

**Министерство образования и науки Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ ИУ

КАФЕДРА \_Программное обеспечение ЭВМ и информационные технологии

**ОТЧЕТ ДЛЯ СМОТРА ПО КУРСОВОМУ ПРОЕКТУ**

Студент

Нестеренко Александр Александрович

*фамилия, имя, отчество*

Группа ИУ7-54

Тип практики стационарная

Студент

Нестеренко А.А

*подпись, дата фамилия, и.о.*

Научный руководитель Романова Т.Н

*подпись, дата фамилия, и.о.*

*2018*

# Алгоритм, использующий z-буфер

Пусть мы хотим изобразить отобразить нашу сцену на некотором окне размера x\*y пикселей. Создадим некоторую структуру, которую будем использовать как буфер, например, двумерный массив размера x\*y, в котором будем хранить для каждого пикселя такой параметр, как глубина (z-координата, отсюда и название алгоритма). В начале инициализируем все элементы нашего массива минус бесконечностью. Затем, при отрисовке очередного пикселя на экране будем сравнивать значение его глубины с той, что находится в нашем массиве. Если глубина текущего пикселя больше той, что находится в буфере, то данный пиксель отрисовывается на экране, а элемент буфера инциализируется значением глубины отрисовываемого пикселя.

Общая схема алгоритма с Z-буфером:

1) Инициализировать Z-буфер. Z-буфер заполняется минимальным значением Z.

2) Выполнить преобразование каждого многоугольника сцены в растровую форму. При этом для каждого пиксела вычисляется его глубина z. Если вычисленная глубина текущего пикселя больше, чем глубина, уже имеющаяся в Z-буфере по соответствующим координатам, то занести в Z-буфер его глубину, иначе никаких занесений не выполнять.

Преимущества этого алгоритма:

* Простота реализации
* Скорость работы

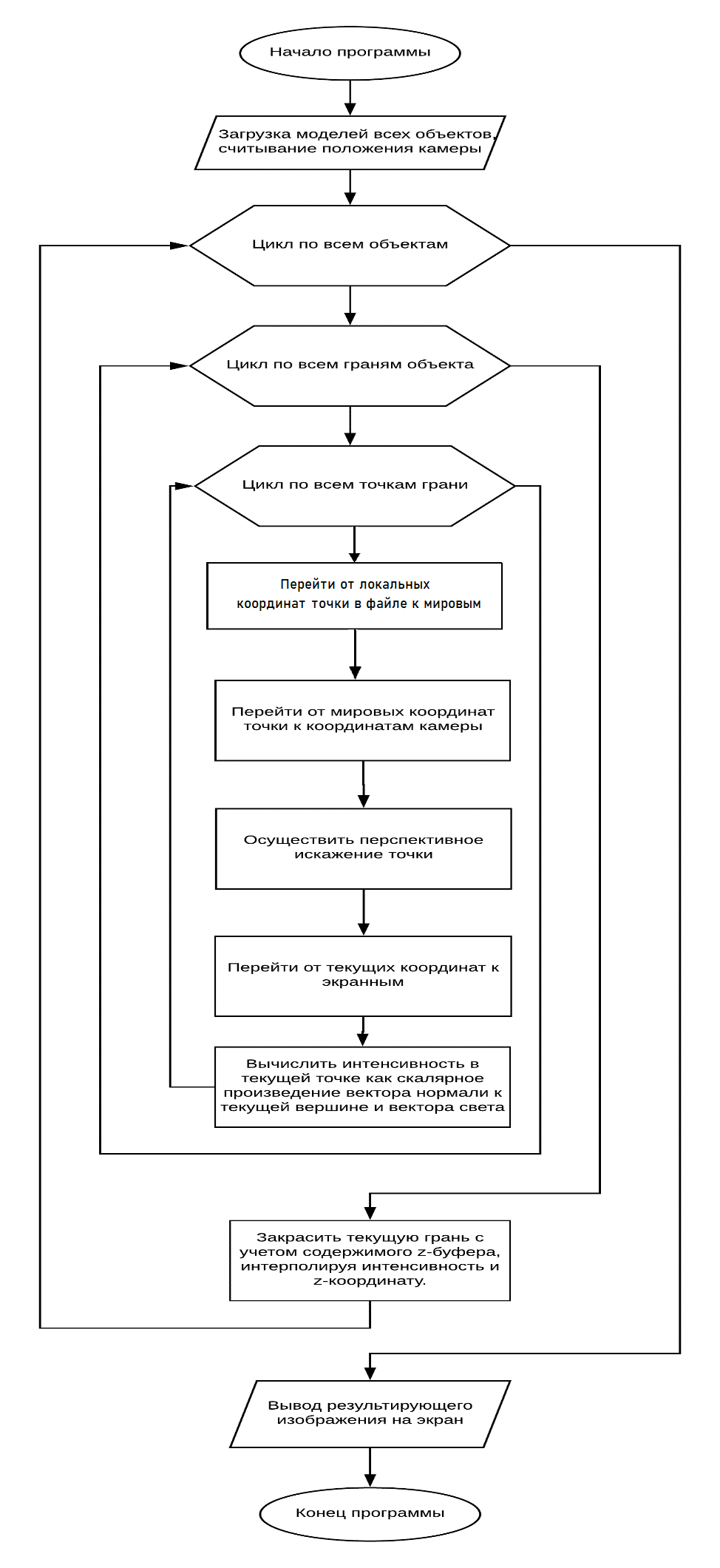
Недостатки этого алгоритма:

* Если некоторые объекты имеют близкую z координату, то иногда, в зависимости от точки обзора, результирующее изображение может быть полосатым (часть пикселей будет принадлежать одному объекту, а часть – другому). Это называют Z-конфликтом, и решают этот конфликт с помощью сдвига одного объекта относительно другого на величину, превышающую погрешность Z-буфера.
* Буфер требует на своё содержание память. Так, например, при условии, что буфер представляет собой целочисленный двумерный массив типа unsigned char, картинка 800\*800 пикселей для отображения потребует 640000 байт. Однако в наше время это незначительное количество памяти.

# Конструкторская часть

В качестве алгоритма отсечения поверхностей был выбран алгоритм, использующий z-буфер. Этот метод является очень эффективным по скорости, а его главный недостаток (потребление памяти на содержание z-буфера) в современном мире является очень незначительным, так как даже для изображения сцены на весь экран на современных мониторах (1920x1080 пикселей) потребуется всего 1920\*1080 байт (2,0736 Мегабайт) при использовании в качества буфера двумерного массива, каждый элемент которого – 1 байт.

Для придания реалистичности изображению также используем метод тонирования Гуро, учтем в изображении перспективу и, пользуясь знаниями из аналитической геометрии, реализуем подпрограмму “камера”, которая позволит рассматривать изображение с разных сторон.



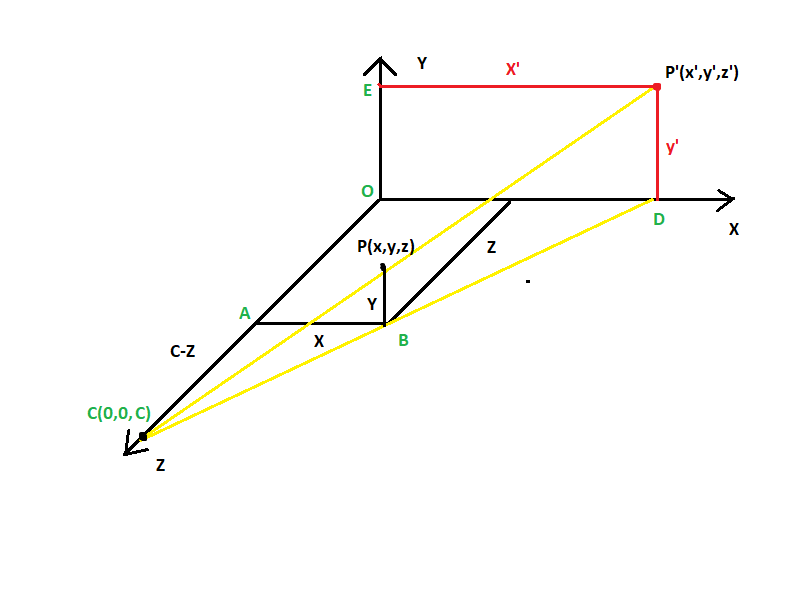
Блок-схема, описывающая принцип работы программы

Прежде чем перейти непосредственно к рассмотрению алгоритма z-буфера, посмотрим, как отобразить каркасную модель объекта, заданную своими вершинами и ребрами, на экране, учтя перспективу, и как поворачивать камеру.

# Учет перспективы

В курсе двумерной графики преобразования точки на плоскости при использовании матриц делались следующим образом: мы добавляли к нашей точке еще одну единичную компоненту, помещая тем самым точку в трехмерное пространство, затем уже в этом трехмерном пространстве производили какие-либо преобразования, и в конце проецировали точку обратно в 2d, поделив координаты x и y на 3-ю добавочную компоненту (обычно она была единицей). Попробуем сделать то же самое и в трехмерном пространстве.

Пусть у нас есть точка P с координатами (x, y, z).



И мы хотим спроецировать её на плоскость z = 0, при этом наша камера находится на оси z на расстоянии C от центра координат.

Треугольники ABC и ODC будут подобны (по двум пропорциональным сторонам и углу между ними), поэтому мы можем записать:

|AB|/|AC| = |OD|/|OC| => x/(c-z) = x’/c (2)

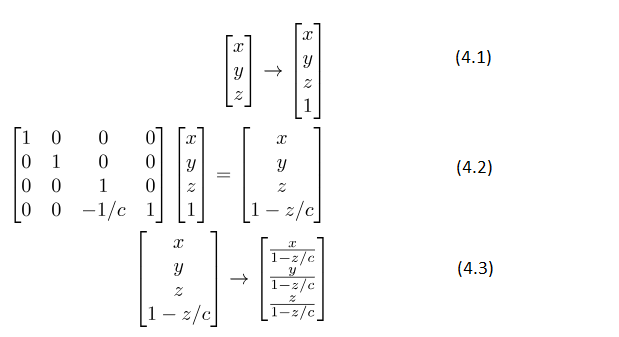
Из (2) выразим x’:

x’ = x/(1-z/c) (3.1)

Рассматривая треугольники CPB и CP’D можно прийти для подобной записи для координаты y:

y’ = y/(1-z/c) (3.2)

Таким образом, с учетом уравнений (3.1) и (3.2) мы можем составить последовательность матричных преобразований для точки, которые будут давать нам новые координаты этой же точки, но уже с учетом перспективы:



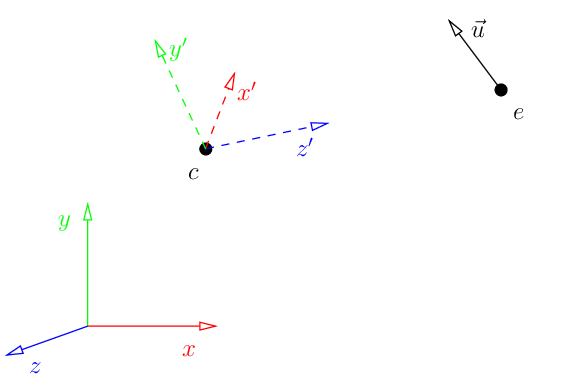
В (4.1) – переводим трехмерную точку в четырехмерное пространство.

В (4.2) – Выполняем перспективное преобразование (**c** – расстояние от начала координат по оси **z**, наша камера тоже расположена на оси **z**)

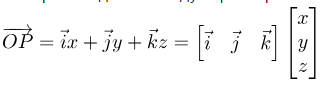
В (4.3) – Проецируем результирующую точку из четырехмерного пространства в трехмерное.

# Поворот камеры

Пусть у нас есть некоторые трехмерные точки **е**, **c** и вектор **u**. Мы хотим сделать так, чтобы наша камера находилась в точке **e**, смотрела на точку **c**, **а** также хотим, чтобы вектор **u** в финальной картинке был вертикален.

****

Любая система координат задается точкой отсчёта и базисом пространства. Если некоторая точка P имеет координаты (x, y, z) в системе координат (O, i, j, k), это значит, что вектор OP можно задать следующим образом.



Пусть у нас также есть вторая система координат (O’, i’, j’, j’).

Тогда должна существовать невырожденная матрица M, что

Screenshot_52

Итак:

1) Вычислим вектор **z'** (он будет представлять собой вектор **ce**)

2) **x’** найдем как векторное произведение между **u** и **ce**

3) **y’** найдем как векторное произведение x’ и z’

4) Нормализуем полученные вектора

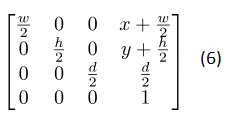
5) Делаем параллельный перенос в **с**

В итоге получим матрицу, где 4x4, где на первой строчке в первых трех столбцах – компоненты x, y, z вектора **x’,** на второй – вектора **y’**, на третей – **z’**, а последний (4-ый) столбец будет представлять собой компоненты вектора **c**, элемент (4,4) будет единицей

Т.е мы получим матрицу, умножив на которую любую точку с координатами (x,y,z,1) в старом базисе, получим координаты в новом базисе. Эту матрицу назовем матрицей обзора.

# Размер объекта на экране, переход к экранным координатам.

С помощью этой матрицы:



мы отображаем куб мировых координат [-1,1]\*[-1,1]\*[-1,1] в куб экранных координат [x,x+w]\*[y,y+h]\*[0,d] . Параметры w и h нужны для задания размера объекта, x и y – для задания объекта в экранной системе координат, а параметр d – глубина, пригодится для использования его в z-буфере (величина этого параметра зависит от того, какой размер у элемента буфера).

# Итоговая цепь преобразований

Пусть объекты заданы точками в своей локальной системе координат.

Сначала мы помещаем эти точки в четырехмерное пространство, добавив к ним компоненту 1 (матрица 4.1), далее переходим к мировым с помощью умножения их на матрицу обзора, затем умножаем результат на матрицы (4.2-4.3) для получения перспективного искажения, затем умножаем результат на матрицу (6) получаем результирующую точку уже в экранных координатах.

# Растеризация треугольников и z-буфер.

Закрашивать треугольники будем следующим образом:

На вход будем подавать 3 точки – вершины треугольника, имеющие координаты x, y и z.

1) Отсортируем все вершины треугольника (t0, t1 и t2) по возрастанию координаты y. Для примера пусть пусть t2>t1>t0.

2) Посчитаем разность максимальной координаты y и минимальной:

total\_height = (t2.y-t0.y)

3) Закрашиваем первую половину треугольника:

3.1) В цикле от y = t0.y до t1.y с шагом 1:

3.1.1) Высчитываем строку, на которой сейчас находимся как segment\_height = t1.y-t0.y+1

3.1.2) Высчитываем параметр alpha для интерполяции как

(y-t0.y)/total\_height

3.1.3) Высчитываем параметр beta для интерполяции как

(y-t0.y)/segment\_height

3.1.4) Создаем точку A, имеющую координаты t0 + (t2-t0)\*alpha

3.1.5) Создаем точку B, имеющую координаты t0 + (t1-t0)\*beta

3.1.5) Если A.x > B.x, то меняем их местами

3.1.6) Закрашиваем текущую строку от A.x до B.x цветом color

4) Закрашиваем вторую половину треугольника:

4.1) В цикле от y = t1.y до t2.y с шагом 1:

4.1.1) Высчитываем строку, на которой сейчас находимся как segment\_height = t2.y-t0.y+1

4.1.2) Высчитываем параметр alpha для интерполяции как

(y-t0.y)/total\_height

4.1.3) Высчитываем параметр beta для интерполяции как

(y-t1.y)/segment\_height

4.1.4) Создаем точку A, имеющую координаты t0 + (t2-t0)\*alpha

4.1.5) Создаем точку B, имеющую координаты t0 + (t2-t1)\*beta

4.1.5) Если A.x > B.x, то меняем их местами

4.1.6) Закрашиваем текущую строку от A.x до B.x цветом color

5) Конец работы алгоритма

Этот алгоритм очень удобен для его последующей модификации с учетом использования z-буфера и дополнения каким-либо алгоритмом освещенности (например, тонировкой Гуро).

В пунктах алгоритма (3.1.4), (3.1.5), (4.1.4), (4.1.5) мы находим коэффициенты alpha и beta, тем самым делая линейную интерполяцию. Аналогично можно поступить и с координатой z:

На этапе закраски очередной строки (пункт 3.1.6 и 4.1.6 в алгоритме) мы вводим новый параметр phi

Если B.x == A.x, то phi = 1 иначе phi = (текущаяя коорината x – A.x)/(B.x-A.x)

Затем находим искомую координату z как

current\_z = A.z + (B.z – A.z)\*phi. (7)

Создадим z-буфер – двумерный массив, все элементы которого изначально инициализированы минус бесконечностью.

Теперь, перед закраской очередного пикселя, необходимо сделать проверку:

Если current\_z, полученное в (7), больше чем значение z в z-буфере, то занести current\_z в z-буфер и закрасить текущий пиксель, иначе ничего не делать.

# Учет освещенности

Интенсивность освещённости равна скалярному произведению вектора света и нормали к данному треугольнику. Для поиска нормали к треугольнику можно воспользоваться векторным произведением его ребер, однако мы вычислим их заранее и будем считывать с файла модели. При этом если скалярное произведение получится отрицательным, то это означает, что свет падает позади треугольника. Такие треугольники можно не отрисовывать, это увеличит скорость работы. При этом если мы будем использовать интенсивность не в какой-то одной точке треугольника, а посчитаем произведения нормалей и вектора света к каждой вершине треугольника, то, интерполируя эту интенсивность по аналогии с координатой z, мы получим более плавную и реалистичную освещенность наших моделей. Этот прием называется тонированием Гуро.

# Технологический раздел

# Структуры и классы программы

**class Vec3 –** класс, хранящая информацию о трехмерном векторе. Может также использоваться для хранения информации о трехмерной точке. Этот класс имеет методы:

1) Векторное произведение с другим трехмерным вектором

2) Сложение с другим трехмерным вектором

3) Разность с другим трехмерным вектором

4) Умножение трехмерного вектора на число

5) Нормирование вектора

6) Скалярное произведение с другим трехмерным вектором

**сlass Matrix –** класс матриц. Имеет конструктор, позволяющий создавать матирцы произвольного размера. Имеет следующие методы:

1) Матричное умножение

2) Транспонирование матрицы

3) Нахождение обратное матрицы

**class Model –** класс, хранящий информацию о 3d-модели. Имеет конструктор с параметром, инициализирующий поля вершин, граней и нормалей. Имеет следующие методы:

1) Возврат количества вершин

2) Возврат количества граней

3) Возврат грани по её индексу

4) Возврат вершины по её индексу

5) Возврат нормали по её индексу

**class Color** - описывает цвет по трём составляющим (RGB – red green blue), каждая из составляющих принимает значение от 0 до 255.

**Class Image –** содержит в себе методы:

1) растеризации треугольника

2) учета перспективы

3) Преобразования мировых координат в экранные

4) Определения той части сцены, которую мы видим (отрисовываем на полотне).

## 

## **Выбор среды программирования**

Среда программирования была выбрана среда Visual Studio 2017. Одно из главных преимуществ этой среды программирование – отсутствие необходимости вручную составлять make-файлы, среда делает это за программиста. Среда обладает хорошей документацией на русском языке. С помощью встроенного класса bitmap можно легко создавать графические приложения, а с помощью пакета Windows Forms можно легко создать пользовательский интерфейс, при этом изображение, полученное в классе bitmap, можно легко отобразить на полотне picture box пакета Windows Forms.

# Входные и выходные данные

На вход программе подаются объекты: стол, два стула, 3 шара и лампа, параллелепипед(комната), которые описаны в отдельных текстовых файлах в следующем формате: сначала на отдельных строках указываем координаты вершин в формате v x/y/z, где v – индикатор для синтаксического анализатора, что v – вершина, а x, y и z – некоторые вещественные числа. После вершин в формате vn x/y/z указываем векторы нормалей к вершинам, где vn - индикатор для синтаксического анализатора, что v – вершина, а x, y и z – некоторые вещественные числа. Далее в формате f v1/n1 v2/n2 v3/n3 определяем поверхности, здесь v1..v3 – индексы вершин, и n1..n3 – индексы нормалей, описанных ранее (нумерации начинается с единицы). Также с пользовательского интерфейса вводятся координаты положения наблюдателя в формате x/y/z, где x/y/z – вещественные числа.

Выходными данными является вывод конечного изображения на экран пользователя.

Примеры работы программы:

