МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

ОТЧЕТ

по лабораторной работе № 5

по дисциплине «Компьютерная графика»

Тема: Исследование алгоритмов выявления видимости сложных сцен

Студентка гр. 1361 _____ Галунина Е.С.

Студентка гр. 1361 _____ Горбунова Д.А.

Студентка гр. 1361 _____ Токарева У.В.

Преподаватель _____ Колев Г.Ю.

Санкт-Петербург

2023

Цель работы.

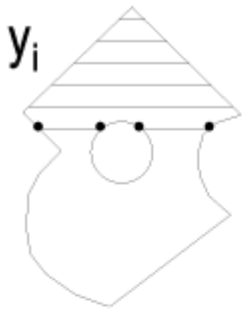
Обеспечить реализацию видимости совокупности произвольных многогранников на основе алгоритма построчного сканирования.

Основные теоретические положения.

Основные теоретические сведения представлены на рисунках 1, 2, 3, 4

Алгоритм выявления видимости сложной сцены путем построчного сканирования изображения.

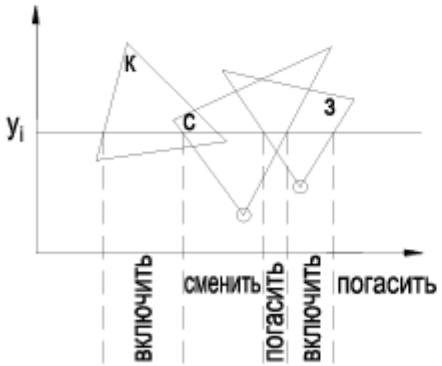
В основе алгоритма лежит алгоритм разложения многоугольника в растр:



Этапы:

1. При определенных значениях Yi определяются точки пересечение с ребрами многоугольников, соответствующих граням объектов визуализации.
2. Выполняется сортировка найденных точек пересечения в порядке возрастания (или убывания) по координате X.
3. Обеспечивается закраска (переключение цвета луча развертки в соответствии с цветом “Ц” соответствующей грани) вдоль горизонтальной линии. По первой точке включается луч, по второй точке он либо выключается, либо изменяется (переключается) его цвет и так до конца сканируемой строки. Изначально он выключен. Затем осуществляется переход на следующую сканирующую строку.

Таким образом, осуществляется построчная закраска экрана в соответствии с обрабатываемой сценой.



Идея алгоритма: При развертке луча для конкретных значений Yi и Xj необходимо определить какой многоугольник становится активным, т.е. в какой цвет (какое значения параметра “Ц” активно) следует закрашивать разворачивающийся луч (при следующем многоугольнике надо луч либо погасить, либо сменить его цвет, либо оставить прежний цвет).

Для реализации идеи:

Рисунок 1 – Теоретические сведения

– на первом подготовительном этапе работы алгоритма создаются таблицы многоугольников ТМ и ребер ТР.

В таблицу многоугольников ТМ заносятся параметры (A, B, C, D) уравнения плоскости, определяемые на основе координат вершин каждой грани, и цветность соответствующей грани (многоугольника) – ПЭЛ цветности (Ц):

Ax+By+Cz+D=0 – уравнение плоскости.

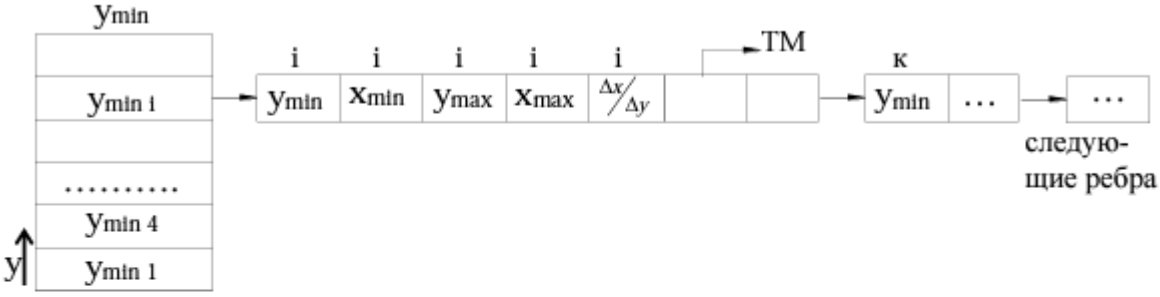
В таблицу ребер ТР заносятся параметры всех негоризонтальных ребер, т.е. следующая информация:

$y_{\min}$; $x_{\min}$; $y_{\max}$; $tg\alpha = \dfrac{\Delta x}{\Delta y}$,где $tg\alpha$ - угол наклона ребра к оси абсцисс (y).

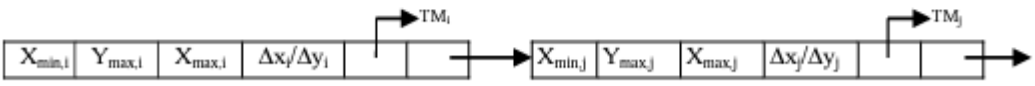
Для каждого ребра в ТР есть ссылка к какому многоугольнику (грани) в таблице многоугольников ТМ эти ребра относятся (т.е. указатель отношения к многоугольнику, т.е. к его плоскости и цвету).

– на втором подготовительном этапе на основе ТР обеспечивается групповая сортировка ребер, во время которой все ребра сортируются по возрастанию (или убыванию) их Ymin и объединяются в таблицу групповой сортировки ТГС. При одинаковых значениях Ymin ребра дополнительно сортируются по убыванию $tg\alpha$. Такая группировка выполняется в рамках всего растрового диапазона дисплея (содержит столько элементов сколько строк в нем по Y, если используется горизонтальная развертка).

Пример групповой сортировки ребер:

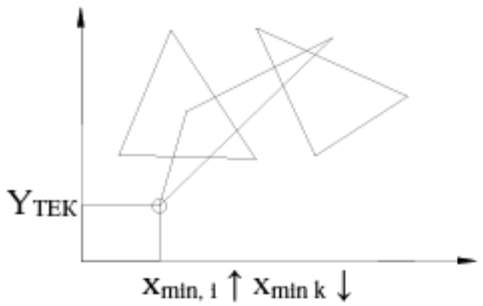


На следующем этапе работы алгоритма устанавливается $Y_{TEK} = Y_{\min i}$, соответствующее минимальной значимой строке таблицы групповой сортировки, и на ее основе формируется таблица активных ребер ТАР, в которую переносятся параметры ребер, соответствующих Yтек:



Далее начинается обработка элементов ТАР .

Рисунок 2 – Теоретические сведения



1. В ТАР считываются параметры первого ребра, определяются соответствующий ребру i номер многоугольника в ТМ и его цвет ($Ц_i$=1), поднимается флаг этого многоугольника и в соответствии с этими данными определяется цвет сканирующего луча – луч подсвечивается соответствующим цветом при значении $x_{min,i}$. Количество поднятых флагов N (активных могоугольников) устанавливается равным 1 (N=1).
2. Считываются параметры второго ребра j в ТАР и из ТМ определяется его номер.

   Если номер следующего многоугольника равен номеру предыдущего, то при значении $x_{min,j}$ флаг его сбрасывается, цвет луча в этот момент подавляется (луч гасится), а число N принимает значение 0 (N=0).

   Если номер следующего многоугольника отличен от номера предыдущего, то при значении X = $x_{min,j}$ (для очередного ребра) поднимается флаг его многоугольника, число учитываемых (активных) многоугольников увеличивается на 1 (N = N + 1). В этот момент необходимо определить, какой многоугольник ближе к наблюдателю (для какого из них $Z_{i,j}$ имеет большее значение при значении X = $x_{min,j}$) – для старого активного или нового (подключенного):

   если $Z_{i,j}^{(с)} > Z_{i,j}^{(н)}$, то значение цвета луча не меняется,

   иначе (если $Z_{i,j}^{(с)} < Z_{i,j}^{(н)}$), то цвет луча развертки изменяется на цвет, соответствующий окраске нового активного многоугольника ($Ц=Ц^{(н)}$).

   Последняя ситуация возникает в случаях, при которых в ТАР находится более двух ребер, относящихся к двум или более многоугольникам.

   Тогда, если, например, имеется 4 ребра (при двух активных многоугольниках), то при пересечении сканирующим лучем третьего ребра:

   – считываются параметры этого третьего ребра k в ТАР и из ТМ определяется его номер:

   – если это ребро относится к многоугольнику, к которому относится предыдущее ребро (номера многоугольников одинаковы), то:

   – при значении $x_{min,k}$ флаг его сбрасывается, но остается

Рисунок 3 – Теоретические сведения

поднятым флаг 1-го многоугольника. Число активных многоугольников N уменьшается на 1 и принимает значение 1 (N=1). Цвет луча второго многоугольника в этот момент подавляется, но луч не гасится, а переключается на цвет, соответствующий 1-му многоугольнику;

– если номер следующего многоугольника равен номеру первого, а не второго, то:

– при значении $x_{min,k}$ флаг первого многоугольника сбрасывается, но остается поднятым флаг 2-го многоугольника. Цвет луча остается прежним, так как он соответствует второму многоугольнику. Число активных многоугольников N уменьшается на 1 и принимает значение 1 (N=1).

Аналогичные действия должны выполняться и в случаях, когда активных многоугольников может быть больше двух.

При достижении сканирующим лучем последнего ребра активного многоугольника в ТАР его флаг сбрасывается, число активных многоугольников становится равным 0 (N=0) и цвет луча подавляется.

На этом завершается сканирование информации в ТАР и визуализация строки изображения, соответствующей значению Y_{TEK} .

3. Осуществляется изменение Y_{TEK} :

$$Y_{TEK} = Y_{TEK} + 1$$

4. Если $Y_{TEK} > Y_{\max}^{(k)}\big|_{TAP}$, то такое ребро выкидывается из ТАР, т.к. оно уже полностью обработано.
5. Корректируется $x_{\min}$ на величину Δx/Δy.
6. Проверяется таблица групповой сортировки для значения Y_{TEK} :

– если $Y_{TEK} > Y_{\min}^{(\max)}$ (больше последней значащей строки), то следует перейти на п.7,

– если Y_{TEK} соответствует строке таблицы групповой сортировки с подключенными к ней ребрами, то их описание переносится в ТАР.

7. Если в ТАР число ребер не меньше двух:

– то осуществляется сортировка этих ребер по возрастанию $x_{\min}$,

– иначе работа алгоритма завершается.

8. Осуществляют переход на п.1

Рисунок 4 – Теоретические сведения

Формализация.

Работа выполнена на языке программирования Python. Для визуализации данных трехмерной графики используется библиотека matplotlib.

Экспериментальные результаты.

На рисунке 5 представлен результат работы программы

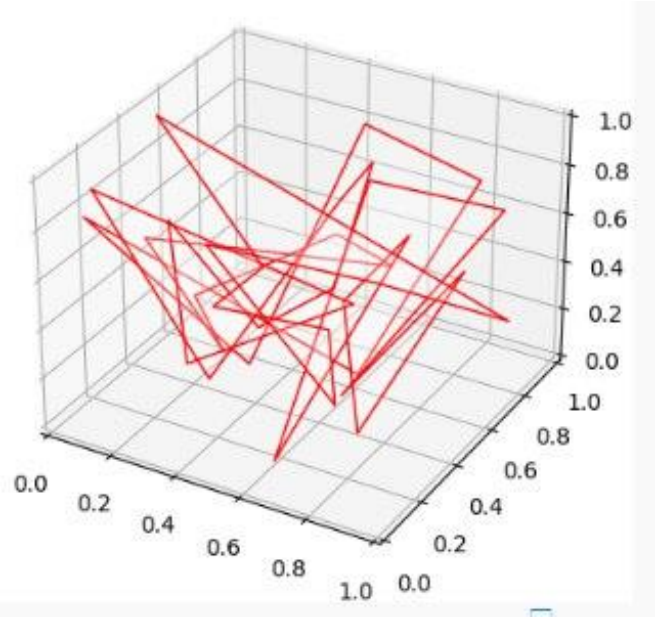


Рисунок 5 – Результат эксперимента

Исходный код программы

```
import numpy as npimport matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.art3d import Poly3DCollection# Функция для генерации многоугольников
def generate_polygons(n, min_vertices=3, max_vertices=10):  polygons = []
    for _ in range(n):  num_vertices = np.random.randint(min_vertices, max_vertices+1)
        vertices = np.random.rand(num_vertices, 3) # генерируем 3D вершины
polygons.append(vertices)
    return polygons# Функция для определения пересечения между многоугольником и сканирующей строкой
def find_intersections(polygon, y):
intersections = []
    for i in range(len(polygon)):  p1 = polygon[i]
```

```
            p2 = polygon[(i+1) % len(polygon)]
if min(p1[1], p2[1]) <= y <= max(p1[1], p2[1]):
                x = p1[0] + (p2[0] - p1[0]) * (y -
p1[1]) / (p2[1] - p1[1])
intersections.append(x)
        intersections.sort()    return intersections
    #Функция для вычисления глубину (z-координату)
пикселядef compute_z(polygon, x, y):
        return polygon[0][2]# Функция для визуализации
многоугольников с использованием z-буфера
    def visualize_polygons(polygons, screen_width,
screen_height):     # Инициализация z-буфера и буфера
кадра
        z_buffer = np.full((screen_width,
screen_height), np.inf)             frame_buffer =
np.zeros((screen_width, screen_height, 3))
        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')
        for polygon in polygons:            for y in
range(screen_height):        # Обход каждой строки
изображения
                intersections =
find_intersections(polygon, y)    # Находим пересечения
многоугольника со сканирующей строкой                if
intersections:  # Если пересечения существуют
                    for x in range(intersections[0],
intersections[1]):     # Обход каждого пикселя между
парой пересечений                                z =
compute_z(polygon, x, y)   # Вычисляем глубину пикселя
```

```
                        if z < z_buffer[x, y]:  # Если глубина пикселя меньше значения в z-буфере
z_buffer[x, y] = z  # Обновляем значение в z-буфере
                            frame_buffer[x, y] = polygon.color  # Заносим цвет многоугольника в буфер кадра
        # Визуализация многоугольника
            poly3d = [polygon.tolist()]  # Исправленная строка
ax.add_collection3d(Poly3DCollection(poly3d, facecolors='white', linewidths=1, edgecolors='r', alpha=.25))
        plt.show()
    # Генерация 5 многоугольниковpolygons = generate_polygons(5)
    # Визуализация многоугольниковvisualize_polygons(polygons, 800, 600)
```

Выводы.

В результате выполнения работы нами был реализован код алгоритма реализации видимости совокупности произвольных многогранников на основе алгоритма построчного сканирования. на языке Python.