A Tool for Automated Performance Testing of Java3D Applications in Agile Environments

Xueling Shu
Department of Computer Science
University of Calgary
Calgary, Alberta, Canada
shu@cpsc.ucalgary.ca

Frank Maurer
Department of Computer Science
University of Calgary
Calgary, Alberta, Canada
maurer@cpsc.ucalgary.ca

Abstract

Following the agile philosophy that all core features of a system need an automated test harness, performance requirements also need such a check when they are essential for the success of a project. The purpose of this paper is to describe a tool, J3DPerfUnit, which supports automated performance testing for Java3D applications in agile environments. We elicited tool requirements from domain experts through a survey and evaluated J3DPerfUnit using code from our partner's bioinformatics project and Java3D official tutorials. The evaluation pointed out that the tool is effective in detecting performance problems and in identifying where they come from when loading/unloading a 3D object.

1. Introduction

Rendering performance is a key quality requirement in diverse 3D applications such as, computer games and medical imaging. Failure to test and tune performance regularly will accumulate performance issues to an extent where they are difficult to resolve [4]. 3D systems can be useless without an acceptable performance as many other projects [16] became.

The iterative and evolutionary development process in agile methods allows for a frequent and timely treatment of system issues, including functional and non-functional requirements. Little published work [3] describes performance testing in agile environments, but many agile practitioners have realized the importance of conducting early and frequent performance investigations along with continuous functional/acceptance testing [3, 15].

In our opinion, agile performance testing needs to be test-driven and automated. Using story-test driven development (or: executable acceptance test driven development), developers should set up performance test cases at the same time as they set up functional acceptance tests. These tests provide guidance for the development and help ensuring a timely delivery of solutions that fulfill performance requirements [14]. Test automation bears a major advantage: automated performance test cases can be executed repeatedly through continuous integration, iteratively exploring the current status of the system. Thus, breakdowns against system performance criteria will be noted in a timely manner and a continual build up of unresolved problems will be avoided. At the same time, test results will also allow for a reexamination of performance measures to see whether they need to be revised or not for the current iteration.

To carry out effective and efficient agile performance testing, a tool that determines meaningful metrics to fit various system needs is needed. From September 2005 to September 2006, we worked with an agile team that developed a bioinformatics project based on Java3D [12]. During the development, serious performance issues were uncovered and needed to be tested continuously whenever a tuning was done. With no appropriate Java3D performance testing tools available, we were forced to follow a tedious and error-prone manual testing process. This motivated our research. Our aim is developing a tool which executes automated performance tests for Java3D applications and then evaluating the effectiveness of the tool for agile development of Java 3D applications. Other quality attributes of Java3D applications, such as availability and usability, although important, are not focuses of this research.

We carried out a survey to investigate the current state of art in performance testing of 3D applications and collected requirements for tool support. The availability of an automatic 3D performance testing tool makes it possible for a team to continuously check if an application stays "fast enough" while the team incrementally develops more and more functionality following an iterative process.

The paper is organized as follows: Section 2 gives an overview on Java3D and discusses current strategies for performance testing. Section 3 analyzes the survey that collected tool requirements. Section 4 demonstrates

features in J3DPerfUnit by showing real tests. Section 5 discusses the achievements. Section 6 outlines future plans. A conclusion will be drawn in Section 7.

2. Background

2.1. A brief introduction of Java3D

Java3D offers a high-level object oriented structure for developers to manipulate and display objects in a three dimensional virtual world. Everything in Java3D links to a tree structure, called a "scene graph". This tree specifies information about 3D objects, including shapes, locations, orientations, hierarchical relationship and so on [10]. Through manipulating nodes on this tree, one can implement visualization, animation and interaction.

The process to show ("load") 3D object can be divided into three steps. In the first step, an application reads and analyzes the object's geometry data from files or memory, and generates a corresponding BranchGroup [10] which manages the geometry data. In the second step, the Java3D engine places the BranchGroup onto on a tree ("scene graph") branch and makes the 3D object ready for rendering. Lastly, the engine sends the object to a graphic device for rendering. Knowledge of time spent on each step allows locating performance problems easier.

Unloading a 3D object consists of two phases. First, the Java3D engine unloads the BranchGroup representing the object from a scene graph. Second, the engine sends a request asking the graphics device to delete the 3D object.

2.2. Existing tools and strategies

Currently, there is no tool specifically designed for automated performance testing of Java3D applications. The execution time reported by testing frameworks, such as JUnit [8] and JUnitPerf [13], is so general that their test results make it difficult for Java3D developers to figure out where the time is spent. For example, regarding the loading process of a 3D object discussed in Section 2.1, both frameworks can only measure the time for the first half of the first phase. Therefore, for Java3D applications, those frameworks provide very limited help in locating performance problems.

With user intervention, Fraps [7] can report frame rate as 3D objects get rendered. Frame rate [6] usually measured in FPS (frames per second) represents the number of frames produced by a graphics device every second. The higher the frame rate the smoother the rendering is. To get a frame rate with Fraps, a user needs to press hot keys to manually define starting and ending points of capture during the run of an application, and then read the result from the generated reports. The accuracy of these reports is highly dependent on hand-eye coordination. Delays in human reactions usually result in inaccurate testing blocks. Thus, this testing process can be categorized as manual and non-repeatable. Another major limitation is that Fraps is Windows based only. It cannot be used on Linux or Solaris, the operating systems underlying many Java3D applications.

Without a suitable tool in hand, it is not uncommon for developers to build a custom testing program for their 3D applications. However, with limited support available from Java3D, this customization requires a lot of effort and is affected by a number of limitations. Another way to do the performance testing is visually benchmarking and manually interacting with only one or several specific objects. In crunch time, these manual performance tests are then often neglected in the same way as manual regression testing is skipped under time pressure.

In a word, the absence of proper tools can make performance testing of Java3D applications too much effort to be continuously and consistently done.

2.3. The bioinformatics project

Our partner's bioinformatics project is funded by the Governments of Canada and Alberta. The project develops a software tool that allows medical researchers to view a complete image of disease mechanisms in a 3D environment. Examples in the following sections are test code for this project.

3. The survey

We carried out a survey to improve our understanding of the needs for performance testing of Java3D applications. The main results are: the survey responses pointed out that at present only insufficient and incomplete tool support exists for Java3D performance testing (confirming our own investigations into the subject); and the survey provided a set of basic requirements for a 3D performance testing tool.

3.1. Participants

To create wider survey responses, we invited not only the bioinformatics team, but also Master and PhD students in the graphics lab on campus, and developers from several Java3D forums. Java3D comes in two versions; one is based on OpenGL and the other on DirectX, both of which are robust software interfaces for graphics devices [2, 5]. We did not consider DirectX experience as a prerequisite for participation, because the technology can only work on Windows systems. Therefore, DirectX has a narrower application scope and is less popular within the Java3D programming community.

The following table outlines the programming experiences with Java3D and OpenGL among the participants. The number of participants totaled 9.

Table 1. Java3D and OpenGL experience

Java3D	OpenGL	Developers
1 year	3 years	developer 1
3 years	1 year	developer 2
3 years	0.5 year	developer 3
4 years	2 years	developer 4
1 year	0 years	developer 5
1 year	0 years	developer 6
0 years	11 years	developer 7
0 years	13 years	developer 8
0 years	1 year	developer 9

3.2. Test cases

To develop a tool that can effectively measure system performance, understanding the design of corresponding test cases has a significant value. The example test cases given in the survey can be roughly presented as: 1. Apply a scenario on specific test scenes with a required load. For example, load the heart, which has 3000 polygons, into the human body. A test scene, in this case, is the view displaying the human body; 2. Attempt various interactions with the test scene. For instance, rotate the scene; 3. Check specific performance metrics, such as: the frame rate during the scene rotate.

Several words in the examples appeared repeatedly and thus grabbed our attention: "load", "interactions" and "statistics". They provided a clue of what parameters should be used when designing a performance test case. A further interview with the developers disclosed a similarity in the meanings of these words: "load" refers to the dataset size of objects, "interactions" means zoom, rotate or translate a scene or 3D objects, and "statistics" mainly covers execution time and frame rate.

3.3. Metrics

From Table 2, one can clearly see the importance level of each metric in 3D performance testing from the participants' perspective. "Loading time of objects" refers to the time spent from the point when an application starts reading a 3D object data from files or memory to the point when the object is displayed. "Interaction time or lag time" means how long it takes from triggering an interaction, such as: rotate, to observing the final view. These metrics are not limited to the bioinformatics domain as six out of the nine participants came from other 3D projects.

Table 2. Performance metrics

Metric name	Number of affirmative responses
Loading/unloading time of objects	6
Interaction or lag time	6
Frame rate	4
Memory usage	3
Maximum Polygon & Object Counts at X fps (frame per second)	1
Scene graph size	1
Milliseconds/frame	1

3.4. Testing techniques

When asked about performance testing techniques, out of nine participants only two stated that their performance testing was automated. The rest relied on a manual testing process. We were very interested in these two responses and did a follow up interview.

However, the first case ended up with our realization that the developer actually referred to a Java2D testing framework: Jemmy [11]. As their application was a hybrid of Java2D and Java3D, they automated 3D test cases by firing 2D events. A further investigation uncovered that Jemmy was only designed to trigger events on 2D controls to simply examine functional requirements for Swing applications and cannot automate scene interactions in Java3D programs and generate performance statistics.

The second developer mentioned a customized auto-tester, but then gave no further response, thus no details could be obtained. Yet based on the description, J3DPerfUnit subsumes the auto-tester because our framework can test any scenes in a real application without building a "sample-app" and without worrying about the limitation that "the scenes did not change from one test to the next".

4. Performance testing with J3DPerfUnit

4.1. Performance testing with our tool

J3DPerfUnit can evaluate three aspects of Java3D application performance: execution time to build a scene; execution time to load/unload 3D objects to/from a scene graph; and frame rate during user interactions,

Test output is similar to command line messages in JUnit. Detailed statistics can be found in reports under a specified directory. Table 3 gives a brief explanation of the parameters used to define test cases.

Table 3. Test parameters

Parameter	Meaning / Definition
Title of a test window	the title of a window where a test scene rests on
Index of test canvas	test canvas index on the test window (A canvas is a blank rectangular area on a window. One can make drawings or trigger events on it. Multiple canvases can exist on a window, therefore we need to point out which canvas is under testing)
BranchGroup identifier	either the name or the user data of a BranchGroup
Time expectation	maximum time for a successful test
Frame rate expectation	minimum frames produced per second for a successful test
Interaction data	which mouse or keyboard button to press, start and end point for mouse movement, etc.

4.2. Execution time to build a scene

As the soul of a Java3D application, a scene graph must be constructed properly. Otherwise displaying a scene can be extremely slow or simply fail.

The following example investigates the execution time to show a human body. To define such a test, we need two parameters: the title of the test window, which, in our case, is "center" shown on line (2). We use this parameter to locate the scene where a test case should execute on; and the maximum execution time for passing a test which is 2000 milliseconds specified on line (3) for this example.

Concretely, the test code is:

```
(1) TestCase testcase1 =
            j3dTest.getTestCase();
(2) testcase1.testSceneBuild ("center");
(3) testcase1.assertExeTimeEquals(2000);
(4) classRunner.runApplication
             ("WepaMain");
```

The result is displayed on the console:

```
Running 1 test in j3dperftest.WepaTest
Fail (0 tests)
```

This indicates the human body was successfully shown within 2 seconds.

4.3. Execution time to load/unload a 3D object

Loading intricate 3D objects, such as those with a high polygon count or rich textures requires a large data set to be processed, which tends to produce performance issues.

J3DPerfUnit can tell which component needs the most time during the loading process mentioned in Section 2.1: the application or the Java3D engine. As no Java3D APIs is available for capturing any information about graphics card rendering, J3DPerfUnit cannot capture an execution time for the third step in the loading process. However, information about the first two steps can still be useful to locate problems. This is because the information clearly points out where the execution time costs most: the application codes or the Java3D engine. Accurately locating performance problems can help developers work out appropriate solutions. For the third step, J3DPerfUnit provides FPS (frames per second) as a replacement metric for those interaction tests explained in Section 4.5.

The test in this section checks whether loading a skeleton into a human body can be finished in 2 seconds. Screenshots are provided in Figure 1, demonstrating the scenes before and after the skeleton is loaded.

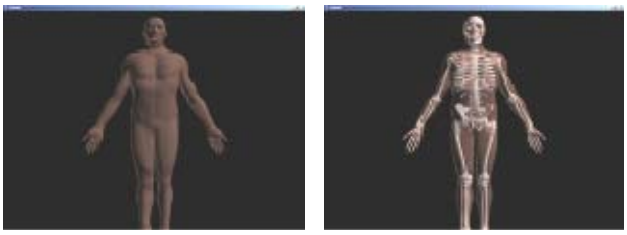

Figure 1. Load a skeleton

For a loading test, two parameters are used to locate a test canvas: the title of the test window and the test canvas index, indicating which canvas on the test window that the loading test will be executed on. Another three parameters used to run such a test are: the name or user data of a BranchGroup to be loaded; an identifier indicating whether we use the name or the user data of the loaded BranchGroup. J3DPerfUnit defines two constants to represent this identifier: BranchIdentifier.BRANCH_NAME and BranchIdentifier.USER_DATA; and a maximum execution time allowed.

This example test failed. Details are printed in a generated report:

```
skeletonTest fails:
     Event -- loading a branch group;
     BranchGroup Name -- skeleton;
     Time Expectation -- 2000ms;
     2000ms was totally spent by the
     application itself to process
     3D data from files or memory.
```

The report shows the execution time is exhausted by the application itself, not the Java3D engine. Therefore, during the performance tuning phase, developers should work on their own code to find out what causes the problem.

Test code, output and reports for unloading a 3D object are very similar to what is described above.

4.4. Frame rate while interacting with a scene

Interaction with a scene involves rotate, translate and zoom with mouse and/or keyboard, all of which have been fully automated by J3DPerfUnit.

Reporting "interaction time" was dropped out because we cannot generate accurate reports for this metric using Java3D as its API does not provide access to this information. Therefore, J3DPerfUnit uses frame rate to evaluate interaction performance. A higher frame rate indicates a smoother interaction and in general 30fps is equivalent to movie performance.

In this section, an automatic mouse scroll will be triggered to zoom in a test scene where a human body with organs represented by a large data set is displayed and examine the frame rate. Other tests, such as automating keyboard and/or mouse buttons to implement interactions with a 3D scene, have similar test code, output and reports.

For tests using a mouse scroll, four parameters need to be specified: a predefined point to position the mouse; symbols representing the mouse wheel; the number of "notches" to move the mouse wheel (negative values indicate movement up/away from the user; positive values indicate movement down/towards the user); and the expected frame rate.

Figure 2 shows two screenshots demonstrating the scenes before and after the body is zoomed in. The test failed. Details are printed in the report like this:

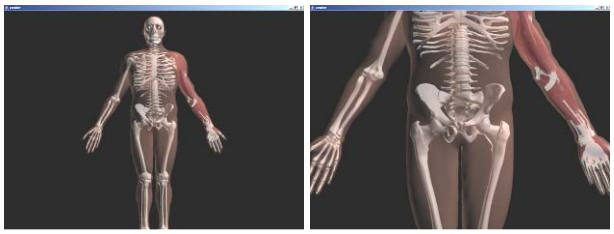

Figure 2. Zoom in a human body with organs

```
zoomInTest failed:
     Event - zoom in the scene;
     Frame Rate Expectation:
            30 frames/second;
     Actual Frame Rate:
            7 frames/second;
     The zoom is not smooth.
```

The message highlights a very low frame rate during the zoom. Carrying organs with complex geometric data, the human body yields a very low interaction performance. As a result, developers need to optimize the implementation that handles the complex geometric data to make the test pass or discuss a change of the performance requirements with their customers.

4.5. Continuous integration

To facilitate continuous integration, J3DPerfUnit offers an Ant task to run performance tests as part of an automated build process. Defining ant tasks for performance test is very similar to specifying those for JUnit tests.

The following demonstrates how we included performance test cases into the build script for our partner's bioinformatics project:

```
<j3dperfunit maxmemory="512M">
     <classpath refid="unittest.class.path" />
     <batchtest todir="${j3dReports.dir}">
            <fileset dir="${src.dir}">
                   <include
                   name="**/tests/gui/*J3DPer
                   fUnit*.java"/>
            </fileset>
     </batchtest>
</j3dperfunit>
```

The test output and detailed test reports are similar as shown in the previous sections.

5. Discussion

Existing tools and strategies cannot provide an efficient and effective way to conduct automated performance testing on Java3D applications. Thus, one core requirement – application performance – can not be addressed as part of agile development for these kinds of applications. Our research has made three major contributions to improve this situation. Firstly, we provide tool support to conduct automated performance testing for Java3D applications so that Java3D performance testing can be continuously conducted as regression testing and be efficiently reexamined whenever a tuning is done. Secondly, we provide a customized Ant task so that our tool can be easily integrated into a continuous build process. Thirdly, we can specify performance criteria in test cases, thus allowing agile developers to use test-driven development covering performance requirements.

An initial evaluation of our tool using our partner's bioinformatics project and samples in Java3D tutorials as examples indicated that the tool was effective in detecting performance problems and helpful in identifying whether the problems come from an application itself or the Java3D engine during the 3D-object loading/unloading process

6. Future Work

6.1. Further evaluation

We have verified the fitness for purposes of J3DPerfUnit by applying them to evaluate some Java3D tutorials and the system from our partner's bioinformatics project. In initial discussions, our partner considered the automated performance testing useful. To extensively verify the tool's usefulness and usability, we will invite other Java 3D developers to use J3DPerfUnit for their performance testing. The study participants will fill out questionnaires regarding tool evaluation and ways of improvement. The experiments will be conducted in May-June 2007.

6.2. Tool enhancement

System load testing is a vital part of performance testing which indicates a system's highest capacity. This kind of testing is popularly used for tuning web application performance [1, 9]. Likewise, the ability to process a large number of objects is crucial in 3D applications. Games, for example, require smooth translation, rotation or zoom over all the objects on a scene. We plan to advance current loading/uploading features in J3DPerfUnit to make system load testing easier.

7. Conclusions

In this paper, we introduced a new tool, J3DPerfUnit that can effectively automate performance testing of Java3D applications and support test-driven development as well a continuous integration incorporating performance requirements. We conducted a survey to gather performance metrics beneficial for 3D developers, and to collect requirements for our tool development. An initial tool evaluation was conducted using a Java 3D application from one of our partners and Java3D tutorials. A more detailed, formal and extensive evaluation planned for Summer 2007 will provide guidelines for our further development, and insights into performance testing for Java3D applications developed using agile methods.

8. References

[1] B. M. Subraya, *Integrated Approach to Web Performance Testing: A Practitioner's Guide*, IRM Press, Hershey, PA, USA and London, UK. 2006.

[2] Bradley Bargen, Terence Peter Donnelly, *Inside Directx (Microsoft Programming Series),* Microsoft Pr, Redmond, Washington, 1998.

[3] Chih-Wei Ho, Michael J. Johnson, Laurie Williams, and E. Michael Maximilien, "On Agile Performance Requirements Specification and Testing", *AGILE 2006 Conference,* 23-28 July 2006, on page(s): 6 pp.-.

[4] Chung, L., B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2000.

[5] Dave Shreiner, Opengl Architecture Review Board, *OpenGL(R) Reference Manual: The Official Reference Document to OpenGL, Version 1.2 (3rd Edition),* Addison-Wesley Professional, 1999.

[6] Frame Rate, Online: http://en.wikipedia.org/wiki/Frame_rate, visited Dec 22, 2006.

[7] Fraps: Overview, Online: http://www.fraps.com/, visited Dec 20, 2006.

[8] Gunjan Doshi, "JUnit 4.0 in 10 minutes", Online: http://www.instrumentalservices.com/content/view/45/52/#_Timing_out_a_test, visited Dec 20, 2006.

[9] Hung Q. Nguyen, *Testing Applications on the Web: Test Planning for Internet-Based System*, Wiley, 2000.

[10] Java3D: Java3D API Tutorial, Online: http://java.sun.com/developer/onlineTraining/java3d/, visited Dec 22, 2006.

[11] Jemmy: Overview, Online: http://jemmy.netbeans.org/, visited Jan 1, 2007.

[12] Jim X. Chen, Edward J. Wegman, *Foundations of 3D Graphics Programming: Using JOGL and Java3D*, Springer, London, 2006.

[13] JUnitPerf, Online: http://www.clarkware.com/software/JUnitPerf.html, visited Jan 1, 2007

[14] Lloyd G. Williams, Connie U. Smith, "PASASM: a method for the performance assessment of software architectures", Workshop on Software and Performance, Proceedings of the 3rd international workshop on Software and performance, 2002, p179-189.

[15] Neill Mccarthy, "Agile Performance Testing", Online: http://www.testingreflections.com/node/view/2186, visited Dec 20, 2006.

[16] Sommerville, I., *Software Engineering 7th Edition*, Addison-Wesley, Boston, MA, 2004.