items are used.

The following problem has also been reported and recreated several times. When the system cursor is moved off of a playing Flash movie and subsequently moved back onto the movie's area, CPU load become very high and the movie suffers from (extreme) slowdown. It seems to occur much more often if the window in which the movie is played is small. The issue has been investigated, but the source of the problem could not be identified. While it is possible that this phenomenon is caused by a bug in the PowerCursor code, it could also be by brought about by the Flash player platform in which the movie runs.

4 Conclusion

4.1 Discussion

As already stated in the introduction, PowerCursor's basic use of cursor displacements has been verified [2]. However, there are still many questions about the use of cursor displacements, its possibilities and its limits. Numerous new ideas and concepts can now be explored.

Although PowerCursor is based upon the use of cursor displacements, this basis is not without its drawbacks. Users are accommodated to having full control over the mouse cursor, and introducing a new element which also affects the cursor may have a negative effect. Users' (perceived) control over the cursor will likely decrease, which might make mouse usage less pleasurable or even less efficient if users are constantly irritated by the computer fiddling with 'their' cursor. Should this prove to be a common problem, turning the PowerCursor's displacements off should be made easy, for example by pressing a key combination or by making a special mouse gesture.

PowerCursor's possibilities for mixed-initiative user interfaces were touched upon section 2.3.3. In these mixed-initiative interfaces, the computer is given a more active role in steering the cursor. Using this technique, the cursor could become more than a pointing device. Normally, the mouse cursor is used only to indicate the user's position on the screen, the point where he or she can act and manipulate, and has no other meaning or effect. But the cursor could be employed in other roles as well. It could, for example, be used as a marker when moving through a decision tree or flowchart. The cursor could be automatically guided from decision to decision, stopping to let the user make a choice or input some data. With the cursor being guided from every decision to the next, the user could get a clear overview of the entire decision chart and, more importantly, where in this scheme he or she stands and how he or she got there. Since the cursor is used as a position marker, re-starting or undoing any number of decisions could be easily and intuitively accomplished by moving the cursor to a previous location in the decision tree.

Also in section 2.3.3, a way to improve wizards using PowerCursor behaviours was discussed. Instead of enhancing wizards, it is not unthinkable that PowerCursor applications might replace wizards as a whole: instead of using a wizard to present the user with a sequence of mandatory choices, the computer could guide a user past the required options by moving the mouse cursor there and opening menus and windows when necessary. Apart from wizards, computer-initiated movement might also be applied in help information or tutorials. Such information could become more interactive, with the computer actually showing and teaching the user how to accomplish things, instead of simply showing the use a manual with instructions.

4.2 Suggestions for further work

PowerCursor is a toolkit built to allow interface designers to experiment with cursor displacements in their designs. As such, it is only the first step in a large project, and with the completion of the toolkit many new possibilities for further work have opened up. Some of these are discussed below.

The PowerCursor toolkit has several imperfections that may be improved upon, even before receiving feedback from users. At present, the toolkit already has various features that make it relatively easy to use for non-programmers. Yet it is still rather difficult to create dynamic structures, such as forces that change over time or react to mouse movement, without resorting to programming. And although the toolkit has been constructed for Flash designers, its' accessibility for first-time Flash users might also be considered. Overall, the toolkit's ease of use could be improved, especially for inexperienced users.

As interface designers will start working with the PowerCursor toolkit, we hope to receive their comments and ideas to further improve the package. Although this version of the toolkit has been worked on for a year, it is still only a first version, and it is likely to be improved when several users' remarks and suggestions have been collected.

Apart from improving the toolkit to better suit the designers that use it, it may also be interesting to research how PowerCursor's displacements can be used best. There are no models yet on how or where cursor displacements are most effectively or most pleasurably applied, how strong or how subtle cursor displacements should be, or whether cursor displacements are equally applicable for all form of computer usage.

At some point, when we have a clear idea of what PowerCursor's capabilities, uses and users are, the toolkit could be rebuilt as a plug-in for an actual interface design IDE such as Borland Delphi, MS Visual Studio, or NetBeans. With such a plug-in, cursor displacements could operate on the system cursor instead of a mock cursor, and GUIs of existing programs could be enhanced with PowerCursor capabilities while retaining all their original functionality.

PowerCursor approaches new cursor behaviours by simulating force feedback displacements. There are other systems that explore new cursor behaviours in other ways [6]. It may be interesting to perform a comparative analysis of different cursor behaviour systems. By evaluating their strengths, weaknesses, and possibilities, it may be possible to find new ways to improve PowerCursor or a new method that approaches new cursor behaviours in a more universal manner.

5 References

[1] K. v. Mensvoort, "What you see is what you feel: Exploiting the dominanace of the visual over the haptic domain to simulate force-feedback with cursor displacements". Proceedings of DIS2002, June 25-28, London, 2002.

[2] K. v. Mensvoort, D.J. Hermes, M. v. Montfort, "Usability of visually simulated force-feedback". Submitted to the International Journal of Human-Computer Studies, 2003.

[3] D. Ahlström, "Modeling and improving selection in cascading pull-down menu's using Fitts' law, the steering law and force fields". Proceedings of CHI 2005, April2-7, Portland, Oregon, USA.

[4] E. Horvitz, "Principles of mixed-initiative user interfaces". Proceedings of CHI '99, May 15-20, Pittsburgh, Pennsylvania, USA.

[5] Macromedia Flash MX 2004 documentation, Macromedia, 2004.
[5a] page "New object-oriented programming model".
[5b] page "What's new in version 2 components".
[5c] page "Using event listerners".

[6] A. Lécuyer, L. Etienne, B. Arnaldi, J. Burkhardt, "Tactile Images: feeling the relief of images". INRIA institute. http://www.irisa.fr/tactiles/

Appendix A: Components

In this appendix, all parts of the PowerCursor toolkit are described. Every entry consists of a short paragraph describing the part's function or use and, where applicable, details of the inner workings. A table then lists the object's public properties, which can be accessed using Actionscipt code. All of these properties are both readable and writable.

A.1 General components

A.1.1 PC_Engine

The PC_Engine care of general tasks that concern more than one component, such as frame rate handling, bouncing balls, and system cursor tracking. More importantly, it computes cursor displacements by combining cursor properties, objects' forces, and mouse movements.

Property	*Type (default)*	*Use*	*Parameter?*
`baseForce`	*Number (100)*	*This number affects the magnitude of all forces. Setting it above or below 100 will scale all forces accordingly.*	*Base Force*
`scaleToAlpha`	*Boolean (false)*	*Enabling Scale-to-alpha multiplies all forces by their object's alpha transparency (0 - 1).*	*Scale force to _alpha*
`combineForces`	*"Strongest", "Topmost", or "Add" (Add)*	*How the engine combines several forces on a single cursor: By using only the strongest, using only the topmost, or by adding the forces together.*	*Combine Forces*
`hideRealCursor`	*Boolean (true)*	*Whether the system cursor should be hidden when it is within (above) the Flash movie.*	*Hide Real Cursor*
`bouncingBalls`	*Boolean (true)*	*Whether balls should bounce off each other or simply pass through each other like other cursors.*	*Bouncing Balls*

A.1.2 PC_Watcher

The Watcher displays some information about a single cursor. It shows the cursor's position, speed (gained from forces), and by which objects the cursor is currently affected. Objects that try to affect the cursor but are dismissed by the engine's force combination method are displayed between brackets. The Watcher was included in the toolkit for educational and debugging purposes.

Property	*Type (default)*	*Use*	*Parameter?*
`averaging`	*Number (7)*	*Over how many frames the watcher's data should be averaged. Not averaging (by setting this parameter to 1) leads to a very irregular display.*	*Averaging (frames)*
`updateTime`	*Number (1)*	*How often the watcher should redraw its data. Minimum = 1 (every frame).*	*Update time (frames)*
`target`	*Number (1)*	*Which cursor the watcher should study. Cursor 1 is the cursor that was added to the stage first, cursor 2 was added second, and so on.*	*Target Cursor*

A.2 Cursors

A.2.1 Cursors in general

These are the properties of the PC_Cursor class, from which all other cursors descend. The PC_Cursor class contains all variables, properties and functions that are common to all cursors. Event dispatching is also handled in this class. Its subclasses set certain variables, such as `isBall` and `graphics_mc`, to a value to make specify their behaviour.

All cursors broadcast some events. These are detailed in section A.5 .

Property	*Type (default)*	*Use*	*Parameter?*
mouseAsForce	*Boolean*	*Whether this cursor should use mouse movements as forces (instead of using them as simple displacements). Default false for the arrow cursor, but true for ball cursors.*	-
grip	*Number (50)*	*If mouse movements are used as forces, grip determines how fast cursor loses speed. High grip means the cursor is fast to lose speed.*	*Force Grip*
inertia	*Number (10)*	*If mouse movements are used as forces, inertia determines how fast the cursor picks up speed. High inertia means the cursor is slow to pick up speed.*	*Force Inertia*
isBall	*Boolean*	*Whether the cursor is a ball or not. The engine can make all cursors for which this is true bounce off each other. Default true for the BallCursor and FreeBallCursor, false for other cursors.*	-
mouseLink	*Boolean*	*Whether the cursor should move with the mouse. True for most cursors, except the FreeBallCursor and the ForceSensor.*	-
surfaceGrip	*Number (1)*	*The grip of the surface the cursor is currently on.*	-
xSpeed	*Number*	*The cursor's horizontal speed (positive = right).*	-
ySpeed	*Number*	*The cursor's vertical speed (positive = down).*	-

A.2.2 PC_ArrowCursor

The PC_ArrowCursor is the default cursor of the PowerCursor toolkit. It has the same appearance as the standard Windows operating system cursor, except that it is coloured red by default.

Property	*Type (default)*	*Use*	*Parameter?*
mouseAsForce	*Boolean (false)*	*Whether this cursor should use mouse movements as forces (instead of using them as simple displacements).*	*Mouse as Force*
tint	*Color (#EC3E3E)*	*The colour of the arrow's body.*	*Graphics Tint*

A.2.3 PC_BallCursor, PC_FreeBallCursor

The BallCursor is a cursor that looks like a round ball. By default, all balls bounce off each other, but this behaviour can be disabled in the PC_Engine. As opposed as to the ArrowCursor, the BallCursor interprets mouse movement as forces by default.

The PC_FreeBallCursor is exactly the same as the BallCursor, except that it's not linked to the mouse (and as such, it will not move unless it is hit by another ball or rolls over an object).

Property	*Type*	*Use*	*Parameter?*
`tint`	*Color (#EC3E3E)*	*The colour of the ball.*	*Graphics Tint*

A.2.4 PC_CursorMaker

The PC_CursorMaker is a modifier that can turn any symbol into a cursor. The CursorMaker symbol should be combined with another symbol in a new compound symbol. The CursorMaker will then render itself invisible and apply its displacements and events on the new compound symbol.

A.2.5 PC_ForceSensor

The ForceSensor is a special kind of cursor. When the Engine sends it a displacement, it does not reposition itself, but instead it draws an arrow indicating the direction and strength of the displacement. In this way, the ForceSensor can be used to inspect the forces at a particular point of the stage. It was included as an authoring tool, not to be included in any final design.

Property	*Type (default)*	*Use*	*Parameter?*
`averaging`	*Number (7)*	*Over how many frames the ForceSensor should average its forces. Not averaging (by setting this parameter to 1) leads to a very irregular display.*	*Averaging (frames)*
`updateTime`	*Number (1)*	*How often the ForceSensor should redraw itself. Minimum = 1 (every frame).*	*Update time (frames)*

A.3 Ready-made Objects

A.3.1 PC_RoundReadyMadeObject

The PC_RoundReadyMadeObject is a parent class for the Hill, Hole and Twister objects, and was created because those objects (due to their similar shape) had a lot of code in common. The PC_RoundReadyMadeObject also handles the pc_plateau event, which is triggered when a cursor hits the objects' centre (its 'hilltop' or 'floor').

Property	*Type (default)*	*Use*	*Parameter?*
`strength`	*Number (3)*	*Strength indicates how much force this object applies. Increase this to push or pull cursors with more force.*	*Force Strength*
`plateauDiameter`	*Number (10)*	*The Plateau is the force-free hilltop (or floor bottom) in the center of the object. This parameter determines its size, in % of the total diameter. Set to 0 to disable the plateau.*	*Force Plateau Diam.*
`shape`	*"V-shaped", "U-shaped" or "Sine-shaped" ("U-shaped")*	*The shape of the hill or hole. V-shaped indicates a constant slope, U-shaped a linear slope. A Sine-shaped hill is sloped as a sine-wave, which results in smooth edges.*	*Force Shape*
`style`	*"None", "Circle", "Disc", "Just*	*The graphical style of the object determines how the object should be drawn: Nothing, an open*	*Graphics Style*

	shading", "Circle with shading" or "Disc with shading" ("Disc with shading")	*circle or a filled circle (disc), and with or without shading.*	
`Tint`	*Color (#FF6600)*	*The colour of the object's circle or disc.*	*Graphics Tint*

A.3.1.1 PC_Hill, PC_Hole

The PC_Hill is an circular object that pushes cursors away from its centre. A round, force-free 'hilltop' can be created in its centre using the Plateau Diameter parameter. Depending on its Strength, the Hill can push harder or weaker, and the shape parameter determines how its force changes with distance from the centre. The different slope-shapes are illustrated in Figure 14.

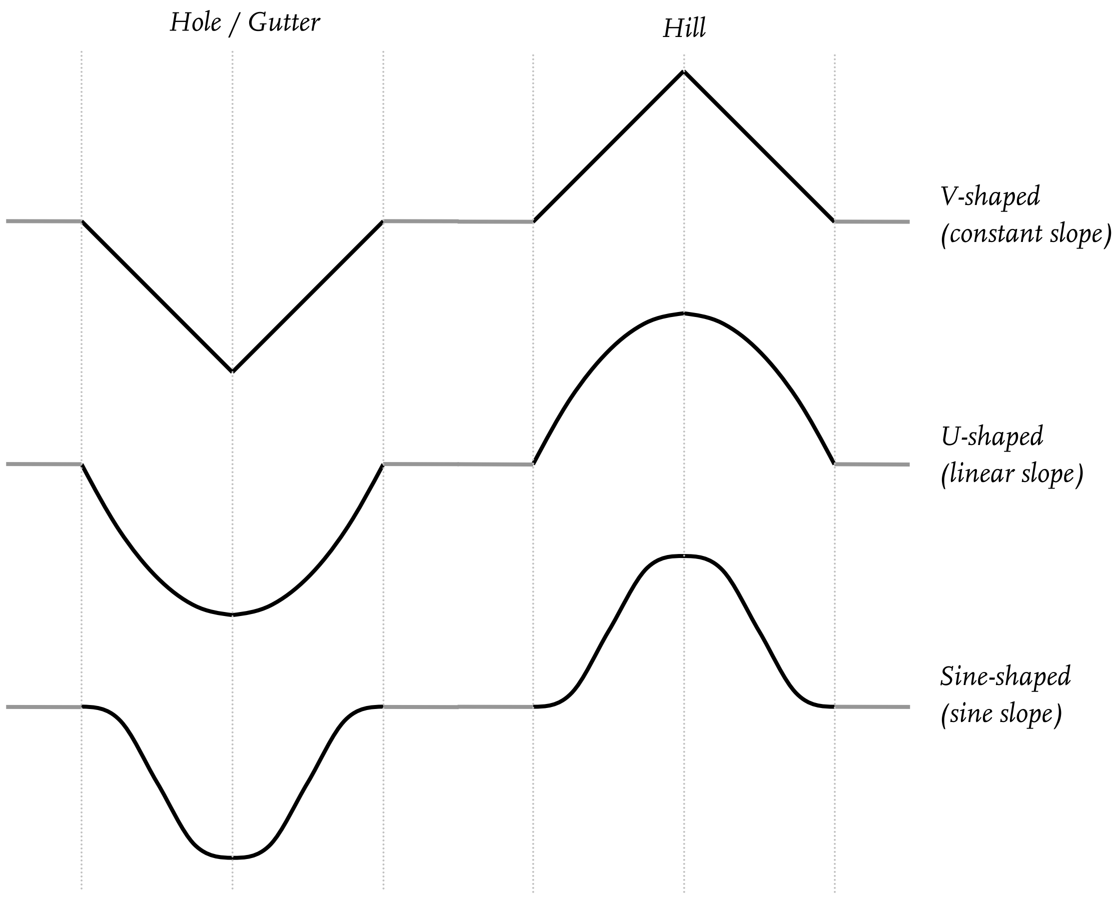


Figure 14: The slope-shapes available for the Hill, Hole, Twister and (Push)Gutter.

The PC_Hole works exactly like the Hill, except it pulls cursors towards its centre instead of pushing them away. It also looks the same, with the exception of its shading. When the PC_Hill or PC_Hole is assigned a negative Strength parameter, it will act like the other type (a Hill will became a Hole and vice-versa).

A.3.1.2 PC_Twister

The PC_Twister acts much like a Hill or Hole, but instead of affecting cursors straight to or from its centre, it spins them around. This can give a 'drainy' effect, where cursors twirl down the vortex like water down a drain. This can be especially useful in combination with the pc_plateau event, which is activated whenever a cursor hits the Twister's centre – in other words, the bottom of the drain.

Property	*Type (default)*	*Use*	*Parameter?*
`spin`	*"Right In", "Right Out", "Left In", "Left Out", ("Right In")*	*Spin indicates whether the twister pulls cursors in or pushes them out, and in what direction. This is based on default ArrowCursors; depending on certain parameters, such as very low grip, a cursor might be pushed out even by a pulling twister.*	*Force Strength*
`vortexStyle`	*"None", "Still", "Animated"*	*The visualisation for the twister's vortex can be set with this parameter. Note that the vortex's animation speed does not match the cursors' spinning speed, as this speed can be different for different cursors.*	*Force Plateau Diam.*

A.3.2 PC_Dimmer

The DimmerMaker is a special object that does not exert forces itself. Instead, it can create areas that weaken or disable forces (or mouse movements). Dimmers add their dimming effect to cursors just like other objects add forces to them. When forces and mouse movement are combined in the Engine, it inspects whether any of those should be weakened by a dimmer, and scales the forces appropriately.

Dimmers affect either the mouse or objects' forces, and object dimmers can have a global or a local scope. Mouse dimmers reduce the effect of mouse movements. Global force dimmers weaken every force on the cursor they affect, while local dimmers only weaken forces from objects with the same parent symbol as the dimmer itself.

Property	*Type (default)*	*Use*	*Parameter?*
`dimmingFactor`	*Number (0)*	*This indicates to what percentage forces are dimmed by the Dimmer. 0 indicates forces are completely dimmed, 100 indicates no dimming, By setting this value above 0, forces are weakened instead of deactivated completely.*	*Dim to ... (%)*
`scope`	*"Mouse", "Global forces" or "Local forces" ("Global forces")*	*If the scope is set to Global, this Dimmer dims all forces on cursors it touches. If it is set to Local, the Dimmer only dims forces originating from objects with the same parent as itself (its siblings).*	*Scope*
`directionDependency`	*"None", "One-way", "Two-way" ("None")*	*When turned on, direction dependency makes this object's dimming depend on the angle of the cursor's movement. One-way makes this object apply its full dimming in one direction and no dimming in the opposite (180°) direction, Two-way applies its full dimming effect at the set angle and its opposite, and has no effect at 90° and 270°.*	*Direction dependency*

`dirDepAngle`	*Number (0)*	*If direction dependency is turned on, this is the angle where the dimmer has maximum effect. If the cursor moves in this direction (or if two-way, in the opposite direction), the dimmer will apply its full dimming effect.*	*Dir. dep. angle*

A.3.3 PC_Gutter, PC_PushGutter

The PC_Gutter is a rectangular area which draws cursors towards its bottom, which is its centre line. In this respect, it works like a hole, but in only one dimension instead of two. The Gutter is useful for guiding cursors along a line. If the Gutter's Strength parameter were to be set to a negative value, the Gutter would turn into a sort of wall. The different shapes available for Gutters are the same as those for hills and holes (see Figure 14).

Property	*Type (default)*	*Use*	*Parameter?*
`strength`	*Number (3)*	*Strength indicates how much force this object applies. Increase this to push or pull cursors with more force.*	*Force Strength*
`plateauDiameter`	*Number (10)*	*The Plateau is the force-free gutter bottom in the center of the object. This parameter determines its size, in % of the total width. Set to 0 to disable the plateau.*	*Force Plateau Diam.*
`shape`	*"V-shaped", "U-shaped" or "Sine-shaped" ("U-shaped")*	*The shape of the gutter. V-shaped indicates a constant slope, U-shaped a linear slope. A Sine-shaped hill is sloped as a sine-wave, which results in smooth edges.*	*Force Shape*
`style`	*"None", "Rectangle", "Block", "Just shading", "Rectangle with shading" or "Block with shading" ("Block with shading")*	*The graphical style of the object determines how the object should be drawn: Nothing, an open rectangle or a filled rectangle (block), and with or without shading.*	*Graphics Style*
`tint`	*Color (#99CC00)*	*The colour of the object's rectangle or block.*	*Graphics Tint*

The PC_PushGutter draws cursors in like a normal Gutter, but it also pushes cursors along its length. It even keeps pushing cursors along when they are on the gutter's bottom (plateau). The PushGutter's two effects – to draw cursors in and to push cursors along – use different strength settings. To indicate in which direction cursors will be forced, arrows can be displayed on the PushGutter.

Property	*Type (default)*	*Use*	*Parameter?*
`pushStrength`	*Number (3)*	*Push Strength is the strength with which the PushGutter moves cursors along its length. This operates independently of the PushGutter's normal 'gutter' strength.*	*Push Strength*
`arrowStyle`	*"None", "Still", "Animated" ("Still")*	*The visualisation for the PushGutter's arrows can be set with this parameter. Note that the arrows' animation speed does not match the cursors' pushed speed, as this speed can be different for*	*Graphics Arrows*

		different cursors.	

A.3.4 PC_Rough

The PC_Rough object is an area that is hard for cursors to get through. It can be used to simulate a sandy or sticky texture. It has both a Roughness and a Stickyness parameter: the first controls the semi-random forces this object applies to the cursor, the second governs the additional speed decrease this object imposes.

The forces this object applies should hinder the cursor when it moves, but be absent when the cursor is still. To achieve this, the force calculated is in proportion to the cursor's current speed and in the opposite direction. Because a random factor is added as well, the exact force applied is different per frame, which suggests a rough texture.

Property	*Type (default)*	*Use*	*Parameter?*
`strengthRoughness`	*Number (3)*	*Roughness determines how much random displacement cursors receive from this object.*	*Strength (Roughness)*
`strengthStickyness`	*Number (3)*	*Stickyness indicates how much extra slowdown the cursor receives from this object. A high stickyness makes the cursor lose speed very fast. When set to 0, there is no additional slowdown, though it is not recommended to set this parameter that low.*	*Strength (Stickyness)*
`style`	*"None", "Square", "Just shading", "Square with shading" ("Square with shading")*	*The graphical style of the object determines how the object should be drawn: Not at all, a flat square, just rough shading or a square with shading.*	*Graphics Style*
`tint`	*Color (#CC9966)*	*The colour of the object's square.*	*Graphics Tint*
`directionDependency`	*"None", "One-way", "Two-way" ("None")*	*When turned on, direction dependency makes this object's force depend on the angle of the cursor's movement. One-way makes this object apply its full force in one direction and nothing in the opposite (180°) direction, Two-way applies its full effect at the set angle and its opposite, and no effect at 90° and 270°.*	*Direction dependency*
`dirDepAngle`	*Number (0)*	*If direction dependency is turned on, this is the angle where the object has maximum effect. If the cursor moves in this direction (or if two-way, in the opposite direction), the object will apply its full rough- and stickyness.*	*Dir. dep. angle*

A.3.5 PC_Slick

The PC_Slick object does not actually exert forces on cursors. Instead, it is an area with a low surface grip, which causes cursors to lose speed more slowly. Cursors will maintain higher speeds on a Slick object, giving the impression of an slippery 'icy' surface. Setting the

Strength parameter of this object to a negative value effectively turns it into a sticky surface, where cursors lose speed more quickly than normal.

Property	*Type (default)*	*Use*	*Parameter?*
`strength`	*Number (3)*	*For this object, the Strength indicates how much the surface grip is reduced for cursors that touch the object. High strength means grip is reduced a lot and cursors will lose speed slowly.*	*Strength*
`style`	*"None", "Square", "Icy" ("Icy")*	*The graphical style of the object determines how the object should be drawn: Not at all, a flat-coloured square, or as an icy light-blue gradient.*	*Graphics Style*
`tint`	*Color (#99CCFF)*	*The colour of the object's square.*	*Graphics Tint*
`directionDependency`	*"None", "One-way", "Two-way" ("None")*	*When turned on, direction dependency makes this object's force depend on the angle of the cursor's movement. One-way makes this object apply its full force in one direction and nothing in the opposite (180°) direction, Two-way applies its full effect at the set angle and its opposite, and no effect at 90° and 270°.*	*Direction dependency*
`dirDepAngle`	*Number (0)*	*If direction dependency is turned on, this is the angle where the object has maximum effect. If the cursor moves in this direction (or if two-way, in the opposite direction), the object will apply its full slipperyness.*	*Dir. dep. angle*

A.3.6 PC_Slope

The PC_Slope is a very basic object, which simply pushes cursors away in a certain direction with constant force. This direction is specified by an angle parameter (in degrees). It is relative to the object, so when a Slope is rotated, it will push cursors in a different direction as well.

Property	*Type (default)*	*Use*	*Parameter?*
`strength`	*Number (3)*	*Strength indicates how much force this object on cursors. Increase this to push cursors harder.*	*Force Strength*
`angle`	*Number (0)*	*The angle at which cursors are pushed. 0 or 360 = up, 90 = right, 180 = down, 270 = left. When the object is rotated, these angles are rotated as well.*	*Force Angle*
`style`	*"None", "Square".*	*The graphical style of the object determines how the object should be drawn: As a flat square or not at all.*	*Graphics Style*
`tint`	*Color (#6600CC)*	*The colour of the object's square.*	*Graphics Tint*

A.3.7 PC_Wall

While other objects act on any cursor that touches it, the PC_Wall is an object that cannot be touched at all. When the Engine is about to displace a cursor, it first checks whether this displacement would position it on a wall. If this is the case, the displacement is cancelled and the Engine will try to find an alternative movement that will not put the cursor on any wall (see Figure 7). Although this makes it impossible to move a cursor onto a wall area, it is possible to 'jump' over a wall by moving the mouse briskly. This can cause the cursor to move from one side of the wall object to the other without touching it.

Property	*Type (default)*	*Use*	*Parameter?*
`tint`	*Color (#333333)*	*The colour of the object's square.*	*Graphics Tint*

A.3.8 PC_Wedge

The PC_Wedge object is similar the PC_Slope object, it simply pushes cursors away with constant force. However, the PC_Wedge pushes cursors away from a single point – like a Hill – rather than in a specific direction. It is shaped like a 45° wedge, or 1/8 of a pie chart, and will force cursors away from its sharp point towards its rounded edge.

Property	*Type (default)*	*Use*	*Parameter?*
`strength`	*Number (3)*	*Strength indicates how much force this object on cursors. Increase this to push cursors harder.*	*Force Strength*
`angle`	*Number (0)*	*The angle at which cursors are pushed. 0 or 360 = up, 90 = right, 180 = down, 270 = left. When the object is rotated, these angles are rotated as well.*	*Force Angle*
`style`	*"None", "Square".*	*The graphical style of the object determines how the object should be drawn: As a flat square or not at all.*	*Graphics Style*
`tint`	*Color (#DD2424)*	*The colour of the object's wedge.*	*Graphics Tint*

A.4 Modifiers

Modifiers are objects which can achieve similar effects as the ready-made objects, but they have no shade themselves. Instead, they can add PowerCursor functionality to any symbol or shape in Flash. For example, it is possible to draw any custom shape in Flash using the Brush tool and subsequently make that surface sticky or rough or impassable by adding the appropriate modifier to it.

All modifiers apply their functionality to their parent symbol. To use a modifier, the modifier itself should be included in a new symbol, together with the symbol or shape it should modify. Although the new parent symbol defines the object's shape and also broadcasts it events, the modifier's parameters are still maintained within the modifier symbol itself.

A.4.1 PC_ButtonMaker

The PC_ButtonMaker does not displace cursors. Instead, it uses PowerCursor's events and an event listener to create button-like behaviour. When the ButtonMaker is touched by a cursor, it can jump to a different frame of its parent MovieClip. On a mouse click, the ButtonMaker can move to yet another frame. These target frames can be set using parameters

The ButtonMaker was included in the toolkit because the button-like use of event dispatchers it implements is probably one of the most used applications of PowerCursor's events. This modifier could also be created by coding an appropriate event listener in an EventDispatcher, but to accommodate for non-programmers this modifier was included separately.

Property	*Type (default)*	*Use*	*Parameter?*

`upframe`	*String or Number (1)*	*This is the frame the ButtonMaker's parent MovieClip will jump to when there is no cursor on it. This can be either a frame number or a frame label.*	*'Up' frame*
`overframe`	*String or Number (2)*	*This is the frame the ButtonMaker's parent MovieClip will jump to when a cursor enters its aera. This can be either a frame number or a frame label.*	*'Over' frame*
`downframe`	*String or Number (3)*	*This is the frame the ButtonMaker's parent MovieClip will jump to when there there is a mouseclick while it has cursors on it. This can be either a frame number or a frame label.*	*'Down' frame*

A.4.2 PC_DimmerMaker

The DimmerMaker is a special modifier that does not exert forces. Instead, it can create areas that weaken or disable forces (or mouse movements). Dimmers add their dimming effect to cursors just like other objects add forces to them. When forces and mouse movement are combined in the Engine, it inspects whether any of those should be weakened by a dimmer, and scales the forces appropriately.

Dimmers affect either the mouse or objects' forces, and object dimmers can have a global or a local scope. Mouse dimmers reduce the effect of mouse movements. Global force dimmers weaken every force on the cursor they affect, while local dimmers only weaken forces from objects with the same parent symbol as the dimmer itself.

Property	*Type (default)*	*Use*	*Parameter?*
`dimmingFactor`	*Number (0)*	*This indicates to what percentage forces are dimmed by the Dimmer. 0 indicates forces are completely dimmed, 100 indicates no dimming, By setting this value above 0, forces are weakened instead of deactivated completely.*	*Dim to ... (%)*
`scope`	*"Mouse", "Global forces" or "Local forces" ("Global forces")*	*If the scope is set to Global, this Dimmer dims all forces on cursors it touches. If it is set to Local, the Dimmer only dims forces originating from objects with the same parent as itself (its siblings).*	*Scope*
`directionDependency`	*"None", "One-way", "Two-way" ("None")*	*When turned on, direction dependency makes this object's dimming depend on the angle of the cursor's movement. One-way makes this object apply its full dimming in one direction and no dimming in the opposite (180°) direction, Two-way applies its full dimming effect at the set angle and its opposite, and has no effect at 90° and 270°.*	*Direction dependency*
`dirDepAngle`	*Number (0)*	*If direction dependency is turned on, this is the angle where the dimmer has maximum effect. If the cursor moves in this direction (or if two-way, in the opposite direction), the dimmer will apply its full dimming effect.*	*Dir. dep. angle*

A.4.3 PC_DragMaker

The PC_DragMaker is a modifier that will make any symbol it is applied to draggable by PowerCursor cursors. This works on both normal symbols and PowerCursor objects, so Hills and Roughs and such can also be made draggable; keep in mind that he combination of a cursor dragging an object and the object simulatiously applying forces to the cursor may lead to some unexpected results.

The DragMaker was included in the toolkit because using events to implement draggability is probably one of the most used applications of PowerCursor's events. Like the ButtonMaker, this modifier could also be created by coding an appropriate event listener in an EventDispatcher, but to accommodate for non-programmers this modifier was included separately.

The PC_DragMaker has no adjustable properties or parameters

A.4.4 PC_EventDispatcher

The PC_EventDispatcher is an empty subclass of PC_Modifier. It does nothing more than dispatch the standard PowerCursor events that every object dispatches when its parent its parent MovieClip is touched by cursors.

The Events dispatched by this object (which are also dispatched all other objects) are detailed in section A.5 .

The PC_EventDispatcher has no adjustable properties or parameters

A.4.5 PC_HillMaker

The PC_HillMaker is a modifier with an effect similar to the PC_Hill object. Because the PC_Hill object is of a fixed size, it can displace the cursor according to various interesting slope-shapes, but the HillMaker does not have that advantage. Therefore, the PC_HillMaker simply detects the nearest edge and pushes the cursor in that direction.

Because the HillMaker has no way of knowing the shape of its parent object, it has to find the nearest edge through trial-and-error. This is rather inefficient, and because it happens every frame for every cursor touching it, the HillMaker can be quite a calculation-intensive component. Having numerous HillMaker objects with several cursors touching them may lead to performance issues.

Property	*Type (default)*	*Use*	*Parameter?*
`strength`	*Number (3)*	*Strength indicates how much force this object applies. Increase this to push or pull cursors with more force.*	*Force Strength*

A.4.6 PC_RoughMaker

The PC_RoughMaker works exactly like the PC_Rough object (see A.3.4), except that the RoughMaker has no surface itself but applies the roughness to another symbol. It can be used to create a sandy or sticky surface in a custom shape.

Property	*Type (default)*	*Use*	*Parameter?*
`strengthRoughness`	*Number (3)*	*Roughness determines how much random displacement cursors receive from this object.*	*Strength (Roughness)*
`strengthStickyness`	*Number (3)*	*Stickyness indicates how much extra slowdown the cursor receives from this object. A high stickyness makes the cursor lose speed very fast. When set to 0, there is no additional slowdown, though it is not recommended to set this parameter that low.*	*Strength (Stickyness)*
`directionDependency`	*"None", "One-way", "Two-way" ("None")*	*When turned on, direction dependency makes this object's force depend on the angle of the cursor's movement. One-way makes this object apply its full force in one direction and nothing in the opposite (180°) direction, Two-way applies its full effect at the set angle and its opposite, and no effect at 90° and 270°.*	*Direction dependency*
`dirDepAngle`	*Number (0)*	*If direction dependency is turned on, this is the angle where the object has maximum effect. If the cursor moves in this direction (or if two-way, in the opposite direction), the object will apply its full rough- and stickyness.*	*Dir. dep. angle*

A.4.7 PC_SlickMaker

The PC_SlickMaker works exactly like the PC_Slick object (see A.3.5), except that the SlickMaker has no surface itself but applies the roughness to another symbol. It can be used to create a slippery or icy surface in a custom shape.

Property	*Type (default)*	*Use*	*Parameter?*
`strength`	*Number (3)*	*For this object, the Strength indicates how much the surface grip is reduced for cursors that touch the object's parent. High strength means grip is reduced a lot and cursors will lose speed slowly.*	*Strength*
`directionDependency`	*"None", "One-way", "Two-way" ("None")*	*When turned on, direction dependency makes this object's force depend on the angle of the cursor's movement. One-way makes this object apply its full force in one direction and nothing in the opposite (180°) direction, Two-way applies its full effect at the set angle and its opposite, and no effect at 90° and 270°.*	*Direction dependency*
`dirDepAngle`	*Number (0)*	*If direction dependency is turned on, this is the angle where the object has maximum effect. If the cursor moves in this direction (or if two-way, in the opposite direction), the object will apply its full slipperyness.*	*Dir. dep. angle*

A.4.8 PC_SlopeMaker

The PC_SlopeMaker is a modifier version of the PC_Slope object (see 0). It works exactly like that object, except that the SlopeMaker has no surface itself but applies its cursor-pushing ability to the surface of another symbol.

Property	*Type (default)*	*Use*	*Parameter?*
`strength`	*Number (3)*	*Strength indicates how much force this object on cursors. Increase this to push cursors harder.*	*Force Strength*
`angle`	*Number (0)*	*The angle at which cursors are pushed. 0 or 360 = up, 90 = right, 180 = down, 270 = left. When the object is rotated, these angles are rotated as well.*	*Force Angle*

A.4.9 PC_Synchronizer

The PC_Synchronizer is a object that synchronizes PowerCursor's mock cursors with the system cursor. This is accomplished by applying forces aimed at the system cursor. This will push the cursor towards the system cursor until their disparity is negligible. The subtlety parameter can make the synchronization process more unobtrusive, but a low strength parameter also goes a long way in achieving this.

Property	*Type (default)*	*Use*	*Parameter?*
`strength`	*Number (3)*	*Strength indicates how much force this object on cursors. Increase this to push cursors harder.*	*Force Strength*
`subtle`	*Boolean (true)*	*When subtle is turned on, the Synchronizer will only apply synchronizing forces when the mouse is moved. Turned off, the synchronizing effect will move the cursor even when the mouse is held still.*	*Subtle*

A.4.10 PC_WallMaker

The PC_WallMaker is a modifier version of the PC_Wall object (see A.3.7). It works exactly like that object, except that the WallMaker has no shape itself and is meant to create a wall in a custom shape. The WallMaker applies its wall-making ability to its parent object.

A.5 Events

As stated in section 3.5.2, PowerCursor events are dispatched in a universal way. PowerCursor events are dispatched from its cursors and its objects. Because the events are implemented in the PC_Object and PC_Cursor superclasses, all individual subclasses handle events in the same way. The specifications of these events can be found below.

Three different events are dispatched by cursors. These can be dispatched once per frame per cursor at maximum.

Event name	*Description*
`pc_forceStart`	*Triggered when objects try to apply forces to this cursor, and this did not happen in the previous frame.*
`pc_force`	*Triggered when objects try to apply forces to this cursor*
`pc_forceEnd`	*Triggered when no object tries to apply forces to this cursor, although this did happen in the previous frame.*

Six distinct events are dispatched by objects. These can be dispatched once per frame per object per cursor at maximum.

Event name	*Description*
`pc_rollin`	*Triggered when the number of cursors touching this object increases.*
`pc_rollover`	*Triggered whenever any cursor touches this object.*
`pc_rollout`	*Triggered when the number of cursors touching this object decreases.*
`pc_mousedown`	*Triggered when, at the moment the mouse button is pressed, any cursors touch this object.*
`pc_mouseup`	*Triggered when, at the moment the mouse button is released, any cursors touch this object.*
`Pc_plateau`	*Triggered whenever any cursor hits the force-free plateau in the centre of certain objects.*

All object events carry an event object with them, which contains information on the cursor that triggered the event. (The cursor events have no event object.) To access the event object, an event listener needs to be coded in Flash. The code required for such an event listener may differ for distinct Flash versions but instructions can be found in the accompanying Flash documentation [5c].

The event object that is attached to the PowerCursor objects' events contains the following information:

Property	*Type*	*Description*
`type`	*String*	*The event type, such as* `pc_rollin` *or* `pc_mousedown`.
`target`	*String*	*The path to the object that dispatched this event.*
`x`	*Number*	*The x-coordinate of the cursor that caused this event.*
`y`	*Number*	*The y-coordinate of the cursor that caused this event.*
`xLastDisp`	*Number*	*The last horizontal displacement (in other words, the horizontal speed) of the cursor that triggered this event. A positive number denotes rightward movement.*
`yLastDisp`	*Number*	*The last horizontal displacement (in other words, the vertical speed) of the cursor that triggered this event. A positive number denotes downward movement.*
`path`	*String*	*The path to the cursor symbol that triggered this event.*