

Оптимизация

Оптимизация производительности графики. Батчинг вызовов отрисовки (Draw Call Batching). Моделирование персонажей для оптимальной производительности. Окно Rendering Statistics.

Оптимизация с использованием системы LOD.

Камера

С помощью параметра **Render Path** можно указать в Unity, как обрабатывать рендеринг света и теней в игре. В Unity поддерживаются три типа рендеринга, вот они в порядке от наиболее ресурсоемкого к наименее ресурсоемкому: **Deferred (отложенный, только в Unity Pro)**, **Forward (заблаговременный)** и **Vertex Lit (освещение вертексов)**. В каждом случае тени и свет обрабатываются немного иначе, и для их обработки требуется разный объем ресурсов ЦП и ГП.

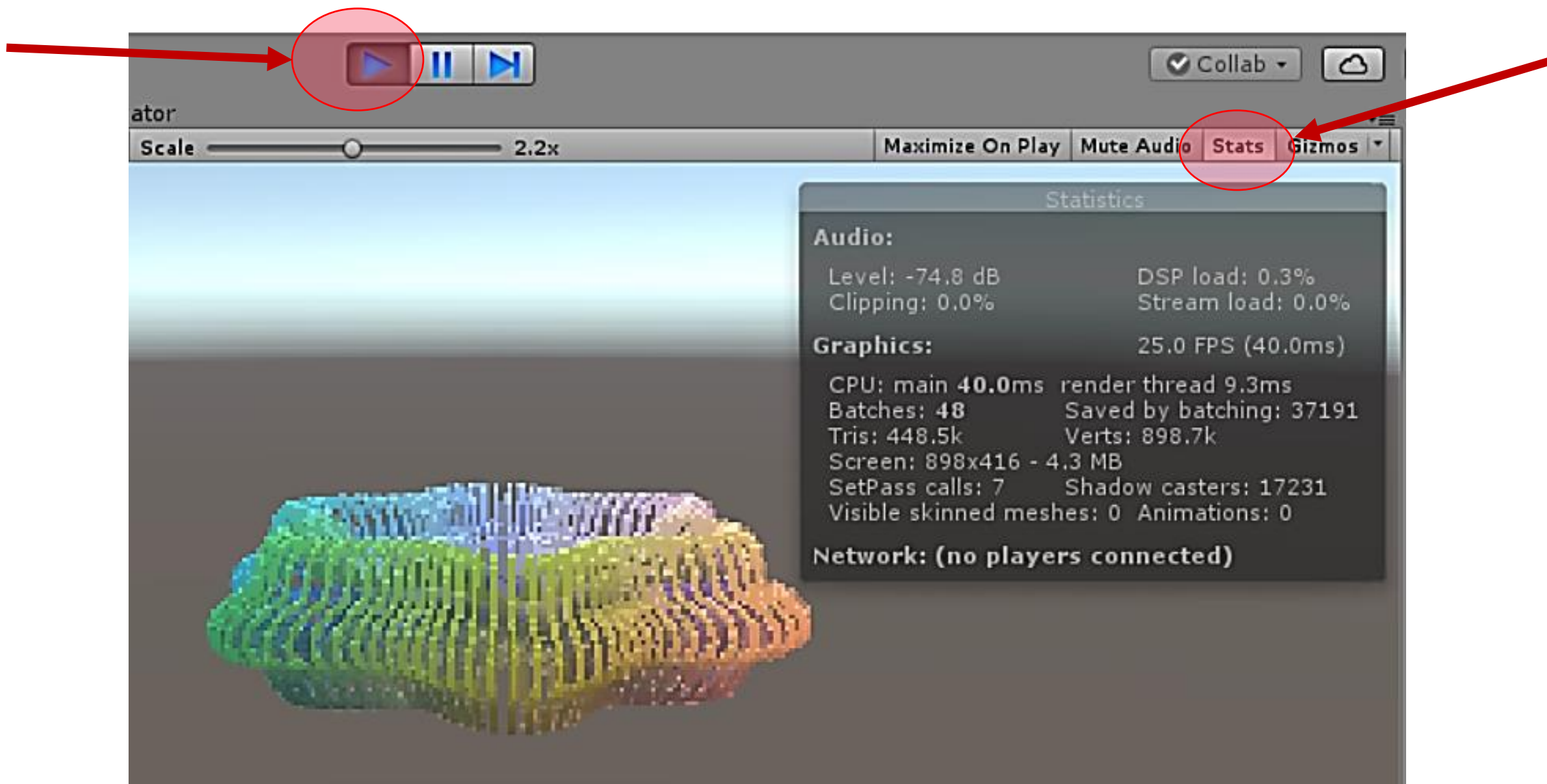
Если выбрать рендерер, который не поддерживается графическим адаптером, **Unity автоматически переключится на менее ресурсоемкий способ рендеринга.**

Разработчики должны знать, целевую аудиторию своей игры и платформу, на которой потенциальные пользователи будут играть в эту игру. Это поможет выбрать соответствующий способ рендеринга для данной платформы. Например, если игра содержит несколько источников света и графических эффектов, использующих отложенный рендеринг, то на компьютерах с маломощным графическим адаптером играть в такую игру будет попросту невозможно.

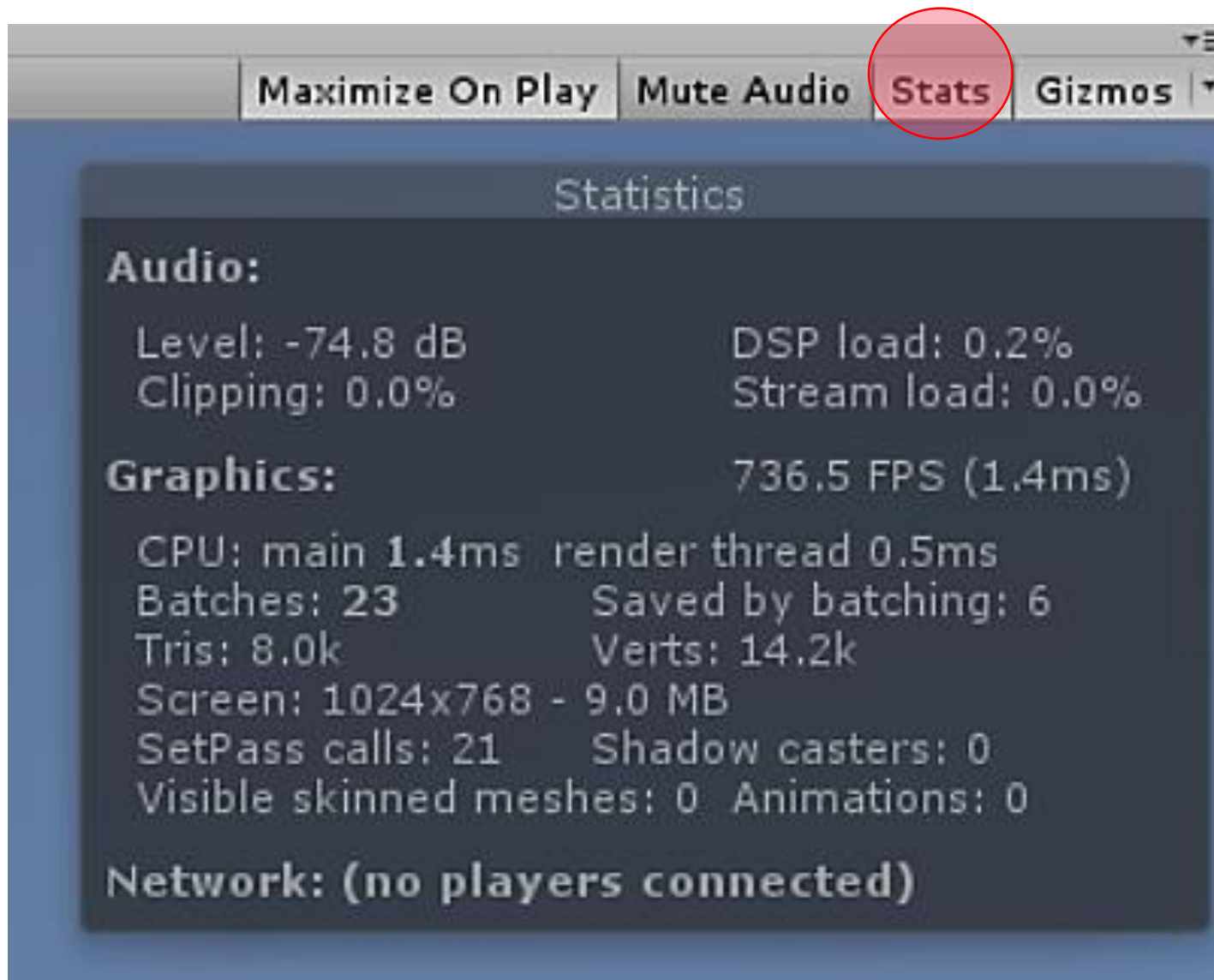
Какова стоимость графики

Графическая часть вашего 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.

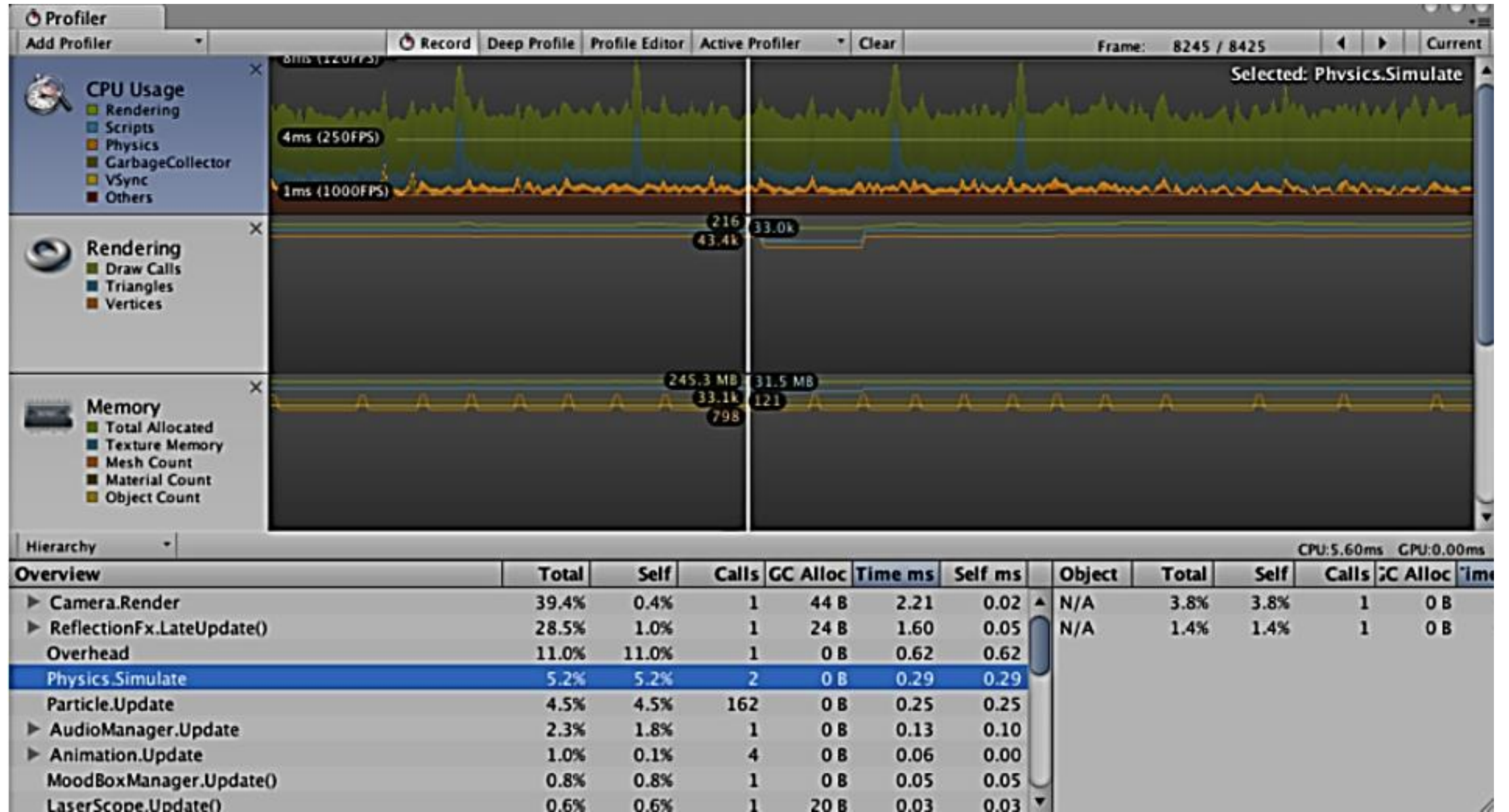
Окно Rendering Statistics отображает статистику рендеринга в реальном времени, полезно для оптимизации производительности.

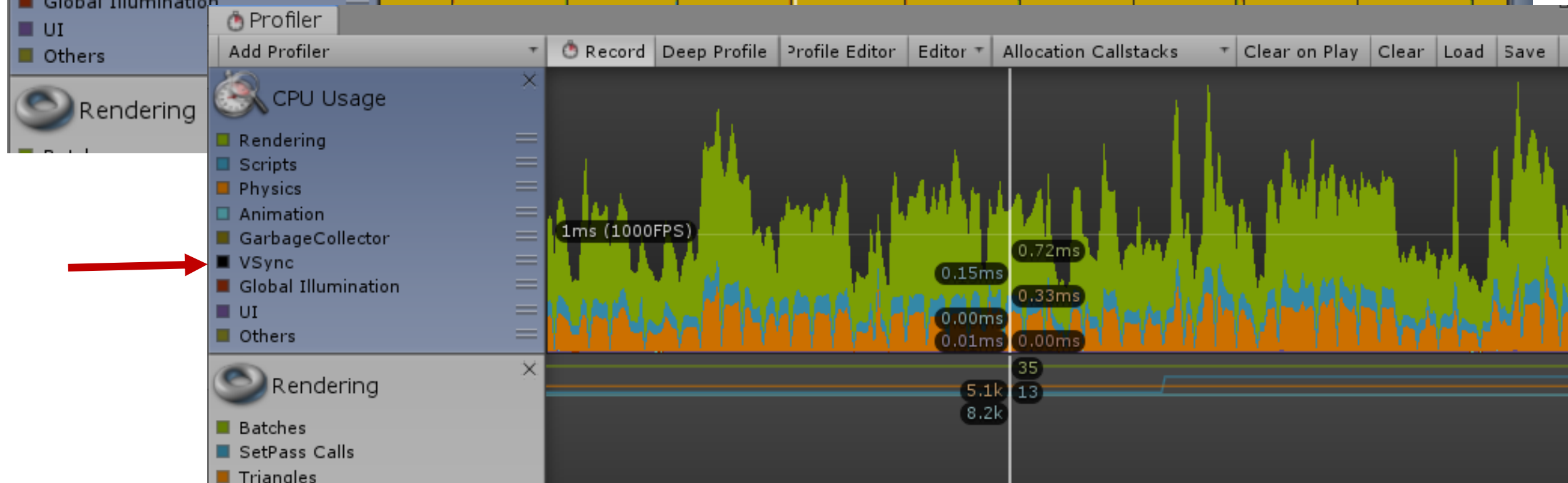
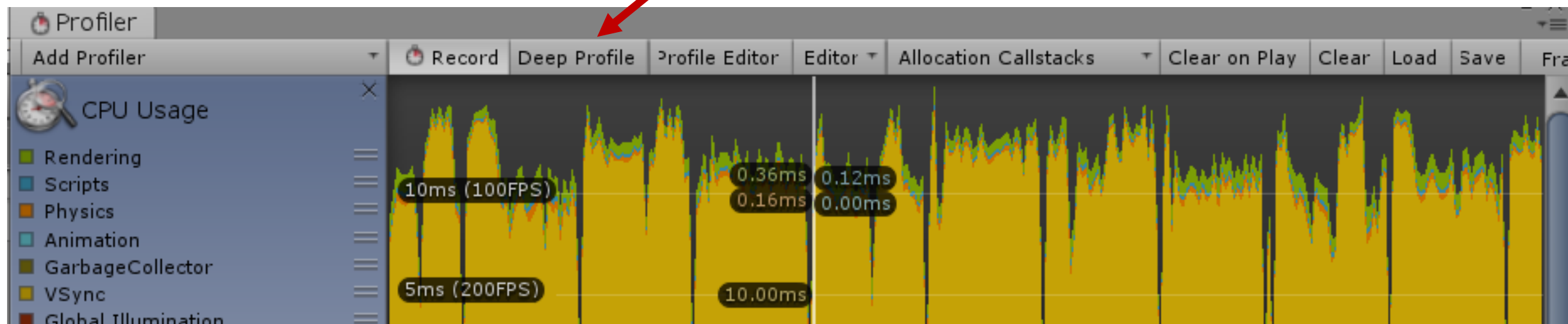


Окно Rendering Statistics отображает статистику рендеринга в реальном времени, полезно для оптимизации производительности. Содержимое панели может быть разным, в зависимости от целевой платформы.



The Profiler Window





Типичные узкие места и их проверка:

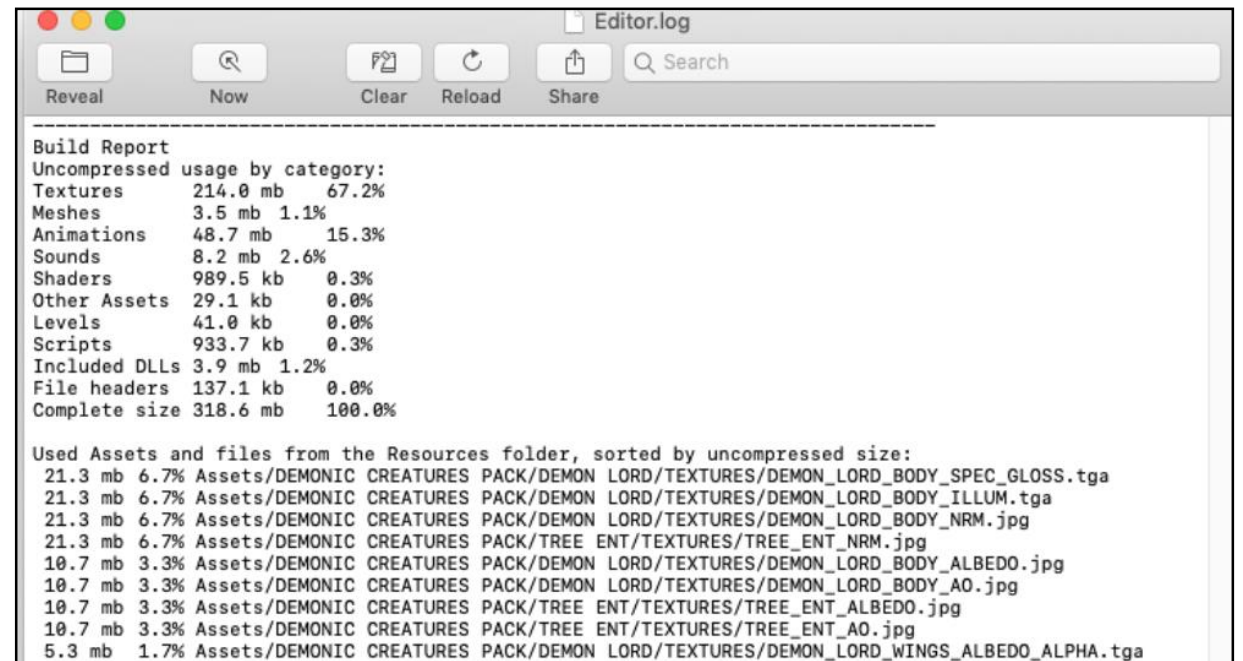
- GPU часто ограничен филлрейтом (fillrate) или пропускной способностью памяти.
- Запуск игры с более низким разрешением экрана увеличивает производительность? Тогда вы скорее всего ограничены филлрейтом GPU.
- CPU часто ограничен количеством вещей, которые должны быть отрисованы, также известно, как “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. Проблема может быть, к примеру, в скриптах или физике. Используйте **профайлер** для поиска источника проблемы.

Журнал редактора

Первым 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:












Настройки качества

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.






Levels   







Very Low	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Low	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Medium	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
High	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Very High	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Ultra	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	




Default   

Add Quality

- Very Low
- Low
- ☒ Medium
- High
- Very High
- Ultra

Levels   

Very Low	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Low	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Medium	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
High	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Very High	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Ultra	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

Default   

Add Quality Level

Name

Rendering

Pixel Light Count

Texture Quality

Anisotropic Textures

Anti Aliasing

Soft Particles ☐

Realtime Reflection ☒

Billboards Face Cam ☒

Resolution Scaling

Shadows

Shadows

Shadow Resolution

Shadow Projection

Shadow Distance

Shadowmask Mode

Shadow Near Plane

Shadow Cascades

Cascade splits

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).

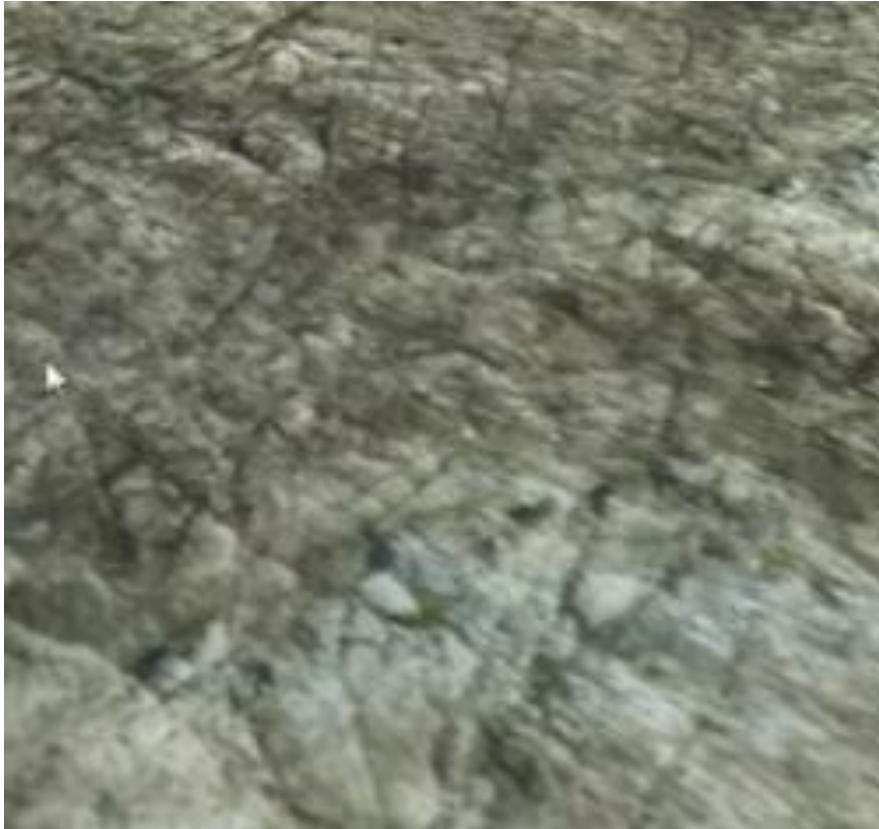
AntiAliasing устанавливает уровень сглаживания, который будет использоваться.

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.

[Далее](#)

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

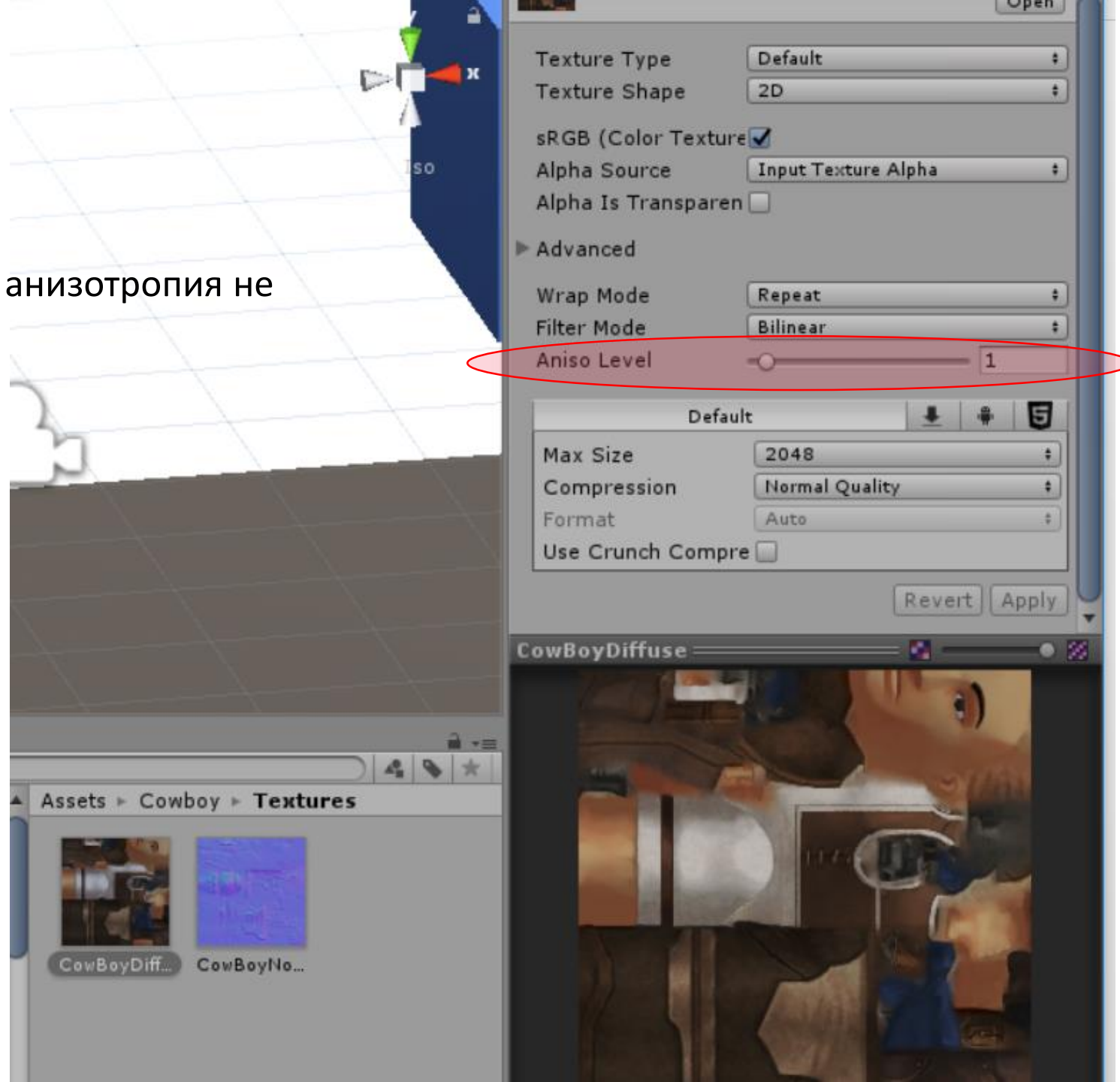


Полный размер



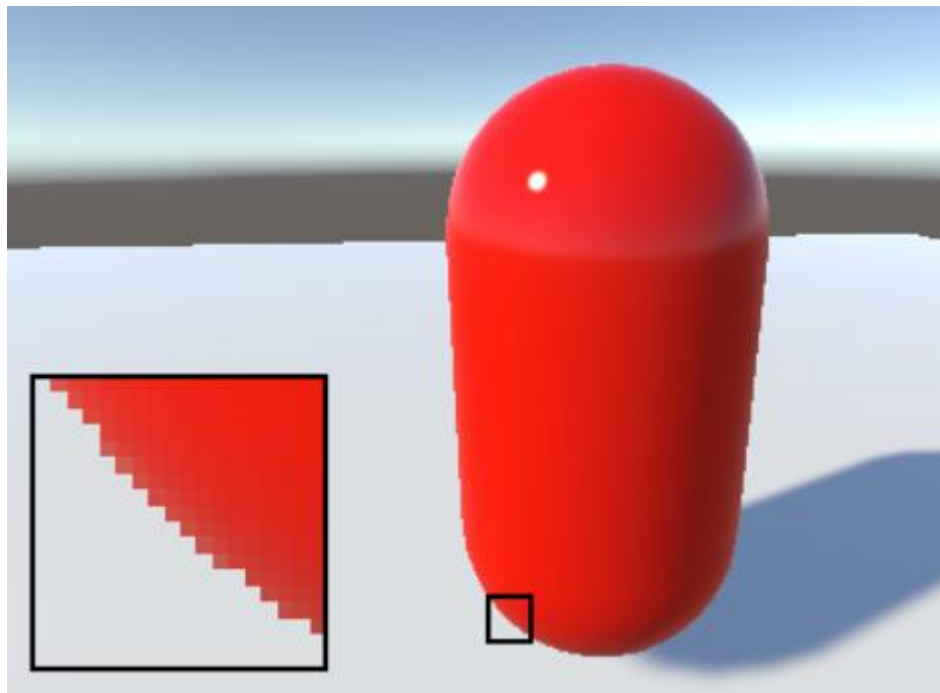
1/8 от размера

Если стоит ноль, анизотропия не применяется

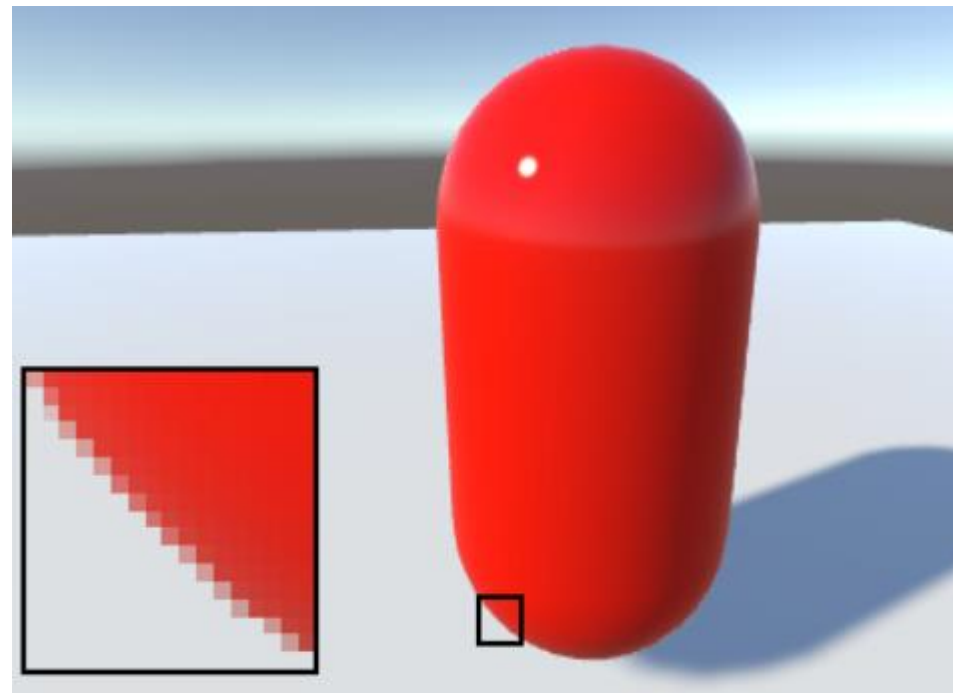


[назад](#)

AntiAliasing устанавливает уровень сглаживания

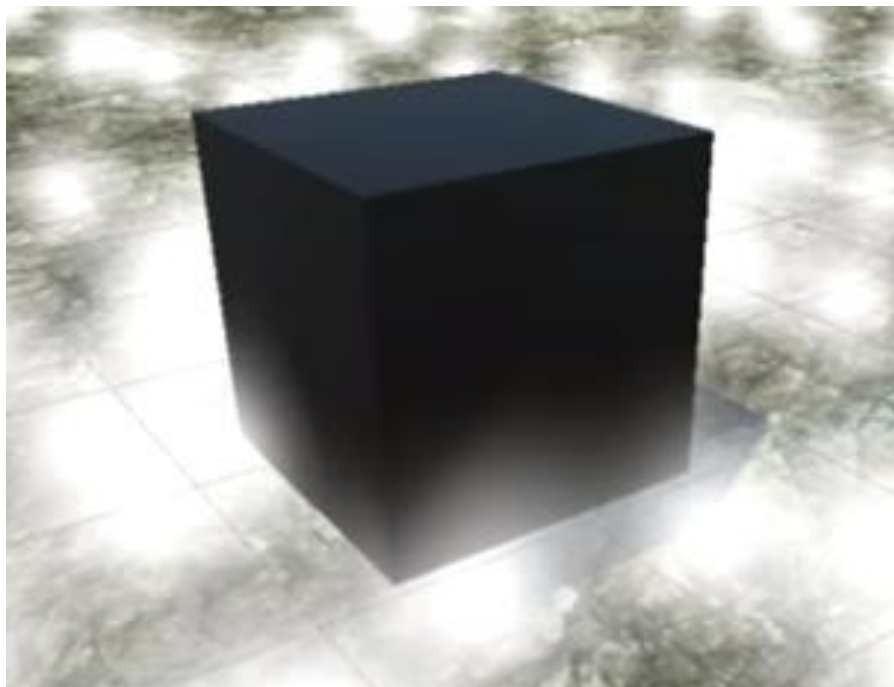


Disabled

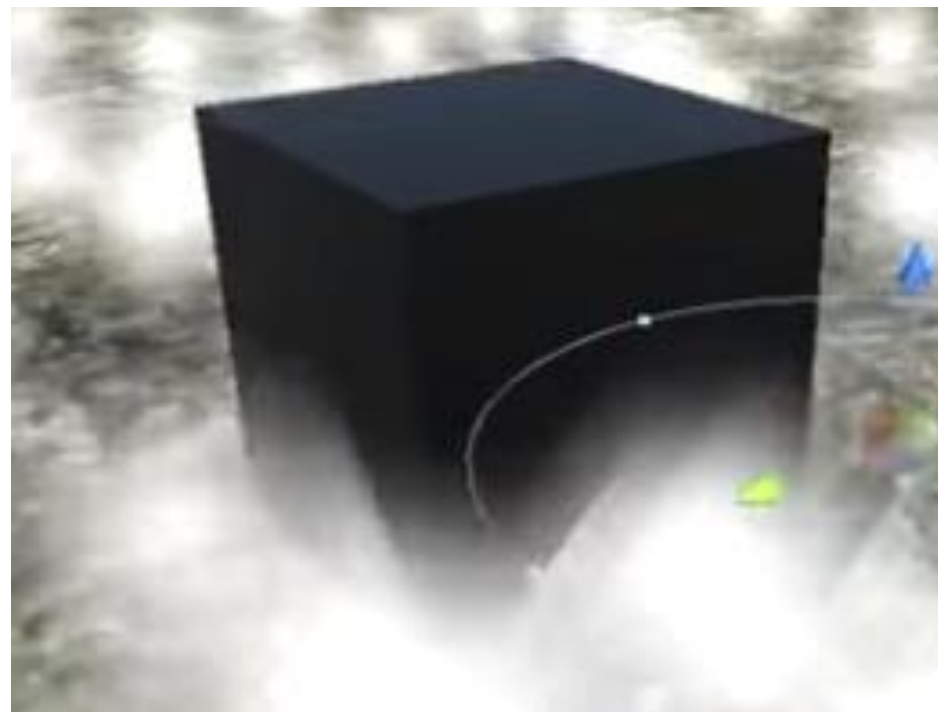


8x

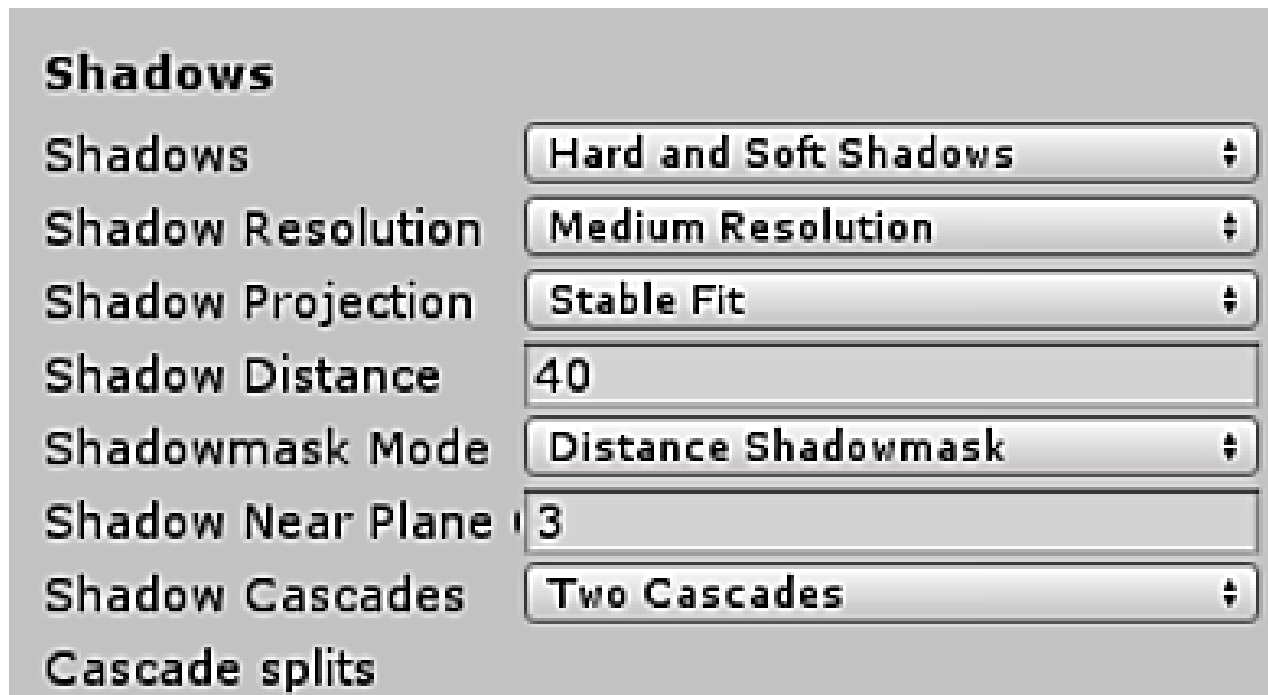
Soft Particles нагружает систему, не всегда целесообразно для мобильных устройств



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 и Очень Высокий*.

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

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

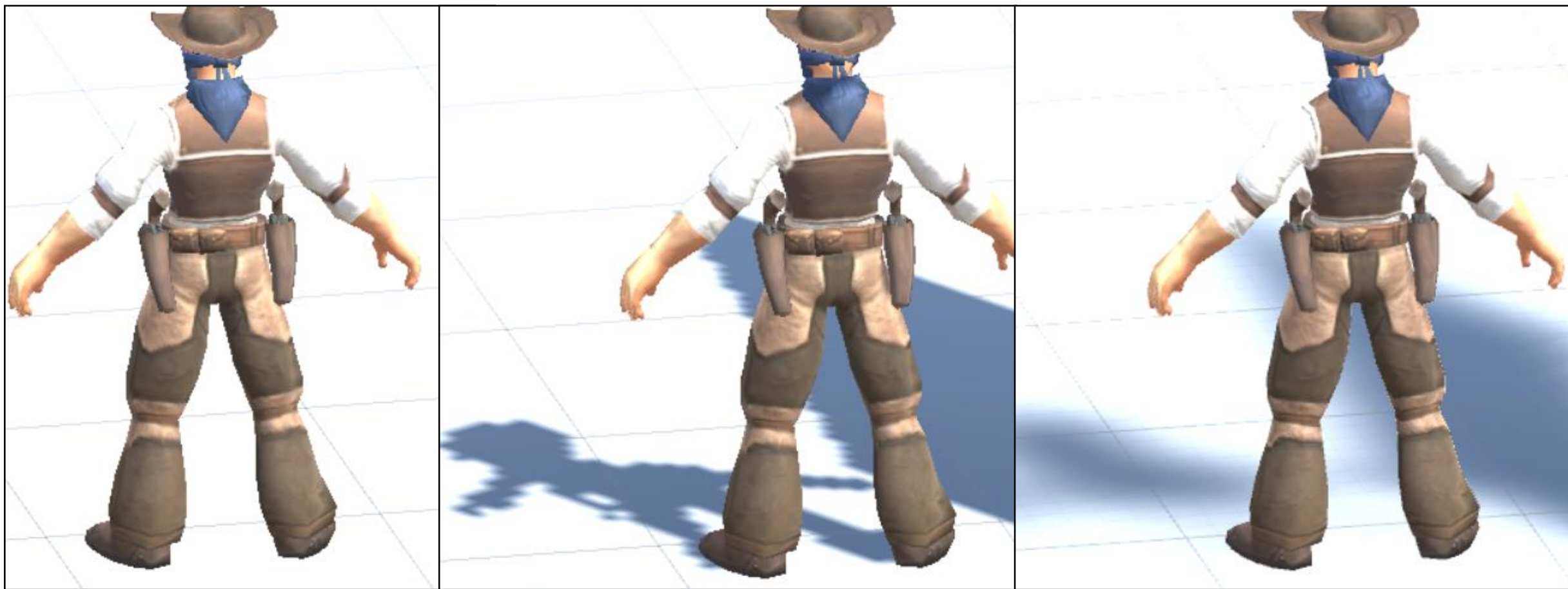
Shadowmask Mode *Distance Shadowmask* использует тени в реальном времени до Shadow Distance и запеченные тени за его пределами.

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.

[Далее](#)



[Назад](#)

Низкое разрешение



высокое



[Назад](#)

количество каскадов: нет



количество каскадов 4



[Назад](#)

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.

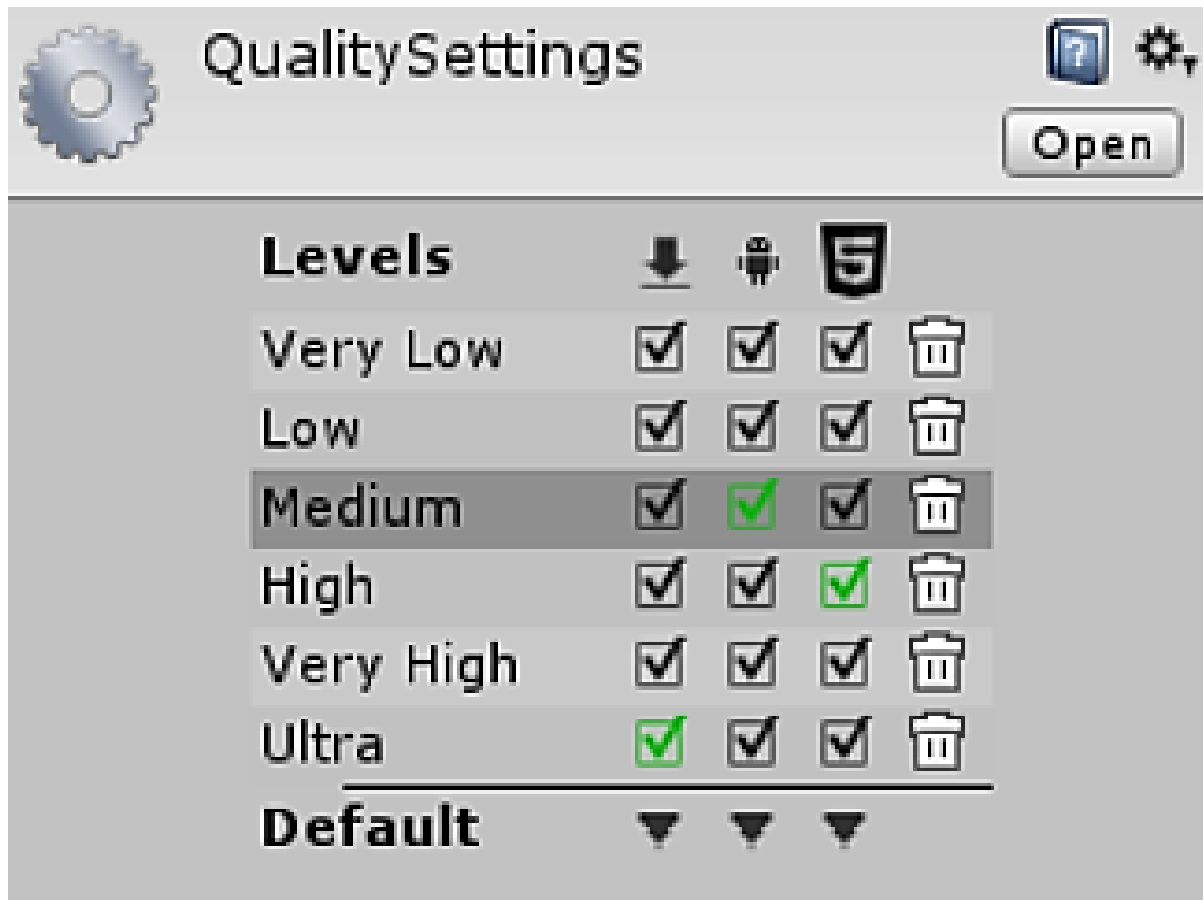
VSync Count синхронизация с частотой обновления устройства, чтобы избежать «[разрыва](#)» артефактов

LOD Bias чем больше, тем лучше от 0 до 2

Particle Raycast Budget Максимальное количество raycasts для использования при обнаружениях столкновений частиц ([см.](#))

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

[Далее](#)



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

4

64

4096

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.



[Назад](#)

Батчинг вызовов отрисовки (Draw Call Batching)

Для отрисовки объекта на экране движок отправляет команду (draw call) графическому API (например, OpenGL или Direct3D). Графический API производит значительную работу для каждого DC, что сильно влияет на производительность CPU.

- **Static Batching:** объединение статических (то есть не движущихся) объектов в большие мэши. Статичный батчинг позволяет движку снизить количество DC для геометрии любого размера, если она не двигается и использует общий материал. Статичный батчинг более эффективен, чем динамический.
- **Dynamic Batching:** для достаточно небольших мешей, сгруппируйте много похожих друг с другом и рисуйте за один раз. Динамический батчинг применяется автоматически и не требует дополнительных действий.

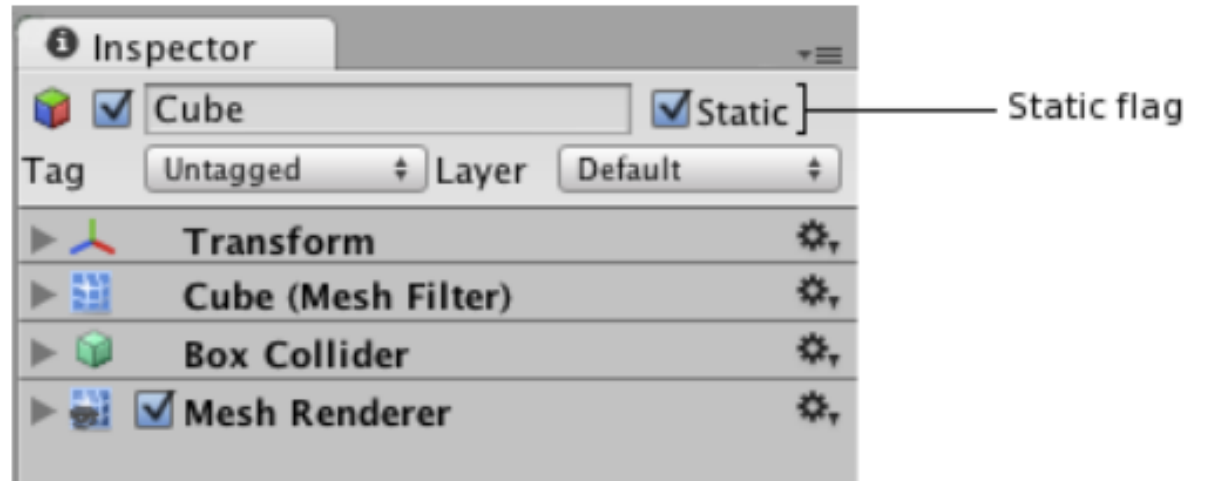
На данный момент, только **Mesh Renderers** и **Particle Systems** могут батчиться.

Батчатся только объекты, имеющие один и тот же материал. Соответственно, если это возможно необходимо делать материалы общими для множества объектов.

Если у вас есть два одинаковых материала, отличающихся только текстурами, можно объединить эти текстуры в одну большую — процесс часто называемый созданием текстурного атласа.

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.

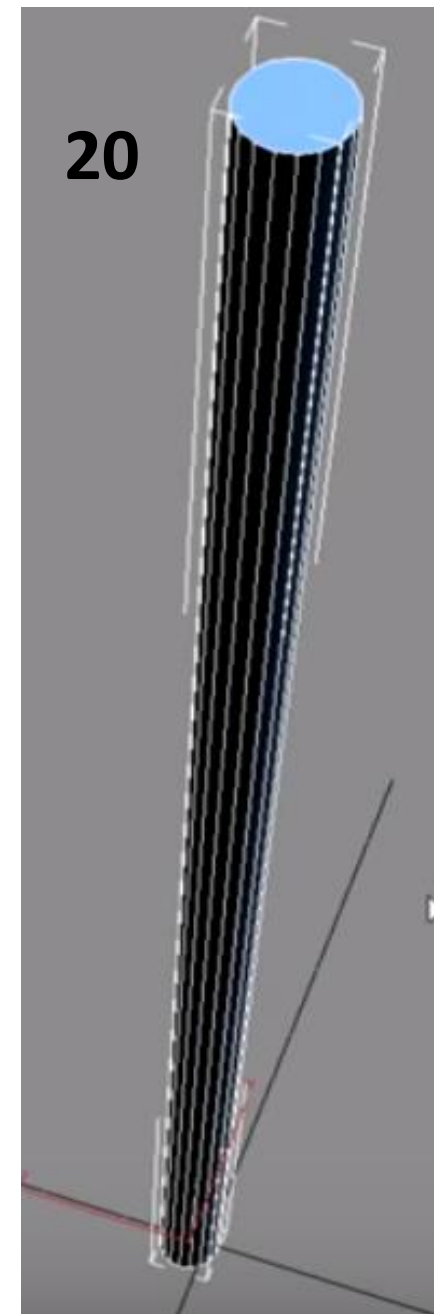
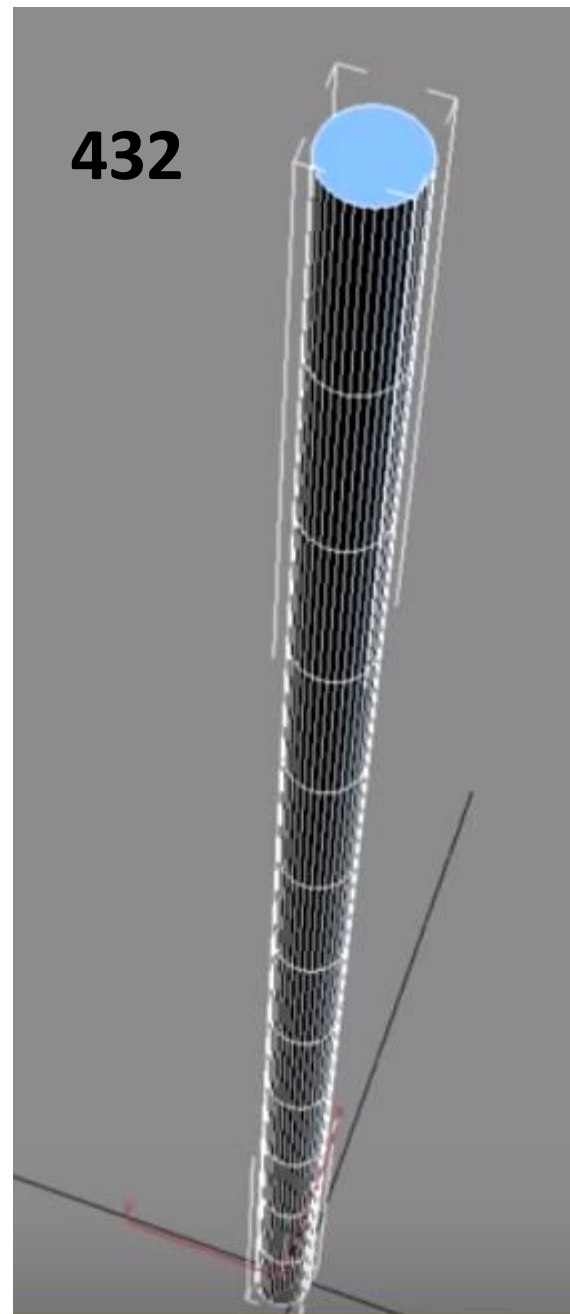
Динамический батчинг (Dynamic Batching)

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 с вашей стороны.

- Динамический батчинг связан с дополнительной нагрузкой для **каждой вершины**, так что он применим только к мешам, содержащим менее **900** вершин в сумме.
- Если ваш шейдер использует Vertex Position, Normal и единственный UV, то вы можете батчить до 300 вершин; тогда как, если шейдер использует Vertex Position, Normal, UV0, UV1 и Tangent, то только 180 вершин.
- 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.

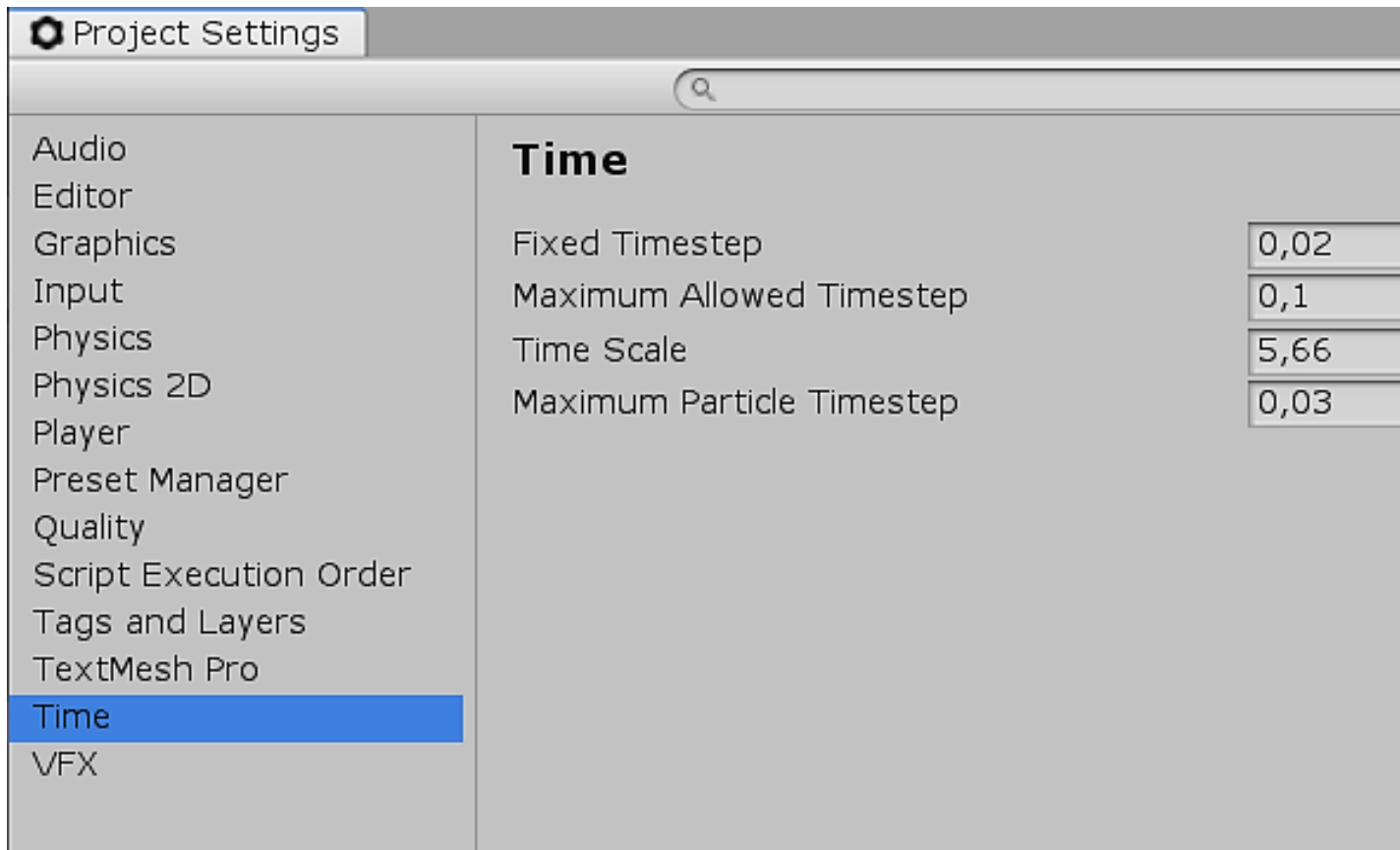
Советы по оптимизации графики

1. Полигонаж
2. Лучше если мелкие объекты монолитные
3. Текстуры (использование атласов текстур)



Оптимизация физики

Edit >> Project Settings >> Time

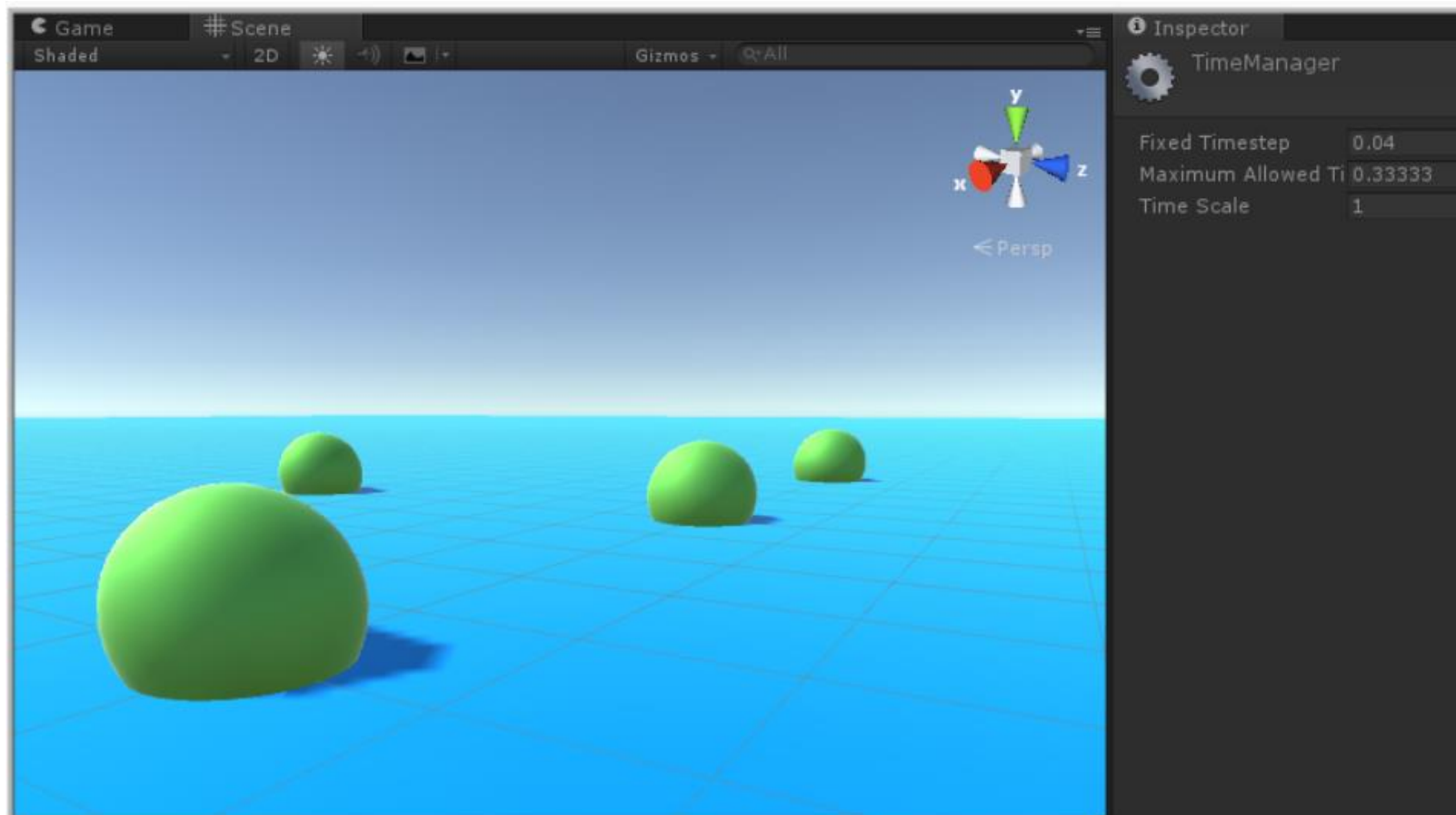


FIXED TIMESTEP

Значение по умолчанию - 0,02 (в секундах) , это означает , что каждые 20 мс будет выполняться обновление физики.

Все *FixedUpdate ()* также будут вызываться каждые 20 мс.

Но если ваша игра не сильно зависит от физики, вы всегда можете увеличить TimeStep, чтобы получить лучшие результаты. (т.е. уменьшить физические вызовы)



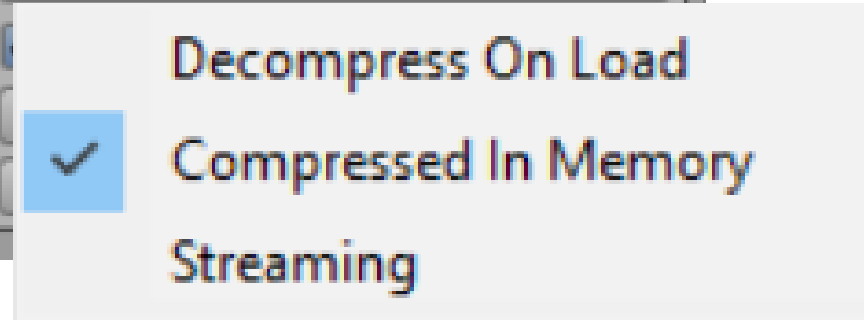
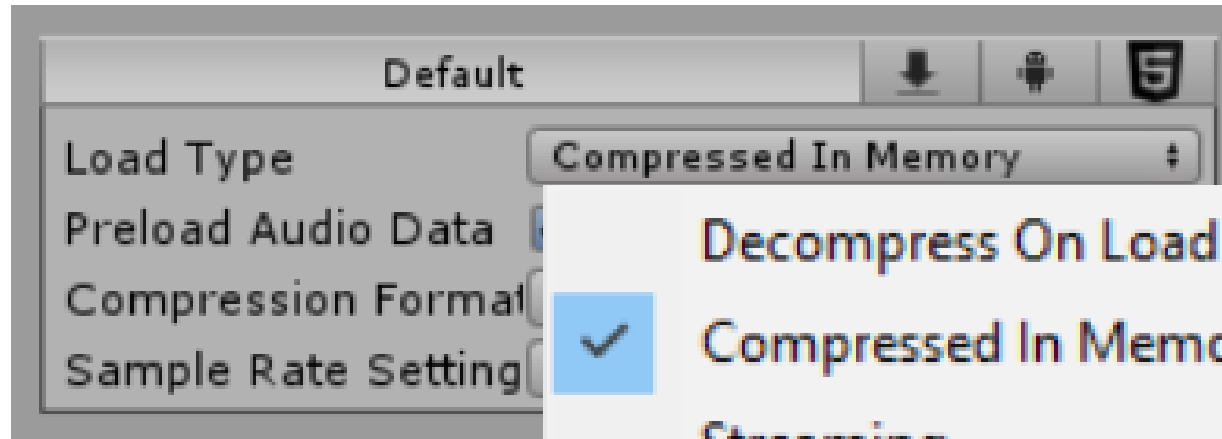
НО!!! Например, при $\text{FIXED TIMESTEP}=0.1$, сферы пролетят через поверхность, а при 0.04 (см рис.) физическое столкновение будет обнаружено с небольшим опозданием и будет возвращаться в норму только после обновления кадра. Это может быть не видно при нормальном FPS в вашем игровом процессе, но это может повлиять на физику.

Советы по оптимизации Физики от Unity

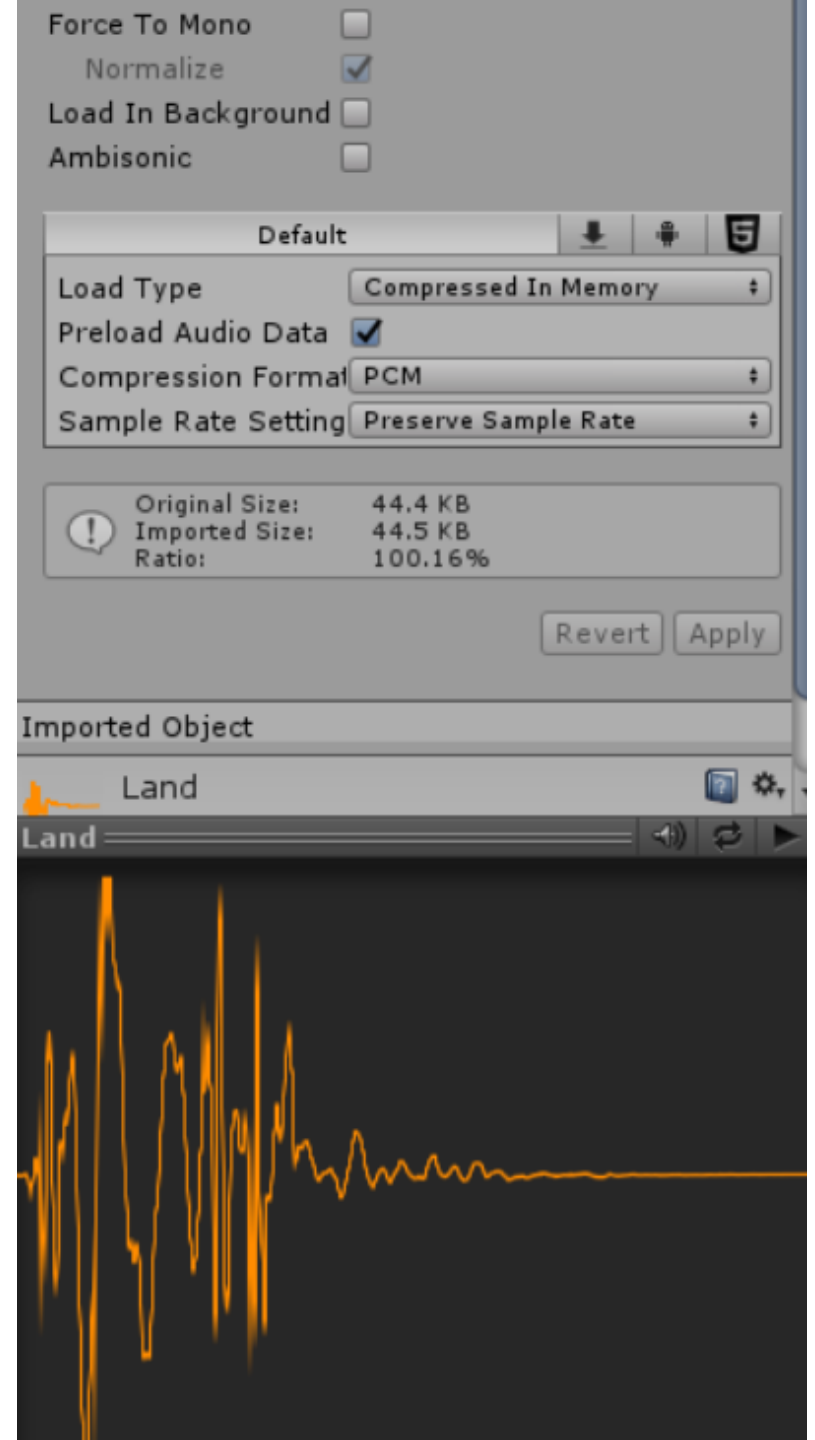
<https://docs.unity3d.com/ru/current/Manual/MobileOptimisation.html>

- 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 звука



- Находится в оперативной памяти
- Тоже но в сжатом виде
- Проигрывается с жесткого диска



Статьи с анализом



saul 31 марта 2015 в 08:52 Разработка

Планирование оптимизации с Unity

Качество текстур

Изменение кадровой скорости при переключении между различным качеством текстур в 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

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;  
}
```

Исключение заслоненных объектов

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.

Исключение заслоненных объектов (окклюзия)

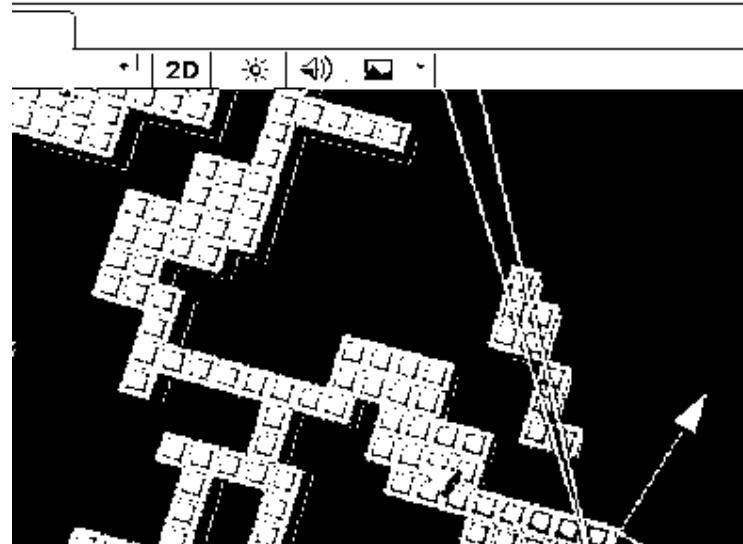
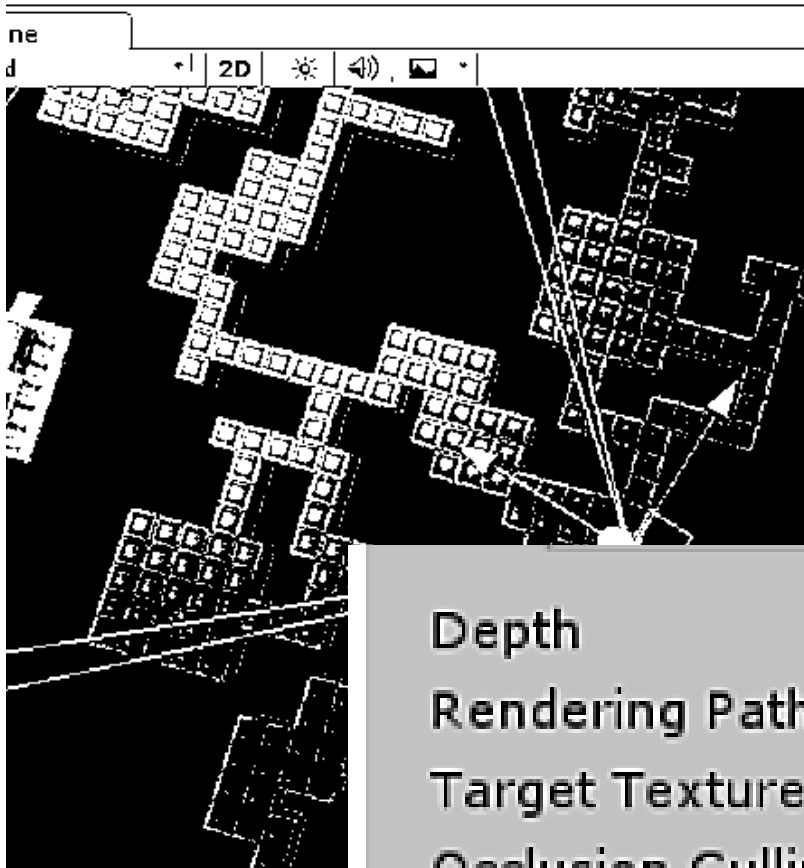
<https://habrahabr.ru/company/intel/blog/254353/>



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.

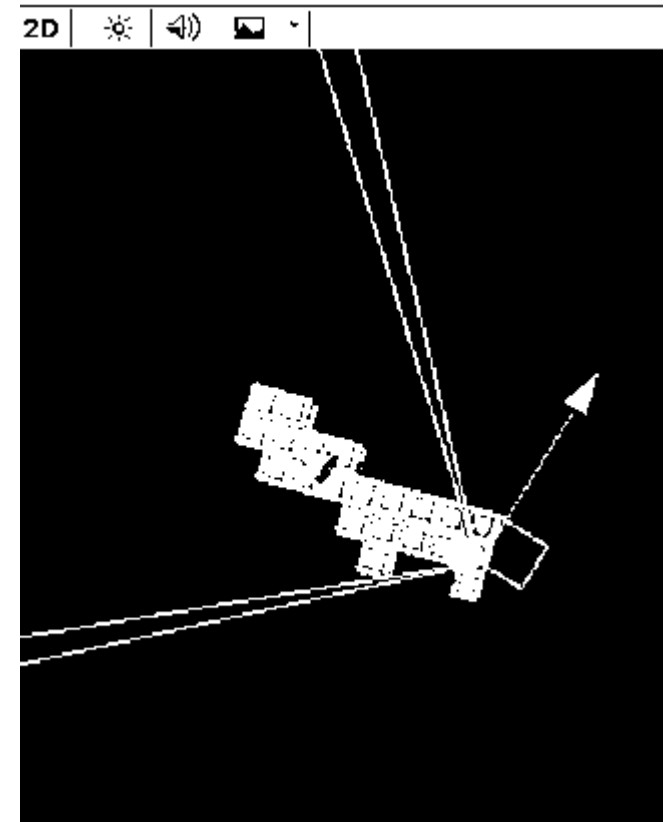
Настройка окклюзии

Frustum Culling

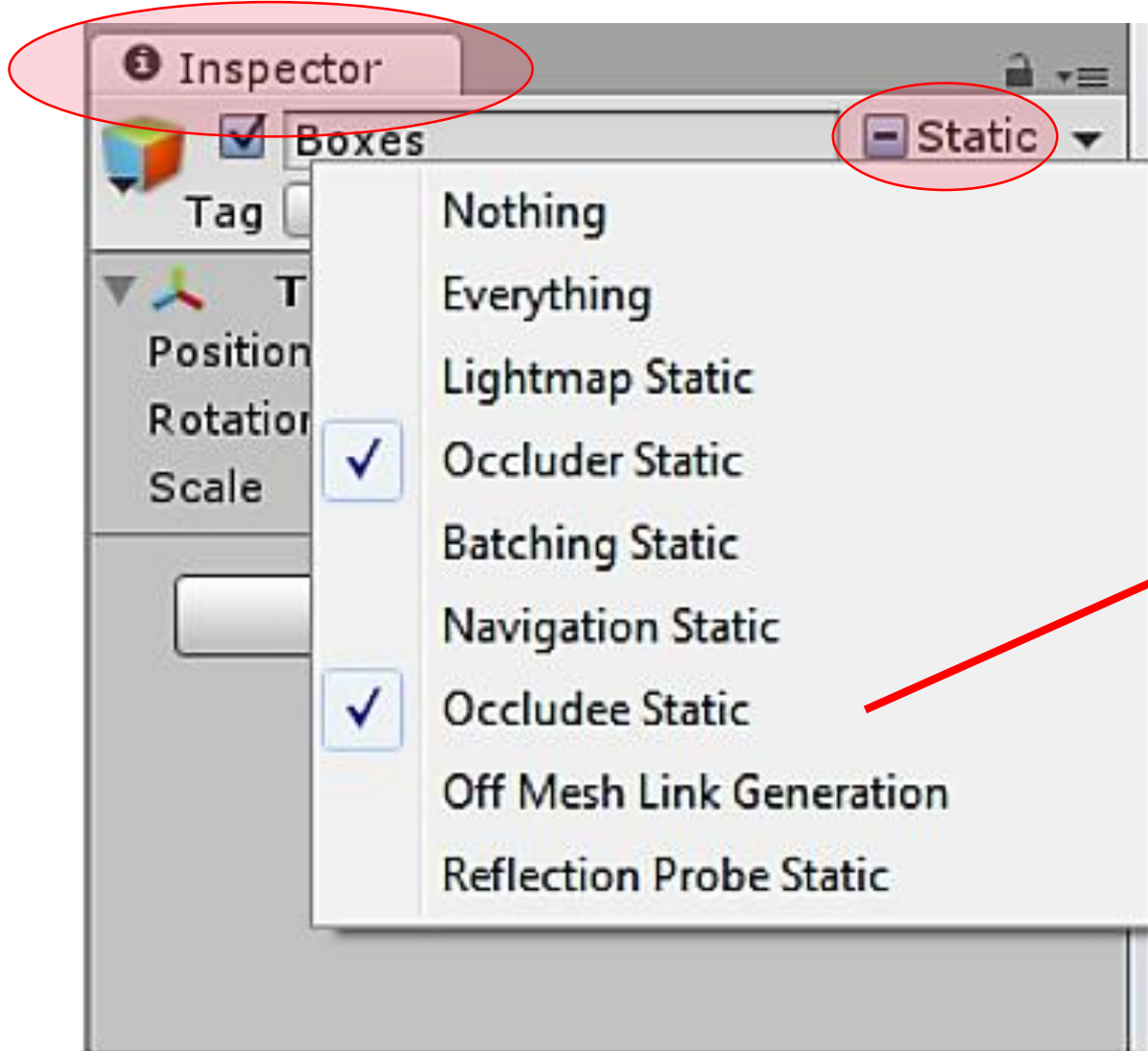


Depth	1
Rendering Path	Use Graphics Settings
Target Texture	minimap
<u>Occlusion Culling</u>	<input checked="" type="checkbox"/>
Allow HDR	<input type="checkbox"/>
Allow MSAA	<input checked="" type="checkbox"/>

Occlusion Culling



Настройка окклюзии



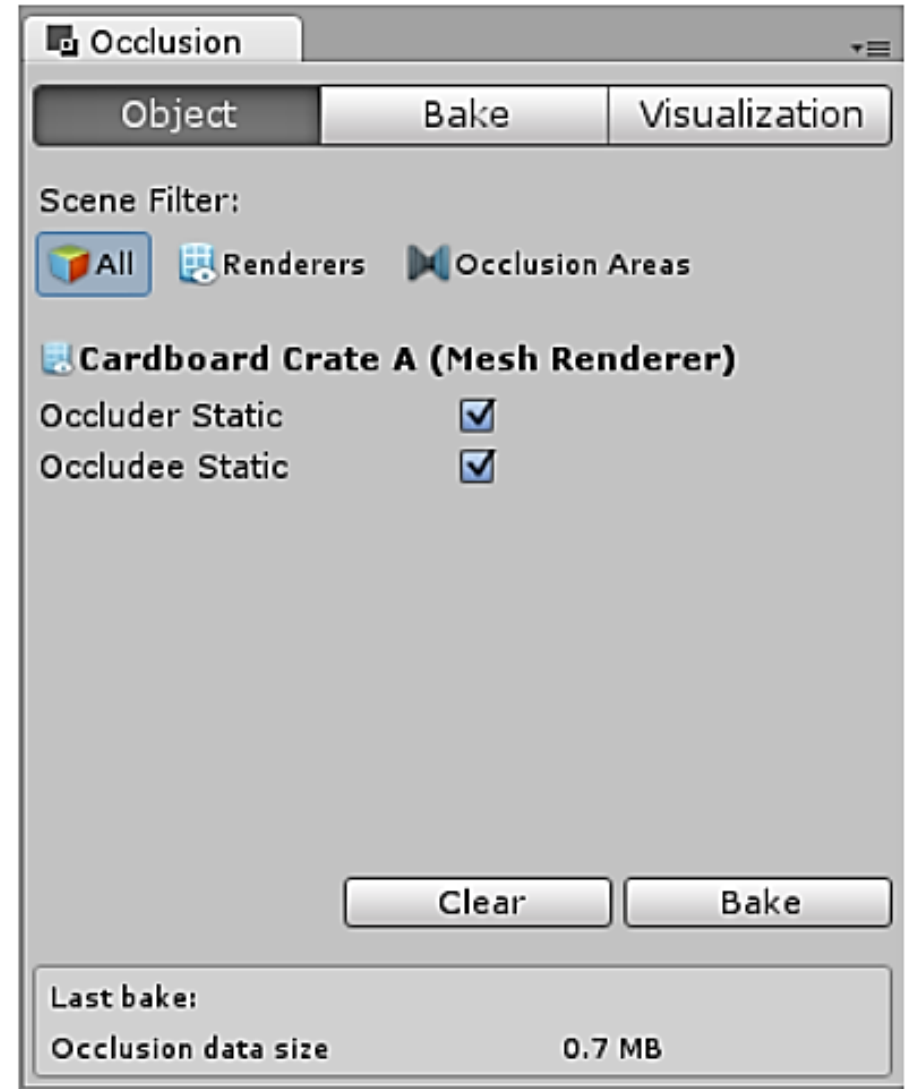
Для того, чтобы на объекты действовал occlusion culling, их нужно пометить тегом **Occluder Static** в Inspector.

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).



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)

LOD	Level 0	Level 1	Level 2
FPS	160	186	240

Пакетная обработка

Чем больше на экране объектов, тем больше вызовов рендеринга нужно сделать. В Unity поддерживается так называемая пакетная обработка, позволяющая поместить несколько игровых объектов в один вызов рендеринга. **Статическая пакетная обработка предназначена для статических объектов, а динамическая — для движущихся объектов.**

Динамическая пакетная обработка выполняется автоматически при выполнении всех требований, а статическую пакетную обработку требуется задавать вручную. Для получения наибольшего преимущества от пакетной обработки старайтесь объединить как можно больше объектов в пакет для одного вызова рендеринга.

Сведения о том, какие объекты подходят для динамической или статической пакетной обработки, см. в документации Unity.

Пакетная обработка



<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 включенной и отключенной статической пакетной обработке



CooperMaster 26 января 2016 в 16:38 Разработка

Оптимизация Android-игр, созданных на Unity для платформы Intel: пример из жизни

Оптимизация Android-игр, созданных на Unity для платформы Intel: пример из жизни

<https://habrahabr.ru/company/intel/blog/275927/>



У компании Innospark, разработчика Hero Sky: Epic Guild Wars, имеется большой опыт в создании мобильных игр с использованием различных коммерческих игровых движков. Hero Sky: Epic Guild Wars – это первая игра компании, разработанная с использованием Unity и выпущенная на всемирном рынке. После публикации в Google Play, с ростом числа загрузок, компания начала сталкиваться с жалобами пользователей. На некоторых Android-устройствах, основанных на платформе Intel, игра попросту не работала, на других её производительность оставляла желать лучшего. В итоге в компании было принято решение портировать игру на платформу x86 и оптимизировать её.

Для поиска узких мест в производительности в процессе разработки использовали Intel GPA . Данные анализа приложения применялись для решения проблем с графикой и улучшения производительности игры.



После того, как игра была портирована для архитектуры x86, нагрузка на процессор снизилась примерно на 7,1%, FPS выросла на 27,8%, а время исполнения уменьшилось примерно на 32,6%. Однако, нагрузка на видеоядро выросла на 26,7% из-за роста частоты кадров.

Вот график, на котором сравнивается производительность нативного и эмулированного кода на устройстве.



Выигрыш в производительности, которого можно достичь благодаря поддержке x86

Устранение ненужного альфа-смешивания

Когда при отображении объектов используется альфа-смешивание, программа должна скомбинировать, в реальном времени, цветовые значения всех перекрывающихся объектов и фона для того, чтобы выяснить, каким будет итоговый цвет. Таким образом, вывод цветов, получаемых в результате альфа-смешивания, может оказать большую нагрузку на процессор, нежели отображение непрозрачных цветов. Эти дополнительные вычисления способны повредить производительности на медленных устройствах. Поэтому было решено избавиться от ненужного альфа-смешивания.

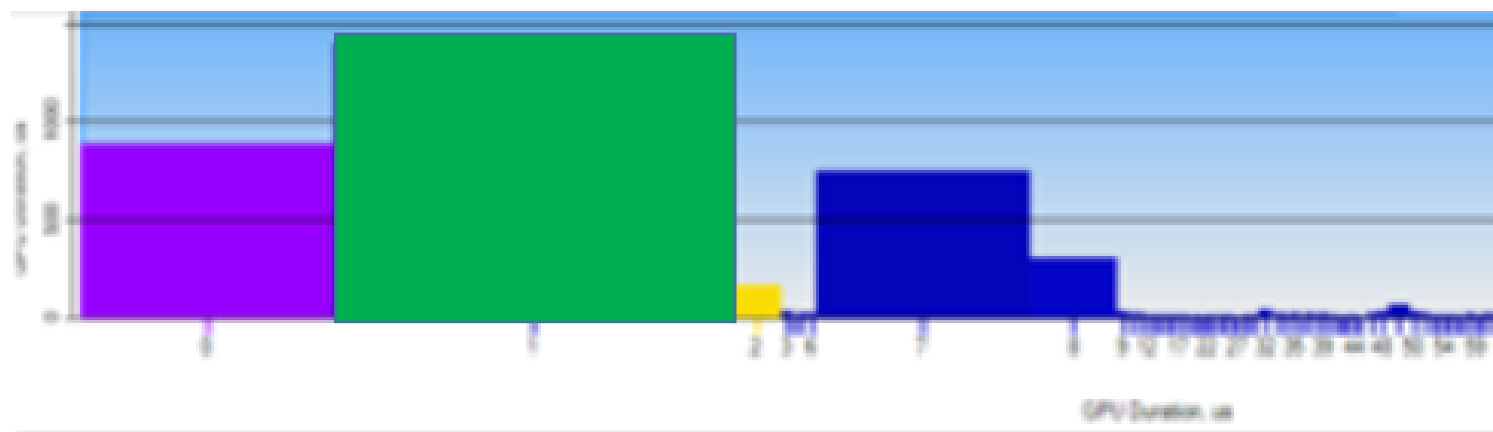
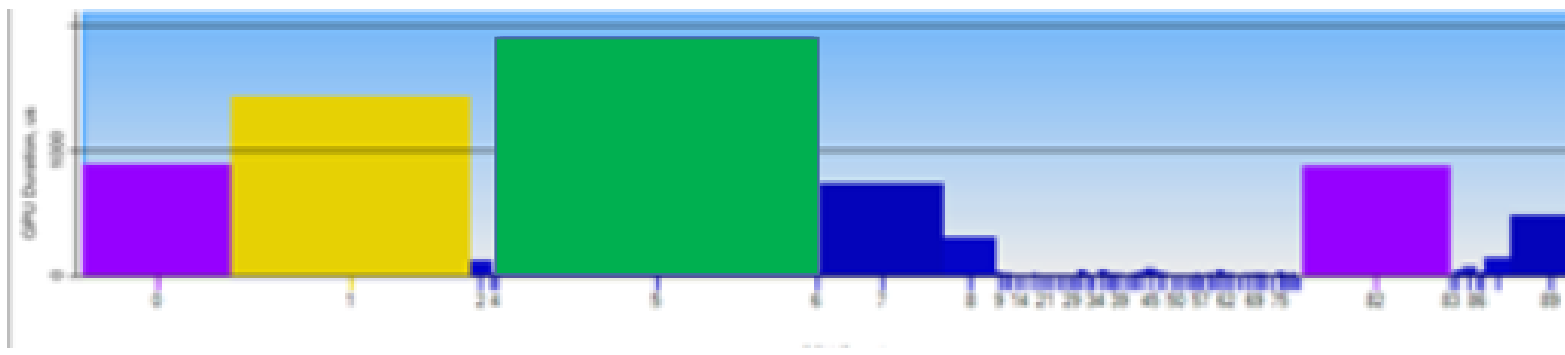
Показатель	Базовая версия	Отключение ненужного альфа-смешивания (земля)
Циклы GPU	1466843	1085794,5
Длительность работы GPU	1896,6 мкс	1398,4 мкс
Чтение из памяти, GPU	9,4 Мб	0,2 Мб
Запись в память, GPU	8,2 Мб	8,2 Мб

В таблице показана более подробная информация о рисовании травы после выключения альфа-смешивания. Длительность работы GPU, в результате, **снизилась на 26,0%**. Кроме того, обратите внимание на то, что показатель чтения из памяти **снизился на 97,2%**.

Эффективное применение Z-отсечения

Z-отсечение- это...

В игре есть два вида вывода окружения: небо (эрг №1) и трава (эрг №5). Так как большая часть неба находится позади травы, значительная площадь неба никогда в процессе игры показана не будет. Однако небо выводится первым, что препятствует эффективному использованию Z-отсечения.



Сравнение нагрузки на систему до и после изменения порядка вывода объектов в Graphics Frame Analyzer.

В таблице показаны более подробные сведения о рисовании неба после изменения порядка вывода объектов. Время работы GPU, в частности, уменьшилось на 88%. Обратите внимание и на то, что объём данных, записанных в память, сократился примерно на 98,9%.

Показатель	Базовая версия	Изменение порядка рисования (небо)
Циклы GPU	1113276	133975
Длительность работы GPU	1443 мкс	174,2 мкс
Раннее Z-отбрасывание	0	2145344
Количество записанных образцов	2165760	20416
Чтение из памяти, GPU	0,2 Мб	0,0 Мб
Запись в память, GPU	8,2 Мб	0,1 Мб

Выводы

Когда вы приступаете к оптимизации игры для Android x86, сначала следует портировать её на эту платформу, а затем – отыскать узкие места. Средства профилирования способны помочь в измерении производительности и в поиске проблемы с быстродействием, относящимся к GPU. Мощный аналитический инструмент Intel GPA может дать возможность поэкспериментировать с графической составляющей приложения без внесения изменений в исходный код.

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

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



Компрессия текстур

Первые версии Subway Simulator включали в себя статичную локацию с заранее настроенными источниками света (в том числе и в Realtime). Довольно быстро стало понятно, что такая концепция работает лишь до тех пор, пока в игре не используются многополигональные модели и большое количество текстур, как для интерфейса, так и для локаций, поездов и людей. В противном же случае необходимо подобрать другой вариант, который обеспечивал бы хорошее качество при умеренных затратах памяти.

В первую очередь **были написаны шейдеры с заранее запеченными картами света (LightMaps)** — решили отказаться от просчета освещения в принципе, заменив его специальными шейдерами на материалах и прожекторами на поезде.

Таким образом удалось сократить потребление памяти со стороны как локаций, так поездов — в общем счете **почти в два раза**.

Компрессия звука и моделей

В рамках дополнительной оптимизации мы **сжали все звуки на максимальное значение — 1**. Опыт тестирования показал, что с мобильных девайсов практически не чувствуется потеря в качестве звука. Выставление Mono-режима всех звуковых файлов также снизило потребление памяти **почти в полтора раза**.

Отладка метрики

Еще одна 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.

Связано это с тем, что в Unity нет понятия бесконечного пространства, только условные границы, до которых можно без проблем просчитывать значения координат. Чем дальше мы будем уходить от этих границ, тем с большей погрешностью будет происходить рендеринг сцены.

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. Такое решение снижает вероятность погрешностей в просчете движения, если игрок прокатается, скажем, полчаса или больше.