

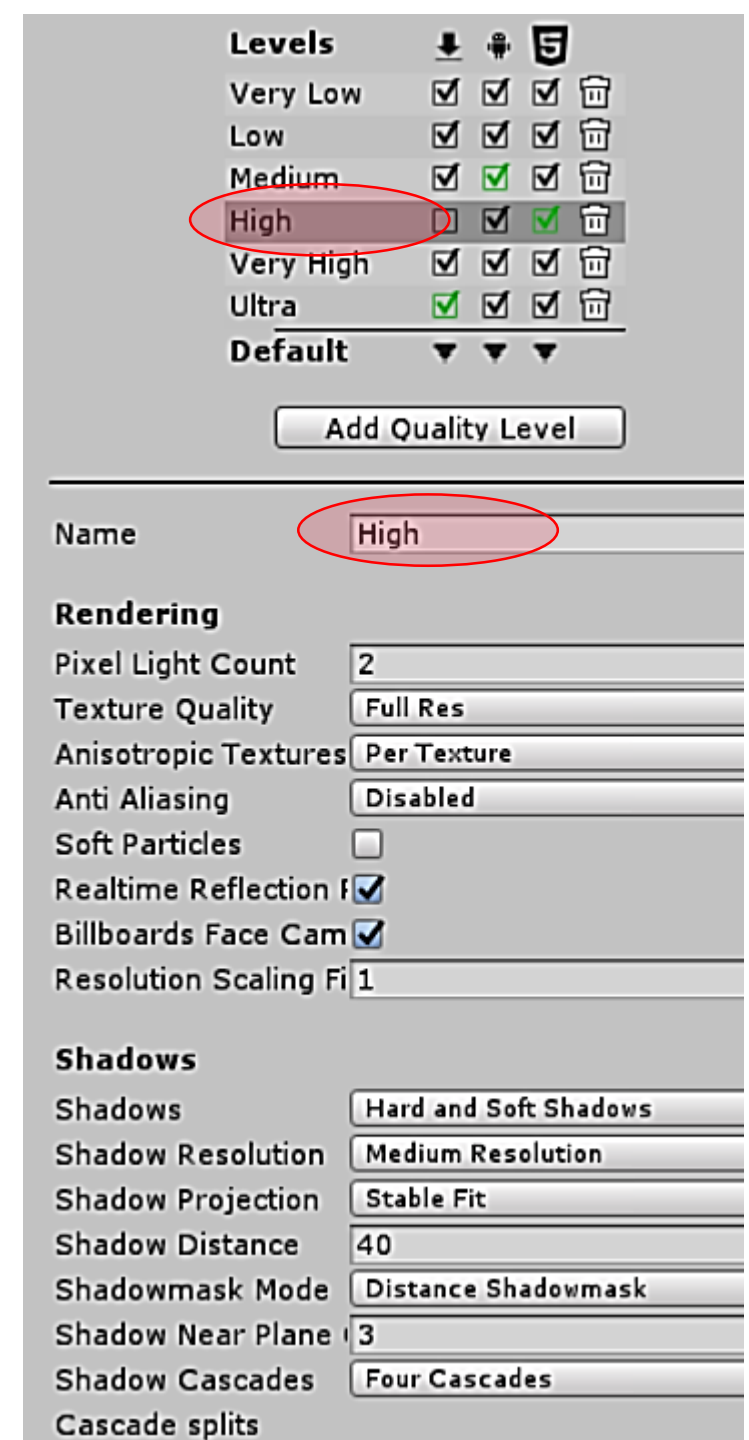
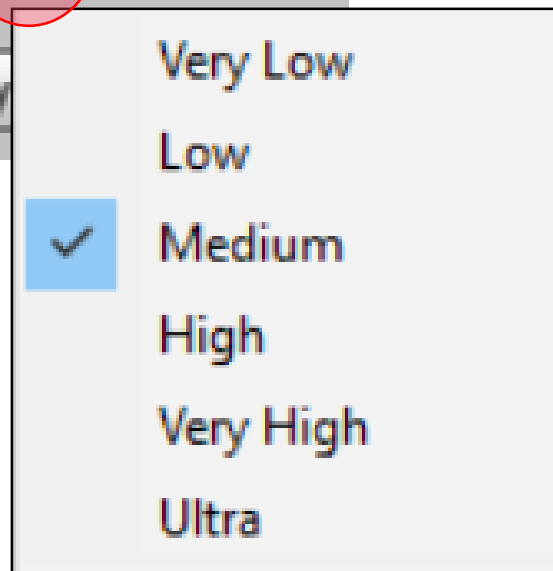
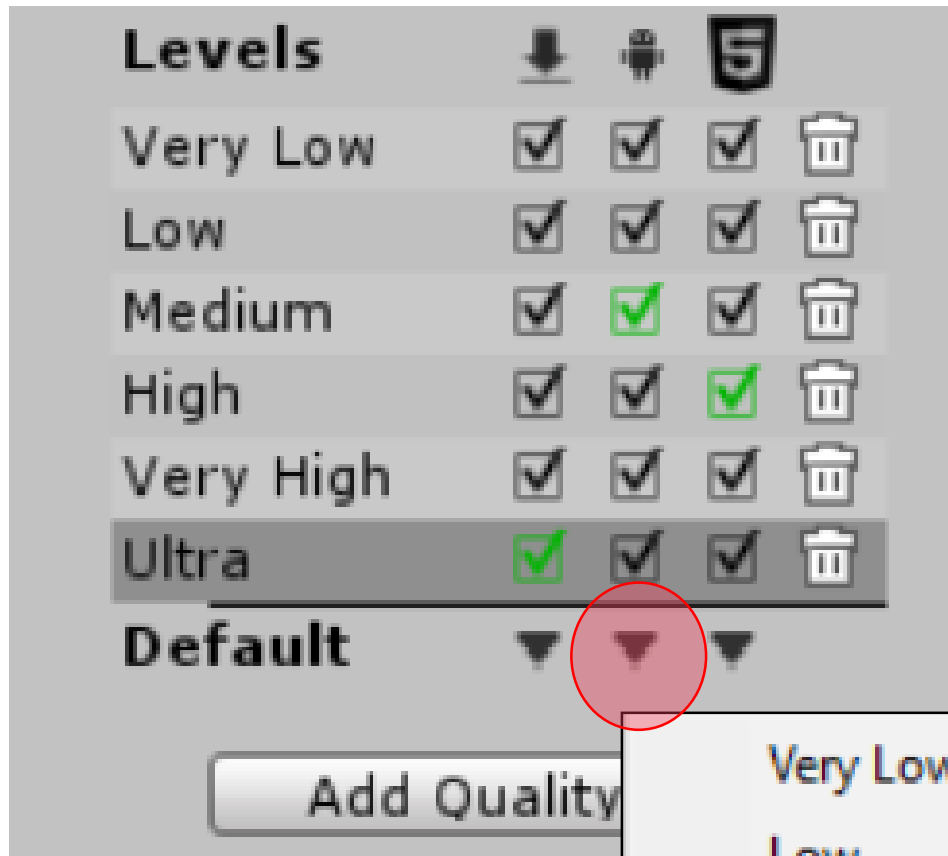
Unity

Оптимизация проектов

Настройки качества графики

Unity позволяет установить уровень графического качества, который он попытается выполнить. Инспектор параметров качества **Edit > Project Settings > Quality** используется для выбора уровня качества в редакторе для выбранного устройства.





Name	High
Rendering	
Pixel Light Count	2
Texture Quality	Full Res
Anisotropic Textures	Per Texture
Anti Aliasing	Disabled
Soft Particles	<input type="checkbox"/>
Realtime Reflections	<input checked="" type="checkbox"/>
Billboards Face Camera	<input checked="" type="checkbox"/>
Resolution Scaling Factor	1

Pixel Light Count количество просчитываемых источников света для пикселя при использовании Forward Rendering (упреждающий рендеринг)

Texture Quality отображать ли текстуры с максимальным разрешением или меньше. Возможные значения: **Full Res**, **Half Res**, **Quarter Res** и **Eighth Res**.

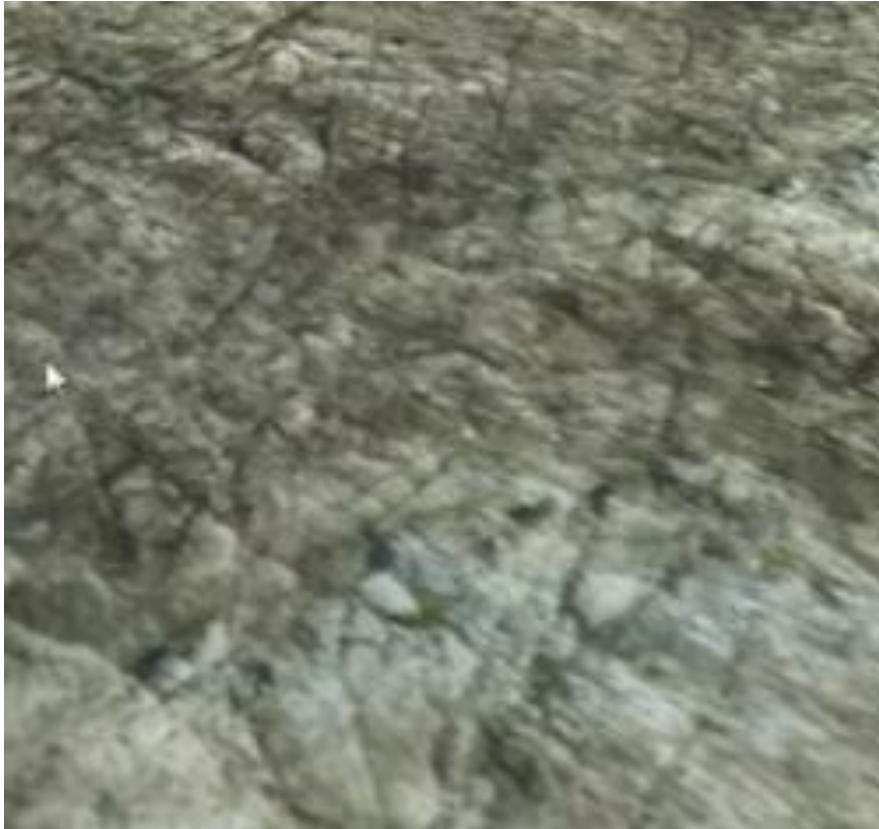
Анизотропные текстуры - Это позволяет использовать анизотропные текстуры. Параметры: **Disabled**, **Per Texture** и **Forced On** (т.е. всегда включены).

AntiAliasing устанавливает уровень сглаживания, который будет использоваться.

Soft Particles Следует использовать мягкое смешивание для частиц?

Billboard Face Camera Position объекты (например, трава) поворачиваются к позиции камеры (true) или по направлению камеры (false). Включение сильно нагружает систему.

Texture Quality отображать ли текстуры с максимальным разрешением или меньше. Возможные значения: **Full Res**, **Half Res**, **Quarter Res** и **Eighth Res**.

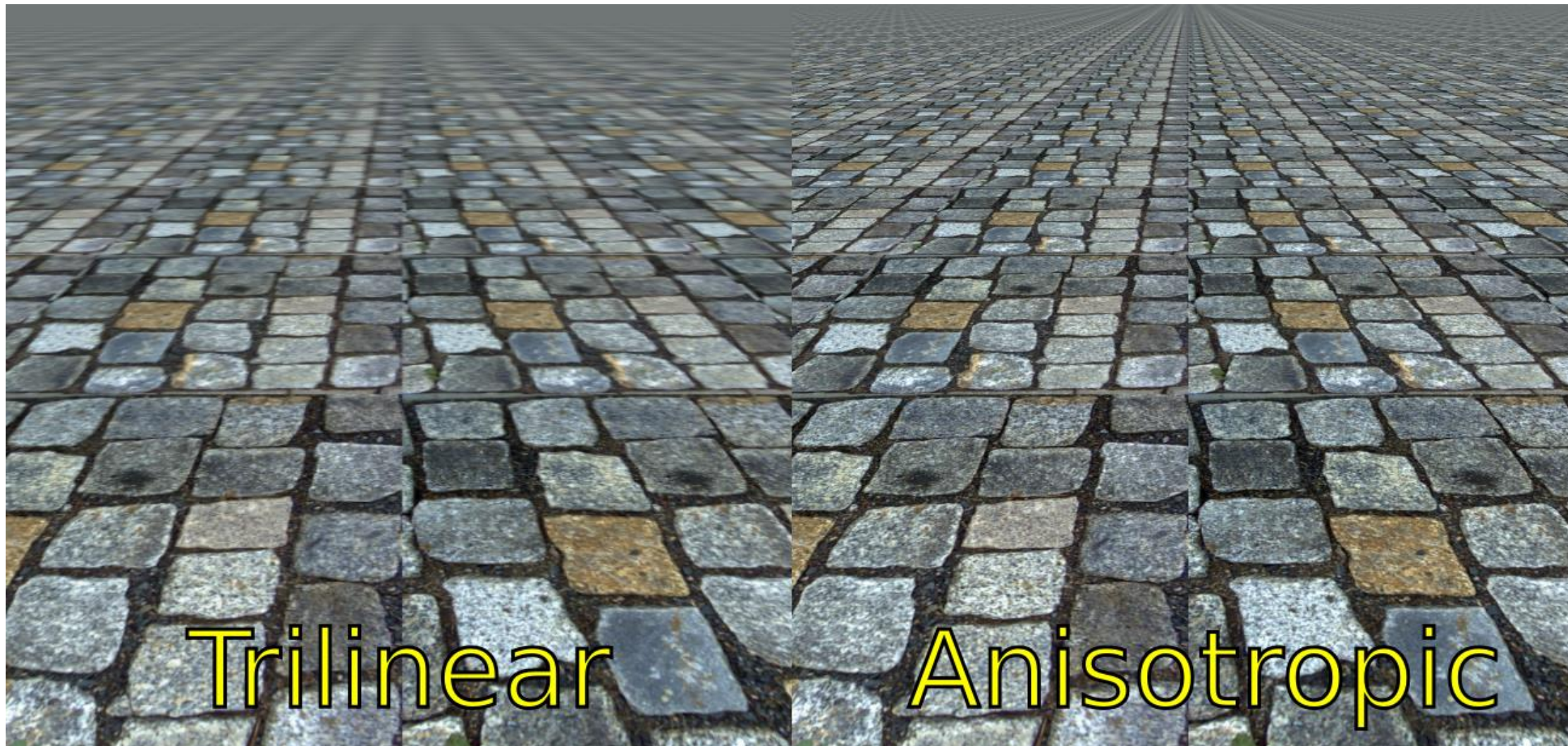


Полный размер

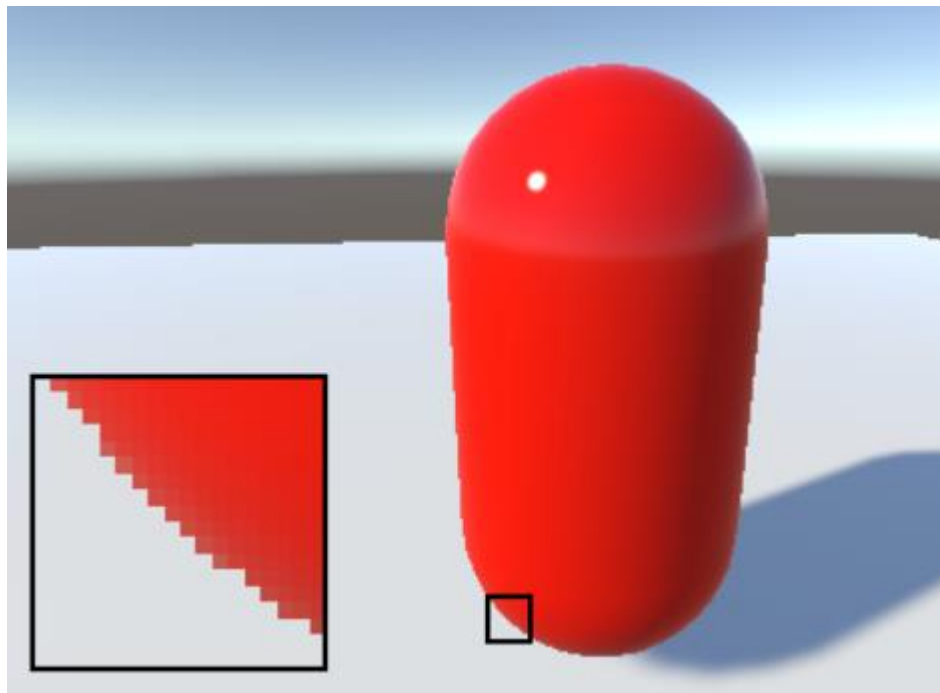


1/8 от размера

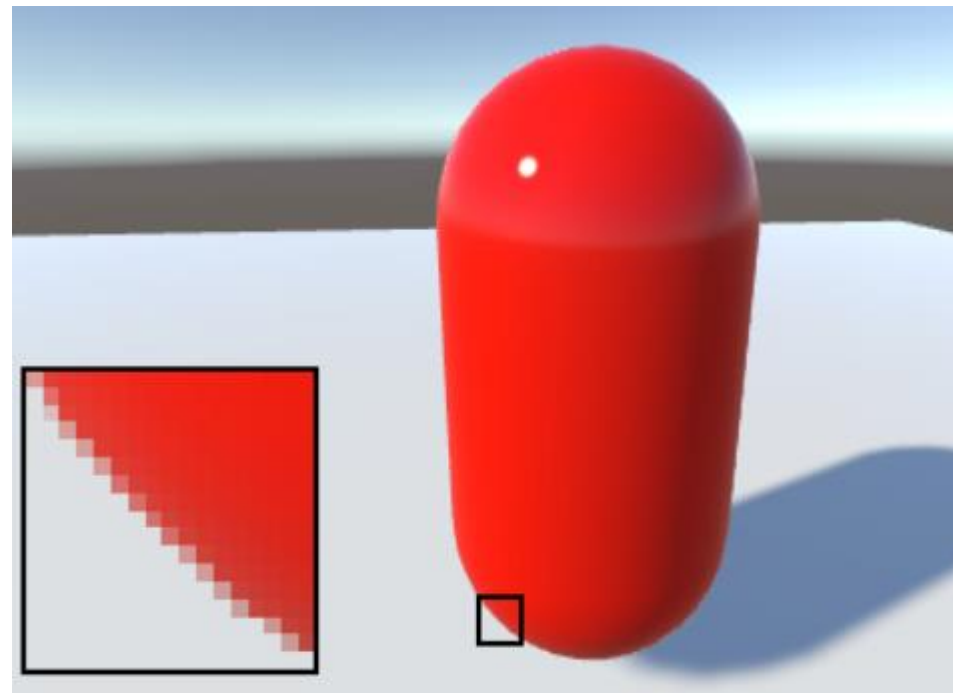
Анизотропные текстуры



AntiAliasing устанавливает уровень сглаживания

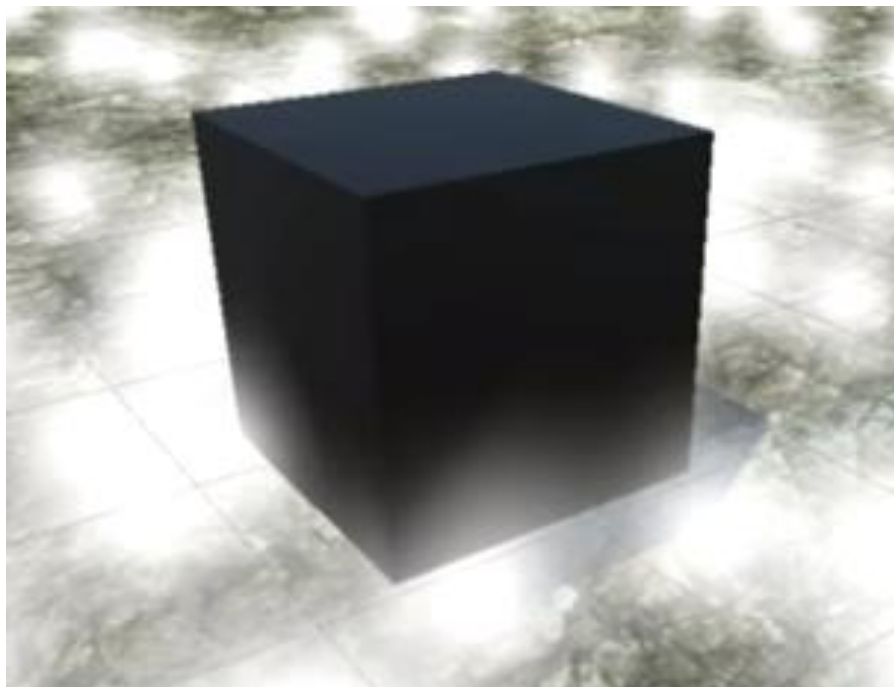


Disabled

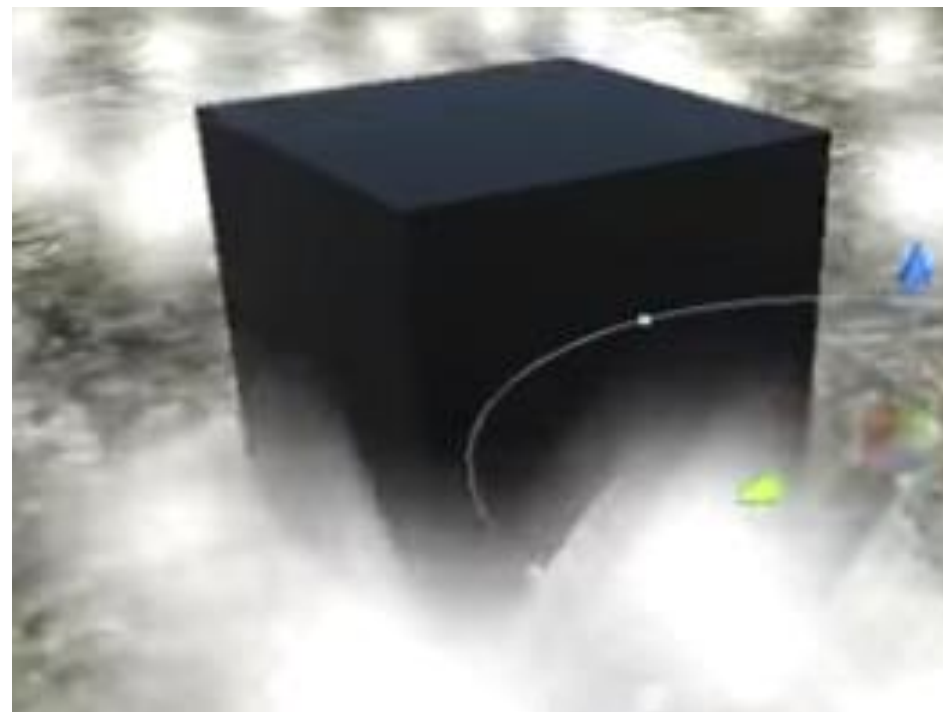


8x

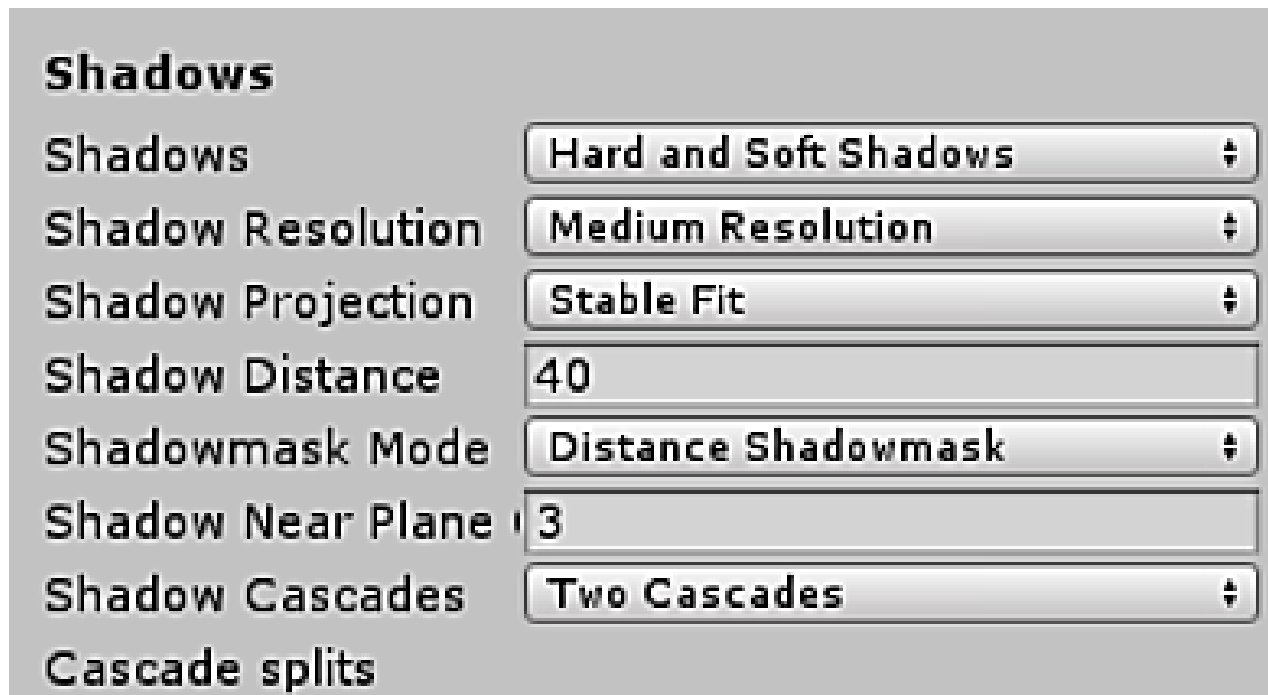
Soft Particles (мягкое смешивание для частиц) нагружает систему, не всегда целесообразно для мобильных устройств



Soft Particles включено



выключено



Shadows определяет, какой тип теней должен использоваться. Доступны опции *Hard u Soft Shadows, Hard Shadows Only u Disable Shadows*.

Shadow Resolution Тени могут отображаться в нескольких разных разрешениях: *Низкий, Средний, Высокий u Очень Высокий*.

Shadow Projection Существует два разных способа проецирования теней из направленного света *Close Fit u Stable Fit*

Shadow Distance Максимальное расстояние от камеры, при которой будут видны тени.

Shadowmask Mode *Distance Shadowmask* использует тени в реальном времени до Shadow Distance и запеченные тени за его пределами.

Shadowmask : Static GameObjects всегда бросают запеченные тени.

Shadow Near Plane Offset Смещение тени

Shadow Cascades может быть установлено равным 0, 2 или 4. Более высокое количество каскадов дает лучшее качество.

Параметр **Shadows**



Параметр **Shadow Resolutions**

Низкое разрешение



высокое



Other

Blend Weights	4 Bones
V Sync Count	Every V Blank
Lod Bias	2
Maximum LOD Level	0
Particle Raycast Budget	4096
Async Upload Time Slice	2
Async Upload Buffer Size	4

Blend Weights Количество костей, которые могут повлиять на заданную вершину во время анимации.

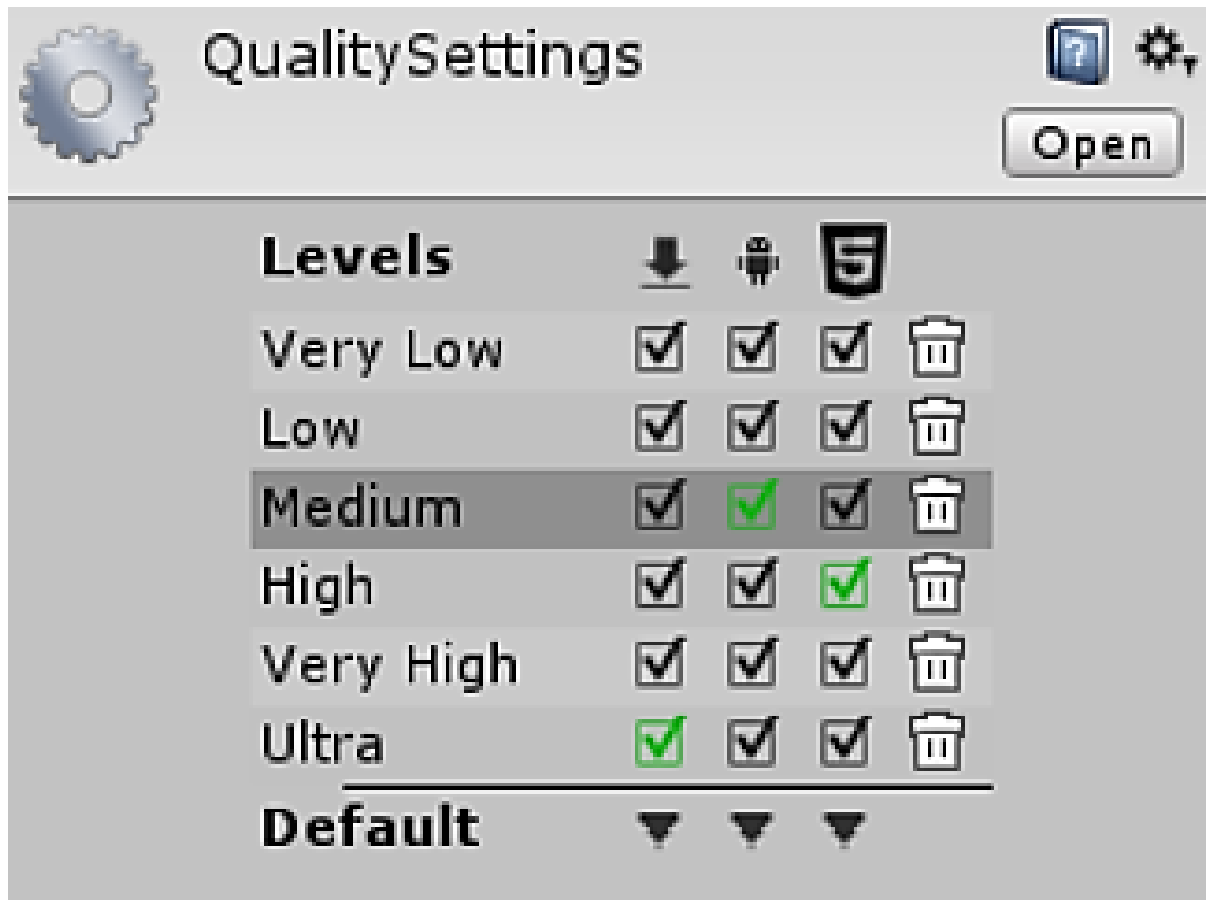
VSync Count синхронизация с частотой обновления устройства, чтобы избежать «разрыва» артефактов

LOD Bias чем больше, тем лучше от 0 до 2

Particle Raycast Budget Максимальное количество raycasts, используемых для аппроксимации столкновений систем частиц (см.)

Async Upload Time Slice и **Async Upload Buffer Size** параметры загрузки асинхронных текстур

Количество Raycasts для использования при обнаружениях столкновений частиц при различных уровнях качества



4

64

4096

Параметр **VSync Count**

Обновления Unity не обязательно синхронизируются с обновлениями на дисплее, поэтому Unity может выпускать новый кадр, пока дисплей все еще отображает предыдущий. Это приведет к визуальному артефакту, называемому «разрывом» в позиции на экране, где происходит смена кадра.



Установка типа рендерига

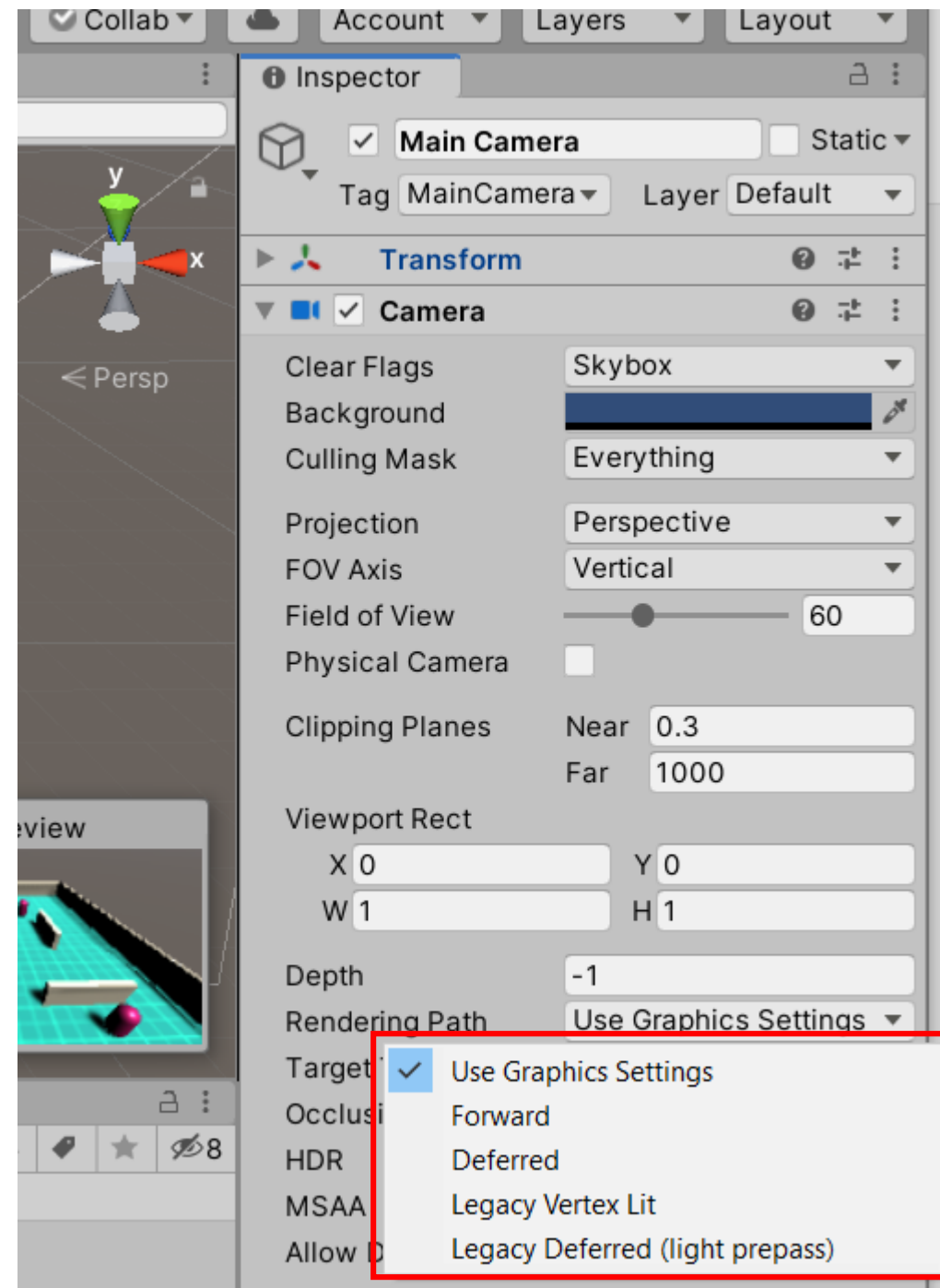
Камера ->Inspector

С помощью параметра **Rendering Path** можно указать в Unity, как обрабатывать рендеринг света и теней в игре.

Unity поддерживает разные параметры **Rendering Path**. Различные **Rendering Path** имеют разные возможности и характеристики производительности. Решение о том, какой тип рендеринга наиболее подходит для проекта, зависит от типа проекта и целевого оборудования.

Этот параметр может быть определен для каждой камеры.

Если графический процессор на устройстве, на котором запущен проект, не поддерживает тип рендеринга, который вы выбрали, Unity автоматически использует тип рендеринга с более низкой точностью.



Rendering Path

Forward (Прямая отрисовка) - это тип рендеринга по умолчанию. Освещение в реальном времени при прямом рендеринге очень дорого обходится. Чтобы компенсировать эту стоимость, можно выбрать, сколько источников света Unity будет отображать на пиксель одновременно. Unity визуализирует остальные источники света в сцене с более низкой точностью.

Подходит если в проекте не используется большое количество источников света в реальном времени или если точность освещения не важна для проекта.

Deferred (Отложенная отрисовка) – это тип рендеринга с высочайшим качеством отрисовки освещения и теней.

При таком типе не существует ограничений на количество источников света, влияющих на один объект.

Отложенная отрисовка требует поддержки графического процессора и имеет некоторые ограничения. Например, не поддерживаются полупрозрачные объекты (Unity отображает их с использованием прямого рендеринга).

Подходит если в проекте большое количество источников света в реальном времени и требуется высокий уровень точности освещения, а целевое оборудование поддерживает отложенное затенение.

Legacy Vertex Lit - это самый быстрый тип рендеринга с самой низкой точностью освещения. У него самая широкая аппаратная поддержка. Этот тип рендеринга не поддерживает большинство попиксельных эффектов: тени, отображение нормалей, зеркальные блики не поддерживаются.

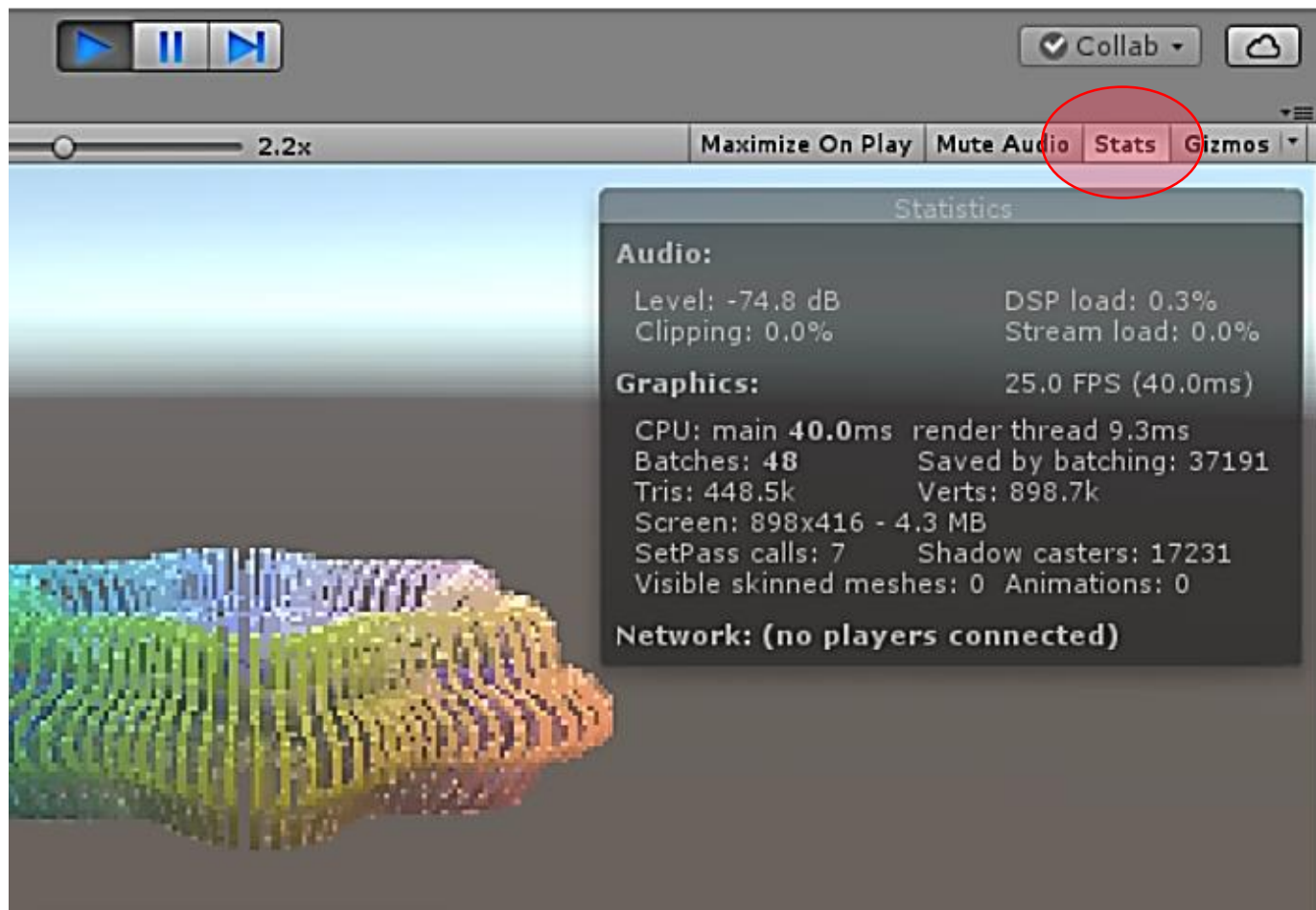
Legacy Deferred - это подмножество отложенной отрисовки.

Какова стоимость графики

Графическая часть приложения нагружает в первую очередь две системы компьютера: GPU (графический процессор) и CPU (центральный процессор). **Первое правило любой оптимизации: найти, где возникает проблема**, так как стратегия оптимизации для GPU и CPU имеет существенные различия.

Найти проблему позволяют различные инструменты профилирования работы приложения.

Окно Rendering Statistics отображает статистику рендеринга в реальном времени, полезно для оптимизации производительности.



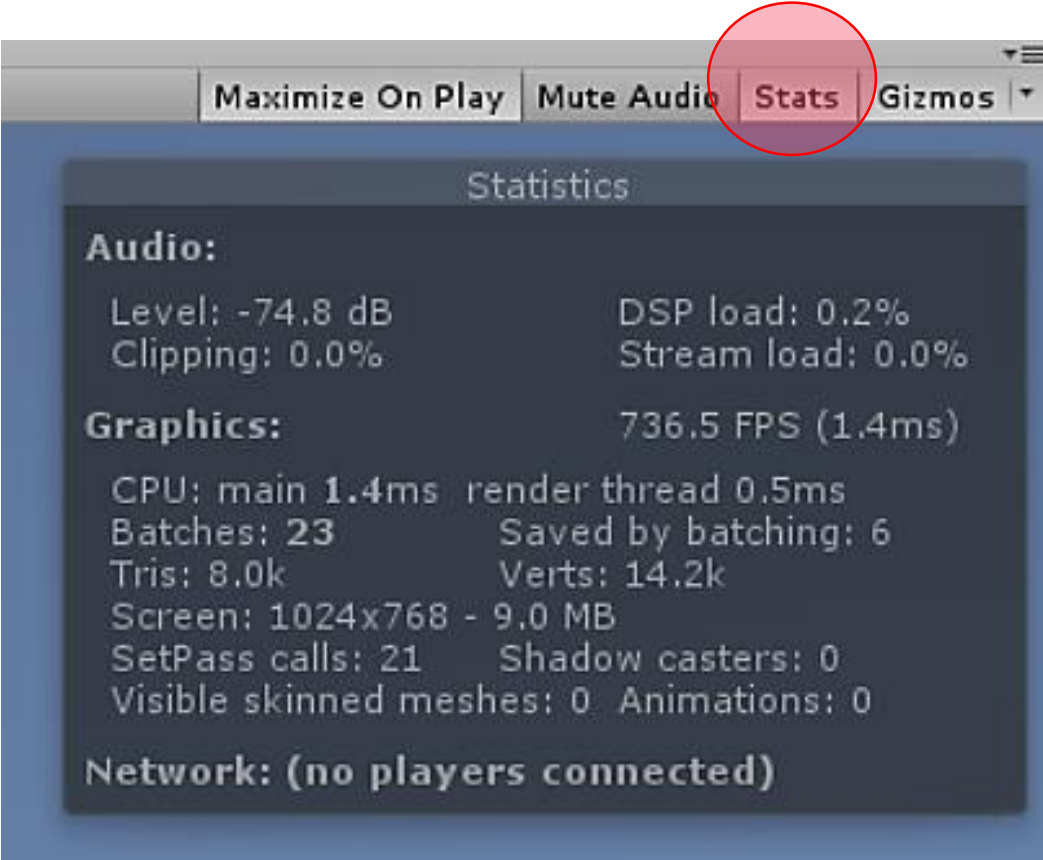
«**Время кадра**» — это время в миллисекундах, за которое может быть построен и отрисован один кадр. Чтобы получить 60 кадров в секунду, один кадр должен быть обчислен за 16 миллисекунд или еще быстрее.

Draw Call (вызов отрисовки) — это одна графическая команда, которая должна что-то отрисовать.

Сложность кадра можно оценивать именно с помощью вызовов отрисовки. Чем их больше, тем дольше они обчисляются, тем медленнее выполняется 1 кадр, тем меньше кадров в одной секунде и тем меньше итоговый **FPS**.

И наоборот — чем меньше вызовов отрисовки, тем быстрее считается кадр, и тем выше FPS.

Окно Rendering Statistics



Time per frame and FPS	Время, затрачиваемое на обработку и рендеринг одного кадра и его обратная величина - количество кадров в секунду (FPS).
Batches	Количество сбатченных DC
Saved by batching	Количество объединенных батчей.
Tris and Verts	Количество отрисованных треугольников и вершин.
Screen	Размер экрана, вместе с уровнем anti-aliasing сглаживания и использованием памяти.
SetPass	Количество проходов рендеринга. Каждый проход требует, чтобы среда выполнения Unity привязала новый шейдер, что может вызвать накладные расходы ЦП.
Visible Skinned Meshes	Количество отрендеренных skinned мешей.
Animations	Количество проигрываемых анимаций.

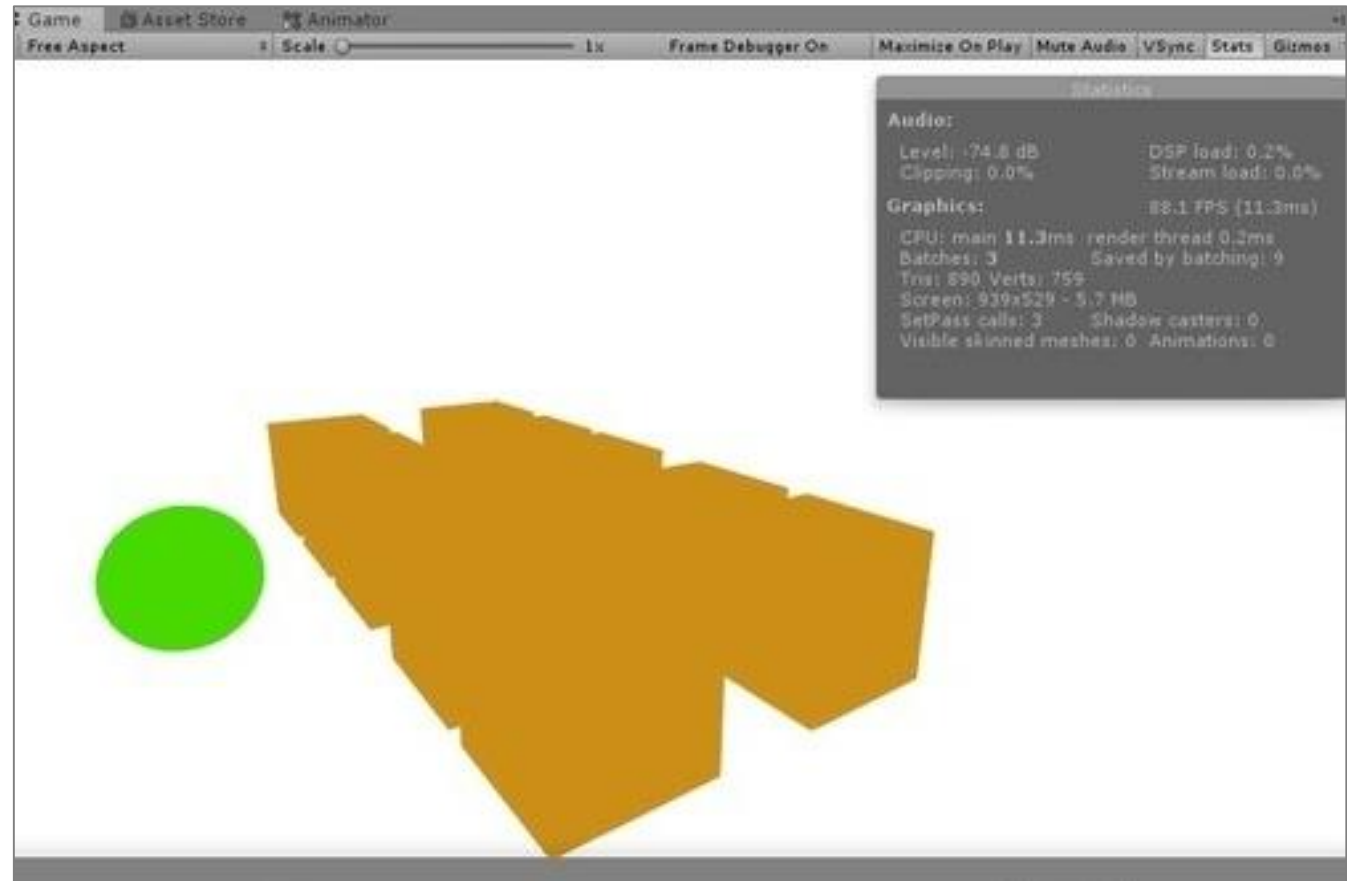
Вызовы отрисовки (Draw Call)

Простой пример: у нас есть 10 ящиков и одна сфера. Все десять ящиков имеют один и тот же шейдер, материал и количество полигонов. Логично предположить, что у нас будет 11 вызовов, ведь в кадре 11 объектов.

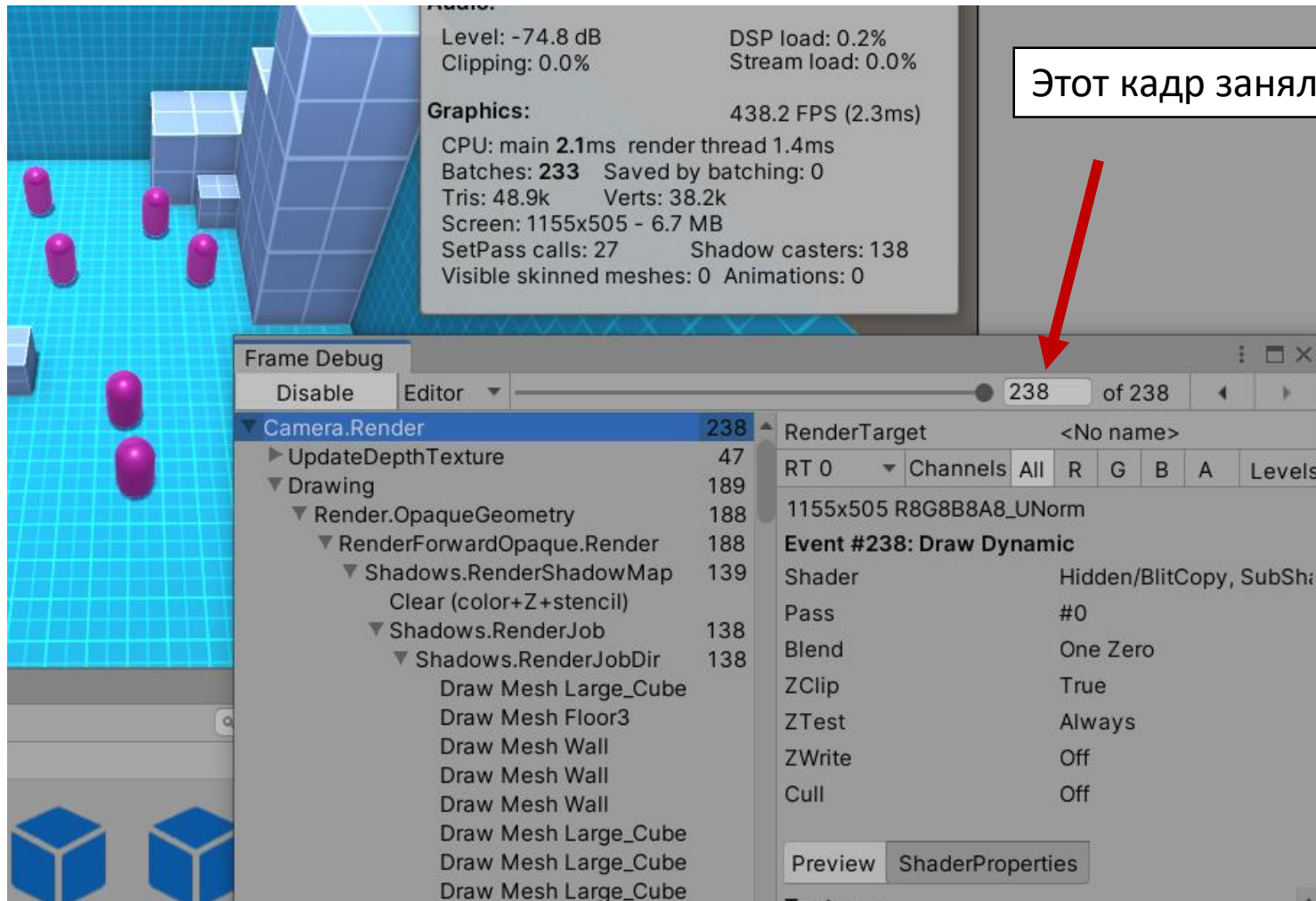
Всё почти так и есть. Программа рендерит так: «ящик, ящик, ящик, ящик, ящик... и потом сферу. Кадр построен».

Однако, судя по статистике, у нас всего три **батча** (вызова отрисовки).

Дело в том, что движок сам понимает, что перед ним десять одинаковых ящиков. Он сам «сшивает» ящики в один объект, и теперь процессор отдает нашей видеопамяти на отрисовку один кусок меша, который мы видим как 10 отдельных ящиков. Этот процесс называется «батчинг».



Окно Frame Debugger (меню: **Window > Analysis > Frame Debugger**) позволяет просматривать отдельные *вызовы отрисовки (DC)* в конкретном кадре. Помимо перечисления вызовов отрисовки, отладчик также позволяет выполнять их пошагово.



Этот кадр занял 238 отрисовок

Примерный разброс лимитов на Draw Call можно описать так:

- **Мобильные игры** — примерно **100** на средне-высоких настройках, на топовых устройствах можно и больше.
- **Игры для ПК и консолей** — до **4000**.

Например, *Battelfield* и третий «Ведьмак» в среднем работают на 1000-2000 вызовах отрисовки.

<https://dtf.ru/gamedev/101398-optimizaciya-pochemu-vremya-vazhnee-poligonov>.

Батчинг вызовов отрисовки

В Unity поддерживается так называемая пакетная обработка (батчинг), позволяющая поместить несколько игровых объектов в один вызов рендеринга. **Статический батчинг предназначен для статических объектов, а динамический — для движущихся объектов.**

Чтобы батчинг сработал, объекты должны быть «одинаковы». Объекты должны иметь один и тот же шейдер, материал на этом шейдере, текстуру и остальные параметры объекта. Если у вас есть два одинаковых материала, отличающихся только текстурами, можно объединить эти текстуры в одну большую — процесс часто называемый созданием текстурного атласа.

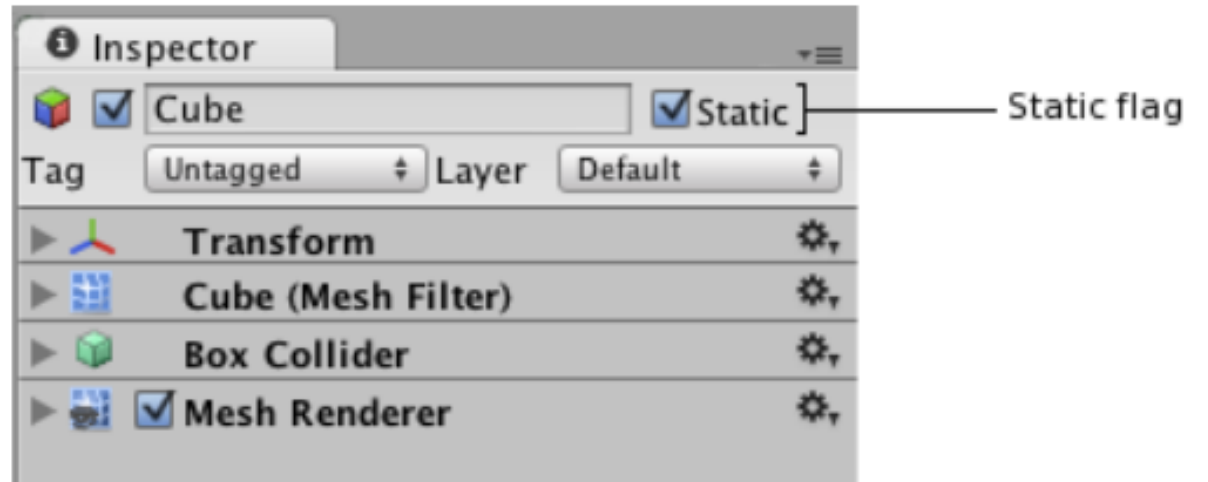
Динамический батчинг выполняется автоматически. Обычно у динамического батчинга очень жёсткие лимиты; он не позволяет сшивать тяжёлую геометрию и занимается в основном мелкими деталями в кадре. Например, если от стены динамически отделились мелкие осколки, то они, скорее всего, будут сшиты динамическим батчингом в каждом отдельном кадре.

Партикли (система частиц) умеют «сшиваться» сами по себе, поэтому их стоит использовать, если нужно нарисовать много одинаковых мелких объектов. (Например, тысячи рыбок)

Статический батчинг требуется задавать вручную. Для этого нужно указать что этот объект статичен. Статичный батчинг позволяет снизить количество DC для геометрии любого размера, если она не двигается и использует общий материал. Статичный батчинг более эффективен, чем динамический.

Статический батчинг (Static Batching)

При использовании статичного батчинга вы должны убедиться, что объекты статичны и не двигаются, не вращаются и не масштабируются во время выполнения. Если эти условия соблюдаются, можно пометить объекты как статичные, поставив галочку Static в Inspector:



НО!!! Использование статичного батчинга требует дополнительной памяти для хранения объединённой геометрии. Если несколько объектов используют общую геометрию перед статичным батчингом, то копия геометрии создаётся для каждого объекта, либо в рантайме, либо в редакторе. Это может быть не очень удачной идеей — иногда вы можете пожертвовать производительностью визуализации некоторых объектов для снижения затрат памяти. Для примера, пометив все деревья как статичные на лесистом уровне вы можете получить серьёзный удар по объёму доступной памяти.

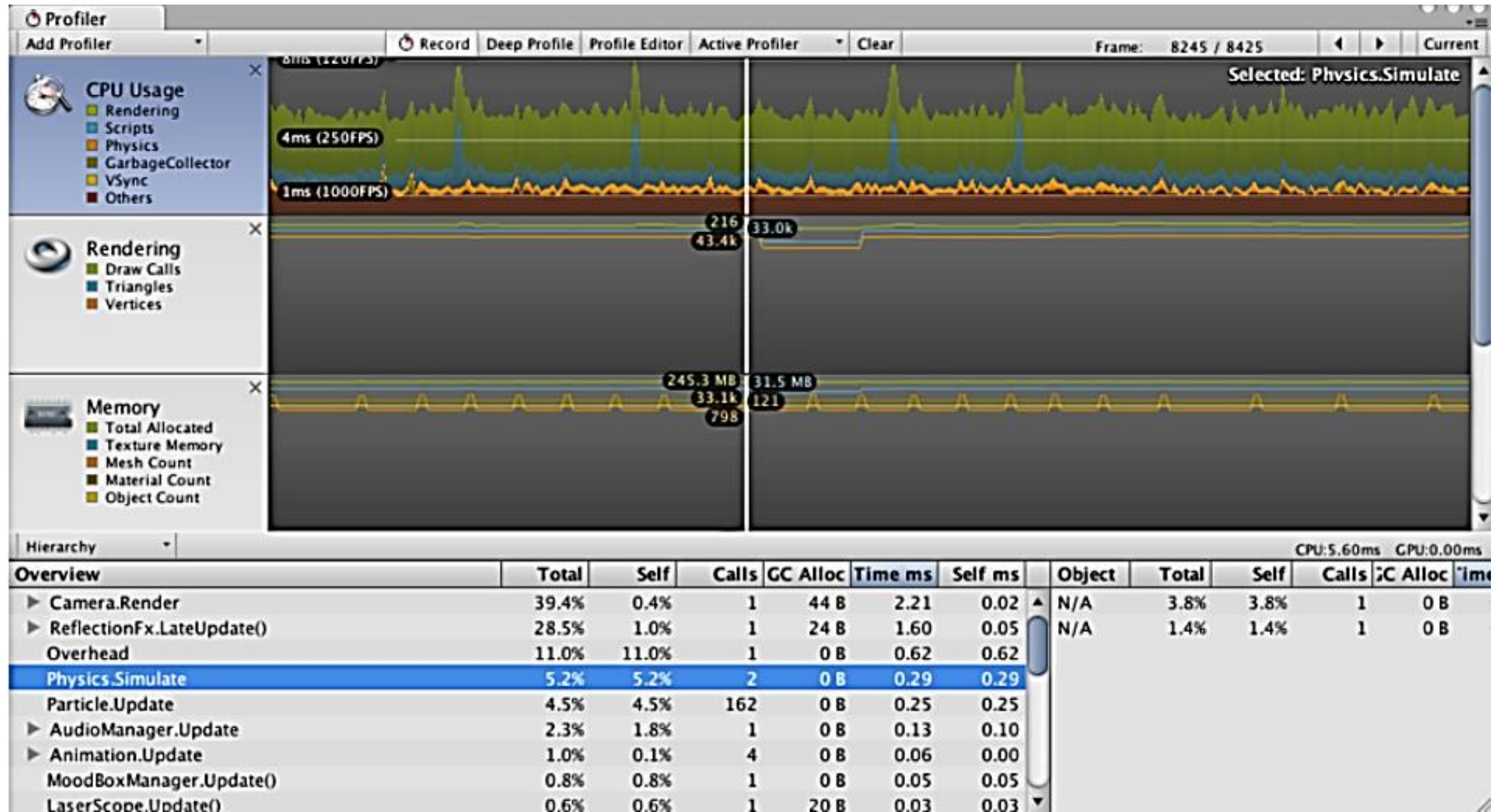
<https://docs.unity3d.com/ru/current/Manual/DrawCallBatching.html>

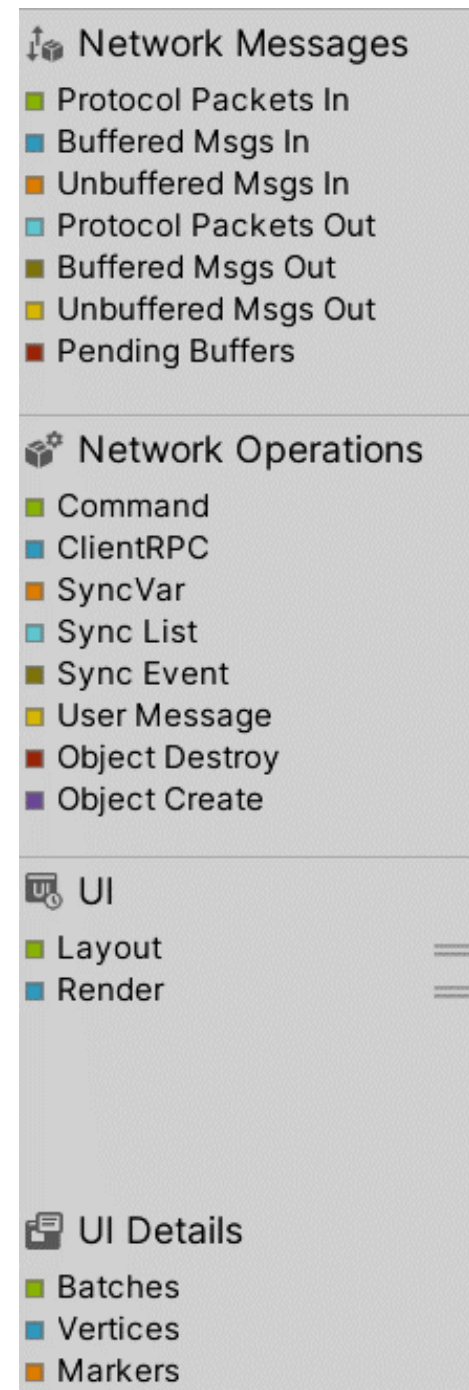
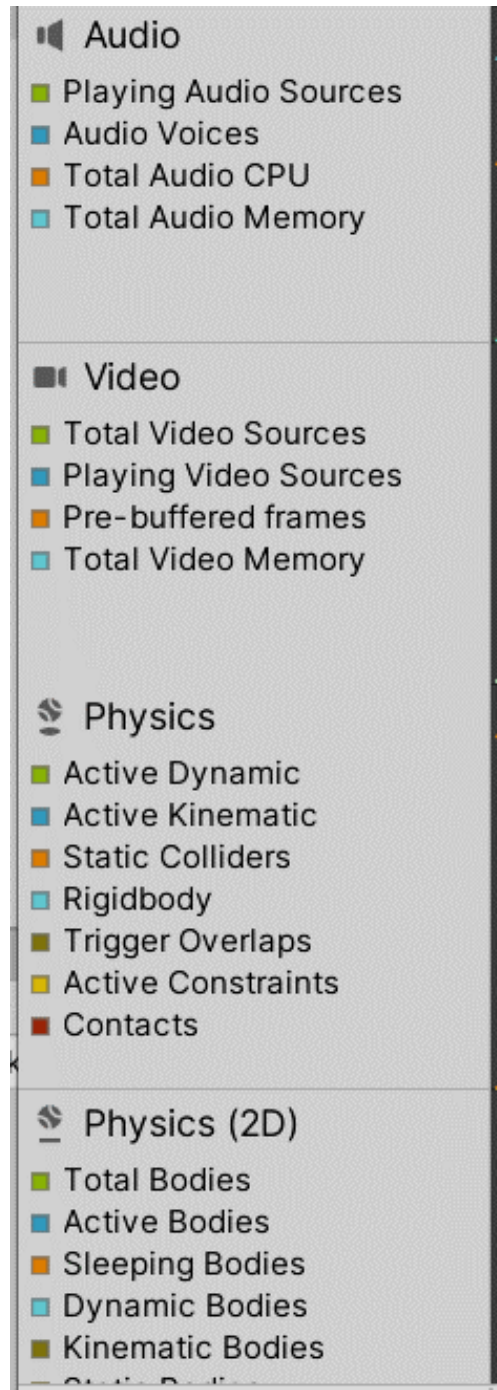
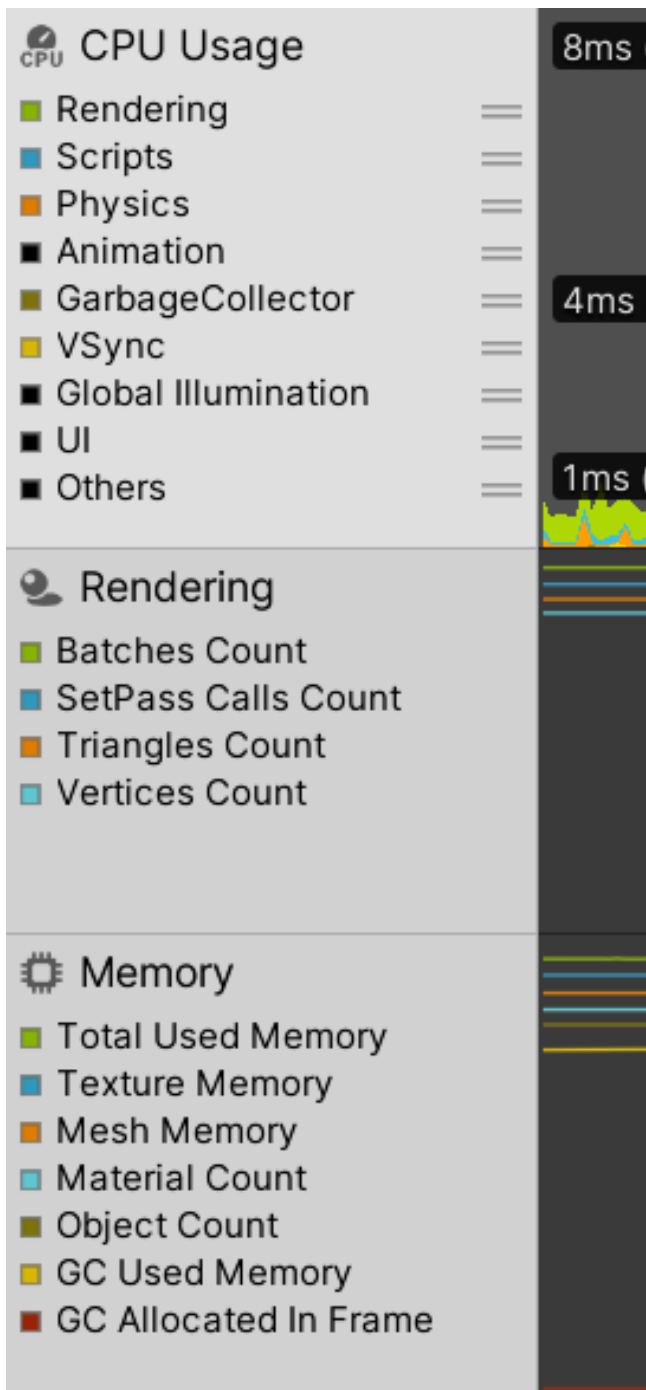
Динамический батчинг (Dynamic Batching)

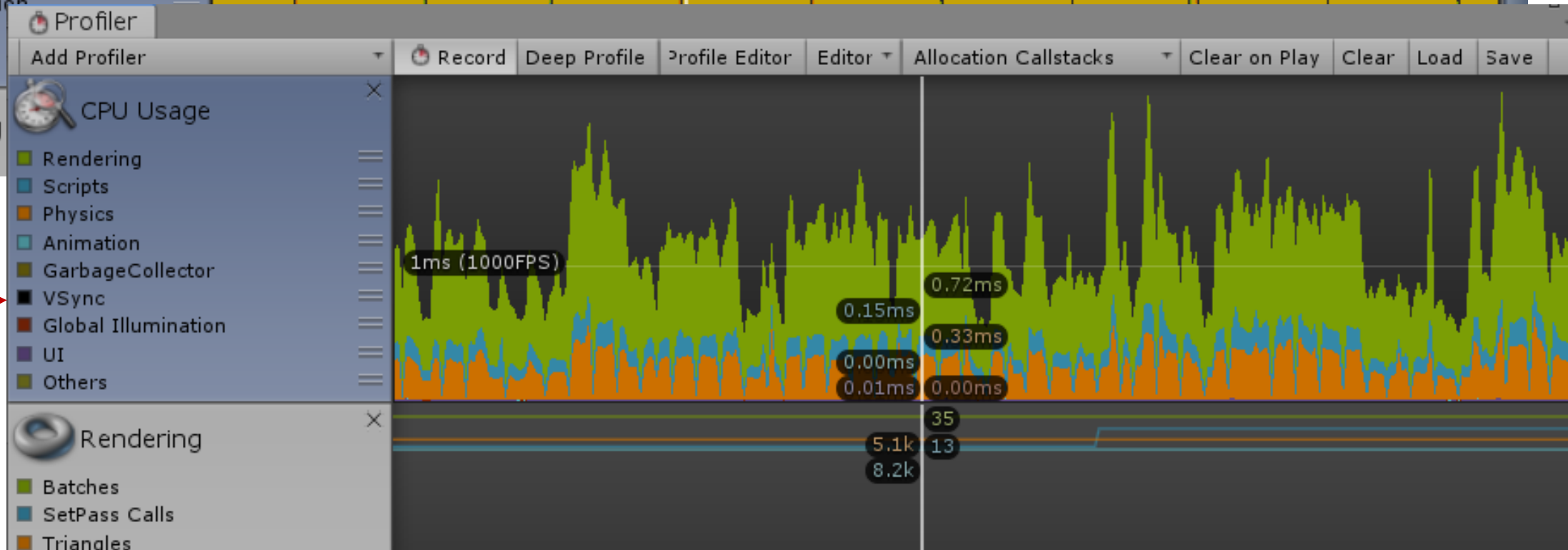
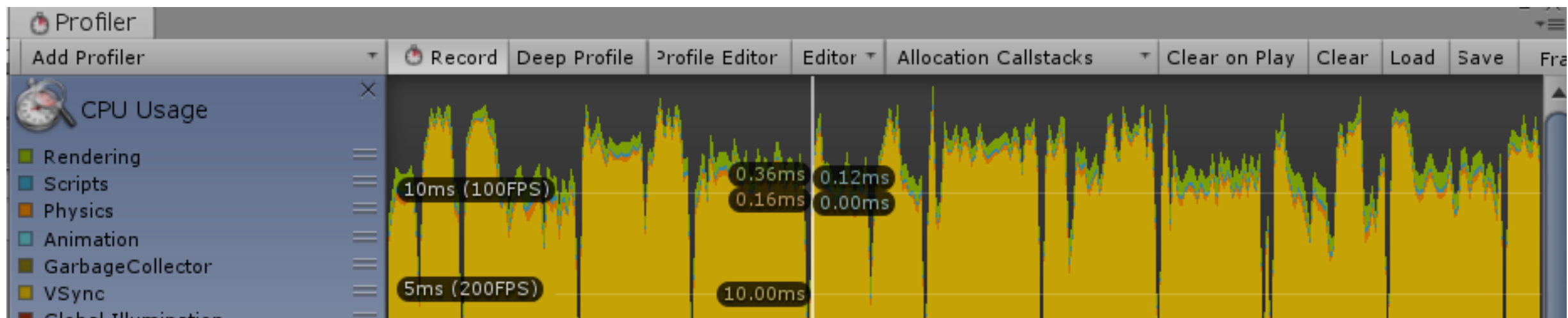
Unity может автоматически батчить движущиеся объекты в один DC, если они используют общий материал и отвечают ряду других критериев. Динамический батчинг применяется автоматически и не требует дополнительных действий.

- Динамический батчинг связан с дополнительной нагрузкой для **каждой вершины**, так что он применим только к мешам, содержащим менее **900** вершин в сумме.
- Использование разных экземпляров материалов — даже если они по сути своей являются одним материалом — сделает динамический батчинг невозможным.
- Объекты с картами освещения имеют дополнительное свойство: индекс карты освещения и смещение/масштаб внутри карты освещения. Для батчинга объекты должны ссылаться на одно и то же место в карте освещения.

The Profiler Window







Типичные узкие места и их проверка:

GPU часто ограничен филлрейтом (fillrate) или пропускной способностью памяти.

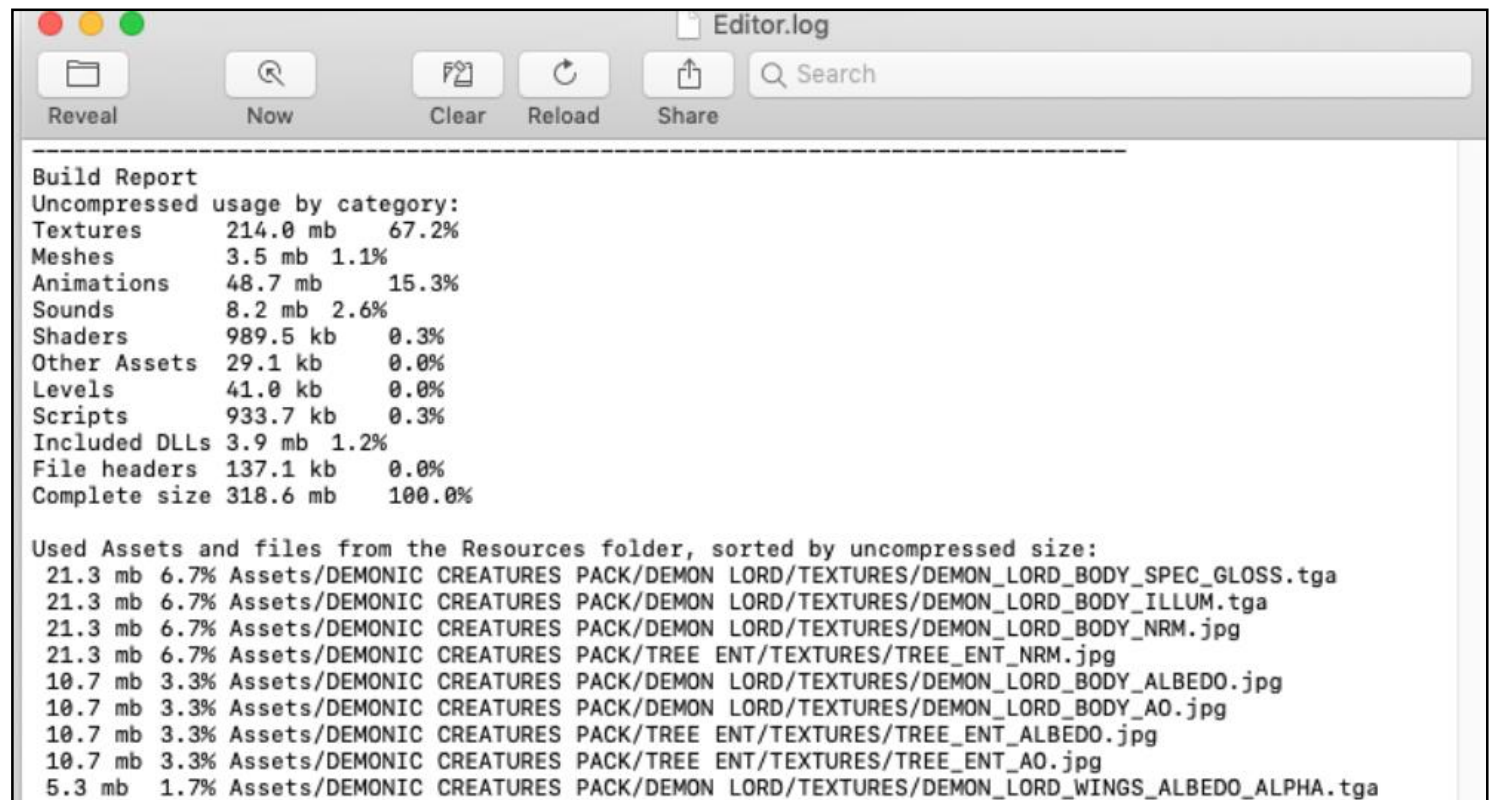
- **Запуск игры с более низким разрешением экрана увеличивает производительность?** Тогда вы скорее всего ограничены филлрейтом GPU.
- CPU часто ограничен количеством вещей, которые должны быть отрисованы, также известно, как “draw calls”. Проверьте показатель “draw calls” в окне **Rendering Statistics**; если он составляет больше нескольких тысяч (для PC) или нескольких сотен (для мобильных устройств), то вам может потребоваться оптимизация количества объектов.
- GPU обрабатывает слишком много вершин. Какое количество вершин является нормальным, определяется GPU и набором вертексных шейдеров. Можно посоветовать использовать не более 100 тысяч для мобильных устройств и не более нескольких миллионов для PC.
- **Рендеринг не создаёт проблем ни для GPU, ни для CPU?** Тогда проблема может быть, к примеру, в скриптах или физике. Используйте **профайлер** для поиска источника проблемы.

Журнал редактора

Еще одним шагом в уменьшении размера приложения является определение того, какие активы вносят в него больший вклад, поскольку эти активы являются наиболее вероятными кандидатами для оптимизации. Эта информация доступна в журнале редактора сразу после выполнения сборки. Перейдите в окно консоли и нажмите на маленькую выпадающие панели в правом верхнем углу, и выберите **Open Editor Log**.

Журнал редактора предоставляет сводку активов в разбивке по типу, а затем перечисляет все отдельные активы в порядке внесения размера. Как правило, такие объекты, как текстуры, звуки и анимация, занимают больше всего места, а **сценарии**, уровни и **шейдеры** как правило, имеют наименьшее влияние.

Журнал редактора помогает вам идентифицировать активы, которые вы, возможно, захотите удалить или оптимизировать, но вы должны рассмотреть следующее перед началом работы:



Советы по оптимизации Физики от Unity

<https://docs.unity3d.com/ru/current/Manual/MobileOptimisation.html>

Физика может сильно нагрузить процессор. Можно проследить это с помощью профайлера редактора. Если физика сильно нагружает процессор:

- Настройте *Time.fixedDeltaTime* (в Project settings -> Time) так, чтобы он был как можно более высоким. Если ваша игра с медленным движением, то, вероятно, вам понадобится меньше фиксированных обновлений, чем игре с быстрым движением. Быстрый темп игры нуждается в более частых расчетах, поэтому, чтобы не было сбоев с коллизиями, *fixedDeltaTime* должен быть ниже.
- Rigidbodies используйте только там, где это необходимо.
- Вместо меш коллайдеров старайтесь использовать примитивные коллайдеры.
- **Никогда не двигайте статический коллайдер (т.е. коллайдер без Rigidbody),** так как это сильно скажется на производительности. В профайлере это отобразится как “Static Collider.Move”, но на самом деле будет обрабатываться в *Physics.Simulate*. **Если понадобится, добавьте Rigidbody и установите *isKinematic* в true.**

Оптимизация скриптов

Есть несколько алгоритмов с сомнительной репутацией, хотя на первый взгляд они выглядят невинно. Постоянное объединение строк - классический пример:

```
void Update() {  
    string scoreText = "Score: " + score.ToString();  
    scoreBoard.text = scoreText;  
}
```

...будет выделять новые строки при каждом вызове Update и генерировать постоянный поток нового мусора. Большую часть этого можно сохранить, обновляя текст только тогда, когда счёт меняется:

```
void Update() {  
    if (score != oldScore) {  
        scoreText = "Score: " + score.ToString();  
        scoreBoard.text = scoreText;  
        oldScore = score;  
    }  
}
```

Статьи с анализом



saul 31 марта 2015 в 08:52 Разработка

Планирование оптимизации с Unity

Разбор статьи с анализом влияния приемов оптимизации на fps

<https://habrahabr.ru/company/intel/blog/254353/>

Качество текстур

Изменение кадровой скорости при переключении между различным качеством текстур в Unity



saul 31 марта 2015 в 08:52 Разработка

Планирование оптимизации с Unity



Сцена с разрешением 1/8



Сцена с полным разрешением

Texture Quality:	1/8	¼	½	Full
Fantastic	72	73	72	69

Shadow Distance

Shadow Distance — это параметр, определяющий глубину отбраковки, используемую для теней игровых объектов. Если игровой объект находится в пределах заданного расстояния от камеры, то тени этого объекта отрисовываются, если же объект находится дальше этого расстояния, то тени такого объекта не отображаются (исключаются из отрисовки).

Shadow Distance:	0	1	5	10	15	25	50
Simple	124	114	96	92	82	77	73
Good	79	63	56	55	52	50	46
Beautiful	39	35	34	33	32	30	28
Fantastic	35	32	31	30	29	28	26

Изменение кадровой скорости при изменении значения параметра Shadow Distance в тестовом примере

Батчинг

<https://habrahabr.ru/company/intel/blog/254353/>



<i>Static Batching:</i>	<i>Off</i>	<i>On</i>
<i>FPS</i>	24	58
<i>Draw Calls</i>	5144	390

Разница в кадровой скорости и количестве вызовов рендеринга при включенном и отключенном статическом батчинге

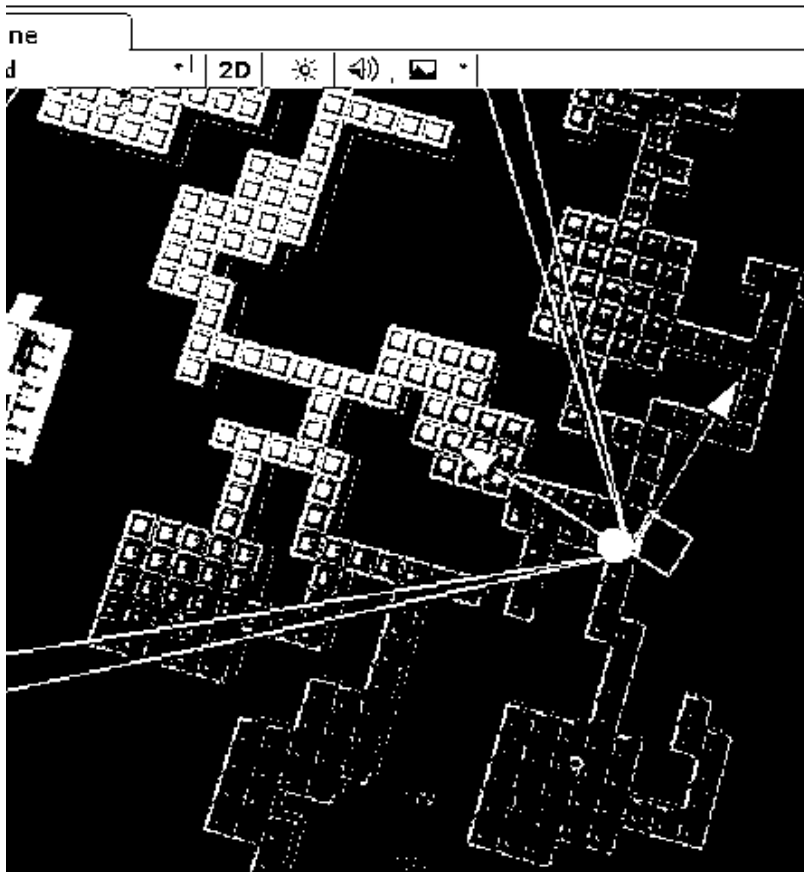
Расстояния отбраковки слоев (layerCullDistances)

Камера не будет отрисовывать игровые объекты, находящиеся за пределами плоскости отсечения в Unity. С помощью сценариев Unity можно задать для определенных слоев более короткое расстояние до плоскости отсечения.

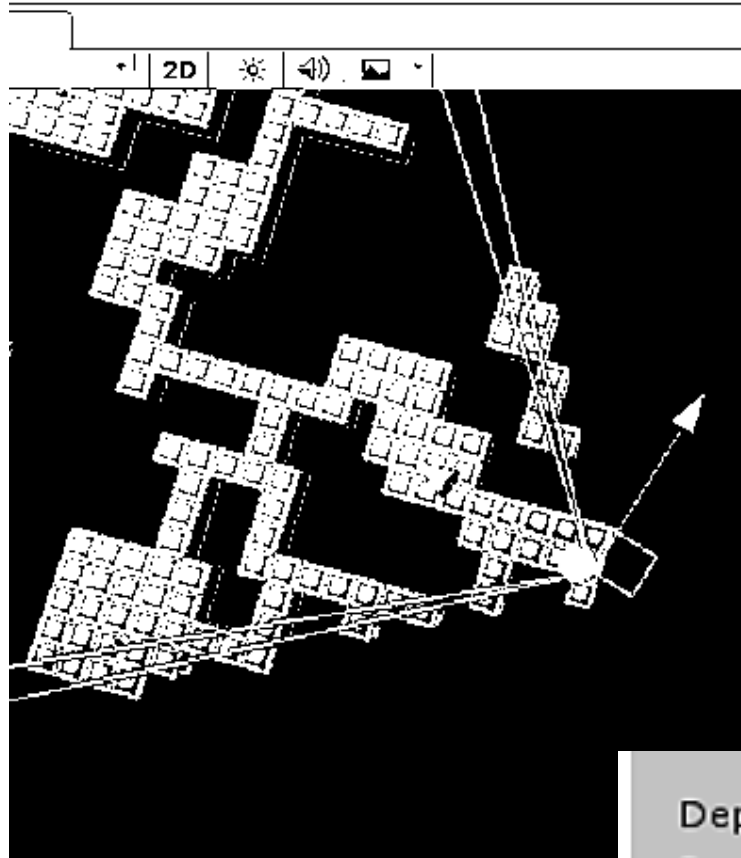
Настройка более короткого расстояния отбраковки для игровых объектов требует некоторой работы. Сначала нужно разместить объекты на слое. Затем нужно написать сценарий, чтобы изменить расстояние отбраковки этого конкретного слоя, и назначить сценарий камере. Образец сценария на рис. показывает, как создается массив 32 значений с плавающей запятой, соответствующий 32 доступным слоям. Если изменить значение индекса в этом массиве и присвоить его *camera.layerCullDistances*, то для соответствующего слоя изменится расстояние отбраковки. Если не назначить число индексу, то соответствующий слой будет использовать дальнюю плоскость отсечения камеры.

```
function Start () {  
    var distances = new float[32];  
    // Set up layer 10 to cull at 15 meters distance.  
    // All other layers use the far clip plane distance.  
    distances[10] = 15;  
    camera.layerCullDistances = distances;  
}
```

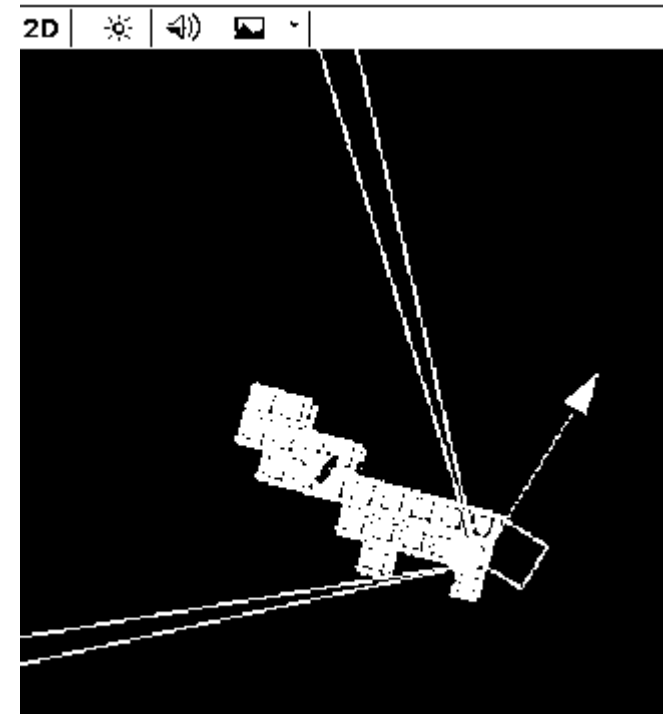
Настройка окклюзии



Frustum Culling



Occlusion Culling



<https://docs.unity3d.com/Manual/OcclusionCulling.html>

Depth	1
Rendering Path	Use Graphics Settings
Target Texture	minimap
Occlusion Culling	<input checked="" type="checkbox"/>
Allow HDR	<input type="checkbox"/>
Allow MSAA	<input checked="" type="checkbox"/>

Исключение заслоненных объектов (окклюзия)

Исключение заслоненных объектов — это отключение рендеринга объектов, скрытых за другими объектами. Это очень выгодно с точки зрения производительности, поскольку значительно сокращается объем информации, которую следует обработать.

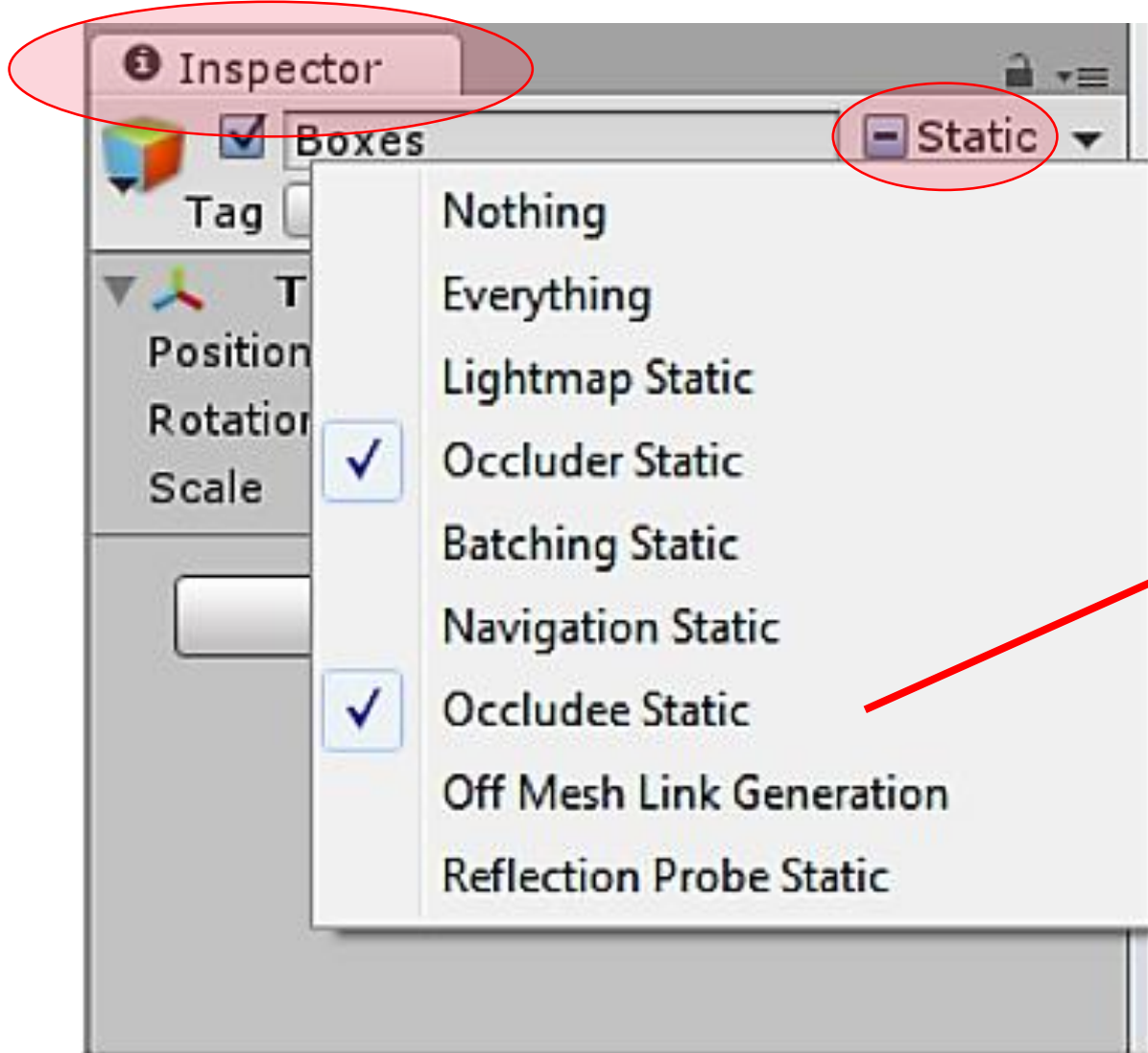
Заслоняющий объект, выступает в качестве преграды: все загороженные им объекты, помеченные как заслоняемые, не отрисовываются.

Заслоняемый объект не будет отрисовываться в Unity, если его загораживает заслоняющий объект.

Например, если пометить все объекты, находящиеся внутри дома, как заслоняемые, то сам дом можно пометить как заслоняющий. Если игровой персонаж будет находиться снаружи этого дома, то все объекты внутри дома, помеченные как заслоняемые, не будут отрисовываться. При этом ускорится обработка на ЦП и ГП.

Использование и настройка исключения заслоненных объектов задокументированы в Unity.

Настройка окклюзии



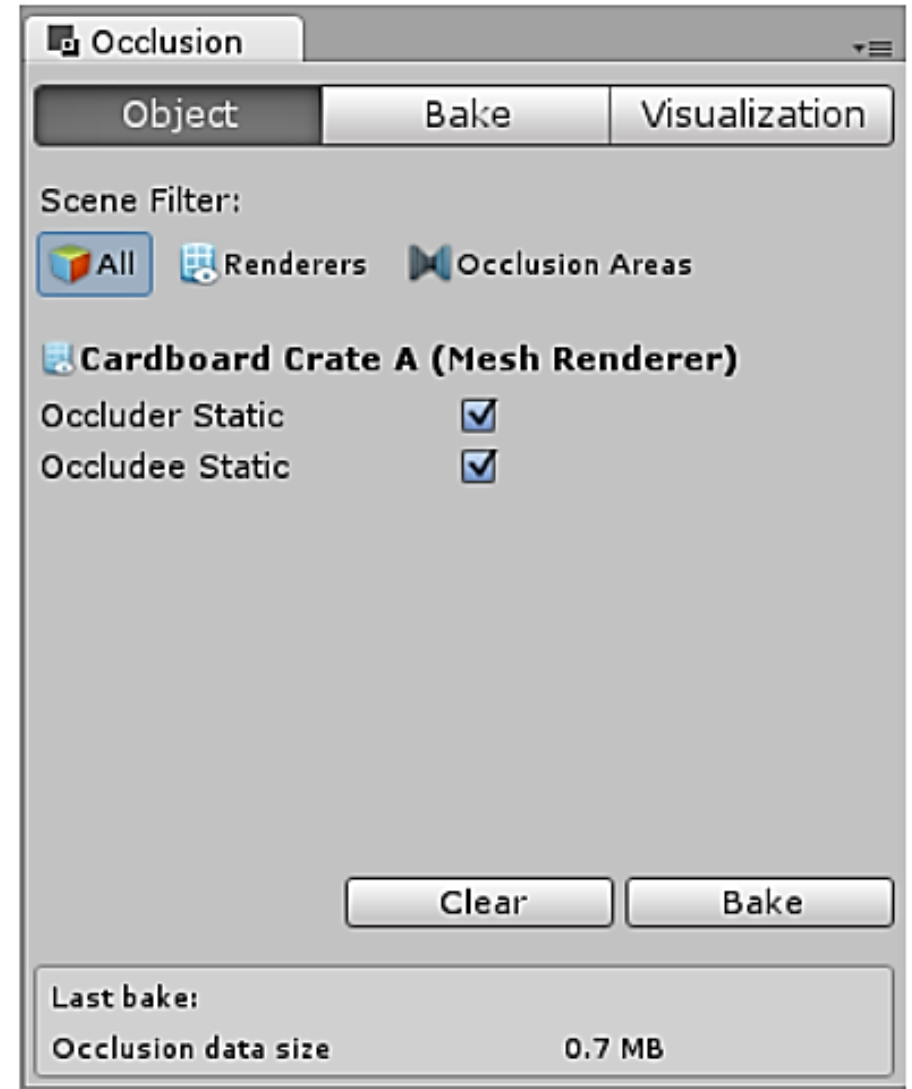
Для того, чтобы на объекты действовал occlusion culling, их нужно пометить тегом **Occluder Static** в Inspector.

Когда следует использовать **Occludee Static**?
Прозрачные объекты, которые не закрывают видимость, а также небольшие объекты, не перекрывающие других объектов, должны быть помечены как Occludees, а не Occluders. Это означает, что они не будут рендериться при закрытии их другими объектами. Однако, когда сами эти объекты закрывают видимость других, рендеринг выключаться не будет.

Occlusion Culling Window

Для большинства операций следует использовать окно Occlusion Culling (Window->Occlusion Culling)

В окне Occlusion Culling вы можете работать с мешами и Occlusion Areas(Областями Окклюзии).



Исключение заслоненных объектов (окклюзия)

<https://habrahabr.ru/company/intel/blog/254353/>



На изображении слева исключение заслоненных объектов отключено, отрисовываются все объекты, расположенные за стеной, поэтому кадровая скорость составляет **31 кадр** в секунду. На изображении справа исключение заслоненных объектов включено, объекты, заслоненные стеной, не отрисовываются, поэтому скорость возросла до **126 кадров** в секунду.

Уровень детализации LOD (Level of Detail)

Упрощение или удаление объектов по мере их удаления от камеры. С помощью уровня детализации (LOD) можно присвоить одному игровому объекту несколько моделей разной сложности и переключаться между ними в зависимости от расстояния до камеры.

+ повышает производительность

- увеличивает работу художников



Наилучшее качество (Level 0)




Среднее качество (Level 1)



Низкое качество (Level 2)

<i>LOD</i>	<i>level 0</i>	<i>level 1</i>	<i>level 2</i>
<i>FPS</i>	160	186	240

 EverydayTools 23 января в 16:10 Разработка

Оптимизация механики и графики в игре жанра «симулятор» на iOS

<https://habrahabr.ru/company/everydaytools/blog/319990/>

Проблема, связанная с движением, заключалась в соотношении расстояния и метрики в Unity. Если путь поезда всегда будет генерироваться блоками непрерывно вперед, рано или поздно объект окажется в таких координатах, что адекватный просчет кадров будет невозможен.



Кабина поезда, состоящая из нескольких Mesh'ей, начинала в буквальном смысле этого слова трястись, когда координаты достигали слишком высоких значений.

Исходя из этого, решили дополнить движок игры **респауном** — иными словами, сделать так, чтобы, приезжая на станцию, поезд возвращался вместе с ней и ближайшими отстроенными блоками тоннеля в нулевую точку координат.