Оглавление

Введение **3**

1 Аналитическая часть **5**
1.1 Постановка задачи 5
1.2 Описание предметной области 5
1.3 Алгоритмы удаления невидимых линий 5
1.3.1 Алгоритм Робертса 5
1.3.2 Алгоритм Варнока 6
1.3.3 Алгоритм Вейлера-Азертона 7
1.3.4 Алгоритм Z буфера 7
1.3.5 Выбор алгоритма удаления невидимых граней . . . 8
1.4 Алгоритмы закраски 8
1.4.1 Закраска методом Гуро 8
1.4.2 Закраска Фонга 10
1.4.3 Выбор алгоритма закраски 10
1.5 Модель освещения 10
1.5.1 Локальная модель освещения 10
1.5.2 Глобальная модель освещения 11
1.5.3 Выбор модели осещения 11
1.6 Преобразования 11
1.6.1 Перенос 11
1.6.2 Масштабирование 12
1.6.3 Поворот 13
1.7 Вывод 15

2 Конструкторская часть **16**
2.1 Требования к программе 16
2.2 Полигональная сетка 16

2.2.1 Построение полигональной сетки 17
2.3 Вычисление интенсивности освещения в вершинах треугольника 19
2.4 Реализация метода Гуро с использованием Z-буфера 20
2.5 Выбор используемых типов и структур данных 23
2.6 Вывод 23

Заключение **24**

Список литературы **25**

Введение

В наши дни научные деятели активно изучают космические пространства и объекты. Ученые стремятся максимально реалистично изобразить моделируемые системы, поэтому сегодня важно иметь инструменты для разработки реалистичных трехмерных изображений. На помощь приходит компьютерная графика.

Компьютерная графика - наука, представляющая 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, которое предоставляет визуализацию космических объектов.

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 алгоритмов.
4. Выбрать способ хранения данных о моделируемом объектею.
5. Подробно изучить выбранные алгоритмы.
6. Выбрать среду и язык программирования.
7. Разработать программу на основе существующих алгоритмов.

8. Снизить время работы выбранного алгоритма.

9. Выбрать и обосновать выбор языка программирования, для решения данной задачи.

1 Аналитическая часть

1.1 Постановка задачи

Необходимо реализовать программный продукт, предоставляющий визуализацию системы космических объектов. Также необходимо осуществить выбор метода решения. Пользователь должен иметь возможность запустить и остановить визуализируемую систему. Полученное изображение должно четко показывать работу данной системы.

1.2 Описание предметной области

Компьютерная графика играет важную роль в IT сфере. Она решает множество проблем, акутальных на данный момент. Разработка алгоритмов визуализации трехмерных объектов является обязательной задачей компьютерной графики. К примеру, визуализацию различных систем и процессов для демонстрации определенных явлений.

1.3 Алгоритмы удаления невидимых линий

1.3.1 Алгоритм Робертса

Алгоритм Робертса позволяет произвести удаление невидимых линий при помощи математических вычислений. Алгоритм работает в объектном пространстве с выпуклыми телами. Если имеются невыпуклые тела, необходимо сначала разбить их на выпуклые.

Сначала удаляются ребра и грани, экранируемые самим объектом, а затем линии, экранируемые другими объектами сцены.

К достоинствам алгоритма можно отнести точность и быстроту работы математических методов.

Недостатком метода является то, что вычислительная сложность алгоритма растет как квадрат числа объектов сцены.

Существуют модификации данного алгоритма. При использовании приоритетной сортировки вдоль оси z и простых габаритных или минимаксных тестов, вычислительная сложность линейно зависит от количества объектов. Однако реализация модифицированных алгоритмов достаточно сложна.

1.3.2 Алгоритм Варнока

Алгоритм Варнока работает в пространстве изображений. В пространстве изображений анализируется информация, содержащаяся в окне. Если окно не перекрывается ни одним многоугольником, оно считается пустым. Если окно содержит пустые области и области, покрываемые многоугольниками, то окно разбивается на подокна до тех пор, пока остаются области, содержащие не один многоугольник. Если результирующее окно имеет размер один пиксел, но оно перекрывается несколькими многоугольниками, необходимо сравнить глубину этих многоугольников и высветить пиксел цветом ближайшего к наблюдателю многоугольника.

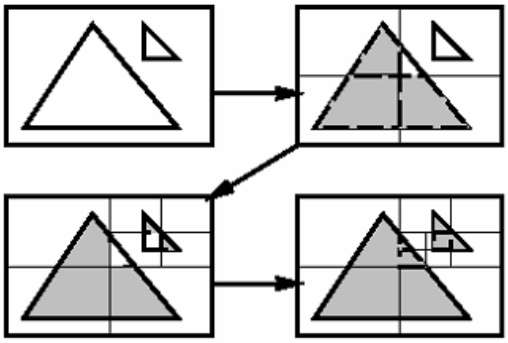

Рис. 1.1 — Пример разбиения Алгоритмом Варнока

Алгоритм достаточно прост с точки зрения понимания, однако может потребоваться большое количество разбиений, соответственно потребуется много времени на анализ и отображение содержимого всего окна.

1.3.3 Алгоритм Вейлера-Азертона

Алгоритм Вейлера-Азертона предназначен для минимизации количества 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 пересекаются, то для корректной работы данного алгоритма нужно плоскость одного разбить другим на две части. Эффективность алгоритма Вейлера-Азертона, как и алгоритма Варнока, зависит от эффективности разбиений. В дальнейшем этот алгоритм был распространен на сплайновые поверхности. К достоинствам алгоритма можно отнести скорость работы, учет когерентности изображения. Недостатками алгоритма является сложность реализации, а также невозможность передачи зеркальных эффектов и преломления света.

1.3.4 Алгоритм Z буфера

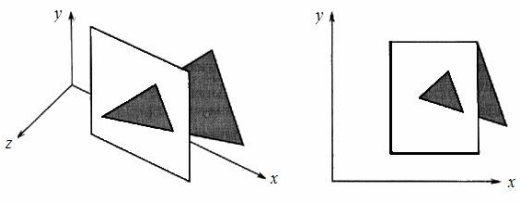

Рис. 1.2 — Пример работы алгоритма Z буфера

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 нужно подготовить буферы. Для этого в Z буфер заносятся максимально возможные значения z, а буфер кадра заполняется значениями пикселя, который описывает фон. Также 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 достоинствам алгоритма, использующего z-буффер можно отнести простоту реализации, а также отсутствие предварительной сортировки элементов сцены. Недостатками алгоритма являются трудоёмкость реализации эффектов прозрачности, а также перерасход по памяти - алгоритм предполагает хранение двух двумерных массивов, размер которых увеличивается с увеличением размеров изображения.

1.3.5 Выбор алгоритма удаления невидимых граней

При реализации программного продукта использован алгоритм Z-буфера. Этот алгоритм позволяет строить сцены любой сложности, а также достаточно быстро работает за счет отсутствия сложных вычислений. Так как на сегодняшний день размер оперативной памяти достаточно велик, использование большого объема памяти не является существенным недостатком применения данного алгоритма в программном продукте. Для увеличения скорости отрисовки каждого объекта треугольники, составляющие объект, отсортированы по оси z по возрастанию (ось Z направлена от наблюдателя).

1.4 Алгоритмы закраски

1.4.1 Закраска методом Гуро

При данном виде закраски интерполируется значение интенсивности, что позволяет получить сглаженное изображение. Данный алгоритм закраски проще, чем алгоритм закраски Фонга. Недостатками метода Гуро является появление полос Маха (алгоритм не обеспечивает непрерыв-

ность изменения интенсивности), а также то, что при линейной интерполяции значение интенсивности на некоторых участках может быть получится равной, и поверхность на этих участках будет выглядеть плоской. Метод основан на идее закрашивания грани не одним цветом, а плавно изменяющимися оттенками. Для вычисления оттенка, интерполируются цвета примыкающих граней. Для реализации метода Гуро необходимо вычислить нормали к каждой грани. Затем, необходимо определить усредненные нормали в вершинах. Для этого усредняются значения нормалей примыкающих граней

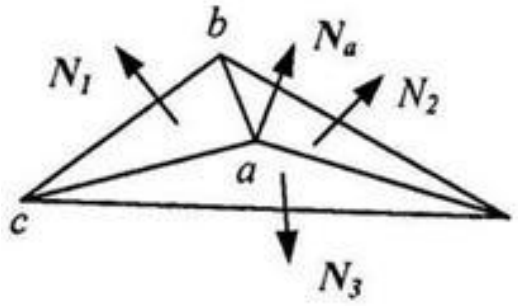

Рис. 1.3

Далее, на основе усредненных нормалей вычисляются интенсивности в вершинах. Грань закрашивается цветом, полученным на основе интерполяции интенсивности в вершинах.

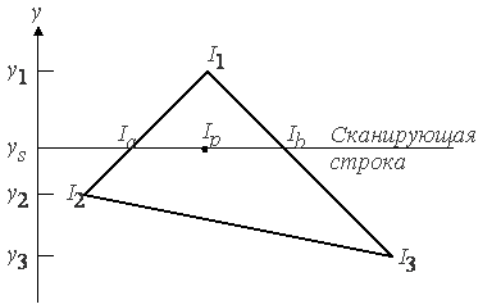

Рис. 1.4

1.4.2 Закраска Фонга

Закраска Фонга сложнее в реализации, однако она позволяет устранить недостатки закраски методом Гуро за счет того, что интерполируются нормали к полигонам, а не значения интенсивности. Это позволяет уменьшить количество полос Маха. Более реалистично выглядят зеркальные блики.

Для реализации метода Фонга необходимо проделать следующие операции. Сначала, определяются нормали к граням. Затем, по нормалям к граням определяются усредненные нормали в вершинах. Далее, с помощью интерполяции вычисляются интерполированные векторы нормали в каждой точке грани. На основе направления нормали определяется интенсивность каждой точки.

1.4.3 Выбор алгоритма закраски

В разрабатываемом программном продукте использована закраска Гуро, так как она требует меньшего количества вычислений, что приведет к увеличению быстродействия программы. К тому же, как показал Дафф, эффект полос Маха проявляется сильнее для сфер при использовании закраски Фонга. Учитывая, что сфера является одним из наиболее часто используемых объектов сцены, целесообразно учесть это замечание и прибегнуть к закраске Гуро. Также, моделируемые объекты имеют матовую структуру, а превосходство метода Фонга над методом Гуро наиболее ярко проявляется при наличии зеркальных бликов матовых поверхностей.

1.5 Модель освещения

1.5.1 Локальная модель освещения

Локальная модель учитывает освещение точечных источников, непосредственно освещающих сцену. Многократное отражение и преломление лучей от различных объектов сцены не учитывается.

1.5.2 Глобальная модель освещения

В глобальной модели освещения отслеживается весь путь луча до тех пор, пока он не покинет сцену или переносимая им энергия не станет достаточно малой, чтобы не учитывать ее. Для этого вводится ограничение количество отражений, поскольку при каждом отражении теряется часть энергии луча. Глобальная модель освещения является более точной, так как точнее передает то, как распространяется свет в реальности. Однако при работе с матовыми объектами, локальная модель освещения позволяет достаточно точно визуализировать рассматриваемые объекты.

1.5.3 Выбор модели осещения

В разрабатываемом продукте используется локальная модель освещения, так как для реализации глобальной модели необходима трассировка лучей, что увеличит количество вычислений и время выполнения программы. К тому же, моделируемые объекты будут матовыми, и при первом отражении энергия луча будет сильно уменьшаться.

1.6 Преобразования

1.6.1 Перенос

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

$$\begin{cases} x_1 = x + dx \\ y_1 = y + dy \end{cases} \tag{1.1}$$

Матрица переноса в двухмерном пространстве:

$$Mmove = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ dx & dy & 1 \end{pmatrix} \tag{1.2}$$

Матрица переноса в трехмерном пространстве:

$$Mmove = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ dx & dy & dz & 1 \end{pmatrix} \tag{1.3}$$

1.6.2 Масштабирование

Масштабирование - изменение размера. Задается коэффициентами масштабирования kx, ky и центром масштабирования xm, ym. Иллюстрация представлена на рисунке 1.5

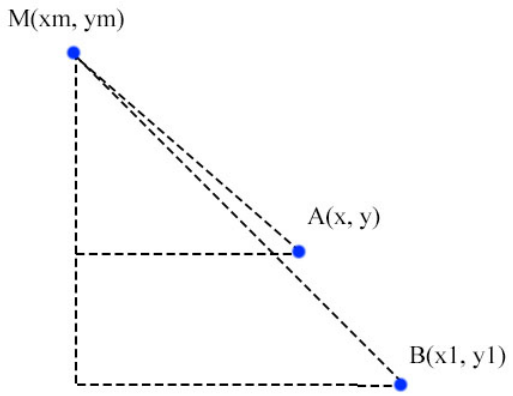

Рис. 1.5 — Перемещение точки

При kx = ky масштабирование однородное.
При 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 к центру и уменьшается.

Из Рис. 1.5 получаем:

$$\begin{cases} x_1 - x_m = kx(x - x_m) \\ y_1 - y_m = ky(y - y_m) \end{cases} \tag{1.4}$$

Упростим (1.4) и получим:

$$\begin{cases} x_1 = kx * x + (1 - kx) * x_m \\ y_1 = ky * y + (1 - ky) * y_m \end{cases} \tag{1.5}$$

Матрица масштабирования в двухмерном пространстве:

$$Mscale = \begin{pmatrix} kx & 0 & 0 \\ 0 & ky & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{1.6}$$

Матрица масштабирования в трехмерном пространстве (добавляется коэффициент масштабирования: kz):

$$Mscale = \begin{pmatrix} kx & 0 & 0 & 0 \\ 0 & ky & 0 & 0 \\ 0 & 0 & kz & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{1.7}$$

1.6.3 Поворот

Для поворота потребуется угол θ и центр C(xc, yc) (рис. 1.6).
Поворот против часовой стрелки:

$$\begin{gathered} x_1 = x_c + R * cos(180^\circ - (\varphi + \theta)) = \\ x_c - R * cos(\varphi) * cos(\theta) + R * sin(\varphi) * sin(\theta) = \\ x_c - (x_c - x)cos(\theta) + (y - y_c)sin(\theta) = \\ x_c + (x - x_c)cos(\theta) + (y - y_c)sin(\theta) \end{gathered} \tag{1.8}$$

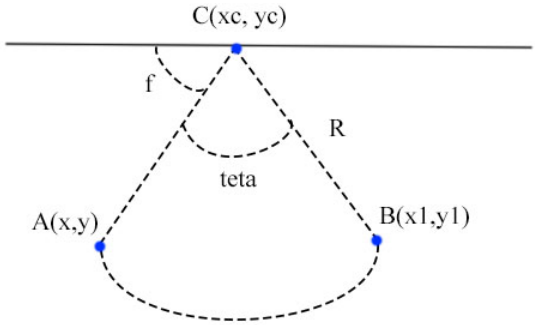

Рис. 1.6 — Поворот точки

$$
\begin{gathered}
y_1 = y_c + R * sin(180^\circ - (\varphi + \theta)) = \\
y_c + R * sin(\varphi) * cos(\theta) + R * cos(\varphi) * sin(\theta) = \\
y_c + (y - y_c)cos(\theta) + (x_c - x)sin(\theta) = \\
y_c + (y - y_c)cos(\theta) - (x - x_c)sin(\theta)
\end{gathered} \tag{1.9}
$$

Матрица поворота против часовой стрелки в двухмерном пространстве:

$$
Mrotate = \begin{pmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{1.10}
$$

Матрица поворота в трехмерном пространстве вокруг оси Z :

$$
Mrotate = \begin{pmatrix} cos(\theta) & -sin(\theta) & 0 & 0 \\ sin(\theta) & cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{1.11}
$$

1.7 Вывод

Оценив все изложенные выше алгоритмы для реализации программного обеспечения, можно сделать вывод, что для данной работы подходит следующий набор алгоритмов: закраска Гуро совместно с z-буфером, локальная модель освещения.

2 | Конструкторская часть

2.1 Требования к программе

Программа должна предоставлять следующие возможности.

1. Визуализацию сцены.
2. Запуск системы.
3. Останов системы.

2.2 Полигональная сетка

Для работы с трехмерными объектами, удобно использовать полигональную сетку. Это позволяет получить реалистичное изображение, показать объем тела. В разрабатываемом программном продукте использована треугольная полигональная сетка. Выбор треугольной полигональной сетки обосновывается следующими причинами:

1. Разбиение тел на треугольники упрощает рендеринг, так как треугольник является выпуклой фигурой, а многие алгоритмы работают именно с выпуклыми полигонами
2. Любой многоугольник может быть разбит на некоторое количество треугольников
3. Невырожденный треугольник всегда расположен в одной плоскости, к нему легко восстановить нормаль

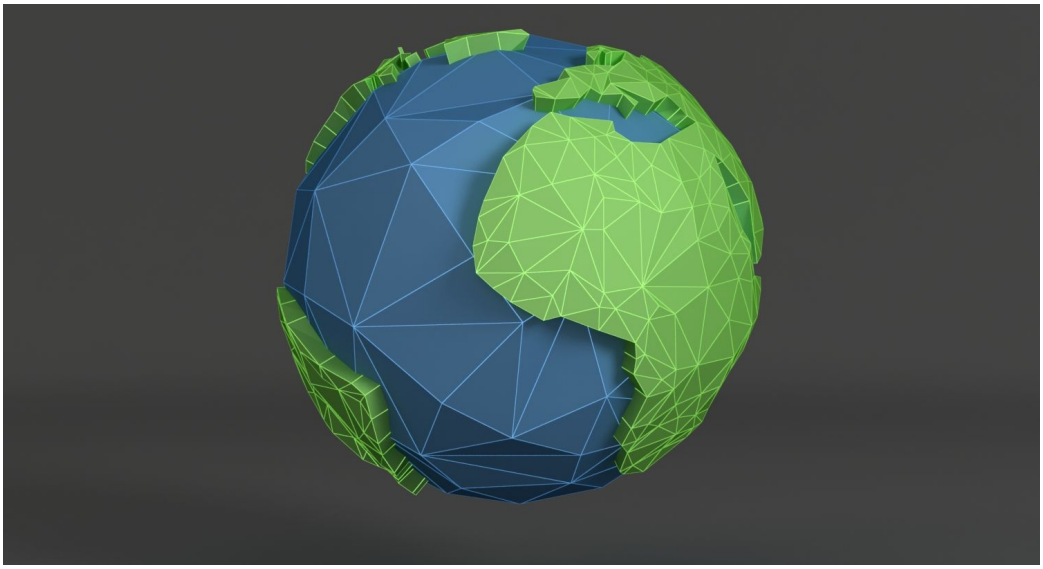

Рис. 2.1 — Пример полигональной сетки, изображающей планету

2.2.1 Построение полигональной сетки

Для построения полигональной сетки тел вращения используется следующий алгоритм. Задается количество разбиений основания тела вращения и количество разбиений по высоте. Затем вычисляется угол, на который необходимо каждый раз поворачивать радиус-вектор основания, чтобы получить необходимое количество треугольников:

$$angle = \frac{2 * \pi}{n * \phi}, \tag{2.1}$$

где n * ϕ - число секторов на которые делится основание.

Основание тела при его создании лежит в плоскости xOz, поэтому координаты точек окружности, которые послужат вершинами треугольников, вычисляются следующим образом:

$$\begin{gathered} x = x_0 + R * \cos(angle * i) \\ y = y_0 \\ z = z_0 + R * \sin(angle * i) \end{gathered} \quad , \tag{2.2}$$

где R - радиус основания вращения, i - номер итерации цикла, на котоом находится очередная точка.

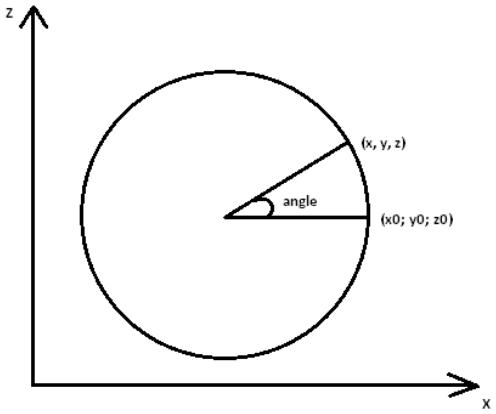

Рис. 2.2

Затем, зная зависимость радиуса от высоты, аналогичным способом проставляются точки на уровнях тела вращения с шагом:

$$dh = \frac{yTop - yBottom}{heightSegments} \tag{2.3}$$

где yTop - координата Y центра верхнего основания, yBottom - координата Y центра нижнего основания, heightSegments - число разбиений тела по высоте.

Затем все полученные точки соединяются в четырехугольники. Точка соединяется с соседней сверху, соседней справа и точкой, которая лежит выше по диагонали справа. Таким образом, тело оказывается разделенным на четырехугольники. Затем в каждом четырехугольнике проводится диагональ, и тело оказывается покрытым треугольной полигональной сеткой.

Этот алгоритм подходит для триангуляции цилиндра, конуса, усеченного конуса.

Чтобы разбить на треугольники сферу, необходимо разбить ее на усеченные конусы, а затем каждый из конусов триангулировать по описанному выше алгоритму.

2.3 Вычисление интенсивности освещения в вершинах треугольника

Для вычисления интенсивностей в вершинах треугольника сначала были вычислены нормали к его вершинам с учетом нормалей прилегающих к нему треугольников.

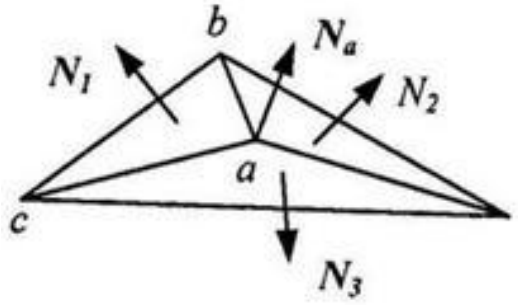

Рис. 2.3 — Нахождение нормалей к вершинам треугольника

$$N_a = \frac{N_1 + N_2 + N_3}{3} \tag{2.4}$$

Затем были найдены векторы от источника освещения к вершинам треугольника и углы между этими векторами и нормалями к каждой из вершин. Далее, интенсивность вычислялась по формуле:

$$I[i] = angleCos[i] * source.GetI() + constLight, \tag{2.5}$$

где I[i] – интенсивность в i-й вершине треугольника, angleCos[i] – косинус угла между вектором от источника освещения к i-й вершине и нормалью к этой вершине, source.GetI() – интенсивность источника освещения, constLight - интенсивность рассеянного света, подобранная опытным путем.

Если в результате вычисления получалась интенсивность меньшая, чем constLight или большая, чем 1, значение корректировалось до constLight или 1 соответственно.

2.4 Реализация метода Гуро с использованием Z-буфера

Так как все тела, используемые в программе, были триангулированы, алгоритм закраски Гуро реализован для закраски треугольника. Для решения задачи удаления невидимых линий перед закраской координата Z каждого пиксела сравнивается со значением в буфере глубины, после этого принимается решение о необходимости закрашивать данный пиксел.

Для того чтобы при отрисовке выводить меньше пикселей, которые в дальнейшем будут перекрашены, было принято решение отсортировать треугольники по возрастанию по координате Z. Критерием сортировки является среднее значение координаты Z трех вершин треугольника. Такой подход позволяет сначала вывести на канву большую часть видимых треугольников, а затем при проверке не отрисовывать невидимые части треугольников, лежащих дальше от наблюдателя. Так как операция высвечивания пиксела занимает довольно много времени, такой подход должен ускорить процесс получения изображения.

Предварительно вершины треугольника сортируются по возрастанию их координат Y.

Поясняющий рисунок, на котором обозначены используемые в алгоритме обозначения:

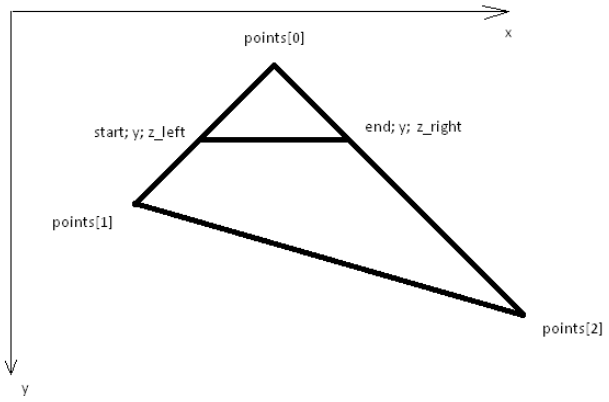

Рис. 2.4 — Поясняющий рисунок, на котором обозначены используемые в алгоритме обозначения:

Ниже представлен алгоритм закраски.

1. Вычислить интенсивности в вершинах треугольника.

2. Рассчитать изменение координаты Z в зависимости от Y.

3. Цикл по сканирующим строкам от вершины с минимальным значение Y до вершины со средним значением Y.

4. Определить координаты X левой и правой точек пересечения сканирующей строки со сторонами треугольника.

5. Вычислить интенсивность в точках, найденных в пункте 4. Расчет для левой точки выполняется по следующей формуле:

$$I = (1 - (\frac{len2}{len1})) * points[0].I + (\frac{len2}{len1}) * points[1].I, \tag{2.6}$$

   где len2 – расстояние по Y от сканирующей строки до вершины с минимальным Y, len1 – длина стороны, на которой лежит точка, points[0].I – интенсивность в вершине с минимальным Y, points[1].I - интенсивность во второй вершине этой же стороны. Аналогичным образом рассчитывается интенсивность в правой точке сканирующей строки.

6. Вычислить координаты Z крайних точек сканирующей строки по формулам:

$$\begin{aligned} z_{left} &= point1.getZ() + dz_{12} * (y - point1.getY()); \\ z_{right} &= point1.getZ() + dz_{13} * (y - point1.getY()); \end{aligned} \tag{2.7}$$

   z_{left} и z_{right} – координаты Z левой и правой точек пересечения сканирующей строки со сторонами треугольника, point1 - вершина с минимальным Y, dz_{12} и dz_{13} – изменение координат Z при смещение на 1 по Y для сторон треугольника, соединяющих вершины с минимальным и средним значением Y и вершины с минимальным и максимальным значением Y соответственно.

7. Вычислить изменение координаты Z при смещении по X по сканирующей строке по формуле:

$$dzx = \frac{z_{right} - z_{left}}{end - start}, \tag{2.8}$$

где dzx – изменение координаты Z, z_{right} и end – координаты Z и X правой крайней точки сканирующей строки, z_{left} и start – координаты Z и X левой крайней точки сканирующей строки.

8. Цикл по X от start до end по сканирующей строке.

9. Проверить, не лежат ли координаты X и Y рассматриваемой точки за пределами канвы.

10. Рассчитать координату Z рассматриваемой точки:

$$z = z_{left} + dzx * (x - start), \tag{2.9}$$

где x и z – координаты точки, рассматриваемой на данной итерации.

11. Сравнить полученную координату Z со значением в буфере глубины. Если найденная координата Z меньше значения в буфере глубины, то

12. Поместить координату Z в буфер глубины.

13. Вычислить параметр изменения интенсивности:

$$iP = (1 - \frac{lenQP}{lenQR}) * iQ + \frac{lenQP}{lenQR} * iR \tag{2.10}$$

где iP – вычисляемый коэффициент, iQ и iR – значения интенсивностей в левой и правой точках сканирующей строки соответсвенно, lenQP и lenQR – расстояния от рассматриваемой точки до левой и правой точек сканирующей строки соответственно.

14. Занести в буфер цвета пиксела с его базовым цветом, каждая компонента которого умножена на полученный коэффициент iP.

15. Конец условия п.11

16. Конец цикла п.8
17. Конец цикла п.3
18. Провести аналогичные действия для части треугольника, лежащей между вершиной со средним и максимальным значениями координаты Y.

2.5 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. Планета – хранит радиус, цвет и центр планеты.
5. Ракета – вектор из координат, описывающих ракету.
6. Сцена – список объектов, заданных планетой и ракетой.
7. Источник света – положение и направление света.

2.6 Вывод

В данном разделе были описаны требования к программе, подробно рассмотрены алгоритмы z-буффера и закраски Гуро, описаны типы и структуры данных, которые будут реализованы. Также рассмотрен способ вычисления интенсивности освещения и построения полигональной сетки.

Заключение

Во время выполнения поставленной задачи были рассмотрены и проанализированны основные алгоритмы удаления невидимых линий и методы закрашивания. Были проанализированны их достоинства и недостатки, выбраны наиболее подходящие для решения поставленной задачи. 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 изучены выбранные алгоритмы.

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] Stopwatch Класс, [Электронный ресурс], режим доступа: https://docs.microsoft.com/ru-ru/dotnet/api/system.diagnostics.stopwatch?view=net-5.0 (дата обращения: 12.29.2020)

[7] 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)

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

[9] 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)