

Содержание

Об авторе	9
Благодарности	10
О технических рецензентах	11
Предисловие	13
Глава 1. Выявление проблем с производительностью	19
Профилировщик Unity Profiler	20
Запуск профилировщика.....	21
Окно профилировщика	25
Методы анализа производительности.....	32
Проверка присутствия сценария	33
Проверка количества сценариев	34
Сведение к минимуму изменений в текущем коде.....	35
Сведение к минимуму внутренних помех	35
Сведение к минимуму внешних помех	37
Выборочное профилирование сегментов кода	37
Управление профилировщиком из сценариев	38
Нестандартное профилирование использования центрального процессора.....	40
Сохранение и загрузка данных профилировщика	44
Сохранение данных профилировщика	45
Загрузка данных профилировщика	48
Заключительные соображения о профилировании и анализе.....	52
Освоение профилировщика.....	52
Уменьшение шума	53
Сосредоточение внимания на проблеме.....	54
Итоги	54
Глава 2. Приемы разработки сценариев	56
Кэширование ссылок на компоненты.....	56
Самый быстрый метод получения ссылок на компоненты.....	58
Удаление пустых объявлений обратных вызовов.....	61
Не используйте методов Find() и SendMessage()	63
Статические классы.....	65
Компоненты-одиночки.....	67
Сохранение ссылок на существующие объекты.....	71
Глобальная система обмена сообщениями	74
Отключение неиспользуемых сценариев и объектов.....	85
Отключение невидимых объектов	85
Отключение отдаленных объектов	86

Замена расстояния квадратом расстояния	87
Избегайте извлечения строковых свойств объектов игры	89
Метод Update, сопрограммы и метод InvokeRepeating	91
Кэширование изменений компонента Transform	97
Ускорение проверки отсутствия ссылки на игровой объект	98
Итоги	99
Глава 3. Преимущества пакетной обработки	100
Вызовы системы визуализации	101
Материалы и шейдеры	103
Динамическая пакетная обработка	107
Атрибуты вершин	108
Однородное масштабирование	109
Краткие выводы о динамической пакетной обработке	110
Статическая пакетная обработка	111
Флаг Static	111
Требования к памяти	112
Ссылки на материалы	113
Особенности использования статической пакетной обработки	113
Краткие выводы о статической пакетной обработке	115
Итоги	116
Глава 4. Привнесение искусства	117
Аудио	118
Загрузка аудиофайлов	118
Форматы кодирования и уровни качества	121
Улучшение производительности аудио	123
Файлы текстур	128
Форматы сжатия	129
Улучшение производительности обработки текстур	131
Файлы мешей и анимаций	141
Уменьшение количества полигонов	141
Импорт/расчет только необходимого	143
Встраиваемые анимации	143
Оптимизация мешей движком Unity	144
Объединение мешей	145
Итоги	145
Глава 5. Разгон физического движка	147
Внутреннее устройство физического движка	148
Физические движки и время	148
Статические и динамические коллайдеры	152
Обнаружение столкновений	153
Виды коллайдеров	154
Матрица столкновений	155
Активное и неактивное состояния компонента Rigidbody	156
Отбрасывание лучей и объектов	157
Оптимизация производительности физической системы	157
Настройка сцены	157

Правильное использование статических коллайдеров	160
Оптимизация матрицы столкновений	161
Предпочтение дискретного обнаружения столкновений	162
Изменение частоты фиксированных обновлений	163
Настройка максимально допустимой длительности	165
Уменьшение отбрасывания лучей и ограничение проверяемого объема	165
Избегайте сложных меш-коллайдеров	167
Избегайте сложных физических компонентов	170
Пусть физические объекты поспят	170
Изменение количества итераций	172
Оптимизация тряпичных кукол	173
Когда следует использовать физическую систему	175
О возможности перехода на Unity 5	176
Итого	177
Глава 6. Динамическая графика	178
Профилирование проблем отображения	179
Профилирование графического процессора	181
Отладка кадров	184
Поиск методом перебора	185
Основная нагрузка приходится на центральный процессор	185
Узкие места на этапе предварительной обработки	189
Уровень детализации	189
Отключение скиннинга графическим процессором	191
Уменьшение тесселяции	192
Узкие места на этапе окончательной обработки	192
Скорость заполнения	192
Пропускная способность памяти	206
Ограничения видеопамати	210
Освещение и затенение	212
Непосредственное отображение	213
Отложенное затенение	214
Обработка освещения в вершинном шейдере (устаревший способ)	214
Обработка теней в реальном времени	215
Оптимизация освещения	215
Оптимизация графики для мобильных устройств	217
Минимизация обращений к системе визуализации	218
Минимизация количества материалов	218
Уменьшение размеров текстур и количества материалов	218
Квадратные текстуры с размером стороны, кратной степени числа 2	219
Использование в шейдерах форматов с минимально допустимой точностью	219
Избегайте альфа-тестирования	219
Итого	219
Глава 7. Мастерство управления памятью	221
Платформа Mono	222
Процесс компиляции	224

Оптимизация использования памяти.....	226
Области памяти Unity	227
Значения и ссылки.....	236
Важность порядка размещения данных	249
Прикладной программный интерфейс Unity	250
Циклы foreach	251
Сопрограммы.....	252
Замыкания	252
Функции в библиотеке .NET.....	253
Временные рабочие буферы	254
Пулы объектов.....	254
Пулы шаблонных объектов	257
Компоненты пула	260
Система пулов шаблонных объектов	263
Пулы шаблонных объектов.....	267
Активация объектов	268
Предварительное создание экземпляров	269
Деактивация объектов	270
Тестирование пула шаблонных объектов.....	271
Организация пулов шаблонных объектов и загрузка сцены	272
Итоговые замечания об организации пулов шаблонных объектов	273
Дальнейшее развитие Mono и Unity	274
Итоги	276

Глава 8. Тактические советы и подсказки 278

Подсказки по клавишам быстрого доступа в редакторе	279
Игровые объекты.....	279
Представление сцены.....	279
Массивы.....	280
Интерфейс.....	280
Прочее.....	281
Советы, касающиеся интерфейса редактора	281
Общие	281
Представление инспектора	284
Представление проекта	286
Представление иерархии	287
Представления сцены и игры	288
Режим воспроизведения	289
Советы для сценариев	290
Общие	290
Атрибуты	291
Регистрация.....	293
Полезные ссылки.....	293
Советы по настройке редактора и меню	294
Советы, не касающиеся Unity напрямую	296
Другие советы	297
Итоги	298

Предметный указатель 300

Об авторе

Крис Дикинсон (Chris Dickinson) вырос в Англии, увлекается естественными науками, математикой и компьютерными играми. Получив в 2005 году степень магистра в области физики и электроники, сразу же переехал в Калифорнию, чтобы заниматься научными исследованиями в самом сердце Кремниевой долины. Затем, разочаровавшись в своем выборе, переключился на работу в области разработки программного обеспечения.

За последнее десятилетие он многого достиг, сделал карьеру в области разработки программного обеспечения, стал ведущим разработчиком. Крис в основном занимался программным обеспечением для автоматизации и создавал инструменты для внутреннего тестирования в процессе разработки, но страсть к видеоиграм не потерял. В 2010 году он занялся изучением секретов разработки игр и 3-мерной графики и получил еще одну степень, степень бакалавра в области разработки игр и программного моделирования. Написал учебник о применении физических законов в играх (*Learning Game Physics with Bullet Physics and OpenGL*, издана в Packt Publishing). В настоящее время продолжает разрабатывать программное обеспечение, свободное время посвящая работе над независимыми игровыми проектами, использующими Unity 3D.

О технических рецензентах

Клиффорд Чемпион (Clifford Champion) обладает обширным, многолетним опытом разработки программного обеспечения, полученным во время работы над 3D-играми, интернет-приложениями и программами искусственного интеллекта. Имеет ученые степени в области математики и информатики, присвоенные в Калифорнийском университете в Сан-Диего и в Калифорнийском университете в Лос-Анджелесе. В прошлом Клиффорд работал в компании Navok (ныне часть Microsoft), занимающейся разработкой игровых технологий, и в известной дизайн-студии PlainJoe Studios. В настоящее время возглавляет группу разработки программного обеспечения в компании zSpace (zspace.com), специализирующейся на 3D-приложениях виртуальной реальности для учебных заведений и промышленности.

Клиффорда можно найти в Twitter (@duckmaestro), и он готов вести дискуссии на любые темы.

Доктор Себастьян Т. Кёниг (Dr. Sebastian T. Koenig) получил степень доктора наук в области систем «человек–машина» в университете Кентербери, Новая Зеландия, за разработку основы системы виртуальной реальности для индивидуализированной когнитивной реабилитации. Имеет диплом психолога в области клинической нейропсихологии и реабилитации с применением систем виртуальной реальности, полученный в университете Регенсбурга, Германия.

Является основателем и генеральным директором Katana Simulations, где курирует проектирование, разработку и проведение когнитивной экспертизы систем моделирования для профессиональной подготовки. Его профессиональный опыт включает более 10 лет клинической работы в сфере когнитивной реабилитации и более 8 лет в области исследования, разработки и тестирования пользователями виртуальной реальности. Он часто выступает на международных конференциях и приглашается в качестве рецензента научных публикаций в области реабилитации, когнитивной психологии, нейропсихологии, программной инженерии, разработки игр, исследований восприятия пользователями игр и виртуальной реальности.

Разработал ряд прикладных программ для когнитивной оценки и профессиональной подготовки. За работу над задачей виртуальной памяти в 2011 году был удостоен престижной премии Laval Virtual в категории медицины и здравоохранения. Другими его достижениями являются: приложение виртуальной реальности для оценки восприятия, разработанное в сотрудничестве с Фондом Кесслера (Нью-Джерси, США), и запатентованное приложение для моторной и когнитивной подготовки JewelMine/Mystic Isle, основанное на Microsoft Kinect и созданное в Калифорнийском институте креативных технологий (Калифорния, США).

Поддерживает веб-сайт www.virtualgamelab.com, где представлены результаты его исследований и проекты разработки программного обеспечения. Сайт также содержит исчерпывающий список учебных материалов, посвященных игровому движку Unity.

Предисловие

Привлекательность – чрезвычайно важный компонент любой игры. Она определяется не только сценарием игры и игровым процессом, но также гладкостью графики, надежностью соединения с многопользовательскими серверами, скоростью реакции на действия пользователя и даже величиной конечного файла приложения, что важно при распространении из хранилищ или из облака. Барьер включения в разработку игр был значительно снижен благодаря появлению недорогих и качественных инструментов, таких как Unity. Однако требования к конечному продукту, предъявляемые игроками, растут с каждым днем. Следует иметь в виду, что все аспекты игры могут и будут тщательно изучены как игроками, так и критиками.

Цели оптимизации производительности тесно связаны с восприятием игры пользователями. Плохо оптимизированная игра, как правило, имеет низкую частоту кадров, зависает, сбивает, реагирует на действия пользователя с опозданием, долго загружается, ведет себя непоследовательно и дергано во время выполнения, работа физического движка часто нарушается и даже имеет чрезмерно высокое энергопотребление (что крайне важно в эпоху мобильных устройств). Наличие хотя бы одной из этих проблем должно приводить в ужас разработчиков, поскольку авторы обзоров склонны делать упор на отрицательных качествах, оставляя в стороне положительные.

Высокая производительность обеспечивается оптимальным использованием всех имеющихся ресурсов, включая ресурсы центрального процессора, такие как частота процессора и объем оперативной памяти, ресурсы графического процессора, такие как объем видеопамати и ее пропускная способность, и т. д. Поэтому основной задачей оптимизации является такое распределение ресурсов, которое обеспечит эффективное их использование и выполнение высокоприоритетных задач в первую очередь. Даже небольшие, кратковременные задержки и потери производительности снижают привлекательность игры, ухудшают погружение в нее и ограничивают потенциал разработчиков.

Важно также выбрать момент, когда следует остановиться и прекратить заниматься улучшением производительности. В идеальном мире, где время и ресурсы ничем не ограничены, всегда найдется еще один способ достичь более удачного, быстрого и простого решения.

Но в реальном мире должен существовать момент в процессе разработки, когда следует заявить, что продукт достиг приемлемого качества. Если этого не сделать, дальнейшие усилия принесут мало пользы или вообще не дадут ощутимых результатов.

Лучший способ выбрать момент прекращения работ над улучшением производительности – ответить на вопрос: «А пользователь это заметит?» Если ответ на этот вопрос – нет, дальнейшая оптимизация не стоит затраченных на нее усилий. Существует старая поговорка, относящаяся к разработке программного обеспечения:

Преждевременная оптимизация – корень всех зол.

Под преждевременной оптимизацией понимается переработка и рефакторинг кода для повышения производительности, когда нет твердой уверенности в необходимости этих действий, например при полном отсутствии проблем с производительностью (отвечаем на вопрос о заметности для пользователя) или при наличии предположений, что проблема производительности вызвана определенной областью, но нет доказательств, что так и есть на самом деле. Такие ошибки обходятся разработчикам удручающе большим количеством рабочих часов, потраченных впустую.

Цель этой книги – дать вам инструменты, знания и навыки, необходимые для выявления и исправления проблем производительности, независимо от их причин. Подобные проблемы могут вызывать компоненты оборудования, такие как центральный процессор, графический процессор или оперативная память; подсистемы программного обеспечения, такие как подсистема физики, визуализации или же сам движок Unity. Кроме того, чем больше ресурсов удастся сэкономить, тем больше можно будет сделать с помощью движка Unity на одном и том же оборудовании, что позволит реализовать более интересный и динамичный процесс игры.

А это даст вашей игре больше шансов на успех и позволит ей выделиться из массы других, представленных на рынке, куда ежедневно выбрасываются новые, высококачественные игры.

О чем рассказывается в этой книге

Глава 1 «Выявление проблем с производительностью» посвящена знакомству с профилировщиком Unity Profiler и методами профилирования приложений, выявлению узких мест с точки зрения производительности и анализу основных причин их возникновения.

Глава 2 «Приемы разработки сценариев» описывает эффективные приемы разработки сценариев для Unity на языке C#, минимизации дублирования компонентов, оптимизации взаимодействий между объектами и многое другое.

Глава 3 «Преимущества пакетной обработки» посвящена изучению применения систем динамической и статической пакетной обработки в Unity для снижения нагрузки на систему визуализации.

Глава 4 «Привнесение искусства» поможет разобраться в закулисных процессах работы с ресурсами и узнать, как избежать распространенных ошибок при их импорте, сжатии и перекодировании.

Глава 5 «Разгон физического движка» посвящена особенностям физической системы Unity для 3D- и 2D-игр и правильной организации физических объектов для повышения производительности.

Глава 6 «Динамическая графика» подробно рассматривает систему визуализации и способы решения проблем производительности приложений, возникающих в системе визуализации, узкими местами которой являются графический процессор или центральный процессор, а также специальные методы для мобильных устройств.

Глава 7 «Мастерство управления памятью» анализирует внутренние рабочие процессы движка Unity, фреймворка Mono и управление памятью в этих компонентах, чтобы защитить приложение от распределения памяти в куче и сборки мусора во время выполнения.

Глава 8 «Тактические советы и приемы» знакомит со множеством полезных методов, используемых профессионалами для улучшения управления рабочими процессами и сценами.

Что потребуется для работы с книгой

Основное внимание в этой книге уделяется возможностям версии Unity 5.x. Большинство методов, описанных в книге применимы к проектам, созданным в Unity 4.x, но для некоторых из них требуется обновить Unity 4 до версии Pro (например, окклюзивная выбраковка, статическая пакетная обработка и даже сам профилировщик).

Кому адресована эта книга

Эта книга адресована разработчикам среднего и продвинутого уровня, уже имеющим опыт работы с большей частью функций Unity, заинтересованных в улучшении производительности игр или в решении конкретных проблем. Если вы испытываете проблемы, связанные

с перегрузкой процессора, скачками нагрузки во время выполнения, замедленным доступом к памяти, фрагментацией, уборкой мусора, снижением частоты графического процессора или пропускной способностью памяти, эта книга научит вас выявлять источники проблем и выбирать способы уменьшения их воздействия на приложение.

Вам потребуется знание языка C# при чтении разделов, содержащих сценарии и посвященных использованию памяти, а также базовое понимание библиотеки Cg – при рассмотрении оптимизации шейдеров.

Соглашения

В этой книге используется несколько разных стилей оформления текста для выделения разных видов информации. Ниже приводятся примеры этих стилей и объясняется их назначение.

Программный код в тексте, имена таблиц баз данных, имена папок, имена файлов, расширения файлов, фиктивные адреса URL, пользовательский ввод и ссылки в Twitter будут выглядеть так: «Главной функцией обратного вызова Unity является функция обновления Update().»

Блоки программного кода оформляются так:

```
public class TestComponent : MonoBehaviour {
    void Update() {
        if (Input.GetKeyDown(KeyCode.Space)) {
            PerformProfilingTest();
        }
    }
}
```

Когда потребуется привлечь ваше внимание к определенному фрагменту в блоке программного кода, он будет выделяться жирным шрифтом:

```
public class TestComponent : MonoBehaviour {
    void Update() {
        if (Input.GetKeyDown(KeyCode.Space)) {
            PerformProfilingTest();
        }
    }
}
```

Новые термины и важные слова будут выделены жирным. Текст, отображаемый на экране, например в меню или в диалогах, будет

оформляться так: «Пороговое значение для перехода в состояние сна может быть изменено при помощи выбора пункта **Edit** ⇒ **Project Settings** ⇒ **Physics** ⇒ **Sleep Threshold** (Редактирование ⇒ Параметры проекта ⇒ Физика ⇒ Интервал для перехода в состояние сна)».



Так будут оформляться предупреждения и важные примечания.



Так будут оформляться советы и рекомендации.

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте www.dmkpress.com, зайдя на страницу книги, и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Загрузка исходного кода примеров

Загрузить файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте www.dmkpress.com или www.dmk.ru в разделе «Читателям – Файлы к книгам».

Список опечаток

Хотя мы приняли все возможные меры, чтобы удостовериться в качестве наших текстов, ошибки всё равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в тексте или в коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии этой книги.

Если вы нашли опечатку, пожалуйста, сообщите о них главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в Интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и «Ракт» очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в Интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли принять меры.

Пожалуйста, свяжитесь с нами по адресу электронной почты dmkpress@gmail.com со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

Глава 1

Выявление проблем с производительностью

Оценка производительности большинства программных продуктов является исследовательским процессом, включающим: определение максимальных поддерживаемых параметров (количества одновременно работающих пользователей, максимального объема доступной памяти, ресурсов центрального процессора и т. д.); выполнение нагрузочного тестирования путем воссоздания реальных ситуаций; инструментальный сбор результатов тестирования; анализ данных для выявления узких мест; комплексный анализ причин; внесение изменений в конфигурацию или код приложения для исправления проблем; повторение всего перечисленного еще раз.

Разработка игр является творческим процессом, но это вовсе не означает, что ее не следует рассматривать с объективной и исследовательской точек зрения. Игра направлена на конкретную целевую аудиторию, что и определяет ее аппаратные ограничения. Нужно протестировать приложение, собрать данные о работе нескольких компонентов (центральный процессор, графический процессор, память, физика, визуализация и т. д.) и сравнить их с требуемыми характеристиками. Эти данные используются для выявления узких мест в работе приложения, с помощью дополнительных инструментов определяются главные причины проблем, после чего проблемы рассматриваются с разных точек зрения.

Для успешного выполнения этого процесса необходимы инструменты и знания, методы, с которыми знакомит эта глава. Они будут использоваться на протяжении всей книги для выявления проблем с производительностью и определения их главных причин. Такие навыки составляют методику выявления и анализа проблем с произ-

водительностью **Unity**-приложения и выбора мест для внесения изменений. В этой главе будет проделана подготовительная работа для освоения материала остальных глав, описывающего пути решения выявленных проблем.

Начнем с изучения **профилировщика Unity Profiler** и его многочисленных функций. Затем опишем несколько способов выявления узких мест и в завершение приведем несколько советов, касающихся применения этих технологий.

Профилировщик Unity Profiler

Профилировщик Unity Profiler встроен в **редактор Unity** и обеспечивает целенаправленный поиск узких мест производительности путем сбора информации о работе компонентов **Unity3D** во время выполнения, то есть:

- использование центрального процессора отдельно каждым компонентом движка Unity3D;
- затраты на визуализацию;
- использование графического процессора GPU на разных этапах работы графического конвейера;
- использование памяти;
- затраты на воспроизведение аудиоэффектов;
- использование физического движка.



Начиная с **Unity 5.0**, компания Unity Technologies открыла доступ к профилировщику в редакции Personal Edition (прежде эта редакция называлась Free Edition).

Пользователям, использующим редакцию Unity 4 Free Edition, необходимо обновить ее до Unity 5 или приобрести лицензию для редакции Unity 4 Pro Edition.

Однако использование профилировщика влечет дополнительные накладные расходы. При включении флагов профилирования в настройках компилятора будут генерироваться журнал событий времени выполнения и прочий код поддержки профилирования, что вызовет дополнительные затраты ресурсов центрального процессора и памяти во время выполнения. Однако затраты на профилирование не очень значительны и не приводят к непредсказуемой разнице в поведении при включенном и отключенном профилировщике.

Кроме того, всегда избегайте использования режима редактирования во время профилирования и проведения сравнительного анализа

из-за накладных расходов на работу интерфейса редактора и затрат памяти на размещение различных объектов и компонентов. Для получения более точных и надежных данных проверку приложения лучше выполнять автономно и на целевом устройстве.



Читатели, знакомые с процедурой подключения профилировщика к приложениям, могут пропустить следующий раздел и перейти сразу разделу «*Окно профилировщика*».

Запуск профилировщика

Начнем с краткого обзора способов подключения профилировщика Unity в различных контекстах:

- к локальным экземплярам приложения через редактор или автономное профилирование самого редактора;
- к локальным экземплярам приложения в **веб-плеере Unity**;
- к удаленным экземплярам приложения на iOS-устройствах (iPad или iPhone);
- к удаленным экземплярам приложения на Android-устройствах (планшет или телефон с ОС Android).

Далее мы кратко рассмотрим особенности настройки профилировщика в каждом из этих контекстов.

Редактор или автономные экземпляры

Запустить профилировщик можно только из редактора Unity, после чего его следует подключить к запущенному экземпляру приложения. Здесь возможны случаи, когда игра запущена из редактора или как автономное приложение на локальном или удаленном устройстве, или когда требуется выполнить профилирование самого редактора.

Чтобы запустить профилировщик, выберите в меню редактора пункт **Window** ⇒ **Profiler** (Окно ⇒ Профилировщик), как показано на рис. 1.1. При работе редактора в режиме воспроизведения в окне профилировщика появится отчет.



Для профилирования автономных проектов при их сборке следует установить флаги Use Development Mode (Использовать режим разработки) и Autocconnect Profiler (Автоматически подключать профилировщик).

Выбор экземпляра приложения, запущенного в редакторе (в режиме воспроизведения) или автономного (собранныго отдельно и рабо-

тающего в фоновом режиме), производится с помощью меню **Active Profiler** (Активный профилировщик) в окне профилировщика, как показано на рис. 1.2.

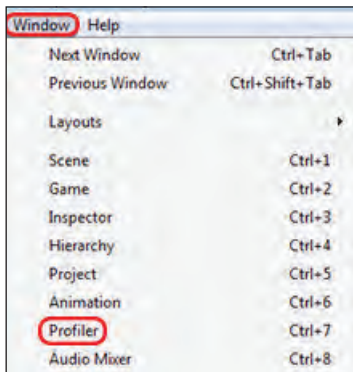


Рис. 1.1 ❖ Запуск профилировщика Unity

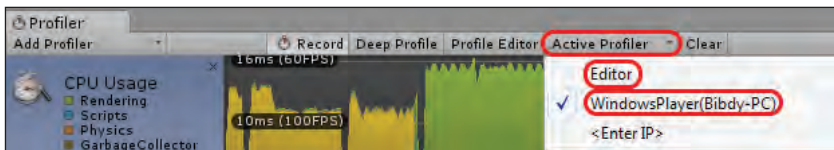


Рис. 1.2 ❖ Профилировщик Unity

Профилирование редактора

Профилирование редактора, например редактора сценариев, можно включить выбором пункта меню **Profile Editor** (Профилирование редактора) в окне профилировщика, как показано на рис. 1.3. Обратите внимание, что для этого требуется отметить параметр **Editor** (Редактор) в меню **Active Profiler** (Активный профилировщик).

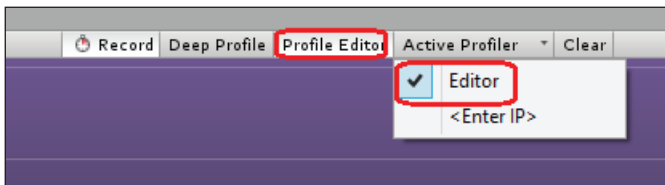


Рис. 1.3 ❖ Профилирование редактора сценариев

Подключение к веб-плееру Unity

Профилировщик можно также подключить к экземпляру приложения в веб-плеере Unity, запущенном в браузере. Это позволит выполнить профилирование веб-приложения в наиболее реальной среде с помощью целевого браузера и протестировать несколько типов браузеров для выявления несоответствий.

1. Перед сборкой приложения для веб-плеера включите флаг **Use Development Mode** (Использовать режим разработки).
2. Запустите скомпилированное для веб-плеера приложение в браузере, активируйте окно браузера и, удерживая нажатой клавишу **Alt** (*Option* на компьютере Mac), щелкните правой кнопкой мыши на объекте веб-плеера в браузере, чтобы открыть меню **Release Channel Selection** (Выбор канала вывода). Затем выберите канал **Development** (Разработка), как показано на рис. 1.4.



Обратите внимание, что изменение параметра Release Channel (Канал вывода) вызовет перезапуск приложения в веб-плеере.

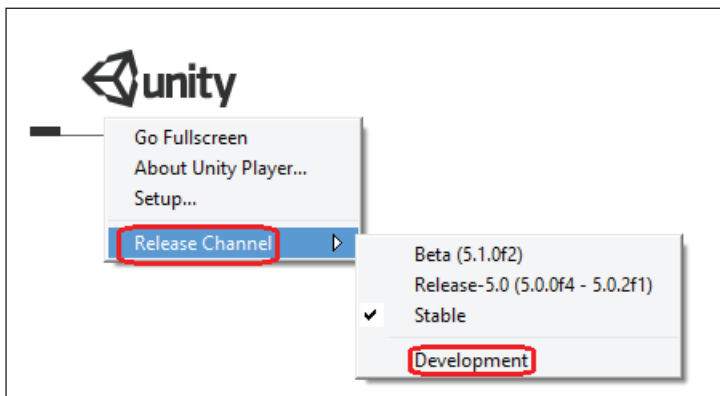


Рис. 1.4 ❖ Выбор канала вывода

3. Откройте профилировщик в окне редактора Unity и выберите пункт **Active Profiler** ⇒ **WindowsWebPlayer(COMPUTERNAME)** или **Active Profiler** ⇒ **OSXWebPlayer(COMPUTERNAME)** в зависимости от используемой операционной системы, как показано на рис. 1.5.

После этого в окне профилировщика должна появиться собранная им информация.

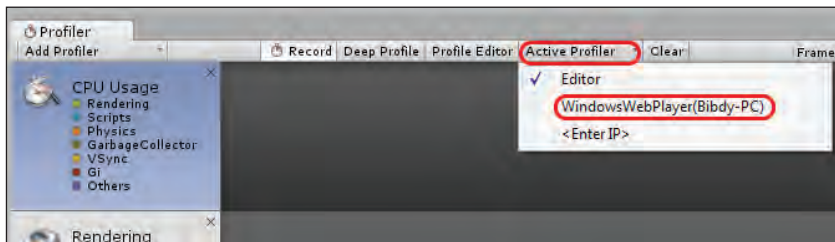



Рис. 1.5 ❖ Выбор веб-плеера


Подключение к удаленным iOS-устройствам

Профилировщик можно также подключить к активному экземпляру приложения, запущенному на удаленном iOS-устройстве, таком как iPad или iPhone, например через Wi-Fi-подключение. Чтобы подключить профилировщик к устройству Apple, выполните следующие действия:

 Обратите внимание, что подключение к устройству Apple возможно, только если профилировщик выполняется на устройстве Apple Mac.

1. Перед сборкой приложения включите флаги **Use Development Mode** (Использовать режим разработки) и **Autocconnect Profiler** (Автоматически подключать профилировщик).
2. Подключите оба устройства, iOS и Mac, к локальной сети или сети ADHOC WiFi.
3. Соедините устройства iOS и Mac кабелем USB или Lightning.
4. Запустите сборку приложения, выбрав в меню пункт **Build & Run** (Собрать и запустить), как обычно.
5. Откройте окно профилировщика в редакторе Unity и выберите устройство в меню **Active Profiler** (Активный профилировщик).

После этого в окне профилировщика должна появиться собранная им информация.

 Для передачи данных профилировщик использует порты с номерами от 54998 до 55511. Убедитесь, что эти порты доступны для исходящего трафика, если в сети используется брандмауэр.

Подключение к удаленным Android-устройствам

Существуют два способа подключения профилировщика к Android-устройству: через Wi-Fi-соединение или с помощью инструмента отладки **Android Debug Bridge (ADB)**. ADB – это набор инструментов отладки, входящий в состав пакета Android SDK.

Для профилирования через Wi-Fi-соединение выполните следующие действия:

1. Перед сборкой приложения включите флаги **Use Development Mode** (Использовать режим разработки) и **Autoconnect Profiler** (Автоматически подключать профилировщик).
2. Подключите Android-устройство и компьютер к локальной Wi-Fi-сети.
3. Соедините Android-устройство с компьютером кабелем USB.
4. Запустите сборку приложения, выбрав в меню пункт **Build & Run** (Собрать и запустить), как обычно.
5. Откройте окно профилировщика в редакторе Unity и выберите устройство в меню **Active Profiler** (Активный профилировщик).

После этого в окне профилировщика должна появиться собранная им информация.

Для профилирования через ADB выполните следующие действия:

1. В окне командной строки Windows выполните команду `adb devices`, которая проверит, распознается ли устройство отладчиком ADB (если устройство не распознается, для него придется установить специальные драйверы и/или включить на целевом устройстве отладку через USB).



Обратите внимание, что если команда `adb devices` не будет найдена при попытке запустить ее из командной строки, добавьте путь к папке с Android SDK в переменную окружения `PATH`.

2. Перед сборкой приложения включите флаги **Use Development Mode** (Использовать режим разработки) и **Autoconnect Profiler** (Автоматически подключать профилировщик).
3. Соедините Android-устройство с компьютером кабелем USB.
4. Запустите сборку приложения, выбрав в меню пункт **Build & Run** (Собрать и запустить), как обычно.
5. Откройте окно профилировщика в редакторе Unity и выберите устройство в меню **Active Profiler** (Активный профилировщик).

После этого в окне профилировщика должна появиться собранная им информация.

Окно профилировщика

А теперь рассмотрим основные функции профилировщика.



Обратите внимание, что в этом разделе описываются функции профилировщика Unity 5. В этой версии появились новые функции,

которые в профилировщике Unity 4 могут называться по-другому или вообще отсутствовать.

Окно профилировщика делится на три основные области:

- панель управления;
- временная шкала;
- подробное описание.

Эти области показаны на рис. 1.6.

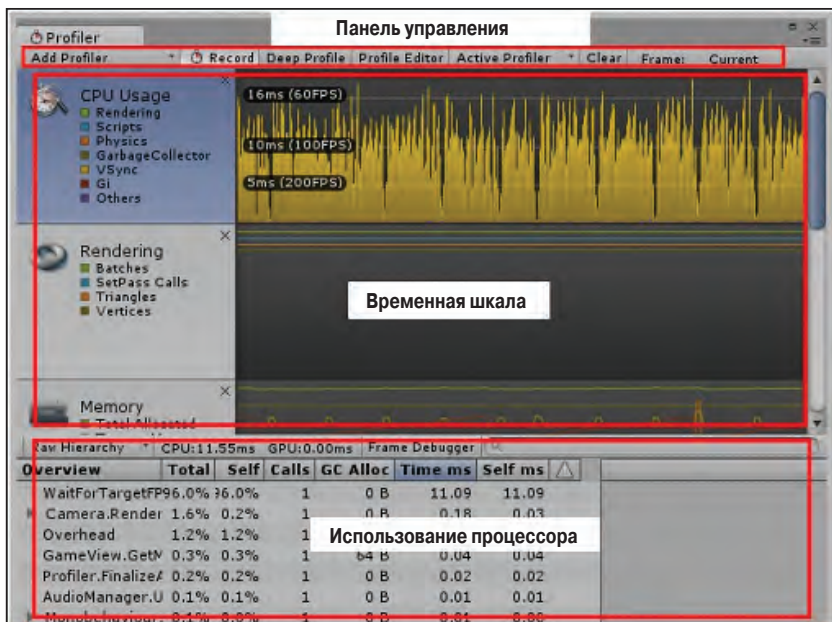


Рис. 1.6 ❖ Окно профилировщика

Элементы управления

Верхняя панель содержит несколько элементов управления профилированием и глубиной сбора системных данных:

- **Add Profiler** (Добавить профилировщик): по умолчанию профилировщик отображает в области временной шкалы несколько компонентов движка Unity, но с помощью пункта **Add Profiler** можно добавлять другие элементы. Полный список компонентов, доступных для профилирования, приводится в разделе «Временная шкала».

- **Record** (Запись): включение этого параметра заставит профилировщик непрерывно записывать данные профилирования. Обратите внимание, что данные записываются только в режиме воспроизведения (и если не включена пауза) или если установлен параметр **Profile Editor** (Профилирование редактора).
- **Deep Profile** (Глубокое профилирование): при обычном профилировании записываются только время и память, использованная всеми методами обратного вызова, такими как `Awake()`, `Start()`, `Update()`, `FixedUpdate()`, и т. д. Включение параметра **Deep Profile** (Глубокое профилирование) приведет к повторной компиляции сценариев для обеспечения возможности отслеживания характеристик всех вызываемых методов. Это вызовет увеличение накладных расходов во время выполнения и значительный рост использования памяти, поскольку отбрасываемые данные будут охватывать весь стек вызовов. Как следствие глубокое профилирование невозможно производить на больших проектах при слабом оборудовании, поскольку Unity может не хватить памяти даже для запуска тестирования!



Обратите внимание, что переход к глубокому профилированию требует повторной компиляции проекта, поэтому следует избегать переключения этого параметра во время выполнения.

Поскольку установка этого параметра приводит к слепому профилированию всех методов, он редко используется на практике. Этот параметр следует использовать, только когда профилирование по умолчанию не дало результатов или для небольших тестовых сцен, использующих небольшое подмножество функций игры.

Если большой проект или сцена требует глубокого профилирования, а установка параметра **Deep Profile** (Глубокое профилирование) сильно затрудняет выполнение, можно использовать альтернативные способы, описанные в разделе «Выборочное профилирование сегментов кода» ниже;

- **Profile Editor** (Профилирование редактора): этот параметр включает профилирование редактора, то есть производится сбор данных профилирования для самого редактора Unity. Он может пригодиться для профилирования нестандартных редакторов сценариев, разработанных пользователями;



Обратите внимание, что перед использованием этой функции следует выбрать параметр **Editor** (Редактор) в меню **Active Profiler** (Активный профилировщик).

- **Active Profiler** (Активный профилировщик): это раскрывающееся меню с вариантами выбора целевого экземпляра Unity для профилирования. Как уже упоминалось, таким экземпляром может быть редактор, а также экземпляр автономного, локального или удаленного приложения;
- **Clear** (Очистка): очищает данные профилирования в области временной шкалы;
- **Frame Selection** (Выбор кадра): счетчик кадров **Frame** отображает количество кадров, подвергшихся профилированию, и текущий кадр в шкале времени. Две кнопки позволяют перемещаться между кадрами вперед и назад, и еще одна кнопка (**Current** (Текущий)) выбирает последний, текущий кадр, и сохраняет эту позицию. Таким образом, в разделе «Подробное описание» во время профилирования всегда будут отображаться данные для текущего кадра;
- **Timeline View** (Временная шкала): временная шкала отображает данные, организованные по областям, в зависимости от исследуемого компонента движка.

В каждой области выводится несколько графиков разного цвета для различных подразделов соответствующих компонентов. Эти графики могут скрываться и отображаться по желанию.

Каждая область специализируется на профилировании различных компонентов движка Unity. При выборе области во временной шкале в разделе использования процессора появится информация о соответствующем компоненте для выбранного кадра.

В разделе «Подробное описание» может отображаться самая разная информация, в зависимости от выбранной области.



Области можно удалять из временной шкалы щелчком на значке «X» в правом верхнем углу области. Чтобы восстановить закрытую область, выберите соответствующий пункт в меню **Add Profiler** (Добавить профилировщик).

Область использования центрального процессора

Раздел использования центрального процессора отображает информацию об использовании центрального процессора подсистемами Unity, такими как компоненты MonoBehaviour, камеры, процессы

визуализации, процессы моделирования физической среды, пользовательский интерфейс (включая интерфейс редактора при профилировании редактора), аудиопроцессы, сам профилировщик и т. д.

Существуют три режима отображения данных об использовании центрального процессора в разделе «Подробное описание»:

- иерархия;
- простая иерархия;
- шкала времени.

В режиме **Hierarchy Mode** (Режим иерархии) схожие элементы данных и вызовы глобальных функций Unity для удобства группируются вместе, например в этом режиме объединяются вызовы разделителей визуализации, такие как `BeginGUI()` и `EndGUI()`.

В режиме **Raw Hierarchy Mode** (Режим простой иерархии) глобальные вызовы функций Unity не группируются. Это, как правило, усложняет анализ содержимого раздела «Подробное описание», но позволяет подсчитать, сколько раз был вызван конкретный глобальный метод, или определить, не потребовал ли один из этих вызовов больших затрат процессорного времени и памяти, чем ожидалось. Например, вызовы `BeginGUI()` и `EndGUI()` будут представлены отдельными элементами, что приведет к захламлению раздела и затруднит его чтение.

Наиболее полезным, пожалуй, режимом в разделе «Подробное описание» является режим **Timeline Mode** (Режим шкалы времени, не путайте его с разделом «Временная шкала»). В этом режиме отображаются затраты времени центрального процессора для текущего кадра в виде стека вызовов. Блоки в верхней части соответствуют методам, вызванным непосредственно движком Unity (например, методы `Start()`, `Awake()` или `Update()`), а блоки под ними представляют методы, вызванные этими методами, которые, в свою очередь, могут включать вызовы методов других компонентов или объектов.

Ширина каждого блока определяется относительным временем выполнения метода в сравнении с окружающими его блоками. Кроме того, методы, выполняющиеся относительно быстро в сравнении с более *тяжелыми* методами, отображаются серыми прямоугольниками, чтобы не заострять на них внимания.

Раздел «Подробное описание» в режиме временной шкалы предлагает наглядный и хорошо организованный способ, позволяющий выявить наиболее тяжелые методы в стеке вызовов и оценить, как время их работы соотносится со временем выполнения других методов, вызываемых в этом же кадре. Это значительно облегчает выявление виновников проблем.

Например, предположим, что необходимо выявить причину проблемы с производительностью по результатам, изображенным на рис. 1.7. Здесь сразу бросаются в глаза три метода, затрачивающих на выполнение примерно одинаковое время, о чем свидетельствует примерно равная ширина представляющих их блоков.

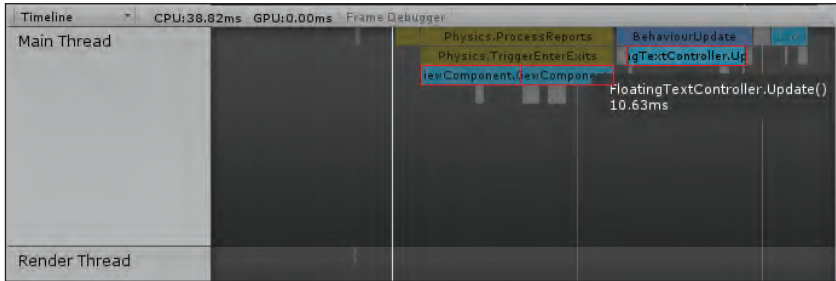


Рис. 1.7 ❖ Результаты профилирования с тремя проблемными методами

Здесь имеются целых три метода, за счет улучшения которых можно повысить производительность, что увеличивает вероятность найти фрагмент, который можно оптимизировать. Но улучшение производительности каждого метода приведет к уменьшению общего времени обработки кадра лишь на одну треть. Следовательно, придется проанализировать и оптимизировать все три метода, чтобы минимизировать время обработки кадра.

Область использования процессора будет подробно рассматриваться в *главе 2 «Приемы разработки сценариев»*.

Область использования графического процессора

Область использования графического процессора (GPU Area) аналогична области использования центрального процессора, но отображает вызовы методов и их длительность, которые выполняются на графическом процессоре. Соответственно, здесь отображаются вызовы методов Unity, связанных с камерами, рисованием, освещением, тенями и т. д.

Область использования графического процессора будет подробно рассматриваться в *главе 6 «Динамическая графика»*.

Область визуализации

Область визуализации (Rendering Area) содержит статистику визуализации, например количество вызовов `SetPass`, общее количество

пакетов для отображения сцены, количество пакетов, сохраненных при динамической и статической обработке, потребление памяти для текстур и т. д.

Область визуализации будет подробно рассматриваться в *главе 3 «Преимущества пакетной обработки»*.

Область памяти

Область памяти (Memory Area) позволяет исследовать потребление памяти приложением в двух режимах:

- простой режим;
- подробный режим.

Простой режим (Simple Mode) позволяет получить лишь общее представление о потреблении памяти такими компонентами, как низкоуровневый движок Unity, фреймворк Mono (размер общей кучи, обрабатываемой сборщиком мусора), графика, аудио (FMOD) и даже сам профилировщик.

Подробный режим (Detailed Mode) позволяет оценить потребление памяти отдельными игровыми объектами и компонентами в виде их собственных управляемых представлений. В этом режиме отображается дополнительный столбец, поясняющий причину потребления памяти объектом, и когда она может быть освобождена.

Область памяти будет подробно рассматриваться в *главе 7 «Мастерство управления памятью»*.

Область аудио

Область аудио (Audio Area) дает возможность исследовать статистику использования аудиоустройств и позволяет оценить, какую нагрузку оказывает аудиосистема на центральный процессор, а также общий объем памяти, потребляемой **источниками звука** (для выполняющих воспроизведение и в режиме паузы).

Область аудио подробно будет рассматриваться в *главе 4 «Привнесение искусства»*.



Об аудиосистеме часто забывают, когда дело касается улучшения производительности, однако функции воспроизведения аудио могут стать главным источником проблем, если не управлять ими должным образом. Поэтому периодически проверяйте статистику использования памяти и центрального процессора аудиосистемой.

Область физических движков 3D/2D

Имеются две области физических движков: одна – для 3D-системы (Nvidia PhysX) и другая – для 2D-системы (Box2D), – интегрированных в Unity, начиная с версии 4.5. Эти области отображают разнообразные статистические данные для таких компонентов, как Rigidbody, Collider и счетчики контактов.

Эта область будет рассматриваться в *главе 5 «Разгон физического движка»*.



На момент публикации этой книги в последней версии Unity v5.2.2f1 область Physics3D Area предоставляла незначительное количество элементов, в то время как область Physics2D Area предлагала значительно больше информации.

Методы анализа производительности

Применение надлежащих приемов программирования и управления ресурсами проекта значительно упрощает поиск причин низкой производительности, и в этом случае единственной реальной проблемой остается оптимизация кода. Например, если метод состоит из единственного гигантского цикла `for`, можно с уверенностью утверждать, что проблема связана либо с количеством итераций, либо со временем выполнения отдельных итераций.

Конечно, значительная часть кода не всегда отличается ясностью, поэтому придется время от времени выполнять профилирование подозрительных участков кода. Иногда *грубые, неуниверсальные* решения неизбежны, и не всегда появляется время вернуться назад и реорганизовать код, чтобы привести его в соответствие с рекомендуемыми методиками.

Легко упустить из виду очевидное, когда устранение проблемы и оптимизация производительности являются лишь еще одной формой решения той же задачи. Цель использования профилировщиков и анализа собранных ими данных заключается в поиске источника проблемы и определении ее важности. Очень часто можно получить неверные данные или сделать неправильные выводы из-за нетерпения или отсутствия желания разобраться в тонкостях. Часто при отладке программного обеспечения выясняется, что причину можно было найти гораздо быстрее, если остановиться и еще раз проверить сделанные ранее предположения. Уверенность, что проблема весьма