Technical Report CAIP-TR-259

WORLDTOOLKIT VS. JAVA3D: A PERFORMANCE COMPARISON

Rares F. Boian and Grigore C. Burdea

26 April 2001

CENTER FOR ADVANCED INFORMATION PROCESSING
CoRE Building - Frelinghuysen Road
Rutgers - The State University of New Jersey
Piscataway, New Jersey 08855-1390
(732) 445-4208

Abstract

This report compares the performance of WorldToolKit (releases 8 and 9) and Java3D (versions 1.1.3 and 1.2) running a VR simulation on a dual-processor 450 MHz PC. The simulation was designed to run in several configurations having different interaction levels (no interaction, fly-through and haptic interaction), rendering modes (wireframe, Gouraud and textured), graphics modes (mono and stereo) and scene complexities (5,000–50,000 polygons). Results show that overall Java3D is faster than WTK in terms of frame refresh rates. However, WTK has a more uniform frame rendering time, which results in more predictable visual feedback.

Acknowledgement

The research reported here was made possible through the support of the New Jersey Commission on Science and Technology (R&D Excellence Grant).

Table of Contents

List of Figures

1. Introduction

Virtual reality libraries can be classified as commercial toolkits and non-commercial (free) software [3]. The most widespread commercial VR library is WorldToolKit (WTK) sold by Engineering Animation Inc. [4]. Over the past decade is has evolved in many versions, and now has reached release 9. The number of functions in the library has steadily increased, but price has kept an order of magnitude above standard PC software. Sun Microsystems has been developing Java3D, a portable simulation library that is free [7][5]. It too has evolved in recent years, and now has version 1.2.

Selecting between WTK and Java3D means weighing a significant up-front cost vs. long term savings in development costs. These long-term savings are due to the VR-specific drivers included in WTK, but which are missing in the current version of Java3D. Regardless of the cost, is there a performance penalty in selecting Java3D? To begin to answer this question a comparative study was done with a benchmark simulation that ran on a dual-processor 450 MHz PC. The study compared WorldToolKit (releases 8 and 9) and Java3D (versions 1.1.3 and 1.2). Section 2 describes the virtual scene and simulation modes. Experimental results are given in Section 3. Section 4 concludes this paper.

2. Experimental Setup

The simulation ran on a Dell Workstation Dual PentiumII 450MHz, 128M RAM, with an Accel-Galaxy graphics accelerator with 56M RAM. The I/O tools for interaction with the virtual world were a Polhemus Insidetrack [6] and an Rutgers Master II-ND Force Feedback Glove [1]. This haptic glove reads user's hand gesture and provides force feedback to the fingertips [2].

The benchmark consisted of a 3D simulation that ran in several different configurations meant to test three important aspects of virtual reality simulations: graphical refresh rate, constancy of the graphical refresh rate and i/o device update rates. The simulation was implemented in WorldToolKit and Java3D using identical geometrical models. The benchmark was run using two versions of each toolkit.

The comparisons were made between WTK R8 and Java3D 1.1.3 and between WTK R9 and Java3D 1.2. The scene consisted of a number of spheres of 420 polygons each arranged in a star-like structure, as well as a 2,270 polygon virtual hand. The group of spheres rotated constantly around an arbitrarily chosen axis. Depending on the configuration in which the simulation was run the virtual hand rotated around an axis or it moved according to the user hand movement. Five parameters were set in the simulation, to see how the two toolkits behaved. These parameters were interactivity, graphics mode, rendering mode, number of polygons in the scene and number of light sources, as summarized in table 2.1.

The three values of the interactivity variables were:

1. "Non interactive". In this configuration, the application was a simple 3D animation. This was used as a baseline for comparison.
2. "Hand input". The application read the glove sensors and updated the virtual hand's position accordingly. From a computational point of view this configuration is similar to a walk-through in which there is no manipulation of virtual objects.
3. "Force feedback". The application performed bounding box collision detection between the virtual hand and the spheres. A simple gesture recognition algorithm decided if the hand was closed or opened. When a sphere was grasped the user had force feedback through the RMII-ND glove.

The number of spheres contained in the scene was a function of the parameter giving the total number of polygons. For a certain value of this parameter, the scene contained the virtual hand

Table 2.1: Benchmark parameters

Number of polygons	5,000, 10,000, 20,000, 30,000, 40,000, 50,000
Number of lights	1, 5, 10
Interaction	Non interactive, Hand input, Force feedback
Rendering mode	Wireframe, Gouraud, Textured
Graphics mode	Mono, Stereo

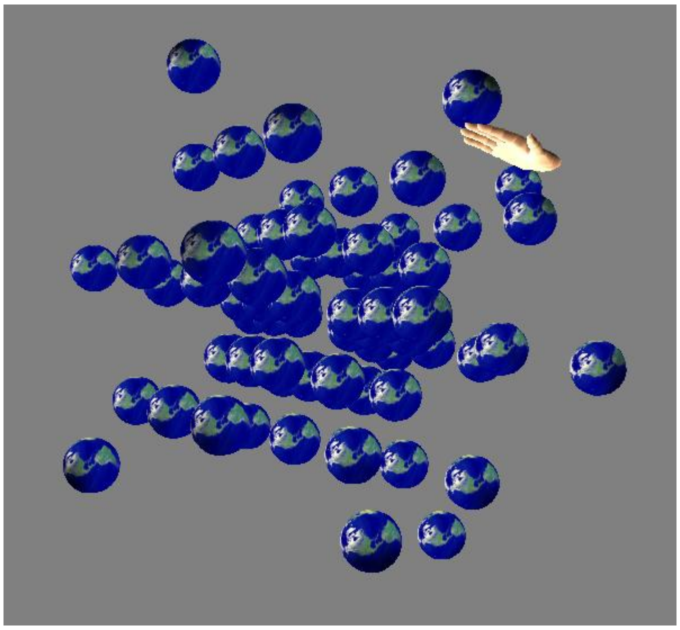

Figure 2.1: The simulation scene

and a number of spheres computed with the formula:

$$NumerOfSpheres = floor\left(\frac{NumberOfPolygons - 2270}{420}\right)$$

3. Results

The application was tested for a number of 324 configurations. Each configuration was executed for 1000 frames or for one minute, whichever came first. The application stored in a buffer the system time at the beginning of each frame. At the end of the simulation the buffer was written into a file. For the configurations that read the glove sensors the application wrote into files the update rate of the i/o drivers. This data was used to plot the frame rendering time for each of the 324 configurations. To make data more comprehensible the average graphical refresh rates and driver update rates were also plotted for configurations that differed only in the number of polygons and number of light sources.

3.1 Graphical Refresh Rate

Figures 3.1 and 3.2 contain the average refresh rates for configurations differing only in the number of polygons and light sources. The notations on the X-axis of the graphs have the following meaning:

- On the lowest level are the three values of the interactivity parameter (non-interactive, hand input and force feedback).
- The middle level shows the rendering mode (W for wireframe, G for Gouraud and T for textured).
- The level closest to the axis shows the graphics mode parameter (M for mono and S for stereo).

For the configurations with wireframe rendering, Java3D 1.1.3 has a lower frame rate than WTK R8 (figure 3.1). In figure 3.2 showing the next releases side by side, it can be seen that Java3D 1.2 improved its performance and is sometimes faster than WTK. One needs to mention that in wireframe rendering Java3D computes the lighting while WTK does not. This is an important factor that affects Java3D's performance. The authors could not disable the lighting computation of a wireframe scene in Java3D nor could they enable it in WTK.

For all the textured configurations WTK had higher refresh rates than Java3D. Measurements showed that Java3D seemed to have a memory leak. The frame rendering times of Java3D started at lower values than WTK's. However, they were steadily increasing as the application was running, as seen in figure 3.3. Watching the memory used while the benchmark was running, it was determined that size increased over time. When rendering was done in textured mode, this behavior was present in both versions of Java3D.

For the Gouraud shaded configurations, the graphical frame rate of Java3D was equal or higher than WTK's in both 1.1.3 and 1.2 versions. Since these configurations were not affected by memory artifacts, they were used in the remainder of this study.

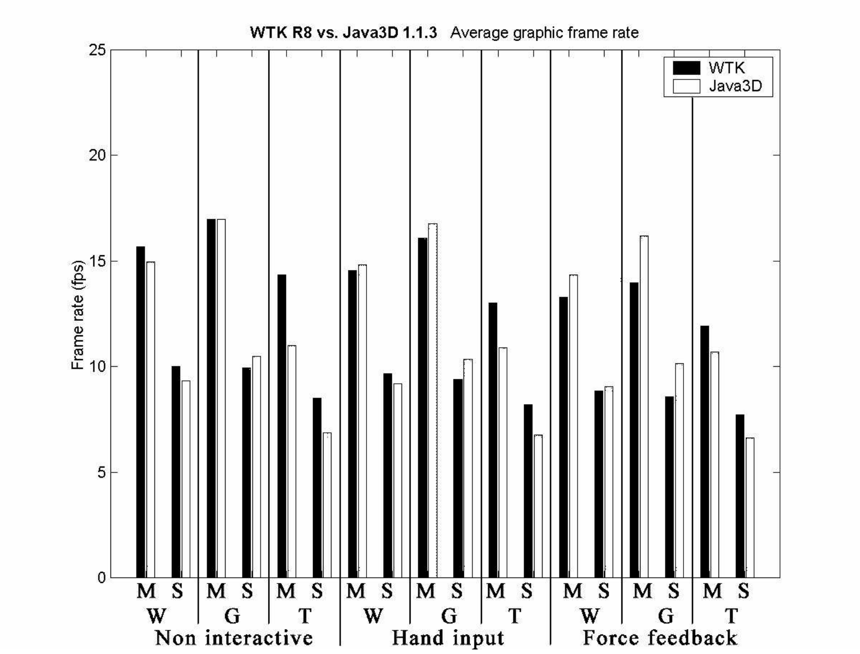

Figure 3.1: Average frame rates - WTK R8 vs. Java3D 1.1.3

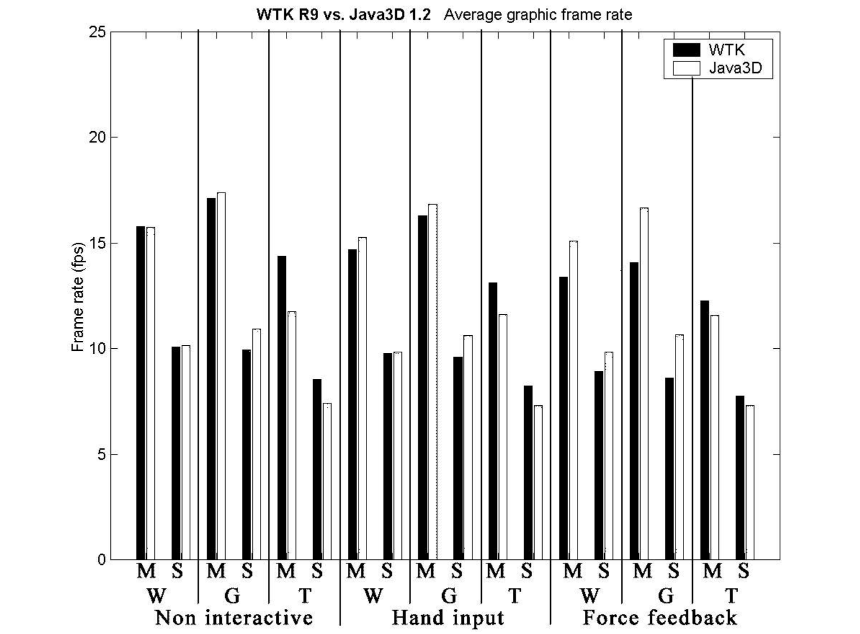

Figure 3.2: Average frame rates - WTK R9 vs. Java3D 1.2

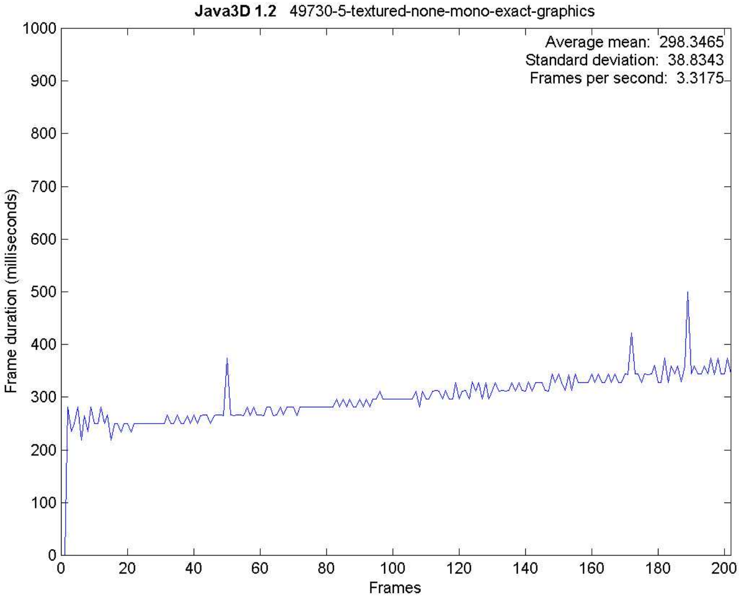

Figure 3.3: Frame rendering times for Java3D in a textured configuration

3.2 Frame Rendering Time

It was shown previously that Java3D has a higher frame rate than WTK for equivalent configurations. Figure 3.4 to 3.7 plot the frame rendering times of both toolkits for a configuration with 50,000 polygons, five light sources, Gouraud shading, interactivity and stereo graphics mode. It can be seen that both WTK release 8 and 9 have very good constancy in frame rendering time. The earlier version of Java3D exhibits numerous spikes (poor constancy), presumably due to the garbage collection mechanism which is activated periodically. Such spikes means that the virtual scene freezes momentary, which in turn affects negatively user immersion and interactivity. Fortunately, the latest version of Java3D (1.2) shows a nice improvement in frame rendering times constancy. As seen in the figure 3.7 the spike frequency has decreased substantially.

A summarizing graph, containing the standard deviations of the frame rendering times for configurations differing only in the number of polygons and the number of lights can be seen in figures 3.8 and 3.9. Java3D is less constant than WorldToolKit in all configurations. For the textured configurations the variation is much higher due to the constant increasing in the frame rendering duration (as explained previously). Java3D's 1.2 has slightly higher standard deviation of the frame rendering time vs. version 1.1.3 for the first two levels of interaction. This standard deviation drops by some 10% for the simulation configurations using haptics. These are the ones performing collision detection, an essential component of physical simulation.

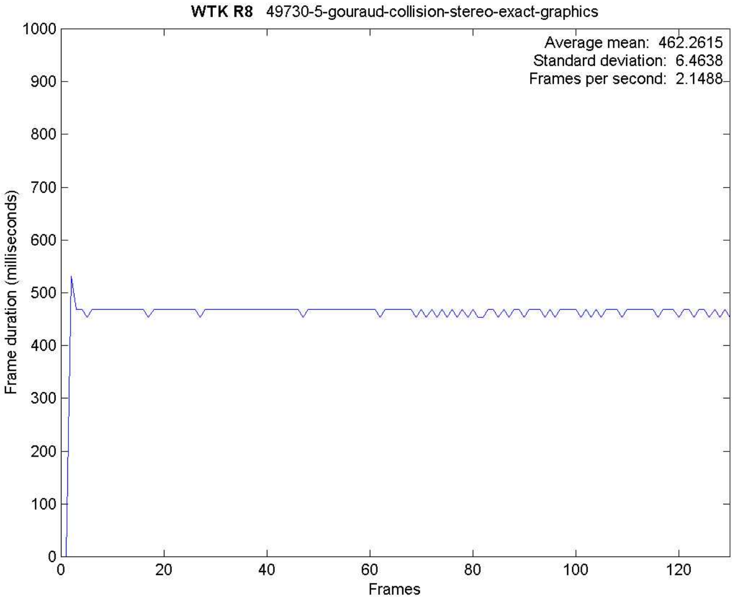

Figure 3.4: Frame rendering times fro WTK R8

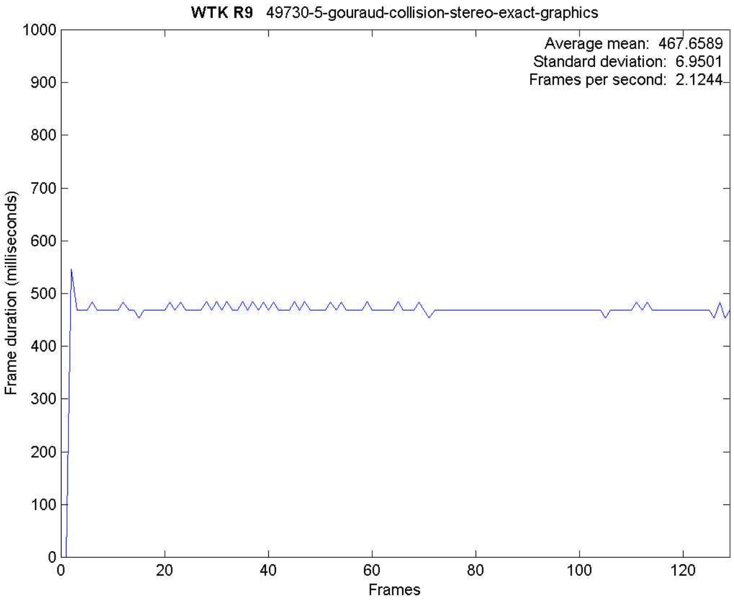

Figure 3.5: Frame rendering times for WTK R9

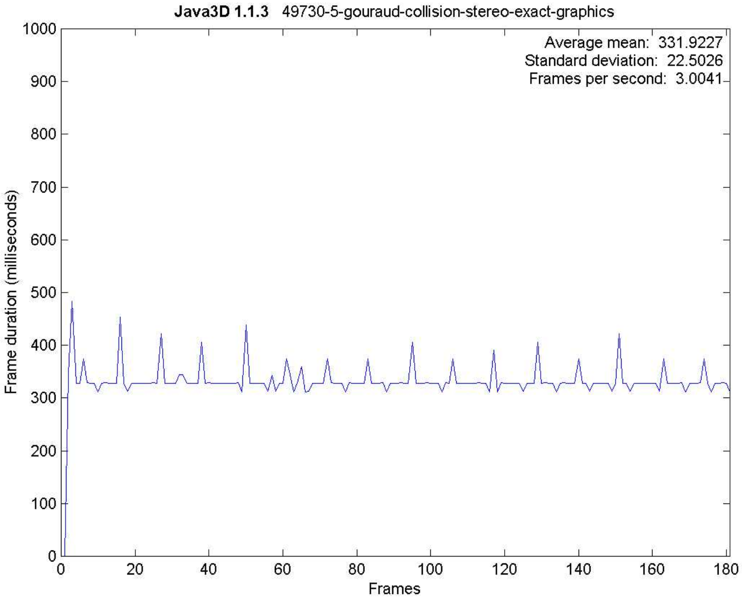

Figure 3.6: Frame rendering times for Java3D 1.1.3

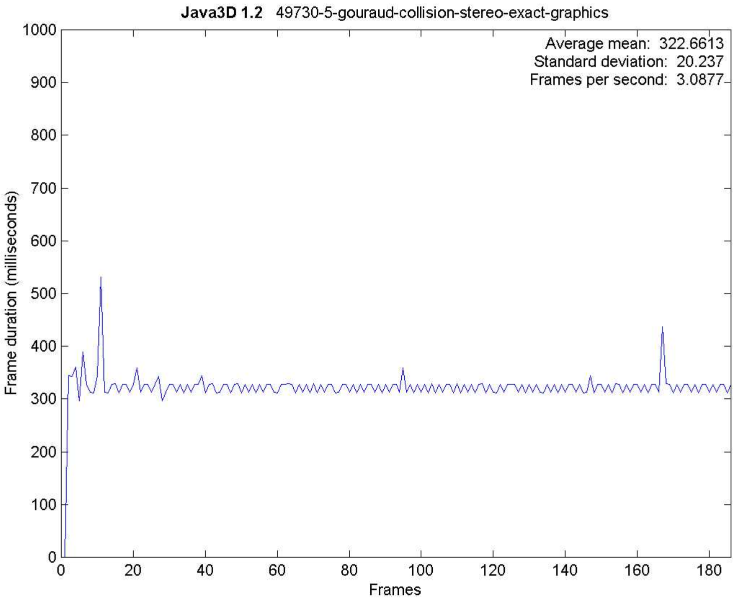

Figure 3.7: Frame rendering times for Java3D 1.2

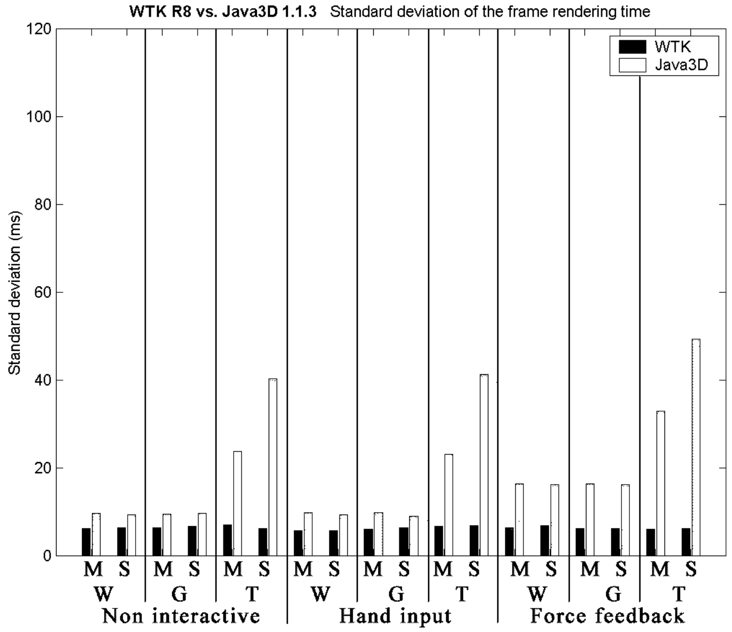

Figure 3.8: Variation of the frame rendering time - WTK R8 vs. Java3D 1.1.3

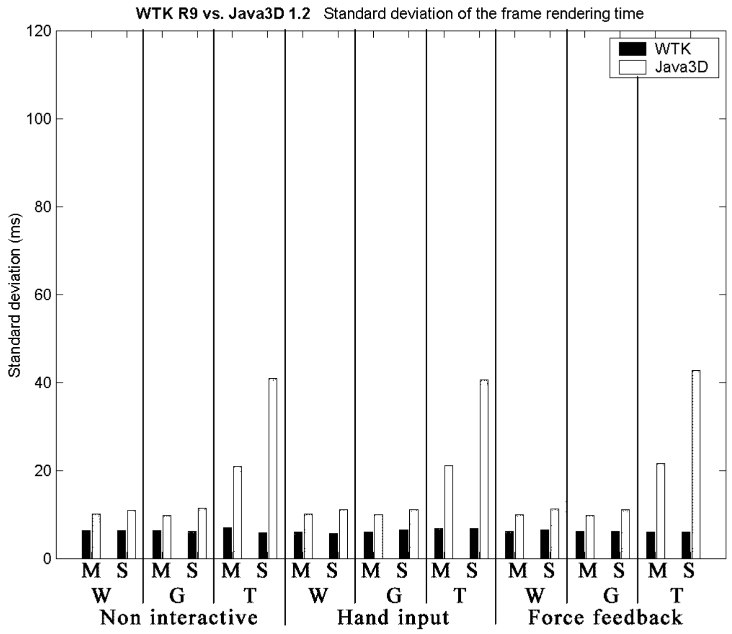

Figure 3.9: Variation of the frame rendering time - WTK R9 vs. Java3D 1.2

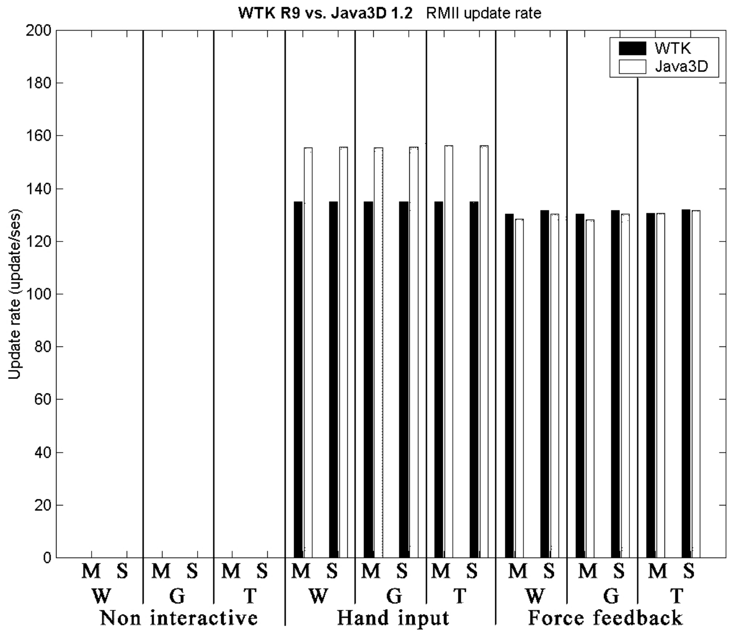

Figure 3.10: RMII driver update rate

3.3 Sensor Update Rates

The devices that provided input for the benchmark simulation were a Polhemus Insidetrack position sensor and a Rutgers Master II-ND Force Feedback Glove. Measurements showed that the Insidetrack had a constant rate of 60 updates/sec regardless of the simulation configuration.

The custom driver for the haptic glove was a thread written in C (accessed with JNI in Java3D). It communicated with the RMII system over the serial port of the host computer. Figures 3.10 and 3.11 show that the driver update rate was higher for Java3D when no force feedback was present (no collision detection performed). This rate dropped by about 17% for the simulation configuration that had haptics. In such case it was substantially equal to that of WTK. This behavior was present in both versions of Java3D, and may be due to the additional driver thread computation load for bounding box collision detection. Both WTK releases tested had essentially the same driver update rate, wether collision detection was done or not.

3.4 The Effect of the Scene Complexity on Performance

The complexity of the rendered scene has a major impact on the performance of a simulation. The complexity of scenes used in this study varied from very simple to beyond the point where the simulation was usable. When using a toolkit, it is important to know how the performance of a simulation degrades due to scene complexity.The behavior of the graphical frame rate can be seen in figure 3.12. Both toolkits follow similar descending paths, with Java3D showing a better performance.

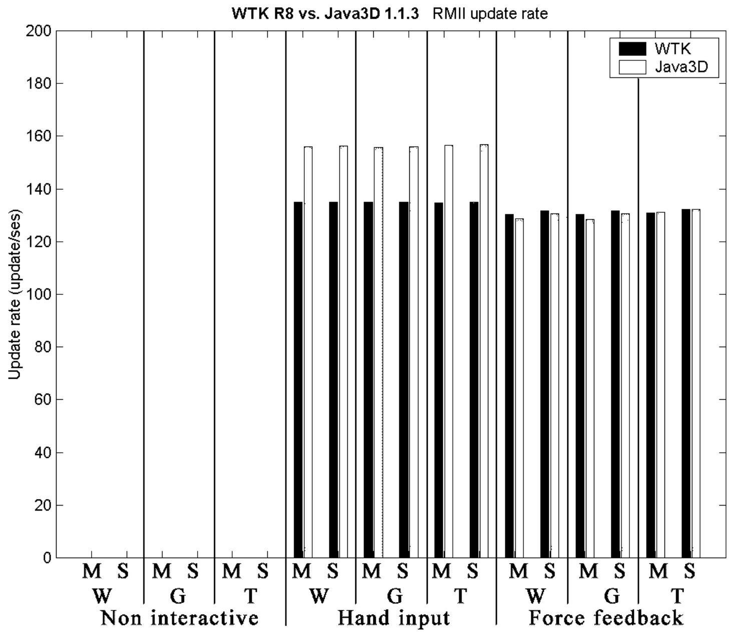

Figure 3.11: RMII driver update rate

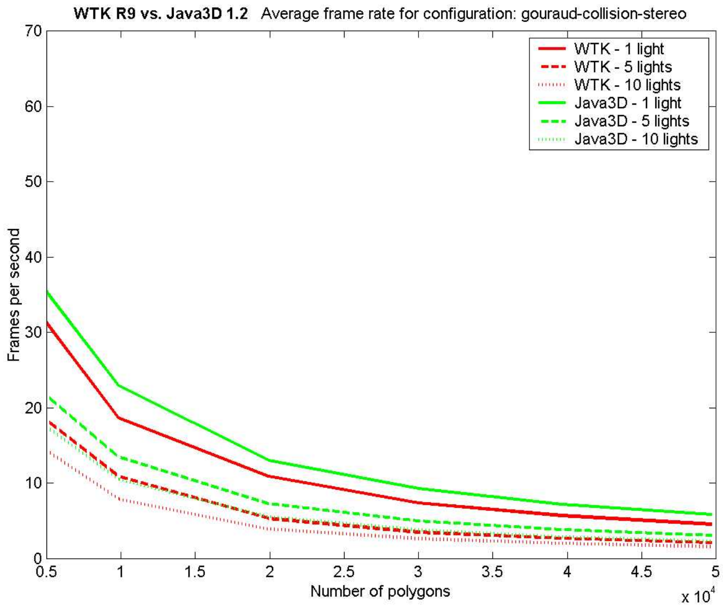

Figure 3.12: Graphic frame rate at different scene complexities

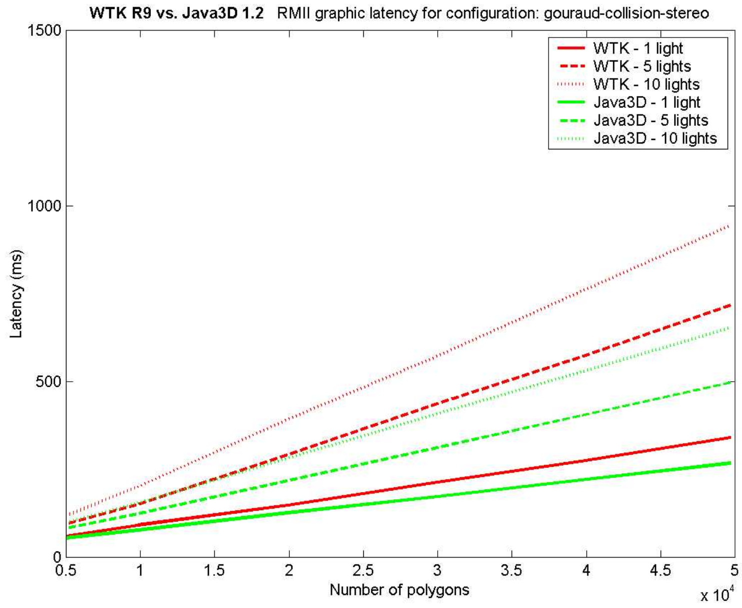

Figure 3.13: Latencies at different scene complexities for the RMII dirver

The latency of the sensors increases as the graphic frame rate decreases. The latencies of the the two input devices used in this study (RMII Glove and Polhemus Insidetrak) can be seen in figures 3.13 and 3.14. We can see that the scene complexity affects the latency of the input devices more than it affects the graphic frame rate. That is caused by the thread running the graphic rendering engine having a higher priority. When the scene is complex the rendering engine takes more CPU time slowing down the driver threads.

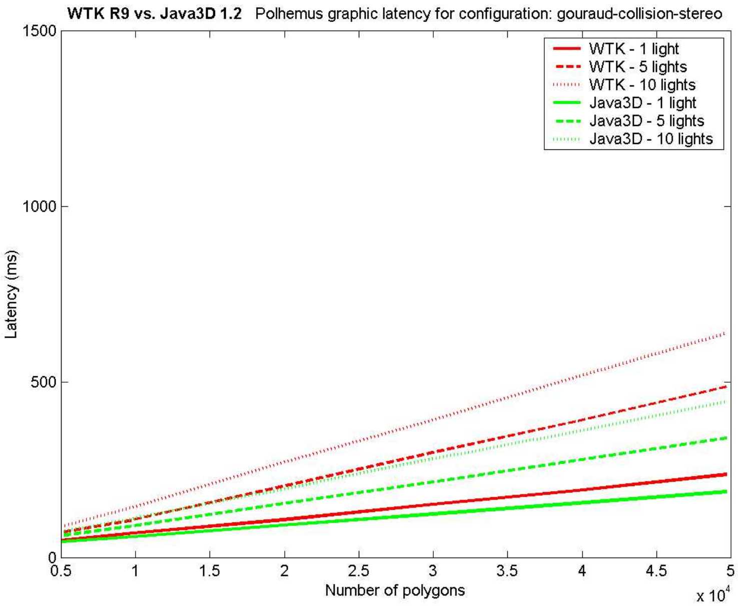

Figure 3.14: Latencies at different scene complexities for the PolhemusInsidetrack driver

4. Conclusions

Looking at the performance offered by the WTK and Java3D, it appears that Java3D is slightly faster in terms of rendering speed and sensor update rates. Unfortunately, version 1.1.1 has poor constancy in terms of frame rendering time, owing to its garbage collection mechanism. The newer version of Java3D has improved significantly in this respect, yet it is still behind WTK in terms of rendering speed constancy.

Although, the two toolkits have comparable scene graph handling capabilities, at development time WorldToolKit offers features not implemented in Java3D. The most important feature missing in Java3D is the support for third party I/O devices. WTK supports a wide range of sensors, displays and feedback devices while the only devices supported by the Java3D are keyboard and mouse. If any other devices are to be used with the application (and this is often the case) the developer has to integrate the drivers himself. Because the device divers are usually written in C/C++, a Java3D application is not so easy to port on a new platform. To do this, one would have to obtain or write drivers for the I/O devices and than re-integrate everything. From this point of view, WorldToolKit is a painless experience, due to its cross-platform I/O device support.

References

1. M. Bouzit, G. Burdea, V. Popescu, and R. Boian. The rutgers master force feedback glove. *IEEE Transactions on Mechatronics, Special Issue on Haptic Interfaces*, July 2000.

2. C. Grigore Burdea. *Force and Touch Feedback for Virtual Reality.* John Wiley & Sons, New York, 1996.

3. C. Grigore Burdea and Philippe Coiffet. *Virtual Reality Technology.* John Wiley & Sons, New York, 1994.

4. Engineering Animation, Inc. *WorldToolkit Reference Manual*, 1999.

5. Merlin Hughes. 3d graphic java: Render fractal landscapes. *JavaWorld*, August 1998.

6. Polhemus, Inc., http://www.polhemus.com.

7. Sun Microsystems, Inc. *Java 3DTM Documentation*, 1999-2000.