UNIVERSITY OF CALIFORNIA
Santa Barbara

Imagination in Media Arts: Technological Constraints and Creative Freedom

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Media Arts & Technology

by

Rama C. Hoetzlein

December 2010

The Dissertation of
Rama C. Hoetzlein is approved:

Professor George Legrady

Professor Laurie Monahan

Professor Tobias Höllerer, Committee Chairperson

November 2010

Imagination in Media Arts: Technological Constraints and Creative Freedom

Acknowledgements

I would like to thank my friends and family who have supported me through everything; Mischa Hoetzlein, Abram Connelly, Mark Zifchock, Marty White, Claire Fox, Scott McCasland, Ying Liu, Mary Li, and my parents, Nanci Rose-Ritter and Lawrence Hoetzlein. I'd also like to thank the professors in Media Arts & Technology, and other departments at the University of California Santa Barbara, who advised me and whom I worked with including Tobias Hollerer (CS/MAT), George Legrady (MAT), Laurie Monahan (Art History), Alan Liu (English), Curtis Roads (MAT), and Matthew Turk (CS/MAT).

Curriculum Vitæ

Rama C. Hoetzlein

Education

2004-2007	MS, Media Arts and Technology, University of California Santa Barbara. *The Organization of Human Knowledge: Systems for Interdisciplinary Research*
1995-2001	BA, Computer Science, Cornell University - Computer Graphics
1995-2001	BFA, Fine Arts, Cornell University - Mechanical Sculpture

Experience

2009	DreamWorks Animation. *R&D Rendering Intern.* Glendale, CA.
2008-2010	Transliteracies, Department of English. *Project and Team Lead.* Santa Barbara, CA.
2007	Interactive Digital Multimedia, NSF IGERT. *Researh Fellow.* Santa Barbara, CA.
2005	George Legrady Studio, Seattle Library Visualization Project. *Production Lead.* Santa Barbara, CA.
2001-2004	Game Design Initiative at Cornell University. *Co-founder, Lecturer and Outreach Coordinator.* Ithaca, NY.
1998-2000	Mediartspace, Digital Media Lab. *Co-founder and Lecturer.* Department of Fine Arts, Cornell University. Ithaca, NY.

Selected Publications

Oct 2010	R. Hoetzlein, "Reflections on Deep Structure", In *5th Annual Nebraska Digital Workshop.* Lincoln, Nebraska. Oct 2010.
Jul 2009	R. Hoetzlein, "Subjective Media: A Historic Context for New Media in Art", In *Fourth International Conference on the Arts in Society.* Venice, Italy. July 2009.
Oct 2009	R. Hoetzlein, "Alternatives to Author-centric Knowledge Organization", *Implementing New Knowledge Environments (INKE 2009).* Victoria, Canada.

Mar 2009	R. Hoetzlein and T. Hllerer, "Interactive Water Streams with Sphere Scan Conversion". *ACM Interactive Graphics and Games (i3D) 2009.* Boston.
Jun 2009	R. Hoetzlein, "Real-Time Water Dynamics: Practical Rendering of Fluids" *Game Developers Conference (GDC) 2009.* San Francisco, CA.
Jan 2009	R. Hoetzlein and D. Adderton, "MINT/VXF: A High-Performance Computing Framework for Interactive Multimedia." *Future of Media Arts, Science and Technology Workshop (MAST) 2009.* UCSB
Mar 2008	M. Turk, T.Hllerer, S.Arisona, J.Kuchera-Morin, C. Coffin, R. Hoetzlein, et al] "Creative Collaborative Exploration in Multiple Environments" *Association for the Advancement of Artificial Intelligence*, 2008 Symposium
Jun 2007	R. Hoetzlein, "The Organization of Human Knowledge: Systems for Interdisciplinary Research". *Master's Thesis.* U. of California Santa Barbara. 2007.
Aug 2005	R. Hoetzlein and D. Schwartz, "GameX: A Platform for Incremental Instruction in Computer Graphics and Game Design." *ACM SIGGRAPH Educators Program 2005.* Los Angeles, CA. 2005.
Oct 2003	R. Hoetzlein and D. Schwartz, "Computer Game Design as a Tool for Interdisciplinary Education", American Society for Engineering Education. St. Lawrence Section Conference, Queens University, Oct 2003.

Abstract

Imagination in Media Arts: Technological Constraints and Creative Freedom

Rama C. Hoetzlein

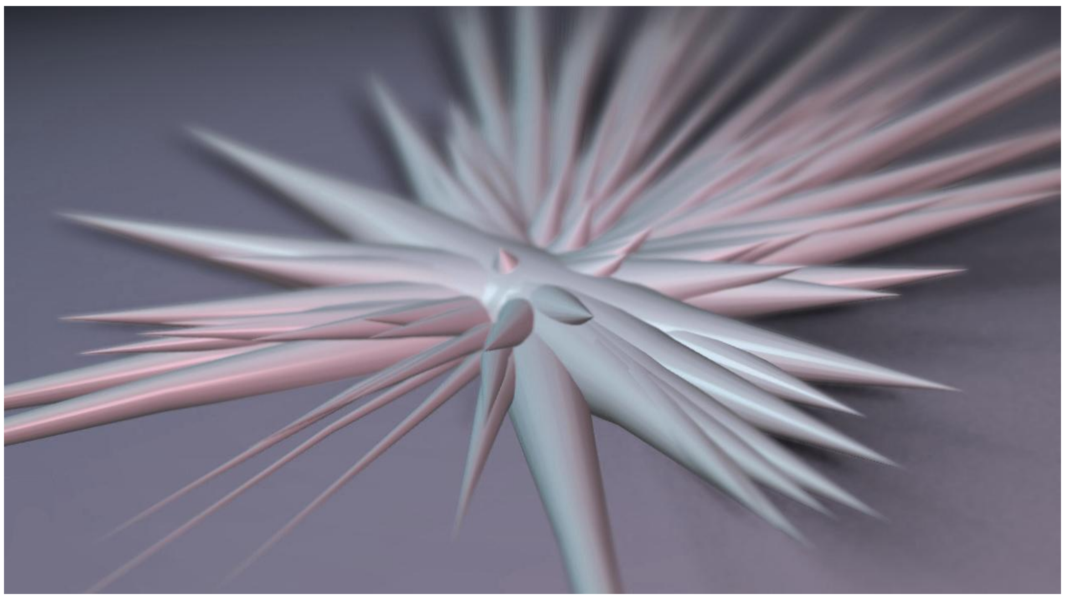
Media artists explore a wide range of techniques for developing art. These techniques, found in current tools for artists, come with inherent limitations which, over time, may separate creative communities into groups that are familiar with particular methods. The question examined here is whether these techniques can be integrated into shared, multi-dimensional frameworks for use by a larger community. To study this question, dimensions of creative interest to media artists are examined, including: 1) programming and language, 2) modality and media, 3) live performance, 4) motion, dynamics and autonomy, 5) structure and surface, and 6) image and idea. A novel visual dataflow language, LUNA, is developed to integrate these aspects of creativity into an open system for collaborative exploration. Technical aspects of modeling, language development, and rendering are addressed and evaluated according to creativity metrics, comparisons with other tools, and measured performance. Visual results cover multiple projects in interactive art, game design, biology and media arts.

Contents

List of Figures

Chapter 1

Introduction



1.1 Overview

Media artists and engineers engage in a wide variety of techniques for making art. These techniques may include working with physical media or devices, apply existing tools, or developing their own tools. In the 1950s scientists such as Michaell Noll, Ken Knowlton and Freider Nake developed the first digital images, and the computer presented itself as a new form for making art [Dietrich, 1986]. Since then the variety of forms of expression using digital techniques has expanded rapidly. Some digital artists, such as the Algorists, began making art by writing computer code expressing a particular aesthetic. In other areas, artist-engineers such as William Reeves, Craig Reynolds and Karl Sims developed code to explore naturalistic behaviors through simulation. Artists began working with databases, computer interfaces, simulations, and generative systems, resulting in many new forms.

As these new fields emerged, many artists and engineers began making their own tools, including the development of integrated frameworks for shared use in research and education. By the mid 1990s, graphics languages such as GINO, PHIGS, IRIS Inventor, OpenGL, and DirectX provided generic interfaces for expressing primitive graphic elements to machines [Dam, 1998]. Languages such as Java, Flash and Processing expanded the use of text-based languages to make them more adept at expressing visual ideas, and allowing media artists greater access to learning how to program. Commercial applications such as Maya, 3D Studio MAX, and Houdini, have allowed digital artists to develop sophisticated work flows for building complex digital worlds, while tools such as Max/MSP, VVVV, Quartz Composer, and Soundium have allowed artists to work with visual and auditory media in installations, exhibits, and live performances.

While there is a great deal of overlap in these forms of expression, over time tools for artists have specialized to serve the needs of each group. As Linda Candy observes, any tool comes with certain *inherent constraints* which limit the range of expression [Candy, 2007]. These may be distinguished from the creative constraints the artist imposes in resolving a conceptual problem, since the inherent constraints of the tool limit creative freedoms beyond the control of the artist. Chapter 1 explores the specialized constraints of existing tools for media artists, focusing specifically on systems for representing shape, form and behavior, areas of particular interest to the author.

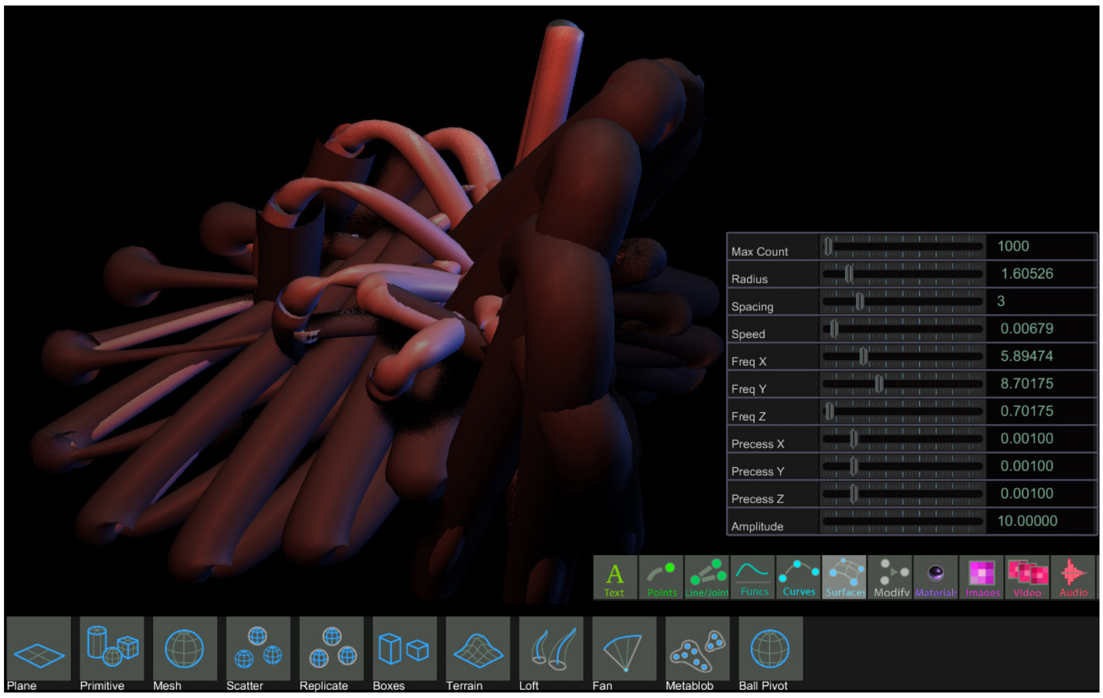
Although every tool has constraints, the question addressed in this work is: Whether the specific constraints and boundaries between current tools for digital artists are necessary or if they are a by-products of the various communities which have developed over time? Are our current technical constraints really inherent constraints of the digital media itself, or are they constraints resulting from the evolution of goals in creative communities? If the former case is true, it would indicate some real distinction in media between different forms of artistic expression. If the latter is true, then it should be possible to develop tools which combine the expressive capabilities of many different tools, reducing work and allowing for greater cross-over in techniques between artistic communities.

This question is addressed by examining current practices by media artists, and through the development of LUNA, a novel visual language for creative expression which integrates these disparate practices. The dimensions of technique investigated here, and present in varying degrees in currently existing tools, are not intended to be an exhaustive map of creative practice, but were selected as a representative set of some essential, recurrent issues for media artists.

The dimensions explored in this thesis include:

1. Programming and Language
2. Modality and Media
3. Live Performance and Computation
4. Motion, Dynamics and Autonomy
5. Structure and Surface
6. Image and Idea

The motivation for these particular dimensions are based on issues that touch several different groups in digital and media arts, and are explored in detail in Chapters 5 and 6. The dimension of Programming and Language addresses the need for low level rule-making in text-based languages in comparison to the desire to quickly mix ideas in visual languages for non-programmers. The dimension of Modality and Media addresses the desire of media artists to work with many different media, including three-dimensional form, images, video, audio, databases, and bringing these into computers from the real world through various devices. Live Performance and Computation addresses the distinction between real-time systems for live performance and offline computation for high fidelity structures and imagery. Motion, Dynamics and Autonomy addresses the desire of media artists to control the dynamic behavior, motion and growth of digital forms, while Structure and Surface addresses the need of certain artists to control and manipulate the visual appearance, surface and style of these forms. Finally, Image and Idea addresses the relationship between images and words, i.e. the ability of machines to look at digital images as semantic objects.



Max Count	1000
Radius	1.60526
Spacing	3
Speed	0.00679
Freq X	5.89474
Freq Y	8.70175
Freq Z	0.70175
Precess X	0.00100
Precess Y	0.00100
Precess Z	0.00100
Amplitude	10.00000

1.2 Summary of Contributions

The primary contribution of this thesis is the development of LUNA, a visual data flow language for exploring the different dimensions in creative expression described above. LUNA was developed over a period of three years, with influences from several other frameworks. Conceptually, the visual language for LUNA is strongly influenced by the board game Scrabble. LUNA is also influenced by other animation languages such as ConMan, Stephen May's AL (Animation Language), Processing, and Soundium.

The novel aspects of LUNA are the contribution of a real-time language for procedural modeling, rendering by deferred shading and a natural, minimal graphical interface for expressing structure and behavior.

Specific contributions of this thesis are as follows:

1) LUNA allows for creative expression along the various dimensions described above, demonstrating that it is possible to develop tools which integrate these creative possibilities into a single framework. These dimensions, and how LUNA addresses them, are explored in Chapters 5 and 6.

2) The graphical interface for LUNA enables non-programmers to rapidly develop complex structures and behaviors in dataflow diagrams. The graphical interface is also distinguished from other commercial packages such as Maya and Houdini in that its method of interaction is based on media rather than work flow. Top-level tool bars in LUNA express structures, while second-level tool bars express behavior. This arrangement, and its benefits, are described in Chapters 3 and 5.

3) While high level behaviors can be combined in the interface, LUNA allows programmers to develop new low level behaviors by authoring new nodes in C++ (programming language). This method of node development relies on the LUNA API to provide a common, shared framework for expressing geometric, image, and object structures while allowing the author to generate new behaviors as desired. This is described in detail in Chapter 5.

4) The procedural language of LUNA allows artists to build and express complex organic structures. This language is based on manipulating and generating multiple objects with standardized, yet expandable, memory structures for representing discrete geometry. (This non-standard representation does not use C++ class variables to store data, but instead uses variable length buffers with named semantics). The language of LUNA is described in Chapter 4.

5) LUNA is shown to be measurably faster than Houdini, a commercial application for procedural modeling, for a procedural reference model developed for testing. The reference model, a woven sphere, cannot be easily created by other modeling techniques, such as physical simulation or hand manipulation of polygonal models, and is presented as a novel object for testing procedural frameworks. Performance results for LUNA are found in Chapter 5.

6) LUNA incorporates a deferred shading engine. Commonly found in game engines, and unique to frameworks for media artists, deferred shading allows for real-time shadows, depth-of-field, multiple light sources, and other advanced rendering techniques, discussed in Chapter 5. The deferred shading engine is capable of running custom shaders on multiple monitors with any number of GPUs and displays running from a single system.[1]

[1]Mutliple display rendering was developed in MINT/VFX based on NSF IGERT work by the author on cluster rendering for the Allosphere, an immersive 30 foot diameter near-spherical collaborative environment for media artists and scientists at the University of California Santa Barbara

7) Profiling tools, which allow artists to tune the computational resources in LUNA, are also incorporated into the framework and provide real-time feedback for live performances. These profiling tools are described in Chapter 5.

8) LUNA allows media artists to explore, mix, and combine a range of different behavioral systems. Some of the current components in LUNA include particle systems, fluid simulation, spiroids (oscillators), spring-systems and others. These can be combined in the interface with high level operators which merge these objects to create more complex behaviors. The behavioral aspects of LUNA are described in Chapter 6.

9) LUNA can express a range of different media structures. These currently include points, trees, curves, surfaces, images and materials, and may be extended in the future to include volumes, audio, video, databases, and networking. Structures in LUNA may be procedurally generated, such as trees, or may be loaded from static data, such as meshes. Currently available structures in LUNA are described in Chapter 6.

10) A theory for digital semantics is presented in Chapter 6 which considers the difference between the digital model and its expression. Based in part on the work of Jorg Shirra, this theory motivates new directions in tools for media arts that consider images and words (or semantics).

11) Experiments by the author, in Chaper 6, which combine hand-sketched drawings with digitally generated compositions show that rule-less images expressed by drawing and photography cannot be completely replaced by digital models, and are thus important elements for visual synthesis into tools for media artists.

Overall, LUNA is presented as a novel language and an open system for media artists. The development of LUNA focuses first on providing a flexible modeling language, which is then populated with a number of specific behaviors or *nodes*. The current nodes were developed according to the author's own interests in sculptural form, geometry and behavior, and to demonstrate certain capabilities across different media (such as interactive shader manipulation). In the future, it is hoped that the capabilities and available nodes in LUNA will be further expanded by the artistic community.

1.3 Criteria for Evaluation

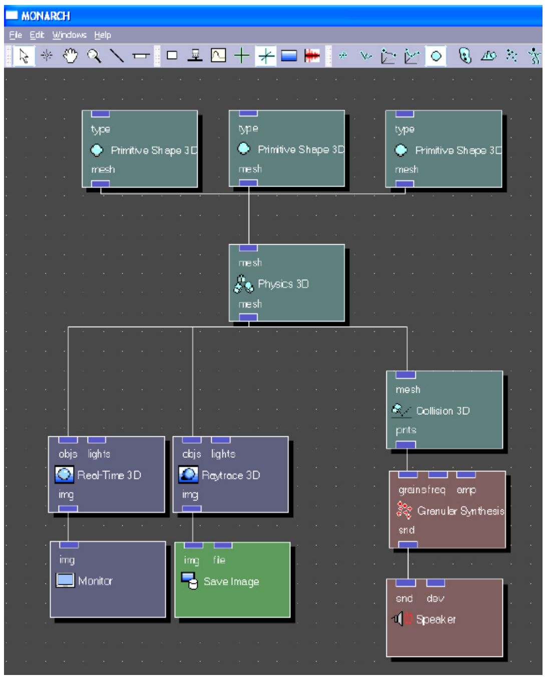
Evaluation of this work is based on a number of different metrics. Overall, the final test of any creative tool is its future adoption by the artistic community. Ideally, to meet the goals set out by LUNA it would be used by artists in several communities to demonstrate the ability to bridge different practices. Prior to adoption, LUNA is evaluated through a number of collaborative projects it has been applied to, by quantitative performance metrics, through the visual results it achieves, and by criteria established by a 2005 NSF Workshop on Creative Support Tools [Shneiderman et al., 2005].

The primary evaluation of LUNA as a tool for different communities is based on the result of its application to a number of collaborations and inter-disciplinary projects described in Chapter 4. The creative dimensions discussed above are demonstrated in LUNA by showing the system is capable of expressing at least two points along each of the dimensions described. These results represent the abilities of the system according to the particular ways that media artists frame their tools. For example, results in Chapter 6 show that artists can create structural objects such as trees with a realistic appearance in LUNA, but can also manipulate and distort this structure for other aesthetic ends, or to re-contextualize the tree to be applied to fractal structures in biological modeling. Thus, these dimensions represent not only a parametric change in the model, but conceptual shifts in how the artist conceives of using the tool. In this way, LUNA is demonstrated to meet several different aesthetics of interest to media artists.

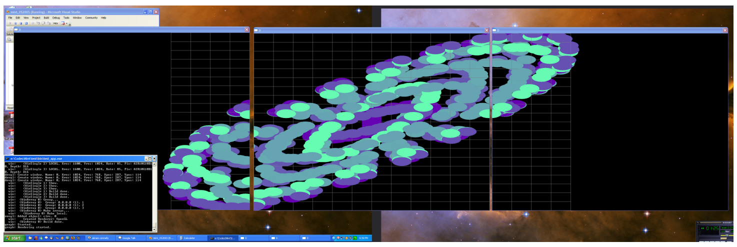
To evaluate LUNA as a tool for supporting creative expression, metrics are introduced from the 2005 NSF Workshop on Creative Support Tools. These metrics are based on the observation that creative tools in various disciplines ideally have 1) low thresholds, 2) high ceilings, and 3) wide walls. Low threshold means "that the interface should not be intimidating, and should give users immediate confidence that they can succeed," high ceiling means that "the tools are powerful and can create sophisticated, complete solutions," and wide walls means that "creativity support tools should support and suggest a wide range of explorations [Resnick et al., 2005]." These metrics, while not quantitative, provide a subjective criteria for evaluating the ability of LUNA to support creativity. The relative relationship between LUNA and other tools with regard to these metrics is explored in Chapter 6.

The computational performance of LUNA is evaluated using a novel procedural reference model, and compared to both Houdini and a baseline model in OpenGL.

Although no user study is developed for the LUNA interface, this reference model is also used to evaluate the ability to perform interface tasks in relation to Houdini.



(a)



(b)

Figure 1.1: Influences on the systems aspects of LUNA include a) Monarch, for interface aspects, b) MINT/VFX, for events and multi-screen rendering, and GameX for graphics architecture (not shown).

1.4 History and Development

The development of LUNA occurred in a series of stages. The early stages of LUNA were influenced by several other media systems frameworks. MINT, a collaboration among graduate students participating in an NSF IGERT project from 2005-2007 at the University of California Santa Barbara contributed ideas regarding the event system and multi-display aspects of LUNA, Figure 1.1a [Hoetzlein and Adderton, 2009]. A prototype interface for media artists, Monarch, was developed by R. Hoetzlein and Jorge Castellanos in 2006 to experiment with creative interactions, Figure 1.1b, although the system had no internal capabilities to simulate objects [Hoetzlein and Castellanos, 2006]. GameX, an earlier project used to co-found the Game Design Initiative at Cornell University in 2002, influenced the rendering system of LUNA [Hoetzlein and Schwartz, 2005].

The next key stage in LUNA, developed through 2009, was the creation of the visual dataflow language. Based on design sketches by the author since 1998, these sketches suggest a method by which different geometric structures may be combined to create new functional structures. Similar to procedural modeling in Houdini or Xfrog, a key difference is that these objects are envisioned as dynamic, complex, high-level systems that may be combined and connected while details are revealed only on demand. The board game Scrabble, in which combinations of single-letter tiles can create a wide range of expressive power, provided inspiration for the layout of the language itself. Each object assumes a basic behavior in which no other information

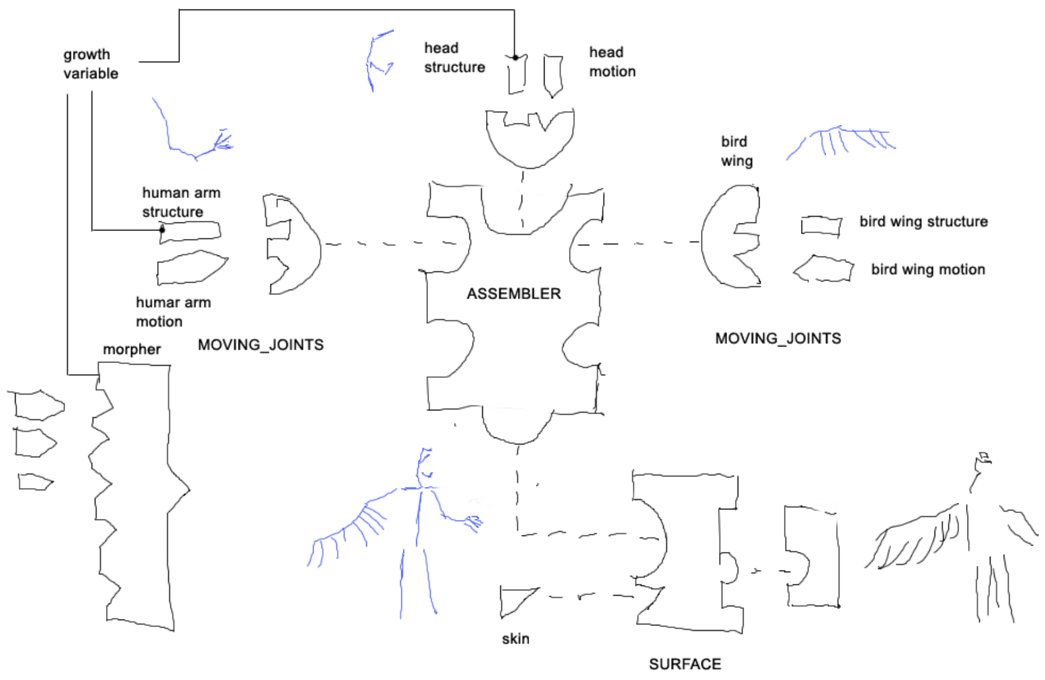


Figure 1.2: Sketches of the LUNA visual language.

is needed for construction, making the system usable by non-programmers. The visual dataflow language is described in detail in Chapters 3 and 4.

Between 2007 and 2010, LUNA was used to develop a number of creative and experimental collaborations. These include *Presence*, with Dennis Adderton and Jeff Elings (2008), an interactive panoramic high resolution display of natural scenes on a custom six-screen display exhibited at the University of California Davidson Library, Figure 1.3a, and *Blocks*, a massive virtual world of dynamic cubes developed with Mark Zifchock, Abram Connelly and Marty White (started in 2003), Figure 1.3b. LUNA was also used in a scientific collaboration with Mock (Panuakdet) Suwannatat and Tobias Höllerer, based on astrocyte imaging results by Gabe Luna, Geoffrey Lewis, and Steve

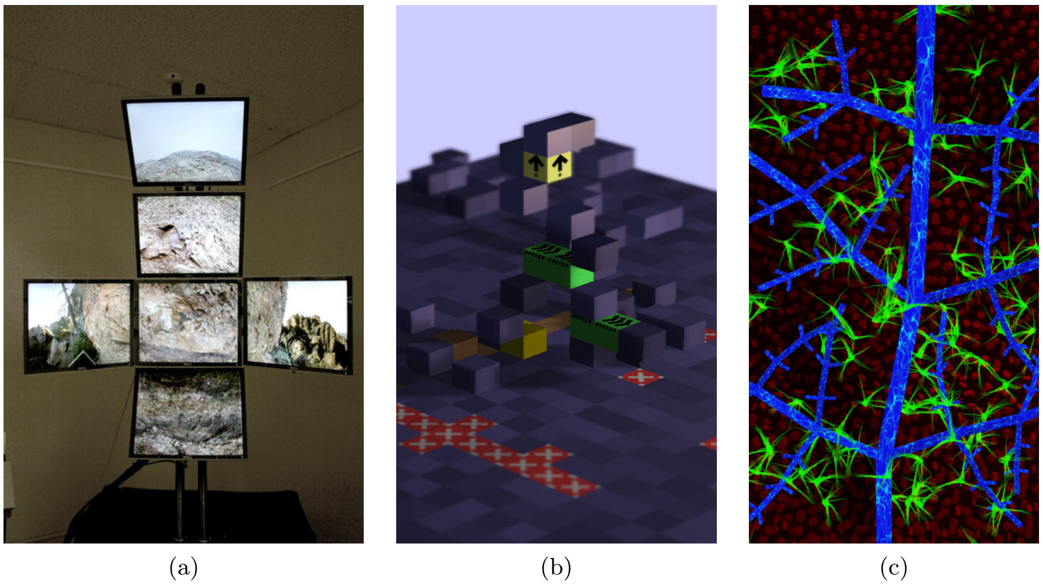
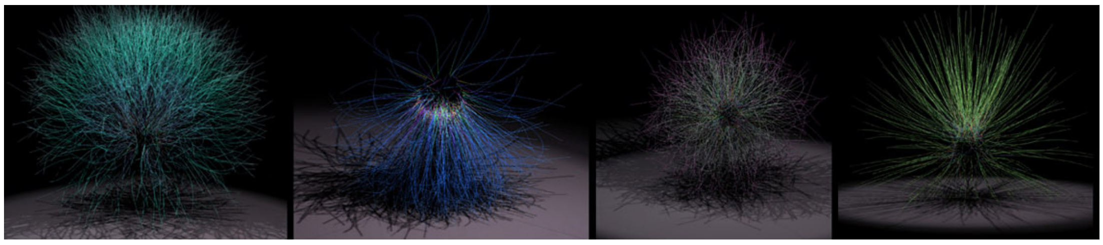


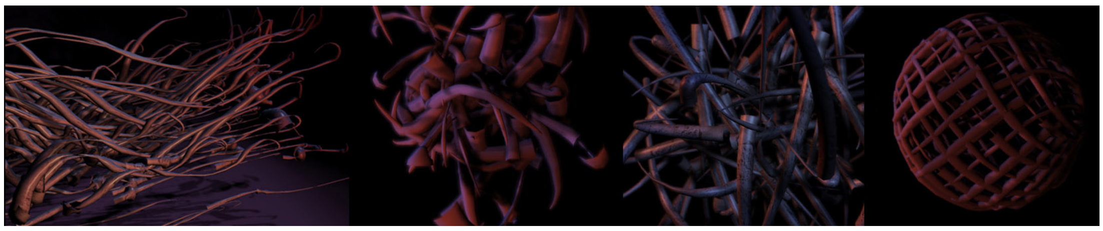
Figure 1.3: Collaborative projects created with LUNA include a) *Presence*, with Dennis Adderton, b) *Blocks*, with Mark Zifchock, and c) *Synthetic Rendering* with Mock Suwannata and the Neuroscience Research Institute (c) 2010. See text for project descriptions.

Fisher (Neuroscience Research Institute, Univ. of California Santa Barbara), and B.S. Manjunath (Dept. of Electrical and Computer Engineering). This was a project to explore *synthetic rendering*, the use of digital modeling to reproduce and render microscopic structures of retinal astrocyte images, showing the systems capabilities in simulating complex models (Figure 1.3c).

The final stage of LUNA (in the development of this dissertation) has been to improve the interface and expressiveness of the system to handle complex structural models. The tree object was added in 2010, as well as a pipeline for material and surface appearances using Cg shaders. Profiling tools were introduced to enable artists to interactively evaluate the performance of the system. The graphical user interface



(a)



(b)

Figure 1.4: Procedural modeling of various organic forms in LUNA based on different behaviors and structures.

was extended to include elements for nested 2D and 3D views, tool bars, scroll bars, and interactive sliders. The visual results of Figure 1.4, show that LUNA is capable of a range of behaviors as well as being able to model complex systems, Figures 1.5.

The most recent additions to LUNA include nodes that synthesize images from image sets (collections of images), using hand-sketched drawings in combination with generative composition. This has resulted in a number of aesthetic experiments and observations that reveal potentially novel art forms, see Figure 1.6,. Artists seeking to work with the hand drawn image may find, in the future, a number of different ways in which digital and traditional media may be combined in LUNA. Details of these techniques are found in Chapter 6.

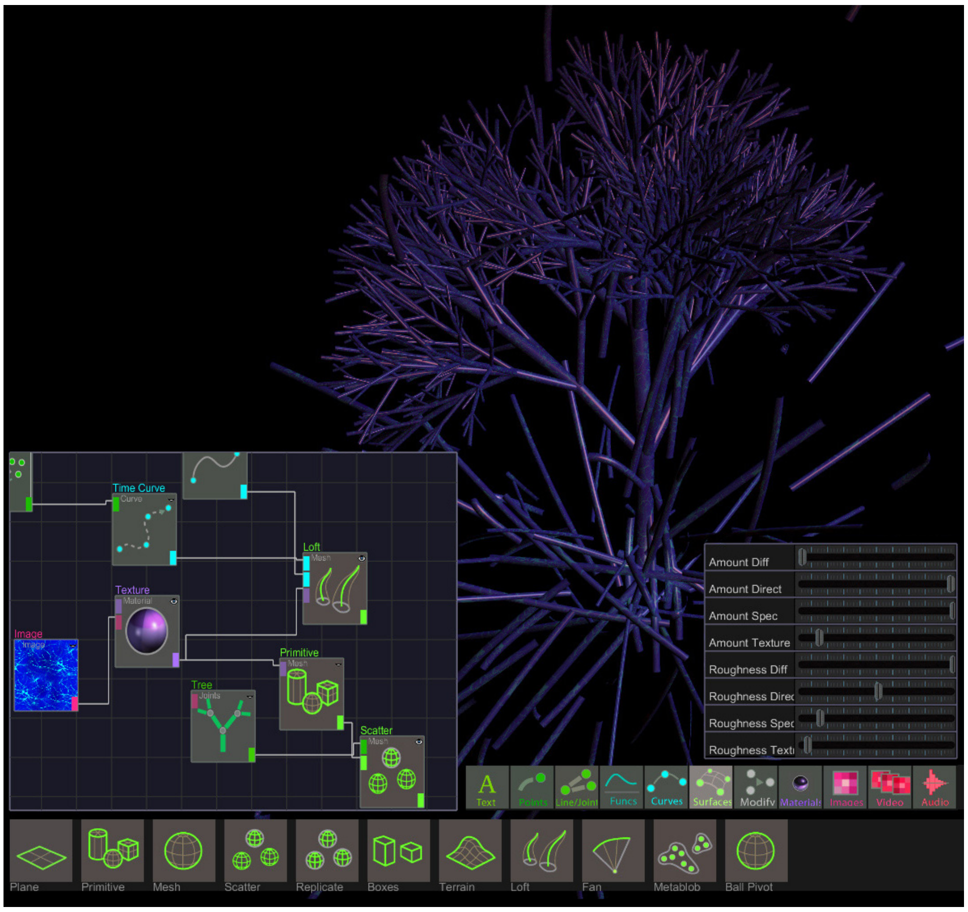


Figure 1.5: Tree modelled in LUNA. Different aesthetics are achieved by modifying the graph, which expresses behavior, geometry, and appearance.

1.5 Chapter Outline

The notion that media artists work best by using a handful of different tools suited to specialized tasks for creative work is challenged by the development of a novel visual data flow language, LUNA, which meets the needs of several creative techniques simultaneously. While LUNA achieves depth only in certain areas, such as procedural modeling, it shows that tools that integrate many different modes of working are possible while resolving many of the issues that arise from developing cross-disciplinary tools. The dimensions along which LUNA explores these creative boundaries include 1) Programming and Language, 2) Modality and Media, 3) Live Performance and Computation, 4) Motion, Complexity and Autonomy, 5) Structure and Surface, and 6) Image and Idea.

LUNA is presented as an experimental system for digital and media artists to easily and rapidly explore visual possibilities through creative bricolage, to engage in a range of techniques without the need to learn new programming languages.

The remainder of the thesis is organized as follows:

- Chapter 2. Tool Survey. Covers existing tools and comparisons of major features of interest to media artists.

- Chapter 3. Interface Design. Addresses the design of the graphical interface for LUNA, with interface comparisons to other systems.

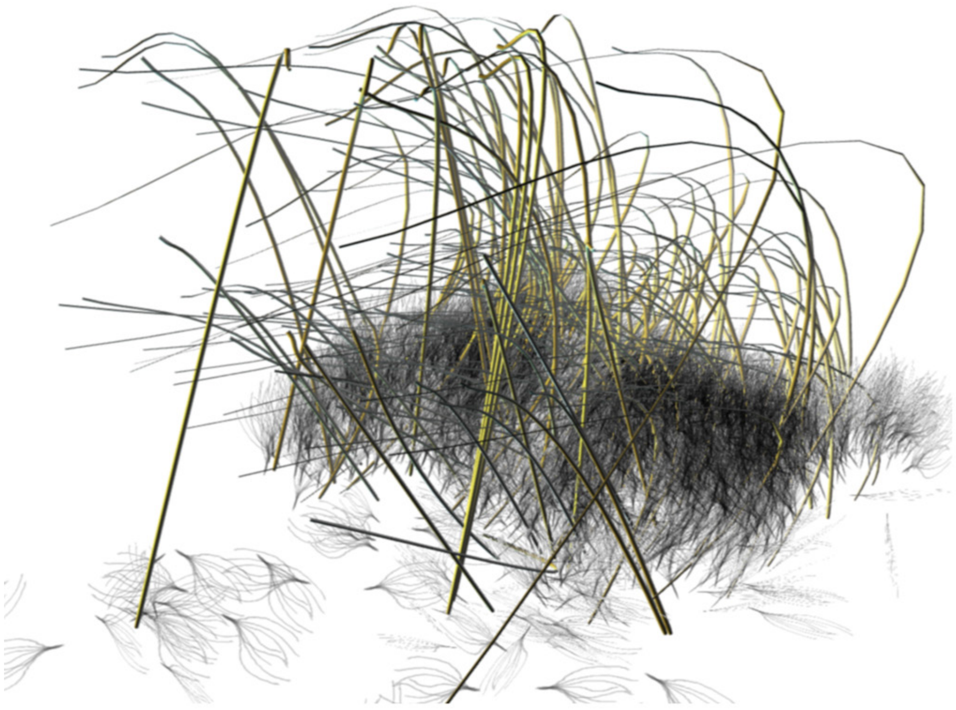
- Chapter 4. Procedural Modeling. Develops the procedural language for LUNA itself, and establishes the storage and memory structures used, along with performance comparisons to Houdini.

- Chatper 5. Creative Workflows for Media Artists. Establishes six dimensions of creative exploration of interest to media artists, and outlines the evaluation criteria to be used. Discusses the first three dimensions, which relate to the language of LUNA.

- Chapter 6. Structure in Dynamic Media. Discusses the last three dimensions of creativity which deal with the content aspects of LUNA, specifically dynamics, structure, and image.

- Chapter 7. Conclusions. Covers limitations of, and potential future directions for LUNA.

LUNA demonstrates that it is possible to develop a system that combines the conceptual needs of various techniques without sacrificing the primary goals of each. Although specific tools may each have inherent constraints, the division of creative techniques in media arts into separate tools is more likely to be a function of the separation and evolution of goals in different creative communities than a result of any inherent limitations of digital media. LUNA demonstrates this by presenting an alternative tool for media artists designed to explicitly resolve several of these techniques into a single tool, enabling artists to work together across creative boundaries.



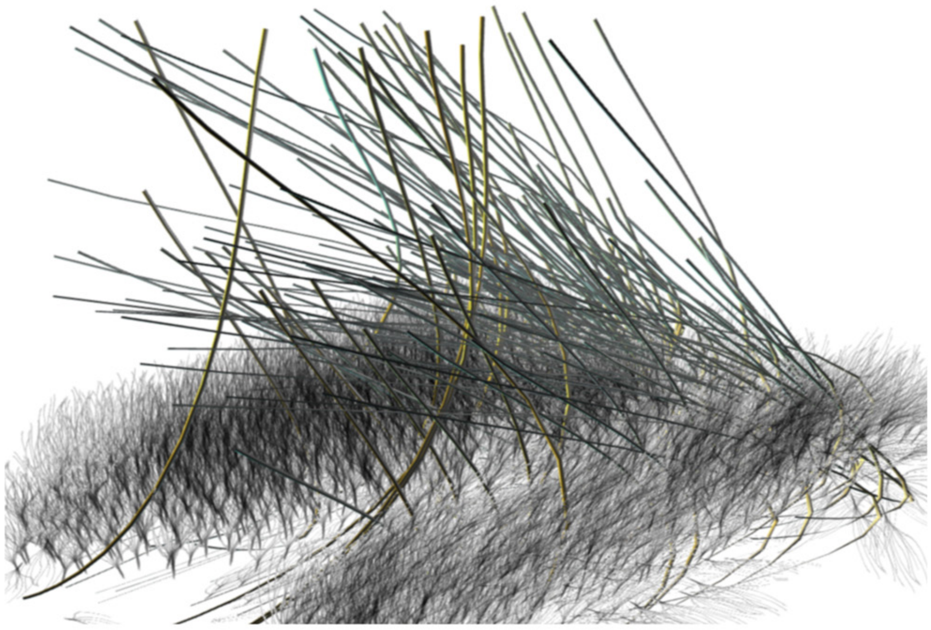


Figure 1.6: Hand-sketched images combined with generative modeling. The technique is described in detail in Chapter 6.

Chapter 2

Tools for the Visual Media Artist: A Survey

2.1 Changing Practice in Media Arts

The practices of first generation media artists are significantly different from current ones, as artists engaging with the computer for the first time had to deal with a different set of problems than those working today. For example, first generation artists - those working in the late 1960s and early '70s (Michael Noll, Freider Nake, Charles Csuri) - did not have generic graphics languages that could describe basic shapes, and found it necessary to implement these directly [Dietrich, 1986]. Artist-scientists at the time began developing the first computer languages for visual elements, such as Frieder Nake's COMPART ER 56 and Leslie Mezei's SPARTA. These tools, while mostly experimental, set the context for graphics systems that would follow.

Today, graphics tools for visual artists are abundant. Many languages, such as Java, Flash, and Processing, are based on the metaphors of earlier text-based languages, and invite the artist to be programmers themselves. Such systems allow a great deal of flexibility in describing behaviors. Other tools, such as Maya, Houdini, Xfrog and Massive, present the artist with an application environment in which to express visual objects. These systems generally make it easier to represent complex geometries, with some focusing on hand-manipulated and articulated digital modeling while others focus on procedural, or computationally, generated models. Still other tools present the artist with visual data flow languages for interactively connecting objects to express ideas. Since visual languages allow one to rapidly experiment with different configurations and behaviors, these are often used in live performances with real-time graphics, examples of which include Max/MSP, Soundium, and Quartz Composer. A final class of tools are research frameworks, prototype systems which give a glimpse at how certain aspects that are critical to artists may be resolved in the future, and include systems such as Squeak, Scratch, and Alice. Artists have used these to experiment with programming education, virtual worlds and robotics.

With such a prolific choice of tools, one may wonder if it is possible to integrate these approaches into more unified frameworks. While choice is generally agreed to be an asset to artists, there are numerous problems presented by having so many different tools. First, if an artist learns a particular language such as Java, and then wishes to explore geometric structures, he or she may need to invest additional time in a new language. Second, some tools are better than others at certain tasks, which may force the artist to switch tools. Maya, for example, provides extensive support for human character modeling while Processing does not. Artists with such interests are either forced to find another tool or must implement these structures themselves (at great cost in time). The nature of interaction with the tool is also critical. Soundium allows artists to dynamically, and interactively modify visual output while the system is running. For those interested in live performance, this eliminates all other tools that are not oriented toward real-time interaction and output. Finally, some features critical to particular groups of artists, for example those interested in multi-screen output, may be limited to only a few tools not capable of other aspects they wish to explore.

The problem may be summarized as one of inter-operability. While all of the tools available to artists cover the totality of what digital artists may currently do, their lack of communication means that this totality is not actually realized without years of learning many different systems. One approach to this problem is to connect various tools together using communication and scripting languages such as OpenSC and Lua [Cerqueira et al., 1999]. However, this does not address the fact that certain structures are common across several tools, and can therefore end up in conflict with one another. Maya, for example, supports character modeling but uses its own proprietary renderer for real-time viewport rendering. Chromium is a low-level graphics system that supports multi-screen rendering, yet combining this with Maya may result in a dramatic loss of performance. To give another example, Houdini allows one to build objects declaratively (as a procedural model), while Max/MSP output is based on the idea of signals flowing through a graph. There are certain similarities between these languages yet their integration must take into account both ways of thinking of data.

Can systems be built which address the multiple dimensions of existing tools? This question is considered throughout this dissertation by examining several dimensions of interest to media artists. These include: 1) programming, 2) modality and media, 3) live performance, 4) dynamics and behavior, 5) structure and surface, and 6) image and idea. While these dimensions are not exhaustive, they cover aspects of sculptural form, live performance, and behavior, which are of interest to the author. A similar set of questions could be formed around sound, information aesthetics (data), or game design.

The dimensions examined in this thesis cover forms of expression which may be in conflict in current tools. To examine the current state of tools for visual media artists more carefully, this chapter provides a survey of a few tools in the above areas of interest. The tools considered here, and the reasons for their inclusion, are:

a) Processing - for its ability to express complex behaviors in a text-based language
b) Max/MSP - for its signal processing metaphor, and its use in live performance
c) VVVV - for its ability to achieve high performance visuals on multiple displays
d) Xfrog 5 - for its ability to declaratively model complex, organic objects
e) Groboto - for its ability to model abstract objects through generative, grammatic rules
f) Houdini 10 - for its ability to procedurally model dynamic, complex behaviors and moving systems

Five of the six languages above are visual data flow languages, as this is the approach taken toward LUNA, the integrated system described in this thesis. While many other languages could be examined, these represent a sufficient challenge in terms of the cross-section of features they offer to different communities. Processing is used widely in education, while Max/MSP and VVVV are used in professional live performances. Xfrog 5 is used as a professional system in commercial film for building organic virtual worlds, while Houdini is used in film for visual special effects. Groboto is used primarily by Braid Media to create organic art, and presented to the artistic community as an experimental system for playing with grammatic forms. From a creative perspective, it would be ideal if one could use the features of each without having to learn each of these systems.

2.2 Methodology: Inherent versus Creative Constraints

There are many ways that digital tools for media artists might be evaluated. As a basis for understanding these tools we might begin by considering their features. However, it would be useful to be able to connect these features to artistic practice rather than considering them in isolation. One possible approach, suggested by Linda Candy, is to consider digital tools as materials which *constrain* artistic process.

> "Constraints in creativity are both limiting and liberating. They are used to impose boundaries upon the creative space we occupy and at the same time enable us to grapple with inherent tensions between different demands, which may lead to a new idea, direction or artifact. When we choose particular forms, materials and tools for our creative work, we are also choosing the kinds of constraints that will shape our process and its outcomes [Candy, 2007]."

We would like to answer questions such as: When do digital tools help the artist? When are they barriers? Answers to these questions would suggest ways to improve our tools. However, as Candy mentions, for the artist, constraints may be both a positive, useful factor or an imposing one. Thus it is not entirely clear in which direction the tools should evolve. Consider, however, that the "choice of a tool" directly leads to the "kind of constraints" that shapes its outcome. This implies that there are *inherent constraints* which are not at all associated with the artist, but are naturally part of the tool itself. Consider that traditional painting requires a finite flat space while digital painting does not (it may be infinite). This may be understood as an inherent aspect of the art object coming into being through a media, and may be analyzed through Aristotle's four causes, presented here by Heiddeger:

> "For centuries philosophy has taught that there are four causes:
> (1) the *causa materialis*, the matter out of which, for example, a silver chalice is made
> (2) the *causa formalis*, the form, the shape into which the material enters
> (3) the *causa finalis*, the end, for example, the sacrificial rite in relation to which the chalice is required, determined as to its form and matter;
> (4) the causa efficiens, which brings about the effect that is the finished, actual chalice, in this instance, the silversmith [Heidegger, 1982]."

The silver chalice itself has certain *inherent constraints* due to it being made of silver, that have only to do with the choice of silver: ductility, weight, and color. This may be distinguished from the artistic choice of using the metal silver, the shape of the chalice, its purpose, or the message it conveys. The following definitions help to distinguish *inherent constraints* from *creative constraints* in examining digital tools.

Inherent constraints: Rules imposed by the medium selected by the artist to resolve the material cause of the work.

Creative constraints: Rules imposed by the artist to resolve the process and idea toward the formal and final causes.

Of course, the artist is free to choose a particular tool, which by itself is a creative constraint, but once the choice is made the tool brings its own additional inherent constraints. George Whales explains that it can be quite difficult to determine "which limitations are real and which are illusory [Candy and Edmonds, 2002, p. 251]." He gives the example of a virtual reality system, initialized with four walls by its creators, seen as an artificial limitation by the artist. In this example, the four walls at first appear to be an inherent constraint imposed by the system. Yet these are easily removed. Thus, the flexible limitations in technical media are due to different layers of the medium being either loosely constrained or deeply constrained - to the programmer there is no hard boundary between inherent constraints. It is easy to remove the ground plane from a 3D modeling program; it is more difficult to convert a 2D modeling system into a 3D one. Thus, it is essential that as artists work they learn the fundamental premise of particular systems.

Creative constraints, on the other hand, are those introduced by the artist him or herself to resolve a boundary or inner tension in the work. These are positive factors in that they are conscious, free choices by the artist. For example, to upset the cultural status of painting, Joan Miró chose to work for a time with only black charcoal and found objects in paintings during his “assassination of painting [Miró, 2008].” This is a creative constraint intended to resolve a particular conceptual challenge.

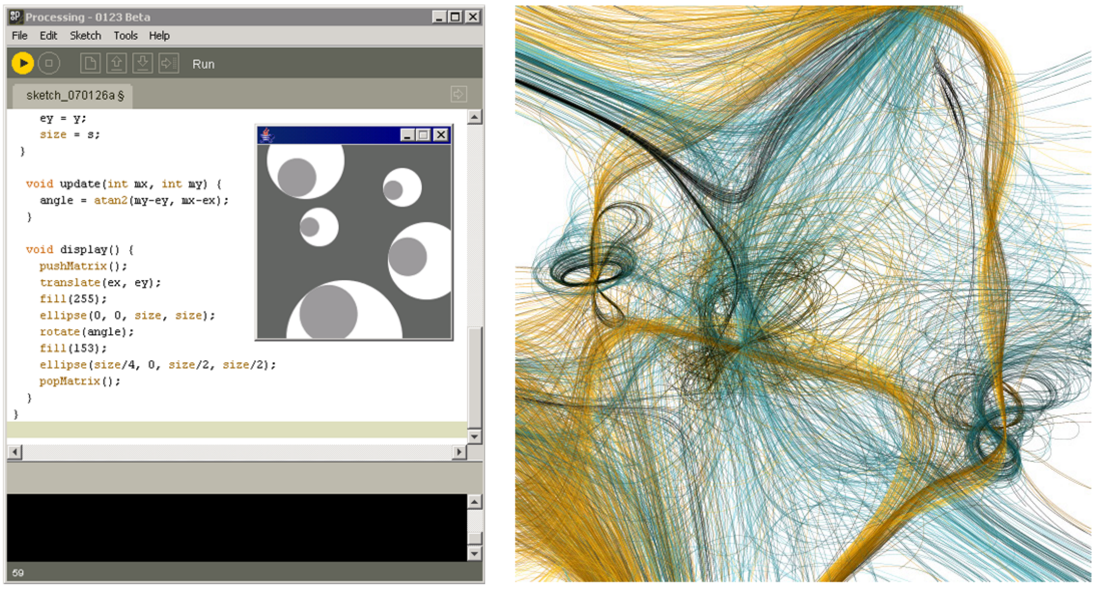
Thus, when speaking of software tools, the *inherent constraints* are those which are most deeply embedded in the system, and one way to conceptualize them is by stating that they remove the element of choice from the artist. When choosing a tool, artists may not know all of the constraints involved, but after a time they learn that some inherent constraints are immovable, and they must either contact the tool developers, re-engineer the system, or select another tool. Whales’ point that this is not easily determined reflects the fact that digital tools are complex systems whose boundaries may not even be fully understood by its developers.

As a basis for analyzing digital media, inherent constraints provide a way to examine tools irrespective of their use. We can ask: Regardless of whether the artist may choose to embrace or abandon a particular tool, what are its inherent constraints? While a “feature set” describes its unique capabilities, beyond basic functionality, inherent constraints describe what would be fundamentally difficult for the artist to do at each level of the tool. This may be a more valuable representation of where digital tools should focus next as it explores what choices the artist would *like* to have available, while a feature only describe what is currently available.

The tools examined in this survey include Processing, Max/MSP/Jitter, VVVV, Xfrog Plants, Groboto, and Houdini. The focus in this analysis is on tools for visual media arts (rather than music), and these tools represent a cross-section of different approaches to the exploration of form and space in media arts. From this point forward the word "constraint" will be used to refer to inherent constraints introduced by the tool, versus creative decisions made by the artist.

2.3 Survey of Tools

2.3.1 Processing



```
    ey = y;
    size = s;
  }

  void update(int mx, int my) {
    angle = atan2(my-ey, mx-ex);
  }

  void display() {
    pushMatrix();
    translate(ex, ey);
    fill(255);
    ellipse(0, 0, size, size);
    rotate(angle);
    fill(153);
    ellipse(size/4, 0, size/2, size/2);
    popMatrix();
  }
}
```

Figure 2.1: Processing, software for media arts developed by Casey Reas and Benjamin Fry, shown next to artwork by Casey Reas. *Path 00*, 2001. Print on velvet, 32"x32". Image by Casey Reas (c) 2001.

Processing was developed and first released in 2001 by Casey Reas and Benjamin Fry, both originally from the Aesthetics and Computation Group of the MIT Media Lab. Processing is a free, text-based language derived from Java which was written to "promote easy-of-use" in the creation of media artworks.

> "Processing was created to teach fundamentals of computer programming within a visual context, to serve as software sketchbook, and to be used as a production tool. Students, artists, design professionals, and researchers use it for learning, prototyping, and production [Reas and Fry, 2006]."

Users of Processing have created a wide array of projects, examples of which can be found on the processing.org website. Due to the authors' background in information visualization, projects created with Processing tend to have an information aesthetic, although this may also be influenced, in part, by the use of its base language Java. Processing's functionality, for example, is not particular well suited to 3D graphics and Java is not the language most commonly used for 3D due to its performance.[1] However, Java is a hardware-independent language, which means that Processing projects are more easily capable of being run directly in a web browser.

As a text-based language, Processing requires some programming experience, but this is exactly what it was intended to teach. Processing is one of the first tools to allow novice programmers the ability to quickly prototype and experiment with simple, animated, and generative two-dimensional images and shapes. Although more involved, data generated in Processing may be exported to other tools, such as Open Sound Control (OSC) for audio synthesis, or to third-party rendering tools, an example of which is *Platonic Solids* by Michael Hansmeyer [Hansmeyer, 2010]. Artists have continued to extend Processing with hardware input and output (camera tracking, LEDs), and have used Processing in exhibitions worldwide.

[1]The most common language for 3D graphics is C/C++

In understanding the inherent constraints of a tool, the best resource is a language reference. An online reference shows the base functionality that Processing offers.[2] This includes lines, arcs, quads, images, Bezier curves, noise, matrices and mathematical operations, a tool set which is oriented primarily toward drawing of fundamental two-dimensional shapes. While this language is natural as a learning tool, it constrains the output to a certain class of objects. Output resolution may be limited in size, and while intended for 2D, performance may not easily allow tens of thousands of objects. Although it would be possible for an artist to author code to animate two-dimensional articulated figures, these are not part of the base language. While some 3D features are available in Processing, its ability to animate solid, three-dimensional forms is not its primary use, and while it allows for single-frame video processing, it is also not intended as a video editing tool. Processing's strength is in the autonomous generation of abstract two-dimensional shapes, and its ease of use as a programming language, which can be seen in project samples (see Figure 2.1).

2.3.2 Max/MSP/Jitter

Max/MSP was created by Miller Puckette, who was also at the MIT Media Lab from 1985 to 1987. Since then, he developed Max/MSP and Pure Data (Pd) as graphical programming languages for music synthesis. While primarily a tool for music synthesis, Max/MSP/Jitter is considered here due to the introduction of Jitter in 2003 by Joshua

[2]This reference can be found at http://processing.org/reference/

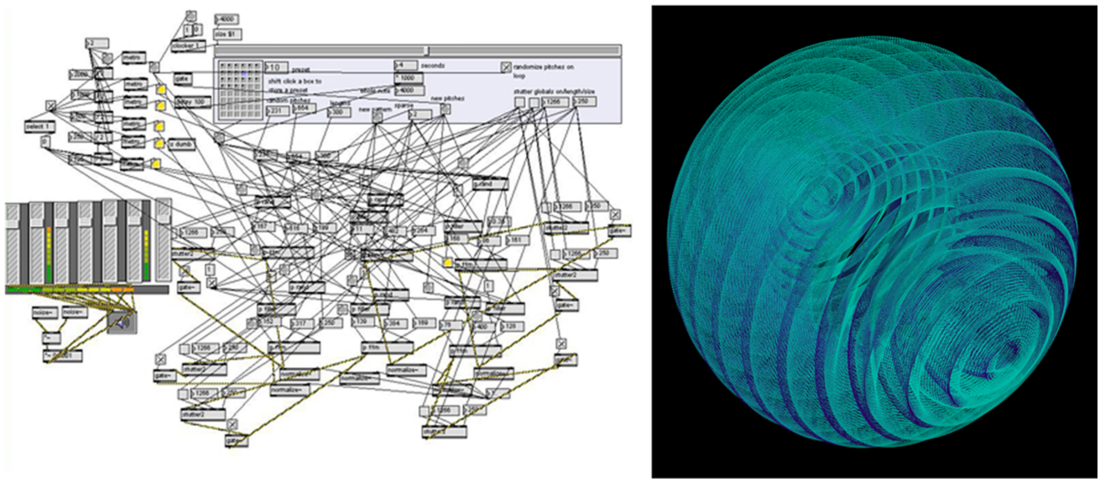


Figure 2.2: Max/MSP, software by Cycling 74, developed by Miller Puckette with visuals using Jitter developed by Joshua Kit Clayton in 2003. Artwork by Christopher James (c) 2006, Third Space Mind.

Kit Clayton, which provides support for matrices, and visual output in OpenGL. Matrices are an essential aspect to the use of Max/MSP/Jitter as a visual tool:

> "It is important to note that what we have called the spatial dimensions of a matrix need not be interpreted spatially. For instance, as we will see later, it is possible to transcode audio signals into one-dimensional matrices for Jitter-based processing, or to represent the vertices of an OpenGL geometric model as a multi-plane, one-dimensional matrix [Jones and Nevile, 2005]."

The concept of transcoding is central to the Max/MSP/Jitter workflow. The strength of this is that any object may be interpreted by another component as a different type. A drawback, however, is that the user must be constantly aware of the internal matrix structure, which is not directly visible, as it flows through the graph. In addition, transcoding from a one-dimensional audio signal to a three-dimensional object is not typically a direct process. Thus, it is more common to introduce translators that transcode into the desired output. Nonetheless, the metaphor is valuable for the flexibility it offers.

Max/MSP/Jitter has found a wide user-base in the audio synthesis world with an increasing number of projects using visual output. The interface to Max/MSP is a visual data flow language, which benefits the author by placing the code in the same place as user interface controls. Distinct from text-based languages like Processing, Max/MSP patches look very much like both a visual graph and sound mixing boards [Cycling74, 2010].

The visual programming interface is also a point of some contention as patches can become cluttered. In studying visual data flow languages, Johnston has found that this may be due to visual languages being used to mimic text-based programming [Johnston et al., 2004]. When expressions and equations are represented as nodes in a graph, it requires a larger number of connections to create modules with high-level functionality. As Max/MSP is primary a signal processing tool, this is often the case as signals flow through filter nodes expressed by equations.

Max/MSP/Jitter performs visual output using OpenGL, resulting in higher graphics performance than Processing can provide. Using OpenGL also allows for three-dimensional geometry, Cg shaders and more complex graphical effects. However, typically the author must code these directly as they are not part of the base feature set of Max/MSP/Jitter. This requires knowledge of other languages such as C/C++ or Cg, and also limits possibilities for generative modeling. However, it is important to emphasize that Max/MSP was originally a signal processing tool for music and only recently a system for visual arts.

A large user community has developed around Max/MSP which exchanges code, patches, and modules for reuse by the community. Overall, the Max/MSP/Jitter allows novice artists to develop ideas in a visual interface, and is used increasingly by media artists for professional performances.

2.3.3 VVVV

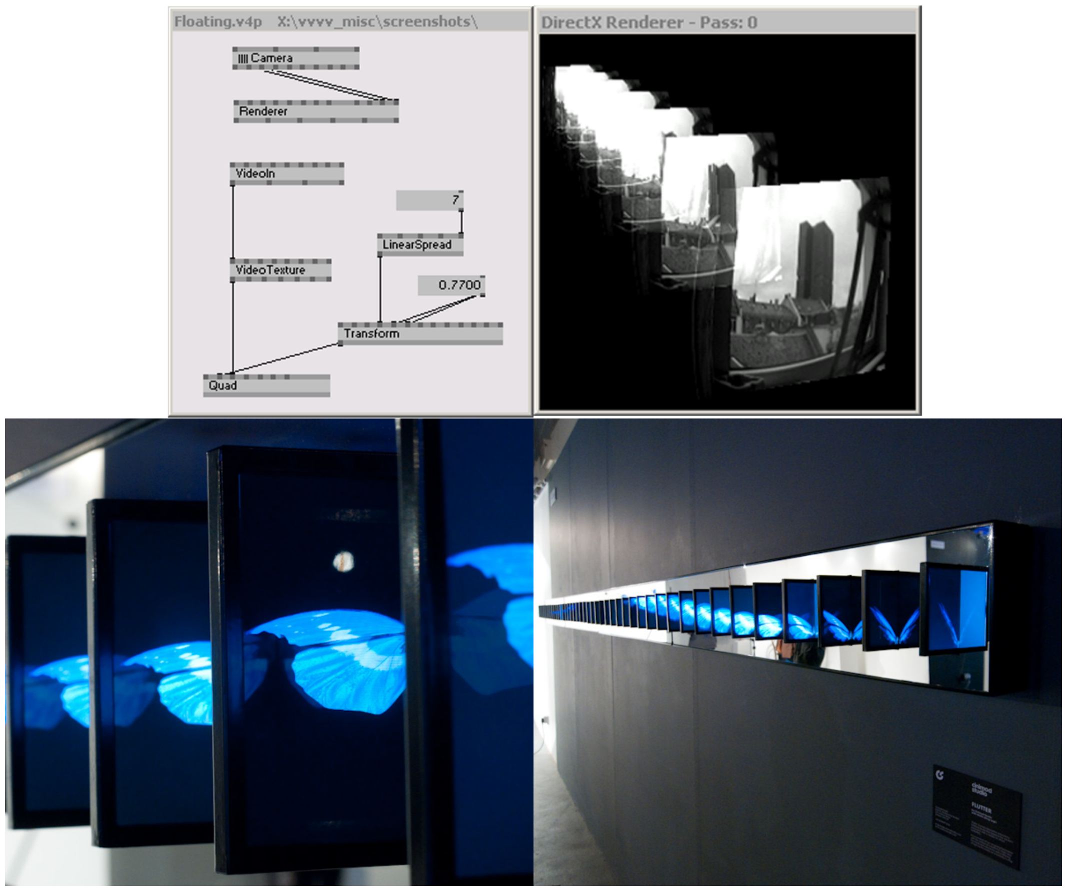


Figure 2.3: VVVV, software for media artists developed by Meso studios and made freely available in 2002. The butterfly sequence, entitled Flutter, is composed on 88 double-sided screens using VVVV. Images courtesy Cinimod Studio (c) 2010.

VVVV was created by Sebastian Oschatz, Max Wolf and Joreg through a company called MESO. MESO was founded in 1987 as a design team of computer scientists and artists to work on large, interactive installations. VVVV was primarily an in-house tool until it was released as free software in 2002 [Meso, 1998].

VVVV also uses a visual programming language to prototype media artworks. Unlike Max/MSP, however, VVVV focuses on the visual arts and includes some high-level components for graphical transformations. VVVV lies between the low-level signal processing of Max/MSP and the generative modeling capabilities of Houdini. A node library provides a wide range of capabilities, from quaternions to 3D animation, to color and video. While VVVV can load static 3D geometry, and has 3D modules, these are not as abstract or generative as a procedural language like Houdini, and creating dynamic three-dimensional forms is equally as difficult as with Max/MSP.

VVVV is best suited to large scale, interactive, visual installations. The Galería shows a number of major projects created with VVVV as well as many gallery installations.[3] VVVV may be considered an installation tool as its workflow and user modules are focused on real-time imagery. A tutorial on the site shows how to use VVVV to project live images onto physical surfaces.

Rendering to multiple displays is a desirable feature among professional artists. Unlike the other systems mentioned, VVVV includes direct support for multiple displays using a client-server system called "boygrouping," in which many client computers are controlled from a server. However, VVVV relies on DirectX for rendering, which restricts

[3]http://vvvv.org

its use to Microsoft Windows systems. DirectX has many of the same features as OpenGL, and is an industry standard for game development, so it benefits from the most recent graphics hardware developments. In VVVV, this can be found in shader support which, as with Max/MSP, must be coded in another language such as Cg or HLSL by the artist.

VVVV is unique as a tool for visual media artists, and as a visual programming language it allows users to create projects quickly and easily. Its language is focused more toward visual output, and features a large number of modules for graphics, images, and hardware. VVVV is used by VJs and artists to create high quality, interactive installations and performances.

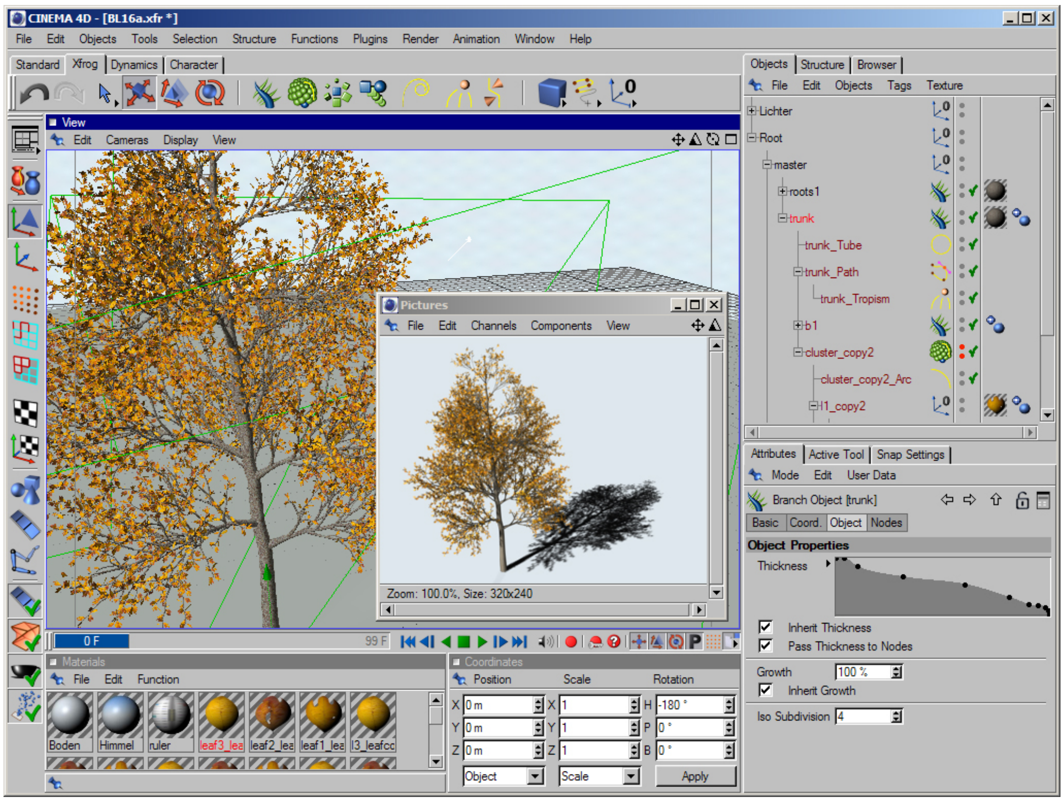

CINEMA 4D - [BL16a.xfr *]
File Edit Objects Tools Selection Structure Functions Plugins Render Animation Window Help
Standard Xfrog Dynamics Character
View
Pictures
Zoom: 100.0%, Size: 320x240
Objects Structure Browser
Lichter
Root
master
roots1
trunk
trunk_Tube
trunk_Path
trunk_Tropism
b1
cluster_copy2
cluster_copy2_Arc
l1_copy2
Attributes Active Tool Snap Settings
Branch Object [trunk]
Object Properties
Thickness
Inherit Thickness
Pass Thickness to Nodes
Growth 100 %
Inherit Growth
Iso Subdivision 4
Materials
Boden Himmel ruler leaf3_lea leaf2_lea leaf1_lea l3_leafcc
Coordinates
Position Scale Rotation
Apply

Figure 2.4: Xfrog is a commercial system for organic modeling and plants by Xfrog, Inc. Xfrog is often integrated into the workflow of other modeling tools, such as Maya or Cinema4D, as shown here. Images by Xfrog, Inc. (c) 2004.

2.3.4 Xfrog 5

Xfrog is a procedural modeling tool developed by Oliver Deussen and Bernd Lintermann for Xfrog, Inc. The authors, originally from the ZKM Karlsruhe institute in Germany, created Xfrog to allow for generative modeling of organic forms [Deussen and Lintermann, 2004]. Unlike the other system, Xfrog is the first tool considered here which uses a procedural modeling workflow to create forms. This method is similar to the way a sculptor works, by successively manipulating models with a specific structure.

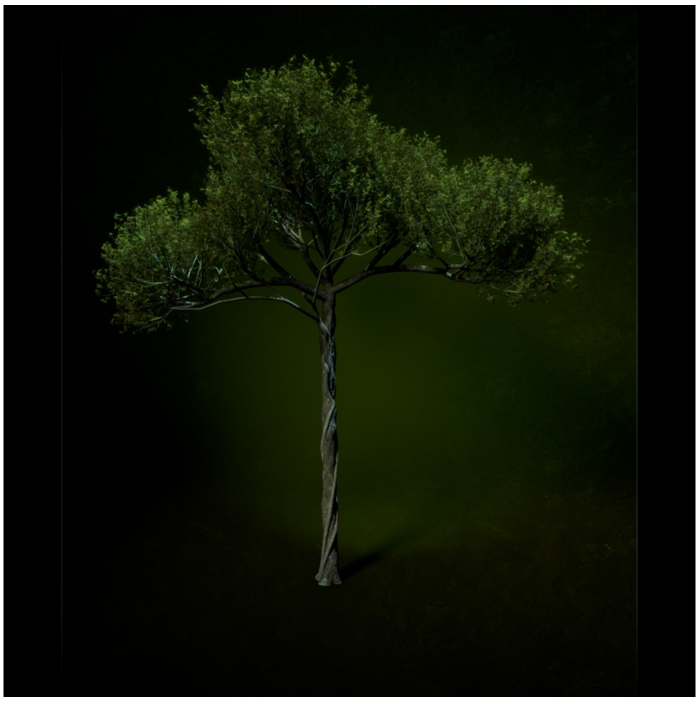


Figure 2.5: *Kleine Spielerei* (2009) demonstrates high quality renderings produced using Xfrog. Image copyright Jan Walter Shliep (c) 2009, http://www.wallis-eck.de

Xfrog is primarily a tool for the visual effects community, and focuses especially on organic and architectural models. The visual dataflow language of Xfrog allows artists to easily create three-dimensional structures like plants, as exemplified by its key modules: Branch object, Phyllotaxis object, Tropism object, Curve object. These structures can be combined in a procedural workflow that allows the artist to work with generative functions [Deussen and Lintermann, 2004, p. 251].

As a production level tool, Xfrog outputs primarily to third-party rendering systems such as V-Ray, MentalRay, or Maya, for high quality, photo-realistic output. Due to its focus on organic modeling, its capabilities for information aesthetics, hardware interfacing, and real-time performance are limited. Although it has a real-time viewport, it is unable to render quality images in real-time for interaction or live performance. However, due to its offline rendering workflow, unlike Max/MSP or VVVV, it can easily render high quality images for large format printing.

While used by digital artists more than by media artists, it is mentioned here because it offers a procedural workflow distinct from the other performance-oriented systems. This workflow, while also employing a visual language, enables structurally defined geometric models to be described using visual grammars. These grammars can express organic relationships such as branching structures or spiral phyllotaxis (e.g. the compact, spiral arrangement of buds on a sunflower), using models and textures defined by the user. The benefit of Xfrogs to graphically-oriented artists is that geometric objects can be expressed as *functional* models that respond to structural changes without having to directly implement primitive geometries oneself.

2.3.5 Groboto

Groboto is a procedural tool created by Darrel Anderson of BRAID Media Artists. Like Xfrog, Groboto uses a visual modeling workflow for generating three-dimensional forms. One distinction, however, is that Groboto focuses more on the behavioral and abstract generative aspects of form than Xfrog, which is realized more as a procedural modeling tool.

Groboto employs a rule-based system for modeling which introduces specific benefits and constraints in the types of objects it can express. This is a system in which objects

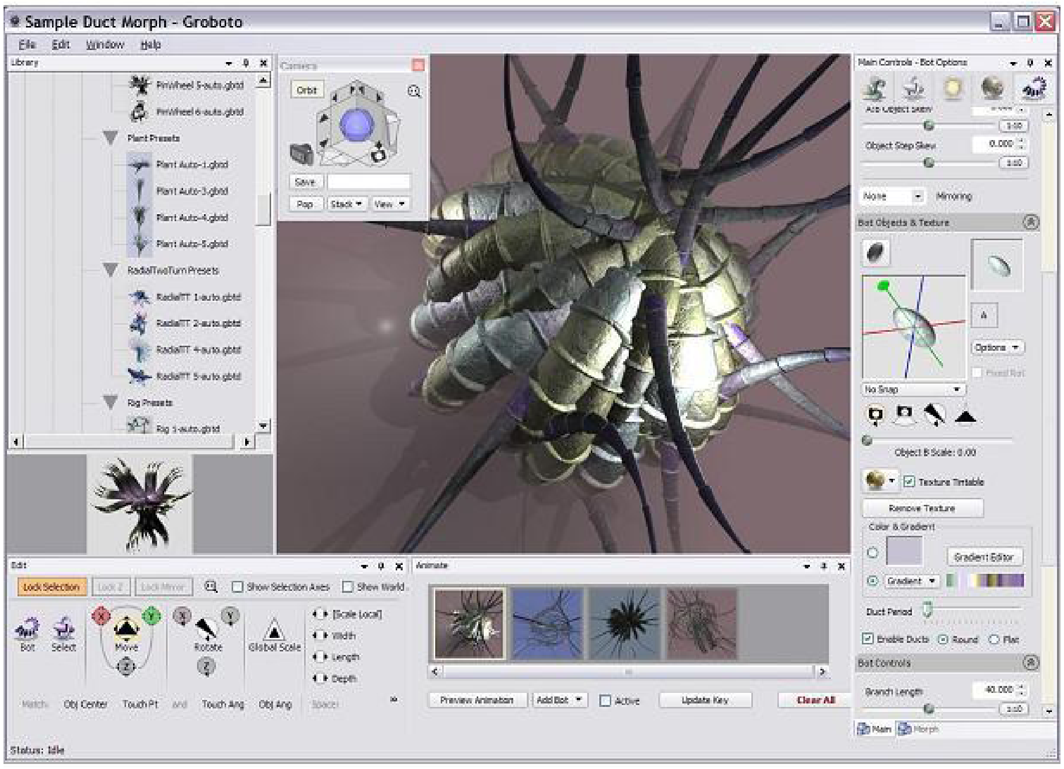


Figure 2.6: *Groboto* software for generative modeling. Image copyright Braid Art Labs LLP (c) 2008.

generate similar forms, or replace forms, in proximity to one another using an automated logic, or grammar [Anderson, 2008]. These grammars are capable of producing complex structures from a very compact initial set of rules. However, unlike procedural models found in Xfrog, these models typically do not have a dynamic functional aspect; they may exist as complex forms in space but cannot also move or respond to changes over time. Later developments in functional systems, as found in Houdini for example (see section 2.3.6), combine the benefits of both grammar systems and procedural modeling.

The gallery examples presented by Groboto exemplify the playful nature of using the system. While Groboto also outputs to third-party renderers, and is therefore similar to Xfrog in this regard, it allows users to very quickly create models of a specific class but with arbitrary complexity. The results, which can be found on the BRAID Media Arts website (http://braid.com), are abstract, generative three-dimensional structures which resemble gravity-free architectures.

These two approaches, procedural modeling in Xfrog plants, and rule-based modeling in Groboto, exemplify potential workflows for media arts which are not yet fully realized as a whole. They are distinct from one another in that they present differing degrees of control and different structural tools to the artist. In addition, Xfrog and Groboto are offline systems which typically do not have the support for real-time rendering, hardware input, and information design which are needed for interactive performances by media artists. Nonetheless, their ability to express complex geometric forms is much greater than the previously examined tools for media artists. The audience for Groboto is targeted toward experimental visual artists, while Xfrog is focused on organic worlds for commercial film.

2.3.6 Houdini 10

Houdini 10 is the flagship software product of Side Effects Software, Inc. Among other Computer Generated Imaging (CGI) companies for motion pictures and entertainment, which includes Wavefront, Alias, Autodesk and Softimage, Side Effects Software was developed to support the special effects industry. Houdini is mentioned here because

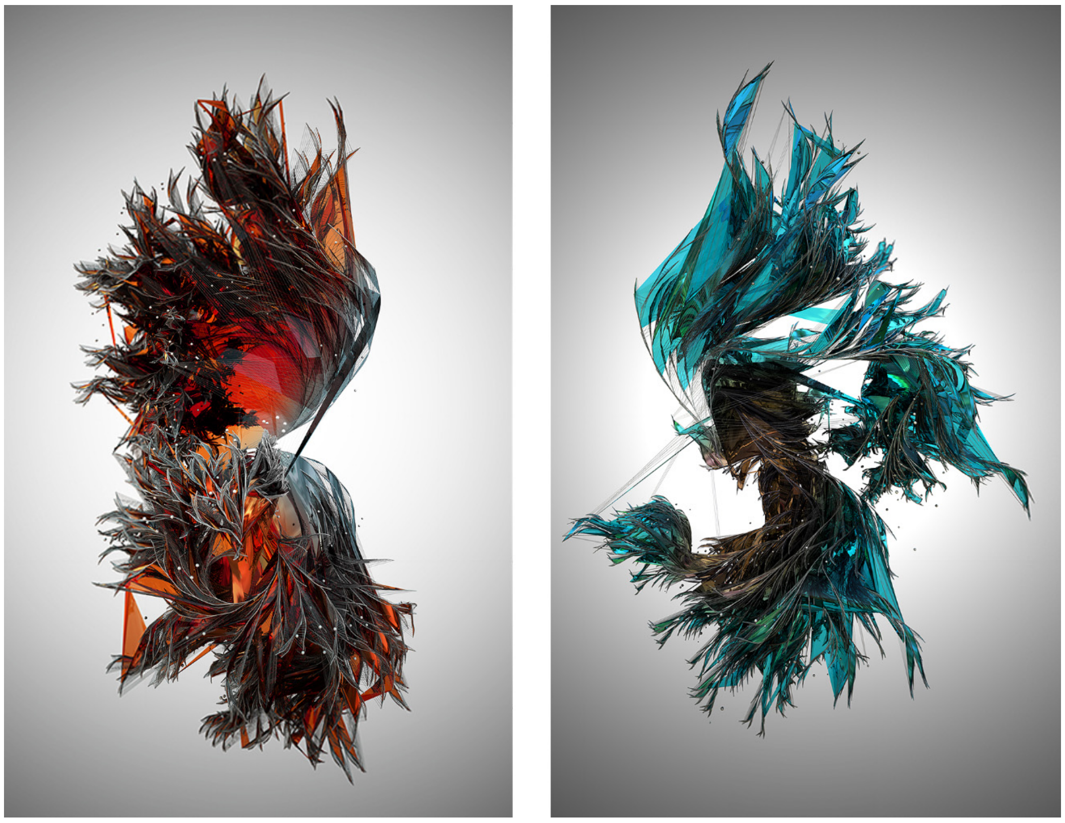


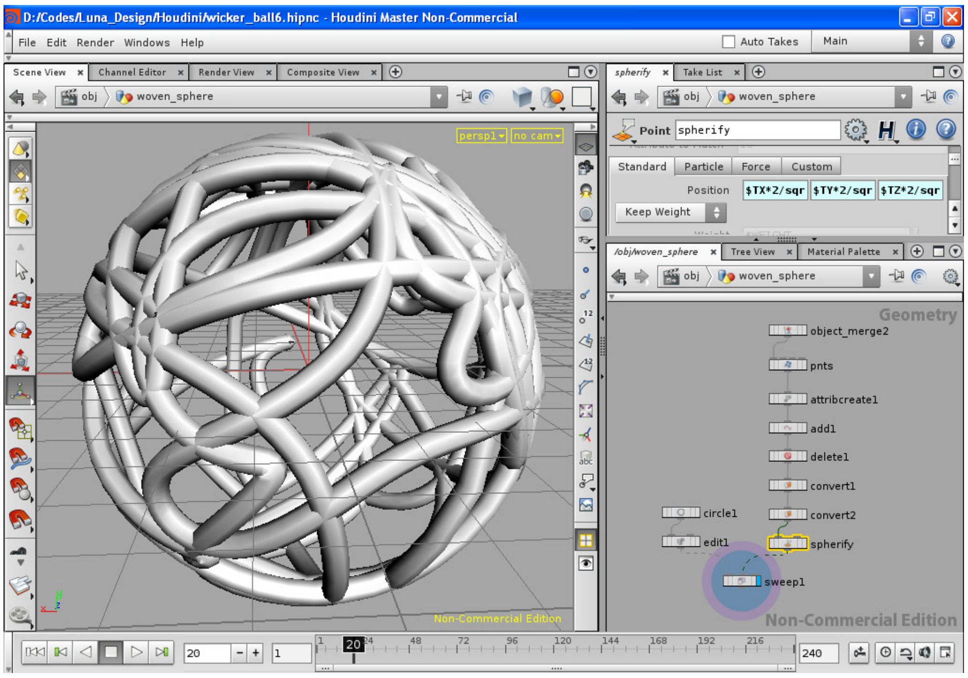
Figure 2.7: *Gestrüpp* (2010) demonstrates complex, organic modeling created using Houdini 10, by Side Effects Software. Image copyright depotVisuals GbR (c) 2010, http://art.depotvisuals.de/

of its unique support for procedural modeling (rather than scene-based modeling), and has been used in a wide number of feature films.

Houdini is also a visual programming language with an extensive set of procedural tools. As a production level tool, it has its own benefits and drawbacks. Primary among its benefits is the power and flexibility in modeling that can be achieved once the user overcomes its learning curve. As with other production tools, a drawback is that this power comes at the cost of a complex interface and significant time needed to learn the language. Houdini is capable of very specific operations, and of performing these on detailed geometric models which may be either generated or captured from real physical models.

Houdini uses graphs to express both hierarchical relationships and functional flow. This is another example of the complexity of a problem informing interface design, as Houdini is intended to model real world objects in sufficient detail for film production. Unlike the other platforms studied, Houdini is the only system here which features complex systems such as characters, fluids, fire, and smoke. Thus, Houdini may be considered the counterpoint to the low-level information aesthetic, and shape-based designs of Processing or Max/MSP. While many other effects are possible, typically only advanced users are capable of exploring the full expressiveness of the tool [Carlson, 2010].

Where Houdini excels is in procedural modeling, which it supports through a visual data flow interface. This highly developed interface allows artists to create complex models that change in both structure and behavior over time. Nested graphs (modules) let users express objects that can be affected by other objects. The cost of this flexibility, however, is that peculiarities of the system can take a great deal of time to master. Some of the specific issues in using Houdini are explored in further detail in Chapters 3 and 4. One finding is that achieving complex behavior often requires that the user write expressions, so that despite its visual interface it still requires mathematical knowledge to be used effectively. More importantly, this work flow is orthogonal to other goals of media artists, such as live performance. The detail-oriented nature of modeling in


D:/Codes/Luna_Design/Houdini/wicker_ball6.hipnc - Houdini Master Non-Commercial

Point spherify

Position $TX*2/sqr $TY*2/sqr $TZ*2/sqr

Keep Weight

Figure 2.8: Interface to Houdini 10, showing a reference model used later in this thesis (see Chapter 4). Software by Side Effects Software, Inc. (c) 2010

Houdini, suitable for commercial film, could not be easily modified to enable dynamic changes during a performance. In general, Houdini is a highly successful, powerful application for developing complex special effects in an offline environment.

2.4 Tools Summary

Tools available to visual media artists range from low-level signal processing to high-level procedural modeling of complex objects. The tools explored here, Processing, Max/MSP, VVVV, Xfrog, Groboto and Houdini are summarized in Figure 2.9. This table gives an overview of their history and design, a consideration of the output options available in each, and a look at the types of objects that they can express.

A. *Overview*	Processing	Max/MSP Jitter	vvvv	Groboto	Xfrog	Houdini
First Release	2002	1989 (2003 jit)	2002	1996	1996	1996
Authors	Benjamin Fry & Casey Reas	Miller Pluckette	Oschatz & Wolf	Darrel Anderson	Deussen & Lintermann	Davidson & Hermanovic
History	Info vis.	DSP, audio	Meso studio	Braid Media	ZKM	Side Effects
Real-time graphics	Java	OpenGL	DirectX	offline	offline	offline
Cost	free	$60 / year $500 full	free	$80	$200 min $1000 full	$99 student $325 univ. $9000 full
B. *Object Capabilities*						
Information design	▬	▬	▬			
Image compositing	▬ [1]	▬	▬			▬
Static 3D models		▬	▬	▬		▬
Rule-based models				▬		▬
Functional modeling					▬	▬
High level / Figures						▬
C. *Output Capabilities*						
Hardware in/out	▬	▬	▬			
Real-time output	▬ 2D only	▬ OpenGL	▬ DirectX			
Graphics shaders		▬ [2]	▬ [2]			▬
Multiple screens			▬			
Audio synthesis		▬				
High-res printing				▬	▬	▬
3rd party Rendering				▬	▬	▬

[1] Possible to write, but performance may be slower than hardware-based image compositing.
[2] Must be coded directinly in HLSL or Cg. Shader libraries not included in release.

Figure 2.9: Survey of tools for visual media artists. Major categories are a) Overview and history, b) Object representation workflows, and c) Output modalities.

It is important to note that emphasis is placed on the ease with which the tool supports a given modality, but this does not imply a tool cannot achieve another feature listed, only that it would be relatively difficult or time consuming for the artist to do so. For example, it is possible to build a three-dimensional procedural modeling system on top of Processing, but this would be a long term development problem in itself, and Processing is not necessarily the best environment to explore this. The table thus reflects the current, deep constraints, of the systems shown.

An interesting aspect of these results is the difference between low-level and high-level modeling. In terms of object support, this expresses itself as a difference in ability to support information aesthetics versus complex object models like characters. This may be due to a divergence of practice between media artists and commercial artists, with the later specifically targeting offline computation of complex systems and real world objects for use in film. Yet there is no inherent reason why tools could not be created which support both. Rather, this is a consequence of the different paths these communities have taken in their aesthetic goals.

Another key distinction, related to the previous one, is a difference in output modalities. Processing, Max/MSP, and VVVV all offer real-time, full screen, interactive output for live performance. Xfrog, Groboto, and Houdini don't allow this, but they do offer high resolution printing, and offline third-party rendering for photo realistic, anti-aliased (high quality) image generation. As can be see in modern video games, however, there is an increasing shift toward high quality rendering with real-time interaction provided by modern graphics cards. However, this technology is generally not yet available to media artists in a way which is not restricted to the specific objects of gaming, such as characters and terrains.

The only system which comes with multiple display support is VVVV. This is unfortunate, considering that this is a common format for architectural media installations. The present solution, for many artists, is to run several instances of the same application and synchronize their output using message passing. This allows the artwork to exist on multiple displays but makes poor use of system resources and can require significant overhead in development time for the artist or engineer. Multiple screen rendering is an ongoing field of research, while only a few existing tools take advantage of this format.

Even among the tools specifically designed for media artists, Processing, Max/MSP and VVVV, there are no similarities in terms of programming language or output graphics system. The first uses the Java-language with Java graphics output while the others use visual data flow languages with output in either OpenGL or DirectX. Since learning a programming language is a large time investment, this means that the upcoming artist is required to pick a tool which may dictate the next several years of project design. Unless the artist is willing to invest in learning multiple tools, the implication is that media artists will gather around particular languages. These communities are not distinguished by creative vision (movements centered around ideas), but by artificial communities based only on underlying language, and thus impose an unnecessary restriction on cross-communication. More importantly, as examples show, tool selection guides the creative ideas of artists into particular, constrained paths.

In general, there is currently no one tool which supports all of the primary work flows desirable to the media artist. The above list represents a range of tools currently available, yet none of these covers all of the dimensions of interest to artists. Of course, it is questionable whether a single tool could be designed to expressing all of these dimensions of creativity together, yet the current situation is equally challenging as existing tools do not necessarily provide the right set of features needed to explore any idea. An integrated tool would ideally combine many different styles of expression.

These results may explain why many media artists still choose to learn text-based low-level languages such as C/C++, Java, or Flash. First generation media artists learned more fundamental languages to retain the ability to explore whatever concepts were desired. Yet, even in examples such as Cohen's AARON [Cohen, 1979], which is not a system designed for animation, we can see that learning a low-level language introduces inherent constraints in the types of output it can generate. In many cases, the artist accepts the inherent constraint as a creative constraint and uses it to guide the work forward. Learning a low-level language can thus provide the basic theory needed to cross different domains, while higher level tools are need to explore other ideas or to extend into other output modalities without spending years in implementing basic structures oneself.

One clear conclusion is that there is an artificial divide between tools which directly impacts the goals of media artists. Visual artists with a sculptural background, for example, will find that there is no tool yet which offers procedural modeling of complex structural forms with real-time, high quality output for live performance. Support for complex objects such as characters, fluids and terrain are currently restricted to high-end modeling packages, and while not all artists will want to use these, we can imagine that many may wish to. Thus, the current mode of creative expression for media artists lags behind the more well funded film and game industry by several years in terms of complex geometric forms. However, to a greater extent than industry, media artists have created tools to explore real-time rendering, multiple displays, hardware interaction, and live performance. There is no theoretical reason, from a software perspective, that this divide need exist.

Chapter 3

LUNA: A Puzzle-Based Metaphor for Procedural Modeling

3.1 Introduction

Creative tools for digital artists have evolved considerably over the years. Since early systems such as Sutherland's Sketchpad [Sutherland, 1988] presented the first opportunity to directly interact with computers, users have been able to paint textures directly on surfaces [Blinn and Newell, 1976] [DeBry et al., 2002], and to interactively sculpt three-dimensional objects themselves [Lawrence, 2004]. These kinds of direct interactions, similar to physical artistic practices, are shifting more recently toward interaction in augment reality [Bandyopadhyay et al., 2001] [Ryokai et al., 2004] [Jacucci et al., 2005]. Yet tools for conceptual artists, who may view the art object as a dynamic system or model, have evolved more slowly.

While digital interfaces for the plastic arts are now common, such as painting with Photoshop or sculpting with Zbrush, visual interfaces for conceptual artists in the form of visual dataflow languages are still relatively new and still evolving. Early prototypes

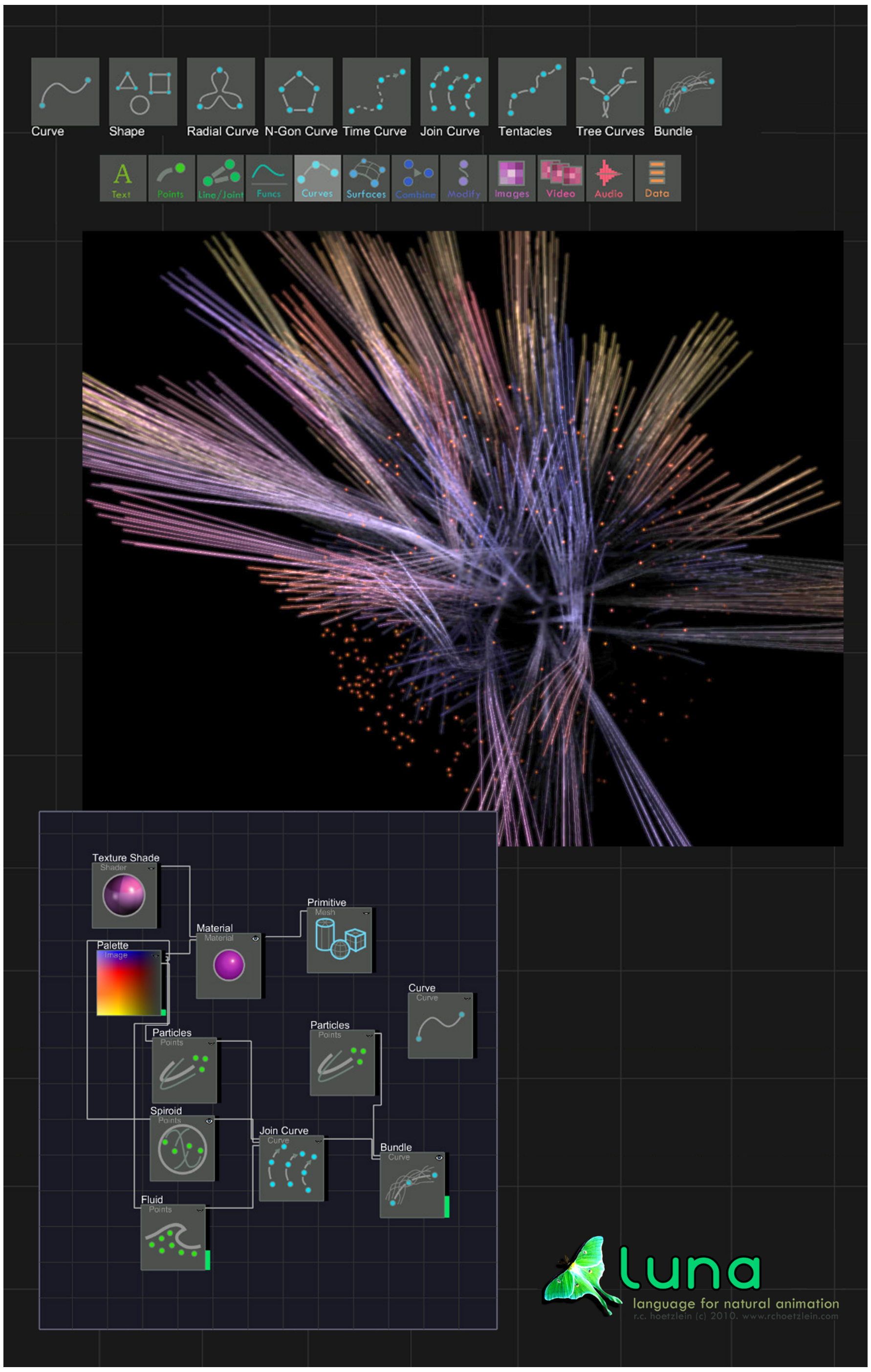


Figure 3.1: Full screen vertical layout of the LUNA graphical user interface.

such as ConMan [Haeberli, 1988] introduced the notion of dataflow interaction for graphical objects, while IRIS (SGI) Explorer [Foulser, 1995], the Application Visualization System (AVS) [Upson, 1989], and IBM's Visualization Data Explorer [IBM, 1999] provided a wide range of tools for processing scientific data. Educational systems such as Alice and Scratch employ a visual data flow metaphor, but these are designed largely to teach programming concepts rather than to simplify artistic development [Conway and Pausch, 1997] [Resnick et al., 2009]. Commercial systems such as Houdini enable content creation for artists in the entertainment industry [Bannink, 2009], using procedural methods in an offline setting to develop physical simulations and special effects. However, there is relatively little current academic research on novel designs for visual data flow languages in comparison to augmented interfaces for physical manipulation.

For media artists working with live performance, visual data flow languages such as Max/MSP/Jitter, 'vvvv', and Soundium offer an interactive node-based interface. These systems have been employed in major international live exhibition artworks. Max/MSP/Jitter was founded on digital signal processing, with objects that can process audio signals in real-time, and can transform these signals into graphic primitives [Jones and Nevile, 2005]. Soundium is a visual language that allows for interactive composition of visual elements, such as shapes, images and video, during a live performance [Müller et al., 2006] while 'vvvv' provides for graphical interaction with support for multiple display rendering and geometric primitives such as mesh objects [Meso, 1998].

While each of these systems have different affordances none of them employs a procedural modeling paradigm to allow for complex geometries. Although systems for media artists have evolved to support live performance they have remained relatively low-level in comparison to the model complexity offered by offline procedural tools.

Studies examining how conceptual artists interact, or would like to interact, with procedural dataflow systems are not as common as studies of painting or drawing interfaces. In a cognitive study of visual dataflow programming languages, Green defines and explores several aspects of their interaction: 1) *commitment*, when the language requires early decisions, 2) *progressive evaluation*, the ability to see intermediate results, 3) *expressiveness*, how easy it is to say what you want, 4) *viscosity*, how much the interface resists change, and 5) *visibility*, how easily you can see what you're creating [Green and Petre, 1996]. While Green admits it may be difficult to establish quantitative measures of these, his criteria establish guidelines for evaluating visual languages.

To understand interface issues that are directly relevant to media artists, several dimensions of creativity of interest to this group were examined prior and during software development. These dimensions, 1) programming and language, 2) modality and media, 3) live performance and computation, 4) motion, dynamics and autonomy, 5) structure and surface, and 6) image and idea, were found to represent common themes that media artists working with visual forms seek to explore (see Chapters 5 and 6 for a detailed discussion of these dimensions). *Programming and Language* refers to the desire of some artist to engage in programming, while others seek to explore ideas visually, without programmatic knowledge. *Modality* is the ability to engage with different types of media, such as images, surface, video, and audio. *Live Performance* refers to the desire of some artists to have immediate, real time feedback, with high quality output in full screen for installation and performance. *Motion, Dynamics and Autonomy* refers to certain artists' interests in exploring motion and behavior, while *Structure and Surface* reflects those interested in expressing geometric forms and their material appearance. Finally, *Image and Idea* refers to the desire of some artists to work with the image as a conceptual object with semantic content.

These dimensions establish the basis on which interface decisions were made to create LUNA, a novel visual dataflow language for procedural modeling. LUNA is inspired by the board game Scrabble, where the ability to express a wide range of words comes from the rearrangement of only a few tiles. The dimensions of creative exploration for media artists are used to inform the design decisions of the language, resulting in a minimalist approach which supports real time interaction for complex procedural models. Result consist of a number of cross-disciplinary projects and comparisons of user interactions performed in Luna and Houdini.

3.2 Interface Design

Design of the visual dataflow language for LUNA is motivated by a combinatorial arrangement of minimally designed tiles. While similar systems use graphs to allow for such combinations, they are often superimposed with a number of other interface elements and widgets which must be read by the user in order to be understood. This need to "read" the interface reduces the rate at which different conceptual designs may be explored by the artist. LUNA is developed using a minimalist approach in which the dynamic rearrangement of tiles (objects) provides a top level interaction which is then continually refined through more specific interactions. Design decisions are motivated toward eliminating the act of reading in favor of visual metaphor.

3.2.1 Design Decisions

Support for creative exploration influenced the primary design decisions in the development of LUNA. Among these was the desire to develop a tool that builds on the dimensions of creativity established earlier by informing the goals of the language. The dimensions explored, and their impacts on language interface design, are as follows:

1. Programming - Mathematical knowledge should be optional to the artist. The primary mode of interaction should be conceptual and exploratory, suggesting a visual dataflow language with a minimalist design aesthetic.

2. Modality - The language itself should offer a range of different media types. These are supported through a tool set that includes points, curves, surfaces, images, and materials. The interface should make the media types, which are the primary objects being operated on, apparent to the user while working.

3. Live Performance - The interface should enable live performance by allowing for full screen, high quality output, with real time design changes. This aspect informs the

overall visual design of LUNA, resulting in an inverted window layout that floats the menus and interface elements as a layer in front of a full screen output canvas, rather than surrounding the canvas by interaction tools.

4. Dynamics and Behavior - The system should allow users to interact with dynamic objects and receive immediate feedback. This aspect informs LUNA through the use of property panels that resemble mixing boards, with large, intuitive sliders that modify on-screen behavior.

5. Image and Idea - The language should allow complex semantic transformations to be performed easily and without lengthy interactions. For example, setting up motion capture or a computer vision process (e.g. transforming and image into semantic labels), should be possible without a length set up process to define parameters. This criteria influenced the decision in LUNA to require that each node provide intuitive default behaviors as soon as it is first placed.

The expressive power of a procedural modeling language is influenced by both its interface and the underlying structures which support it. In the development of LUNA interface design decisions have had direct effect on the structure and design of the modeling language. This language structure, discussed in detail in Chapter 4, consists of geometric and media elements that flow through an abstract graph, similar to systems such as ConMan and Houdini. The primary contributions explored in this chapter are the graphical user interface features that directly enable the abstract goals described above.

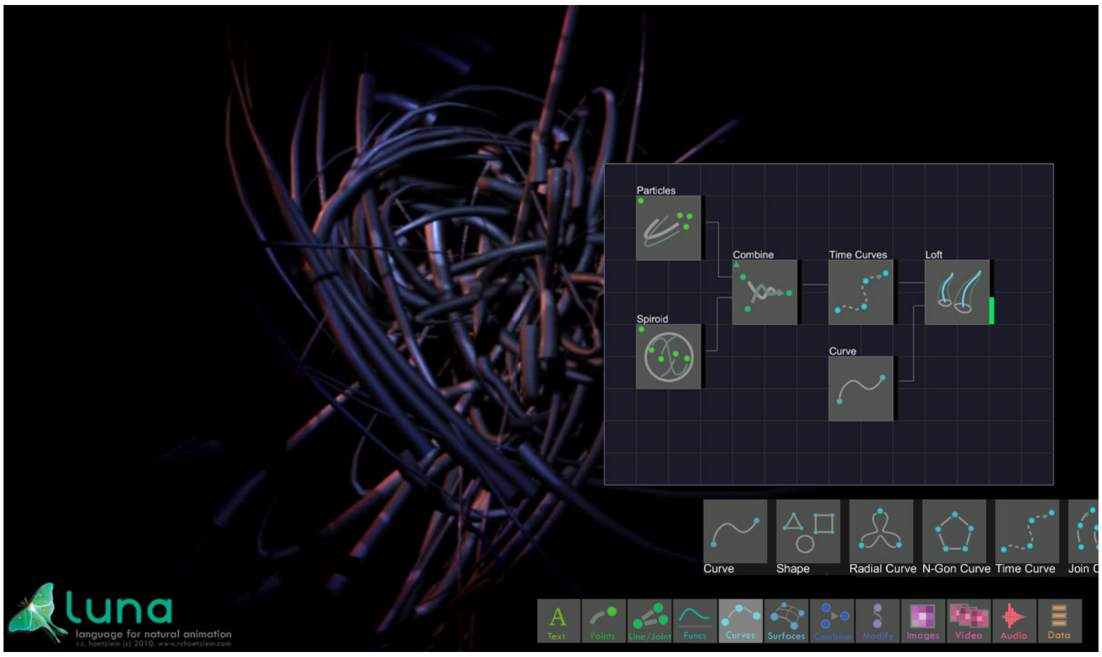


Figure 3.2: LUNA interface with full-screen background rendering and floating foreground elements. Interface elements may be hidden, or moved to a second display, to allow for live performances.

3.2.2 Workspace Layout

The general design of LUNA follows a workspace model, but one which has been inverted from the common layout (compare Figures 3.2 and 3.10). Typically, the work flow of a procedural modeling tool employs a central view surrounded by menus and interaction panels. In the interest of live performance, and in order to emphasize the result of aesthetic explorations, this layout is inverted by using the entire display area as the space for the output, while floating the interaction panels above the output. Rather than consider the viewport as an intermediate result, as is common in other commercial packages (e.g. Maya, Houdini), it is designed as a primary output window with high quality rendering using deferred shading techniques typically found in gaming. Of course, the layout is flexible and all windows, in including the deferred shading output window, are resizable and movable.

3.2.3 Object Icons

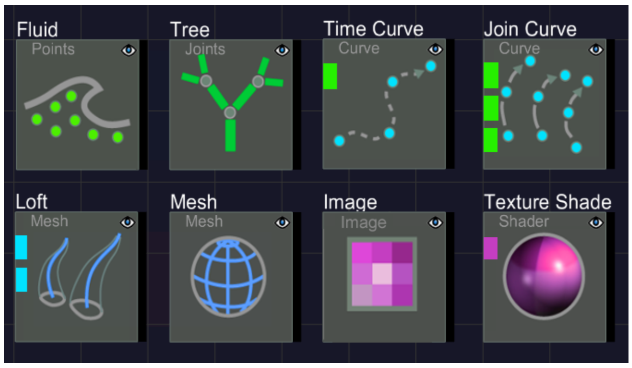


Figure 3.3: Symbolic icons in LUNA with dominant color used to indicate the base output type. The symbol uses the majority of the icon space.

In many visual graph languages, a tool bar with an iconic depiction of the object is used to quickly identify objects of interest. However, once the object is placed on the graph, the icon is removed and replaced by a textual description, object or class name. The LUNA interface retains the symbolic icon, and enlarges it, with minimal text above the graph object to describe its type, Figure 3.3. This allows the user to see, at a glance, the visual meaning of each object in the graph. Careful design of the icons gives a strong impression of the actual output the graph will produce.

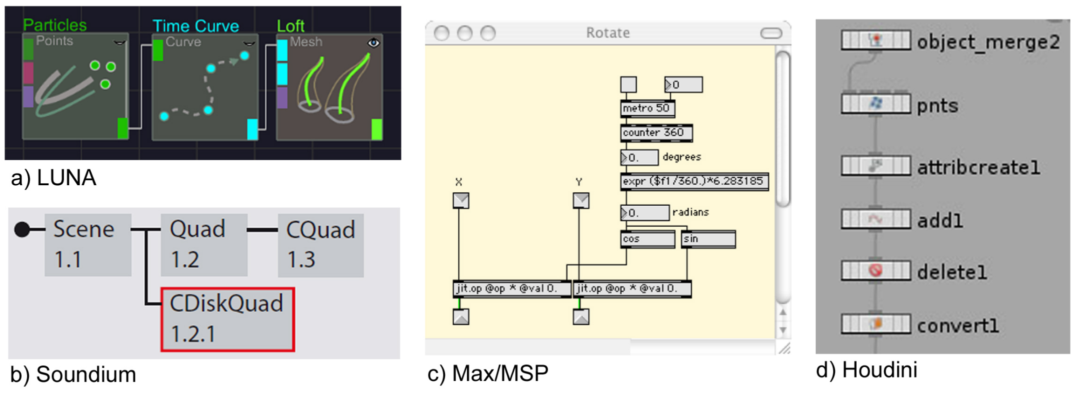


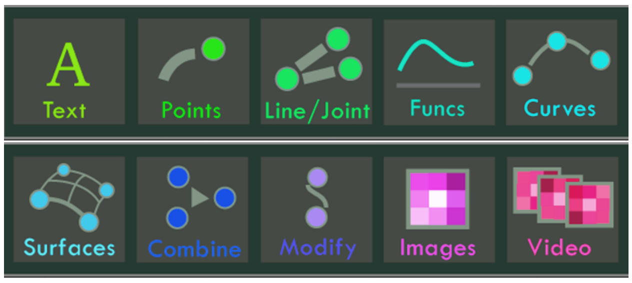
Figure 3.4: Colored tabs in LUNA (a) indicate media types as they flow through the graph, with green for points and blue for curves, in comparisons with other data flow interfaces from b) Soundium, c) Max/MSP, and d) Houdini.

3.2.4 Colored Inputs

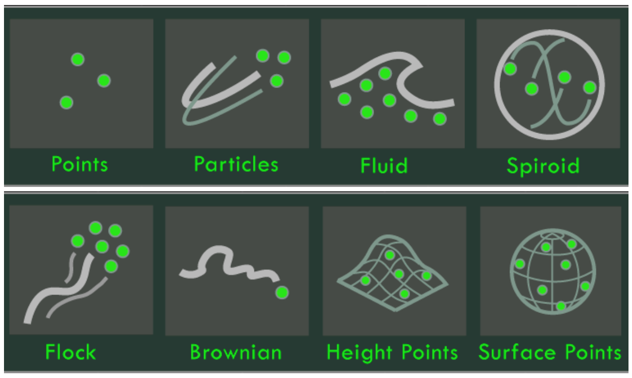
Procedural graphs often use nodes with input and output tabs to represent the arguments to a function. When constructing graphs, it can be difficult to determine which nodes are compatible with which inputs and outputs, often necessitating a help reference in order to construct syntactically valid graphs. In order to alleviate this problem, LUNA uses colored tabs to identify compatible input nodes, shown in Figure 3.4. This allows the artist to quickly see what *modality* they are working with throughout the design process. Objects without input tabs are generator objects, while objects colored grey are modifiers which accept several media types as input.

3.2.5 Toolbar Design

The design of toolbars for large numbers of objects in procedural systems is an ongoing challenge. In many systems, objects are categorized according to *workflow*



(a) Primary toolbar with categories for geometry output types



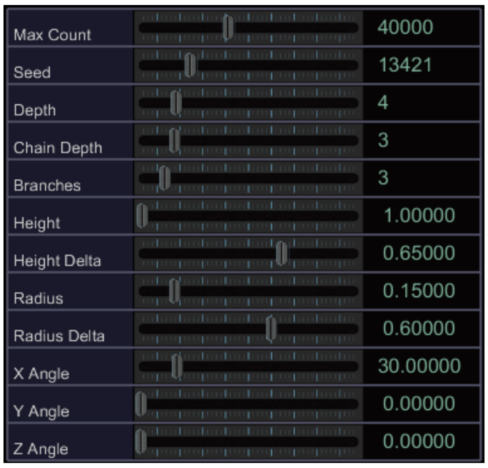
(b) Secondary toolbar showing behavioral models, or functional variants, of a given primary type. Each object also has parameter inputs which are specific to it.

Figure 3.5: Two-level tool bar design intended to reflect the structure and function of procedural objects. In this example, the secondary tool bar shows all behavioral objects whose output type is a Point set (all are colored green).

categories of modeling, animation, characters, dynamics and rendering.[1] In LUNA, a two-level system is introduced. The primary tool bar represents the *structural* objects of discrete geometry. These include: points, lines, curves, surfaces, images and video. Selection of a geometry type at level one exposes a set of objects in the secondary tool bar. The secondary bar shows all the different *behavioral* choices available for a given primary type. These objects are interchangeable, such that any behavior which outputs

[1]These workflows represent stages of production common in the motion picture and visual effects communities

points can be input to any node accepting points. For example, Particles, Fluids, Spiroids, Brownian, and Flocking all generate unique behaviors for a time-evolving point set. Any of these may be input into other objects which accept points as input (e.g. TimeCurves). This ability to quickly interchange behaviors is a major advantage in creative explorations.



Max Count	40000
Seed	13421
Depth	4
Chain Depth	3
Branches	3
Height	1.00000
Height Delta	0.65000
Radius	0.15000
Radius Delta	0.60000
X Angle	30.00000
Y Angle	0.00000
Z Angle	0.00000

Figure 3.6: Property panel in LUNA with parameter sliders shown for the Tree object.

3.2.6 Property Panel

To further enable live performance and to provide precise control over node parameters a property panel is introduced with an aesthetic based on sound mixing. While the interface appearance is reconfigurable, this design encourages large sliders with clear labels over numeric entry. The property panel is optionally accessed by clicking on a graph node, a top-down approach that places emphasis on the graph, where high level decisions are made first rather than on more exacting parameter changes which can be made later. While only sliders are currently available, in the future the panel may be extended to support other controllers.

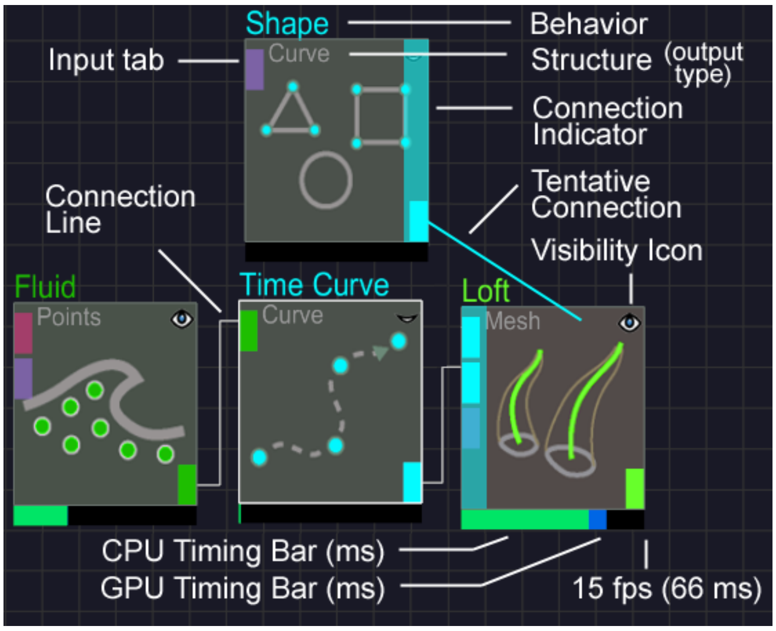


Figure 3.7: Dataflow interface elements show the base behavior of an object, its output structure, input tabs and connected lines. When connecting two objects, *connection indicators* show the media type by color. Blue 'bumpers' in this case indicate the Shape curve is being connected to the Loft object. A visibility icon allows the user to render the output of any object in the graph. Performance bars show CPU (green) and GPU (blue) timings for each node. In this example, a smooth particle hydrodynamic (particle-based) fluid simulation runs on the CPU with 10k particles, while 400 swept surfaces are generated from this data (1000 polys per object), resulting in 400k rendered polys/frame at a frame time of roughly 50 ms (on an Intel Core i7 with a GeForce GTX 420).

3.2.7 Smart Connections

Typically, in order to connect two nodes, it is necessary to know the type and meaning of arguments to both. In many cases, however, there is only one combination of input types possible. For example, a generative terrain object may only require a two-dimensional image representing height. We distinguish between primary, required, inputs and secondary, optional ones. When connecting objects, LUNA detects the semantics of incoming objects and, wherever possible, directly connects these objects without the user having to specify a particular input and output tab (Figure 3.7). With a single click-drag motion, it is possible to quickly connect many objects in this way. In the future, for more refined control, the user might hover over an input tab to specify a particular input.

3.2.8 Required Defaults

While implementing LUNA, it was realized that a drawback of some systems is that inputs often must be exactly defined in order to produce any output. One design goal, reflected in the design dimension of image/idea, is that complex operations should not be prohibitively difficult to specify to get an initial result. Therefore, every node in LUNA is required to have default behavior that produces output as soon as all required inputs are connected. This often means that a generative node must be capable of resampling or reducing the input size to a meaningful level to avoid stalling the system. Creating any node, and connecting its visible input tabs, produces an immediate result.

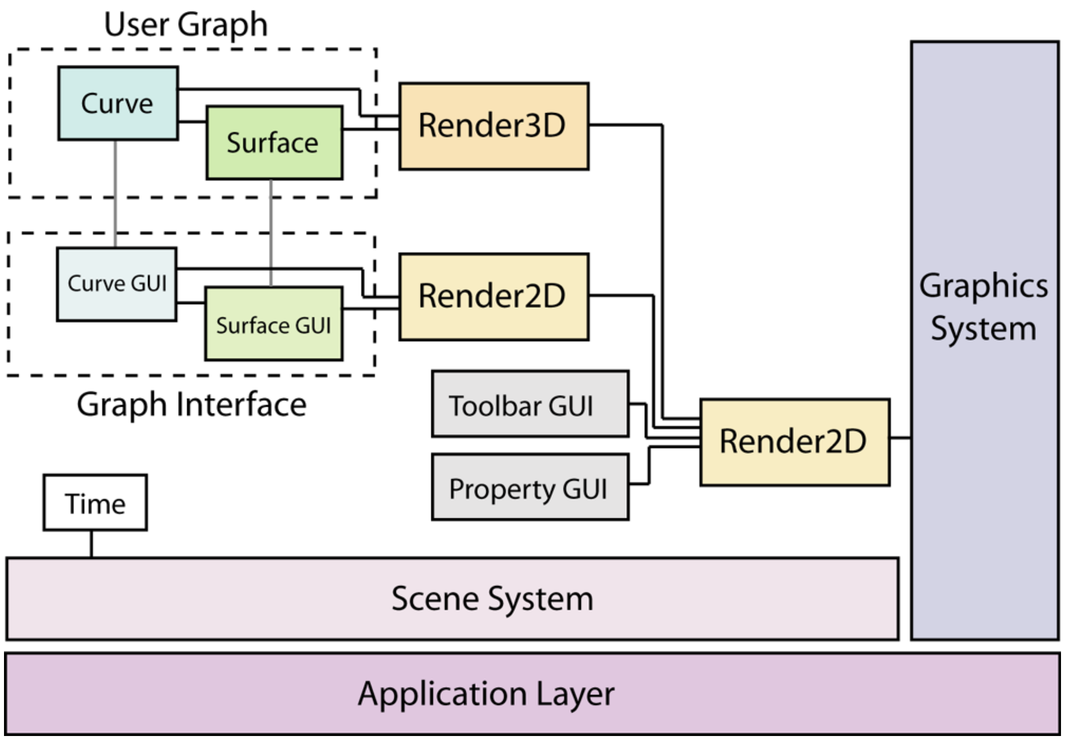


Figure 3.8

3.3 Graphical Language Implementation

The structure of LUNA consists of a procedural modeling language and a graphical user interface combined into a single graph architecture. The scene graph is a directed graph which contains objects for behavior, geometry, and interface. Unlike traditional model-view-controller designs, in which the scene DAG is kept separate from the interface graph, LUNA combines these objects into a single graph and uses rich connection semantics to keep them organized.[2] Thus, multiple sub-graphs may overlap in the same graph. The *render graph* is a set of input/output connections that describe how to ren-

[2]An interface graph is different from the procedural and scene graphs, which are distinguished based on semantic context as described in Chapter 4

der both GUI elements and three-dimensional objects, see Figure 3.8. This allows the graphical interface and procedural models to be displayed by the rendering sub-system in a uniform way. Of course, these objects must be handled differently during rendering, so any object can indicate the style of evaluation it requires: 1) *self-draw*, used by GUIs to draw themselves and their contents, 2) *proxy*, used by geometry to request that the renderer build vertex buffers on the GPU, and 3) *resources*, used by images and materials to request persistent data available to multiple objects. The method of evaluation differs for procedural models and graphical interface components. In general, the generation of complex procedural models are discussed in detail in Chapter 4, while this chapter focuses on the evaluation on GUI elements, i.e. the visual and interactive combination of two and three-dimensional components in the system.

The overall architecture of LUNA consists of multiple sub-systems, including graphics, video, networking, and input. These sub-systems are responsible for hardware or device-level interactions with the scene graph, and each may communicate with the graph in different ways. The graphics system, for example, renders any desired object and keeps track of graphics state and GUI buffers for performance. The renderer traverses the graph, conceptually, from right-to-left starting from the top level 2D desktop GUI node, Render2D, which covers the workspace. This node may have multiple Render2D and Render3D nodes connected to it as well, allowing for nested views and three-dimensional windows. The input sub-system handles user interface events, and traverses the same scene graph from root to leaf, but using a different set of functions for event handling.

Time, i.e. motion, is handled by the application by inserting a global Time node into the procedural graph. The time node is unique in that it traverses the graph from the opposite direction, triggering any behavioral nodes which accept time as an input. This causes notifications to travel *up* the graph, informing any objects whose geometries or interfaces must be updated on the next render frame. With this overall design, it is possible to combine multiple semantics into a single graph architecture which simplifies the issues related to maintaining separate graphs for model, view and control.[3]

The idea of rendering in two and three-dimensions is implemented using nodes also found in the scene graph. While a single graphics sub-system handles actual rendering, the presence of rendering nodes allows the graph to invoke different coordinate spaces, views, and windows as the graph is traversed during rendering. This allows both the graphical interface and the user output to be generated by the same rendering evaluation model. In Figure 3.8, for example, the Render3D node prepares the graphics system to generate geometry for the Curves and Surfaces connected to it, while the Render2D node prepares a local two-dimensional canvas on which the GUI boxes representing these objects are drawn. If the user interacts with the three-dimensional curve itself (moving a vertex), this event passes down the Render3D portion of the graph from the

[3]While model, view and control are still present, the objects which represent these different facets of the application are combined in the same graph through the diverse functionality present in each object.

root, while if the user moves the two-dimensional box representing the curve, this event passes down the Render2D portion of the graph.

3.4 Interaction Study and Evaluation

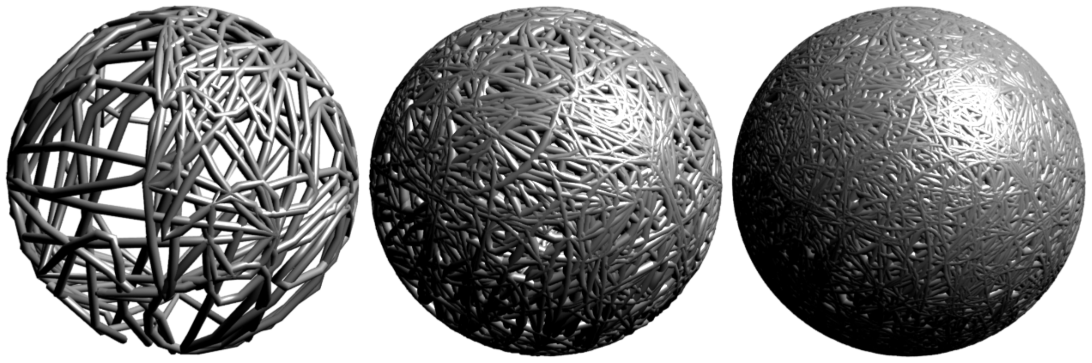
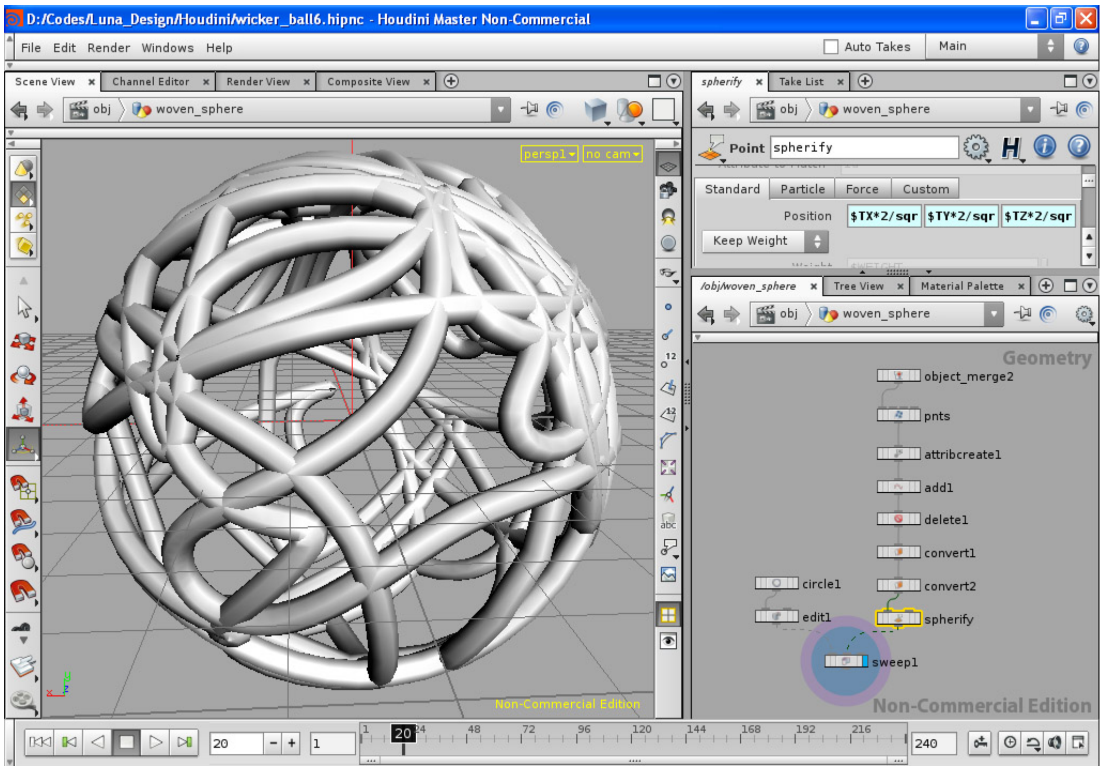


Figure 3.9: Reference object for interface testing, a woven sphere, is described in detail in Appendix A.

To demonstrate the usability of LUNA in a practical context, a series of interaction studies were performed by the author. While there are few procedural dataflow languages with similar capabilities the most similar system is Houdini, chosen as a comparison reference system due to its support for generic procedural modeling. As there are no common reference models for interactions with procedural dataflow systems, a novel object is introduced here. The object used for testing is a woven sphere, shown in Figure 3.9, which consists of a simple system of moving particles sampled to generate Bézier curves, normalized to a sphere, and lofted to create a set of tubes lying on the sphere. This model is described in further detail in Appendix A. The woven sphere is a suitable object for interface testing because it represents several specific steps which are unique to procedural systems. It includes an animated system, intermediate objects of different types (points, curves and meshes), and steps which must be introduced at correct stages in the model graph to produce the correct output. In addition, this object is uniquely procedural, and cannot be constructed using traditional modeling techniques.



Point: spherify
Position: $TX*2/sqr $TY*2/sqr $TZ*2/sqr
Keep Weight

Figure 3.10: Reference model created in the Houdini interface. The model uses a Particle SOP for initial point locations, a Point SOP /w a normalize expression to generate curves and map these curves to the surface of a sphere, a Circle SOP (set to 'polygons') to define the loft cross-section, and a Sweep SOP to build swept surfaces from these curves. The Copy Stamping method is used to generate different random instances of curves, with point inputs scaled to (0,0,0) so that the curves are not translated in space.

To perform interface testing, the author constructed the reference model in both Luna and Houdini.[4] A log was kept of the challenges and problems encountered during model construction, as well as a record of the time at each phase, reported in Appendix B. Figure 3.10 shows the reference model being constructed in Houdini. In general, the model requires knowledge of points, curves, and surfaces as it is being constructed. However, these transformations in media type are not entirely clear in the Houdini graph, which may necessitate the user having to conceptualize the data format implicit in each node. Other interactions were also found to be difficult in Houdini. At certain stages, detailed in Appendix B, it was necessary to know a non-obvious feature of a node in order to correctly produce the output type needed for the next step. For example, to generate the loft surfaces required by the model (tubes), the circle object in Houdini must be changed from "primitive" output to a "polygon" output in order to generate swept surfaces from cross-sections, otherwise the output remains blank. This knowledge is generally found by referring to the reference documentation for particular objects, which further detracts from the work flow. Overall, four hours were required by the author to create the reference model in Houdini.

There are several ways to create this same model in LUNA. One may work from right-to-left, imagining the form of the final result and filling in pre-requisite nodes that are needed to activate it. Or, one may work from left-to-right, starting from the basic structure of the model and building it into a final form. This latter approach is taken in

[4]These interactions were the author's first experiences with Houdini, so no prior knowledge was assumed.

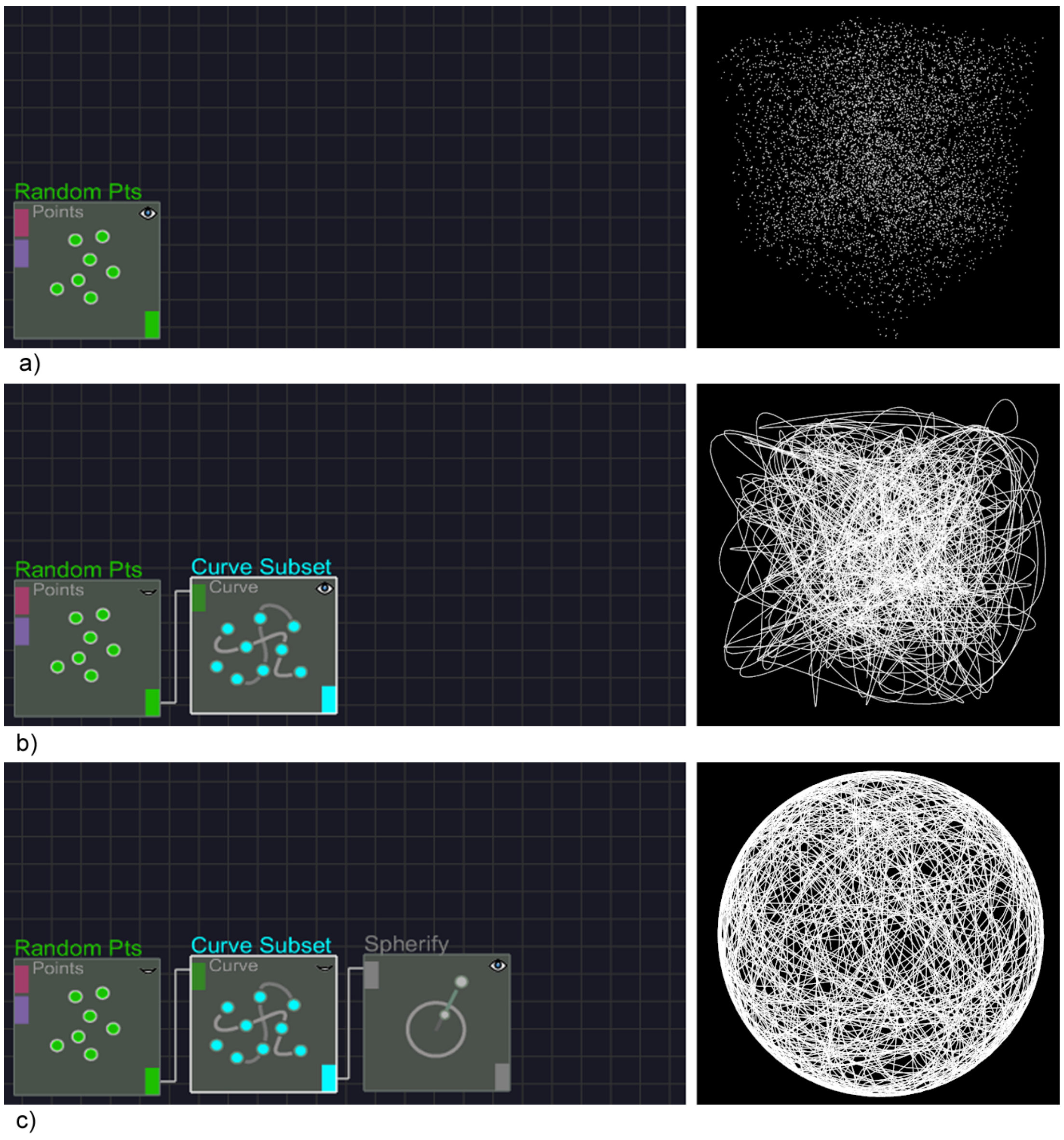


Figure 3.11: Steps in creating the reference model in LUNA include a) creating a set of Random Points as starting positions, b) producing curves by randomly sampling subsets of the input points using the Subset Curves object, and c) normalizing the curves to a sphere using a Spherify modifier.

the steps described below, the first of which is to select the Points type from the main toolbar and drop a Random Points node onto the canvas, producing the result shown in Figure 3.11a.

The second step in this model, detailed in Appendix A, involves selection of several random subsets of these points to be used as the CV control points for Bézier curves. This is accomplished by dropping a Curve Subset object, and connecting the Random Points into it, shown in Figure 3.11b. This produces a set of curves which randomly fills the space occupied by the points. Graph connections are made using a single click-drag motion from input to output. The third step is to map these curves onto a sphere. This is accomplish in LUNA using the Spherify modifier, Figure 3.11c, which transforms any object onto a sphere (by normalizing its points); in this case the curves are spherified. In general, modifiers in LUNA are able to operate on any object, and their output takes on the type of the input connected to them. Thus the output of Spherify in this case is another set of curves.

The final step in this example involves constructing swept surfaces from these curves. The Loft object in LUNA performs this function and takes two curves as required inputs. The first specifies the cross-sectional shape of the surface, in this case a circle Primitive object is connected at the top. The second input is the curve set which represents multiple paths along which this section will be swept. As the Spherify modifier outputs a set of curves, this is used to express the paths we wish to loft, producing the final results shown in Figure 3.12a. Using these nodes, the reference model can be created in this interface with literally ten clicks (5 to select objects, 5 to drop them), and four click-drag motions (connecting each node to the next).

Admittedly there are several advantages given to LUNA here. First, many of the nodes used here do not exist in Houdini. For example, the Spherify operation is not found in Houdini by default, and it was necessary (see Appendix B) to write an expression to perform the spherify operation in Houdini, which reduces performance. In addition, the author's familiarity with LUNA suggests that the choice of nodes, and their order of operation, may not be obvious to a new user of the language. The issue of language familiarity, however, is partly addressed by the ease with which one can discover different models in LUNA using very simple interactions. As the Spherify node operates on any geometry type, it can be connected at different stages in the graph. In building the reference model, for example, the user may have thought to spherify the points *before* generating curves. The result of this is shown in Figure 3.12b. Since the Subset Curve object uses the points as control vertices in Bézier curves, the resulting curves themselves may penetrate into or protrude away from the sphere surface, producing an incorrect model (relative to the reference goal). Using LUNA, the user can transition from this model to the correct one with only three click-drag motions. Thus a unique contribution of LUNA is the ease with which new models can be generated through its interface.

This basic interface, which favors combinatorial rearrangement over detailed parameter controls, allows one to rapidly explore the power of LUNA as a language for

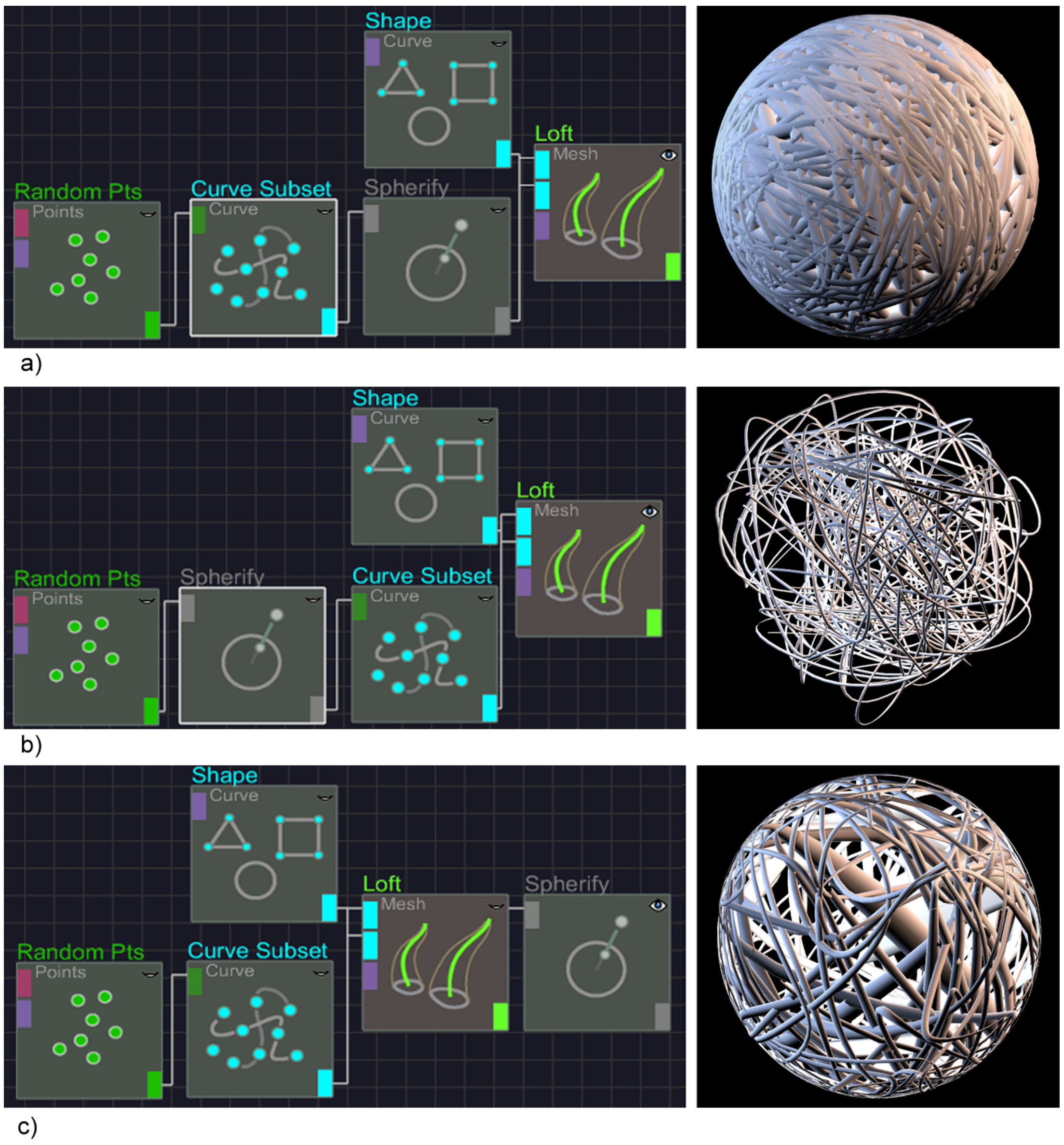


Figure 3.12: The final step in creating the reference model, a) connects the spherified curves to a Loft object to generate swept surfaces. Alternative models are easily explored by changing the order in which these high level actions are performed, such as b) where the spherify operation acts directly on points, and c) where spherify is applied to a mesh object.

procedural modeling. By connecting the spherify operation to the outcome of the loft node, one gets a different result altogether (Figure 3.12c). This causes the vertices of the tube meshes to be spherified, which distorts their cross-sectional geometry and volume in interesting ways. Modifiers may be connected to other modifiers, providing a limitless source of possible outcomes. While languages such as Houdini focus on the detailed control of each node, also possible in LUNA using the property panel, LUNA does not require this kind of detailed work flow in order create valid output. This makes LUNA particularly suited to its original goal of serving creative exploration by media artists.

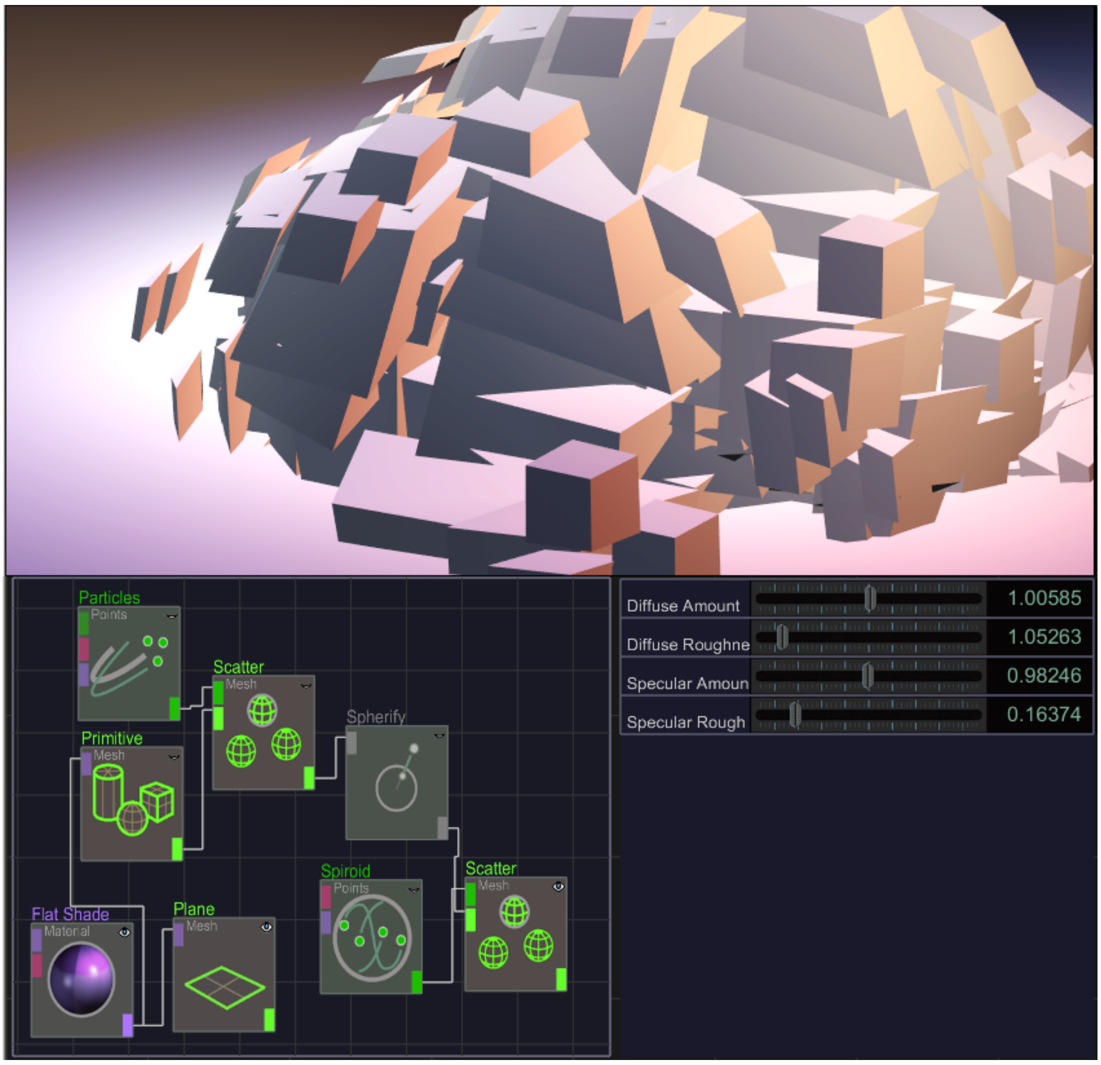
This brief demonstration was intended to show how one can quickly create novel objects in LUNA. However, a more thorough interface study could be designed to reveal more detailed results. To create a fair analysis, both Houdini and Luna would be provided with the same object set by implementing custom nodes in Houdini. Although it is difficult to find a task that represents overall creative exploration, one could ask users to create *any* object meeting certain criteria, such as incorporating points, curves and surfaces together. A detailed study might also reveal how these systems balance expressive power and flexibility. These are future areas for possible examination. In any case, there are few generic visual dataflow languages for procedural modeling.[5]

Prior to developing such user-based interaction studies, LUNA may be evaluated according to Green's criteria for visual dataflow languages defined earlier. Early de-

[5]Others include Xfrog and Groboto, but these are specialized systems. Artists' tools like Soundium are generative, but do not have a procedural aspect. See Chapter 2 for a comparison of systems.

cisions made in LUNA can be easily modified by reconnecting objects, so its level of *commitment* is lower than that of Houdini. *Progressive evaluation* is supported in several ways, by allowing the user to see intermediate results (using the 'eye' icon) and by giving direct feedback on parametric changes, features also available in Houdini, although LUNA's performance is better overall in this regard (see Chapter 4). The ability to convey what one wants, *expressiveness*, can be interpreted in two ways. First, the complexity of an expression, i.e. power, is potentially higher in Houdini as it is a more developed commercial application with support for more complex objects. However, expressiveness can also imply the ease with which one can describe an idea, and in that respect LUNA may provide a better experience as its interaction produces more immediate results. Regarding *viscosity* (resistance to change), LUNA was intentionally designed to make it easy to modify objects and ideas interactively. Additionally, the use of color to denote media type, the large iconic representations of tasks, and the overall layout of the system give it a level of *visibility* which makes it easier to see what one is making in comparison to Houdini. LUNA is thus presented as a modern, dynamic, interactive alternative to current commercial procedural modeling systems.

A more complex LUNA graph is shown in Figure 3.13. In this example, two particle systems and a cube primitive are used to generate arrangements using two Scatter nodes. A material node, Flat Shade, is used to change the visual appearance of both the overall object and the ground plane. The parameters to this material are shown in the Property panel to the right, with the results shown above.



Diffuse Amount 1.00585
Diffuse Roughne 1.05263
Specular Amoun 0.98246
Specular Rough 0.16374

Figure 3.13: Example of a complicated graph in LUNA, incorporating multiple surface objects, modifiers, and materials. Cube primitives are scattered at point locations, distorted, and then scattered again to create shifting, rectilinear forms.

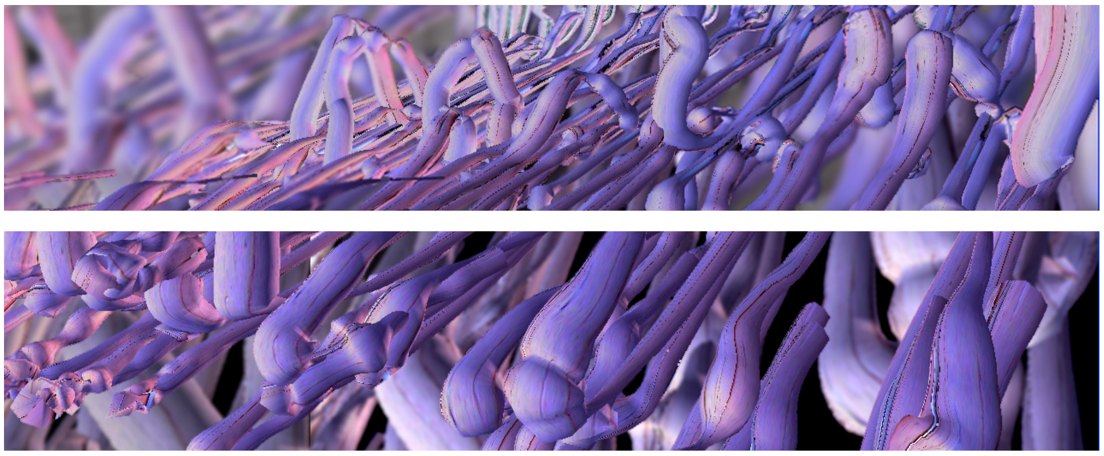


Figure 3.14: *The Bones of Maria.* Generative art. Exhibit online at The Cultor, Italy. 2010.

3.5 Project Results

3.5.1 *The Bones of Maria*, Organic Art

To explore the expressive range of LUNA, several creative, interdisciplinary projects were developed with the system. These include works with styles in digital and media arts. *The Bones of Maria*, shown in Figure 3.14, is a generative art project using a smoothed particle hydrodynamic (SPH) fluid system and time analysis to create organic, three-dimensional, textured forms. These were shown in an online exhibition at The Cultor, an arts and culture organization based in Torino, Italy [6]

[6]http://www.cultor.it/Pinacoteca2.html

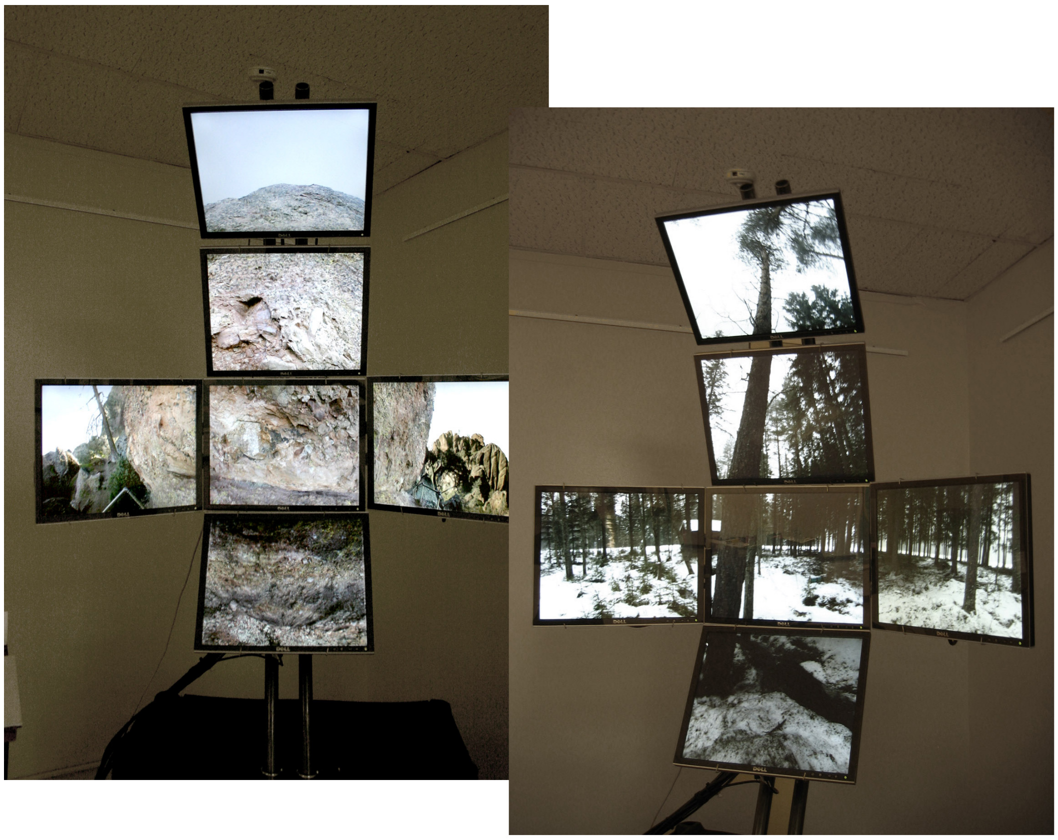


Figure 3.15: *Presence* by Dennis Adderton, Jeff Elings and R. Hoetzlein (2009). Interactive artwork showing 360 degree panoramic images of natural spaces, exhibited in the Davidson Library at the University of California Santa Barbara.

3.5.2 *Presence*, Interactive Art

Presence was an interactive, site-based installation exhibited at the University of California Santa Barbara's Davidson Library in 2009, Figure 3.15. A collaboration between R. Hoetzlein, Dennis Adderton, and Jeff Elings, *Presence* consists of high resolution, virtual 360 degree panoramic photographs displayed on six screens. A camera detects the motion of passing library patrons and rotates the panorama as they walk past the exhibit.

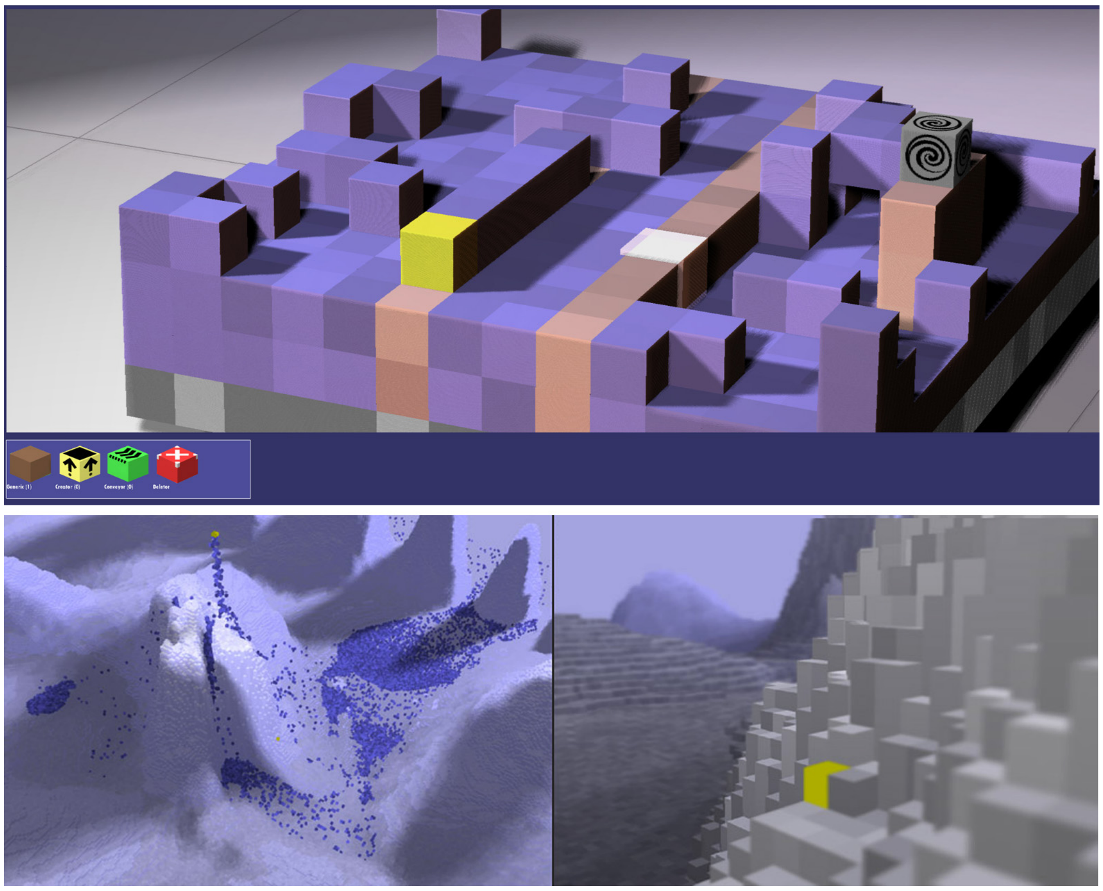


Figure 3.16: *Blocks*, a universe of cubes that function as bridges, terrain, fluids, and logic puzzles in an interactive game (2003-2010). Designed by Mark Zifchock and Rama Hoetzlein. Created using LUNA.

3.5.3 *Blocks*, Game Design

An experimental game project called *Blocks*, started in 2003, was created with Mark Zifchock using LUNA, with additional input by Abram Connelly and Marty White. *Blocks* is a universe of cubes where each has a unique function: some blocks act like water flows, moving at right angles into lower spaces. Others are used to build terrain, bridges, and barriers. Logic blocks introduce computational AND, OR and NOT gates expressed in cubes, while special blocks allow for teleportation and wireless signaling. *Blocks* was implemented using a custom node in LUNA for the block-world simulation, with the blocks universe consisting of millions of cubes rendered using textures and Cg shaders. A unique aspect of Blocks is its own graphical interface, which includes a custom tool bar for selecting block types. This interface was implemented using the same GUI graph architecture used for LUNA itself, and both GUI elements (the Blocks tool bar and LUNA's object tool bars) are rendered in the same graph together.

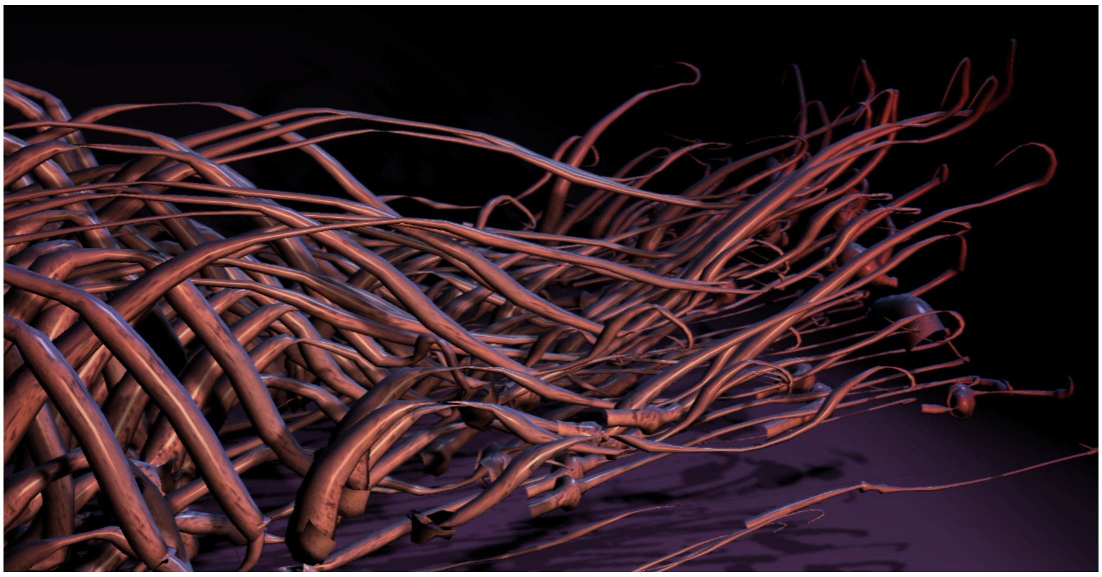


Figure 3.17: Loft surface with high quality rendering created in LUNA using shadows and depth-of-field

3.5.4 Procedural Modeling

Other experiments with procedural modeling in LUNA are shown in Figures 3.17 and Figure 3.18. In many cases, these images were constructed, generated and rendered in a matter of seconds. The dynamic nature of the interface makes it very easy to quickly replace an object with another node of a similar type. Thus, an artist is able to rapidly experiment with different dynamic behaviors as this sequence shows.

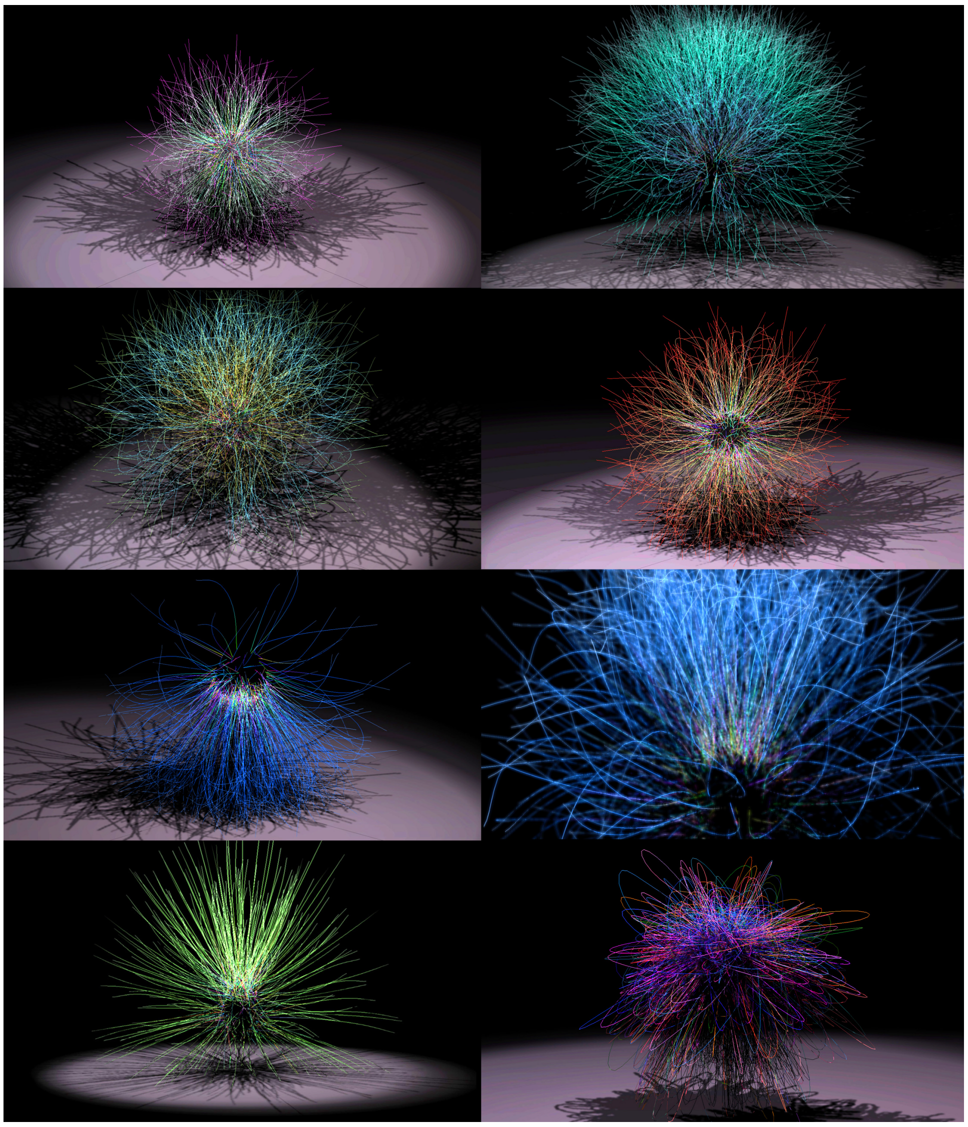


Figure 3.18: *Les Motif*, R. Hoetzlein (2010). Experiments with curve sets.

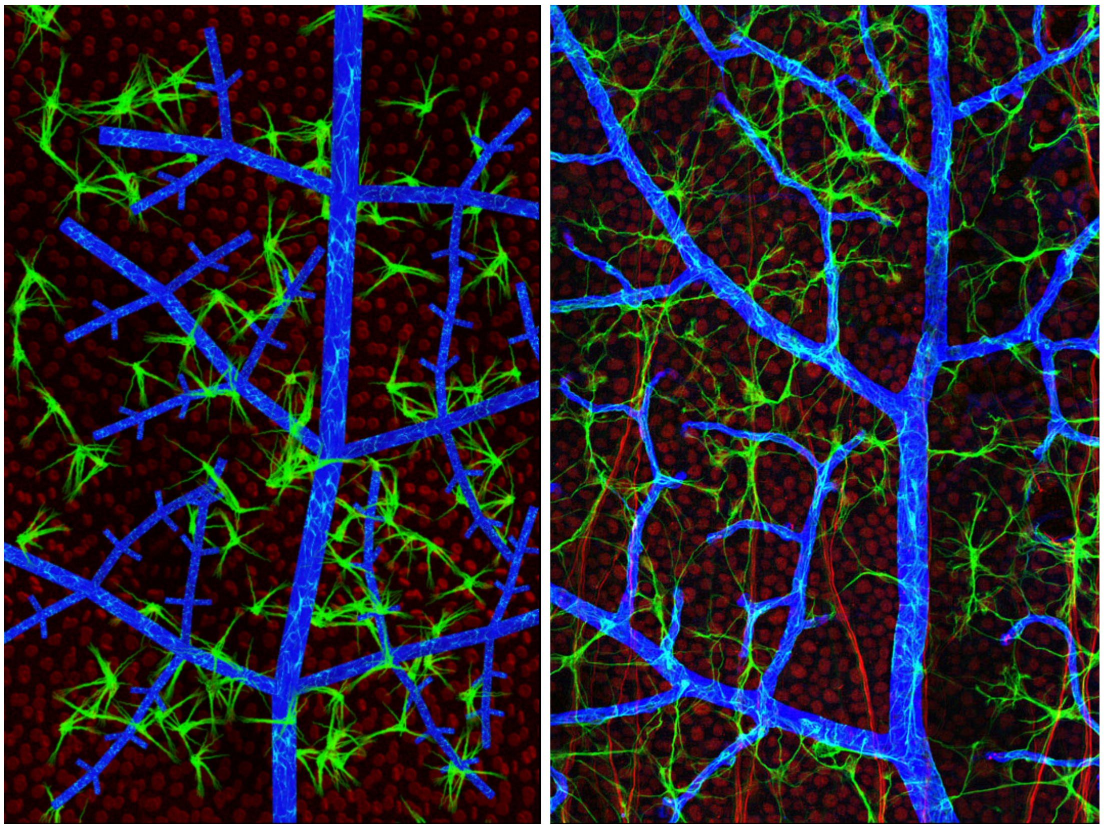


Figure 3.19: Synthetic rendering created in LUNA (left) compared with real astrocyte imagery of a rabbit retina. Blood vessels (blue) were modeled using a tree object, while astrocytes (green) were modeled as Bézier curves using a physically based spring-system with added noise. Left image simulated by the author using LUNA. Right image courtesy of the Neuroscience Research Institute, University of California Santa Barbara (c) 2010, Gabe Luna, Geoffrey Lewis, and Steve Fisher.

3.5.5 Biological Modeling

These examples demonstrate that LUNA is able to achieve many distinct creative styles. In addition to creative projects, LUNA was used in a scientific collaboration with Panuakdet (Mock) Suwannatat and Tobias Höllerer, based on astrocyte imaging results by Gabe Luna, Geoffrey Lewis, and Steve Fisher (Neuroscience Research Institute, Univ.

of California Santa Barbara), and B.S. Manjunath (Dept. of Electrical and Computer Engineering). This project explored the use of LUNA to create synthetic models of real world biology. Figure 3.19 shows an astrocyte image of a rabbit retina, with blood vessels (blue) shown next to astrocyte cells (green). The synthetic model, at left, is the first example of a procedural model whose goal is to visually match the structure of a micro-cellular network whose biological organization is unknown. The motivation for this ongoing project is to generate images sufficiently similar to real world microscopic images that vision algorithms used to detect astrocyte cell centers and geometry could be evaluated on synthetic data with known ground truth.

3.6 Conclusions

A visual dataflow language, LUNA, is presented for the creative exploration of procedural models using an intuitive, minimalist interface. Its design follows from a combinatorial approach influenced by a series of design goals established from creative dimensions that are of particular interest to media artists. Experiments with the interface show that it is possible to rapidly explore interesting, alternative designs by quickly connecting and arranging high level tiles representing procedural objects. The graphical interface in LUNA enables this by making specific use of layout, color, and connection behavior to meet these design goals. In addition, LUNA itself is capable of many different creative styles, including procedural and organic modeling, interactive art using video input, game design, and high quality rendering with deferred shading.

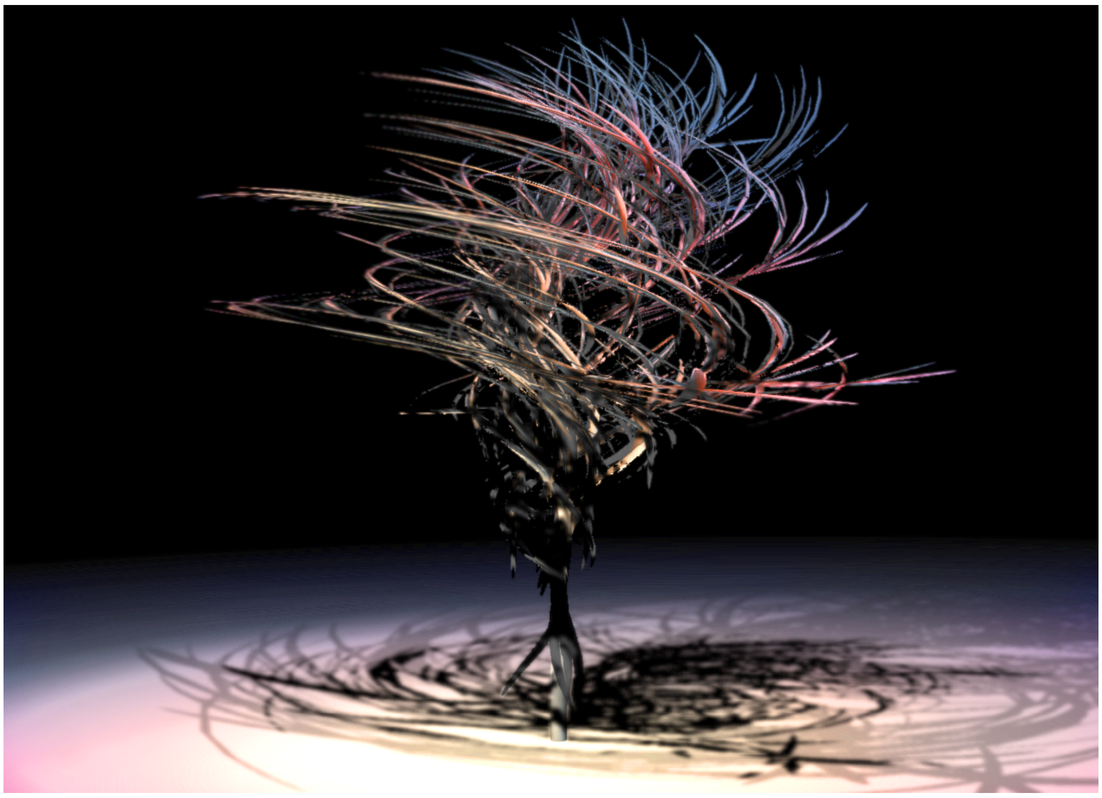
Although any visual dataflow language requires some symbolic interaction, the features of LUNA are constructed to meet the needs of artists, allowing them to focus on the task of exploring creative possibilities. The dimension of *modality*, for example, is embedded in the two-level toolbar design and in the currently available media types, while the dimensions of *dynamics* and *structure* are embedded in the temporal and geometric behavior of objects as they are manipulated by the graph. The critical features of LUNA, established by these creative dimensions, are thus incorporated into both the interface and the structure of the language.

While LUNA presents interesting possibilities, it is a new system which would benefit from further development and testing. Currently (October 2010), there are 29 nodes available in LUNA. One future goal is to expand this vocabulary to include audio, video, and device interaction. In the area of interface design the issue of temporality is not yet addressed as all nodes perform their actions continuously, making it difficult to script or trigger different behaviors over time. This suggest an interactive timeline is needed in addition to the canvas area. Specific areas, such as the types of parameter controls in the property panel (currently only sliders are present), also deserves more attention. Finally, user studies may make it possible to establish real differences in expressive power between LUNA and other languages.

LUNA is presented here as a novel interface for the construction of dynamic, creative objects with interactive feedback, with specific examples showing how artists can use this visual language to quickly create new and interesting models. Six creative dimensions of interest to artists contribute to both the interface and structure of the language, resulting in a system which is intentionally designed to meet the needs of media artists, with the hope of unifying many of the diverse practices and techniques found in the digital visual arts.

Chapter 4

Procedural Modeling of Complex Objects



4.1 Introduction

Procedural modeling involves the use of functional specifications of objects to generate complex geometry from a relatively compact set of descriptors, allowing artists to build detailed models without manually specifying each element, with applications to organic modeling, natural scene generation, city layout, and building architecture. Early systems such as those by Lindenmayer and Stiny used grammatic rules to construct new geometric elements [Lindenmayer, 1968] [Stiny and Gips, 1971]. A key feature of procedural models is that geometry is generated as needed by the grammar or language. This may be contrasted with static geometric data structures[1] which were historically used in systems such as IRIS (SGI) Inventor to provide a flexbile way to represent geometric data, and later in IRIS Performer for efficient hardware rendering of large scenes [Strauss, 1993] [Rohlf and Helman, 1994]. Static data structures take advantage of persistent GPU buffers and reordering of graphics state switches to render scenes in real time, and are ideally suited to geometry with few scene changes. A more recent trend is to transmit simplified geometry to graphics hardware and to allow the GPU to dynamically build detailed models without returning to the CPU. This technique has been successfully applied to mesh refinement, character skinning, displacement mapping, and terrain rendering [Lorenz and Dollner, 2008] [Rhee et al., 2006] [Szirmay-Kalos and Umenhoffer, 2006]. However, there are still few generic methods for efficient procedural modeling that take advantage of graphics hardware. A taxonomy

[1]Also referred to as *scene graphs*, although this term has been applied in many other contexts.

of dynamic geometric data types is developed here for representing various classes of static and non-static geometric objects for interactive procedural modeling.

Several early procedural systems developed from the study of nature. L-systems, introduced by Lindenmayer and Prusinkiewicz, use grammars based on string substitution to model plants [Lindenmayer, 1968]. Kawaguchi models shells, fossils and branches by iteratively and recursively constructing complete models from transformed shapes [Kawaguchi, 1982]. A method related to fractal geometry is employed by Stiny, who constructs models of painted regions using *shape grammars* that replace shape patterns with other shapes. Grammars have also been applied to the modeling of building architectures [Müller et al., 2006], whereby large volumes such as walls are replaced with windows, doors and framing. While grammars have a range of applications, they are often limited to a certain class of objects, are not easily animated, and it is difficult to determine how their evaluation might be parallelized.

An alternative to grammars is to rely on programming languages themselves to generate complex forms. Synder uses a C-interpreter to model volumes and surfaces for CAD objects, with *opreators* that can perform a variety of tasks, including constraint solving, integration and spatial deformation [Snyder, 1992]. In the animation language AL, May uses a Scheme interpreter to dynamically generate and animate complex objects, such as generative architectures and anatomical muscle models [May et al., 1996]. AL uses functional operations that arbitrarily transform, deform, or generate new geometry. The commerical software Maya includes MEL, a interpreted scripting language that can be used to generate geometry in relation to efficient, underling C/C++ object models [Gould, 2002]. Although interpreted languages offer a great deal of flexibility, user interaction with such systems is difficult to define, and low-level interaction with graphics APIs makes it difficult to efficiently group geometry for hardware acceleration.

Visual dataflow languages, VDFLs, provide another solution to procedural modeling. An early VDFL for procedural modeling is ConMan, a system by Paul Haeberli that uses a directed acyclic graph (DAG) of behavioral nodes to generate objects [Haeberli, 1988]. These nodes are similar to the language Squeak, whose inventor Luca Cardelli describes them as “fragments of behavior”, which perform actions on various data [Cardelli, 1985]. Xfrog, on the other hand, is a natural modeling system in which *p-graphs* (a layout of labeled vertices), are converted into an *i-tree* by replacing vertices with primitives that are then rendering interactively [Deussen and Lintermann, 2004]. The commerical software Houdini uses VDFLs to describe and animate graphs consisting of surface operators (SOPs), particle operators (POPs), dynamic operators (DOPs), and render operators (ROPs), among others. In addition to developing an integrated VDFL framework for procedural modeling (discussed later), Ganster provides an overview of the features and drawbacks of several other procedural languages [Ganster and Klein, 2007]. VDFLs provide a more natural interface and also a great deal of flexibility as nodes may perform any operation, but it is not entirely clear how these functional graphs should be connected to performance-based scene graphs.

While procedural languages have generally increased in expressiveness over time, methods for efficiently representing static models with moving parts have evolved to meet other needs. Early systems employed a hierarchical description of object transformations, introduced in platforms such as GKS and PHIGS+ in the 1980s [Dam, 1998]. In 1993, SGI introduced IRIS Inventor to flexibly represent models using a *scene graph* with static geometric data structures represented by a directed acyclic graph (DAG) and supplemented with nodes that include "shapes, properties, and groups [Strauss, 1993]." One year later, IRIS Performer was introduced to efficiently manage graphics hardware with improvements such as sorting by texture state and spatial culling [Rohlf and Helman, 1994]. Building on this, OpenSceneGraph and OpenSG sought to bring these enhancements to consumers by expanding to different platforms, porting from SGI IRIX to Linux, and new rendering APIs such as Microsoft's Fahrenheit, later replaced by DirectX [Harrison, 2007]. Over time, the term *scene graph* has come to signify a wide range of approaches to storing geometric objects. In a panel discussion at SIGGRAPH 1999, graphics professionals were found to have differing views on the definition of scene graphs [Bethel, 1999]. A central issue to organizing geometric objects is that they have competing needs in terms of spatial organization (for culling), material properties (for state switching), functional behavior (animation), and generative abilities (procedural modeling). Avi Bar-Zeev collects and discusses several of these competing goals [Bar-Zeev, 2007].

The problem addressed here is how to integrate the performance benefits of scene graph organizations with the flexibility of procedural modeling languages, to develop a system for real time procedural animation. LUNA is introduced as an intuitive, high performance, visual dataflow language for procedural modeling. The system is used to interactively model complex organic objects with a deferred shading engine for high quality rendering. For evaluation, a procedural reference model is proposed to do performance comparison with Houdini (the only other generic, modern procedural dataflow language for procedural modeling the author is aware of).

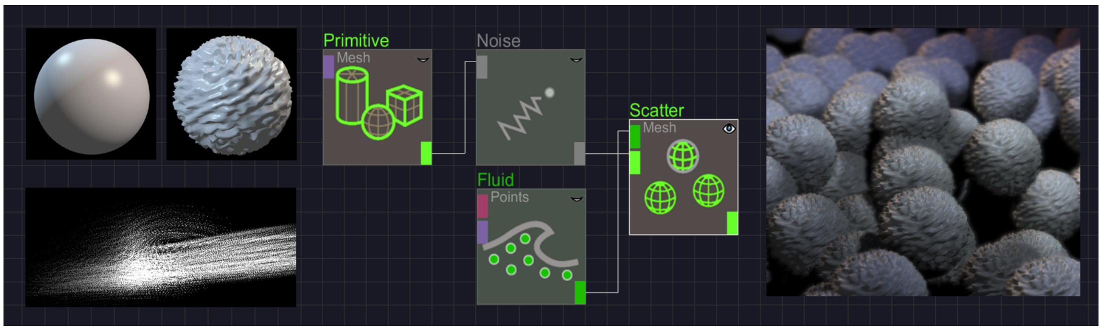


Figure 4.1: This example shows a graphical interface representing a *procedure graph* that generates the results shown on the right. The inputs consist of a sphere with animated noise displacement (top left), and a particle-based fluid system (bottom left).

4.2 Language Design

4.2.1 Overview

The approach taken here is to consider how to integrate the flexibility of procedural modeling languages with the performance benefits of scene graph to develop a language for real time procedural animation. The technique is based on the following design principles:

1) *A Procedure graph generates multiple scene sub-graphs.* A procedural node represents the behavior of a high-level concept, and is capable of taking multiple scene nodes as input and generating multiple scene nodes as output.

2) *A Scene graph uses API-dependent proxy objects to manage rendering.* A scene node specifies a local coordinate system (LCS), surface materials, and geometry buffers. A procedural node groups its output scene graph into a set of objects of similar type, which are rendered using an API-dependent *proxy object* which maintains GPU buffers.

3) *Execution takes advantage of scene graph organization, hardware rendering, and GPU kernel execution.* LUNA takes advantage of the GPU in three ways: 1) The behavior of procedural nodes can be optionally executed on the GPU using GPU-specific node kernels, 2) Scene graph organization is optimized through grouping of geometry, 3) Rendering is optimized for a hardware-based deferred shading engine.

The explicit distinction between behavior and structure can be found in several newer visual dataflow languages. In the integrated system by Ganster and Klein, a *model graph* represents functional operations that generate structure [Ganster and Klein, 2007]. Although the system is flexible, complex models such as trees are generated through iteration which occurs at the model level, reducing the performance of the system considerably. Forbes develops a system in the area of information visualization that models behaviors and structures as distinct graphs that can operate on one another

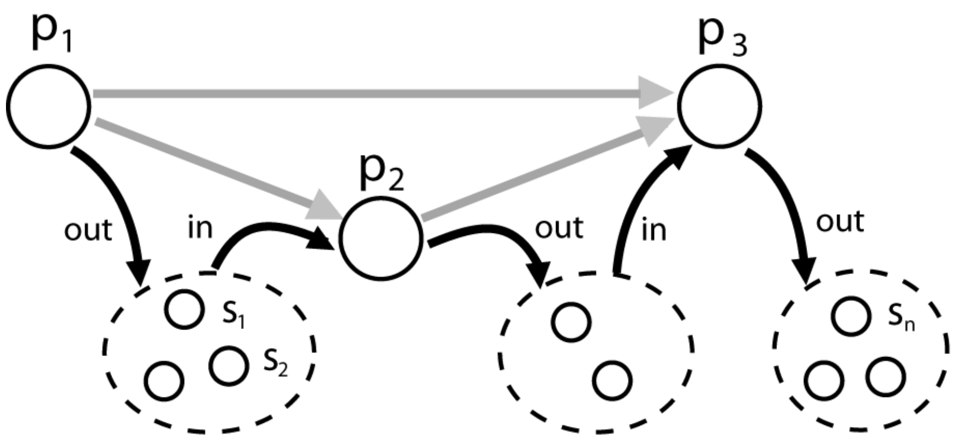
[Forbes et al., 2010]. This system creates complex transitions for visual data but does not deal with generative geometric structures.

In LUNA, procedure nodes represent high level operations and iteration inside their kernels produces multiple scene outputs. An example is shown in Figure 4.1, in which a dynamically moving fluid system (represented as particles and animated using smoothed particle hydrodynamics) is used as the point locations for instancing a dynamically deforming, animated sphere. This high level graph, visibile to the user, is the *procedure graph* (P). While the scene graph (S) formally represents the total visual output of the system, and remains hidden in the graphical interface, because of its generative nature it is useful to conceive of the system as a set of procedure nodes in which scene sub-graphs flow through the system. A procedure node can essentially create, modify, destroy or transform an input scene graph to generate an output scene graph. All of these changes are found in the complete graph, as the scene graph holds the inputs and outputs of all procedural nodes.

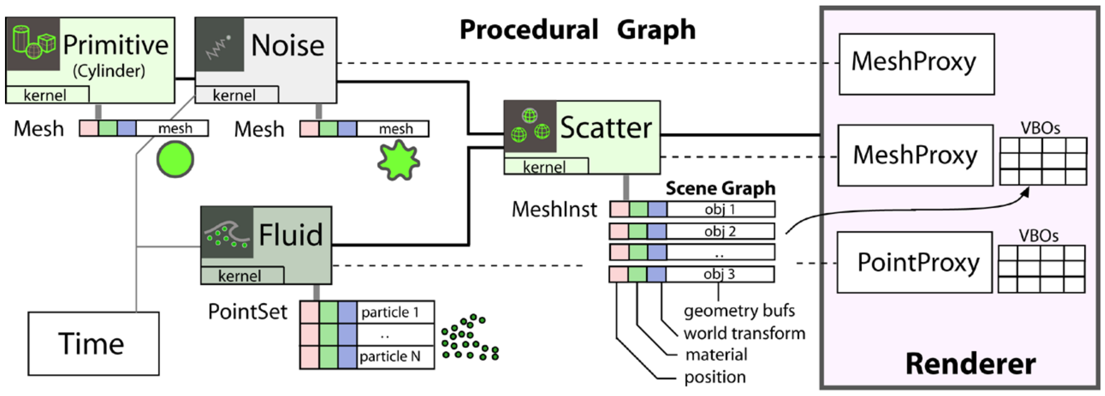
4.2.2 Formal Definition

Formally, LUNA scenes are defined by two directed acyclic graphs, a procedure graph (P) that modifies input and output subsets of a scene graph (S). Each graph is a set of nodes and edges, while the P vertices additionally reference subsets of S:

$$P = \{P_v, P_e\} \tag{4.1}$$



(a)



(b)

Figure 4.2: Structure of the LUNA graphs in which, a) a *procedure graph* operates on subsets of a *scene graph*, and a specific example b) in which four procedural nodes are used to generate multiple scene nodes to render the image shown in Figure 4.1. In the actual storage of the graph, all nodes are maintained by a single graph system, and their usage and behavior are distinguished by object semantics.

$$S = \{S_v, S_e\} \tag{4.2}$$

Each of the nodes and edges of these graphs have a particular interpretation. A procedural node, $P_v \in P$, defines behavior that consists of a CPU kernel, an optional GPU kernel, which operate on an input subset of the scene graph, $S_{in} \subseteq S$, to produce an output subset, $S_{out} \subseteq S$. Procedural nodes also include a *proxy object*, P_x, which is referenced by the node but managed by renderer, and maintains the vertex buffers and graphics state for the object's output sub-graph. A procedural edge (P_e) connects two P-nodes and defines the functional inputs to a behavior (such as an image and a mesh being the input to a displacement).

$$P_v = \{S_{in}, S_{out}, P_x, kernel(CPU/GPU)\} \tag{4.3}$$

$$P_e = \{P_a, P_b\} \tag{4.4}$$

A scene node, $S_e \in S$, defines a media object, which may be an image, shape, mesh, sound, or other structure. For geometric objects, a scene node maintains a local coordinate system (LCS), references to material nodes that define shaders and textures, and geometry buffers which maintain vertices, edges and faces. Scene edges, S_v, define relationships between two geometric objects, the most common example of which is an *instance* relation (e.g. MeshInst refers to a Mesh), described more later.

$$S_v = \{\text{LCS, material, data buffers}\} \tag{4.5}$$

$$S_e = \{S_a, S_b\} \tag{4.6}$$

A graph demonstrating this structure is shown in Figure 4.2. As the system is procedural, the scene graph is hidden from the user since the output scene geometry can grow rapidly. Users create edges by click-dragging from an input node to an output node, as shown in Figure 4.1. In this example graph, the Scatter P-node is connected to the renderer, and in the graphical interface this is indicated by the 'eye' icon. As such, any object in the procedure graph may be visualized by clicking on this icon, allowing multiple objects to be connected to the renderer. A special Time node is automatically introduced by the system to trigger time-dependent changes per frame.

4.2.3 Evaluation Model

The evaluation model consists of two basic steps. First, the procedure graph is traversed from the Time node outward to any dependent nodes that require updating, from left to right in Figure 4.3a. In this example, it requests that all nodes except the primitive (sphere) be re-evaluated. The second step is to traverse the graph in depth-first order starting from any *visible* P-node connected to the renderer, to evaluate that node and update its geometry. In Figure 4.3b, the only connected visible node is the Scatter node, which acts as the current root of the traversal. Repeated evaluation along different traversal paths is avoided by tagging the nodes that have already been evaluated on the current frame. For each visited node, its CPU or GPU kernel is called, which will generate the output sub-graph S_{out}. The proxy object P_x for that P-node is then called to update vertex buffers and to render the resulting sub-objects.

The example of Figure 4.2 uses a point-based smoothed particle hydrodynamic fluid simulation, a sphere primitive, and a noise generator, to render a number of organic

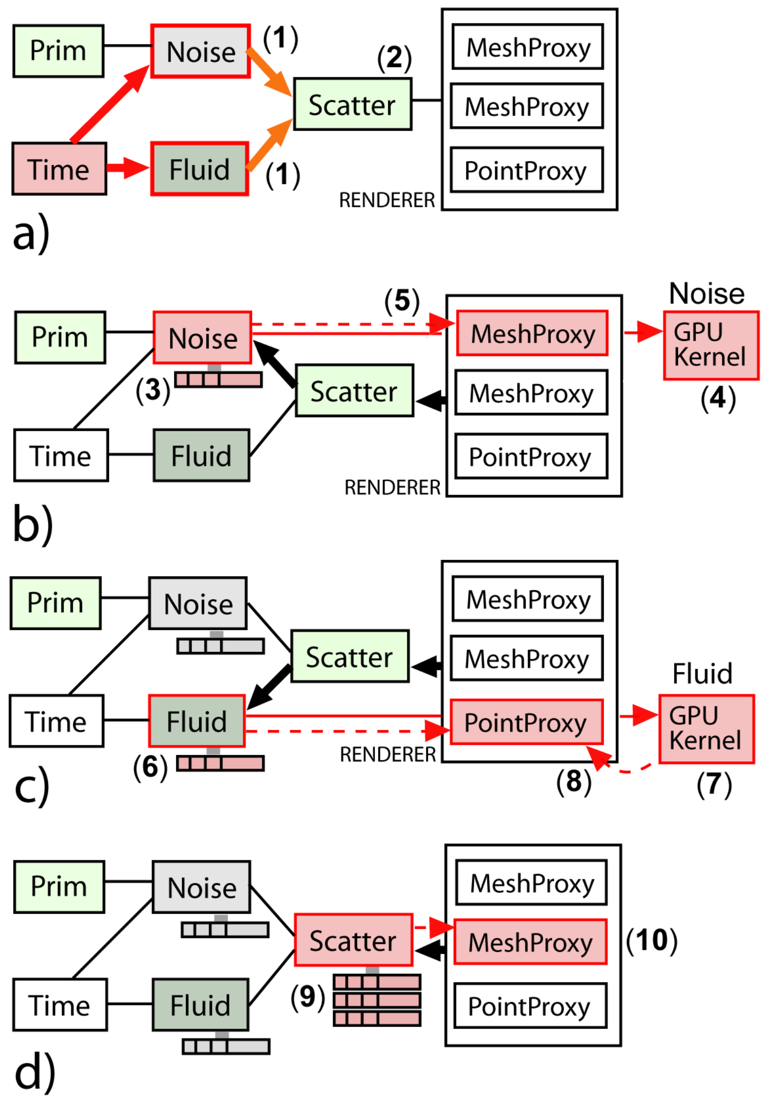


Figure 4.3: Sequence of steps in the evaluation model: a) The Time node requests updates by traversing graph dependencies, b) The Noise node is traversed in depth-first order to update a noise-animated mesh using a GPU kernel, and the result is stored on the GPU via a MeshProxy, c) The Fluid node is visited to update fluid particle locations using a GPU kernel, d) The Scatter node generates or updates the world transform of mesh instances, using the GPU mesh from step b as an instance that is scattered at the point locations of step c.

noise-animated spheres moving in a fluid. The Primitive node creates a mesh scene output which is generated only once. Figures 4.3b through 4.3d show the results of the traversal process on each node in the example. The Noise node generates an animated noisy sphere which is updated on every frame, while the Fluid node generates a PointSet output which updates the fluid particle locations per frame.

The output scene graphs of Noise and Fluid, consisting of a mesh and a point set, are used as inputs to the Scatter node, which generates a list of MeshInst objects at each of the point locations. A MeshInst is a sub-class of a Mesh scene node which supports much of its functionality through a reference to another Mesh node. In this case, many MeshInst nodes with different transforms refer to a single input mesh, providing a way to represent geometry instancing. To avoid repeatedly generating the MeshInst output scene graph, the Scatter P-node detects changes in the number of elements of its inputs, and only rebuilds when necessary. This allows nodes to generate more geometry as needed, or to efficiently animate the geometry present based on update notifications.

4.3 Implementation

4.3.1 Discrete Geometry

Scene nodes store and transmit geometry through the graph. Their representation consists of discrete geometry stored in the *data buffers* of each node. These buffers have a direct mapping to GPU buffer objects and can be directly copied from CPU memory to the GPU. While a common programming method of storing objects uses

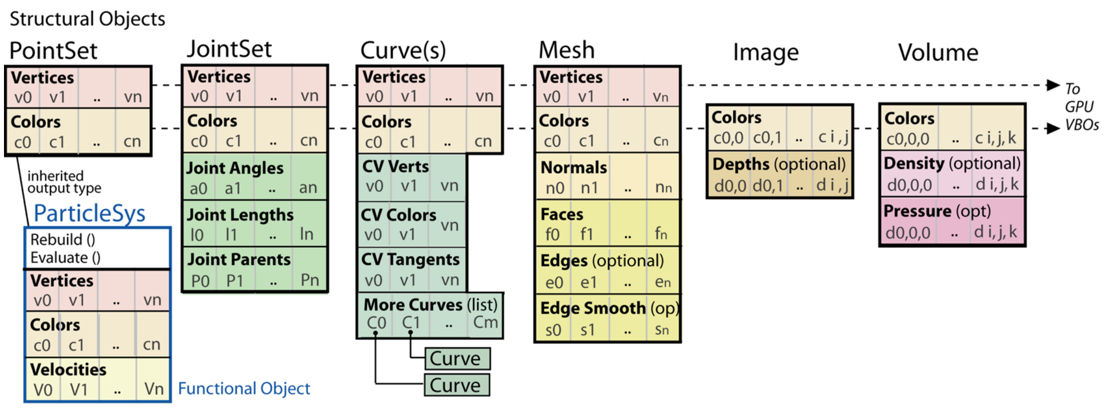


Figure 4.4: Objects are represented as discrete geometry in LUNA with uniform buffers. Buffers may vary in length, as in the example of control vertices (CVs) for the Curve object shown. Other buffers might store pointers to allow lists of the same object class (C0,C1,..,Cn). Data is sent to the GPU in a unified way using proxy objects. Some buffers are allocated by derived nodes, such as particle velocity. Other functional objects may operate across many object types. A bend transform, for example, works with any object that has a vertex buffer.

statically linked data structures to describe geometric elements, this can make it difficult to expand object attributes at run time, whereas GPU performance and flexibility are improved by storing structures in contiguous buffers of uniform type. Our technique uses uniformly typed, variable length buffers on the CPU to represent discrete sets with named semantics, as shown in Figure 4.4.

In creating geometric types for LUNA, the concept of uniform buffers is abstracted to an arbitrary number of named buffers, each with variable length, and fixed sized elements within a given buffer that implement a particular discrete geometry. Their layout is designed to match graphics hardware buffer objects, yet store every per-element variable needed for high level objects including hierarchical relationships. Trees for example, are stored in JointSet scene objects, which contain buffers that hold references to parent and child branches.

A key benefit of using uniform buffers is that broad classes of objects can be treated similarly both during evaluation and rendering, without the need to define new functions. Points, curves, joints and meshes are all defined with their three-dimensional vertices as the initial buffer. Thus, modifiers such as twist, bend and distort can be performed in a consistent way across many different geometry types, including points, curves, and surfaces. The renderer requires vertex-face information and normals which are available as additional buffers in mesh objects.

4.3.2 Performance

The structure of LUNA supports procedural modeling by allowing multiple outputs for each behavioral operator, while its evaluation model supports several performance enhancements. First, only the portions of the graph that change over time are re-evaluated per frame. Second, the proxy objects efficiently transfer only the sub-graph geometry that has changed from CPU to GPU, and also group that geometry to share render state transitions (e.g. materials). This introduces a potential performance issue common to VDFLs: the replication of data buffers at later points in the graph. In the current system, modifiers copy input objects to outputs, and then perform transformations on the output buffers. This resolves situations where a single input is modifier in two different ways, but is inefficient when there is only one output. In the future, the evaluation model could be modified to selectively remove this inefficiency.

The LUNA design takes advantage of the ability of modern graphics hardware to invoke re-entrant kernels. Giden et al. develop such a system consisting of graphs of CUDA kernels for the eXtensible Imaging Platform (XIP) of the National Cancer Institute [Giden et al., 2008]. In LUNA, each procedural node may have an optional CUDA GPU kernel that performs simulation, or geometry generation, on the GPU. Notice that there can be multiple GPU kernels in a graph in Figure 4.3b, and these can be interspersed with rendering calls to allow several high level nodes to be evaluated on the GPU while also using the GPU for rendering. At present, only the GPU kernel for the SPH fluid simulator is written, and has not yet been tested in the context of other GPU nodes although the design allows for it.

Data replication between CPU and GPU versions of an object are avoided through the proxy objects. At present, the CPU maintains the final copies of all data. For objects with single outputs, the proxy object transfers that data to the GPU and renders it, updating only changed buffers. In the future, for nodes evaluated with GPUs kernels, it should be possible to directly update the proxy object, as in Figure 4.3c, without returning the data to the CPU. By authoring multiple nodes for the GPU, this should allow complete graphs to be evaluated entirely on the GPU. The CPU-GPU bus transfer is thus an optional step based on which portion of the graph requires the data next.

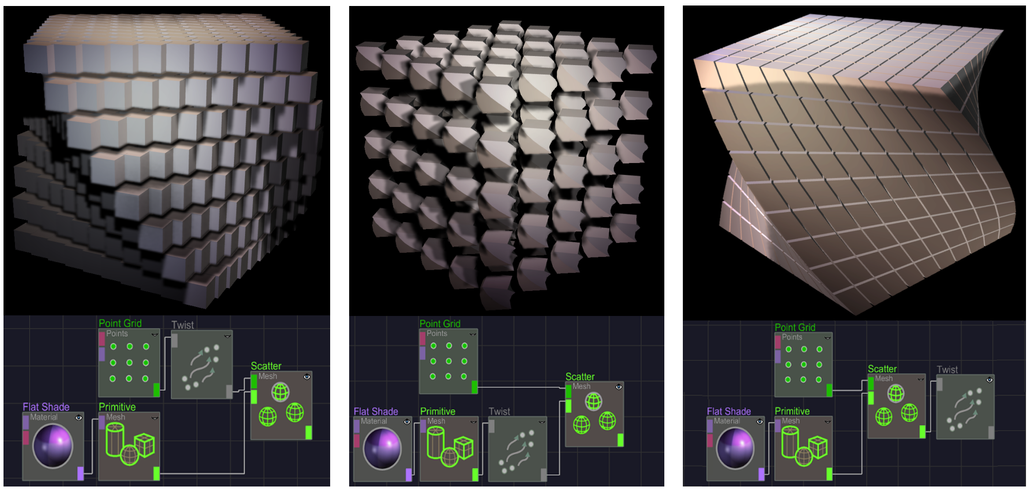
Although CPU-GPU bus transfers incur a cost for dynamic geometry this can be alleviated by incremental data transfers. In RenderAnts, Zhou et al. develop an adaptive parallel pipeline for geometry transfer to the GPU in a Reyes rendering environment. Dynamic loading of the GPU allows for more complex models since scene detail is not bound by GPU memory. This process of incremental data transfer can be applied to procedural modeling more directly since geometry is generated on-the-fly, and LUNA proxy objects support this by *reusing* the same vertex buffer objects for different mesh geometries in the same render frame. Although not adaptive in the same sense as RenderAnts, each P-node generates an output sub-graph of varying size based on user requested detail levels, and the proxy can quickly load different meshes in a single set of VBOs allowing for much greater scene complexity. This is demonstrated in particular by the examples in Figure 4.7a.

4.3.3 Rendering

The integration of procedural modeling with high quality real-time rendering is a relatively unexplored research area. Although deferred shading is typically found only in game engines with objects of particular class types (terrain, etc), the LUNA rendering engine supports deferred shading and multi-pass rendering of procedural models using OpenGL. Individual objects in LUNA may have vertex, pixel, and geometry shaders assigned to them which permit advanced visual effects on generated models. These shaders are included in procedure graphs as material inputs to geometric nodes. At run time, the proxy object for a particular node runs any Cg shaders prior to primitive drawing. Screen space shaders allow for effects such as depth-of-field, deferred soft shadows, motion blur and light bloom. Increased performance due to geometry classes, lack of temporary objects, and streamlined buffer transfers allows procedural models to be generated and shaded in a real time, high quality rendering environment.

4.4 Procedural Results

4.4.1 *Twist*: Modifiers and Order of Operation



(a) (b) (c)

Figure 4.5: Modifiers operate on different geometry types. In these examples, a cube Primitive (bottom left) is instanced at PointGrid locations (top left) by a Scatter node (right). Placing a twist at different stages in the graph produces a) A twisted grid with untwisted cubes, b) Twisted cubes located at an untwisted grid, and c) A regular grid with regular cubes, the whole of which is twisted.

Modifiers are procedural nodes that operate on *any* input geometry type. This is accomplished by copying the input sub-graph, which may be points, lines, curves, or

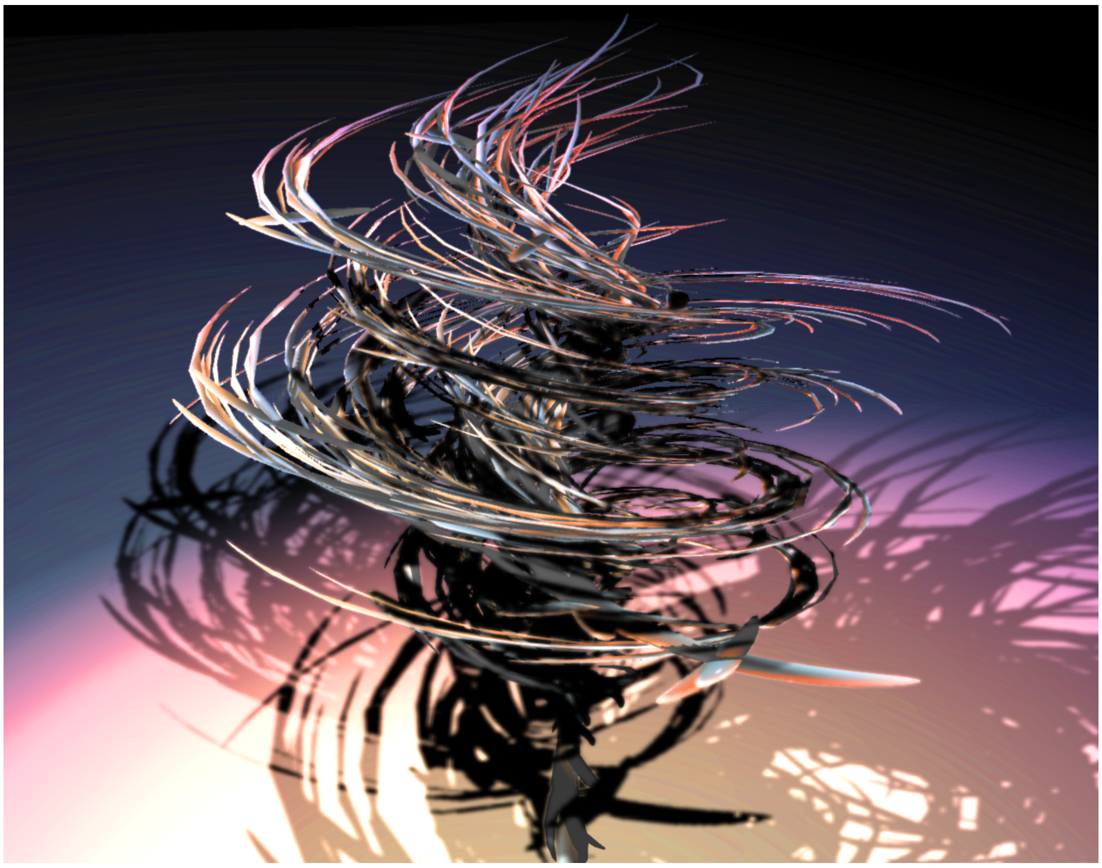


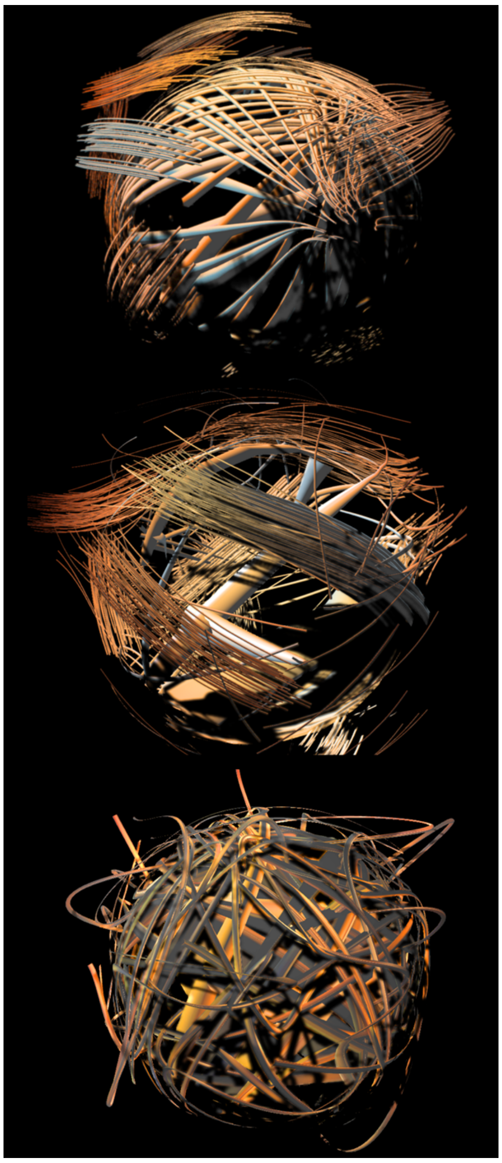
Figure 4.6: *Twist*, R. Hoetzlein (2010). A tree is constructed from a JointSet hierarchy which is then built up using cylinder Primitives. The overall shape is then twisted to produce these results, rendered in real time using deferred shading, shadows, and depth of field.

surface geometries, to the output sub-graph, and then uniformly modifying the point locations according to some global transform. The order of these operations is also important, as the examples in Figure 4.5 show, where a cube is instanced at grid locations. Depending on what stage the Twist operation is performed produces different variations of twisted/untwisted objects. In this example, the grid locations, the cube mesh, or the entire object are twisted separately. Although a point cloud (PointSet) can be twisted by its vertices, to correctly twist a mesh requires transforming both vertices and normals. This is done by allowing the Twist node to inspect the type of its output and perform an additional transformation on the normal buffer.

A more complex twisting example is shown in Figure 4.6. Here, a tree is constructed from a JointSet hierarchy, which is made into a solid object by scattering Cylinder primitives at the joint locations. Unlike an interpreted function, which might recursively traverse the tree, determine joint angles, and instance cylinders as it proceeds, in LUNA the structure of the entire tree is generated first. Subsequently, any object may be instanced at this tree structure separately from the tree definition. A twist is added to the final result, distorting the transformed cylinder geometry and producing the image shown.

4.4.2 *Scatter*: Compound instancing

One of the key features of procedural modeling systems identified by Reeves is replication, the ability to generate instances of a model with subtle variation [Reeves et al., 1990]. The LUNA language can distinguish between *instancing*, which is the copying of a single mesh at multiple locations and orientations, and *replication*, the ability to repeatedly re-evaluate a procedural model so that each instance is different from the last. Currently, the node for LUNA to support replication is not yet written, as this would require repeated evaluation of procedural sub-graphs. However, LUNA


(a)

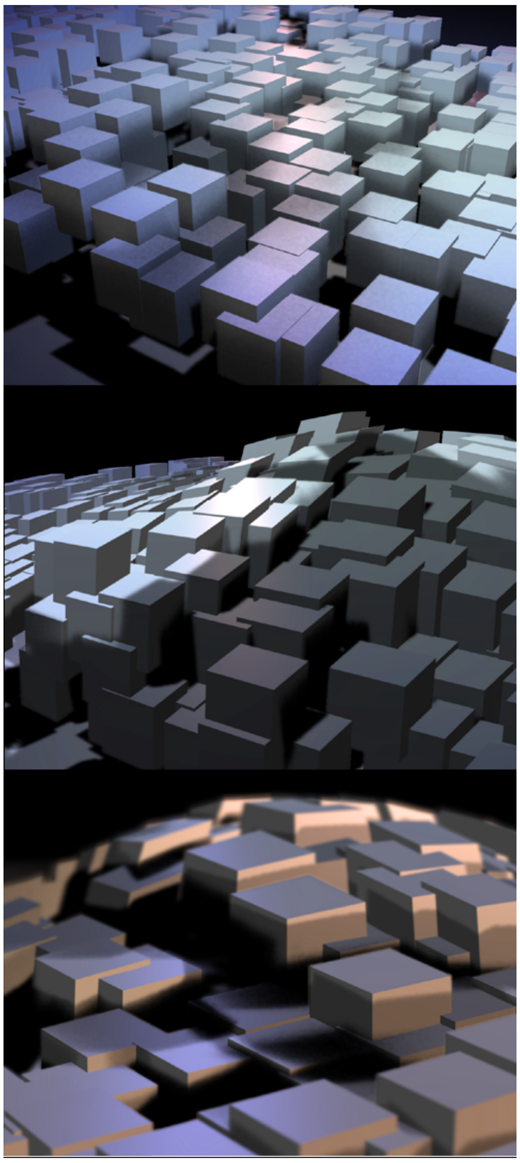
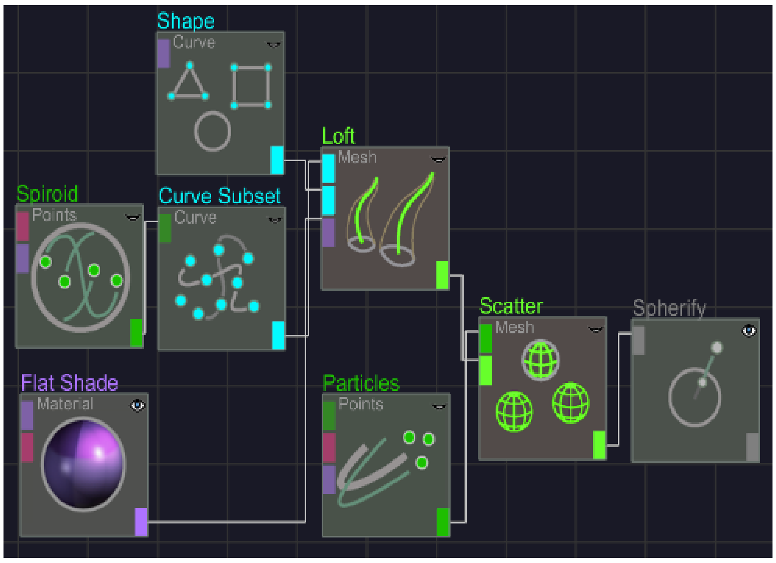
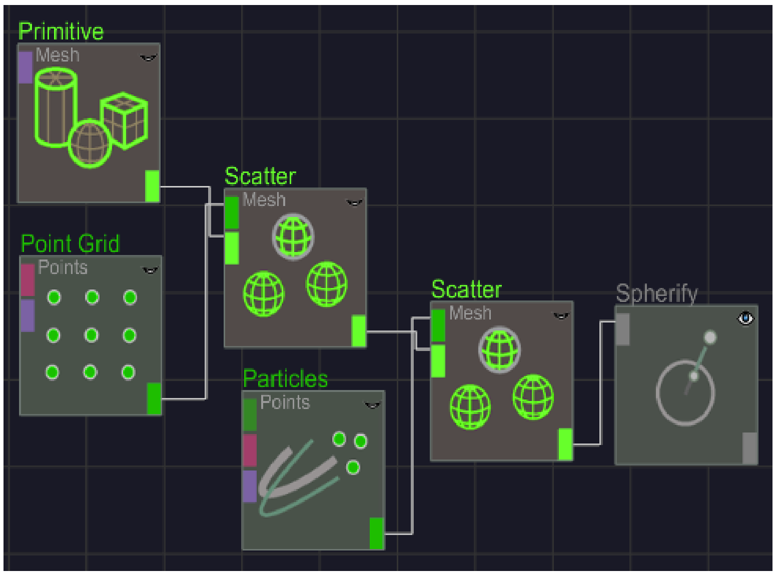

(b)

Figure 4.7: Compound systems created using a variety of objects, a) Swept surfaces are generated from a particle system, which are then scattered to create the appearance of three dimensional brushstrokes composed of tubes, b) A collection of cube primitive is scattered using a grid, then scattered again to produce variation. In both systems, space is distorted by using a spherify operation on the output.



(a)



(b)

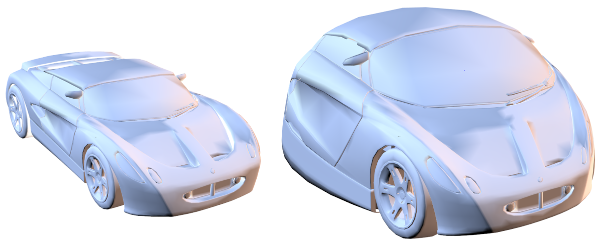
Figure 4.8: Graphs for the visual results of Figures 4.7a and 4.7b, showing a) a collection of loft surfaces which are scattered, and b) a collection of scattered cubes which are scattered again. In both cases the compound result is *spherified* to distort the final space.

does support nested or compound instancing, the ability to create instances of models that themselves contain instances.

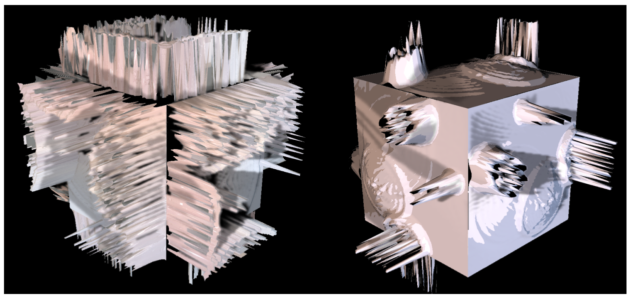
Examples of compound instances are shown in Figure 4.7 and 4.8 using a Scatter node. In the first example, 4.7a, a Loft object is used to generate N swept surfaces as output scene sub-objects according to Subset Curves defined over a set of points. These sub-objects are then Scattered to produce N*M swept curves, whose parameters in this case give the appearance of three dimensional brush strokes. In the second example, 4.7b, a cube primitive is Scattered at regular locations on a point grid. The result is then Scattered again at moving particle locations to create N*M output cubes with more variation. As scattering multiplies its input by N point positions, nested scattering can result in an exponential generation of mesh geometry growing as $O(N^M)$, assuming N particles per scatter, and M scatter operations. To prevent system stalling, a user controllable maximum count is present on each Scattering node, forcing a limit on the maximum output at each to step. In the future, it should be possible to allow the renderer itself to dynamically adjust these limits to meet performance or quality goals for real time or offline rendering.

4.4.3 *Displace, Wave, Cube*: Other experiments

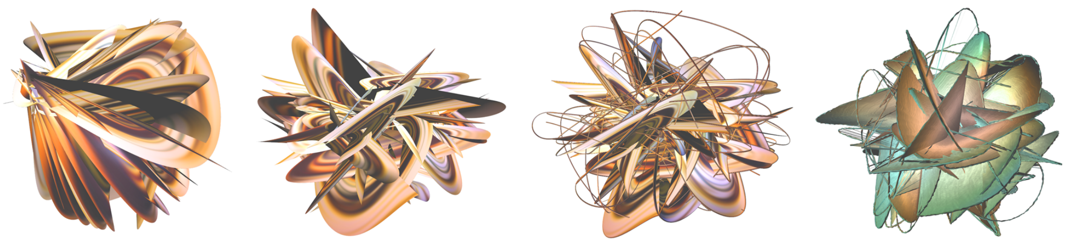
A number of other experiments are shown in Figure 4.9. These experiments include a) a spherified car mesh, b) a cube primitive with wave displacement rendering with toon shading, c) architectural forms created by using a combination of loft and fan



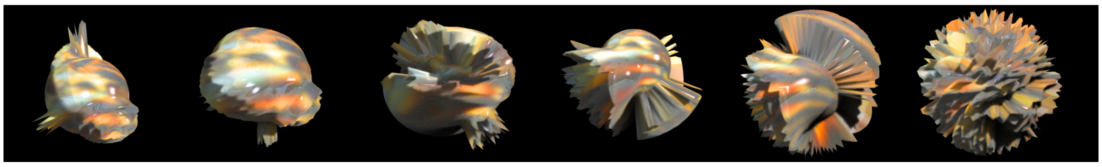
(a)



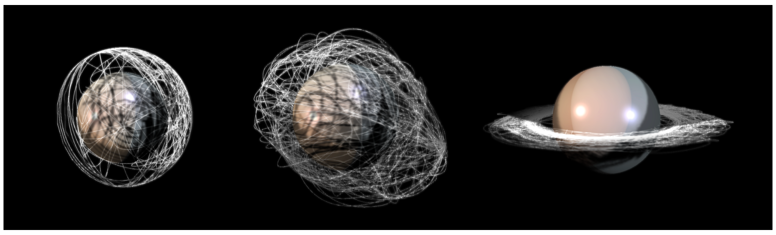
(b)



(c)



(d)



(e)

Figure 4.9

surfaces, with texturing, d) shells created by animated wave displacement of a sphere, rendered with environment mapping, and e) a planet-like form rendered by using curve subsets on a set of points surrounding a sphere.

All objects in these examples are rendered at interactive rates in real time with soft shadows, depth of field, and texturing. Notice in example e), *planets*, the rendering system allows for combinations of curve and mesh primitives, showing different portions of the procedure graph simultaneously. Example b) and d) show the use of other media types in conjunction with surfaces, where an animated image is used to displace the surface points of a mesh along its normals. This is an ideal node for implementation as a GPU kernel which, in the future, could perform mesh refinement and displacement in one step.

4.5 Performance Results

Standard reference models for procedural modeling do not yet exist. Baseline models for static geometry exist (Utah teapot [Torrence, 2006], the Stanford Bunny [Turk and Levoy, 1994], and the Happy Buddha [Curless and Levoy, 1996]), and similar test objects are available for volumetric data.[2] Thus, a novel test object for procedural modeling is proposed: a woven sphere composed of swept surfaces residing on a sphere, Figure 4.10. A detailed description of the woven sphere is presented in Appendix A and also available online.

[2]Several volumetric data sets are available from http://www.volvis.org/ with references to original authors of the data

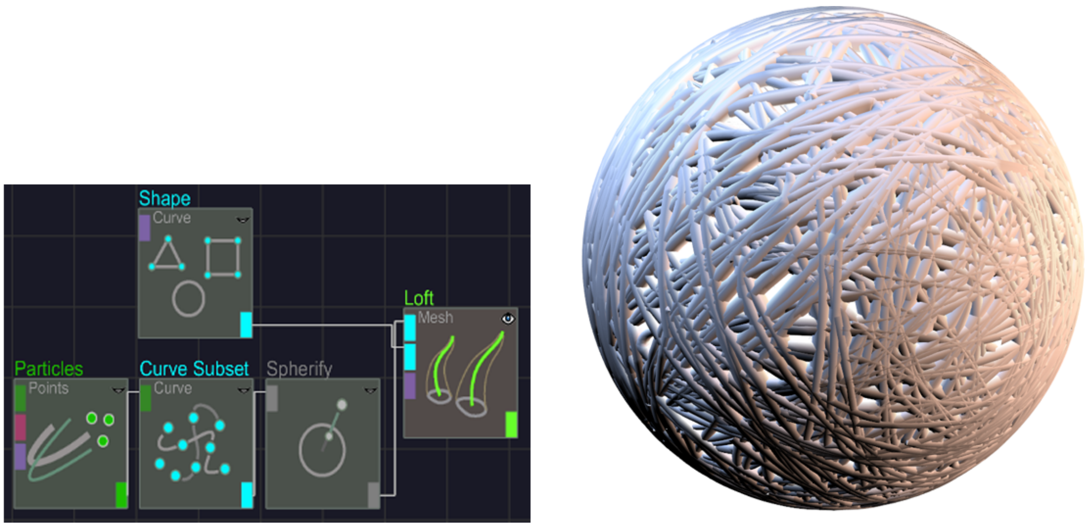


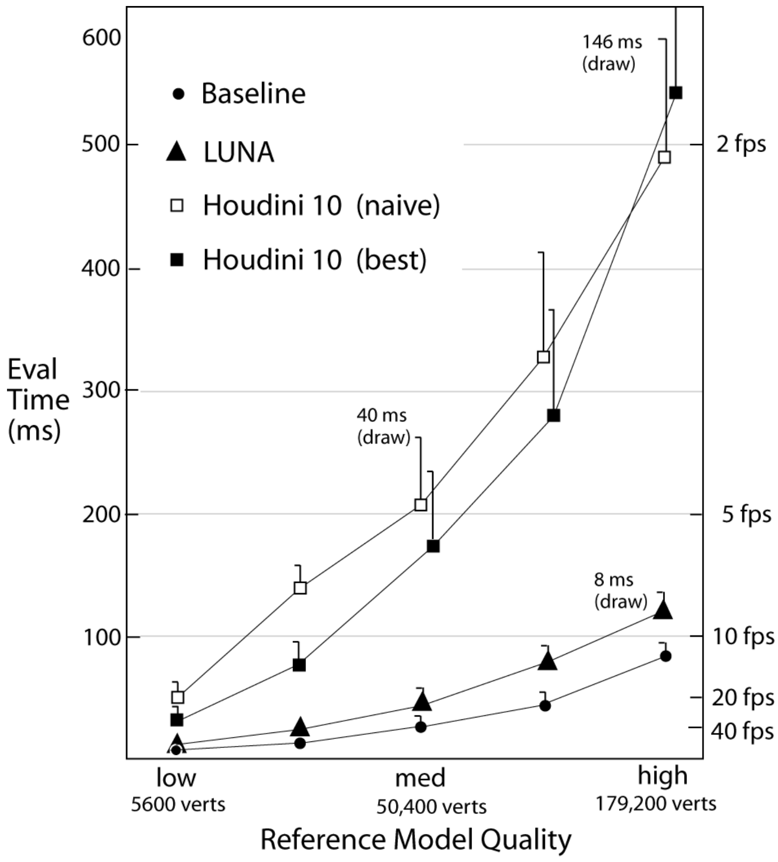
Figure 4.10: Procedural reference model introduced for performance comparisons with Houdini and a baseline OpenGL model. The object is specified as a set of curves defined from random subsets of points, normalizing the curves to a sphere, and lofting them to create tubes. See Appendix A for a detailed definition. The LUNA graph used to make this object is also shown.

The woven sphere is a suitable reference for procedural modeling for several reasons. First, it is relatively simple, and although it contains physical animation, it cannot be created from a physics simulation alone. Second, later construction steps are dependent on earlier time-dependent motions. Third, it requires evaluation of random point subsets to create curves, which necessitates a pseudo-random number generator or storing intermediate data for animation. Fourth, it generates multiple loft surfaces which cannot be created through instancing or static geometry deformation, so it is unique to procedural systems. Finally, the meshes generated require vertex re-evaluation per frame which forces a CPU-to-GPU transfer, or direct evaluation on the GPU. Thus, as hardware support for procedural generation of dynamic geometry improves, this object can be used to investigate bus transfer overhead during rendering. In addition to these specific features, the object is relatively simple from a procedural standpoint and its storage complexity can be computed theoretically.

To advance the woven sphere as a test case in the graphics community, a baseline model is implemented directly in C++ using GLUT without a procedural modeling language. This provides an absolute reference for the best possible performance since it eliminates any overhead due to language evaluation. The woven sphere was also generated in both Houdini and in LUNA for comparison.

Specific parameters for low, medium and high resolution reference models are recommended in Appendix A. For Houdini, evaluation time and viewport drawing cost are measured using Houdini's "performance monitor". To guarantee the level of detail settings match the baseline, the number of sample vertices per curve are individually counted, and the total number of vertices and faces are checked to make sure they are close to the theoretical number for a given test (Houdini does not allow one to set resampled curve resolution exactly). For LUNA, a procedure graph for the woven sphere was created with sampling parameters set interactively to match the baseline.

Performance results for the woven sphere for the baseline, Houdini and LUNA models are shown in Figure 4.11, computed on a Sager NP5320 with a Pentium M 770 at 2.13 Ghz and an ATI X700 graphics card. The LUNA reference model is implemented using five nodes: Particles, Subset Curves, Spherify, Curve (shape) and Loft, and can be constructed in the interface in under a minute with ten clicks and four click-drag



(a)

Model	Verts	Baseline (ms)		Houdini[1]		Houdini[2]		Luna (ms)	
		eval	draw	eval	draw	eval	draw	eval	draw
Low res	5k	2	<1	40	4	27	4	5	1
.	22k	10	<1	140	13	69	18	22	3
Med res	50k	24	1	203	39	183	47	45	4
.	89k	44	2	318	71	276	71	89	7
High res	179k	88	5	536	130	555	146	118	9

[1] Naive method. Created using Copy Stamping SOP and expressions.
[2] Suggested method. Created using AttribCreate SOP and Add SOP.

(b)

Figure 4.11: Performance comparison for graph evaluation and rendering time (in milliseconds) for the reference model in LUNA, Houdini 10, and OpenGL baseline.

motions. On average, LUNA is consistently 4x to 7x faster than Houdini and only 1.3x slower than the baseline model (a direct C implementation with no overhead). All LUNA models run in real time with the low res model at 150 fps (7 ms total) and the high res model at 8 fps (126 ms total)

The baseline model is implemented in C++ using GLUT for rendering. The first, naive implementation, in Houdini uses the Copy Stamping SOP to evaluate an expression to generate point subsets for curves. Repeated interpretation of string expressions is likely the cause of reduced performance here. Following discussions in online Houdini forums, a better method uses the AttribCreate OP to tag particles into groups and then the Add SOP to generate curves from the groups by attribute name. This Houdini graph requires ten nodes. Although performance is improved in most cases, this technique is slower than Copy Stamping for the high-res model and averages 10x slower than the baseline model overall.

Interestingly, although this object represents the worst case for bus transfer overhead due to dynamically generated geometry per frame, in the baseline test it represents only 4 to 6% of the total cost (5 ms out of 93 ms for the high-res model). In a game engine, however, a 5 ms cost for a 180k vertex object is unacceptable, and in such a context this object would be solved using fixed vertex shaders on the GPU while sacrificing flexibility. LUNA render times closely match the baseline. In Houdini, the viewport drawing cost averaged between 10 to 15% of the total cost (the reasons for this additional overhead are not fully understood by the author).

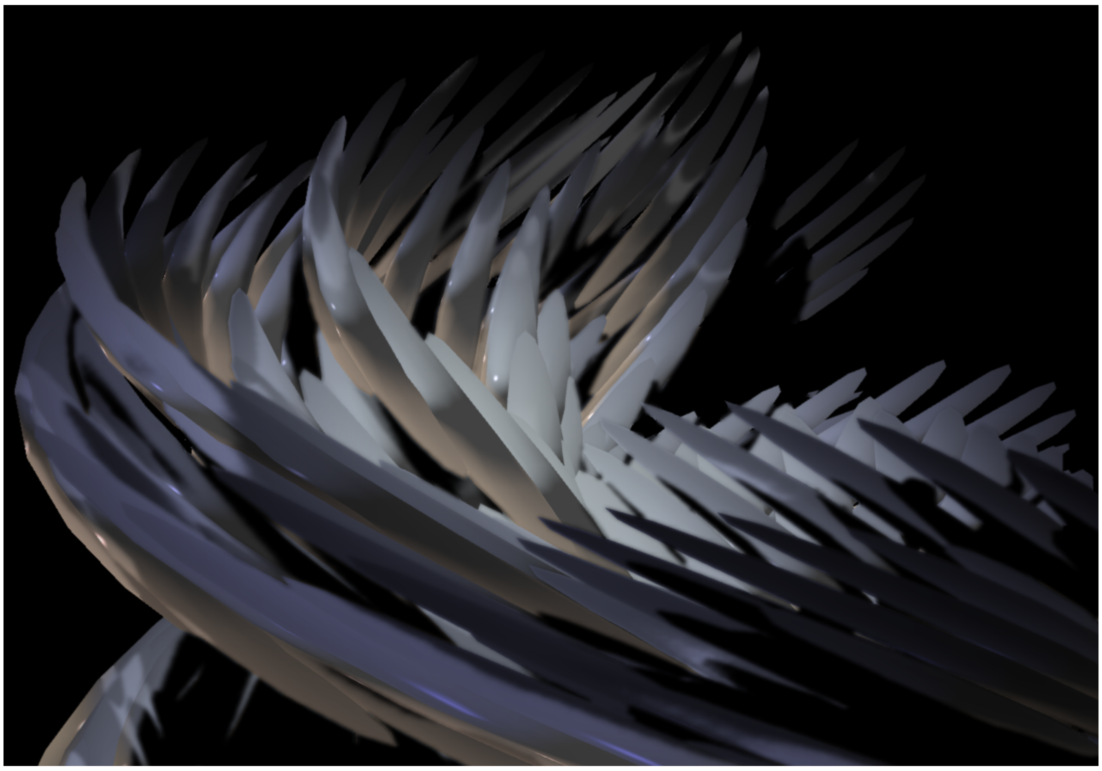


Figure 4.12: *Twist*, R. Hoetzlein (2010). Surfaces generated by twisting tubes, rendered with glass-like material properties.

4.6 Conclusions

LUNA, a visual dataflow language for procedural modeling, is introduced as an efficient and flexible system with interactive output. A benefit of our language model is that it allows for different interpretations while presenting a generic solution to the relationship between functional models and scene graphs. LUNA is defined as a procedure graph that makes generative changes to subsets of a scene graph, adding or modifying scene nodes as needed. The interface permits novice users to quickly prototype objects without the need to understand detailed controls, and the performance of the system enables direct feedback on the structure, appearance, and surfacing of complex models[3]. LUNA is tested and evaluated through comparisons to Houdini using a new model for procedural systems, the woven sphere reference model.

There are several limitations still present in LUNA that could be address in the future. First, although individual objects may contain hierarchies (e.g. JointSet, Tree), the scene graph output is currently a list of scene nodes rather than a hierarchy. A more complete model would implement a scene graph hierarchy for each procedural node, allowing for multiple procedure node output types and more complex object relationships. For example, in the future character models should be possible using named scene nodes with joint relationship handled internally.

Several performance improvements are still possible. At present, LUNA takes advantage of scene graph performance primarily by grouping material and texture render state, and by intelligent updating of hardware geometry buffers. Although the necessary information is present in LUNA, spatial culling and other acceleration structures have not yet been implemented. Dynamic overlapping geometry, such the woven sphere reference model, present a difficult performance challenge which can only be addressed through better polygon-level rendering. However, it should be possible to accelerate disjoint objects using known techniques since some objects are naturally separated both spatially and conceptually as high level entities by the procedure graph.

While LUNA does not yet contain a detailed vocabulary for physical simulation, crowds, or character modeling, the structure of the language should be make it relatively

[3]A demo version of LUNA is currently available online at http://www.rchoetzlein.com/luna/

easy for communities to add these in the future. To my knowledge, LUNA is the first system to allow for high quality, deferred shading of complex procedural models with interactive feedback.

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.

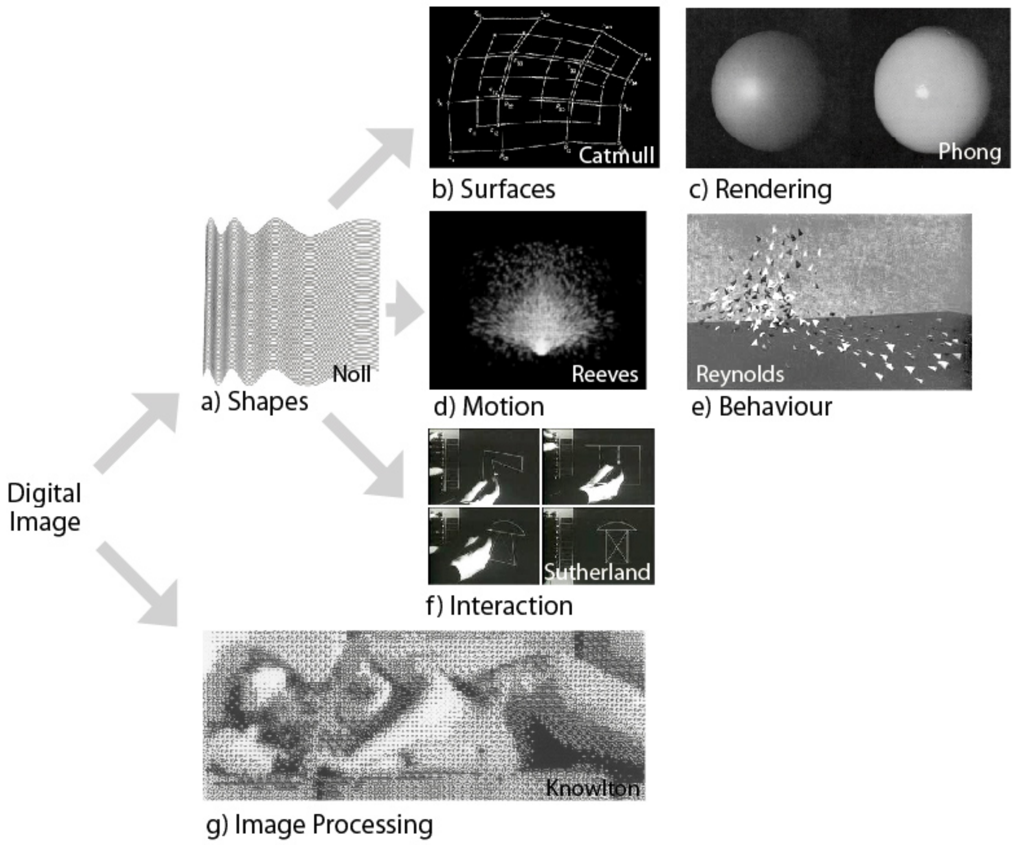


Figure 5.1: Early history of the digital image, including a) shapes (Michael Noll), b) surfaces (Ed Catmull), c) rendering (Bui Phong), d) motion (William Reeves), e) behavior (Craig Reynolds), f) interaction (Ivan Sutherland), and g) image processing (Ken Knowlton).

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 characters 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.

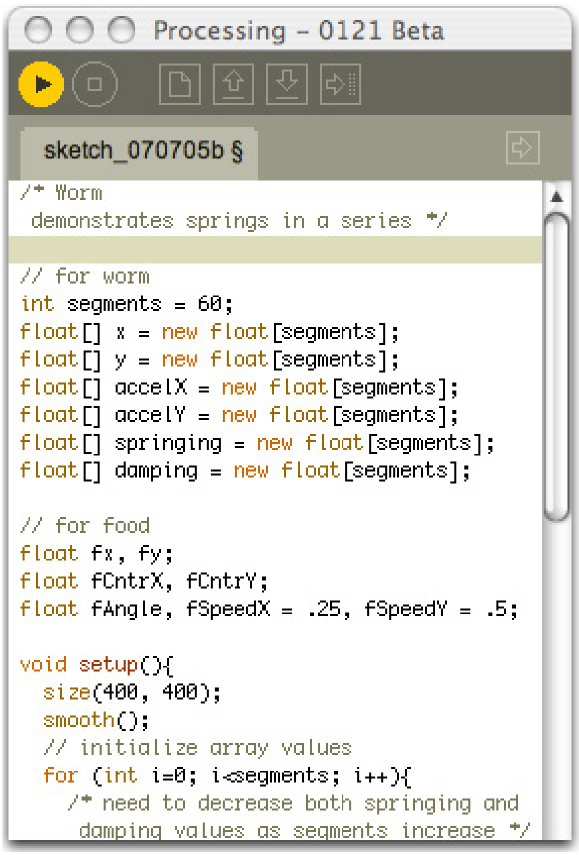
> "As an animation language it provided instructions for several motion effects as well as for camera control. Knowlton had initially hoped that artists would learn the language to program their own movies, but he came to realize that they usually wanted to create something the language could not facilitate, 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.



```
/* Worm
 demonstrates springs in a series */

// for worm
int segments = 60;
float[] x = new float[segments];
float[] y = new float[segments];
float[] accelX = new float[segments];
float[] accelY = new float[segments];
float[] springing = new float[segments];
float[] damping = new float[segments];

// for food
float fx, fy;
float fCntrX, fCntrY;
float fAngle, fSpeedX = .25, fSpeedY = .5;

void setup(){
  size(400, 400);
  smooth();
  // initialize array values
  for (int i=0; i<segments; i++){
    /* need to decrease both springing and
     damping values as segments increase */
```

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

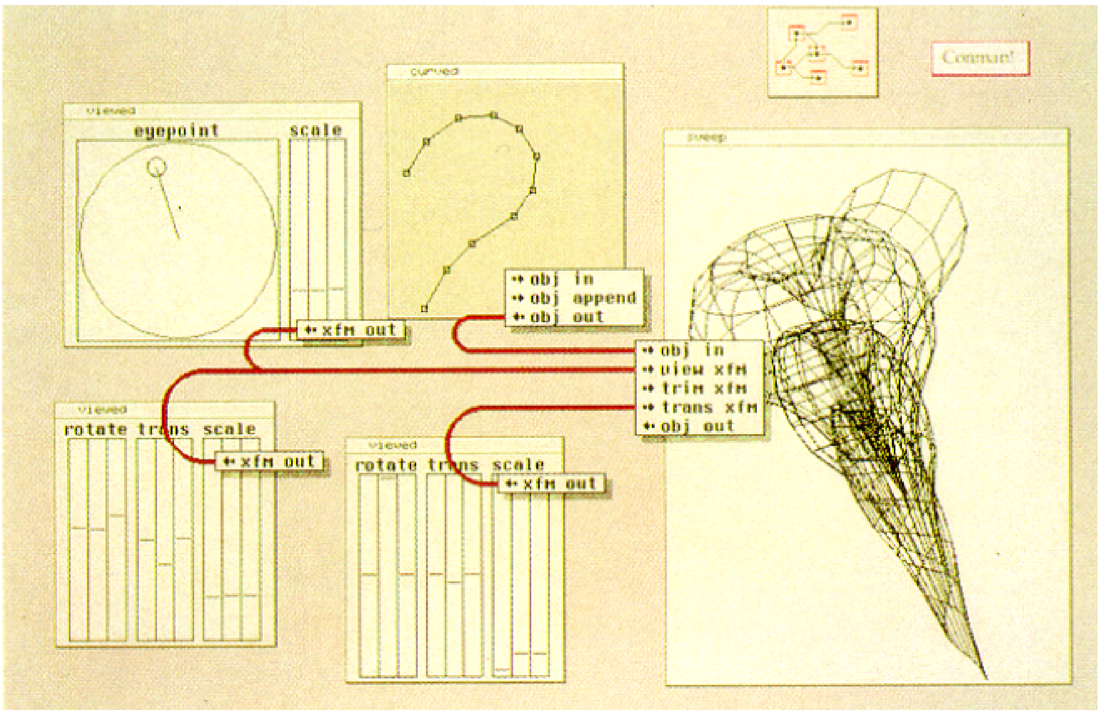


Figure 5.3: Con Man, an early visual language for procedural graphics. Image copyright Paul Haeberli (c) 1988.

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].

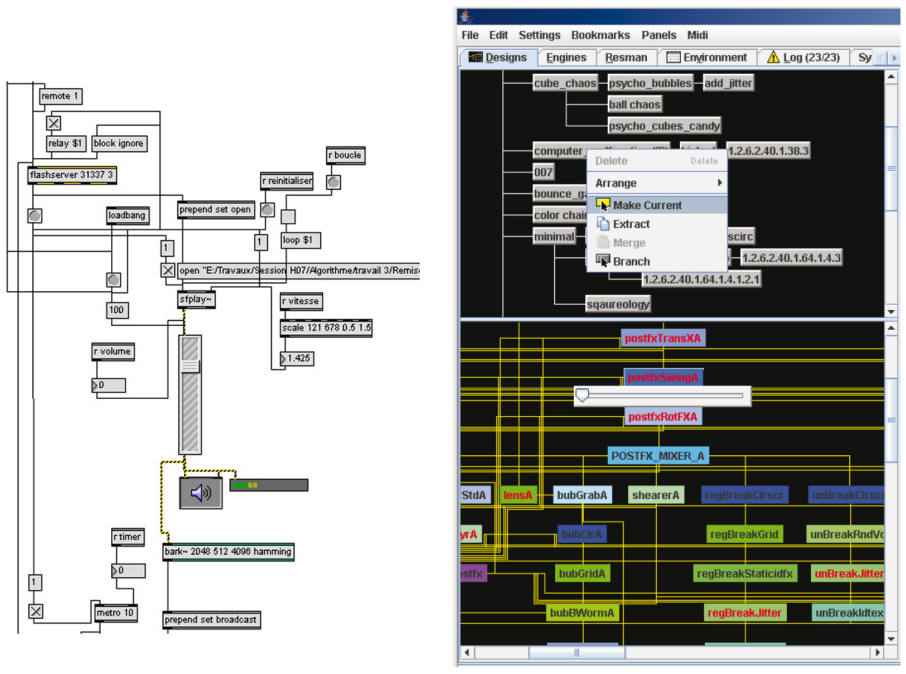
The idea of the media artist as programmer is an on-going trend, so the interest of artists in visual dataflow languages is relatively new. Interactive artworks, such as *A-Volve* by Sommerer & Mignonneau (1994), use text-based programming to achieve a particular, dynamic relationship between the art 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,

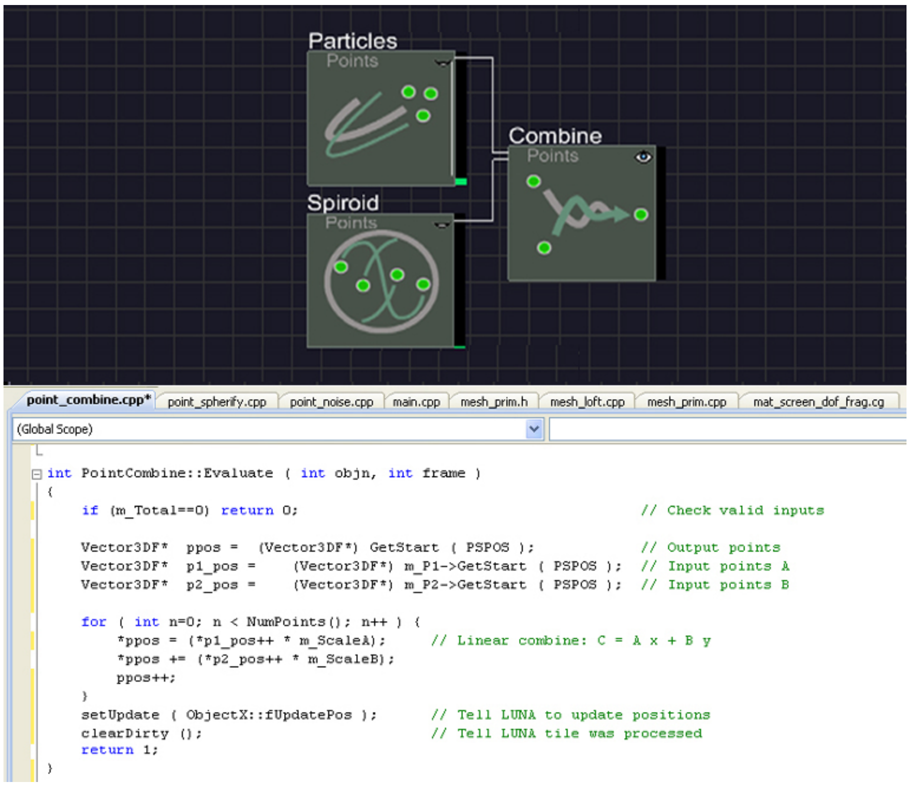
[4]Pascal Müller, Stefan Müller Arisona, Simon Schubiger-Banz and Matthias Specht, from ETH Zurich, University of Fribourg, and University of Zurich

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.



```
int PointCombine::Evaluate ( int objn, int frame )
{
    if (m_Total==0) return 0;                                        // Check valid inputs

    Vector3DF*  ppos =  (Vector3DF*) GetStart ( PSPOS );             // Output points
    Vector3DF*  p1_pos =    (Vector3DF*) m_P1->GetStart ( PSPOS );  // Input points A
    Vector3DF*  p2_pos =    (Vector3DF*) m_P2->GetStart ( PSPOS );  // Input points B

    for ( int n=0; n < NumPoints(); n++ ) {
        *ppos = (*p1_pos++ * m_ScaleA);      // Linear combine: C = A x + B y
        *ppos += (*p2_pos++ * m_ScaleB);
        ppos++;
    }
    setUpdate ( ObjectX::fUpdatePos );       // Tell LUNA to update positions
    clearDirty ();                           // Tell LUNA tile was processed
    return 1;
}
```

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

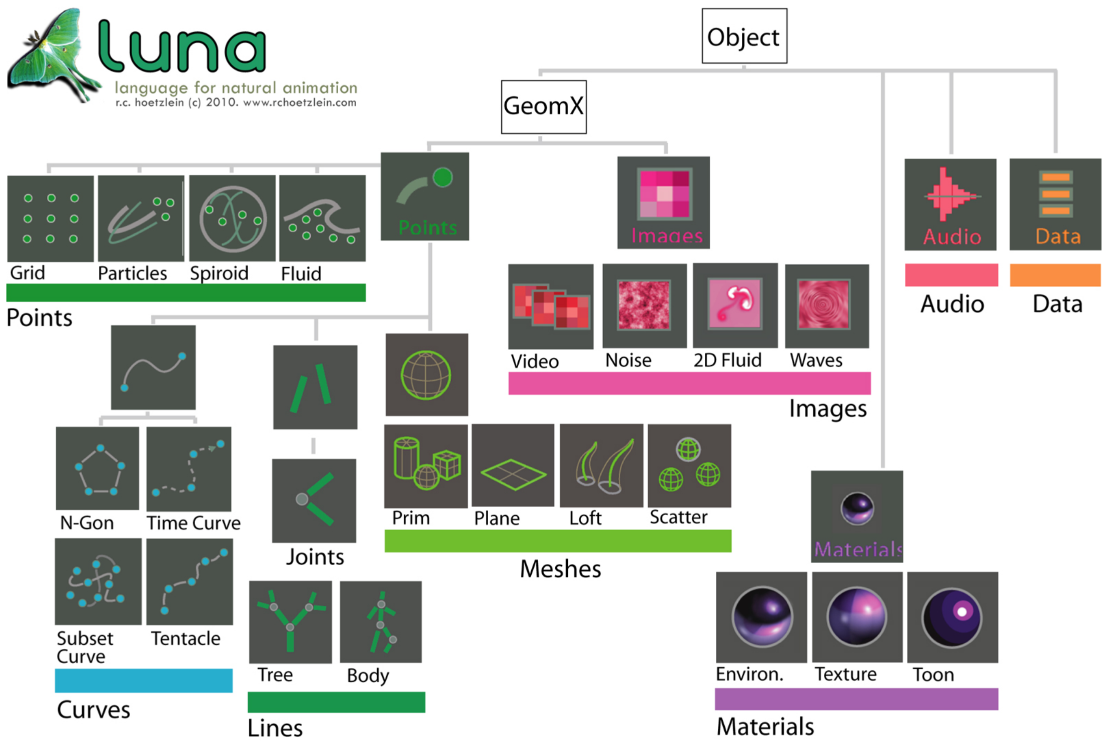


Figure 5.6: Taxonomy of classes and currently implemented nodes in LUNA.

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

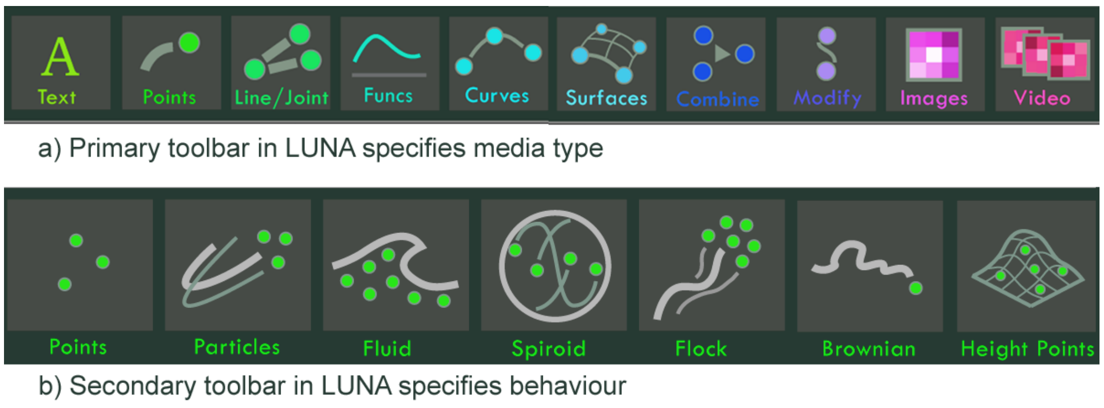


Figure 5.7

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.

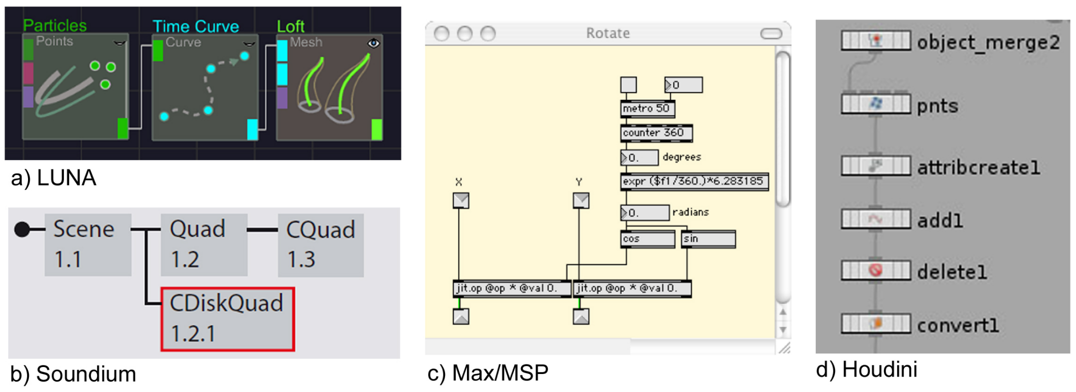


Figure 5.8: Detailed view of connections in various visual design platforms. Colored tabs in LUNA indicate changes in the type of media as it flow through the system.

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).

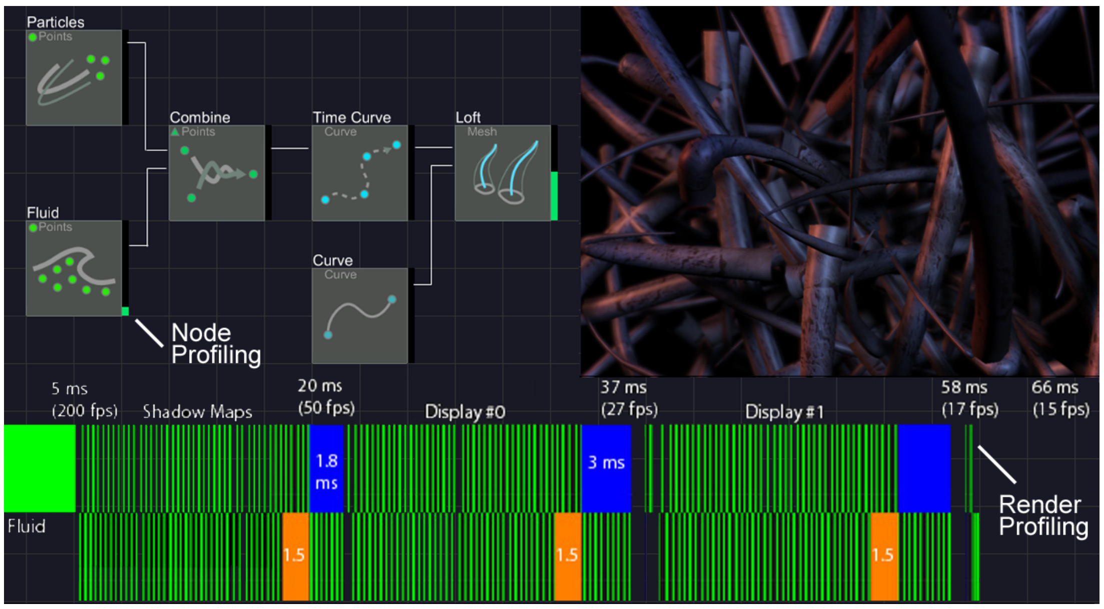


Figure 5.9: Performance profiling in LUNA. Node profiling shows CPU/GPU resources used for each object in the user graph. Render profiling shows the resources used by the CPU for computation and GPU for rendering.

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.

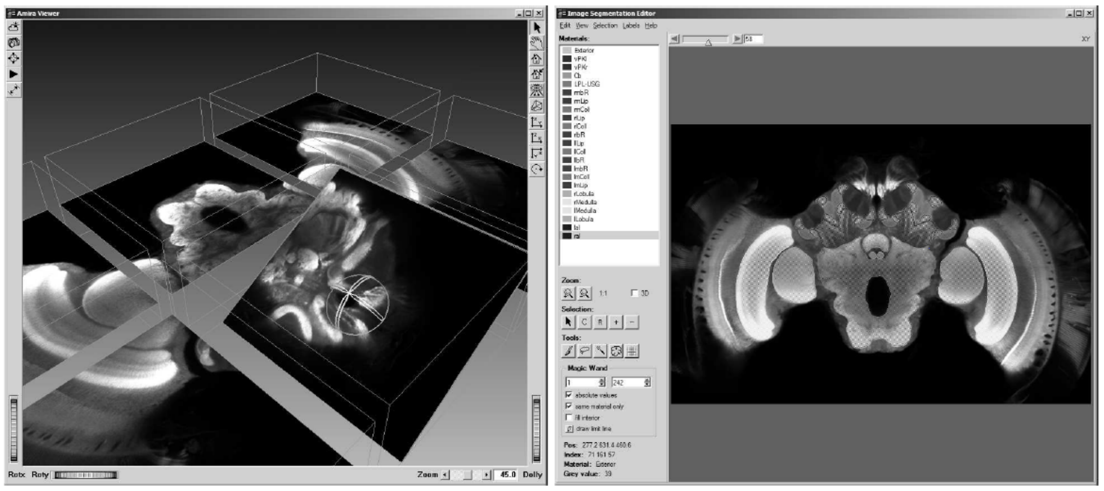


Figure 5.10: High-performance computing in Amira, a system dedicated to interactive manipulation and visualization of scientific datasets. Image copyright Mercury Computer Systems SA (c) 2010, based on the work of [Stalling et al., 2005]

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.

Chapter 6

Structure in Dynamic Media

6.1 Overview

The production of image from code was one of the first major challenges to early computer artists and scientists. Once this barrier was overcome, artists began to explore a range of topics exposed by the use of digital technology. Languages for graphics focused on the production of the image using complex algorithms to produce dynamic content. Low-level languages such as Processing and Max/MSP provide a means to create basic shapes and forms while leaving the behavior to be determined by an artist-programmer. High level environments such as Houdini, Maya, and Xfrog offer a range of structures including particles, fluids, and L-systems (trees). LUNA was designed specifically to allow artists to remix different dynamic, autonomous structures and behaviors in an interactive environment.

Jörg Schirra observes a difference between the structure of image as a set of pixels and the structures found *within* the image, that is its content [Schirra, 2005, p. 96] [1]. Abstract structures for content include basic primitives such as lines and curves. Harold Cohen proposes that the content of an image may be defined through art-making as a set of underlying rule-based activities which derive from the nature of cognition. Cohen's AARON, software designed to draw scenes of computer generated characters, is a system for determining the placement of line based on a set of rules, and therefore informs the content of the image through such rules [Cohen, 1979].

This chapter is concerned with several dimensions related to the content of the image in media art. One dimension which will be explored is the nature of the rules which determine the placement of forms and shapes. Structure, in three dimensions, creates the appearance of surface and volume, so another axis for creative freedom resides in the materiality of the surface. Finally, the content of an image need not be generated by a set of rules but might also come from hand drawn figures, or content borrowed from other sources, i.e. photography or collage. This use of content from the real world, whether it is drawn by an artist or taken from nature, is yet another choice available to the artist. The ideal scenario for creative tools for media artists would allow the exploration of each of these ideas together. The objects currently found in LUNA were chosen to represent points along these dimensions to show that this kind of bricolage is possible at an abstract level in a digital environment.

[1] A structuralist argument may also define content as the values of each image pixel, but this perspective avoids the organization of pixels into groups such as lines, curves and forms which I consider here.

6.2 Motion, Dynamics, Autonomy

The rules for determining the placement, style, and orientation of lines, curves, and forms can be exceedingly "complex". Lev Manovich uses the word complex to describe his experience of contemporary digital media in contrast to abstract minimalism of the preceding era:

> "What is important is that having realized the limits of linear top-down models and reductionism, we are prepared to embrace a very different approach, one that looks at complexity not as a nuisance which needs to be quickly reduced to simple elements and rules, but instead as the source of life.. I am now finally ready to name the larger paradigm I see behind the visual diversity of this practice. This paradigm is complexity [Manovich, 2007, p. 346]."

Setting up a duality between movements of abstraction and complexity in art creates several problems, however. First, this opposition avoids the fact that work of the Russian Constructivists, such as Malevich's Supremus No. 18 (Figure 6.1), are both abstract and complex . This image, showing a detailed and subtle arrangement of lines and rectangles, was created after Malevich's reductionist period [Gray, 1962, p. 166], so Malevich was aware of the reductive possibilities of abstract forms while simultaneously engaging in them as a multitude. Second, the source of the complex image in new media extends in part from early experiments in artificial life. The scientists who explored this field, including Alan Turing, John Von Neumann and John Conway, found that very simple rules could lead to complicated structures, thus complexity may reside only at a particular level. The problem is that the term complexity can be applied to such a wide range of human experiences despite the fact that in some cases very simple or

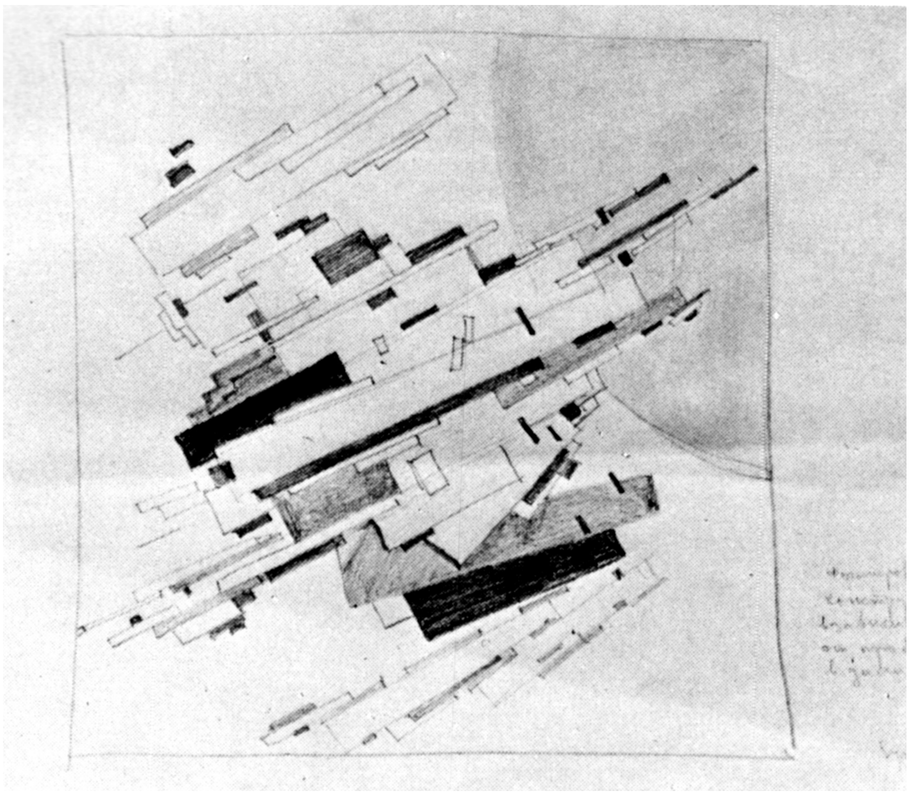


Figure 6.1: Supremus No. 18, Kasimir Malevich, 1916-1917

abstract ideas underlying these experiences are present simultaneously, and that certain aspects of an image may be complex (e.g. placement) while others may not (e.g. form). Art of nearly any period may be described as complex in some sense. In what *way* is something complex?

A number of terms have been used by media artists to more precisely define types of digital art, which may be viewed as an evolving taxonomy. 'Functional' in computer science refers to that which is based on a set of rules, but may conflict with "serving a useful purpose." 'Autonomous' forms are capable of independent action, such as self-motion and self-reproduction, while 'Generative' art is capable of creating new structures and may or may not also be autonomous. ‘Organic’ forms are higher order structures which grow, change, or appear like those in nature. ‘Behavioral’ objects may be said to *act* on other objects, which has a connotation of human cognition. The term ‘Computable’ expresses that which can be decided by rules using a machine, and appears to cover all of these domains of media art, but has the essential drawback that we do not know if all organic, behavioral or autonomous objects in nature are also computable. I prefer the term ‘Dynamic’, as it expresses the idea that something is in *motion.* Even if the image itself is static, all media art is dynamic in the sense that something in the external world, a device, computer or kinetic object, is or was *changing*, that is acting on its own in the world. This concept of externalized change is new to art, since prior to the 20th century most art was created by ideas expressed through the human body.

To consider the content of media art in more detail involves looking at ideas which have motivated it. One possible starting point can be found in the science of *dynamic systems.* Edward Lorentz, Henri Poincare, and Benoit Mandelbrot explored iterative and non-linear systems in nature, leading to fractal and chaos theory. James Gleick collects and summarizes these ideas in the book *Chaos* [Gleick, 1987]. The term chaos itself is interesting as it expresses that which is beyond formal understanding. In a literal sense such systems can only be approximated with numerical methods since they general defy analytic solutions:

“Chaos is the irregular, unpredictable behavior of deterministic, nonlinear dynamical systems.” Roderick Jensen, Yale University [Gleick, 1987, p. 306].

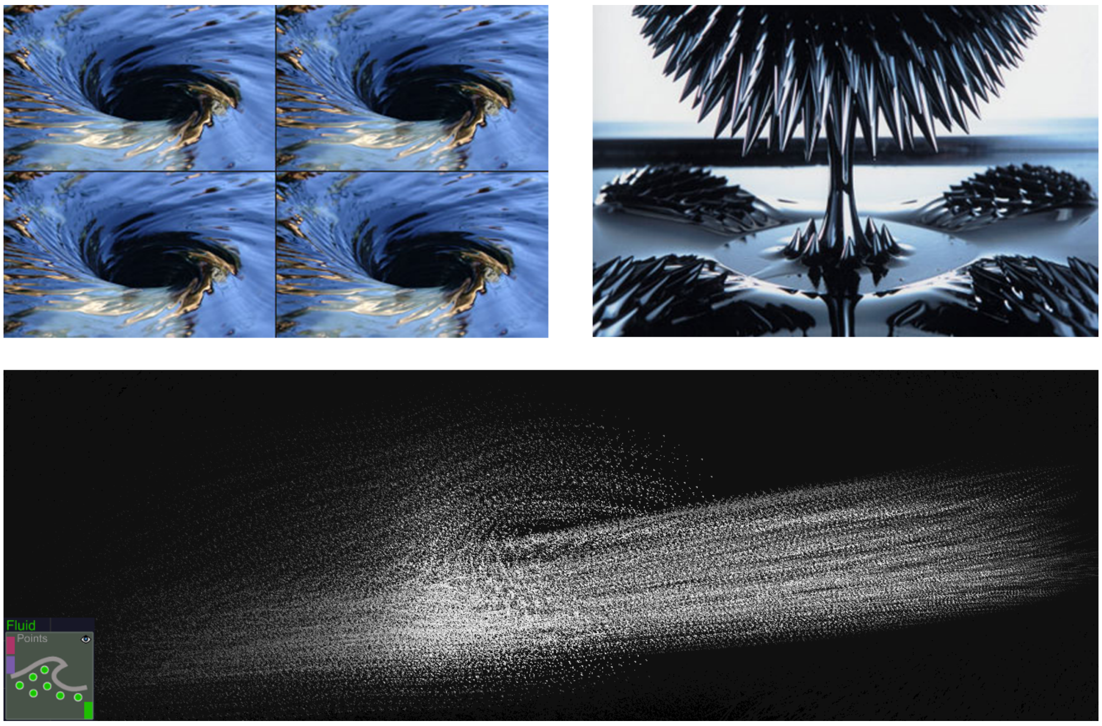


Figure 6.2: Fluid systems generate chaotic motions with vortices by Ned Kahn in *Basin of Attraction* (c) 1989, Artpark, Lewiston New York (top left), and *Protrude, Flow* by Sachiko Kodama (c) 2000, Tokyo (top right). The bottom image shows fluid waves created by the author by merging the Fluids v.2 software with LUNA [Hoetzlein, 2010].

A particular kind of non-linear dynamic system is the motion of fluids. While the science of fluids dates back to ancient history (Archimedes studied fluid mechanics), artists have explored irregular patterns in fluids as a form of beauty. Ned Khan uses architectural and sculptural elements, in a systematic way, to visually expose the flow of real wind and water [Mather, 2006]. Sachiko Kodama and Minako Takeno look at the dynamics of ferrofluids[2] in the work *Protrude, Flow*, 2000. The most interesting aspect of these works is the motion created by non-linear systems. LUNA allows artists to explore this motion through simulated fluids, a component in the system which

[2]Ferro-fluids are liquids with magnetic particles suspended in them

embodies the rules that govern fluid dynamics, Figure 6.2. The fact that these systems are chaotic invites creative experimentation since the variety of motions they produce can never be exhausted, an experience described by Kant as the 'mathematical sublime'.

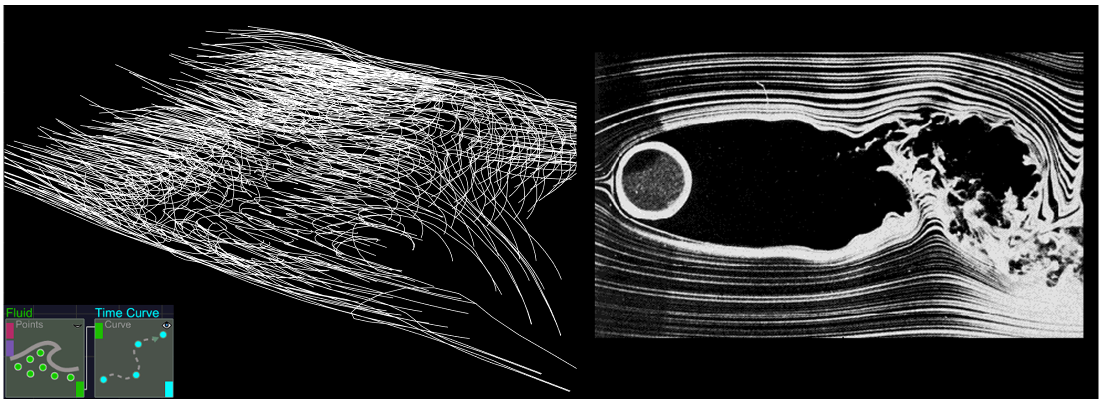


Figure 6.3: Fluid flow visualizations create by tracking the motion of small fluid volumes over time. The left image shows the Time Curve object being used to follow particles in a point-based fluid simulation. The right image shows real fluid lines imaged by taking time-elapsed photographs of helium bubbles suspended in a fluid.

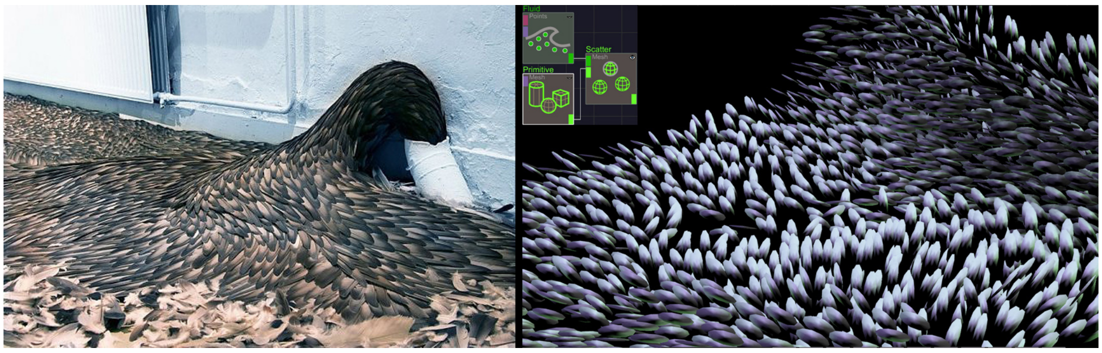


Figure 6.4: *Sluice* (c) 2009. Kate MccGwire, http://www.katemccgwire.com. Feathers assembled to follow a fluid-like path through a real space. *Digital Sluice* (c) 2010. R. Hoetzlein. Virtual feathers arranged to follow the paths of a simulated fluid.

From a creative perspective, what is interesting about LUNA is its ability to easily combine fluids with other systems, to interact with or reveal this motion in unique ways.

In 1961, Asher Shapiro developed a thirty nine video series on the mechanics of fluids for the National Committee for Fluid Mechanics Films, an educational institution to promote understanding of science [Shapiro, 1961]. One particular film, Flow Visualization, shows how helium bubbles suspended in a fluid can be used to *see* how the fluid moves over time, Figure 6.3. In a similar way, connecting the Fluid node to a Time Curve node allows the path of wave to be revealed. Kate MccGwire, in *Sluice* (2009), arranges feathers along the possible path of fluids in city landscapes. Although many software frameworks only allow fluids to appear as liquid surfaces, this kind of aesthetic is possible in LUNA by connecting the Fluid node, and a mesh describing any object, into a Scatter node which places the object at each point in the fluid. While LUNA is implicitly a simulation, these kinds of experiments can be performed with literally five or fewer mouse motions used to connect the tiles. These rearrangements are similar to the work of the bricoleur, since they consist of a playful arrangement of the *connections* between found objects (found by the user), except that the objects themselves are dynamic systems.

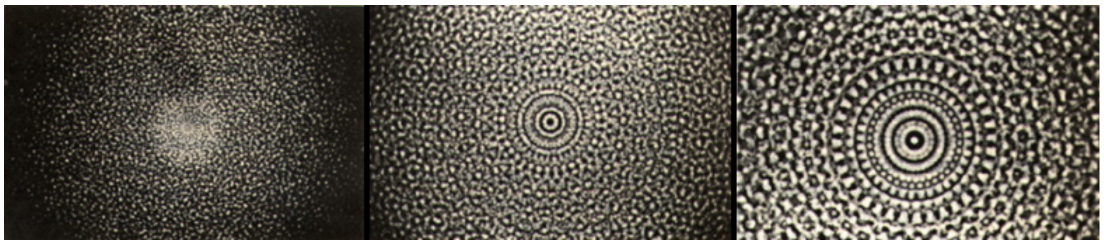


Figure 6.5: *Lapis*, John Whitney (c) 1967.

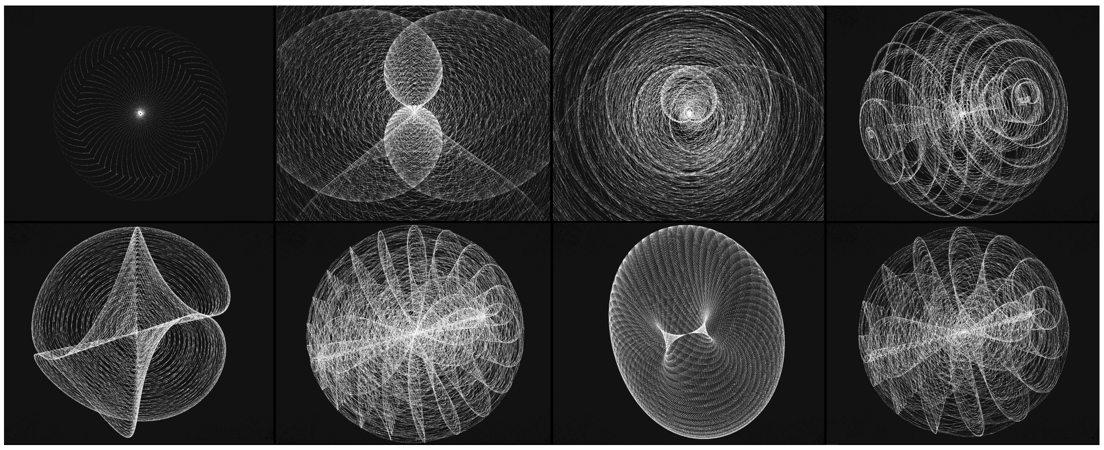


Figure 6.6: *Spira*, R. Hoetzlein (c) 2010. Created in LUNA.

Other disciplines have looked at motion in different ways. In music, the study of sound waves lead to ideas of frequency and phase in oscillations, while oscillations (tones) taken together form harmonics. John Whitney began looking at these dynamics visually using an oscilloscope, an instrument which measures waves in electronic signals [Russet and Starr, 1976]. Eventually, through a grant with IBM, Whitney found that computers could be used to directly study these patterns, Figure 6.5. In figure 6.6, images entitled *Spira* were created with LUNA using the Sinusoid object implementing a similar kind of behavior. The Sinusoid object was created to show that these dynamics are possible using the same grammar as physical systems such as fluids, while a more complete language for harmonics would require a much larger set of other nodes which could be added to the system in the future. Square waves, sin waves, saw tooth waves, and filters which are found in music synthesis, might be added to the Function and Audio media types in LUNA.

One of the ideas present in structuralism is that hidden rules may determine the form of objects. This idea is common in the sciences, where scientists try to determine the structure of the world by looking at nature. The whole of science may, in fact, be summarized as the discovery of rules from the observation of forms [Peirce, 1866]. One example is the process of self-replication, observed as a natural phenomenon and considered by René Descarte, Samuel Buttler, and William Palley as a concept which could possibly be applied to machines. However, it was John von Neumann that first began to developed specific instructions for machines that could self-reproduce with out relying on parts outside the environment in which they existed [Levy, 1992, p. 44]. The first true self-reproducing machine of this kind was demonstrated in John Conway's Game of Life by William Gosper [Wainwright, 1974] [Levy, 1992, p. 56-57], thus showing in a dramatic way that natural rules can be deduced and reapplied formally. Many other natural processes can also be understood through the abstraction of rules. Craig Reynolds created a system for simulating the motion of birds [Reynolds, 1987], while William Reeves developed a way of representing fuzzy objects like fire and clouds [Reeves, 1983]. In LUNA, Reeves fuzzy objects are implemented as a particle system tile. Yet, in the sciences, there remains a great deal of uncertainty as to whether the whole of nature can be understood through the deduction of hidden rules [Kuhn, 1962]. This presents a challenge to structuralism as well, as it is not clear whether all image forms can be explained as a set of rule-generating behaviors? If so, then all that remains is to discover more of these rules. If not, what about images makes them unique? This may be more of a question regarding natural reality, since any image is a picture of some reality. This issue will be revisited in the next section.

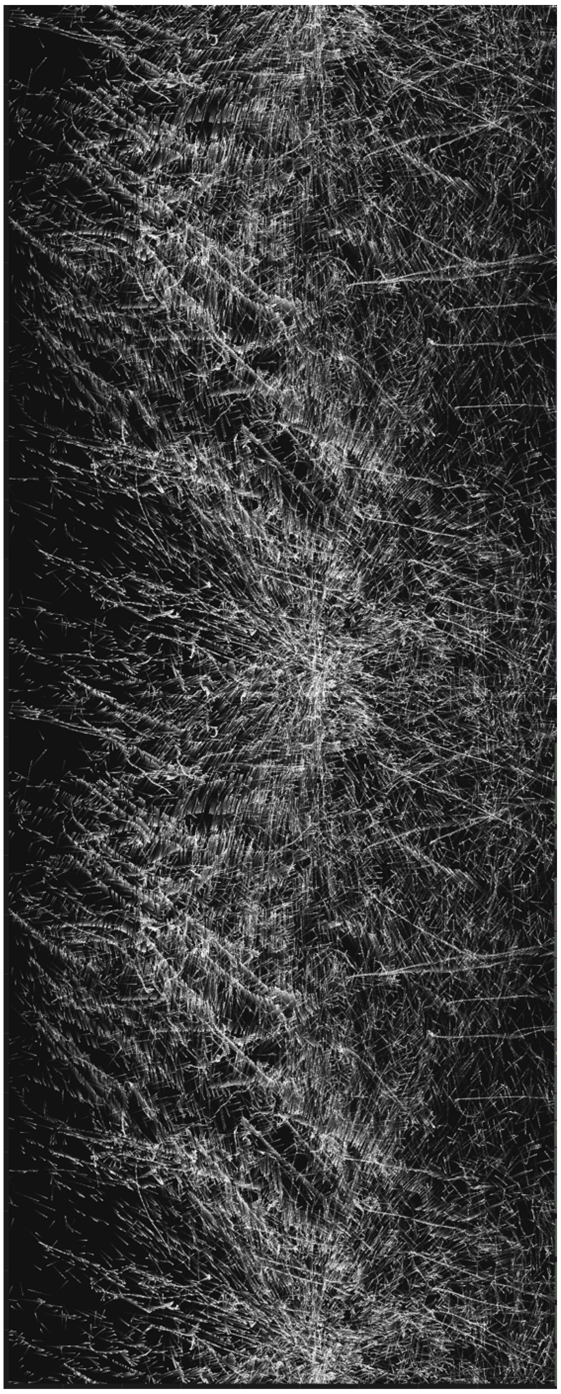


Figure 6.7

Overall, the Point tiles in LUNA[3] were selected to represent a cross-section of rule-based behaviors in fields which have been explored by artists and engineers in the past. These tiles are considered the beginnings of a visual grammar that play with dynamic systems themselves, allowing artists to mix and match different rule-based motions. The total range of behaviors in this language-of-systems is thus greater than any one model. A node which embodies the idea of mixing dynamics is the Combine node. The entire internal code for the Combine node can be found in Figure 5.5 (see chapter five). In one respect, this node is simple as it represents the linear combination (weighted addition) of two points in space. Yet the merging of all points in two moving systems creates a new kind of dynamic itself, a new aesthetic. Figure 6.7 shows the result of combining a Fluid system with a Particle system. Whereas programming in text-based languages changes the rules "underneath" the system, a visual grammar changes the rules "over" the systems. In other languages, such as Houdini, Max/MSP, or Processing, mathematical knowledge would be required to create this combination by writing an expression. This idea of transforming dynamic systems as malleable, whole entities is an interesting way of re-conceptualizing the process of making media art.

New behaviors are revealed to the artist by playing with the system. The *spherify* node takes the points of any dynamic system or object, and maps it to a sphere. Fluid-Spherify results in points on a sphere whose motion, *constrained* to a sphere, resembles a fluid. Grid-Spherify maps the points of a grid onto the points of a sphere, creating a

[3]These include several implemented nodes, Particles, Fluids, Spiroid, Combine, Scatter and some not yet implemented ones, Flocking, Brownian, and Surface Points.

kind of crystalline lattice of points. Tree-Spherify takes a tree structure and appears to *press* it against the surface of a sphere. These operations begin to take on the richness and semantic complexity of sculptural processes performed on real materials, but which have been translated into the unique language of computer simulated behaviors. The aspect of LUNA which makes this particularly suited to creative workflows for media artists is the ability to quickly connect tiles without having to remember, program, or 'lookup' the syntax of the language, to interact with these changes in structure.

6.3 Structure and Surface

While the study of simple systems has been greatly enhanced by the ability of artists and scientists to generate images from a set of rules, there are many kinds of systems to be explored. As early experimenters discovered even a single formula, such as those for fluid equations or fractals, can take a lifetime to explore. One aspect that these non-linear systems share is the observation that structure *emerges* from them; there is nothing needed other than the original formula [Gleick, 1987, p. 23-24]. In essence, these systems have no explicit parts; but what do we make of objects that do have parts, such as trees, molecules, and crystals?

What is structure? While this may be impossible to answer concretely, several common ideas emerge. In the natural world, when we observe trees we find that any species is reproduced in a similar but not identical way. We now know that DNA is partly responsible for remembering the form of a particular species. In computer graphics, to generate three dimensional structures, one of the simplest systems is the Lindenmayer system (L-system), a set of typographical rules that transforms one string into another, to *grow* trees similar in form to living ones. In computational theory, Alan Turing and John von Neuman explored the idea of an infinite tape, or universal machine which can encode any algorithm by means of an infinite tape onto which symbols are printed. Each of these ideas share the concept of *memory*, a continuity of experience, a relation or dependency between parts that takes place over time.

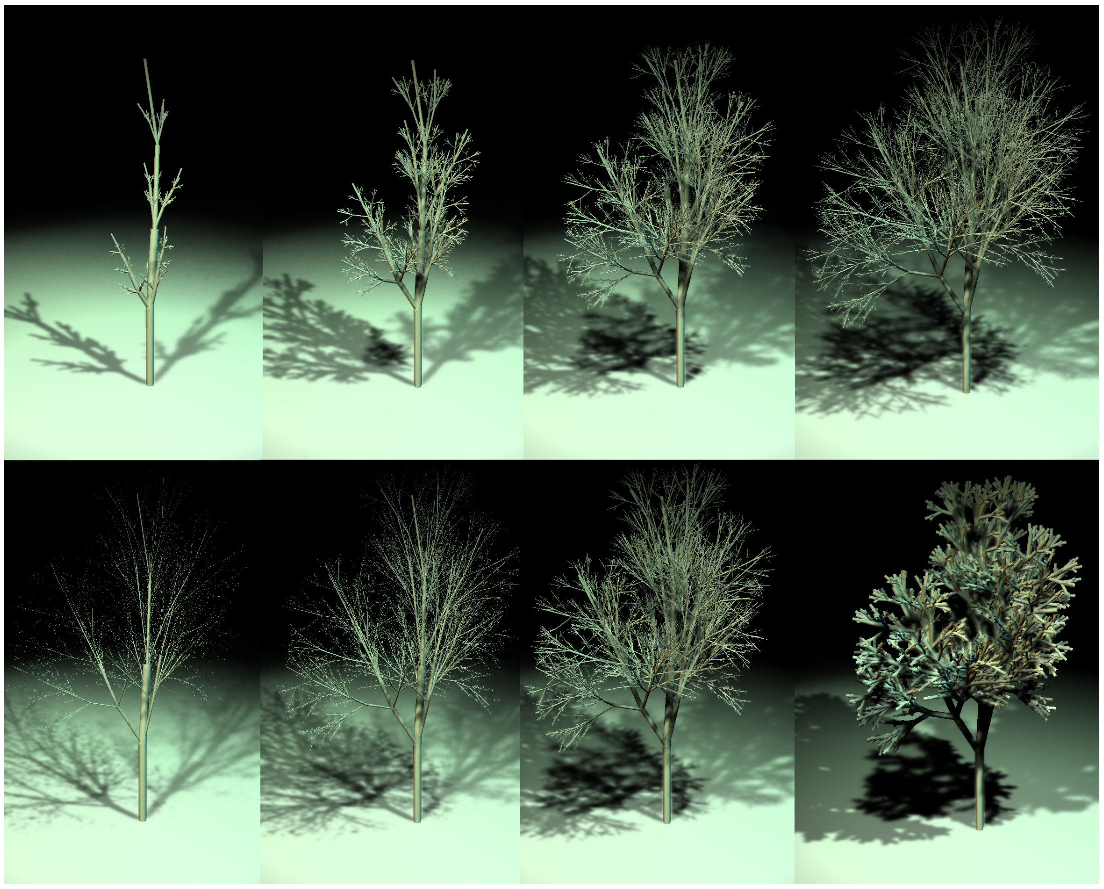


Figure 6.8

What does it mean for an artist to have freedom along the dimension of structure? A simple approach can be derived from the embodiment of a form which is remembered from one instantiation to the next, as a seed produces a unique tree which is also a member of its species. An example of this kind of memory is shown in Figure 6.8, modeled and rendered with LUNA. In this example, parameters of the tree are modified while the rules which produce the particular angles between branches are stored so that each change appears to be the same tree in different stages of growth (top row). In the bottom row, this same kind of memory can be applied to other features, resulting in a kind of growth not found in nature, i.e. the outer branches thicken while maintaining their overall length. The structure retains its shape as various parameters affecting its outcome are modified.

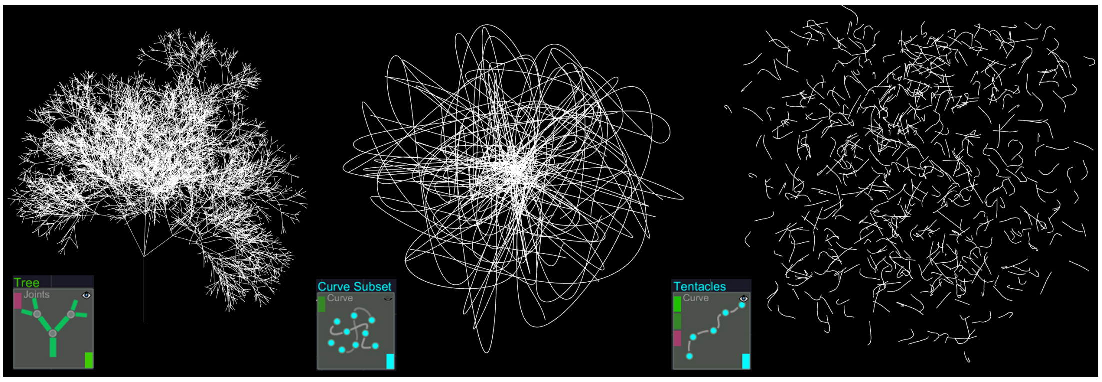


Figure 6.9: Three different structural systems: a) Trees, b) Curve subsets, c) Tentacles

Another kind of freedom in exploring structure extends from selecting different kinds of structures. A tree is an example in which each branch has a particular relationship to its parent branch, continuing recursively back to the trunk. In chemistry, molecules can attach to each other without this hierarchical restriction. In LUNA, Curve Subsets and

Tentacles are two other examples of structures chosen for implementation because they both require only a set of points as input, Figure 6.9. Curve Subsets randomly selects and remembers a single set of points and connects these with curves. Tentacles chooses a set of points and then uses a spring system to randomly *reach out* to a different set of points. Many other structures also exist which could be included in LUNA, such as crystals, molecules, articulated bodies, or other arbitrarily generated shapes. It would be interesting to consider whether there are generalizations that could embody the abstract concept of remembered relations between shapes.[4]

What causes an artist to explore a particular structure over another? Why, for example, did I choose one recognizable structure in Figure 6.9, the tree, and two unrecognizable ones to develop in LUNA? The psychology of vision is of importance to the artist since the artist may seek to uphold or upset that vision either consciously or unconsciously. Formalists, such as Wolflinn, considered the psychological development of vision to be central to understand art in different cultures, proposing that there were universal structures in the development of mankind which parallel the development of the individual [Hatt and Klonk, 1992]. This is one way to explain why different forms are of interest in art over time, but does not easily explain how they differ across cultures or periods.

[4]This could potentially allow one to create unpredictable structures with definite forms. While genetic algorithms are another abstraction of memory, the idea here is a way of iteratively producing structures from physical constraints, rather than generating structures from evolving genotypes based on physical performance. The interesting question is how geometric constraints relate to physical structure in an abstract sense.

In the 20th century Saussure explores perception through language, that the memory of a *sign* creates meaning when an image, icon or word (signifier) evokes an idea (signifie). In the example above, the tree-form is an abstract signifier than generates the idea of tree. Interestingly, it is not a word or icon but an abstract simulation of tree that evokes the concept. Artists are interested in how forms, icons and signs evoke meaning as this is how ideas are conveyed through art. This use of forms is not necessarily a literal process, that is artists do not always wish to convey the concept of a tree by simply modeling or showing a tree. Yet this is often the assumption made in authoring tools focused on building virtual worlds. How do we design media frameworks to support the indirect development of meaning? One way in which this is resolved through LUNA is to reveal the internal structures of objects at each stage, making the different layers of the object available to the artist. Rather than provide a completed form, the artist should be able to navigate the assumptions that went into constructing a particular model, as shown in the next example.

The rendering in Figure 6.10c creates an illusion of space, and delineates the surface, form and texture in detail using LUNA's ability to render dynamic scenes in real-time, with shadows, texturing and depth of field (features typically found only in game engines). The same forms may be presented in LUNA in other ways, as stylized shapes in Figure 6.10b or simply as curves in 6.10a. The presentation of structure or surface is a choice available at each stage in the construction of the object. To facilitate this, all

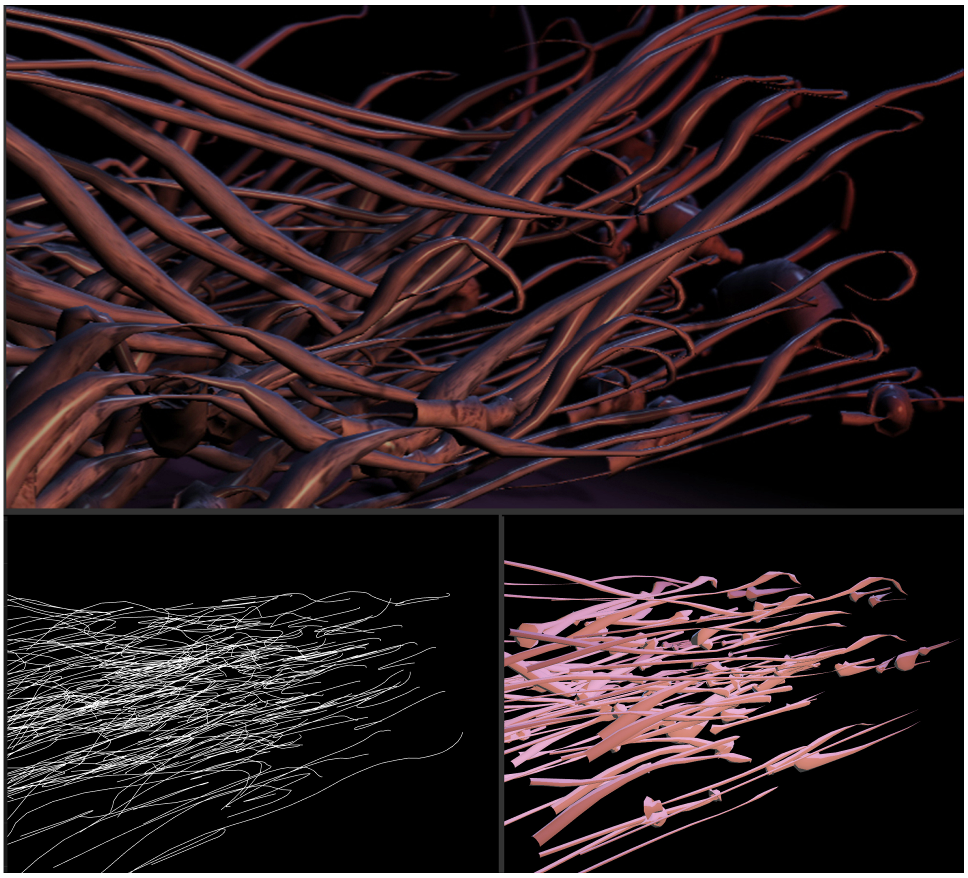


Figure 6.10: Loft surfaces create with different choices in appearance including inner structure, shapes, and illuminated surfaces.

LUNA nodes have an 'eye' icon that allows the user to enable or disable the intermediate appearances of the object (a similar feature is found in Houdini).

Surface realism was a major area of research while implementing LUNA, since one of the goals was to provide not only the option of realism, but also the ability to control appearance interactively. While game technologies creates a high degree of realism in real-time, the ability to modify or author different looks is typically not part of their interactive workflow and is often performed using offline tools. Therefore, LUNA borrows the idea of material graphs from tools like Maya and combines this with real-time rendering techniques found in gaming. From a graphics perspective, LUNA makes no distinction between interactive appearance editing and high-quality rendering. Thus, artists do not need to wait for image results to experiment with different looks (in the motion picture industry this is the emerging field of pre-visualization). Artists are also free to author their own appearances by writing Cg shaders which plug into LUNA.[5]

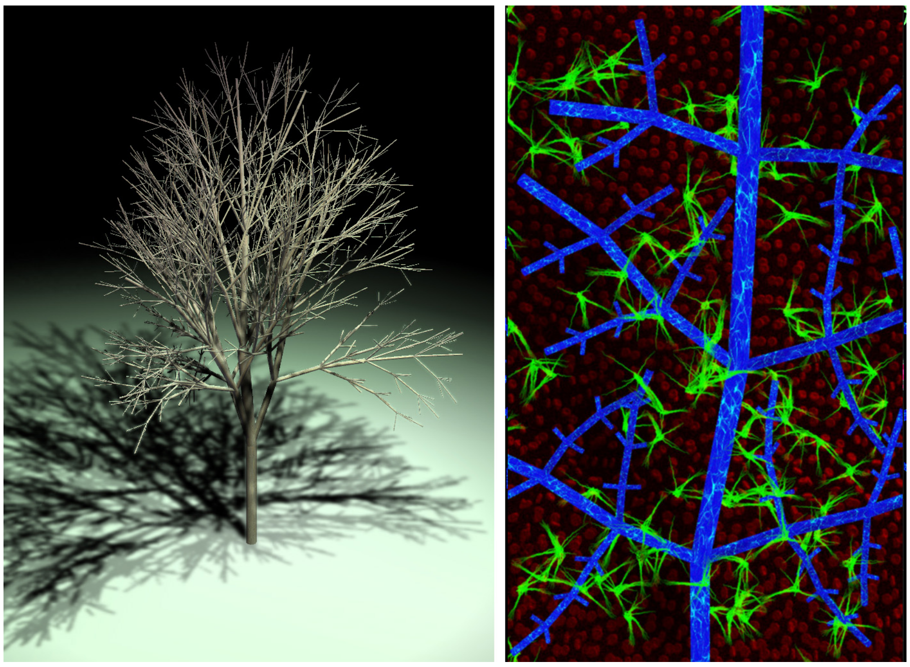


Figure 6.11: Organic and biological tree forms created using the same generative structure, demonstrating how context can play a part in understanding the sign of an object. Both images were simulated by the author using LUNA. The right image is based on matching retinal imaging results from the Neuroscience Research Institute (c) 2010, Gabe Luna, Geoffrey Lewis, and Steve Fisher. See text for details.

Images do not necessarily require the appearance of a surface to signify an idea, since realism may reside on many levels. As a result, another kind of choice is the ability

[5]Cg is a language for specify the shaded appearance of an object to the GPU.

to modify the context of a structure. The images of Figure 6.11 each uphold a sign in different ways. The left figure is recognizable as a natural (yet simulated) tree due to the ground plane, shadows, and overhead lighting. The right figure is recognized as a biological image, even if we do not know the particular structure it represents (blood vessels and astrocytes) because the structures, colors, and scale relationships between them remind us of microscopic neuronal images. Both objects were rendered using the same Tree object in LUNA, whose context is modified through its relationship to other structures within the graph.

The idea of image as a form of manipulation led to a reductive and abstractionist phase in art as artists reconsidered the nature of image making. What changes in each work, what remains constant? Structuralists found that the image itself, its context and rules, provide an underlying system for explaining works of art. While views of structuralism differ, Alison Assiter collects a number of these and finds five commonalities:

> "1) The whole forms a system whose elements are interconnected where the structure of the whole determines the position of each element. 2) Structuralists believe every system has a structure: the task of science is to find out what that structure is. 3) Structuralists laws deal not in changes but with co-existence. 4) Structuralists would not deny that dynamics is important in science, but would say that this is complementary to synchronic analysis. 5) Structuralism is a method which examines phenomena as the outward expressions of their inner, invisible structures [Assiter, 1984]."

Aesthetic structuralism is thus a way of analyzing art to reveal the sign it upholds, yet from a scientific perspective the idea of structure still contains many challenges. The concept of structure presents a paradox from the perspective of memory. If an image contains remembered signs, then the only way to avoid its status as myth is to consider structures which do not produce recognizable forms. However, once these forms are exhibited, their structures are committed to memory and they become signs for the future. Thus, the creative dimension of freedom in structure for the artist implies the ability to continually create and destroy the available forms to both break the rules and create new ones.

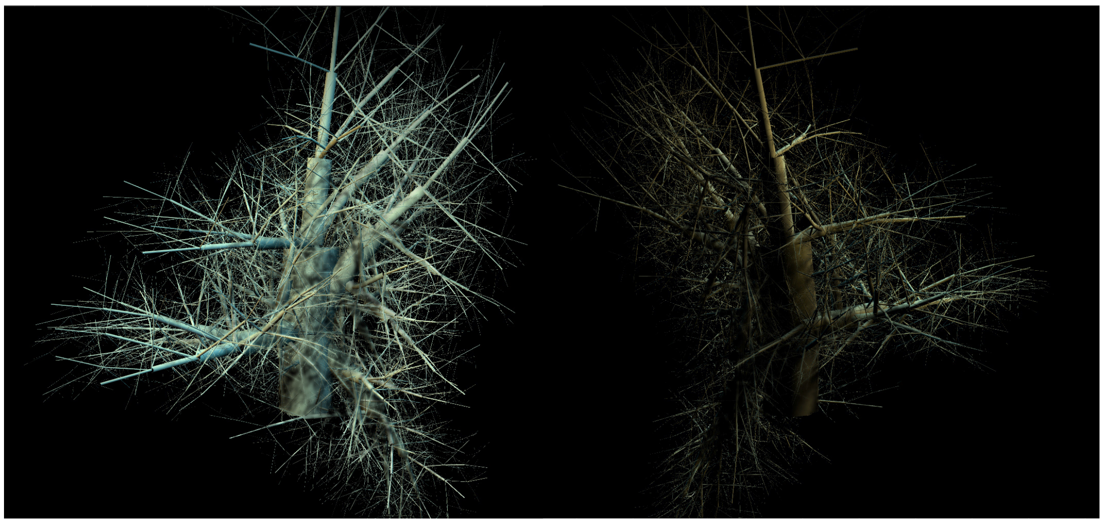


Figure 6.12: *Tmods*, R. Hoetzlein (c) 2010. Variations in a tree structure are explored by changing the parameters and rules used to generate the new forms.

As low-level languages were initially the only choice available, the approach found in current tools for media artists such as Processing have allowed the artist to create or destroy structures by coding these directly. Consider a tree form needed for a particular project. These structures are interesting to the artist not for the tree itself, but for the various dimensions of change that are possible with them. Examples of some interesting transformations are shown in Figure 6.12. In the current paradigm, to program such a system requires implementing the necessary data structures oneself or acquiring them from a community of artists to be integrated into one's own project. Instead, creative tools for artists would ideally allow for processes of construction, destruction, growth, modification, transformation, and appearance on a variety of existing structures presented in the tool itself. These forms are present not to uphold a sign implied by the structure, but to allow the artist to use them in a larger context, to change or reform them altogether through language. The artist is also still free to build new structures in low-level languages (LUNA's node authorship in C++ for example), but this new choice of transforming structures themselves engages the media artist at a different level than was previously possible.

6.4 Image and Idea

In the Foundation of Computational Visualistics, Jörg Schirra sets out to define a meta-field of study based on the image. His proposal is guided by the observation that different disciplines manipulate images in a variety of ways, and this establishes a basis for understanding how artists, engineers and scientists work with images. This is proposed according to the relation between image and not-image, briefly summarized here from [Schirra, 2005, p. 17]:

Type of Algorithm	Act / Field(s)
≫image≪ to ≫image≪	Image processing
≫image≪ to ≫not-image≪	Pattern recognition, Computer vision
≫not-image≪ to ≫image≪	Computer graphics, Information visualization

The not-image which Schirra speaks of is everything which is "not defined by the media type image." For example, in computer vision one goal may be to take an image and create a word-label for each object in it. In computer graphics the task of rendering is to take a structure and produce an image. For Shirra, the *concept* must exist in a context as a structure, an essentially connectionist viewpoint:[6] "Objects (as we usually understand the expression) are members of many contexts. What we usually call the identity of objects is basically the question of connecting an object in different contexts [Schirra, 2005, p. 52]." This helps to explain for Schirra how the tree used in computer graphics, which is a structure embedded in code, is the same "tree" which is a word embedded in the sign-perception of the viewer. Both of these not-images are connectionist ideas, however, while the concept of not-image does not fully represent the status of the other object involved in these processes.

Combining some of the processes collected by Schirra with the different media types developed in this thesis allows us to construct a more complete theory of the digital semantics of the image, shown in Figure 6.13. By digital semantics I mean the different meanings of images and processes as represented by machines. Schirra develops computational visualistics essentially on the functions performed on digital images, which can be found embedded in this figure as the in-going and out-going arrows (processes) acting on the image. Developing the media types further reveals that the sound, written

[6]Connectionism is the philosophical view that mental processes can be understood as emergent phenomenon of symbol processing machines or brains. Precursors can be found in Descartes' Treatise on Man (1633) and David Hartley's Observations on Man (1749), while the movement gained prominence in 19th century psychology with the discovery of the neuron.

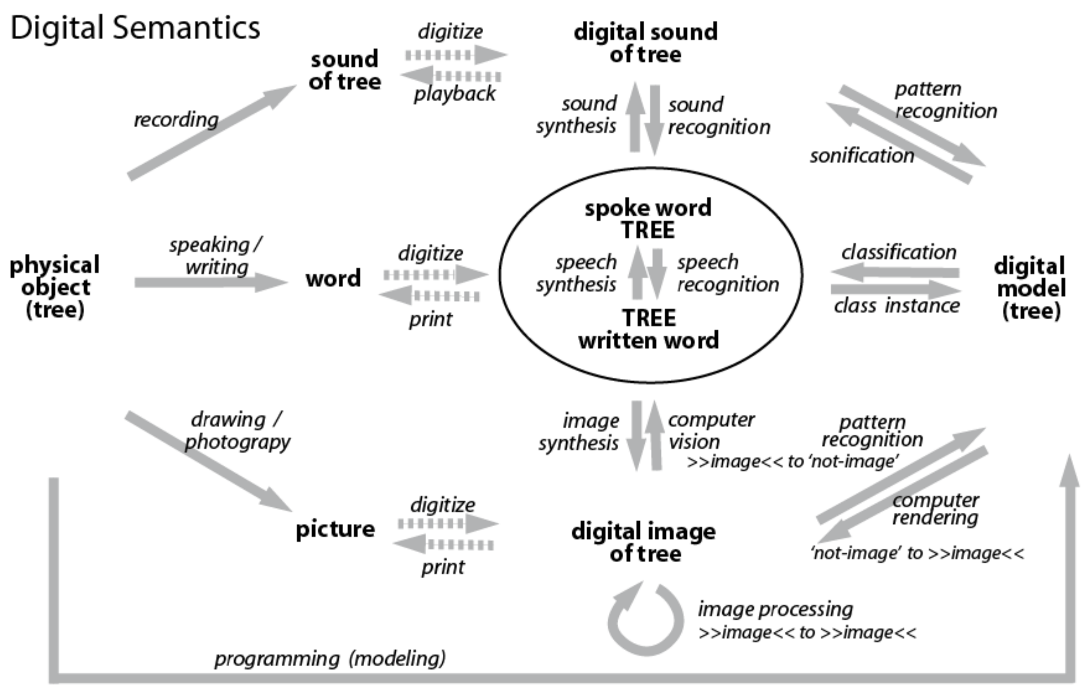


Figure 6.13: Semantics of the digital image and its relation to other forms of representation. Diagram by the author.

or spoken word, and the digital model are quite distinct from the image, and acted on in different ways by machines.

Overall the diagram in Figure 6.13 is arranged as a relationship between the physical object (left side) and the digital model (right side). The digital model is, for example, the structure of a tree as represented by machine as numbers. Interestingly, the digital model has no sensate form, i.e. it cannot be perceived by the senses, without passing through a sense-based media type such as sound or image. We cannot see the data structure of the tree as it exists in the machine without rendering it as an image of a tree. How are digital models constructed? This is one of the acts of programming, to create a mathematical or conceptual model of a real object. However, it is not necessary to have a physical referent to create digital models, which may also be constructed from abstract rules or principles. Thus the physical object is not necessary to make digital models.

The one media type found in this picture (center of Figure 6.13), but not yet explored in previous chapters, is the *word*. A word is unique among media types in that it is a symbol, in the sense used by Pierce to distinguish icon, index, and symbol; it is an abstract representation of a thing [Peirce, 1931, 2.228]. Put another way, the word for “tree” could refer to *any* tree, and does not embody a particular tree whereas all the other types of media do. So, whereas a digital model can provide a particular shape or form without a word, a word can provide an idea without the particulars. Neither of these yet involve the digital image, so we are speak here of language. In fact, in programming languages, the translation of a word into a replicated model by machine is called *instantiation*[7], and the reverse process of deducing a word from the pattern of a model is *classification*, both of which can be seen in the figure.

Two processes involving the digital image, not studied in great detail by artists relative to others areas mentioned here, are image synthesis and computer vision. These processes, shown in the connections between words and image at the bottom of Figure 6.13, are areas that artists could explore further in the future. Image synthesis is the process of creating images from semantic labels (words), while computer vision is the

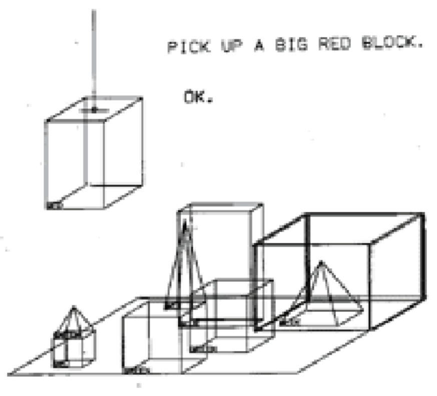
[7]Instantiation is the process of creating an object from a class, and the resulting object is called an instance. The computer must be told what the model is in a particular way, along with the word that defines it. Any number of *different* instances can then be requested on demand. Another term, *instancing* is a graphics concept which refers to making copies of a three dimensional model without duplicating the memory of its data structures.

process of determining a word given an image. The final dimension of freedom for media artists considered in this chapter is the relationship between images and words. It should be emphasized that this does not imply a particular value structure toward specific words or their referents. In Saussure's terms, there is no *signifié* anywhere in the figure above as every media type is a referent, including written words, since a word only becomes a complete sign in relation to human perception and memory (or context) [Saussure, 1965]. Words evoke meanings, but to the machine they are another type of data. That meaning should be a matter of choice for the artist, where possible processes are offered by the tool.

6.4.1 Image Synthesis

In 1968, Terry Winograd developed a system called SHRDLU, a system in which various shapes of different colors and sizes (blocks, spheres, cones) could be arranged in a simulated world [Winograd, 1971]. The user could then instruct the machine, using English sentences, to rearrange the blocks based on language. Figure 6.14 gives an example of a dialogue with SHRDLU.

The unique aspect of SHRDLU is the connection made between a visual, three-dimensional world and the semantics of language. SHRDLU contains a toy model of a world which the user creates and modifies entirely through written dialogue. This world is interesting from a media arts perspective since the representation is a visual one, and is thus a form of image synthesis. What is the relationship between computer rendering



Person: Pick up a big red block.
Computer: OK.
Person: Grasp the pyramid.
Computer: I don't understand which pyramid you mean.
Person (changing their mind): Find a block which is taller than the one you are holding and put it into the box.
Computer: By "it", I assume you mean the block which is taller than the one I am holding.
Computer: OK.
Person: What does the box contain?
Computer: The blue pyramid and the blue block.
Person: What is the pyramid supported by?
Computer: The box.
Person: How many blocks are not in the box?
Computer: Four of them.
Person: Is at least one of them narrower than the one which I told you to pick up?
Computer: Yes, the red cube.

Figure 6.14: SHRDLU, by Terry Winograd, allows a user to move around blocks in a virtual world using natural language. Image copyright Terry Winograd (c) 1968.

and image synthesis? From a graphics perspective, rendering is the process of generating an image regardless of its source, which may be a memory of images or a digital model, and thus encompasses all image-making. In the present context, rendering is defined as forming an image from a digital model which *represents* the object it refers to - for example, a hierarchical three dimensional model produces a tree. This is distinct from image synthesis, in which a *picture of a tree* can be used to render a tree, and is called image-based rendering in the graphics community [McMillan and Bishop, 1995]. Image-based rendering was introduced in 1995 as a way of rendering three-dimensional scenes using composited collections of two-dimensional images, although I am using the term *image synthesis* more broadly to refer to any image produced by another set of images, such as a collage. In the examples of *Trees* in the preceding section, a digital model is used to create a tree without reference to an image of a tree. The unique aspect of image synthesis is that the only distinguishing feature in the model is the concept 'tree' (present in the picture), since there is no structural representation in the machine.

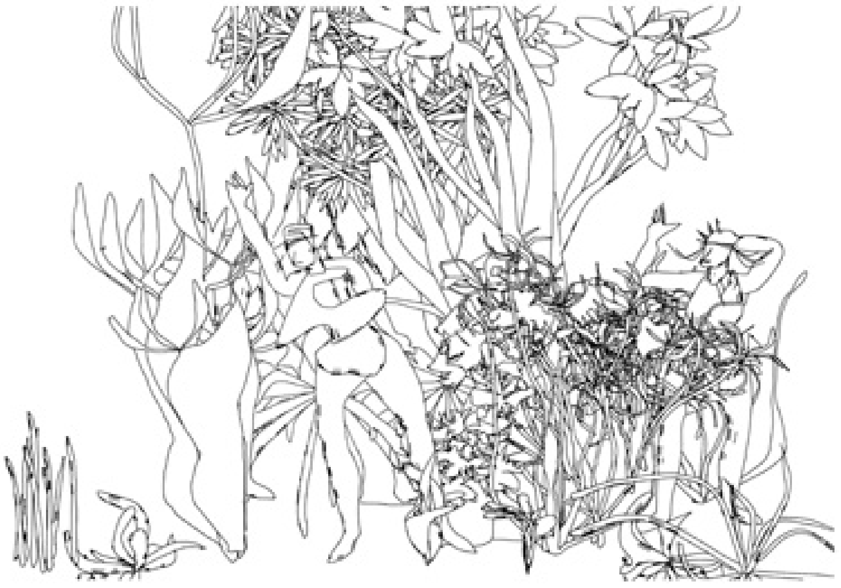


Figure 6.15: Aaron's Garden, 1989. Pen and ink drawing by AARON software created by Harold Cohen. Image courtesy of Harold Cohen (c) 1989

In certain areas of visual arts such semantic systems already exist. Cohen's unique AARON software achieves automatic drawing of human figures and plants using a set of semantic rules [Cohen, 1979], Figure 6.15, and is thus not strictly a form of image synthesis but an abstracted *symbolic* model of a human figure. AARON is instructed on the semantic, word-meaning relationships between parts of the body to automatically generate compositions. In organic art, systems like Xfrog are capable of generating plants with roots, stems, leaves, and branches based on the labels of words assigned to different parts. Although any model is ultimately a simplification of reality, it is interesting to consider how far semantic systems could go in describing aesthetic or imaginative objects. Is it possible to describe and animate a complete, imagined world by semantic means? I consider this an open question, and one which is not presently answered by LUNA. The language of LUNA was developed to move toward this goal, however, by observing that such a system would need to create *structural* and *functional* relations between objects. Currently these are embodied in the line connections created within a visual graph.

The relationships between the parts of different objects in Aaron's Garden are physical attachments, A is connected to B. Parts have physical relations to other parts. However, relationships between objects are also conceptual and symbolic. The ways in which these word-relations might be potentially interpreted by machines is as complex and varied as the behaviors and structures explored in the preceding sections, yet aside from AARON there are few other examples of semantic drawing in media art.

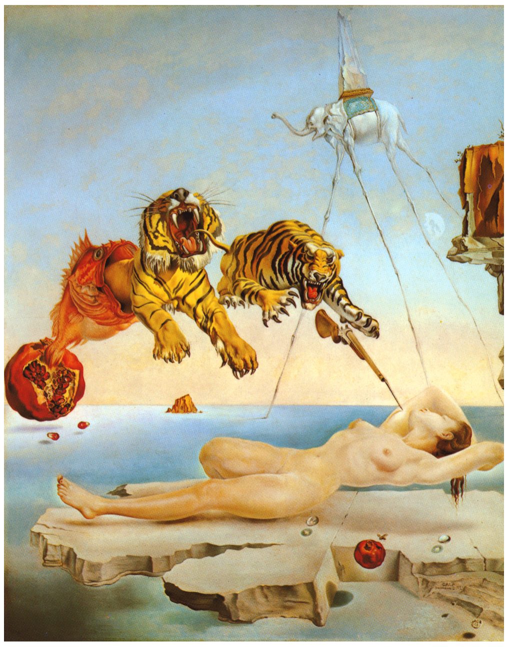


Figure 6.16: *Dream Caused by the Flight of a Bee Around a Pomegranate a Second Before Awakening*, Salvador Dali, 1944. Oil on Canvas. 20" x 15.9". Thyssen-Bornemisza Museum, Madrid.

Consider the image of Figure 6.16. This image, *Dream Caused by the Flight of a Bee Around a Pomegranate a Second Before Awakening* by Salvador Dali (1944), presents a more complex semantic. We might describe this image formally as "A nude figure floats over a stone slab suspended above a blue plane. Over her body, poised at the neck, a gun with a bayonet is released by a tiger jumping from the mouth of a lion, which jumps from the mouth of a goldfish, which jumps out of a pomegranate. In the distance, an elephant with extremely long bony legs carries an obelisk." Certainly no system could ever be expected to reproduce this image exactly without further details relating to scale, placement and shading of the objects. It is interesting to consider whether this might be possible at all. Such a system would need to be familiar with the objects in the image, as well as the parts of fruits, animals, and humans.

This example by Dali points out several paths, as image synthesis may take many forms. The form explored by AARON is a verb relationship between parts - 'An arm is next to a body. A head is above it.' The objects in Dali's painting have more to do with action relationships between objects suspended at a moment in time. One approach might be to *build* a scene from a large collection of three dimensional models stored in memory, while another could be to allow dynamic models to actively engage in word actions described by the user which are then "recorded" by rendering a moment in time. This example by Dali was chosen due to the wide range of meanings present, but also to highlight the complexity of these tasks with real world objects. However, the meaning-system of Dali's surreal representation is not necessary to explore this dimension of creativity. The words used could be the sequences of DNA strands (a typographic system), or perhaps poems, producing dynamic images based on abstract models as Michael Rees does in his work *Putti*.

The simplest way to explore image synthesis is through memory. A word evokes an idea, which in its simplest form may be represented in the machine as a database of images from which a choice is made. *Social Evolution* is a project by the author which pre-dates LUNA, but which could be more easily represented in LUNA using constructs for image synthesis. Social Evolution consists of characters engaged in verbal acts such as walking, running, sleeping, harvesting, killing and eating, and was developed by hand-sketching a series of characters in various poses (Figure 6.17). The system for Social Evolution required programming of image selection, behavior, and image synthesis. This example has not yet been converted to LUNA, but could be accomplished using

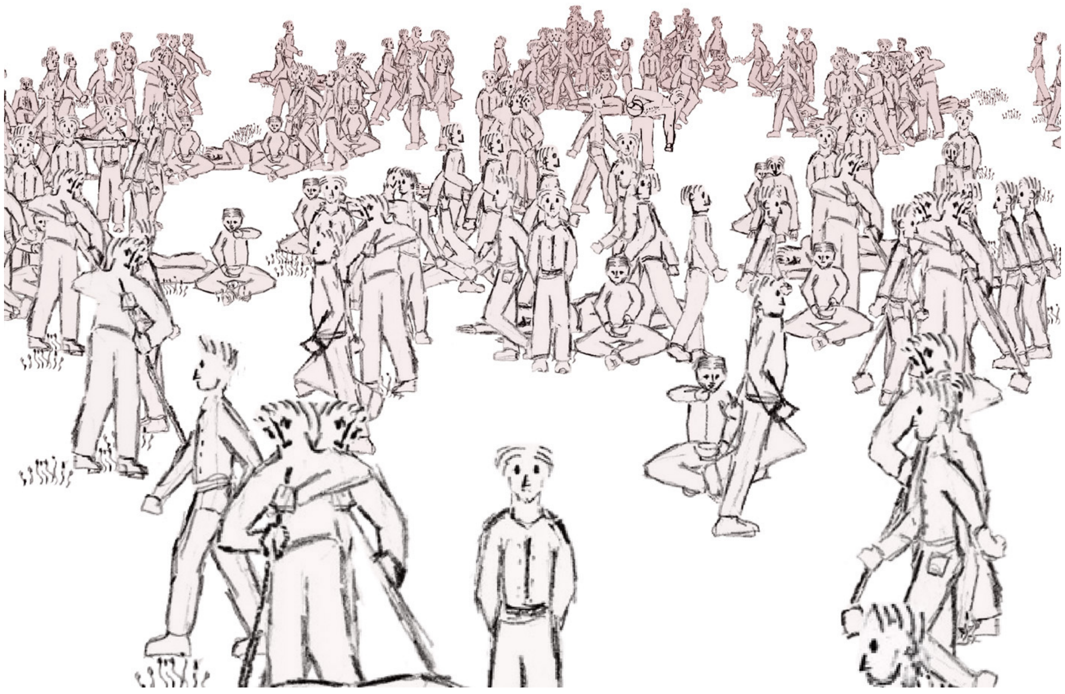


Figure 6.17: *Social Evolution*, R. Hoetzlein (c) 2007-2010. Hand drawn characters autonomously engage in activities including sleeping, eating, walking, killing, harvesting and running. Exhibited at the 2nd Beijing International Arts & Science Exhibition (2007), Tsinghua University in Beijing, China, and the Center for the Contemporary Image in Geneva, Switzerland (2009).

the Image Scatter object, taking a set of locations determined by behavior and placing these at locations in a scene.

At one point, I considered developing digital models to generate the image database for new experiments in image synthesis, but made an interesting discovery with hand drawn structuralist figures. These images, titled *Puzzles*, are shown in Figure 6.18. A digital model is a structure which by definition involves a set of known rules used to produce a form, since it must be interpreted by machine. What are the rules used in

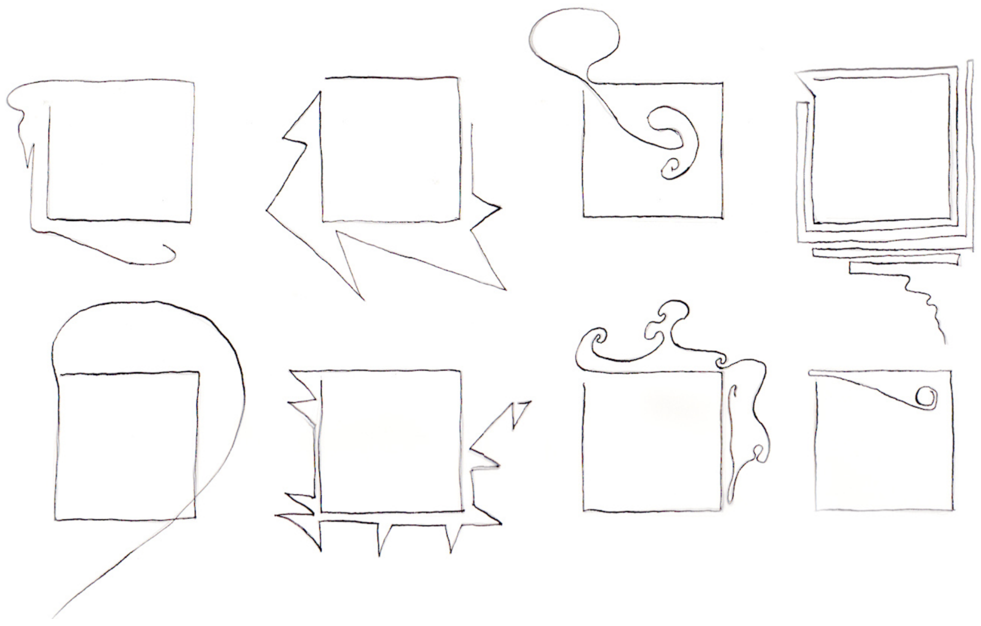
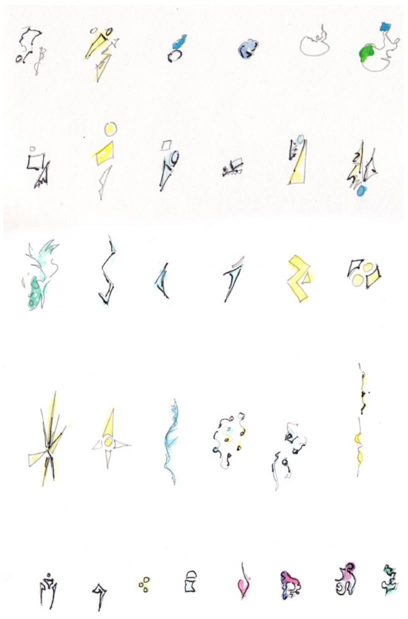


Figure 6.18: *Puzzles*, R. Hoetzlein, 2010. Ink on paper.

figure 6.18? Two obvious rules are the concept of the nearly closed square as a starting point, and the use of an uninterrupted line. A third might be the idea of deviation but this can be deduced from the first two. However, these three rules alone are insufficient to digitally reproduce the shapes create here. One might state that there is an abstract rule which is, “Create a series of curves which try to break a pattern.”, which could describe the psychological or conceptual process involved, but this also is insufficient to reproduce these images since even knowing this fact does not allow one to explain why these *particular* shapes were created. While these figures may have been created by some internal rule within the artist, that process is inaccessible to us. This aspect of the hand drawn image, to have content without known rules, is a unique distinction from the digital model, which *requires* rules for machines to interpret them. The integrated use of images in LUNA is essential in allowing artists to mix digital models with non-formal images.



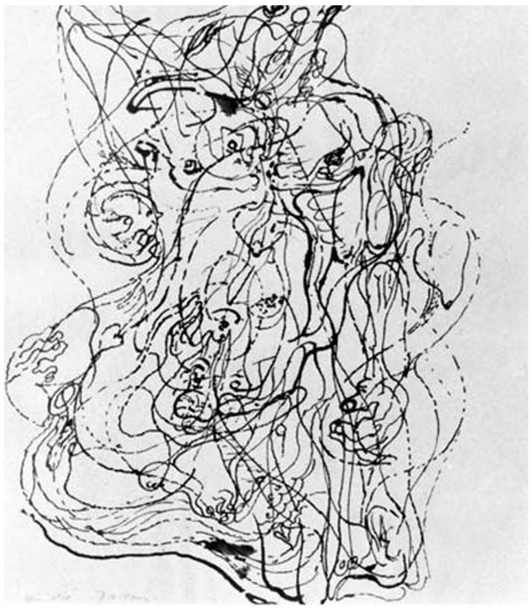


Figure 6.19: a) *Automatic Fragments*, R.C. Hoetzlein, 2010. Ink and water color on paper. b) *Automatic Drawing*, André Masson, 1924. Ink on paper. 9 1/4" x 8 1/8". Museum of Modern Art, New York.

Figure 6.19a show a series of fragments created entirely as hand drawn sketches (no computerized process is used). The process is similar to that of *automatic drawing* employed by André Masson to shift between an "unconscious process" and brief fragments of a recognizable figure in Figure 6.19b. These sketches were each created in less than one minute, by starting with the idea of a splotch which guides an unresolved curve (or vice versa), i.e. the form of the curve is not conceived of ahead of time. To create the curve, the mind is free to sketch whatever appears to it at that moment in the splotch, thus there is a similarity in process to automatic drawing in allowing the "unconscious mind" to help in forming the result. These forms are interesting in that they represent the combination of a rule-based process and a rule-free process, and thus cannot be understood in terms of any formalized rule system. However, unlike Masson's drawings, whose fragments make reference to figurative forms, they also remain abstractions that exist as pure form. While the overall pattern is known, the pattern of each shape is unknown, and no computer process could be found to reproduce them.

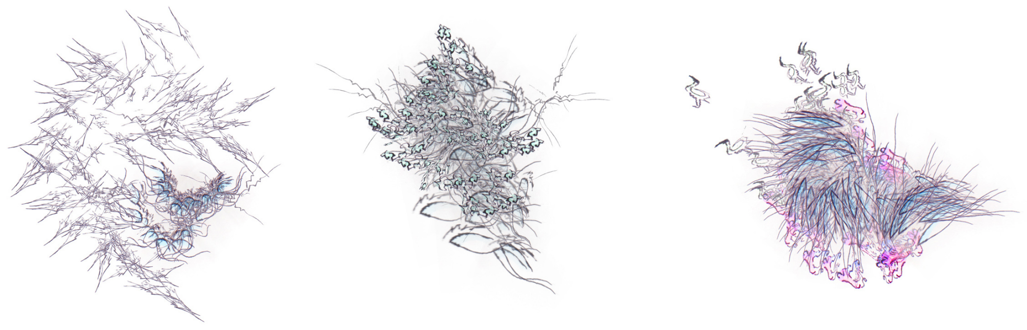


Figure 6.20: *Fragment Collage*, R. Hoetzlein, 2010. Fragments of hand drawn images are composited by the computer using generative algorithms.

Although machines require rules to create models, in the next step *rule free* fragments are incorporated into digital works through computer generated compositions. The images in figure 6.20 were created by allowing the machine to determine the arrangement of hand drawn fragments of 6.19. Thus, these images contain two conceptual elements: 1) a formalized system of rules for placing shapes in space, and 2) a set of shapes based on a rule-free process, created by hand as described above. The result is a man-machine collaboration in which the total image contains abstract formalized rules, yet carries no descriptive connotation, and which cannot be recreated in its entirety by any set of known rules. These images show that digitally generated images may contain a human element without connotating something specific to human reality (such as a body or a tree). While algorist art contains a unique human aspect in the computer code itself [Verostko, 2006], that aspect remains hidden from the viewer except as an overall pattern in the outcome. In the present images, the unknown human element of the drawing co-exists with deterministic patterns in the visual forms themselves.

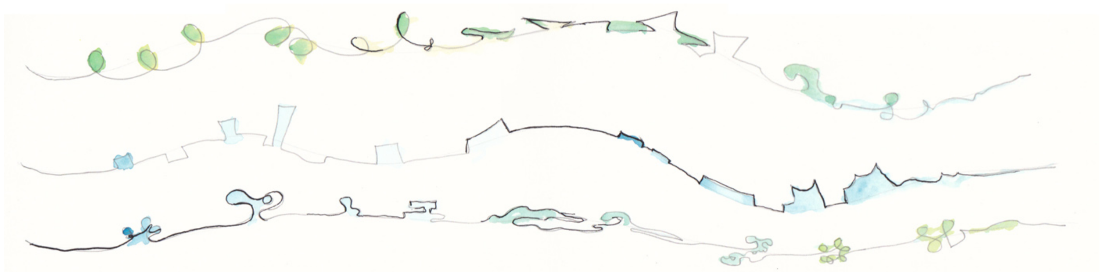


Figure 6.21: *Sequence*, R. Hoetzlein, 2010. A continuous curve with a sequence of structural deviations presents a particularly difficult challenge for computer synthesis.

A final series of drawings further demonstrates the idea. *Sequence*, figure 6.21 is a number of continuous curves with structural deviations along its length, such that the whole line would be very difficult to formalize by machine (no computerized process is used here). These experiments show that there is potentially a continuum between object or word-based image synthesis as found in Social Evolution and purely generative synthesis found in *Fragment Collage.* The contribution of image synthesis, as a process, is that it may or may not involve an idea (word form) but in both cases, by relying on external hand drawn shapes, the image need not be formalized. While a digital model is always a formal structure, the image has a formal structure (pixels) yet its content may extend from any source: natural, human, or digital.

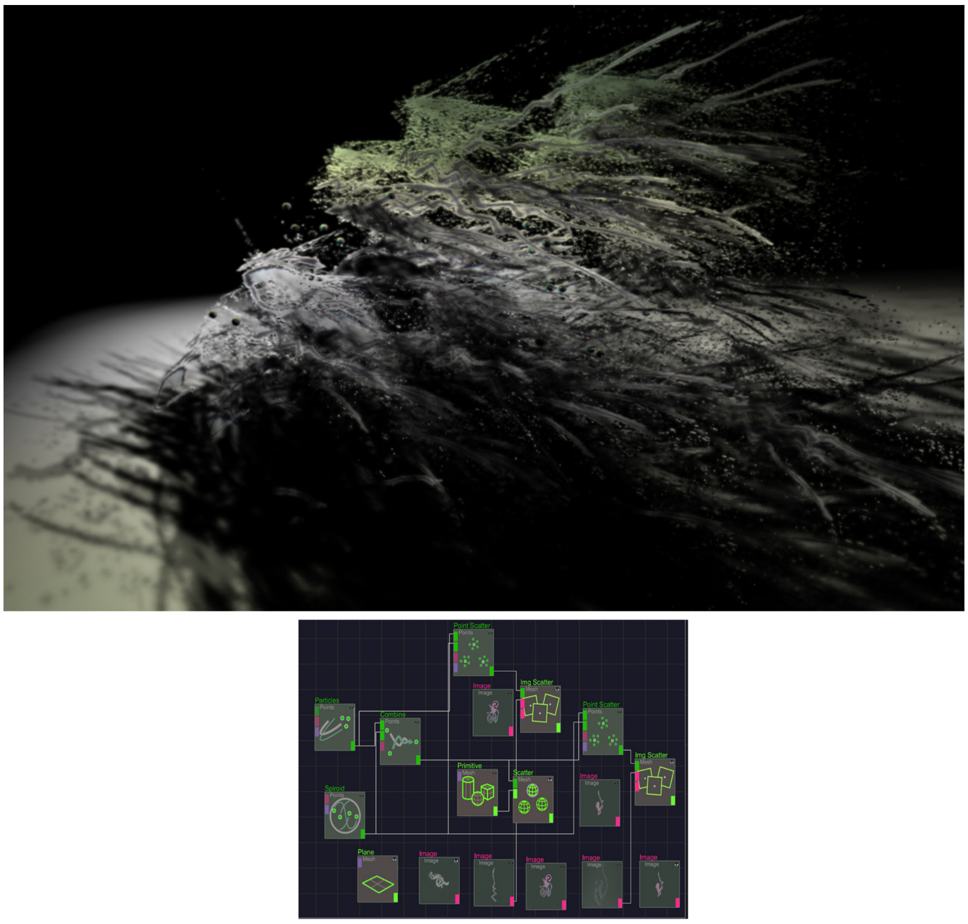


Figure 6.22: *Dark Fragments*, R. Hoetzlein, 2010.

This final image, figure 6.22 shows more rendered version of a Fragment Collage along with the LUNA graph used to create it.

6.4.2 Computer Vision

As a general technique, computer vision has found a presence in media arts since the 1970s, exemplified in Myron Krueger's *Videoplace*, an environment for human interaction with virtual spaces [Krueger, 1991]. More recent examples, such as Golan Levin's *Footfalls* (2006) encourages participants to play with digital circles released by the sound of stomping participants while visual detection enables them to hold and collect these virtual balls. Both works essentially treat the participant as a body-form whose silhouette is recognized as an interacting shape. In a later project, Opto-isolator (2006), Levin collaborates with engineer Greg Baltus to develop a mechanical eye which follows the visitor's own gaze, blinks and looks away. In general, these techniques use computer vision to transmit the gestural interaction or facial features of the participant into the machine. This method was also used in *Presence*, a collaboration between myself and Dennis Adderton, using LUNA to present a 360 degree panoramic photograph which reorients toward the view as one walks around it. A recent article by Levin [Levin, 2006], demonstrates that computer vision is still a developing field in the arts, as techniques for object recognition of the image, i.e. semantic labeling, have not yet made their way to media artists.

In other communities, however, engineers that specialize in computer vision have been making rapid progress. The Computer Vision Laboratory (BIWI) at ETH Zurich developed a cell phone application which allows art museum participants to photograph a work of art and receive an instant identification of the object, with an accuracy of

82% [Bay et al., 2006]. Stanford University and the Nokia Research Center created an outdoor application that identifies parts of an image in photographs taken using a cell phone [Takacs et al., 2008]. These examples represent the current state of the art in computer object recognition, transforming a digital image into a set of word labels.

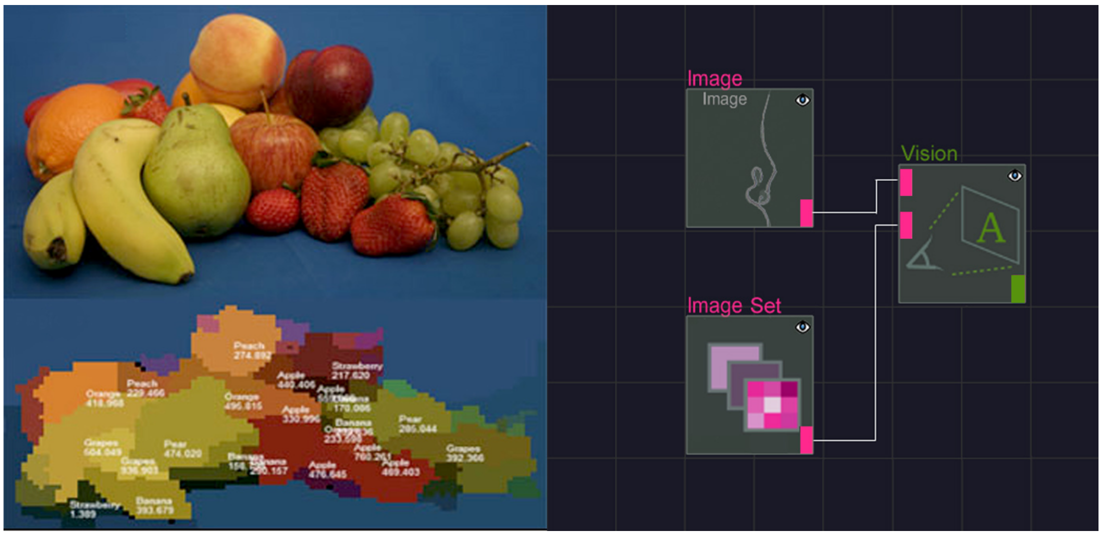


Figure 6.23: Computer vision in LUNA with two inputs, 1) an image set representing a memory of objects, fruits from various angles and 2) a target image of a still life to be detected (top left), produces a set of labeled points (bottom left) of detected objects.

A similar experiment could be performed in LUNA in the future by using computer vision processes that detect regions of color in images. Figure 6.23 shows a potential example, based on prior work done for a class[8], in which colored regions of a still life photograph are used to label fruits. In a reversal of image synthesis, which uses image memory to generate new compositions, this method takes an image and detects regions

[8]Based on work done in Computer vision, a graduate class with Professor Matthew Turk at the University of California Santa Barbara.

in relation to that image memory. This introduces a new media type in LUNA, a Text object, which derives from a point set that includes labels. This Text object may also be used in the future for information visualization, as labels are assigned to geometric locations based on information contained in a database.

The generation of images from digital models is a well developed area of computer graphics since the need to create visual representations of computer models (whether they are abstract, realistic, rule-based or informational) is an important technique in several fields. Relatively speaking, the relation between *words* and images is still largely unexplored by media artists, due to the fact that methods for image synthesis and computer vision are potentially more complex. To facilitate the exploration and expansion of media arts into other media, processes, and ideas, these facets of the digital image are introduced in LUNA as building blocks to be interacted with, while their future growth is in the hands of the artistic community.

6.5 Conclusions

The development of this thesis has proceeded along several lines of thought, or dimensions, related to form, data and technique in media arts. The creative features of digital tools to provide low threshold, high ceiling, and wide walls are employed as evaluative criteria to varying degrees in LUNA through the design of the language, its interface, interaction, and the number and types of media both potentially and currently available in the system. LUNA allows several basic dimensions for creative workflow.

These include: 1) the ability to author new models in text-based languages and also through the interactive visual interface, 2) the ability to work with different media types including video, images, and geometric shapes, surfaces, and materials, and 3) the ability to adjust performance and rendering quality to meet the particular needs of live performance.

In the area of content, this work has explored: 4) motion and dynamics, 5) structure, and 6) image and words. The construction of these arguments is based on the concept of creative *dimensions* in media arts. In the process of exploring content, however, it becomes apparent that there are several facets to each particular mode of working. Thus, these dimensions should not be considered as Cartesian generalization of the space of work explored by media artists. Certainly the examples considered here may be thought of as points within a particular dimension or context, yet beyond this, each dimension should not imply a linear map for expression. These dimensions should be understood to refer more loosely to sets of opposing and complimentary practices which take place within media arts. The actual choices themselves, however, exist along a complex number of features related to each type of media based on specific decisions of the artist. The content of LUNA explores a particular space of media arts in the area of geometry, form and media, one which is possibly wider overall by design than other tools for artists, yet still a series of choices based on the author's interest in sculptural, interactive, and generative forms in relation to word-concepts.

How the artistic community conceptualizes and develops creative tools plays a major role in defining the kinds of art that will be created in the future. Computer vision and image synthesis, for example, are novel approaches to considering images and ideas, but are largely unexplored due to the high threshold for entry into these methods. This limitation currently focuses and drives artistic work into more easily accessible areas such as modeling and rendering and thus forces decisions in content. To overcome such restrictions it is necessary to redefine creative tools not in terms of their current abilities and features, but in terms of the space of possibilities which can be expanded as the field evolves. The contribution of this work has been to propose and show that several of these dimensions which have become separated over time into tools for distinct communities, may be brought into existence together through a consideration of their relationships and constraints. The final purpose of which is to enable and encourage creative work with digital media in different, potentially better ways, without having to repeat work while simultaneously coming closer to expressing what one hopes to as an artist.

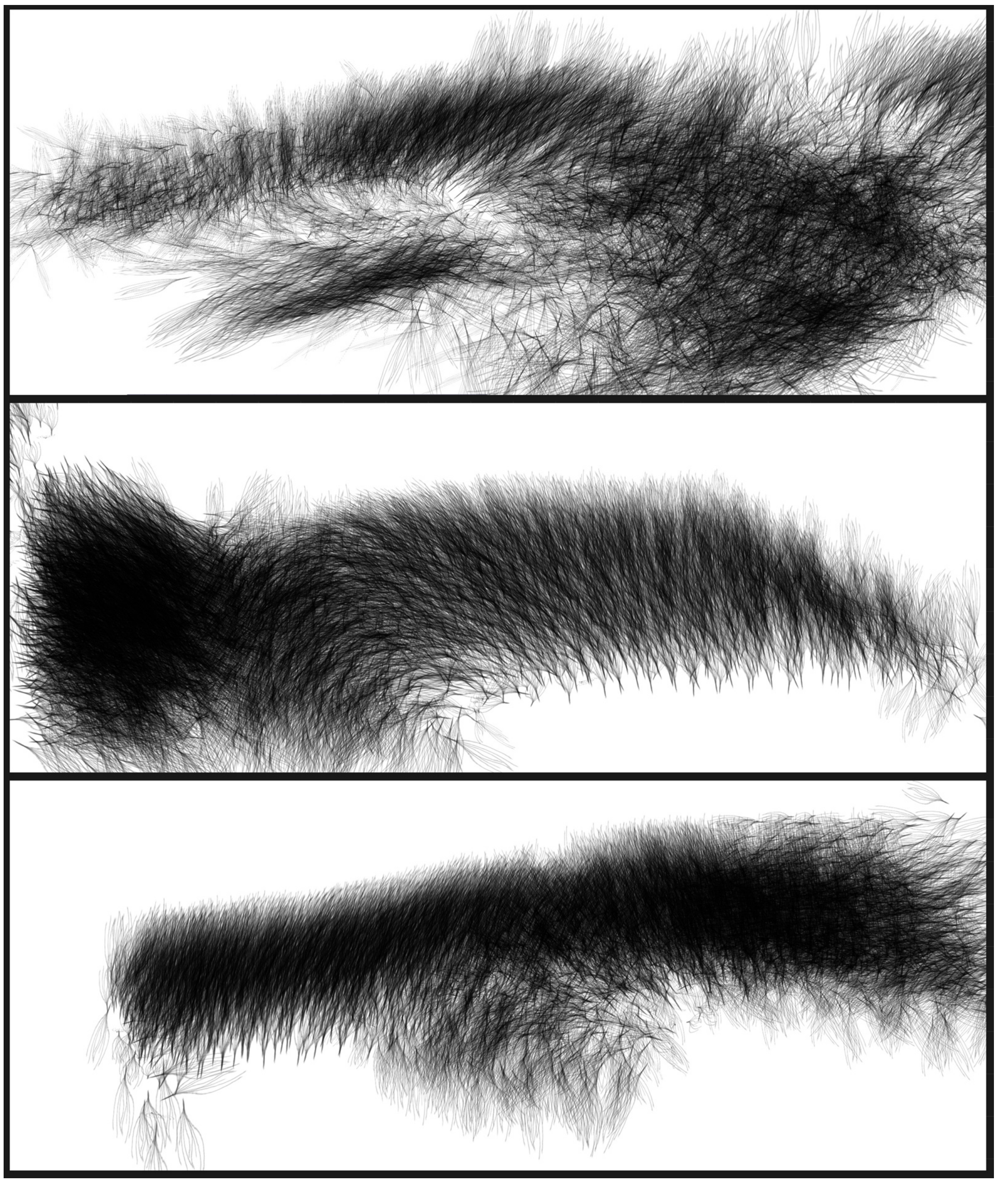


Figure 6.24: *Soft Sketches*, R.C. Hoetzlein, 2010. Ink on paper with computer generated composition.

Chapter 7

Conclusions

7.1 Summary of the Dissertation

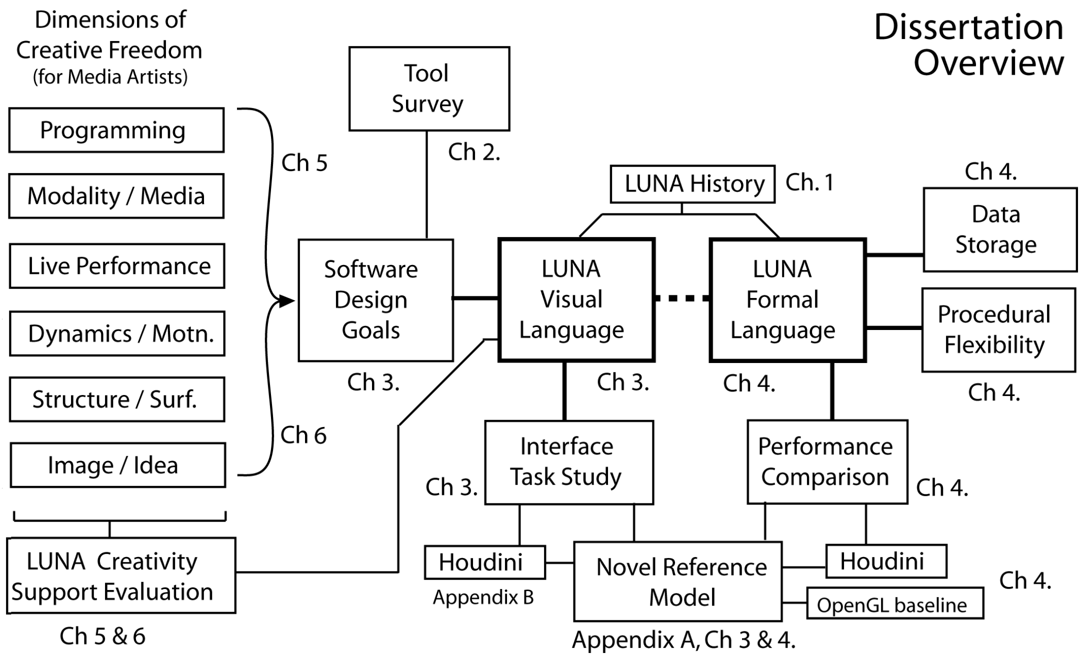


Figure 7.1: Summary of the dissertation showing relationships between the various chapters in the development and evaluation of LUNA.

This dissertation introduces LUNA, a novel visual dataflow language for media artists. The software was developed by considering six creative dimensions of interest to media artists, and exploring the possibility of developing integrated tools that combine these various freedoms. These creative dimensions were used to motivate the software goals and graphical user interface of LUNA, which influenced the formal structures of LUNA's procedural language. LUNA is evaluated in several ways, first by the user interface and its flexibility in comparison to Houdini (Chapter 3), second by the procedural and visual results it generates (Chapter 4), and third by its computational performance relative to Houdini and a baseline OpenGL model (Chapter 4). Finally, LUNA is evaluated as a tool for artists based on metrics developed by the Creative Support Tools workshop held by the National Science Foundation in 2005. These metrics, 1) low threshold, 2) high ceiling, and 3) wide walls, are considered relative to the six creative dimensions set out in the dissertation. An overview of these relationships can be found in Figure 7.1. This chapter provides a summary of the dissertation and its conclusions, and explores current limitations and future directions.

7.2 Creative Dimensions

Six creative dimensions of interest to media artists were used to establish a basis for exploring freedoms in visual design tools. These dimensions cover a range of topics, including procedural modeling of sculptural forms, live performance, and dynamic motion. Some dimensions, such as programming and modality, were considered because of their essential relationship to tools for the media arts. Finally, some relatively new directions are explored in the dimension of image and idea. The six dimensions explored are:

1. Programming and Language
2. Modality and Media
3. Live Performance and Computation
4. Motion, Dynamics and Autonomy
5. Structure and Surface
6. Image and Idea

Further motivation for this choice of topics can be found in Chapter 5. The first three topics address issues of *language* in the use of digital tools for making art, considered in Chapter 5, while the last three cover issues related to *content* in media arts, explored in Chapter 6.

One clear observation of these dimensions is that they do not cover all the ways in which media artists work. Major areas such as device interaction and audio synthesis are not addressed here, nor are the topics of web-based art, or information aesthetics based on the database. However, these areas are considered as future directions in which LUNA could expand.[1] One area in which LUNA is most likely not to be able to contribute to easily is web-based art, since LUNA is developed in the application language C++. While the LUNA front end could be ported to Java or Flash, it would loose many of the performance benefits it currently has.

[1]Just prior to publication of this work, an audio system was started for LUNA using OpenAL and PortAudio that allows for sound playback.

Another area not explored in this dissertation is network-based art, telepresence, or distributed computing. This is viewed by the author as another sub-system which could be added to the LUNA toolkit. Ideally, the interface would include send/receive nodes that could be used to transmit any type of media to another computer also running LUNA. Messages from other protocols such as OpenSC could also be received by LUNA. These are future directions that would be interesting to explore.

7.3 Graphical User Interface

Motivations for the LUNA's graphical user interface are influenced by the creative dimensions presented above. The board game Scrabble provides the inspiration for a minimal design based on a combinatorial arrangement of iconic tiles. Other interface features of LUNA are intended to simplify the process of creative exploration by artists. These include:

1. Programming - Mathematical knowledge is optional for the artist, where the primary mode of interaction is conceptual and exploratory.

2. Modality - The language offers a range of different media types, supported through a tool set that includes points, curves, surfaces, images, and materials. Through the application of color and object labeling the interface is designed to make it clear to the user what media types are in use while working.

3. Live Performance - The interface enables live performance by allowing for full screen, high quality output, with real time design changes. This aspect informs the

overall visual design of LUNA, resulting in an inverted window layout that floats the menus and interface elements *above* an output canvas.

4. Dynamics and Behavior - The system allows users to modify dynamic objects and receive immediate feedback, with a property panel to control behavior interactively.

5. Structure and Surface - Users can change materials, textures, and the surface appearance of two and three dimensional geometry in addition to their behavior and structure.

6. Image and Idea - Default behaviors allow users to explore complex objects without lengthy interactions needed for setup.

A primary contribution of this work is the integration of these design features into a single tool. Chapter 3 provides an example of how to create a simple procedural model using LUNA, a reference model resembling a woven sphere, which is also used later for performance tests. The steps needed to construct this model are compared to interface tasks in Houdini (Appendix B). The LUNA interface is also evaluated according to Green's criteria for visual dataflow languages, which include 1) *commitment*, when the language requires early decisions, 2) *progressive evaluation*, the ability to see intermediate results, 3) *expressiveness*, how easy it is to say what you want, 4) *viscosity*, how much the interface resists change, and 5) *visibility*, how easily you can see what you're creating [Green and Petre, 1996]. Subjectively, LUNA is shown to be better than Houdini along several of these measures.

A limitation of the graphical interface evaluation presented in this work is the lack of a user study. One problem is how to define an exploratory creative task that could be reasonably compared in two or more different systems, although this may be resolved by asking users to create *any* object that includes certain features of the language. Another challenge with a user study for media artists using LUNA is the lack of other generic frameworks for procedural modeling. To my knowledge, Houdini is the only modern visual dataflow language which has features similar to LUNA. Despite the lack of a user study, the flexibility of LUNA for creative tasks is shown by example.

Overall, the intention of LUNA is that the interface design serves as a key contribution to the creative community. Artists are supported through a language that allows immediate construction of high level objects without the need for detailed or repetitive interaction tasks and without mathematical knowledge or conceptualization of the structures present in the language. These high level objects are capable of procedurally generating numerous scene objects in conjunction with an internal language discussed in detail in the next section.

7.4 Procedural Modeling

The formal language which supports the interactions possible in LUNA is described in Chapter 4. LUNA consists of two directed acyclic graphs, a *procedural graph* that represents abstract behaviors, and a *scene graph* representing geometric structures and other media types (such as images and materials). The procedural graph operates on input and output subsets of the scene graph, allowing multiple scene objects to be generated or modified at run-time. The distinction between these two graphs, while handled internally in LUNA using a single graph system, allows for a functional distinction between what high level objects do, and the visual objects they create or generate.

Although such a distinction between behavior and structure can be found in other systems such as Squeak, ConMan, and Houdini, a unique contribution of LUNA is the representation of structure using discrete geometry stored in uniform buffers. These buffers allow for interoperability between media types and for direct copying of data into GPU buffer objects. The combination of LUNA's graph structure and its storage mechanism support the efficient generation and rendering of procedural objects in real time. The evaluation model, which takes advantage of the CPU and GPU, and defines how LUNA itself functions, is also described in Chapter 4.

The language capabilities of LUNA are evaluated according to the flexibility of its output. Although LUNA does not yet support replication (building duplicate models with slight variations), it does allow for compound instancing (creating instances of objects which also contain instances), and for changes in the order of operations. The LUNA language includes *modifiers* that can operate on objects at different stages in the graph, greatly varying the resulting output. These interactions can be performed immediately and interactively with just a few click-drag motions. The rendering system in LUNA uses a deferred shading engine to support high quality images with soft shadows and depth of field, while materials and shaders may be added to any graph, allowing users to modify the surface appearance of objects interactively.

Although LUNA is a flexible, interactive system for procedural modeling it still has several limitations. While the data buffers of scene nodes permit hierarchical relationships within a single object, such as a JointSet used to represent trees, the graph does not allow for hierarchical relationships among scene nodes and is thus a simplified scene model. At present, behavioral nodes may only output lists of scene nodes, limiting the potential range of the results. For example, character animation is not yet present in LUNA but could be added in the future. Another limitation is the lack of support for integrated physics simulations. While LUNA currently includes a fluid system (using smoothed particle hydrodynamics), a model that allows groups of complex objects to be treated as rigid bodies is not yet supported.

In general the design strategy of LUNA has focused on its interactivity, live performance features, and capabilities as a real time system. In a commerical environment, complex objects such as characters and physics simulations would also be present. LUNA was created with the idea that these may be added to the system by artists and engineers in the future. LUNA is thus a novel language for combining generative, procedural modeling of geometric structures with a focus on live performance and interactive feedback.

7.5 Performance

To deliver results suitable for interactive feedback and live performance, LUNA incorporates several features to make it more efficient. First, the memory layout of data buffers in the system matches the layout of GPU buffer objects, allowing data to move easily between the CPU and the Graphics Processing Unit (GPU). Secondly, the evaluation model of LUNA enables the system to detect and update only those objects that have changed. Finally, the system takes advantage of hardware rendering and vertex buffer objects to efficiently render objects. These features are described in Chapter 4.

Performance features which directly benefit the user include an interactive profiler showing immediate feedback on the computational resources being used by any node in the graph. This visual feedback allows the user to determine which objects might be overloading the system and allows them to scale back the amount of data being processed. In the future, the ability to control the quantity of data flowing through the system could be automatically updated by the rendering system, so that detail is selectively added or reduced based on the goals of the artist for live performance or offline results, or based on the performance of the computer being used. This feature is described in Chapter 5 (Live Performance).

The performance of LUNA is demonstrated directly by using a novel reference model, a woven sphere, in comparison to the same object in Houdini and to a baseline model written directly in OpenGL (a low level graphics language). These results are presented in Chapter 4. With this example, LUNA is shown to be 7x to 10x faster than Houdini. Despite these improvements in model evaluation and rendering, there are other areas of performance that have not yet been explored. The use of spatial culling, found in scene graph systems such as IRIS Performer, is not yet implemented in LUNA. Similarly, several improvements could still be made to the renderer to avoid changes in graphics state (slow changes that occur in hardware).

Current graphics processing units have a computing power that is equivalent to *fifty* or more CPUs, yet this technology is new enough that few applications take full advantage of it. Although only one node in the system, the fluid simulator, currently uses GPGPU computing to advance fluid particles using CUDA, the design of LUNA potentially supports multiple re-entrant GPU kernels to allow entire procedural graphs to be simulated entirely on the GPU.[2] Presently there are few generic visual dataflow languages for the GPU so, in the future, LUNA could be the first procedural language to support installation projects with computations taking place primarily in graphics hardware.

7.6 Creativity Support

The larger goal of this research has been to show that different communities of artists need not be limited to working with the inherent constraints of particular tools in their

[2]GPGPU computing is the use of the Graphics Processing Unit (GPU) to perform generic computation typically done by the CPU. CUDA is a language developed for NVidia graphics cards to support GPGPU computing.

area of interest. While some existing tools used by media artists are described in Chapter 2, LUNA shows that it is possible to develop integrated systems that bring together the benefits of different creative groups. In particular, LUNA was used to explore the combination of procedural modeling and interactive output for live performance.

To examine whether or not an integration of different creative techniques is possible, this work starts by examining dimensions of interest to media artists. LUNA was then designed and evaluated according to these creative dimensions, discussed in Chapters 5 and 6, using the criteria set out by the Creative Support Tools workshop held by the NSF in 2005. Briefly, these criteria measure creative support tools by 1) low threshold, giving users “immediate confidence that they can succeed” through simple interfaces, 2) high ceiling, providing tools that are “powerful and complete”, and 3) wide walls, meaning that the tool “suggests a wide range of explorations [Resnick et al., 2005].”

These results are briefly summarized here:

1) Programming - Some media artists are interested in programming, while others are not. LUNA addresses this in the same way that Houdini does, by offering programmers the ability to author nodes in a low-level language (C/C++). This allows artists to work on at least two different programming levels in LUNA, low-level authorship and the high-level interface. In the future, it may also be possible to incorporate a scripting language in LUNA, similar in presentation to Processing’s integrated development environment. LUNA’s visual language provides possibly the lowest threshold of any other

dataflow language for conceptual artists, since minimalist icons and smart connections allow users to very quickly build complete projects.

2) Modality and Media - Artists seek to work with a range of different media types, including images, audio, video, and geometric shapes. LUNA explores the media of geometry; points, lines, curves and surfaces while also including images and materials (surface styles). In the future, audio, video and other data may be added. The range of media available relates to the criteria of *high ceiling*, i.e. the power of the system, which from a language perspective is unbounded as future authors may continue to contribute to it. From a content perspective, the currently available nodes developed for this dissertation provide a set of media types focused primarily on procedural modeling.

3) Live Performance - Unlike offline tools designed to support commercial film, live performance tools must necessarily give instantaneous output. Live performance is another way in which LUNA provides a high ceiling, allowing for a more "complete" solution meeting the needs of a particular group of artists through the ability of the system to render full screen, high quality results across multiple displays. Another live performance feature in LUNA is a property panel which is designed like a mixing board to allow artists to easily and interactively modifying the behavior of the system. Using deferred shading, LUNA's real time results are of high quality enough that they could be used for print or film, but a desirable (not yet implemented) future extension to the system would be to connect LUNA to other third party rendering engines such as V-Ray or MentalRay for high resolution, anti-aliased, globally shaded images.

4) Motion, Dynamics and Autonomy - Artists interested in algorithmic art, motion, and dynamics wish to work with behavior as a fundamental unit by exploring different rules, patterns, and systems for expressing behavior. This dimension is supported in LUNA by making behavior and time basic aspects of the system, visible in the range of objects currently available. Behaviors may be connected to one another, indicated through colored tabs on tiles, suggesting other possible behaviors and supporting the criteria of *wide walls* (a broad range of explorations). Although the system is partially limited in the sense that new low-level behaviors must be authored as nodes in C++, i.e. to change the rules 'under' the system, LUNA's ability to interconnect different nodes together allows for a new kind of authorship which changes behaviors 'over' the system by relating complete dynamic systems to one another.

5) Structure and Surface - Artists with an interest in sculptural form, generative art, and procedural modeling wish to construct structures. This dimension, as well as the previous one of dynamics and behavior, further expands the current ceiling of the system by introducing new objects. Unlike other live performance tools such as Max/MSP or VVVV, procedural modeling of three-dimensional structures is an implicit part of the LUNA language. Users can create forms based on simpler shapes (such as points or curves), or generate a multitude of different forms (e.g. using the scatter node). As with offline tools, surfaces can be shaded using materials and textures (written in the language Cg) allowing the user to decide on both the form and the appearance of objects.

6) Image and Idea - A final dimension explored here relates to the semantic content of the image. Discussed in relation to Jörg Schirra's Computational Visualistics, Chapter 6 introduces a new theory of digital semantics which associates the digital model with the digital image, the 'word' (or concept), and their physical counterparts. Whereas rendering is viewed as a relationship between the digital model and the image, areas such as image synthesis, computer vision, and symbolic logic are presented as new paths for creative exploration in LUNA. Artists such as Harold Cohen and Golan Levin use the image in these ways, and a number of projects experimenting with LUNA for image synthesis of hand drawn images are presented. Finally, the potential to expand into computer vision is discussed via not-yet-implemented future examples of graphs in LUNA.

The flexibility of the system to meet the needs of artists is demonstrated via a number of cross-disciplinary projects. *Presence*, a collaboration with Dennis Adderton and Jeff Elings, was presented at the University of California Davidson Library to show interactive, 360 degree panoramic photographs of natural spaces that change as the viewer walks past them. *Blocks*, a collaboration with Mark Zifchock, is a video game project started in 2002. It consists of a massive world of cubes for creating functional bridges, machines, and logic puzzles. *The Bones of Maria* is a series of experiments in generative art exploring our psychological relationship to the body, exhibited online at The Cultor (Torino, Italy). Finally, *Synthetic Renering* is a collaboration with Mock (Panuakdet) Suwannatat and Tobias Höllerer, based on the work of Gabe Luna, Geoffrey

Lewis, and Steve Fisher (Neuroscience Research Institute, Univ. of California Santa Barbara), with B.S. Manjunath (Dept. of Electrical and Computer Engineering), to develop a system that can model and replicate the visual appearance of microscopic images of astrocyte cells. These projects are described in more detail in Chapter 3.

LUNA demonstrates that artists working with different techniques do not necessarily need to work with different tools. An often-heard view is that having many tools is beneficial to the creative community as each tool supports a different method of working, with each artist choosing a tool to engage in a particular format. A problem with this view is that it requires media artists to learn a number of different systems to become proficient in their field across a range of techniques. Additionally, some projects may require techniques that can only be found in two different tools. While exposure to multiple tools may still be beneficial for educational reasons, the idea that media artists *must* learn many tools because this is the only way to explore the digital arts is refuted by LUNA. LUNA is an environment in which several, previously disparate methods in media arts are integrated into a single framework, as demonstrated through the creative dimensions supported above. Although one argument against such integrated frameworks is that artists may wish to program their own tools, this is partly addressed by LUNA since artists are free to author their own objects through the system (i.e. writing low-level nodes), similar to the community of developers that surrounds Max/MSP or Processing. Another argument may be that artists are not interested in monolithic integrated systems, but rather in 'breaking' or undoing the tools they use.

This, however, could be considered an argument against any form of tool-use, and any system can be hacked if desired.

While artists naturally work in a variety of different ways, this dissertation supports the idea that communities of artists are more capable of exchanging techniques, methods and ideas when the tools they use are sufficiently integrated so that basic media types can flow between them. Integrated frameworks such as LUNA show this can be achieved by presenting a system rich enough to allow different styles and forms of output; forms such as procedural modeling that previously existed only in offline tools. Although artists are always free to choose how they work, communities that use common tools have an advantage in the sense that they can share experiences without continually reinventing the wheel, so to speak. This can be observed in the growing communities of artist-engineers who exchange ideas and code. LUNA is presented with the hope that media artists using it will overlap with other groups experimenting with different output formats and methods of presentation while continuing to explore a growing range of interesting ideas.

Bibliography

[Anderson, 2008] Anderson, D. (2008). Groboto. http://www.groboto.com, accessed June 2010. Published by Braid Arts Labs.

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

[Assiter, 1984] Assiter, A. (1984). Althusser and structuralism. In *The British Journal of Sociology*, volume 35, pages 272–296.

[Bandyopadhyay et al., 2001] Bandyopadhyay, D., Raskar, R., and Fuchs, H. (2001). Dynamic Shader Lamps: Painting on Movable Objects. In *ISAR '01: Proceedings of the IEEE and ACM International Symposium on Augmented Reality (ISAR'01)*, page 207, Washington, DC, USA. IEEE Computer Society.

[Bannink, 2009] Bannink, P. (2009). Houdini in a games pipeline. In *SIGGRAPH '09: SIGGRAPH 2009: Talks*, pages 1–1, New York, NY, USA. ACM.

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

[Bay et al., 2006] Bay, H., Fasel, B., and Gool, L. V. (2006). Interactive Museum Guide: Fast and Robust Recognition of Museum Objects. In *Proc. Of Intl. Workshop on Mobile Vision*.

[Bethel, 1999] Bethel, W. (1999). Scene Graph APIs: Wired or Tired? Panel discussion, SIGGRAPH 1999.

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

[Blinn and Newell, 1976] Blinn, J. F. and Newell, M. E. (1976). Texture and reflection in computer generated images. *Commun. ACM*, 19(10):542–547.

[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.

[Candy and Edmonds, 2002] Candy, L. and Edmonds, E. (2002). *Explorations in art and technology.* Springer-Verlag, London, UK.

[Cardelli, 1985] Cardelli, L. (1985). Fragments of Behavior. In *Personal Communication. DEC Systems Research Center*, Palo Alto, CA.

[Carlson, 2010] Carlson, W. (2010). Animation Software Companies and Individuals. http://design.osu.edu/carlson/history/tree/ani-software.html, accessed June 2010. Lecture notes.

[Cerqueira et al., 1999] Cerqueira, R., Cassino, C., and Ierusalimschy, R. (1999). Dynamic component gluing across different componentware systems. *Proceedings of the International Symposium on Distributed Objects and Applications*, pages 362–371.

[Cohen, 1979] Cohen, H. (1979). What is an image? In *Conference Proceedings of IJCAI 1979.*

[Conway and Pausch, 1997] Conway, M. J. and Pausch, R. (1997). Alice: easy to learn interactive 3D graphics. *SIGGRAPH Comput. Graph.*, 31(3):58–59.

[Curless and Levoy, 1996] Curless, B. and Levoy, M. (1996). A volumetric method for building complex models from range images. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312, New York, NY, USA. ACM.

[Cycling74, 2010] Cycling74 (2010). Max/MSP software. http://cycling74.com/, accessed Oct 2010.

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

[DeBry et al., 2002] DeBry, D., Gibbs, J., Petty, D. D., and Robins, N. (2002). Painting and rendering textures on unparameterized models. *ACM Trans. Graph.*, 21(3):763–768.

[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.

[Deussen and Lintermann, 2004] Deussen, O. and Lintermann, B. (2004). *Digital Design of Nature: Computer Generated Plants and Organics.* SpringerVerlag.

[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.

[Forbes et al., 2010] Forbes, A., Hollerer, T., and Legrady, G. (2010). behaviorism: A Framework for Dynamic Data Visualization. In *Proceedings of InfoVis 2010. Oct 24-29th.*, Salt Lake City, Utah. IEEE.

[Foulser, 1995] Foulser, D. (1995). IRIS Explorer: a framework for investigation. *SIGGRAPH Comput. Graph.*, 29:13–16.

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

[Ganster and Klein, 2007] Ganster, B. and Klein, R. (2007). An Integrated Framework for Procedural Modeling. In *Spring Conference on Computer Graphics*, pages 150–157, Comenius University, Bratislava.

[Giden et al., 2008] Giden, V., Moeller, T., Ljung, P., and Paladini, G. (2008). Scene graph-based construction of CUDA kernel pipelines for XIP. *Proceedings of High-Performance Medical Image Computing and Computer Aided Intervation (HP-MICCAI) Workshop, Sept 2008.*

[Gleick, 1987] Gleick, J. (1987). *Chaos: Making a New Science.* Viking. Donnelley & Sons, Harrisonburg, Virginia.

[Gould, 2002] Gould, D. (2002). Complete Maya programming - An extensive guide to MEL and the C++ API. Elsevier, San Francisco.

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

[Green and Petre, 1996] Green, T. R. G. and Petre, M. (1996). Usability analysis of visual programming environments: A 'cognitive dimensions' framework. *Journal of Visual Languages and Computing*, 7(2):131–174.

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

[Hansmeyer, 2010] Hansmeyer, M. (2010). Platonic Solids. http://www.michael-hansmeyer.com/, accessed Oct 2010.

[Harrison, 2007] Harrison, D. (2007). Evaluation of Open Source Scene Graph Implementations. Technical Report. Visualization & Virtual Reality Research Group.

[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.

[Heidegger, 1982] Heidegger, M. (1982). *The Question Concerning Technology, and Other Essays*. Harper Perennial.

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

[Hoetzlein and Adderton, 2009] Hoetzlein, R. C. and Adderton, D. (2009). MINT/VFX - A High-Performance Computing Framework for Interactive Multimedia.

[Hoetzlein and Castellanos, 2006] Hoetzlein, R. C. and Castellanos, J. (2006). Monarch: Interface for a graphical programming language for digital artists.

[Hoetzlein and Schwartz, 2005] Hoetzlein, R. C. and Schwartz, D. I. (2005). GameX: a platform for incremental instruction in computer graphics and game design. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Educators program*, page 36, New York, NY, USA. ACM.

[IBM, 1999] IBM (1999). IBM Visualization Data Explorer. The proprietary IBM Visualization Data Explorer became open source in 1999 as OpenDX. Viewed at http://www.research.ibm.com/dx/, accessed June 2010.

[Jacucci et al., 2005] Jacucci, G., Oulasvirta, A., Salovaara, A., Psik, T., and Wagner, I. (2005). Augmented Reality Painting and Collage: Evaluating Tangible Interaction in a Field Study. In *Human-Computer Interaction, INTERACT 2005*, pages 43–56, Heidelberg, Berlin. Springer.

[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.

[Johnston et al., 2004] Johnston, W. M., Hanna, J. R. P., and Millar, R. J. (2004). Advances in dataflow programming languages. *ACM Comput. Surv.*, 36(1):1–34.

[Jones and Nevile, 2005] Jones, R. and Nevile, B. (2005). Creating Visual Music in Jitter: Approaches and Techniques. *Comput. Music J.*, 29(4):55–70.

[Kawaguchi, 1982] Kawaguchi, Y. (1982). A Morphological Study of the Form of Nature. In *Computer Graphics*, volume 16.

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

[Krueger, 1991] Krueger, M. K. (1991). *Artificial Reality 2, 2nd ed.* Addison-Wesley Professional.

[Kuhn, 1962] Kuhn, T. (1962). *The Structure of Scientific Revolutions.* University of Chicago Press.

[Lawrence, 2004] Lawrence, J. (2004). A painting interface for interactive surface deformations. *Graph. Models*, 66(6):418–438.

[Levin, 2006] Levin, G. (2006). Computer Vision for Artists and Designers: Pedagogic Tools and Techniques for Novice Programmers. In *Journal of Artificial Intelligence and Society*, volume 20. Springer Verlag.

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

[Lindenmayer, 1968] Lindenmayer, A. (1968). Mathematical models for cellular interactions in development. In *Journal of Theoretical Biology*, volume 18, pages 280–315.

[Lorenz and Dollner, 2008] Lorenz, H. and Dollner, J. (2008). Dynamic Mesh Refinement on GPU using Geometry Shaders. In *WSCH 2008 Full Papers Proceedings*, pages 97–104.

[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.

[Manovich, 2007] Manovich, L. (2007). Abstraction and Complexity. In *MediaArtHistories.*, Cambridge. 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.

[Mather, 2006] Mather, D. (2006). An Aesthetic of Turbulence: The Works of Ned Kahn. In *Sarai Reader 06: Turbulence. Sarai Media Lab, Delhi.*

[May et al., 1996] May, S. F., Carlson, W., and Phillips, F. (1996). AL: A Language for Procedural Modeling and Animation.

[McMillan and Bishop, 1995] McMillan, L. and Bishop, G. (1995). Plenoptic Modeling: An Image-Based Rendering System. In *Proceedings of ACM SIGGRAPH '95*, Los Angeles, CA.

[Meso, 1998] Meso (1998). VVVV: A multipurpose toolkit. http://www.meso.net/vvvv, accessed June 2010.

[Miró, 2008] Miró, J. (2008). Joan Miró: Painting and Anti-Painting 19271937. Exhibition catalog.

[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.

[Müller et al., 2006] Müller, P., Wonka, P., Haegler, S., Ulmer, A., and Gool, L. (2006). Procedural modeling of buildings. In *Transactions on Graphics*, volume 25, page 614623.

[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.

[Peirce, 1866] Peirce, C. S. (1866). The Fixation of Belief. 12:115.

[Peirce, 1931] Peirce, C. S. (1931). *Collected Papers of C.S. Peirce.* 8 vols., Harvard University Press. Hartshorne, C., Weiss, P. and Burks, A., editors., Cambridge, MA.

[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.

[Reeves, 1983] Reeves, W. T. (1983). Particle Systems - a Technique for Modeling a Class of Fuzzy Objects. *ACM Trans. Graph.*, 2(2):91–108.

[Reeves et al., 1990] Reeves, W. T., Ostby, E. F., and Lefler, S. J. (1990). The menv modelling and animation environment. In *Journal of Visualization and Computer Animation*, volume 1, pages 33–40.

[Resnick et al., 2009] Resnick, M., Maloney, J., and Monroy-Hernandez, A. (2009). Scratch: Programming for All. *Commun. ACM*, 52(11).

[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.*

[Reynolds, 1987] Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 25–34, New York, NY, USA. ACM.

[Rhee et al., 2006] Rhee, T., Lewis, J., and Neumann, U. (2006). Real-time Weighted Pose-Space Deformations on the GPU. In *Computer Graphics Forum*, volume 25, pages 439–448.

[Rohlf and Helman, 1994] Rohlf, J. and Helman, J. (1994). IRIS Performer: A high performance multiprocessing toolkit for real-time 3D graphics. *Computer Graphics, Proceedings of ACM SIGGRAPH '94*, pages 381–394.

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

[Russet and Starr, 1976] Russet, R. and Starr, C. (1976). *Experimental Animation: Origins of a New Art.* Da Capo Press.

[Ryokai et al., 2004] Ryokai, K., Marti, S., and Ishii, H. (2004). I/O brush: drawing with everyday objects as ink. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 303–310, New York, NY, USA. ACM.

[Saussure, 1965] Saussure, F. d. (1965). *Course in General Linguistics.* McGraw-Hill Humanities/Social Science/Languages.

[Schirra, 2005] Schirra, J. R. (2005). *Foundation of Computational Visualistics.* Deutscher Universitats-Verlag/GWV Fachverlage GmbH, Wiesbaden.

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

[Shapiro, 1961] Shapiro, A. (1961). *Illustrated Experiments in Fluid Mechanics.* MIT Press.

[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.*

[Snyder, 1992] Snyder, J. (1992). Generative modeling for computer graphics and CAD: symbolic shape design using interval analysis. In *Academic Press Professional, San Diego.*

[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.

[Stiny and Gips, 1971] Stiny, G. and Gips, J. (1971). Shape grammars and the generative specification of painting and sculpture. In *IFIP Congress 1971. North Holland Publishing.*

[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.

[Sutherland, 1988] Sutherland, I. E. (1988). Sketchpad a man-machine graphical communication system. In *25 years of DAC: Papers on Twenty-five years of electronic design automation*, pages 507–524, New York, NY, USA. ACM.

[Szirmay-Kalos and Umenhoffer, 2006] Szirmay-Kalos, L. and Umenhoffer, T. (2006). Displacement Mapping on the GPU - State of the Art. In *Proceedings of Eurographics*, volume 25, pages 1–24.

[Takacs et al., 2008] Takacs, G., Chandrasakhar, V., Gelfand, N., Xiong, Y., Chen, W., Bismpigiannis, T., Grzeszczuk, R., Pulli, K., and Girod, B. (Oct 30-31, 2008). Outdoors Augmented Realty on Mobile Phone using Loxel-Based Visual Feature Organization. In *ACM MIR '2008.*, Vancouver, Canada.

[Torrence, 2006] Torrence, A. (2006). Martin Newell's original teapot. In *SIGGRAPH '06: ACM SIGGRAPH 2006*, page 29, New York, NY, USA. ACM.

[Turk and Levoy, 1994] Turk, G. and Levoy, M. (1994). Zippered polygon meshes from range images. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 311–318, New York, NY, USA. ACM.

[Upson, 1989] Upson, C. (1989). The Application Visualization System: a computational environment for scientific visualization. In *IEEE Computer Graphics and Applications.*

[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.

[Wainwright, 1974] Wainwright, R. T. (1974). Life is universal! In *Proc. of the 7th Winter Simulation Conference.*, pages 449–459, Washington, DC.

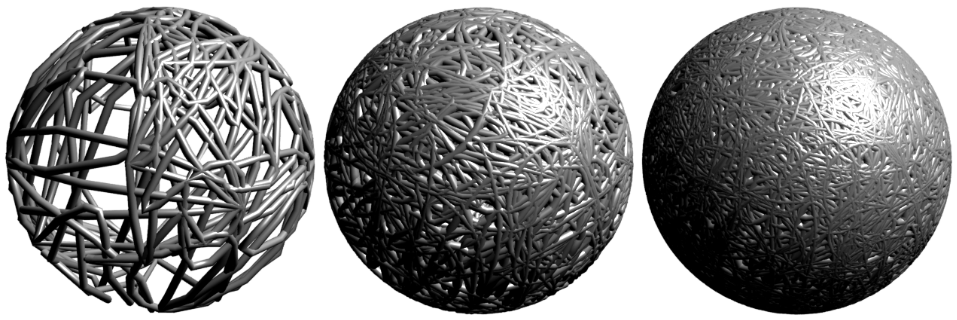
[Wands, 2007] Wands, B. (2007). *Art of the Digital Age.* Thames & Hudson.

[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.

[Winograd, 1971] Winograd, T. (1971). Procedures as a Representation for Data in a Computer Program for Understanding Natural Language. In *MIT AI Technical Report 235.*

Appendix A

Reference Model



Model	P	C	K	V	U	Verts	Tris
Low res	50	25	8	28	8	5,600	9,450
Med res	150	100	8	42	12	50,400	90,200
High res	250	200	8	56	16	179,200	330,000

Figure A.1: Woven sphere reference model with parameter values for low, medium and high resolution models.

The woven sphere is a procedural model defined as follows. Input consists of a particle system with P points, generated randomly in a box from (-1,-1,-1) to (1,1,1) and moving with a uniform velocity of 0.0025 in a random direction (arbitrary units, time step is 1.0). As the points animate, they reflect off boundaries to remain inside the initial volume. Described in LUNA notation:

PSYS$_{points}$ (P, init_min $< -1, -1, -1 >$, init_max $< 1, 1, 1 >$, init_vel $< 0.0025 >$)

From these points, random subsets are selected in groups of K to become the CV control keys of C Bézier spline curves. The Bézier curves are sampled to a resolution of V total sample vertices per curve. The curve order is 3 (cubic). The function is:

SUBSET$_{curves}$ (POINTS$_{points}$, num_keys K, num_curves C, num_samples V)

This generates C curves with K keys and V sampled points in each. These curves are then spherified to a unit sphere (radius 1) by normalizing the points in each curve. Note that it is incorrect to normalize the CV keys as the resulting curve may still penetrate the sphere. The spherify function should operate on the final sampled points to guarantee the sampled curve lies on the sphere. In procedural modeling terms, the spherify function takes any geometric object (points, curves, meshes) and normalizes its verticies. It is a typeless function defined by p' =$|p|$:

SPHERIFY (OBJ)

Finally, loft surfaces are generated by sweeping a circle along the curves. A circle of radius 0.025, sampled with U verticies, is used as the cross-section. The paths are the spherified curves of the previous step. The loft surface has a cylindrical topology with only triangular faces, and no end caps. This produces a total of U*V verticies per loft, and C*U*V verticies for the entire woven sphere object, with 2(U-1)(V-1) triangles per loft, and 2(U-1)(V-1)C triangles for the whole object.

CIRCLE_{curve} (samples U)

LOFT_{mesh} (PATH_{curves}, SHAPE_{curve})

The total function is:

LOFT_{mesh} (SPHERIFY(SUBSET_{curves} (PSYS_{points}(P, init_vol, init_vel), K, C, V)), CIRCLE_{curve} (U))

Parameter values and sample representations for the low, medium and high-res models used in our tests can be found in Figure A.1. For render performance testing in real-time systems, it should be rendered at 1024x768 using a single Phong light source and no shadows or anti-aliasing. When reporting results, ideally evaluation should be separated from render time. Animation of the underlying particle system causes the curves to gradually morph along the sphere surface.

Appendix B

Houdini Interaction Study

Results of the interface test in Houdini for the reference model are shown here. No prior knowledge of Houdini is assumed, although the author is familiar with procedural modeling concepts. In total, it took around 4 hours to create this model in Houdini.

Elps Time	Task Time	Description
0:02	2 min	Figure out how to create objects (must press enter)
0:08	6 min	Cannot use Source on Particles (only Fluids)
0:13	5 min	Source for Geometry used to emit particles. Exploration of help docs to find that Emission type parameter can be set to Volume.
0:44	31 min	Trying to figure out how to build a curve from particles. No obvious function to generate curve from points. Found an online forum: ”moving curves points to the particle locations using a Point SOP”
0:59	15 min	Determined how to connect object sub-graphs to one another. Incorrect assumption about how Houdini works.
1:36	37 min	Output: Now produces points moving on surface of a sphere. Created a point SOP to shrink points to a sphere. Learned that top-level graphs are not flow networks, but heirarchy networks. So it is not possible to connect object sub-graphs. Must copy nodes into an object’s flow graph.

1:51	15 min	Moved the 'spherify' node after the curve input, to properly match reference model. Attempting to use the Copy Stamping method to generate many curve instances, after further reading of documentation.
2:06	15 min	Discovery that graphs in Houdini compute entire objects first. I should not generate multiple curves, but generate a complete curve-loft, then replicate.
2:18	12 min	Skin Output of the Sweep SOP is not producing output (see next step).
2:36	18 min	Circle primitive type was changed from Primitive to Polygon in order to generate swept surfaces, explaining why Sweep SOP appeared not to work.
2:56	20 min	Curve points are not yet spherified, only control keys. To spherify curve itself, a Convert operator is introduced to make a Polyline.
3:01	5 min	Output: Now produces curves moving on surface of sphere. Determining relation between Level of Detail and number of points generated, as I cannot precisely control the curve sampling.
3:24	23 min	Found that 'stamp' instancing was not being used correctly. Took time to figure out it must be an expression of the form: point("particles", $PT +
3:34	10 min	Some time lost due to object path naming. Interface automatically inserts paths like "obj/group/particles/"
3:51	17 min	Output: Complete graph is working, with curves becoming loft tubes. There is probably a more efficient method than Copy stamping for this task. Cannot stop it from translating curves to the particle locations.
3:54	3 min	To solve the translation problem, a noded is added to scale all particles by (0,0,0), causing a null translation during in the Copy to Points node.
3:54		Output: Produces results that match the reference model.