

ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

Отчет о программном проекте на тему:

Оптимизация рендеринга в компьютерных играх на основе трекинга взгляда

Выполнили студенты:

группы #БПМИ236, 2 курса Хорт Матвей Дмитриевич
группы #БПМИ234, 2 курса Щербаков Алексей Дмитриевич

Принял руководитель проекта:

Галицкий Борис Васильевич
Приглашенный преподаватель
Факультет компьютерных наук НИУ ВШЭ

Москва 2025

Содержание

1 Аннотация	5
2 Введение	6
2.1 Разделение задач по участникам	6
2.2 Терминология	6
2.3 Foveated Rendering	8
2.4 Обзор существующих решений	8
2.5 Проблемы разработки для ПК	9
3 План реализации foveated rendering	11
3.1 Variable Rate shading	11
3.1.1 Преимущества	11
3.1.2 Недостатки	12
3.1.3 Нагрузка на ресурсы	12
3.1.4 Шаги реализации	12
3.2 LOD (Level of Detail) Based Foveated Rendering	12
3.2.1 Преимущества	13
3.2.2 Недостатки	13
3.2.3 Нагрузка на ресурсы	14
3.2.4 Шаги реализации	14
3.3 Особенности разработки в Unity	14
3.3.1 Built-in Render Pipeline (BiRP)	14
3.3.2 Universal Render Pipeline (URP)	17
3.3.3 High Definition Render Pipeline (HDRP)	18
4 План реализации Gaze tracking	19
4.1 Актуальность технологии и сценарии применения	19
4.2 Доступность технологии массовому пользователю	19
4.3 Обзор подходов к реализации	20
4.4 Современный подход	20
4.5 Сложности appearance-based подхода	21
4.6 Гибридные решения и перспективы развития	22
4.7 Современное состояние исследований	23

5 Реализация Gaze Tracking	24
5.1 EyeGestures v3	24
5.2 Интеграция в Unity	25
5.3 Eyeware Beam Eye Tracker	26
5.4 Базовый пример без использования ML	27
5.5 Исторический контекст создания MediaPipe	28
5.5.1 MediaPipe Face Mesh и отслеживание взгляда	28
5.6 Сравнение трех подходов	29
6 Реализация VRS на стороне Unity	30
6.1 Общий Workflow	30
6.2 Получение данных о взгляде	31
6.3 Интеграция с BiRP	31
6.4 Интеграция с URP	31
6.5 Настройка <code>foveal regions</code> и разрешения растеризации	32
7 Реализация VRS на стороне нативного плагина	32
7.1 Основные компоненты плагина	32
7.2 Инициализация плагина и подключение к Unity	32
7.3 Обработка данных взгляда	33
7.4 Применение VRS	33
7.5 Обработка <code>rendering events</code> и очистка	34
8 Реализация LOD Foveated rendering	34
8.1 Общий workflow	35
8.2 Заключение и важность всех методов	36
9 Настройка демо сцен	36
9.1 Сцена Night City [7]	36
9.2 Сцена Mountains [1]	36
10 Результаты	37
11 Перспективы развития для метода VRS	39
11.1 Luminance-Contrast-Aware Foveated Rendering	39
11.2 Сглаживающие фильтры	39

11.3 Upscaling	40
Список литературы	41

1 Аннотация

В VR широко используется оптимизация под названием Foveated Rendering, которая значительно повышает эффективность рендеринга. Основная идея заключается в том, чтобы отображать небольшую область, на которую направлен взгляд игрока (foveal region), с высоким уровнем детализации, в то время как остальные участки отображаются с пониженным уровнем детализации. Такой подход основан на особенностях человеческого зрения: мы менее детально различаем объекты, расположенные за центральной областью поля зрения.

Данная оптимизация применяется только VR и AR шлемах и обеспечивает существенный прирост FPS. Цель нашего проекта — адаптировать Foveated Rendering для использования в обычных компьютерных играх на стандартных мониторах. Для этого мы будем использовать вебкамеру для отслеживания направления взгляда пользователя, что позволит аналогичным образом динамически изменять детализацию рендеринга в зависимости от направления взгляда.

С каждым годом графика в компьютерных играх входит на новый уровень, и требует все большей и большей производительности. Поэтому, если наша идея сработает, то данная оптимизация может оказаться крайне актуальной. Результат проекта - плагин для графического движка Unity, который можно будет легко подключить и использовать в любом проекте.

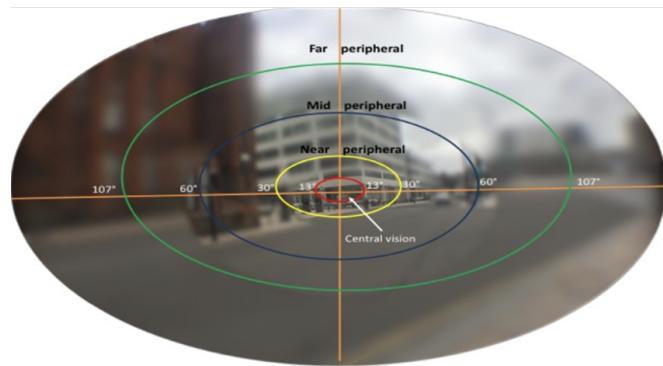


Рис. 1.1: Пример работы статического foveated rendering в VR шлеме

Ключевые слова

Foveated Rendering, Gaze Tracking, Unity, Variable Rate Shading, Eye Gestures

2 Введение

2.1 Разделение задач по участникам

Хорт Матвей будет отвечать за отслеживание направления взгляда человека на мониторе и подключением этой части к Unity. Часть отчета, посвященная Gaze Tracking была написана Матвеем.

Щербаков Алексей будет отвечать за разработку Foveated Rendering в Unity. Часть отчета, посвященная Foveated Rendering в контексте Unity была написана Алексеем.

2.2 Терминология

Фовеа (Fovea), фовеальная область (Foveal region) Область сетчатки глаза с наивысшей остротой зрения.

Периферийная область (Peripheral region) Область человеческого зрения, расположенная за пределами foveal region.

Фовеативный рендеринг (Foveated rendering) Общий термин для всех оптимизаций, которые снижают качество рендеринга в peripheral region зрения с целью повышения производительности.

Отслеживание взгляда (Gaze tracking) Совокупность алгоритмов, позволяющих определить положение зрачков относительно глаза, чтобы определить направление взгляда.

Динамический фовеативный рендеринг (Dynamic Foveated rendering) Метод рендеринга, при котором foveal region и peripheral region определяются с помощью отслеживания направления взгляда и автоматически обновляются в реальном времени.

Статический фовеативный рендеринг (Static Foveated rendering) Метод рендеринга, при котором foveal region фиксирован в центре экрана и не изменяется в зависимости от направления взгляда пользователя.

Variable Rate Shading Технология от Nvidia, которая позволяет задавать разрешение рендеринга отдельно для каждой группы пикселей.

Unity Популярный кросс-платформенный игровой движок, который используется для создания 2D и 3D игр, а также интерактивных приложений. Он поддерживает множество

платформ, включая Windows, macOS, Android, iOS, консоли и платформы для виртуальной реальности. Unity предоставляет мощный набор инструментов для разработки, включая редактор сцен, систему физики, шейдеры, анимацию, поддержку многопользовательских игр и многие другие возможности. Unity широко используется как для инди-разработки, так и для крупных коммерческих проектов.

Rendering Pipeline Процесс в Unity, через который проходят все объекты сцены для того, чтобы быть визуализированными на экране. Он включает в себя несколько этапов, которые могут быть настроены и модифицированы в зависимости от нужд игры или приложения.

Frames Per Second (FPS) Количество кадров, которые успевают отрендериться за секунду. Чем больше FPS, тем лучше воспринимает игровой процесс.

Нативный плагин (Native plugin) Unity Плагин, скомпилированный в виде динамической библиотеки, с которым можно взаимодействовать через Unity.

Командный буфер (Command Buffer) Структура данных в Unity, которая используется на BiRP и позволяет создавать и управлять низкоуровневыми командами рендеринга.

ScriptableRendererFeature Компонент, используемый в URP, который позволяет добавлять пользовательские шаги рендеринга в граф рендеринга.

Webcam-based Это подход, при котором для выполнения задач компьютерного зрения используется стандартная веб-камера.

Structure from Motion (SfM) Метод компьютерного зрения, позволяющий реконструировать трёхмерную структуру сцены из последовательности двумерных изображений, снятых с разных ракурсов.

Stable fixation Это состояние, при котором взгляд человека относительно стабилен и фиксирован на определённом объекте или точке.

Саккады Быстрые, резкие перемещения глаз между разными точками.

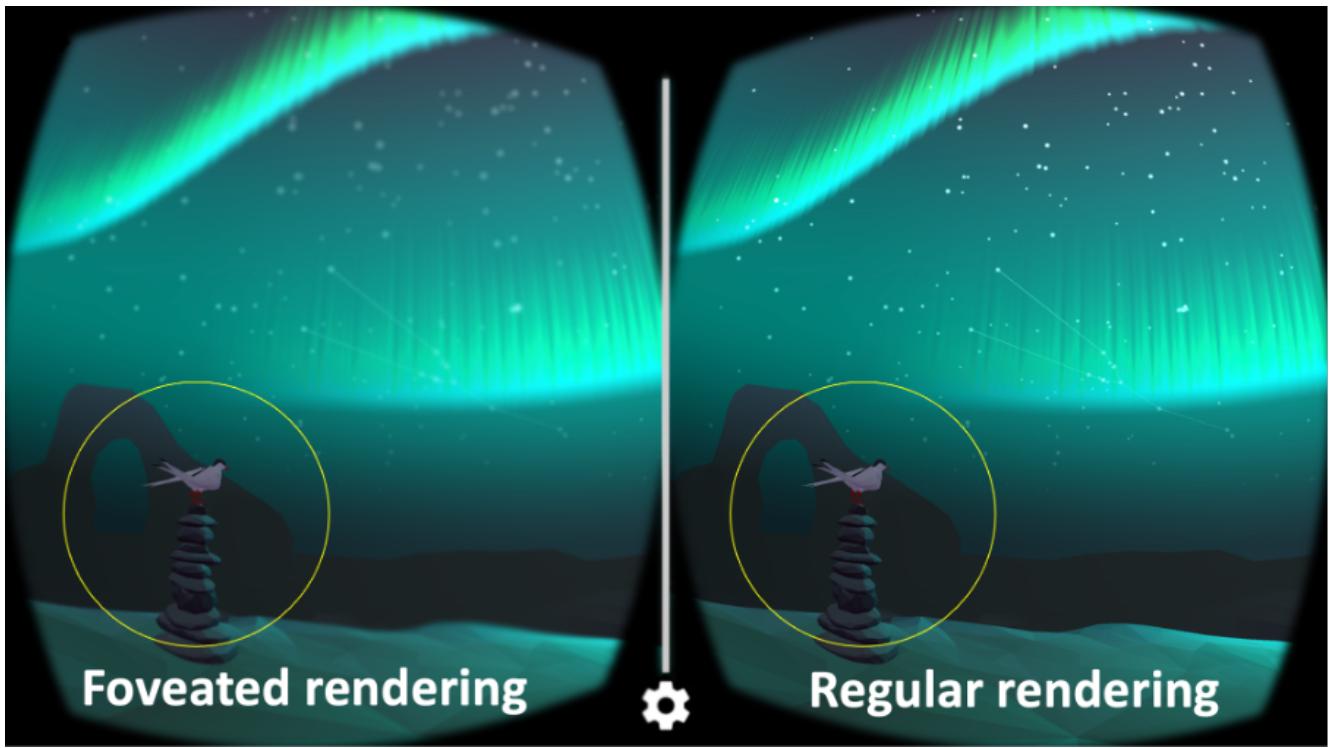


Рис. 2.1: Пример работы динамического Foveated rendering

2.3 Foveated Rendering

Человеческий глаз имеет центральную область сетчатки, называемую fovea. Эта зона охватывает примерно 1° - 2° углового обзора [2] и отвечает за крайне детальное восприятие объектов, на которые направлен взгляд. Периферийное зрение менее острое, но важное для общего восприятия сцены. Именно эта особенность зрения легла в основу технологии foveated rendering.

В устройствах виртуальной (VR) и дополненной реальности (AR), оснащенных дисплеями высокого разрешения эта технология особенно эффективна, так как дисплеи расположены очень близко к глазам, и переход между foveal region и peripheral будет не сильно заметен. К тому же, во многих шлемах уже есть встроенный gaze tracking, который обладает высокой точностью за счет близкого расположения к глазам. Но даже со статическим foveated rendering пользовательский опыт ухудшается не сильно, учитывая, что foveated rendering, как оптимизация, способная уменьшить нагрузку на GPU на 60-70 % и обеспечить прирост fps до 50 процентов [3].

2.4 Обзор существующих решений

Foveated rendering в VR шлемах используется уже достаточно давно, поэтому в Unity достаточно просто установить несколько плагинов и все начнет работать. Однако использо-

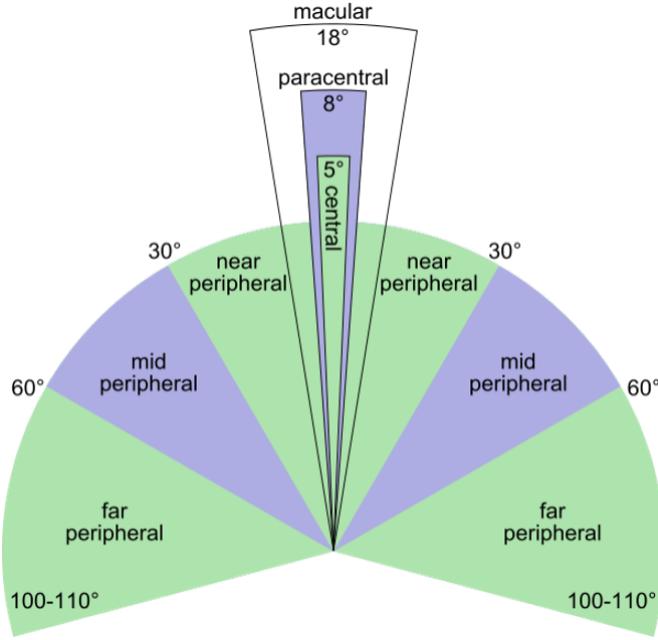


Рис. 2.2: Зоны человеческого зрения

вать те же плагины без VR шлема не получится. Многие из этих плагинов проприетарные, и во многих шлемах оптимизации не программные, а аппаратные (и тоже проприетарные). Поэтому каких либо настраиваемых плагинов с открытым исходным кодом найдено не было.

К сожалению, задача ухудшения рендеринга отдельных частей изображения непроста в реализации, хотя, может быть, и кажется несложной. Проблема в том, что мы хотим как-то на очень низком уровне изменить части процесса рендеринга, что через API движков сделать невозможно. Разработчикам игр никогда не приходится сталкиваться с такой задачей, поэтому такого функционала и не добавлено.

Большинство готовых плагинов для VR шлемов используют технологию Variable Rate Shading (или ее вариации) для достижения динамического foveated rendering. Также мы выделили оптимизацию под названием LOD-based Foveated Rendering, которая пока нигде не используется. В проекте будут рассмотрены эти методы, описание принципов их работы, план реализации, преимущества, недостатки и влияние на системные ресурсы.

2.5 Проблемы разработки для ПК

Как уже говорилось, реализовать Foveated rendering не так уж и просто. Вот некоторые проблемы, с которыми нам придется столкнуться:

- Не получится просто использовать готовые плагины, в которых уже реализован Foveated rendering, потому что они почти всегда используют аппаратные особенности VR платформ. Поэтому придется писать собственный плагин с нуля.

- Теперь дисплей расположен не вплотную к глазам, и foveal region будет занимать большую часть дисплея. То есть прирост производительности будет точно меньше, чем в VR шлемах.
- К тому же, теперь и камера, отслеживающая направление взгляда расположена не вплотную к глазам, и отслеживание направления взгляда будет работать менее точно.

3 План реализации foveated rendering

3.1 Variable Rate shading

Variable Rate Shading (VRS) - это технология от Nvidia, которая способная делать в точности то, что мы и хотим: она регулирует разрешение растеризации в зависимости от области изображения. Метод работает на уровне тайлов (блоков пикселей). Обычно все пиксели разбиваются на блоки 16×16 . Каждому такому блоку можно указать плотность шейдинга: блок может быть отрендерен как 2×2 (подблоки 4×4 склеиваются в один пиксель), как 4×4 , либо как 16×16 (не меняется качество рендеринга). Также есть возможность указать большее разрешение, чем разрешение монитора - таким образом можно достичь методологии под названием anti-aliasing¹. Перед применением метода нужно составить специальную карту и указать каждому блоку разрешение растеризации.



Рис. 3.1: Пример возможностей VRS

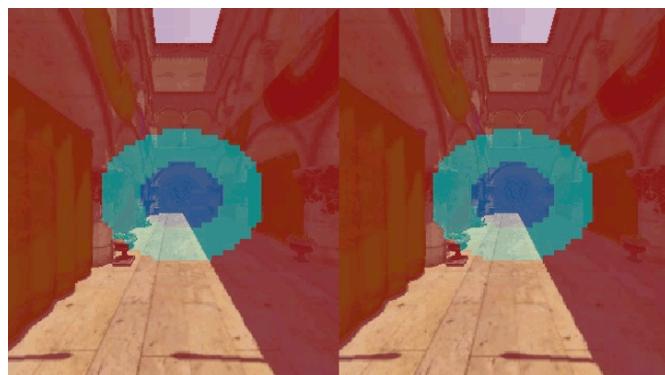


Рис. 3.2: Пример работы VRS для Foveated rendering

3.1.1 Преимущества

- Значительное снижение нагрузки на GPU

¹Антиалиасинг (Anti-Aliasing, AA) в играх — это технология сглаживания зубчатых краёв (aliasing) на объектах и текстурах для улучшения визуального качества изображения. Она устраняет резкие ступенчатые линии на границах, делая картинку более гладкой и реалистичной.

- Гибкость в настройке уровней шейдинга - можно сделать плавный переход между fovea region и peripheral region
- Отличная совместимость со многими компонентами игровых движков

3.1.2 Недостатки

- Требует поддержки на аппаратном уровне, что ограничивает использование на старых видеокартах
- Для разных вендоров необходимо разрабатывать разные решения
- Сложно разрабатывать

3.1.3 Нагрузка на ресурсы

Уменьшает нагрузку на GPU без существенного влияния на CPU и память.

3.1.4 Шаги реализации

- Написать нативный плагин, используя API Unity и Nvidia, который инициализирует VRS и обновляет карту тайлов
- Скомпилировать нативный плагин как динамическую библиотеку Windows
- Используя положение мыши на экране вместо направления взгляда на экране (для демонстрации работы), синхронизировать метод Update нативного плагина и метод Update от Unity.
- Скомпилировать Gaze tracking как динамическую библиотеку Windows.
- Разработать алгоритм калибровки направления взгляда.
- Синхронизировать VRS и Gaze tracking.

3.2 LOD (Level of Detail) Based Foveated Rendering

Level of Detail (LOD) — традиционный метод оптимизации рендеринга, который изменяет уровень детализации объектов в зависимости от их расстояния до камеры. Чтобы

применить данную оптимизацию у каждого объекта должно быть 3-4 модели с разным разрешением. Например, первая содержит 1000n треугольников и прекрасную текстуру, а четвертая содержит n треугольников и плохую текстуру. Для каждого объекта рассчитывается расстояние до камеры и в зависимости от этого выбирается нужная модель.

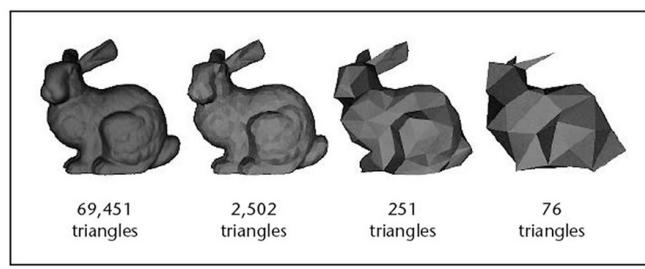


Рис. 3.3: Пример моделей разных уровней детализации для LOD оптимизации

Мы хотим попробовать использовать схожий подход для Foveated rendering - для каждого объекта рассчитывать LOD не только на основе расстояния до камеры, но и на основе расстояния от положения объекта на экране до положения взгляда на экране.

3.2.1 Преимущества

- Может быть эффективен в сложных сценах с большим количеством объектов
- Снижение нагрузки на GPU за счет уменьшения детализации в периферии
- Работает на всех GPU и на всех архитектурах

3.2.2 Недостатки

- Могут быть ощутимые артефакты (Например, на рисунке 3.3, переход от первого к четвертому зайцу может быть заметен даже в периферии).
- Нужно с самого начала процесса разработки учитывать, что для каждого объекта нужны 3-4 модели (хотя в большинстве игр и так используется LOD).
- Увеличенное использование памяти для хранения различных уровней детализации и сложность в реализации плавных переходов между уровнями. (Обычно LOD level не может измениться быстро. Но с foveated rendering мы можем быстро перевести взгляд в угол монитора. Поэтому видеокарте либо придется быстро загружать нужную модель с диска в видеопамять, что плохо. Либо хранить в видеопамяти больше моделей, что тоже плохо)

3.2.3 Нагрузка на ресурсы

Снижает нагрузку на GPU за счет использования менее детализированных объектов в периферии, но увеличивает использование памяти для хранения нескольких уровней детализации.

3.2.4 Шаги реализации

- При инициализации получить список всех объектов, у которых уже есть несколько моделей.
- В методе Update перерасчитывать для каждого объекта необходимых LOD.
- Использовать положение мыши на экране вместо направления взгляда на экране для демонстрации работы.
- Скомпилировать Gaze tracking как динамическую библиотеку Windows.
- Разработать алгоритм калибровки направления взгляда.
- Синхронизировать LOD и Gaze tracking.

3.3 Особенности разработки в Unity

Unity предлагает три основных рендеринг-пайплайна, каждый из которых предназначен для различных типов проектов и обладает своими уникальными особенностями. Рассмотрим их в контексте реализации foveated rendering.

3.3.1 Built-in Render Pipeline (BiRP)

Built-in Render Pipeline (BiRP) — классический пайплайн, используемый во многих старых проектах Unity (примерно до 2020 года). Он предоставляет стандартные инструменты рендеринга, которые были включены изначально в Unity. Хотя этот пайплайн гибок, он не обладает встроенной поддержкой современных технологий оптимизации и не позволяет удобно вмешиваться в процесс рендеринга.

Особенности:

- Универсальность и широкая поддержка старых проектов: BiRP поддерживает множество старых шейдеров и инструментов, что облегчает работу над существующими проектами.

Spotlight on Render pipelines

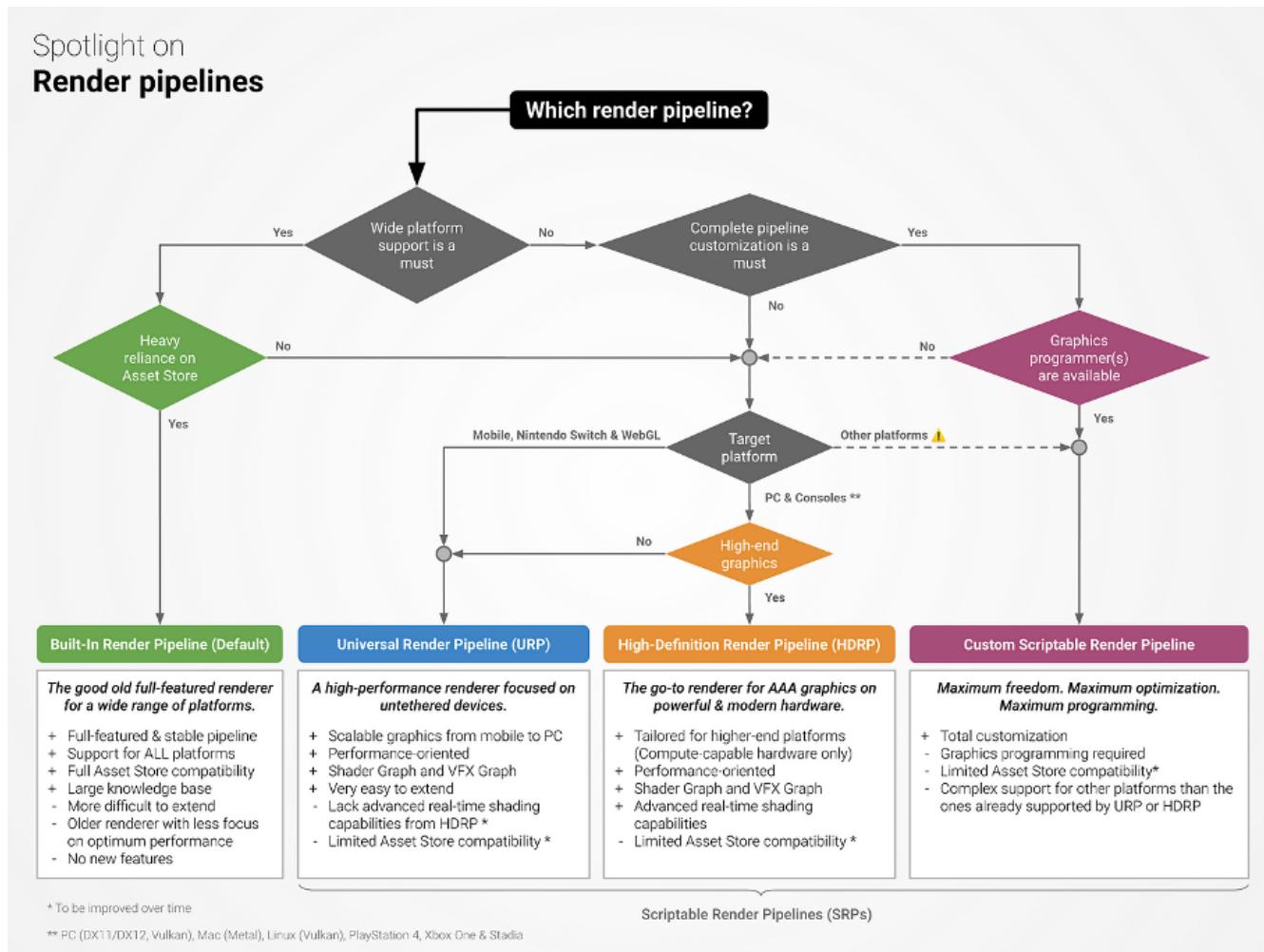


Рис. 3.4: Различные Rendering Pipelines для различных целей

- Ограниченные возможности изменения процесса рендеринга: возможно только через создание командных буферов.
- Простота использования и интеграции: BiRP легко настраивается и интегрируется в проекты без необходимости глубоких знаний о внутренней архитектуре рендеринга.
- Поддержка стандартных шейдеров и материалов: широкий набор стандартных шейдеров и материалов позволяет быстро создавать визуальные эффекты без дополнительной настройки.
- Широкий набор инструментов для разработчиков: предоставляет базовые инструменты для работы с освещением, тенями и пост-обработкой.

Преимущества BiRP:

- Широкая совместимость с существующими проектами: позволяет легко поддерживать и обновлять старые проекты без необходимости полной переработки рендеринга.
- Простота настройки для начинающих: интуитивно понятный интерфейс и минимальная необходимость в кастомизации делают BiRP подходящим для новичков.
- Обширная документация и сообщество: большое количество обучающих материалов и активное сообщество облегчают решение возникающих проблем.

Недостатки BiRP:

- Низкая производительность на современных устройствах: BiRP не оптимизирован для использования современных графических возможностей и может быть менее эффективен по сравнению с новыми пайплайнами.
- Ограниченная поддержка современных графических технологий: такие технологии, как физически корректное освещение (PBR), теневые карты высокой точности и пост-обработка, могут быть недоступны или сложнее в реализации.
- Сложность кастомизации: внесение изменений в процесс рендеринга требует глубоких знаний внутренней архитектуры Unity и рендеринга в целом.
- Меньшая масштабируемость: BiRP может испытывать трудности при адаптации под различные платформы и устройства, особенно мобильные и VR/AR.

3.3.2 Universal Render Pipeline (URP)

Universal Render Pipeline (URP) был представлен в Unity как современная замена BiRP, ориентированная на обеспечение высокой производительности и гибкости на различных устройствах, включая мобильные платформы и VR/AR устройства. URP предоставляет улучшенные инструменты для оптимизации рендеринга и гибкого управления качеством изображения. Однако обратная совместимость с BiRP не сохранена, что требует миграции проектов для использования URP.

Особенности:

- Высокая производительность и оптимизация для широкого спектра устройств: URP разработан с учетом современных требований к производительности, что позволяет использовать его на разнообразных платформах без значительных потерь качества.
- Поддержка современных технологий: URP включает поддержку современных графических технологий, таких как физически корректное освещение (PBR), улучшенные теневые карты, пост-обработка и эффекты частиц.
- Гибкость в настройке параметров рендеринга через Renderer Features и Rendering Passes: URP предоставляет возможности для настройки различных этапов рендеринга.
- Scriptable Render Pipeline (SRP): URP основан на SRP, что дает разработчикам возможность полностью настраивать процесс рендеринга с помощью скриптов, обеспечивая высокую степень кастомизации. Однако степень кастомизации ограничена и может потребовать написания нативных плагинов для специфических задач.
- Поддержка Shader Graph: интеграция с Shader Graph позволяет создавать сложные шейдеры визуально, без необходимости написания кода.

Преимущества URP по сравнению с BiRP:

- Лучшая производительность: благодаря оптимизациям и современным методам рендеринга, URP обеспечивает более высокую производительность, особенно на мобильных и VR/AR устройствах.
- Гибкость и расширяемость: возможность добавления собственных Renderer Features и Rendering Passes позволяет легко внедрять новые эффекты и оптимизации.

- Совместимость с современными инструментами Unity: интеграция с Shader Graph и другими современными инструментами облегчает процесс разработки и улучшает качество графики.
- Упрощенная настройка освещения и пост-обработки: URP предоставляет более интуитивные инструменты для настройки освещения и пост-обработки, что ускоряет процесс разработки визуальных эффектов.
- Кроссплатформенность: URP оптимизирован для работы на различных платформах, включая мобильные устройства, консоли и ПК, обеспечивая единообразие визуального качества.

Недостатки URP:

- Отсутствие полной обратной совместимости с BiRP: миграция проектов с BiRP на URP может потребовать значительных изменений в коде и настройках материалов.
- Ограниченнная поддержка некоторых сложных эффектов: некоторые специфические или кастомные эффекты, реализованные в BiRP, могут требовать переработки для работы с URP.
- Необходимость обучения: разработчикам, привыкшим к BiRP, может потребоваться время для освоения новых инструментов и возможностей URP.
- Зависимость от обновлений Unity: некоторые функции URP могут быть доступны только в определенных версиях Unity, что требует поддержания актуальности используемой версии движка.

Характеристика	Built-in Render Pipeline (BiRP)	Universal Render Pipeline (URP)
Производительность	Низкая до средней	Высокая
Гибкость настройки	Ограниченнная	Высокая благодаря SRP
Поддержка современных технологий	Ограниченнная	Широкая
Совместимость с старыми проектами	Высокая	Низкая (требуется миграция)
Кастомизация шейдеров	Сложная и требовательная	Упрощенная с помощью Shader Graph
Инструменты оптимизации	Ограниченные	Расширенные и гибкие
Модульность	Низкая	Высокая

Таблица 3.1: Сравнение BiRP и URP

3.3.3 High Definition Render Pipeline (HDRP)

High Definition Render Pipeline (HDRP) предназначен для проектов с высоким уровнем графики, таких как AAA-игры и визуализации, требующие значительных вычислительных

ресурсов. HDRP фокусируется на обеспечении максимального качества изображения за счет использования продвинутых графических техник и эффектов. Однако никто пока не добавил поддержку foveated rendering в HDRP², даже сам Unity или Meta, так что мы не будем рассматривать данный Rendering Pipeline.

4 План реализации Gaze tracking

Предисловие

4.1 Актуальность технологии и сценарии применения

В последние годы наблюдается бурный рост интереса к технологиям отслеживания взгляда. *Gaze tracking* стал востребованным инструментом для понимания внимания пользователя. Он находит применение в самых разных областях – от исследований пользовательского опыта (UX) и анализа удобства интерфейсов до видеоигр и систем виртуальной реальности. Например, по движениям глаз можно оценивать, какие элементы интерфейса привлекают внимание, адаптировать контент в реальном времени или управлять игровым процессом. Кроме того, отслеживание взгляда активно используют для оптимизации графики с помощью *foveated rendering* – динамического изменения детализации изображения в зависимости от точки фиксации взгляда. Все это делает данную технологию чрезвычайно актуальной.

4.2 Доступность технологии массовому пользователю

Однако, массовое распространение технологии *eye tracking* маловероятно в том числе из-за стоимости оборудования. Традиционные системы отслеживания взгляда часто требуют специализированного аппаратного обеспечения: высокоскоростных камер, инфракрасных подсветок, датчиков положения головы. Такие решения, как правило, дороги и малодоступны для широкого круга пользователей. Например, коммерческие трекеры взгляда вроде *Tobii* [8] считаются эталоном точности, но стоят дорого и требуют подключения отдельного устройства. По этой причине в последние годы все больше внимания уделяется программным *webcam-based* решениям – то есть отслеживанию взгляда с помощью обычной веб-камеры без дополнительных сенсоров.

²Обсуждение Foveated Rendering для HDRP: <https://discussions.unity.com/t/hdrp-foveated-rendering/1541424/6>, дата обр. 18.01.2025

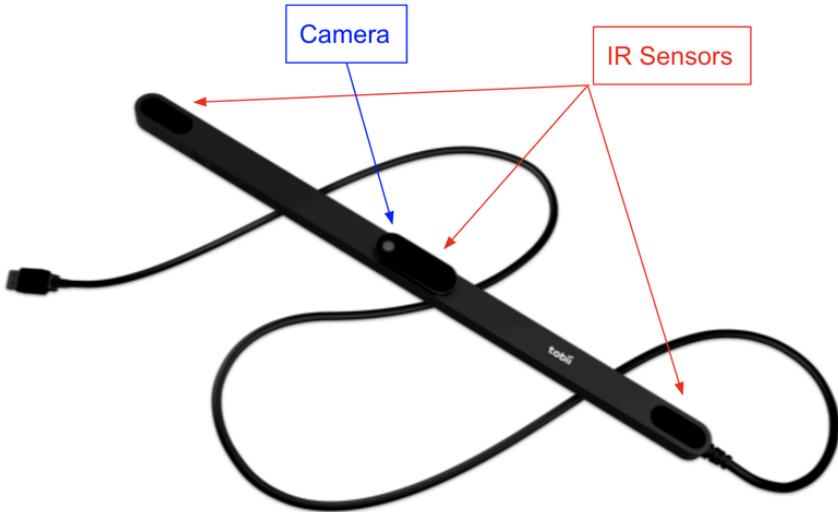


Рис. 4.1: Tobii eye tracker 5 – устройство отслеживания взгляда

4.3 Обзор подходов к реализации

Методы отслеживания взгляда исторически можно разделить на два основных подхода: (*model-based*) и по изображению (*appearance-based*). Модельно-ориентированные методы опираются на геометрическую модель глаза и оптические особенности зрачка. Как правило, они используют специальные метки или подсветку: например, инфракрасные LED-огни создают блики на роговице, а камера фиксирует положение зрачка и этих бликов. По взаимному расположению зрачка и бликов вычисляется направление взгляда относительно головы, а зная геометрию глаза – можно определить точку пересечения взгляда с экраном. Подобные алгоритмы были подробно проработаны в ранних исследованиях и способны давать высокую точность при идеальных условиях. Однако им присущи существенные недостатки: во-первых, требуется специальное оборудование (например, камеры с ИК-подсветкой) и длительная калибровка для каждого пользователя; во-вторых, сложность таких моделей делает их крайне не универсальными.

4.4 Современный подход

Appearance-based подходы предлагают иной путь. Они напрямую используют изображения глаз и лица, пытаясь извлечь направление взгляда из самих визуальных данных, без явного физического моделирования глаза. С развитием методов машинного обучения и компьютерного зрения стали популярны именно такие методы: большие выборки данных о глазах и взгляде используются для обучения моделей, которые сами находят сложные зависимости между изображением глаз и направлением взгляда. По сути, задача определения

взгляда сводится к задаче регрессии по изображению – модель получает сырье снимки глаз и лица и выдает оценку направления взгляда или точки фиксации. Современные подходы этого типа почти всегда основаны на сверточных нейросетях (CNN). Appearance-based методы значительно проще в использовании: они почти не требуют ручной настройки под каждого пользователя (в нашем проекте всего 25 точек для калибровки) или установки специальных маячков, достаточно лишь камеры и предварительно обученной модели. Кроме того, они лучше справляются со сложными случаями – различными формами глаз, отражениями, помехами, изменениями освещения – за счет того, что эти вариации могут быть охвачены обучающим датасетом.

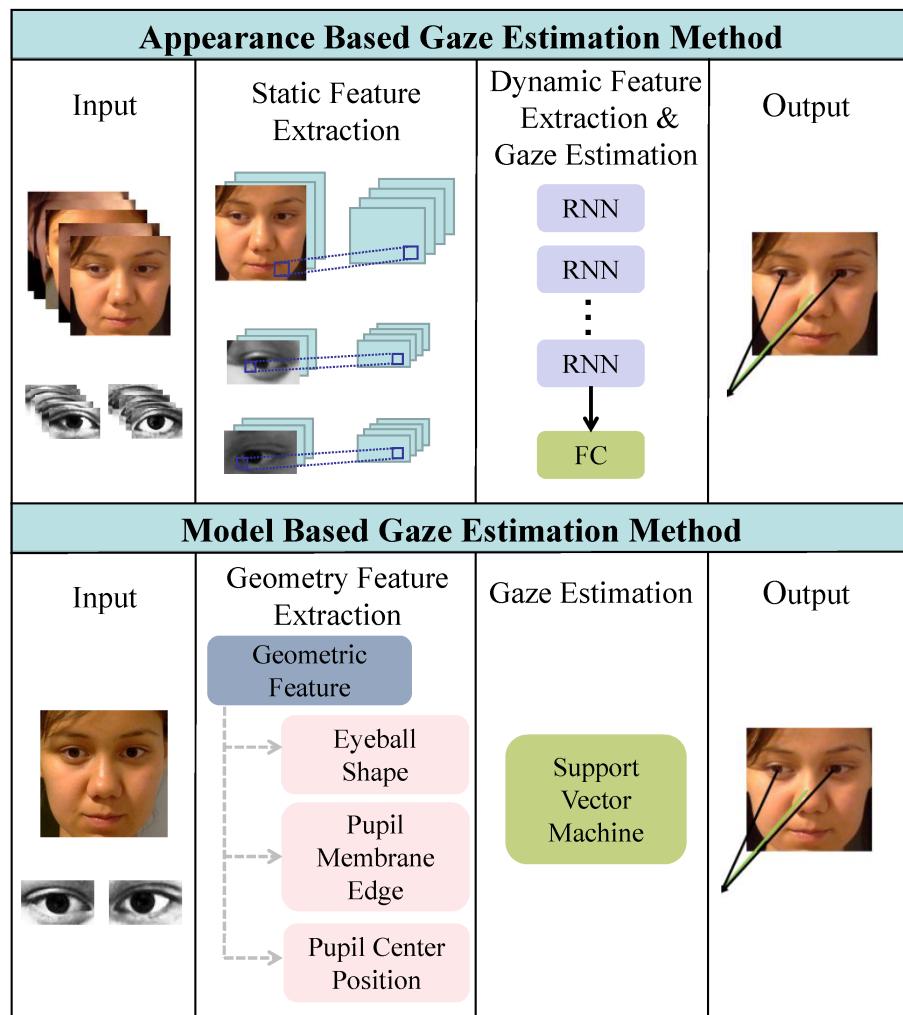


Рис. 4.2: model-based и appearance-based подходы

4.5 Сложности appearance-based подхода

Нейросети способны интерпретировать информацию о положении головы, зрачков, форме век – и вычислять направление взгляда как совокупность этих факторов. Поэтому

именно appearance-based подходы с глубоким обучением сегодня считаются наиболее перспективными. Тем не менее, даже на фоне этих успехов точное отслеживание взгляда по обычной веб-камере остается сложной задачей. Проблема в том, что система должна быть чрезвычайно чувствительной к едва уловимым изменениям, но при этом устойчивой к шумам и вариациям окружения. Бытовые веб-камеры обычно не обладают высоким разрешением или высокой частотой кадров по сравнению со специализированными трекерами. Существующие программные решения на базе веб-камер пока уступают по точности и скорости даже относительно простым коммерческим трекерам. Именно поэтому задача, рассматриваемая в нашей работе, остается актуальной.

4.6 Гибридные решения и перспективы развития

Появляются и гибридные варианты, нацеленные на удешевление технологии. Например, Eyeware Beam (один из вариантов отслеживания взгляда в нашем проекте) предлагает использовать обычную камеру смартфона и его LiDAR сенсор вместо специализированного устройства. Подобные решения выглядят многообещающе, они избавляют от необходимости покупать отдельное устройство, используя уже имеющиеся камеры и мощность потребительского ПК. Тем не менее, даже им пока не удалось полностью заменить возможности специализированных трекеров. Кроме того, программы вроде Beam пока распространены мало, требуют платной подписки и работают только с новыми моделями iPhone. Таким образом, аппаратные и полуаппаратные альтернативы хотя и существуют, но либо малодоступны, либо не решают всех проблем подхода с использованием веб-камеры.

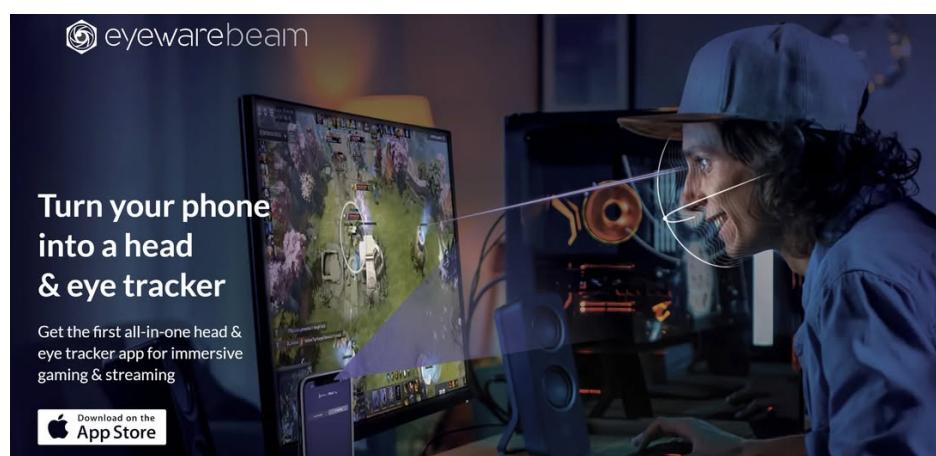


Рис. 4.3: Eyeware Beam

4.7 Современное состояние исследований

Текущее состояние исследований в области *gaze tracking* неоднозначно [4]. С одной стороны, сообществом собраны обширные датасеты изображений глаз и лиц с метками направления взгляда, что сильно продвинуло вперед обучение моделей. Начиная примерно с 2015 года были созданы общедоступные базы данных, такие как **MPIIGaze** (фотографии глаз людей в реальных условиях с камер ноутбуков) [5]. С другой стороны, открытые проблемы все еще активно обсуждаются. Одна из главных — это вариативность позы головы и положения пользователя относительно экрана. В идеальных условиях испытуемый сидит неподвижно прямо перед камерой, но при реальном использовании компьютера или телефона человек может наклоняться, поворачивать голову, менять расстояние до экрана. Это сильно осложняет задачу. Для того чтобы *webcam-based* отслеживание взгляда стало действительно практическим, важно компенсировать движения головой и оценивать положения пользователя относительно экрана. Система должна уметь определять, где находятся глаза в пространстве: на каком расстоянии и под каким углом относительно монитора. Без этой информации невозможно надежно перевести направление взгляда в координаты на дисплее. Если пользователь подался вперед или отклонился назад, изменится масштаб сцены и точка пересечения взгляда с плоскостью экрана. Поэтому современные исследования стремятся интегрировать в *gaze tracking* дополнительные модули, учитывающие позу головы. Один из подходов — применять методы компьютерного зрения, такие как **Structure from Motion** (SfM) или стереозрение, чтобы по изображению с одной веб-камеры восстановить 3D-положение головы и глаз. В целом, задача отслеживания взгляда в неидеальных условиях постепенно смещается от поиска только направления глаз к отслеживанию глаз и головы по-отдельности.

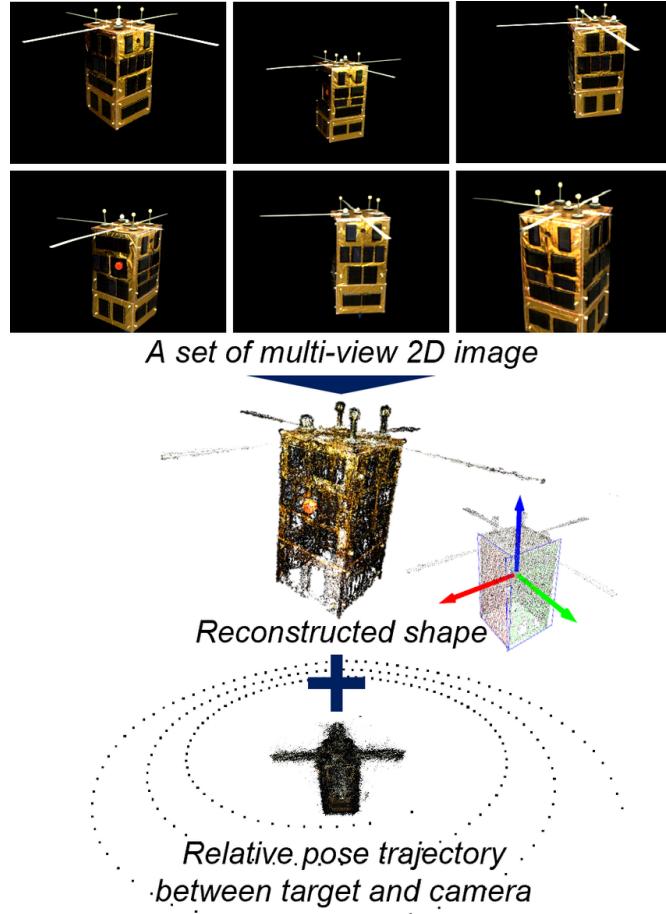


Рис. 4.4: Иллюстрация Structure from Motion

5 Реализация Gaze Tracking

5.1 EyeGestures v3

EyeGestures v3 – это Open Source проект для отслеживания взгляда с использованием обычной камеры. Есть несколько версий: ранняя V1 опиралась на классические модельные алгоритмы, V2 добавила машинное обучение поверх V1 для повышения точности, а V3 представляет собой самую новую, более быструю и точную реализацию. Библиотека использует такие инструменты как MediaPipe (для детекции лица и глаз) и scikit-learn (для ML-калибровки). Библиотека способна определять координаты взгляда на экране, а также сопутствующие характеристики: например, состояние фиксации взгляда (`stable fixation`) и саккад (резких движений глаза). Предусмотрено и распознавание моргания (в версии V2 возвращался флаг `blink`, в V3 он тоже доступен). Библиотека содержит встроенный режим калибровки: пользователю последовательно показываются специальные точки на экране (по умолчанию около 25 точек) в разных местах, на которых нужно сфокусироваться. Технически калибровка реализована как сбор пар «наблюдаемые характеристики глаз – реаль-

ные координаты целевой точки». Эти данные затем используются для обучения модели, которая и прогнозирует направление взгляда. В EyeGestures V2, например, метод `step()` в режиме калибровки возвращал координаты текущей целевой точки калибровки, а также радиусы кругов калибровки – `acceptance_radius` (величины, показывающие насколько точно нужно попасть взглядом в цель) и `calibration_radius` (когда взгляд считается достаточно устойчивым на цели, чтобы зафиксировать точку). В V3 интерфейс упростили. Метод `step()` и выходные данные: В основной рабочий цикл EyeGestures интегрируется вызов `event, cevent = gestures.step(frame, calibrate, screen_width, screen_height, context="...")`. Каждый кадр камеры передается в `step()`, который возвращает два объекта-события: `event` – рассчитанные параметры взгляда, и `cevent` – данные калибровочного процесса. В случае обычного отслеживания (калибровка уже пройдена) основной объект `event` содержит:

- `event.point` – координаты точки фиксации взгляда на экране,
- `event.fixation` – индикатор фиксации (насколько взгляд стабилен),
- `event.saccadess` – детектор резких движений глаз.

Если же идет калибровка (`calibrate=True`), дополнительно возвращается `cevent` – в нем библиотека предоставляет информацию о текущей калибровочной точке. Например, в предыдущей версии V2 аналогично выдавались `calibration_point` (координаты той точки, на которую сейчас должен смотреть пользователь) и радиусы допусков. Таким образом, приложение, использующее EyeGestures, может отображать нужные визуальные подсказки для калибровки (например, рисовать мишень в позиции `calibration_point`) и знать, когда пользователь достаточно точно навёл взгляд (взгляд войдет в область `acceptance_radius`), чтобы зафиксировать эту точку.

5.2 Интеграция в Unity

Включение EyeGestures в игровой движок (Unity) – задача нетривиальная, учитывая что библиотека написана на Python. Прямо вызвать Python-код из C# нельзя, поэтому существует несколько подходов:

- **Отдельный процесс:** Запустить EyeGestures как внешний Python-процесс, который будет получать картинку с веб-камеры и вычислять координаты взгляда, а затем передавать их в Unity. Передача может осуществляться через сокеты (UDP/TCP), shared memory, Named Pipes итд. Unity скрипт, в свою очередь, будет получать эти данные. Этот способ обеспечивает разделение, но существует задержка пару миллисекунд.

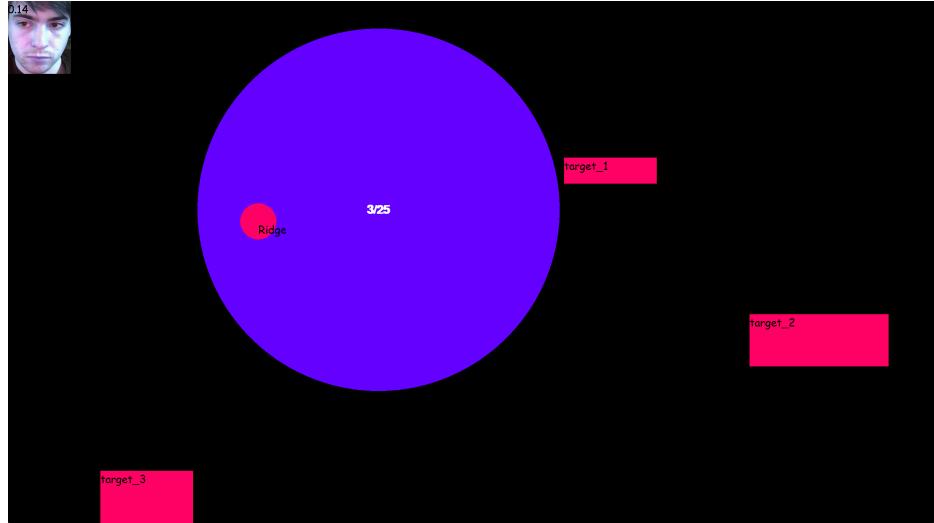


Рис. 5.1: Процесс калибровки в EyeGestures

- Использование Unity + Python: Существуют решения, позволяющие вызывать Python внутри Unity (например, Unity ML-Agents в обратную сторону или IronPython), но для реального времени с камерой это не самый выгодный путь, также из-за большого количества нестандартных библиотек этот вариант невозможен в нашем проекте.

5.3 Eyeware Beam Eye Tracker

Eyeware Beam – это отдельное приложение, которое выполняет все вычисления, и набор SDK-библиотек, через которые сторонние программы (например, игра на Unity) могут получать готовые данные о взгляде.

Как работает Beam: Пользователь устанавливает и запускает приложение Eyeware Beam на своем ПК. Это приложение захватывает изображение с камеры и с помощью своих моделей отслеживает глаза и голову. В нём же обычно проводится калибровка – Beam предоставляет пользователю свой интерфейс калибровки. После успешной калибровки Beam начинает выдавать скорректированные координаты взгляда. Стороннее приложение подключается к Beam через SDK: для этого Beam поднимает локальный сервер-трекер, к которому можно подключиться. В C++ это делается созданием объекта `eyeware::TrackerClient` и проверкой

```
if (tracker.connected())
```

Далее мы запрашиваем нужные данные – `ScreenGazeInfo` через метод `get_screen_gaze_info()`. Аналогично можно получить `HeadPoseInfo` для данных о положении головы. Данные, которые предоставляет SDK:

- 2D-координаты взгляда на экране. Beam прямо выдает положение точки взгляда в пикселях экрана.
- **Confidence**. В каждый момент Beam оценивает надёжность определения взгляда – SDK возвращает enum качества трекинга (низкий, средний, высокий).
- **Статус отслеживания**. Флаг `is_lost` указывает, потерян ли в текущем кадре взгляд/ положение головы. Если глаза временно не обнаружены, Beam также сообщит об этом.

Минусы Beam:

- **Закрытость Beam** – проприетарное решение. Исходный код недоступен, поэтому невозможно изменить или кастомизировать алгоритмы трекинга. Мы полагаемся на настройки, предоставленные SDK.
- **Лицензирование и стоимость**. Eyeware – коммерческая компания, и их SDK невозмож но использовать бесплатно.

В целом, Beam SDK – отличный выбор, потому что нам не нужно думать о реализации Gaze Tracing, но мы также теряем гибкость и контроль. Для задачи `foveation rendering` в играх Beam дает качественные данные о положении взгляда, позволяющие сразу приступить к реализации рендеринга, не погружаясь в компьютерное зрение.

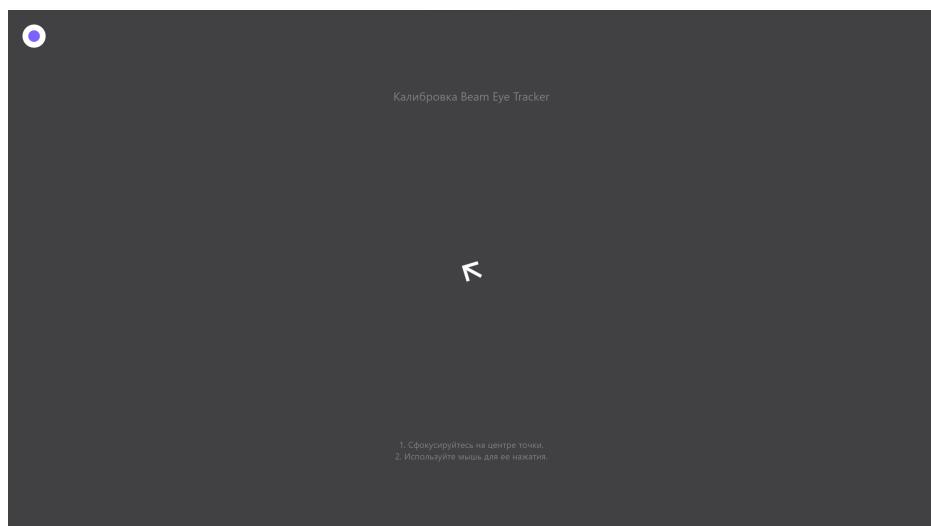


Рис. 5.2: Процесс калибровки в Beam

5.4 Базовый пример без использования ML

Это Python скрипт, который захватывает изображение с веб-камеры, находит на лице глаза и оценивает направление взгляда без сложных моделей. Сначала на видеопотоке нахо-

дится лицо, затем области глаз. Это происходит с помощью MediaPipe Face Mesh, который сразу дает координаты основных черт лица.

Минусы:

- **Низкая точность без ML.** Точный расчет точки на экране, куда смотрит пользователь, требует учета многих факторов: глаза, угол и расстояние до камеры, оптические искажения и т.д. Простой геометрический подход (насколько пикселей сместился зрачок) не может учесть всего этого. Без специальной калибровки или обученной модели ошибка будет очень большая.

Стоит упомянуть, что оба решения с открытым кодом (python-скрипт и EyeGestures) используют библиотеку MediaPipe, в частности Face Mesh. На этом решении построено давляющее большинство продуктов, решающих проблему Screen-based Gaze Tracking'а и это не совпадение. Мне показалось разумным посвятить несколько абзацев этому фреймворку, так что ниже я вкратце расскажу, что я о нем узнал.

5.5 Исторический контекст создания MediaPipe

MediaPipe — это кроссплатформенный фреймворк с открытым исходным кодом, разработанный компанией Google для обработки мультимедийных данных в реальном времени. Внутренняя версия этой системы начала создаваться в начале 2010-х годов: уже в 2012 году технологии MediaPipe применялись для анализа видео и аудио на YouTube в режиме реального времени. Со временем фреймворк стал использоваться во многих продуктах Google (например, в камерах Nest Cam, приложении Google Lens, для AR-фильтров в рекламе, в сервисе Google Photos и др.), демонстрируя свою эффективность.

5.5.1 MediaPipe Face Mesh и отслеживание взгляда

Одним из ключевых компонентов MediaPipe, применимых к задаче отслеживания взгляда, является Face Mesh — модель для детекции лица и определения его геометрии (точек на лице). Face Mesh выделяет на лице пользователя сотни опорных точек (468 точек в базовом варианте) в трёхмерном пространстве, используя в качестве входных данных изображение с камеры. MediaPipe в реальном времени, строит 3D-меш лица. Дополнительно MediaPipe предоставляет расширенный режим с применением модели Attention Mesh, которая повышает точность локализации отдельных зон лица — в частности, области глаз и радужки — за счёт механизма `attention`. Этот режим (активируемый параметром `refine_landmarks`)

добавляет около 10 точек вокруг радужных оболочек глаз, доводя общее число ключевых точек до 478. На практике Face Mesh совместно с модулем Face Transform позволяет не только получать координаты черт лица, но и оценивать поворот головы — что особенно полезно для последующего вычисления направления взгляда.



Рис. 5.3: Media Pipe Face Mesh(это я)

5.6 Сравнение трех подходов

Точность отслеживания: Beam SDK обеспечивает наивысшую точность – после калибровки его ошибки минимальны, сопоставимы с профессиональными устройствами. EyeGestures v3 дает хорошую точность при правильной калибровке – на центральной части экрана ошибка может быть небольшой, но по краям или при изменении положения пользователя точность может снижаться. Кроме того, если пользователь отклонится от позы, в которой проходила калибровка, точность ухудшится, поэтому периодически может требоваться перекалибровка. Baseline – работает, это уже успех.

Стабильность: Beam крайне стабилен в работе: благодаря комбинированному трекингу головы и глаз, он редко «теряет» взгляд, плавно восстанавливается после морганий, работает при разнообразном освещении (насколько позволяет обычная камера). EyeGestures довольно стабилен при фронтальном положении и хорошем освещении – он опирается на MediaPipe, который хорошо детектирует лицо. Однако в сравнении с Beam, устойчивость ниже: например, резкий поворот головы вбок может вывести глаза из поля зрения камеры и

EyeGestures потеряет трекинг (Beam же, зная поворот головы, может приблизительно дать координаты или хотя бы сразу отметить потерю).

Доступность и простота использования: EyeGestures – проект энтузиастов, его качество хорошее, но может не быть гарантий, обновления зависят от сообщества. Beam в плане конечного пользователя достаточно удобен: обычно калибровку проходят один раз при настройке Beam-приложения, дальше оно работает в фоновом режиме для всех игр. Но Beam не является бесплатным, и также нужно совместимое устройство (например, iPhone для лучшего трекинга или качественную веб-камеру на ПК). Для разработчика интеграция Beam – самая простая: документация и SDK упрощают задачу, примеры кода показывают, как получить данные.

Таблица 5.1: Сравнение методов отслеживания взгляда

Параметр	Beam SDK	EyeGestures v3	Baseline (Базовый пример)
Точность отслеживания	Наивысшая точность после калибровки	Хорошая при правильной калибровке, снижается по краям	Работает, это успех
Стабильность	Крайне стабилен, плавное восстановление после морганий	Довольно стабилен при фронтальном положении, может терять трекинг при резких движениях	—
Доступность и простота использования	Простая интеграция через SDK, требует лицензирования	Проект энтузиастов, качество хорошее, но зависит от сообщества	—

6 Реализация VRS на стороне Untiy

6.1 Общий Workflow

- Инициализация нативных плагинов из Unity.
- В методе Update: Получение данных о направлении взгляда с помощью класса `GazePluginUpdater`.
- В методе Update: Отправка этих данных в нативный плагин VRS с помощью метода `RefreshGazeDirection`.
- Очистка ресурсов всех плагинов.

6.2 Получение данных о взгляде

В классе `GazePluginUpdater` используются следующие ключевые методы:

- `Initialize` — инициализирует плагин для отслеживания взгляда.
- `Cleanup` — освобождает ресурсы плагина.
- `GetGazeDirectionVector` — получает координаты взгляда, нормализует их и передает в нативный плагин VRS.

После того, как данные о взгляде получены, они преобразуются в координаты и передаются на GPU через вызов `VrsPluginApi.UpdateGazeDirection`.

6.3 Интеграция с BiRP

В BiRP Variable Rate Shading управляется через класс `VrsBirpController`, который использует `command buffers` для внедрения событий в нужные моменты рендеринга, а именно в моменты `CameraEvent.BeforeForwardOpaque` и `CameraEvent.AfterForwardAlpha` добавляются команды, которые соответственно включают и отключают плагин.

При работе с `command buffers` используются методы:

- `bufferManager.AddCommandBuffer` — добавляет `command buffers` с событиями для инициализации и очистки плагина VRS.
- `bufferManager.EnableBuffers` — активирует `command buffers`.
- `bufferManager.DisableBuffers` — деактивирует `command buffers`.

6.4 Интеграция с URP

В URP используется `ScriptableRendererFeature`, который позволяет добавлять низкоуровневые команды рендеринга в граф рендеринга. Как раз в классе `VrsUrpController` добавляются шаги, которые отвечают за управление VRS.

При работе с `ScriptableRendererFeature` используются методы:

- `EnableFoveatedRenderingPass` и `DisableFoveatedRenderingPass` — методы для включения и отключения `RenderingPass`.
- `cmd.IssuePluginEvent` — метод, который передает команду плагину для включения или отключения VRS.

URP позволяет более гибко интегрировать плагин, благодаря использованию rendering graph, который контролирует порядок выполнения команд.

6.5 Настройка foveal regions и разрешения растеризации

В обоих подходах (BiRP и URP) реализована возможность настройки foveal region и качества рендеринга для каждой из областей. Для этого используются методы:

- `ConfigureShadingRatePreset` — настройка предустановленных значений для частоты рендеринга.
- `ConfigureRegionRadii` — настройка радиусов для VRS.
- `AssignShadingRate` — настройка качества рендеринга для каждой из областей.

7 Реализация VRS на стороне нативного плагина

7.1 Основные компоненты плагина

Плагин состоит из нескольких ключевых частей:

- `GazeManager` — отвечает за обработку данных отслеживания взгляда пользователя.
- `VrsManager` — управляет настройками VRS, взаимодействует с API NVidia для применения паттернов VRS.
- `RenderEventHandler` — обрабатывает события рендеринга и отправляет соответствующие команды в `VrsManager` и `GazeManager`.
- `PluginInterface` — интерфейс плагина, который взаимодействует с Unity, инициализирует все компоненты и обрабатывает вызовы рендеринга.
- `NvApiWrapper` — оборачивает вызовы NVidia API для удобства и облегченного управления настройками VRS.

7.2 Инициализация плагина и подключение к Unity

Когда плагин загружается в Unity, вызывается функция `UnityPluginLoad`, которая выполняет следующие действия:

- Создается экземпляр класса `PluginInterface`, который будет управлять всеми процессами плагина.
- Плагин регистрирует графическое устройство с помощью NVidia API через `NvApiWrapper`.
- Инициализируется `GazeManager` - обработчик данных взгляда.
- Создается объект `RenderEventHandler`, который будет обрабатывать события от Unity, а именно команды на включение и выключение.

После такой инициализации плагина, он готов к обработке данных.

7.3 Обработка данных взгляда

Каждый раз, когда Unity передает информацию о направлении взгляда пользователя, `GazeManager` выполняет следующие действия:

- Метод `UpdateGazeDirection` принимает вектор направления взгляда (`Vector3`) и нормализует его.
- После нормализации, метод `HasGazeChanged` проверяет, изменилось ли положение взгляда относительно предыдущего. Если изменение превышает заданный порог, данные обновляются, чтобы не пересчитывать карту тайлов.
- В методе `RefreshGazeData` обновляются данные взгляда, которые затем передаются в NVidia API.
- Обновленные данные взгляда передаются в `VrsManager` через метод `CommitGazeData`, чтобы GPU мог применить карту тайлов, сформированную на основе текущего положения взгляда.

7.4 Применение VRS

Когда `Foveated rendering` включен, `RenderEventHandler` получает соответствующее событие и передает его в `VrsManager`. Процесс применения фоеативного рендеринга включает следующие шаги:

- Метод `ApplyShadingRatePattern` в `VrsManager` вычисляет, какие области экрана должны быть отрендерены с более высоким разрешением, а какие - с меньшим.

- `VrsManager` заполняет параметры рендеринга и передает их в NVidia VRS API с помощью метода `vrsHelper->Enable`.
- Параметры могут быть настроены вручную или выбраны из заранее определенных настроек.

7.5 Обработка rendering events и очистка

Плагин отслеживает и обрабатывает `rendering events`, связанные с рендерингом, такие как включение/выключение фовеативного рендеринга или обновление данных взгляда. Для этого используется класс `RenderEventHandler`, который реагирует на различные типы событий:

- Когда часть из Unity сообщает о необходимости включить фовеативный рендеринг, `RenderEventHandler` вызывает метод `ApplyShadingRatePattern` для применения фовеативного паттерна.
- При отключении фовеативного рендеринга вызывается метод `RemoveShadingRatePattern`, который сбрасывает настройки.
- При получении обновленных данных взгляда, `RenderEventHandler` обновляет информацию о положении взгляда и передает её в `GazeManager` для последующей обработки.

Когда плагин выгружается, вызывается метод `Unload`, который выполняет очистку всех ресурсов, включая:

- Освобождение памяти, занятой обработчиками данных взгляда.
- Удаление объекта `RenderEventHandler`.
- Деинициализация NVidia API с помощью `nvApiWrapper.Unload()`.

Эти шаги гарантируют, что плагин не оставляет следов в системе после завершения работы, что важно для правильной выгрузки плагинов в Unity.

8 Реализация LOD Foveated rendering

`FoveatedLODController.cs` представляет собой скрипт для Unity, который управляет уровнем детализации (LOD) объектов сцены в зависимости от того, куда смотрит пользователь.

8.1 Общий workflow

1. Инициализация компонентов: При старте сцены скрипт сначала находит все объекты с компонентом `LODGroup`. Эти группы содержат модели различных уровней детализации для каждого объекта.

2. Обновление на каждом кадре: В методе `Update` происходят следующие действия:

- Определяется нормализованное положение взгляда пользователя (в диапазоне от -1 до 1), используя либо данные от плагина VRS, либо положение мыши.
- В объекте `ZoneVisualizer` обновляется положение центра эллипса в зависимости от текущего положения взгляда.
- Далее вызывается метод `UpdateLODGroups`, который обновляет уровень детализации объектов в зависимости от того, где находится взгляд.

3. Вычисление нормализованных координат мыши: Для работы с мышью используется метод `GetNormalizedMousePosition`, который переводит экранные координаты мыши в нормализованные координаты в диапазоне [-1, 1].

4. Определение нужного уровня детализации для объектов: В методе `UpdateLODGroups` происходит вычисление того, в какой области взгляда находится каждый объект. Для этого сначала рассчитываются нормализованные координаты экрана для позиции объекта. Затем определяется расстояние между текущей позицией взгляда и позицией объекта. Если объект находится в *foveal region*, ему присваивается самый высокий уровень детализации. Все объекты, находящиеся в *peripheral region*, получают самый низкий уровень детализации.

5. Метод `IsWithinEllipse`: Это вспомогательный метод, который проверяет, находится ли точка в пределах эллипса. Для этого используется стандартное уравнение эллипса:

$$\frac{dx^2}{rx^2} + \frac{dy^2}{ry^2} \leq 1$$

где dx и dy — это отклонения по осям x и y от центра, а rx и ry — радиусы эллипса для *foveal region* или *peripheral region*.

6. Завершение работы: Очищается память, выделенная для нативных плагинов.

8.2 Заключение и важность всех методов

9 Настройка демо сцен

Мы использовали две сцены для демонстрации работы оптимизации.

9.1 Сцена Night City [7]

Этот пакет содержит готовую к использованию сцену с небольшой частью ночного города и длинным шоссе:

- Есть поддержка BiRP, URP
- Нет поддержки LOD
- Есть хорошо проработанное и настроенное освещение, а также его отражения на дороге
- Количество треугольников для всех объектов варьируется от 26 до 10000
- Разрешение текстур от 256 до 2048



Рис. 9.1: Сцена "Night city"

9.2 Сцена Mountains [1]

- Есть поддержка BiRP, URP
- Для большинства объектов есть поддержка LOD
- Сцена хорошо проработана и есть возможность регулировать скорость ветра и освещения
- Количество треугольников для всех объектов варьируется от 26 до 100000



Рис. 9.2: Сцена "Mountains"

- Разрешение текстур от 256 до 4096

Для того, чтобы сравнить эффективность методов и сравнение было репрезентативным, для каждой сцены был составлен маршрут движения камеры и маршрут "движения взгляда". Маршрут занимает около одной минуты для всех сцен.

Тестирование скомпилированной игры проводилось на компьютере с процессором Intel Core i5-13400 и видеокартой GeForce RTX 3070.

10 Результаты

Исходный код проекта и инструкция по настройке плагина доступны по следующей [ссылке](#).

Скомпилированное демо проекта (папка `build`) доступно по этой [ссылке](#).

Видео демонстрации работы также доступны по этой [ссылке](#).

Первое, что бросается в глаза - наша оптимизация LOD ухудшила FPS вместо того, чтобы улучшить. Есть два объяснения, что могло пойти не так:

- Перед рендерингом каждого кадра приходится пересчитывать расстояние до каждого объекта. Но я это делаю на высоком уровне и, вероятно, не так эффективно, как это происходит внутри Unity.
- Обычно оптимизация LOD эффективна именно потому, что расстояние от объекта до камеры не может измениться быстро. Однако в данном случае оно может меняться очень быстро, из-за чего происходит большой расход видеопамяти. Из-за этого могут происходить задержки в случаях, когда модели нет в видеопамяти и ее нужно быстро загрузить туда.

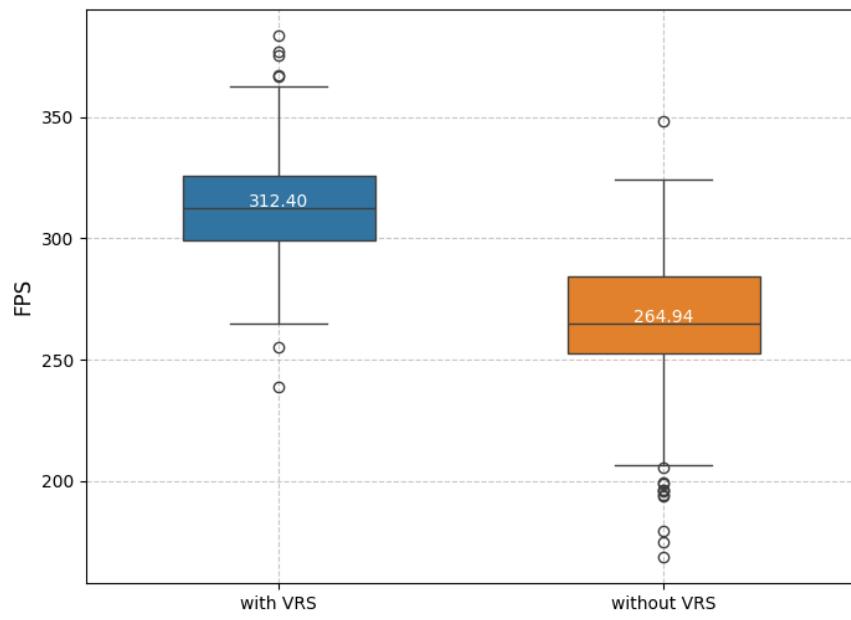


Рис. 10.1: Сравнение FPS на сцене Night City

Прирост производительности от VRS существенный и сопоставим с приростом производительности в VR шлемах, особенно учитывая, что теперь foveal region занимает большую площадь дисплея. Для Night City прирост FPS составляет 18%, для Mountains - 32.5 %.

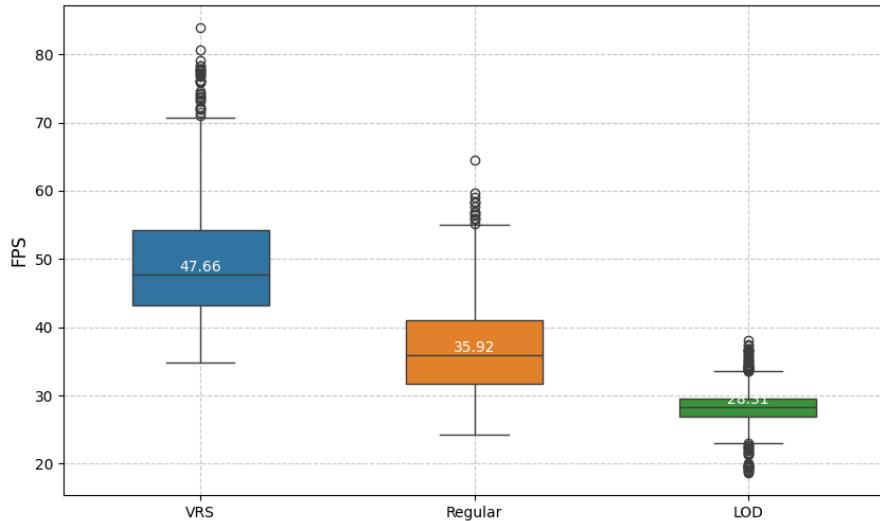


Рис. 10.2: Сравнение FPS на сцене Mountains

11 Перспективы развития для метода VRS

11.1 Luminance-Contrast-Aware Foveated Rendering

Одним из перспективных направлений развития foveated rendering является учет локального контраста изображения для более плавных переходов между зонами разного разрешения. Метод Luminance-Contrast-Aware Foveated Rendering [6] анализирует локальные контрасты и уровни освещенности, чтобы адаптировать детализацию не только на основе направления взгляда, но и на основе визуальных характеристик сцены. Это позволяет минимизировать переходы между уровнями детализации и сильно улучшает общее восприятие изображения

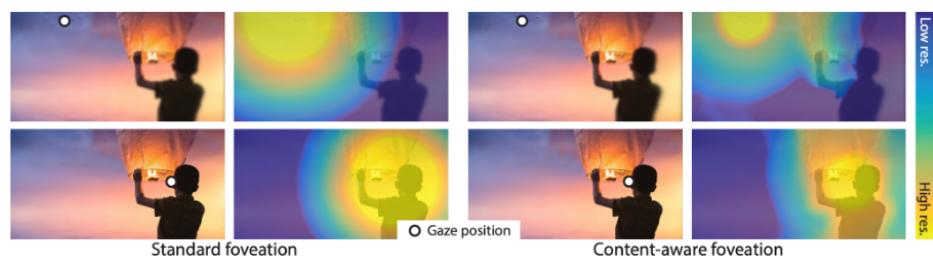


Рис. 11.1: Пример работы Content-aware foveation

11.2 Сглаживающие фильтры

Учитывая, что периферальные области рендерятся с пониженным разрешением, может появиться заметная рябь. Чтобы ее избежать, можно накладывать быстрый сглажива-

ющий фильтр, ведь разрешение в тех областях и так пониженное.

11.3 Upscaling

Upscaling, как оптимизация работает достаточно хорошо сам по себе, и может применяться независимо от других оптимизаций. Однако с Foveated Rendering он может работать еще лучше³.



Рис. 11.2: Пример работы технологии DLSS

³Туториал, в котором упоминается что upscaling очень хорошо сочетается с FR <https://mbucchia.github.io/OpenXR-Toolkit/fr.html#combining-with-the-upscaling-feature>, дата обр. 29.01.2025

Список литературы

- [1] BK. *Сцена Pure Nature 2: Mountains*. URL: <https://assetstore.unity.com/packages/3d/environments/pure-nature-2-mountains-269088> (дата обр. 24.01.2025).
- [2] Olivier Auguste Coubard, Marika Urbanski, Clémence Bourlon и Marie Gaumet. “Educating the blind brain: A panorama of neural bases of vision and of training programs in organic neurovisual deficits”. B: *Frontiers in Integrative Neuroscience* 8.89 (2014). DOI: [10.3389/fnint.2014.00089](https://doi.org/10.3389/fnint.2014.00089).
- [3] David Heaney. *Here's The Exact Performance Benefit Of Foveated Rendering On Quest Pro*. URL: <https://www.uploadvr.com/quest-pro-foveated-rendering-performance/> (дата обр. 29.01.2025).
- [4] Katrin Solveig Lohan Lucas Falch. *Webcam-based gaze estimation for computer screen interaction*. URL: <https://www.frontiersin.org/journals/robotics-and-ai/articles/10.3389/frobt.2024.1369566/full> (дата обр. 31.03.2025).
- [5] MPIIGaze. *dataset for appearance-based gaze estimation*. URL: <https://paperswithcode.com/dataset/mpigaze> (дата обр. 31.03.2025).
- [6] USI PDF Research Group. *Adaptive Foveation*. URL: <https://www.pdf.inf.usi.ch/projects/AdaptiveFoveation/> (дата обр. 24.01.2025).
- [7] Versatile Studio. *Сцена Demo City*. URL: <https://assetstore.unity.com/packages/3d/environments/urban/demo-city-by-versatile-studio-mobile-friendly-269772> (дата обр. 24.01.2025).
- [8] Tobii. *EyeTracking utility*. URL: <https://www.tobii.com/> (дата обр. 31.03.2025).