**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

**отчет**

**по лабораторной работе №5**

**по дисциплине «Компьютерная графика»**

**Тема: Исследование алгоритмов выявления видимости сложных сцен**

**Вариант 1**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 7301 |  | Гарцев Е.А.  Литвинов К.Л. |
| Преподаватель |  | Матвеева И. В. |

Санкт-Петербург

2020 г.

**Цель работы**

Научиться корректно выявлять видимость сложных сцен.

**Задание**

Обеспечить реализацию алгоритма выявления видимых граней и ребер для одиночного выпуклого объемного тела.

**Математическая модель**

У нас существует выпуклое объёмное тело, в данном случае куб, и две точки: внутренняя точка куба О, являющаяся его центром, и точка камеры Н. Для того, чтобы понять, какие грани являются видимыми, проделываются следующие действия.

Для каждой грани куба составляется функция, обозначающая определитель матрицы плоскости в виде:

Где x1-3, y1-3, z1-3 – координаты точки грани, а x,y,z – координаты проверяемой точки. Если во время подстановки значений определитель данной матрицы будет равен нулю, то получается, что проверяемая точка находится на плоскости грани.

Для того, чтобы узнать, является ли грань видимой, сравнивают две функции: с внутренней точкой куба f(x0,y0,z0) = f(O) и с точкой камеры f(xh,yh,zh) = f(H). Мы можем рассмотреть две ситуации:

1) f(O) \* f(H) < 0 – данная грань является видимой

2) f(O) \* f(H) ≥ 0 – данная грань является невидимой

Равенство нулю означает, что плоскость выстраивается в линию.

**Ход работы**

Данная работа выполнялась на языке Python с использованием PyQT для реализации интерфейса, PyOpenGL для удобного отображения сцены, а также системных встроенных библиотек sys, design, math и numpy.

В программе реализовано следующее:

1. Возможность отобразить все грани многогранника (отображение идёт с учётом того, какие части граней могут быть видны благодарю принципу «что по порядку отображается позже, то и видит пользователь»).
2. Возможность показать только видимые грани куба.

При выборе всех возможностей пользователю будет виден куб с правильным отображением всех видимых и невидимых граней.

**Выводы**

В процессе выполнения данной лабораторной работы был получен опыт корректного отображения видимости сложных сцен.

**Листинг кода**

*Файл lab5.py*

import sys

from OpenGL import GL, GLU

from PyQt5.QtWidgets import QApplication, QMainWindow, QPushButton, QMessageBox

from PyQt5.QtGui import QIcon

from PyQt5.QtCore import pyqtSlot

import design

import math

from numpy import matmul

import openglwidget

class App(QMainWindow, design.Ui\_MainWindow):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.setupUi(self)

        self.openGLWidget

        self.edgesCheckBox.stateChanged.connect(self.change)

        self.visibilityCheckBox.stateChanged.connect(self.change)

    def change(self):

        if self.edgesCheckBox.isChecked():

            openglwidget.flag = True

        else:

            openglwidget.flag = False

        if self.visibilityCheckBox.isChecked():

            openglwidget.flag2 = True

        else:

            openglwidget.flag2 = False

        self.openGLWidget.update()

if \_\_name\_\_ == '\_\_main\_\_':

    app = QApplication(sys.argv)

    ex = App()

    ex.show()

    sys.exit(app.exec\_())

*Файл cube.py*

from OpenGL.GL import \*

from random import random

from numpy import linalg

class Point():

    def \_\_init\_\_(self, x, y, z):

        self.x = x

        self.y = y

        self.z = z

class Lines():

    def \_\_init\_\_(self, start, end):

        self.start = start

        self.end = end

class Surfaces():

    def \_\_init\_\_(self, line1, line2, line3, line4, point1, point2, point3, point4):

        self.lines = [line1, line2, line3, line4]

        self.points = [point1, point2, point3, point4]

        self.visible = True

class Cube():

    def \_\_init\_\_(self):

        a = Point(1, 1, 1) # 0

        b = Point(3, 1, 1) # 1

        c = Point(3, 3, 1) # 2

        d = Point(1, 3, 1) # 3

        e = Point(3, 1, 3) # 4

        f = Point(1, 1, 3) # 5

        k = Point(1, 3, 3) # 6

        j = Point(3, 3, 3) # 7

        self.verticies = [a, b, c, d, e, f, k, j]

        self.eternalPoint = Point(2, 2, 2)

        self.cameraPoint = Point(7, 7, 7)

        line1 = Lines(a, b) # 0

        line2 = Lines(a, d) # 1

        line3 = Lines(a, f) # 2

        line4 = Lines(c, b) # 3

        line5 = Lines(c, d) # 4

        line6 = Lines(c, j) # 5

        line7 = Lines(k, d) # 6

        line8 = Lines(k, f) # 7

        line9 = Lines(k, j) # 8

        line10 = Lines(e, f) # 9

        line11 = Lines(e, b) # 10

        line12 = Lines(e, j) # 11

        self.Lines = [

            line1,

            line2,

            line3,

            line4,

            line5,

            line6,

            line7,

            line8,

            line9,

            line10,

            line11,

            line12

        ]

        self.surfaces = [

            Surfaces(line1, line6, line3, line7, a, f, k, d),

            Surfaces(line6, line7, line9, line5, k, d, c, j),

            Surfaces(line6, line4, line12, line11, e, b, c, j),

            Surfaces(line1, line3, line11, line10, a, b, e, f),

            Surfaces(line1, line2, line5, line4, a, b, c, d),

            Surfaces(line10, line12, line8, line9, e, f, k, j),

        ]

    def draw\_cube(self, flag, flag2):

        if flag == True:

            glBegin(GL\_QUADS)

            for surface in self.surfaces:

                if surface.visible or flag2 == False:

                    glColor3f(random(), random(), random())

                    for point in surface.points:

                        glVertex3d(point.x, point.y, point.z)

            glEnd()

        else:

            for surface in self.surfaces:

                glBegin(GL\_LINES)

                if surface.visible or flag2 == False:

                    for line in surface.lines:

                        glVertex3d(line.start.x, line.start.y, line.start.z)

                        glVertex3d(line.end.x, line.end.y, line.end.z)

                glEnd()

    def determineVisibility(self):

        for surface in self.surfaces:

            line1 = surface.lines[0]

            line2 = surface.lines[1]

            point1 = line1.start

            point2 = line1.end

            point3 = line2.end

            x1 = point1.y

            y1 = point1.z

            z1 = point1.x

            x2 = point2.y

            y2 = point2.z

            z2 = point2.x

            x3 = point3.y

            y3 = point3.z

            z3 = point3.x

            matrixStart = [

                [self.eternalPoint.x - point1.x, self.eternalPoint.y - point1.y, self.eternalPoint.z - point1.z],

                [point2.x - point1.x, point2.y - point1.y, point2.z - point1.z],

                [point3.x - point1.x, point3.y - point1.y, point3.z - point1.z]

            ]

            matrixCamera =  [

                [self.cameraPoint.x - point1.x, self.cameraPoint.y - point1.y, self.cameraPoint.z - point1.z],

                [point2.x - point1.x, point2.y - point1.y, point2.z - point1.z],

                [point3.x - point1.x, point3.y - point1.y, point3.z - point1.z]

            ]

            detStart = linalg.det(matrixStart)

            detCamera = linalg.det(matrixCamera)

            if detStart\*detCamera >= 0:

                surface.visible = False

*Файл design.py*

# -\*- coding: utf-8 -\*-

# Form implementation generated from reading ui file 'design.ui'

#

# Created by: PyQt5 UI code generator 5.11.3

#

# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui\_MainWindow(object):

    def setupUi(self, MainWindow):

        MainWindow.setObjectName("MainWindow")

        MainWindow.resize(800, 600)

        self.centralwidget = QtWidgets.QWidget(MainWindow)

        self.centralwidget.setObjectName("centralwidget")

        self.openGLWidget = OpenGLWidget(self.centralwidget)

        self.openGLWidget.setGeometry(QtCore.QRect(90, 50, 451, 471))

        self.openGLWidget.setObjectName("openGLWidget")

        self.edgesCheckBox = QtWidgets.QCheckBox(self.centralwidget)

        self.edgesCheckBox.setGeometry(QtCore.QRect(590, 110, 131, 17))

        self.edgesCheckBox.setObjectName("edgesCheckBox")

        self.visibilityCheckBox = QtWidgets.QCheckBox(self.centralwidget)

        self.visibilityCheckBox.setGeometry(QtCore.QRect(590, 140, 201, 17))

        self.visibilityCheckBox.setObjectName("visibilityCheckBox")

        MainWindow.setCentralWidget(self.centralwidget)

        self.menubar = QtWidgets.QMenuBar(MainWindow)

        self.menubar.setGeometry(QtCore.QRect(0, 0, 800, 21))

        self.menubar.setObjectName("menubar")

        MainWindow.setMenuBar(self.menubar)

        self.statusbar = QtWidgets.QStatusBar(MainWindow)

        self.statusbar.setObjectName("statusbar")

        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)

        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):

        \_translate = QtCore.QCoreApplication.translate

        MainWindow.setWindowTitle(\_translate("MainWindow", "MainWindow"))

        self.edgesCheckBox.setText(\_translate("MainWindow", "Отображать грани"))

        self.visibilityCheckBox.setText(\_translate("MainWindow", "Показывать только видимые грани"))

from openglwidget import OpenGLWidget

if \_\_name\_\_ == "\_\_main\_\_":

    import sys

    app = QtWidgets.QApplication(sys.argv)

    MainWindow = QtWidgets.QMainWindow()

    ui = Ui\_MainWindow()

    ui.setupUi(MainWindow)

    MainWindow.show()

    sys.exit(app.exec\_())

*Файл openglwidget.py*

from PyQt5 import QtWidgets

from OpenGL.GL import \*

from OpenGL.GLU import \*

from OpenGL.GLUT import \*

from math import sqrt

from cube import \*

flag = False

flag2 = False

class OpenGLWidget(QtWidgets.QOpenGLWidget):

    def initializeGL(self):

        glClearDepth(1.0)

        # glDepthFunc(GL\_LESS)

        # glEnable(GL\_DEPTH\_TEST)

        glMatrixMode(GL\_PROJECTION)

        glLoadIdentity()

        self.cube = Cube()

    def paintGL(self):

        # Настройка камеры для изометрического вида

        glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT)

        glMatrixMode(GL\_PROJECTION)

        glLoadIdentity()

        dist = sqrt(1 / 3.0)

        gluPerspective(90, 491/461, 0.1, 70.0)

        glTranslatef(0, 0, -6)

        gluLookAt(dist, dist, dist, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0)

        glMatrixMode(GL\_MODELVIEW)

        # Рисуем координатные оси

        glBegin(GL\_LINES)

        glColor3d(1.0, 0.0, 0.0)

        glVertex3d(0.0, 0.0, 0.0)

        glVertex3d(3.0, 0.0, 0.0)

        glColor3d(0.0, 3.0, 0.0)

        glVertex3d(0.0, 0.0, 0.0)

        glVertex3d(0.0, 3.0, 0.0)

        glColor3d(0.0, 0.0, 1.0)

        glVertex3d(0.0, 0.0, 0.0)

        glVertex3d(0.0, 0.0, 3.0)

        # Ось до камеры

        glColor3d(0.5, 1.0, 0.5)

        glVertex3d(0.0, 0.0, 0.0)

        glVertex3d(dist+6, dist+6, dist+6)

        glEnd()

        # Рисуем куб

        glColor3d(0, 1.0, 1.0)

        glColor3f(1,1,0)

        self.cube.determineVisibility()

        self.cube.draw\_cube(flag, flag2)

        glFlush()