УДК 004.021

МЕТОДЫ ОПТИМИЗАЦИИ ПРОИЗВОДИТЕЛЬНОСТИ МОБИЛЬНЫХ ИГР В UNITY3D

А.Д. КАРПОВИЧ
(Представлено: Д.В. ПЯТКИН)

Рассматриваются наиболее эффективные способы оптимизации мобильных игр, написанных в Unity3d.

Unity3d [1] – один из самых популярных игровых движков для мобильных платформ. Множество разработчиков используют его для создания и выпуска игр. При разработке игр для мобильных устройств важно учитывать их ограниченные технические возможности. Существуют слабые и мощные по производительности телефоны. Новые поколения мобильной GPU могут быть в 5 раз более производительнее своих предшественников. Несмотря на это важно вовремя оптимизировать приложение, для того чтобы добиться максимальной производительности на большинстве устройств и охватить большее количество пользователей.

Основной раздел

Оптимизация [2] — модификация системы для улучшения её эффективности.

Некоторые задачи часто могут быть выполнены более эффективно. Например, программный код, который вычисляет выражение, можно записать следующим образом:

```
int a = b*c + b*c + b*c;
```

Листинг 1. – Вычисление выражения без оптимизации

Здесь мы видим, что выполняется 3 операции умножения и 2 операции сложения. Этот код можно улучшить, вычислив произведение заранее. Теперь вместо 3 операций умножения мы обошлись только одной.

```
int d = b*c;
int a = d + d + d;
```

Листинг 2. – Вычисление выражения с использованием математической оптимизации

Обычной практикой считается оптимизация после реализации основного функционала игры. Преждевременная оптимизация кода может вызвать множество проблем и сильно замедлить разработку. Так как оптимизированный код менее гибкий и его сложнее читать.

Британский ученый Майкл А. Джексон часто цитирует свои правила оптимизации программ: Первое правило оптимизации программы: не делайте ее. Второе правило оптимизации программы (только для экспертов!): не делайте ее пока что [3]. Он обосновал это тем, что учитывая рост скорости компьютеров, программа будет достаточно быстрой. Кроме того, если пытаться слишком много оптимизировать, то код сильно усложниться, и появится много ошибок.

Однако при разработке мобильной игры оптимизацию не следует считать последней стадией разработки проекта: аппаратное обеспечение, представленное сейчас на рынке, сильно ограничено по сравнению с компьютерами. Поэтому высок риск того, что игра не будет работать на большинстве устройств и про оптимизацию следует думать с самого начала разработки.

Для оптимизации требуется найти узкое место: критическую часть кода, которая является основным потребителем необходимого ресурса. Улучшение примерно 20 % кода иногда влечёт за собой изменение 80 % результатов, согласно *принципу Парето* [4]. Утечка ресурсов (памяти, дескрипторов и т.д.) также может привести к падению скорости выполнения программы. Для поиска таких утечек используются специальные отладочные инструменты, а для обнаружения узких мест применяются программы – *профайлеры* [5].

Профайлер – инструмент, позволяющий программисту увидеть, сколько времени у программы уходит на выполнение каждой функции и ранжировать их по порядку. Когда верхняя функция занимает 3% времени, это означает, что если удастся вдвое сократить время ее выполнения, то общая производительность программы ускорится на 1,5%.

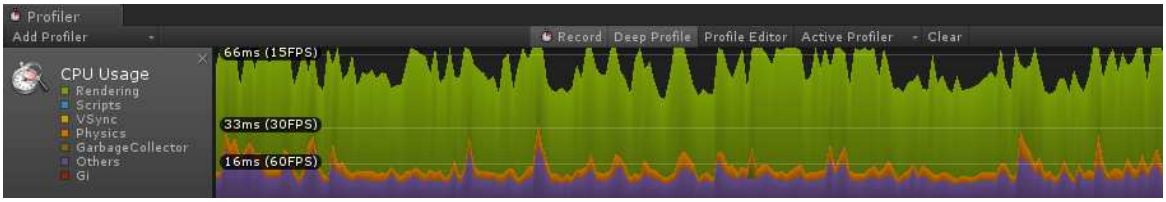

Рисунок. – Окно профайлера в Unity

Профилирование важно, потому что оно поможет выяснить, какие оптимизации действительно приведут к большому приросту производительности, а какие являются пустой тратой времени. Благодаря тому, что рендеринг обрабатывается на отдельном чипе – GPU, отрисовка одного кадра занимает в два раза меньше времени (только GPU, а не CPU + GPU). Это означает, что если CPU замедляет работу, оптимизация графики не повысит частоту кадров, и если GPU замедляет работу, не помогут оптимизация физики и скриптов.

Оптимизация в основном фокусируется на одиночном или повторном времени выполнения, использовании памяти, дискового пространства, пропускной способности или некотором другом ресурсе. Это обычно требует компромиссов: один параметр оптимизируется за счёт других. Например, увеличение размера программного кэша улучшает производительность времени выполнения, но также увеличивает потребление памяти. Прозрачность кода и его выразительность получается за счет деоптимизации. Сложные специализированные алгоритмы требуют больше усилий по отладке и увеличивают вероятность ошибок.

Элементы больше всего нуждающиеся в оптимизации:

– графика – 70%;
– расчет физики – 20%;
– скрипты – 10%.

Выделение памяти для новых объектов – дорогая операция. Использование методов создания новых объектов во время игры нужно максимально минимизировать. Для большого количества однотипных объектов можно использовать пул объектов. Один раз инициализировав определенное количество объектов, они будут использоваться повторно, вместо создания и уничтожения каждый раз, что существенно снизит нагрузку на CPU и GPU.

Все, что будет использоваться больше одного раза лучше *кешировать* [6]. Операции типа GameObject.Find() или GetComponent() достаточно ресурсозатратны, а если они вызываются где-нибудь в цикле или в Update(), то производительность может упасть. Пример кеширования можно увидеть в листинге 3.

```
Transform _cachedTransform;

void Awake () {
 _cachedTransform = transform;
}

void Update () {
 _cachedTransform.localPosition = _newPosition;
}
```

Листинг 3. – Кеширование компонента Transform

Для оптимизации отрисовки графики для начала нужно уменьшить количество *draw-call* [7].

Draw-call – команда графическому API (например, OpenGL или Direct3D) на отрисовку. Графический API производит значительную работу для каждого draw-call, что сильно влияет на производительность CPU.

Для мобильных устройств рекомендуется иметь до 100 draw-calls. Для уменьшения их количества можно использовать *батчинг* [7] – отрисовка однотипных объектов использующих один и тот же материал за один подход.

Для графики упаковывать все спрайты в *атласы* [8]. Для этого есть как встроенный инструмент Sprite Packer, так и продвинутый TexturePacker. Таким образом уменьшается количество вызовов отрисовок спрайтов.

Шейдеры управляют всеми визуальными элементами в игре. Нужно знать, что оптимизированные шейдеры могут значительно повысить производительность, поскольку они не требуют большого количества вычислений.

Еще один способ ускорить игру – использовать *мобильные шейдеры* [9], даже на более высокопроизводительных платформах. При работе над мобильной игрой, обязательно нужно использовать мобильные шейдеры, потому что они требуют меньше вычислений.

Так же для улучшения отрисовки графики можно использовать *occlusion culling* [10].

Occlusion Culling – функция, отключающая рендеринг тех объектов, которые в данные момент не видит камера. Используя виртуальную камеру, происходит проход по сцене для построения иерархии потенциально видимых объектов. Эти данные используются каждой камерой для определения того, что она видит, и что нет. Опираясь на полученную информацию, происходит рендеринг только видимых объектов. Это уменьшает количество draw-calls и увеличивает производительность игры.

В Unity есть возможность использовать разные типы *коллайдеров* [11] – компонентов, позволяющих обрабатывать столкновения объектов. По возможности лучше использовать примитивные коллайдеры такие как Box Collider и Circle Collider для ускорения расчёта физики.

Компонент Rigidbody обычно используются для добавления веса к объекту. Если объект привязан к Rigidbody, он может зависеть от физики, такой как сила тяжести других сил. Большое количество объектов Rigidbody, отрицательно повлияет на производительность.

Unity поддерживает несколько типов аудио, по умолчанию он будет импортировать аудиоклипы для использования типа загрузки *Decompress On Load* [12] вместе со сжатием *Vorbis* [13]. Звуковые эффекты обычно короткие и, следовательно, имеют небольшие требования к памяти. Для них настройка *Decompress on Load* будет работать лучше всего, но тип сжатия должен быть либо *PCM*, либо *ADPCM* [14]. PCM обеспечивает более высокое качество, но поставляется с большим размером файла, что отлично подходит для очень короткого, но важного звукового эффекта. ADPCM имеет коэффициент сжатия в 3,5 раза меньше, чем PCM, и лучше всего используется для аудио эффектов, которые используются очень часто.

Для более длинных аудиоклипов, таких как фоновая музыка или другие большие файлы, лучше использовать сжатыми в памяти, что приводит к распаковке файла прямо перед воспроизведением.

Таким образом, в данной статье рассмотрены основные методы оптимизации мобильных игр, однако не стоит забывать, что методы оптимизации сильно зависят от типа разрабатываемой игры. Некоторые методы оптимизации применимы только к узкому спектру задач и могут негативно сказаться на итоговой производительности. Так же не стоит пытаться полностью оптимизировать программу. Оптимизацию стоит заканчивать тогда, когда достигнута необходимая производительность

ЛИТЕРАТУРА

1. Unity (игровой движок) [Электронный ресурс] / Wikipedia – The Free Encyclopedia. – Режим доступа: https://ru.wikipedia.org/wiki/Unity_(игровой_движок). – Дата доступа: 14.09.18.
2. Оптимизация (информатика) [Электронный ресурс] / Wikipedia – The Free Encyclopedia. – Режим доступа: https://ru.wikipedia.org/wiki/Оптимизация_(информатика). –Дата доступа: 14.09.18.
3. Руководство по оптимизации [Электронный ресурс] / Docs Unity3d. – Режим доступа: https://docs.unity3d.com/ru/current/Manual/MobileOptimizationPracticalGuide.html. – Дата доступа: 14.09.18.
4. 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: 14.09.18.
5. The Profiler Window [Электронный ресурс] / Docs Unity3d. –Режим доступа: https://docs.unity3d.com/ru/current/Manual/Profiler.html. – Дата доступа: 14.09.18.
6. Оптимизация скриптов [Электронный ресурс] / Docs Unity3d. – Режим доступа: https://docs.unity3d.com/ru/current/Manual/MobileOptimizationPracticalScriptingOptimizations.html. – Дата доступа: 15.09.18.
7. Батчинг вызовов отрисовки (Draw Call Batching) [Электронный ресурс] / Docs Unity3d. – Режим доступа: https://docs.unity3d.com/ru/current/Manual/DrawCallBatching.html. – Дата доступа: 15.09.18.
8. Sprite Atlas [Электронный ресурс] / Docs Unity3d. – Режим доступа: https://docs.unity3d.com/Manual/SpriteAtlas.html. – Дата доступа: 15.09.18.
9. Использование и производительность встроенных шейдеров [Электронный ресурс] / Docs Unity3d. – Режим доступа: https://docs.unity3d.com/ru/current/Manual/shader-Performance.html. – Дата доступа: 16.09.18.

10. Occlusion Culling [Электронный ресурс] / Docs Unity3d. – Режим доступа: https://docs.unity3d.com/ru/current/Manual/OcclusionCulling.html. – Дата доступа: 16.09.18.
11. Коллайдеры (Colliders) [Электронный ресурс] / Docs Unity3d. – Режим доступа: https://docs.unity3d.com/ru/current/Manual/CollidersOverview.html. – Дата доступа: 16.09.18.
12. Audio Clip [Электронный ресурс] / Docs Unity3d. – Режим доступа: https://docs.unity3d.com/Manual/class-AudioClip.html. – Дата доступа: 16.09.18.
13. Vorbis [Электронный ресурс] / Docs Unity3d. – Режим доступа: https://docs.unity3d.com/ScriptReference/AudioCompressionFormat.Vorbis.html. – Дата доступа: 16.09.18.
14. Audio compression format [Электронный ресурс] / Docs Unity3d. – Режим доступа: https://docs.unity3d.com/ScriptReference/AudioCompressionFormat.html. – Дата доступа: 16.09.18.