МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
БУРЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Институт математики и информатики
Кафедра прикладной математики

Выпускная квалификационная работа

Кроссплатформеное приложение для 3D-визуализации 2D-модели плана помещения методом бросания лучей

Выполнил : студент 4 курса группы 05230
Шорников Александр Евгеньевич
Научный руководитель: к.ф.-м.н., ст. преп.
Трунин Дмитрий Олегович
Научный консультант: асс. каф. ИТ
Брагин Александр Фёдорович

Улан-Удэ
2017

Оглавление

Стр.

ВВЕДЕНИЕ

Интерактивные планы помещений - доступный и современный способ разобраться в незнакомом здании. Сегодня интерактивная карта в отличие от обычной плоской в виде изображения отличается тем, что ее элементами можно управлять. Пользователь, находясь на странице, может свободно перемещаться по карте, находить объекты, схему прохода и просматривать информацию. При необходимости элементы карты могут включать в себя помимо реальных физических объектов (дома, улицы, парки, дороги и т.д.) дополнительно текстовую информацию, видеозаписи и ссылки на сайты.

К таким интерактивным планам предъявляются весьма суровые системные требования: поскольку они предназначены для самых разных платформ, и должны запускаться не только на можных десктопах, на и на мобильных устройствах, слабых нодах типа Raspberry Pi, встраивымх системах типа Arduino, а так же крутиться на сайтах. Не на всех из вышеперечисленых платформ может запускаться стандартная для современных 3D-визуализаций библиотека OpenGL(или WebGL в случае сайтов).

Для построения трехмерной визуализации выбран метод бросания лучей. Данный метод, являясь одним из простейших для трехмерной отрисовки, имеет ряд свойств согласующихся с задачами проекта: малую вычислительную сложность и простую реализацию без использования готовых библиотек трехмерной графики. Плюсами данного метода так же является то, что метод работает с плоской двумерной моделью помещения. То есть для визуализации помещения не надо строить полностью трёхмерную карту для рендеринга - достаточно дать плоскую карту.

Цель: создание кроссплатформеного псевдотрёхмерного движка для

3D визуализации здания по 2D плану.

Задачи:
–Модификация алгоритма рейкастинга для вещественных координат;
–Разработка математической модели для рейкастового рендерера;
–Изучение и освоение технологии кросс-компиляции Cheerp;

Объектом исследования является задача построения псевдотрёхмерной картинки.

Предметом исследования является изучение основных принципов построения псевдотрёхмерной картинки методом бросания лучей.

Глава 1

Постановка задачи

Итак, перед нами встала задача: создание интерактивного плана помещения методом бросания лучей. Поскольку современный человек предпочитает универсальные механизмы для любых платформ, мы так же озадачили себя кроссплатформеностью одного и того же кода, причём на таких казалось бы несовместимых платформах как web и нативная платформа(десктоп и мобильное приложение). Для достижения кроссплатформы на любой десктопной операционной системы, в разработке был применён язык $C++$‘14 с фреймворком Qt 5.8.0. Использование именно этой среды и языка обеспечило полную совместимость как в $*nix$ системах (в частности, в $MacOS\ X$, системах на основе ядра $Linux$, BSD - системах и тд), так и в среде $Windows$. Так же фреймворк Qt позволяет с лёгкостью перекомпилировать тот же самый код без особых изменений на любое $Android$ - устройство и устройство с iOs, например $iPhone$, что весьма актуально ввиду всеобщей распространённости таких гаджетов.

Однако, с платформой Web для встраивания на сайты пришлось повозиться. Любая web-страничка в Интернете представляет собой совокупность документа $HTML$, каскадную страницу стилей CSS, скриптовую часть на языке $JavaScript$ и внутренее содержание в виде текста, документов содержащих в себе картинки, аудиоокнтент, видеоконтент, мультимедиа, апплеты и гиперссылки на другие страницы. И для реализации на сайте нашего приложения необходимо было создать скриптовый апплет. Ввиду абсолютной несовместимости web-платформы с языком $C++$ в качестве скриптового языка для встраиваемых апплетов, необходимо было перенести реализацию на язык $JavScript$ с сохранением работоспособности десктопной версии.

Для решения этой проблемы мы разделили кодовую базу, сделав общей часть непосредственно решающую задачу, и отделив код специфичный для каждой из платформ. Код решающий задачу оперирует абстрактными классами канвы ($Core :: Canvas$) и буфера ($Core :: ImageBuffer$), которые в свою очередь конкретизируются для каждой платформы.

В итоге, в общую часть у нас вошло: Алгоритм рейкастинга, Представление карт уровней, Абстракции канвы и буфера кадров. И части для каждой платформ вошли конкретные реализации канвы и буфера кадров. Сейчас специальная часть для платформ занимают лишь 30% от общей кодовой базы, и нет причин для увеличения этой части с ростом проекта.

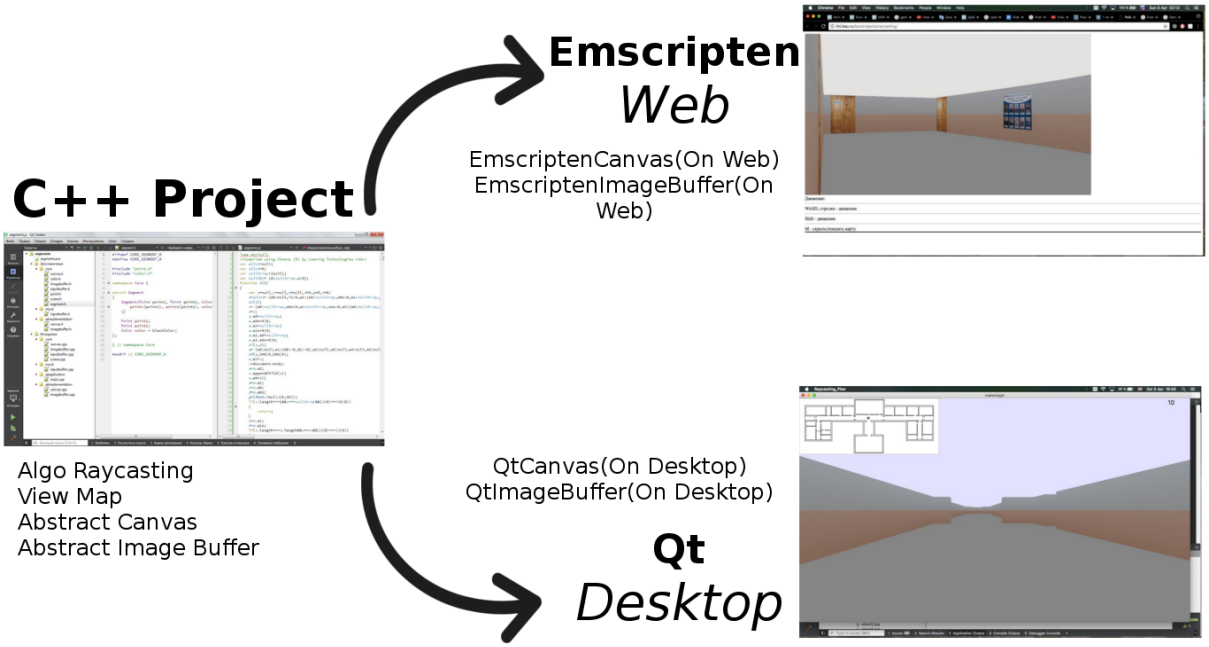

Рис. 1.1: Разделение конкретных реализаций

Для переноса общей части на *JavaScript* был использован кросс-компилятор $C++$ в *JavaScript* под названием *Emscripten*. *Emscripten* — ком-

пилятор из $LLVM$ байт-кода в $JavaScript$. $C/C++$ код может быть скомпилирован в $LLVM$ байт-код с помощью компилятора $Clang$. Некоторые другие языки так же имеют компиляторы в $LLVM$ байт-код. $Emscripten$ на основе байт-кода генерирует соответствующий $JavaScript$-код, который может быть выполнен любым интерпретатором $JavaScript$, например современным браузером. $Emscripten$ предоставляет: $emconfigure$ – утилита настройки окружения и последующего запуска $./configure$; $emmake$ – утилита для настройки окружения и последующего запуска $make$; $emcc$ – компилятор $LLVM$ в $JavaScript$.

Для автоматической сборки проекта и компиляции использована утилита $CMake$. $CMake$ - это универсальная кроссплатформенная утилита для автоматической сборки программы из исходных кодов. При этом сама $CMake$ непосредственно сборкой кода не занимается, а выступает в качестве front-end'а для back-end компилятора. И в итоге для общей сборки проекта необходим скрипт сборки для $CMake$, который выглядит следующим образом:

```
cmake -G"MinGW Makefiles" -DCMAKE_TOOLCHAIN_FILE=
C:\cheerp\share\cmake\Modules\CheerpToolchain.cmake
../segments && mingw32-make
```

Подобного рода скрипты позволяют скомпилировать проект и сразу в нативную версию, и в версию для web-приложения с условием готового платформозависимого канваса и буффера кадров для странички $HTML$. Выходной $JavaScript$-файл встраивается на заранее установленную страничку автоматически.

Глава 2

Математическая модель

2.1. Способ представления карты

Поскольку метод бросания лучей работает с двумерной моделью помещения, необходимо было придумать способ представления карты для его адекватного считывания подпрограммой считывания уровня и представления функцией обработки. Для классического рейкастинга уровень представляет собой двумерный массив, где значение каждого элемента массива является квадратом мира. Если значение ячейки равно 0, то квадрат оказывается пустым, и через него можно пройти. Если же значение больше 0, квадрат представляет собой стену определённого цвета или текстуры.

Для нашего проекта мы изменили алгоритм метода бросания лучей для работы в вещественных координатах. Благодаря этому мы избавились от ограничений старого задания уровней. Все графические примитивы на карте мы стали обозначать отрезками задаваемыми двумя вещественными точками, благодаря чему размер карты стал ограничиваться только ресурсами платформ, а так же увеличилась точность самого рендеринга карты.

Для того что бы представить карту каждого этажа в уровне нашего движка необходимо составить модель этого этажа. Как уже говорилось выше, каждый из графических примитивов в карте задаётся отрезком. И для представления карты внутри программы мы можем создать $C++$-класс, описывающий тип данных отрезков в качестве совокупности координат $(x_1, y_1) - (x_2, y_2)$, а так же флага принадлежности текстуры к данному отрезку. Таким образом мы полностью описываем программно

уровень карты.

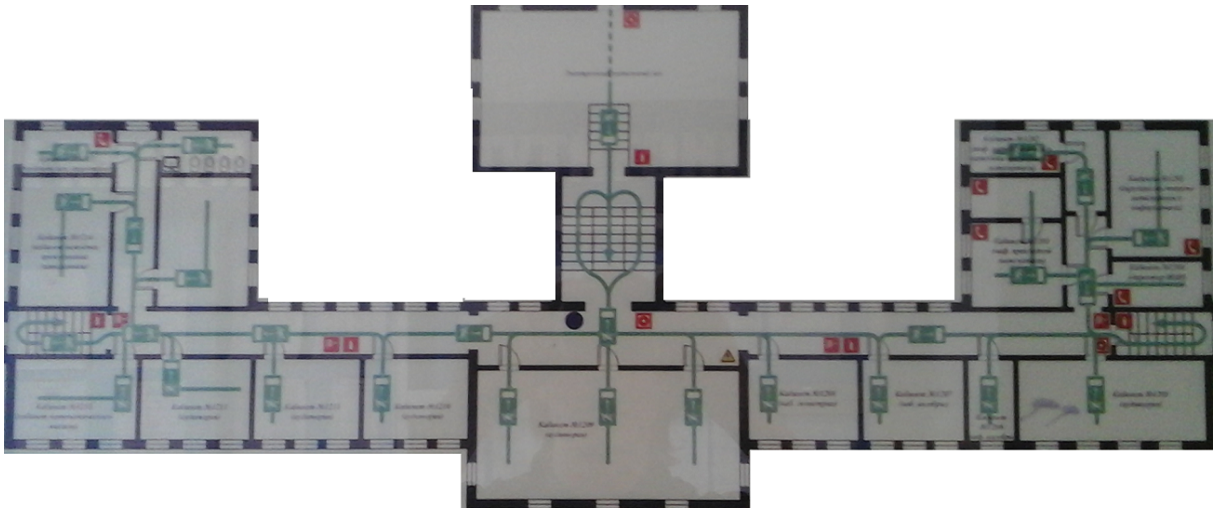

Рис. 2.1: Обработанное изображение 2 этажа 1 корпуса БГУ

Теперь для задания необходимого изображения этажа в карту можно представить её в векторном формате *SVG*. *SVG* (от англ. Scalable Vector Graphics — масштабируемая векторная графика) — язык разметки масштабируемой векторной графики, предназначеный для описания двумерной векторной и смешанной векторно/растровой графики в формате *XML*. Выбор пал имеенно на этот формат, поскольку внутрене формат *SVG* представляет собой *XML*-документ, содержащий помимо встроеной разметки координаты точек котрезков. Необходимо было только написать парсер для фильтрации полезной информации из *SVG*. Пример внутренего содержания *SVG*:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- 2017-04-04 23:45:02 Generated by QCAD SVG Exporter -->
<svg width="460mm" height="191mm" viewBox="0 0 460 191"
version="1.1" xmlns="http://www.w3.org/2000/svg"
style="stroke-linecap:round;stroke-linejoin:round;fill:none">
    <g transform="scale(1,-1)">
```

```
        <!-- Line -->
        <path d="M208,-65 L208,-112 "
        style="stroke:#000000;stroke-width:0.25;"/>
        <!-- Line -->
        <path d="M212,-112 L212,-65 "
        style="stroke:#000000;stroke-width:0.25;"/>
        <!-- Line -->
        <path d="M180,-2 L278,-2 "
        style="stroke:#000000;stroke-width:0.25;"/>
        <!-- Line -->
        <path d="M278,-2 L278,-61 "
        style="stroke:#000000;stroke-width:0.25;"/>
        <!-- Line -->
        <path d="M236,-61 L236,-65 "
        style="stroke:#000000;stroke-width:0.25;"/>
    </g>
</svg>
```

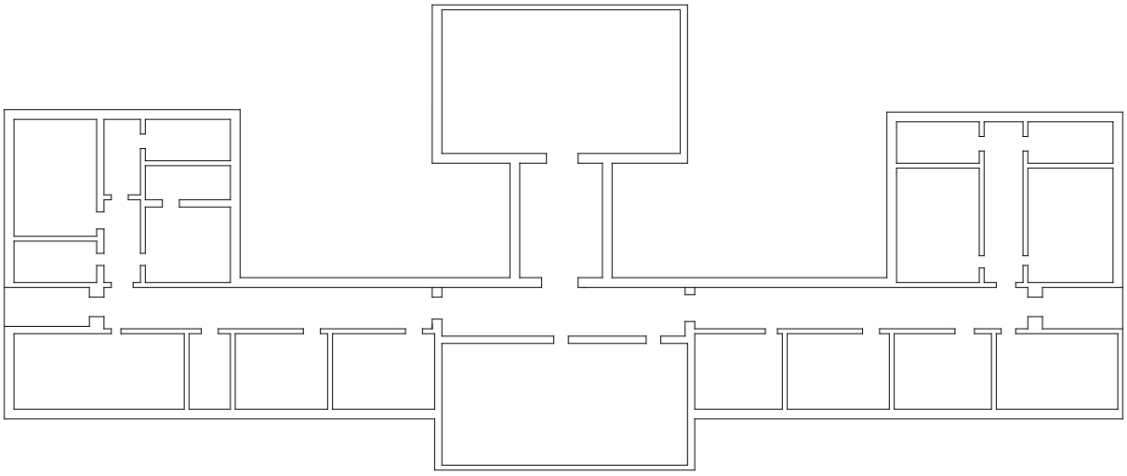

Рис. 2.2: Векторное представление изображения карты

В итоге, общая задач оцифрования уровня карты сводится к сканированию или простому фотографированию пожарного плана помеще-

ния, перевод данного изображения в векторный формат SVG, редактирование получившегося изображения при необходимости и "скармливание"получившегося файла программе считывания уровня. Полученное векторное изображение представляет собой уже размеченый xml-документ, который подпрограмма считывания уровня просто парсит, вычленяя необходимую информацию в виде координат точек отрезков.

2.2. Рейкастинг как алгоритм отрисовки

Итак, в общем случае у нас на карте задан уровень в виде множества отрезков, а также задана точка, интерпретирующая положение камеры на карте. Также задан угол обзора, показывающий какие сегменты попали в наблюдение камеры.

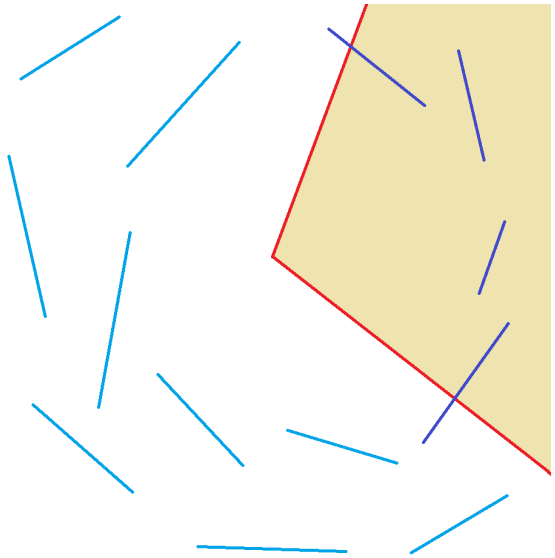

Рис. 2.3: Общий случай описания карты

Вектор описывающий положение камеры назовём $\overline{pos}$, а вектором $\overline{dir}$ назовем направление угла обзора. Длина этого вектора не будет влиять на угол обзора.

При помощи оценивания знака векторного произведения определяется принадлежность сегмента к углу обзора. Из местоположения наблюдателя (вектор $\overline{pos}$) через каждую точку отрезка экрана, представляющую собой одну колонку пикселей на мониторе, проводятся лучи. Их количество равно горизонтальному разрешению экрана. Расстояние между точками на отрезке равно полутора длинам вектора $\overline{dir}$ деленное на го-

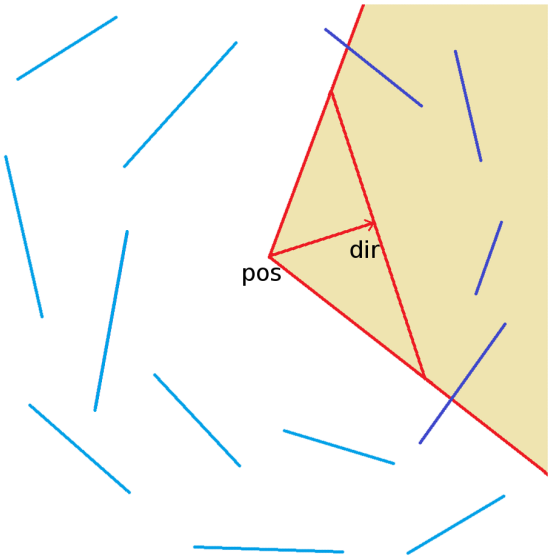

Рис. 2.4: Задание векторов

ризонталь разрешения экрана.

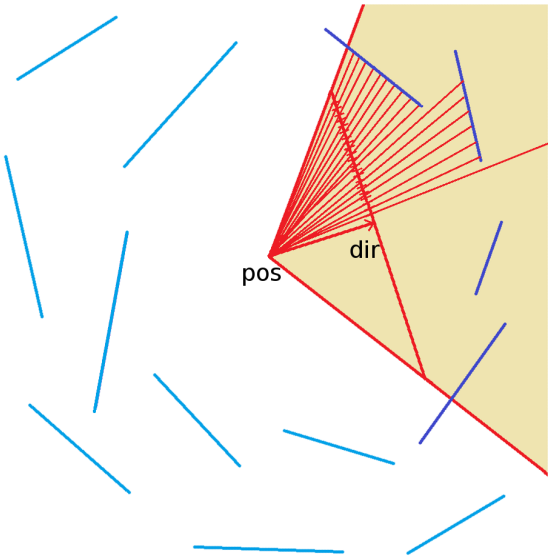

Рис. 2.5: Бросание лучей

Далее для каждой точки рассчитывается присутствие пересечения луча с сегментами по формуле Краммера и расчитывается расстояние по

лучу до препятствия. В соответствии с полученым расстоянием отображаются линии которые формируют стены. Длина линии обратно пропорциональна найденному расстоянию. Т.е. чем дальше от нас объект, тем он меньше. Эти линии формируют стены.

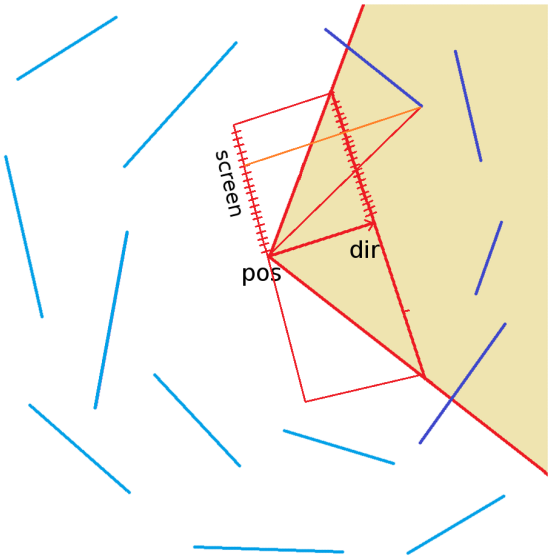

Рис. 2.6: Формирование изображения

К сожалению, при бросании луча из одной точки (точки где находится наблюдатель) возникает так называемый “эффект рыбьего глаза”, или широкоугольного объектива. Это происходит потому что расстояние пройденное по лучу, когда один конец луча неподвижен, а другой скользит вдоль прямой линии, изменяется по квадратичному закону относительно расстояния пройденного вдоль экрана. В результате этого границы стен на изображении описываются кривой второго порядка. Если мы хотим избежать этого эффекта, то нам нужно добиться того, чтобы расстояние пройденной лучом изменялось линейно от расстояния пройденного вдоль экрана. Для этого достаточно пускать лучи перпендикулярно экрану. Этого можно добиться минимальными изменениями в алгоритме, всего лишь изменив параметры передаваемые подпрограм-

ме по расчету расстояния - всего лишь умножив получение расстояние до пересечения на косинус угла между лучом и логическим экраном.

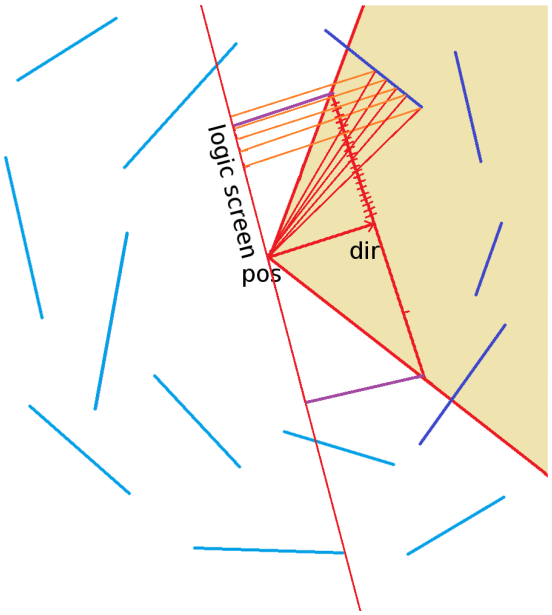

Рис. 2.7: Избавление от "эффекта рыбьего глаза"

2.3. Алгоритм текстурирования

2.4. Паттерн для интерактивного взаимодействия

Глава 3

Реализация

3.1. Инструментарий

3.2. Организация работы над проектом

3.3. Структура проекта (UML)

3.4. Реализация на целевых платформах

3.5. Внедрение ссылка на сайт ИМИ БГУ

ЗАКЛЮЧЕНИЕ

Литература

1. Бьерн Страуструп. Язык программирования C++(3 издание). -СПб.: Невский Диалект, 2008. - 504 с.

П Р И Л О Ж Е Н И Е 1

Программный код приложения для реализации метода бросания лучей