ICT, Electronics, Programming

UDC 004.021

IMPROVING THE PERFORMANCE OF UNITY 3D MOBILE GAMES

ALEXEY KARPOVICH, DMITRY PYATKIN
Polotsk State University, Belarus

This article presents ways to create high-performance applications on Unity 3d. Describedways to improve the performance of applications for mobile platforms and architectural design patterns that are most often used when programming mobile games.

Unity3d [1] is one of the most popular game engines for mobile platforms. Many developers use it to create and release games. When developing games for mobile devices, it is important to consider their limited technical capabilities. There are weak and powerful phones in performance. New generations of mobile GPUs can be 5 times more productive than their predecessors. Despite this, it is important to optimize the application on time in order to achieve maximum performance on most devices and reach a greater number of users. Also of great importance is the architecture of the gaming application. Properly chosen architecture allows us to greatly simplify and speed up the process of developing a game, as well as improve the overall performance of the application as a whole.

To create high-performance applications that will work equally well on any device, good optimization of the program code and graphics rendering is necessary.

Optimization [2] - modification of the system to improve its effectiveness.

Optimization requires finding a bottleneck: a critical part of the code, which is the main consumer of the necessary resource. Improving about 20% of the code sometimes entails a change in 80% of the results, according to the Pareto principle [3]. A resource leak (memory, handles, etc.) can also lead to a drop in the speed of program execution.

However, optimizing code from the very beginning of game development can be a pretty bad idea. Premature code optimization can cause many problems and slow down development. Since the optimized code is less flexible and harder to read. In addition, it is most likely that most of the optimizations that you are going to do at the initial stage of the project will most likely not affect the final project as a whole. Therefore, the best solution is to optimize the application only if necessary at the final stage of game development.

However, there are a couple of exceptions to this rule, which are especially important when developing for mobile devices, because this rule is more suitable for PC projects and they are less limited in resources. Optimization can be carried out immediately:

– If there is a code, or a construction that is already known to write better, with less memory / processor cost, and so on, and this will not damage the overall perception of the program code.

– If there is an action that will be repeated many times, and from the very beginning it can be optimized. Then everything will go automatically, and you will not need to correct the same thing several times.

Before you begin to optimize the code, you need to determine what exactly needs to be optimized. In Unity there is a convenient tool for finding places needing optimization - profiler [4].

A profiler is a tool that allows a programmer to see how much time a program takes to perform each function and rank them in order. When the top function takes 3% of the time, this means that if you can halve the time for its execution, the overall program performance will accelerate by 1.5%.

Optimization mainly focuses on single or repeated runtime, memory usage, disk space, bandwidth, or some other resource.

Most of all, the performance of the game is affected by graphics rendering, so you should pay more attention to it. Usually the best way to increase graphics rendering performance is to reduce the number of draw calls [5].

Draw-call is a drawing API command (for example, OpenGL or Direct3D) for drawing. The graphics API does significant work for each draw-call, which greatly affects the CPU performance.

For mobile devices it is recommended to have up to 100 draw-calls, for older devices it is better that the number of draw-calls does not exceed 40. There are many ways to reduce their number:

– Use batching [6] for drawing objects of the same type using the same material for one approach.
– Use atlases to combine multiple textures into one large one.
– Try to ensure that objects with different materials do not overlap each other.
– Do not change Transform->Scale

– Try to use fewer particle systems. Each particle system gives 1 Draw Call

Another factor affecting the performance is the so-called performance jumps. Performance jumps are short-term game hangs. Here is a list of rules for reducing performance jumps:

– Try not to use Instantiate (), especially for complex objects.

– Minimize the number of calls Destroy (), gameObject.SetActiveRecursively (), Object.Find (), Resources.UnloadUnusedAssets () and GC.Collect () functions, as they require a large amount of resources.

Unity supports several types of audio; by default, it will import audio clips for use with the Decompress On Load [7] download type along with Vorbis compression [8]. Sound effects are usually short and, therefore, have small memory requirements. For them, the setting of Decompress on Load will work best, but the type of compression should be either PCM or ADPCM [9]. PCM provides higher quality, but comes with a large file size, which is great for a very short but important sound effect. ADPCM has a compression ratio of 3.5 times less than PCM, and is best used for audio effects that are used very often.

Were considered the most common ways to increase productivity, but also to simplify the development of the game and improve performance, it may be important to choose the right game architecture.

The architecture of a computer game [10] is a system for organizing a program that defines the internal logic of building code, the choice of structural elements and the definition of connections between them.

A complex gaming application system consists of several subsystems - functional modules, services, layers, subroutines, connected in a specific sequence. With this functional partitioning, developers get not very connected code, but a set of clear elements that interact according to simple rules.

It is important in the development process to follow the signs of a good architecture:

– It is easy to make edits. New fragments do not require rewriting existing ones.

– The system is effective. The code solves the tasks and works in any conditions.

– Development time can be reduced by increasing the team. Tasks are easily shared between developers.

With an incorrectly written architecture, the game will turn out to be poorly scalable and inefficient in terms of performance.

One of the simplest architectural patterns used in games, as well as one of the oldest, is considered the Game Loop [11]. This template is used in almost every game.

Game Loop (Game Loop) - this is a general flow control for the entire game program. Each iteration of the game loop is known as a frame. Most real-time games are updated several times per second: 30 and 60 are the two most common intervals. If the game runs at 60 frames per second, then the game loop completes 60 iterations every second.

Game Loop works throughout the game. At each iteration, the game loop processes user input without blocking, updates the state of the game, and renders the game.

Another frequently encountered architectural solution in game programming is the Model-View-Controller (MVC) [12]. It is often used in applications with a graphical user interface (GUI) [13]. MVC divides the program into Controllers, Views and Models:

– The model provides data and responds to controller commands, changing its state.

– The view is responsible for displaying model data to the user, responding to changes in the model.

– The controller interprets the user's actions, notifying the model of the need for changes.

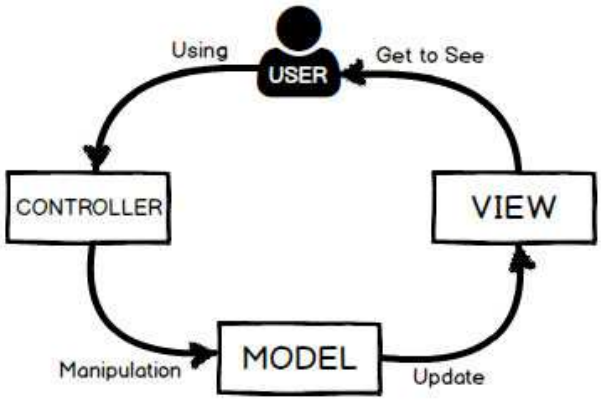

Figure 1. – MVC Chart

The main purpose of the application of this concept is in the separation of business logic (model) from its visualization (presentation, type). Due to this separation increases the possibility of code reuse. Also this separation allows a programmer who develops business logic to work independently of theGUI developers.

However, MVC is usually not fully utilized. The role of the Controller in controlling the flow of data can be assumed by the game cycle and the set of Systems, and the number of different Components is much larger and is not described by simply dividing into Model and View. In addition, many components are optional. In spite of all this, MVC pattern is the most popular among mobile application developers, simple and efficient applications are created with its help.

This article has reviewed the main ways to develop high-performance mobile games. Optimization recommendations were given to help improve application performance. Architectural design patterns of games optimal for the development of mobile applications were also considered.

REFERENCES

1. Unity (игровойдвижок) [Электронныйресурс] / Wikipedia – TheFreeEncyclopedia. – Режим доступа: https://ru.wikipedia.org/wiki/Unity_(игровой_движок). – Дата доступа: 14.02.2019.
2. Оптимизация (информатика)[Электронный ресурс] / Wikipedia – TheFreeEncyclopedia. – Режим доступа: https://ru.wikipedia.org/wiki/Оптимизация_(информатика). – Дата доступа: 14.02.2019.
3. Закон Парето или Принцип 80 на 20 [Электронный ресурс] / Элитариум – центр дополнительного образования. – Режим доступа: http://www.elitarium.ru/zakon-pareto-princip-80-na-20-pravilo-jurana-kachestvo-resursy-raspredelenie-rabota-vazhnost/. – Дата доступа: 14.02.2019.
4. TheProfilerWindow [Электронныйресурс] / DocsUnity3d. – Режим доступа: https://docs.unity3d.com/ru/current/Manual/Profiler.html. – Дата доступа: 14.02.2019.
5. Руководство по оптимизации [Электронный ресурс] / DocsUnity3d. – Режим доступа: https://docs.unity3d.com/ru/current/Manual/MobileOptimizationPracticalGuide.html. – Дата доступа: 14.02.2019.
6. Батчинг вызовов отрисовки (DrawCallBatching) [Электронный ресурс] / DocsUnity3d. – Режим доступа: https://docs.unity3d.com/ru/current/Manual/DrawCallBatching.html. – Дата доступа: 15.02.2019.
7. Sprite Atlas [Электронныйресурс] / DocsUnity3d. – Режим доступа: https://docs.unity3d.com/Manual/SpriteAtlas.html. – Дата доступа: 15.02.2019.
8. AudioClip [Электронный ресурс] / DocsUnity3d. Режим доступа: https://docs.unity3d.com/Manual/class-AudioClip.html. – Дата доступа: 16.2.2019.
9. Vorbis [Электронный ресурс] / DocsUnity3d. – Режим доступа: https://docs.unity3d.com/ScriptReference/AudioCompressionFormat.Vorbis.html. – Дата доступа: 16.02.2019.
10. Audio compressionformat [Электронныйресурс] / DocsUnity3d. – Режим доступа: https://docs.unity3d.com/ScriptReference/AudioCompressionFormat.html. – Дата доступа: 16.02.2019.
11. Архитектура, производительность и игры [Электронный ресурс]/ GameProgramming Patterns. Режим доступа: https://martalex.gitbooks.io/gameprogrammingpatterns/content/chapter-1/1.1-architecture-performance-and-games.html. – Дата доступа: 16.02.2019.
12. Игровой цикл [Электронный ресурс] /GameProgramming Patterns. – Режим доступа: https://martalex.gitbooks.io/gameprogrammingpatterns/content/chapter-3/3.2-game-loop.html. – Дата доступа: 18.2.19.
13. Model-View-Controller [Электронныйресурс] / Wikipedia – TheFreeEncyclopedia. Режим доступа: https://ru.wikipedia.org/wiki/Model-View-Controller. – Дата доступа: 18.02.2019.