*Book Chapter to appear in: Visual Languages for Interactive Computing: Definitions and Formalizations Editor: Fernando Ferri, Publisher: Idea Group Inc.*

# Aesthetic Computing: A Brief Tutorial

Paul A. Fishwick, University of Florida

## Introduction

The purpose of *aesthetic computing* is to apply the theory and practice of art and design to the field of computing. The range of aesthetics within the arts is broader than those in mathematics and computing where aesthetics is often synonymous with *optimality criteria* (i.e., elegant proof, minimal line crossings). Therefore, the new field encourages applying an artistically-based *expanded range* of "aesthetic" to basic elements of computing such as programs, models, and data.

A Dagstuhl workshop in 2002 surfaced the core areas of aesthetic computing, culminating in a recently published edited volume (Fishwick 2006). Aesthetic computing is associated with three levels of art-computing integration: *cultural*, *implementation*, and *representation*. The *cultural level* is one where computing artifacts (i.e., structures) or process (i.e., collaboration) are affected by an introduction of the expanding role of aesthetics or contact with designers and artists. The *Processing* language (Greenberg 2006) is a good example of this level, where a strong community is sustained by people who are programmers, designers, or both. Other examples of cultural infusion are the collaborations of artists with scientists (Prophet et al. 2006, Cox 2006). In this level, the practice of computing is modified through a cultural form of integration: artists and computer scientists working together in teams.

The *implementation level* creates a situation where computing artifacts have a tight *behavioral coupling* with design and art artifacts. Each computing artifact, when executed, exhibits an artistic consequence. Processing also reflects this level through its extensive documentation where concise code fragments that demonstrate a specific Java method have corresponding design equivalents. For example, the "while statement" in the online reference manual is specified through a five-line Java program. Thus, there is a tight coupling between code and design: for any piece of code, there is a commensurate, behaviorally generated, visual design. This creates a strong bond to where the visual pattern becomes partially synonymous with the *concept of conditional iteration* associated with the computing artifact. Another system that exhibits aesthetic computing via the implementation level is Alice (Dann et. al 2006) since every programming fragment can have audio or visual consequence in the 3D display space.

The final *representation level* is an extension of the implementation level since the coupling between design and computing artifacts occurs at a structural, as well as behavioral, level. The artifacts reflect each other to the extent that they can be used interchangeably, suggesting a strong level of one artifact *structurally representing* the other. Moreover, the artifact and the behavior may occupy the same human interface space, allowing one to be juxtaposed with the other, or otherwise connected through morphing or transitioning. Information visualization (Ware 2004), if expanded as done by the Processing community to span the range from pure utility to artistic freedom, is one example of the representation level. The computing artifact for information visualization is generally a simple data structure such as an array or tree; however, other possibilities emerge in representing formulae, code, and model structures. In our work, we attempt to embrace all three of these levels:

- *Cultural:* artists and computer scientists take the same classes as part of the Digital Arts and Sciences degree programs

- *Implementation:* mathematical and computing elements are taught within the context of design and art, forming new bonds between both.

- *Representation:* the Aesthetic Computing class emphasizes a process where student create different types of representations of formal structures using a variety of styles and metaphors.

Four years ago, we created the aesthetic computing class at the University of Florida with the goal of exploring the third level of art-computing integration as previously specified. The class is organized with lecture, invited speakers, student team speakers and projects. The projects are divided into four categories: 1D, 2D, 3D, and Physical. the 1D project is one where a student takes a text-based structure and represents it in text, but perhaps with embedded interaction. The 2D and 3D projects are similar, except that the visual represents are similarly limited to those dimensions. The physical project involves a juried exhibition of sculptures and prototypes. For Spring 2006, we had an exhibition in a refurbished warehouse called *WARPhaus* (Warphaus 2006). The process of representation involves a careful study of semiotics, a brief introduction to categories, basic parsing, and computational manipulation methods from analogy to simple graph transformation. Figure 1 illustrates the process.

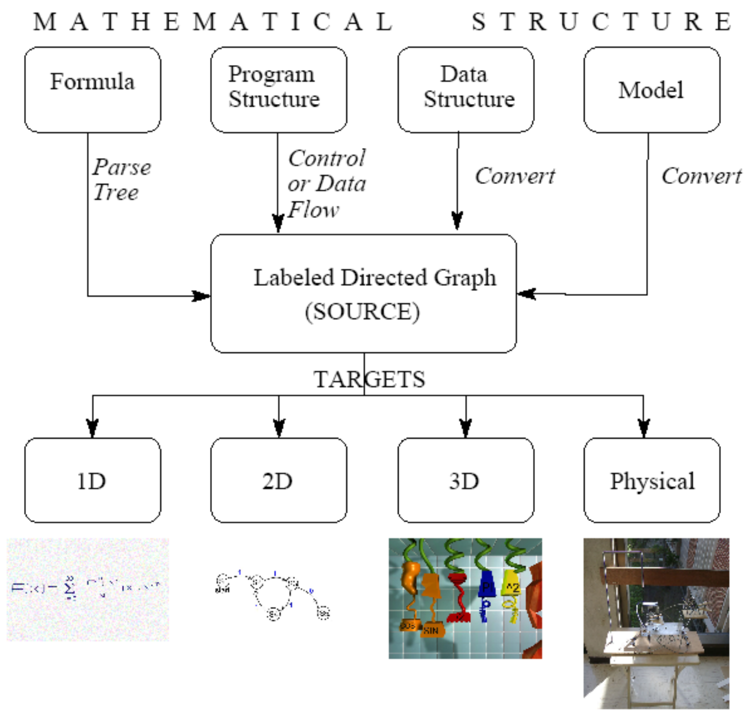


Figure 1: A simplified process flow of beginning with a source structure and applying a wide range of aesthetics to create a target structure

## A Five Step Process

Encouraging artists, computer scientists, and mathematicians to collaborate is a good way to initiate any task involving representation (i.e., cultural level). Having tools that help forge connections (i.e., implementation level) between art and mathematical structures provides additional assistance. However, we wish to focus on the representation level where the potential for representing structure is given a framework for creative exploration. This framework is defined in five steps:

1. *Identification:* one needs a source structure to represent. The following are the sorts of structures that are presented as possibilities in the class: number, variable, formula, function, program, data, and model. The first four are commonly found in K-12 mathematics, whereas program and data structures are often not introduced to students until university computing classes.

2. *Graph:* the structure identified in the first step should be represented as a fully labeled graph. As indicated in figure 1, traditionally represented formulas and functions in 1D are parsed to create graphs. Programs can be translated into data or control flow graphs. Data structures are often represented in graph form. Models of various sorts (i.e., data, information, dynamic) generally have underlying graph-based formalisms.

3. *Ontology:* even though it may seem a source graph is enough to begin a representation process, there are many missing pieces of information and knowledge that need to be clearly surfaced if the mapping is to be made clear, complete, and consistent. Semantic networks should be created to describe the ontological foundation for the source item to be represented.

4. *Map:* how does one take a source graph and an ontological framework, and actually craft a representation? A mapping must occur from source to target. This mapping begins with a determination of the sort of target to forge. Are we to transform a structure into a dance, a song, a landscape, or a cityscape? The use of metaphor is warranted, but even before specific metaphors, we employ the use of a simple table that provides ideas as to how the key parts of a graph (i.e., relations) can be mapped to concepts in *spacetime*.

5. *Representation:* the final stage is the actual representation with the assistance of the table in step 4. This is where the designer employs the most creativity.

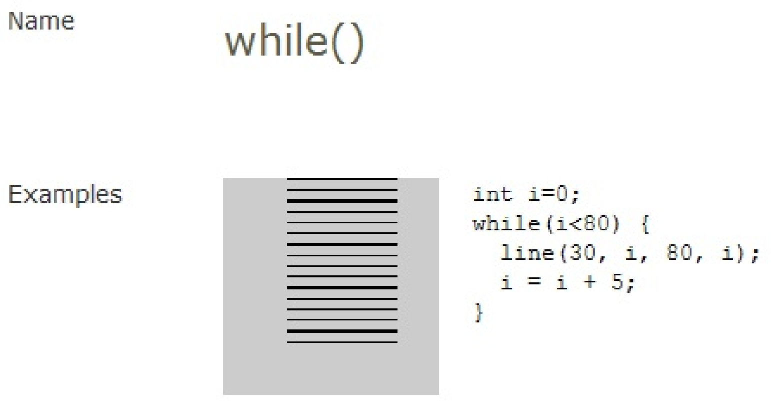
6. *Assessment:* after creating a representation, it should be critiqued through peer evaluation or heuristics.

It is certainly possible to skip one or more steps. One might immediately envision the Pythagorean theorem as a tree, the tree as a set of enclosed rings, and then go about sketching the rings or producing them from a rapid prototyping machine. However, having the steps allows us to capture a guiding *method* even if all steps are not rigorously followed.

We will use several examples to illustrate the first five steps and then provide assessment criteria for the final step. In general, like most processes for creating large products (i.e., software engineering), there is a significant iterative component where steps are revisited as required.

## STEP 1: Identification

Consider the code-design coupling shown within the Processing reference guide (Processing 2006) for the *while statement*, shown in Figure 2.



```
int i=0;
while(i<80) {
  line(30, i, 80, i);
  i = i + 5;
}
```

Figure 2: The while statement code-design coupling in the Processing reference (Processing 2006).

This example is typical of the tight relationship promoted between text-based code and a visual behavior in Processing. One can imagine a number of creative representations of the behavior from using other iconic and geometric forms in the iteration to using sound and music.

## STEP 2: Graph

Our source structure is a textual computer program in figure 2, and so our first goal is to represent this as a graph of some sort. Figures 3 and 4 illustrate control and data flow representations of the program. Other type of graphs, especially "models" used to encode process, are possible; however, data and control flow diagrams are two of the most common generic graphs.

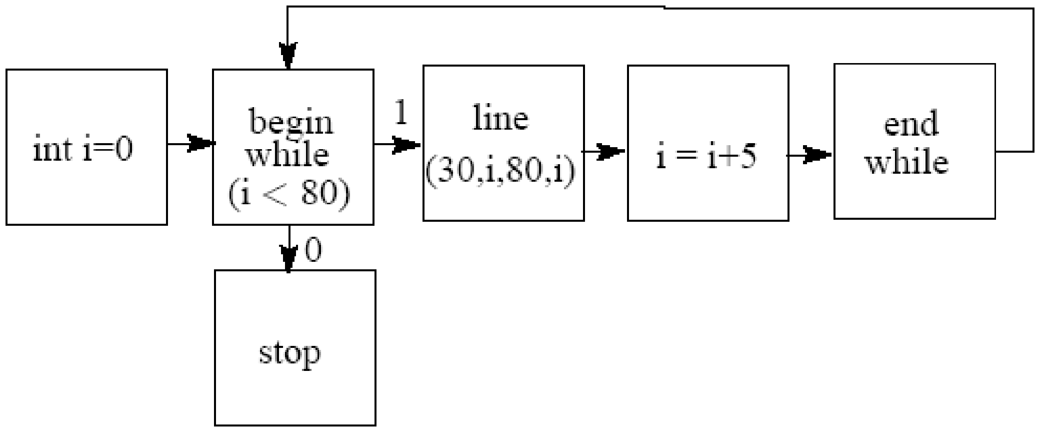


Figure 3: Control flow diagram for the code in figure 2.

Graph nodes in both Figures 3 and 4 are shown as boxes. The control flow diagram is often termed a *flowchart*. The edges labeled 0 and 1 represent false and true, respectively. Figure 4 shows the corresponding data flow diagram, which emphasizes the flow of data rather than of control. A key difference between these models is that the control flow is sequential, while the data flow is parallel: all nodes in the data flow graph execute simultaneously.

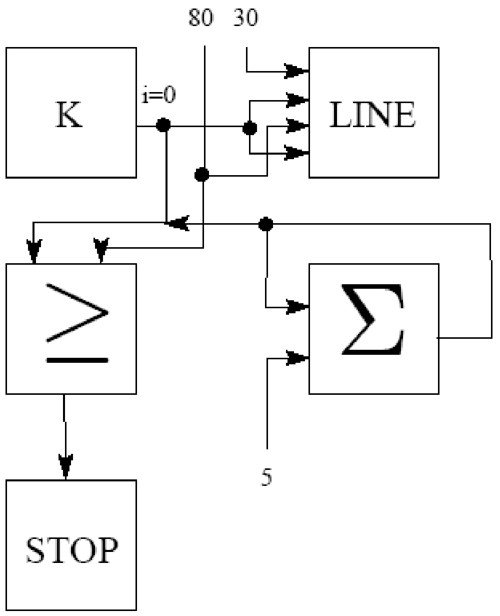


Figure 4: Data flow diagram for the code in figure 2.

Now that we have the core program in Figure 2 in terms of data and control flow graphs, or models, we can proceed with the semantic networks needed to capture the ontological level.

## STEP 3: Ontology

A semantic network is a labeled graph of nodes defining concepts. An ontology is based on a semantic network and usually includes additional arithmetic or logical-based constraints: for example: mammals have four legs or a transport vehicle may have one more wheels. Semantic networks and ontologies are important not only for representing what we know about a source structure, but also, what we know about the chosen target domain identified in the subsequent step. For the computer program, we need to define what it means *to be* a program, and then similarly, what it means to be a product of architecture if architecture captures our target domain. In this example, we are going to mix concepts of steps 3 and 4 together and then use our definition of step 4 to illustrate a higher level approach to mapping that would normally be used before settling on a specific target—such as architecture. Figure 5 displays three semantic networks: program, building, and graph. Squares represent aggregation, composition, or definition while circles represent generalization. For example, a Program is composed of an Initialization, a Body, a Stop, and a Sequence whereas a Building contains a Portico, a Primary Structure, an Exit and Portal. These are concepts that are to be viewed in the same light as classes in Java or C++, and so, they can be used to instantiate multiple objects of that particular *type*. As for the circle relations, there are two types of statements (simple and complex) and two types of areas for buildings (room and block). The numbers that appear as superscripts are defined in step 4. Simple ontological constraints are defined using relational operators such as “=2”.

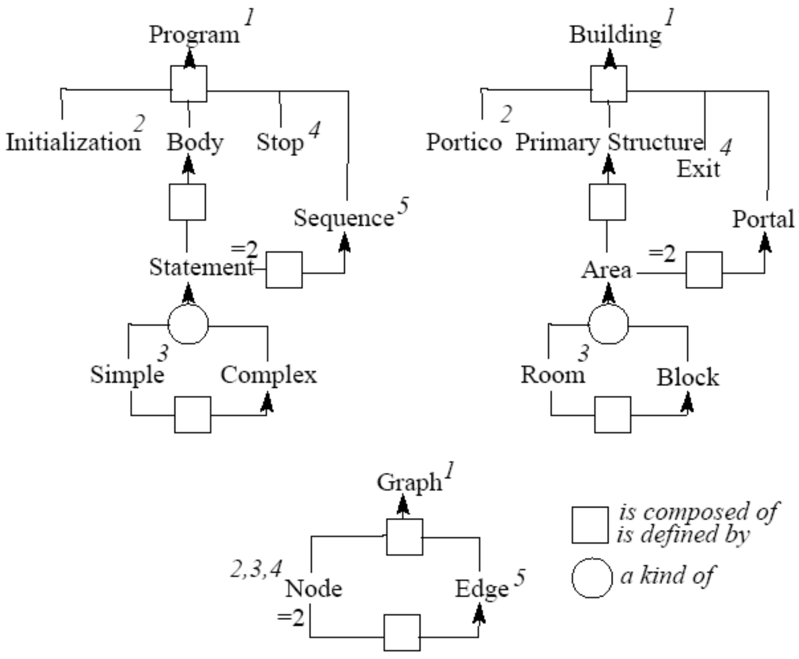


Figure 5: A simple ontology containing knowledge of what it means to be a program (upper left), building (upper right), and graph (bottom).

## STEP 4: Map

Given that we have source structures as graphs and that graphs are composed of multiple relations (i.e., on edges), how can a relation be reified? Let’s take an example of two elements X and Y. We can show a relationship between X and Y by drawing an arrow between them. This is conventional and requires knowledge of what an “arrow” is, and yet this is an effective method for capturing a directional relationship of X to Y.

Figure 6 is a general method for this approach: a spatiotemporal matrix defining a relation. The arrow case is an example of employing the upper left most box. The target is a static visual representation of X being related to Y: connect X and Y through a third “path” object. However, X can also be related to Y by putting Y inside of X (or vice versa). This method of encapsulation relies on an entirely different metaphor (Lakoff et al. 1980). The column “dynamic” suggests that the relationship is shown dynamically with X, for example, moving toward Y (in the connection row), X turning into Y upon magnification (encapsulation row), or X morphing “in place” to Y over some period of time.

The column labeled “aural” is also dynamic, yet non-visual: the connection-type is where there exists a musical or sonic passage that separates X from Y, thus identifying X as being related to

Y. For the encapsulation row of aural, X "contains" Y as a subcomponent musical element. For the proximity row, X is followed by Y.

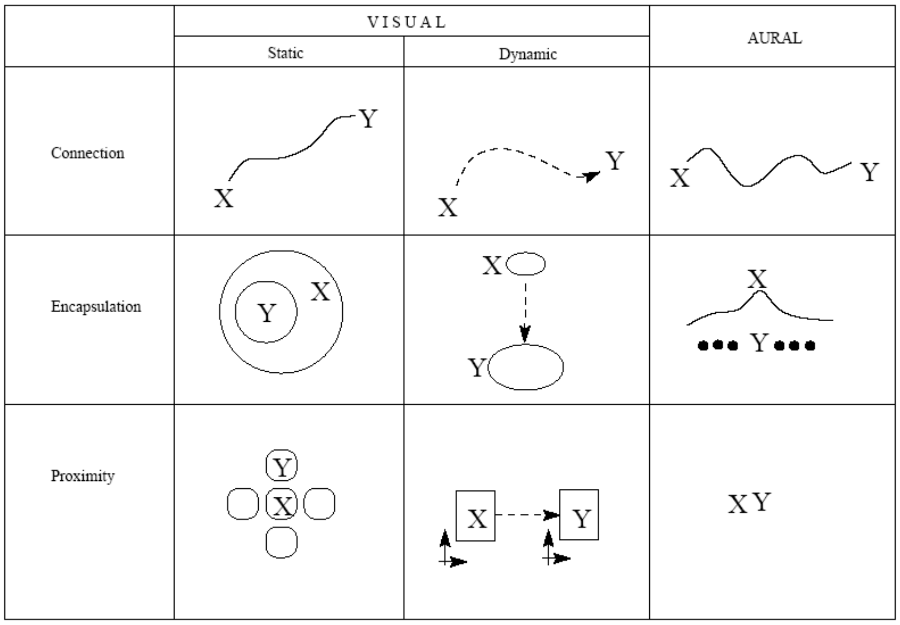


Figure 6: A matrix capturing different ways of reifying the concept of a relation in spacetime

For our program representation, we will use the static/visual column and "connection" row by way of portals (i..e, doorways) from one area into another. We further refine our mapping by revisiting figure 5, and noting the correspondence between the concepts in the top two ontologies as indicated by the superscript identifiers. A Program is a Building (using superscript 1) and a Stop is an Exit (using superscript 4.

## STEP 5: Representation

For architecture, one may use a spatial data structure that captures *adjacency*, which means that the originally embedded, implicit semantics of "next" for the edge between the nodes labeled "int i=0" and "begin while (i<80)" maps to "adjacent to" in the target adjacency graph that must be created to eventually yield an architectural schematic. The original style of "square box" for each node in figure 3 will map to "room" in the target. The graph edges themselves will map to "portals" in the target. The detailed mapping, along these lines, uses semantic networks and simple ontology construction (i.e., figure 5 superscript identifiers) to map from source to target. Then, the representation step yields one of many possible interpretations of the target graph such as the one shown in figure 7. This is a top-down floor layout (i.e., plan view) for a large building with an entrance portico on the left, denoting the place where an imaginary person enters to begin the computation. The small circles represent columns and the diagonal lines with circular arcs represent doorways. For the "while" box, all doorways leading in the linear direction are associated with a condition of true, whereas a false condition indicates an exit through the other doorway to the semi-circular "stop" room.

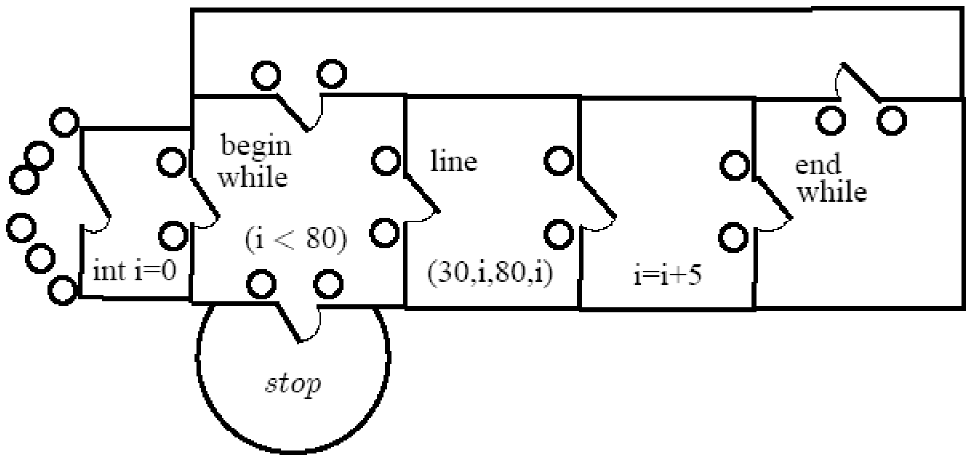


Figure 7: Architectural 2D floor plan target for the control flow model in figure 3.

In two recent aesthetic computing classes, students were not only taught how to represent formal structures along the lines of figure 7, but they also created narrative *targets*. Often, the narrative was made consistent with a 2D or 3D representation. For example, the floor plan in figure 6 can serve as the basis for an accompanying narrative:

*… as Shelly returned through the door at the end of the room containing the metallic switch machine, she noticed that a new silver bar had been positioned next to the previous one. This cast an eerie array of shadows against the far wall. It seemed as if by running through the outer corridor she returned to find the number of bars growing—perhaps to eventually create a jail-like barrier to leaving the room…*

This particular type of narrative is consistent with elements from science fiction and fantasy, for example with “The Hall of Machines” (Jones 1972). The variety of targets is, thus, not limited to visual representations, but can encompass both music/sound (Vickers et al. 2002) as well as narrative dimensions. The use of narrative extends the connections of aesthetic computing to the literary arts. There are numerous other architectural styles that may be used to represent the original program, but there are also many other possibilities such as the abstract model depicted in figure 8.

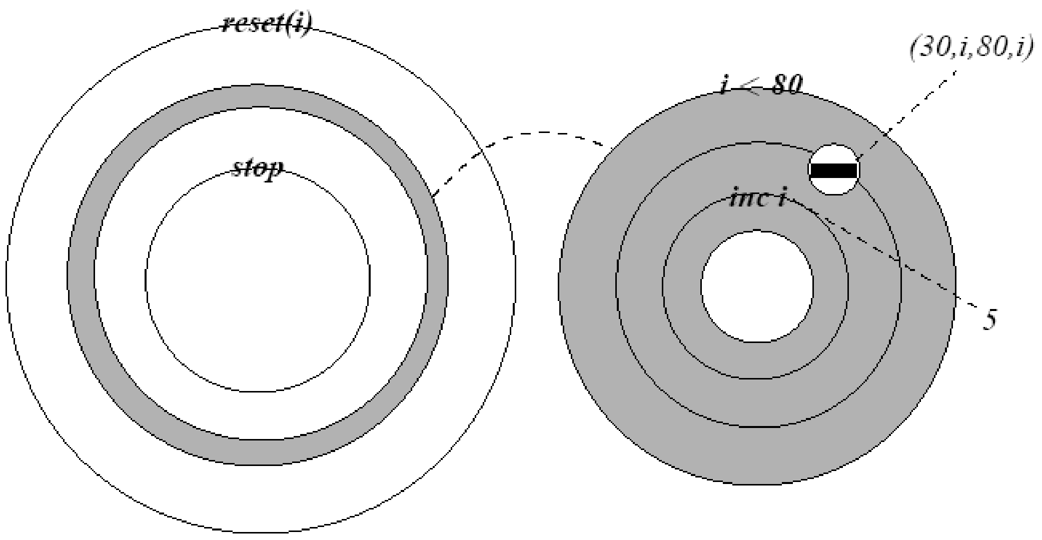


Figure 8: Using a 2D container metaphor to represent the control flow.

The first step in realizing the model in figure 8 is aggregating the control flow diagram into a three-block linear graph: initialization→ iteration→ stop, with this graph being shown as the left-most circle. The original control semantics of “next” is translated to “contains”, so that the original initialization step of resetting variable *i* is a circle that contains a loop designated as shaded ring. The dashed arcs represent the interaction of touching with an input device such as the mouse. So when the ring is touched, the right-most circle appears to show what is inside of the iteration block. The small circle with the horizontal black bar is an icon representing “drawing a line.” As a final example, figure 9 shows a 3D representation of the dataflow in figure 4. A fluid metaphor is used so that constants in figure 4 are captures with valves that set flow rate and the output of each block in figure 4 becomes a copper pipe exiting from a liquid-filled container.

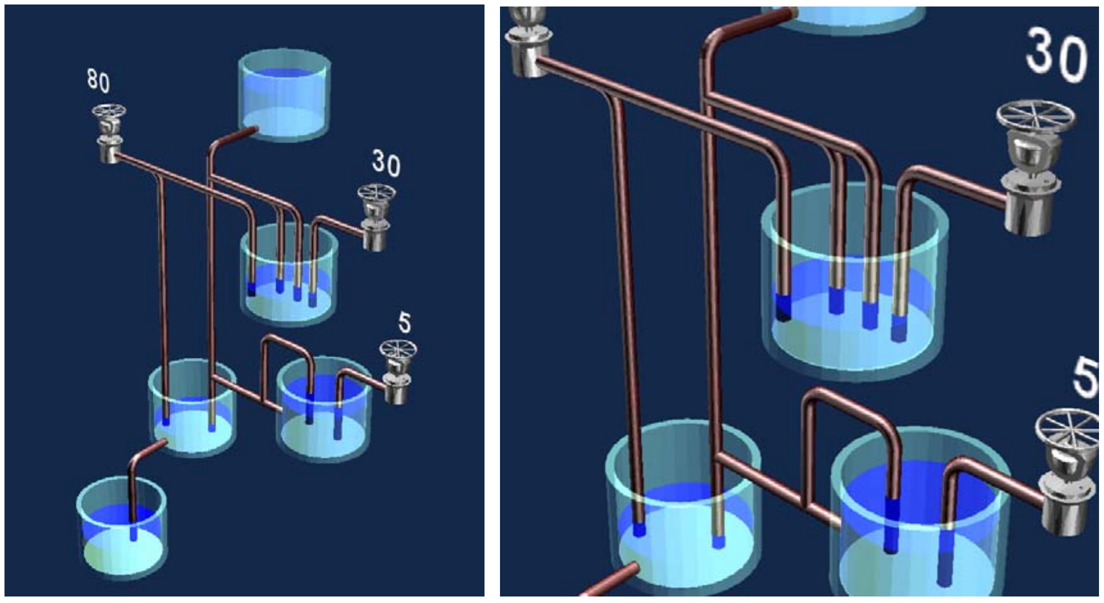


Figure 9: A 3D fluid metaphor representation of the dataflow (fig. 4) with two views

## STEP 6: Assessment

We have used the following criteria in the aesthetic computing class:

- *Syntax*: It is necessary for the visual and aural constructs to be complete: there needs to be an isomorphic mapping between source and target syntactical structures. This can sometimes be made difficult because even with mathematically rigorous source structures, there is often an abundance of external natural language text necessary for one to understand the formalism. So, in some sense, natural language undergirds the most formal of structures. Still, it is common to forget to map certain source elements, and this is problematic from a utilitarian perspective.
- *Metaphor*: The chosen analogy or metaphor needs to be consistently applied from source to target, with similar source components being respectively mapped to similar target structures. For example, mapping two different variables to two types of fruit in a target scenario is counterproductive since the fruit are far more different than the original two variables in both semantic and presentational context.
- *Scalability*: Does the target scale well with a commensurate increase in components in the source structure? The students must analyze the scalability of their products.
- *Semantics*: Models are executed using Java for Processing. Students write their own code, and must convince the assessor that their code accurately relates to the syntactical components.
- *Communication*: The student must fully document the mapping so that the average reader may fully understand the isomorphism. This is the task of effective communication on the part of the creator. Each creator must make the target comprehensible. The task is to avoid mixed metaphors in the target.
- *Utility*: What are the actual and perceived utilities of the target structure? An actual utility estimates the usability of the target model with today's software and hardware technology, whereas perceived utility assumes an advanced technology where the 2D and 3D models are easy to construct, manipulate and interact-with. The actual utility is often fairly low in the case of 3D for reasons of that area still being nascent in terms of ease-of-use given our current state of technology. In the spirit of modeling research, students are encouraged to create designs which may be usable in the future when interaction technology has improved to support their modeling technique. In some cases, a student might take a source component such as a function and map this to a temporal component--a sound or motion, so the user would have to experience and re-experience this component instead of being able to see the model *all at once*. This approach must be carefully weighed against usability requirements.
- *Aesthetics*: All aspects of aesthetics are covered, with a focus on effective and pleasing design. Naturally, there is a great deal of subjectivity, and yet other students can assist in providing a *juried evaluation*.

## Related Work

The use of aesthetics and artistic design as a motivator for the creative representation of discrete structures finds a locus in aesthetic computing (Fishwick 2006). The purpose of aesthetic computing is to explore the application of aesthetics, in all of its diversity, to the field of computing. While the overall effect of aesthetic computing covers many sub-areas, our work interprets aesthetic computing as artistic exploration of representation of discrete structures such as equations, trees, graphs, and the more complex structures built upon these: program and data structures. While many researchers in visual languages employ aesthetics, the definition is one generally limited to quantitatively defined metrics (Purchase 2002, Battista et al. 1999), often defined as optima: *minimizing* line crossings and *maximizing* symmetry. However, when we peruse a comprehensive resource on aesthetics (Kelly 1998), we find that aesthetics is much broader than quantitatively defined optimality criteria. Aesthetics also connotes exploring form and beauty for the sake of the potential in visual and artistic design.

The need to apply this expanded role of aesthetics in visual languages is based in several areas. First, implementations in visual software and algorithms come with the aesthetics of their designers. For example, Lieberman's 3D code representation (Lieberman 1991) and Najork's CUBE (Najoork 1996) present a primitive block environment that has its own visual simplicity and elegance. Diehl (2001) emphasizes *cross-cutting* and interdisciplinary approaches for future software visualization research. To a great extent, diversity of presentation is at the heart of aesthetics, which is frequently catalogued by genre, or indeed by the artist whose design aesthetic is under consideration. Gardner (1984) reinforces the strong relationship between the creative act and art.

A balance between form and function is necessary if all aspects of aesthetics are to be employed in visual language development. The HCI community speaks to this in ways that emotion (i.e., frequently associated with artistic practice) plays a significant role in the interface (Norman 2004, Tractinsky et al, 1997,2000). Moreover, this balance is viewed as fundamental in design with all interfaces vacillating between transparency and reflectivity, or seeing through the interface to complete utility versus reflecting on its visual or aural structure (Bolter et al. 2003). In a recent study by Lavie and Tractinsky (2004), this duality between focus on function versus form, or transparency versus reflectivity, might be interpreted along the two chronologically demarcated approaches to aesthetics: *classical* versus *expressive*. The classical definition of aesthetics was prevalent prior to Baumgarten (1750) and Kant (1790), where aesthetics—and what we would term *usability* in today's parlance—were tightly coupled. Then, in the latter half of the seventeenth century until the present day, aesthetics can be viewed on a more diverse scale as evidenced in (Kelly 1998), allowing for artistic expression, creativity, and pleasure.

The arts have historically defined modern aesthetics as capturing the philosophy of art, and perhaps extending the definition. Within the arts, one finds all aspects of aesthetics. Certainly, expression and creativity are to be found in any museum or magazine devoted to art and design. However, we also find what the artists term *formal* aspects, which correlate well with the previously described optimality conditions in CS. Arnheim (1954,1966) defines a number of abstractions necessary to view art from a formal perspective. Concepts such as balance, dynamics, perspective, and symmetry would fit easily into the lexicon of the visual language or mathematics aesthete. The challenge is for visual languages to explore the remainder of the arts, outside of pure formalism, to where the totality of aesthetics is considered. Aesthetics in visual languages needs to be more than usability and the establishment of optimal metrics--there is a need to bring in the *artistic half* which naturally promotes creativity and personal expression.

## Summary

The goal of aesthetic computing is to apply elements of art and design to the field of computing. We have defined the overall goals of aesthetic computing by creating three levels: cultural, implementation and representation. The cultural level is achieved by social interaction—having artists and computer scientists work together in groups either in close collaboration or through web-based interaction. The implementation level results from computing artifacts such as programs creating artistic behaviors. This level differs from the overall field of digital art in terms of *scale*: each small scale programming element creates an artistic behavior. The representation level includes both structure and behavior so that computing artifacts are represented by artistic artifacts.

The goal of these representations is not to determine an optimal configuration or even to ensure that every mapping is completely isomorphic from source to target, but instead to leverage the creative and customizable aspects of art and design in learning about mathematically-based structures such as computer programs. We have had significant success in applying these goals within the aesthetic computing class itself (Fishwick et al. 2005). The procedure for applying the concept of multiple representations has played a significant and similar role in mathematics education (Kaput 1985). Our work extends that tradition in two ways: leveraging design and art, and applying the concept at the university level in CS.

Aesthetic computing, in terms of the third level of understanding, is all about *representation as a field of study*. There is tension at meeting point between mathematics and computing on one hand and art and design on the other. Artists may feel inordinately constrained by the rules of representation, and computing professionals may be put off by the expanded role of “aesthetic”. However, given that formation of media has in general followed the current state of technology, we are in a position to explore different ways of achieving representations, and to encourage students to design entirely new human-computer interfaces for formal structures.

## Acknowledgments

The author would like to thank all students of past aesthetic computing classes as well as partial support from the National Science Foundation under grant EIA-0119532 entitled "An Investigation into Aesthetic Computing within the Digital Arts and Sciences Curricula" (with co-PIs Tim Davis and Jane Douglas). In particular, many thanks to Salam Daher and Hyunju Shim for their 3D representations, and to Yuna Park for the WARPhaus web page design.

## References

Arnheim, R., Art and Visual Perception: A Psychology of the Creative Eye, University of California Press, 1954.

Arnheim, R. Toward a Psychology of Art: Collected Essays, University of California Press, 1966.

Battista, G. D., Eades, P, Tamassia, R., and Tollis, I. G., 1999, Graph Drawing: Algorithms for the Visualization of Graphs, Prentice Hall.

Baumgarten, A. G., 1750, Aesthetica, Hildesheim, translated by G. Olms 1968.

Bolter, J. D. and Gromala, D., 2003, Windows and Mirrors: Interaction Design, Digital Art, and the Myth of Transparency, MIT Press.

Cox, D., 2006, Metaphoric Mappings: The Art of Visualization In. Fishwick, P., Ed., Aesthetic Computing, MIT Press, pp. 89-114.

Dann, W. P., Cooper, S., and Pausch, R., 2006, Learning to Program in Alice, Pearson Custom Publishing.

Diehl, S., 2001, Ed. Software Visualization, Springer Verlag, May 2001, LLCS 2269.

Fishwick, P., Davis, T. and Douglas, J. , 2005, ``An Empirical Study of Aesthetic Computing'', ACM Transactions on Modeling and Computer Simulation, 18(3), July 2005, pp. 254-279.

Fishwick, P., 2006, Ed., Aesthetic Computing, MIT Press.

Gardner, H. E., 1984, Art, Mind, and Brain: A Cognitive Approach to Creativity, Basic Books.

Greenberg, I., 2006, Foundation Processing, Friends of ED.

Jones, L., 1972, *The Hall of Machines* In The Eye of the Lens, The MacMillan Company.

Kant, I., 1790, The Critique of Judgement, Oxford: Clarendon Press, translated by James Creed Meredith (1952).

Kaput, J. 1985, Representation and Problem Solving: Some Methodological Issues, In Silver, E. Teaching and Learning Problem Solving: Multiple Research Perspectives,
Hillsdale, pp. 381-398.

Kelly, M., Ed., 1998, Encyclopedia of Aesthetics (4 Volumes), Oxford University Press.

Lakoff, G. and Johnson, M., 2003, 2nd Ed., Metaphors We Live By, University of Chicago Press.

Lavie, T. and Tractinsky, N.,  2004, Assessing dimensions of perceived visual aesthetics of web sites, International Journal of Human-Computer Studies, 60:269-298.

Lieberman, H., 1991, A Three-Dimensional Representation of Program Execution, in Visual Programming Environments: Applications and Issues, Glinert, E. P., Ed., IEEE Press.

Najoork, M.,1996, Programming in Three Dimensions, Journal of Visual Languages and Computing, 7 (2), pp. 219-242.

Norman, D. A., 2004, Emotional Design: Why we Love (or Hate) Everyday Things, Basic Books.

Processing Reference Guide, 2006, http://www.processing.org/reference/index.html

Prophet, J. and d'Inverno, M., 2006, Transdisciplinary Collaboration in "Cell", In. Fishwick, P., Ed., Aesthetic Computing, MIT Press, pp. 185-196.

Purchase, H. C., 2002, Metrics for Graph Drawing Aesthetics, Journal of Visual Languages and Computing, 13 (5), pp. 501-516.

Tractinsky, N., 1997, Aesthetics and Apparent Usability: Empirically Assessing Cultural and Methodological Issues in 1997 Proceedings of the Conference on Human Factors in Computing Systems, Association of Computing Machinery (ACM), pp. 115-122.

Tractinsky, N., Shoval-Katz, A., and Ikar, D., 2000, What is Beautiful is Usable: Interacting with Computers, 13, pp. 127-145.

Vickers, P. and Alty, J. 2002, Using Music to Communicate Computing Information, Interacting with Computers, 14(5), pp. 435-456.

Ware, C., 2004, Information Visualization: Perception for Design, Morgan Kaufman, 2nd Edition.

Warphaus, 2006. WARPhaus exhibition, http://www.cise.ufl.edu/class/cap4403sp06/Gallery/WARPhaus_exhibition/index.html