Development of a Graphical Sound Synthesis Controller Exploring Cross-Modal Perceptual Analogies

Liam O’Sullivan

Music & Media Technologies
School of Engineering
&
School of Drama, Film and Music
Trinity College Dublin

Submitted as part fulfilment for the degree of M.Phil.

2007
Declaration

I hereby declare that this thesis has not been submitted as an exercise for a degree at this or any other University and that it is entirely my own work.

I agree that the Library may lend or copy this thesis upon request.

Signed: ____________________   Date: ____________________
Abstract

Analogous perceptual experiences in multiple sensory modalities provide possibilities for abstract expression using modern computer-based audio-visual systems. This work examines the phenomenon of such cross-modal links with a focus on mappings between the auditory and visual realms.

The design and implementation of a graphics-based sound synthesis controller using such inter-sensory associations is presented. A review of literature relevant to the design of computer-based musical instruments is provided, including discussions of parameter mapping, the use of graphical displays and computer vision as a gesture capture mechanism.

An analysis of the software instruments realised and the physical interface setup is provided. System improvements and possible applications are also discussed with the direction of future work with the system being suggested.
Acknowledgements

My sincere appreciation goes to…

…my thesis supervisors; Hugh Denman and Eamonn Doyle- who have put up with endless questions, atrocious grammar and minor nervous breakdowns over the last three months…

…Fionnuala Conway- who has excelled as the captain of the MMT ship for the past year or so…

…the extended MMT family- lecturers, tutors, researchers and staff, and the long-suffering Nora Moore, who helps to hold it all together…

…Brian Solon- for help and tech advice…

… my classmates, both ’06 and ’07. A motley crew, and all the better for it!

But above all, my thanks go to Claire- the most loving, supportive, understanding person I know.
# Table of Contents

1. **Introduction** 1  
   1.1 Motivation  
   1.2 Project Goals  
   1.3 Thesis Structure  

2. **Analogous Perceptual Experiences in Multiple Sensory Modalities** 4  
   2.0 Introduction  
   2.1 Synaesthesia  
   2.1.1 Exploration of Synaesthesia in Art and Science  
   2.1.2 More General Synaesthetic Trends  
   2.2 Sound Symbolism  
   2.2.1 Sound Symbolism Using Simple Shapes  
   2.2.2 Correlations between the Attributes of Sounds and Shapes  
   2.3 Cross-Modal Associations based on Physical Models and Experimental Results  
   2.3.1 Real World Models  
   2.3.2 Empirical Study  
   2.4 Conclusions  

3. **Interface Design Considerations** 19  
   3.0 Introduction  
   3.1 Parameter Mapping  
   3.1.1 Types of Mapping  
   3.1.2 Multi-layered Mapping  
   3.2 Musical Expression  
   3.2.1 Measuring Expressivity of Electronic Instruments  
   3.2.2 Supplemental Performance Gesture  
   3.3 Graphic Displays  
   3.3.1 Graphics-Based Musical Controllers  
   3.3.2 Mouse-Based Graphics Applications  
   3.3.3 Combining Control and Display Surfaces
3.4 Computer Vision as a Gesture Capture Mechanism
  3.4.1 Computer Vision Techniques
  3.4.2 The Camera Musicale

3.5 Outline of Approach

3.6 Conclusions

4. System Framework Design and Implementation

4.0 Introduction

4.1 Software System Framework Design
  4.1.1 The Max/MSP and Processing Environments
  4.1.2 Computer Vision System
    4.1.2.1 reacTIVision
    4.1.2.2 TUIO Simulator
  4.1.3 Software System Overview

4.2 Hardware Setup

4.3 Conclusions

5. Software Instrument Implementation

5.0 Introduction
  5.0.1 Analysis of Design Considerations

5.1 MaluTak
  5.1.1 Overview
  5.1.2 Graphics Generation Using Bezier Curves
  5.1.3 Technical Overview

5.2 DynAudio
  5.2.1 Overview
  5.2.2 Gesture Analysis
  5.2.3 Sound Synthesis
  5.2.4 Graphics

5.3 TwoHand
  5.3.1 Overview
  5.3.2 Gesture Analysis
  5.3.3 Auditory-Visual Mappings
    5.3.3.1 Pitch and Colour, Pitch and Size
5.3.3.2 Amplitude and Size
5.3.3.3 Timbral Brightness and Object Curvature,
Timbral Brightness and Slope
5.3.3.4 Harmonicity and Texture Periodicity
5.3.4 Technical implementation
5.3.4.1 Functional Overview
5.3.4.2 Gesture Analysis and Mapping
5.3.4.3 Graphics
5.3.4.4 Synthesis

5.4 Conclusions

6. Discussion and Analysis

6.0 Introduction
6.1 Software System Framework
6.2 Instrument Evaluations
  6.2.1 MaluTak
  6.2.2 DynAudio
  6.2.3 TwoHand: Technical Implementation
6.3 Installation Evaluation: TwoHand on the Table
  6.3.1 Prototype System Performance
  6.3.2 The TwoHand Experience
6.4 Conclusion

7 Conclusion

7.1 Future Work

Appendix A. Max/MSP Patches
  A.1 MaluTak Max/MSP Code
  A.2 DynAudio Max/MSP Code
  A.3 TwoHand Max/MSP Code

Appendix B. Processing Code
  B.1 MaluTak Processing Sketch
  B.2 DynAudio Processing Sketch
<table>
<thead>
<tr>
<th>Appendix C.</th>
<th>List of <em>Max/MSP</em> Objects</th>
<th>127</th>
</tr>
</thead>
<tbody>
<tr>
<td>C.1</td>
<td>Max Objects</td>
<td></td>
</tr>
<tr>
<td>C.2</td>
<td>MSP Objects</td>
<td></td>
</tr>
<tr>
<td>Appendix D.</td>
<td>List of <em>Processing</em> Methods</td>
<td>131</td>
</tr>
<tr>
<td>References</td>
<td></td>
<td>133</td>
</tr>
</tbody>
</table>
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Grapheme-Colour Synaesthesia</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>Scriabin's 'Keyboard of Light'</td>
<td>7</td>
</tr>
<tr>
<td>2.3</td>
<td>Shapes Used in Kohler’s ‘Maulma/Takete’ Experiment</td>
<td>11</td>
</tr>
<tr>
<td>2.4</td>
<td>Sound Mosaics Software</td>
<td>16</td>
</tr>
<tr>
<td>2.5</td>
<td>Sound Mosaicss Auditory-Visual Mappings</td>
<td>17</td>
</tr>
<tr>
<td>3.1</td>
<td>The Violin</td>
<td>20</td>
</tr>
<tr>
<td>3.2</td>
<td>Kraftwerk on Stage</td>
<td>21</td>
</tr>
<tr>
<td>3.3</td>
<td>Multiple Mapping Layers</td>
<td>24</td>
</tr>
<tr>
<td>3.4</td>
<td>Graphical Music Software</td>
<td>28</td>
</tr>
<tr>
<td>3.5</td>
<td>Pete Rice’s ‘Stretchables’ Installation</td>
<td>29</td>
</tr>
<tr>
<td>3.6</td>
<td>Mouse-based Graphics Software</td>
<td>31</td>
</tr>
<tr>
<td>3.7</td>
<td>Graphics from Abbado’s ‘Dynamics’ Composition</td>
<td>32</td>
</tr>
<tr>
<td>3.8</td>
<td>Combined Display/Control Interface</td>
<td>34</td>
</tr>
<tr>
<td>3.9</td>
<td>The Camera Musicale Setup</td>
<td>38</td>
</tr>
<tr>
<td>3.10</td>
<td>The Camera Musicale’s Scene Analysis System</td>
<td>39</td>
</tr>
<tr>
<td>4.1</td>
<td>Max/MSP Graphical Environment</td>
<td>44</td>
</tr>
<tr>
<td>4.2</td>
<td>Processing Integrated Development Environment</td>
<td>45</td>
</tr>
<tr>
<td>4.3</td>
<td>reacTIVision Fiducial Markers</td>
<td>46</td>
</tr>
<tr>
<td>4.4</td>
<td>reacTIVision Computer Vision Framework</td>
<td>47</td>
</tr>
<tr>
<td>4.5</td>
<td>reacTIVision TUIO Simulator</td>
<td>48</td>
</tr>
<tr>
<td>4.6</td>
<td>Project Software Framework Overview</td>
<td>49</td>
</tr>
<tr>
<td>4.7</td>
<td>Physical Setup of the System Interface</td>
<td>50</td>
</tr>
<tr>
<td>4.8</td>
<td>Infrared LED Light Disc</td>
<td>51</td>
</tr>
<tr>
<td>5.1</td>
<td>MaluTak Graphics Output</td>
<td>55</td>
</tr>
<tr>
<td>5.2</td>
<td>Bezier Curves</td>
<td>55</td>
</tr>
<tr>
<td>5.3</td>
<td>MaluTak Max/MSP Patch</td>
<td>56</td>
</tr>
<tr>
<td>5.4</td>
<td>DynAudio Graphical Output</td>
<td>58</td>
</tr>
<tr>
<td>5.5</td>
<td>Instantaneous Curvature of a Line</td>
<td>59</td>
</tr>
<tr>
<td>5.6</td>
<td>Expression to Measure the Instantaneous Curvature of a Line</td>
<td>60</td>
</tr>
<tr>
<td>5.7</td>
<td>Simple FM Synthesis</td>
<td>61</td>
</tr>
<tr>
<td>5.8</td>
<td>The simpleFM~ Max/MSP Object</td>
<td>62</td>
</tr>
</tbody>
</table>
List of Figures (cont.)

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.9</td>
<td>DynAudio Graphical Output</td>
<td>63</td>
</tr>
<tr>
<td>5.10</td>
<td>TwoHand Graphical Output</td>
<td>65</td>
</tr>
<tr>
<td>5.11</td>
<td>TwoHand Graphics for Varying Inter-Puck Distances</td>
<td>67</td>
</tr>
<tr>
<td>5.12</td>
<td>TwoHand Graphics for Varying Shape Sizes</td>
<td>67</td>
</tr>
<tr>
<td>5.13</td>
<td>TwoHand Graphics for Varying Shapes Curvatures</td>
<td>68</td>
</tr>
<tr>
<td>5.14</td>
<td>TwoHand Graphics for Varying Texture Periodicities</td>
<td>69</td>
</tr>
<tr>
<td>5.15</td>
<td>Functional Overview of TwoHand</td>
<td>71</td>
</tr>
<tr>
<td>5.16</td>
<td>frequency_Scaler sub-patch</td>
<td>72</td>
</tr>
<tr>
<td>5.17</td>
<td>Effect of Different Slopes on Envelope Shape in the scale_And_Generate_Variable_Envelope sub-patch</td>
<td>73</td>
</tr>
<tr>
<td>6.1</td>
<td>Implementation of Sound in MultiTak</td>
<td>80</td>
</tr>
<tr>
<td>6.2</td>
<td>Photograph of the Prototype System Setup with Graphics</td>
<td>85</td>
</tr>
<tr>
<td>6.3</td>
<td>Photograph of the Prototype System Setup without Graphics</td>
<td>86</td>
</tr>
</tbody>
</table>
1. Introduction

"I am fascinated by how abstraction can connect us to a reality beyond language, and the ways in which our gestures and traces, thus abstracted, can reveal the unique signatures of our spirits."

-Golan Levin [Levin, 2005]

1.1 Motivation

The advance of new technology continuously opens up possibilities for new musical instrument interfaces based on the modern computer. All too often however, this potential goes untapped, with commercial demands and uninspired design forcing reliance on the familiar, rather than the revolutionary. Graphical applications that emulate hardware controllers and the dependence on a mouse-and-keyboard interface are two examples of the survival of legacy techniques in the face of creative invention.

It is the premise of this thesis that a more abstract, but no less well-defined, approach to musical control is desirable- an approach that treats the human-machine interaction holistically- specifying it in terms of the human intention rather than a set of discrete machine parameters.

Exposure to Golan Levin's Manual Input Sessions was the catalyst for the project. These audiovisual pieces exhibit a strong intuitive link between the visual shapes and gestures performed and the audio produced by the synthesis system. The relationships between graphical form and auditory event require no grounding in music theory to comprehend, no cultural conventions to be appreciated. Such links between sensory modalities highlight the intention behind the gesture, the abstract meanings of the sound.

This work explores how the seemingly unrelated realms of sound and vision, communicated to us by different senses, can somehow seem to work in union, providing an insight into abstract feelings of expressive intention.
1.2 Project Goals

The goals of the project may be summarised as follows:

- The exploration of the phenomenon of the cross-modal perceptual experience, its possible origin and nature, with an emphasis on associations formed between the auditory and visual domains.

- The design and implementation of software instruments that facilitate musical control of sound synthesis and the generation of correlated graphics using such inter-sensory analogies.

- The specification of an audiovisual environment to serve as a platform for these instruments and in doing so provide an immersive, engaging experience for the user that fosters an appreciation of abstract expression.

1.3 Thesis Structure

Chapter 2 examines the phenomenon of cross-modal perceptual experiences in humans, with an emphasis on the inter-relation of sound and visual form. A line is traced from the subjective experiences of synaesthetic individuals, through more general theories of the origins of inter-sensory associations and on to specific contemporary approaches of mapping characteristic parameters across the auditory and visual realms.

Chapter 3 reviews literature concerning the design of musical instrument interfaces and examines examples of designs facilitated by computer technology.

Chapter 4 documents the design and implementation of a suitable framework to facilitate such an interface. A number of alternative software environments are assessed, the implemented framework is presented and a hardware setup is specified.
Chapter 5 describes the realisation of three software instruments that operate within the framework defined in chapter 4. The progression of the project is traced by the evolution of these instruments in increasing complexity and sophistication.

Chapter 6 analyses the work carried out in chapters four and five, assessing the choice of system framework used and evaluating the implementation of the software instruments. A prototype installation setup is considered from a technical standpoint and from the experience of interacting with it.

Chapter 7 concludes the thesis with some general observations and suggests future work to be carried out in the area.
2. Analogous Perceptual Experiences in Multiple Sensory Modalities

“A studious blind man, who had mightily beat his head about visible objects… bragged one day, that he now understood what scarlet signified. Upon which, his friend demanding what scarlet was? The blind man answered, It was like the sound of a trumpet.”

- John Locke [Locke, 1690]

2.0 Introduction

It is very natural in ‘normal’ conversation to convey to others the experience of a perception gained through the mechanism of one sense, or sense modality, using terminology commonly associated with another: The taste of cheese may be described as sharp or soft. A piece of music may seem dark or sweet. More adventurous types might say that the sound of a trumpet can have colour. In 'less-normal' conversation, as in the writings of poets, more esoteric examples of such descriptions may be found. An ancient Egyptian love poem contained in hieroglyphics uses an analogy drawn between sound and taste: "The sound of your voice is sweet, full like the taste of wine" [Marks, 1978: 75 after Foster, 1971].

Such omnipresent comparisons of qualities from different sensory modalities belies their somewhat oddball make-up. How can a voice sound like a wine taste? No scale exists that can quantify and compare both experiences. Yet the meaning is clear and the emotional impact of the line is heightened by the association. Individual interpretations may differ on the specifics of meaning contained in the line, like what constitutes a full taste for example. On a more holistic level however, the line evokes emotion using a natural and intuitive link between the two pleasurable experiences of drinking a good wine and hearing a lover’s voice.

This chapter looks at some of the mechanisms that allow such analogies to be drawn between the senses and discovers that there is more at work here than mere quirks of language. The way that perceptual experiences can be expressed in terms of seemingly
unrelated sense modalities is not a result of limitations of vocabulary - it is not that there are not enough words to go around and some need to be used more than once. Rather it is that the processes associated with generating these perceptions in one sense have analogous processes that are normally associated with other senses. The experiences themselves can be said to be cross-modal. In this chapter, examples of cross-modality are studied with a focus on the relationships between the aural and visual senses.

The first section examines the phenomenon of synaesthesia, an emphatic example of the intrusion of one sense upon another. The synaesthetic experience can be highly individualistic, yet a history of attempts to tame its mysteries and employ it in art and science exists, and will be broadly investigated. Section 2 introduces more scientific rigor to the discussion and attempts to find more universal models of inter-sensory relationships. The fields of sound symbolism and Gestalt psychology are shown to offer explanations for the origins of correspondences between different perceptual experiences. Experimental evidence from the work of Wolfgang Kohler will verify the existence of such correspondences, specifically for the case of word sounds and graphic shapes. The third and last section looks at some other sense-matching techniques that utilise empirical user data studies and physically-modelled systems to provide the groundwork for demonstrable cross-modal associations.

The overarching aim of the chapter is to ascertain if links exist that allow the mapping of the properties of one sensory modality to those of another. What is important is that such mappings be universal in nature—a result of the human condition rather than a learned set of arbitrary associations. Such a mapping framework might, in turn, facilitate simultaneous artistic expression in multiple modalities that would be widely appreciated at an intuitive level.

2.1 Synaesthesia

The phenomenon of synaesthesia may be described as a mixing of the senses in ‘otherwise normal individuals’ and is now seen as a demonstrable perceptual effect [Ramachandran & Hubbard, 2003]. Individuals who experience synaesthesia, or synaesthetes, consistently associate the quality of one sensory event with that of another. For example, in the variant known as coloured hearing, an individual may see a particular
musical note in a certain colour: as the sound is heard, the synaesthete has a simultaneous experience of being presented with a definite colour. Another common example is a type of grapheme-colour synaesthesia, in which an individual associates a colour with a particular letter- or number-shape, as illustrated in figure 2.1. Tests on individuals with this type of synaesthesia reveal that they can recognise such graphemes within a complex pattern more quickly than non-synaesthetes. To the synaesthete, particular characters seen in different colours stand out more clearly from the background [Ramachandran & Hubbard, 2001].

![SYNESTHESIA](https://via.placeholder.com/150)

*Figure 2.1: How an individual with grapheme-colour synaesthesia might view a collection of letters and numbers. The figure shows an alternative spelling of the word [Image from Wikipedia, 2007].*

Although in-depth discussion of the subject is beyond the scope of this paper, there is strong support for the physiological origins for synaesthesia, with unpublished observations estimating the proportion of people that experience the effect at up to 1 in 200 [Ramachandran & Hubbard, 2001]. The types of synaesthesia are found to vary widely, with certain examples, such as the aforementioned grapheme-colour variant, being more common than others. Furthermore, researchers express the belief that the mechanisms of synaesthesia are linked with the phenomenon of sound symbolism, discussed in section 2.2 below. The physiology of the synaesthete may simply be more suited to a heightened cross-modal perceptual awareness when compared to that of the general populous. This suggests that we all have some capacity for some type of synaesthetic experience.
2.1.1 Exploration of Synaesthesia in Art and Science

Although research has undoubtedly verified the existence of synaesthesia as a fact of the sensory and perceptual systems, it does not follow that a universal model may be developed to utilise the effect. Nevertheless, attempts to employ the synaesthetic experience in such a manner have been made. For example, Louis-Bertrand Castel was aware of synaesthetic individuals when he undertook to invent the first colour organ as far back as the early eighteenth century [Marks, 1978: 83 - 86].

Interest in the subject, particularly in coloured hearing, grew steadily up until the 1940’s, with artists and scientists endeavouring to understand and/or exploit the phenomenon in a myriad of ways. Multimodal concerts were not uncommon, and while it is not suggested that all such performances were based on the synaesthetic experience per se, it is inferred that synaesthetes generated at least some of the interest in such light-music synthesis. A notable work is Alexander Scriabin’s ‘Prometheus’. The score for this composition for piano, voice, and orchestra also specified the use of a tastiera per luce or keyboard of light. This instrument, also called a clavier à lumières, produced an illumination with musical pitches mapped to colours as shown in figure 2.2.

![Scriabin's 'keyboard of light' mapping pitches of the piano keyboard to colours](image)

*Figure 2.2: Scriabin’s 'keyboard of light' mapping pitches of the piano keyboard to colours [Wikipedia, 2007].*

---

1 It is debatable whether Scriabin was a synaesthete himself; see Galeyev and Vanechkina- "Was Scriabin a Synaesthete?" at <http://prometheus.kai.ru/skriab_e.htm> Accessed: June, 2007.
The example of Scriabin’s colour-mapping keyboard points to one of the major failures of the attempts to utilise synaesthesia artistically. The choice of mapping from pitches to colours was completely arbitrary, reflecting the highly subjective nature of the synaesthetic experience across individuals. Synaesthetes rarely agree on the colour associated with a musical note. [Marks, 1978: 93].

2.1.2 More General Synaesthetic Trends

Despite this disparity, there is evidence of certain trends in the experiences of synaesthetes. Just as certain types of synaesthesia are found to be more common, specific examples of the synaesthetic experience occur more frequently than others. The reports of synaesthetes show certain trends that reveal underlying general principles [Marks, 1978: 98 - 99]. Two qualities that exhibit strong trend behaviour are the properties of brightness and size.

**Brightness** is an auditory descriptor that has an analogy in the visual medium. As a general rule, higher-frequency sounds are associated with more brilliantly-coloured light. It has been seen that different arbitrarily-assigned mapping systems may relate pitches to colours, such that there is no general consensus as to which colour represents which pitch. However, there is general agreement that higher pitches produce brighter colours. For example, if a particular individual associates the note C with the colour blue, then the C an octave above will be a brighter blue [Marks, 1978: 99].

**Size** is another such quantity and there is a broad feeling that size grows with descending pitch. At the same time, size is directly related to perceived loudness, as the secondary visuals generated by an auditory event are larger with increasing amplitude. The idea that the size property may be used to describe qualities in different sensory modalities will again be raised in section 2.2.2 during a discussion of sound symbolism.

Although the experience of synaesthesia suggests strong relationships between the visual and aural domains as discussed above, some researchers doubt whether the associations drawn by synaesthetes can be useful or practically applied. The subjective nature of these
experiences across individuals does not appear to provide a coherent structure for developing trans-modal parameter mappings in musical instruments, for example [Jones & Neville, 2005].

2.2 Sound Symbolism

A more general theory that may unlock the mechanisms behind the synaesthetic experience and go further to explore the relationships between different senses is that of sound symbolism.

The following discussion draws considerably on the work of Lawrence E. Marks, author of one of the most definitive works on the subject of cross-modal perception and metaphor among the senses [Kogan, 1991]. In his book The Unity of the Senses Marks examines phonetic symbolism, or the application of sound symbolism to speech sounds. This field has been described as the study of “(the) apparent appropriateness of the sound-structures of many individual words for their meanings” [Allott, 1995].

Such relationships between word sounds and their meanings may be seen on a number of levels [Marks, 1978]. At the topmost level, the choices of words used to describe a sensory experience are governed by the rules of a particular language, which may be seen as a learned set of arbitrary associations. For example, the object we understand as a house is referred to by the word we have come to associate with it. In addition, words may have onomatopoeic connections with the sounds they produce when spoken; for instance in English, the word “buzz” mimics the sound that it represents.

Beyond this however, it may be found that the actual sounds of words express something about a quality of the referent that is normally associated with another sense altogether. Such a sound may be understood on a perceptual level and the character of this experience may have an analogy in a sense other than the auditory. This is an example of cross-modality, in which the qualities of an experience in one sensory mode may be ascribed to those of another. For example, there is experimental evidence that the sounds of particular words may be associated with the characteristic forms of diagrammatic shapes, as will be shown in section 2.2.1.
In the more general case, it is highly significant that the sounds giving rise to cross-modal experiences need not even have been produced by the voice. When reviewing the results of relevant studies, Marks notes that: “At this level, it is important to bear in mind, speech sounds arose perceptions that, in some ways, behave like perceptions aroused by non-speech sounds” [Marks, 1978: 77].

The suggestion is that the psychoacoustic processes that illicit human response in one sense, may have analogous processes in another sense that illicit the same, or similar responses. These processes may have an inherent structure that is not arbitrarily assigned, and so may be seen as somewhat universal among humans.

2.2.1 Sound Symbolism Using Simple Shapes

The branch of psychology called Gestalt theory provides some insights into cross-modal perceptual experiences. The origins of modern Gestalt theory lie at the beginning of the twentieth century in a reaction to the more traditional approaches to psychological study. Instead of viewing the brain as a set of component processes that each contributed a fixed amount to a machine-like operation, a more holistic approach was deemed necessary to more fully understand the mind. The word gestalt may be translated as whole form and the approach cast aside a rigorous application of scientific reductionism in favour of an all-encompassing theory of inter-relationships. In a landmark essay, one of the pioneers of the discipline offers a definition of the approach: “The fundamental 'formula' of Gestalt theory might be expressed in this way. There are wholes, the behaviour of which is not determined by that of their individual elements, but where the part-processes are themselves determined by the intrinsic nature of the whole. It is the hope of Gestalt theory to determine the nature of such wholes” [Wertheimer, 1924].

The work of Gestalt theorist Wolfgang Kohler gives substantiated illustration of some of the theory that specifically examines the perception of form. The work also serves to verify the existence of relationships between the sensory modalities, such as those suggested in the discussion of synaesthesia. Kohler’s experiment asked subjects to relate each of the invented words ‘maluma’ and ‘takete’ to either of a pair of graphical forms as
shown in figure 2.3. The results show overwhelmingly that ‘maluma’ was attributed to the rounder shape on the left-hand side and ‘takete’ to the more angular shape on the right-hand side. Refined versions of the test have produced similar results [Holland & Wertheimer, 1964], significantly even among children with different native tongues [Davis, 1961]

![Shapes from Kohler experiment](image)

*Figure 2.3: Shapes similar to those used in the Kohler experiment. The graphic on the left was almost universally accepted as representing the word ‘maluma’ while that on the right was named ‘takete’.*

Interestingly, Marks even notes that his own children, when asked to draw shapes to represent the sounds of the same concocted words, produced graphics with remarkable similarity to the original shape choices of Kohler’s test [Marks, 1978: 77].

### 2.2.2 Correlations between the Attributes of Sounds and Shapes

An analysis of both the word sounds and the shapes of the Kohler experiment yield attributes that may be used to describe each of them [Marks, 1978: 77]. Tables 2-1 and 2-2 below gather these descriptors under three clear headings.

Correlations may be drawn between the aspects of sound and form shown in the headings of tables 2.1 and 2.2. The quality of *duration* describes the extent of the word sounds in the temporal dimension e.g. ‘maluma’ takes longer to say than ‘takete’. In the case of the graphical objects, their spatial extent is described by the *area* attribute. Both of these properties may be seen as a measure of *size* for each of the sounds or shapes. Such a common descriptor effectively conveys the similarities and differences between both
the pseudo-words and the graphical figures: The sound of the word ‘maluma’ may be said to be bigger than that of the sound of ‘takete.’ The maluma shape may be said to be bigger than the takete shape.

The idea that sounds can convey a meaning other than that denoted by language therefore suggests the existence of attributes that evoke similar perceptions from different sensory input. Indeed, it is very natural to describe sensory perceptions from one sense using qualities usually associated with another—as noted earlier, we may refer to sharp cheese or a soft sound.

<table>
<thead>
<tr>
<th>Word</th>
<th>Duration</th>
<th>Vowel Sounds</th>
<th>Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘Maluma’</td>
<td>Longer</td>
<td>Low frequency</td>
<td>Sibilant</td>
</tr>
<tr>
<td>‘Takete’</td>
<td>Shorter</td>
<td>High frequency</td>
<td>Sharp</td>
</tr>
</tbody>
</table>

Table 2-1: Attributes of the word choices from the Kohler experiment.

<table>
<thead>
<tr>
<th>Shape</th>
<th>Area</th>
<th>Local curvature</th>
<th>Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Larger</td>
<td>Gradual</td>
<td>Rounded</td>
</tr>
<tr>
<td></td>
<td>Smaller</td>
<td>Abrupt</td>
<td>Spiked</td>
</tr>
</tbody>
</table>

Table 2-2: Attributes of the shape choices from the Kohler experiment.

Empirical data supports the suggestion that there is a strong correlation between a vowels’ perceived frequencies and the imagined sizes of the objects represented by the sounds [Marks, 1978: 79]. The trend observed suggests that an increase in perceived pitch reflects a decrease in the size attribute. Furthermore, there is a link between this expected object size and the imagined size of the vowel sound itself. The sound can
actually be understood as having a spatial extent of its own. A sound with a higher pitch insinuates a sound object of smaller dimensions.

In a similar fashion, the quality of brightness may also be equated to the frequency of a vowel’s second formant. In this case, the perceived brightness of a sound is directly related to the perceived pitch content—the higher the pitch, the more brilliant the sound.

It has been shown that the qualities of perceptual experiences in multiple modalities can lead to common descriptors that go some way towards unifying such experiences. A study of subjects’ descriptions of words and word-fragments from languages unfamiliar to them yields a further notable result [Mc Murray, 1960]. The perceived pitches of vowel sounds do not necessarily generate unique descriptors and the results gathered in table 2.3 show just how many words may be used to qualify such a sound. It seems that there may be more than one way of describing the quality of a perceptive experience. This plurisignificance, as it has been termed, has ramifications for any attempt to generate a universal model that describes experience across modalities.

Although much of the research discussed thus far is specific to sound symbolism in the case of speech, it has already been noted that the actual mechanism of sound production is not necessarily important. While it has been suggested that the act of vocalisation affects the perceptive process, it is the sound itself that should be considered the main factor. Supporting evidence comes from the example of deaf individuals who have learned to speak [Johnson et al, 1964]. Whereas such individuals may have learned to form the mouth shapes and perform the necessary actions to produce the correct sounds, they do not exhibit the same symbolic sound relationships as hearing individuals do. So it is the actual sound itself which generates perceptual experiences whose qualities may be transferred to, or described in terms of, other senses.
Table 2.3: Examples of words used to describe vowel sounds of different apparent pitch.

<table>
<thead>
<tr>
<th>Apparent Vowel Pitch</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>small</td>
</tr>
<tr>
<td></td>
<td>sharp</td>
</tr>
<tr>
<td></td>
<td>bright</td>
</tr>
<tr>
<td></td>
<td>quick</td>
</tr>
<tr>
<td></td>
<td>active</td>
</tr>
<tr>
<td>Low</td>
<td>large</td>
</tr>
<tr>
<td></td>
<td>dull</td>
</tr>
<tr>
<td></td>
<td>dark</td>
</tr>
<tr>
<td></td>
<td>slow</td>
</tr>
<tr>
<td></td>
<td>passive</td>
</tr>
</tbody>
</table>

2.3 Cross-Modal Associations based on Physical Models and Experimental Results

While research has shown that most people do have a capacity for at least some synaesthetic experience as previously mentioned, and sound symbolism provides a strong basis for generating mappings based on more widely appreciated cross-modal links, these approaches remain largely subjective. Other avenues have been explored in an effort to provide consistent parameter mapping strategies that are universally effective. Two such approaches are the use of physically modelled systems, and data-gathering experiments; these are discussed below.
2.3.1 Real-World Models

Common experience of the physical world may be used to produce some rudimentary mappings that link the qualities of graphical objects to some of the most basic musical parameters: - pitch, perceived loudness and timbre [Jones & Neville, 2005].

The pitch of a sound that is associated with a given visual object may be seen as inversely related to its size. This supports the earlier findings of sound symbolism, when it was the apparent frequency of vowel sounds that gave rise to a feeling of spatial extent that could in turn be related in a graphical medium. In this case however, the association of size across the modalities stems from the common observation of the sounds produced by resonating real-world objects. For example, the pitch of a smaller bell is higher than that of a larger one when struck.

The amplitude or perceived loudness of a sound may be associated with the brightness of an image, as each represents the strength of the stimulus in either medium. The intensity of a sound may be related to that of an image.

Jones and Neville note the difficulty in ascribing a suitable descriptor to the aspect of musical timbre. The timbre of a sound is a more difficult quality to effectively link with a visual dimension, but there is a measurable physical quantity that may allow such a link. The curvature of a graphical shape or form may be related to the brightness of sound. Once again, this concurs with the aforementioned work of Kohler that also correlated the brightness of a timbre to the curvature of a shape. The real-world basis for such a relationship lies in the fact that the frequency content of the sound is mapped to the frequency of curvature of form for the visual object. With an increase in the frequency of curvature, the shape appears sharper. At the same time, as more high frequency harmonics are included, the sound appears brighter.

2.3.2 Empirical Study

Another approach to unearthing more universally-appreciated cross-modal associations is to gather the responses of subjects in experimental conditions to ascertain which are the
most common [Giannakis, 2006]. The work of Giannakis attempts to address the lack of empirical data in this area by using computer-based tests to collect trend data for auditory-visual mappings. The *Sound Mosaics* software utilised in this study is shown in figure 2.4. It allows a choice of graphical textures to be associated with different sounds presented to the test subject. The graphics deemed to best represent the audio sequences are dragged to the empty blocks at the top of the frame. The subjects’ choices are continuously recorded, allowing the preferences of users towards certain audio-visual links to be analysed from the resulting experimental data.

*Figure 2.4: Prototype software used in the experiments of Giannakis. Subjects could drag images onto the empty display area at top after listening to a sound sequence. Images were alternating according to each visualisation framework and auditory dimension under investigation [Giannakis, 2006]*
Compared with the *Sound Mosaics* data, it is found that these sonogram-type graphs are not as intuitive to use for the non-musical test subjects. The study concludes that high-level auditory-visual mappings are better understood than any arbitrary associations made between discrete physical attributes of sounds and graphics. As a first foray into this type of analysis, the study is necessarily limited to a small sample set of subjects and the testing of pitch, loudness and steady-state timbre. Nevertheless, the results obtained are an initial step towards a greater understanding of the linked perceptions of different sensory inputs.

<table>
<thead>
<tr>
<th>Auditory</th>
<th>Sound mosaics</th>
<th>Frequency-domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pitch height</td>
<td>Color brightness</td>
<td>Line height</td>
</tr>
<tr>
<td>Loudness</td>
<td>Color saturation</td>
<td>Color brightness</td>
</tr>
<tr>
<td>Sharpness</td>
<td>Texture coarseness</td>
<td>Line addition</td>
</tr>
<tr>
<td>Compactness</td>
<td>Texture granularity</td>
<td>Pixelation</td>
</tr>
<tr>
<td>Sensory dissonance</td>
<td>Texture repetitiveness</td>
<td>Line density</td>
</tr>
</tbody>
</table>

*Figure 2.5: The target auditory-visual mappings for the Sound Mosaics and frequency-domain visualisation frameworks in the Giannakis experiments [Giannakis, 2006].*

More traditional frequency-domain graphical representations are also tested. The contrast between the attributes mapped from the auditory to the visual domains for both the *Sound Mosaics* graphics and the sonogram-type displays are shown in figure 2.5.

### 2.4 Conclusions

This chapter considered the phenomenon of cross-modal perceptual experiences—the human trait allowing the features of observations made through one sensory modality to be expressed in terms of another. Three broad headings attempted to explore some of the underlying principles involved: the experiences of synaesthetic individuals; insights gained from the study of sound symbolism and Gestalt psychology and recent approaches to more generalised formulae for generating inter-sensory link frameworks. The emphasis was principally on the relationships that could be drawn between the realms of sound and vision.

In the case of synaesthesia, it was shown that this was a very real phenomenon that caused certain individuals to experience spontaneous intrusions of the qualities of one
sense into another. While being quite subjective in nature, it was found that certain more general trends could be seen that linked groups of synaesthetes together. For example, in the case of the common variant known as coloured-hearing, it is notable that while these individuals do not generally agree on discrete colour-to-pitch mappings, the broader association of brighter colours with increasing pitch is widely encountered. Historical attempts at employing synaesthetic relationships in creative endeavour were reviewed; the mappings employed in such works being arbitrary decisions of the composers.

The consideration of sound symbolism offered more general results with experimental evidence suggesting strong correlations between word-sounds and graphical shapes. Even more significant were the results of psychological studies that suggested that cross-modality could be an inherent human trait, making certain associations between sense experiences quite intuitive in nature.

The final section looked at two recent studies that attempted to find cross-modal links based on the emulation of common worldly experience and empirical data gathering. The first of these suggested that graphical objects could be associated with auditory events by physically modelling them on the behaviour of real-world objects. The size of a graphical shape could be inversely related to the pitch of its associated sound for example, mirroring the physical behaviour of resonating objects. It was notable that the suggested techniques mapped the qualities of visuals to musical parameters in sympathy with some of the results of sound symbolism. The reasons for this are not known, as the real-world framework formulated cross-modal correlations independently using very definite physical relationships. The last exploration of inter-sensory relationships examined experimental evidence for the user preferences towards certain auditory-visual mappings among non-musicians. High-level mapping of graphical qualities to musical parameters provides evidence that these types of associations are inherently more natural than traditional sonogram-type visual representations of sound.

Material reviewed in this chapter suggests that the comparisons, analogies and metaphors found in everyday language linking perceptual experiences between the senses could have a complex basis in human physiology, psychology and everyday life. Some such cross-modalities can be intuitive and universal, enriching the experience of the individual and enabling deeper expressive capabilities that are not based on learned conventions.
3 Interface Design Considerations

3.0 Introduction

This chapter explores some of the important factors for the design of a modern computer-based musical instrument interface. This is approached through a review of relevant literature and an examination of some previously implemented systems.

Consideration is first given to the subject of parameter mapping. This is the link between the input gesture and output sound of an instrument. It will be shown how computer-based instruments are inherently different to more traditional ones in this respect, and how different couplings between input and output may be implemented in contemporary electronic devices. The contemplation of mapping schemes leads on to a reflection on the concept of expressivity, as any mapping scheme may have a large effect on how the intentions of the performer may be effectively transmitted. A review of the literature on measuring how expressive an electronic interface may be is presented, giving some insight as to how the design of such a system should be approached.

The next section examines how the graphics' capabilities of the modern computer may be exploited to provide performance feedback. Examples of different approaches to graphic-based applications will be examined. Some of these are purely visual implementations while others offer control of musical parameters. The superposition of display and control surfaces will be addressed as an important design consideration.

The following section takes a look at computer vision techniques that forego the need for legacy devices such as the computer keyboard and mouse. An example of an interface using such input will be examined in the Camera Musicale.

An analysis of all these approaches is contained in each section followed by an outline of the approach to design that this project will take. Through this analysis and outline it is hoped that the inadequacies of these systems to address the design considerations of this work will be discussed.
3.1 Parameter Mapping

Most traditional instruments exhibit an inherent coupling between the gestures of the performer and the output sound. The energy and movement of the player are visibly and directly translated into an audible result, often with even the most subtle nuance of performance being expressively portrayed. This coupling is observed from two perspectives; that of the performer playing the instrument and that of the audience. The former affects the expressivity of an instrument and how intuitive it is to control. The latter helps to convey the performer's intentions through the direct association of particular gestures with musical result and supplemental performance gesture cues.

In the case of a violin, for example, the selection of pitches demonstrates a clear link between cause and effect: as the strings are shortened by the performer's hand movement, the sounds produced rise in pitch (figure 3.1). Also, the energy and technique of the performance may be seen to directly influence the properties of the music heard. For example, powerful quick bow strokes can create loud staccato notes while slower, less forceful gestures result in a quieter, legato feel. Such gestures can be mimicked by others or discovered first-hand relatively quickly, allowing particular musical effects to be elicited from the instrument. The tight input-output coupling makes the instrument more intuitive to play.

In the case of electronic and computer-based instruments, however, this natural connection between input gesture and output sound is not present. The means of sound
production of these instruments may be hidden within electronic circuits or computer code, and may not even be directly accessible by the performer. An example of this type of disparity between a traditional instrument and a modern music-generating device is that of a xylophone and a laptop-based modular synthesiser. The mallet striking a xylophone has a strong apparent mapping of action to sound, while the application of a time-varying envelope to a synthesiser sound produces a large dynamic change in timbre with only the press of a button. Figure 3.2 shows a photo of the computer-based group Kraftwerk in concert, whose static onstage stance offers little onstage insight as to which actions (if any) are producing which audible effects. For these artists, knowledge of the instruments' internal mappings facilitates performance. Others would find it more difficult to achieve the same sounds from the instruments without this knowledge. Unlike the case of the violin, there is no intuitive, natural input-output coupling.

Figure 3.2: Kraftwerk remain largely static in concert with little or no apparent link between performance gesture and musical output [http://perso.orange.fr].
3.1.1 Types of Mapping

The process of converting the input of the user through some gesture capture device to parameters that sound synthesisers can use is called mapping. The field is a diverse and involved one with much research being devoted to the subject. An excellent overview of the topic is provided by the annual proceedings of the conference on ‘New Interfaces for Musical Expression’ (NIME). One such paper gives a thorough review of the development of major ideas in the field [Hunt & Wanderley, 2002] and helps to give a good introduction to some basic concepts that will now be broadly examined.

For a modern electronic musical instrument, the input data is generally gathered from some gesture capture device such as a mouse, a keyboard or other such purpose built hardware. Different numbers of available input gesture parameters and of adjustable controllers offered by a synthesis engine give rise to three types of mapping arrangements. These techniques generally offer trade-offs between the interface’s potential for expressivity and its intuitiveness or ease of use [Rovan et al, 1997].

- **One-to-one Mapping**: The simplest form of mapping correlates each individual input stream to a single synthesis parameter. This provides the most intuitive interface for first-time users, but may inherently limit the instrument’s expressivity. An example is the typical use of hardware sliders to control a single synthesis parameter such as filter cutoff, resonance etc.

- **Divergent Mapping**: One gestural input is used to simultaneously control more than one musical parameter. This expands the potential for meaningful fine distinction in performance but may make playing the instrument less intuitive. This classification may be called a one-to-many mapping, an example being the finger position on a stringed instrument which controls base pitch and vibrato.

- **Convergent Mapping**: Many gestures are coupled to modify one musical parameter. This often results in a most expressive interface with good potential for highly nuanced performance. However, this generally comes at the expense of a steep learning curve, with more time required to achieve any level of mastery of the
A single-reed instrument exhibits such a many-to-one mapping scheme in the combination of lip pressure and airflow in the control of loudness.

Mapping techniques may be further described as static or dynamic depending on the instrument’s ability to adapt to input over time. Furthermore, explicit mappings describe fixed, quantifiable links between the measurable quantities of input gestures and the various parameters of the output. Implicit mappings are more like general rules on how the mapping module should perform and do not usually provide as transparent a link between input and output.

The results of applying these schemes have been explored in numerous papers [Hunt & Wanderley, 2002]. Notable developments include the use of various interpolation schemes to assure continuity of control and methods for allowing manageable interaction with the large number of modifiable controls that may be associated with a given synthesis method.

### 3.1.2 Multi-layered Mapping

Because of the flexibility of software-driven computers and some electronic systems, data does not necessarily have to be directed straight from the gesture capture device to the synthesis engine. Input streams may first be mapped to a collection of intermediate parameters, before being mapped onwards to the synthesiser controls. In fact, there is no inherent limit as to how many mapping stages the input data can go through before reaching the sound generator. Each of these layers of mapping will analyse and/or transform the data in some way, with their number and performance being constrained by considerations of processor power and any undesirable time-lags introduced by each process.

One implementation uses intermediate mapping layers based on abstract or perceptual connotations of the input streams and output engine parameters [Arfib et al, 2002]. Figure 3.3 shows the scheme suggested. A gesture capture device such as a mouse, machine vision system or any other hardware transducer sends a stream of data that is interpreted in terms of the input gesture. This related-to-gesture data is then mapped to more general perceptual measures. These may then be transformed into parameters...
describing perceptual aspects of the output mode, before in turn being mapped to the actual synthesis engine controllers.

![Figure 3.3: One proposed method of implementing multiple mapping layers [Arfib et al, 2002: 128]](image)

### 3.2 Musical Expression

The ultimate aim of any mapping scheme must be the provision of a suitably expressive interface. But what is musical expression and how can it be measured?

Expression has been described as 'a clear, felicitous indication of mood or intent' [Dobrian & Koppelman, 2006]. Performance expression is further described as the inclusion of material by the interpreter of a piece that is supplemental to the direction contained in the original score. If an audience is to experience the intended meaning of these nuances in performance, it must speak the same language of expression as the artist. That is, the desired effect of a certain expressive device must be understood by both parties.

Performers commonly use variations to the sound of individual notes and larger-scale phrase shaping to impose their interpretation on a piece. An example of the former is subtle variation of the amplitude envelope of a note or chord, while the latter is exemplified by the executed levels of a crescendo section.

#### 3.2.1 Measuring Expressivity of Electronic Instruments

A machine may not be defined as expressive in itself as it does not have anything to express. However, an effective instrument may serve as a conduit for expression by
accurately transmitting the intentions of the performer. A truly successful instrument will produce results that are very close to the user’s goal before performance. Some researchers have defined the expressivity of an interface in terms of the transparency, to both the performer and the audience, of the aforementioned mapping between gesture and aural result [Fels, Gadd & Mulder, 2002]. This may be accomplished by either the simplistic one-to-one mapping schemes described previously or by sophisticated interpretation of input by the system. To produce high levels of expressivity the former method may be insufficient, while the latter requires more of the artist.

The number of control streams offered by an instrument is not necessarily an accurate measure of its expressive capability. Adequate control is a prerequisite of an expressive instrument, however. It is the combination and organisation of the control input in the context of the instrument that enables expression [Dobrian & Koppelman, 2006].

An expressive instrument has three qualities as defined by Dobrian and Koppelman

- Accurate capture of gesture.
- A suitable level of transparency between gesture and output.
- Appropriate mappings that enhance expressivity.

An instrument that affords the depth of expression characterised by virtuosic mastery may use one-to-many mapping or other sophisticated input interpretation techniques. This leads back to the idea that the machine is not in itself expressive, so the amount of control delegated to the device by such fly by wire methods must be carefully considered. For truly virtuosic performance the interface should allow the artist to learn to use the basic functions of the instrument at a subconscious level. The performer is then free to concentrate on decisions of nuance that go beyond a purely mechanical rendition of the score.

3.2.2 Supplemental Performance Gesture

Dobrian and Koppelman go on to note how performance gesture in itself may colour the perception of a piece. Syntony may be characterised in this context as an emotional response or understanding of the presented environment that goes beyond the literal. The authors cite the valuable example of the experience elicited by a hand-drawn line: the
viewer may recognise the quantifiable geometric qualities of the line, while also sympathising with the act of drawing that produced it [Dobrian & Koppelman, 2006]. Computer interfaces often lack these extra performance cues. As seen previously in the consideration of mapping, the link between gesture and aural event can be broken in modern electronic or computer instruments. However, the flexibility and power of the modern computer does offer ways around this, in the form of the computer’s display facilities.

3.3 Graphic Displays

The computer is a powerful tool in the development of new musical instruments for a number of reasons. The flexibility of the design process and the sheer computational power of modern machines facilitate experimentation with new instrument concepts using advanced mapping strategies to link performance gesture and output. The graphical user interface (GUI) also has the potential to provide valuable feedback to a performer [Couturier & Arfib, 2003]. Haptic interfaces, such as are found in most traditional musical instruments, provide a continuous, subconscious flow of feedback that aids play. Electronic instruments commonly use non–haptic interfaces where the performer is fed only the aural result of performance. Such feedback is retrospective in nature and as such affords no chance for revision. It would be a somewhat tedious experience to witness to a performance where the artist needs to find a particular note or sound on her instrument. Haptic feedback typically eliminates such trial and error in traditional instruments through muscle memory. Cues are provided as to the eventual sound that will be produced. An advantage of the use of computers is the provision for visual feedback using the machine’s GUI that can also provide such cues.

3.3.1 Graphic Based Musical Controllers

Three methods of visualising musical control are shown in figure 3.4: graphical score-type displays, onscreen controller representations and interactive widgets [Levin, 2000: 41]. The first of these necessitates familiarity with a learned language of musical representation. In figure 3.4(a), the pitches of notes are distributed along a conventional keyboard. A scrolling bar indicates the current play position, mimicking the way a performer reads through a musical score. The second method uses virtual emulation of
hardware devices with adjustable knobs and sliders, such as in figure 3.4(b), and are commonplace in contemporary consumer music production packages.

Perhaps more interesting are the third class of music visualisations known as ‘Interactive Widgets’. These take the form of objects or sprites whose behaviour is linked to the production of aural effects. In figure 3.4(c), for example, the coloured boxes represent sound objects. The boxes may be moved around the screen and the circle plays these objects when they fall within its circumference. Various combinations of chords can therefore be chosen by positioning the boxes and moving the circle. The example above requires the user to use pull down menus to assign sounds to each box in the form of samples.

Of the three types of graphical displays discussed above that offer musical control, both the score-type presentations and the emulations of hardware controllers have drawbacks. The former require familiarity with learned or cultural conventions and so have an inherent barrier-to-entry. The latter use computer screen space inefficiently, as objects are constantly displayed even when inactive. With mouse operation, these interfaces are limited as a single parameter may only be modified at a time. Such discrete parameter entry does not promote a sense of creative flow.
Figure 3.4 (a): The Nuendo sequencer package from Steinberg. An example of a score-type display interface [Steinberg, 2007].

Figure 3.4 (b): The Kompakt software sampler from Native Instruments. An example of an emulated hardware interface using graphic controls [Native Instruments, 2007].

Figure 3.4 (c): The Stocksynth synthesiser form IXI software. An example of a musical interface employing interactive widgets for control purposes [IXI Software, 2007]
Another example of a *widget-type* implementation is the *Stretchable Music* or *Stretchables* software developed by Pete Rice and illustrated in figure 3.5 [Rice, 1998]. This allows users to modify the output of a pre-determined score by way of the manipulation of graphical widget objects. In the implementation shown in the figure, a scanning laser technique provides the gesture capture mechanism. Objects may be moved or stretched, as the name suggests, resulting in corresponding changes in the timbre of generated sounds. The broader music structure is pre-composed however, highlighting a weakness in a large proportion of graphics based musical controllers. The reliance on pre-recorded material that may be ‘tweaked’ by the user sometimes reduces these types of interfaces to interactive compositional pieces rather than instruments in their own right [Levin, 2000, pg 41].

*Figure 3.5: An implementation of Pete Rice’s Stretchables using a scanning laser interface. The graphics may be move and distorted to effect change in the music.*

*Stretchables* does offer a greater malleability of its graphical shapes by way of a scanning laser interface. Manipulation of the widget graphics cause coherent correlations with the timbre of the produced audio. The deficiencies of this system relate to the performance opportunities, which are finite and quickly exhaustible. This is because the reliance on pre-composed musical material means that they may only be modified in real-time to a limited extent.
Widget graphics offer scope for more inventive methods of musical control, but in some simpler implementations the manipulation of the onscreen objects has only an indirect effect on musical output, as previously seen. As with the previous two types of interface, the control offered operates on discrete parameters rather than providing a flowing, continuous correlation between input gesture and musical result.

### 3.3.2 Mouse-Based Graphics Applications

Levin offers another insight into how graphical applications can afford another type of musical control in a review of several examples that allow the production of visuals from input gesture. [Levin, 2000: 46 - 52] The resulting displays may be viewed as visual music in its silent form, as they allow the composition of visual imagery that is as temporally complex as aural music.

Such interfaces are not restricted to the kinds of discrete parameter-entry tasks such as those discussed in section 3.3.1. Most offer an interface that allows the flowing input of freehand gesture. Works by Paul Haeberli and Scott Snibbe, among others, all allow hand input that is converted to shapes in different ways. Examples of these systems are illustrated in figure 3.6.

What is most interesting here is the character imparted to the different systems using various underlying principles. The mechanisms of the various systems are very different, but what they have in common is that each has a personality that raises it above other mouse coordinate capture and graphics rendering applications. For example, Haeberli’s *Dynadraw* shown in figure 3.6(a) models a springy physical system to smooth out the input hand drawings, resulting in a very pleasing freehand experience. Similarly, Snibbe’s *Lazyline* instrument (part of his *Dynamics Systems Series*) applies a user-variable filter to the input line gestures. The amount of curve smoothing or exaggeration depends on the nature of the filtering and it is the interaction with this smoothing function that the user becomes conscious of.
As discussed above, mouse-based applications for graphic generation offer much freer gesture-entry methods than the previous examples of musical controllers. The addition of characteristic behaviour such as line smoothing implemented in software enhances the experience of the user and makes the program more fun to use. A disadvantage is the
use of the computer mouse, but this is not nearly as intrusive in a drawing package as in the musical control software.

Another use of the computer’s graphic capabilities is as a combined compositional aid and visual accompaniment to music [Abbado, 1988]. Some of the graphics generated by Adriano Abbado during the composition of his Dynamics work are illustrated in figure 3.7. In the development of this timbre-based piece, Abbado investigates correlations between the perceived natures of sounds and graphics such as those discussed in Chapter 2. Sonic events are perceived as objects and are related to discrete visual objects occurring at the same time.

![Figure 3.7: Graphics generated as part of Abbado's Dynamics Composition. The different shapes represent sounds of different timbres](image)

The basis for Abbado’s piece was an exploration of the concept of a melody of timbres, previously suggested by composer Arnold Schonberg.1 Abbado describes how the visualisations aid in the management of new timbres, such as those facilitated by modern synthesis methods, by graphically representing them in a meaningful way. The shapes of visual objects are related to timbre using intuitive links, while other graphical qualities such as spatial localisation and intensity are also useful compositional techniques with representations in the aural realm.

On the use of spatial distribution as a compositional tool, it is noted how the localisation techniques of the ear do not match those of the eyes in precision. As such, the size of a visual object can be related to the apparent spread of an aural event. This may compete with other relationships drawn between graphical object size and musical parameters.

---

1 Schonberg viewed pitch as just a single dimension of the larger timbre space, sympathizing with the non-Western musical tradition which more often used the manipulation of timbre as a compositional tool [Abbado, 1988].
such as pitch, but can ultimately provide contributing components of a convergent mapping scheme as described in section 3.1.1.

The application of graphics in *Dynamics* has some good points. The idea that graphical shapes can help in the management of new timbres by associating the graphics with the sounds is an interesting one. Although Abbado’s approach admittedly uses subjective judgments when equating parameters of sound and vision, some useful relationships are expressed that may be more general in nature. The use of more a jagged shape to represent a brighter sound, as shown in figure 3.7, echoes the ideas of the Kohler experiment outlined in Chapter 2. The playback of the visuals in concert with the finished musical piece also opens up possibilities for multi-modal-style performances. It is notable that the graphics application used is not a type of real-time controller, however.

All these graphics-based applications suggest methods for visual feedback that aid musical composition and/or performance, and they can provide an engaging interactive experience as well as entertaining visual accompaniment. However, they are all united by one limitation: they employ the native computer interface of mouse, keyboard and computer screen. With the possibilities made available for capturing gesture data and the use of mapping strategies outlined earlier, this is seen as something of a missed opportunity.

### 3.3.3 Combining Control and Display Surfaces

As has been described, the modern computer offers visual feedback possibilities that promote expressive interface possibilities. The use of the mouse to manipulate such displays is inherently limiting however, as only a single object may be adjusted at a given time. This can result in a dependence on the triggering of pre-arranged sequences and the use of automation, provoking a break in the link between performance gesture and result and effectively nullifying the advantages of the GUI [Couturier & Arfib, 2003].
A more effective technique is the superposition of the control area and feedback display and the simultaneous use of multiple control parameters on a single interface as shown in Figure 3.8.

Figure 3.8: An example of a combined display/ control interface. This version allows direct manipulation of the shape of a virtual string used to create sound by scanned synthesis [Couturier & Arfib, 2003: 187].

This interface eliminates the need for expensive touch-screen technology by utilising data gloves with built in switches and 3D sensors. The switches on the fingertips activate when the screen is touched, so the system defines a coordinate system relative to the plane of the screen. The system uses a patch in Max/MSP to manage the multiple control regions used in the system, while other patches provide the sound synthesis engines. An emulation of a photosonic synthesizer is one implementation incorporated into the prototype interface. A virtual light source, sensor and filter arrangement generates audio depending on their relative positions, properties and orientations. The virtual objects may be freely moved around the display area to vary the output sound. Another version of the same setup uses a scanned synthesis patch to provide a virtual string that may be manipulated using the data glove. This provides a strong intuitive link between the physical appearance of the oscillating string and the synthesized sound. The interface allows the string to be struck at different points to produce a corresponding oscillation, with various sliders and an angular frequency control allowing further manipulation of pitch and timbre.
The superposition of display and control surfaces in this example offers more possibilities than a mouse-based system. The use of such a virtual controller object arrangement offers a number of advantages:

- Objects may be manipulated directly on the display surface providing transparent and intuitive control and feedback.
- The constraints of the real world need not apply; objects can have flexible properties and qualities that can in turn be used as control data. Interactions may be defined within the system that would not be physically possible otherwise.
- The display may be augmented with supplemental information.

The combination of display and control on a single screen engages the user and focuses attention, promoting the aforementioned experience of immersive creative flow. Although the use of sliders and fixed controls is somewhat limiting, the use of a physically-modelled sound generating system has appeal. Because such a system is seen to oscillate at haptic rates, it is well suited to the type of hands-on control used. The appearance of such a system also provides intuitive visual feedback during performance corresponding with the sounds being produced.

The touch screen implementation has some limitations, however. The inherent latency of the location detection system would generally be unacceptable for the performance of musical material. The sensor arrangement also experiences some electromagnetic interference with CRT and LCD displays. The need for fingertip switches and sensors on the hands is also somewhat cumbersome.

### 3.4 Computer Vision as a Gesture Capture Mechanism

Another method of inputting a stream of data for control purposes lies in computer vision. This technique is less intrusive than the use of data gloves or other capture devices attached to the user.

A thorough consideration of computer vision theory is beyond the scope of this paper, but Golan Levin gives an excellent overview of the subject with an emphasis on the practical implementation of the technique from the viewpoint of modern artists and designers [Levin, 2004]. The review also gives examples of very different
implementations of the method in installation scenarios, from Myron Krueger’s first use of computer vision in *Videoplace* to some of Levin’s own recent work with various collaborators. For now, however, it is necessary to very briefly assert some of the requirements that make a modern computer vision system practical to implement.

### 3.4.1 Computer Vision Techniques

Various tried-and-tested techniques now exist that may be implemented on robust platforms without the need for bespoke software development. *Motion detection* (or *frame differencing*), *background subtraction*, *brightness thresholding* and other well documented techniques allow the novice programmer to utilise computer vision in artistic projects [Levin, 2006]. Such techniques are not entirely new, but the ever increasing power of computers and the evolution of software environments certainly make them more accessible.

Informed design decisions can simplify the realisation of a robust machine vision system. In particular, the demands on signal processing can be reduced by appropriate modification of the physical environment. For example using a two-dimensional field of view on a high-contrast background and employing a non-tracking method of contour detection alleviates the need for calibrated lighting setups and simplifies the processing algorithms required. Considered choice of hardware and software components also plays a part.

An advantage of the computer vision techniques discussed above is the lack of any requirement for additional components such as data gloves. These techniques come with their own considerations however, with the selection of suitable hardware and software being one. Another is the need for adequate lighting conditions and well controlled environmental conditions. A sufficiently powerful computer processor is also required.

### 3.4.2 The Camera Musicale

An installation that has employed computer vision for musical control for a number of years is the *Camera Musicale*. 
The Camera Musicale is a control interface for a sound synthesiser that utilises a video camera to capture the hand- and finger-movements of the user. The first incarnation of the system was used throughout the eighties to generate music for choreographed performances, before being developed further by Remus, Aubin and Bure Soh in 1992 [Remus, 2006]. This implementation allowed an audience to control the system, activating large, mechanical musical devices by waving their hands in the air. In subsequent years, the Camera Musicale system continuously evolved as the creators learned from the experiences of different user audiences.

The ‘modern’ Camera Musicale setup consists of the basic structure shown in figure 3.9, comprising the following readily available components:

- A black & white camera, with infrared (IR)-pass filter.
- An IR light source.
- A computer with a video capture card.
- Max/MSP environment to perform signal analysis on camera data and to control mapping of data from camera to musical instrument.
- A software synthesiser or other audio generator.
- Display screens for visual feedback displaying any control regions used
The camera captures the scene in IR only because of the attached filter. The analysis of the camera scene by the computer system yields a measurable gesture \textit{spot} as graphically represented in figure 3.10. The grey spot area corresponds to a high contrast part of the camera scene such as a hand or finger held in mid-air. Movement relative to the reference point generates X and Y position data. The extents of the spot are labelled \( dX \) and \( dY \) for their dimensions in the X and Y directions. Calculus is used to determine the rates of change of these quantities in time to yield speed and acceleration magnitudes. In addition to these accessible control parameters, a spot may be assigned different areas of effectiveness so that its function changes depending on which area of the scene it is located in. Multiple independent spots may also be implemented.
Independent controllable parameters are limited to between two and four, just as in the case of the majority of traditional instruments. Control of pitch, timbre, intensity, articulation and rhythm (or succession of events) has been found to be sufficient for expressive play and more than this does not improve performance regardless of the experience of the user [Remus, 2006].

Remus gives three examples of the implementations of parameter mappings as modes of play used in the Camera Musicale:

- Continuous position data from the X- and Y- axes are used to control pitch and intensity respectively. The dY parameter controls attack time, which is effectively a timbre modifier.
- Small regions in space are represented on the screen and touching these triggers different sounds. The speed at which the hand moves from one point to the next provides another parameter to effect a characteristic of the triggered sound.
- Pre-written sequences of notes are operated on, with measurements such as X position controlling speed of playback.
The body language of the performer can be considered as a point of interest for the audience, as precision and self-control is needed due to lack of tactile feedback. This supports the findings of earlier research discussed in previous sections that recommends the use of additional performance cues in expressive interfaces.

Although mappings are flexible, users tend to gravitate towards certain modes of expression, even when limited to less than 4 streams of data:

- Use of a drumming style of triggering sounds by regions.
- A relationship between intensity of movement and musical output, often on the Y-axis.
- Piano-style X-axis distribution of low to high frequency sounds.

Remus provides empirical data that summarises the frequency of use of different parameters based on these three basic playing modes. This identified four mapping types: *Orchestral*, *Theremin*, *Instrument* and *Direct Audio*. Trends show that the first two types make more frequent use of imaginative control-to-musical-parameter links, while the latter favour simpler movements [Remus, 2006: 252].

The setup was deemed most effective if used with pre-recorded sequences of some form, as this reduces the need for accurate note selection and progression. This suggests that the system is more effective if used as a type of musical effect or real time arranger, rather than as an instrument with low-level musical control.

The *Camera Musicale* is an example of a working synthesis controller, the technical implementation of which points to what may be achievable with a computer vision system. The free gesture input is just that, and the system has operated in public for many years, in various guises. The system falls down in that the visual feedback to the user is contained on display screens which show the active areas within the camera scene. The suggestion that it is more suited to arranging pre-recorded musical sequences also lessens its impact on the design of the system presented here.
3.5 Outline of Approach

This project develops a system that allows musical control of sound synthesis using graphics-based instruments implemented in an installation environment. Using rudimentary perceptual models such as those discussed in chapter 2, the system explores basic cross-modal associations that offer an engaging experience without a steep learning curve. The user is not required to have experience of music theory or other such learned conventions, but encouraged to explore more intuitive auditory-visual links. As such, the expressivity of the interface allows the accurate rendition of the intent of the user, but does not extend to providing an instrument capable of virtuosic play with lengthy study and practice. Nevertheless, the points raised in connection with the design of instruments destined for more advanced performance are taken into account in the instrument designs.

Computer vision facilitates gesture capture in the system. The gesture data is transformed through multiple mapping layers to the sound synthesis device. The use of intermediate mapping layers allows the generation of graphics for performance feedback to the user and provides a modular system arrangement. These visuals are superimposed on the control area to create an immersive experience.

The system is intended to be as engaging and as fun an experience as possible. The playfulness of the graphics applications discussed in section 3.3.2 is important to this project. The seemingly intelligent nature of these systems, achieved through the implemented rules of interaction of the virtual objects, changes the experience of using them. Instead of a set of discrete parameter entry tasks, the user experiences a continuous, flowing interaction with the interface.

The malleability of graphic shapes of Pete Rice’s Stretchables and their coherent correlations with the timbre of the produced audio is characteristic of the type of control in this project. Larger-scale musical structures are not adjustable, but rather the lower-level parameters such as pitch, amplitude and timbre. It is hoped that a balance is found between an instantly-knowable intuitively understood interface and one that offers good scope for expressive play, with a slight bias towards the former. The implementation of
the developed software into an installation setting provides an alternative to the traditional computer interface.

All of the systems reviewed above have various advantages and disadvantages in their treatment of interface design. Aspects of each of them will have relevance to certain parts of this project, but no one example of an interface system answers all the present design questions. The system described in this work uses a combination of the strongest points made above, resulting in instruments that address the areas that these implementations do not. The final system is thus a hybrid of the better ideas from each of these interfaces.

3.6 Conclusions

This chapter reviewed literature surrounding the design of computer based musical interfaces and some examples of previous implementations. The mapping of gesture input to output was considered and aspects of the expressivity of such an interface were investigated. Examples of graphics-based systems of various types were examined along with computer vision based methods of gesture capture. Analysis of the advantages of each topic of design was undertaken in each section and the failures or inadequacies of certain approaches were observed. This facilitated the formulation of the design approach undertaken in this project.
4. System Framework Design and Implementation

4.0 Introduction

This chapter presents the system framework which implements a graphics-based synthesis controller interface exploring the use of cross-modal associations such as those described in chapter 2. The system design is observant of considerations raised in chapter 3 on the design of audio-visual instruments.

The review of the literature identified several design features of the system:

- The superposition of display and control surfaces.
- The use of computer vision as a gesture capture mechanism.
- A flexible software environment that facilitates the use of multiple parameter mapping stages.

A functional description of this framework is first presented, followed by technical details of the hardware setup for the system. Details of several possible approaches to implementing the software system are included to highlight the advantages of the chosen framework.

4.1 Software System Framework Design

The software system contains the following functional units:

- Gesture capture.
- Gesture analysis and mapping.
- Sound synthesis.
- Graphics generation.

For portability of installation and because of the resources available, the software framework is contained in a single PC running the Windows XP operating system. The PC has Firewire capability and a graphics card capable of OpenGL hardware accelerated rendering.
4.1.1 The Max/MSP and Processing Environments

Max/MSP is a graphical environment for the creation of music, audio and multimedia works with a substantial development community [cycling74.com, 2007]. Available as a commercial product for over a decade, the software is more robust than some open-source environments assessed such as Pure Data. Furthermore, numerous third party contributors have enriched the functionality of the application by providing hundreds of objects and add-ons available in the form of patches, externals and function libraries. The gesture mapping and sound synthesis stages are implemented in Max/MSP.

![Max/MSP graphical environment](image)

Figure 4.1: The Max/MSP graphical environment.

The graphics generation functions of this project are carried out in Processing, which is an open-source integrated development environment (IDE) for the programming of image graphics, animation and sound [Fry & Reas, 2006]. The environment uses a streamlined version of the Java language to allow rapid prototyping of applications in the form of sketches. An active development community, substantial third-party function libraries and other resources such as example sketches make it suitable for the exploration of creative applications at a novice programming level. The environment supports OpenGL hardware-accelerated graphics, which lessens the computational overhead of the system’s graphics generation module.
4.1.2 Computer Vision System

Several software platforms now exist capable of performing the necessary signal processing tasks to facilitate computer vision [Levin, 2006]. An examination of several of these alternatives informed the initial system design framework for this project. The three systems evaluated were EyesWeb, Processing, and the reacTIVision framework.

The existence of previously documented problems implementing computer vision for gesture recognition in the EyesWeb environment was confirmed [Solon, 2006]. The latency associated with using the system as a gesture capture device made it unsuitable for the musical applications of this project. It is acknowledged that this may have been down to a lack of expertise, but the short project time-frame did not allow prolonged experimentation with the setup.

It was found infeasible to use Processing as a computer vision module. The WinVDIG drivers that handle video capture for the system and reliance on the QuickTime video format introduced high latency. The computer vision implemented in the environment is not as reliable as the platform’s well-developed graphics generation capabilities. Numerous bugs and technical issues arose which discounted the use of Processing as a viable computer vision module.
4.1.2.1 ReacTIVision

The application implemented in the project uses the reacTIVision system. ReacTIVision is described as an open-source, cross-platform computer-vision framework primarily designed for the construction of table-based tangible user interfaces (TUIs) [Kaltenbrunner & Bencina, 2007]. A standalone application tracks special marker tags called fiducials (see figure 4.3) that may be affixed to puck-like objects placed atop a transparent table surface. The tracking is of sufficiently low latency, and sufficiently robust, to provide a continuous stream of usable data for musical control. Figure 4.4 shows a particular implementation of the framework as realized by the system developers, which is also the approach to system design taken here.

Figure 4.3: Examples of the fiducial markers that are tracked by the reacTIVision computer vision system [http://mtg.upf.edu, 2007].

The system transmits data via Open Sound Control (OSC) messages on the UDP network transport layer. The unique requirements of tangible user interfaces are handled by a specialised TUIO protocol. OSC messages are a standardised communications method used in numerous musical applications such as Max/MSP and SuperCollider, as well as graphics-based development environments such as Processing and Flash. As a result, the reacTIVision system can interface reliably with these systems using a specific TUIO client. The system developers provide examples of these clients that can be effectively implemented in each host environment.
4.1.2.2 TUIO Simulator

An advantage of the reacTIVision framework is the provision of a table simulator in the form of a Java application. This is an onscreen emulation of the physical interface complete with control objects called *pucks*. Manipulation of the onscreen objects produces the same data output as the *reacTIVision* application and is recognised and decoded as such by the TUIO client in *Max/MSP*. This is an invaluable tool in the development and testing of software applications as it eliminates the need for a fully-functional physical setup.
4.1.3 Software System Overview

Figure 4.6 illustrates the design for the system using the software platforms described above. Gesture data is captured using the reacTIVision application and is sent to the TUIO client in Max/MSP where it is decoded into useful control messages and data signals. Mapping techniques implemented in Max/MSP transform the data from raw gesture input to perceptual quantities associated with gesture. These are in turn mapped to perceptual quantities in the auditory and visual realms. Sound synthesis is also carried out in Max/MSP while graphics generation is handled by Processing.

The TUIO clients can only receive on a single specified UDP port, and the reacTIVision and TUIO Simulator applications can also only broadcast on one port at any given time. This means that when the Max/MSP client is receiving data, a similar client cannot be
used simultaneously in Processing\(^1\). However, this limitation does not impact on the system design, as data can be sent to Processing from Max/MSP using the maxlink library developed by Jesse Kriss [Kriss, 2007]. When installed in the computer system, this library of java classes facilitates UDP communication between \texttt{mxj} object inlets and outlets in Max/MSP with ports specified using a special naming method in Processing. The library requires the existence of a network connection to run, so the Microsoft Loopback Connection is installed in the system. This negates the need for an actual physical link to another computer or network.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure4_6.png}
\caption{Overview of the project software system framework}
\end{figure}

### 4.2 Hardware Setup

Figure 4.7 illustrates the physical setup of the system implementation. The PC running the system software is not shown. Semi-opaque satin glass of dimensions 600x600 mm is used as the table surface. The minimum projection path to fill this area is typically about

\footnote{It was not possible to implement changes in the \texttt{TUIO Server} java class source code that would rectify this problem by allowing data broadcast to multiple ports [Kaltenbrunner, 2007: personal correspondence].}
1.2m depending on the projector model, so a mirror at a 45 degree angle is used to lengthen the light path.

An infrared (IR) light source and a camera with an attached infrared-pass filter facilitate the use of a combined display and control surface. The projector works in the visible spectrum, but this light is cut from the camera signal by the filter. An IR source illuminates the underside of the table so that the fiducial markers on the control pucks may be seen by the computer vision system. This arrangement serves to keep the projected graphics elements from confusing the computer vision system. It is necessary to diffuse the IR illumination in order to prevent the formation of a bright spot of IR light on the glass, as such bright spots obscure the camera’s view of any fiducial placed above it.

![Figure 4.7: The physical setup of the interface system. PC not shown.](image)

The camera must provide full frame, progressive scan output for the reacTIVision system to be able to recognise the fiducial markers. Attempts to use a PAL format DV camera were therefore unsuccessful as the interlaced signal destroyed the structure of the fiducials. A Unibrain Fire-i webcam is suitable, as it provides 30 frames per second (fps) output over a Firewire connection. The use of a wide angle lens on this camera enables the use of larger table areas.
IR Light Emitting Diodes (LEDs) are used to provide illumination of the camera scene. One such LED disk is shown in figure 4.8.

![Figure 4.8: Infrared LED Light Disc](image)

### 4.3 Conclusions

This chapter described the software and hardware frameworks for the interface system implemented in this project. Key functional factors of the software system were described based on the use of the reacTIVision computer vision framework. The use of the Max/MSP and Processing environments to provide gesture analysis, mapping, sound synthesis and graphics generation were explained. Technical details of the hardware setup were also presented.
5. Software Instrument Implementation

5.0 Introduction

This chapter outlines the realisation of three software instruments designed for use with the project hardware setup presented in section 4.2. Experimentation with different approaches to the implementation of mapping, sound and graphics generation took several directions during the course of this project. While it is not feasible to document all these approaches here, the examples presented serve as a broad outline of the progression of the project. The instruments increase in complexity and sophistication from first to last.

The first of these instruments, MaluTak, is a basic silent experiment in the generation of graphics. The second DynAudio instrument adds sound output. The final instrument, TwoHand, is that intended for gallery installation and as such is the most highly developed. However, all three examples do run on the hardware setup.

The instruments are based around the exploration of the types of cross-modal perceptual associations discussed in chapter 2. MaluTak allows the control of the shape quality of its graphical objects by gesture input. DynAudio maps gesture input to both the auditory and visual realms. The TwoHand system generates sound and visuals from gestures using multiple inter-sensory analogies in a complex mapping scheme.

Each of these instruments was developed using the TUIO Simulator but with reference to the hardware table setup. They are each first described in terms of their basic playing technique and output. Explanations of their chief functional units are then presented, followed by more in-depth accounts of their technical workings. Details of specific coding techniques important to the understanding of each instrument are described, but kept to a minimum. Full, commented code is available in Appendices A and B, while descriptions of all Max/MSP objects used and details of the Processing java methods are presented in Appendices C and D respectively. Max/MSP patch files and Processing sketches are included in the project support files on the attached compact disc.
5.0.1 Analysis of Design Considerations

Some general software design approaches are observed in all instruments: calibration and scaling capabilities allow enough flexibility to accommodate different hardware components without altering code functionality. Basic principles of encapsulation and modularity are respected. In Max/MSP, communication between sub-patches is implemented using patch cords rather than send and receive objects, resulting in a top-down data flow. Object-oriented techniques are observed in Processing to allow adaptability in the future, although these were not fully implemented in all sketches due to time constraints.

The approaches to interface design discussed in Chapter 3 are also observed in the implementations of the presented instruments.

The review of mapping strategies underlines the importance of suitable consideration of the link between input gesture and musical result. In particular, the use of multiple mapping layers provides a number of advantages:

- A modular arrangement that allows the interchange of controllers and/or synthesis models. This is useful for rapid prototyping and experimentation, as well as offering the possibility to exploit a user’s existing virtuosity with a particular instrument or interface.
- The facility for multiple feedback ‘taps’ at different positions in the chain, creating the opportunity for output in more than one medium. This opens up options for different feedback mechanisms to the player and simultaneous multimodal performance.

The focus on the use of perceptual models in intermediate mapping stages promotes finding common metaphors that link the audio and video realms. These metaphors allow the practical application of cross-modal cognitive theory such as that discussed in chapter 2.

The observations of Dobrian and Koppelman on the subject of expressivity are similarly relevant. Although originally targeting the design of concert-grade instruments capable of
highly nuanced play by a master, the principles are applicable to all musical interfaces. It is clearly established that expressivity is of central importance. The suggestion that the design of an instrument interface should also consider the impact of supplemental performance gesture is also noted: animated use of an instrument can be an aid to transparency between action and result and hence evoke an empathic response in an audience. However, returning to the example of the computer-based performances of Kraftwerk from section 3.1, it must be noted that the spectacle of seeing this seminal act in concert is not lessened by the lack of onstage antics. Furthermore, it is not always be necessary to require a wide range of extended techniques from a user that only indirectly affect musical output.

As discussed in section 3.6, the interface is designed to allow the non-musician to produce results in a casual gallery situation. Therefore the expressivities of the instruments do not stretch to the level that requires virtuosic mastery with the investment of prolonged practice.

5.1 MaluTak

5.1.1 Overview

MaluTak is a silent system used to test the communications between various modules and to experiment with graphics drawing techniques. The output of the system is shown in figure 5.1. Actions on the TUIO Simulator result in the addition, deletion and modification of graphical widget objects. Up to four control pucks (fiducial ID nos. 0, 4, 8, and 12) may be used simultaneously. The positions of the pucks are mapped to the x- and y- coordinates of the relevant widgets, and the angle of rotation of each control object affects the shape of the corresponding graphic.

The widgets are animated by continuously and randomly adjusting their sizes. The ranges of random values used are constrained so that the graphics maintain an average screen size while changing shape. A demonstration video of the software working on the computer desktop is included in the project support files under the video directory as MaluTak_Demo.avi.
5.1.2 Graphics Generation Using Bezier Curves

Each widget is drawn using multiple Bezier curves, and then colour filled. A Bezier curve (figure 5.2) is drawn between two specified anchor points, with the shape of the curve determined by two additional control points. The curve formed is actually a series of short straight-line segments that appears as a curve when the number of segments used is large.

*Processing* provides the ability to draw Bezier-type curve segments using the `bezier()` method. The resolution of the curves—the number of straight line segments that are contained between the endpoints—is controlled with the `bezierDetail()` method. Higher resolutions result in smoother curves, while lower ones produce visible straight line sections that connect at abrupt angles. Setting the `bezierDetail()` parameter to 1 results in a single line drawn between the endpoints. The default Java rendering engine does not use
this Bezier resolution information, so these graphics are rendered using the OpenGL graphics library. Figure 5.1 illustrates the shapes obtained for different curve resolution values. The more rounded shapes result from higher parameter values being sent to the bezierDetail() method.

5.1.3 Technical Overview

The MaluTak patch (figure 5.3) manages the input from TUIO_Client and sends UDP messages out to Processing via the maxlink mxj object. These messages contain the fiducial identification number of the control puck being updated, integer values for the x and y coordinates, and a float value for the rotation angle of the puck. When a puck is introduced to-, or removed from- the board, a data string is also sent containing an Add or Take control message. This message is used by Processing to increase or decrease the numbers of graphical objects being drawn.

![MaluTak Max/MSP patch](image)

*Figure 5.3: the MaluTak Max/MSP patch.*
A bug in the TUIO client was identified in the implementation of this patch: it was found that when updated rotation angle values were being sent, a continuous stream of x/y coordinate data was also transmitted, even if the position had not changed. This caused widgets to spontaneously jump to the last position transmitted when their rotation value was adjusted, as the new fiducial number was updated with old coordinate data. The Filter_Out_Reps_In_Input sub-patch shown in figure 5.3 rectifies this by only sending changes in coordinates.

Processing manages the graphical objects using a Puck class that defines x, y and angle data for each object as well as a method for drawing it to the screen. The Puck objects are instantiated and stored in an array as Add messages are received, and removed from the array when Take messages are received. A switch statement manages which array elements are updated using the fiducial identification numbers sent over maxlink.

5.2 DynAudio

5.2.1 Overview

The second instrument experiment takes inspiration from Paul Haeberli's Dynadraw graphics application discussed in section 3.3.2. DynAudio takes input in the form of stroke gestures and produces simultaneous graphics and sound output. The visuals reproduce a version of the entered stroke, shaped by a physically-modelled system. The sound generated has attributes linked to the nature of the stroke drawn: a more jagged stroke produces a brighter timbre.

The instrument is played by drawing a puck across the table controller in a painterly motion. Sound and vision are activated by the angle of rotation of the puck; turning it clockwise past a trigger point switches on the audio output and draws the visuals, while turning it anti-clockwise stops audio and clears all drawn strokes. An example of the graphical output is presented in figure 5.4
5.2.2 Gesture Analysis

The auditory and visual output of DynAudio is controlled by the rotation of the control puck. The puck rotation values received by the TUIO Client are scaled to integers in the range -49 and +50, with zero corresponding to the twelve o'clock position. Moving past this point generates a trigger message (a bang in Max/MSP) via an edge detection function. This in turn generates an activate or deactivate message depending on the direction of movement. These message strings are sent on to the synthesiser section in Max/MSP and to Processing over maxlink.
As shown in figure 5.5, the curvature of a stroke may be assessed from the angles between successive component straight line segments. Increasing the number of line segments results in closer approximations to the instantaneous curvature at any point on the curve. A Max/MSP sub-patch called *cosAngleCalculator* uses this principle to calculate an approximate measure of the stroke curvature. In this patch, the three most recent points from the stream of x/y input coordinates are used to generate a succession of output values. These values are the cosines of the angles between the line segments formed by the three points. The points are labelled X(n-1), X(n) and X(n+1); denoting the most recent, current and oldest points received respectively. The \( n \) subscript signifies the index of an array. As new values are received they are inserted at index (0), the older values being shifted one element along the array. This function is implemented in the sub-patch using the *bucket* object. The expression used in the calculations is shown in figure 5.6. The mean of the twelve most recent cosine values is then found to provide smoother control data. This control is also scaled to provide more musically relevant results in the variations of timbres produced- the range of cosine values of between -1 and 1 produce modulation indexes of between 0 and 50.
\[ \cos \text{ angle} = \frac{(a*b) + (a'*b')}{\sqrt{a^2 + a'^2} \times \sqrt{b^2 + b'^2}} \]

where:
- \(a = X(n-1) - X(n)\)
- \(a' = Y(n-1) - Y(n)\)
- \(b = X(n+1) - X(n)\)
- \(b' = Y(n+1) - Y(n)\)

*Figure 5.6 Expression implemented in Max/MSP that calculates the cosine of the angle between successive line segments.*

### 5.2.3 Sound Synthesis

The *DynAudio* instrument uses *frequency modulation* (FM) synthesis for sound generation. FM is a family of digital synthesis methods that use a non-linear oscillating function for wavetable lookup [Roads, 1996: 226]. Originally developed by John Chowning in the 1970's, the technique generates dynamic sound with time-varying spectral qualities in a computationally inexpensive manner. The most basic FM technique is called *simple FM* and uses an oscillator to modulate the frequency of an oscillating carrier signal, as shown in figure 5.7.

FM of two sinusoidal signals generates a series of sidebands around the carrier frequency. The number of sidebands generated is related to the amplitude of the modulating oscillator. In this specific case, the distance between each of the sidebands and the carrier is an integer multiple of the modulator frequency—i.e. the spectrum is *harmonic*. This occurs when the ratio of the modulator frequency to that of the carrier is a simple whole number relationship e.g. 1:4. When this \(M:C\) or *harmonicity ratio* is not a ratio of two integers, an *inharmonic* spectrum is observed in the FM output. The harmonicity ratio can therefore be used to control the amount of *dissonance or roughness* in the timbre of an FM-synthesised sound.
The number of sidebands generated in an FM signal is related to the modulation index, $I$. This is the frequency deviation of each sideband from the carrier frequency multiplied by the modulator frequency. Increasing $I$ therefore results in a brighter sound, as more partials are added to the carrier frequency. The schematic shown in figure 5.7 must be modified in order to be able to specify modulation index as an input parameter. Figure 5.8 shows how this is achieved in the Max/MSP simpleFM~ object. The input parameters are the carrier frequency, harmonicity ratio and modulation index. By applying different envelope arrangements to these inputs, as well as another amplitude envelope to the output, many different dynamic sounds may be generated. Research has yielded many methods for emulating real-world instrument sounds as well as ways to produce completely new timbres using various FM techniques [Roads, 1996; 226-250]. This broad timbral capacity comes at a relatively low computational cost compared to synthesis based on physical modelling and other methods.
Sound is triggered in Dyn.Audio by rotation of the drawing puck as previously described. To prevent clicks in the audio, the activate and deactivate messages trigger short attack- and release- envelope sections respectively.

The base pitch of the sound produced by the instrument is controlled via the carrier frequency value sent to the synthesiser. This is extracted from the y-coordinate of the control puck position. The values are scaled so that the full 480-pixel height of the control surface covers a frequency range of 110 Hz to 440Hz. Thus control spans the two octaves from A2 to A4.

The timbre of the sound produced by the instrument is controlled by the output of the cos.AngleCalculator sub-patch described in section 5.2.2. These values are sent to the modulation index of the FM synthesiser section. Higher values, produced from sharp direction changes at abrupt angles, result in more partials being generated in the audio output i.e. a brighter sound. Smoother strokes produce purer, duller tones.
5.2.4 Graphics

The Processing sketch used to generate graphics is based on Haeberli's Dynadraw [Processing.org, 2007]. Although the source code for Haeberli's original application is available [graficaobscura.com, 2007] the version presented here is not a strict port of this to the Processing environment. Instead, the DynAudio sketch is an attempt to incorporate gesture-augmentation techniques similar to those of that original application.

The activate and deactivate messages generated in Max/MSP are sent to Processing as string data, where they are used to set Boolean conditional variables. These variables determine if Processing executes the drawTrace() method to render the graphics. The graphical output is continuously refreshed to white at the application frame rate, so deactivation of the drawTrace() method effectively clears the visuals.

Figure 5.9: Graphical output of DynAudio for different values of the spring constant, k: (left) k = 0.1 and (right) k = 0.9.

A second method called updateTrace() continuously modifies the drawing position using the x/y coordinate data sent from Max/MSP. The virtual brush is modelled as a physical object with mass, velocity and friction, and the moving input data point 'pulls' the brush with a simulated spring. The simulation implements Hooke's law for the action of a spring under a force and computes the displacement of the line drawn from the actual x/y input. In effect, the program is applying a simple filter to the input coordinates. By manipulating the variables of the system, different graphical effects are achieved. For example, a high value of the spring constant, k, causes the resultant drawing to pitch wildly.
about any changes in direction of the input stroke- a lower value for \( k \) results in smoother drawing, as shown in figure 5.9.

### 5.3 TwoHand

#### 5.3.1 Overview

The *TwoHand* instrument uses two pucks for control and simultaneously generates sound and visuals. Manipulation of the two pucks on the control surface varies the synthesized sound output. Pitch is controlled by the distance between the two pucks: in the manner of traditional string instruments, longer lengths produce lower frequency sounds. The position and orientation of puck #1 controls certain aspects of timbre. Puck #2 controls sound output level and some other aspects of timbre. The slope of the line that joins the centres of the pucks also affects the evolution of timbre over time.

On the control/display surface, a series of shape elements are drawn in the space between the pucks. A new shape appears when the distance between them exceeds a multiple of a specified unit length. Each of the fully formed shapes therefore has a diameter equal to this unit length. Although the frequency output of the synthesiser is continuous, these shapes represent the lengths at which particular notes of the scale are produced. The last element adapts its shape to occupy the space between the highest-multiple unit length and the centre of puck #2. That is, it fills the gap between the fully formed shapes and the second control puck. An example of the graphical output of the *TwoHand* system is shown in figure 5.10. The positions of the control pucks (fiducial IDs 0 and 4) are shown by the red circles.
5.3.2 Gesture Analysis

The angle of rotation of puck #2 is used with a zero-crossover trigger similar to the method used in the DynAudio instrument presented in section 5.2. However, in TwoHand, the speed of rotation of the puck is measured before it crosses the trigger point moving in the clockwise direction. This data is then used to scale envelopes that in turn affect sound synthesis parameters and graphics. Capturing the rotation speed in this way facilitates different playing styles. Using a quicker turn of the wrist produces a short staccato–type sound. A more legato style is produced by slower rotation. In addition to the trigger and speed signals, puck #2 also returns its positive, scaled rotation values.

The distance between the table pucks and the slope of the line joining them are continuously computed and used as control data. The orientation of puck #1 is scaled to values linearly increasing with clockwise movement, while the y coordinate of puck #1 is also calibrated for control purposes.
5.3.3 Auditory-Visual Mappings

The variation of the sound generated by the manipulation of the control pucks is mirrored in the appearance of the graphics superimposed on the playing surface. Qualities of the visuals relate to those of the sound by a series of auditory-visual mappings described in this section.

5.3.3.1 Pitch and Colour, Pitch and Size

As the distance between the control pucks is lengthened, the base pitch of the sound synthesised becomes lower in frequency. At the same time, more shape elements are added to the visual scene in the space between the controllers. The current pitch being played is the shape element nearest to puck #2, highlighted by a red border. Pitch is mapped to size, as successive shapes added are larger than preceding ones. Fully formed, non-distorted objects actually represent the points at which notes in the scale are obtained. The highest note is that corresponding to the formation of a single shape element.

As well as growing larger, successively added shapes are also darker in colour. The highest note achievable therefore corresponds to the shape with the highest aggregate RGB colour values. This effectively is a mapping of pitch to colour brightness.

Figure 5.11 presents the graphical output of the system as inter-puck distance increases. On the left of the figure, a note is being played that is three notes distant from the highest note that is achievable- as the highest note available is a C3, this note is an F2. In the right of the figure, an E2 is being played. The shape element corresponding to this latter note is darker in colour and larger than those of the higher notes.
5.3.3.2 Amplitude and Size

The amplitude value of the sound output also contributes to the sizes of the shape elements produced. Sound output level is increased by turning puck #2 in a clockwise direction. Larger output values generate larger shapes, but with the correspondence between pitch and size still being preserved. Figure 5.12 shows how the graphics are affected.
5.3.3.3 Timbral Brightness and Object Curvature, Timbral Brightness and Slope

The rotation angle of puck #1 controls the overall brightness of the timbre of sounds produced by the instrument. This is reflected in alterations to the shapes of the graphic elements. As the brightness control is turned clockwise, the shape outlines change from smoother curves to a series of straight-line segments. The overall effect is illustrated in figure 5.13, showing a change from softer, rounder shapes to more angular, spiked ones.

![Figure 5.13: Changes in the shapes of the graphic elements obtained from increasing the overall timbral brightness control.](image)

In experiments with the system, it was found that the slope of the line between the control pucks should be correlated with some aspect of the sound output. This incline therefore affects how the sound evolves in TwoHand. When the elements form a downward incline moving from puck #1 to #2, the brightness of the timbre lessens with time. An upward gradient corresponds with an apparent gradual increase in the brilliance of the output sound. This dynamic variation of timbre is achieved using an adjustable envelope that is modified by the slope of the line between the control pucks.

5.3.3.4 Harmonicity and Texture Periodicity

The position of puck #1 on the y-axis also contributes to the control of timbre. Here it is the *dissonance* or *inharmonicity* of the sound that is affected. At the position halfway between the top and bottom of the control surface, the sound produced contains
harmonic partials. Deviations from this rest position introduce more inharmonicity into the sound.

The effect is represented visually by the internal textures of the shape elements. The textures are generated by regular, overlapping curves of identical colour and reduced opacity. Increasing inharmonicity results in aperiodicity in the shapes. That is, the graphics move from static, regular textures to dynamic, irregular ones. The animation becomes more frenetic and the height relationships between the constituent curves of the shapes more random. Figure 5.14 illustrates this for various positions of puck #1 along the y-axis.

![Figure 5.14: Increasing inharmonicity by displacement along the y-axis produces irregular internal textures in the shape elements.](image)
5.3.4 Technical implementation

5.3.4.1 Functional Overview

Figure 5.15 illustrates the TwoHand system from a functional standpoint. The model is based around the discussion of design considerations from Chapter 3, in particular the consideration of mapping strategies. The Max/MSP mapping layer referred to in the diagram is not a distinct sub-patch in itself, rather it is the connections between the various sub-patches used to analyse the input gesture data and the sound synthesis and maxlink objects. As the flow of control illustrates, the mapping scheme is multi-layered and may be described in terms of a perceptual mapping layer. The mapping is complex, with control streams converging and diverging in different ways. In terms of the mapping schemes discussed in section 3.1, the mapping is a many-to-many scheme. This mirrors the inter-dependant nature of the cross-modal associations used in the instrument.

5.3.4.2 Gesture Analysis and Mapping

The extraction of perceptual parameters from input and the subsequent mapping of these parameters to synthesis- and graphics-based parameters are carried out by a number of Max/MSP sub-patches and within the Processing code. A description of the major functional aspects of the sub-patches not previously explained for the MaluTak and DynAudio instruments is now presented. Full, commented code appears in the Appendices.
Figure 5.15: Functional overview of the TwoHand system showing the flow of control streams expressed in terms of parameter mapping layers.

frequency_Scaler

The appearance of new shape elements in the graphics is governed by the distance between the control pucks and the specification of a unit length in the code. To ensure that musically relevant pitches coincide with the emergence of a new shape requires some sophisticated scaling of input to output. This operation is carried out by the frequency_Scaler patch. Other attempts using Max/MSP scale and expr objects were
unsuccessful: patches became more and more complicated in trying to provide the necessary mapping of linear spatial dimension to a logarithmic musical frequency distribution, while also taking account of the different interval steps of musical scales. The `frequency_Scaler` succeeds in generating a continuous range of output frequencies while guaranteeing that certain pitch values coincide with integer multiples of inter-puck distances. Different unit lengths can be input, and the scale and root note to be used are also assignable.

![Image of the frequency_Scaler sub-patch](image)

**Figure 5.16: The frequency_Scaler sub-patch**
**measure_Puck_Rotation_Rate**

The generation of triggers and puck rotation speed data is accomplished in the *measure_Puck_Rotation_Rate* sub-patch. The angle of rotation of the puck is mapped from input values of 0 to 360 degrees to an output range of -49 to 50 as mentioned previously. This patch measures the time that elapses between the detection of an orientation within a specified range of values, and the detection of the zero crossover. It was necessary to utilise a value range in this way, as discrete values were not always detected for quick rotation of the puck. The range of (negative) values chosen to start the timing mechanism is based on the amount of puck rotation comfortable for the user. This takes precedence over considerations of tradeoffs between the control's robustness for different puck speeds and accuracy. Attempts to use a stored history of data input also produced spurious results. When a zero point crossing in either direction is detected, the patch outputs a *bang* trigger signal. For clockwise movement the measured *dial time* (inversely proportional to rotation speed) is also output in milliseconds. These are used to scale and trigger envelopes as described next.

**scale_And_Generate_Variable_Envelope**

The *scale_And_Generate_Variable_Envelope* uses input in the form of the *dial time* sent from *measure_Puck_Rotation_Rate*, a trigger (*bang*) and slope values to generate an envelope. The first of these inputs scales the extent of the envelope along the temporal x-axis. The second outputs all the envelope point data. The values of input slope alter the shape of the four-point envelope generated. Positive and negative slope values are applied to different points on the envelope. For example, a positive slope varies the height of point three of the envelope, effectively controlling decay time. A negative slope affects point two, resulting in an envelope with a longer attack stage. The effect of the slope on the envelope shape is shown in figure 5.17. It is necessary to hold the first and last points of the envelope fixed due to limitations of the Max/MSP breakpoint function object. It was found that the object re-organises the points list and reassigns indexes as points are moved around. This meant that the split input streams each acted on different points after every adjustment, causing unpredictable output.
wiggle_data

wiggle_data is a simple patch used to add basic animation to the visuals. It uses a metronome (*metro*) object to periodically add 1 to the data passing through it. It modifies the *detail* parameter specified by puck #1 before this is passed to the graphics engine. Thus, the shapes of the elements change as the detail value is adjusted. When the metronome is switched off, data passes through the patch unaffected and the animation of the graphics stops.

**Figure 5.17: Effect of different slopes on the envelope shape generated in the scale_Domain_And_Generate_Envelope sub-patch**

*scale_Domain_And_Generate_Envelope*

This patch uses two trigger values obtained from the *measure_Puck_Rotation_Rate* sub-patch previously described to provide an amplitude envelope for the synthesis section.
The first trigger signal from movement in the clockwise direction triggers the attack phase of the envelope. A sustain level is then held until a trigger is received from anticlockwise movement of the puck past the zero point. Using the puck rotation speed to scale the amplitude envelope was found to result in a less intuitive playing experience. Because the amplitude of the synthesiser output is also controlled by the rotation values of puck #2, the addition of another time-varying amplitude envelope did not offer any benefits.

### 5.3.4.3 Graphics

The *TwoHand Processing* sketch generates the visual output of the system. Data obtained via *maxlink* is stored in two arrays. One contains x/y position and orientation data for the two control pucks used, while the other stores a list of points on the x/y plane. Basic trigonometry is used to calculate the x- and y-axis displacement components of the inter-puck length. Conditional statements in the *Processing* code allow placement and manipulation of the widgets anywhere on the display surface and preserve the consistency of calculations of slope when the pucks are swapped between hands.

The distance between the two pucks is calculated in *Processing* even though a similar calculation is also carried out in *Max/MSP*. It was found to be more effective to do this, rather than send specific distance values from *Max/MSP* to *Processing* in addition to sending the x/y position data that must be sent to highlight the puck positions in the graphics. Repeating the calculation in this way does not adversely affect computational efficiency to any measurable degree.

The inter-puck distance value and the unit length amount previously described are used to calculate the number of full and partial segments that fill the space between the pucks. A loop structure draws successive shape elements and allows the qualities of each shape to vary with each iteration.

Each shape segment is rendered using Bezier curves drawn between anchor points computed both above and below the line joining the pucks. The control points for the Bezier curves have a vertical displacement from the anchor points depending on the *size* input variable and the shape element’s position in the chain. As puck #2 is moved
further from puck #1 the height displacement of any added elements increases, scaled by the $size$ value.

The curvature of the shapes is controlled by altering the number of Bezier curve elements as described in section 5.1.2. The Bezier quality is an input variable passed to the bezierDetail() method based on the orientation of puck #1.

The fill colours of the elements are darker for more recently added shapes. RGB parameter values are multiplied by the number of the loop iteration and subtracted from a maximum value of 255. The opacities of the shapes are adjusted similarly, with the last drawn shape having 100% opacity while the others are set to be semi-translucent. A highlighting border is drawn around the last element by adjusting the stroke weight. These adjustments to the stroke weight and shape opacity signify that the most recent shape corresponds to the pitch being played.

The periodicity of the internal shape textures is controlled by independently randomising the heights of the Bezier curves. The range of random values is gradually enlarged using the increasing values of the $periodicity$ value sent from Max/MSP. Decreasing the periodicity value reduces the range of random values being used so the curves become more regularly spaced.

5.3.4.4 Synthesis

The TwoHand instrument uses frequency modulation (FM) synthesis for sound generation as outlined in section 5.2.3. More varied timbres are produced than those in the Dyn.Audio instrument by the application of different envelopes to the synthesis parameters.

The scale_Domain_And_Generate_Envelope outputs an amplitude envelope as described in section 5.3.4.2, and this is applied to the amplitude of the carrier signal in the FM synthesiser. The positive linear values obtained from the rotation angle of puck #2 are fitted to an inverse exponential curve and applied directly to the synthesizer output volume. This is preferable to using the control to scale the synthesiser amplitude input, as the discrete values of the controller were found to introduce clicks or zipper noise in the output.
The frequency_scaler sub-patch from section 5.3.4.2 is applied to the carrier frequency input of the synthesiser, providing a value for the base pitch of produced sounds. Although this patch effectively quantises pitches for use in the graphical display as previously outlined, the output of the synthesiser is continuous in the frequency domain.

The output from scale_And_Generate_Variable_Envelope, also described in section 5.3.4.2, is used to dynamically vary the FM modulation index and so affects the brightness of timbres produced. The envelope is scaled by the rotation of puck #1, controlling the overall brightness level.

The C:M ratio input of the synthesiser is adjusted by the y-axis position of puck #1. This allows control of the nature of the harmonics generated in sound output, as presented in section 5.2.3.

5.4 Conclusions

This chapter presented three software instruments realised for use with system framework outlined in chapter 4. The instruments increase in complexity from the basic graphical manipulation of MaluTak, through the Haeberli-inspired DynAudio audio-visual instrument, to the more sophisticated gesture analysis and complex mapping strategies of TwoHand.

Modes of operation, typical graphical output and the basic functional units of the systems were described. Descriptions of some of the less successful early attempts at implementing various aspects of the systems were also presented.
6. Discussion and Analysis

6.0 Introduction

In this chapter the implementation of the graphics-based synthesis controller is evaluated and some areas for possible improvement are identified. Particular attention is given to the implementations of the theoretical considerations described in chapters two and three.

The first section looks at the system framework used to realise the project goals. Section 6.2 studies the individual software instruments. *MaluTak* and *Dyn.Audio* are analysed more holistically because of their basic functionality and because to a large extent they can be assessed without the use of the hardware setup. *TwoHand*, being a more sophisticated system, is first examined from the perspective of its technical implementation and functional aspects. Section 6.3 details the performance of a prototype of the installation setup and describes the experience of using *TwoHand* in that environment.

6.1 Software System Framework

*ReacTIVision* provides an excellent computer vision framework. It eliminates the need for the development of specialised signal processing algorithms, as would be necessary in environments such as *EyesWeb*, and offers better stability and performance in its video capture capabilities than *Processing*. The availability of a *Max/MSP* client for the *TUIO* communications protocol and a software table-controller simulator made the realisation of this project feasible within the project timeframe.

*Max/MSP* is a flexible and robust environment that offers many possibilities for rapid development of creative multimedia projects. It has a somewhat steep initial learning curve, but previous familiarity with its graphical interface facilitated rapid prototyping of ideas throughout this project. The *Jitter* graphics generation component of Max was not used however, as it was found to use more computational resources and had a steeper learning curve than *Processing*.
Processing does require some familiarity with object-oriented programming but its streamlined Java-type language appealed more than the alternative graphics packages that were tested. The logical structure of its language facilitated quick development of ideas. The availability of the maxlink java class library allows reliable communication with Max/MSP, while OpenGL graphics capabilities lessen the demand on system resources.

The system framework performs remarkably well from an efficiency perspective. The processing capabilities of the computer are not taxed, with CPU load remaining under 60% in the most efficient reactTIVision camera mode. This is in stark contrast with some other systems that utilise computer vision as a gesture-capture mechanism [Solon, 2006].

6.2 Instrument Evaluations

The three software instruments developed for use are now examined. Where deficiencies are identified in the implementation of the instruments, suggestions are made as to how these may be rectified. These are typically system improvements that were not realised during the project due to time constraints.

6.2.1 MaluTak

As a basic instrument exploring the capabilities of the system framework and graphics drawing techniques, MaluTak is successful. The use of Bezier curves to provide shape-changing capabilities is effective. However, while the bezierDetail() method in Processing does allow variation between smooth and jagged curves, the apparent effect of changing the curve detail parameter in the lower value ranges greatly outweighs the change in shape evident from changing the parameters at higher values. At resolutions greater than about 20 or so, the curves are still becoming smoother as the numbers of their constituent line segments grow larger, but it is quite difficult to see any change. Manipulation of the control points of the curves in tandem with changes in the Bezier resolution could provide more extreme and controllable changes of form. Such a technique would require more experimentation in computer graphics than the time-frame of this project permitted, however.
The *Processing* sketch contains a minor operational bug. Pucks must be removed from the scene in reverse order to which they were added. That is, the last puck to be added must be the first to be removed, and so on. This is because of the way the puck coordinates are stored in an array with each *Add* or *Take* message string sent from *Max/MSP*. An *Add* message simply expands the size of the array by one while a *Take* message removes the last array index. As a result, the removal of the second puck to be added to the scene causes the removal of the last puck added. The problem could be rectified by indexing the array with fiducial identification numbers, but an attempt at implementing this during the project was unsuccessful.

The lack of sound output means *MaluTak* does not hold interest as a musical instrument. A possible mechanism to add sound output is illustrated in figure 6.1. The use of an animated *read head* in the form of a radiating circle would allow the widgets to act as triggers for sound events. Different rhythmic patterns would be obtainable by moving the graphical objects around. A timbral control could be implemented based on the control puck rotation, correlated with the widget shape. Pitch could be controlled by the x/y positions of the widgets. Such modifications are just beyond reach for inclusion in this project, as they require the development of collision detection or a similar mechanism to generate sound triggers as the *read circle* passes over the objects.

*Figure 6.1: Possible implementation of sound in MaluTak. The animated circle radiates out from the centre and triggers a sound when it passes over a graphical widget, resulting in different rhythmic patterns.*
6.2.2  DynAudio

*DynAudio* is surprisingly successful despite it being somewhat of a program hack—the graphics component is an adaptation of code that is nearly twenty years old. The instrument’s sophistication is elevated somewhat by the curvature analysis sub-patch implemented in *Max/MSP*. The algorithm is not always entirely accurate, but works effectively in a musical context as there is a transparent link between the energy and movement invested in sketching a shape, and the resulting audio. The cross-modal association of curvature of graphical form with brightness of timbre is intuitively understood. This reflects the fact that it implements a simple *one-to-one* mapping. As discussed in section 3.1.1, these types of mappings do present the most intuitive interfaces, often at the expense of depth of expression. Indeed, *DynAudio* does not offer any deeply expressive capabilities and does not demand prolonged practice for its mastery, but it is not designed as such. Above all else, the instrument is *fun*.

In the current version, there is no capacity to change the constants of the simulated physical system while it is running. The graphics rendered by the *Processing* code are representations of the input strokes as shaped by the simulation’s parameters at runtime. In future versions, the curvature value computed from the input stroke in *Max/MSP* could be linked to the spring constant of the system. This would mean that the strokes would vary in character along their lengths in correlation with the sounds being produced. As it is, the spring constant is set to a medium value. This means that more *jagged* strokes produce similar visual output, while *smoother* strokes are softened, linking well with the audio produced simultaneously. More complex control of the other simulated physical constants could open up different system behaviours and suggest new associations between the auditory and visual realms.

6.2.3  TwoHand: Technical implementation

The experience of using *TwoHand* in the installation setup is discussed in section 6.3, but a brief analysis of the functional aspects of the instrument is presented here.

The *Max/MSP* patch is quite complex, employing a *many-to-many* mapping scheme with numerous cross-connections between functional units. The top-down data flow and the
use of encapsulated sub-patches that perform a limited number of specific tasks does clarify the functional architecture, however.

The various sub-patches function well overall. The *frequency_scaler* module is a particularly useful method of relating a linear length measurement to a musical scale. The calculation of puck rotation speed to use in scaling envelopes is also useful. This technique ensures that timbres vary dynamically in different ways depending on how they are played. For instance, the modulation index envelope is applied after an interval based on this speed of rotation, ensuring that the same attack phase is not played at the start of every sound. The *wiggle_data* sub-patch is less successful, being somewhat devalued by the more sophisticated animation driven by the periodicity parameter. This sub-patch may be bypassed, however.

What the system lacks is an adaptable mapping scheme. This would facilitate experimentation with different mappings at different mapping stages. The rotation of puck #2 could be swapped with the y-axis coordinate of puck #1, for instance, so that harmonicity of timbre and periodicity of texture would now be controlled by rotation. To do this would require that the outputs of all sub-patches be scaled to the same ranges, a non-trivial task considering the mixture of linear and non-linear controls used in the system. Time constraints forced a focus on developing a robust system with a definite mapping scheme, rather than on providing this flexibility. It is open to speculation whether such flexibility would give rise to useful mapping arrangements.

The use of FM synthesis was a compromise chosen to facilitate more rapid development of the system. It was originally envisaged that the system would use a physically-modelled synthesis technique that would facilitate haptic rate manipulation of the sound generation simulation, as discussed in section 3.3.3. The more complex behaviour of *scanned synthesis* and the larger number of input parameters meant that it could not be implemented effectively within the project timeframe. This technique also demands more of the computer processor, and would necessitate more than one PC in the system. FM was found to be more than adequate however, due to its wide range of available timbres and its comparatively high computational efficiency.
The graphics generation in Processing is simplistic, but performs as intended. The use of Bezier curves to change graphical shapes suffers from the minor aesthetic problem of an apparent non-linear response to variations in resolution, as discussed in section 6.2.1. Optimisation of the code and implementation of a more object-oriented approach would release more processor resources and provide better application flexibility in the future. In its current state neither of these aspects affects performance to any discernible degree, however.

The choice of auditory-visual mappings used in the system is not arbitrary, but informed by the discussion of cross-modal perceptual analogies of chapter 2. The correlation between curvature of graphical form and brightness of timbre stems from the results of Kohler’s Maluma/Takete experiment discussed in section 2.2.1 and the physics-based techniques of Jones and Nevile described in section 2.3.1. The use of brighter colours to relate higher musical pitches is one of the most common trends found in synaesthetes, as observed in section 2.1.2. The simultaneous mapping of pitch to size (in an inverse relationship) is another real-world observation. The use of a texture/sensory-dissonance analogy has its basis in the empirical findings of Giannakis from section 2.3.2. The addition of another timbre modifier, based on the slope of the line between the control pucks, is a mapping that developed from experimentation and feedback of users of an early version of the system. The correlation seems to be an aesthetic one and somewhat culturally-based. It is as if users read the sound from left to right, and expect qualities of the sound to perform temporally as the graphics are portrayed spatially. A positive slope from left to right suggests an increase in the brightness of the corresponding sound. A negative slope suggests a feeling of decrease or going down.

Whether or not these mappings are successful is subjective; perhaps the only real indication must come from observation of gallery users’ reactions it is hoped that the cross-modal associations employed in the system do offer intuitive links between visuals and sound.
6.3      Installation Evaluation: TwoHand on the Table

A prototype of the system used to test the TwoHand instrument is shown in figure 6.2. This system was constructed with the resources available and is a scaled down version of the full assembly. As such, it suffers a number of performance deficiencies, but nevertheless provides a useful environment for evaluating the experience of using the TwoHand software.

6.3.1      Prototype System Performance

This experimental setup has a number of limitations due to the resources available for its construction. The lighting arrangement of the prototype setup is not ideal and the control pucks are paper prints of the fiducial symbols.

The LED disk assembled to provide infrared (IR) illumination is not powerful enough to allow the use of an IR pass filter on the Fire-i camera. Attaching the filter results in a noisy camera scene in realTIVision and the loss of fiducial position and orientation. Bringing the IR source closer to the table surface produces a bright spot in the camera field, and a suitable diffuser to spread the light more uniformly across the table underside could not be obtained. Without the capacity to split the camera scene into visible and non-visible regions, it is not possible to control the system when projecting the graphics onto the control surface as shown in figure 6.2.
Figure 6.2: Photograph of the prototype system setup. In this configuration, control does not function when the graphics are projected as depicted.

With graphic output disabled, or projected somewhere other than the table surface, the system can be controlled as intended by the movement of fiducials on the table surface as shown in figure 6.3.

The system responds well as a musical control but there are some noticeable improvements to be made. The lighting setup of the prototype does not illuminate the table underside uniformly. As a result, fiducials tend to get lost in areas of increased lightness or darkness, resulting in discontinuity in control value output. The effect does not totally inhibit playing the instrument, but rather manifests as discrete changes in output parameters. For example, changing the length between pucks sometimes results in a discernible stepping through pitch values rather than a smooth, continuous frequency adjustment. The lighting also causes some noise in the captured image, resulting in small variations in the control data being sent. This can be observed in the output graphics when the pucks are left stationary on the table surface. The representations of the puck positions in the visuals can be seen to move slightly away from their position and back
again. The movement is typically only a pixel in any direction and the timings of the movements are random, as would be expected from a noisy signal.

Figure 6.3: Photograph of the prototype system setup. In the configuration depicted the fiducial controls function, as visual feedback is projected away from the table surface.

Attempts to rectify these problems, or lessen their effects, could be made in software. An interpolation method could be used to smoothly manage movement of the controllers. A noise reduction technique could be employed to steady the random fluctuations in control. It is felt, however, that this would be somewhat over-engineering a solution. Improvements in lighting setup must be the first approach adopted.

The latency of the system does not seem to be functionally inhibitive, though this is a somewhat subjective judgement. As will be discussed below, the interface tends to evoke gestures that seek to gradually mould and shape the sound and graphics output. Users do not tend to use quick, percussive-style movements on the instrument. Any latency in the system may become more apparent when the graphics are successfully superimposed on
the control area, but for now the response of the system seems to be suitably fast for musical application.

6.3.2 The TwoHand Experience

A full evaluation of the experience of using the interface can not be obtained until the installation has been fully implemented. Even then, the considered opinions of a wide range of users over a significant time period may not offer any kind of definitive empirical measure of the interface’s success or failure. A truly objective assessment may not even be possible. Can universal agreement be found for such an abstract concept as the *successfulness* of a musical instrument?

Such philosophical considerations aside, documentation of a user’s comments can give an insight into the positive and negative aspects of that user’s experience. At this point, feedback from initial users of the prototype version is the subjective measure that may be used to loosely quantify the prototype setup.

The experience of using TwoHand is immersive and engaging. The playing technique gives the impression of pulling and shaping sound with both hands. The effects of different gestures on the output are quite transparent for the more basic functions such as pitch, amplitude and overall brightness of timbre. The effect of the slope of the line between the pucks on the output is an example of a less apparent relationship between cause and effect.

Not all playing techniques of the interface should be immediately apparent, however. Such an instrument would quickly exhaust its appeal. The ultimate aim of any interface may be the provision of an instantly knowable and infinitely deep experience, but such an ideal system may not be realisable. The inclusion of the harmonicity control using the y-axis displacement of puck #1 is included as a type of extended technique but the experience of its execution is not as unnatural as such a label would suggest. In fact, the discovery of the control method becomes apparent quickly due to the graphical feedback. As the harmonicity of timbre is mapped to the periodicity of the shapes’ internal textures, users see the introduction of irregularity in the visuals and retrace their movements to try to undo their actions. The return of the sound to its original timbre in
concert with the graphics return to a periodic state is recognised and learned. The act of
discovery adds to the enjoyment of the experience. In this regard, at least, the interface
must be viewed as a success.

6.4 Conclusion

This chapter analysed and discussed the implementation of a graphics-based synthesis
controller system. The decisions made on the choice of system framework were
evaluated and assessments of the individual software instruments were presented. A
prototype system was analysed from a technical standpoint and from the experience it
offers users.
7 Conclusion

A review of the associations made in the mind between experiences in different sensory modalities highlights the universal nature of at least some of these links. Such cross-modal correlations intensify the human experience, enriching life with abstract perceptual meanings that go beyond a mechanical transmission of rigidly-defined physical reality.

This project presents the development of a graphics-based synthesis controller interface that employs such inter-sensory relationships to reinforce and intensify the perceived consequences of user action. By relating input gesture to perceptual quantities and simultaneously representing these in the auditory and visual realms, an engaging and immersive experience is provided.

7.1 Future Work

The first task ahead is the full implementation the system presented in the form of a gallery installation. Such a setup will provide a rigorous test of the system’s technical performance and will no doubt suggest design improvements. User interaction will also inform on the experience of using the interface. The intuitiveness of the cross-modal associations used to link the auditory and visual output and the mapping of gesture input to the sound and graphics parameters will be more fully assessed. To this end, it may be desirable to record user interaction by some means to correlate such data. The technical implementation of this is still to be considered.

The system has possible application to teaching. The use of intuitive, perception-based control may cultivate an appreciation of abstract creativity that supplements traditional musical theory. Children and non-musicians could find it particularly insightful, but the interface should offer a new mode of expression for a wide range of users.

Possible modification and improvements of the software instruments have already been suggested in chapter 6 and there are a multitude of possibilities to take in this direction. Physically-modelled systems and cellular automata are examples of two techniques that
use existing Java libraries to enrich graphics presentation and could add new means of control. The use of a physical model for sound generation, such as scanned synthesis, would also open new avenues of audio manipulation.

The computer vision system is under constant development and an imminent release includes the capability for finger-tracking (as of June 2007). This would require a reconsideration of the control mapping used in the system, but could ultimately provide an even more immersive experience of using the interface.
Appendix A  Max/MSP Patches

A.1  MaluTak Max/MSP Patches
A.2 DynAudio Max/MSP Patches

[Diagram of Max/MSP patchbay with various nodes and connections.]

- Generates activate/deactivate messages based on clockwise/anti-clockwise rotation.
- Detects core transition.
- Simple Envelope Generator: Prevents audio clicks.
- Loadmess 100: Sets the maximum load value.
- Pack 0.0: Packs data.
- Speedlim 100: Limits speed.
- Loadmess 50: Sets the load value.
- M:O Ratio: Modulates signal.
- Amp: Amplifies signal.
- Volume: Controls volume.

Additional parameters:
- Carrier Frequency: Modulates carrier.
- Mod Index: Modulates index.
- M:O Ratio: Modulates modulation index.
- Amp: Modulates amplitude.
- Audio on/off: Activates or deactivates audio output.
Max - [simp...]

File Edit View Object
Font Options Trace Extras
Window Help

Max - [invertYCoords]

File Edit View Object Font Options
Trace Extras Window Help

0, 120 1, 0 200

scale 0 480 440 220 1.
Max - [measure_Puck_Rotation_Rate]

- Zero transitions from positive and negative directions

Generate trigger

Need to test for dial passing through range of values as every discrete value is not always included in fast movements

Capacity for measuring speed in an anti-clockwise direction (not usual)

Max - [scalePeriodicityRegions]

- Scale 0.0 - 640.0 0.1
- Scale 0.0 - 640.0 0.1
- Scale 0.0 - 640.0 0.1
- Scale 0.0 - 640.0 0.1
makes sure the denominator is never 0.
Max - [scale_And_Generate_Variable_Envelope]

Split positive and negative slopes to control different envelope points.

Points 1 & 2 are affected by slope, points 3 is scaled and point 0 is always at origin.

Max - [wiggle_data]

leadbang 0
metro 500
+ 1
+ 1
39
Appendix B.  

**Processing Code**

B.1  

*MaluTak Processing Code*

```java
/////////////////////////////////////////////////////////////////////
///////////
//”MaluTak”  Graphics Generation Sketch
//
//by
//Liam O'Sullivan
///////////
//This sketch generates the graphics for the "MaluTak" Graphics Application
//The system uses up to four control pucks of the reacTIVision system- Fiducial //IDs 0, 4, 8 and 12
//Input from Max/MSP over "maxlink" communications link
/////////////////////////////////////////////////////////////////////
///////////
//The Maxlink Library that facilitates communication with Max/MSP
import maxlink.*;
//OpenGL rendering library greatly improves performance
import processing.opengl.*;
//This is the name that must be specified in the maxlink mxj object in Max/MSP
MaxLink link = new MaxLink(this, "MaluTak");

//Declaring a global variable to store an array of Puck objects
Puck[] mypucks;

//Declaring the inlet variables
int int_x = 0;
int int_y = 0;
float f_angle = 0;
int f_id = 1;
String string = "";

int puckCount = 0;

//Setup Function////////////////////////////////////////////////////////////////////
///
void setup()
{
    size(600, 600, OPENGL);
    frameRate(12);
    // creating an empty array, capable of holding Puck objects
    mypucks = new Puck[6];

    // Filling array with elements of type Puck
    for( int i = 1; i < 6; i++)
    {
        mypucks[i] = new Puck();
        mypucks[i].x = 0;
        mypucks[i].y = 0;
    }
}
```

113
// Declare the variable names and the setter function names
link.declareInlet("int_x", "setint_x");
link.declareInlet("int_y", "setint_y");
link.declareInlet("f_angle", "setf_angle");
link.declareInlet("f_id", "setf_id");
link.declareInlet("string", "setstring");
}

// Draw Function: Continuously iterates////////////////////////////////////////////
void draw()
{
    background(255);
    for( int i = 1; i < puckCount + 1; i++)
    {
        // This selects which x and y positions are updated based on id number
        switch(f_id)
        {
            case 1:
                mypucks[1].x = int_x;
                mypucks[1].y = int_y;
                mypucks[1].a = f_angle;
                break;
            case 2:
                mypucks[2].x = int_x;
                mypucks[2].y = int_y;
                mypucks[2].a = f_angle;
                break;
            case 3:
                mypucks[3].x = int_x;
                mypucks[3].y = int_y;
                mypucks[3].a = f_angle;
                break;
            case 4:
                mypucks[4].x = int_x;
                mypucks[4].y = int_y;
                mypucks[4].a = f_angle;
                break;
            default:
                break;
        }
        // invoking the object's method (named run)
        mypucks[i].drawShape(i);
    }
}

// Setter functions //////////////////////////////////////////////////////////////////
void setint_x(int newint_x) {
    int_x = newint_x;
}
void setint_y(int newint_y) {
    int_y = newint_y;
}

void setf_angle(float newf_angle) {
    f_angle = newf_angle;
}

void setf_id(int newf_id) {
    f_id = newf_id;
}

void setstring(String newstring) {
    string = newstring;
    // Increase and decrease no of objects with each 'Add'/ 'Take'
    // message:
    if (string.equals("Add") && (puckCount != 4)) {
        mypucks[f_id].active = 1;
        puckCount++;
    } else if (string.equals("Take") && (puckCount != 0)) {
        mypucks[f_id].active = 0;
        puckCount--;
    }
}

// Defining a custom class called Puck. Instances of this class are
// put into the array 'puckList'

class Puck {
    int x;
    int y;
    float a;
    int f_id;
    int active;
    int r, g, b;

    void drawShape(int i) {
        if (puckCount > 0) {
            // The random numbers generated are used to scale the displacements of
            // the control points each iteration, effectively animating the
            // widgets
            float seed = random(50, 100);
            float r1 = random(50, seed);
            float r2 = random(50, seed);
            float r3 = random(50, seed);
            float r4 = random(50, seed);
            float r5 = random(50, seed);
            float r6 = random(50, seed);
            float r7 = random(50, seed);
            float r8 = random(50, seed);

            ellipseMode(CENTER);
            if (active == 1) {

                }
ellipse(x, y+100, 20, 20);
}
smooth();

// following statements allocate different colours for each graphical
// widget drawn
if (i == 1){
    fill(200*i, 0, 0 );
}
if (i == 2){
    fill(0, 200*i, 0 );
}
if (i == 3){
    fill(0, 0, 200*i );
}
if (i == 4){
    fill(10*i, 50*i, 100*i );
}

// converts to int for use as parameter in bezierDetail() method
int detail = int(a);

// controls the bezier curve quality: 1 is straight line, default is 20
bezierDetail(detail);

// these are the curves that make up the graphical shapes
bezier( x, y, x+r1, y+r2, x+r3, y-r4, x, y);
bezier( x, y, x-r5, y+r6, x-r7, y-r8, x, y);
bezier( x, y, x+r1, y+r2, x-r3, y+r4, x, y);
bezier( x, y, x+r5, y-r6, x-r7, y-r8, x, y);
bezier( x, y, x+r1, x+r2, y, x, y);
bezier( x, y, x-r3, x+r4, y, x, y);
bezier( x, y, x-r5, x-r6, y, x, y);
bezier( x, y, x, y+r7, x-r8, y, x, y);
B.2  DynAudio Processing Code

//////////////////////////////////////////////////////////////////// /
///////////
//"DynAudio" Synthesis Controller System- Graphics Generation Sketch
//by
//Liam O'Sullivan
//////////////////////////////////////////////////////////////////// /
//This sketch generates the graphics for the "DynAudio" Graphics- 
//Based Synthesis Controller
//The system uses a single control puck of the reacTIVision system
//Input from Max/MSP over "maxlink" communications link
//"DynAudio" is based on a port of "Dynadraw", which is described at: 
// Originally created in June 1989 by Paul Haeberli
// Ported to Proce55ing January 2004: www.processing.org
//////////////////////////////////////////////////////////////////// /

//The Maxlink Library that facilitates communication with Max/MSP
import maxlink.*;

//OpenGL rendering library greatly improves performance
import processing.opengl.*;

//This is the name that must be specified in the maxlink mxj object
in Max/MSP
MaxLink link = new MaxLink(this, "DynAudio");

int fiducialId = 0;
float px, py;                               // current position of spring
float vx, vy;                               // current velocity
float ppx, ppy;                             // our previous position
float k = 0.4;                               // bounciness, stiffness of spring
float damping = 0.88;                        // friction
float speedVThick = 0.5;                     // this constant relates stroke width to
speed
float max_th = 15.0;                         // maximum stroke thickness
float mass = 1;                              // mass of simulated pen

//sliders are not used in graphics but the parameters are used as variables
int sliderh = 25;
float max_K_val = 0.2;
float min_K_val = 0.01;
float max_D_val = 0.999;
float min_D_val = 0.250;
boolean editing_K = false;
boolean editing_D = false;
int xDataIn = 0;
int yDataIn = 0;
float angleDataIn = 0;
String string = "deactivate";
boolean drawOn = true;
float r;

//Setup
Function

//////////////////////////////////////////////////////////////////// /
///
void setup(){
  size(640, 480, OPENGL);
  ellipseMode(CENTER);
  background(255);
  stroke(0);
  noFill();

  // Declare the variable names and the setter function names
  link.declareInlet("xDataIn", "setxDataIn");
  link.declareInlet("yDataIn", "setyDataIn");
  link.declareInlet("angleDataIn", "setangleDataIn");
  link.declareInlet("fiducialId", "setfiducialId");
  link.declareInlet("string", "setstring");

  ppy = py = height/2;
  ppx = px = width/2;
  vx = vy = 0;
}

//Draw Function: Continuously iterates///////////////////////////////////////
void draw(){

  updateTrace();
  drawTrace();
  ppx = px; // Update the previous positions so lines
  ppy = py; // can be drawn next time through the
  loop.

  if (drawOn){
    activate();
  } else if (!drawOn) {
    deactivate();
  }
}

///////////////////////////////////////////////////////////////////////////
void updateTrace(){
  float dy = py - yDataIn; // Compute displacement from the cursor
  float dx = px - xDataIn;
  float fx = -k * dx; // Hooke's law, Force = - k *

  float fy = -k * dy; // Hooke's law, Force = - k *

  float ay = fy / mass; // Acceleration, computed from F = ma
  float ax = fx / mass;

  vx = vx + ax; // Integrate once to get the next
  vy = vy + ay; // velocity from the acceleration

  vx = vx * damping; // Apply damping, which is a force
  vy = vy * damping; // negatively proportional to velocity

  px = px + vx; // Integrate a second time to get the
  py = py + vy; // next position from the velocity
}
void drawTrace() {
    // Compute the (Pythagorean) velocity
    float vh = sqrt(vx*vx + vy*vy);
    // Scale, clamp and invert
    float th = max_th - min(vh*speedVThick, max_th);
    th = max(1.0, th);
    // Calculate the stroke weight
    strokeWeight(th);
    stroke(0);
    smooth();
    line(ppx, ppy, px, py);
}

void activate() {
    // the slider variables are used even though the slider output
    // graphics have been removed from this version
    float tol = 40;
    float K_x = (float)width*(k - min_K_val)/(max_K_val - min_K_val);
    float D_x = (float)width*(damping - min_D_val)/(max_D_val -
                    min_D_val);
    if ((abs(xDataIn - K_x) < tol) && (yDataIn > 0) && (yDataIn < sliderh)){
        editing_K = true;
        editing_D = false;
    } else if ((abs(xDataIn - D_x) < tol) && (yDataIn > sliderh) &&
            (yDataIn < sliderh*2)){
        editing_D = true;
        editing_K = false;
    } else {
        editing_K = false;
        editing_D = false;
        background (255);
    }
}

void deactivate(){
    editing_K = false;
    editing_D = false;
}

void setxDataIn(int newxDataIn) {
    xDataIn = newxDataIn;
}

void setyDataIn(int newyDataIn) {
    yDataIn = newyDataIn;
}
void setangleDataIn(float newangleDataIn) {
    angleDataIn = newangleDataIn;
}

void setfiducialId(int newfiducialId) {
    fiducialId = newfiducialId;
}

void setstring(String newstring) {
    string = newstring;

    //Increase and decrease no of objects with each 'Add'/ 'Take'
    //message:
    if (string.equals("activate")) {
        drawOn = false;
    } else if (string.equals("deactivate")) {
        drawOn = true
    }
}
B.3  TwoHand Processing Code

/////////////////////////////////////////////////////////////////////
///////////
// TwoHand Synthesis Controller System- Graphics Generation Sketch
// by
// Liam O'Sullivan
/////////////////////////////////////////////////////////////////////
///////////
// This sketch generates the graphics for the "TwoHand" Graphics-Based
// Synthesis //Controller
// The system uses two control pucks of the reacTIVision system-
// Fiducial IDs 0 //and 4
// Input from Max/MSP over "maxlink" communications link
/////////////////////////////////////////////////////////////////////
///////////
// The Maxlink Library that facilitates communication with Max/MSP
import maxlink.*;
// OpenGL rendering library greatly improves performance
import processing.opengl.*;
// This is the name that must be specified in the maxlink mxj object
in Max/MSP
MaxLink link = new MaxLink(this, "TwoHand");

// Global
variables/////////////////////////////////////////////////////////////////////
int uD = 66;  // The unit length of graphical shape segments
float uDx = 0.0;  // Components of uD along x- and y-axes
float uDy = 0.0;
float dx = 0.0;  // Components of the displacement between the
control
float dy = 0.0;
Puck[] puckList;  // Stores the array that holds two puck
locations
Point[] pointList;  // Stores the array that holds the start and
end points for each segment
int puckCount = 2;  // Keeps track of number of pucks in the
puckList array-  // not varied in this
implementation

// Input from
Maxlink/////////////////////////////////////////////////////////////////////
float fiducialId = 1;  // The ID number of the puck being updated,
added,  // removed etc.
int xDataIn = 0;
int yDataIn = 0;  // x,y position data and puck rotation
angle
float angleDataIn = 0;
float periodicity = 1.0;  // this variable is used to change the
regularity
internal textures
int sizes = 0;
int detail = 20;
the  // Input variable used to scale the size of
// graphic elements
String string = "";  // Input message string for providing "addObject" and other messages- NOT USED

//Setup
Function////////////////////////////////////////////////////////////////////////
/

void setup(){
    size(640, 480, OPENGL);
    frameRate(25);
    // Declaration of the variable names and the setter function used to assign //values based on input
    link.declareInlet("xDataIn", "setxDataIn");
    link.declareInlet("yDataIn", "setyDataIn");
    link.declareInlet("angleDataIn", "setAngleDataIn");
    link.declareInlet("fiducialId", "setFiducialId");
    link.declareInlet("sizes", "setSizes");
    link.declareInlet("detail", "setDetail");
    link.declareInlet("string", "setString");
    link.declareInlet("periodicity", "setPeriodicity");

    // Make space in array for Puck objects
    puckList = new Puck[3];

    // Fills array with Puck Objects
    for(int i = 0; i <= 2; i++){
        puckList[i] = new Puck();
    }

    //Use the window's diagonal size to get max possible segments
    int num = int((sqrt(sq(width)+sq(height))/uD);

    //Make array big enough to hold enough points for a 'num' number of shape //elements
    pointList = new Point[num + 1];

    // Fills array with Point Objects
    for(int i = 0; i < num + 1; i++){
        pointList[i] = new Point();
    }

//The Processing Draw
Loop////////////////////////////////////////////////////////////////////////

void draw(){
    background(250);
    for( int j = 1; j < puckCount + 1; j++){
        int fId = int (fiducialId);

        // This selects which x & y positions and rotation angles are updated based on //fiducial ID number
        switch(fId){
            case 0:
                puckList[1].x = xDataIn;
                puckList[1].y = yDataIn;
                puckList[1].angle = angleDataIn;
                break;
            case 4:
                puckList[2].x = xDataIn;
        }
    }
}
puckList[2].y = yDataIn;
puckList[2].angle = angleDataIn;
break;
default:
    break;
}
}

//These conditional statements ensure that the pucks can be placed anywhere on
//the control surface and calculations of distance and slope etc. are consistent

if(puckList[2].x > puckList[1].x){
dx = (puckList[2].x - puckList[1].x);
} else{
    dx = (puckList[1].x - puckList[2].x);
}
if(puckList[2].y > puckList[1].y){
dy = (puckList[2].y - puckList[1].y);
} else{
    dy = (puckList[1].y - puckList[2].y);
}

//Calculates the displacement length of the line between the centres
//of the pucks using trigonometry.
floating d = sqrt(sq(dx)+sq(dy));

//Ratio of the x & y components of displacement
floating r = dx/dy;

//x & y components of the unit length; effectively calculated for
different //slope values
uDx = sqrt (sq(uD)/(1+sq(1/r)));
uDy = sqrt (sq(uD)/(1+sq(r)));

// Calculates number of whole segments that fit between the pucks

int n = int(d/uD);
int a = sizes;
smooth();
fill(255, 255);
stroke(225, 0, 0);
strokeWeight(5);

//Draws highlights for the puck positions
ellipse(puckList[1].x, puckList[1].y, uD, uD);
ellipse(puckList[2].x, puckList[2].y, uD, uD);

//Loop structure to draw multiple shapes looping 'backwards' ensures
//the 'last' element is to foreground

for(int i = n; i >= 0; i--){
    //Conditional statements ensure free-positioning of pucks
    if(puckList[2].x > puckList[1].x){
        pointList[i].x = (puckList[1].x + i*uDx);
    } else{
        pointList[i].x = (puckList[1].x - i*uDx);
    }
    if(puckList[2].y > puckList[1].y){
        pointList[i].y = (puckList[1].y + i*uDy);
    } else{
        pointList[i].y = (puckList[1].y - i*uDy);
    }
}

123
pointList[i].y = (puckList[1].y + i*uDy);
}
else{
    pointList[i].y = (puckList[1].y - i*uDy);
}

if (i !=0) {
    // Shapes not drawn at < 1 unit length
    int rd, gn, bl, alph, thick;
    rd = 0;
    gn = 255-(30*i);
    // Darkens successive elements
    bl = 0;
    if (i==n) {
        // Last element has opacity of about 60%
        alph = 155;
        thick = 7;
    } else {
        alph = 75;
        // Other segments are more translucent
        thick = 0;
    }
    fill(rd, gn, bl, alph);
    stroke(255, 0, 0);
    strokeWeight(thick);
    float seed = periodicity;
    float r0 = random(seed, 1.0);
    float r1 = random(seed, 1.0);
    float r2 = random(seed, 1.0);
    float r3 = random(seed, 1.0);
    // Need to convert detail to integer for use in bezierDetail() method
    int idetail = int (detail);
    bezierDetail(idetail);
    float arX1 = pointList[i-1].x;
    float arY1 = pointList[i-1].y;
    float arX2 = pointList[i].x;
    float arY2 = pointList[i].y;
    if (i != n) {
        // These are the bezier curves that make up the shapes
        bezier(arX1, arY1, arX1, arY1 - (a*i), arX2, arY1 - (a*i), arX2, arY1);
        bezier(arX1, arY1, arX1, arY1 + (a*i), arX2, arY1 + (a*i), arX2, arY1);
        bezier(arX1, arY1, arX1, arY1 - (a*0.75*i*r1), arX2, arY1 - (a*0.75*i*r1), arX2, arY1);
        bezier(arX1, arY1, arX1, arY1 + (a*0.75*i*r1), arX2, arY1 + (a*0.75*i*r1), arX2, arY1);
        bezier(arX1, arY1, arX1, arY1 - (a*0.5*i*r2), arX2, arY1 - (a*0.5*i*r2), arX2, arY1);
        bezier(arX1, arY1, arX1, arY1 + (a*0.5*i*r2), arX2, arY1 + (a*0.5*i*r2), arX2, arY1);
        bezier(arX1, arY1, arX1, arY1 - (a*0.25*i*r3), arX2, arY1 - (a*0.25*i*r3), arX2, arY1);
        bezier(arX1, arY1, arX1, arY1 + (a*0.25*i*r3), arX2, arY1 + (a*0.25*i*r3), arX2, arY1);
    }
    // This is the shape element nearest puck #2 that changes size to fill the gap // to the last unit-sized shape element
    else {

bezier(arX1, arY1, arX1, arY1 - (a*i), arX2, arY1 - (a*i), puckList[2].x, puckList[2].y);
    bezier(arX1, arY1, arX1, arY1 + (a*i), arX2, arY1 + (a*i), puckList[2].x, puckList[2].y);

    bezier(arX1, arY1, arX1, arY1 - (a*0.75*i*r1), arX2, arY1 - (a*0.75*i*r2), puckList[2].x, puckList[2].y);
    bezier(arX1, arY1, arX1, arY1 + (a*0.75*i*r1), arX2, arY1 + (a*0.75*i*r2), puckList[2].x, puckList[2].y);

    bezier(arX1, arY1, arX1, arY1 - (a*0.5*i*r2), arX2, arY1 - (a*0.5*i*r2), puckList[2].x, puckList[2].y);
    bezier(arX1, arY1, arX1, arY1 + (a*0.5*i*r2), arX2, arY1 + (a*0.5*i*r2), puckList[2].x, puckList[2].y);

    bezier(arX1, arY1, arX1, arY1 - (a*0.25*i*r3), arX2, arY1 - (a*0.25*i*r3), puckList[2].x, puckList[2].y);
    bezier(arX1, arY1, arX1, arY1 + (a*0.25*i*r3), arX2, arY1 + (a*0.25*i*r3), puckList[2].x, puckList[2].y);
}
}
}

//End of loop structure//

// Setter functions

#endif

void setxDataIn(int newxDataIn) {
    xDataIn = newxDataIn;
}

void setyDataIn(int newyDataIn) {
    yDataIn = newyDataIn;
}

void setAngleDataIn(float newAngleDataIn) {
    angleDataIn = newAngleDataIn;
}

void setFiducialId(float newFiducialId) {
    fiducialId = newFiducialId;
}

void setString(String newString) {
    string = newString;
}

void setSize(int newSize) {
    size = newSize;
}

void setDetail(int newDetail) {
detail = newDetail;

void setPeriodicity(float newPeriodicity) {
    periodicity = newPeriodicity;
}

//These are the classes stored in the arrays

class Puck{
    float x;
    float y;
    float angle;
}

class Point{
    float x;
    float y;
}

//These are the classes stored in the arrays

class Puck{
    float x;
    float y;
    float angle;
}

class Point{
    float x;
    float y;
}

//These are the classes stored in the arrays
Appendix C.  List of Max/MSP Objects

C.1  Max Objects

The following is a list of the Max objects used in the various Max/MSP patches implemented in the project.

-  Arithmetic operator. The output is the left input minus the typed argument or right input.

+  Arithmetic operator. The output is the left input plus the typed argument or right input.

*  Arithmetic operator. The output is the left input multiplied the typed argument or right input.

/  Arithmetic operator. The output is the left input divided by the typed argument or right input.

==  Relational operator Output is 1 when the left input is equal to the typed argument or right outlet, 0 if not.

>=  Relational operator Output is 1 when the left input is greater than or equal to the typed argument or right outlet, 0 if not.

abs  Outputs the absolute value of the input.

append  Adds arguments to the end of message.

button  Outputs a bang when clicked or when any input is received.

bucket  An n-stage shift register that passes input along its outputs from left to right.
coll Stores a collection of ints, floats symbols or lists and assigns them an
address to enable retrieval.

expr Evaluates a C-type mathematical expression.

ftom Converts frequency value to midi note number.

gate Messages received in the right inlet are passed out the outlet
specified by the number in the left inlet.

hypot Calculates the hypotenuse from two side lengths. Included in the Jasch
library.

if Conditional statement in if/then/else form. The message after the
then or else portion of the arguments is sent out the outlet if the
condition is met.

inlet In a sub-patch sends out whatever messages it receives through patch
cords from the patch that contains it.

int Stores an integer data type. An integer received at the left inlet is output.
An integer received at the right inlet is stored and output after a bang is
received to the left inlet.

loadmess Sends out a message argument when a patch is loaded.

mean Finds the running average of a stream of input numbers.

message Sends any message Objects can be programmed to respond to message
input.

metro Outputs a bang at a regular interval specified by argument or left inlet
value.
**mt0f**  Converts a midi note number to frequency.

**number box**  Display and output a number (float or int versions).

**nth**  Outputs the nth member of a list received at the right inlet when n is received at the left.

**outlet**  Sends data out from a sub-patch to the patch that contains it.

**pack**  Combine numbers or symbols into a list.

**patcher**  Creates a sub-patch within a patch. Numbers of inlets and outlets are decided by the numbers of inlets and outlets within the sub-patch's patcher window.

**prepend**  Adds the argument specified to the start of a list.

**random**  Outputs a random number between 0 and argument or right input.

**route**  Pass the input out a specific outlet when it matches an argument.

**scale**  Maps an input range to an output range.

**speedlim**  Limits the speed that messages passes through based on argument specified in milliseconds.

**timer**  Measures and reports the time elapsed between events received at the left, and then right inlets.

**togedgedge**  Sends out a bang when input stream has a zero transition. Positive direction transition bangs left outlet, negative bangs right outlet.

**toggle**  Switches between on (1) and off (0).
trigger\t Sends input to many outlets in right to left order. Data type of outlets is set in arguments.

unpack Breaks a list up and sends to outlets depending on arguments.

C.2 *MSP Objects*

The following is a list of the MSP objects used in the various Max/MSP patches implemented in the project.

+ ~ Add one signal to another.

* ~ Multiply one signal by another.

cycle~ Table lookup oscillator that outputs a periodic signal at the specified frequency.

ezdac~ Audio output with on/off switch.

line~ Generates a signal ramp that can be used as an envelope.

sig~ Outputs a constant value in the form of a signal.
Appendix D. List of Processing Methods

The following methods are described as used in the project's various Processing sketches. A more full description is available in the environment's documentation reference or at <www.processing.org/reference>.

background() sets the colour used for the background of the Processing window. In the draw() function, the background colour is used to refresh the display window between frames.

beginShape() begins recording vertices for a shape.

bezier() Draws a Bezier curve on the screen. Parameters are 1st anchor point, 1st control point, 2nd control point, 2nd anchor point.

bezierDetail() Sets the resolution at which Bezier curves display.

draw() Called automatically directly after setup() and continuously executes the lines of code contained inside its block until the program is stopped.

ellipse() Draws an ellipse in the display window. Parameters are x and y coordinates and horizontal and vertical sizes.

ellipseMode() Specifies the origin of the ellipse drawn. Parameters such as CENTER specified the mode used.

endshape() companion to beginShape(). All image data defined since the previous call to beginShape() is written into the image buffer.

fill() Sets the colour used to fill shapes.

frameRate() Used within setup() to specify the number of frames to be displayed every second. Parameter is fps.
int() Converts a primitive data type, string, or array to its integer representation.

noFill() Disables filling so shape keeps background colour.

noSmooth() Draws all geometry with jagged (aliased) edges.

noStroke() Disables the drawing of stroke outlines.

random() Generates random numbers.

setup() Called once when the program is started to define initial conditions (screen size, background colour etc.).

smooth() Draws all shapes with smooth, anti-aliased edges.

stroke() Sets the colour used to draw outlines and borders around shapes.
References


Mc Murray, G. *Meaning associated with the phonetic structure of unfamiliar foreign words.* Canadian Journal of Psychology, 1960, 14: 166-174


Steinberg Media Technologies GmbH. *Nuendo Software*. 
