1. Титульная страница

2. ТЗ

Оглавление

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 данного курсового проекта является разработка ПО по дисциплине компьютерная графика на тему "визуализация маятника Ньютона".

В рамках выполнения работы необходимо решить следующие задачи:

1. описать предметную область работы;
2. рассмотреть существующие алгоритмы построения реалистичных изображений;
3. выбрать и обосновать выбор реализуемых методов или алгоритмов;
4. подробно изучить выбранный алгоритм;
5. разработать программу на основе одного из существующих алгоритмов;
6. увеличить скорость работы выбранного алгоритма;

7. выбрать и обосновать выбор языка программирования, для решения данной задачи.

1 | Аналитическая часть

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 [7]. 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, нужно дать некоторые определения, чтобы читателю было проще воспринимать рассказанные ниже алгоритмы [7].

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:

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

2. Изображений - в экранных координатах, системе координат, связанной с тем устройством в котором отображается результат. (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 [6].

Основные этапы:

1. подготовка исходных данных;

2. удаление линий, экранируемых самим телом;

3. удаление линий, экранируемых другими телами;

4. удаление линий пересечения тел, экранируемых самими телами, связанными отношением протыкания и другими телами.

1. 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.

2. 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.

3. Удаление линий, экранируемых другими телами.

Наблюдатель находится в плюс бесконечности на оси z. Если на 1 этапе использовался вектор взгляда, то на 2 этапе используются 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.

4. 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 повторяются 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 естественного 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.

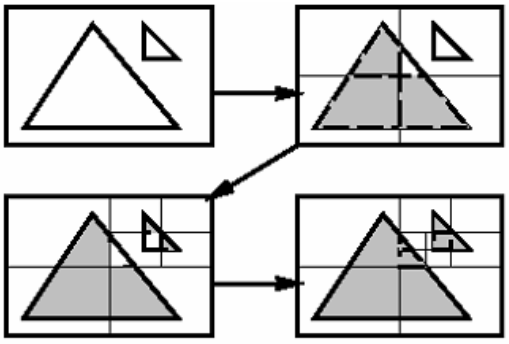

Рис. 1.1 — Пример разбиения Алгоритмом Варнока

Основная идея состоит в том, что необходимо найти ответ на вопрос о том, что изображать в очередном окне. Сначала окно имеет размеры экрана. Если нельзя точно дать ответ, то окно делится 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:

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

2. 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;

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

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

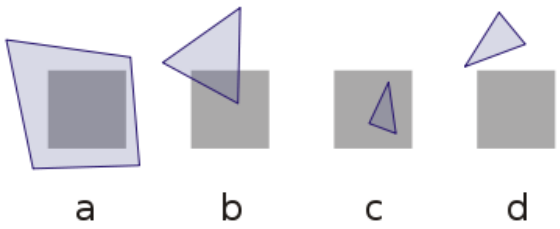

Рис. 1.2 — Классификация многоугольников в Алгоритме Варнока

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 [7].

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 занесен 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 объемы 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 [8]. 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 [8]. 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 оболочкой рассматриваемого объекта. Под оболочкой понимается некоторый простой объект, внутрь которого можно поместить рассматриваемый объект, к примеру параллелепипед или сфера.

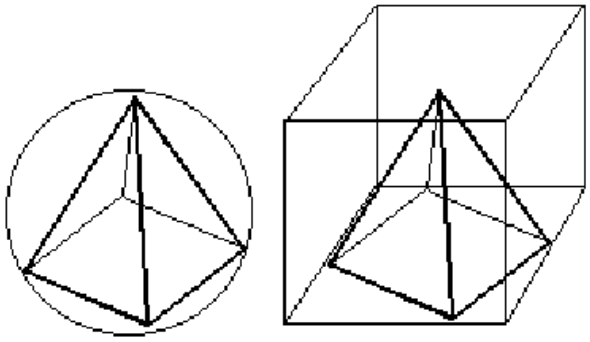

Рис. 1.3 — Сферическая и прямоугольная оболочки

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 найдено 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 расчета интенсивности отраженного 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:

1. Диффузную составляющую отражения.
2. Отражающую составляющую отражения.
3. Рассеянное освещение.

Глобальная модель - учитывается еще и свет, отраженный от других поверхностей или пропущенный через них

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, можно 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 отлично от 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

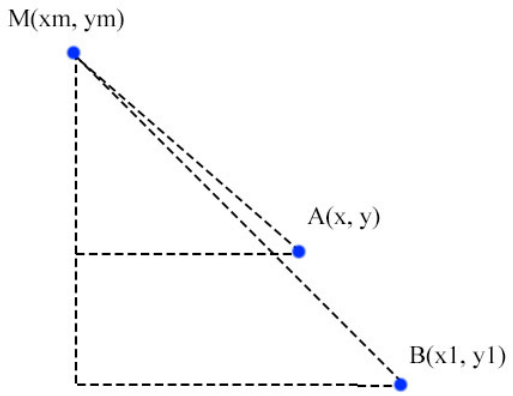

Рис. 1.4 — Перемещение точки

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:

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

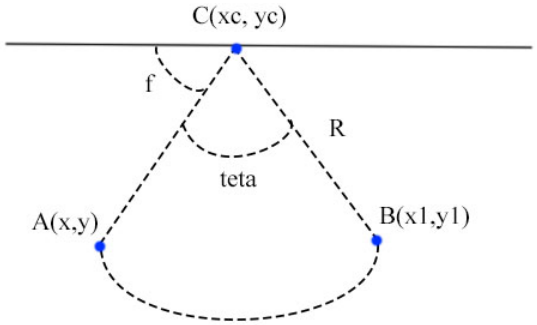

Рис. 1.5 — Поворот точки

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

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 Увеличение скорости работы трассировки лучей

Параллельные вычисления часто используются для увеличения скорости выполнения программ.

Поскольку алгоритм обратной трассировки лучей обрабатывает каждый пиксель экрана независимо, можно распараллелить обработку всего экрана, разбив его на некоторые части. Разбиение экрана можно производить горизонтально или вертикально. После разбиения каждый поток будет обрабатывать свой участок экрана.

На рис. 1.6 показано, как можно вертикально разбить экран на несколько частей. На рис. 1.7 показано горизонтальное разбиение экрана. Разбив экран на части можно реализовывать параллельное вычисление цвета пикселей каждой части экрана.

Рис. 1.6 — Вертикальное разбиение экрана

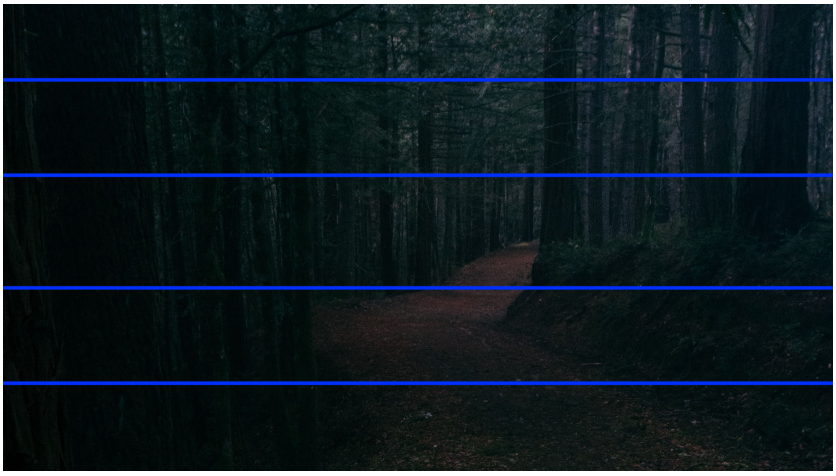

Рис. 1.7 — Горизонтальное разбиение экрана

1.7 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. Также для улучшения скорости работы алгоритма будет применено многопоточное выполнение.

2 | Конструкторская часть

2.1 Требования к программе

Программа должна предоставлять следующие возможности:

1. визуализацию сцены;
2. запуск системы;
3. останов системы.

2.2 Алгоритм обратной трассировки лучей

В данном проекте алгоритм обратной трассировки лучей будет применяться к сфере и цилиндру, для того, чтобы воспользоваться данным алгоритмом нам нужно уметь находить пересечение луча со сферой и цилиндром.

2.2.1 Пересечения луча и сферы

Уравнение луча представлено ниже:

$$P = O + t(V - O), t \geq 0 \tag{2.1}$$

Где O - центр камеры, а V - текущий пиксель.
Обозначим направление луча:

$$\overrightarrow{D} = V - O \tag{2.2}$$

Уравнение 2.3 эквивалентно уравнению 2.1

$$\begin{cases} x(t) = x_O + tx_D \\ y(t) = y_O + ty_D \\ z(t) = z_O + tz_D \end{cases} \tag{2.3}$$

Рассмотрим, что из себя представляет сфера.

Сфера — это множество точек P, лежащих на постоянном расстоянии r от фиксированной точки C. Тогда можно записать уравнение, удовлетворяющее этому условию:

$$distance(P, C) = r \tag{2.4}$$

Запишем расстояние (2.4) между P и C, как длину вектора из P в C.

$$|P - C| = r \tag{2.5}$$

Заменим на скалярное произведения вектора на себя:

$$\sqrt{\langle P - C\rangle, \langle P - C\rangle} = r \tag{2.6}$$

И избавимся от корня:

$$\langle P - C\rangle, \langle P - C\rangle = r^2 \tag{2.7}$$

В итоге мы имеем два уравнения - уравнение луча и сферы. Найдем пересечение луча со сферой. Для этого подставим (2.1) в (2.7)

$$\langle O + t\overrightarrow{D} - C\rangle, \langle O + t\overrightarrow{D} - C\rangle = r^2 \tag{2.8}$$

Разложим скалярное произведение и преобразуем его. В результате получим:

$$t^2\langle\overrightarrow{D}, \overrightarrow{D}\rangle + 2t\langle\overrightarrow{OC}, \overrightarrow{D}\rangle + \langle\overrightarrow{OC}, \overrightarrow{OC}\rangle - r^2 = 0 \tag{2.9}$$

Решение представленного уравнения (2.9) можно получить решив дискриминант. В итоге у нас может получится одно решение - это обозначает, что луч касается сферы. Два решения обозначают, то что луч входит в сферу и выходит из неё. И если нет решений, значит луч не пересекается со сферой.

2.2.2 Пересечения луча и цилиндра

Бесконечный цилиндр, образующие которого параллельны оси z имеет уравнение 2.10

$$x^2 + y^2 = r^2 \tag{2.10}$$

Аналогичные уравнение 2.11 и 2.12 для бесконечных цилиндров, образующие которых параллельны осям x и y соответственно.

$$y^2 + z^2 = r^2 \tag{2.11}$$

$$x^2 + z^2 = r^2 \tag{2.12}$$

Рассмотрим цилиндр, выровненный вдоль оси z (уравнение 2.10). Аналогичные решения получатся для 2.11 и 2.12. Найдем пересечение луча с цилиндром. Для этого подставим 2.3 в 2.10

$$(x_O + tx_D)^2 + (y_O + ty_D)^2 = r^2 \tag{2.13}$$

Вынесем t из скобок в уравнение 2.13 и получим 2.14

$$t^2(x_D^2 + y_D^2) + 2t(x_O x_D + y_O y_D) + (x_O^2 + y_O^2 - r^2) = 0 \tag{2.14}$$

Решение представленного уравнения (2.14) можно получить решив дискриминант.

Для конечного цилиндра нужно ввести ограничение по оси. Для цилиндра, записанного уравнением 2.10 данные ограничения продемонстрированны в 2.15

$$z_{min} < z < z_{max} \tag{2.15}$$

Получив t_1 и t_2 из уравнения 2.14 найдем z_1 и z_2 с помощью уравнения 2.3. Далее необходимо произвести проверку 2.16

$$\begin{cases} z_{min} < z_1 < z_{max} \\ z_{min} < z_2 < z_{max} \end{cases} \tag{2.16}$$

Если обе точки проходят данные тест, то наименьшее неотрицательное значение t – это ближайшая точка пересечения луча с конечным цилиндром.

2.2.3 Алгоритм

Суть алгоритма состоит в следующем:

Из некоторой точки пространства, называемой виртуальным глазом или камерой, через каждый пиксель изображения испускается луч и находится точка пересечения с объектом сцены (2.9 и 2.14). Далее из найденной точки пересечения испускаются лучи до каждого источника освещения. Если данные лучи пересекают другие объекты сцены, значит точка пересечения находится в тени относительно рассматриваемого источника освещения и освещать ее не нужно. Освещение со всех видимых источников света складываются. Далее, если материал рассматриваемого объекта имеет отражающие или преломляющие свойства, из точки испускается отраженный луч и для него вся процедура трассировки рекурсивно повторяется.

На рис. 2.1 представлена схема алгоритма.

2.3 Выбор используемых типов и структур данных

В данной работе нужно будет реализовать следующие типы и структуры данных:

1. точка – хранит положение, задается координатами x, y, z;
2. вектор – хранит направление, задается x, y, z;
3. цвет – вектор из трех чисел (синий, красный, зеленый);
4. сфера – хранит радиус, цвет и центр сферы;
5. цилиндр – хранит радиус, цвет, центр цилиндра и ось (x, y или z), образующие цилиндра параллельны ей;
6. сцена – список объектов, заданных сферами и цилиндрами;
7. источник света – положение и направление света.

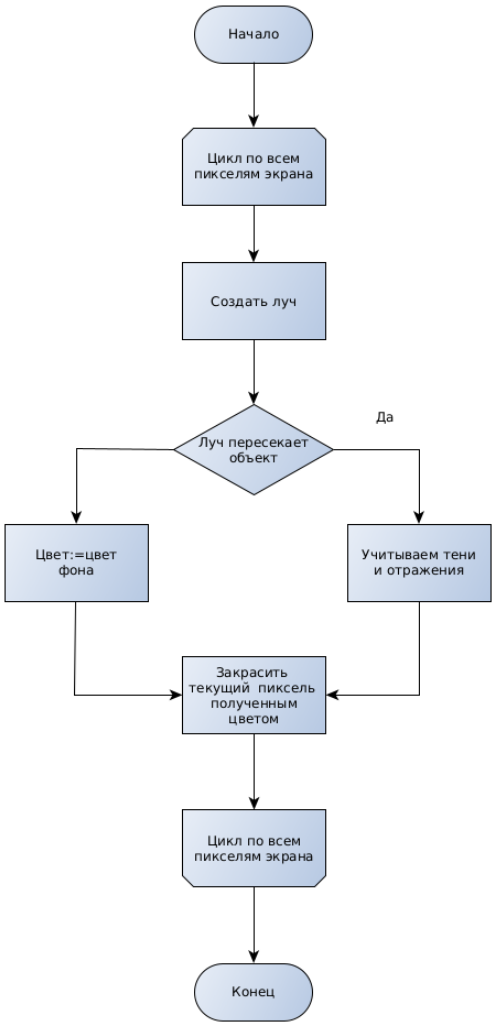

Рис. 2.1 — Блок-схема алгоритма трассировки лучей

2.4 Вывод

В данном разделе были описаны требования к программе, подробно рассмотрен выбранный алгоритм, описаны типы и структуры данных, которые будут реализованы, и приведена схема алгоритма трассировки лучей 2.1.

3 | Технологическая часть

3.1 Выбор языка программирования и среды разработки.

Для решения описанной задачи был выбран язык программирования - C# [4]. Данный язык был выбран по следующим причинам:

- он использует объектно-ориентированный подход к программированию, что позволяет работать по принципу черного ящика;
- в языке присутствует обилие синтаксических возможностей, которые помогают использовать готовые конструкции, вместо того, чтобы переписывать однотипные строки кода;
- он имеет наличие большого количества библиотек и шаблонов, позволяющих не тратить время на изобретение готовых конструкций;
- язык является строго типизированным, что позволяет защититься от непроконтролированных ошибок;
- он является нативным, что необходимо для увеличения скорости работы алгоритмов с помощью распараллеливания.

В качестве среды разработки был использован Visual Studio Code [1]. Visual Studio Code подходит не только для Windows [2], но и для Linux [3], это причина, по которой был выбран VS code, т.к. на моем компьютере установлена ОС Ubuntu 18.04.4 [5]. Также моей архитектуре присутствует 8 ядер.

3.2 Структура программы

Данная программа состоит из следующих модулей:

- Program.cs - файл, содержащий точку входа в программу.
- Vector.cs - файл, содержащий класс Vector, в котором написаны основные методы для работы с вектором;
- Color.cs - файл, содержащий класс Color, в котором написаны основные методы для работы с цветом;
- Constants.cs - файл, содержащий константы;
- Shape.cs - файл, содержащий базовый класс Shape и унаследованные от него классы Sphere и Cylinder;
- Light.cs - файл, содержащий класс Light;
- MinForm.cs - файл, содержащий интерфейс и алгоритм трассировки лучей.

На рисунках 3.1 - 3.2 показана структура классов.

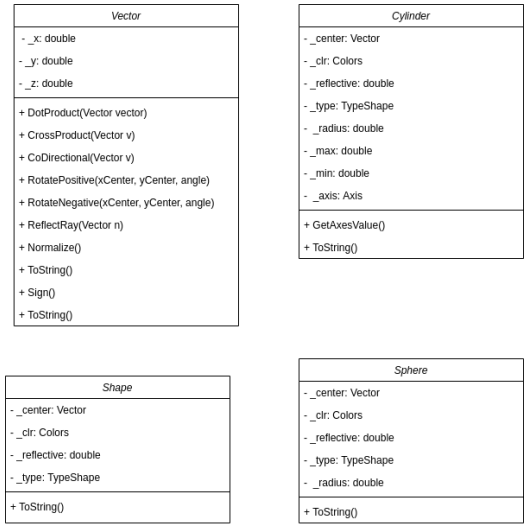

Рис. 3.1 — Структура классов

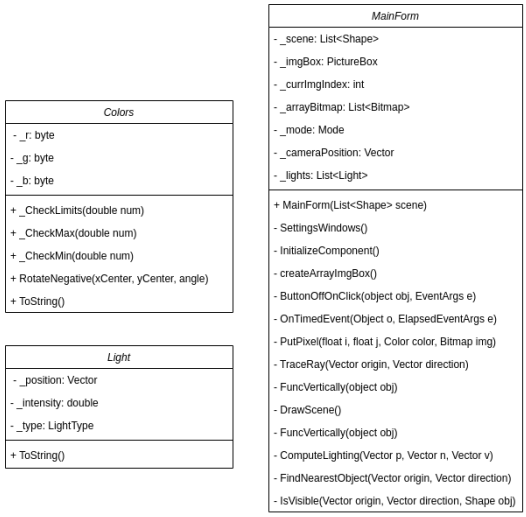

Рис. 3.2 — Структура классов

На листинге 3.1 представлен основной код алгоритма.

Листинг 3.1 — Трассировка лучей.

```
1  private Colors TraceRay(Vector origin, Vector direction,
       double min_t, double max_t, int depth = 3)
2  {
3    double closest_t = Double.PositiveInfinity;
4    Shape ClosestObject = FindNearestObject(origin, direction
       , min_t, max_t, ref closest_t);
5    if (ClosestObject == null)
6      return new Colors(0, 0, 0);
7
8    Vector point = origin + direction * closest_t;
9    Vector tmp = null;
10   if (ClosestObject.Type == TypeShape.Sphere)
11   {
12     tmp = ClosestObject.Center;
13   }
14   else if (ClosestObject.Type == TypeShape.Cylinder)
15   {
16     Cylinder cylinder = ClosestObject as Cylinder;
17
18     Vector axesValue = cylinder.GetAxesValue();
```

```
19      Vector axesValueRev = cylinder.GetAxesValue();
20      axesValueRev.Reverse();
21
22      axesValue = axesValue * cylinder.Center;
23      axesValueRev = axesValueRev * point;
24      tmp = axesValue + axesValueRev;
25    }
26
27    Vector normal = point - tmp;
28    normal = normal * (1.0d / normal.Length);
29    Vector view = direction * -1.0d;
30    double lighting = ComputeLighting(point, normal, view);
31
32    Colors result = ClosestObject.Clr * lighting;
33
34    if (depth <= 0)
35      return result;
36
37    Vector reflected_ray = ReflectRay(view, normal);
38    Colors reflected_color = TraceRay(point, reflected_ray,
        0.001d, Double.PositiveInfinity, depth - 1);
39
40    double reflective = ClosestObject.Reflective;
41    return result * (1 - reflective) + reflected_color *
        reflective;
42  }
```

3.3 Вывод

В данном разделе был выбран ЯП и среда разработки. Также описаны модули и разобран листинг рис 3.1, показывающий работу алгоритма.

4 | Экспериментальная часть

4.1 Замеры времени

Так как одной из цели данного курсового проекта было достаточно быстрое построение реалистичного изображения, то было применено распараллеливание алгоритма обратной трассировки лучей.

Для замеров времени используется изображение размером 1200×600. Так как трассировка данного изображения считается достаточно короткой задачей, используется усреднение массового эксперимента. Сравнение произведено при n = 10.

Результат представлен на рис. 4.1.

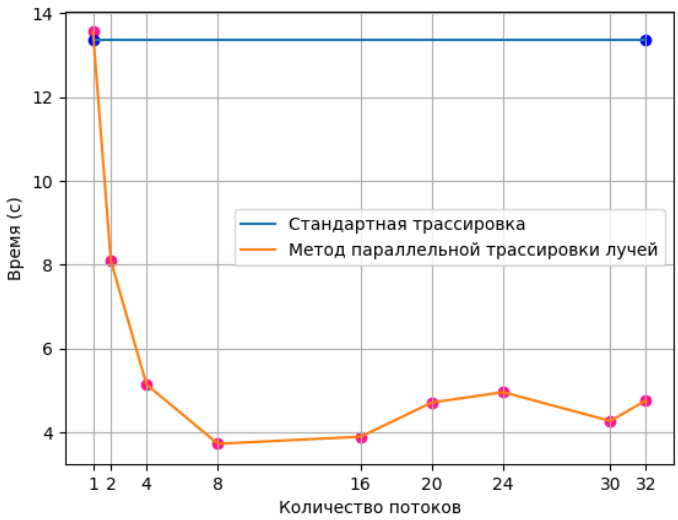

Рис. 4.1 — Временные характеристики

Обычная реализация работает быстрее, чем создание одного потока, потому что на создание потока тратится некоторое время. При двух потоках выигрыш получается почти в два раза, так как два потока трассируют одновременно свою часть экрана. При увеличении числа потоков отслеживается уменьшение времени трассировки. При 8 потоках достигается пик, при котором все ядра процессора одновременно выполняют трассировку экрана. Далее при увеличении числа потоков производительность падает. Это объясняется тем, что создается очередь потоков, которая замедляет работу программы.

4.2 Результат работы алгоритма

На рисунке 4.2 и 4.3 показана работа алгоритма трассировки лучей.

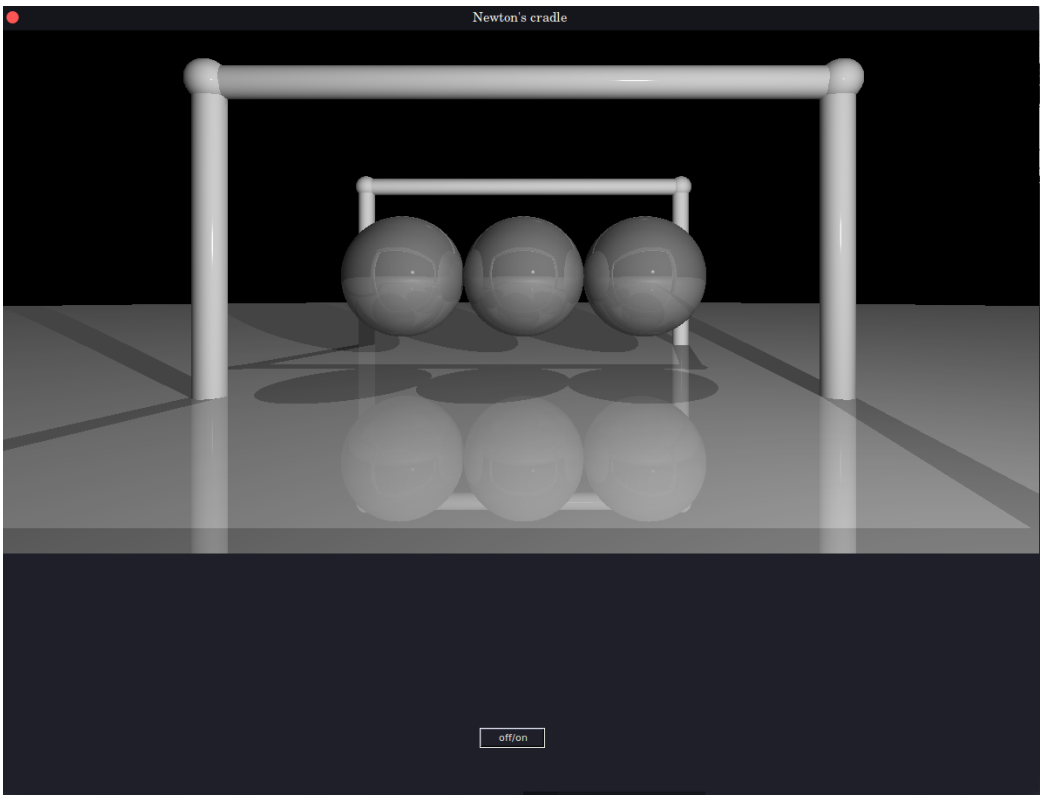

Рис. 4.2 — Результат работы программы

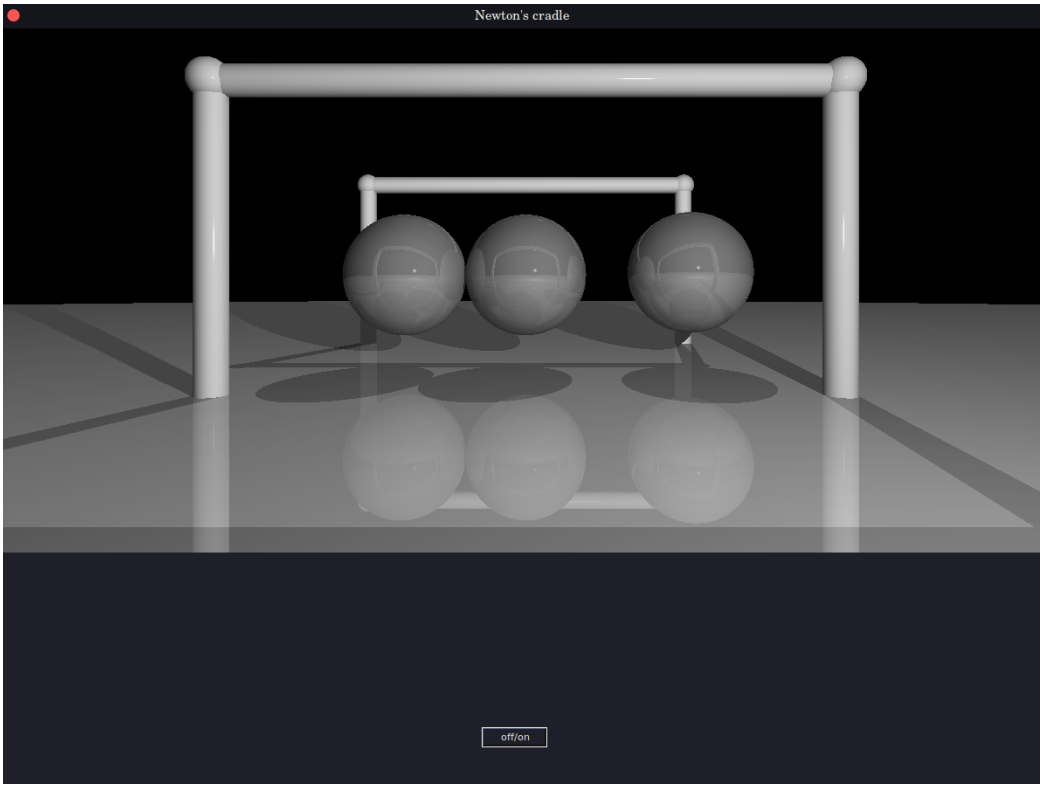

Рис. 4.3 — Результат работы программы

4.3 Вывод

В данном разделе было произведено сравнение алгоритма трассировки лучей при простой реализации и многопоточной (рис. 4.1). Результат показал, что выгоднее всего использовать все ядра процессора. Так же была продемонстрированна работа программы (рис. 4.2 и 4.3).

Заключение

В данном проекте были рассмотрены основополагающие материалы которые в дальнейшем потребовались при реализации алгоритма трассировки лучей. Была рассмотрена схема алгоритма (рис. 2.1). Также был разобран листинг 3.1, показывающие работу изучаемого алгоритма и были приведены рисунки 4.2 и 4.3, показывающие работу алгоритма.

В рамках выполнения работы решены следующие задачи:

1. описана предметная область работы;
2. рассмотрены существующие алгоритмы построения реалистичных изображений;
3. выбран и обоснован выбор реализуемых алгоритмов;
4. подробно изучен выбранный алгоритм;
5. разработана программа;
6. увеличена скорость работы выбранного алгоритма;
7. выбран и обоснован выбор языка программирования, для решения поставленной задачи.

Литература

[1] Visual Studio Code [Электронный ресурс], режим доступа: https://code.visualstudio.com/ (дата обращения: 02.10.2020)

[2] Windows [Электронный ресурс], режим доступа:https://www.microsoft.com/ru-ru/windows/ (дата обращения: 02.10.2020)

[3] Linux [Электронный ресурс], режим доступа:https://www.linux.org.ru/ (дата обращения: 02.10.2020)

[4] Руководство по языку C#[Электронный ресурс], - режим доступа: https://docs.microsoft.com/ru-ru/dotnet/csharp/ (дата обращения: 02.10.2020)

[5] Ubuntu 18.04 [Электронный ресурс], режим доступа:https://releases.ubuntu.com/18.04/ (дата обращения: 02.10.2020)

[6] Дымченко, Лев. Пример реализации в реальном времени метода трассировки лучей: необычные возможности и принцип работы. Оптимизация под SSE [Электронный ресурс], режим доступа:https://www.ixbt.com/video/ rt-raytracing.shtml. (дата обращения: 02.20.2020)

[7] Роджерс, Д. Алгоритмические основы машинной графики / Д. Роджерс. — Москва «Мир», 1989. — Р. 512.

[8] Ю.М.Баяковский. Трассировка лучей из книги Джефа Проузиса [Электронный ресурс], режим доступа: ttps://www.graphicon.ru/oldgr/courses/cg99/notes/lect12/prouzis/raytrace.htm. (дата обращения: 02.20.2020)