Position paper:
MetaViz – Issues in Software Visualizing Beyond 3D

Juergen Rilling, Jianqun Wang, S. P. Mudur
Department of Computer Science, Concordia University
{rilling, jianq_wa, mudur}@cs.concordia.ca

Abstract

In this research, we present our MetaViz project that was initiated to investigate software visualization issues in the context of novel modeling and visualization techniques. The focus of our research is the visualization of software structures and their dynamic behavior using 3D visualization techniques. Rather than applying traditional 2D visualization techniques and transfer these to the 3D space, we are exploring novel visualization techniques and investigate computational, grouping and layout related issues of these visualization techniques.

1. Introduction

For large, complex software systems, the comprehension of such diagrammatic depictions is restricted by the resolution limits of the visual medium (2D computer screen) and the limits of user's cognitive and perceptual capacities. One approach to overcome or reduce the limitations of the visual medium is to make use of a third dimension by mapping source code structures and program executions to a 3D space [6]. Mapping these program artifacts into the 3D space allows users to identify common shapes or common configurations that may become apparent, and which could then be related directly to design features in the code. Scalability becomes a well-known barrier that exists in both 2D and 3D software visualization [9]. Improving layout algorithm and clustering management is one way to make visualization techniques scalable. In this paper, we will investigate some of the readability issues of visuals representations in 3D space, and present improvements in three different aspects. First, we will introduce the use of metaballs as a metaphor to visualize software systems to improve the more traditional representation of "Nodes and Arcs". Second, we will use hierarchic grouping of entities to abstract higher level entities and improve the usability. This will lead to an "overview first, zoom and filter, then details on demand" approach. Finally, we will address issues related to gird based 3D layout algorithms as some of the techniques to improve readability of the 3D visuals created [12].

The remainder of the paper is organized as follows. Section 2 introduces the MetaViz project; section 3 discusses the metaball visualization approach and the Marching Cube algorithm used to create the metaballs. Section 4 introduces 3D layout algorithms and quality aspects. In section 5 we discuss grouping and clustering to improve the comprehensibility of visual representations. Section 6 illustrates applications of MetaViz tool, followed by a discussion of future challenges in section 7.

2. Overview of MetaViz

The MetaViz tool was developed using Java 3D as an independent, but reusable 3D visualization environment to investigate the various application domains for the metaball metaphor. The tool provides a programming interface that allows for an easy extension and further reuse of the tool. The MetaViz tool consists of three major parts: the grid-layout, a clustering and grouping algorithm, and the metaball rendering engine (see Figure 1).

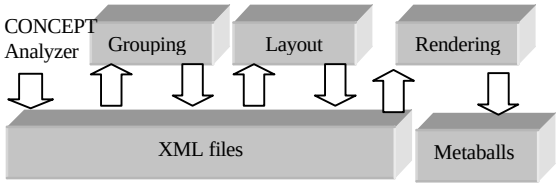

Figure 1. MetaViz architecture

The grid layout plug-in is based on an XML input file, which describes the software artifacts and their internal relationships. Within the MetaViz tool, users can select among 4 different layout algorithms, and provides users with feedback about the progress made in the layout optimization by displaying snapshots of the current layout state. Once a predefined optimum is reached, the layout optimization is completed and will be stored in an XML file for further processing.

The rendering engine of the MetaViz tool reads this XML file as input to generate and render the metaballs in

3D space, by mapping the structural software properties to the properties of the metaballs. After the completion of the rendering process, the user can navigate through the visuals and apply overview, select and zoom techniques to refine the current view.

3. Metaball Metaphor

Creating intuitive and useful abstraction is one of the major research issues in 3D software visualization [5]. As part of the MetaViz project we are investigating new metaphors for 3D software visualization that can provide additional guidance in supporting the comprehension process [6].

3.1. Metaball vs. "Nodes and Arcs"

Metaballs provide a three dimensional picture with smooth connection between metaballs and shading, which eases the difficulties of building mental model. Figure 2 illustrates the advantage of metaball over a 3D sphere-line graph. Both visuals display the same information and use the same layout. One of problems of the sphere-line graph is that it cannot convey some structural information in the same way as, for example, the metaballs. The fusion (the thickness of the connection) among two or several metaballs can be used to show clearly structural dependencies. The fusion can also be used to indicate the relationship among different software artifacts. Shading and blending are other options that can be applied to convey additional information not available in most traditional software visualization techniques.

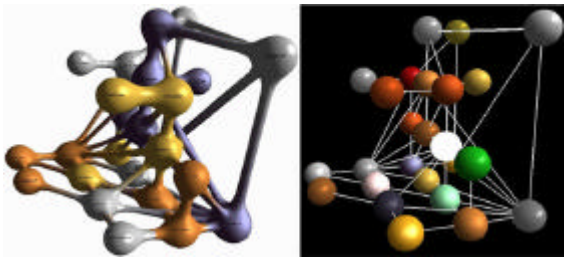

Figure 2. Metaball vs. sphere-line graph

However, it should be mentioned that the metaball has one major disadvantage - its computational complexity, caused by the Marching-cube algorithm that is used for rendering the metaballs. For visualization techniques to be useful and applicable, performance plays an important role. In what follows we discuss an optimization for the general Marching-cube algorithm that improves on the rendering speed of the algorithm.

3.2. Marching-cube Algorithm Optimization

The performance problem with the metaball rendering is directly related to the computations carried out by the Marching cube algorithm that computes the iso-surface of the metaballs [15]. One way to improve the performance of the algorithm is to avoid any surface computations for grid cells that do not contain any parts of the metaball iso-surface. The following table (Table 1) shows our test results for 64 metaballs with and without this optimization. For the experiment, we limited the recursion level to 9. It should be noted that the recursion level directly corresponds to the rendering quality of the metaballs. For a recursion level of 5 or lower, the resulting metaball surface becomes too coarse (polygonized) to be useful. For more than 9 recursions, the computation complexity becomes too large to be applicable.

Recursion times	Non-optimized	Optimized
5	1,093 ms	157 ms
7	55,469 ms	1,750ms
8	459,609 ms	9,031 ms
9	3,286,978 ms	58,343 ms

Table 1. Test results for 64 metaballs rendered by the marching cube algorithms optimization. (CPU: P4 2.8GHz, 1GB RAM)

4. 3D Grid Layout Algorithms

Within the MetaViz tool, we applied a grid layout approach, where the position of each node corresponds to integer coordinates. There are two major reasons for applying this approach for 3D visualizations. Firstly, the layout algorithm allows for a space efficient visualization of metaballs. For example, 1000 entities can be placed within a 10*10*10 grid using the metaball visualization. On the other hand, displaying the same 1000 entities using a cone tree [2], the size of each entity becomes too small at the lower levels of the tree.

Secondly the grid layout is reusability and can be applied for other visualization techniques that have similar layout and readability problems. The presented layout algorithm is developed as a separate Java package plug-in that can be reused by other visualization approaches within our CONCEPT project (e.g. UML diagrams, 3D worlds, etc.).

It should be noted that grid layout algorithms have two major shortcomings. One is their lack of real-time responsiveness, which makes them not suitable for interactive/animated applications. The other shortcoming is that the rearrangement of graphs may not preserve some designing properties, and the context (e.g. grouping, clustering) of the original design might be lost. [8].

4.1. Readability criteria

Software visualization techniques were originally introduced to support people during software and system comprehension. For a visualization technique to be useful, its readability becomes a major quality factor. Readability criteria have already been established and applied in information visualization to evaluate the quality of layout algorithms [10]. Figure 3 shows the importance ranking of different criteria for the readability of a visualization technique in general. In our implementation, we take into account most of these criteria. The objective function is a weighted sum of these numbers, with line object crossing having the highest weight, and the length of arcs having the lowest weight. Drawing space and density distribution can be further optimized by choosing a minimum grid size that can accommodate the given number of metaballs.

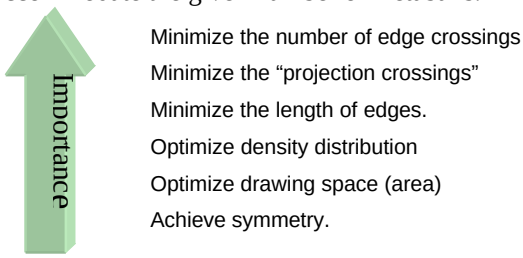

Figure 3. Visualization criteria

4.2. 3D Grid Layout as a State Searching Problem

The 3D grid layout problem can be described as placing m entities into n^3 positions to satisfy certain readability constrains, where $m <= n^3$.

The computationally intensive nature of 3D layouts has already been shown in existing layout algorithms in other application domains [3]. The complexity of evaluating a 3D layout is O (n3!). A brute force search is not feasible, because of the size of the search space [13]. One solution to the search problem is to apply a heuristic search. In what follows we apply the hill-climbing algorithm to improve the grid layout algorithm.

4.3. The Hill-climbing Search as a 3D Layout Problem

Initially, a search tree has to be created that starts from a root state and moves on to its children. In the case of a root state, the algorithm randomly places m entities within the given n^3 space. The algorithm swaps some of the entity positions and creates new states for the children. The number of children corresponds to the "branch factor", which directly influences the complexity of searching algorithm. We calculate the branch factor for certain operators (Table 2).

Operator	n entities into n positions	Branch factor
Switching two entities	(n-1)*n/2	351
Switching three entities	(n-1)*…*(n-4)	15,600
Switching four entities	(n-1)*…*(n-5)	3.6e+5
Switching five entities	(n-1)*…*(n-6)	7.9e+6

Table 2. The branch factors of searching trees for placing 27 entities into 27 positions

The hill-climbing algorithm is a state searching algorithm, with the goal to minimize the memory requirements to perform the search. A detailed analysis of existing searching algorithms that are studied extensively in the field of Artificial Intelligence can be found in [13]. The limitation of the hill climbing algorithm is that it can only find a local peak. In our research, we try to overcome this limitation by modifying the hill-climbing algorithm. In our extended version called as the competition hill-climber, we modify the hill-climbing algorithm by using a more expensive comparison to break away from the local peak and jump to another hill which has a higher peak than the current hill.

Thread	Projection crossings	Line object cross	Lines length	Objective Function value	Compute time (sec)
0	22	0	125	279	1284
1	10	1	130	221	1134
2	24	1	149	338	953
3	33	0	126	357	1161
4	20	0	126	266	1230
5	17	1	123	263	1353
6	18	0	124	250	1348
7	12	0	114	198	1325
8	16	0	129	241	968
9	11	1	115	213	1329

Table 3. Results of competition hill-climber (Test condition: P4 2.8GHz, 1 GB RAM)

Moreover, we apply a random positioning of the entities into the grid and use these as random starting states. These random starting states will then lead to different peaks. In our implementation, we use ten threads to evaluate ten different starting states using one of the above strategies. Finally, we compare the ten computed peaks and choose the peak with the best result. In our example (see Table 3), thread number 7 has the best estimation value.

5. Grouping

"Program analysis is a crucial part of many program understanding tools" [1]. Grouping can be described as a process of program analysis prior to the layout management. The layout algorithms are constrained by the amount of information to be displayed and the limited

screen space. Even, if one manages to create a layout, the resulting visual might have far too much information, causing an information overload. Therefore, limiting the number of entities to be displayed to the user is one of the key challenges in software visualization. For the visualization of large software systems, it is essential to provide some type of grouping to create a decomposition of the system. It has been shown that grouping can improve readability [11], by supporting a representation that is closely related to the mental model a programmer forms of a system [14] during typical comprehension tasks. Grouping or clustering can be applied to generate suitable abstraction levels and therefore allow for a reduction of the amount of information to be displayed on the screen. In this paper, we present two methods of grouping: metric-based grouping and feature-based grouping.

Metric-based grouping.

For metric-based grouping an internal relation table is created that analyzed the coupling among different classes. The number of function calls defines the weight of relationships among the different entities (coupling). In a first step, we identify entities that are strongly coupled with each other and group them closely together. During the second processing step, we identify a threshold that corresponds to the maximum number of entities displayed on the screen. Based on the coupling relationships and the maximum threshold, objects are placed on the screen. For example, the threshold for Figure 4 is four. After the first grouping iteration, ten groups are identified. Each of these groups can be treated as a separate entity, for which the process can be re-applied recursively to create the next higher level of abstraction.

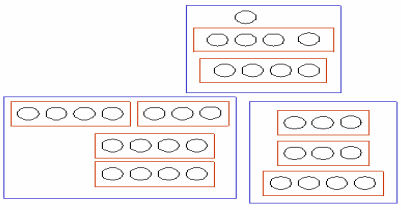

Figure 4. Metric-based grouping

Feature-based grouping

There exist several techniques for identifying features in software systems, e.g. (program slicing [12], concept analysis [7], etc.). For some applications, such as testing and debugging, programmers might be interested in focusing on particular features instead of the whole software system. Therefore, grouping software entities based on their features can help programmers to focus on related software parts and therefore reduces the cognitive and comprehension load.

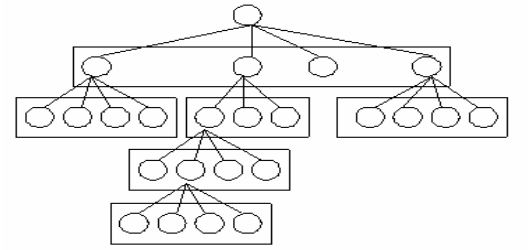

Figure 5. Feature-based grouping

Figure 5 shows an example of such a feature-based grouping. A program might be represented, like in this example as a hierarchical structure, with a feature consisting of several sub features.

6. Applying MetaViz

In this section, we illustrate some applications of MetaViz and its visualization techniques, layout algorithms and grouping techniques. For illustration purposes, we use the MetaViz implementation itself. MetaViz consists of 64 classes with a total of approximate 10,000 LOC. The examples will illustrate how metaballs in combination with different source code analysis techniques guide programmers during program comprehension. The examples include: a hierarchical representation, grouping based on coupling among different software entities, the animation of the layout algorithms and visualization of dynamic program aspects using metaballs.

6.1. Hierarchical Structure of Software

Typically, any larger software system is organized as a hierarchical structure and many software visualization techniques are developed for visualizing these hierarchies (e.g., tree structures such as cone-tree, cam-tree, and information-cube)[4]. Within MetaViz, we visualize these hierarchies by applying the metaball metaphor.

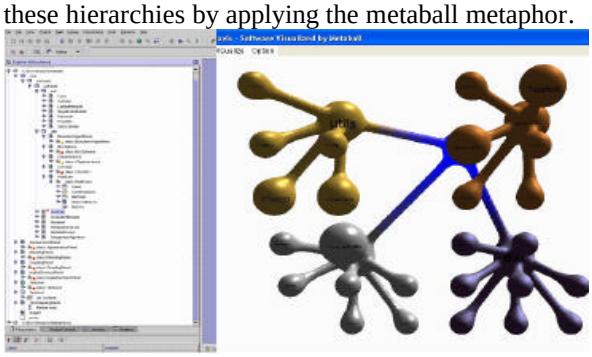

Figure 6. Hierarchical structure in SunONE (left); same structure using MetaViz's visualization

Figure 6 shows the MetaViz program structure in both a textual representation as it can be found in most IDEs (left) and a graphical representation using the metaball

approach (right). We additionally apply color coding to indicate package dependencies and hierarchy information.

6.2. Applying the Metaball Metaphor to Visualize Coupling Measurements

Visualizing the internal relationships and call dependencies of software systems is an essential part of many software visualization tools. In this application example, we apply coupling between object classes (CBO) as the relationship measure among software artifacts. The cylinders diameter connecting two metaballs corresponds directly to the existing CBO coupling among two entities. Furthermore, the cylinder provides a visual feedback of the strength of the coupling.

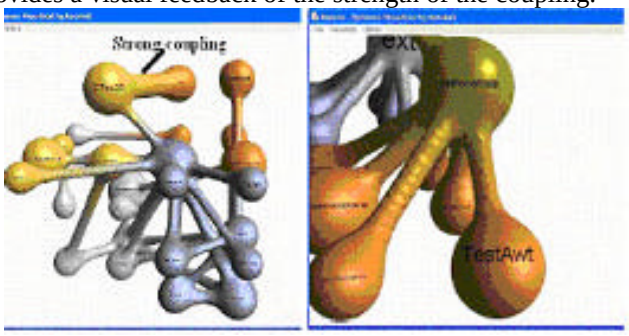

Figure 7. Metaball visuals of MPC measurements

6.3. Layout Management Animation

The MetaViz tool not only implements an optimized hill-climbing layout algorithm, but also has the option to visualize the layout computation and optimization process. The following two sample snapshots (Figure 8) are from one of these layout optimization sessions, providing users with instant visual feedback on the progress of the current layout optimization. It also allows the user to terminate the optimization process once layout meets the visual expectations (quality).

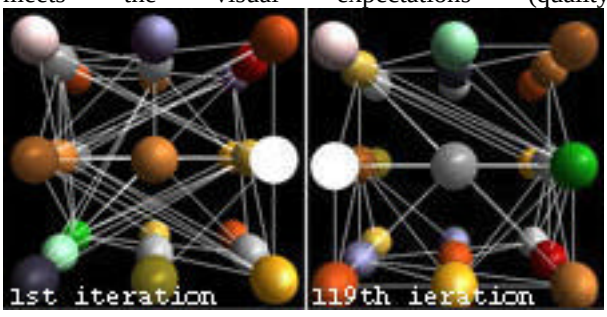

Figure 8. The Layout optimization Process

6.4. Visualizing Dynamic Information

Currently most visualization tools are restricted in their ability to support dynamic program information and aspects. Within the MetaViz tool, an execution trace is recorded for a specific program execution; allowing for a stepwise re-execution that can be displayed as a sequence of frames. Figure 9 shows four frames of such an animated re-execution. The metaballs in the picture represent executed classes, with the diameter of the metaball corresponding to the number of executed statements and the white ball indicating the current execution position.

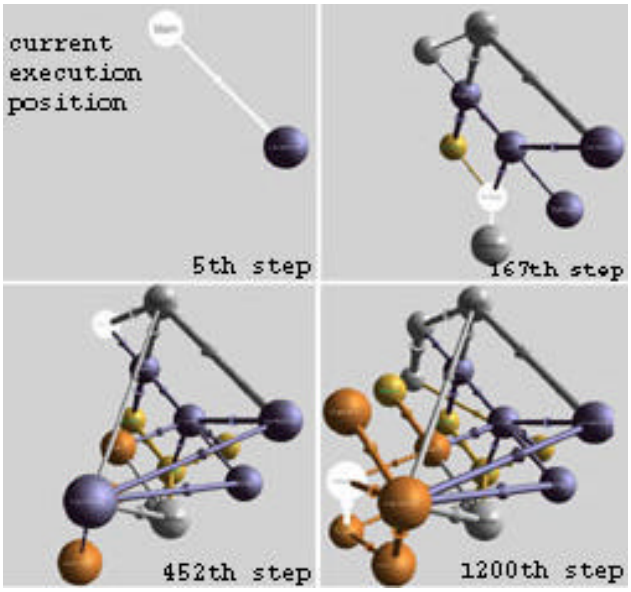

Figure 9. Dynamic visualization

7. Software visualization beyond the third dimension

In this article we presented our MetaViz tool, in which we address the following visualization issues: We introduced the metaball metaphor as a basic notation for our software visualizations, a grid based 3D layout and grouping techniques. One of the initial motivations for the MetaViz project was to explore novel software visualization techniques in 3D space. One of the immediate results of our prototype implementation was that 3D visualization techniques can enhance and benefit the comprehension process by enhanced utilization of screen space and additional visualization effects (shading, transparency, fusion, etc.). Source code analysis can provide additional insights and guidance in filtering and visualizing the information. Our grid layout technique improves the readability of 3D visuals on screen. Specifically, the ability to dynamically observe the behavior of the layout algorithm as it progresses helps the user to be an active participant in this visual generation process. Grouping based on object coupling and slicing based features were demonstrated, to avoid

information overloading. However, several main challenges remain that go beyond just moving to 3D space or applying some layout or clustering techniques.

Remaining key challenges are the re-creation of a mental model that closely corresponds to the mental model designers/programmers developed during the originally forward engineering process. No matter what layout, clustering or grouping algorithm one applies to identification and analysis of logical relationships among different software entities, they always will be limited by the quality of the algorithm and the lack of domain knowledge modeling. Overcoming these limitations requires additional information sources (other than source code) and domain knowledge has to be incorporated in existing layout, grouping and clustering algorithms.

With more and more applications moving into distributed and network centered environments; software visualization, analysis techniques, as well as grouping and layout approaches have to keep up with these changing requirements. Visualizing dynamic and behavioral aspects has additional challenges in the form of filtering large amount of information, the creation of meaningful abstractions and

8. References

[1] A. van Deursen and J. Visser. Building Program Understanding Tools Using Visitor Combinators. In Proceedings 10th International Workshop on Program Comprehension (IWPC'02), pages 137-146, IEEE Computer Society, 2002.

[2] Cockburn, A. & McKenzie, B., “An evaluation of cone trees,” In People and Computers XV. Proceedings of the 2000 British Computer Society Conference on Human-Computer Interaction, University of Sunderland, 4--8 September, 2000.

[3] Fabien Jourdan, Guy Melançon A scalable force-directed method for the visualization of large graphs Workshop on info visualization. LIRMM, Monptellier. January 2002

[4] Ivan Herman, Guy Melancon, and M. Scoot Marshall, “Graph Visualization and Navigation in Information Visualization: a Survey”, IEEE Transactions on Visualization and Computer Graphics, Vol6 No 3, 2000 Computer Science, 1995.

[5] Knight, C. and Munro, M. “Software Visualization conundrum”, Department of Computer Science Technical Report 05/01, July 2001.

[6] Knight, C., and Munro, M. “The Power of (Software) Visualization”, Department of Computer Science Technical Report 01/00, January 2000.

[7] Koschke, R., `An Semi-Automatic Method for Component Recovery', Proceedings of the Sixth Working Conference on Reverse Engineering, pp.256-267, Atlanta, October 1999.

[8] M. A. Storey, and H. A. Muller. “Graph layout adjustment strategies”. In Graph Drawing '95, pages 487--499, 1995

[9] Maletic, J.I., Marcus, A., Collard, M. "A Task Oriented View of Software Visualization", in Proceedings of the the IEEE Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2002), Paris, France, June 26, 2002, pp. 32-40

[10] R. Tamassia. New layout techniques for entity-relationship diagrams. In Proc. 4th Int. Conf. on Entity-Relationship Approach, pages 304--311, 1985

[11] S. Mancorids. B. S. Mitchell, Y. Chen, E. R. Gansner “Bunch: A clustering tool for the recovery and maintenance of software structures” In Proc; IEEE Inter. Conference on Software Maintenance, IEEE Computer Society Press, 1999, pp 50-59.

[12] J. Rilling, and S. P..”The Metaball Metaphor for Slicing Based Software Visualization”, 2003.

[13] S. Russell and P. Norvig, “Artificial Intelligence A modern approach”, Prentice Hall, 1995.

[14] V. Tzerpos, R.C Holt. “ACDC: An algorithm for comprehension-driven clustering”, Int. Working Conference on Reverse Engineering, 2001

[15] William E. Lorensen and Harvey E. Cline, Marching Cubes: A High Resolution 3D Surface Construction Algorithm", Computer Graphics (Proceedings of SIGGRAPH '87), Vol. 21, No. 4, pp. 163-169