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 60s 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 Kenneth Knowlton'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 modelling 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. These are often used in education, 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 wonders 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, they may need to invest addition time in a new language. Second, some tools are better at certain tasks than others, which may force the artist to switch tools. Maya, for example, provides extensive support for human character modelling while Processing does not. This means that artists who enjoy Processing are either forced to switch tools or must implement such 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 which are not oriented toward real-time interaction and output. Finally, some features critical to particular groups of artists, such as those interested in multi-screen output for example, 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 [**?**]. However, this does not address the fact that certain structures are common across several tools, and therefore in conflict with one another. For example, Maya supports character modelling, but uses its own propriety 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 questions 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, for example.

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 methpor, 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 ceative perspective it would be ideal if one could use the features of each without having to learn each system.

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 nice 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 answers 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). The may be understood as an inherent aspect of the art object coming into being through a media, and may be analysed 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, for example, 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 silver itself, or the shape of the chalice, or the purpose or message it conveys. The following definitions help to distinguish *inherent constraints* from *creative constraints* in examining digital tools.

> *Inherent constraints*: Rules imposed by the media 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, and this is a creative constraint, but once the choice is made the tool brings with itself its own 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, that were 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. They are positive factors in that they are conscious, free choices by the artist. For example, to upset the cultural status of painting, Joan Miró choose 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.

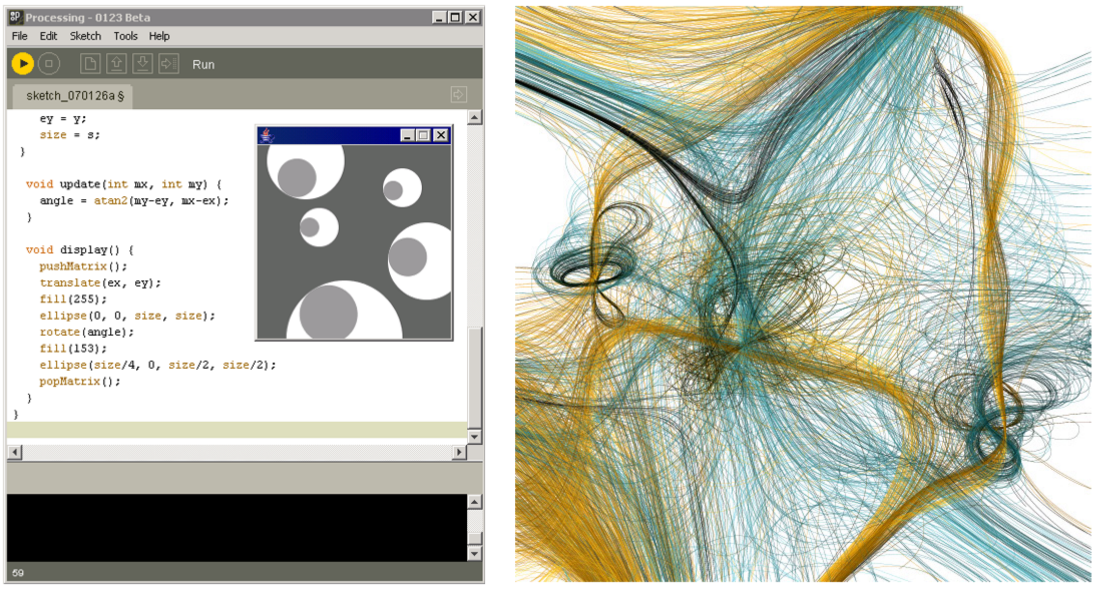
Thus, when speaking of software tools, the *inherent constraints* are those which are most deeply embedded in the system, and one way to conceptualize them is that they remove the element of choice from the artist. When choosing a tool the artist may not know all of the constraints involved, but after a time they learn that some inherent constraints are immovable, and they must either contact the tool developers, re-engineer the system, or switch tools. 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 analysing 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 constraint what are the inherent constraints of a given tool? While a "feature set" describes the unique capabilities of a tool (beyond its 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”

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 project, examples of which can be found on the processing.org website. Due to the authors backgrounds in information visualization, projects created with Processing tend to have an information aesthetic. This may be partly due to the authors backgrounds, but also to the 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 (which is C/C++). However, Java is a hardware-independent language, which means that Processing projects are more capable of being run directly on the web in a 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, 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 (camera tracking, LEDs), and have used Processing in exhibitions worldwide.

In understanding the inherent constraints of a tool, the best resource is a language reference. A online reference shows the base functionality that Processing offers[1]. 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 abstraction two-dimensional shapes, and its ease of use as a programming language, which can be seen in project samples (see Figure 1.1).

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,

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

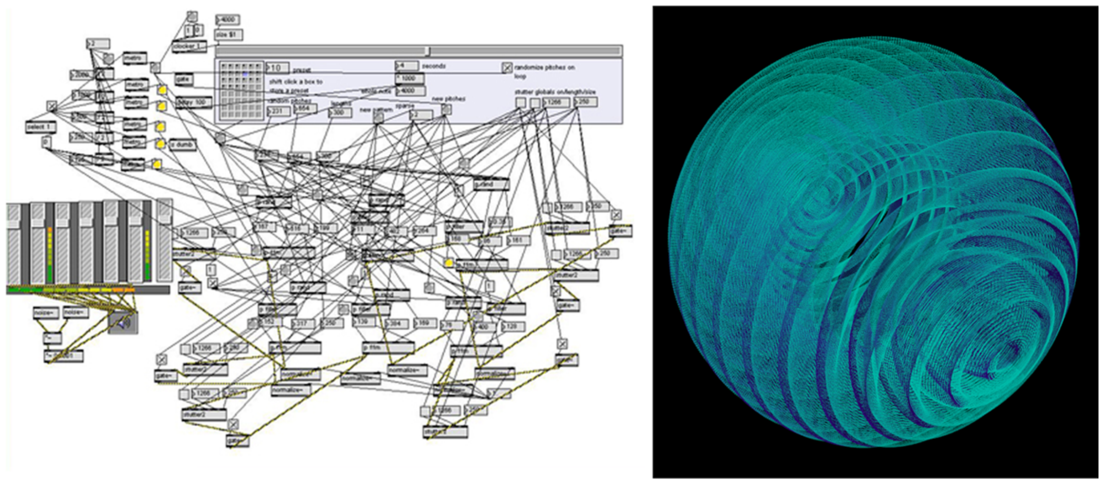


Figure 2.2: Max/MSP, software by Cycling 74, developed by Miller Puckette with visuals using Jitter developed by Joshua Kit Clayton in 2003. Artwork by Christopher James, 2006 from Third Space Mind.

Max/MSP/Jitter is considered here due to the introduction of Jitter in 2003 by Joshua 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, its 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, which will be higher performance than Processing. 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 Max/MSP is 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

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 equal difficult as with Max/MSP.

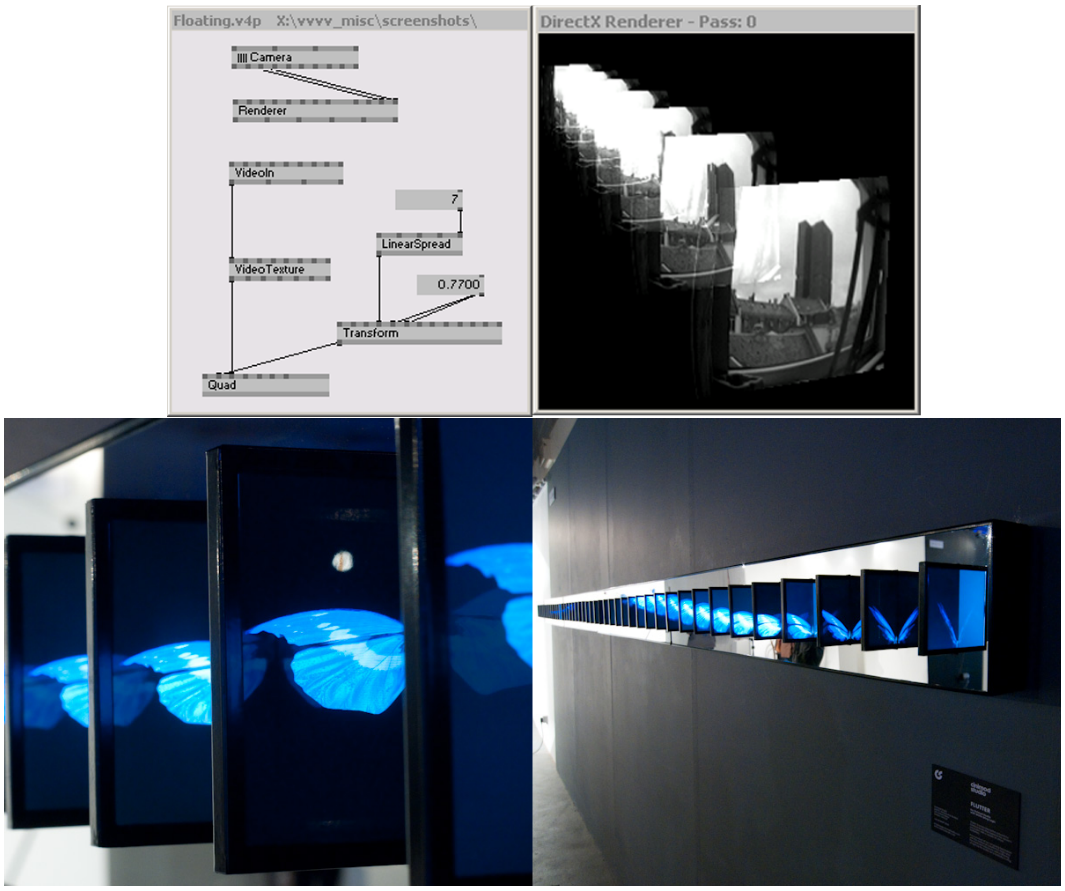


Figure 2.3: VVVV, software for media artists developed by Meso studios and made freely available in 2002. The butterfly sequence, entitled Flutter, is composed on 88 double-sided screens using VVVV by Cinimod Studio, 2010.

VVVV is best suited to large scale, interactive, visual installations. The Galería (http://vvvv.org) shows a number of major projects created with VVVV as well as many gallery installations. VVVV may be considered an installation tool as its workflow and user modules are focused on real-time imagery. A tutorial, for example, 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 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, like Max/MSP, must be coded in another language (HLSL) by the artist.

VVVV is unique as a tool for visual media artists, and as a visual programming language it is easy to create projects quickly. 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.

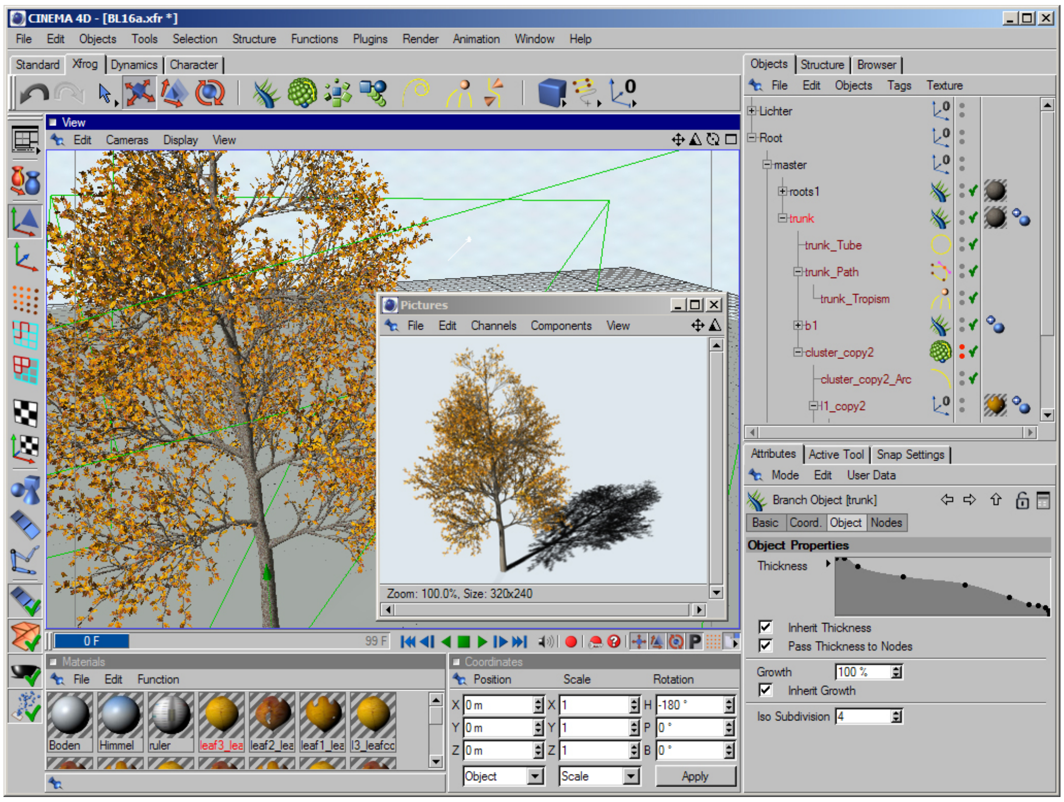

CINEMA 4D - [BL16a.xfr *]
File Edit Objects Tools Selection Structure Functions Plugins Render Animation Window Help
Standard Xfrog Dynamics Character
View
Pictures
Zoom: 100.0%, Size: 320x240
Objects Structure Browser
Lichter
Root
master
roots1
trunk
trunk_Tube
trunk_Path
trunk_Tropism
b1
cluster_copy2
cluster_copy2_Arc
l1_copy2
Attributes Active Tool Snap Settings
Branch Object [trunk]
Basic Coord. Object Nodes
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_leafco
Coordinates
Position Scale Rotation
Object Scale 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.

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.

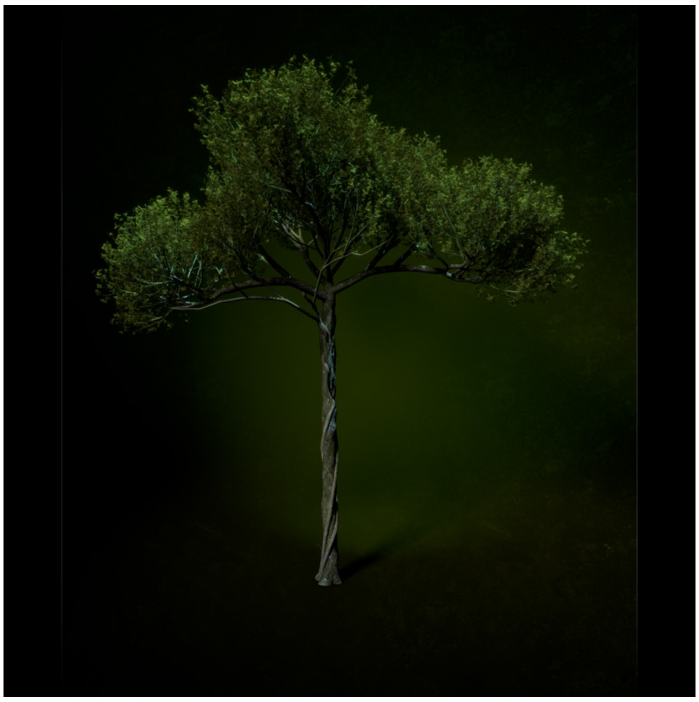


Figure 2.5: *Kleine Spielerei*, by Jan Walter Schliep (2009), demonstrates high quality renderings produced using Xfrog.

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

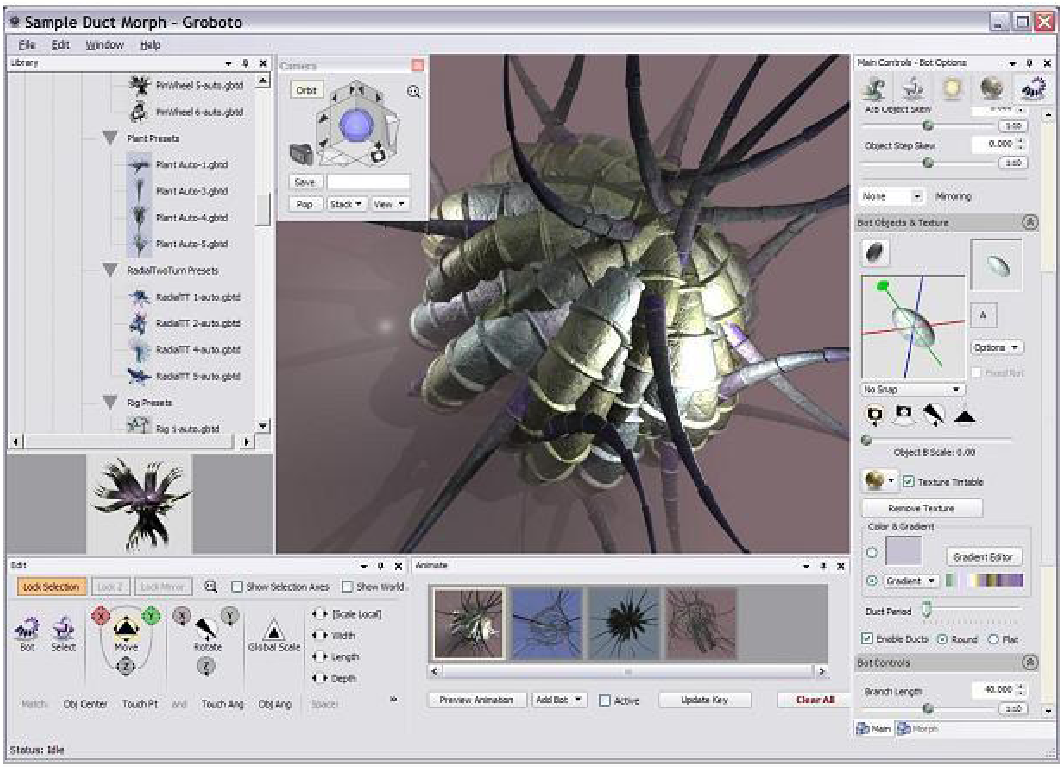


Figure 2.6: Groboto, created by BRAID Media Artists and Darrel Anderson.

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 in the next section for example, 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 Grobot 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 flag-ship software product of Side Effects Software. A member of the original CGI film companies which include Wavefront, Alias, Autodesk and Softimage, Side Effects Software was developed to support the special effects industry. Houdini is mentioned here because of its unique support for procedural modeling (rather than scene-based modeling), and has been used in a wide number of feature films.

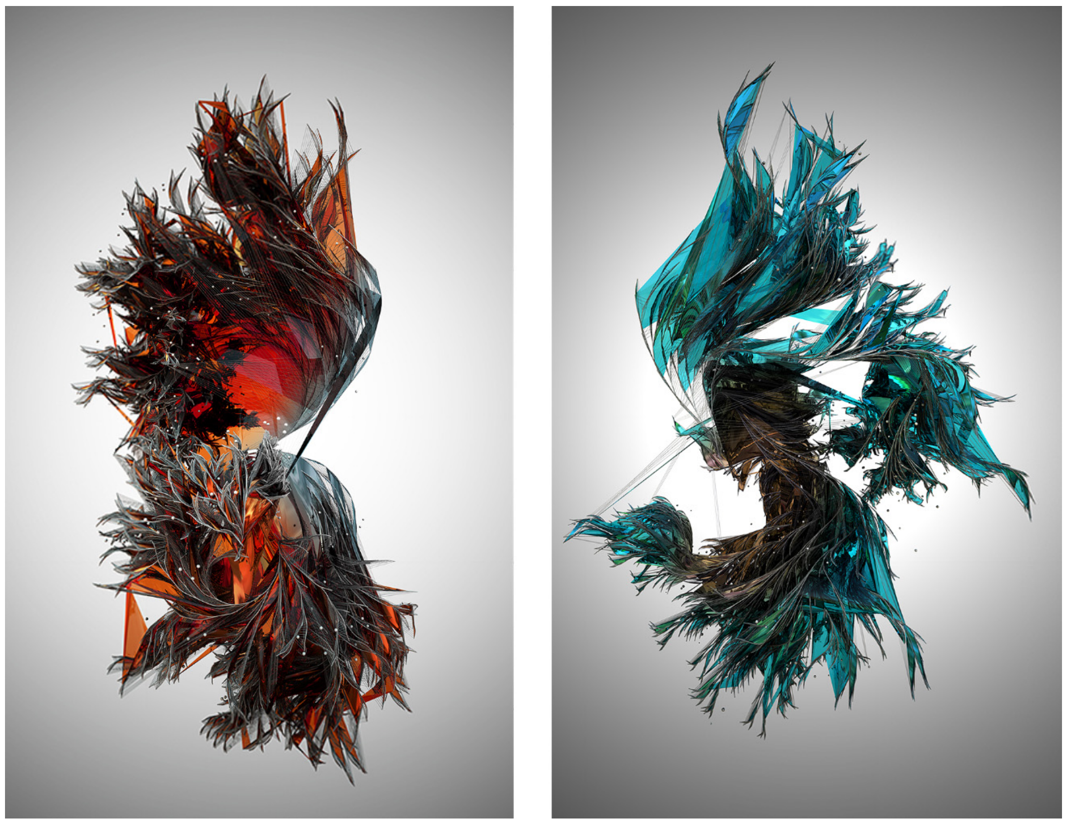


Figure 2.7: *Gestrüpp*, by depotVisuals GbR (2010), demonstrates complex, organic modeling created using Houdini 10, by Side Effects Software.

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 the benefits are 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-world 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].

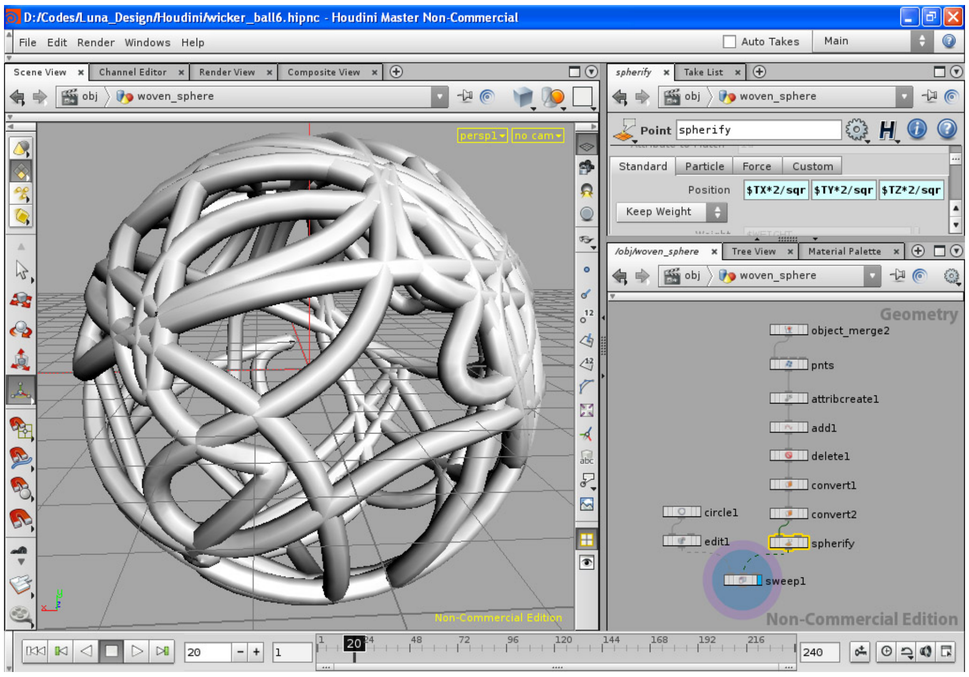


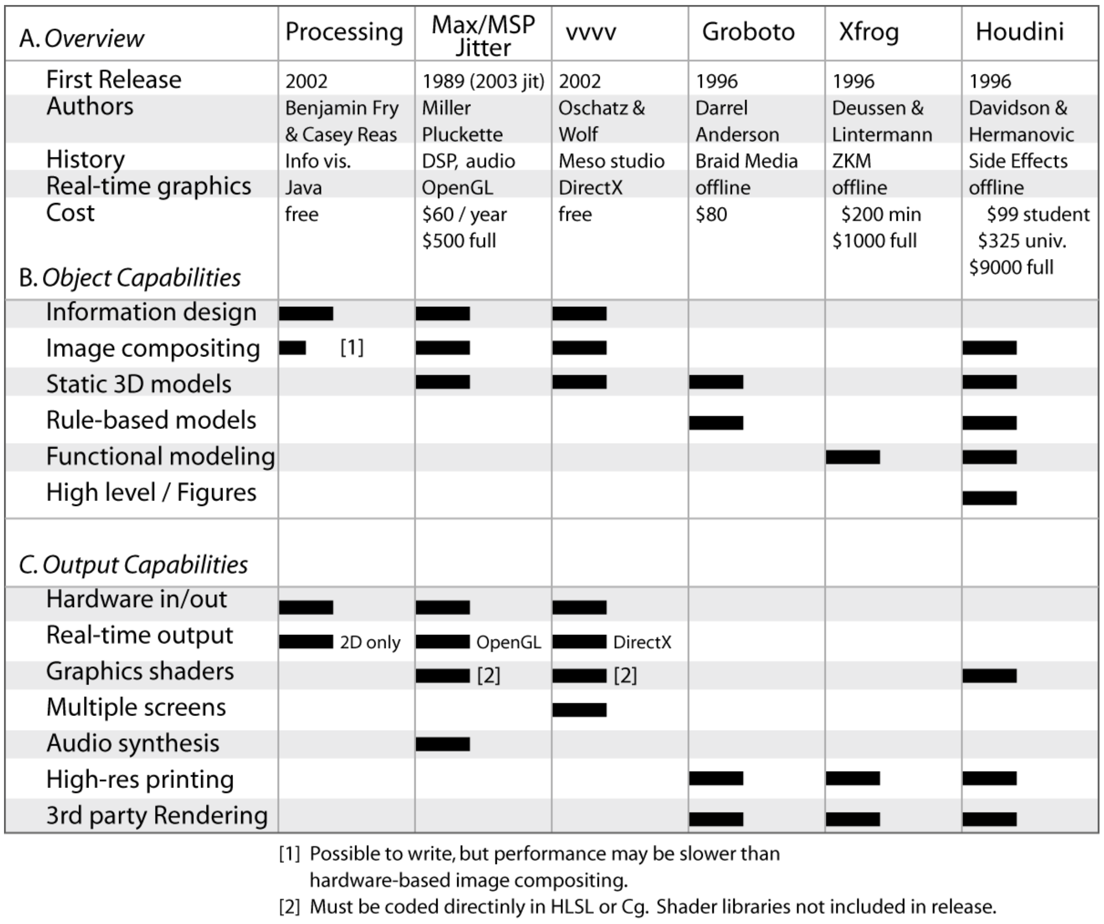
Figure 2.8: Interface to Houdini 10, showing a reference model used later in this thesis (see Chapter 4).

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 behaviour 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 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 1.9. 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.

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



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.

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 can require significant overhead in development time for the artist or engineer, and makes poor use of system resources. Multiple screen rendering is an on-going 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 using 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, this implies that media artists will gather around 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 this full range, yet the current situation is equally challenging as current tools do not necessarily provide the right set of features needed to explore an 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, we can see that learning a low-level language introduces inherent constraints in the types of output. AARON, for example, is not a system designed for animation. 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 like 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 state 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.

Bibliography

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

[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.

[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.

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

[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.

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

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

[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.

[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.

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