Platform Perils

Test Report

Steven Palmer
⟨palmes4⟩
Chao Ye
⟨yec6⟩

April 26, 2016

Contents

List of Tables

List of Figures

1 Introduction

This document provides a report of the results of the testing performed on the Platform Perils application. Both system testing and quality testing are covered. Traceability between testing and both requirements and modules is given in the final section.

2 System Testing

****NOTE 1:** Many of the automated tests described in the test plan will not be implemented in this version of the game. The test plan will be modified for revision 1 submission to achieve consistency. **

****NOTE 2:** Additional automated testing is planned for revision 1.**

2.1 Automated Testing

A suite of automated unit tests has been created to test the game for basic functionality. The unit tests cover input response as well as collision physics. These tests are useful to ensure that modifications to the game code do not break the fundamentals of the game. A description of the unit tests that were carried out and their results are given in the remainder of this section.

Revision History

Date	**Version**	**Notes**
March 24, 2016	1.0	Created document skeleton
March 28, 2016	1.1	Final version for rev 0

Test 2.1.1:	**Walk left, started from stationary**
Description:	Tests if the hero walks left when the corresponding input is received when the hero is initially stationary
Type:	Unit Test (dynamic, automated)
Initial State:	Custom in-game state with a hero object having x-velocity of zero
Input:	Keyboard function called with simulated left key down stroke
Output:	Hero object x-velocity
Expected:	Hero object x-velocity is less than zero
Result:	PASS

Test 2.1.2:	**Walk right, started from stationary**
Description:	Tests if the hero walks right when the corresponding input is received when the hero is initially stationary
Type:	Unit Test (dynamic, automated)
Initial State:	Custom in-game state with a hero object having x-velocity of zero
Input:	Keyboard function called with simulated right key down stroke
Output:	Hero object x-velocity
Expected:	Hero object x-velocity is greater than zero
Result:	PASS

Test 2.1.3:	**Supported by platform**
Description:	Tests if the hero is supported when standing on a platform
Type:	Unit Test (dynamic, automated)
Initial State:	Custom in-game state with a hero object having zero velocity directly above a platform
Input:	The game is stepped forward in time
Output:	Hero object y-position
Expected:	Hero object y-position is unchanged
Result:	PASS

Test 2.1.4:	**Obstructed by wall, approaching from left side**
Description:	Tests if the hero is obstructed by a wall when approaching from the left side of the wall
Type:	Unit Test (dynamic, automated)
Initial State:	Custom in-game state with a hero object on a platform with a wall directly to the right
Input:	Keyboard function called with simulated right key down stroke and the game is stepped forward in time
Output:	Hero object x-position
Expected:	Hero object x-position is unchanged
Result:	PASS

Test 2.1.5:	**Obstructed by wall, approaching from right side**
Description:	Tests if the hero is obstructed by a wall when approaching from the right side of the wall
Type:	Unit Test (dynamic, automated)
Initial State:	Custom in-game state with a hero object on a platform with a wall directly to the left
Input:	Keyboard function called with simulated left key down stroke and the game is stepped forward in time
Output:	Hero object x-position
Expected:	Hero object x-position is unchanged
Result:	PASS

2.2 Manual Testing

Manual testing was the main method used to ensure that the game functions as intended. A summary of the results of the manual testing are given in Table 1.

[What kind of tests did you do for gameplay, sound, etc.? You should describe your methods. —DS]

Table 1: Manual test results

Test category	**Result**
Gameplay	PASS
Sound	PASS
Graphics	Texture problem on arch needs fixing.

3 Quality Testing

3.1 Usability Testing

A user experience survey will be used to assess useability. This survey has not yet been administered but is expected to occur in the near future.

3.2 Performance Testing

Stress testing was used to evaluate performance. In these tests, different types of game objects were continuously introduced to a stage while measuring changes in framerate. A high performance and low performance system were used in these tests. The specifications of each system is given in Table 2.

Table 2: Systems used in performance testing

System	Hardware
High performance	i7 4770K @ 4.4 GHz AMD Radeon HD 7970
Low performance	i5 2430M @ 2.4 GHz nVidia GT 540M

The objects used by the game can be broken down into two main categories: dynamic and static. Dynamic objects are free moving and subject to all physics calculations, while static objects do not move and are only involved in collision calculations. All dynamic objects used in the game are essentially the same: they consist of a loaded mesh, textures, and a single shader. Most static objects also fit this description, with one notable exception: platform objects were implemented using 3 separate shaders.

Plots showing the variation in framerate with respect to the number of objects present in a stage are given in Figure 1 (dynamic) and Figure 2 (static). The results show that a large number of objects can be incorporated into a stage before performance begins to be affected: the high performance system maintained a framerate of 60 fps until roughly 1500 dynamic objects or 3000 static objects were introduced. Even the low performance system was able to maintain a framerate above 60 until around 300 dynamics objects. This means that the game will likely be able to run at 60 fps on most systems since the number of objects in a single stage is unlikely to approach 300.

A plot showing the variation in framerate with respect to the number of platforms was also produced and is shown in Figure 3. The results of this test show a drastic drop in framerate with very few objects, and this framerate drop appears to be consistent between the high end and low end systems. This suggests that the problem is in using multiple shaders: switching shaders is a resource intensive procedure and the consistencies in framerate between the two systems can be explained by the fact that the amount of time required for switching a shader would be approximately constant between the two systems.

Additional testing was performed to assess whether the framerate was predominantly affected by procedures involving physics calculations or rendering. The results of these tests are given in Figure 4 and Figure 5 for the high and low performance systems, respectively. Interestingly, from the plot of the high performance system it appears that neither is a dominant factor: both make a roughly equal contribution to the overall framerate. This does not appear to be the case in the low performance system, where it is clear that the rendering step limits the framerate.

3.3 Robustness Testing

Robustness of the application with regard to erroneous input was not formally tested. User inputs are only accepted as defined keystrokes and mouse clicks. The way these inputs map to setting/modifying variables is entirely controlled by the game code and thus the user is never able to directly change any variables via inputs. This means that all variables are maintained within their expected ranges and no explicit testing is required.

Robustness tests in the form of stress testing were covered under performance testing.

4 Summary of Changes

A summary of the changes made in response to testing are given in Table 3.

** **Note:** Changes have not yet been made. **

Table 3: Changes made in response to testing

System	Hardware

5 Traceability

5.1 Trace of Testing to Requirements

A trace between requirements and testing is given in Table 4.

** **NOTE:** Requirements need updating. This will be done before revision 1. **

Table 4: Trace between requirements and tests

Requirement	Test(s)

5.2 Trace of Testing to Modules

A trace between modules and testing is given in Table 5.

** **NOTE:** The design document needs updating. This will be done before revision 1. **

Table 5: Trace between modules and tests

Module	Test(s)

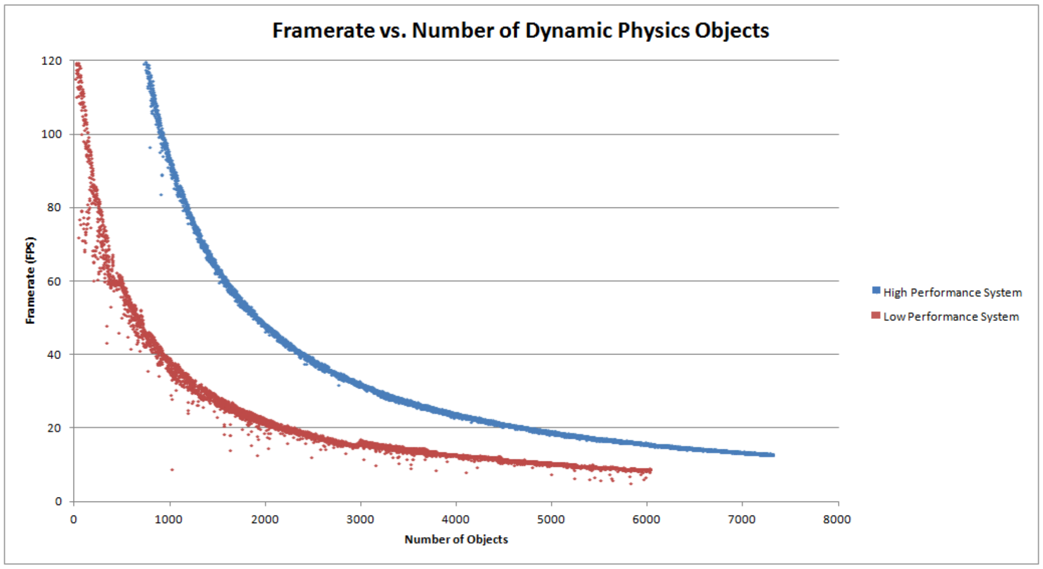

Figure 1: Framerate vs. number of dynamic objects

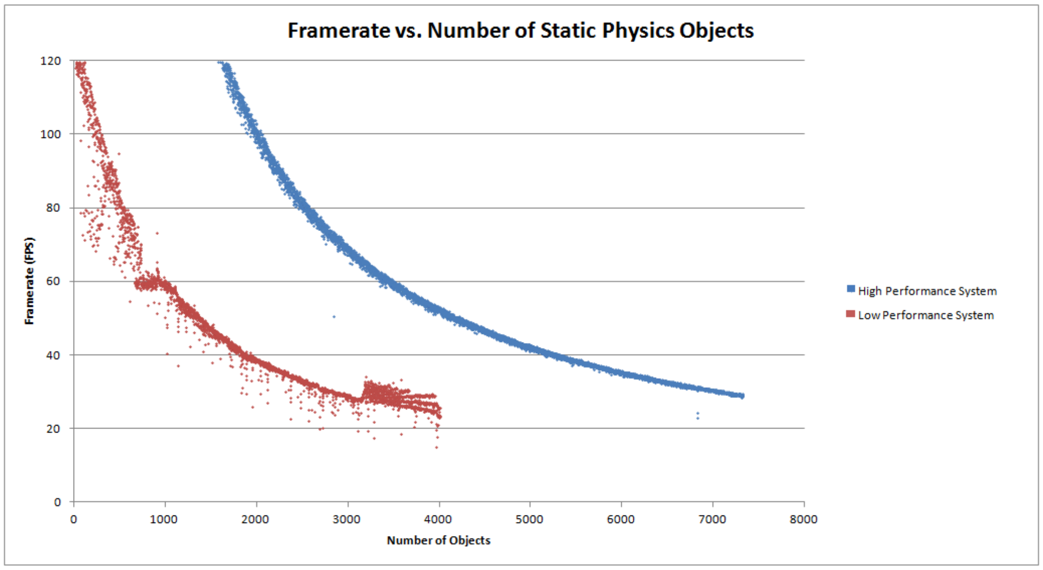

Figure 2: Framerate vs. number of static objects

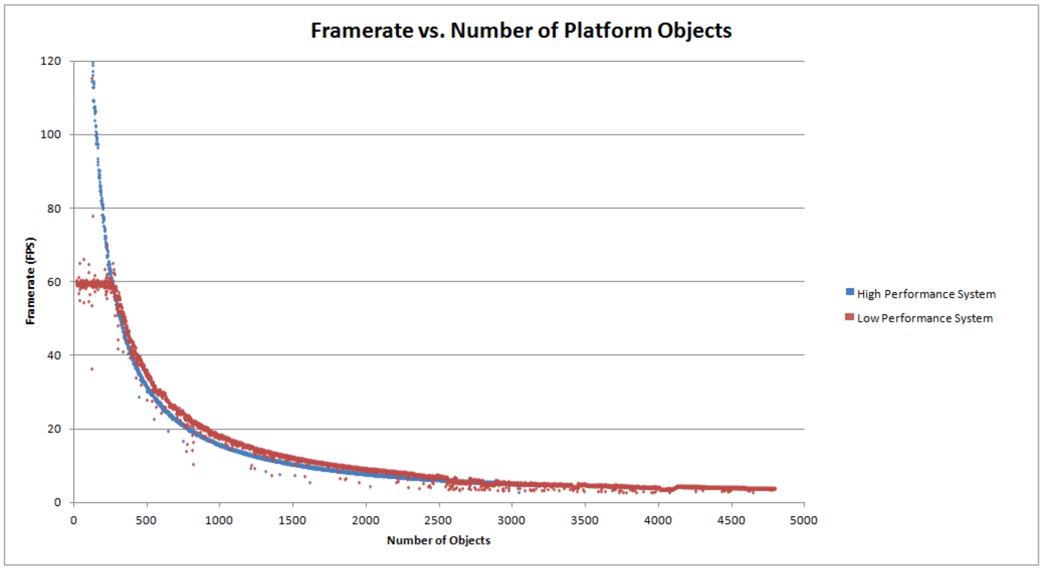

Figure 3: Framerate vs. number of platform objects

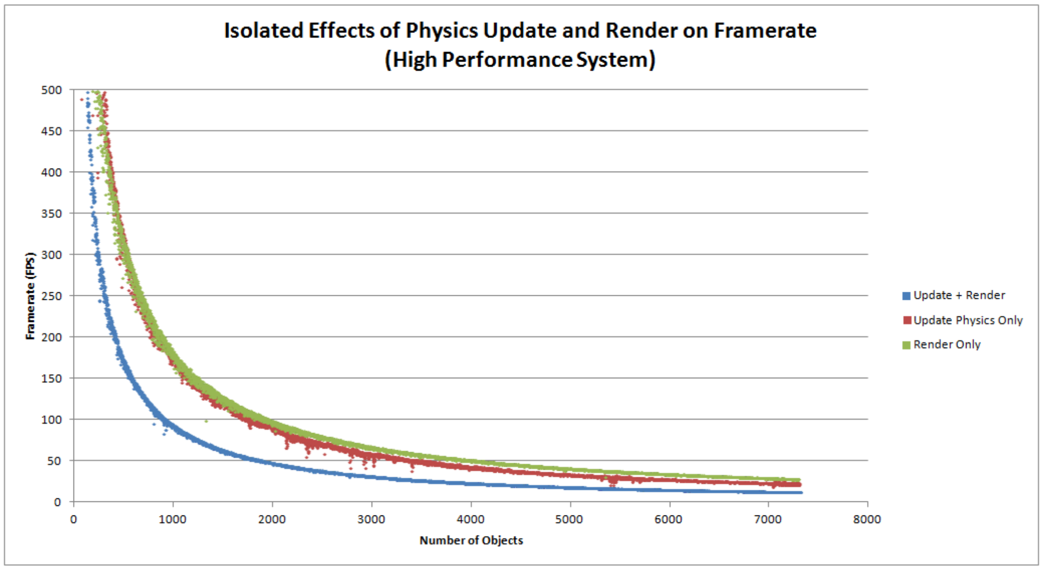

Figure 4: Effects of physics update and render on framerate on high performance system

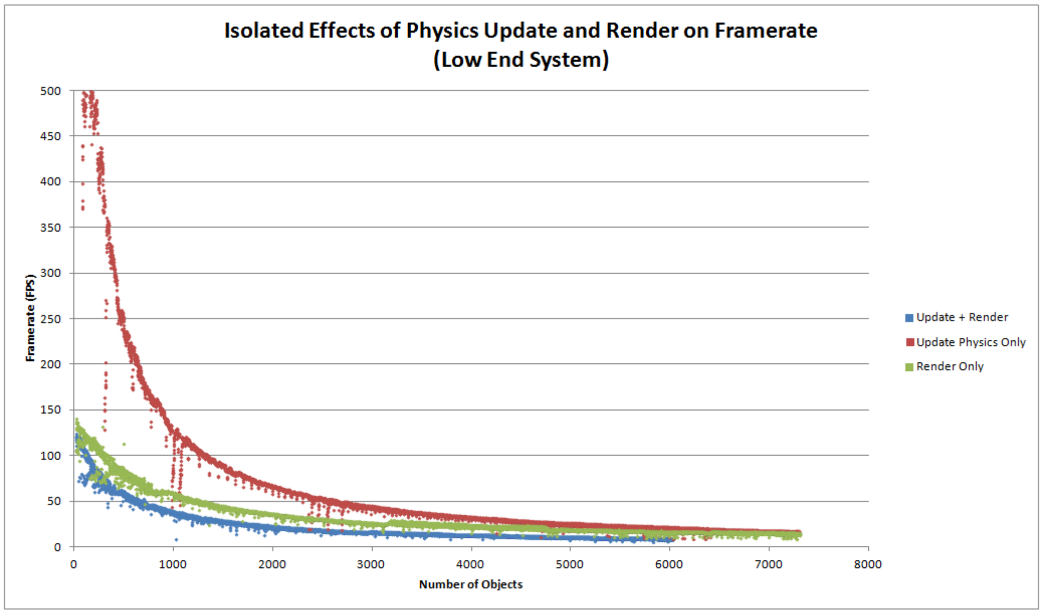

Figure 5: Effects of physics update and render on framerate on low performance system