АННОТАЦИЯ

Настоящий программный документ представляет собой пояснительную записку к программному проекту «3D Renderer».

Раздел «Введение» включает в себя наименование программы и документ, на основании которого ведётся разработка, с указанием организации, утвердившей данный документ.

В разделе «Назначение и область применения» содержатся функциональное и эксплуатационное назначение программы и краткая характеристика области её применения.

В разделе «Технические характеристики» присутствуют следующие подразделы: постановка задачи на разработку программы, описание функционирования программы, описание и обоснование алгоритма работы программы, описание и обоснование математического алгоритма работы программы, описание выбора метода организации входных и выходных данных, описание работы с файловой системой, описание и обоснование выбора состава технических и программных средств.

В разделе «Ожидаемые технико-экономические показатели» указана предполагаемая потребность и экономические преимущества разработки по сравнению с отечественными и зарубежными образцами или аналогами.

Программный документ разработан в соответствии с требованиями:

1. ГОСТ 19.101-77 Виды программ и программных документов [1];

2. ГОСТ 19.102-77 Стадии разработки [2];

3. ГОСТ 19.103-77 Обозначения программ и программных документов [3];

4. ГОСТ 19.104-78 Основные надписи [4];

5. ГОСТ 19.105-78 Общие требования к программным документам [5];

6. ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом [6];

7. ГОСТ 19.404-79 Пояснительная записка. Требования к содержанию и оформлению [7].

Изменения к Пояснительной записке оформляются согласно ГОСТ 19.603-78 [8], ГОСТ

19.604-78 [9].

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

1.1. Наименование программы

Наименование программы – «3D Renderer».

Наименование программы на английском языке – «3D Renderer».

Краткое наименование программы, используемое далее в документе – Приложение.

1.2. Документ, на основании которого ведётся разработка

Учебный план подготовки бакалавров по направлению 09.03.04 «Программная инженерия» и утвержденная академическим руководителем программы тема курсового проекта.

2. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ

2.1. Назначение программы

2.1.1. Функциональное назначение

Функциональным назначением данного программного продукта является предоставление возможности просмотра 3D объектов на экране компьютера. Приложение должно отображать 3D объекты, которые были добавлены на сцену, причём которые видны камере. Приложение предоставляет возможность пользователю перемещать камеру для просмотра сцены и добавлять объекты на сцену.

2.1.2. Эксплуатационное назначение

Данное приложение может использоваться пользователями, которые хотят изучить процесс отрисовки 3D объектов на экране компьютера.

2.2. Краткая характеристика области применения

«3D Renderer» - интерактивный образовательный проект. Его цель – научиться имплементировать последовательность отрисовки 3D объектов.

Интерактивность Приложения заключается в том, что есть возможность не только наблюдать за статической картинкой, на которой изображены 3D объекты, но и перемещать камеру, через которую просматриваются объекты, в пространстве.

3. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ

3.1. Постановка задачи на разработку программы

1. Демонстрационное приложение

   При запуске приложения должно появиться два окна:

   1. Окно 1 – окно для наблюдения за состоянием 3D сцены
   2. Окно 2 – окно для наблюдения за сообщениями приложения, содержащие информацию об ошибках или времени, затраченном на отрисовку сцены в окне 1, диалог с пользователем.

   Демонстрационное приложение может находиться в двух состояниях:

   a. Состояние “a” – происходит отрисовка 3D сцены после каждого действия пользователя. В этом состоянии приложение пользователь управляет камерой. В окне 2 появляется информация о времени, затраченном на отрисовку 3D сцены. Сцена отрисовывается только после изменения положения камеры или добавления на неё нового объекта.
   b. Состояние “b” – происходит добавление нового объекта на сцену с помощью ввода в окно 2 запрашиваемой информации в формате, указанном в п. 4.1.2 “Технического задания”.

   Демонстрационное приложение в состоянии “a” должно давать пользователю возможность:

   a. Наблюдать за текущим состоянием 3D сцены в окне 1.
   b. Изменять положение камеры на 3D сцене в системе координат камеры посредством нажатия клавиш WASD.
   c. Изменять наклоны камеры на 3D сцене в системе координат камеры посредством нажатия клавиш: “стрелка вверх”, “стрелка вниз”, “стрелка влево”, “стрелка вправо”.
   d. Переходить в состояние “b” посредством нажатия клавиши P.

   Демонстрационное приложение в состоянии “b” должно предоставлять возможность:

   a. Вводить путь к файлу, содержащий информацию о новом объекте, при запросе пути.
   b. Вводить цвет нового объекта при запросе цвета объекта.
   c. Вводить координаты нового объекта при запросе координат объекта.

d. Переходить в состояние “a” при вводе команды “exit” вместо ввода параметров запрашиваемых данных на любом этапе ввода информации об объекте.
e. Переходить в состояние “a” при успешном вводе всей запрашиваемой информации о новом объекте.
f. Сообщать в окне 2 о виде запрашиваемой информации.
g. Сообщать в окне 2 о несоответствии формата вводимых данных ожидаемому формату, указанному в п. 4.1.2 “Технического задания”.

2. Библиотека отрисовки

Библиотека отрисовки должна содержать алгоритмы отрисовки 3D объектов на 3D сцене. Библиотека должна поддерживать отрисовку объектов, которые представляются в виде набора треугольников. Библиотека должна поддерживать обработку фонового и направленного света и его влияние на итоговый цвет объектов, находящихся на сцене.

В библиотеке должны находиться следующие классы:

1. World, который содержит в себе информацию о глобальной системе координат, находящихся на ней объектах, источниках света и камеры.
2. Renderer, который отрисовывает на пиксельном экране состояние объекта World, которое видит камера.
3. Camera, через которую пользователь может видеть текущее состояние сцены.
4. TriangulatedObject, который описывает в пространстве 3D объект, представимый в виде набора треугольников.
5. Triangle, который задаёт треугольник в пространстве.
6. Color, который описывает цвет.
7. AmbientLight, который содержит информацию о фоновом свете: его цвете.
8. DirectionalLight, который содержит информацию о направленном свете: его направлении и его цвете.
9. PixelScreen, который задаёт пиксельный экран, содержащий отрисованные на нём 3D объекты - результат работы класса Renderer.

3.2. Описание применяемых математических методов

Для обработки процесса отрисовки 3D объектов на 2D экране применяются методы линейной алгебры. Используемые картинки и методы описаны в книге “3-D Computer Graphics, A Mathematical Introduction with OpenGL”.

1. Хранение точек, задающих объект.

   Для удобства работы с точками в пространстве точка задаётся четырёхмерным вектором, где первая, вторя и третья координаты – координаты по осям ox, oy, oz соответственно, а четвёртая координата равна 1.

$$P_{vertex} = \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix}$$

2. Хранение направлений

   Для удобства работы с векторами-направлениями в пространстве вектор-направление задаётся четырёхмерным вектором, где первая, вторя и третья координаты – координаты по осям ox, oy, oz соответственно, а четвёртая координата равна 0.

$$V_{direction} = \begin{pmatrix} V_x \\ V_y \\ V_z \\ 0 \end{pmatrix}$$

3. Хранение позиций объектов в глобальной системе координат

   Каждый объект хранит в себе треугольники, которые находятся в локальной системе координат объекта, для того чтобы перевести треугольники в глобальную систему координат необходимо сместить все точки на позицию объекта.

   Позиция объекта не является точкой, которая задаёт грани объекта. Она является вектором-направлением, на который смещаются все точки объекта. Позиция объекта задаётся аналогично вектору направлению:

$$P_{object} = \begin{pmatrix} P_x \\ P_y \\ P_z \\ 0 \end{pmatrix}$$

4. Хранение цвета объекта. Арифметика цветов.

   Цвет задаётся в виде трёхмерного вектора, каждая координата которого – красная, зелёная или синяя составляющая.

$$\mathrm{C} = (\mathrm{C}_r \quad C_g \quad C_b)$$

   Есть два варианта задания цвета:

a. Каждая координата – целое число от 0 до 255
b. Каждая координата – вещественное число от 0 до 1

Используются координаты от 0 до 1. Преобразовать одни координаты в другие можно посредством умножения или деления на 255.

Цвета можно складывать:

$$C + D = (\mathrm{C}_r + D_r \quad C_g + D_g \quad C_b + D_b)$$

Цвета можно умножать друг на друга:

$$C \cdot D = (\mathrm{C}_r \cdot D_r \quad C_g \cdot D_g \quad C_b \cdot D_b)$$

Цвет можно умножать на скаляр:

$$\alpha \cdot \mathrm{C} = (\alpha \cdot \mathrm{C}_r \quad \alpha \cdot C_g \quad \alpha \cdot C_b)$$

Более подробно арифметика цветов описана в главе 7 части 1 книги “3-D Computer Graphics, A Mathematical Introduction with OpenGL”.

5. Система координат камеры.

Система координат камеры устроена таким образом, что ось oz направленна против направления камеры, ось ox направлена вправо относительно камеры, ось oy направленна вверх относительно камеры.

6. Обработка и хранение поворотов камеры в системе координат камеры.

Матрица, столбцами которой являются направления осей ox, oy, oz системы координаты камеры, вектора заданы в глобальной системе координат, называется матрицей поворота камеры.

$$R_{camera} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Добавление четвёртого столбца не будет влиять на дальнейшие вычисления.

При повороте камеры влево или право, то есть при повороте относительно оси oy, матрицу поворота камеры нужно умножить слева на следующую матрицу поворота:

$$R_{oy} = \begin{pmatrix} \cos(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Угол α>0, если поворот происходит направо, α<0, если поворот влево.

При повороте камеры вверх или вниз, то есть при повороте относительно оси ox, матрицу поворота камеры нужно умножить слева на следующую матрицу поворота:

$$R_{ox} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Угол α>0, если поворот происходит вниз, α<0, если поворот вверх.

7. Обработка перемещения камеры в системе координат камеры.

   При передвижении камеры прямо, то есть вдоль отрицательного направления oz в системе координат камеры, нужно изменить вектор перемещения в глобальных координатах таким образом, чтобы в системе координат камеры этот вектор соответствовал вектору отрицательного направления оси oz. Этот вектор находится следующим образом:

$$V_{movement} = R^{-1} \cdot \begin{pmatrix} 0 \\ 0 \\ -1 \\ 0 \end{pmatrix}$$

   В этом уравнении матрица R – матрица поворота камеры, $V_{movement}$ – вектор, на который нужно передвинуть камеру в глобальной системе координат.

   Важно заметить, что матрица поворота камеры всегда обратима, так как определитель исходной матрицы равен нулю и определители всех матриц поворота, которые умножаются на матрицу поворота камеры, равны 1.

8. Понятие области видимости камеры.

   Область видимости камеры выглядит следующим образом:

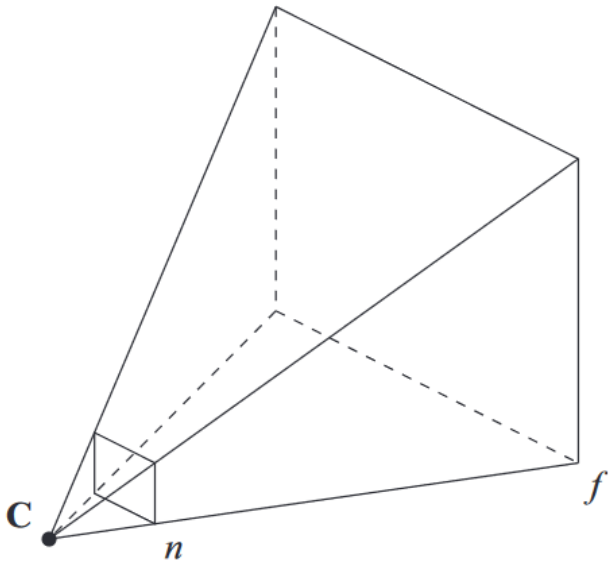

Figure 1 Section 5.3 View Frustrum

Точка C – позиция камеры. Область видимости камеры выглядит как усечённая пирамида, далее пирамида зрения. Всё что находится внутри этой пирамиды зрения видно камере. Отрезанная сверху пирамида имеет высоту n, общая высота пирамиды – f. Число n называют near plane distance или расстоянием до ближайшей плоскости, оно задаёт расстояние, начиная с которого объекты видны камере. Число f называют far plane distance или расстоянием до дальней плоскости. Всё, что находится за дальней плоскостью не видно камере.

Координаты точек, где боковые плоскости пересекают ближайшую плоскость, ищутся следующим образом:

Горизонтальный угол обзора камеры обозначается как α.

Необходимо построить плоскость, перпендикулярную направлению камеры таким образом, что в системе координат камеры она пересекает боковые плоскости на координатах -1 и 1 по оси x.

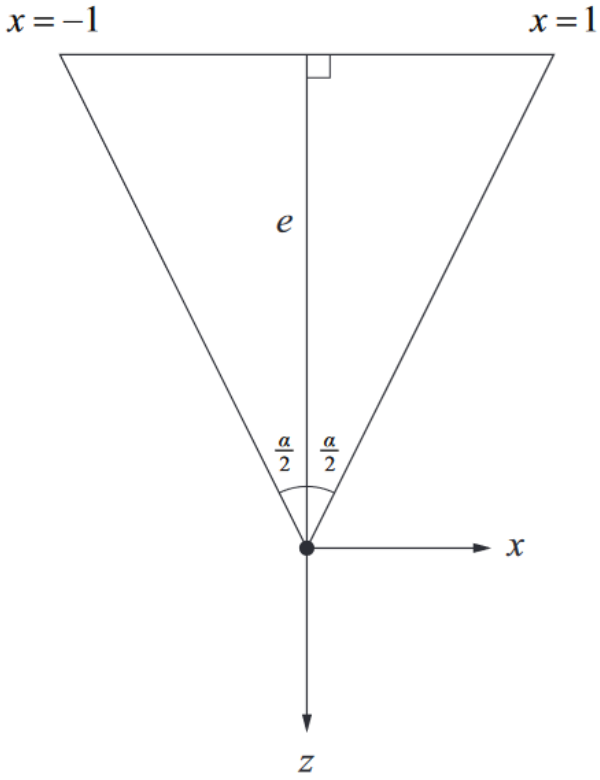

Figure 2 Section 5.3.1 Field of view

Расстояние e – фокальное расстояние камеры. Оно находится по следующей формуле:

$$e = \frac{1}{tan\left(\frac{\alpha}{2}\right)}$$

Вертикальный угол обзора камеры ищется следующим образом:

Найдём соотношение сторон экрана, необходимо поделить высоту окна на его длину, обозначим это число за a. Вертикальный угол обзора:

$$\beta = 2\,tan^{-1}\left(\frac{a}{e}\right)$$

Координаты у пересечения плоскости и пирамиды зрения ищутся следующим образом:

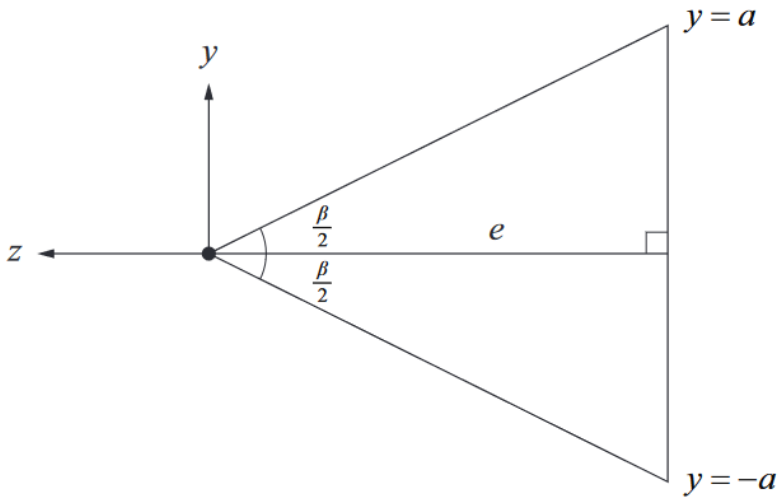

Figure 3 Section 5.3.1 Vertical field of view

Плоскость находится на расстоянии e. Необходимо умножить координаты пересечения по x и по y на $\frac{n}{e}$, чтобы плоскость находилась на расстоянии n. Левая грань пересекается на координате x равной $-\frac{n}{e}$, правая грань: $\frac{n}{e}$. Верхняя грань пересекается по координате y равной $\frac{an}{e}$, нижняя: $-\frac{an}{e}$. Эти числа l, r, t, b соответственно. Они понадобятся в пункте 10 “описания применяемых математических методов”.

9. Преобразование точек и нормалей объектов из глобальной системы координат в систему координат камеры.

   Матрица перехода из глобальной системы координат в систему координат камеры следующим образом:

$$A = \begin{pmatrix} R_{0,0} & R_{0,1} & R_{0,2} & 0 \\ R_{1,0} & R_{1,1} & R_{1,2} & 0 \\ R_{2,0} & R_{2,1} & R_{2,2} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

   R – матрица поворота камеры.

   К последнему столбцу необходимо прибавить вектор (V) положения камеры в пространстве умноженный на матрицу поворота камеры.

   Получаем матрицу:

$$A = \begin{pmatrix} R_{0,0} & R_{0,1} & R_{0,2} & RV_0 \\ R_{1,0} & R_{1,1} & R_{1,2} & RV_1 \\ R_{2,0} & R_{2,1} & R_{2,2} & RV_2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Пусть V_{norm} – это вектор нормали некоторой плоскости, а P – точка некоторой плоскости, заданная в глобальной системе координат.

$$AV_{norm} = \begin{pmatrix} V'_0 \\ V'_1 \\ V'_2 \\ 0 \end{pmatrix}$$

При умножении вектора нормали на матрицу перехода мы получаем такой же вектор нормали, но уже в системе координат камеры.

$$AP = \begin{pmatrix} P'_x \\ P'_y \\ P'_z \\ 1 \end{pmatrix}$$

При умножении точки на матрицу перехода мы получаем точку в системе координат камеры. Последний столбец матрицы перехода позволяет корректно переводить точки в глобальной системе координат в систему координат камеры.

10. Преобразование точек и нормалей объектов из системы координат камеры в систему координат, где каждый видимая точка имеет координату по каждой оси в интервале (-1;1) (Normalized device coordinates).

    Более подробное описание преобразования в Normalized device coordinates описано в главе 5 части 5 книги “3-D Computer Graphics, A Mathematical Introduction with OpenGL”.

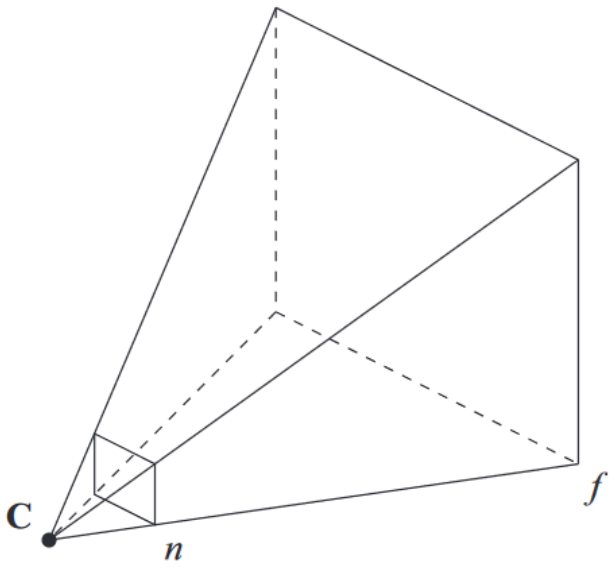

Figure 3 Section 5.3 View Frustrum

Пирамида зрения выглядит так, как изображено на Figure 3. Проецирование объектов на ближайшую плоскость в таком случае является трудоёмкой задачей. Для того чтобы облегчить эту задачу стоит превратить пирамиду зрения в куб зрения.

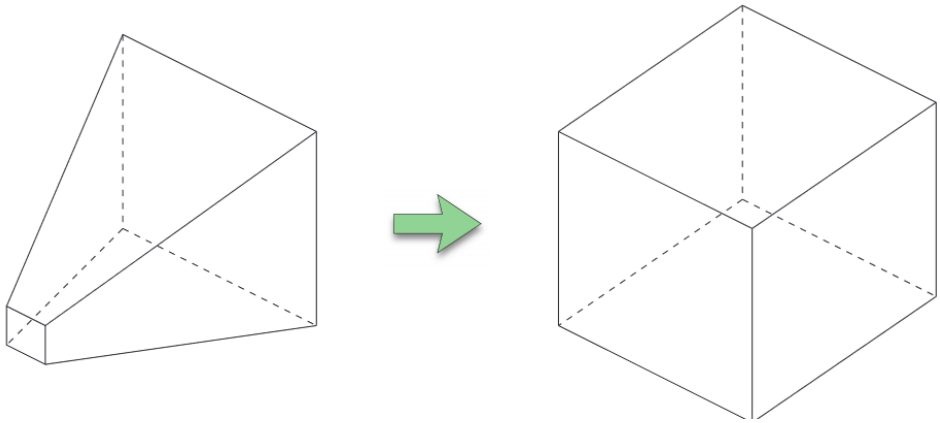

Figure 4 Section 5.5.1 Perspective projections

В кубе зрения все видимые точки имеют координаты от -1 до 1 по всем осям.

Для такого преобразования понадобится следующая матрица:

$$M_{frustum} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2nf}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Пусть $P = (P_x \quad P_y \quad P_z \quad 1)$. Тогда:

$$M_{frustum}P = \begin{pmatrix} P'_x \\ P'_y \\ P'_z \\ -P_z \end{pmatrix}$$

В четвёртой координате получено отрицательное значение координаты z у исходной точки. Необходимо поделить полученный вектор на последнюю координату, чтобы получить ожидаемый результат.

$$P'' = \begin{pmatrix} \frac{P'_x}{-P_z} \\ \frac{P'_y}{-P_z} \\ \frac{P'_z}{-P_z} \\ 1 \end{pmatrix}$$

Если исходная точка P находилась внутри пирамиды зрения камеры, то точка P'' будет находиться внутри куба зрения.

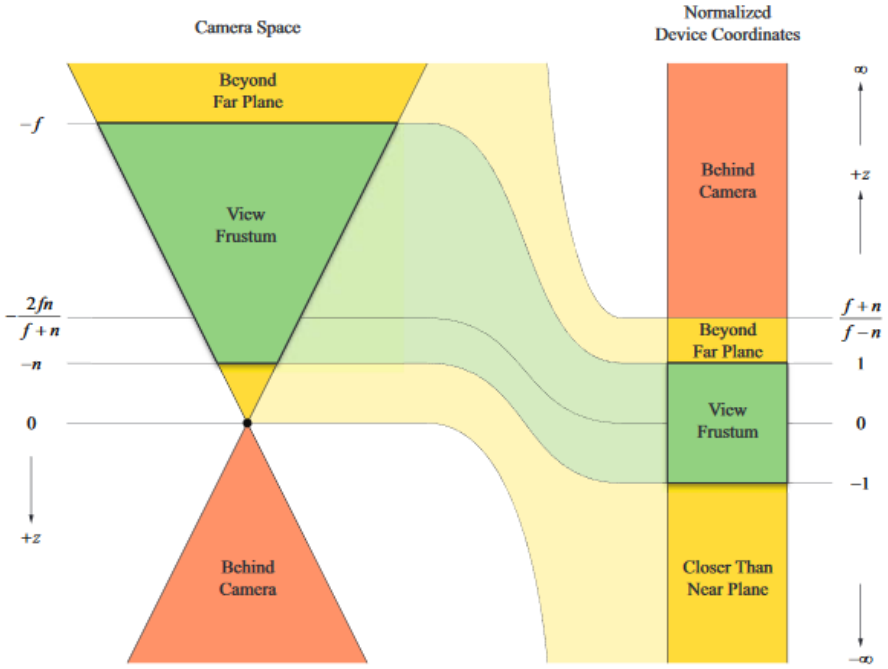

Figure 5 Section 5.5.1 Normalized device coordinates

На Figure 5 изображено каким образом преобразуются координаты исходных точек в системе координат камеры в новую систему координат с кубом зрения.

В некоторых случаях может появиться необходимость в том, чтобы камера видела бесконечно далеко, тогда необходимо f устремить к бесконечности:

$$M_{inffrustum} = \lim_{f \to \infty} M_{frustum} \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -1 & -2n \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Тогда при переходе к кубу зрения необходимо использовать матрицу $M_{inffrustum}$.

11. Определение видимости поверхности камерой.

Пусть дан вектор нормали плоскости V и вектор направления камеры V_{cam}.

Нормализуем эти два вектора. Найдём скалярное произведение этих двух векторов. Если результат меньше нуля, то плоскость гарантированно видна. Если результат близок к единице, то поверхность не видна.

12. Пересечение луча и треугольника.

Более подробное описание нахождения пересечения луча и треугольника описано в главе 6 части 2 пункте 1 книги “3-D Computer Graphics, A Mathematical Introduction with OpenGL”.

Пусть дан луч с начальной точкой S и вектором направления V, точки P_0, P_1, P_2, задающие треугольник. Найдём вектор нормали для треугольника:

$$N = (P_1 - P_0) \times (P_2 - P_0)$$

Положим $L = (N_0 \quad N_1 \quad N_2 \quad -N \cdot P_0)$. Найдём число t, относящееся к пересечению с плоскостью, содержащий треугольник.

$$t = -\frac{L \cdot S}{L \cdot V}$$

Если $L \cdot V = 0$, то пересечений нет, иначе:

$P = S + tV$ – точка пересечения луча и плоскости.

Если существуют вещественные числа $w_0, w_1, w_2 > 0$, $w_0 + w_1 + w_2 = 1$, такие что

$$P = w_0 P_0 + w_1 P_1 + w_2 P_2$$

то точка P лежит внутри треугольника. (в последнем равенстве четвёртой координатой в точках следует пренебречь).

Введём дополнительные обозначения:

$$R = P - P_0$$
$$Q_1 = P_1 - P_0$$
$$Q_2 = P_2 - P_0$$

Чтобы найти коэффициенты, необходимо решить матричное уравнение:

$$\begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} Q_1^2 & Q_1 \cdot Q_2 \\ Q_1 \cdot Q_2 & Q_2^2 \end{pmatrix}^{-1} \cdot \begin{pmatrix} R \cdot Q_1 \\ R \cdot Q_2 \end{pmatrix}$$

Третий коэффициент можно найти следующим образом:

$$w_0 = 1 - w_1 - w_2$$

13. Определение цвета плоскости при наличии фонового цвета

Пусть дан цвет $C_{ambient}$ – цвет фонового света и C – цвет плоскости, тогда цвет

$C_{ambient} \cdot C$ является итоговым цветом плоскости

14. Определение цвета плоскости при наличии направленного света

    Пусть дан цвет $C_{directonal}$ – цвет направленного света, C – цвет плоскости, $V_{directional}$ – направление направленного света, V_{norm} – вектор нормали плоскости. Если результат скалярного произведения вектора нормали и вектора направления света больше нуля, то плоскость не освещается направленным светом, иначе итоговый цвет можно найти следующим образом:

$$-(V_{directional} \cdot V_{norm}) \cdot (C_{directional} \cdot C)$$

15. Определение цвета плоскости при наличии фонового и направленного света.

    Итоговый цвет плоскости является суммой цвета, найденного в пункте 13, и всех цветов найденных в пункте 14, то есть:

$$C_{res} = C_{ambient} \cdot C - \sum_i (V_i \cdot V_{norm}) \cdot (C_i \cdot C)$$

3.3. Описание архитектуры программы

Приложение состоит из двух компонент – библиотеки отрисовки и демонстрационного приложения, использующего библиотеку.

Более подробно о каждом классе можно прочитать в приложениях 2–5.

1. Демонстрационное приложение

   Класс App отвечает за инициализацию всех необходимых программе ресурсов, запуск run-time loop, в котором считываются все действия пользователя. содержит в себе run-time loop, который считывает, какие клавиши нажал пользователь, а также направление обработки событий другим классам.

   Класс ObjectParser отвечает за парсинг нового объекта из файла формата .obj, цвета объекта и его позиции. Класс участвует в обработке события добавления объекта на сцену.

   Класс Kernel отвечает за инициализацию 3D сцены, камеры и за контроль за наполнением сцены и положением камеры. Класс участвует в обработке событий передвижения и поворота камеры, добавления объекта на сцену.

   Класс Logger отвечает за вывод общей информации и информации об ошибках в окно 2.

   Класс Timer отвечает за замер времени. Класс является таймером.

   Класс View отвечает за отрисовку пиксельного экрана в окне 1.

2. Библиотека отрисовки

   Класс AmbientLight хранит информацию о фоновом свете.

   Класс DirectionalLight хранит информацию о направленном свете.

   Класс Color хранит информацию о цвете и реализует арифметику цветов.

   Класс Triangle является хранилищем данных для треугольника в пространстве.

   Класс TriangulatedObject является хранилищем данных для триангулируемого объекта.

   Класс Camera является хранилищем данных о камере, а также реализует операции над камерой: её поворот и её передвижение.

   Класс World является хранилищем данных о 3D сцене.

   Класс PixelScreen является хранилищем данных о пиксельном экране.

   Класс Renderer является логическим обработчиком отрисовки 3D объектов на сцене, которую видит камера.

3.4.Описание алгоритма работы программы

1. Демонстрационное приложение

   Класс App запускает run-time loop, в котором считываются действия пользователя. События связанные с перемещением камеры направляются классу Kernel. Событие связанное сс добавлением нового объекта обрабатываются классами App, ObjectParser и Kernel.

2. Библиотека классов

   Поворот камеры осуществляется за счёт умножения матрицы поворота камеры на матрицу поворота. Передвижение камеры осуществляется за счёт прибавления к текущей позиции камеры вектора перемещения.

   Процесс отрисовки состоит следующих этапов:

   a. Нахождение всех необходимых чисел для отрисовки.
   b. Подсчёт матрицы перехода в Normalized device coordinates.
   c. Считывание информации обо всех объектах на сцене, направленных и фоновых источниках света.
   d. Перевод координат каждого объекта из глобальных координат в систему координат камеры, а потом в Normalized device coordinates.
   e. Перевод векторов нормали каждого объекта в систему координат камеры.
   f. Определение видимости каждой поверхности каждого объекта.

g. Нахождение точек пересечения лучей, выпущенных из каждого пикселя экрана, и поверхностей объекта. Определение цвета каждого пикселя на основе данных о пересечениях лучей и поверхностей, фоновом свете и направленных источниках света.

3.5. Обоснование выбора схемы алгоритма

Для написания приложения и библиотеки был выбран ЯП C++ по причине своей скорости и количеству свободы, который он предоставляет программисту, а также требования научного руководителя проекта.

Описанная в пункте 3.3 схема алгоритма выбрана с желанием сделать библиотеку отрисовки максимально независимой от того кода, который будет её использовать.

1. Демонстрационное приложение

   Приложение разбито на множество классов, так как высокоуровневые классы, например, App, не должны обрабатывать обращение с необходимыми параметрами напрямую к классам библиотеки отрисовки. Было принято решение ввести класс Kernel, который будет обрабатывать логику работы с классами библиотеки, будет являться посредником между App, который считывает действия пользователя, и классами библиотеки отрисовки, которые требуют параметризованных вызовов функций.
2. Библиотека классов

   Библиотека содержит инкапсулированные контейнеры данных: свет, объекты, треугольники, задающие объект, камеру. Также есть класс Render, который в полном объёме содержит логику отрисовки 3D объектов. Максимальное деление логических единиц отрисовки объекта сделано для максимального распределения логики изменения и хранения данных между объектами. Намного удобнее создавать новый 3D объект, ограничившись списком треугольников, нежели списком точек, векторов нормалей и связей между ними.

3.6. Описание выбора метода организации входных и выходных данных

Кнопки WASD выбраны для перемещения камеры из-за широкого распространения такого вида управления в видеоиграх.

Стрелки выбраны для изменения угла наклона камеры, так как они лучше всего подходят по своему назначению к изменению угла наклона камеры.

Формат .obj выбран для ввода нового объекта в связи со своей распространённостью.

Формат RGB для ввода цвета выбран по причине своей распространённости и лёгкого понимания формата.

Формат ввода координат выбран по причине использования общераспространённых математических методов.

Формат информации об ошибках выбран таким образом, чтобы пользователю потребовалось минимальное количество времени на идентификацию сути проблемы и её решение.

Формат информационных сообщений выбран таким образом, чтобы пользователь получал обратную связь от приложения после своих действий.

3.7. Описание и обоснование выбора состава технических и программных средств

3.7.1. Состав технических и программных средств

Для работы окнами и их событиями была выбрана библиотека SFML по причине своей простоты и масштаба приложения. Требуется реализовать лишь два окна – для этой задачи лучше всего подходит библиотека SFML.

Для реализации библиотеки была использована библиотека Eigen со всеми необходимыми методами линейной алгебры. Эта библиотека предоставляет быстрый подсчёт результатов перемножения матриц на процессоре.

Для контроля за кодстайлом была выбрана утилита clang-format.

Для контроля за качеством кода была выбрана утилита clang-tidy.

Для сборки проекта была выбрана система сборки CMake.

Для контроля версий проекта используются git и github.com.

Для нормального функционирования программы требуется персональный компьютер, оснащенный следующими техническими компонентами:

1. Для Windows:

   Дисплей: Минимальное разрешение 1280 x 720 пикселей.

   Операционная система: Windows 10 или более новые версии.

   Оперативная память: 4 ГБ или более.

   Хранилище: не менее 100 МБ свободного места для хранения.

2. Для Linux:

   Дисплей: Минимальное разрешение 1280 x 720 пикселей.

   Дистрибутив: Ubuntu 20.04 или более новые версии.

   Оперативная память: 4 ГБ или более.

   Хранилище: не менее 100 МБ свободного места для хранения.

3.7.2. Обоснование выбора технических и программных средств

ОС Windows и Linux являются очень популярными среди обычных и продвинутых пользователей компьютеров. На них более вероятно пользователь, желающий изучить процесс 3D рендеринга, будет заниматься изучением.

Утилиты, системы контроля версий, системы сборки выбраны по причине своей распространённости.

4. ОЖИДАЕМЫЕ ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

4.1. Ориентировочная экономическая эффективность

В рамках данного задания экономическая эффективность не предусмотрена.

4.2. Предполагаемая потребность

Данный программный продукт будет интересен людям, которые хотят изучить базовый процесс отрисовки 3D объектов на экране компьютера, состоящий из отображения 3D объекта на 2D экран и обработки направленного и фонового света.

4.3. Экономические преимущества разработки по сравнению с отечественными и зарубежными образцами или аналогами

В рамках данного задания экономические преимущества по сравнения с отечественными и зарубежными аналогами не предусмотрена.

ПРИЛОЖЕНИЕ 1

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. ГОСТ 19.101-77 Виды программ и программных документов. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
2. ГОСТ 19.102-77 Стадии разработки. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
3. ГОСТ 19.103-77 Обозначения программ и программных документов. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
4. ГОСТ 19.104-78 Основные надписи. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
5. ГОСТ 19.105-78 Общие требования к программным документам. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
6. ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
7. ГОСТ 19.201-78 Техническое задание. Требования к содержанию и оформлению. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
8. ГОСТ 19.603-78 Общие правила внесения изменений. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
9. ГОСТ 19.604-78 Правила внесения изменений в программные документы, выполненные печатным способом. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
10. 3-D Computer Graphics, A Mathematical Introduction with OpenGL – Samuel R. Buss, Cambridge 2003
11. Mathematics for 3D Game Programming and Computer Graphics Third Edition – Eric Lengyel, Course Technology PTR 2012
12. Компьютерная графика. Динамика, реалистические изображения – Е.В. Шикин, А. В. Боресков, Москва “Диалог-МИФИ” 1995
13. Компьютерная графика. Полигональные модели – Е.В. Шикин, А. В. Боресков, Москва “Диалог-МИФИ” 2001
14. Eigen. Eigen: A C++ template library for linear algebra. URL: https://eigen.tuxfamily.org/index.php?title=Main_Page
15. SFML. SFML: Simple and Fast Multimedia Library. URL: https://www.sfml-dev.org/index.php

16. Cppreference. Cppreference - complete online reference for the C and C++ languages and standard libraries. URL: https://en.cppreference.com/w/

ПРИЛОЖЕНИЕ 2

ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ КЛАССОВ ДЕМОНСТРАЦИОННОГО ПРИЛОЖЕНИЯ

Класс	Назначение
App	Класс, обрабатывающий действия пользователя в двух окнах
ObjectParser	Класс, считывающий из файла .obj триангулируемый объект, если это возможно
Kernel	Класс, отвечающий за изменения состояния камеры и наполнения сцены
Logger	Класс, который предоставляет возможность удобно писать в консоль или в файл сообщения с дополнительными данными, например, текущим временем или типом сообщения
Timer	Класс, который замеряет время
View	Класс, отвечающий за отрисовку данных в окне 1

ПРИЛОЖЕНИЕ 3

ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ ПОЛЕЙ И МЕТОДОВ ДЕМОНСТРАЦИОННОГО ПРИЛОЖЕНИЯ

Описание полей и методов класса App				
Поля				
Наименование	Модификаторы	Тип поля	Назначение	
kWidth	private static	const int	Стандартная ширина окна 1	
kHeight	private static	const int	Стандартная высота окна 1	
kAppName	private static constexpr	const char*	Стандартное название окна 1	
kernel_	private	Kernel	Ядро, которое управляет состоянием сцены	
window_	private	sf::RenderWindow	Окно 1	
view_	private	View	Отвечает за отрисовку состояния сцены, полученной из ядра	
Методы				
Наименование	Модификаторы	Тип аргумента	Тип возвращаемого значения	Назначение
Run	public	-	void	Основной цикл программы
HandleEvent_	private	const sf::Event& event	void	Обработка полученного действия пользователя
HandleKeyEvent_	private	const sf::Keyboard::Key& key	void	Обработка нажатия клавиши

AddNewObject_	private	-	void	Ввод информации о новом объекте и добавление его в ядро при корректном вводе
ShowNewFrame_	private	-	void	Обновление состояния окна 1

Описание полей и методов класса ObjectParser				
Поля отсутствуют				
Методы				
Наименование	Модификаторы	Тип аргумента	Тип возвращаемого значения	Назначение
ParseObject	public const	const std::string& path	TriangulatedObject	Считывает файл .obj, преобразует данные из файла в TriangulatedObject
ParseColor_	public const	const std::string& input	sf::Color	Считывает из строки input цвет в формате RGB
ParsePosition_	public const	const std::string& input	Eigen::Vector3d	Считывает из строки input координаты x, y, z

Описание полей и методов класса Kernel				
Поля				
Наименование	Модификаторы	Тип поля	Назначение	
kScreenWidth	private static	const int	Стандартная ширина буфера экрана	
kScreenHeight	private static	const int	Стандартная высота буфера экрана	
kMovementSpeed	private static constexpr	double	Стандартная скорость перемещения камеры	
kRotationSpeedDeg_	private static constexpr	double	Стандартный угол поворота камеры	
cam_	private	Camera	Камера, которой управляет ядро	
world_	private	World	Мир, состоянием которого управляет ядро	
screen_buffer_	private	PixelScreen	Буффер экрана	
renderer_	private	Renderer	Обрабатывает логику отрисовки объектов на сцене	
Методы				
Наименование	Модификаторы	Тип аргумента	Тип возвращаемого значения	Назначение
MoveCameraForward	public	-	void	Передвижение камеры вперёд
MoveCameraBackwads	public	-	void	Передвижение камеры назад

MoveCameraLeft	public	-	void	Передвижение камеры влево
MoveCameraRight	public	-	void	Передвижение камеры вправо
RotateCameraUp	public	-	void	Поворот камеры вверх
RotateCameraDown	public	-	void	Поворот камеры вниз
RotateCameraLeft	public	-	void	Поворот камеры влево
RotateCameraRight	public	-	void	Поворот камеры направо
AddObject	public	TriangulatedObject && obj	void	Добавляет объект obj на сцену (в world_)
SetAmbientLight	public	AmbientLight&& light	void	Меняет фоновый свет на сцене
AddDirectionalLight	public	DirectionalLight&& light	void	Добавляет направленный свет на сцену
MakeScene	public	-	const PixelScreen&	Возвращает текущее состояние сцены,

				которое видит камера

Описание полей и методов класса Logger				
Поля				
Наименование	Модификаторы	Тип поля	Назначение	
kEmpty	private static constexpr	const char*	Пустой тип сообщения	
kError	private static constexpr	const char*	Тип сообщения, который говорит об ошибке	
kInfo	private static constexpr	const char*	Тип сообщения, который несёт осведомительный характер	
kConsole	public static	const Logger	Логгер, который пишет в окно 2 без временной метки	
kConsoleTimeSpan	public static	const Logger	Логгер, который пишет в окно 2 с временной меткой	
kLoggerFile	public static	const Logger	Логгер, который пишет в файл	
write_to_file_	private	bool	Флаг, который показывает, пишет ли логгер в файл	
use_time_stamp_	private	bool	Флаг, который показывает, необходимо ли писать временную метку	
path_	private	std::string	Путь к файлу. Если логгер пишет в окно 2, то path_ - пустая строка	
Методы				
Наименование	Модификаторы	Тип аргумента	Тип возвращаемого значения	Назначение
Log	public const	const char* message	void	Логгирует message без типа сообщения
Log	public template <class T> const	const T& message	void	Логгирует message без

Изм	Лист	№ докум.	Подп.	Дата
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

				типа сообщения
Error	public const	const char* message	void	Логгирует message с сообщением об ошибке
Error	public template <class T> const	const T& message	void	Логгирует message с сообщением об ошибке
Info	public const	const char* message	void	Логгирует message с сообщением INFO
Info	public template <class T> const	const T& message	void	Логгирует message с сообщением INFO
LogWithType_	private const	const char* message , const std::string& type	void	Пишет в файл или окно 2 сообщение message с типом type и временной меткой, если use_time_stan_ - true
LogWithType_	private template <class T> const	const T& message, const std::string& type	void	Пишет в файл или окно 2 сообщение message с типом type и

				временной меткой, если use_time_stan_ - true
GetCurrentTimeLog_	private const static	-	std::string	Возвращает текущую временную метку
GetCurrentDay_	private const static	-	std::string	Возвращает текущий день

Описание полей и методов класса Timer				
Поля				
Наименование	Модификаторы	Тип поля	Назначение	
start_	private	std::chrono::_V2::system_clock::time_point	Время, в которое начался отсчёт	
Методы				
Наименование	Модификаторы	Тип аргумента	Тип возвращаемого значения	Назначение
GetMilliseconds	Public const	-	int64_t	Возвращает количество миллисекунд, которое прошло с момента start_ до текущего момента
Reset	public	-	void	Обновляет время отсчёта

Описание полей и методов класса View				
Поля				
Наименование	Модификаторы	Тип поля	Назначение	
window_	private	sf::RenderWindow*	Указатель на окно 1	
Методы				
Наименование	Модификаторы	Тип аргумента	Тип возвращаемого значения	Назначение
Draw	public	const PixelScreen& ps	void	Рисует в окне 1 состояние пиксельного экрана

ПРИЛОЖЕНИЕ 4

ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ КЛАССОВ БИБЛИОТЕКИ

Класс	Назначение
AmbientLight	Класс, который содержит информацию о фоновом свете
DirectionalLight	Класс, который содержит информацию о направленном свете
Color	Класс, который реализует арифметику цветов и содержит информацию о цвете
Triagle	Класс, который представляет собой треугольник
TriangulatedObject	Класс, который является объектом, представленным в виде множества треугольников
Camera	Класс, который инкапсулирует действия с камерой на сцене и данные камеры
World	Класс, который содержит в себе информацию о сцене
PixelScreen	Класс, который представляет собой пиксельный экран
Renderer	Класс, который содержит логику отрисовки 3D объектов на пиксельном экране

ПРИЛОЖЕНИЕ 5

ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ ПОЛЕЙ И МЕТОДОВ БИБЛИОТЕКИ

Описание полей и методов класса AmbientLight				
Поля				
Наименование	Модификаторы	Тип поля	Назначение	
color_	private	Color	Цвет фонового света	
Методы				
Наименование	Модификаторы	Тип аргумента	Тип возвращаемого значения	Назначение
GetColor	Public const	-	const Color&	Геттер для цвета

Описание полей и методов класса Color				
Поля				
Наименование	Модификаторы	Тип поля	Назначение	
color_vector_	private	Eigen::Vector3d	Вектор цвета, хранящий цвет в формате RGB. Каждая координата от 0 до 1	
Методы				
Наименование	Модификаторы	Тип аргумента	Тип возвращаемого значения	Назначение
GetColorVector	Public const	-	const Vector3&	Геттер для вектора цвета
ConvertToHexColor	Public const	-	sf::Color	Преобразует текущий цвет из формата, где каждая компонента от 0 до 1, в формат, где каждая компонента от 0 до 255
operator+=	public	const Color& other	Color&	Оператор сложения цвета с другим цветом
operator*=	public	const Color& other	Color&	Оператор умножения

				цвета на другой цвет
operator*=	public	double coef	Color&	Оператор умножения цвета на скаляр
operator+	friend	const Color& lhv, const Color& rhv	Color	Оператор сложения двух цветов
operator*	friend	const Color& lhv, const Color& rhv	Color	Оператор умножения двух цветов
operator*	friend	double coef, const Color& color	Color	Оператор умножения цвета на скаляр
Normalize_	private	-	void	Сохраняет корректное состояние вектора-цвета

Описание полей и методов класса DirectionalLight				
Поля				
Наименование	Модификаторы	Тип поля	Назначение	
color_	private	Color	Цвет света	
direction_	private	Eigen::Vector4d	Направление света	
Методы				
Наименование	Модификаторы	Тип аргумента	Тип возвращаемого значения	Назначение
GetColor	Public const	-	const Color&	Геттер для цвета
GetDirection	Public const	-	const Eigen::Vector4d &	Геттер для направления

Описание полей и методов класса Triangle				
Поля				
Наименование	Модификаторы	Тип поля	Назначение	
point0_	private	Eigen::Vector4d	Первая точка треугольника	
point1_	private	Eigen::Vector4d	Вторая точка треугольника	
point2_	private	Eigen::Vector4d	Третья точка треугольника	
normal_	private	Eigen::Vector4d	Вектор нормали треугольника	
Методы				
Наименование	Модификаторы	Тип аргумента	Тип возвращаемого значения	Назначение
GetPoint	Public const	Int index	const Eigen::Vector4d &	Геттер для точек. Возвращает точку, соответствующую индексу
GetNormalVector	Public const	-	const Eigen::Vector4d &	Геттер для вектора нормали
MakePositionMatrix	Public const	-	Eigen::Matrix<double, 4, 3>	Возвращает матрицу 4x3, содержащую координаты треугольника

Описание полей и методов класса TriangulatedObject				
Поля				
Наименование	Модификаторы	Тип поля	Назначение	
position_	private	Eigen::Vector4d	Позиция объекта в глобальной системе координат	
color_	private	Color	Цвет объекта	
surfaces_	private	std::vector<Triangle>	Список всех треугольников, задающих объект	
Методы				
Наименование	Модификаторы	Тип аргумента	Тип возвращаемого значения	Назначение
SetPosition	Public	const Eigen::Vector3d & pos	void	Сеттер для позиции
SetColor	Public	sf::Color color	void	Сеттер для цвета
GetPosition	Public const	-	Eigen::Vector4d	Геттер для позиции
GetColor	Public const	-	Color	Геттер для цвета
GetSurfaces	Public const	-	const std::vector<Triangle>&	Геттер для списка треугольников
MakeVertexesLocal	Public const	-	Eigen::Matrix4Xd	Возвращает матрицу, где каждый столбец – точка одного из треугольника

				в локальной системе координат
MakeVertexesGlobal	Public const	-	Eigen::Matrix4Xd	Возвращает матрицу, где каждый столбец – точка одного из треугольника в глобальной системе координат
MakeNormalVectorMatrix	Public const	-	Eigen::Matrix3Xd	Возвращает матрицу с векторами нормалей каждого треугольника
MakeNormalVectors	Public const	-	std::vector< Eigen::Vector4d >	Возвращает список с векторами нормалей каждого треугольника

Описание полей и методов класса Camera				
Поля				
Наименование	Модификаторы	Тип поля	Назначение	
kNearPlaneDistance	private	Const double	Стандартное расстояние до ближайшей плоскости камеры	
kHorizontalFieldOfViewAngleRad	private	Const double	Стандартный горизонтальный угол обзора камеры	
position_	private	Eigen::Vector4d	Позиция камеры	
directionMatrix_	private	Eigen::Matrix4d	Матрица поворота камеры	
near_plane_distance_	private	double	Расстояние до ближайшей плоскости	
horizontal_field_of_view_angle_rad_	private	double	Горизонтальный угол обзора	
Методы				
Наименование	Модификаторы	Тип аргумента	Тип возвращаемого значения	Назначение
MoveForward	Public	double distance	void	Перемещает камеру вперёд на расстояние distance
MoveBackwards	Public	double distance	void	Перемещает камеру назад на расстояние distance
MoveLeft	Public	double distance	void	Перемещает камеру влево

				на расстояние distance
MoveRight	Public	double distance	void	Перемещает камеру вправо на расстояние distance
RotateUpRad	Public	double angle	void	Поворачивает камеру вверх на угол, заданный в радианах
RotateUpDeg	Public	double angle	void	Поворачивает камеру вверх на угол, заданный в градусах
RotateDownRad	Public	double angle	Void	Поворачивает камеру вниз на угол, заданный в радианах
RotateDownDeg	Public	double angle	Void	Поворачивает камеру вниз на угол, заданный в градусах
RotateLeftRad	Public	double angle	Void	Поворачивает камеру влево на угол, заданный в радианах
RotateLeftDeg	Public	double angle	Void	Поворачивает камеру влево на угол,

				заданный в градусах
RotateRightRad	Public	double angle	Void	Поворачивает камеру вправо на угол, заданный в радианах
RotateRightDeg	Public	double angle	Void	Поворачивает камеру вправо на угол, заданный в градусах
GetNearPlaneDistance	Public const	-	double	Геттер для расстояния до ближайшей плоскости камеры
GetHorizontalFieldOfViewRad	Public const	-	double	Геттер для угла обзора камеры
GetPosition	Public const	-	const Eigen::Vector4d&	Геттер для позиции камеры
GetForwardDirectionOfCamera	Public const	-	Eigen::Vector4d	Геттер для вектора направления “прямо” относительно камеры
GetRightDirectionOfCamera	Public const	-	Eigen::Vector4d	Геттер для вектора направления

				“вправо” относительно камеры
GetDirectionMatrix	Public const	-	Eigen::Matrix 4d	Геттер для матрицы поворота камеры
GetTransformToCameraSpaceMatrix	Public const	-	Eigen::Matrix 4d	Геттер для матрицы перехода в систему координат камеры

Описание полей и методов класса World				
Поля				
Наименование	Модификаторы	Тип поля	Назначение	
objects_	private	std::vector<TriangulatedObject>	Список всех объектов, находящихся на сцене	
directional_light_sources_	private	std::vector<DirectionalLight>	Список всех направленных источников света	
ambient_light_	private	AmbientLight	Фоновый свет	
Методы				
Наименование	Модификаторы	Тип аргумента	Тип возвращаемого значения	Назначение
SetAmbientLight	Public	AmbientLight light	void	Сеттер для фонового света
AddObject	Public	TriangulatedObject && obj	void	Добавляет obj в список объектов сцены
AddDirectionalLight	Public	DirectionalLight light	void	Добавляет направленный свет в список направленных источников света

GetObjects	Public const	-	const std::vector< TriangulatedObject >&	Геттер для списка объектов
GetDirectionalLight	Public const	-	const std::vector<DirectionalLight>&	Геттер для списка направленных источников света
GetAmbientLight	Public const	-	const AmbientLight&	Геттер для фонового света

Описание полей и методов класса PixelScreen				
Поля				
Наименование	Модификаторы	Тип поля	Назначение	
kWidth	private static	int	Стандартная ширина экрана	
kHeight	Private static	int	Стандартная высота экрана	
width_	private	int	Ширина экрана	
height_	private	int	Высота экрана	
screen_	private	sf::VertexArray	Список всех пикселей на экране	
Методы				
Наименование	Модификаторы	Тип аргумента	Тип возвращаемого значения	Назначение
Pixel	Public	int row, int column	sf::Vertex&	Геттер для пикселя, находящегося на координатах (row; column)
Pixel	Public const	int row, int column	const sf::Vertex&	Константный геттер для пикселя, находящегося на координатах (row; column)
GetWidth	Public const	-	int	Геттер для ширины экрана
GetHeight	Public const	-	int	Геттер для высоты экрана

GetPixels	Public const	-	const PixelScreen::Pixels&	Геттер для списка всех пикселей

Описание полей и методов класса Renderer				
Поля				
Наименование	Модификаторы	Тип поля	Назначение	
z_buffer_	private	Eigen::MatrixXd	Буффер, хранящий z координату ближайщей точки пересечения луча, направленного из пикселя, и некоторого треугольника	
Методы				
Наименование	Модификаторы	Тип аргумента	Тип возвращаемого значения	Назначение
Render	Public	const World& w, const Camera& c, PixelScreen& buffer	void	Отрисовывает на пиксельном экране buffer состояние сцены, содержащейся в w, которую видит камера c
IsSurfaceVisible_	Public const	const TriangulatedObject::Matrix4xN& coordinates_of_object, Eigen::Vector4d normal, Eigen::Vector4d camera_direction, int surface_index	bool	Определяет, видна ли плоскость с индексом surface_index камеры по направлени

				ю камеры, вектору нормали поверхности и точкам этой поверхности

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

Лист регистрации изменений

Номера листов (страниц)					Всего листов (страниц в докум.)	№ документа	Входящий № сопроводительного докум. и дата	Подп.	Дата
Изм.	Измененных	Замененных	Новых	Аннулированных					

Изм	Лист	№ докум.	Подп.	Дата
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата

Изм	Лист	№ докум.	Подп.	Дата
Инв. № подл.	Подп. и дата	Взам. инв №	Инв. № дубл.	Подп. и дата