УДК 004.94

OPTIMIZED GAME-OBJECTS UPDATE SYSTEM FOR VIDEO GAMES

DOI 10.36994/2788-5518-2021-01-01-12[12]

***Dmitriy Markov**, NTUU "Igor Sikorsky KPI", Kiev, Ukraine, markov.dmitriy.dev@gmail.com*
***Walery Rogoza**, Dr.habil. Prof. NTUU "Igor Sikorsky KPI", Kiev, Ukraine,, rosvetnik@gmail.com*

***Abstract**. With the growing sophistication of computer games, real-time events and simulations are virtually a standard in today's game architectures. What is needed is a way to manage and execute multiple events per frame, or many times within a frames time-step. A scheduler can manage update of the objects in a very flexible fashion, as well as facilitate a modular approach for extensibility. Given paper describes a way to optimize objects update using specially developed frame pipeline, internal optimizations inside objects and scheduling of updates. In the given system tasks would run only on some frames. Frame consists of several stages such as Serial, MT and End of the frame. In this work we have described all pros and cons of the mentioned parts of the frame. Also work presents scheduler technology which allows objects to be updated with gaps but still be responsive and with minimal visual or logic lags.*

***Keywords**: CPU, optimization, multithreading, video games, objects update system*

ОПТИМІЗОВАНА СИСТЕМА АПДЕЙТУ ГЕЙМ ОБ'ЄКТІВ ДЛЯ ВІДЕО ІГОР

***Марков Д. К.**, НТУУ "КПІ імені Ігоря Сікорського", Київ, Україна, markov.dmitriy.dev@gmail.com*
***Рогоза В. С.**, Д.т.нг.,проф. НТУУ "КПІ імені Ігоря Сікорського", м. Київ, Україна rosvetnik@gmail.com*

***Анотація**. Управління подіями в грі, такими як зміни анімації і зіткнення об'єктів, може бути складним завданням, якщо ви не розумієте, як вони структуровані і реалізовані. Ця стаття покаже вам, як планувальник допоможе вашій ігровій системі стати більш організованою і гнучкою. Фізичне моделювання, анімація персонажів, виявлення зіткнень, ігровий штучний інтелект і рендеринг - це лише кілька прикладів ігрових технологій, які можуть отримати вигоду з використання планувальника. Час - важливий фактор в комп'ютерних іграх. При зміні сотень окремих об'єктів і процесів в різні проміжки часу багато хто з цих технологій моделювання можуть стати надзвичайно складними. Наприклад, симуляція фізики розбиває час на невеликі дискретні інтервали для кожного об'єкта, щоб оновити рух об'єктів. Моделювання мало б набагато більш високу ступінь точності, якби дозвіл за часом було більш точним. Оскільки в цій ситуації кілька об'єктів і тимчасових інтервалів обробляються одним і тим же кодом планування, надійність має вирішальне значення для запобігання вузьких місць при плануванні. З*

[12] Dmitriy Markov, Walery Rogoza

ростом складності комп'ютерних ігор події і симуляції в реальному часі стали практично стандартом в сучасних ігрових архітектурах. Що необхідно, так це спосіб управляти і виконувати кілька подій за кадр або багато разів в межах тимчасового кроку кадру. Планувальник може дуже гнучко управляти оновленням об'єктів, а також сприяти модульному підходу до розширюваності. У даній статті описаний спосіб оптимізації відновлення об'єктів з використанням спеціально розробленого конвеєра кадрів, внутрішньої оптимізації всередині об'єктів і планування оновлень. У даній системі апдейти проходять тільки на деяких фреймах. Кадр складається з декількох частин, таких як Серіальна частина, Багатопоточна частина і Кінець кадру. У цій роботі ми описали всі плюси і мінуси згаданих частин кадру. Також в роботі представлена технологія планувальника, яка дозволяє оновлювати об'єкти з пропусками, але при цьому реагувати на них з мінімальними візуальними або логічними затримками.

Ключові слова: *ЦПУ, оптимізації, багатопоточність, відео ігри, система апдейту об’єктів*

ОПТИМИЗИРОВАННАЯ СИСТЕМА АПДЕЙТА ГЕЙМ ОБЪЕКТОВ ДЛЯ ВИДЕО ИГР

Марков Д. К.*, НТУУ «КПИ имени Игоря Сикорского», Киев, Украина, markov.dmitriy.dev@gmail.com*

Рогоза В. С*,. Д.т.н.,проф. НТУУ «КПИ имени Игоря Сикорского», . Киев, Украина rosvetnik@gmail.com*

Аннотация*. Управление событиями в игре, такими как изменения анимации и столкновения объектов, может быть сложной задачей, если вы не понимаете, как они структурированы и реализованы. Эта статья покажет вам, как планировщик поможет вашей игровой системе стать более организованной и гибкой. Физическое моделирование, анимация персонажей, обнаружение столкновений, игровой искусственный интеллект и рендеринг - это лишь несколько примеров игровых технологий, которые могут извлечь выгоду из использования планировщика. Время - важный фактор в компьютерных играх. При изменении сотен отдельных объектов и процессов в разные промежутки времени многие из этих технологий моделирования могут стать чрезвычайно сложными. Например, симуляция физики разбивает время на небольшие дискретные интервалы для каждого объекта, чтобы обновить движение объектов. Моделирование имело бы гораздо более высокую степень точности, если бы разрешение по времени было более точным. Поскольку в этой ситуации несколько объектов и временных интервалов обрабатываются одним и тем же кодом планирования, надежность имеет решающее значение для предотвращения узких мест при планировании. С ростом сложности компьютерных игр события и симуляции в реальном времени стали практически стандартом в современных игровых архитектурах. Что необходимо, так это способ управлять и выполнять несколько событий за кадр или много раз в пределах временного шага кадра. Планировщик может очень гибко управлять обновлением объектов, а также способствовать модульному подходу к расширяемости. В данной статье описан способ оптимизации обновления объектов с использованием специально разработанного конвейера кадров, внутренней оптимизации внутри объектов и планирования обновлений. В данной системе апдейты проходят только на некоторых фреймах. Кадр состоит из*

нескольких частей, таких как Сериальная часть, Многопотчная часть и Конец кадра. В этой работе мы описали все плюсы и минусы упомянутых частей кадра. Также в работе представлена технология планировщика, которая позволяет обновлять объекты с пропусками, но при этом реагировать на них с минимальными визуальными или логическими задержками.

***Ключевые слова**: ЦПУ, оптимизации, многопоточность, видео игры, система апдейта объектов*

Introduction

Managing events in a game, such as animation changes and object collisions, can be a difficult task if you don't understand how they're structured and implemented. This article will show you how a scheduler will help your game system be more organized and flexible.

Physics simulations, character animation, collision detection, game AI, and rendering are only a few examples of game technology that can benefit from using a scheduler. Time is an important factor in both of these innovations. With hundreds of individual objects and processes being modified at different time intervals, many of these simulation technologies may become extremely complex. For instance, a physics simulation will break down time into small, discrete intervals for each object in order to update the objects motion. The simulation would have a much higher degree of accuracy if the time resolution is finer. Since several objects and time intervals are handled by the same scheduling code in this situation, reliability is critical in avoiding scheduling bottlenecks.

The scheduler's ability to add and delete items on the fly is another essential feature. This allows new entities to join a game and participate in simulations alongside the rest of the game's entities without missing a beat, before being removed from the schedule when they are no longer needed.

Structure of the frame

The main goal of all this optimization is to achieve best performance with minimal losses. Scheduler is the one who will distribute load among the frames – this process is called throttling. The other thing that will help us in optimizing is multi-threading. Modern computers have at least 8 logical cores and it becomes common to see PC with 16-24 logical cores. So we have to use most of it for our system. In order to achieve best multi-threading we have to do some work on structure of the frame and find best solution to thread count and their tasks.

Structure of the frame: Threads

In our system there are two types of threads: primary threads and worker threads. Each primary thread is locked on particular logical core, which means that nobody can override out primary tasks. Workers, on the other hand, are not locked at all and they can migrate to any core they want. Priority-wise primary threads have greater priority so workers won’t conflict with them.

The best performance we have managed to get with the next setup: 4 primary threads and, using num_logical as number of logical cores in CPU on current machine, we have to spawn:

```
int num_primary = 4;
int cpus_left = num_logical - num_primary;
int num_workers = cpus_left - cpus_left / 4
```

This is needed because system sometimes wants to do something and if this work will be scheduled on the CPUs with primary threads – we will be stalled for the duration of that work. For example, basic interruption for mouse or keyboard can hold the core for 2-3 ms on average. If we are targeting 60 fps, we have only 16.6 ms for the whole frame and 2-3 ms is crucial on the primary cores.

Primary threads consist of: Primary, Secondary, R-Backend and Physics.

Primary is a thread that controls all engine – the main task of this thread is to start a new frame after we have finished previous. After that he initiates some initial stages of which we going to talk in the next chapters. Also render pipeline is initiated in this thread. One more important thing that lies on Primary thread is synchronization of all other threads we going to talk about later.

Secondary is a parallel helper thread for Primary – initial stage of the frame and some surpassing work is lying on his shoulders.

R-Backend is dedicated almost only for render tasks processing.

Physics thread, same as R-Backend, prioritises physics tasks over all other.

Workers threads are have only one purpose – they are working on tasks that were pushed into our task pool which is called 'mtrpc' which is stands for Multi-Threaded Remote Procedure Call. All of the primary threads are also working on this task pool while they have no 'own' tasks left, for example, R-Backend can work in 'worker-mode' when there are no render tasks left.

Structure of the frame: Frame stages

Frame is divided in three main parts: Serial part, MT (multi-threaded) part and End of the frame. Each of them has their unique purposes and unique pros and cons for using.

Serial part

Serial part is needed for synchronization purposes, that is why it is the most expensive and almost single-threaded part. On this part of the frame we are updating console, state of save if there are any, weather, HUD items, deferred actions (attach, detach, etc.), serial queues (clones, actions from visual scripting logic, etc.), bullet manager, garbage collect and many others. All of these tasks are updating in specific order so the behaviour of the system here is strongly determined. The last but not the least thing that lies on this part is to spawn objects update tasks – they won't be processed on this stage, but we are getting prepared for the next one. Regularly, this part takes up to 2-5% of the frame time.

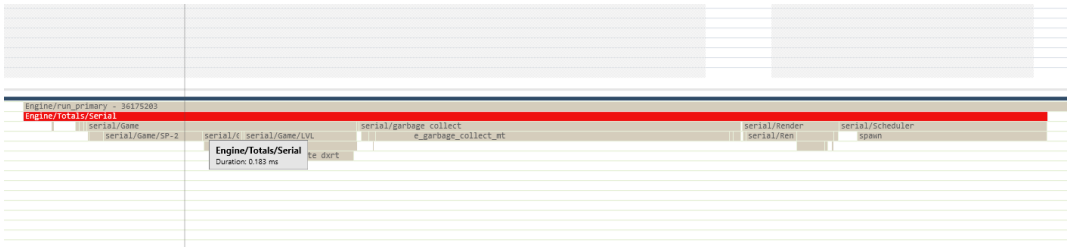

Fig. 1 - Serial part visualization

Pros of Serial part:

- No synchronization needed
- The most protected part (no mt actions, no object deletion)
- Fully determined actions sequence (no reordering, no data racing)

Cons of Serial part:

- Time spent here can be almost multiplied by the number of threads, because there is no MT and other threads are simply waiting

MT part

MT part is needed to speed-up the process of update of all engine. This part relies on maximum usage of all CPU resources and in order to achieve that we have to aim our MT usage to its limits. That idea lies in the foundation of our update system and scheduler itself and it will be described deeper in the next chapters. For now, all we have to know that here we are processing 90-95% of the frame, our objects updates, logic, animations, skeletons, transforms, hierarchies, particles and many other tasks.

Fig. 2 - MT part visualization

Pros of MT part:

- Maximal usage of CPU cores, smaller tasks integrates between larger ones
- The cheapest part of the frame (no stalls for other threads should be applied when task is executed on particular core)

Cons of MT part:

- Minimal sync allowed, locks only for very short amount of time, preferably spin-locks
- All sync-related actions (like change of hierarchy state) should be delayed to the serial part
- Minimum dependencies
- Requires maximum jobs distribution in order to be efficient

End of the frame

End of the frame (EotF) is a small part of the frame needed for only one reason – minimize input latency (but there are some other tasks as well, like update of brain unit storage and others, but they are not performance critical). We have to update input in the closest moment to the next frame, so we can send data to the GPU to render it. But input is not going there by himself, we have to update camera, player, his items and a bunch of other stuff in order to get synchronization of all input-dependent features. Basically this part is a MT part with a grain of Serial part. To start this part we still have to wait for most of the MT tasks, because we can't synchronize anything before the end of MT.

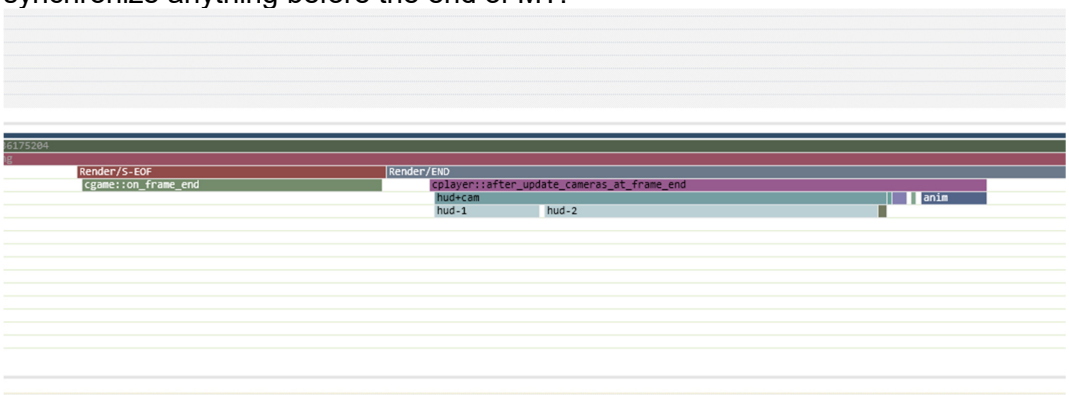

Fig. 3 - End of the frame part visualization

Pros of EotF part:

- Closest to the new frame so it is best place for input
- Still can be executed during part of MT work

Cons of EotF part:

- Should wait for vital tasks like update of player
- Might have poor MT if there is no work has left from MT part

Update of objects

Update of a single object consists of a sequence of three stages: update xform, update state and update dependencies. Update xform is basically an update of position, rotation and scale of an object. Update state updates all kinds of logic. Update dependencies is the most expensive part which updates skeleton and animation of the object.

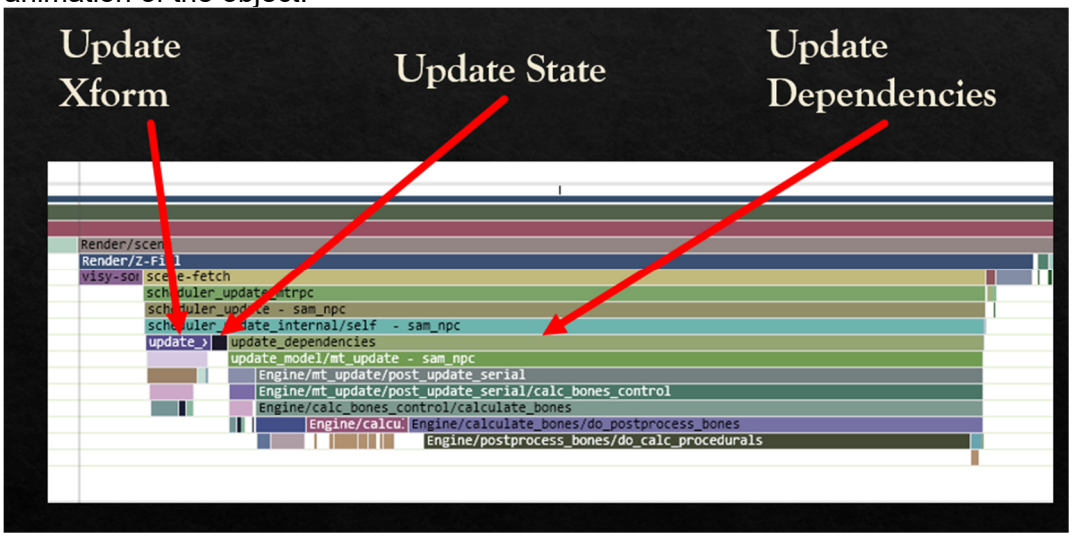

Fig. 4 - Update of object parts visualization

It is important to mention that our objects are not being updated one by one. All objects belong to hierarchies and objects are updated by hierarchies. So each update individual, but only hierarchy can be updated on frame, so we don't have any problems with desynchronization of objects in hierarchy. Start of hierarchy update is an update of root object. On the end of this update root object creates tasks to mtrpc for all of his children. Same procedure will be applied recursively to children. That idea allows us to simultaneously update objects from one level of the same hierarchy (assuming one-level objects should not influence each other directly without synchronization) and hierarchies can be simultaneously too. That allows us to utilize all cores of CPU quite well and minimize time spent in MT phase.

Scheduler

By this point we have good threads idea, simultaneous update thus good multi-threaded performance. But on big levels we can have thousand (literally thousands) of objects which need updates. So we have to minimize time spent on update without visible difference (or very small). Proposed method called throttling. It can be applied to hierarchies to maintain load among frames. For example, there are 100 objects and we can handle only 10 of them per frame. In the primitive approach, we just update each object once per 10 frames which will solve our performance issue, but will bring us stuttering and glitches. Then, let's try to do 'weighted throttling' where each objects will have some kind of mark which means 'how often

should it be updated". In the next section we will go deeper into this idea and rules which will bring us desired behaviour.

Scheduler: Types of update

On the first step scheduler divides objects in two big categories 'RTP', which stands for 'realtime priority' and 'NRM' – 'normal priority'. RTP objects should be updated on every frame, these are NPCs, logic objects that directly influences anything in the current view frustum, etc. NRM objects can be updated with gaps – these we are going to optimize hard.

NRM objects can go to 'sleeping' state, which means they haven't moved in the last 0.5 second. This flag later will be used to lower priority of the object's update.

For flying creatures there is a cheat, because they are much more visible and all their lags are much more noticeable.

The main tweaking point is determination if object is realtime or not. The best rules that were developed are listed below:

- Realtime – important objects
  - If object has been rendered
    - If dist < 55m
    - If dist < 275m & (!sleeping | flying)
    - If !sleeping & flying
  - If dist < 11m
  - If dist < 33m & !sleeping
- Normal – other objects
  - Dist_coef = dist_to_cam – radius
  - Update_time = 15..1000 ms

Conclusion

With the growing sophistication of computer games, real-time events and simulations are virtually a standard in today's game architectures. Paper presents a way to manage and execute multiple events per frame, or many times within a frames time-step. A scheduler can manage update of the objects in a very flexible fashion, as well as facilitate a modular approach for extensibility. Given paper describes a way to optimize objects update using specially developed frame pipeline, internal optimizations inside objects and scheduling of updates. In the given system tasks would run only on some frames. Frame consists of several stages such as Serial, MT and End of the frame. In this work we have described all pros and cons of the mentioned parts of the frame. Also work presents scheduler technology which allows objects to be updated with gaps but still be responsive and with minimal visual or logic lags.

References

1. Game Programming Patters, Update Method, https://gameprogrammingpatterns.com/update-method.html

2. InformIt, Game Programming Algorithms and Techniques: Overview, https://www.informit.com/articles/article.aspx?p=2167437&seqNum=4
3. Objects In Space, Update on Objects in Space, http://objectsgame.com/
4. C.S. Green and D. Bavelier, Enumeration versus multiple object tracking: the case of action video game players, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2896820/