Санкт-Петербургский государственный университет

Кафедра системного программирования

Щербаков Александр Сергеевич

Оптимизация алгоритма коррекции искажений линз в шлемах VR

Курсовая работа

Научный руководитель:
Чурилин К. С.

Санкт-Петербург
2016

Оглавление

Введение **3**

1. Постановка задачи **4**

2. Обзор существующего решения **5**

2.1. Модель искажений Брауна-Конради 5

2.2. Шейдеры 7

2.3. Fibrum SDK 7

3. Алгоритмы коррекции **9**

3.1. На основе полигональной сетки 9

3.2. На основе изменения положения вершин 9

3.3. Преимущества и недостатки ”вершинного” подхода 10

4. Реализация **12**

5. Эксперементальное исследование **14**

Заключение **16**

Список литературы **17**

Введение

Искажения, которое можно увидеть на экране приложений виртуальной реальности, компенсируют физические искажение стеклянных линз в шлемах VR.

Линзы в HMD(Head-Mounted Displays)[5] необходимы для следующих целей:

• Во-первых, они увеличивают поле зрение пользователя как по горизонтали так по вертикали, что позволяет закрыть область обоих глаз.

• Во-вторых, линзы дают возможность располагать экран близко к лицу, за счет чего возможно комфортно смотреть на дисплей, расположенный на малом расстоянии.

Однако, использование линз несет в себе недостатки. Из-за естественного их искажения, некоторые части изображения сжимаются, и, например, текст на изображении может стать нечитаемым. По той же причине по краям изображения наблюдаются цветовые искажения.

Появление шлемов виртуальной реальности, породило вопрос оптимизации приложений для них. Графические требования для VR-приложений выше, потому что на GPU возлагается намного больше работы по сравнению с обычными мобильными приложениями. Одна из причин возросшей рабочей нагрузки заключается в необходимости фильтрации искажений линз. Этот фильтр исполняется на графическом процессоре, нагружая его наряду с построением двух изображений для каждого из глаз вместо одного.

В данной курсовой работе был использован российский экземпляр мобильного шлема VR – Fibrum. Фильтрация искажений, создаваемых линзами в данном шлеме, применяется отдельно для каждого глаза и составляет наибольшую часть нагрузки VR приложения в связи с необходимостью постообработки каждого пикселя итогового изображения. Производительность такого решения оказалась неудовлетворительной. Таким образом, целью данной работы является исследование возможности улучшения производительности VR-приложения за счет оптимизации алгоритма коррекции искажения линз в шлемах VR.

1. Постановка задачи

Целью данной работы являлось исследование возможности улучшения производительности VR-приложения за счет оптимизации алгоритма коррекции искажения линз шлемов VR. Для достижения данной цели были поставлены следующие задачи:

- Исследовать алгоритмы коррекции искажения линз в VR шлемов
- Реализовать и интегрировать в Fibrum SDK[3] более оптимальный алгоритм
- Сравнить производительность реализаций текущего и выбранного алгоритмов

2. Обзор существующего решения

2.1. Модель искажений Брауна-Конради

Несмотря на то, что искажения изображений могут быть неравномерными или следовать сразу нескольким шаблонам, наиболее часто встречающийся вид дисторсий — радиальные искжаения, которые делятся на бочкообразные и подущкообразные. Линзы применяют к изоб-

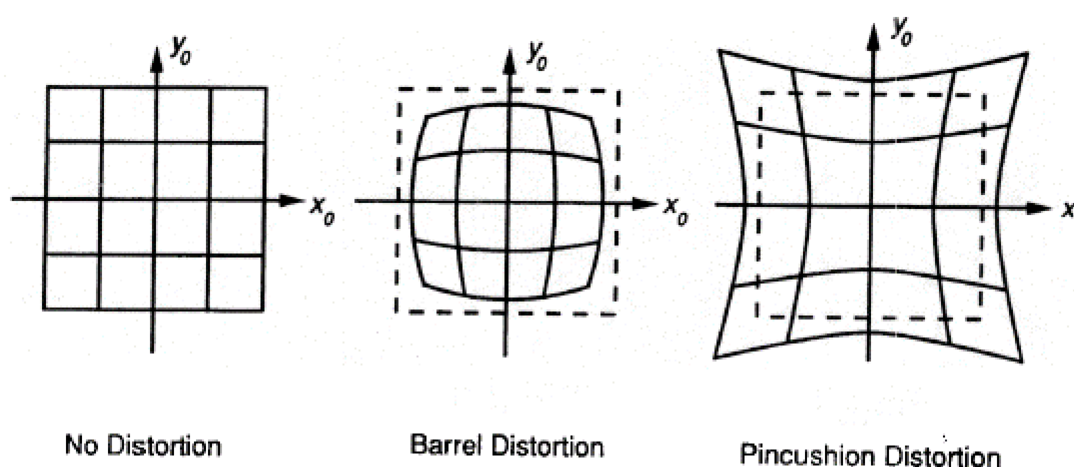

Рис. 1: Радиальные искажения

ражению положительную дисторсию(подушкообразное искажение).[9] В самом центре каждого из двух изображений VR-шлема нет искажения, но при увеличении угла между центром линзы и пользовательским взглядом, уменьшается фактический угол к экрану. Если мы посмотрим на 45° влево, линза исказит поступающий свет, так что мы будем видеть то, что находится всего в 30° от центра. Данные значения зависят от характеристик линз. Шлем Fibrum[3] обеспечивает угол зрения до 110°.

Для того, чтобы компенсировать данный вид искажения, нам надо применить к изображению отрицательную дисторсию(бочкообразное искажение).[1]

Идея состоит в том, чтобы заставить каждую из вертикальных и горизонтальных линий, искаженных линзой вовнутрь, выгнуться наружу на точно такое же значение. Таким образом выполняется обратное

преобразование. Его эффект состоит в том, что оставляя за линзой возможность увеличения обзора и расположения экрана близко к лицу, пользователь может наблюдать неискаженное изображение.

Как было сказано выше, искажение, даваемое линзой, зависит от ее физической конструкции. Есть один общий метод его коррекции - модель искажений Брауна-Конради. Модель Брауна-Конради предна-

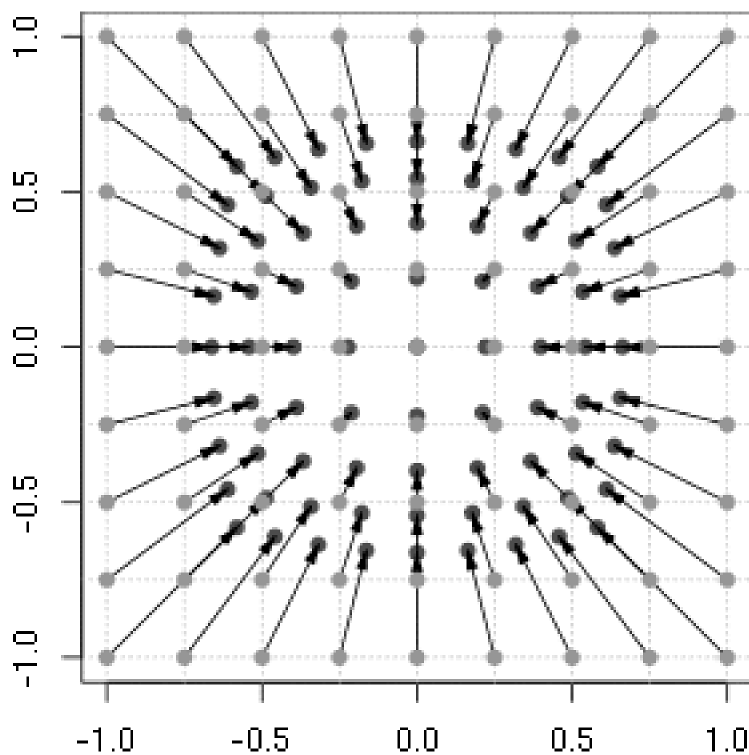

Рис. 2: Бочкообразное искажение

значена для коррекциии как радиальных искажений линз, обосновываемых физическими свойствами, так и для тангенциальных искажений, возникающих из-за их деффектов. Нас интресует только радиальные искажения:

$X_d = X_u * (1 + K_1 * R^2 + K_2 * R^4 + ...)$

$Y_d = Y_u * (1 + K_1 * R^2 + K_2 * R^4 + ...)$

(X_d, Y_d) - искаженное положение точки для указанной линзы

(X_u, Y_u) - неискаженное положение точки, как в случае, если бы линза не давала искажения.

K_n – n-ый коэфицент радиальной дисторсии

R – расстояние от центра искажения до текущей точки.

Для бочкообразного искажения K_1 отрицателен, тогда как для подушкообразного положителен. На основе этих преобразований построено текущее решение.

2.2. Шейдеры

В текущем решении для фильтрации искажений используется шейдер - программа, исполняемая на графическом процессоре. Шейдеры очень сильно влияют на Fill rate – число пикселей в секунду, для которых GPU может выполнить необходимые операции(просчитать освещение,сглаживания изображения и др.), так как добавляют еще один слой рендеринга к общему процессу.

Набольшую опасность представляют шейдеры постэффектов, применяемые ко всему экрану. Так обстоит дело с шейдером для коррекции изображения в Fibrum SDK[3]. Из-за того, что расчеты в шейдере производятся над каждым пикселем(т.е. заново вычисляется цвет и прозрачность пикселя), достаточно сильно понижается производительность: при разрешении 1920*1080 в VR приложении шейдер будет запускаться около 2 миллионов раз, не считая накладных расходов на копирование текстуры, обработки перекрытой геометрии, расходов на переключение контекста.

При частоте кадров 60fps, необходимой для комфортного использования VR-шлемом, на расчет одного кадра отводится не более 15ms, что предъявляет высокие требования производительности к данному шейдеру.

2.3. Fibrum SDK

На текущий момент в Fibrum SDK[3] используется следующее решение: сначала изображение с каждой из двух виртуальных камер(для обоих глаз) рендерится в текстуру. Далее в пиксельном шейдере для каждого из пикселей по формулам расчитывается его новое место с

учетом инверсии искажения линз. После этого изображение копируется в экранный буффер и выводится на дисплей.[6]

Недостатки данного подхода очевидны:

• Во-первых, пиксельный шейдер для каждого из 2-х миллионов пикселей экрана проводит 5 операций умножения, 3 операции сложения и 1 операцию извлечения квадратного корня.

• Во-вторых, копирование полноэкранного изображения на текстуру не является дешевой операцией, из-за чего на старых, маломощных устройствах и устройствах с высоким разрешением экрана могут наблюдаться проблемы с производительностью.

3. Алгоритмы коррекции

После анализа литературы было выделено два решения, которые могли бы пойти на замену текущему.

3.1. На основе полигональной сетки

Первое решение основано на переносе вычислений из пиксельного шейдера в вершинный. Для этого потребуется две плоскости, обладающие достаточной плотностью(например, 32 пикселя на полигон на расстоянии одноу у.е. от камеры). Для FullHD дисплея в данной полигонально сетке будет около 12000 вершин.

Эти плоскости исказим по закону, который будет имитировать инверсию искажения линзы. После чего необходимо наложить текстуры с изображений камер на каждую из них.[2, 7]

Даже если проводить эти операции в момент рендеринга изображения, то в итоге получится 12к проходов шейдера вместо 2м. В таком случае как и в текущем решении останутся расходы на копирование текстуры, чтобы в дальнейшем поместить ее на сетку, и появится расходы на рендеринг итогового изображения. Как вариант оптимизации, возможно вычислить искажения каждой плоскости лишь единожды и использовать их в дальнейшем . Но даже в таком случае расходы на копирование текстуры с камеры на полигональную сетку все еще высоки.

3.2. На основе изменения положения вершин

Второе решение, основано совсем на другом принципе.

Рендеринг изображения в данном алгоритме происходит в следующем порядке: преобразование вершин —> копирование изображения в экранный буфер —> вывод изображения. [4, 8]

Вместо того, чтобы преобразовывать отдельные пиксели из отрендеренного изображения, будем изменять расположение вершин объектов окружающего мира. Чем ближе прямая линия находится к камере,

тем более изогнутой она должна выглядеть на экране. Таким образом, оставляя модель Брауна-Конради, мы вносим в нее единственное изменение: расстояние от центра искажение до искаженной точки расчитывается с учетом расстояния от вершины до камеры.

3.3. Преимущства и недостатки "вершинного" подхода

Преимущества подхода на основе изменения положения вершин объектов очевидны:

Во-первых, шейдер преобразует только вершины объектов, которых на экране гораздо меньше, чем пикселей. Так, например, компания Oculus предлагает придерживаться не более 50-100к полигонов или 150-300к вершин на кадр. Это становится возможным благодаря возможности не отрисовывать геометрию, которая не попадает в область действия камеры. При этом, на сцене может быть гораздо больше вершин, чем в ее поле зрения.

Во-вторых, для преобразования вершин нет необходимости в копировании текстуры по два раза, как это было в предыдущих вариантах, что сокращает одну из значимых статей накладных расходов в алогоритме.

У данного подхода есть и недостатки: искажение нелинейно, и, следовательно, обратное искажение линзы также нелинейно. Когда мы искажаем вершины в вершинном шейдере, прямая между ними остается прямой, в то время когда на самом деле она должна стать изогнутой дугой.

Т.о, если вершины расположены далеко друг от друга в пространстве экрана, то искажение в конечном итоге не будет в должной мере скорректировано и итоговое изображение будет выглядеть не таким, каким его представляли.

Решение данной проблемы очевидно: современные устройства могут обрабатывать до 400к вершин на глаз для одного кадра, поэтому необходимо увеличить количество вершин и найти баланс между коли-

чеством вершин и производительностью.

Можно подумать, что с увеличением числа веришин мы опять вернемся к тем же проблемам, от которых пытались уйти. Но на самом деле современные смартфоны имеют экраны с высокой плотностью - до 1920x1080 пискелей(около 2 миллионов пикселей). Учитывая то, что несмотря на отсечение, часть перекрытой геометрии все равно будет визауализироваться, может получиться до 10 миллионов итераций пиксельного шейдера на кадр. Кроме того, необходимо скопировать изображение с камеры в отдельную текстуру, что добавляет еще обработку двух миллионов пикселей. По сравнению с этими расходами, визуализация 600к вершин выглядит гораздо менее затратно.

4. Реализация

Fibrum SDK поставляется как плагин к среде разработке Unity. Таким образом он является и библиотекой, и интерфейсным решением для данной среды разработки. Было принято решение продолжить разработку Fibrum SDK с сохранением данной идеи – вместе с алгоритмами исправления искажений линз дописать интерфейсный модуль, который позволит выбирать необходимый алгоритм и видеть изменения прямо в редакторе.

Изначально алгоритм коррекции искажений линз был написан в одном из классов, который инициализировался в классе виртуальной камеры, проверял, нужно ли использовать коррекцию искажений и накладывал шейдер коррекции искажений на изображения кадра. Вся остальная логика, касаемая самого искажения, в том числе и его коэффициенты, были описаны в шейдере без каких-либо комментариев относительно работы искажения.

По причине того, что, во-первых, были внедрены два новых алгоритма, отличающиеся по своему принципу действия от текущего, а во-вторых, в связи с желанием разрабатывать не только под шлем Fibrum, мною были реализованы две возможности:

- Стало возможным выбирать алгоритм коррекции искажений из выпадающего списка в интерфейсе Unity.
- При изменении параметров линз или появлении новой версии шлема внести их коэффициенты искажения, получив правильное изображение

Для этого на C-sharp был реализован класс, в который бы заносилась вся информация о шлеме, и методы, получающие необходимые коэффициенты искажения по заданному профилю шлема. Данная особенность была успешно протестирована на Google Cardboard, чьи характеристики искажения линз я нашел в Cardboard SDK. В последствии сюда можно внедрить профили наиболее популярных смартфонов с указанием характеристик экрана для этого чтобы улучшить изображения под них: на текущий момент изображение под разные экраны

может выглядеть по-разному, несмотря на то, что шлем позиционируется как универсальный. Все математически вычисления, для расчета коэффициентов искажения изображения по данным характеристикам линз также вынесены в отдельный класс.

Данные характеристики и коэффициенты передаются в шейдеры, написанные на языке Cg/HLSL, в которых и происходит работа по коррекции искажений.

Для алгоритма на основе искаженных плоскостей, текущий шейдер не подходит, так как необходимо вычислять искажения до начала основных вычислений, что шейдер сделать не в состоянии, поэтому для этого был написан отдельный класс на C-sharp.

Отдельно для вершинного алгоритма также добавлена возможность, назначая в редакторе Unity объектам тег World, включать и отключать искажение для отдельных объектов. Это может быть необходимо для правильной подачи элементов интерфейса, для которых использование стандартного искажения часто ведет к ухудшению пользовательского опыта.

5. Эксперементальное исследование

Сравнение производительности текущего и двух новых алгоритмов было произведено на заранее построенных статичных виртуальных сценах. Данные сцены были подобраны таким образом, чтобы в кадр попадало 600к, 300к, 150к и 75к вершин и соответственно производилось 215, 135, 90, 60 вызовов отрисовки. За основу бралось среднее время построение кадра за 600 кадров, что при 60fps составляет 10 секунд. Матическое ожидание алгоритмов можно увидеть в таблице на рис.3. Среднеквадратическое отклонение для наиболее мощного устройства на текущем алгоритме не превышет 3% от матожидания, а для наименее мощного - 20%. Среднеквадратическое отклонение для наиболее мощного устройства на вершинном алгоритме не превышет 2.4% от матожидания, а для наименее мощного - 8%.

Замеры проводились на четырех устройствах: Nexus7, Sony Xperia Z2, Samsung Galaxy S4 и Asus300. Результаты измерений в таблице на рис3 показывают, что реализации алгоритма на основе изменения положения вершин дает существенный прирост производительности на данных синтетических примерах, в то время как алгоритм на основе полигональной сетки дает минимальный выигрыш или не дает совсем.

	75K Vertices, ms0			150 Vertices, ms			300 Vertices, ms			600 Vertices, ms		
	Current	Plane	Displacement	Current	Plane	Displacement	Current	Plane	Displacement	Current	Plane	Displacement
Nexus7	29	30	31	29	31	35	33	34	42	51	36	60
Xperia Z2	11	16	19	16	13	18	20	16	24	27	24	30
Galaxy S4	14	17	32	18	24	34	19	29	46	29	38	78
ASUS TF 300	22	77	121	29	156	184	50	389	350	91	560	420

Рис. 3: Сравнение времени формирования кадра с использованием различных алгоритмов

На выборке с 600к вершинами ускорение (в количестве раз) для самого слабого из устройств в «вершинной» реализации по сравнению

с текущей составляет 5 раз, в то время как для самого мощного из устройств практически незаметен. Следует отметить, что мы оцениваем время формирования кадра в целом, а не сравниваем скорость работы двух алгоритмов: количество вершин в кадре влияет как на скорость работы данного алгоритма, так и на скорость формирования кадра без учета искажения.

Принцип вычисления искажения в текущей реализации практически идентичен новому за исключением того момента, что расстояние от центра искажения до нового расположения вершины на экране вычисляется с учетом расстояния от камеры до этой вершины. Вычисления же обратных коэффициентов искажения по коэффициентам искажения линз производится до запуска виртуальной сцены. Таким образом, скорость работы почти линейно зависит от количества вершин в кадре, а ускорение по сравнению с текущей версией – от количества видимых и overdraw пикселей на экране, а также расходов на копирование кадра. В текущей реализации на синтетических примерах получилось около 4м запусков пиксельного шейдера на кадр, несмотря на то, что данное число может доходить на 10м. Таким образом ускорения алгоритма на моих примерах для FullHD экрана: от 40 раз на 75к вершин до 7 раз для 600к вершин.

Что касается алгоритма на основе полигональной сетки, то копирование кадра на текстуру оказалось дорогой операцией, перекрывающей преимущества идеи. Однако несмотря на данных тестах результаты показали выигрыш от 1.2 до 2 раз.

Заключение

В ходе выполнения данной работы получены следующие результаты:

• Исследованы алгоритмы коррекции искажения линз в VR шлемов

• Более оптимальный алгоритм реализован и интегрирован в Fibrum SDK[3], несмотря на выявившиеся недостатки.

• Произведено сравнение производительности реализаций алогритмов коррекции радиального искажения линз в VR-шлемах.

Алгоритм, дающий наибольшую производительность, в отличие от текущего подхода и подхода на основе полигональной сетки, перекладывает на плечи разработчика обязанность встраивать код в каждый объект, искажение которого мы хотим исправить. Из-за того, что мы меняем расположение вершин объектов, меняется направление нормалей плогинов, что может повлиять на расчет освещения и другие эффекты, которые были ранее встроены в VR-приложение. Кроме того, появляется необходимость использовать модели с большим числом вершин. Поэтому, могут возникнуть трудности с встраиванием данного метода в готовое VR-приложение. Однако увеличение частоты кадров и общей призводительности перекрывает данные недостатки.

Список литературы

[1] Barrel distortion. — URL: http://github.prideout.net/barrel-distortion.

[2] Correction of lens distortion for the Oculus Rift using LabVIEW. — URL: https://decibel.ni.com/content/docs/DOC-42762.

[3] Fibrum SDK repository. — URL: http://fibrum.com/sdk/.

[4] Google cardboard repository. — URL: https://github.com/itp-vr/osc-demo/tree/master/Assets/Cardboard.

[5] Head-mounted display. — URL: https://en.wikipedia.org/wiki/Head-mounted_display.

[6] OpenVR SDK repository. — URL: https://github.com/ValveSoftware/openvr.

[7] Understanding the Oculus Rift Distortion Shader. — URL: http://rifty-business.blogspot.ru/2013/08/understanding-oculus-rift-distortion.html.

[8] VR Distortion Correction using Vertex Displacement. — URL: http://www.gamasutra.com/blogs/BrianKehrer/20160125/264161/VR_Distortion_Correction_using_Vertex_Displacement.php.

[9] Wikipedia. Distortion(optics). — URL: https://en.wikipedia.org/wiki/Distortion_(optics).