МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
**«Балтийский государственный технический университет «ВОЕНМЕХ» им. Д.Ф. Устинова»**
**(БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова»)**
БГТУ.СМК-Ф-4.2-К5-01

Факультет О Информационные и управляющие системы
шифр наименование
Кафедра О7 Информационные системы и программная инженерия
шифр наименование
Дисциплина Компьютерная геометрия и графика

# КУРСОВАЯ РАБОТА

НА ТЕМУ:

# Игра «Pacman»

Выполнил студент группы И596
Орехов Руслан Вячеславович
Фамилия И.О.

Снежко Е.А.
Фамилия И.О. Подпись

Оценка
«_____» 2022г.

САНКТ-ПЕТЕРБУРГ
2022 г.

# СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

OpenGL является одним из самых популярных прикладных программных интерфейсов (API – Application Programming Interface) для разработки приложений в области двумерной и трехмерной графики.

Стандарт OpenGL (Open Graphics Library – открытая графическая библиотека) был разработан и утвержден в 1992 году ведущими фирмами в области разработки программного обеспечения как эффективный аппаратно-независимый интерфейс, пригодный для реализации на различных платформах. Основой стандарта стала библиотека IRIS GL, разработанная фирмой Silicon Graphics Inc.

Включает более 300 функций для рисования сложных трёхмерных сцен из простых примитивов. Используется при создании компьютерных игр, САПР, виртуальной реальности, визуализации в научных исследованиях.

Целью выполнения курсовой работы является разработка трехмерной сцены с использованием библиотеки OpenGL 2.0. В моем случае сцена должна представлять собой графическую модель игры Pacman. Для достижения поставленной цели необходимо решить следующие задачи:

- создать объекты игры Pacman с использованием графических возможностей библиотеки OpenGL;
- разработать алгоритмы анимации перемещения объектов;
- реализовать освещение с помощью библиотеки OpenGL;
- разработать модуль камеры для визуального взаимодействия со сценой;
- разработать алгоритмы управления и взаимодействия со сценой, ее объектами и камерой.

# 1 Создание объектов графической сцены

Pacman – аркадная видеоигра, разработанная японской компанией Namco и вышедшая в 1980 году. Задача игрока – управляя пакманом, съесть все точки в лабиринте, избегая встречи с привидениями, которые гоняются за героем. Итого нужно было создать модель лабиринта, пакмана, призрака (одного, так как они просто разного цвета), монеты и ягоды.

Для своих моделей, было принято решение использовать воксельный дизайн, то есть модели состоят из вокселей – аналог пикселя только для 3D сцен.

Для создания своих моделей, была использована программа MagicaVoxel. Эта программа была выбрана, потому что она бесплатная, простая в использовании и имеет возможность экспортировать модели в obj формат. На рисунке 1 представлен интерфейс программы. На нем можно увидеть, что в центре есть область взаимодействия с моделью, слева палитра цветов и набор инструментов для создания/удаления вокселей, справа же есть еще одно меню инструментов и список моделей в домашнем каталоге программы.

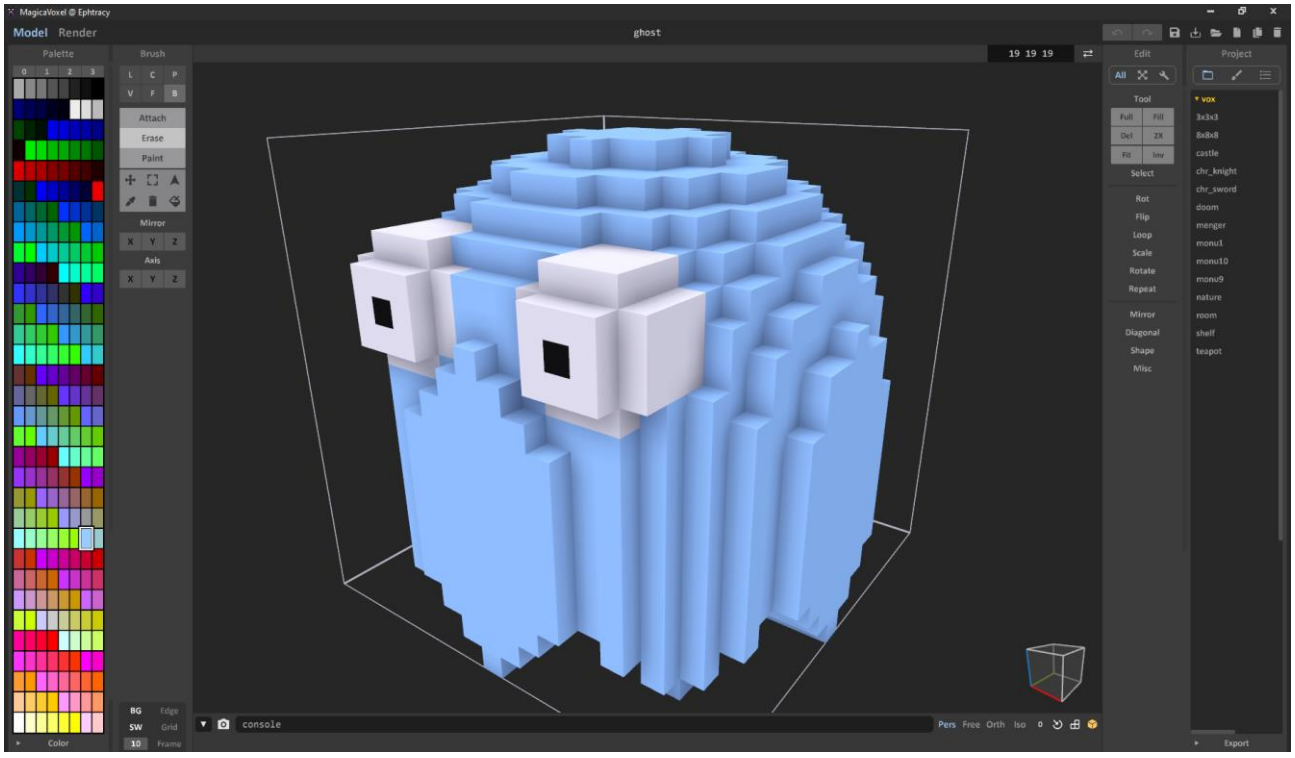


Рисунок 1 – Интерфейс программы MagicaVoxel

На рисунке 1 представлена модель призрака, которая в дальнейшем была использована в графической сцене. Для ее создания использовались

инструменты: создать сферу заданного размера (диаметр – 19 вокселей), удалить и добавить воксель. Аналогичным образом были созданы модели ягодки, лабиринта, монеты и пакмана. На рисунках 2–6 они представлены. Для анимации пакмана были смоделированы 13 моделей. В отчете присутствует только начальная и конечная модель анимации.

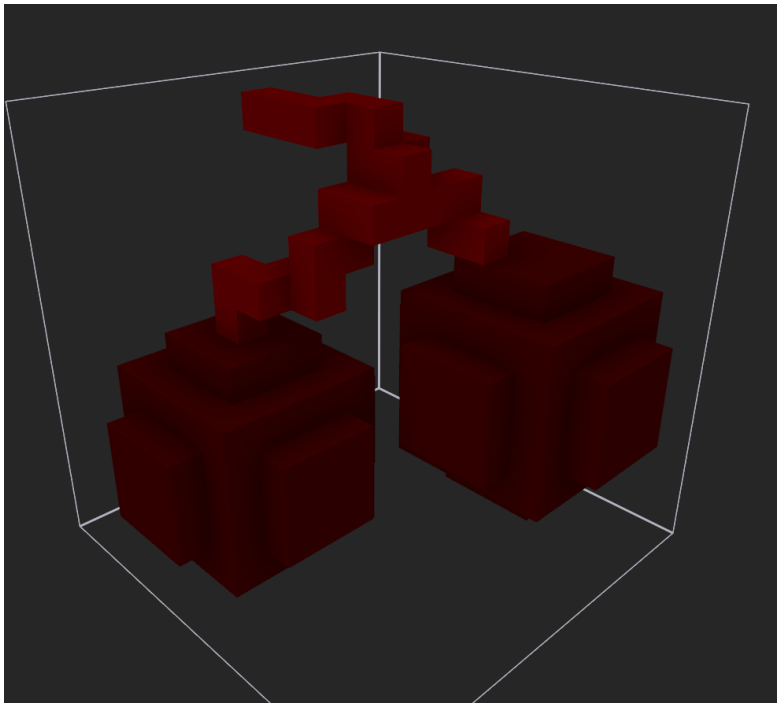

Рисунок 2 – Модель ягодки

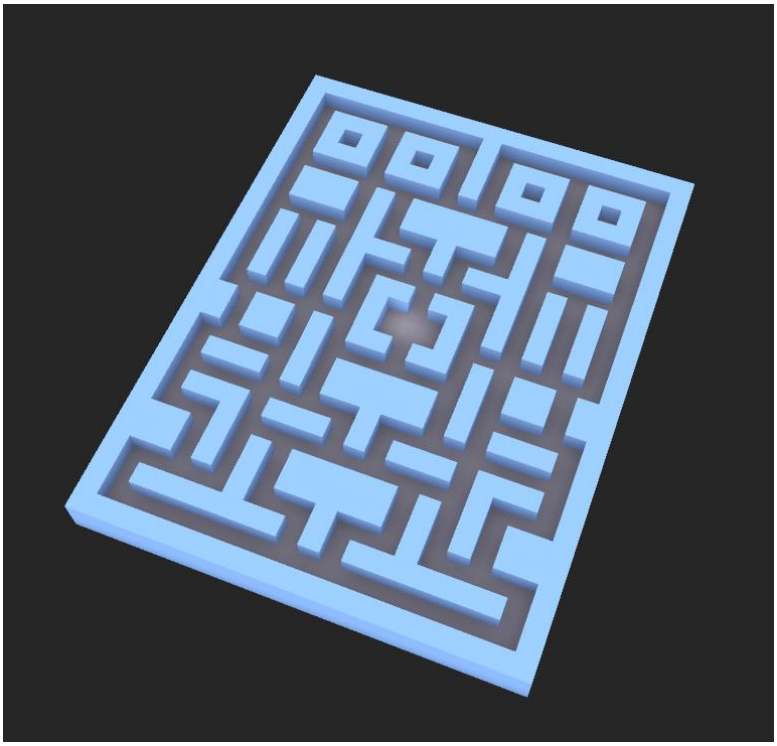

Рисунок 3 – Модель лабиринта

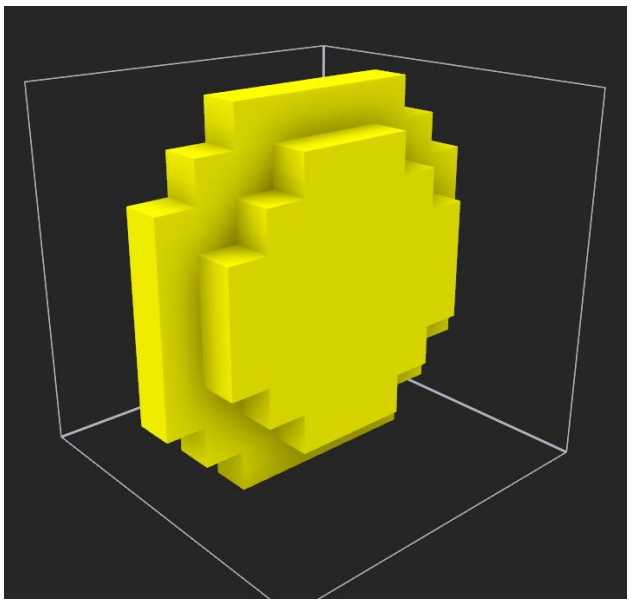

Рисунок 4 – Модель монетки

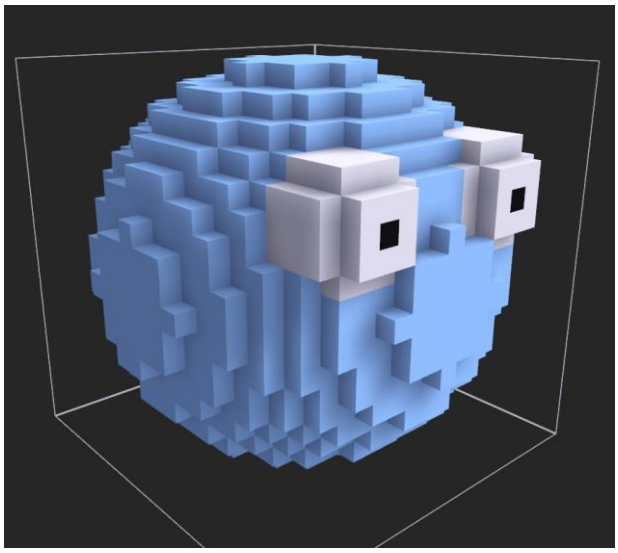

Рисунок 5 – Модель пакмана на начальном этапе анимации

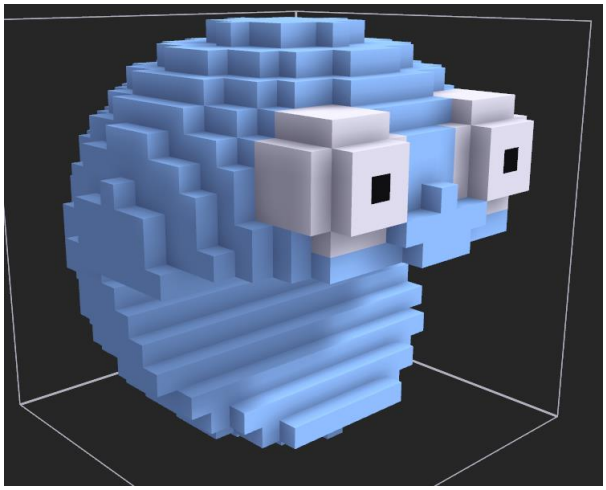

Рисунок 6 – Модель пакмана на конечным этапе анимации

После этого этапа, все модели были экспортированы в формат obj. Формат obj – это формат файла для описания геометрии, разработанный в Wavefront Technologies. Формат файла является открытым и был принят другими разработчиками приложений 3D-графики. Файлы данного формата можно открыть в простом блокноте. Синтаксис obj файлов достаточно простой. На рисунке 7 представлен кусок данного формата.

```
191  v  5.00 0.50 1.50
192  v  5.00 5.50 1.50
193  v  6.00 1.50 -2.50
194  v  6.00 4.50 -2.50
195  v  6.00 1.50 0.50
196  v  6.00 4.50 0.50
197  # 189 vertices
198
199  vn -1.00 0.00 -0.00
200  vn 1.00 0.00 -0.00
201  vn 0.00 -1.00 -0.00
202  vn 0.00 1.00 0.00
203  vn 0.00 0.00 -1.00
204  vn 0.00 -0.00 1.00
205  # 6 vertex normals
206
207  vt 0.70 0.50 0.00
208  vt 0.87 0.50 0.00
209  # 2 texture coords
210
211  f 1/1/1 2/1/1 3/1/1
212  f 4/1/1 2/1/1 1/1/1
213  f 5/1/1 6/1/1 7/1/1
214  f 8/1/1 6/1/1 5/1/1
215  f 9/1/1 5/1/1 10/1/1
```

Рисунок 7 – часть obj файла

В данном формате данные хранятся в виде: ключ + данные, перечисленные через пробел. Для хранения вершин используется ключ-буква ‘v’. Вершина храниться в виде: ключ, координата x, координата y, координата z. Пример хранения вершин можно увидеть на строчках 191–196 на рисунке 7. В данном формате еще есть данные с ключами ‘vn’ и ‘vt’. Ключ ‘vn’ говорит о том, что это нормаль, а ‘vt’ – координаты текстур (как это работает я не уверен, так как не использовал). Каждой вершине, нормали и текстуре задается свой id начиная с 1. Так же есть данные с ключом ‘f’. Это уже информация о какой-либо грани. Хранится в виде: ключ, id вершины 1/id текстуры 1/id нормали 1, id вершины 2/id

текстуры 2/id нормали 2, id вершины 3/id текстуры 3/id нормали 3. Есть и другие способы хранения данных в obj файле, но я их не рассматривал.

Попробовав сразу загрузить модели в OpenGL, выяснилось, что у моделей неподходящий масштаб и неправильное начало координат. Это все можно исправить с помощью функций в OpenGL, но это дополнительная нагрузка при отрисовке каждого кадра. Поэтому было принято решение, загрузить каждую модель в 3Ds max, передвинуть в центр координат, отмасштабировать, повернуть и снова экспортировать в obj. На рисунке 8 можно увидеть модель до обработки, а на рисунке 9 после. Так же на рисунке 10 можно увидеть, что модель состоит из множества треугольных поверхностей.

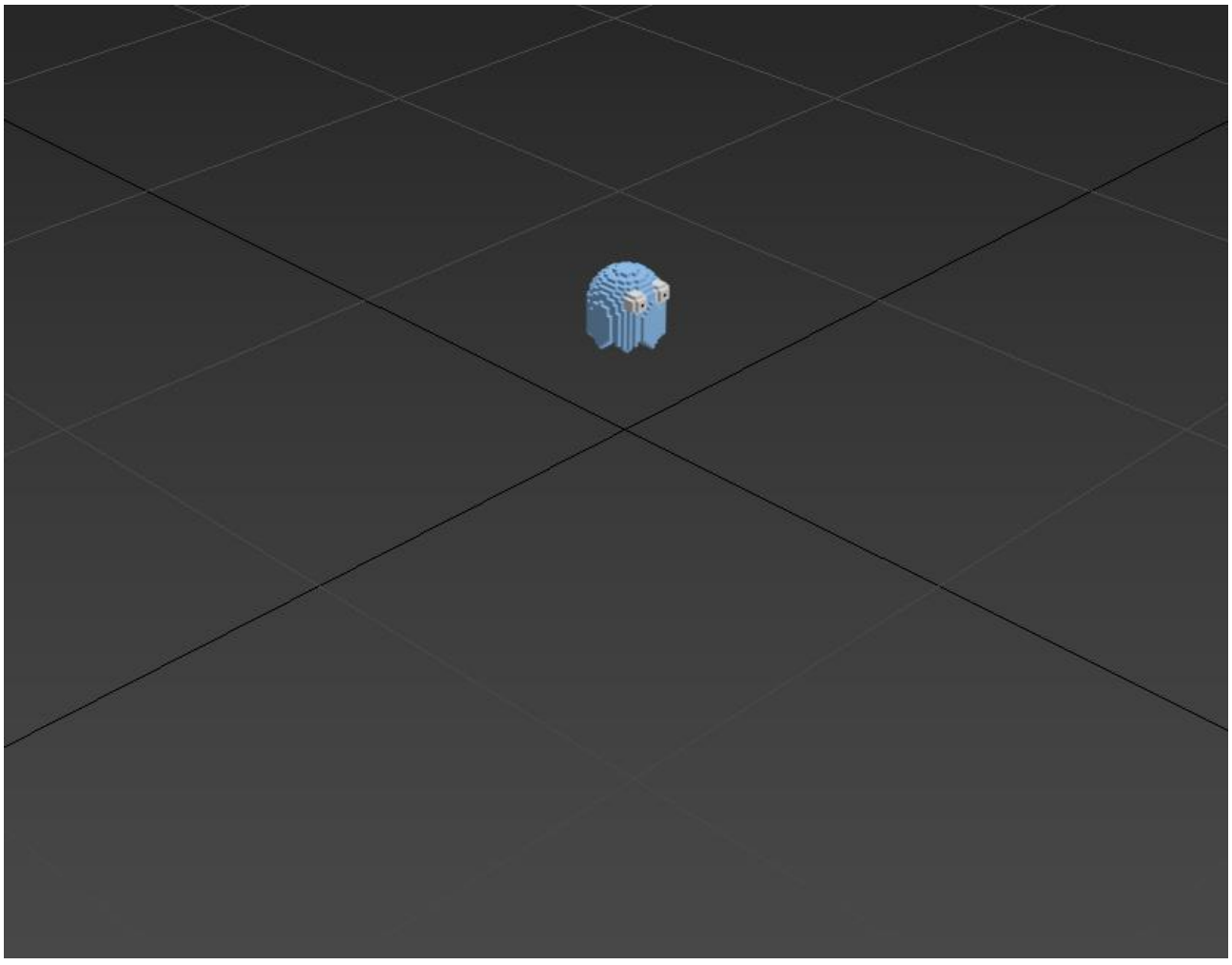

Рисунок 8 – Модель до преобразований

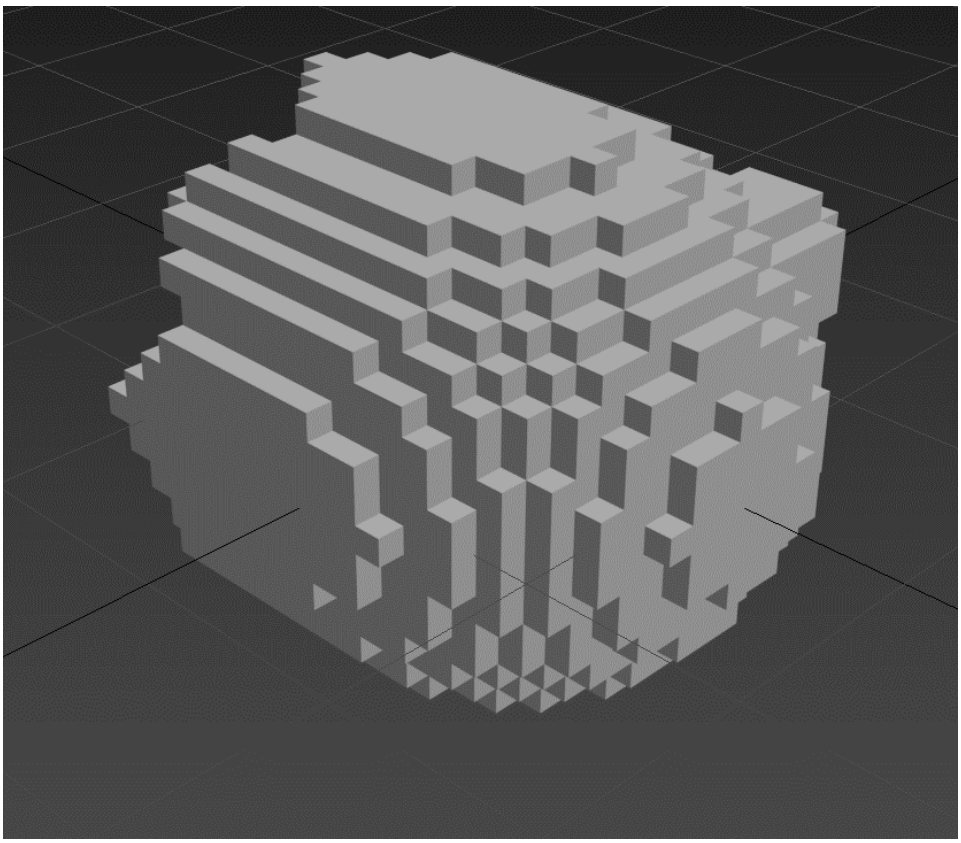

Рисунок 9 – Модель после преобразований

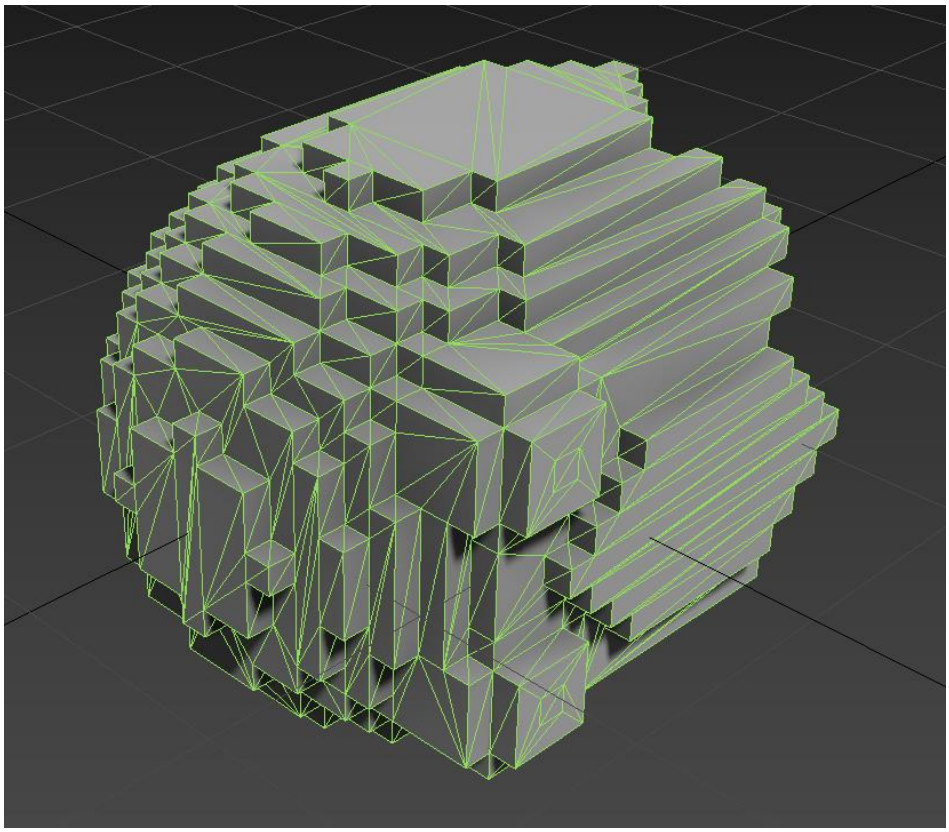

Рисунок 10 – Модель призрака с сеткой граней

Такие же операции были проведены с каждой моделью.

Для дальнейшего использования моделей в сцене, был разработан класс OBJMODEL. Он нужен для импорта и хранения моделей. Далее представлены важные части кода этого класса. На рисунке 11 представлен класс для хранения данных о поверхности, а на рисунке 12 представлена часть кода, отвечающая за чтение данных из файла и запись их во временные списки.

```
class Face {
  public:
    int edge;
    int *vertices;
    int normal;
    unsigned char color;

    Face(int edge, int *vertices, unsigned char inputcolor, int normal = -1) {
        this->edge = edge;
        this->vertices = vertices;
        this->normal = normal;
        color = inputcolor;
    }
};
```

Рисунок 11 – Класс для хранения данных о поверхности

```
if((file = fopen(filename, "r")) == NULL) {
    printf("Cannot load model %s\n", filename);
    return;
}

float a, b, c;
int v0, v1, v2, t0, t1, t2, n;
int *v;

while(fgets(tempstr, 60, file) != NULL) {
    if (tempstr[0] == 'v') {
        if (tempstr[1] == ' ') {
            sscanf(tempstr, "v %f %f %f", &a, &b, &c);
            vertices.push_back(new float[3]{a, b, c});
        } else if (tempstr[1] == 't') {
            sscanf(tempstr, "vt %f %f", &a, &b);
            texcoords.push_back(new float[2]{a, b});
        } else {
            sscanf(tempstr, "vn %f %f %f", &a, &b, &c);
            normals.push_back(new float[3]{a, b, c});
        }
    } else if (tempstr[0] == 'f') {
        sscanf(tempstr, "f %d/%d/%d %d/%d/%d %d/%d/%d", &v0, &t0, &n, &v1, &t1, &n, &v2, &t2, &n);
        v = new int[3]{v0 - 1, v1 - 1, v2 - 1};
        faces.push_back(Face(3, v, t0, n - 1));
    }
}
```

Рисунок 12 – Часть кода, читающая данных из файла

На рисунке 13 представлена часть кода, отвечающая за создание GLuint объекта, со всеми поверхностями.

```
list = glGenLists(1);
glNewList(list, GL_COMPILE);
    for(int i = 0; i < faces.size(); i++) {
        Face face(faces[i]);

        if (face.normal != -1) {
            glNormal3fv(normals[face.normal]);
        } else {
            glDisable(GL_LIGHTING);
        }

        glBegin(GL_TRIANGLES);
            for (int i = 0; i < face.edge; i++) {
                switch (model) { //выбор цвета грани
                glVertex3fv(vertices[face.vertices[i]]);
            }
        glEnd();

        if (face.normal == -1)
            glEnable(GL_LIGHTING);
    }
glEndList();
```

Рисунок 13 – Создание объекта типа GLuint

После этого модель готова к отрисовке на сцене, нужно только вызвать метод glCallList(list). Более подробно реализацию данного класса можно посмотреть в файлах OBGMODEL.h и OBJMODEL.cpp в приложении А. На рисунке 14 представлено, как загружаются все модели в программу.

```
maze.load("maze.obj", 'm');

for(i = 0; i < 13; i++)
    pacman[i].load(wayPacmanModel[i], 't');

ghost[0].load("ghost.obj", 'y');
ghost[1].load("ghost.obj", 'u');
ghost[2].load("ghost.obj", 'i');
ghost[3].load("ghost.obj", 'o');
ghost[4].load("ghost.obj", 'p');

money.load("money.obj", 'n');
cherry.load("cherry.obj", 'c');
```

Рисунок 14 – Загрузка всех моделей

## 2 Упрощенное описание работы сцены

Для работы всей сцены был разработан класс LOGICS. Внутри этого класса были реализованы другие классы:

- MAZE – класс для хранения всей информации о лабиринте (текущее количество и расположение монет, ягод и стен);
- ENTITY – класс для хранения координат, направления движения и скорости перемещения объектов на сцене;
- HERO – класс, унаследованный от ENTITY, для реализации пакмана;
- ENEMY – класс, унаследованный от ENTITY, для реализации призраков.

В классе LOGICS есть 4 основных метода:

- метод step() – метод, где все существа на сцене перемещаются в заданном направление на единицу скорости, генерируются ягодки, выбирается новое направление движения призраков, если они уперлись в стенку (с 33% шансом призраки могу изменить свое направление движения на перекрестке);
- метод eat() – метод для реализации поедания монет и ягод пакманом;
- die_pacman() – метод для проверки столкновений между пакманом и призраками;
- gen(char * way) – этот метод нужен для получения изначальной информации о лабиринте, координат призраков и пакмана.

В виде параметра к методу gen(char * way), требует путь к специальному бинарному файлу (в нем хранится размер лабиринта, координаты монет, стенок, начальные координаты призраков).

Благодаря этому классу работа с OpenGL заключалась в настройке изначальных параметров сцены и отрисовке объектов по заданным координатам.

# 3 Анимации графической сцены

## 3.1 Анимация пакмена

У пакмана есть две анимации. Анимация перемещения и анимация открывания/закрывания рта. Анимация перемещения заключается в изменение координат и перерисовке объекта на новых координатах.

Для отрисовки анимации открывания/закрывания рта были смоделированы 13 моделей (об этом уже упоминалось в пункте про разработку моделей). Анимация заключается в том, что в разных промежутках лабиринта, загружается разная модель пакмана. Какую модель загружать, определяется благодаря методу get_mouth() из класса HERO. В классе LOGICS, на основание координат, высчитывается в какой позиции должен находиться рот (если пакман приближается к монетке, то рот закрывается, а если отдаляется, то рот открывается). На рисунках 15 и 16 представлены разные этапы анимации открывания и закрывания рта, а на рисунке 17 представлено, как в OpenGL отрисовывается пакман.

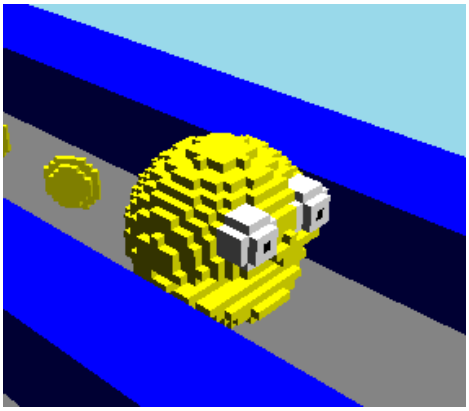

Рисунок 15 – Один из вариантов анимация пакмана

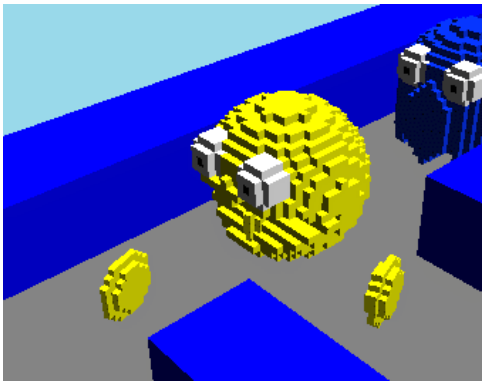

Рисунок 16 – Другой вариант анимация пакмана

```
glPushMatrix();
    pacX = logics->pacman.get_point().x + 0.5;
    pacY = logics->pacman.get_point().y + 0.5;
    glTranslatef(pacX, pacY, 37.5);

    switch(logics->pacman.get_dir()) {
        case 0:
            newAngle = 0 + offsetAngle;
            glRotated(270, 0, 0, 1);
            break;
        case 1:
            newAngle = 270 + offsetAngle;
            break;
        case 2:
            newAngle = 180 + offsetAngle;
            glRotated(90, 0, 0, 1);
            break;
        case 3:
            newAngle = 90 + offsetAngle;
            glRotated(180, 0, 0, 1);
            break;
    }

    pacman[logics->pacman.get_mouth()].draw();
glPopMatrix();
```

Рисунок 17 – Отрисовка пакмана

## 3.2 Анимация призрака

У призрака есть только одна анимация. Эта анимация заключается в перемещение призрака по лабиринту. Расчет координат всех призраков, как и у пакмана, происходит внутри класса LOGICS. На рисунке 18 представлен призрак на сцене.

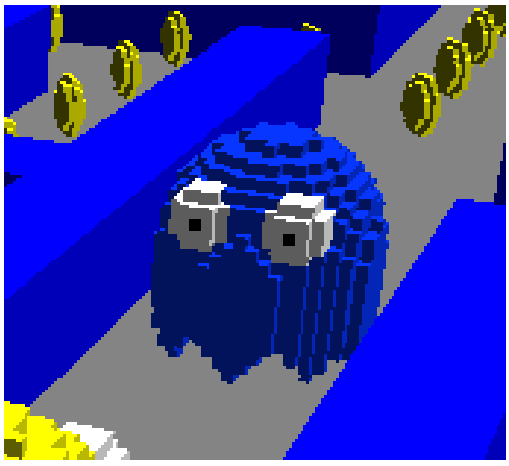

Рисунок 18 – Синий призрак

На рисунке 19 представлено, как отрисовываются все призраки.

```
for(i = 0; i < 4; i++) {
    x = logics->enemy[i].get_point().x + 0.5;
    y = logics->enemy[i].get_point().y + 0.5;

    glPushMatrix();
        glTranslatef(x, y, 37.5);

        switch(logics->enemy[i].get_dir()) {
            case 0:
                glRotated(270, 0, 0, 1);
                break;
            case 1:
                break;
            case 2:
                glRotated(90, 0, 0, 1);
                break;
            case 3:
                glRotated(180, 0, 0, 1);
                break;
        }

        if(logics->enemy[i].get_type()) ghost[i+1].draw();
        else ghost[0].draw();
    glPopMatrix();
}
```

Рисунок 19 – Отрисовка призраков

Так как класс ENEMY и HERO унаследованы от одного класса ENTITY, их отрисовка очень похожа:

– получить координаты объекта;

– повернуть модель в зависимости от направления движения;

– отрисовать объект.

## 3.3 Анимация монет

У монет тоже есть только одна анимация. Это анимация вращения вокруг своей оси. Реализовано это с помощью функции glRotated(angleMoney, 0, 0, 1). На каждом этапе перерисовки, к переменной angelMoney прибавляется единица. Если значение переменной становится больше 360, то ее значение сбрасывается к 0. На рисунке 20 представлен процесс отрисовки всех монет и ягод. Так как лабиринт хранится в виде двумерного массива, то здесь координаты высчитываются вручную.

```
for(float i = 0; i < logics->maze.get_size().x; i++)
    for(float j = 0; j < logics->maze.get_size().y; j++) {
        if(logics->maze.check_value(i, j) == 1) {
            glPushMatrix();
                glTranslatef(j * 25 + 12.5, i * 25 + 12.5, 32.5);
                glRotated(angleMoney, 0, 0, 1);
                money.draw();
            glPopMatrix();
        }
        if(logics->maze.check_value(i, j) == 4) {
            glPushMatrix();
                glTranslatef(j * 25 + 12.5, i * 25 + 12.5, 32.5);
                cherry.draw();
            glPopMatrix();
        }
    }
```

Рисунок 20 – Процесс отрисовки всех монет и ягод

## 3.4 Управление сценой

Для управления сценой прописаны сценарии нажатия заданных клавиш. Список всех доступных клавиш приведен ниже.

Клавиши A, D, S, Q:

– [A] и [D] – поворот пакмана направо или налево на следующем перекресте, относительно его текущего направления движения;

– [S] – развернуться назад;

– [Q] – выход из приложения.

Клавиша ESC – выход из приложения.

# 4 Освещение

Основной источник света на сцене размещен высоко над сценой. В графической сцене используется фоновое освещение для иллюстрации света на объектах.

Фоновое излучение – это свет, который настолько распределен средой (предметами, стенами и так далее), что его направление определить невозможно.

Для работы с освещением сначала необходимо разрешить его использование. Для этого использовалась функция glEnable(GLenum) с параметром GL_LIGHTING и GL_LIGHT0. Затем были указаны параметры излучаемого света и его позиция в пространстве, для этого задались массивы из четырех элементов. Для света — это параметры цветовых каналов RGBA, для позиции – координаты в пространстве XYZ. После были вызваны функции glLightModefv, чтобы установить модель освещения в сцене, и glLighfv для того, чтобы указать положение источника света.

Для более естественного распределения света использовалась функция glEnable(GL_NORMALIZE).

На рисунке 21 располагается код настройки освещения и цвета фона.

```
glClearColor(0.6, 0.851, 0.918, 0);
glEnable(GL_CULL_FACE);

glEnable(GL_DEPTH_TEST);

glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glEnable(GL_NORMALIZE);
glEnable(GL_COLOR_MATERIAL);

const GLfloat light_ambient[]  = {0.3f, 0.3f, 0.3f, 1.0f};
const GLfloat light_position[] = {262.5, 337.5, 500.0f, 0.0f};

glLightModelfv(GL_LIGHT_MODEL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

Рисунок 21 – Код, отвечающий за настройку освещения

# 5 Процесс разработки камеры

Так как данная сцена является трехмерной сценой, необходимо было создать камеру, благодаря которой можно задавать углы обзора для удобства восприятия сцены.

## 5.1 Настройки перспективы

Для задания перспективы использовалась функция glFrustum() – функция для выбора перспективного способа проецирования. Она имеет 6 параметров:

- координата плоскости отсечения слева;
- координата плоскости отсечения справа;
- координата плоскости отсечения снизу;
- координата плоскости отсечения сверху;
- расстояние от наблюдателя до ближней плоскости отсечения;
- расстояние от наблюдателя до дальней плоскости отсечения.

На рисунке 22 представлены настройки перспективы сцены:

```
const float ar = (float) width / (float) height;

glViewport(0, 0, width, height);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(-ar, ar, -1, 1, 1, 3000);

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
```

Рисунок 22 – Настройки перспективы сцены

## 5.2 Настройки камеры

Для моделирования камеры, которая направлена в произвольную точку пространства и расположена в заданном месте, была использована функция gluLookAt.

Функция имеет 9 параметров: первые три параметра определяют координаты местоположения камеры; вторая тройка параметров задает координаты точки, в которую нацелена камера; последние три параметра – направление вектора «вверх» – используются для поворотов изображения в плоскости проецирования.

Центром, куда смотрит камера, был выбран центр пакмана. Координаты самой камеры высчитываются относительно центра пакмена и направления, в которое он двигается. Камера в основном смотрит сзади с отклонением в 45 градусов влево.  При смене направления движения пакмана, камера плавно меняет свое положение. На рисунке 23 изображен код, отвечающий за расположение и перемещение камеры.

```
offsetX = sin(M_PI / 180 * angle) * 100;
offsetY = cos(M_PI / 180 * angle) * 100;

switch(flagOffset) {
    case 1: angle++; break;
    case 2: angle--; break;
}

if(angle == newAngle) flagOffset = 0;
else if(angle >= 360) angle = 0;
else if(angle < 0) angle = 360;

pacX = logics->pacman.get_point().x + 0.5;
pacY = logics->pacman.get_point().y + 0.5;

glLoadIdentity();
gluLookAt(pacX + offsetX, pacY + offsetY, 100, pacX, pacY, 37.5, 0, 0, 1);
```

Рисунок 23 – Код, отвечающий за камеру

Переменная angel отвечает за текущий угол камеры относительно пакмана, newAngel – угол, к которому нужно стремиться камере, flagOffset – флаг, отвечающий за разрешение/запрет изменения угла камеры.

На рисунке 24 изображен пример расположения камеры на сцене.

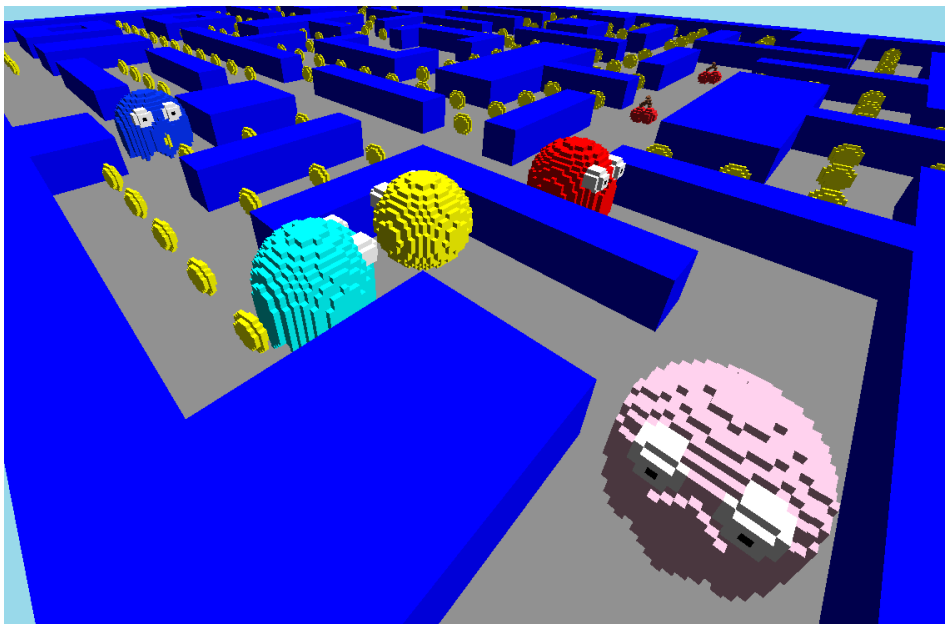

Рисунок 24 – Пример расположения камеры

# ЗАКЛЮЧЕНИЕ

В ходе выполнения поставленных задач для реализации графической модели получены следующие результаты.

Созданы 3D модели пакмана, лабиринта, монетки, призраков и ягодки. Получены их obj файлы. Написан класс для их упрощенного импорта в сцену (нет текстур и материалов).

Разработан класс LOGICS с упрощенной логикой игры Pacman. Алгоритмы анимации перемещения объектов сцены с использованием различных математических вычислений.

Реализовано освещение сцены посредством функций библиотеки OpenGL.

Для визуального взаимодействия со сценой разработан модуль камеры с использованием функций библиотеки GLU. Разработаны алгоритмы управления и взаимодействия со сценой, объектами сцены и камерой посредством тригонометрических и геометрических формул.

Финальный вариант выполненной работы представляет собой трехмерную графическую модель игры Pacman с элементами анимации/управления объектами сцены, освещением и модулем камеры.

Курсовая работа выполнена в IDE Code::Blocks 17.12 на языке СИ++ с использованием графической библиотеки OpenGL 2.0 и библиотеки GLUT.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Е.А. Снижко, Н.А. Флерова, А.В. Воронцов. Программирование компьютерной графики с использованием библиотеки openGL. Лабораторный практикум. СПб.: издательство БГТУ, 2006, -93с.

2. О.В. Арипова, О.А. Палехова, А.Н. Гущин. Программирование на языке высокого уровня. Лабораторный практикум. СПб.: издательство БГТУ, 2014, -94с.

3. О.А. Палехова, Основы программирования на языке С. Практикум. СПб.: издательство БГТУ, 2016, -95с.

# ПРИЛОЖЕНИЕ А

Проект программы, созданные в процессе разработки, находятся в архиве: “И596_ОреховРВ_Курсовая_КомпГеом_и_Граф”.

В папке “../bin/Debug” находится исполняемый файл “Pacman.exe”