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

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

Отчет о программном проекте

на тему: 3D-renderer с нуля

(промежуточный, этап 2)

Выполнил:

Студент группы БПМИ191 _____ К.В. Амеличев
Подпись И.О.Фамилия

16.04.2021
Дата

Принял:

Руководитель проекта Дмитрий Витальевич Трушин
Имя, Отчество, Фамилия

доцент, к.ф.-м.н.
Должность, ученое звание

ФКН НИУ ВШЭ
Место работы (Компания или подразделение НИУ ВШЭ)

Дата проверки _____ 2021 _____ _____
Оценка (по 10-ти бальной шкале) Подпись

Москва 2021

Содержание

Аннотация

В рамках данной работы я разрабатываю библиотеку для языка программирования C++, занимающуюся отрисовкой объектов в трехмерном пространстве. После написания библиотеки планируется сделать тестовое приложение на ее основе.

1 Введение

В рамках изучения тех или иных геометрических объектов возникает необходимость их графического отображения. Сложность в визуализации трехмерных объектов в том, что их нельзя изобразить на бумаге одним рисунком без потери информации. Настоящий трехмерный объект можно повертеть в руках и разглядеть со всех сторон, а вот рисунок на бумаге или экране может быть непонятен.

В рамках данной работы создается библиотека для языка программирования C++, которая визуализирует объекты в трехмерном пространстве, требуя от пользователя минимальных усилий. Важной особенностью этой работы является то, что работа с графикой построена буквально «с нуля» — из пререквизитов нужно только средство для отрисовки пикселей на экране.

Задачами данной работы является:

1. Исследование существующих технологий и принципов в сфере 3д-графики
2. Выбор архитектуры
3. Разработка интерфейса
4. Реализация функционала
5. Создание тестового приложения
6. Последующее документирование и оформление в едином стиле библиотеки с открытым исходным кодом
7. Описание теоретических знаний, которые были получены и применены в рамках исследования и разработки проекта.

Весь исходный код проекта находится в репозитории по следующей ссылке

Также используются следующие источники информации:

1. [1] — сопроводительная инструкция к большому пакету для работы с 3d-графикой, в которой рассказываются основные принципы создания подобных приложений.
2. [3] — математическая основа для библиотеки.
3. [2]— курс лекций по компьютерной графике

1.1 Функциональные требования

- Отрисовка 3д-объектов.
- База стандартных объектов для отрисовки, таких как куб или тетраедр.
- Возможность создать любой триангулируемый объект в пространстве.
- Задание параметров объекта, масштабирование, цвет.
- Перемещение объектов.
- Поворот объектов произвольным образом.
- Возможность последовательной смены кадров, создание анимации
- Обработка взаимодействия пользователя и приложения — пользовательские повороты и перемещения объектов, в частности по взаимодействию с клавиатурой
- Возможность программно создавать и удалять объекты во время анимации.
- Скрытие интерфейса отрисовки от пользователя — пользователю нужно только за создавать объектов и взаимодействовать с ними
- Повороты и перемещение камеры
- Возможность работы в статическом режиме — создать кадр и сохранить его как файл с изображением.

Структура приложения следующая: для создания одного окна с графикой требуется создать одного представителя главного класса Application. После чего Application создает окно, в котором будут отрисовываться объекты. Пользователь может добавлять объекты (такие как куб, тетраедр, или произвольный объект, созданный на основе базового класса) каждому Application'у, давать ему команды для трансформации содержимого (перемещения, переворотов), а также в необходимый момент перерисовывать кадр.

1.2 Технические требования

- Разработка на языке C++.
- Сборка проекта с помощью CMake.
- Работа с 2д-графикой с помощью библиотеки SFML.
- Отрисовка графики на CPU.
- Открытый исходный код.
- Система поддержки версий Git.
- Ошибки программиста отлавливаются assert-ами.
- В случае возникновения системных ошибок выводится критический лог об ошибке. Затем, в случае, если пользователь вызывал отрисовку окна, то функция отрисовки бросит исключение std::runtime_error. Если же пользователь делал произвольное действие с объектами или камерой (добавление объекта, перемещение, поворот), то действие проигнорируется.
- Google C++ styleguide.

Тестируется библиотека с помощью консольного приложения, в котором создаются произвольные 3d-объекты, после чего они переносятся на экран в интерактивном режиме, с возможностью перемещения, сдвигов и поворотов.

2 Изучение аналогов

Поскольку в основном 3d-графика в программировании используется для создания игр, то почти все библиотеки, которые мне удалось найти, имели перегруженный интерфейс под игры, как у Polycode, где надо создавать дополнительные xml-конфигурационные файлы.

Также есть такие популярные фреймворки, как OpenGL или Direct3d, но они настолько низкоуровневые, что при написании простого приложения почти все время уйдет на их изучение.

Получается, что у низкоуровневых библиотек нужно продумывать слишком много параметров, а у высокоуровневых библиотек сложная архитектура и протокол взаимодействия пользователя с библиотекой.

Разумеется, библиотека будет проигрывать в производительности аналогам, как минимум поскольку в ней не предусмотрена отрисовка через GPU. Но важной целью работы является именно создание архитектуры отрисовщика 3d-графики с нуля.

3 Содержательная часть

3.1 Основы 3d-графики

При работе над 3d-рендерером достаточно важной частью работы является понимание математики, происходящей внутри обработчика графики. Основная проблема заключается в том, что на одни и те же объекты можно смотреть с разных сторон, и в зависимости от этого получать разную картинку. Поскольку все случаи не разберешь, на помощь приходит линейная алгебра.

3.1.1 Объекты для отрисовки

Все объекты разделяются на два типа — отрезки и треугольники. На самом деле, достаточно только треугольников, но отрезки рисовать значительно проще, при этом ими можно отображать каркасы объектов, что тоже имеет смысл.

Каждый объект задается набором точек-вершин в глобальной системе координат, а также наборами пар и троек индексов, которые определяют отрезки и треугольники. Отрезки используются для отрисовки ребер фигуры, а поверхности триангулируются. Объекты, созданные из отрезков и треугольников, не меняют своего расположения в глобальной системе координат при передвижении камеры, а также могут двигаться независимо друг от друга.

Например, если задать пирамиду OABCD с квадратом в основании, то у нее будет 8 ребер, 5 вершин, 5 граней и 6 треугольников в триангуляции.

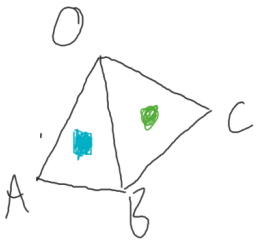


Рис. 1: Пирамида OABCD, вид сбоку

Рис. 2: Пирамида OABCD, вид снизу. Нижняя грань разбита на два треугольника

3.1.2 Экран и камера

Объекты надо как-то отображать на экране. Экран представляется произвольной плоскостью в пространстве, на который проецируются все точки. Поскольку экран имеет ограниченный размер, то и на плоскости выбирается ограниченный прямоугольник. Те точки, которые попадут на прямоугольник экрана, и будут отрисованы. Остальные находятся вне поля зрения.

Для того, чтобы правильно определить, какие объекты видно наблюдателю, используется объект камеры. Этот объект задается своим положением в пространстве ($O_c = \begin{bmatrix} x_{camera} \\ y_{camera} \\ z_{camera} \end{bmatrix}$), а также матрицей текущего пово-

рота камеры. Назовем ее M_c. Еще нам понадобятся габариты параллелипипеда, который видит наша камера — координаты ближней левой нижней и правой верхней дальней точки ((l, b, n) и (r, t, f) соответственно).

3.1.2.1 Шаг 0. Однородные координаты

Поскольку все операции с координатами имеет смысл выражать через векторы и матрицы, возникает проблема с тем, что не все афинные и проективные преобразования представляются произведением трехмерных матриц. Для этого вводят однородную систему координат, которая работает с четырехмерным пространством по следующему правилу:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \to \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}, w \neq 0 \to \begin{pmatrix} \frac{x}{w} \\ \frac{y}{w} \\ \frac{z}{w} \end{pmatrix}$$

Такая система разрешает, во-первых, прибавить произвольное число к любой из координат трехмерного вектора:

$$\begin{bmatrix} 1 & 0 & 0 & a_x \\ 0 & 1 & 0 & a_y \\ 0 & 0 & 1 & a_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + a_x \\ y + a_y \\ z + a_z \\ 1 \end{bmatrix} \to \begin{bmatrix} x + a_x \\ y + a_y \\ z + a_z \end{bmatrix}$$

Также появляется возможноть разделить координаты на произвольную линейную функцию от координат:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ k_x & k_y & k_z & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ k_x x + k_y y + k_z z \end{bmatrix} \to \begin{bmatrix} \frac{x}{k_x x + k_y y + k_z z} \\ \frac{y}{k_x x + k_y y + k_z z} \\ \frac{z}{k_x x + k_y y + k_z z} \end{bmatrix}$$

Утверждается, что такого дополнительного функционала хватает, чтобы сделать необходимые нам преобразования камеры.

3.1.2.2 Шаг 1. Сдвиг камеры

Чтобы переместить камеру в центр координат, нужно из каждой координаты x, y, z вычесть ее текущее значение. Это простая единичная матрица с правым столбцом:

$$\begin{bmatrix} 1 & 0 & 0 & -x(O_c) \\ 0 & 1 & 0 & -y(O_c) \\ 0 & 0 & 1 & -z(O_c) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

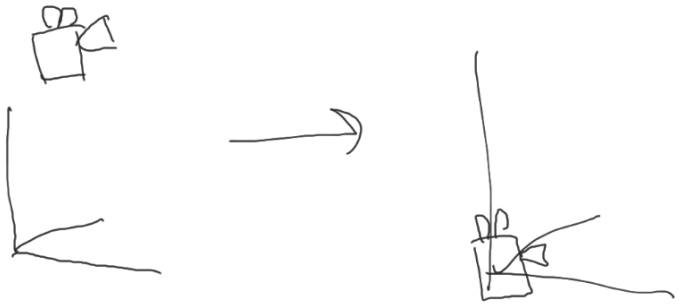


Рис. 3: Шаг 1

3.1.2.3 Шаг 2. Поворот камеры

Поскольку каждый происходящий поворот камеры является ортогональным преобразованием, то обратное преобразование — это просто транспонированная матрица всех поворотов. Поскольку мы хотим получить единичную матрицу, то нам надо просто транспонировать матрицу всех преобразований. А именно, если на текущий момент базисные векторы пространства камеры после всех поворотов равны u, v, w, то надо сделать такое преобразование:

$$\begin{bmatrix} x_u & y_u & z_u & 0 \\ x_v & y_v & z_v & 0 \\ x_w & y_w & z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

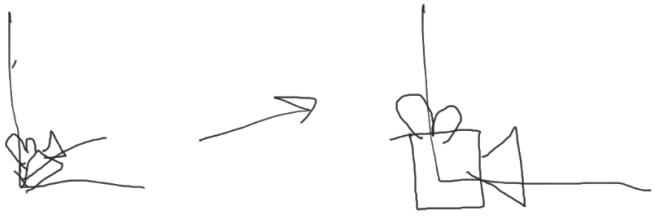


Рис. 4: Шаг 2

3.1.2.4 Шаг 3. Проективное преобразование

Для того, чтобы добавить эффект перспективы, область видимости задается в виде усеченной четырехугольной пирамиды. Такой объект сложно спроецировать на экран, поэтому для начала его проективным преобразованием переводят в прямоугольный параллелипипед. После этого проекция оказывается просто отбрасыванием координаты.

У этого подхода есть еще одно приемущество. Отбрасываемая координата задает глубину точки относительно экрана. Поэтому, при прочих равных, надо будет в конкретном пикселе отрисовывать ту точку, которая имела меньшую глубину. Такой метод называется z-буфером.

Сначала покажу матрицу, а потом объясню, почему она делает то, что нам нужно.

$$\begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -nf \\ 0 & 0 & n & 0 \end{bmatrix}$$

Матрица проективного преобразвания переведет координаты $(x, y, z) \to (\frac{x}{z}, \frac{y}{z}, 1 + \frac{f}{n} - \frac{f}{z})$. Можно заметить, что первые две координаты делятся на z, а от преобразования третьей координаты нам важно только то, что сохраняется неравенство. А именно, если раньше у одной точки z-координата была больше, чем у другой, то и после преобразования будет.

А новые двумерные точки (назовем их x_0, y_0) верны (с точностью до масштаба) из подобия треугольников, образованных направлением, в котором смотрит камера, и вектором (x, y, z):

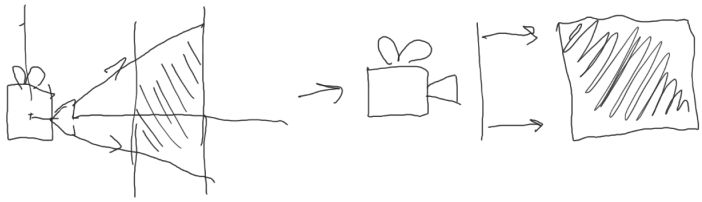


Рис. 5: Шаг 3

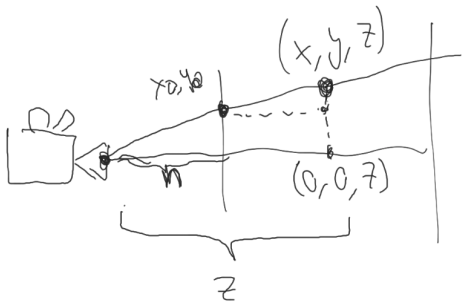


Рис. 6: Вот тут можно видеть подобие треугольников

3.1.2.5 Шаг 4. Перемещение параллелипипеда в центр координат

Чтобы было удобнее дальше работать, параллелипипед перемещается своим центром в центр координат.

$$\begin{bmatrix} 1 & 0 & 0 & \frac{-l-r}{2} \\ 0 & 1 & 0 & \frac{-b-t}{2} \\ 0 & 0 & 1 & \frac{-n-f}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

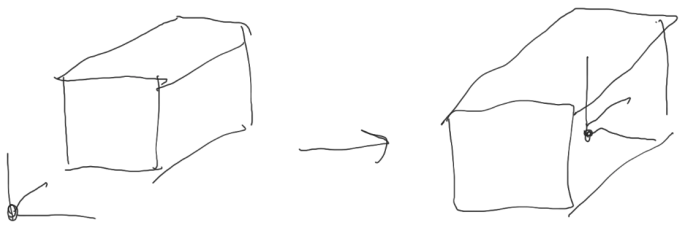


Рис. 7: Шаг 4

3.1.2.6 Шаг 5. Масштабирование в куб $2 \times 2 \times 2$

Теперь объект масштабируется в куб $(-1, -1, -1), (1, 1, 1)$. Это нужно для того, чтобы в дальнейшем можно было работать с любым экраном

$$\begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{n-f} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

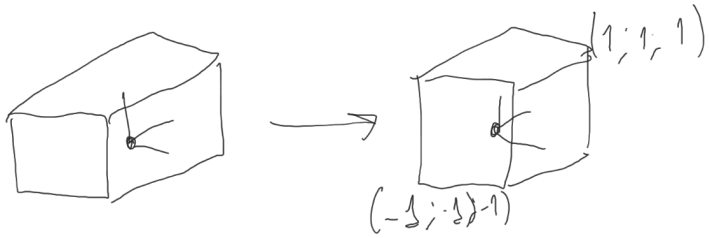


Рис. 8: Шаг 5

3.1.2.7 Шаг 6. Перенос в плоскость экрана

Куб надо отобразить на экране. Для этого мы масштабируем координаты (x, y), а также координата z отвечает за глубину объекта.

$$\begin{bmatrix} \frac{Screen_x}{2} & 0 & 0 & \frac{Screen_x - 1}{2} \\ 0 & \frac{Screen_y}{2} & 0 & \frac{Screen_y - 1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Рис. 9: Шаг 6

3.1.2.8 Здоровенная матрица

В итоге получается, что камера должна сделать большое преобразование из нескольких матриц:

1. Сдвиг камеры в центр координат
2. Поворот всего мира для перехода в базис камеры
3. Проективное преобразование
4. Сдвиг центра видимого параллелепипеда в центр координат
5. Масштабирование в куб $2 \times 2 \times 2$
6. Прообразование для получения пикселей экрана

$$M = \begin{bmatrix} \frac{Screen_x}{2} & 0 & 0 & \frac{Screen_x - 1}{2} \\ 0 & \frac{Screen_y}{2} & 0 & \frac{Screen_y - 1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{n-f} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & \frac{-l-r}{2} \\ 0 & 1 & 0 & \frac{-b-t}{2} \\ 0 & 0 & 1 & \frac{-n-f}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot$$

$$\cdot \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -nf \\ 0 & 0 & n & 0 \end{bmatrix} \begin{bmatrix} x_u & y_u & z_u & 0 \\ x_v & y_v & z_v & 0 \\ x_w & y_w & z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -Camera_x \\ 0 & 1 & 0 & -Camera_y \\ 0 & 0 & 1 & -Camera_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

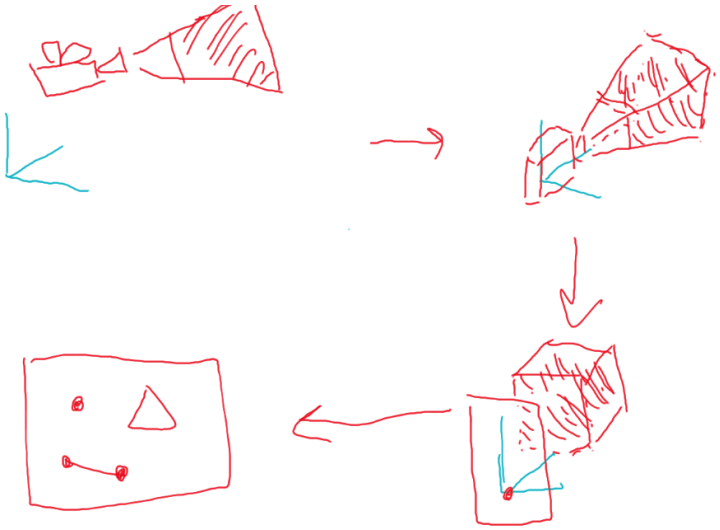


Рис. 10: Весь пайплайн

3.2 Отрисовка двумерных объектов

Дальше происходит отрисовка двумерных объектов на экране по пикселям и заданным вершинам. Для отрезка это достаточно несложная задача, а вот для заливки треугольника нужно придумывать эффективные способы. Автор использует так называемую «сканирующую прямую» — треугольник отрисовывается послойно, на каждом слое выделяется вертикальная полоса внутри треугольника, и заполняется только она. Вертикальные полосы выбираются только между самой левой x-координатой треугольника и правой x-координатой.

Таким образом, мы сначала перебираем x в отрезке $[min\_x, max\_x]$, затем y в отрезке $[min\_y(x), max\_y(x)]$, после чего z вычисляется как $\frac{(z(x,max\_y)-z(x,min\_y))\cdot(y-min\_y)}{max\_y-min\_y}$ (это просто линейное масштабирование вектора между крайними точками вертикальной полосы.

Чтобы посчитать $z(x, y)$, нужно выразить точку (x, y) в базисе из двух сторон треугольника, после чего получившиеся коэффициенты применить к исходному четырехмерному треугольнику:

$z(x, y) = z(\alpha \cdot \vec{AB_4} + \beta \cdot \vec{AC_4})$, где α, β находятся из уравнения $\begin{bmatrix}\vec{AB_2} & \vec{AC_2}\end{bmatrix}\begin{bmatrix}\alpha\\ \beta\end{bmatrix} = \begin{bmatrix}x\\ y\end{bmatrix}$, а индексы 2 и 4 означают размерность треугольника, в котором берутся векторы.

Осталось понять, как найти min_y, max_y.

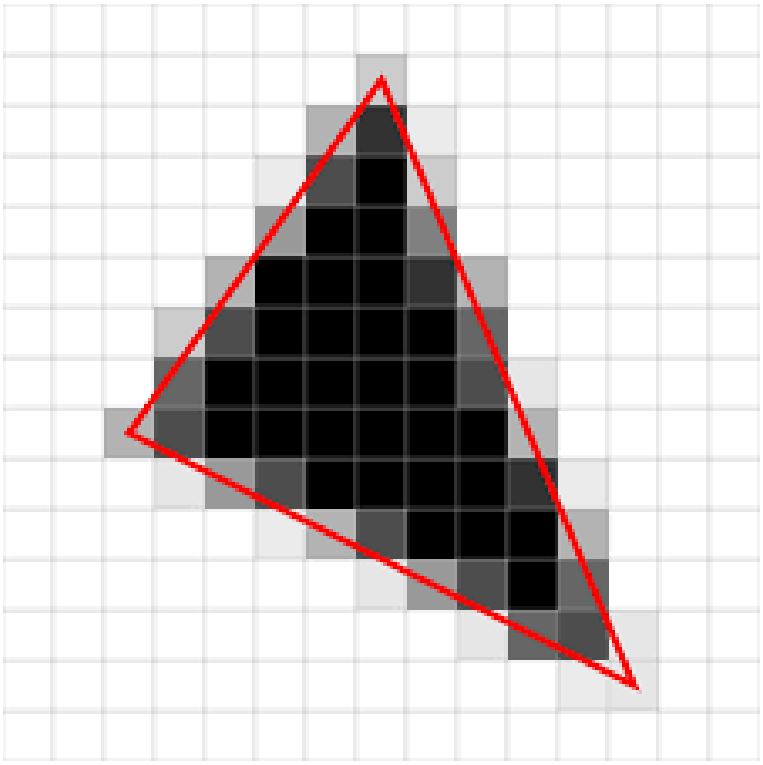


Рис. 11: Двумерный треугольник. Те клетки, которые пересекаются красной линией, имеют минимальную и максимальную y-координату в своей вертикальной полосе. Клетки между ними и надо закрасить.

3.2.1 Сортировка точек в треугольнике

Прежде, чем заполнять полосу в треугольнике, сделаем предобработку. Для того, чтобы уменьшить число случаев, имеет смысл заранее отсортировать все точки треугольника против часовой стрелки, начиная с самой левой (среди самых левых — самой нижней). Для этого нужно найти первую точку как самую левую (среди самых левых — самую нижнюю). Назовем эту точку A, а две оставшиеся B и C (их порядок пока что не определен). После чего достаточно вычислить векторное произведение двух векторов $(B - A) \times (C - A)$. Векторное произведение антисимметрично, поэтому мы можем поменять местами точки B и C так, что произведение будет неотрицательным. А это значит, что точка C будет лежать в левой полуплоскости относительно ориентированного отрезка AB, что и дает нам отсортированность против часовой стрелки.

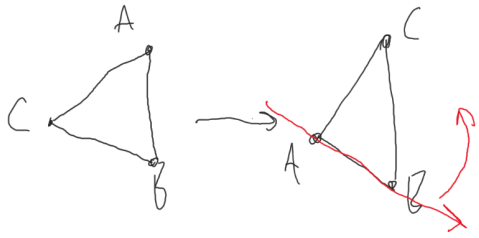


Рис. 12: Треугольник до и после переупорядочивания вершин

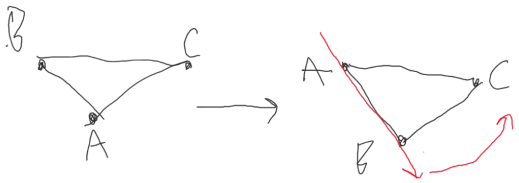


Рис. 13: Еще один треугольник до и после переупорядочивания вершин

3.2.2 Выделение полосы в треугольнике

На текущей стадии мы хотим для каждой вертикальной полосы в треугольнике найти ее границу. Имеющийся у нас инвариант — точки треугольника заранее отсортированы против часовой стрелки, при этом первой в порядке сортировки идет самая левая, а среди самых левых — самая нижняя.

Теперь для выделения полосы есть всего два случая — когда точка C левее точки B, и когда правее. В обоих случаях, зная x-координату полосы, нужные y-координаты вычисляются как точки на соответствующих отрезках.

Дальше я реализую две симметричные функции для нахождения верхней и нижней границы полосы. Рассмотрим поиск нижней границы. Я смотрю на отрезок AB. Если он пересекает полосу, то точка находится на нем, ее можно выразить из линейного уравнения. Если он не пересекает полосу, то точка находится на соседнем отрезке BC, линейное уравнение решается для него. Мы добились такого расположения точек за счет предварительной сортировки.

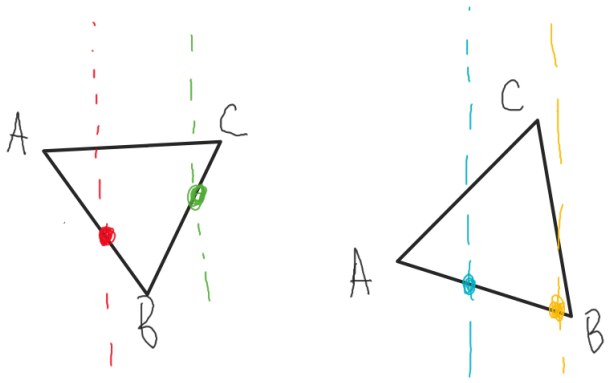


Рис. 14: Два случая расположения треугольников. Точки пересечений показывают минимальный y в полосе. Соответственно, в зависимости от случая, формулы получаются немного разные, а именно:

Для левой картинки	Для правой картинки
$min\_y(x) = \begin{cases} A.y + \frac{(x-A.x)\cdot(B.y-A.y)}{B.x-A.x} & x < B.x \\ C.y + (\frac{(x-C.x)\cdot(B.y-C.y)}{B.x-C.x} & x \geq B.x \end{cases}$	$max\_y(x) = \begin{cases} A.y + \frac{(x-A.x)\cdot(C.y-A.y)}{C.x-A.x} & x < C.x \\ B.y + \frac{(x-B.x)\cdot(C.y-B.y)}{C.x-B.x} & x \geq C.x \end{cases}$
$max\_y(x) = A.y + \frac{(x-A.x)\cdot(C.y-A.y)}{C.x-A.x}$	$min\_y(x) = A.y + \frac{(x-A.x)\cdot(B.y-A.y)}{B.x-A.x}$

3.2.3 Сложность

Если оценить сложность, то мы выделим необходимую область за $O(w)$, где w — ширина треугольника. Таким образом, самая неэффективная часть в отрисовке треугольника — отрисовка каждого пикселя за $O(S_\Delta)$. Но поскольку для любого растрового экрана это действие нам понадобится, можно считать, что алгоритм досточно эффективный для нашей задачи.

3.3 Библиотека: описание элементов

3.3.1 Авторское: Точки, матрицы, геометрия

Матрицы Matrix нужны для работы с матричными вычислениями. Пользователю доступно создание матриц произвольного размера, операторы для сложения, вычитания и перемножения матриц. Также доступен поиск обратных матриц, выражение вектора через базис пространства, транспонирование.

Точки Point4d представляют собой набор из четырех координат или матрицу 4×1. Это объект, который нужен для удобной обертки над несколькими переменными. Для него также доступны арифметические операции, а также отождествление с вектором и возможность нахождения длины. Также есть отождествление 4d-точек с трехмерными через единичную последнюю координату (однородные координаты).

Для работы с двумерной геометрией используется класс двумерного треугольника Triangle2d. Основной его смысл — давать нужную информацию о треугольнике — координаты точек, сохраненные в отсортированном порядке.

Все многомерные треугольники для отрисовки передаются как объекты класса Triangle4d, который выступает в роли контейнера.

3.3.2 Авторское: Объекты, мир

Структура объектов такая: есть объекты типа WireObject, есть объекты типа SurfaceObject, являющиеся наследником типа WireObject. WireObject хранит каркас фигуры: точки и ребра, а SurfaceObject расширяет WireObject данными о триангулированной поверхности объекта. Взаимодействие пользователей происходит с объектами типа SurfaceObject. Мир World является просто контейнером, в котором лежат все объекты. Объекты можно[1] поворачивать и двигать.

И объекты, и мир поддерживают возможность итерирования по своему содержимому: у объектов можно просмотреть все ребра и грани, а у мира можно посмотреть все объекты.

3.3.3 Авторское: Камера

Камера Camera отвечает за то, чтобы переводить объекты из системы координат мира в систему координат экрана. Для этого этот объект генерирует матрицу преобразования и применяет ее ко всем точкам, которые надо отобразить. Также камеру можно поворачивать и двигать, что отражается на матрице преобразования.

3.3.4 Авторское: Экран

Экран Screen нужен для растрового представления экрана. Он представляет собой несколько табличек с данными по каждому пикселю экрана с возможностью доступа и изменения. А именно, позволяет поставить значение цвета пикселя и узнать текущее значение z-буфера в точке.

3.3.5 Авторское: Renderer

Renderer отвечает за логику отрисовки. Он принимает набор объектов SurfaceObject, после чего берет отрезки и треугольники в многомерном пространстве, с помощью камеры Camera переносит их в двумерное пространство, выполняет алгоритм растеризации и переносит весь кадр на экран Screen, после чего отрисовывает кадр с помощью библиотеки SFML.

3.3.6 Авторское: Application

Объект Application является основным объектом приложения для пользователя — именно оно осуществляет взаимодействие между всеми остальными объектами. Иначе говоря, Application берет все объекты SurfaceObject из контейнера World, после чего просит Renderer отрисовать эти объекты.

Также Application поддерживает интерактивный функционал, доступный в SFML — позволяет пользователю настроить listener-ы на различные системные события, которые умеет отлавливать SFML (например, нажатие на клавиши).

[1] когда-нибудь будет можно, строго говоря

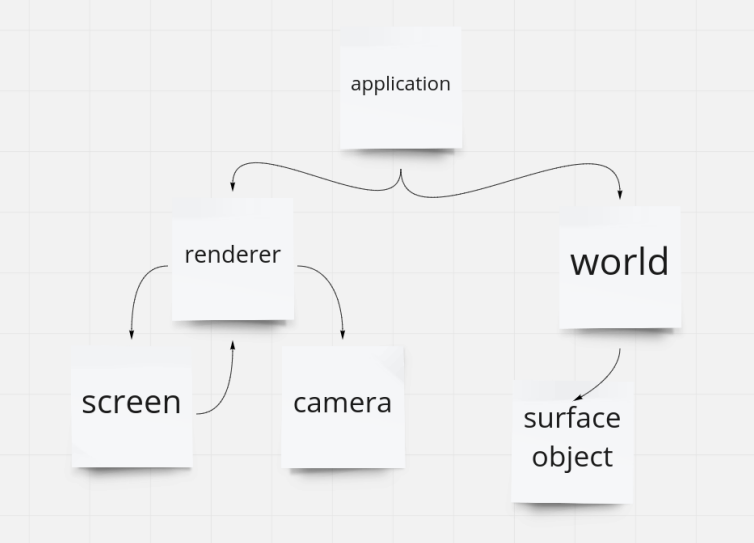


Рис. 15: Взаимоотношение классов. Матрицы, точки и треугольники убраны специально, потому что они используются практически везде, не имеют сложных зависимостей (только треугольники → точки), а при этом сильно усложнили бы схему.

3.3.7 Стороннее: SFML

SFML — это библиотека для отрисовки 2d-графики. У нее есть два основных приемущества:

1. Достаточно простая.
2. Поддерживает создание интерактивных приложений.

Используется для отрисовки массива точек (растеризованный экран), а также для отрисовки отрезков-ребер.

3.4 Библиотека: Pipeline

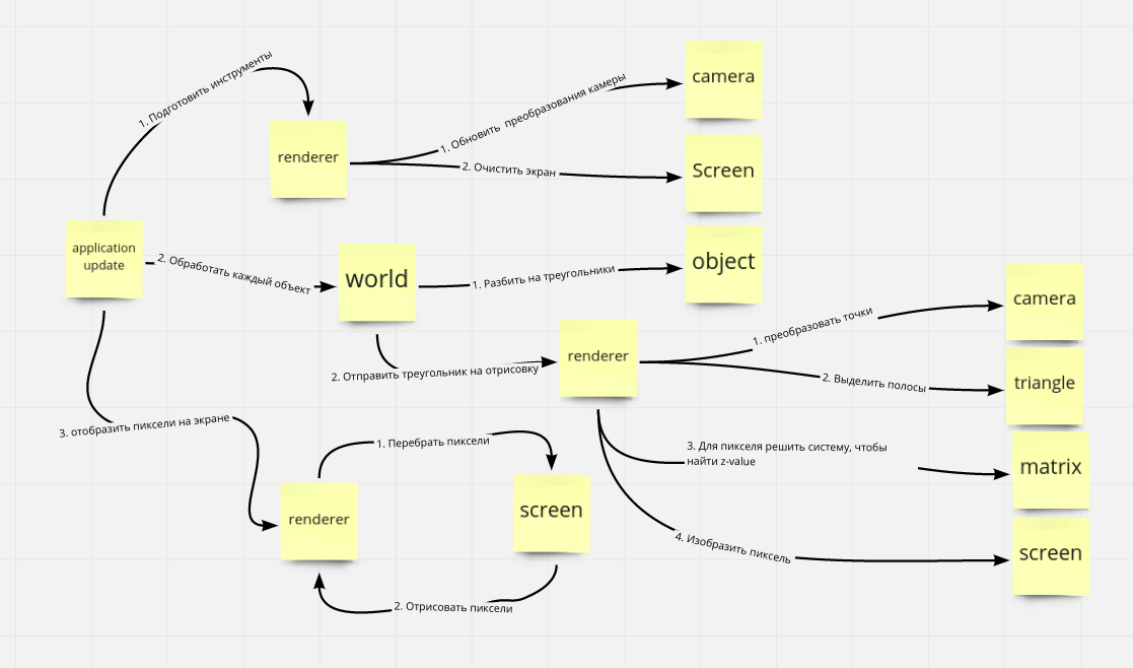
3.4.1 Описание.

В итоге Pipeline отрисовки (Application.update(), обновляющий кадр) получился таким:

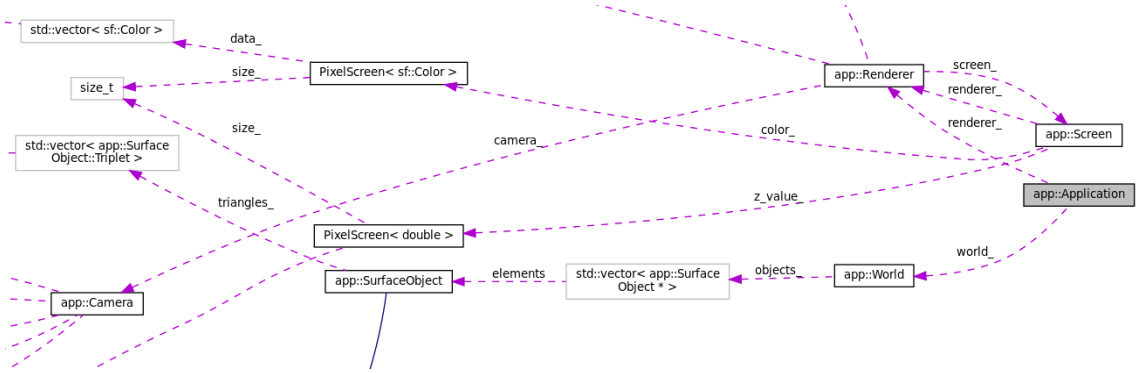
- Application.update() вызовет Renderer.prepare(), чтобы очистить экран Screen.clear() и обновить матрицу преобразования Camera.create_transform()
- Application.update() вложенными циклами переберет сначала все объекты в мире, а потом все треугольники в объекте.
- Эти треугольники отправятся на Renderer.draw() — сначала Camera.project_point() переведет их в координаты на экране, потом прогоняется алгоритм для двумерного треугольника, который получает пиксели треугольника.
- Полученные пиксели отправляются в Screen.draw(), который определит ближайший пиксель к плоскости экрана (минимальная z-value)
- Application.update() в самом конце делает Renderer.update(), который делает Screen.update(), который проходит по всем пикселям и вызывает Renderer.draw() для них, в результате чего множество пикселей оказывается на экране.

Отрисовку кадра вызывает пользователь — на его усмотрение он может это делать как только после изменений, так и постоянно (например, бесконечным циклом).

3.4.2 Схема пайплайна.

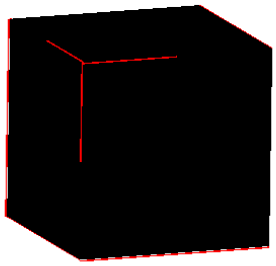


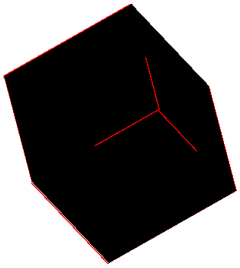
3.4.3 Схема владения объектов



3.5 Тествое приложение и текущие результаты.

Для теста на текущей стадии разработки используется приложение, в котором можно вращать и двигать кубик. Далее привожу две картинки того, как это у меня выглядит сейчас.





Список литературы

[1] https://lorensen.github.io/VTKExamples/site/VTKBook/00Preface/. [VTKBook].

[2] https://www.youtube.com/watch?v=4z6zRet_gkg&list=PLbCDZQXIq7uYaf263gr-zb0wZGoCL-T5G. [Urtech University lectures].

[3] Eric Lengyel. *Mathmatics for 3d game programming and computer graphics*. Course Technology PTR, 2012.