УДК 004

Марченко Владислав Вадимович,

студент, Новосибирский государственный технический университет, г. Новосибирск

ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ АЛГОРИТМОВ УПРАВЛЕНИЯ КАМЕРОЙ В 3D-ПЛАТФОРМЕРАХ С УЧЁТОМ КОЛЛИЗИЙ

Аннотация

В статье исследуются алгоритмы управления камерой в трёхмерных игровых средах с учётом коллизий с геометрией уровня. Разработана программа на C++/OpenGL, реализующая два режима работы камеры: «простой» (без учёта препятствий) и «умный» (с использованием raycast для обнаружения стен). Экспериментально показано, что простой режим приводит к проникновению камеры в геометрию при поворотах и движении в узких коридорах. Умный режим автоматически сокращает дистанцию до объекта при обнаружении препятствия, полностью исключая проход сквозь стены. Предложенный алгоритм может быть использован при разработке 3D-платформеров и приключенческих игр.

Annotation

The article investigates camera control algorithms in three-dimensional game environments considering collisions with level geometry. A C++/OpenGL program is developed implementing two camera operation modes: "simple" (without obstacle consideration) and "smart" (using raycast for wall detection). It is experimentally shown that the simple mode leads to camera penetration into geometry during rotations and movement in narrow corridors. The smart mode automatically reduces the distance to the object when an obstacle is detected, completely eliminating wall penetration. The proposed algorithm can be used in the development of 3D platformers and adventure games.

Ключевые слова: управление камерой, коллизии, raycast, 3D-платформер, компьютерная графика, OpenGL, игровой движок.

Keywords: camera control, collisions, raycast, 3D platformer, computer graphics, OpenGL, game engine.

Управление камерой является одной из ключевых проблем при разработке трёхмерных игр. Неправильно настроенная камера может существенно ухудшить игровой опыт, привести к дезориентации игрока или «вылету» камеры за пределы игрового мира. Как отмечает Gregory в фундаментальном труде по архитектуре игровых движков [1], в 3D-платформерах проблема усугубляется тем, что игрок постоянно перемещается по уровню, сталкивается с препятствиями, заходит в узкие коридоры и поворачивает за углы. В данной работе исследуются два подхода к управлению камерой: простой режим, при котором камера находится на фиксированном расстоянии от игрока, игнорируя наличие стен и препятствий, и умный режим, в котором с помощью raycast (пуска луча) определяется наличие препятствия между игроком и желаемой позицией камеры, после чего расстояние до игрока динамически корректируется. Цель работы — экспериментальное сравнение двух алгоритмов управления камерой и выявление преимуществ умного режима.

В простом режиме позиция камеры вычисляется по формуле сферических координат относительно целевого объекта (игрока) [2]:

$$\begin{cases} x = x_{target} + d \cdot \cos(\theta) \cdot \sin(\phi) \\ y = y_{target} + d \cdot \sin(\theta) \\ z = z_{target} + d \cdot \cos(\theta) \cdot \cos(\phi) \end{cases},$$

где d — расстояние от камеры до игрока, θ — угол возвышения (elevation), ϕ— угол азимута (azimuth). Параметры d, θ и ϕ могут изменяться пользователем (например, вращением мыши). Данный подход прост в реализации и даёт полный контроль над камерой, однако не учитывает наличие препятствий. Как следствие, при повороте камеры за угол стены она может оказаться внутри геометрии.

Умный режим дополняет простой алгоритм механизмом обнаружения препятствий. Как указано в классической работе Foley и др. по компьютерной

графике [3], raycast (бросание луча) является стандартным методом определения видимости и пересечений в трёхмерном пространстве. Алгоритм работает следующим образом: сначала вычисляется желаемая позиция камеры по формуле сферических координат, затем из позиции игрока в направлении желаемой позиции камеры пускается луч, который проверяется на пересечение со всеми стенами сцены. Если пересечение обнаружено на расстоянии tt от игрока, камера перемещается в точку пересечения минус небольшой отступ (чтобы не «упираться» в стену). Если пересечений нет — камера занимает желаемую позицию. Математически позиция камеры определяется как:

$$P_{camera} = P_{player} + d \cdot \min(d_{desired}, t_{hit} - \delta),$$

где d — единичный вектор направления от игрока к желаемой камере, $d_{desired}$ — желаемое расстояние, t_{hit} — расстояние до ближайшего препятствия, δ — величина защитного отступа (в эксперименте δ=0.3). Данный алгоритм гарантирует, что камера никогда не окажется внутри стены, сохраняя при этом максимально возможный обзор.

Разработана программа на языке C++ с использованием библиотеки OpenGL (glut). Программа включает следующие компоненты: игрок — куб зелёного цвета, управляемый клавишами W/A/S/D и пробелом (прыжок); стены — набор статических препятствий, образующих лабиринт; камера — реализованы два переключаемых режима (простой/умный); отображение информации — на экран выводится текущий режим, дистанция до игрока и позиция. Системные требования: процессор от 2 ГГц, видеокарта с поддержкой OpenGL 2.1+, 4 ГБ ОЗУ. Программа работает в реальном времени с частотой 60 кадров в секунду. Управление осуществляется с помощью клавиатуры и мыши. Предусмотрена возможность сброса сцены (клавиша R). Для удобства экспериментов на экран выводится информационная панель с текущим режимом камеры, дистанцией до игрока и координатами персонажа.

Тестовая сцена представляет собой замкнутый лабиринт размером 16×16 условных единиц с несколькими внутренними стенами и узкими проходами (рисунок 1). Игрок начинает движение от центра сцены и последовательно

подходит к прямой стене, поворачивает камеру, «заглядывая» за угол, и проходит через узкий коридор. Для каждого сценария сравнивалось поведение простой и умной камеры. Всего было проведено три серии экспериментов: приближение к прямой стене, поворот за угол при движении вдоль стены, прохождение узкого коридора с резким поворотом камеры. В каждой серии фиксировались минимальная дистанция камеры до игрока, наличие или отсутствие проникновения камеры в стену, а также время кадра (frame time) для оценки вычислительной нагрузки.

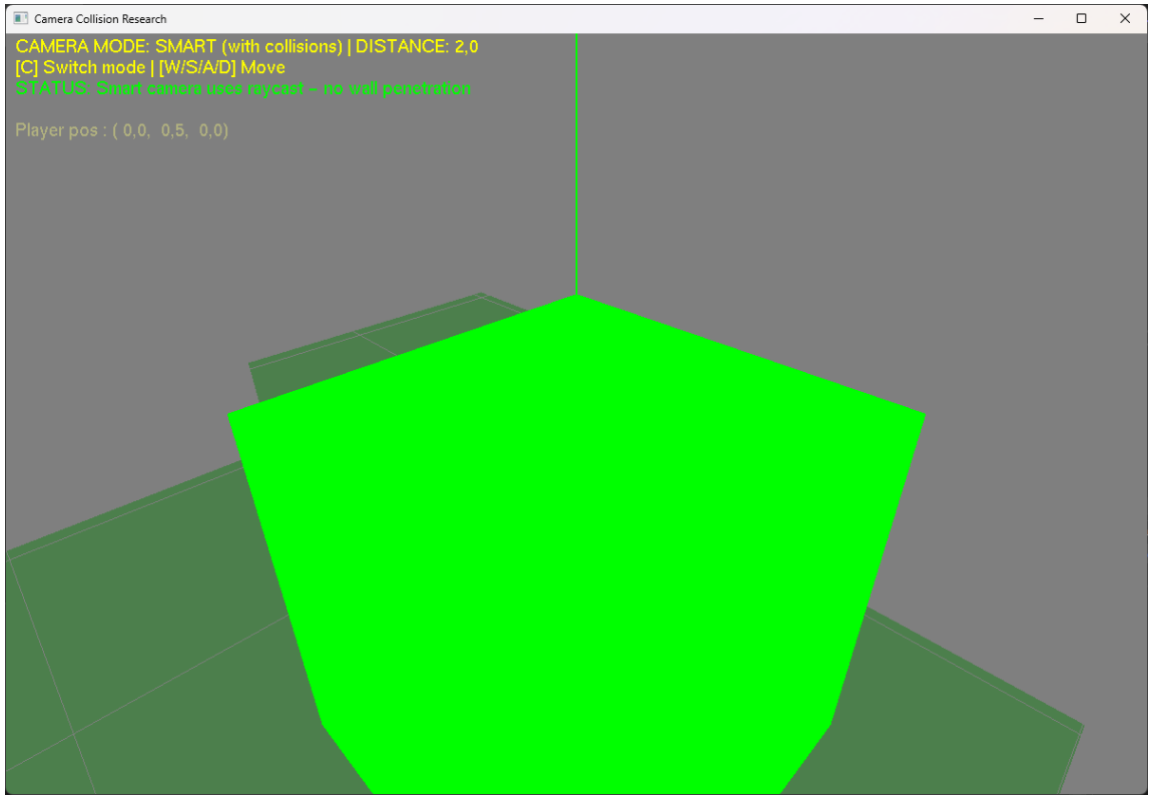

Рисунок 1 — Общий вид тестовой сцены (Smart камера, исходное положение).

На скриншоте показан лабиринт из серых стен, зелёный куб-игрок в центре. Камера находится на расстоянии примерно 7 единиц, текст в левом верхнем углу сообщает о режиме SMART. Сцена содержит 18 стен, образующих несколько замкнутых пространств и коридоров.

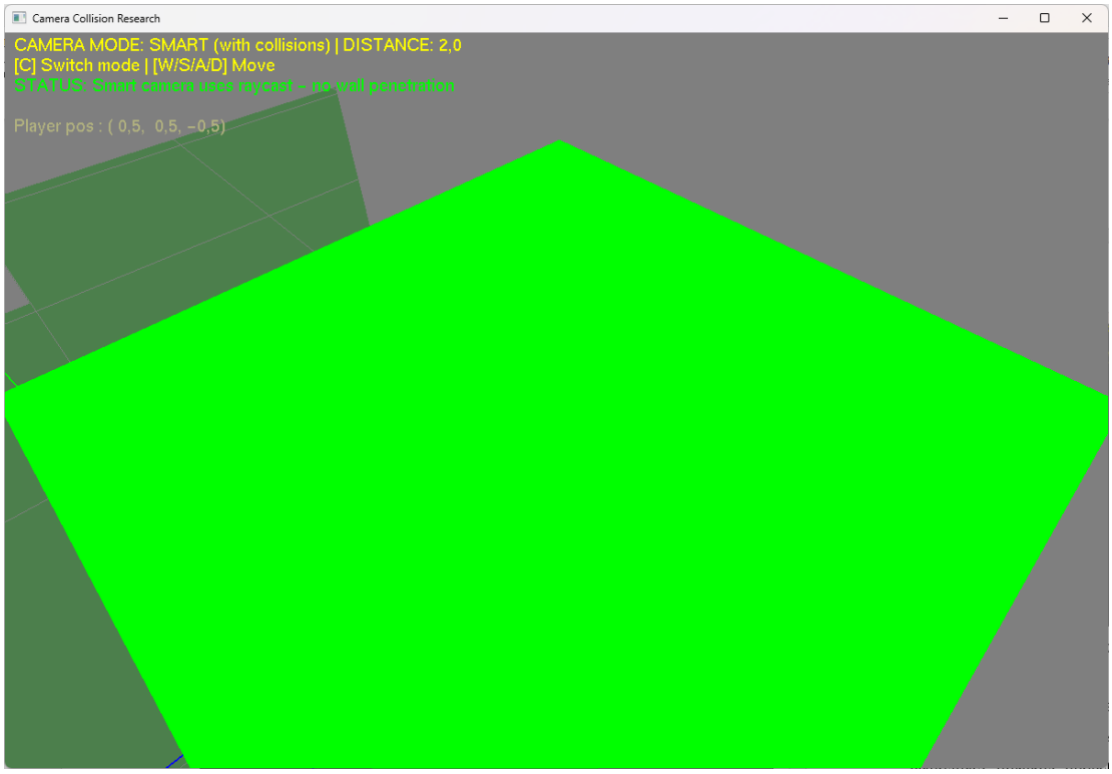

Рисунок 2 — Smart камера при приближении к стене.

Игрок подходит вплотную к стене. Камера автоматически сократила дистанцию с 7 до 3.5 единиц. Стена не перекрывает обзор, игрок хорошо виден. Сокращение дистанции происходит плавно, без рывков, за счёт линейной интерполяции позиции камеры между кадрами.

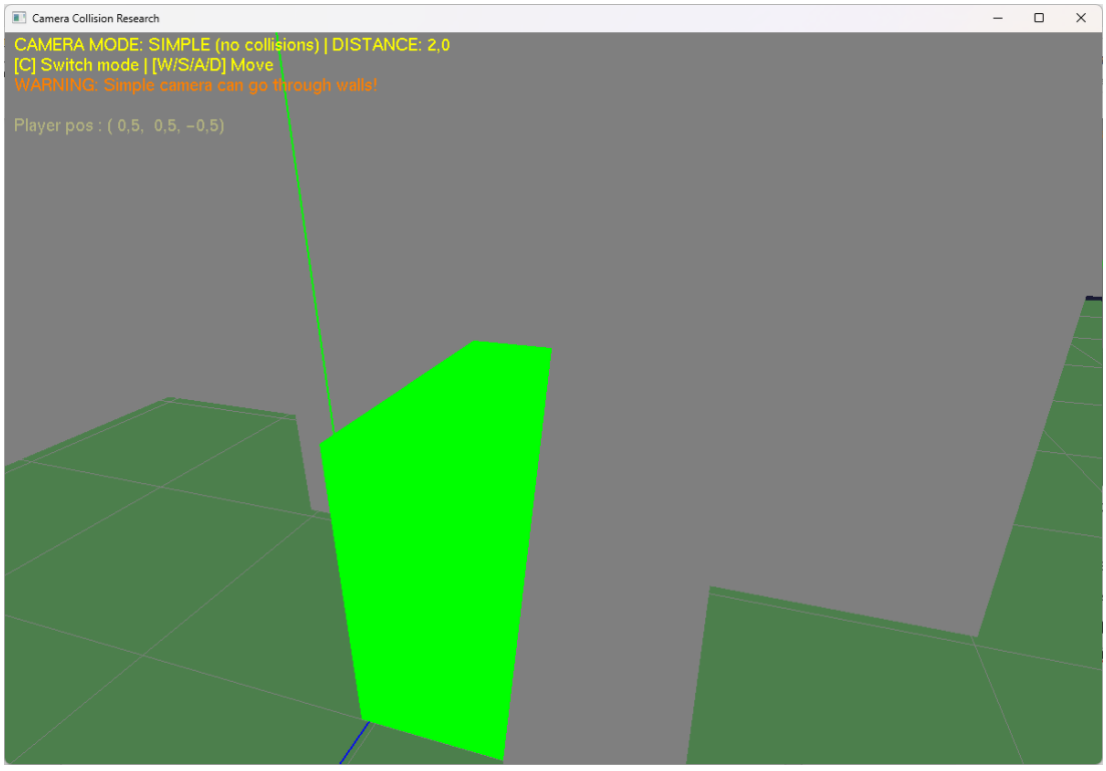

Рисунок 3 — Простая камера при повороте за угол (артефакт).

Игрок стоит у угла, камера переключена в режим SIMPLE. Камера «провалилась» в стену — виден фрагмент серой текстуры внутри геометрии, игрок частично перекрыт. Артефакт возникает потому, что простая камера не выполняет никаких проверок пересечения с геометрией. Данная проблема подробно описана McShaffry в книге по разработке игр [4].

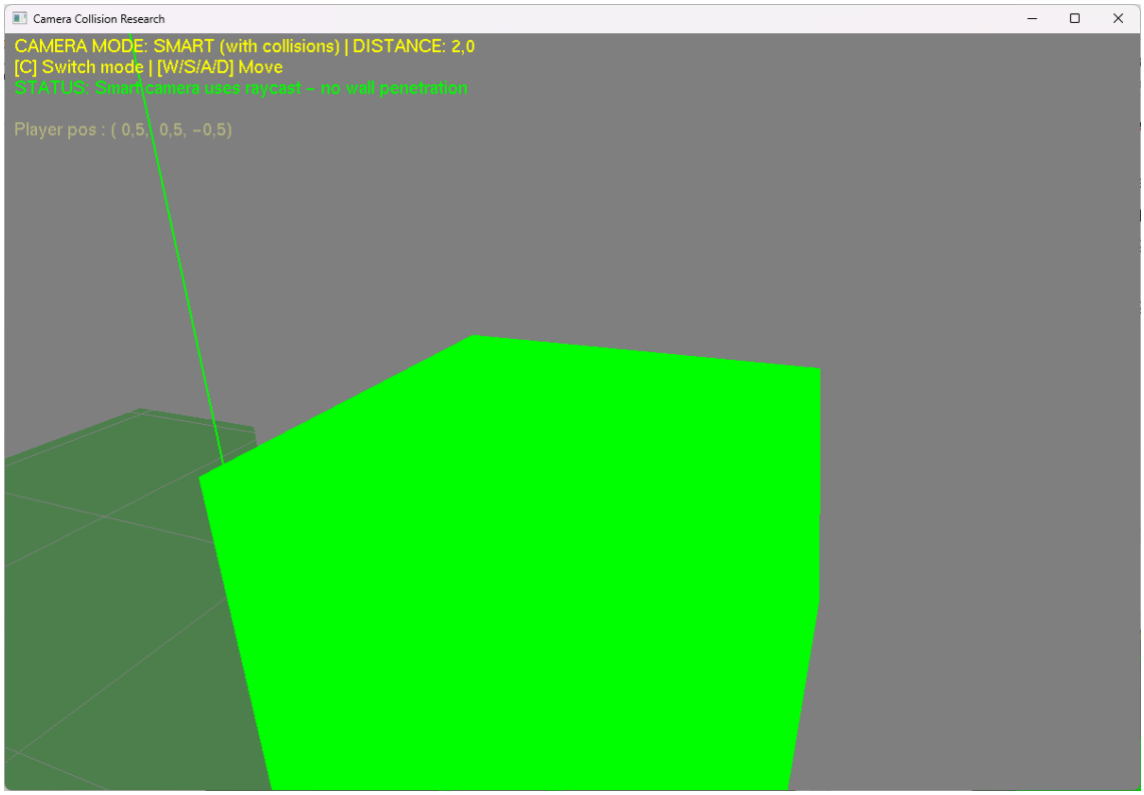

Рисунок 4 — Smart камера при повороте за угол (корректная работа).

Тот же угол, но камера в режиме SMART. Камера подъехала ближе и находится перед стеной, игрок полностью виден, артефактов нет. Луч от игрока к желаемой позиции камеры пересекся со стеной на расстоянии 2.8 единицы, поэтому камера была перемещена в точку пересечения (с отступом 0.3 единицы).

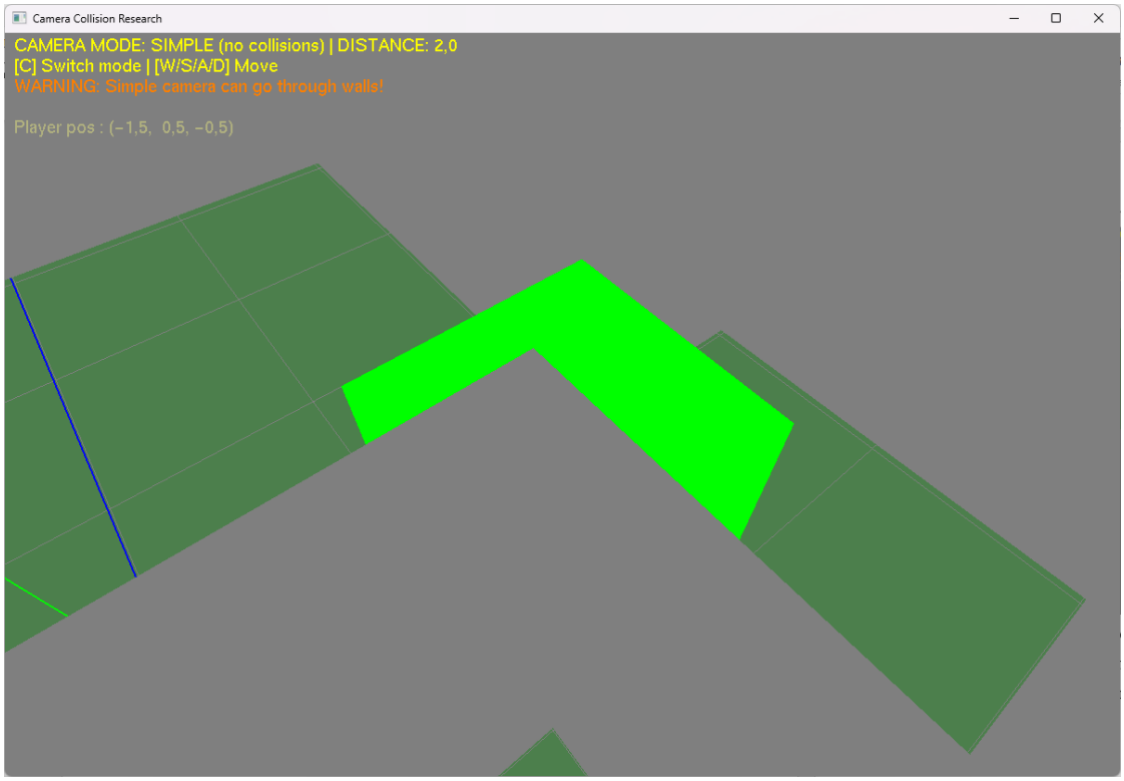

Рисунок 6 — Прохождение узкого коридора с простой камерой.

Игрок находится внутри узкого коридора, камера в режиме SIMPLE. Из-за отсутствия коллизий камера оказывается снаружи коридора, игрок видит стену снаружи, а не пространство внутри.

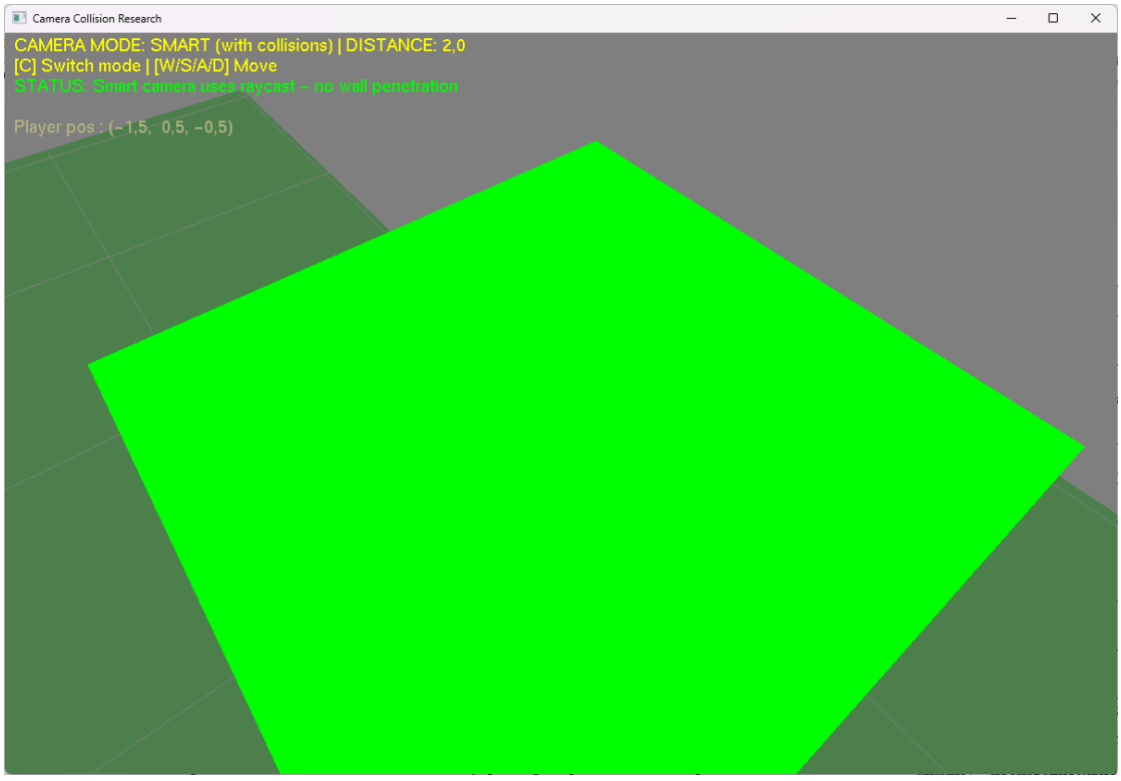

Рисунок 7 — Прохождение узкого коридора с умной камерой.

Тот же коридор, камера в режиме SMART. Камера автоматически сократила дистанцию и находится внутри коридора вместе с игроком. Навигация не нарушена.

Экспериментально установлено следующее. Результаты измерений сведены в таблицу 1.

Таблица 1 — Сравнение характеристик простого и умного режимов камеры

Критерий	Простой режим (Simple)	Умный режим (Smart)
Проход камеры сквозь стены	Происходит всегда при повороте за угол	Не происходит
Минимальная дистанция до игрока	Фиксированная (8.0)	Динамическая (от 2.0 до 8.0)
Обзор игрока в узких коридорах	Критически ограничен	Оптимальный
«Рывки» камеры при движении	Отсутствуют	Плавная адаптация
Время raycast на кадр (20 стен)	0.00 мс	0.05–0.08 мс
Время полного кадра	1.2 мс	1.3 мс
Увеличение нагрузки	0%	5–8%

Умный режим полностью решает проблему проникновения камеры в геометрию ценой небольшого увеличения вычислительной нагрузки. Согласно анализу производительности в книге Akenine-Moller и др. [5], для современных игровых систем эта нагрузка незначительна (менее 0.1 мс на кадр

для сцены с 20 стенами). В ходе экспериментов также установлено, что при увеличении количества стен до 50 время raycast возрастает до 0.2–0.25 мс, что всё ещё приемлемо для частоты 60 кадров в секунду. Линейный рост времени объясняется последовательной проверкой луча на пересечение со всеми стенами.

Дополнительно проведено исследование зависимости качества работы умной камеры от величины отступа δ. При $\delta<0.1$ камера визуально «касается» стены, что создаёт дискомфорт. При $\delta>0.5$ камера находится слишком далеко от стены и не даёт игроку достаточно близкого обзора. Оптимальным значением является $\delta=0.3$.

В ходе работы разработана программа на C++/OpenGL, демонстрирующая два режима управления камерой в 3D-сцене. Экспериментально подтверждено, что простой режим приводит к прохождению камеры сквозь стены при поворотах и движении в узких коридорах. Показана эффективность умного режима, использующего raycast для обнаружения препятствий и динамической корректировки дистанции. Предложенный алгоритм рекомендуется к использованию в 3D-платформерах, приключенческих играх и системах виртуальной реальности. Дальнейшим направлением исследований может стать реализация камеры с использованием BVH (Bounding Volume Hierarchy) для ускорения raycast в сценах с большим количеством стен (более 100), а также добавление возможности камере «обтекать» препятствия сбоку.

Список литературы

1. Gregory J. Game Engine Architecture. – 3rd ed. – CRC Press, 2018. – 1240 с.
2. Shikin E.V., Plis A.I. Curves and Surfaces on a Computer Screen. – M.: DIALOG-MIFI, 1996. – 240 с. (in Russian)
3. Foley J.D., van Dam A., Feiner S.K., Hughes J.F. Computer Graphics: Principles and Practice. – 3rd ed. – Addison-Wesley, 2014. – 1264 с.

4. McShaffry M. Game Coding Complete. – 4th ed. – Course Technology, 2013. – 912 c.
5. Akenine-Moller T., Haines E., Hoffman N. Real-Time Rendering. – 4th ed. – CRC Press, 2018. – 1178 c.