GRADE-IV: Visualizing Graphics Library Operations in an Executing Program

Hidemiko Abe
Google

Takeo Igarashi
The University of Tokyo

Figure 1: To help in debugging errors, our system intercepts OpenGL calls inside of an executing target application (a) and visualizes invisible information through a third person view (b), the line number where each vertex is drawn (c), and spatial transformation chains (d).

Abstract

It is difficult for inexperienced programmers to debug three-dimensional (3D) computer graphics programs. One reason is that the relationship between the visual information rendered on the screen and the numerical information in the source code is difficult to understand. This sketch introduces a system to help programmers understand and debug computer graphics programs by visualizing those hidden relationships. The system allows the user to observe the spatial relationship between the camera and the scene from a third person view. The system also tells the user in which statement of the code each vertex position is specified. In addition, the system visualizes the sequence of spatial transformations applied to each vertex. To support these services, the system continuously observes the access to graphics library commands in the executing program and reconstructs the visual representation associated with the corresponding statements in the code. We implemented a prototype system, called GradeIV, using Java Debug Interface Library and Java Binding for the OpenGL to show the feasibility of the idea.

1 The User Interface

The GradeIV system helps programmers to understand and debug 3D computer graphics programs by visualizing the hidden relationship between the source code and the graphical information on the screen. It provides three services to the user to investigate the behavior of an executing code. First, the system allows the user to observe the spatial relationship between the camera and the scene from a third person view (Figure 1b). This is especially helpful for understanding what is happening when nothing appears on the screen because of an incorrect camera setting. Second, the system tells the user in which statement of the code each vertex position is given (Figure 1c). This facilitates the identification of problematic statements that cause unexpected behavior on the screen. Third, the system visualizes the sequence of spatial transformations applied to each vertex (Figure 1d). This helps the user to understand the basics of spatial transformation, which is particularly difficult for inexperienced programmers.

2 Implementation

The system observes the behavior of an executing target application program from an independent observer process in real time. It can therefore continuously show what is happening inside of the target program during user interactions such as clicking and dragging. This is far more effective for debugging interactive applications than making a log during execution and inspecting it afterward. We implemented a prototype system for debugging Java programs that uses Java Binding for the OpenGL (jogl) as graphics library. It intercepts access to OpenGL commands using the Java Debug Interface Library (JDI) and reconstructs the visual representation associated with the corresponding statements in the code (Figure 2).

3 Related Work

GLTrace intercepts OpenGL API invocations and records the API command sequences [Hawk Software 2005] like our system, but it does not associate program code with the visualization. HijackGL intercepts OpenGL calls to change the rendering style without modifying the original application [Mohr and Gleicher 2002]. The system of Duca et al. records OpenGL command sequences in a database and allows the user to access the data using an SQL-like query language [Duca et al. 2005].

References

