3D Graphics Module

By Ramesh Srigiriraju

Computer Systems Research Lab 2006-07

Abstract

The purpose of this research project is to test different data storage methods for a 3D graphics program. The testing was done using a graphing calculator program that allowed the user to rotate the graphs while viewing them. The program will be included in the student Intranet as a module and thus includes elements of modular design. The program used four different data storage schemes: one used matrix expression trees and column vectors, another stored data as row vectors, and another used no matrix expression trees at all. Another test case used matrix expression trees, but the data points were all calculated at the beginning instead of being recalculated each time, like in the no-tree test case.

Background & Introduction

Previous projects concerning this area of research include The Investigation of Graphics in the Processing Language by C. Fralick, the City Block Project by M. Levoy, and TJForge Iodine for the modular programming component. The 3D graphics projects seemed to use rotation matrices to rotate graphs by an angle a. Iodine used HTML to program in the modules. Possible state-of-the art programs could be MatLab or other computer algebra systems or even the 3D-graphing feature of the TI-89.

Procedures

My lifecycle model the Staged Delivery, where every few weeks I add functionality to an older version of my program with a specific goal in mind. These versions were created to test my data storage schemes for functionality. In the case of my graphing calculator module, they were also used to test my schemes for speed and efficiency. To test my programs, I had my graphing calculator rotate the graphs 10 million times, and the first million test cases were ignored. The exclusion was done because the programs had a tendency to get faster until they reached a “steady state”, and I used 9 million data points to reduce the effects of outliers.

Results

The first data storage scheme (matrix expression trees, data stored as column vectors, data recalculated each time) was by far the slowest, taking 2685 ns. The last scheme (hard-coded formulas) was the fastest, taking only 2440 ns to run. Storing data as row vectors (scheme 2) improved performance, since this only took 2513 ns. However, calculating the data only once at the beginning actually took more time because of the use of temporary variables (scheme 3, 2592 ns).

Average Runtimes (ns)

Scheme 1: 2685
Scheme 2: 2513
Scheme 3: 2592
Scheme 4: 2440