Chapter 5

Creative Workflows for the Media Artist

5.1 Overview

The ways in which artists create art has changed dramatically over the past one hundred years. In the late 1800s, while the science of perception was already having a major impact on the arts through Impressionism (e.g. theories of color led to Pointillism, developed by Georges Seurat), the industrial revolution brought machines such as the railroad and the steam engine to the masses. Dadaists in the early 1920s and 30s, responded by incorporating machines into their works, for example in Marcel Duchamp's *Bicycle Wheel* (1913) and *Rotary Glass Plates* (1920). The Russian Constructivists, Vladimir Tatlin and Kasimir Malevich, among others, created works that incorporated or referenced technology with a monumental or purist aesthetic [Gray, 1962]. Up to the middle of the century, artists continued to respond to the *machine* as an object and idea. In the 1950s, following the invention of the computer, a group of scientists and artists began exploring the production of images by digital means. Led by research institutions including IBM and Bell Labs, Michael Noll, Kenneth Knowlton, and Bela Julesz,

developed early systems for expressing images by digital means [Dietrich, 1986]. This opened up a range of expressive possibilities in which algorists in the USA and Europe, including Freider Nake, Manfred Mohr and Vera Molnar, used computer code to create art [Nake, 2009], and in which scientists like Buckminster Fuller and Benoit Mandelbrot expressed beauty through the mathematics of nature [Kranz, 1974]. In 1978, a DEC VAX-11 cost $120,000, placing research systems out of the reach of most artists. However, the personal computer became available at the same time, and the first Apple II cost $1298 in 1977. In the next decade costs rapidly came down while their power dramatically increased. Digital media opened up new areas not previously possible, such as interactive art, artificial life, and computer generated imagery [Paul, 2003]. This thesis considers major choices in workflow currently available to media artists and proposes an integrated tool to explore some of these new dimensions of media art simultaneously.

The goal of this work is to develop new creative workflows for media artists. In 2005, the National Science Foundation sponsored a workshop to consider Creative Support Tools across several different domains. Discussions covered creativity in tools for children learning to program, engineers creating novel designs, users navigating the web, and new media artists exploring visual and interactive art [Shneiderman et al., 2005]. James & Jennings, in an overview of ACM Multimedia Interactive Art Program on Digital Boundaries, found that artists explored "risk-taking and subversive attitudes" to express cultural acts, often resolved by artists developing their own tools, where each of these tools embodies a particular technique for creating art [Jaimes and Jennings, 2004].

The term *workflow* derives from a sequence of steps in manufacturing, a means of production. While this description is appropriate in some ways, as the media artist creates aesthetic objects using technology, the definition of workflow for the purpose of this thesis is broadened to cover all the ways in which media artists explore or express an idea through technology. A *creative workflow* is a way of generating, exploring, or resolving a particular idea through technology. While a programming language might be used to express an abstract sequence of steps, a *workflow* is defined more broadly to resolve all the possible constraints available to the artist through creative choices.

One of the most common choices for the media artist is the programming language being used. According to the definition above, however, the choice of language is just one of the many dimensions along which a creative workflow may be resolved. Some media artists choose to work with a particular language while others develop their own [Reas and Fry, 2006], yet there are many other dimensions along which a tool can be expressive. A particular tool may support three-dimensional forms better than another or may be more suitable to interactive art. A primary contribution of this work is a consideration of specific dimensions of creative choice which are relevant to media artists. The dimensions considered here include:

1. Programming and Language
2. Modality and Media
3. Live Performance and Computation
4. Motion, Complexity and Autonomy

5. Structure and Surface
6. Image and Idea

A particular *workflow* resolves each of these dimensions through a set of choices. As discussed in Chapter 2, some of these choices may be inherent constraints determined by the tool, while others may be creative constraints resolved by the artist. Workflow is defined as the combination of these constraints and choices across the dimensions above.

The contribution of this work is to conceptualize tools for media art that function along these dimensions simultaneously. The first generation of media artists had no choice but to develop their own tools, since the first computers did not know how to make images [Dietrich, 1986]. This drive to make one's own tools is now considered an essential part of how media artists work but should not be an exclusive requirement for making digitally based art. Ideally, tools for the media artist should allow programming when desired while also supporting the other available dimensions for expression. One motivation for this is that it eliminates the repetitive work required in engineering one's own tools, allowing the artist to work at a higher level. Another motivation is that tools which consider the variety of workflows used by media artists cover a potentially broader expressive range than any one particular tool or language. Thus a central question is how best to support the creative work of media artists.

LUNA, a Language of Natural Animation, is developed to show that these creative dimensions can be resolved together in an integrated tool; its contribution is to support a wide range of different techniques employed by media artists. However, LUNA is not

necessarily an ideal tool for media artists since every tool introduces its own inherent constraints [Candy, 2007]. For example, LUNA does not go into great *depth* either in particular structures (e.g. its support for highly detailed characters is limited) or in particular modalities (e.g. its support for audio is limited). I assume these details may be added in the future with relative ease. Other potential limitations of LUNA will be considered later. Instead, this work focuses on providing a context in which the dimensions above are simultaneously resolved and made available.

5.1.1 Motivation

Motivations for this work come from several directions. My own background in art and computer science has involved an exploration of a variety of forms.[1] In my early work, such as *Atoms* (1994), I investigated generative art through programming based on simple rules which imitated early algorithmic systems similar to Reeve's Fuzzy Objects and Conway's Game of Life. A parallel interest in rendering led to computer software for shading with *Raycast* (1994), and to works in oil painting with a superrealist aesthetic such as *Camera* (1993). Later on, my interests shifted to sculpture and physical forms with autonomous kinetic works including *Creatures* (2001). As complex feedback is possible with very simple programs the immediacy of behavioral systems was always compelling to me, but I could not find tools to achieve that same interactivity with sculptural or procedural forms. Although I experimented with tools for digital modeling, including Maya and 3D Studio MAX, I found few tools capable of combining three-

[1]The works referred to here by the author can all be found at http://www.rchoetzlein.com

dimensional forms with the same level of interactivity of behavioral systems (found in Max/MSP and Processing for example). In addition, in projects such as *Pears* (1998), I began to explore images, video, and other media which I viewed as potential sources for generating unique sculptural forms. The development of LUNA thus captures several dimensions and ideas which I found difficult to combine or express with existing systems, but which I did not perceive as separate in my work.

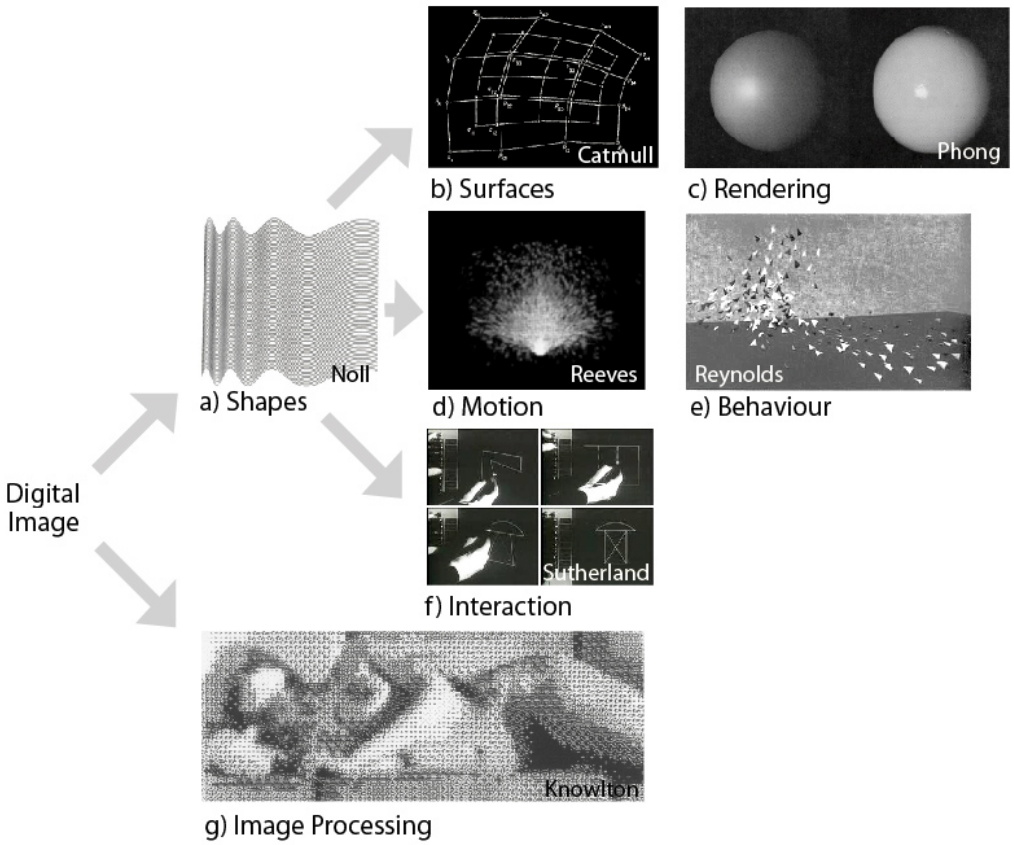


Figure 5.1: Early history of the digital image, including a) shapes (Michael Noll), b) surfaces (Ed Catmull), c) rendering (Bui Phong), d) motion (William Reeves), e) behavior (Craig Reynolds), f) interaction (Ivan Sutherland), and g) image processing (Ken Knowlton).

The history of media art, modern art and computer graphics helps to resolve these problematic aspects. Early artist-scientist pioneers, such as Michael Noll, Ken Knowlton, George Ness and Freider Nake developed the first computer images using simple shapes [Dietrich, 1986]. Shapes such as lines and curve, Figure 5.1a, are the starting point for several new directions. From this point, the vocabulary of shapes can be extended in three-dimensions to curves, surfaces and volumes, as was done by Bresenham, Pierre Bézier, Ed Catmull (Figure 5.1b) [Foley et al., 1997]. Once surfaces were established, other groups focused on the rendering and lighting of these surfaces, including Bui Phong, James Blinn, Don Greenberg, and Turner Whitted (Figure 5.1c). However, one can also take these basic shapes and study their *motion*, resulting in dynamic objects as explored by William Reeves (Figure 5.1d). With the presence of motion, unique behaviors are explored by Craig Reynolds, James Whitney, and others (Figure 5.1e) [Levy, 1992]. *Interaction* with basic shapes leads to real-time systems as investigated by Ivan Sutherland, Doug Engelbart, and Alan Kay (Figure 5.1f). Prior to these interactive systems, the digital image was itself transformed as an object of study through applications to surveillance, satellite research and the military, Figure 5.1g. Rosenfeld surveys these image processing approaches to the image [Rosenfeld, 1983].

These fields define the basic ways in which the digital image may be manipulated while, over time, their separation has resulted in distinct communities. Modern motion pictures and video games take advantage primarily of three-dimensional modeling, lighting and rendering, relying on technical artists to manually create char-

acters and textures to support a particular narrative. Algorithmic artists such as Harold Cohen, Charles Csuri, Freider Nake, and Vera Molnar have explored programmatic, non-representational aspects of image making by looking at motion and behavior [Verostko, 2006]. Information artists work with the database as a source for the layout of shapes and forms. Similarly, interfaces in film and gaming change very gradually due to their means of distribution, while interactive artists such as Ken Feingold, Golan Levin, Michael Naimark, and Simon Penny tend to incorporate or even invent new interfaces regularly in installation works.

A primary motivation for this research is to develop workflows that allow these different practices to be combined in a single tool, to demonstrate that techniques among these communities may be shared. In LUNA, for example, one goal is to incorporate modeling and rendering typically found in animation software, and to combine these with processes for algorithmic, interactive, and performance art. Although the dimensions covered are not designed to address media arts completely – for example aspects of database and web art are not considered – the goal is to bring together several areas of digital art which have evolved into distinct tools by integrating the dimensions in which these choices are made.

This list of dimensions appears to share some similarities to studies of formalism in art history. According to Hatt, in the *Principles of Art History* (1950) Heinrich Wolfflin presents formal properties "meant to provide general descriptive terms, which would capture the development of artistic vision across countries and ages [Hatt and Klonk, 1992]."

[2] Roger Fry develops a familiar set of formal elements in observing line, mass, space, light, color, and plane [Fry, 1926]. These formalist elements examine a work of art based on style in order to understand art created using similar techniques across civilizations and periods of time. They are applied not only to distinguish works of art but to offer explanations for why and how they were created by cultures in time [Preziosi, 1998]. While I also seek to understand the historic aspects of media arts, the dimensions proposed here are based essentially on technique itself rather than style; my primary contribution is not a reflective analysis of works in media art but a workflow for integrating its techniques. Although many of the design choices in LUNA are inspired by historic works of art, these dimensions relate to choices in technique made by the artists during the creation of their work rather than styles derived from looking at art after the fact.

When a sufficient number of examples are considered over time, it may be possible to analyze style in a specific area, such as organic art for example. Steven Levy, in his book *Artificial Life*, explores some of the historic aspects of this type of art and studies the culture in which they developed [Levy, 1992]. Christian Paul, in *Digital Art*, surveys several of the active areas of media art [Paul, 2003]. More recent surveys can be found in *Art and Electronic Media* [Shanken, 2009] and *Art of the Digital Age* [Wands, 2007]. Although it is difficult to distinguish style from technique since both are constantly changing, a study of style in media art could be an interesting area for future research.

[2] Wölfflin's five dualities include 1) Linear versus Painterly, 2) Plane versus Recession, 3) Closed versus Open, 4) Multiplicity versus Unity and 5) Absolute versus Relative Clarity

Each of the current areas of media arts offers a different perspective on the construction of the digital image. Language, computation, modality, autonomy (behavior), structure, image and idea are the primary dimensions which are considered here from the perspective of technique. The cultural significance of the digital image is not addressed here, nor the social effect of these abstract and scientific ideas. In addition, the impact of these techniques on society – which is the study of media theory – is generally not addressed, although certain aspects of structuralism are relevant in examining technique. Finally, it is useful to mention that this is not a goal-oriented process since the creation of a tool for media artists can never have an ideal or final form. Rather, the goal of this work is to propose an integration of several of the dimensions of digital image-making which have diverged into separate disciplines over time, and to show that it is possible to invent new tools which combine these. The motivation for this work is to bring together communities engaged in creative image-making by reducing the natural boundaries that form between tools evolving in particular domains.

5.1.2 Evaluation

The evaluation of tools to support creative tasks can be difficult. The most immediate form of evaluation of any tool is that it achieves the functionality it claims to. With regard to the LUNA, the six dimensions - 1) programming, 2) modality/media, 3) live performance, 4) dynamics/autonomy, 5) structure/surface, and 6) image/idea - should be realized through specific examples found in the system. Since each of these

is treated as a dimension in which choices are made, I consider each as a collection of at least two examples showing a range of behavior. However, my argument is not only that tools can be developed to generate specific examples, but that their integration will provide better overall support for creative workflows among media artists. "Better support" is understood as a general increase in expressiveness, creativity, and ability to explore interesting areas. How would this be evaluated?

Among the outcomes of the Creative Support Tools workshop was the discovery that creative tools cannot be easily evaluated for the quality of the outcome. It is basically impossible to say which tools support creativity to a greater or lesser extent:

> "One important issue with the design of creativity support tools is how they can be evaluated. How do you know if a tool is being helpful? Human-computer interaction professionals are used to measuring the effectiveness and efficiency of tools [for specific goals], but how do you measure if it supports creativity? As discussed above, tools that are not effective and efficient will probably hinder creativity, but it isnt clear that the reverse will hold. To try to measure creativity, the Silk system designers evaluated many different properties, including the number of different designs produced, the variability of the components used, the variety of questions about the designs from collaborators, etc. [Landay 1996], but these still do not really get at the *quality* of the solution. It is still an open question how to measure the extent to which a tool fosters creative thinking [Resnick et al., 2005]."

Despite these difficulties, the above authors found three criteria which were agreed upon to be a good relative measure of creative tools. These are: 1) low threshold, the idea "that the interface should not be intimidating, and should give users immediate confidence that they can succeed," 2) high ceiling, that "the tools are powerful and can create sophisticated, complete solutions," and 3) wide walls, that "creativity support tools should support and suggest a wide range of explorations [Resnick et al., 2005]."

How do these concepts map to creative tools for media artists? While we can agree the visual interface should be simple to use, low threshold may also be taken to mean that the underlying language expressed by the tool is also simple since media artists may wish to work either in a programming language or the visual interface along the dimension of *language*. The idea of 'high ceiling' can have several interpretations. Does this mean powerful in its ability to support different modalities (images, sound, etc.), powerful in the structural detail it can achieve, or powerful in the processes it can perform? I will consider each of these as they arise through examples. Finally, there are several levels on which artists can explore a range of expressions. This may be through the parameters to a particular system, parts of a particular object, or the combination of these systems. Despite distinctions which must be resolved during their evaluation, these criteria provide useful guidelines for evaluating creative tools relative to one another.

LUNA is proposed as an integrated tool for media artists, which suggests that its greatest improvements will occur over time with the active participation of a community of users and developers.[3] In the current design, a number of features are presented as potential directions. Along the dimension of modality, for example, LUNA has icons for audio, video and data. In these areas integration has been considered and designed into the system so that practical details may be completed more easily in the future. During the evaluation of LUNA I thus distinguish several areas of development: 1) inherent

[3]To enable this, the core of LUNA will be released under the open source LGPL license for both Windows and Linux, with the interactive editor released as a free executable for personal use (with dynamically loaded user modules).

limitations of the language, 2) potential ideas which were not explored in any way, 3) partial features which were considered and designed into the basic language of LUNA for future growth but not fully implemented, and 4) features which were completed.

The methodology used in this study is to examine each dimension, to explore its significance for media artists with historic examples, to see how well LUNA integrates that dimension into the overall workflow of the tool, to evaluate the criteria of low threshold, high ceiling, and wide walls, and to evaluate the system according to the actual and potential choices it makes available to the artist.

5.2 Programming and Language

5.2.1 Procedural Languages

The first two exhibits of computer generated images showed works by Bela Julesz and Michael Noll at the Howard Wise Gallery in New York in 1965 (Computer Generated Pictures), and by George Nees and Frieder Nake at the Galerie Niedlich in Stuttgart, Germany the same year [Dietrich, 1986]. All of the early pioneers were also scientists working for institutions such as IBM and Bell Labs in order to provide access to the large, costly computers need to make these images. For the first generation of computer artists, direct programming of the computer was a necessity. Michell Noll attempted to simulate constructivist and abstract works of art using very basic mathematical shapes, in images such as *Bridget Riley's Painting Currents*, 1996 where he reproduces the op art of Bridget Riley's *Currents*, 1966. Using the density of type to reproduced tone, Ken

Knowlton and Leon Harmon created *Studies in Perception I*, 1996, the first digitized and reproduced nude figure. Initial collaborations between artists and engineers took place in Cybernetic Serendipity, an exhibit at the Institute of Contemporary Art in London, curated by Jasia Reichard [MacGregor, 2002]. Computing at this time had a very high threshold, and artist-scientists were required to program in low-level languages using basic mathematics. Despite these difficulties, the presence of science in art was viewed as a cultural shift, through Buckminster Fuller, Benoit Mandelbrot, and others, that explored technology, math and biology as a way of rediscovering nature [Kranz, 1974].

To facilitate making graphical images, scientists began by developing extensions to generic programming languages. George Ness and Leslie Mesei added graphics commands to ALGOL 60 and to Fortran in 1969. Eventually, work by Ken Knowlton and others led to more complete graphic languages. Although they simplified the work for scientists, they were still found to have a high threshold for artists.

> "As an animation language it provided instructions for several motion effects as well as for camera control. Knowlton had initially hoped that artists would learn the language to program their own movies, but he came to realize that they usually wanted to create something the language could not facilitate, and they also shied away from programming... None of the graphics languages mentioned received widespread use, partly because their implementation was machine dependent and also because each language was restricted in scope [Dietrich, 1986]."

Interactive sketching introduced by Ivan Sutherland in 1963, which would allow for direct drawing of shapes using a pen, would not be widely available for another decade [Sutherland, 1963]. For the algorists in the 1970s, these constraints were not a major barrier as the artist's interests contained a strongly constructivist element;

the use of algorithm and mathematics coincided with their creative goals. For artists such as Harold Cohen, Roman Verotsko, and Manfred Moher, programming became the abstract language "through which they created a new reality", a world in which mathematics was understood as a new way to appreciate nature [Verostko, 2002].

In a relatively short time, computing reduced in cost and better graphics hardware became available. Through the 1980s graphical languages proliferated. GINO, Graphical Input/Output System (1971), abstracted the concept of logical output and input devices. Languages such as GPGS (1972) and PHIGS+ (1986) supported hierarchical object arrangements, allowing for descriptive scenes [Dam, 1998]. With the development of graphics standards, OpenGL (1992) and DirectX (1994) were the first languages to be widely used on new graphics hardware, built first as extensions to the generic C/C++ language. Graphical languages finally became widespread in the mid-1990s.

Programming directly in text-based languages continues to be an important way to create visual images as these languages can allow the artist greater control over the specific rules used for indicating new behaviors. To reduce the high threshold of learning full programming languages, Casey Reas and Benjamin Fry introduced the Processing language to promote image making as a way to teach programming in 2001 [Reas and Fry, 2006]. A sample program in Processing is shown in Figure 5.2. In LUNA, text-based programming in C/C++ allows authors to create new interactive nodes, discussed in further detail with the introduction of visual languages below.

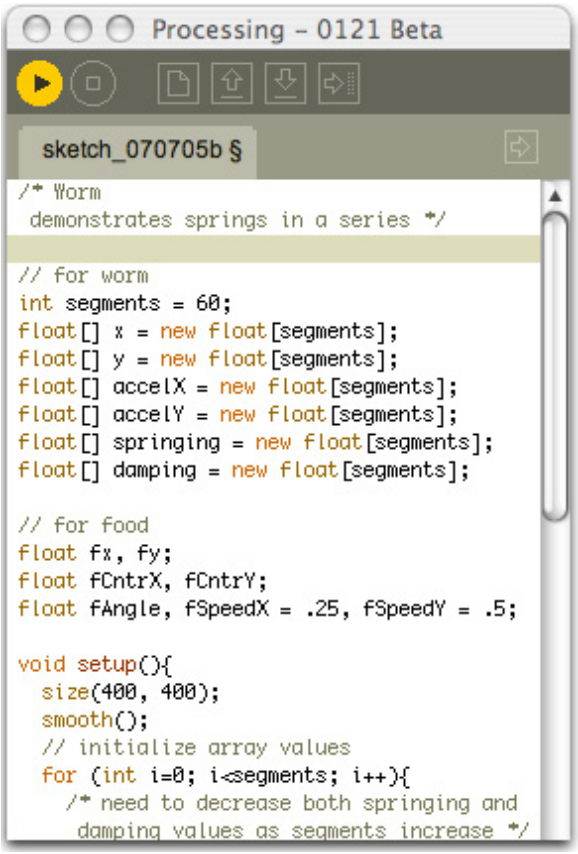


Figure 5.2: Text-based authoring environment in Processing. Image from Processing (c) 2004, Ben Fry and Casey Reas. http://processing.org

Graphics libraries are now available in many other generic text-based languages such as Python, C/C++, Flash and Java (on which Processing is based). Although the use of text-based languages continues to grow as more media artists learn to program, the threshold for achieving complex forms using these languages is still high. While it may be that media artists should learn programming as a part of an education in media arts, it does not necessarily follow that all media artists wish to create exclusively by algebraic or procedural programming.

5.2.2 Visual Languages

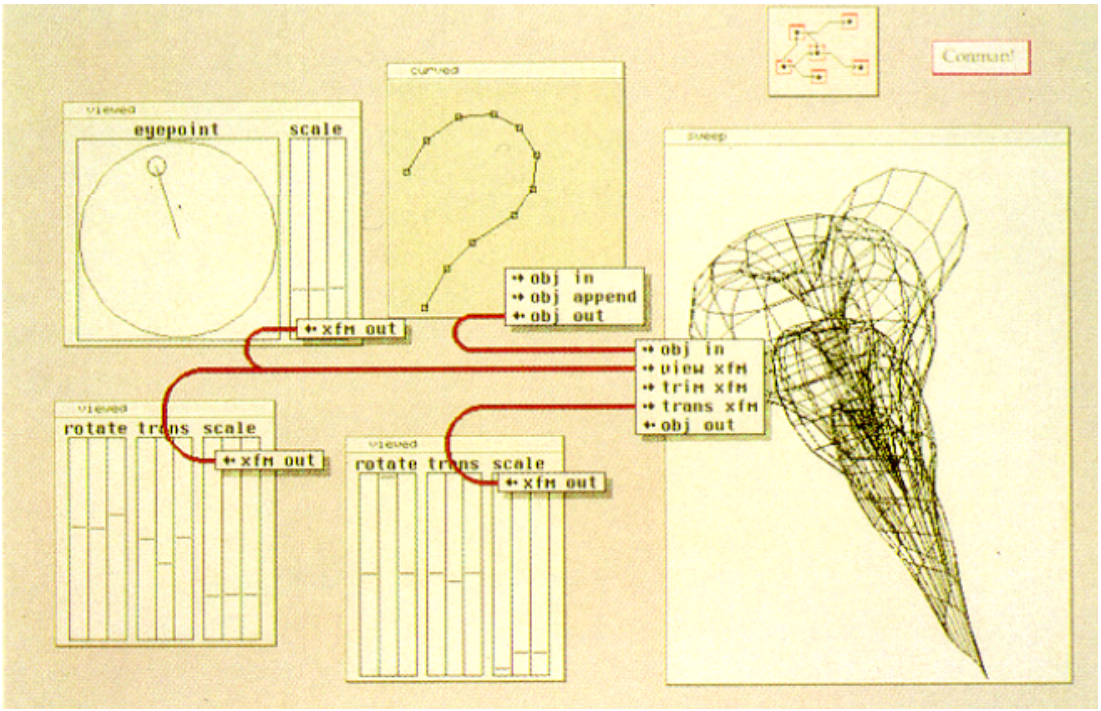


Figure 5.3: Con Man, an early visual language for procedural graphics. Image copyright Paul Haeberli (c) 1988.

The development of Ivan Sutherland's Sketchpad in 1963 led to the first generation of direct interaction technologies. Drawing programs allowed designers and animators to manipulate images manually. In 1968, Ken Pulfer and Grant Bechtold of the National Research Council of Canada created the first hand drawn computer movie entitled *Hunger* by drawing each frame using a wooden mouse. "Markup" and "Superpaint" were the first drawing programs by William Newman and Dick Shoup from research based at Xerox PARC (1974-1975). Myers surveys a number of other parallel developments in early computer interaction [Myers, 1998]. While drawing and sketching are

the first obvious uses of direct interaction with machines, visual interfaces for computational tasks began with visual programming languages (VPL). Early visual languages had similar constructs to text-based languages, and contained iconic representations of mathematical operators, loops and conditionals. A key concept which distinguishes VPLs from text-based programming is the introduction of *nodes* which allow logical operations to be encapsulated and modularized, so that data is viewed as *flowing* through a visual representation of a set of tasks. An early system which experimented with visual languages to perform image processing is ConMan [Haeberli, 1988]. A survey of generic visual languages for goal-oriented tasks, which includes ARK, VIPR, Prograph and IBM Data Explorer, can be found in Boshernitsan and Bownes [Marat and Downes, 2004].

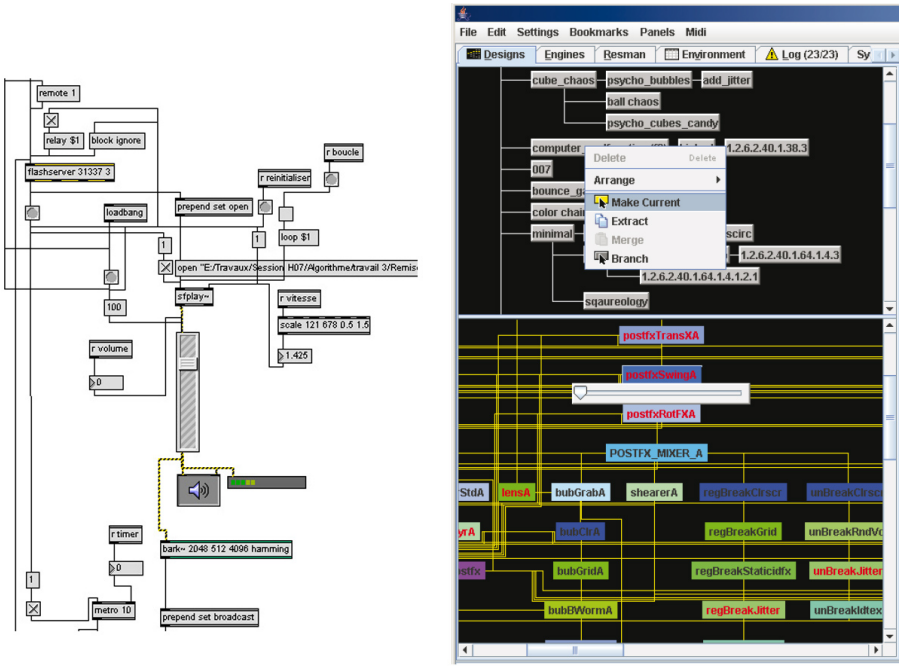
The idea of the media artist as programmer is an on-going trend, so the interest of artists in visual dataflow languages is relatively new. Interactive artworks, such as *A-Volve* by Sommerer & Mignonneau (1994), use text-based programming to achieve a particular, dynamic relationship between the art work and the viewer. This usually requires artist/engineering collaborations since the programs are specific to the installation. A generalized approach to media arts would be to treat each *type* of interface device as a building block with which an exhibit is constructed. Edmonds et al. explore the idea of what tools might be best for media artists:

> "A fundamental question that we have been considering is, what kind of environments best support the development of digital art? There is one answer to this question which, although it may sound a little strange, is, nevertheless, appropriate. In art and technology environments, we need environments for building environments [Edmonds et al., 2004]."

The idea of an environment-for-environments was traditional held by programming languages as these provided a context in which interactive media "environments" could be developed. With the advent of the visual dataflow language, which is itself an environment (or interface), this implies the ability to modularly *design* rather than textually program another environment, application or media installation.

An interactive system mentioned by Edmonds is Max/MSP, created by Cycling '74 and originally written by Miller Puckette (author of PureData), Figure 5.4. Designed for audio synthesis, Max/MSP is a visual language that can generate interactive art using an extension called Jitter. One key contribution of Max/MSP/Jitter is that it allows for many different devices to control audio-visual events by treating all data in the system as a *signal*. David Wessel and Matthew Wright add support for gestural interfaces to Max/MSP by creating Open Sound Control, a communications protocol that allows signals and events to move from device to synthesis, even between remote computers [Wessel and Wright, 2002]. This enables the media artist to work with many different *input devices*, discussed in more detail in the next section.

For artists interested in three-dimensional visual forms, the generalization of data as signal may present some problems. First, three-dimensional forms are represented by computers in a particular way, by using vertices, edges and faces (one possible way), so they are not easily encoded as *signals*. For example, although it is possible to represent a three-dimensional tree structure in Max/MSP, the designer must author a special object to 'encode' each type of geometry. In addition, this encoding requires that the artist-user



a) Max/MSP, Cycling '74 b) Soundium, Pascal Müller et al.

Figure 5.4: Two visual languages for media artists, Max/MSP and Soundium. Max/MSP copyright Cycling '74 (c) 2010, and Soundium by Pascal Müller, Stefan Müller Arisona, et al. (c) 2008.

must continually remember the type of the data as it flows through the system. Finally, the signal processing metaphor results in a visual language which is low-level, consisting of operators and expressions as found in generic programming languages. Although this makes it expressive, it also requires mathematical thinking to understand a Max/MSP patch.

Several top-level decisions drove the design of LUNA, with a special emphasis placed on designing an intuitive interface for artists. This is embodied in workflow principles developed during the project. These are: 1) creative expressiveness in the interface should not require mathematical expressions or logic, 2) the interface should express

high-level concepts first, and particular details only on demand, 3) the language should be capable of complex structures *and* multimedia.

These constraints, especially the first, led to a visual language inspired by the board game Scrabble, which achieves a huge combinatorial variety in word letterings purely through the relative placement of tiles. This suggested a node-based workflow incorporating large iconic tiles with a minimum of extraneous information (i.e. no numbers or values on the graph). The goal of creating a high-level interface helps the design by supporting the idea that each node should represent a particular aesthetic task. A key design of the language which led to its implementation was the discovery that nodes can represent both their *structure* and *behavior* in a way that leads to a natural extension of the language. The structure of the data is carried from node-to-node in the graph, while the behavior determines how each input is processed. In fact, these two ideas are the only textual information presented in the graphical interface. The icon itself expresses a pictographic idea of the *behavior* that will occur, also written in large type above the tile, while the structure of the output is shown in smaller type, depicted in the overall color of the tile.

The graph design of LUNA is not the only way to express structure. In computer graphics the relationships between objects, their physical proximity and orientation in two or three-dimensions, is commonly represented using a *scene graph*. The scene graph became widely used after IRIS Inventor in 1993 to describe scenes consisting of different objects [Strauss, 1993]. Separately from the audio-signal processing community,

the graphics community found that objects in scene graphs can also be expressed in visual languages to facilitate on-screen interaction. Objects in these visual languages, such as Maya's Hypergraph, represent geometric relationships and transformations in space [Bar-Zeev, 2007]. Procedural systems such as Houdini express both behavior and structure, but the language does not make it clear what these structures are, and may require a series of intermediate steps to change the geometry type (see Chapter 4).

A recent approach to this duality is found in Soundium/Decklight, developed by Corebounce.[4]

> "There is a problem we have not dealt with: As a result from unifying multiple graph-based processing entities in a single graph, we have to deal with different graph processing semantics: For example an audio graph, which is typically flow-based, is processed in a different manner than a scene graph (which is basically an object hierarchy). The question is, how we deal with the coexistence of different semantics in the same graph? Our approach is to segment the global graph into individual subgraphs, which correspond to different semantics. Of course, this segmentation will not be made visible to the application layer [Arisona, 2007]."

The authors introduce a *design tree* to express high-level artistic ideas which are used to generate these processed sub-graphs [Müller et al., 2008], see Figure 5.4. This workflow enables simultaneous audio and live visuals (video), interactive editing during a performance, with examples that focus primarily on transformations such as rotation, translation, and scaling that are common to scene graph languages. This approach to multimedia, while it resolves many challenges in simultaneous audio-visual processing,

[4]Pascal Müller, Stefan Müller Arisona, Simon Schubiger-Banz and Matthias Specht, from ETH Zurich, University of Fribourg, and University of Zurich

may make it difficult to address behavioral changes in more complex three-dimensional forms.

The scene graph problem is resolved in LUNA at a high level by allowing each node to contain its own scene graph, while the language conveyed through a combination of nodes expresses dataflow. Unlike Max/MSP, the structure of each object is apparent in the tile color as information moves through the graph. Thus each single tile in LUNA represents not just one object, but arbitrarily many, which allows the system to express complex jointed or articulated structures (a scene graph) while also permitting multimedia processes which operate on these. Due to the storage of these structures (see Chapter 4), modifications to color, position, and orientation can be made at any point in the graph.

Users of LUNA may interact with programming at two levels. The first is by authoring new nodes in the text-based language C/C++ to create new fundamental behaviors. Although Processing was designed to have a lower threshold for text-based languages, node authorship in LUNA is simpler than generic C/C++ on which LUNA is based since it provides all the data structures needed to create complex geometries, Figure 5.5. The second means of creative interaction is through the visual language by mixing and combining existing tiles. Using the visual interface may have a lower threshold than any other language currently available to media artists since the media in use, its flow in the graph, and relation to other processes are all immediate apparent to the user.[5]

[5]At present, node authorship requires rebuilding of LUNA, although this is expected to change soon as new versions will allow dynamic linking.

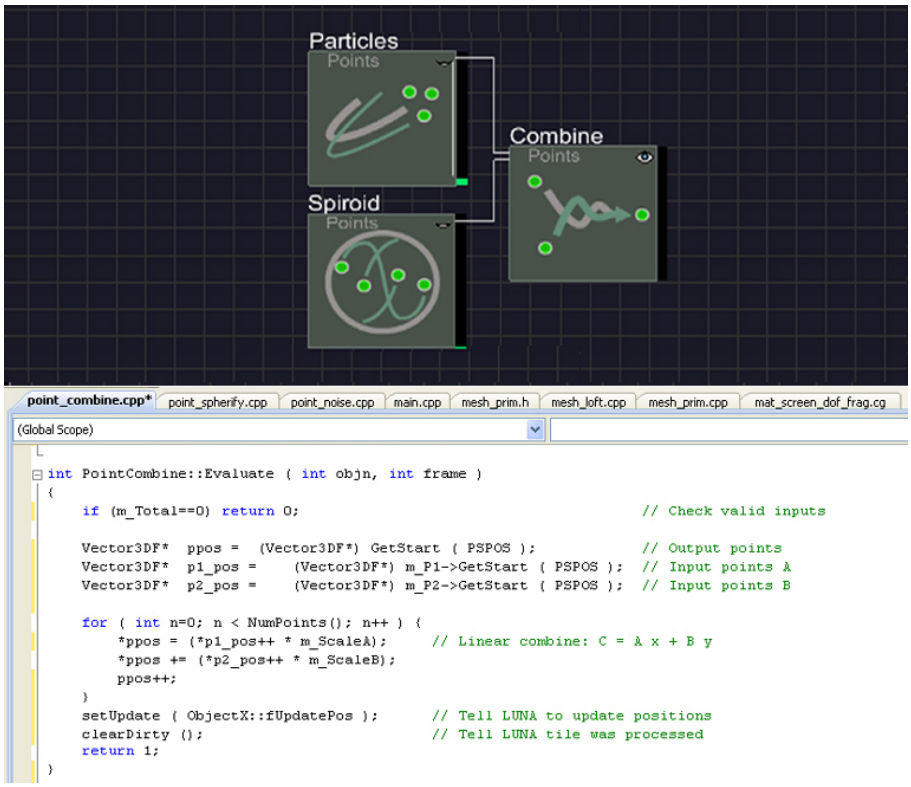


Figure 5.5: Text-based authorship of a point Combine node, and example of its use in the LUNA visual interface. Images by the author.

The system was even shown to a ten year old, who created complex systems simply by matching up input and output colors of the tiles. From the perspective of creative workflow, this low threshold is achieved through the design constraint of supporting non-technical users.

5.3 Modality and Media

Media artists desire to work in a wide variety of ways using different kinds of media and different interface devices. This dimension of creativity has two particular aspects.

Media may be defined as the structure of data, while *modality* is defined in human-computer interaction as a physical system or device that generates a particular media [Bolt, 1980]. For example, a video camera is the modality which produces the media of video. While this may seem obvious, devices like video cameras produce both audio and video, while a music keyboard can generate media which is either audio (a signal) or midi (a sequence of notes). In addition, software may process many types of media, or transcode one media into another [Manovich, 2001].

Max/MSP resolves the issue of media by treating every media type as a signal. This simplifies the base design of the software, and focuses attention on the audio signal, but places a burden on patch developers to handle media other than audio. This is partially alleviated by a large development community, but support for fundamental media types such as meshes or materials is difficult as the community must agree to a library standard.

The design principle for media in LUNA is that each node, individually a) knows what it is, b) knows what it requires, and c) knows what it produces. For example, to make a forest one must know where to plant the trees (point locations), and what the trees should look like (joint structure). To interactively use a computer mouse to change the brightness of an image, one must know the position of the mouse (a point), and the input image. Figure 5.6 shows a hierarchical map of different media in LUNA. Notice that trees, characters, meshes, and curves all have points as a base class, which implies that any process that operates on points - a bend or twist for example - can

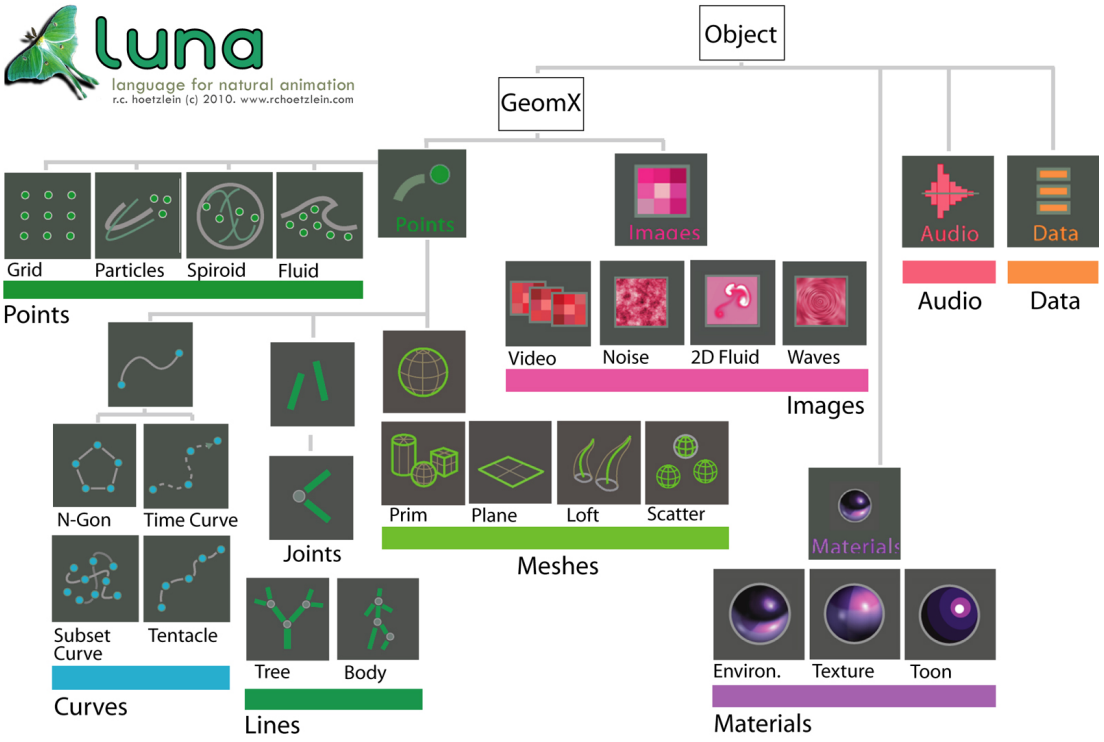


Figure 5.6: Taxonomy of classes and currently implemented nodes in LUNA.

operate on all of these objects. There is an interesting similarity between this taxonomy and the early history of graphics in Figure 5.1, which further supports this architecture as a way of manipulating the digital image in different ways. Although many higher level media types such as audio and events are not yet implemented, as indicated in the figure, this vocabulary of data structures provides a great deal of flexibility.

The primary toolbar in LUNA selects among different media types (structures), while the secondary toolbar provides a choice of processes for generating that particular type (behavior), as shown in Figure 5.7. Attention was placed on geometric media, such as points, lines, curves and meshes, to support the author's interest in sculptural form in real-time systems. Choices made in implementing particular processes are discussed

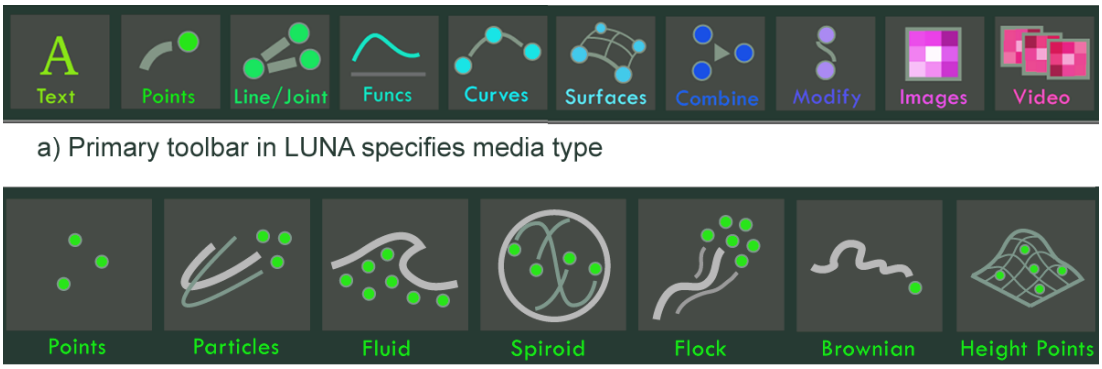


Figure 5.7

in the next chapter. One key goal of the media structures at this level is the ability to distinguish between disconnected points, articulated shapes, and surfaces; features common to graphics systems for modeling three-dimensional forms, such as Houdini, but not currently found in frameworks for real-time multimedia such as Max/MSP or Soundium.

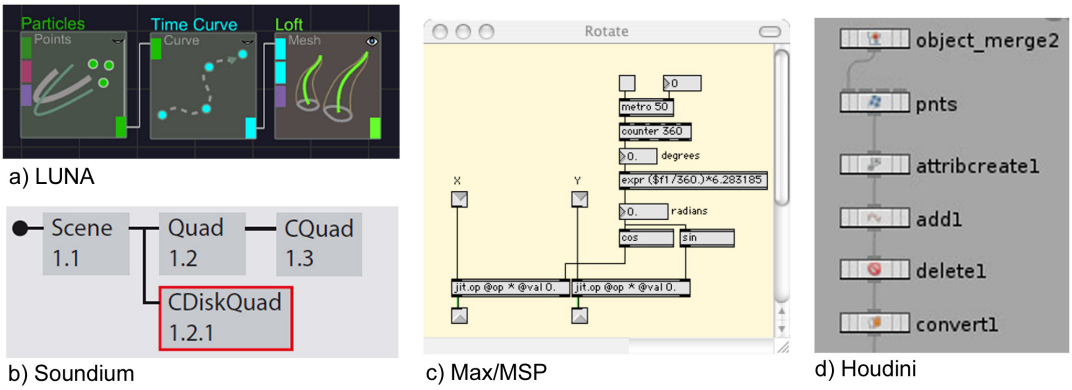


Figure 5.8: Detailed view of connections in various visual design platforms. Colored tabs in LUNA indicate changes in the type of media as it flow through the system.

A unique aspect of LUNA is that inputs are handled by each object. Unlike other media frameworks the type is not fixed by the base system but enforced by individual

nodes, thus LUNA may be described as a loosely-typed visual dataflow language.[6] Figure 5.8. shows how inputs are configured in Houdini, Max/MSP, Soundium, and LUNA. What is the type of media flowing through each example in figure 5.8? Color indicates type in LUNA and this is reflected in the design color of each node, making it easy to identify what media is needed for a particular tile to function. This design lowers the threshold for use of the system by visually re-enforcing the grammar of the language.

The data types supported in any computer language are a basic part of its specification. Within this dimension, LUNA allows the artist to work with lines, points, curves, surfaces, images and materials (shaders) better than other real-time media systems, already providing a high ceiling in terms of potentially realized structures. Other types such as audio, database input, and volumetric data, are considered by the architecture and may be implemented at the level of text-based authorship by a community of developers in the future. For users of the visual interface, low threshold and wide walls are fostered by the tile design and colored tabs which indicate media type as one works.

5.4 Live Performance and Computation

The classical model for the exhibition of the image is the museum, a place where final images are presented as objects for appreciation. More recently, film and gaming have partially displaced the still image with dynamic and interactive forms as a way of communicating with the viewer, while media artists in the tradition of disc and video

[6]The term loosely typed comes from generic text-based languages that do not require the type of a variable to be explicitly stated.

jockeys explore live performance as a venue for new experiences. For these artists, mixing and recombining segments of video and music occur *during* a performance, which requires techniques that can be applied instantaneously. To have this same kind of interactivity with digital three-dimensional forms necessitates high performance computing. This is made possible by new developments in computer graphics such as the Graphics Processing Unit (GPU), a computer chip dedicated to the function of rapidly generating digital images, and now also used for generic parallel computing. In the traditional model of computing, performance was based on CPU clock rate, which meant that the extent of calculations that could be performed was directly related to computing power. This limited real-time interaction to all but the simplest tasks. With the advent of the GPU, calculations can be performed in parallel so that computation is no longer limited by clock rate.

One of the implications of GPUs, which are now being used for many other tasks besides graphics calculations [Bhushan, 2008], is that computing resources can be targeted toward several goals within the same system simultaneously rather than treating each step as a sequence [Farber, 2008]. For media artists this implies that live performance is no longer dependent on how much computing power one has, but rather it is a dimension of choice in which the artist may *focus* various computing resources. This concept of managing computing power is common in the gaming community, where there is a finite budget for graphics, audio, game play, and interaction which must all be realized in 1/30th of a second.

Historically, the use of computing power is one of the fundamental distinctions between different types of graphics tools. Systems such as Maya, 3D Studio MAX, and Houdini all interface with offline rendering software capable of producing very detailed, accurately illuminated images using as much time as needed, while live performance tools such as Max/MSP, VVVV, and Soundium use computing resources to perform simpler tasks in a very short time (real-time). The fact that GPU computing allows large numbers of computations to be performed rapidly implies that future tools for artists will reduce the distance between live performance projects and offline computer generated imagery (although this distinction may never be eliminated entirely).

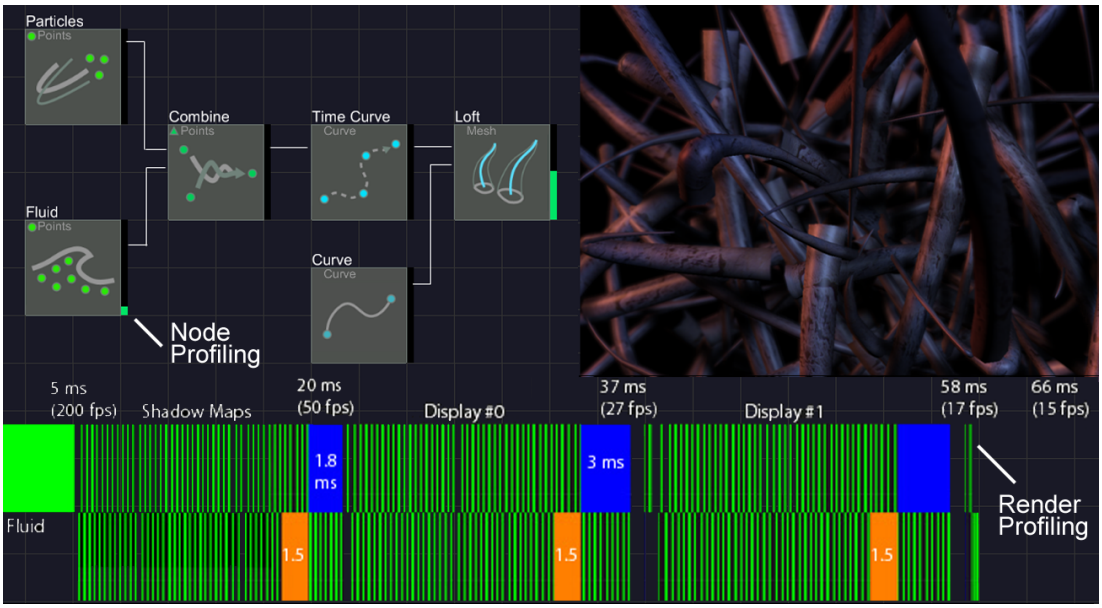


Figure 5.9: Performance profiling in LUNA. Node profiling shows CPU/GPU resources used for each object in the user graph. Render profiling shows the resources used by the CPU for computation and GPU for rendering.

LUNA introduces several novel features to encourage this convergence. First, rendering in LUNA makes heavy use of the GPU to perform deferred shading, a real-time rendering technique capable of achieving realistic results that were previously only possible with offline methods [Deering et al., 1988]. Secondly, the visual language includes dynamic *profiling*, a technique for measuring computational load. Node profiling shows, using vertical bars, how much computing power is being allocated to each node in the LUNA graph, Figure 5.9. Render profiling shows how time slices are allocated to the CPU and GPU. The vertical green bars in Figure 5.9 represent computation of each tube shape in the image, orange is a transfer of data from CPU to GPU (and thus lost time), and blue represents rendering on the GPU (making of the image). These blocks are repeated three times to compute shadow and output for a two-screen image.

To my knowledge, LUNA is the first system for media artists that gives immediate feedback on how computing resources are used. These profiling results already suggest several improvements to the system. For example, the repetition of vertical bars indicates that too much time is being spent on geometric calculation of the Loft node, making this an ideal candidate for authoring this process on entirely on GPU. Some nodes already support this, such as the Fluid system, which can run on either the CPU or the GPU [Hoetzlein, 2010].

More importantly, the artist has direct control over how computing power is allocated to different details in the image. Each node supports the concept of a *maximum count*, a set limit on how many objects it will process regardless of how much data it

receives. In this example, even though the Fluid system is simulating 4000 particles to the drive the motion of the tubes, the Loft node is generating only 200 tube surfaces, restricting the flow of data to a manageable level while keeping the output interesting. At present, the artist controls the maximum count of each node through the interface.

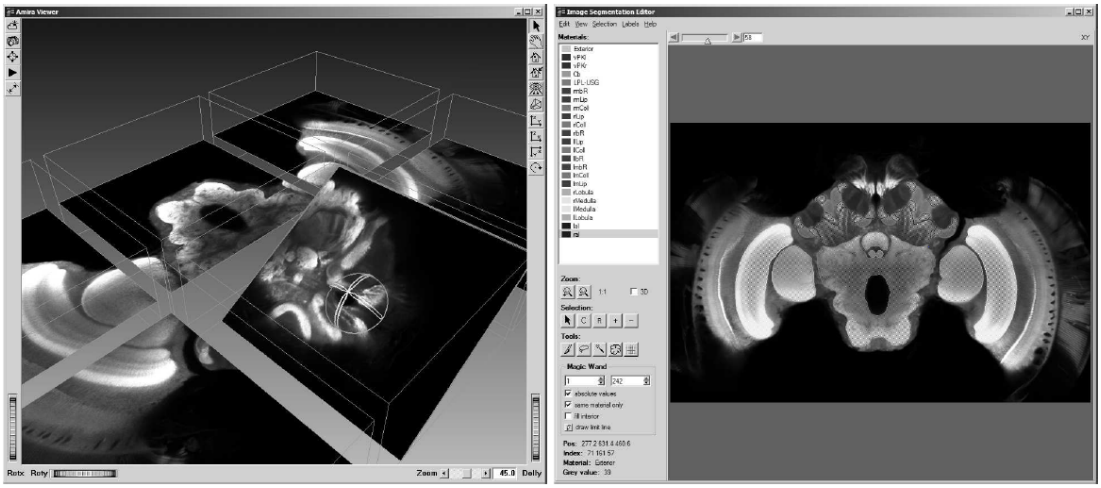


Figure 5.10: High-performance computing in Amira, a system dedicated to interactive manipulation and visualization of scientific datasets. Image copyright Mercury Computer Systems SA (c) 2010, based on the work of [Stalling et al., 2005]

Several newer platforms, such as Amira (Figure 5.10), offer high performance solutions for interactive visualization of large datasets [Stalling et al., 2005]. One major constraint of artist' systems such as Maya and Houdini is that it is possible to "stall" the system (making it appear to crash) by asking it to compute more than it has power to at an interactive rate. For example, creating a particle system beyond a certain number of particles can cause this. While high performance scientific visualizations often deal with a single large-scale data type particularly well (e.g. geographic, volumetric), this issue is potentially greater in creative systems since the system cannot know ahead

of time how much structure or computation the user will request. In the future, due to LUNA's profiling design, it should be possible to have LUNA dynamically adjust dataflow itself to automatically determine scene detail so that the interface never halts. Since the maximum count can be controlled at each step, and the profiler can calculate the time required for a given node, the rendering system could create a feedback loop where scene detail is dynamically adjusted to meet performance needs.

With the advent of parallel computing using GPUs, it is likely that live performance by media artists will change dramatically in the next decade. In addition to providing tools that give the artist direct control over allocation of resources, LUNA is demonstrably faster than other systems at certain tasks (see Chapter 5) due to its dataflow architecture.

5.5 Summary

In retrospect this chapter has focused on three dimensions of media frameworks that *do not* relate to the content of the system. Language syntax, data type (media), and performance establish the rules of the grammar in which meaning can be *potentially* realized by any computing language. In a traditional view of media applications these rules are found together with specific tools. For example, the syntax of Photoshop is the image, while its operations are all image processing tasks. However, with the development of languages for interactive multimedia the syntax may be so broad that the range of tasks is continually changed by future users, at which point its output

cannot be predicted by the inventor since it becomes an open system. At present LUNA generates shapes and tubes with a particular style, but this is because of short-term content decisions, discussed in the next chapter, rather than due to the visual language itself. Interestingly, from the perspective of evaluation criteria, the only criteria directly affected by the 'open ended' aspect of the LUNA language is high ceiling. The ceiling, or expressive power, for a visual language cannot be known in advance since the vocabulary continues to evolve.

This still leaves a question: How do we evaluate languages for media artists? First, future content and changes in style should be supported by allowing node authorship (making new nouns) in addition to visual authorship within the language (making noun phrases), as is the case with LUNA. Secondly, what is the threshold for using the system? How easy is it to learn the language? The constraint of a non-mathematical interface in LUNA's visual language led to choices for a minimal design, specific use of color, and simplicity. Finally, how flexible are the range of ideas one can explore? How easy is it to change ideas? Wide walls relates to the flexibility of the language, which is realized in LUNA through the combinatoric connections between tiles.

Media frameworks may also be evaluated according to their limitations. A simple guide in relation to the goals of media artists concerns modality. Max/MSP and Soundium are limited in their support for three-dimensional forms. LUNA supports such forms but is limited in the areas of audio and device interaction, although these may be expanded in the future. A deeper question relates to constraints of the language

itself: Despite the community authorship of objects, what content might be implicitly unavailable in the system? This is usually not discovered in media frameworks until the vocabulary has been sufficiently explored by others. For example, in the future LUNA could handle complex geometries, real-time displacement, audio, and volumetric data as a source of input through community authorship. Although the following is a speculation, the inherent limits of the LUNA language may reside at level of co-dependence between two complex systems, such as a tree structure growing along an abstract geometric surface which is itself dynamically changing. Suffice it to say that LUNA achieves the workflow goals of low threshold, high ceiling, and wide walls based on the current interface design without knowing where its ceiling lies. In the future, as with any language, it is hoped that the system continues to evolve beyond the initial content defined by the author.

Bibliography

[Arisona, 2007] Arisona, S. M. (2007). Live Performance Tools. Digital Art Techniques. ACM SIGGRAPH 2007. Course Notes.

[Bar-Zeev, 2007] Bar-Zeev, A. (2007). Scenegraphs: Past, Present, and Future. http://www.realityprime.com/scenegraph.php, visited June 2010..

[Bhushan, 2008] Bhushan, A. (2008). Jen-Hsun Huang of NVidia at NVision '08: Visual Computing Era is Now. *CeoWorld Magazine*.

[Bolt, 1980] Bolt, R. A. (1980). Put-that-there: Voice and gesture at the graphics interface. In *ACM SIGGRAPH*, pages 262–270, New York, NY.

[Candy, 2007] Candy, L. (2007). Constraints and Creativity in the Digital Arts. *Leonardo*, 40(4):366–367.

[Dam, 1998] Dam, A. v. (1998). Some Personal Recollections on Graphics Standards. In *The History of Computer Graphics Standards Development.*

[Deering et al., 1988] Deering, M., Winner, S., Schediwy, B., Duffy, C., and Hunt, N. (1988). The triangle processor and normal vector shader: a VLSI system for high performance graphics. *ACM SIGGRAPH Computer Graphics*, 22:21–30.

[Dietrich, 1986] Dietrich, F. (1986). Visual Intelligence: The First Decade of Computer Art (1965-1975). *Leonardo*, 19(2):159–169.

[Edmonds et al., 2004] Edmonds, E., Turner, G., and Candy, L. (2004). Approaches to Interactive Art Systems. In *ACM SIGGRAPH*, pages 113–117, New York, NY.

[Farber, 2008] Farber, R. (2008). CUDA, Supercomputing for the Masses. *Dr Dobb's Journal*.

[Foley et al., 1997] Foley, J. D., Dam, A. v., Feiner, and Hughes (1997). *Computer Graphics: Principles and Practice.* Addison-Wesley Publishing Co.

[Fry, 1926] Fry, R. (1926). *Transformations: Critical and Speculative Essays on Art.* Chatto & Windus, London.

[Gray, 1962] Gray, C. (1962). *The Russian Experiment in Art: 1863-1922*. Thames & Hudson.

[Haeberli, 1988] Haeberli, P. E. (1988). ConMan: a visual programming language for interactive graphics. *SIGGRAPH Computer Graphics*, 22(4):103–111.

[Hatt and Klonk, 1992] Hatt, M. and Klonk, C. (1992). *Art History: A critical introduction to its methods*. Manchester University Press, p. 77, New York, NY.

[Hoetzlein, 2010] Hoetzlein, R. (2010). Fluids v.2: A Fast, Open Source, Fluid Simulator. Available at: http://www.rchoetzlein.com/eng/graphics/fluids.htm.

[Jaimes and Jennings, 2004] Jaimes, J. and Jennings, P. (2004). ACM Multimedia Interactive Art Program: An Introduction to the Digital Boundaries Exhibition. *Proceedings of ACM Multimedia 2004*, pages 979–980.

[Kranz, 1974] Kranz, S. (1974). *Science & Technology in the Arts*. Van Nostrand Reinhold Co.

[Levy, 1992] Levy, S. (1992). *Artificial Life*. Pantheon Books, New York, NY.

[MacGregor, 2002] MacGregor, B. (2002). Cybernetic Serendipity Revisited. In *Creativity & Cognition, Oct 14-16.*, Loughborough, Leic, UK.

[Manovich, 2001] Manovich, L. (2001). *The Language of New Media*. The MIT Press.

[Marat and Downes, 2004] Marat, B. and Downes, M. (2004). Visual Programming Languages: A Survey. UC Berkeley. Technical Report: CSD-04-1368.

[Müller et al., 2008] Müller, P., Arisona, S. M., Schubiger-Banz, S., and Specht, M. (2008). Interactive Editing of Live Visuals. J. Braz, A. Ranchordas, H. Araújo, and J. Jorge (eds.), Advances in Computer Graphics and Computer Vision. Communications in Computer and Information Science, 2007, Vol 4, Part 5, p. 169-184.

[Myers, 1998] Myers, B. A. (March, 1998). A Brief History of Human Computer Interaction Technology. *ACM interactions.*, 5(2):45–54.

[Nake, 2009] Nake, F. (2009). The Semiotic Engine: Notes on the History of Algorithmic Images in Europe. *Art Journal*, pages 76–89.

[Paul, 2003] Paul, C. (2003). *Digital Art*. Thames & Hudson.

[Preziosi, 1998] Preziosi, D. (1998). *The Art of Art History: A critical anthology*. Oxford University Press.

[Reas and Fry, 2006] Reas, C. and Fry, B. (2006). Processing: programming for the media arts. *AI & Society*, 20(4):526–538.

[Resnick et al., 2005] Resnick, M., Myers, B., Nakakoji, K., Schneiderman, B., Pausch, R., Selker, T., and Eisenberg, M. (2005). Design Principles for Tools to Support Creative Thinking. *NSF Workshop on Creative Support Tools. June 13-14*.

[Rosenfeld, 1983] Rosenfeld, A. (1983). Picture Processing: 1982. *Computer Vision, Graphics, and Image Processing*, 22:339–377.

[Shanken, 2009] Shanken, E. A. (2009). *Art and Electronic Media*. Phaidon Press, London, UK.

[Shneiderman et al., 2005] Shneiderman, B., Fischer, G., Czerwinski, M., Myers, B., and Resnick, M. (2005). Creative Support Tools. *NSF Workshop on Creative Support Tools. June 13-14*.

[Stalling et al., 2005] Stalling, D., Westerhoff, M., and Hege, H.-C. (2005). Amira: A highly interactive system for visual data analysis. Hanson, C.D. and Johnson, C.R. The Visualization Handbook.

[Strauss, 1993] Strauss, P. S. (1993). IRIS Inventor, a 3D graphics toolkit. *SIGPLAN Not.*, 28(10):192–200.

[Sutherland, 1963] Sutherland, I. E. (1963). Sketchpad: A man-machine graphical communication system. In *AFIPS Conference Proceedings 23*, pages 323–328.

[Verostko, 2002] Verostko, R. (2002). Algorithmic Fine Art: Composing a Visual Arts Score. In *Explorations in Art and Technology, by Linda Candy and Ernest Edmonds.*, page 131, London, UK. Springer-Verlag.

[Verostko, 2006] Verostko, R. (2006). The Algorists, Historical notes. http://www.verostko.com/algorist.html, accessed Sept 2010.

[Wands, 2007] Wands, B. (2007). *Art of the Digital Age*. Thames & Hudson.

[Wessel and Wright, 2002] Wessel, D. and Wright, M. (2002). Problems and Prospects for Intimate Musical Control of Computers. *Computer Music Journal*, 26(3):11–22.