Reorganizing Data Flow

Takayuki Hamano

2010

Acknowledgements

I would like to express my grateful appreciation to:

My mentor Paul Berg, who with advice and discussion helped me improve throughout the entire two years of my study in The Hague.

Kees Tazelaar, Joel Ryan, Johan van Kreij, Peter Pabon, and Richard Barrett for their valuable comments and support.

René Genis for helping me polish my English.

Nozomi Natsuyama for performing one of my pieces several times.

My family and all my friends for their love and support.

Takayuki Hamano
The Hague, May 2010

Contents

Chapter 1

Introduction

This thesis presents a real-time system for interactive media art creation, which I have developed during the master degree program at the Institute of Sonology in The Hague.

Talking of my personal artistic experience, I consider myself very lucky to have had a lot of chances to think about music performance, especially with regards to the acquisition of playing instruments, because I have had piano lessons since I was 3 years old. In addition, I have studied composition since I was 15.

As for my technical experience, I have to start with my introduction to using a computer when I was 10 years old. Among many experiences during that time, the most impressive was getting acquainted with the programming language LOGO. It was made by computer scientist Seymour Papert, at the MIT, and he construed it as a tool to improve the way children think and solve problems (Harvey, 1997). It was my trigger to learn about computer programming and ultimately I became enthusiastic about making artistic creations with the aid of a computer.

That circumstance spontaneously made me dream about becoming a media artist. After I entered Kunitachi College of Music in Japan, I have studied digital sound processing and real-time image processing. I feel very fortunate that I was facilitated to produce a variety of creations, such as live-electro acoustic pieces, performance pieces using physical motion detection algorithm and installations for digital signage advertisement using infrared LEDs and an infrared video camera.

Following this, I decided to study at the The Hague, Institute of Sonology. The two years there gave me a great opportunity to integrate my interests and achieve the result laid down in this thesis. It was my ambition to consider art creation through software engineering. As a result, I developed a software program with three dimensional visual representation which, I expect, will contribute technological advantages to those people who engage in creating media art.

This thesis adresses the following big issue: how can we create a technological environment in which a truly novel work of art can be produced? It is possible, that this is too wide a question and nobody would ever be able to answer it exhaustively. Nevertheless, I would like to discuss the viewpoints at the basis of the development of my program in an attempt to get nearer to producing an answer to this question. In the first part of this thesis, I would like to explore the general procedure of art creation. Then, the discussion turns to how we can pursue creativity by utilizing a digital interface. Along those lines, a detailed specification of the design of my program will be given. Furthermore, since the system is intended to be applied to many styles of art, some examples of its application are presented in the last part.

Although the term DESIGN is used here in the meaning 'to change how something looks', it also encompasses the realm of program structure and so tackles the matter of how data should be processed in the program. Whenever I deal with making a computer program, all information is treated as digital data. In order to form interactivity between digital data and users, various discussions from different directions will emerge. Therefore, my treatment of the matter includes issues of importing, processing, and exporting data, through analyzing data flows and forming a new model in the domain of art programming. I call this REORGANIZING DATA FLOW.

Chapter 2

Approaches

The primary question of this thesis is: how can we create a digital environment in which a truly novel work of art can be produced? In a first attempt to clarify this matter we can rephrase this question thus: how can we inspire creators of art to imagine the possibility of a variety of ways to pursue their creative experiments? Even though the goal of my project was to produce a computer software program, discussing the concepts that lie at its basis will be helped greatly by first examining the ways in which conventional, non-digital art is created. Therefore, I would like to start with creativity in the domain of non-digital art.

2.1 Creativity

Medium and Interface

I have played the piano since I was a little child. In Japan, one usually learns on pianos made by domestic manufacturers. I was no exception and got used to these instruments. For a long time I did not know any other kind of piano. This is the reason, why I cannot forget my huge surprise when I first played a piano manufactured by Steinway & Sons at a student recital. I didn't feel any unnecessary resistance when hitting the keys, and the speed of the touch was accurately reflected in the sound. On the cheaper pianos it had been far more difficult to get the power of the finger strokes to sound accurately. At that moment, I was really inspired and I wanted to try all kinds of music on that instrument. A keyboard on the piano is, as it were, a kind of interface which connects human action

with a product that can be qualified as art. An interface that inspires our imagination and creativity lets us want to engage in trial and error. This is my ultimate goal.

Motives for creative acts can be as diverse and unique as the number of artists. Some people are inspired by a religion or particular social circumstances. Other people are driven by a personal passion. Though it is too challenging to generalize about the many sides of art, several common facts about motives for creation can be derived from conventional arts.

The first fact I would like to mention, is that art is the act of circulating a sensation. Putting it very simply: what happens in society when someone makes a piece of art? For one thing, works of art can be considered to be the result of the dialogue between the artist and his own sensations, even though there are many exceptions to this. Such resulting work will appeal to the senses of other people which in turn may urge some creators to produce further pieces. Those pieces will motivate yet other creators. Therefore, art can be considered to be an act of motivating others to create by appealing to their senses, and so it circulates repeatedly through society. Each occurrence of this common phenomenon, such as it actually takes place in society, is supported by: a MEDIUM and an INTERFACE. A medium is something that connects the senses of one person to those of another. In case of music, musical instruments are a type of medium, because they interact between human physical actions and the resulting sounds in the air. Of course, the air itself can be also said to be a medium in this case, although it is believed that it is quite hard to control the air. Anyhow, in a more abstract sense, musical instruments can be considered to be the medium within the totality of the process of this particular artistic activity, not just as a tool to achieve a specific purpose.

The next fact is that a medium of art should itself have sufficient potential and ability to facilitate the expression that is produced as the artistic result. The process of giving such a potential to the medium is divided into two steps. The first step is to observe a phenomenon in the world around us. The next one is to make a new model, in order to reproduce the phenomenon based on the observation. For example, to return to the analogy of musical instruments, people in ancient times found objects in the natural world that could be used to make sounds, such as reeds, animal horns, strings, etc. Perhaps,

these things were not at first intended to be used to make music. Later, over time, people improved their instruments to be able to provide controllability that makes for a more efficient result. Haptic media like musical instruments usually have an interface to allow artists and creators to control it through direct touch. Most kinds of tools, like brushes for painting and keyboards on the piano can be included in this realm.

Symbolized Media

Apart from such artifactual media as discussed above, we have to mention so called symbolized media. For its artistic activities, society needed methods to share and to hand down to future generations. This requirement led to the invention of symbolized media, that allow works of art to be reproduced. In this broad sense, western society refined a particular symbolized medium to the highest degree for its music over a long period of time. Musical notation was invented to write down pitch, duration, rhythm, tempo, dynamics and and so on with symbols. In western music, it was important to be able to reproduce and share this art through a notated musical score. It became a common language for players and composers. However, musical score can only convey limited information and passes over many elements that are present in musical performances. This restriction has brought a certain degree of efficiency and composers have produced new ideas based on the rule of scores, while the rule of notation itself has been continually improved. What is interesting here is that, in comparison with European music, Japanese traditional music, e.g. Gagaku, considered the oral tradition to be the more important and as a consequence did not tend to produce musical scores. Therefore, there are many Japanese cultural expressions that have no historical record. This concerns not only music but also drama. European musical culture on the contrary, had a way of writing scores that can be considered to be a medium in artistic activities.

In his dissertation on interfaces, A. J. Bongers succinctly put it as follows:

> An interface is a connection between things. In its simplest form, it can be seen as a line - the line that separates one thing from another. In order to be meaningful however, it will stretch itself and reach out into both domains on either side of the line, and link things together. (Bongers, 2006)

According to this then, interfaces can be said to be the representatives of relationships. The metaphor of medium and interface is significant also to explain the first side of the process of artistic creation which I described. It is an important fact that our imagination can be carried by them. In other words, the DESIGN determines the affordance of the creator. We need to develop the concept of design a little further. In the domain of digital arts, many kinds of electronic devices can become a medium. In them, information is not even a symbol as it is internally treated simply as a sequence of signals consisting of 0 and 1. It may be appropriate to share and preserve information, but the more important matter is how to abstract native computer information into a form we can deal with. Design, then, is the process to solve that, and it is also strongly desired that particular designs stimulate the creativity of artists.

In the next chapter as well as in chapters following that, I will pick up several issues which are, in general terms, essential when designing a digital art medium with the intent, ultimately to synthesize matters and realize a developing program with visual representation for such artistic, creative purposes as I outlined.

2.2 Design

From this section onwards, various kinds of issues concerning the design of a digital medium and interface are explored. During this exploration, matters of graphical design as well as programming design are mixed together inextricably; in fact, they cannot be separated. Therefore, I have chosen to describe them in a chronological order.

2.2.1 Graphical User Interface

First of all, I would like to take a look at the history of GUI's, which stands for Graphical User Interface. A GUI is an important part of general-purpose computers with a display device. Recognizability and controllability of a GUI allows the user to carry on comfortably. The history of GUI's is like a journey of exploration to invent ways to manipulate commands on computers intuitively. Such intuitiveness may provide us with an efficient way to achieve specific tasks. However, there still are some instances for which the most intuitive way is not always the most creative way. This tends to play a relatively more im-

portant role in the domain of creating art. I would like to pursue this point a little further.

Although the origin of user interfaces in computer science dates back to the very time of the invention of the first computer with its physical (i.e. non-virtual) buttons and knobs, the earliest stage of GUI occurred some 50 years ago. Douglas Engelbart, a researcher at the Stanford Research Institute, invented a multiple window system with a cursor manipulated by a mouse in the 1960s. In the early 1970s, this invention led to the development of Xerox PARC, which was the first system to incorporate the desktop metaphor, using windows, icons and menus to manipulate file systems, much in the same way as many operating systems today still use (Smith & Alexander, 1999). It is said that most of the general-purpose GUI's developed later, were influenced by the principles of this system.

The GUI-system invented in the early days has been in use for a long period, until today. This fact implies that it has strong benefits. The main reason for this is that this system is based on the idea of WIMP, which stands for "window, icon, menu, pointer", which term was coined by Merzouga Wilberts in 1980 (Booth, 2008). This method reduces the cognitive burden of learning when we process tasks on it. For instance, the relationships between file structures can be represented by windows and icons, and the user can then choose a command from a list of options shown in a menu by pointing with his/her mouse. It is so very intuitive, because this visualization helps the user to imagine that s/he were manipulating actual objects in the real world. This important idea is strongly related to OBJECT-ACTION INTERFACE which is described next.

The Object-Action Interface model

It is a common requirement - especially on personal general-purpose computers - to convey one's sense directly into the digital device one is working with, through an interface. The OBJECT-ACTION INTERFACE (OAI) model, an extension to GUI, is a concept aimed at achieving this. Amir Khella (2002), a program manager working on the user interface for Microsoft Expression, explains this concept clearly by comparing it with the ACTION-OBJECT INTERFACE (AOI), which is the opposite of OAI.

There are two basic interaction models for any given system :

- Object-Action model : The user first selects an object and then selects the action to be performed on the selected object
- Action-Object model : The user first selects an action to be performed and then selects the objects on which this action will be performed.

Both OAI and AOI divide the task into selecting the target object and selecting the target action. OAI is mostly employed to represent data through the graphical metaphor of the workplace. As Khella also describes, the contrast between these different models can be seen in command based systems (e.g. Unix) and direct manipulation GUI environments (e.g. Windows.) respectively. He gives an example of the task to copy a file from a directory or folder to another location.

> In the command based system which is based on the Action-Object model, the user starts by specifying the action to be performed, which is 'copy' in our case, next he specifies the objects on which this action will be performed, which are the file name and the destination folder. In contrast, a user on a GUI based environment like windows chooses the object by clicking on the file name, and then performs the copy action by dragging the file from the current folder to the new folder. (Khella, 2002)

Before the time GUI was invented, AOI was rather more main stream and was akin to UNIX-shells or its command prompt. As composing a metaphorical space, like OAI, expenses a considerable amount of memory as it needs to load data in advance, it was difficult to realize at this early stage. This problem was solved by the improvement of processing units and the main stream has shifted to OAI, such as on the personal computers we use today. Although OAI is now in common use, it is doubtful that it is always the better system for creative processes. In my opinion, AOI still has advantages in some cases.

The predominancy of OAI will become uncertain by taking an interesting example from linguistics. It is said that, structurally, English and Japanese are as far apart as two languages can possibly be. Grammatical concepts are wholly different and cannot

be compared easily, but the order of words in typical sentences can be exemplified in a quite straightforward manner. The following sentence is rendered in the two respective languages, which then demonstrates the difference in the way of thinking:

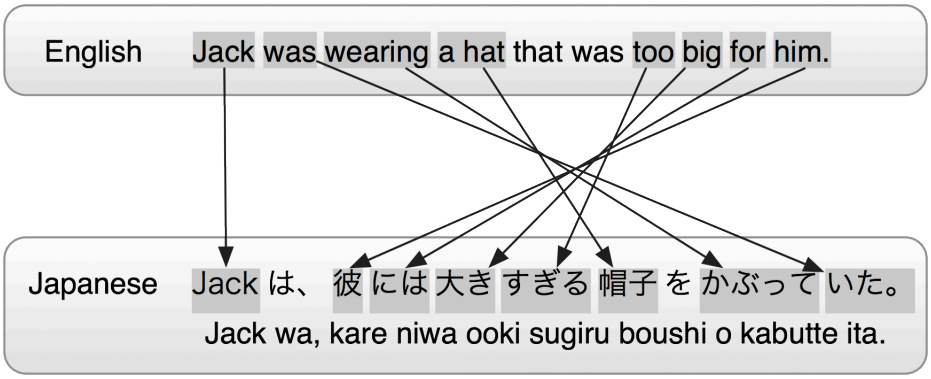

Figure 2.1: Comparison of grammatical structure between English and Japanese (The original sentence in English is an excerpt from *Essential Grammar in Use* by Raymond Murphy (Murphy, 2007))

Both of them start with the subject *Jack*, which in English is followed by the finite verb *was wearing*, and then the object *a hat*. In Japanese the object and verb appear in reversed order. This difference can be directly compared to the issue of OAI versus AOI. *Verb* and *object* then correspond to *command* and *target object* respectively. This may be considered to merely be a difference of custom or habit in the way of thinking to which the user is accustomed. Therefore, it is hard to say which of OAI and AOI is superior. What is truly important here, though, is that the notion of abstracting tasks as an object and an action like WIMP, can reduce our cognitive load when we manipulate tasks, more evidently than the comparison of OAI and AOI. This idea, to represent something clearly in the most comprehensible way for us, is afterwards extended to be applied to programming languages which I will describe next.

2.2.2 Programming Paradigms

Programming paradigms determine the elements of a programming language and their composition. Just like GUI, programming languages have been improved for the sake of usability. There are many kinds of programming paradigms such as procedural programming, imperative programming, declarative programming, and so on. Each programming paradigm involves many abstractions. I would like to examine this by picking up the example of SuperCollider.

Abstraction in Programming Languages

SuperCollider is a environment and programming language for audio synthesis and algorithmic composition originally developed by James McCartney. He explains the aim of SuperCollider:

> A computer music language should provide a set of abstractions that makes expressing compositional and signal processing ideas as easy and direct as possible. The kinds of ideas one wishes to express, however, can be quite different and lead to very different tools. If one is interested in realizing a score that represents a piece of music as a fixed artifact, then a traditional orchestra/score model will suffice. Motivations for the design of SuperCollider were the ability to realize sound processes that were different every time they are played, to write pieces in a way that describes a range of possibilities rather than a fixed entity, and to facilitate live improvisation by a composer/performer. (McCartney, 2002)

The specification of the language SuperCollider is similar to Smalltalk. What McCartney also explains, is that the SuperCollider language consists of many programming abstractions such as variables and functions with a name, control structures, argument passing, data structures, garbage collection, threads, and so on. In SuperCollider, each function of sound synthesis is provided as a unit generator (UGen), and they can be connected in such a way as to create a network by themselves. In particular, the features of object-oriented paradigm and dynamic typing flexibly enable one to construct parts of a program on a more abstract level. In this way, the characteristics of each programming

language are based on the choice of the kind of abstractions that are taken into the specification, but also by the construction, which determines how the abstractions are composed in language.

Object-Oriented Programming

Object-oriented programming (OOP) is a programming paradigm which was famously adopted for the Java programming language. In it, the element that composes the Object-Action Interface Model is given the status of object. This is because OOP has objects for minimum units of compositional elements. The most remarkable feature of OOP is its encapsulation. Encapsulation is the principle of hiding information about an object and its behavior. Grady Booch defined encapsulation as "the process of compartmentalizing the elements of an abstraction that constitute its structure and behavior; encapsulation serves to separate the contractual interface of an abstraction and its implementation." Apart from encapsulation, there are some abstractions which shape OOP: e.g. *inheritance, polymorphism and dynamic binding*. The bigger and more complicated a given project is, the more these characteristics of OOP aid in dividing the whole program into reusable parts.

What is interesting about OOP, is that not only data, but also actions and relationships between objects can be represented as objects. This characteristic can be seen in the *design pattern*, which is a pile of procedural knowledge about software design, that was devised by many past programmers. This design pattern originates from the book *Design Patterns: Elements of Reusable Object-Oriented Software* published by its programmers, the so-called 'Gang of Four', in 1994. They devised collective solutions to common problems in software design and called them DESIGN PATTERNS. Their solutions originally consisted of 23 patterns. From those patterns can be derived that all functions and data of a program are encapsulated and represented as objects; it often seems that they are, in a way, personified.

From the historical viewpoint of software engineering, one could say that shifting to new programming paradigms has always changed the way in which progress was made. Just as the object-oriented programming paradigm provoked innovative productivity for

large scale projects through its technical approach, reorganizing technical paradigms in the context of media arts will also give many possibilities to a broader range of artists.

Visual Programming Environments

Visual programming environments are also a topic for discussion. There are two distinct area's in this matter: visualizing program flows and visualizing data. The modular and node-based programming paradigm is commonly used in the programming of environments for musical and visual art. This paradigm is employed by Max/MSP, Quartz Composer, Node Editor in Blender, and so forth. Here, I would like to discuss matters on the example of Max.

Max is a visual programming language for music and multimedia, which is widely used in the domain of media art. It is also called Max/MSP to express its combination with the enhancing add-ins for sound synthesis on Max. With regards to programming paradigms, Max is a highly modular language. Most programming elements in Max are encapsulated as 'objects' that have inputs and outputs. Among the objects, there are also many GUI-parts for manual data control. A program in Max is made by inputting objects and connecting them in a PATCHER, i.e. a virtual canvas. The connection of objects in the patcher represents the programming data flow. Compared to ordinary coding, this feature has the great advantage that the programmer can trace the data flow very easily. In particular, digital signal processing can benefit from this advantage. Other environments, like Quartz Composer, have borrowed this from Max.

At first glance, Max looks like it employs an object-oriented programming paradigm, but actually, it doesn't. For example, Max cannot let an instances be inherited from other abstract objects, which is possible in SuperCollider. In addition to this, objects are processors that take data and generate new data as a result. Thus, data itself cannot be an object in Max. Max clearly distinguishes data and programming functions, unlike OOP that looked for seamlessness to fuse many kinds of programming elements. Miller Puckette, one of the developers of Max, acknowledged that Max is oriented toward processes more than towards data (Puckette, 2002). Max gained an advantage by sacrificing some programming abstractions in order to simplify the paradigm for visual representation.

Incidentally, a detailed history of GUI in computer music is given in the book by Curtis Roads (1996).

Tangible Bits

Nowadays, design in computer science is not restricted to the GUI or programming paradigms that I described, but has extended to physical human interfaces. One new current is the research into tangible interfaces. This idea was introduced with the research of Hiroshi Ishii, a Japanese tenure professor at MIT. He has invented a new human interface device that can be manipulated by hands, which he refers to as *Tangible Bits* (Ishii & Ullmer, 1997).

An interesting idea of tangible interfaces is that data is represented by an actual object. The user can touch and control this data directly. In this model, the user's senses, his actions and their results coincide with the actions that emerge in the same place simultaneously. In some tangible interfaces, physical laws are directly applied by handling the actual objects. This is possible because everything in daily life behaves according to the laws of physics and we are all aware of these laws as we learn them in a physical way. Making use of this feature, the tangible interface provides users not only with exact data, but also conveys a tangible impression of the data to make them understand in a more physical way.

An even more remarkable achievement by Prof. Ishii is the realization of vision employing artifacts that we an experience in daily life. As programming languages evolved and became recognizable for humans as a means of expression, he recomposed the technical architecture into a metaphorical system in which we don't have to be aware of the techniques employed. This is his approach to intuitiveness.

In April 2010 Apple released the iPad; a tablet computer with multi-touch LCD. This recent trend of multi-touch interfaces shows the great current interest in the possibilities of haptic interfaces. Among the many current research projects in this area, the multi-touch screen made by Jefferson Han, research scientist for New York University's (NYU) Courant Institute of Mathematical Sciences, was particularly impressive (Han, 2005). His group produced many examples of the possible application of the multi-touch screen and it has a lot of potential regarding efficiency, usability, and intuitiveness.

Tangible Systems for Music

Tangible interfaces are applied as electronic musical instruments.

For example, *Audiopad* is a tag-based experimental interface for musical performance and creation (Patten, 2002). It consists of a tabletop interface on which performers can manipulate real-time synthesis processes by means of physical input devices called pucks. The position and orientation of these pucks is recognized by means of Radio-frequency identification (RFID). An interesting feature of Audiopad is that users can specify the parameters to be controlled by putting pucks on the tabletop. The pucks, then, represent the parameters. Thus, the manner of controlling parameters through pucks is universal.

This is somewhat different from *Reactable*, developed by Jordà, Kaltenbrunner, Geiger, and Bencina (2005) of the Music Technology Group at Universitat Pompeu Fabra in Barcelona. In my opinion, Reactable is currently the best application for music creation. Programming units for sound synthesis are each assigned to a transparent solid block. These units are programmable according to their proximity to each other on the table.

The two above examples show how levels of parameter abstraction can vary considerably.

2.2.3 3-D Virtual Space

Visualization in 3-D virtual space gives us more possibilities as compared to 2-D or 1-D; as our eyes have the ability to recognize 3-D images more complex information can be represented in 3-D space.

Voxel Representation

Voxel (= volumetric pixel) is one of the available techniques to represent objects in three dimensional space. With voxel, data is represented as points on a regular grid in 3-D space.

Historically, main stream computer graphics have evolved until today using polygon techniques. This is due to hardware limitations which meant that the amount of calculations had to be reduced. In polygon techniques, 3-D objects are made using vertices and filling faces. Various theories, such as texture mapping and bump mapping, are produced based on polygon and have become progressively more complicated. For example, boolean

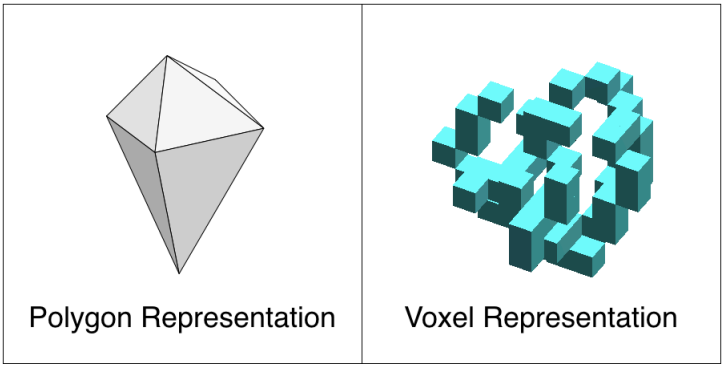

Figure 2.2: Polygon and Voxel representation

calculations of 3-D objects are exceedingly complex, whilst voxel objects can achieve the same result quite easily.

In contrast to polygon, voxel techniques go back to quite simple principles. Polygon only draws the surfaces and textures of an object. A voxel object consists of quite sizable, digital points, each of which has a color. Voxel drawn objects, then, can have their own content as well as a surface. This technique has lately drawn attention in the computer graphics world, as it gets rid of the constraints of polygon.

Voxel has the following advantages:

- Like all actual objects that consist of enormous molecules, voxel represents the shape and volume of an object simultaneously in 3-D space.
- Since a voxel object has its content represented by numerous points, the amount of points can be considered to be the weight of the object.
- A voxel object can be easily modified. Not only rotation, transformation or scaling, but also manual cutting and distortion are done easily and flexibly.

Voxel representation has its strengths and weaknesses just like other methods such as polygon representation and ray tracing. The weakest point of voxel is that it needs a huge amount of point data stored in the storage device. As these devices are continually being improved, it is expected that this problem will also be solved. Solid-state drives

may be one of the possibilities as they are superior in terms of the speed of random access speed. Eventually, if these architectural issues are solved, voxel will begin to have power in the rendering speed. One voxel representation technology, called UNLIMITED DETAIL TECHNOLOGY, treats an algorithm in the same way as does a search engine on the Internet. The engine works out in which direction a given camera is facing and then searches the data to find which points it needs to put on the screen (Dell, 2010). I expect this feature will become a future trend as it realizes a faster representation of 3-D graphics by another trajectory than the ordinary polygon.

The year 2008-2009 was very exciting for people involved in GPU (Graphic processing units) technology: the GPGPU conference was held for the first time. A further reason why voxel representation is expected to spread emerged from there. Nowadays, GPGPU (General-purpose computing on graphics processing units) technology is improving at a remarkable rate. In some cases GPU's perform up to over ten times faster than cpu in FLOPS (Floating point number Operations Per Second). GPGPU is a technique that exploits the power of GPU. CUDA and OpenCL are languages for GPGPU which enables the compilation and running of programs for GPU. It attracts researchers in scientific domains, and sometimes also in medical science. Of course, it can also be applied to graphic techniques combined with shader languages like GLSL, Cg, and HLSL. Environments such as these allow developers to make their own rendering engine and some researchers are trying to realize voxel representations with them. Therefore, I am looking forward to see whether the voxel renderer could be a trend that makes the most use of those technologies.

The concept of voxel representation is taken into my program. Here, voxel is defined as a minimal unit which corresponds to the object in the Object-Action Interface Model (OAI). Although graphic representation will not completely be implemented in terms of real-time graphic techniques, it is the program's goal that the notion of voxels is applied for inventing a new expression.

Controlling 3-D Space

During the development of my program I was confronted with the problem of how to manipulate 3-D space and objects in it. I am thinking here of such manipulations as transformation and rotation. In video games it is quite common to control spatial movement in relation to the object indicating the player's position in 2-D virtual space, even though the actual visualization is realized in 3-D. Then again, almost all 3-D graphic modeling software is designed to be controlled with a mouse, sometimes also with a tablet. The main reason for this is that this kind of program is usually not intended to be used in real-time, such as is the case with live performance. However, there are special devices for modeling software like SpaceNavigator (www.3dconnexion.com) and these provide a better solution.

2.3 Data Mapping

In the Critique of Judgement, Immanuel Kant, the well-known German philosopher, typified beauty as "Zweckmäßigkeit ohne Zweck", which translates into English as "purposiveness without purpose" (and sometimes it is rendered "final without end") (Kant & Klemme, 2007). The following section describes how we can find beauty that can be purposive but arises from something that doesn't have an intended purpose.

Most scientific technologies are invented for a specific purpose, but what should artists do with such technologies? Joel Ryan stated that it is more important how cognitive sense can be made to give results, rather than just applying technologies to art (Ryan, 2003). I will attempt to connect aesthetic motives with actual technology.

2.3.1 Discovering New Values

Human beings are creatures that feel joy when they find new values for something. This principle is what I always take into account when I make a work of art. Taking the method of DATA MAPPING will achieve the realization of works of art. What inspired me to this idea was the book *DATA FLOW* (Klanten, et al). In this book, there are hundreds of pictures visualized by mapping many different kinds of data. This is just an example of

data mapping and in the future data and outcome will be more diverse. For example, making music from the current produced by a school of fish in the ocean could be a kind of data mapping. It is the process of representing something in a different way and so find out new sides of its character. As art actually is an act to cut off aspects of the natural world, increasing the means to recognize occurrences in the nature world, such as data mapping, will help creation.

Randomness

When our nerves receive regular stimuli, they will get used to them and we become numb. In the world of art this fact seems to have provoked a keen interest in randomness, noise, and unpredictablity. In this respect it is interesting to note that the neural signals of our eyes always contain noise.

The noise in these signals has a useful function in the nervous system (Stafford & Webb, 2004). Adding noise to a signal raises the possible maximum signal level and makes it easier to detect. This phenomenon is called stochastic resonance. Our nervous system has evolved to be able to deal with such noisy signals. The brain makes sense of noisy data by canceling out the noise. As such, noise has a close relationship with our body. Our physical system is robust enough to receive and handle this. Adding noisy material to a monotonous stimulus can be a way to enrich its total impression. This may have something to do with our interest in randomness in art. Both in the domain of computer music and computer graphics, chaos and fractals have been big issues for a long time. Allowing the addition of randomness in an environment for artistic creation may inspire artists, whilst any randomized outcome can also be fun and succeed as a work of art.

Simulation

Extending the idea of data mapping, simulation will be the next keyword. Making simulations by analyzing targets and rebuilding simulation models has been adopted as a technique in the field of art creation. Technology for computer music and computer graphics has been invented in order to reproduce or to imitate the real world. It may be legitimate to dream of the computer as being on a par with human beings and thus

extending some kind of reality to it, as long as humans have a conscious and objective awareness of themselves. This dream often has led to technological breakthroughs ...

Deviations from Ordinariness

It is not always true to say that as the number of options increases, creativity improves. Actually, too many possibilities may discourage the eagerness to create. It does seem to be part of human nature to feel well when we chose something of our own accord, as an act of will. There may, then, be a dilemma between the joy of having predominance over efficiency and not restricting ones options. A possible solution we may contribute to the development of a user interface is to construe it in such a way, that the user can actually control optional values concentrating on a single range of aspects at once. In the case of representing tree structure data, the interface should have the ability to show each level from macro to micro.

In addition to this, I would argue for the methodology of adding a deviation on a simulation. There were certain ways of thinking that we are used to. These are historically evolved using reason. I am attempting to give creators an environment where they can reproduce simulations and transcend into a new expressiveness. For example, my program can be just as easily applied to make surround sound as some other programs for music production. However, what is unusual in my program is that users can break the rule of a fixed 3-D space, because axes are not defined from the outset. In this way, users will have more possibilities than with ordinary interfaces.

Chapter 3

Implementation

In this chapter I give a detailed description of the software program 'VDAM' (Volumetric Digital Art Mixer), which I have developed during my studies at the Royal Conservatory of The Hague.

VDAM is a communicator *between*, as well as a convertor of different kinds of art works. Hereafter the term MEDIATOR will be used. It provides a new environment in which users can experiment with producing unexpected, novel outcomes. The program contains a visual programming environment that enables users to manipulate virtual objects in a three dimensional space on screen and control numerous kinds of parameters. It is possible to carry out various experiments of data mapping between different kinds of information through objects in the program. The development process of the program comprised a variety of methodologies such as I described before, including programming paradigms, graphical user interface design and art creation by means of data mapping methods. Since this system is intended for artistic purposes – and not for business or commercial productions - it has the potential to enable inventing new expression in a general sense, rather than to satisfy a specific demand.

Here, I describe the initial stages of the development of VDAM, how it was assembled to realize many ideas and how the program actually works.

Behind the development

In part, the idea of developing VDAM stems from my previous artistic creation 'Saccade', an interactive live performance piece I created in 2008.

The basic concept of this piece was to represent music scores in a manipulable 3-D space. As the ordinary visual MIDI sequencer program has a horizontal timeline with a vertical bar with notes and other symbols, this piece also had these elements. However, the shape of the timeline was automatically bent little by little. As the bar collided with a 'note'-object, it produced sound with a pitch according to the height of that 'note'-object. The performer had a tiny infrared video camera mounted on top of his head to track his body motions. The tracking results were mainly used to manipulate spatial rotation and the position of the bar. What I tried to realize in this piece is to express a contrast between the uncontrollable behavior of the automatically changing score and the elements controlled by the performer, as if they were battling with each other. As the piece went into its concluding part, the program started to progressively limit the performer's control.

Based on the idea behind this piece, I planned to make a program in a more generalized form, allowing it's application to be expanded to many purposes. In this way, a prototype program was created during the first year of my studies, which is the predecessor of VDAM.

3.1 Architecture

The architecture of VDAM is best explained by classifying it into three layers. Here I would like to explain matters by starting from the more general structure and then progressing to more detailed issues.

1. Abstract Layer

The program becomes a mediator by representing data as virtual objects that make seamless connections between a variety of entities like sound, electronic musical instruments, phenomena in the natural world, statistics, user interaction, and any other kind of data (indicated in Fig. 3.1). This is achieved by taking voxel representation as the main principle of the program. Voxel representations give the impression that we are able to manipulate

data as if we were making a sculpture or squeezing clay. This program is intended both for real-time and non-real-time.

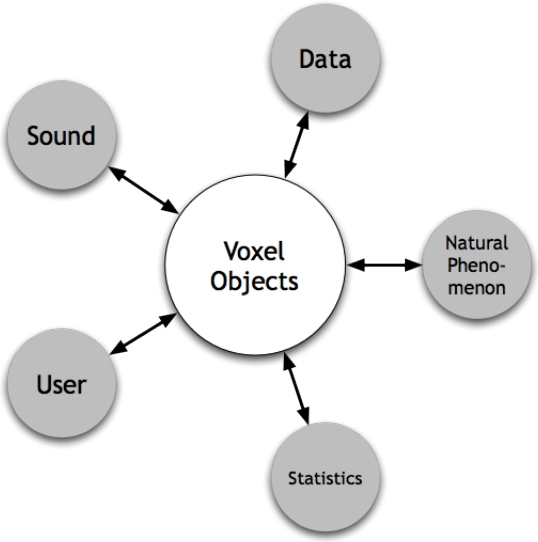

Figure 3.1: Diagram of the system in the abstract layer

2. Technical Layer

The next figure shows the technical layer of how the program is composed and how it communicates with others.

Inputs for the program can be diverse. External devices like keyboard, mouse, pen tablet, WiiRemote are simple suggestions, and the program of course accepts them. File inputs like a standard MIDI file is also accepted. Other ideas include such suggestions as creating a piece from a moving picture of a school of fish. There are two ways to send such kind of data to the program. The first one is by sending it via a network protocol like the open sound control. This enables connections with external applications. The other is through making a new object as a plugin for the program, and give it the functionality to import data.

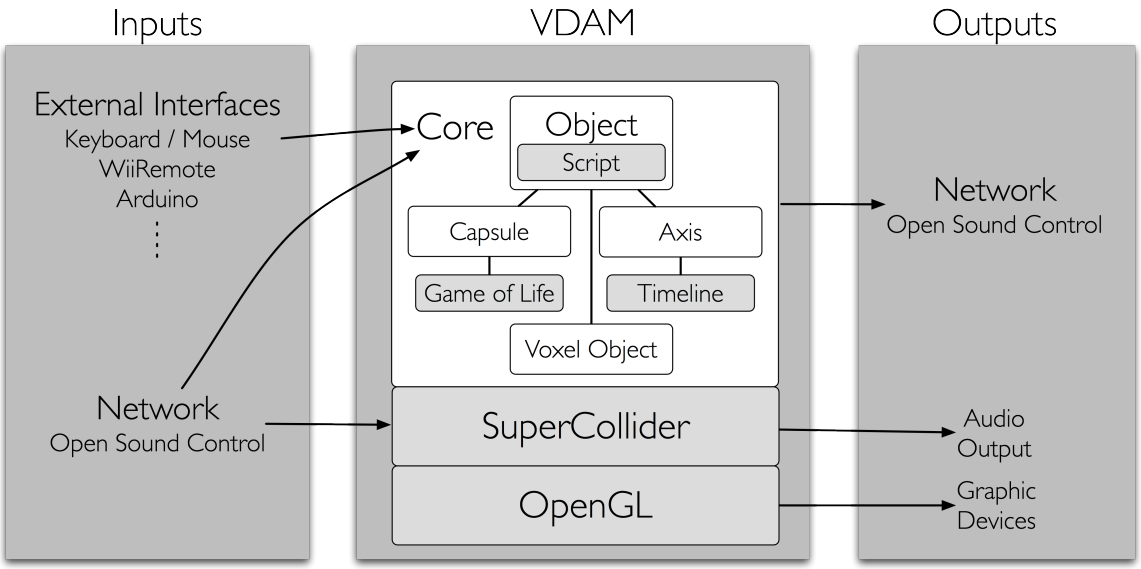

Figure 3.2: Diagram of the system in the technical layer

Imported data is, then, processed by VDAM, and replaced with voxel objects in a given virtual space. The graphical representation is done by means of OpenGL. This means that the program uses the graphics processing unit in such a way that it doesn't interfere with any other tasks on the CPU, like processing sound. Because this program is coded wholly in Java, Java™Binding for the OpenGL®API (JOGL, `http://kenai.com/projects/jogl/`) is used.

Besides, sound engine 'SuperCollider' is integrated in the VDAM and it runs inside it. Users can run SuperCollider codes quickly and achieve a seamless connection between voxel objects and sound generation. It is also possible to write a SuperCollider code into the script of an object,

The sound generated by the SuperCollider is one kind of output from the program. The graphical image of the program is not necessarily intended as a work ofart in itself, but it can also be considered to be just output. Also, transmission via a network is also output, which can be connected to other hardware devices and software programs.

3. Inner Structure Layer

As shown in Figure 3.2, the core of the program consists of several kinds of objects in a hierarchical relationship. Not only voxel objects, but capsule objects and axis objects are children of the top object, which is simply termed 'object'. This hierarchy feature strengthens the principle that objects of different types do communicate with each other.

Each object type has a specific role. Objects can be programmed to work independently, but there may also be objects that only work by collaborating with other objects. Those objects are generated arbitrarily as instances, and always can be controlled to reflect the result in real-time by the user in some way.

In pursuit of simplicity, the connection feature that existed in the former version was abolished. This connection feature enabled the programming of data flow between objects. However, it is no longer needed as the program now supports voxel, which represents data as massive objects. Scrapping this feature allows users to concentrate on the behavior of voxel objects.

3.1.1 Interface

When a user launches the program, only a white screen is shown. Nothing else, not even a timeline, a score editor, or slider is displayed. The screen internally has a three dimensional space and it allows the user to place and transform voxel objects in a variety of ways. In the system, each voxel point is called an 'object', which is handled as a minimum unit representing data. Every object itself can be given attributes determining its position in the space and information about its color. Complex data can be represented as a cluster of objects. Because 'object' is the minimum unit in the system, it cannot be transformed itself. However, clustering objects enables the user to edit (cut, bind, and distort) very easily. The program can be controlled with a mouse, pen tablet, and external MIDI controller.

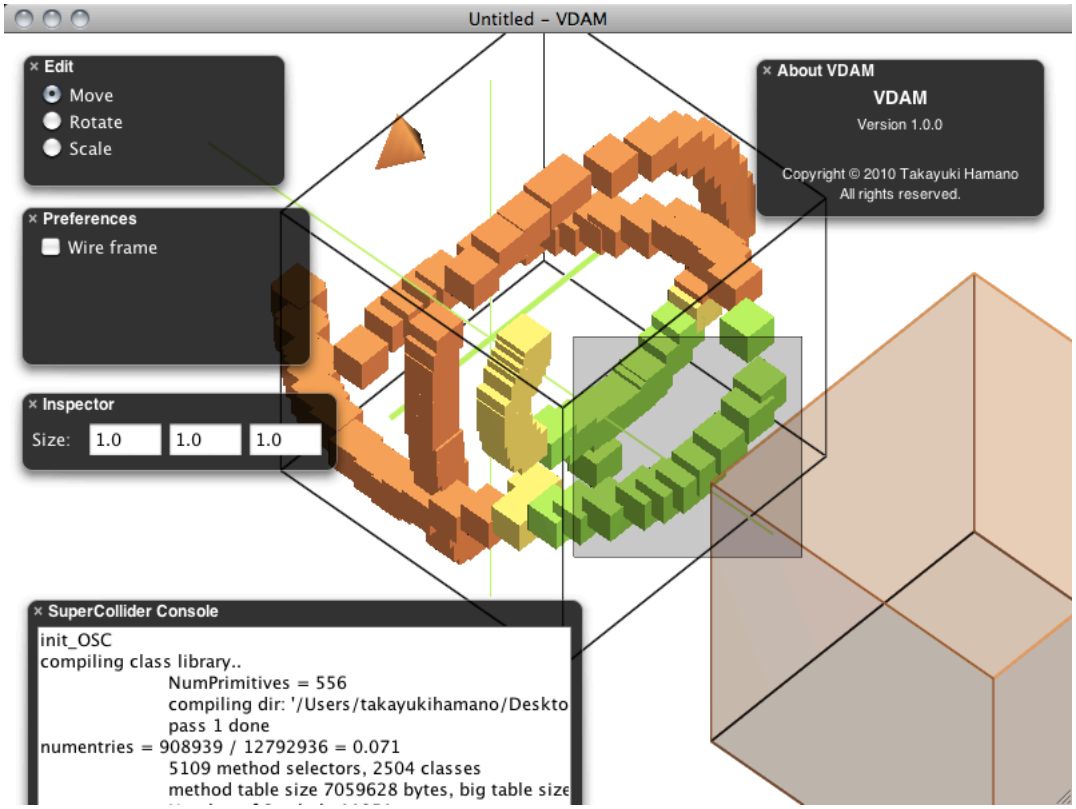

Figure 3.3: Screenshot of VDAM

All objects are written in the Java programming language and can be edited in the script editor incorporated in the system. Written scripts can be compiled and reloaded dynamically at any time using the reflection method of Java. Another benefit of using Java is that objects in the system can be inherited by adding object-oriented program paradigms as a feature.

The program has different states called MODES. Modes can be selected in the mode menu. If the mode is BRUSH, users can draw voxel objects with the mouse. In TRANSFORM mode, manipulations of voxel positions such as moving, rotating and scaling are possible. Various object manipulations can thus be realized; as if the user chooses a tool from his toolbox.

Most objects have properties which are additional settings for the object. Properties are shown in the INSPECTOR as the user selects an object. It can also be changed.

3.2 Object-Oriented 3-D Interface Elements

3.2.1 Objects

As I described before, various kinds of objects are inherited from the object at the topmost level. This latter object determines common functions for all lower level objects but it cannot itself be an instance. In theory, it is possible to make an invisible object that affects other objects.

Voxel Objects

Voxel objects are the most primitive objects. They represents a certain value, and are shown as small cubes on the canvas. Voxel objects can be drawn in brush mode and erased in eraser mode. It is easy to extend the function of the voxel object by making it an inherited class object.

Capsule Objects

Capsule objects are special elements used to encapsulate objects and constrain their movement. Capsule objects also have a script of their own. What is different here from the other kinds of objects, is that the user can write a script defining the movement inside the capsule object and relative to it. For example, the user can script codes to make an object behave along the rule of 'the game of life' which will be described in paragraph 4.4. This object embodies the idea of the Action-Object interface model. Both AOI and OAI are realized in the system and so enable more flexible manipulation. There are three properties appertaining to capsule objects.

The first property is the CONSTRAINTS property. This option affects the movements of voxel objects. Voxel objects cannot move beyond the boundaries of the capsule object when this option is active.

Secondly, there is the SNAP TO GRID property. When this option is on, the position of voxel objects can be placed on the invisible integer grid. Decimals are rounded off.

The last property of capsule objects is the SPACE REPETITION property. This function simulates that the space of a capsule object is reproduced infinitively and they are placed continuously. If the repetition property is turned off, the object in a capsule object

can move over the boundary of the capsule object. If the constraints property is on, it will stops exactly at the boundary. However, if the repetition property is turned on and the encapsulated object reaches to the end of capsule object, it goes back to the other side of the boundary and it will keep moving in the same direction.

3.2.2 Objects for Value Calculation

Voxel objects contain positioning values x, y, and z, as well as color information. There are ways to copy values from other voxel objects by analyzing their complicated state of voxel objects to get an integrated result.

Measure objects

MEASURE OBJECTS are the simplest of the Value Calculation objects. They simply measures the distance from voxel objects to each other.

Axis objects

AXIS OBJECTS are based on the idea of enabling axis transformations freely within the space. They take their value from the crossing point on the axis object where a perpendicular is drawn from a voxel object to the axis object. By combining several axis objects, it is possible to interpolate different kinds of parameters.

Timeline objects

TIMELINE OBJECTS are a variety of axis objects. Like axis objects, timeline objects can be transformed freely within the space. What differs from the axis object is that the timeline object has a certain time scale and it also has a bar that represents its current position in time. When users press the play button in the inspector, the bar starts to move along the timeline. When the moving bar collides with a voxel object, it causes a notification report to the collided object and outputs information about the object. Settings for time scale, beat, and speed can also be changed at any time.

3.3 Scripting

Scripts are bound by all kinds of objects including capsule objects. Scripts determine the behavior of objects. In case of capsule objects, it is also possible to manipulate objects inside the capsule boundaries.

All object scripts are compiled when the user launches the program. Therefore, users can share newly made objects with other users just by copying the source code file. One of the strong features of the program is that users can easily make new objects. As the program contains all environments for compiling codes, users don't need to use any IDE (Integrated Development Environment) with the software development kit. Compilation and reloading updated objects can be done whenever the program is running by merely clicking the update button in the source code editor. This feature became possible through Java's dynamic class loading feature.

As for the graphical commands, users can write legitimate OpenGL codes. This will avoid users having to learn new, divergent ways of scripting.

Chapter 4

Application Examples

VDAM can be used in a variety of ways. This chapter discusses a number of its past applications.

4.1 The Simplest Example

The first example presents a short introduction to a very basic application of VDAM: to produce sounds and control their amplitude.

When the program is launched, it starts with a white screen that represents the virtual 3-D canvas. At the same time, the server of SuperCollider is booted automatically. There are several modes in the program and users can choose one by pressing the space key to show the menu. If the brush mode is selected, objects can be drawn on the canvas using the mouse. In the eraser mode, objects can be removed from the canvas also with the mouse. So, let's draw for example, a large number of objects and call these sound test objects and let's also put in one or two measure objects. Sine tones will be played. The height of each object – actually, a value on Y-axis – is assigned to represent the pitch of the sound. In the edit mode, users can coordinate the position of objects. Let's choose the edit mode and make sure that the item 'Move' is selected in the sub menu. The region to be moved can be defined by moving the mouse on the canvas. Then, users can grab the selected objects, and move them around. The sound will change automatically as the position of the objects changes. Moreover, as the position of measure objects is changed,

the amplitude of each sound will follow accordingly. The distances between the measure object and the sound test objects are calculated automatically, and they are reflected in the amplitude.

To terminate the sound, the user holds the command key and then presses the period key.

How It Works

On condition that both, sound test objects and a measure object, are created, new synths defined in the SuperCollider are generated. The measure object receives the ID of such generated synths and notifies the change of value to each synth. These processes of communication are written into the code of the objects concerned. VDAM features the possibility to select objects on the basis of specific conditions that are attached to these objects. This flexibility is similar to using the wild card character in a command-based interface or in a regular expression. This example features both AOI and OAI, which I described before.

4.2 Sound Space Manipulations with Head-Tracking System

Since my program employs virtual three dimensional space, sound space manipulations are easily achieved. Usually if we try to hear a sound from a specific direction more clearly, we turn our head/body so as to face into the direction of the sound. I have made a system to simulate this in digital sound space.

The system includes an algorithm to track the position of markers in an image taken by a video camera. Markers consist of two filled circles with their borders and are tagged onto the band of a headphone. In this way, the position and direction of the user's head is traced in real-time and reflected onto VDAM. The positional relationship between sound source and hearing point can be simulated in the space in VDAM, as if the user would have placed speakers in real space and were moving around. A head object represents the position and direction of the hearing point, and the result of the marker tracking algorithm affects this object. The head object moves and rotates correlating to the user's head, and calculates the relative distance and direction from virtual sound objects. Finally, a program coded

in SuperCollider executes the necessary physical calculations and produces the resulting simulated sound.

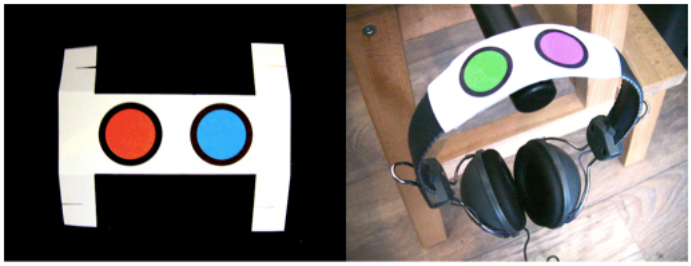

Figure 4.1: Attachable markers to a headphone

This system can be used for monitoring sound space without surround speakers.

4.3 Sonification through Image Analysis

This is one of a number of experiments of digital reinterpretation of non-sound input material. The aim is to produce an sound expression of both still as well as moving image material, and so: make pictures sound. The structure of input images is extracted by an analysis algorithm, and subsequently reformulated into various calculated values.

This feature is realized as a unique capsule object (as indicated in Figure 5.1). The working process of this object is as follows: when an Image Analysis Object is created, it begins to capture screen images in real-time. A captured image is converted to HSB format, which is the abbreviation of hue, saturation, and brightness. Next, a HSB histogram of the image is generated and shown in a preview window. Based on this histogram, feature points, where a large part of similar colors are located in the picture, are detected. Finally, the object outputs values of HSB feature points and their central location. The HSB information is automatically represented as the x-y-z location of voxel objects within the capsule object, whereas the information of the central location is indicated by their color.

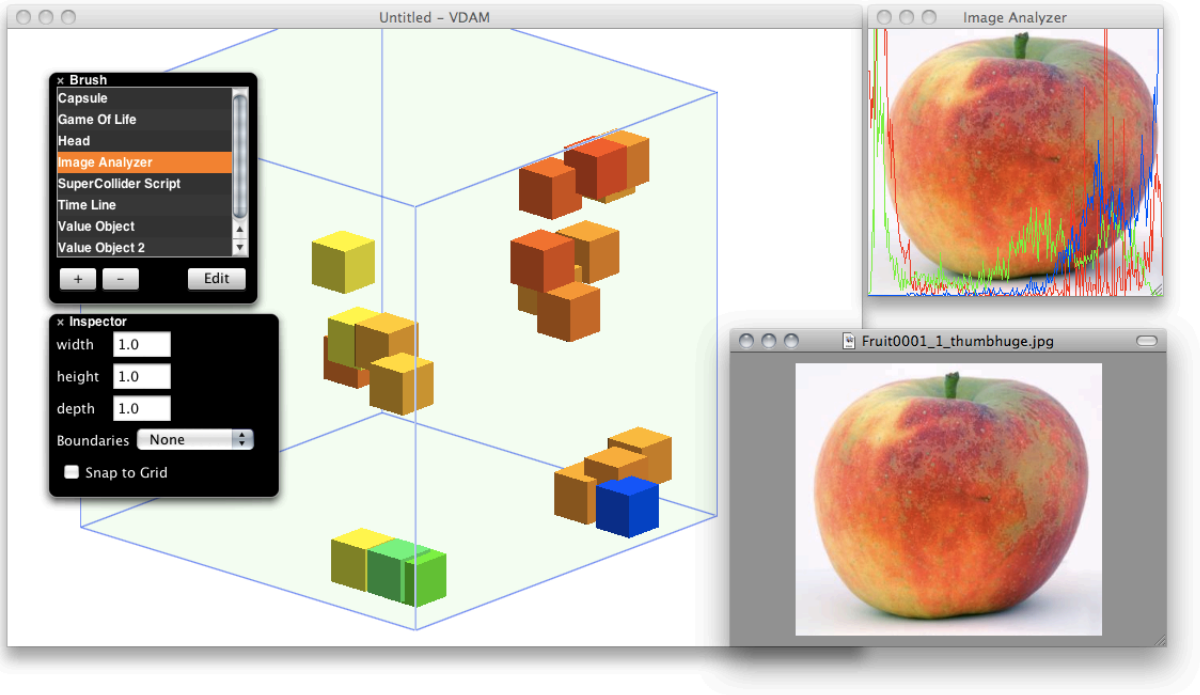

Figure 4.2: Image Analysis Object

Sometimes, when a function needs to work in real-time, a technique optimized for real-time will be used. In the Image Analysis Object, not all the pixels of an image are examined, but the points to be examined are chosen randomly; this is called the MONTE CARLO METHOD. Each round of analyses examines the same number of points. This may not be the best way to get an exact result, but a certain degree of accuracy is guaranteed.

The application of the Image Analysis Object can be extended to other kinds of images. The fact that the system employs the real-time screen capture method, is the feature that makes it possible to import various kinds of images, such as images of live visual performance, network performance with video chatting, artificial computer graphics of physical simulations, and all other kinds of images that can be shown on a screen.

4.4 The Game of Life Simulation

Many researchers in the field of algorithmic music composition have addressed issues concerning Cellular Automata. Cellular Automata are simulations of successive boolean cells in a more than one-dimensional grid space, with rules of how those cells live or die per generation. CONWAY'S GAME OF LIFE is the best-known example of cellular automation. The basic principle of the rule is as follows: if the density around a cell is either too great or too little, the cell dies. In other cases, the cell stays alive. I first attempted to create a cellular automaton as a software application by simply following this original rule. (see the attached CD-ROM). The next step was to extend the principle of the *Game of Life* as a function of VDAM employing two different methods. Both of these are realized as a capsule object with codes that control the behavior of the encapsulated objects.

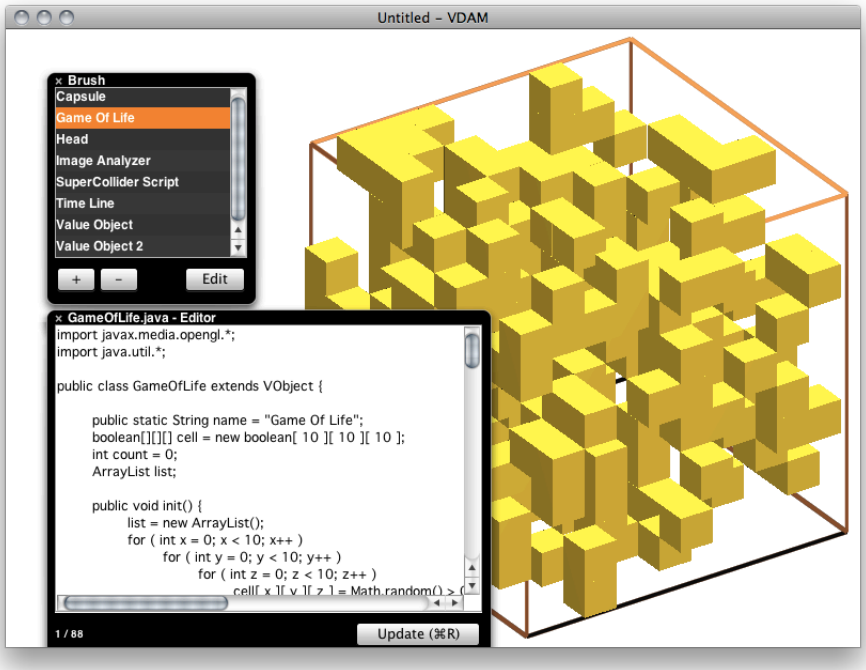

Figure 4.3: 3-D *Game of Life* Simulation

The following is a description of the two methods employed for the implementation of the *Game of Life* principle in 3-D VDAM space.

1. Using Simply an extended algorithm of the ordinary 2-D simulation

The first method works in almost the same way as it does in 2-D. In 3-D VDAM space, the capsule object for this simulation is given the 'snap to grid' feature. Objects that indicate the state of a given cell, be it alive or dead, are placed along grids. Just like in 2-D, the state of a cell is decided by counting the number of surrounding cells. In 3-D VDAM space this should be a total of minimally 26 cells, which is the threshold for a cell to stay alive in the next generation.

This method is realized under the condition that the integer grid setting of the capsule object is on.

2. Using the Monte-Carlo method without grid limitation

The second method enables a more flexible simulation, which might might make events stray from the legitimate rule of the *Game of Life*. This way allows objects to be placed anywhere within the capsule object, not only exactly along the grids. Positions of cells that are generated for following generations are chosen randomly in each generation. The decision whether or not a following generation is produced is made by calculating the density around a given generation point. The decision whether or not existing cells die is also based on the density. The process of density calculation consists of counting the number of cells within a certain distance from that cell.

Either method allows the setting 'function repetition' of a capsule object to be turned on. Under certain conditions it may be possible to produce a pattern of infinite movement.

4.5 Live Performance

The program VDAM is designed for both non-real-time composition and real-time live performance. This is achieved through the design of the user interface which meets all possible different ways of thinking of the user. Besides, it will meet the demand of simultaneous use by multiple users. It is my intention and plan to develop in the future a function to connect multiple computers in a network. VDAM will be able to connect multiple computers that have VDAM installed. The aim is to share complete identical

data by sharing information about changes of the objects in the space and so update the latest state in real-time. The capability for network messaging of Open Sound Control will also help to realize this.

Chapter 5

Conclusions

This thesis recounts the process of the development of the program VDAM. It discusses several issues, both technical and artistic. Throughout the process of the development of VDAM the importance of examining the relationship between design and affordance became clear to me. In this project, sound techniques are fused with graphic techniques into one program that utilizes the advantages of both. Consequently, VDAM is an environment that allows artists to do trial and error in an artistic way.

In the future, I plan to expand its application. For example, I am thinking of making an installation for shows and exhibitions and such like, which will allow visiting guest users, to enjoy themselves with data mapping, bending and simulating. In order to achieve this, more experiments of the possibilities for adopting physical human interfaces need to be carried out. Also, network performances may be another possibility as I already mentioned in application example 4.5. In any case, I believe that I will be able to polish the program as I will receive feed back from users about their experiments.

It is my ideal to proceed in this way, because, if there is an ultimate goal to my project, it is to reorganize our own creative/artistic communications.

Bibliography

Bongers, A. J. (2006) Interactivation : towards an e-cology of people, our technological environment, and the arts. (PhD Dissertation, Vrije Universiteit Amsterdam, 2006). *SIKS dissertation series, no. 2006-12*

Booth, C. (2008). Alan Kay and the Graphical User Interface,
from `http://www.lottiebooth.com/pdf/essay.pdf`

Han, J. Y. (2005). Low-cost multi-touch sensing through frustrated total internal reflection. *Proceedings of the 18th annual ACM symposium on User interface software and technology*, 115-118.

Harvey, B. (1997). Computer Science Logo Style: Symbolic computing (2nd ed.). MIT Press.

Ishii, H. & Ullmer, B. (1997) Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms. *Proceedings of Conference on Human Factors in Computing Systems (CHI '97), ACM, Atlanta, March 1997*, 234-241.

Jordà, Kaltenbrunner, Geiger, & Bencina. (2005). THE REACTABLE. *Proceedings of the International Computer Music Conference 2005*

Kant, I. & Klemme H. F. (2009) Kritik der Urteilskraft Beilage: Erste Einleitung in die Kritik der Urteilskraft. Hamburg Meiner

Khella, A. (2002). Objects-Actions Interface model, from University of Maryland, De-

partment of Computer Science website: `http://www.cs.umd.edu/class/fall2002/cmsc838s/tichi/oai.html`

Klanten, R., et al. (2008) Data Flow, Berlin : Gestalten

McCartney, J. (2002). Rethinking the Computer Music Language: SuperCollider. *Computer Music Journal, 26:4, Winter 2002*, 61-68.

Murphy, R. (2007). Essential grammar in use with answers. Cambridge : Cambridge University Press

Patten, J. (2002). Audiopad: A Tag-based Interface for Musical Performance. *Proceedings of the 2002 conference on New interfaces for musical expression*, 1-6.

Puckette, M. (2002). Max at Seventeen. *Computer Music Journal, 26:4, Winter 2002*, 31-43.

Roads, C., et al. (1996). The computer music tutorial. Cambridge, Mass. : MIT Press

Ryan, J. (2003). MuViz: Visualization Notes. In J. Brouwer and A. Mulder (Eds.), Making Art of Databases. Rotterdam: NAi Publishers

Smith, D. K. & Alexander, R. C. (1999). Fumbling the Future: How Xerox Invented, then Ignored, the First Personal Computer. New York : toExcel

Stafford, T. & Webb, M. (2004). Mind Hacks: Tips & Tricks for Using Your Brain. O'Reilly Media

Appendix A

Activities

A.1 List of works

A.1.1 .f (2009)

Duration:	8 min.
Performer:	Yukari Uto (first version)
	Nozomi Natsuyama (second version)
Performed at:	- International Computer Music Conference 2008
	(August 2008, Queen's University Belfast, UK)
	- Sonology Discussion Concert
	(February 2009, Royal Conservatory in The Hague)
	- Land of the Rising Sense (June 2009, Scheltema Complex, Leiden)

.f (dot-F) is an interactive performance piece with a focus on figures and movements of fingers. The original idea was to augment the characteristics of fingers into sound and visual images. The first version was made in 2007.

In this piece, performance (here: ad hoc finger movements by the performer) is captured in real-time with an iSight (= a video camera on MacBook). The captured image is then analyzed and the data this generates in turn affects sound and generates images. All sounds are created on the basis of a voice sample of the performer. The system of the piece was developed with SuperCollider 3 for the sound processing, and with JOGL (Java

Figure A.1: .f 2.1, performance, Leiden, 2009

Binding for OpenGL) and QuickTime technology for the image processing.

A.1.2 Nagoya Municipal Subway (2009)

Duration: 8 min.
Presented at: - Sonology Discussion Concert
(December 2009, Royal Conservatory in The Hague)
- CASS (February 2009, Royal Conservatory in The Hague)

Nagoya Municipal Subway is a video piece with 8 channel surround sound. There are five keywords that govern this piece; Simulation, Analyzation, Sonification, Visualization, Augmentation.

This piece consists of two parts. The first part is an introduction: some monochrome pictures are shown with recorded sound. The second part is the main part of this piece. In this part, the main concept of this piece is to simulate the metro system visually and

render it in an altered form in this case: from the visual medium to the auditive medium. Simulation is done in a time scale 200x faster than actual speed, so that we can cover an entire day. The reason that I chose the municipal subway of the city of Nagoya is that the scale of the city suited my purposes best.

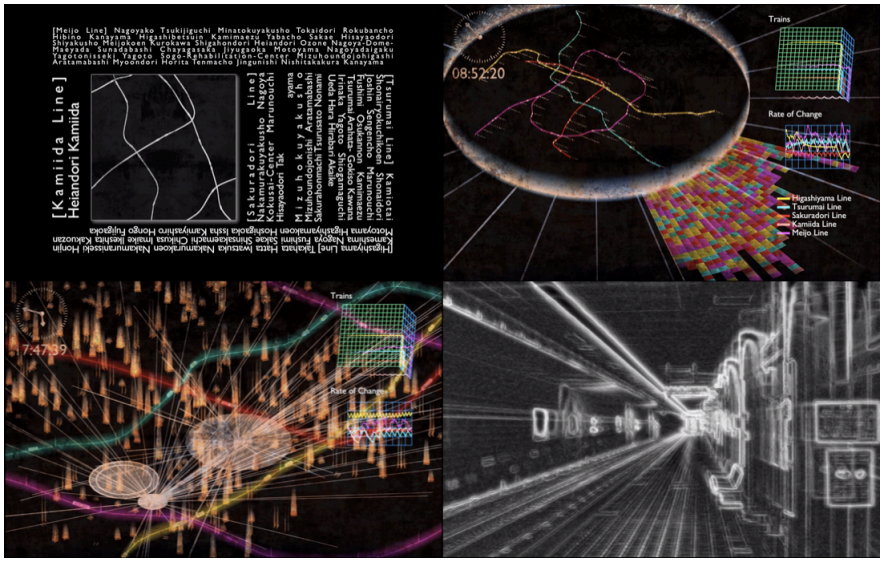

Figure A.2: Thumbnail from Nagoya Municipal Subway

The movements of trains are simulated as well as the statistics about running trains and information about which line the trains run on; all of these matters are presented at the same time. The Rate of change is also a factor that analyzes how the amount of trains on each line increases or decreases. These parameters are applied to sound generation. All generated audio is based on actual recorded sound of a metro station and the inside of a train.[1] In the simulation part, audio material is modulated by granular sound synthesis. The panning of the sound is arranged according to a train's position on the map. After approximately halfway through the second part, a generated simulation of the city appears. This is an augmentation superimposed onto the simulation.

The development of this piece started from collecting actual timetable information of each station. The information was gathered onto a spreadsheet. For the map of the

[1]This recording was made with binaural microphones in the summer of 2009.

train lines, the positions of the stations were traced manually, and the program drew a connecting line by means of a bezier curve. Generating sound in SuperCollider and rendering images were done independently from each other in non-real-time. The simulation program output images and parameters for sounds simultaneously.

In the future, I plan to upload the system of the piece onto a website on the Internet, so that everyone can try to control the time and space themselves.

A.1.3 Concert Étude (2010)

Duration: 10 min.
Presented at: - Final Examination at Institute of Sonology
(June 2010, Royal Conservatory in The Hague)

Concert Étude is my latest work and it uses compilations of most techniques in VDAM. This piece is intended for a real-time interactive live performance on stage. The piece starts with a simple manipulation, and gradually more complicated examples follow. Actually, the piece is like a demonstration of the program and includes a section with experimental improvisation. In that particular section, the program demonstrates its technical ability to incorporate a performer into the complexity of the result. This means that as the performer gets used to manipulate increasingly more functions of the program, increasingly more varied expression is possible. This raises the quality of the piece as a live performance.

Appendix B

Contents of the CD-ROM

What follows below is a list of the contents on the CD-ROM attached to this thesis. It is compiled according to the hierarchy of the file structures. The items in between brackets indicate a directory. This list can also be downloaded from the website: `http://takayukihamano.com/`

+ [Programs]
    + [VDAM]
        - **VDAM.app** - the main application developed in the project
        - [projects]
        - [examples]
        - [objects]
    + [Tutorial]
        - **Tutorial.mov**

+ [GameOfLife]
    - **GameOfLife.app**
    - [examples]
+ [Works] - see Appendix A for details
    - **dotF.mov**
    - **Nagoya Municipal Subway.mov**