Comparison in Performance of Parallel Deformable Object Simulation between OpenGL and Unity

Min-Hyung Choi[1], Min Sang Kim[2], Nak-Jun Sung[2], Yoo-Joo Choi[3] and Min Hong[4]
[1]Department of Computer Science and Engineering, University of Colorado Denver
Colorado, USA
[2]Department of Computer Science, Soonchunhyang University
Asan, South Korea
[3]Deparment of Newmedia, Seoul Media Institute of Technology
Seoul, South Korea
[4]Deparment of Computer Software Engineering, Soonchunhyang University
Asan, South Korea
[e-mail: Min.Choi@ucdenver.edu, ben399399@gmail.com, njsung1217@gmail.com,
yjchoi@smit.ac.kr, mhong@sch.ac.kr]
*Corresponding author: Min Hong

Abstract

Recently, as the performance of mobile processors has increased, it has become possible to run complex computational tasks such as deformable object simulation, which were only possible to simulate for PC in past. Due to these changes, digital contents, including computer graphics, need to be driven not only on smart phones or PCs, but also on a variety of portable device platforms. In this paper, we provided a performance comparison between Unity which is the market-dominant multi-platform digital content production tool, and OpenGL which is cross-platform application programming interface for rendering computer graphics with real-time deformable object simulation using GPU. We believe that this paper provides a guideline for game or smartphone application developers to decide better platforms for creating digital contents with complex deformable object simulation in various situations.

Keywords: Unity, OpenGL, Performance Comparison, Deformable Object Simulation, Multiplatform

1. Introduction

As the performance of mobile processors on smart phones and tablets and other handheld devices has improved, it is possible to run more computationally heavy cost programs on smartphones that have been only able to be performed on computers [1]. Because of these changes, digital contents produced using computer graphics require that they should be run on smart phone, PC, as well as various portable device platforms. One way to create content with complex graphics operations on a variety of platforms is to use a digital content creation tool that supports multiple platforms, such as Unity, or to write code directly using the C ++ language, the OpenGL API, and the GLSL language. Since these contents require many computer operations, it is necessary to perform operations that can be processed in parallel using the GPU in order to compute them in real time.

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(Ministry of Science, ICT & Future Planning) (No. 2017R1A2B1005207).

In this paper, we propose a better method to perform complicated computation using GPU on various platforms in real-time by comparing mass-spring based deformable simulation which requires complex computation in OpenGL and Unity environment.

2. Deformable Object Simulation

In this paper, we used physically-based deformable object simulations to compare the performance of OpenGL and Unity. The real-time deformable object simulation is one of common computational simulations usually for physics or medical areas, to represent deforming objects interacting with external forces such as wind, gravity, and force inputs from users [2, 3]. Simulating deforming objects in precise ways as finite element method (FEM) requires a really expensive computational cost which is not able to present in real-time [4]. Thus, in order to present deformable simulation in real-time, we used the simple mass-spring system to implement deformable objects.

3. Implementation of Parallel Object Simulation in Unity

3.1 Construction of Compute Buffer

The proposed deformable simulation, GPU calculation with compute shader was used for updating information for each edges and nodes. Compute shader is a program that runs on a graphics card, outside of the normal rendering pipeline [5]. Compute shaders enable to perform general-purpose GPU (GPGPU) computing, solving problems which can be parallelized much more efficiently than handling with a single CPU. Compute shaders in Unity are written in High-Level Shader Language (HLSL) and OpenGL uses OpenGL Shader Language (GLSL) [6].

In order to read or write data such as nodes of mass position from GPU in Unity, a compute buffer is needed [7]. Compute buffers carry data across compute shaders and rendering pipeline. **Table 1** shows the compute buffers used in the proposed deformable simulation.

Table. 1. Information of compute buffers

Name	Type	Size
nodePositionData	float4	Number of nodes
nodeVelocityData	float4	Number of nodes
springData	Spring (4 * int , float)	6 * amounts of tetrahedrones
forceData	float array	Number of the maximum adjacent mass point among them

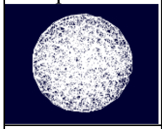
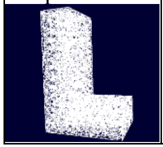
3.2 Scripts and Shaders in Unity

Scripts used for deformable object simulation in Unity are written in C# language. These scripts are used for invoking compute shaders and rendering shaders. The computing script can be divided with two parts. At the preprocessing step, script parses the data files exported from TetGen. After parsing the data, script computes the maximum number of the adjacent mass points among them, which is for generating force buffer. After these steps, script creates compute buffers and passes the parsed data. After preprocessing step is done, computing script passes compute buffer to rendering script for render deformable objects. We rendered computed deformable objects with wireframe rendering to visualize springs inside the deforming objects.

4. Experiment Results

Table 2 shows the deformable objects were used to compare the performance of each platforms. The sizes of models were varied for simulating various conditions.

Table 2. Information of models for experimental tests

Model	Nodes	Faces	Tet.	Springs
Sphere 	732	7,926	3,834	23,004
Alphabet L 	1,642	2,380	6,061	36,366

Alphabet A	2,320	3,724	7,867	47,202

Each simulation was performed 20 times and performances of experimental tests were measured by averaging the elapsed time of 200 frames. **Table 3** is the measured elapsed time per frame and frames per seconds for simulation by each platforms and models.

Table 3. The experimental results with elapsed time per frames and frame per second for each platform and models

Models	Platform	Elapsed time(ms)	Frames per second
Sphere	Unity	1.610	621.12
	OpenGL	0.161	6,207.36
Alphabet L	Unity	2.405	415.75
	OpenGL	0.179	5,583.94
Alphabet A	Unity	3.000	333.31
	OpenGL	0.200	4,990.97

The result shows that simulation by pure OpenGL and GLSL are 12.79 times faster than implementation with Unity.

5. Discussion about Performance

To clearly explain these huge gaps of performance difference between OpenGL and Unity, we analyzed the processes simulation using Unity with the Unity profiler. Since Unity is mainly using for creating gaming, additional processes such as GUI updates or AudioManagers were running on background. However, the simulation implemented on OpenGL doesn't require those features, simulation performed on Unity platform can be much slower than simulation performed on pure OpenGL.

6. Conclusion

In this paper, we provided the performance comparison between OpenGL and Unity platform with complex deformable object simulations. Both environments are widely used in industries for creating graphical digital contents for targeting multi-platforms. The result of our experiment shows that pure OpenGL and shaders written in GLSL were around 12 times faster than performed in Unity since Unity is not just computing simulations but also providing convenient development environment for users. These results may help a decision for choosing better platforms for digital contents containing complex math calculations also targeting devices with various performance.

References

[1] Anand Lal Shimpi, "The ARM Diaries, Part 2: Understanding the Cortex A12", [INTERNET]https://www.anandtech.com/show/7126/the-arm-diaries-part-2-understanding-the-cortex-a12, ANANDTECH, 2013

[2] Meier, U., O. López, C. Monserrat, M. C. Juan, and M. Alcañiz, "Real-Time Deformable Models for Surgery Simulation: A Survey.", *Computer Methods and Programs in Biomedicine*, 2005

[3] Servin, Martin, Claude Lacoursière, N Melin, C Lacoursiere, and N Melin. "Interactive Simulation of Elastic Deformable Materials", *Proceedings of SIGRAD Conference*, no. December: 22–32, 2006.

[4] Hammer, Peter E., et al. "Mass-spring vs. finite element models of anisotropic heart valves: speed and accuracy," *In: ASME 2010 Summer Bioengineering Conference. American Society of Mechanical Engineers*, p. 643-644, 2010.

[5] Shreiner, Dave, et al. "OpenGL programming guide: The Official guide to learning OpenGL, version 4.3." Addison-Wesley, 2013.

[6] Unity Documentation "Compute Shader", [INTERNET] https://docs.unity3d.com/Manual/ComputeShaders.html

[7] Unity Documentation "Compute Buffer", [INTERNET] https://docs.unity3d.com/ScriptReference/ComputeBuffer.html