Государственное образовательное учреждение высшего профессионального образования

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

(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Р А С Ч Ё Т Н О - П О Я С Н И Т Е Л Ь Н А Я З А П И С К А

к курсовой работе на тему:

Реализация маятника Ньютона

Студент _____ Сукочева А.
(Подпись, дата)

Руководитель курсового проекта _____ Волкова Л.Л.
(Подпись, дата)

Москва 2020

Содержание

Введение

Физические тела, окружающие нас, обладают различными оптическими свойствами. Они, к примеру, могу отражать или пропускать световые лучи, также они могут отбрасывать тень. Эти и другие свойства нужно уметь наглядно показывать при помощи электронно-вычислительных машин. Этим и занимается компьютерная графика.

Компьютерная графика – представляет собой совокупность методов и способов преобразования информации в графическое представление при помощи ЭВМ. Без компьютерной графики не обходится ни одна современная программа. В течении нескольких десятилетий компьютерная графика прошла долгий путь, начиная с базовых алгоритмов, таких как вычерчивание линий и отрезков, до построения виртуальной реальности.

Целью данной практики является разработка аналитической части проекта по дисциплине компьютерная графика на тему "визуализация маятника Ньютона".

В рамках выполнения работы необходимо решить следующие задачи:

а) Описать предметную область работы.

б) Рассмотреть существующие алгоритмы построения реалистичных изображений.

в) Выбрать и обосновать выбор реализуемых методов или алгоритмов.

1 Аналитический раздел

1.1 Постановка задачи

В соответствии с техническим заданием в области компьютерной графики, необходимо реализовать программный продукт, предоставляющий визуализацию маятника Ньютона. Нужно определиться с выбором метода решения. Пользователь должен иметь возможность запуска и останова визуализируемой механической системы. В данной системе будут отсутствовать противодействующие силы, поэтому она будет действовать до тех пор, пока что пользователь не решит ее остановить. Полученное изображение должно четко показывать работу механической системы.

1.2 Описание предметной области

Компьютерная графика – занимает большой раздел в IT сфере. Она помогает решать многие задачи, актуальные в нашем времени. В компьютерной графике рассматривают обязательной задачей разработку алгоритмов визуализации трехмерных объектов [1]. К примеру, визуализацию механических систем для демонстрации какого-то существующего явления без нужных, для данного явления, аппаратов.

1.3 Описание алгоритмов

1.3.1 Некоторые теоретические сведения

Прежде чем описывать алгоритмы, я хотела бы дать некоторые определения, чтобы читателю было проще воспринимать рассказанные ниже алгоритмы [1, 2].

Алгоритмы удаления невидимых линий и поверхностей служат для определения линий ребер, поверхностей, которые видимы или невидимы для наблюдателя, находящегося в заданной точке пространства.

Решать задачу можно в пространстве:

а) Объектном - мировая системе координат, высокая точность. Обобщённый подход, основанный на анализе пространства объектов, предполагает попарное сравнение положения всех объектов по отношению к наблюдателю.

б) Изображений - в экранных координатах, системе координат, связанной с тем устройством в котором мы отображаем результат отображения (Графический дисплей).

Под экранированием подразумевается загораживание одного объекта другим.

Под глубиной подразумевается значение координаты Z.

1.3.2 Алгоритм Робертса

Алгоритм Робертса – решает задачу удаления невидимых линий. Работает в объектном пространстве. Он строго работает с выпуклыми телами. Если тело изначально является не выпуклым, то нужно его разбить на выпуклые составляющие. Алгоритм целиком основам на математических предпосылках, которые просты, точны и мощны [3].

Основные этапы:

а) Подготовка исходных данных.

б) Удаление линий, экранируемых самим телом.

в) Удаление линий, экранируемых другими телами.

г) Удаление линий пересечения тел, экранируемых самими телами, связанными отношением протыкания и другими телами.

0. Подготовка исходных данных:

Для каждого тела сцены необходимо сформировать матрицу тела. Обозначают: V. Размерность: 4*n, где n - количество граней. Каждый столбец матрицы - это коэффициенты уравнения плоскости (4 коэффициента), проходящей через очередную грань тела.

Уравнение плоскости:

$$Ax + By + Cz + D = 0 \tag{1.1}$$

В матричной форме выглядит следующим образом:

$$[x\ y\ z\ 1][P] = 0 \tag{1.2}$$

где

$$P = \begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix} \tag{1.3}$$

Тогда матрица тела будет выглядеть следующим образом:

$$V = \begin{pmatrix} A_1 & A_2 & \dots & A_n \\ B_1 & B_2 & \dots & B_n \\ C_1 & C_2 & \dots & C_n \\ D_1 & D_2 & \dots & D_n \end{pmatrix} \tag{1.4}$$

где n - количество граней.

Формирование матрицы:

Нужно найти коэффициенты каждой плоскости, проходящей через каждую грань тела. Располагая информацией о координатах 3 неколлинеарных точках, принадлежащий плоскости, можно найти коэффициенты данной плоскости. Плоскость однозначно задается 3 точками. Делим уравнение на d, при этом свободный член будет равен единице.

$$\begin{cases} Ax_1 + By_1 + Cz_1 = -1 \\ Ax_2 + By_2 + Cz_2 = -1 \\ Ax_3 + By_3 + Cz_3 = -1 \end{cases} \tag{1.5}$$

Запишем в матричной форме:

$$[X][C] = [D] \tag{1.6}$$

где

$$X = \begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{pmatrix} \tag{1.7}$$

$$C = \begin{pmatrix} A \\ B \\ C \end{pmatrix} \tag{1.8}$$

$$D = \begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix} \tag{1.9}$$

Решая уравнение (1.6), получаем значения коэффициентов уравнения плоскости:

$$[C] = [X]^{-1}[D] \tag{1.10}$$

Сформировав матрицу тела нужно проверить ее на корректность.

Корректная матрица тела - это такая матрица, у которой любая точка, расположенная внутри тела, должна располагаться по положительную сторону от каждой грани тела. Т.е. при подставки координат любой точки, расположенной внутри тела в каждое из уравнение плоскостей в результате должно получаться положительное число. Если для очередной грани это условие не выполняется, то соответствующий

столбец матрицы нужно умножить на -1. Чтобы не ошибиться с выбором точки, расположенной внутри матрицы тела, можно усреднить координаты всех вершин тела, либо взять полусумму координат максимальных и минимальных значений.

1. Удаление линий, экранируемых самим телом.

Для решения этой задачи нужно указать место расположения наблюдателя и направление его взгляда.

$$E = |00 - 10| \tag{1.11}$$

С одной стороны (1.11) это вектор взгляда наблюдателя, с другой стороны это вектор однородных координат точки, расположенной в минус бесконечности по оси z.

Для определения невидимых граней следует вектор взгляда E умножим матрицу тела V. Отрицательные компоненты полученного вектора будут соответствовать невидимым граням.

2. Удаление линий, экранируемых другими телами.

Наблюдатель находится в плюс бесконечности на оси z. Если на 1 этапе мы использовали вектор взгляда, то на 2 этапе мы используем координаты точки, в которой находится наблюдатель, обозначим как (1.12).

$$g = |0010| \tag{1.12}$$

Для определения невидимых точек ребра нужно построить луч, соединяющий точку наблюдения с точкой находящейся на ребре. Точка будет невидимой, если луч на своем пути встречает в качестве преграды рассматриваемое тело. Если тело является преградой, то луч должен пройти через тело. Если луч проходит через тело, то он находится по положительную сторону от каждой грани тела.

3. Удаление линий пересечения тел, экранируемых самими телами, связанными отношением протыкания и другими телами.

Если тела связаны отношением взаимного протыкания, то образуются новые ребра, соответствующие линиям пересчения тел. Необходимо найти новые реба. Новые ребра можно получить путем соединения точек протыкания между собой. Вновь полученные ребра надо проверить на экранирование самими телами, связанными отношением протыкания. Оставшиеся видимые части ребер надо проверить на экранирование другими телами сцены.

Фактически на 3 этапе мы повторяем первые два этапа для новых ребер, которые получаются в результате протыкания тел.

Алгоритм Робертса хорошо применим для изображения множества выпуклых многогранников. Однако имеются недостатки: для визуализации более естествен-

ного изображения нам нужно уметь работать с тенью. В этом заключается один из недостатков. Без модификации и привлечения сторонних методов данный алгоритм не позволяет нам реализовывать тень. Также невозможно передать зеркальный эффект и преломление света.

1.3.3 Алгоритм Варнока

Алгоритм Варнока – противоположен алгоритму Робертса. Решает задачу в пространстве изображений. Изображает видимые элементы. Единой версии алгоритма Варнока не существует. Можно рассматривать простейшую версию и более сложные версии алгоритма Варнока.

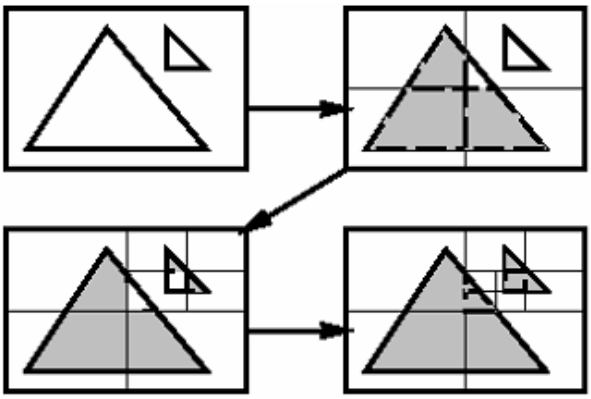

Рисунок 1.1 — Пример разбиения Алгоритмом Варнока

Основная идея состоит в том, что необходимо найти ответ на вопрос о том, что изображать в очередном окне. Сначала окно имеет размеры экрана. Если мы точно не можем дать ответ, то мы окно делим на части (каждую сторону окна делим на две части). Вместо одного окна получаем 4 меньших размеров. Если снова не можем дать ответ, то продолжаем делить каждое окно на 4 части, пока не сможем дать ответ или окно не станет равным в 1 пиксель.

В простейшей версии алгоритма окно делится на подокна всякий раз, если это окно не пусто. Пределом деления является получения окна размером в 1 пиксель. Для одной точки легко определить ближайший к наблюдателю многоугольник (для этого нужно найти глубину каждого многоугольника в этой точки). В более сложных версиях делается попытка решения задачи для окон большего размера (больше одного пикселя). Для этого нужно провести классификацию многоугольников, рассматриваемых в алгоритме Варнока и действия, которые нужно предпринять в том или ином случае.

Классификация многоугольников:

а) С окном связан один охватывающий многоугольник – окно нужно закрасить цветом охватывающего многоугольника. Рисунок 1.2а;

б) С окном связан один пересекающий многоугольник – выполняем отсечение многоугольника по границам окна и получаем один внутренний многоугольник. То есть сводим задачу к случаю с одним внутренним многоугольником. Рисунок 1.2b;

в) С окном связан один внутренний многоугольник – окно нужно закрасить фоновым цветом, а затем выполнить растровую развертку единственного многоугольника. Рисунок 1.2c;

г) Все многоугольники являются внешними по отношению к окну – окно закрасить цветом фона. Рисунок 1.2d;

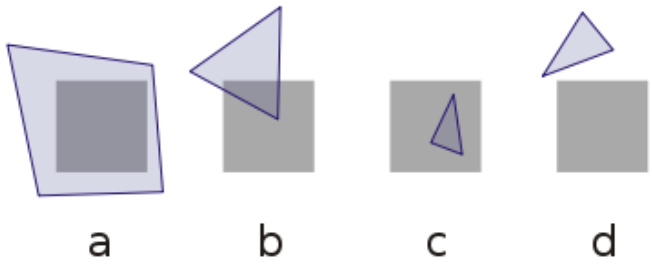

Рисунок 1.2 — Классификация многоугольников в Алгоритме Варнока

К недостаткам алгоритма также можно отнести невозможность передачи зеркальных эффектов и преломления света. Также при визуализации сложной сцены число разбиений может стать очень большим, что приведет к ухудшению скорости.

1.3.4 Алгоритм Вейлера-Азертона

Алгоритм Вейлера-Азертона – пытается минимизировать количество разбиений в алгоритме представленном выше (Алгоритме Варнока) путем разбиения окна вдоль границ многоугольника. Основой служит алгоритм Вейлера-Азертона, который используется для отсечения многоугольников.

Если многоугольники пересекают друг друга, то надо разбить один из них на два линей пересечения плоскостей присущих этим многоугольникам [4].

К недостаткам алгоритма относится сложность реализации, а также невозможность передачи зеркальных эффектов и преломления света.

1.3.5 Алгоритм Z-буфера

Алгоритм Z-буфера – решает задачу в пространстве изображений. Сцены могут быть произвольной сложности, а поскольку размеры изображения ограничены размером экрана дисплея, то трудоемкость алгоритма зависит линейно от числа рассматриваемых поверхностей. Элементы сцены заносятся в буфер кадра в произвольном порядке, поэтому в данном алгоритме не тратится время на выполнение сортировок.

Буфер кадра (регенерации) - используется для заполнения атрибутов (интенсивности) каждого пикселя в пространстве изображения. Для него требуется буфер регенерации, в котором запоминаются значения яркости, а также Z-буфер (буфер глубины), куда можно помещать информацию о координате z для каждого пикселя.

Для начала нам нужно подготовить буферы. Для этого в Z-буфер заносятся максимально возможные значения z, а буфер кадра заполняется значениями пикселя, который описывает фон. Также нам нужно каждый многоугольник преобразовать в растровую форму и записать в буфер кадра. Сам процесс работы заключается в сравнении глубины каждого нового пикселя, который нужно занести в буфер кадра, с глубиной того пикселя, который уже занесён в Z-буфер. В зависимости от сравнения принимается решение, нужно ли заносить новый пиксель в буфер кадра и, если нужно, также корректируется Z-буфер (в него нужно занести глубину нового пикселя).

К недостаткам алгоритма следует отнести довольно большие объёмы требуемой памяти, а также имеются другие недостатки, которые состоят в трудоемкости устранения лестничного эффекта и трудности реализации эффектов прозрачности.

1.3.6 Алгоритм прямой трассировки лучей

Основная идея алгоритма прямой трассировки лучей состоит в том, что наблюдатель видит объекты, благодаря световым лучам, испускаемым некоторым источником, которые падают на объект, отражаются, преломляются или проходят через него и в результате достигают нас [5]. Если проследить за лучами, то становится понятно, что среди них лишь малая часть дойдет до наблюдателя, что приведет к большим затратам ЭВМ. Заменой данному алгоритму служит метод обратный трассировки лучшей.

1.3.7 Алгоритм обратной трассировки лучей

Алгоритм обратной трассировки лучей отслеживает лучи в обратном направление (от наблюдателя к объекту).

Считается, что наблюдатель расположен на положительной полуоси z в бесконечности, поэтому все световые лучи параллельны оси z. В ходе работы испускаются лучи от наблюдателя и ищутся пересечения луча и всех объектов сцены [6]. В результате пересечение с максимальным значением z является видимой частью поверхности и атрибуты данного объекта используются для определения характеристик пикселя, через центр которого проходит данный световой луч. Эффективность процедуры определения пересечений луча с поверхностью объекта оказывает самое большое влияние на эффективность всего алгоритма. Чтобы избавиться от ненужного поиска пересечений было придумано искать пересечение луча с объемной обо-

лочкой рассматриваемого объекта. Под оболочкой понимается некоторый простой объект, внутрь которого можно поместить рассматриваемый объект, к примеру параллелепипед или сфера.

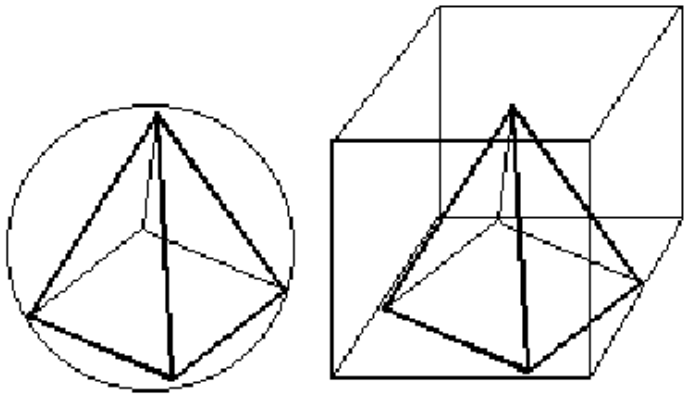

Рисунок 1.3 — Сферическая и прямоугольная оболочки

В дальнейшем при рассмотрении пересечения луча и объемной оболочкой рассматриваемого объекта, если такого пересечения нет, то и соответственно пересечения луча и самого рассматриваемого объекта нет, и наоборот, если мы найдем пересечение, то возможно, есть пересечение луча и рассматриваемого объекта. Для расчета эффектов освещения сцены проводятся вторичные лучи от точек пересечения ко всем источникам света. Если на пути этих лучей встречается непрозрачное тело, значит данная точка находится в тени, иначе он влияет на освещение данной точки. Также для получения более реалистичного изображения сцены, нужно учитывать вклады отраженных и преломленных лучей.

К недостатку алгоритма относится его производительность.

1.4 Модель освещения

Для создания реалистичного изображения в компьютерной графике применяются различные алгоритмы освещения.

Модель освещения предназначена для расчёта интенсивности отражённого к наблюдателю света в каждой точке изображения.

Модель освещения может быть глобальной или локальной.

Локальная модель - учитывается только свет от источников и ориентация поверхности.

Локальная модель включает 3 составляющих:

а) Диффузную составляющую отражения.

б) Отражающую составляющую отражения.

в) Рассеянное освещение.

Глобальная модель - учитывается ещё и свет, отражённый от других поверхностей или пропущенный через них

Глобальная модель воспроизводит чрезвычайно важные эффекты, которые будут показаны в данном проекте.

1.5 Преобразования

1.5.1 Некоторые теоретические сведения

Преобразования на плоскости - изменение значений точек на плоскости:

$$A(x,y) \to B(x_1,y_1) \tag{1.13}$$

Линейное преобразование:

$$\begin{cases} x_1 = Ax + By + C \\ y_1 = Dx + Ey + F \end{cases} \tag{1.14}$$

Однородные координаты:

$$(x, y, w) \tag{1.15}$$

Где w - масштабный множитель (для плоского случая w = 1), в обратном случае:

$$\begin{cases} x' = x/w \\ y' = y/w. \end{cases} \tag{1.16}$$

Матрица преобразований:

$$Mtr = \begin{pmatrix} A & D & 0 \\ B & E & 0 \\ C & F & 1 \end{pmatrix} \tag{1.17}$$

Тогда, используя однородные координаты и матрицу преобразования, мы можем получить результат преобразования:

$$(x_1, y_1, 1) = (x, y, 1) * Mtr \tag{1.18}$$

Методы двумерных преобразований распространяются и на изображения трехмерных объектов. Матрица преобразований в трехмерном пространстве в однородных координатах будет иметь размерность 4*4, а точки (x, y, z) заменяются четверткой (xw, yw, zw, w), $w \neq 0$

1.5.2 Перенос

Для переноса нам потребуется два параметра: dx - смещение по оси абсцисс и dy - смещение по оси ординат.

$$\begin{cases} x_1 = x + dx \\ y_1 = y + dy \end{cases} \tag{1.19}$$

Матрица переноса в двухмерном пространстве:

$$Mmove = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ dx & dy & 1 \end{pmatrix} \tag{1.20}$$

Матрица переноса в трехмерном пространстве:

$$Mmove = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ dx & dy & dz & 1 \end{pmatrix} \tag{1.21}$$

1.5.3 Масштабирование

Масштабирование - изменение размера. Задается коэффициентами масштабирования kx, ky и центром масштабирования xm, ym. Иллюстрация представлена на рисунке 1.4

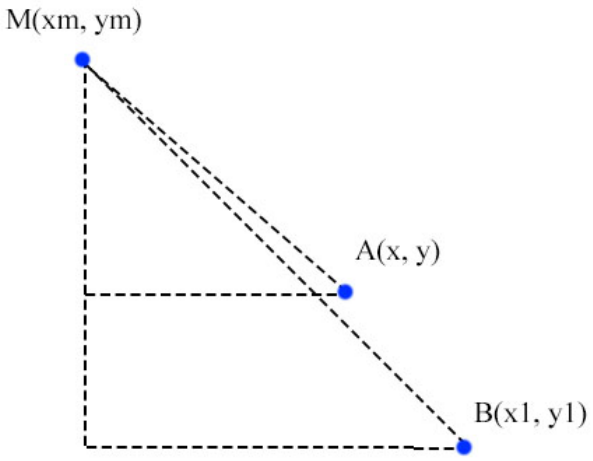

Рисунок 1.4 — Перемещение точки

kx = ky - масштабирование однородное.

kx != ky - масштабирование неоднородное.

Отражение относительно OY: kx = -1, ky = 1.

Отражение относительно OX: kx = 1, ky = -1.

Отражение относительно начала координат: kx = -1, ky = -1.

Если коэффициент масштабирования больше 1, то изображение удаляется от центра и увеличивается.

Если коэффициент масштабирование меньше 1, то изображение приближается от центра и уменьшается.

Из Рис. 1.4 получаем:

$$\begin{cases} x_1 - x_m = kx(x - x_m) \\ y_1 - y_m = ky(y - y_m) \\ \end{cases} \tag{1.22}$$

Упростим (1.22) и получим:

$$\begin{cases} x_1 = kx * x + (1 - kx) * x_m \\ y_1 = ky * y + (1 - ky) * y_m \end{cases} \tag{1.23}$$

Матрица масштабирования в двухмерном пространстве:

$$Mscale = \begin{pmatrix} kx & 0 & 0 \\ 0 & ky & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{1.24}$$

Матрица масштабирования в трехмерном пространстве (Добавляется коэффициентами масштабирования: kz):

$$Mscale = \begin{pmatrix} kx & 0 & 0 & 0 \\ 0 & ky & 0 & 0 \\ 0 & 0 & kz & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{1.25}$$

1.5.4 Поворот

Для поворота нам потребуется угол θ и центр C(xc, yc).

Вывод поворота против часовой стрелки:

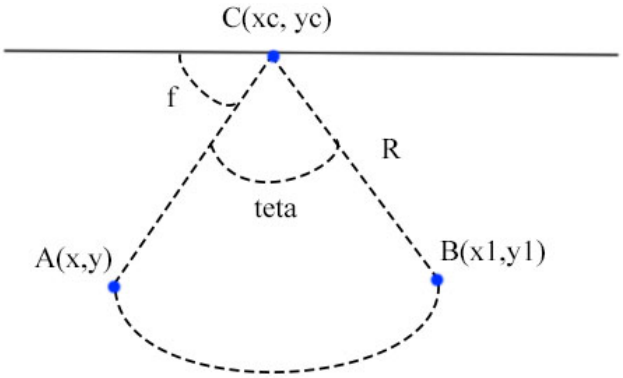

Рисунок 1.5 — Поворот точки

$$
\begin{gathered}
x_1 = x_c + R * cos(180^\circ - (\varphi + \theta)) = \\
x_c - R * cos(\varphi) * cos(\theta) + R * sin(\varphi) * sin(\theta) = \\
x_c - (x_c - x)cos(\theta) + (y - y_c)sin(\theta) = \\
x_c + (x - x_c)cos(\theta) + (y - y_c)sin(\theta)
\end{gathered}
\tag{1.26}
$$

$$
\begin{gathered}
y_1 = y_c + R * sin(180^\circ - (\varphi + \theta)) = \\
y_c + R * sin(\varphi) * cos(\theta) + R * cos(\varphi) * sin(\theta) = \\
y_c + (y - y_c)cos(\theta) + (x_c - x)sin(\theta) = \\
y_c + (y - y_c)cos(\theta) - (x - x_c)sin(\theta)
\end{gathered}
\tag{1.27}
$$

Матрица поворота против часовой стрелки в двухмерном пространстве:

$$
Mrotate = \begin{pmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{1.28}
$$

Матрица поворота в трехмерном пространстве вокруг оси Z :

$$Mrotate = \begin{pmatrix} cos(\theta) & -sin(\theta) & 0 & 0 \\ sin(\theta) & cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{1.29}$$

1.6 Вывод

Оценив все изложенные выше алгоритмы, можно сделать вывод, что для данной работы, которая предполагает визуализацию реалистического изображения, учитывая тени, отражение объектов и тд, подходит алгоритм обратной трассировки лучей, так как он позволяет достичь высокой реалистичности построенного изображения. Он будет использоваться, несмотря на указанные недостатки, так как алгоритм достаточно полно отражает суть физических явлений с приемлемыми затратами производительности.

2 Конструкторский раздел

2.1 Требования к программе

Программа должна предоставлять следующие возможности:

а) Визуализацию сцены.
б) Запуск системы.
в) Останов системы.

2.2 Алгоритм трассировки лучей

2.3 Выбор используемых типов и структур данных

Заключение

В результате выполнения проекта

а) изучены

б) 2

Список использованных источников

1. *Дымченко, Лев.* Пример реализации в реальном времени метода трассировки лучей: необычные возможности и принцип работы. Оптимизация под SSE / Лев Дымченко. — ixbt, 2001. — 11. https://www.ixbt.com/video/rt-raytracing.shtml.

2. Learn Computer Graphics From Scratch! — 2000-2016. https://www.scratchapixel.com/index.php?redirect.

3. *Роджерс, Д.* Алгоритмические основы машинной графики / Д. Роджерс. — Москва «Мир», 1989. — P. 512.

4. *А., Боресков.* Программирование компьютерной графики / Боресков А. — ДМК Пресс, 2019. — P. 370.

5. Ray tracing. — Wikipedia, 2015. https://en.wikipedia.org/wiki/Ray_tracing_(graphics).

6. *Ю.М.Баяковский.* Трассировка лучей из книги Джефа Проузиса / Ю.М.Баяковский. — 1999. https://www.graphicon.ru/oldgr/courses/cg99/notes/lect12/prouzis/raytrace.htm.