Visualizing Software Classes with Geon Diagrams in Second Life - A Botched Attempt

Paul Oppenheim
paul@pauloppenheim.com

INTRODUCTION

There are two related problems:

Language for Algorithm Visualization

There's no easy way for a coder to visually describe software *in* software. The closest thing is graphviz[1], which isn't an inherently visual or intuitive language[2]. The way to solve this would be to find the fundamental visual elements of software diagrams, then write an intuitive language to generate them. Ideally untrained readers would be able to easily read either the source code or the resulting diagram, perhaps with a language similar to "ASCII art".

Validating Software Visualizations

Very few software visualization techniques have been validated[3]. Because of this, there is no consensus about what the fundamental visual elements of software diagrams are. One technique that has been validated for visualizing the semantic content of UML diagrams is using "geon diagrams"[4] based on Biederman's principles of structural cognition[5].

In the interest of developing a base from which to eventually create a software visualization language, there should be a way to create known-effective diagrams from existing code. These diagrams could then be modified and used in experiments to gauge their efficacy.

Second Life[6] is a 3D virtual world, primarily made from "primitives"[7] which almost exactly correspond to the geons in geon diagrams, and as such would be a good environment to reproduce them. Further, Second Life is attractive as an environment because of the ability to easily share the diagrams and talk about them, my familiarity with it[8], and because it would be rather cool.

RELATED WORK

Software Visualization is an entire field unto itself, albeit a smaller one. There's a catalog of many software visualizations, papers, and books related to the subject at algoviz.org[9]. Of only two books on the subject, one asks the question at the core of software visualization:

> ...the fact is that at present there is a shortage of software visualizations explicitly built around cognitive representations. And outside the domain of code, there is not even much idea what a cognitive representation would look like.[10]

There is a body of work around visualizing graphs and networks for mathematics[11], part of which is implemented in the graphviz distribution. A group of researchers in the Object Management Group (OMG)[12] created Unified Modeling Language (UML)[13], a set of techniques for making several types of node-link graphs with additional annotations specifically for software visualization. The question remains whether extensions such as UML are effective, and if there are other techniques which could boost the efficacy of node-link diagrams.

The Diehl book suggested that there *is* an improved model based on approaching the problem from the angle of visual cognition, the geon diagram[14], written about by Irani and Ware in several papers[15]. Controlled experiments show the geon diagram to be more efficient than a semantically equivalent UML diagram[16]. Missing with the papers on geon diagrams is an implementation for others to use, to iterate on, and to run other experiments with. I imagine that this would be very important for others to efficiently carry this line of research forward.

METHODS

This is where there is obvious weakness in my work - I spent so much time researching, deciding which path would be effective, and learning and manipulating existing software that I didn't get far with the implementation. The *intended* techniques and algorithms are as follows.

The original plan for geon diagram generation in Second Life involves

- parsing description of UML diagram
- generating geon layout (in this case, simple force-directed with straight connecting links)
- generating LSL (Second Life scripting code) capable of rezzing the diagram

Prof. Agrawala gave comments on my in-class presentation suggesting that there need to be more “apples-to-apples” comparisons of geon diagrams and UML. As such, a more one-attribute-at-a-time modification from UML to geon is required in the visualization. As the geon diagram compares to the UML class diagram, there must be a way to easily generate

- Object-oriented source code
- equivalent UML class diagram
- equivalent geon diagram
- alternate diagrams for controlled experimentation

Because of the last requirement the system needs to be able to generate UML diagrams and UML-like diagrams as well as geon diagrams and geon-like diagrams.

Regardless of what I should have done, there are several components required:

code there needs to be a set of object-oriented code samples which describe several multi-class algorithms, ideally of varying complexity to create various levels of class diagram complexity. As my work outside the scope of this project is primarily in C++, this would be the ideal language to work with.

parsed code the code needs to be parsed into an AST from which relationships can be discovered. C++ is notoriously hard to parse, as can be seen by the Elkhound[17] and FOG[18] parsers.

relationships relationships between classes need to be used to infer the UML semantics.

UML semantics The information composing the UML class diagram, needed to layout the diagram. Class names, inheritance structure, dependencies, and other attributes are extracted from the code relationships.

UML intermediate form a text description of the semantics of the diagram. The OMG has created an XML-based format for UML known as XMI which is suitable for this purpose[19]. This is useful as an interchange between several tools which may not be able generate UML semantics, but may be able to render them.

UML intermediate form parser a tool to take the UML intermediate form and turn it back into UML semantics.

Layout software software which can turn UML semantics into appropriately sized and positioned nodes and edges.

Drawing software software which can take a layout and render it using a set of given parameters.

In the original plan, many parts of this were handled by existing open source software:

- code - written sample, several open source projects
- parsed code, relationships - bouml [20] (C++ plugout)
- UML semantics, UML intermediate form - bouml
- UML intermediate form parser - python xmiparser[21]
- Drawing software - Second Life

This arrangement leaves the Layout software to be written. It took a while to get to this stack, but when I finally did, I had difficulty in making xmiparser parse bouml’s output. Initial LSL exploration indicated that with decent choice of node objects, there shouldn’t be much difficulty in rezzing them at appropriate locations.

Several other stacks were also attempted:

- parsed code, relationships - doxygen[22]
- Drawing software - Second Life

Leaving UML semantics and Layout software to be written. UML intermediate form serialization and parsing could be skipped in this case, as the software could be written as part of doxygen. This path was abandoned because of the difficulty of working with the doxygen source, and the necessity of inferring UML semantics without assistance. UML is a significant standard, and even inferring class diagram semantics is not a simple undertaking, especially in a codebase as complex as doxygen.

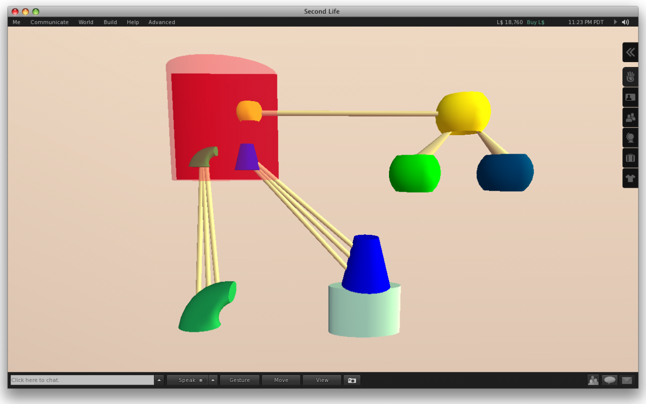
A stack giving optimal flexibility but highest implementation complexity would involve:

- parsed code - elkhound / elsa
- Drawing software - Second Life

This would further involve inferring relationships, but would require relatively simple code to operate on a parsed AST. This could simplify implementation at higher levels.

RESULTS

Unfortunately, the only geon diagrams made were hand-built replicas of the diagrams in the research paper in Second Life:



I learned a tremendous amount about software visualization in the process of researching the original topic. That will aid my future professional work. I consider that a reasonable result as any.

DISCUSSION

Ideally from my research it can at least be seen

- Research shows software visualizations can be effective, some more than others
- There need to be more tools for practical research on software visualization
- Writing these tools may be larger than the scope of a one-month final project in CS294-10 taught by Prof. Agrawala, especially if you already have a full-time job in the software industry

FUTURE WORK

This system still needs to be written. I feel that once the more mundane technical and integration issues are solved, there is a potential to make diagrams sufficient for quantitative testing. The variations in diagrams could be shown using online surveys and large audiences to gauge the cognitive appropriateness of the myriad combinations of graph properties available. If designed correctly, this research could provide results similar to other quantitative perception research, such as the canonical views research by Blanz et al.[23], or the power functions of perceptual magnitude by Stevens[24].

REFERENCES

1. graphviz - http://www.graphviz.org/About.php
2. graphviz example to demonstrate non-intuitiveness - http://www.graphviz.org/Gallery/directed/datastruct.gv.txt
3. Stephan Diehl. ”Software Visualization”. Springer, 2007. p. 157.
4. See [3], p. 60.
5. I. Biederman, ”Recognition-by-Components: A Theory of Human Image Understanding,” Psychological Review, Vol. 94, No. 2, 1987, pp. 115-147.
6. Second Life http://secondlife.com/
7. Second Life Primitives http://wiki.secondlife.com/wiki/Primitive
8. Poppy Linden http://wiki.secondlife.com/wiki/User:Poppy_Linden
9. Algoviz software visualization catalog http://wiki.algoviz.org/AlgovizWiki/Catalog
10. John T. Stasko, John B. Domingue, Marc H. Brown and Blaine A. Price. ”Software Visualization.” MIT Press, 1998.
11. EMDEN R. GANSNER and STEPHEN C. NORTH. ”An open graph visualization system and its applications to software engineering.” SOFTWAREPRACTICE AND EXPERIENCE. John Wiley & Sons, Ltd. 1999.
12. OMG http://www.omg.org/
13. UML http://www.uml.org/
14. see [4]
15. Pourang Irani and Colin Ware. ”Diagrams Based on Structural Object Perception.” Proc. Advanced Visual Interfaces AVI2000, Palermo, Italy, May 2000, pp. 61-67
16. Pourang Irani, Colin Ware, and Maureen Tingley. ”Using Perceptual Syntax to Enhance Semantic Content in Diagrams.” IEEE Computer Graphics & Applications, 21(5):76-84, 2001.
17. Elkhound: A GLR Parser Generator and Elsa: An Elkhound-based C++ Parser - http://scottmcpeak.com/elkhound/
18. Edward D. Willink. Meta-Compilation for C++. PhD Thesis, Computer Science Research Group, University of Surrey, June 2001. http://www.computing.surrey.ac.uk/Research/CSRG/fog/FogThesis.p
19. MOF 2.0/XMI Mapping. Object Management Group, 2007. http://www.omg.org/spec/XMI/2.1.1/
20. bouml http://bouml.free.fr/
21. xmiparser http://pypi.python.org/pypi/xmiparser/1.4 part of ArchGenXML http://plone.org/products/archgenxml
22. doxygen http://www.stack.nl/∼dimitri/doxygen/
23. Volker Blanz, Michael J. Tarr, and Heinrich H. Blthoff. ”What object attributes determine canonical views?” Perception, 28 575-600, 1998.
24. Stevens, S. S. The psychophysics of sensory function. American Scientist. Vol 48, 1960, 226-253.
25. http://media.tumblr.com/tumblr_l20epdB8Nz1qb25dg.jpg