VRCS: Integrating Version Control and Module Management using Interactive Three-Dimensional Graphics

Hideki Koike Hui-Chu Chu
Graduate School of Information Systems
University of Electro-Communications
Chofu, Tokyo 182, Japan
Tel: +81-424-83-2161
Email: {koike, chuchu}@vogue.is.uec.ac.jp

Abstract

Version control and module management are very important in practical software development. In UNIX, RCS or SCCS is used in general as version control tools. However, they (1) lack an ability of explicitly displaying the version changes; (2) require complicated commands to be typed; and (3) lack an ability of managing multiple files and modules.

This paper proposed a solution for these issues by applying 3-D visualization. The prototype system, VRCS, was developed. In our system, each version information stored in a RCS history file is displayed as a 2-D tree by taking the z-axis as time. Other 2-D trees are laid out in 3-D space in the same way. In our visualization, files which compose a certain release of the software are connected by line called relation-link. By using GUIs, users can check in/out each version easily and interactively. More importantly, just by choosing the relation-link, a certain release is rebuilt automatically.

1 Introduction

In practical software development process, version control and module management of program source codes and documents are very important. In UNIX, RCS (Revision Control System) [14] or SCCS (Source Code Control System) [11] are used in general as version control tools. Since RCS/SCCS have enough functions when used with a single file and they are relatively small in their application size, they have been used by UNIX experts.

However, these tools have not been put to good use of by novices because of the following reasons.

- *They lack an ability of explicitly displaying the version history.*
  Each version history is stored in a RCS/SCCS history file. It is, therefore, hard for users to understand intuitively the version history.
- *They require complicated commands to be typed.*
  In order to check in/out files, users have to learn and type complicated RCS/SCCS commands.

These issues can be mostly solved by providing visual interfaces. For example, a 2-D tree visualization of the version history will help users to understand the version history. The GUIs, such as menus, will be useful for reducing programmers typing efforts. Those ideas have been already implemented in some CASE (Computer Aided Software Engineering) tools [8].

Although such visual interfaces might accelerate novices to use RCS/SCCS, these tools still have the following major drawback.

- *They lack an ability of managing multiple files.*
  In general, a software system is composed with multiple files and each file has its own version history. In order to build a specific release of the system, programmers have to check out appropriate versions of all files, then they have to compile and link them. RCS/SCCS, however, do not keep the relation between files. Therefore, programmers, by themselves, have to remember the combination of these different versions.

This is a major reason why experts feel RCS/SCCS are not sufficient in real software development.

A possible solution to this issue is to introduce a new tool which unifies version control and module management, such as CASE tools, AFS [6], and so on. Such integrated tools, however, are often large in their application size and users have to learn many new commands. UNIX users who are familiar with RCS/SCCS do not want to use such tools.

This paper proposes an alternative solution to these issues with effective use of 3-D visual interfaces. We try to use traditional UNIX tools such as RCS and Make as they are, because RCS has enough functions when used with individual files and Make is sufficient for module management. Just by providing 3-D interfaces, we try to unify these existing tools.

The next section describes our 3-D visualization framework. Section 3 describes the system in detail. Section 4 discusses advantages and disadvantages of our system. Finally Section 5 concludes the paper.

2 3-D Visualization Framework

Koike proposed a 3-D visualization framework of software information in [4]. In his 3-D framework, two (or more) relations are visualized in one 3-D diagram so that each relation can be focused on by rotating users' viewing position with interactive animation. On the other hand, when two (or more) relations are visualized in traditional 2-D diagram, visualization tends to be very complex due to the link crossing and so on.

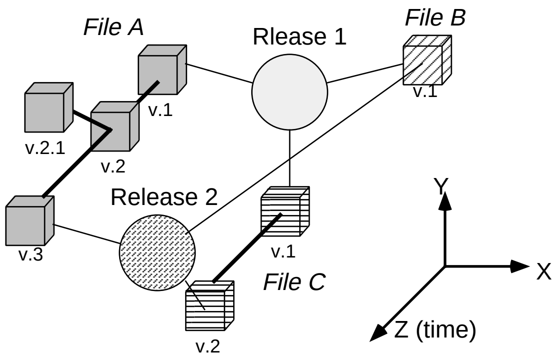


Figure 1: A 3-D framework for visualizing version/module information. Each version history is displayed as a 2-D tree. Files (trees) in the same module are placed physically near when looked along z-axis. Users can obtain both a version history view and a module structure view in one 3-D diagram. A certain release of the system is represented as sphere and versions composing the release are connected by link.

In [4], he also proposed a visualization of version/module information as one of its applications as shown in Figure 1. Each version history is displayed as a 2-D tree which is parallel to the yz-plane. The cube represents each version. Files (i.e. trees) in the same module are placed physically near when looked along z-axis. Versions which are necessary to build a certain executable file are connected by another type of link via special node called *release-node* which is represented as sphere and is placed on z-axis.

When users observe the visualization from the direction which is perpendicular to the z-axis, they can focus on each version history. When users look along z-axis, they can understand which files are used to build each release. Also, users can observe both relations simultaneously by choosing an appropriate viewing position.

3 VRCS

3.1 Implementation

Based on the framework described in the previous section, a system named *VRCS* was developed. VRCS is written in C with our software visualization tool VOGUE [4], and runs on Silicon Graphics workstation. Figure 2 shows an architecture of VRCS.

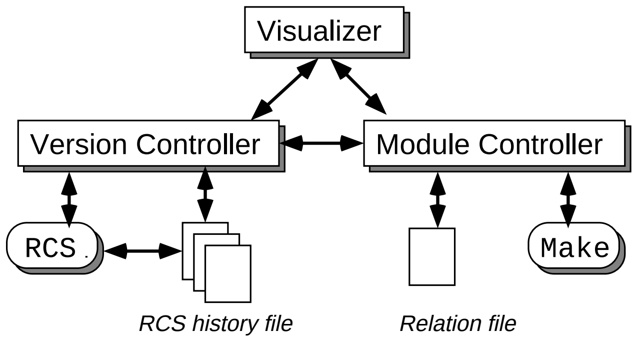


Figure 2: An architecture of VRCS.

The system has three major components, version controller, module controller, and visualizer.

- Version controller
  Version controller parses each RCS history file (which is located in the RCS directory and indicated by the extention ",v"), separates each version information, and connects versions which have parent-child relation by link called *version-link*.

- Module controller
  In order to keep the relation between files, an additional file called *relation-file*, which is also placed in the RCS directory, is introduced. The relation-file keeps release numbers and filenames with their version numbers. Module controller reads the relation-file, creates release nodes, and connects the release and versions by link called *relation-link*.

- Visualizer
  Visualizer displays the structures created by version controller and module controller. Each version history is displayed as 2-D tree where version nodes and version-links are represented as cubes and lines respectively. A different color is assigned to each tree. Also, release nodes and relation-links are represented as spheres and lines respectively.

Figure 3 shows snapshots of VRCS viewed from different positions. In this example, there are two releases, both of which are composed of 10 files. As you can see, some files are changed frequently and others are not. It is also visually clear that which versions are used to build each release.

3.2 Interaction with a Single File

When a user invokes the system with a file name as an argument, a 2-D tree appears on the screen as shown in Figure 4. To check out a certain version of the file, the user picks a corresponding node by mouse and selects "Check Out" from the menu. A target version of the file is retrieved by RCS and is opened with an text editor.

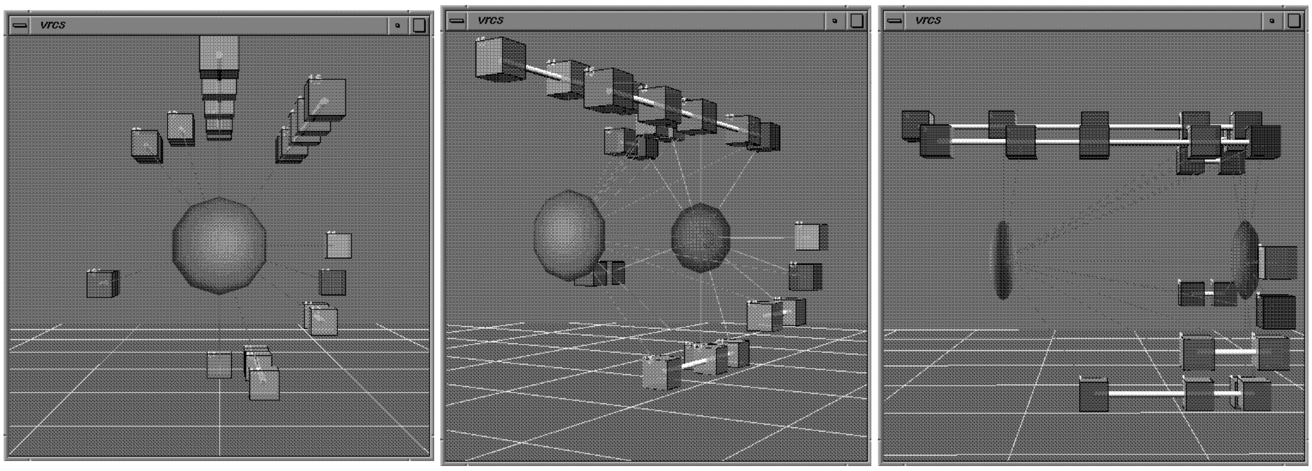


Figure 3: Snapshots of VRCS. In this example, there are 2 releases (represented as sphere) both of which are composed of 10 files. It is visually clear that which file is frequently changed and which file is not. It is also clear that which versions are used to build each release.

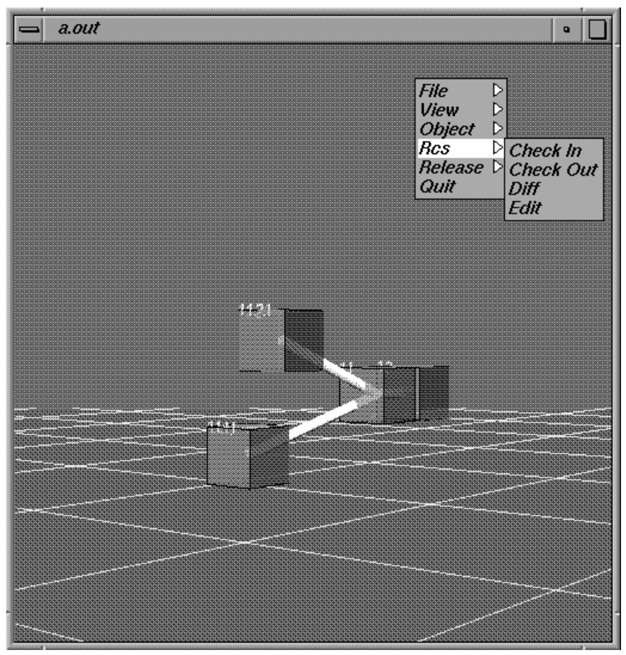


Figure 4: Interacting with a version history of a single file. A user can execute RCS operations by using menu.

In the same way, to check in a file, the user picks any node in the tree and selects "Check In" from the menu. After the file is checked in the RCS history file, a new node appears on the screen. If the user wants to compare two versions, he/she picks two nodes and selects "Diff" from the menu. The difference is displayed on an editor window.

3.3 Interaction with Multiple Files

When the user invokes the system with a directory name, the system first reads a relation file under the RCS directory. Then it parses each RCS history file, visualizes them as individual 2-D trees, makes release nodes on z-axis, makes links between each release node and version nodes which are necessary to build the release.

When the user needs to retrieve all files which compose the release, he/she picks a corresponding release node and selects "Check Out" from the menu. RCS checks out all versions connected by relation links.

If the user needs to get an executable file for the release, he/she selects "Make" from the menu. The system retrieves all files which are necessary to rebuild the executable file, compiles them, and links them. The user does not need to remember which versions are necessary. Just by choosing a release node and by selecting "Make", the target executable file is rebuilt automatically.

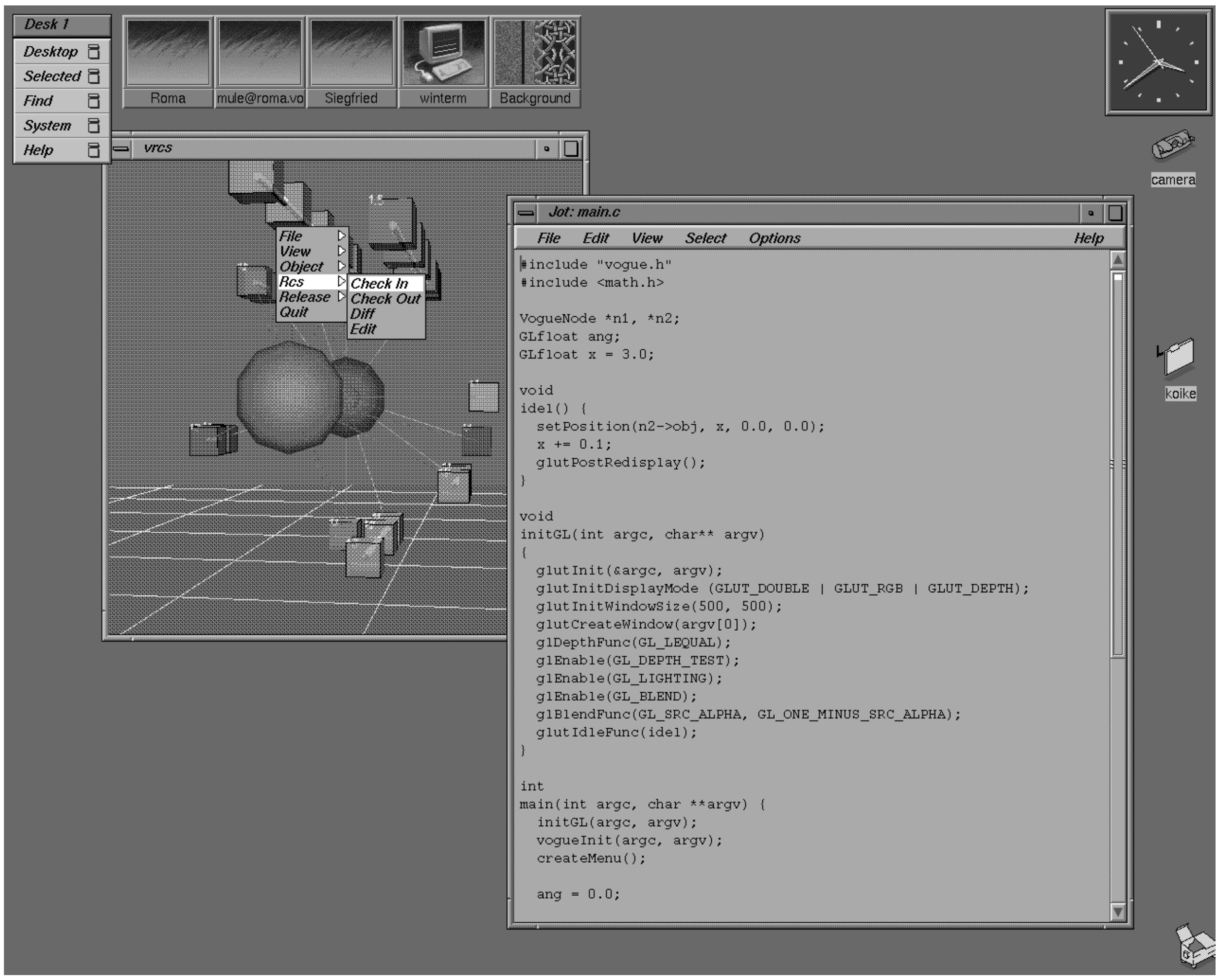
The user can take a snapshot of versions at anytime he/she wants by selecting "Snapshot" from the menu. The system checks in every files in the directory and writes the combination of versions into the relation file.

4 Discussions

VRCS is in experimental use in our laboratory. Through our experiments, the following results are currently observed.

First, since each version history is explicitly shown to users, it gets easier to understand the version history. Because of visual interfaces for RCS, it is easy to check out the previous versions. Traditionally it is necessary to remember the version number and to type RCS commands correctly.

Second, by using 3-D visualization framework, module information and version information are viewed simultaneously. In software development and maintenance, it is very important to recognize both



```
#include "vogue.h"
#include <math.h>

VogueNode *n1, *n2;
GLfloat ang;
GLfloat x = 3.0;

void
idel() {
  setPosition(n2->obj, x, 0.0, 0.0);
  x += 0.1;
  glutPostRedisplay();
}

void
initGL(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
  glutInitWindowSize(500, 500);
  glutCreateWindow(argv[0]);
  glDepthFunc(GL_LEQUAL);
  glEnable(GL_DEPTH_TEST);
  glEnable(GL_LIGHTING);
  glEnable(GL_BLEND);
  glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
  glutIdleFunc(idel);
}

int
main(int argc, char **argv) {
  initGL(argc, argv);
  vogueInit(argc, argv);
  createMenu();

  ang = 0.0;
```

Figure 5: Visualizing version histories of multiple files. Versions composing a certain release are connected by relation link via a release node (represented as sphere). When a user selects "Check Out" from the menu, a corresponding version is retrieved by RCS and is opened with text editor.

information. For example, it is possible to know which module is stable and which module is frequently changed just by seeing our visualization.

Third, it is easy to understand relations between files because the relations are explicitly displayed. Then, in cooperation with Make, users can automatically check out files, compile them, and link them with fewer operations. For example, when N files are used to build a release, programmers usually execute N RCS operations to retrieve appropriate versions. On the other hand, the programmers pick a sphere node and select "Check Out" to retrieve all versions with VRCS.

There are some problems to be solved in our system. First, improvements of visual designs are necessary. The system currently uses simple boxes and lines to represent version histories. Visual vocabularies such as the use of colors, shapes, icons, and so on, should be considered.

Second, one of main concerns in 3-D visualization is how to reduce the number of displayed objects. The refresh rate of graphic displays gets slower with an increase in the number of graphic elements. Moreover, the increase in the number of graphic elements bothers users' cognition. It is necessary to apply some techniques which enable users to control the number of displayed information (e.g. [3, 5]).

Third, our system should be extended to multi-user system. Currently, the system can be used by one user. However, practical softwares are developed by multiple users. If our system is used in such situation, each user will be able to understand the development situations of other programmers. Also, they will be able to understand how their modifications to their programs affect to others through the visualization. It is important to give such macroscopic information to each programmer. We believe our system can support this.

5 Related Work

GNU Make behaves as module management tool just like traditional Make. GNU Make also has an ability of version control in cooperation with traditional version control tools such as RCS. Because of its upper-compatibility with traditional Make, GNU Make is now getting popular in UNIX community. AFS [6] is another example which integrates version control and module management. AFS, however, is not widely accepted by UNIX users because it requires special directory set-ups and is incompatible with traditional tools. Although GNU Make and AFS are capable of managing multiple files, they are command oriented tools, and therefore they are still hard to be used by novices.

CASE tools also integrates version control and module management. Some of them have 2-D visual interfaces, where version histories are displayed as 2-D trees in one window and module structures are displayed as 2-D tree or bubble chart in another window. In general, CASE tools are huge in their application size and they require much effort to learn. Therefore, they are not accepted by ordinary UNIX users. Also, the users have to see the version histories and the module structures in seperate windows although the version information and the module information are inherently unseperable.

The 3-D visualization work was pioneered by SemNet [2] and Information Visualizer [1, 10, 7]. We also have been working with 3-D visualization and proposed a differenct 3-D framework which uses the third axis more actively [4]. SemNet and Information Visualizer display only one relation at a time. Although users change their viewpoints, the obtained information is the same. For example, Cone Tree [10] shows one hierarchical structure whichever viewpoints users choose. On the other hand, our 3-D framework makes it possible to display two different relations without disturbing each other. Such use of the third axis are also seen in [9] and [13].

6 Conclusions

The following conclusions are currently obtained in our research.

1. A complete 3-D visualization framework is proposed, where version information and module information are visualized simultaneously. By effective use of 3-D representation and animated rotation, it is possible to minimize the complexity of a diagram even though two different relations are visualized.

2. Based on this framework, the system was implemented. The system displays each RCS history file as a 2-D tree, and files in the same module are placed physically near each other. Instead of typing RCS commands, these commands are appeared as menu. This makes version control much easier.

3. Versions which should be linked together are explicitly represented by relation links. Moreover, just by simple operations with mouse, it is possible to check out all the related files, compile them, and link them automatically.

Our purpose is not to make new version/module control tool, but to integrate traditional version/module control tools by providing effective 3-D interfaces. This is our ongoing research. And we are now trying to evaluate our system by comparative experiments with existing tools.

Acknowledgments

This work was supported in part by the Foundation "Hattori Hokokai".

References

[1] S. K. Card, G. G. Robertson, and J. D. Mackinlay. The Information Visualizer, an information workspace. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'91)*, pages 181–188. ACM Press, 1991.

[2] K. M. Fairchild, S. E. Poltrock, and G. W. Furnas. SemNet: Three-dimensional graphic representation of large knowledge bases. In R. Guindon, editor, *Cognitive Science And Its Applications For Human-Computer Interaction*, pages 201–233. Lawrence Erlbaum Associates, 1988.

[3] G. W. Furnas. Generalized fisheye views. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'86)*, pages 16–23. ACM Press, 1986.

[4] H. Koike. The role of another spatial dimension in software visualization. *ACM Trans. on Information Systems*, 11(3):266–286, July 1993.

[5] H. Koike. Fractal views: A fractal-based method for controlling information display. *ACM Trans. on Information Systems*, 13(3):305–323, 1995.

[6] A. Lampen. Advanced file to attributed software object. In *USENIX-Winter '91*, 1991.

[7] J. D. Mackinlay, G. G. Robertson, and S. K. Card. The perspective wall: detail and context smoothly integrated. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'91)*, pages 173–179. ACM Press, 1991.

[8] C McClure. *CASE is Software Automation.* Prentice-Hall, 1989.

[9] M.H.Brown and M.Najork. Algorithm animation using 3d interactive graphics. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'93)*, 1993.

[10] G. G. Robertson, J. D. Mackinlay, and S. K. Card. Cone Trees: Animated 3D visualizations of hierarchical information. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'91)*, pages 189–194. ACM Press, 1991.

[11] M. J. Rochkind. The source code cotrol system. *IEEE Trans. on Software Engineering*, pages 255–265, 1975.

[12] J. L. Steffen S. G. Eick and E. E. Sumner Jr. Seesoft - a tool for visualizing line-oriented software statistics. *IEEE Trans. on Software Engineering*, 18(11):957–968, 1992.

[13] J. T. Stasko and J. F.Wehrli. Three-dimensional computation visualization. In *Proceedings of 1993 IEEE/CS Symposium on Visual Languages (VL'93)*, 1993.

[14] W. F. Tichy. Rcs–a system for version control. *Software – Practice & Experience*, pages 637–654, 1985.