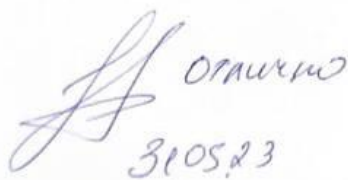


Министерство образования Российской Федерации
Пензенский государственный университет
Кафедра «Системы автоматизированного проектирования»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе
по курсу «Компьютерная графика и 3D-моделирование»
на тему «Разработка программы построения и визуализации
3D объекта»


31.05.23

Выполнили:
студенты группы 21ВВС1
Роганов Д.В.
Федоренко В.А.

Принял:

к.т.н., доцент кафедры САПР Финогеев А.А.

Пенза, 2023

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра

Системы автоматизации проектирования

УТВЕРЖДАЮ

Зав. кафедрой

“ ”

г.

ЗАДАНИЕ

на курсовое проектирование по курсу

Компьютерная графика и 3D моделирование

Студенту Козлову Александру Владимировичу Группа 218БС1

Тема

проекта «Разработка программы построения и визуализации 3D объекта»

Исходные данные (технические требования) на проектирование

Основная цель проекта: Разработка графической программы для построения визуализации 3D модели на языке программирования C++, для реализации процесса динамики объекта, фона и камеры в трехмерной сцене.

Объект моделирования: 3D модель реального объекта -

Задачи проекта:

1. Спроектировать 3D твердотельную модель заданного объекта средствами Autocad (3D Max и т.д.)
2. Синтезировать 2D чертежи проекционных видов 3D модели на плоскости XY, XZ, YZ.
3. Импортировать полученные 3D модели объекта в программную среду C++.
4. Подобрать соответствующие текстуры для объектов сцены.
5. С помощью собственных функций синтезировать 3D объект и визуализировать 3D сцену.
6. Реализовать движения частей объекта относительно друг друга, движение камеры вокруг объекта.
7. Реализовать пользовательское управление движением камеры вокруг 3D объекта посредством клавиатуры (мыши)

Учебно-методический материал: Сервер - CAD-Filer, Диск - Т, Раздел - Компьютерная графика

Программное обеспечение: C++ Builder

CAD Import VCL - DXF DWG PLT HPGL SVG CGM библиотека для Delphi и C++.

Объем работы по курсу

1. Расчетная часть (содержание пояснительной записки)
 1. Анализ и описание процесса синтеза 3D объекта методом твердотельного моделирования
 2. Декомпозиция на графические примитивы, аппроксимация гранями, синтез видов
 3. Подбор текстур, источников освещения и камеры
 4. Описание расположения объекта/сцены в системе координат
 5. Импортирование созданных моделей и ландшафта в программную среду
 6. Разработка программы визуализации объекта/сцены с текстурами, освещением, камерой
 7. Программная реализация динамики объекта и камеры посредством задания их траекторий
 8. Реализация интерфейса в отношении управления объектом или камерой в сцене
 9. Отладка и тестирование программы
 10. Визуализация управляемой анимации объекта, камеры и сцены в целом
11. Графическая часть
 1. Чертежи трех проекций объекта
 2. Структурная схема алгоритма,
 3. Листинг программ.
12. Экспериментальная часть
 1. Тестирование и верификация программного обеспечения

Срок выполнения проекта по разделам

1	Моделирование 3D объекта и чертежей	к	22.02.23
2	Импорт моделей в среду	к	8.03.23
3	Разработка алгоритма и программы	к	22.03.23
4	Реализация движения объекта и камеры	к	5.04.23
5	Реализация интерфейса и элементов управления	к	12.04.23
6	Верификация и отладка ПО	к	19.04.05
7	Подготовка пояснительной записки и защита проекта	к	17.05.23

Дата выдачи задания "8" февраля 2023 г.

Дата защиты проекта "17" мая 2023 г.

Руководитель К.Т.Н., доцент

/А.А. Финогеев/

Задание получил "02" _____

02

2023 г.

Студент Резолюция Н.А.

Н.А.

Содержание

Введение	5
1 Расчётная часть.....	6
1.1 Анализ и описание процесса синтеза 3D объекта методом твердотельного моделирования	6
1.2 Декомпозиция на графические примитивы, аппроксимация гранями	8
1.3 Описание расположения объекта в системе координат	9
1.4 Импортирование созданной модели в программную среду	10
1.5 Разработка программы визуализации объекта	14
1.6 Программная реализация динамики объекта.....	16
1.7 Отладка и тестирование программы	20
Заключение	21
Список литературы	22
Приложение А. Листинг программы.....	23
Приложение Б. Графическая часть. Чертежи трёх проекций объекта. .	30
Приложение В. Окно программы.Визуализация объекта и камеры.....	32

ВВЕДЕНИЕ

Компьютерные технологии предоставляют широкий спектр возможностей для визуализации в трёхмерном пространстве. Трёхмерное моделирование как пространственное изображение объекта в трёхмерной системе координат даёт возможность максимально информативно, детально и реалистично представить его форму, текстуру, размер и цвет, позволяет рассмотреть объект с любого интересующего ракурса.

Трёхмерная графика считается незаменимым средством для демонстрации разного рода сложных технических узлов, многоступенчатых производств, архитектурных сооружений. Трёхмерные модели наглядно показывают все особенности строения объекта, его мельчайшие элементы, которые скрыты от глаз наблюдателя. На сегодняшний момент 3D изображения являются пиком совершенства в рекламной и дизайнерской индустрии.

Компьютерное моделирование — это процесс создания компьютерных моделей. Компьютерная модель — компьютерная программа, работающая на отдельном компьютере или множестве взаимодействующих компьютеров, реализующая абстрактную модель некоторой системы. Компьютерные модели стали обычным инструментом математического моделирования и применяются в физике, астрофизике, механике, химии, биологии, экономике, социологии и других науках. Компьютерные модели используются для получения новых знаний о моделируемом объекте или для приближенной оценки поведения систем, слишком сложных для аналитического исследования.

В качестве среды разработки 3D объекта была выбрана программа Blender 3.4., для синтезирования 2D чертежа – Microsoft Visual Studio 2019, язык программирования C++.

Целью данной курсовой работы является создание программы построения каркасной модели 3D объект

1 РАСЧЁТНАЯ ЧАСТЬ

С каждым годом роль компьютерного моделирования и инженерного анализа при разработке новых изделий растет и, в конечном счете, они должны стать неотъемлемой и органичной частью любого процесса проектирования. Это предполагает тесную взаимосвязь между модулями геометрического моделирования и инженерного анализа компьютерных систем проектирования с тем, чтобы выполнение соответствующих расчетов и моделирования стало для рядового проектировщика таким же обычным и рутинным делом, как создание трехмерной модели или выпуск чертежей.

1.1 Анализ и описание процесса синтеза 3D объекта методом твердотельного моделирования

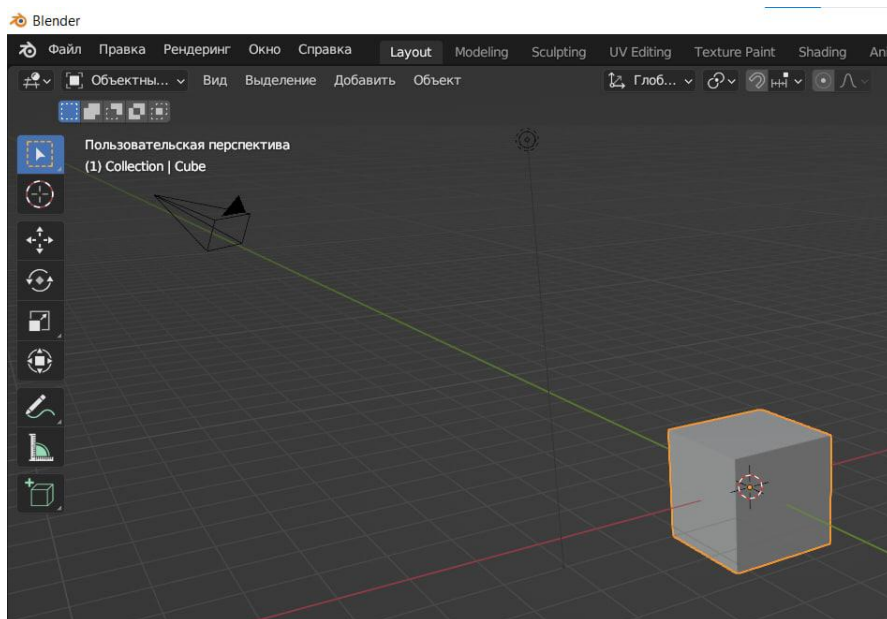


Рисунок 1 – Добавление куба

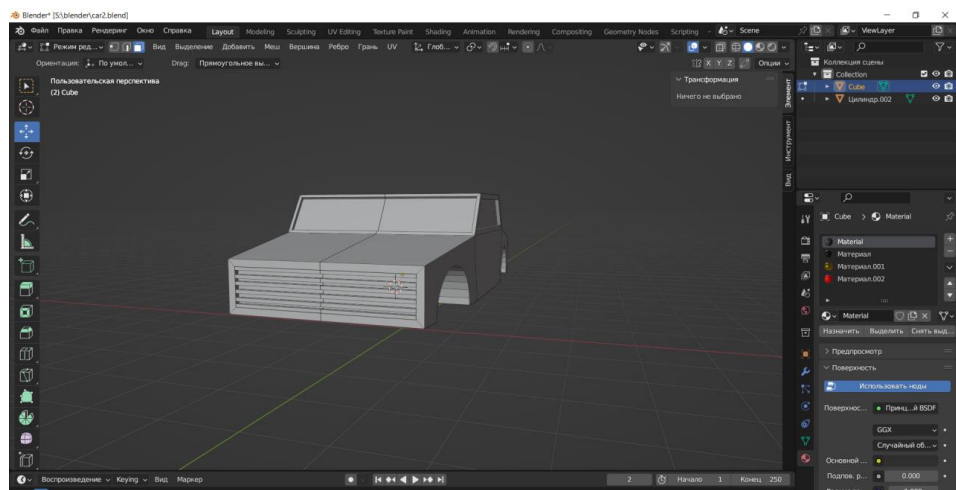


Рисунок 2 – Преобразование куба в каркас машины

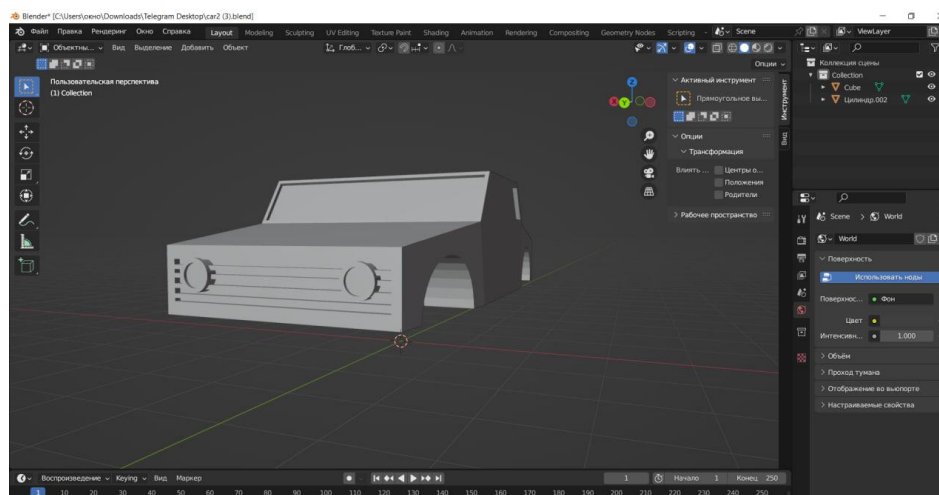


Рисунок 3 – Добавление фар машине

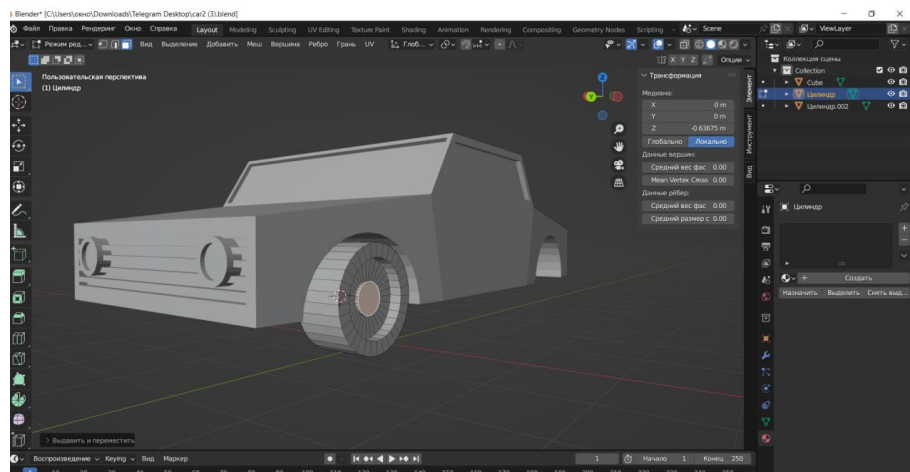


Рисунок 4 – Работа с колесами машины

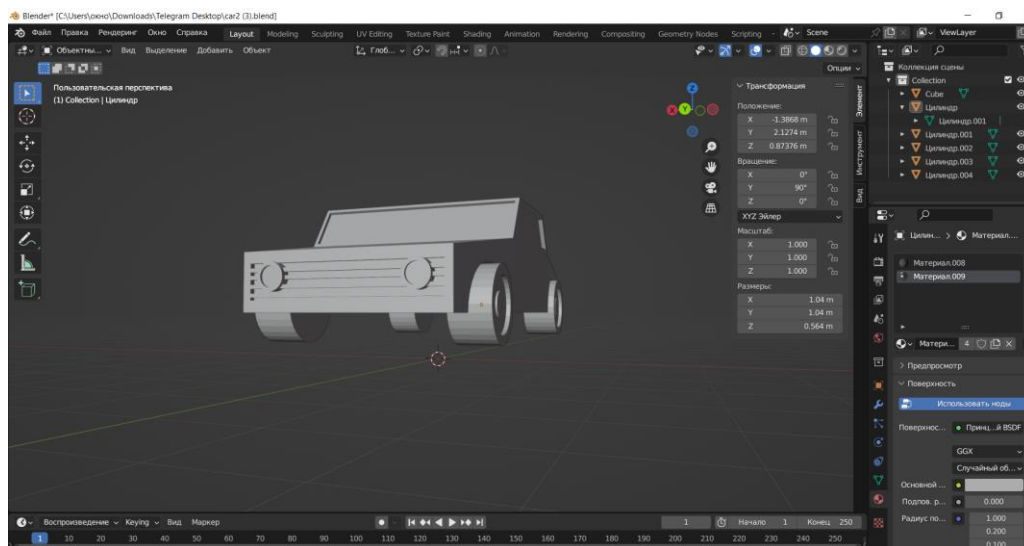


Рисунок 5 – Итог

1.2 Декомпозиция на графические примитивы, аппроксимация гранями

Аппроксимация — научный метод, состоящий в замене одних объектов другими, в каком-то смысле близкими к исходным объектам, но более простыми.

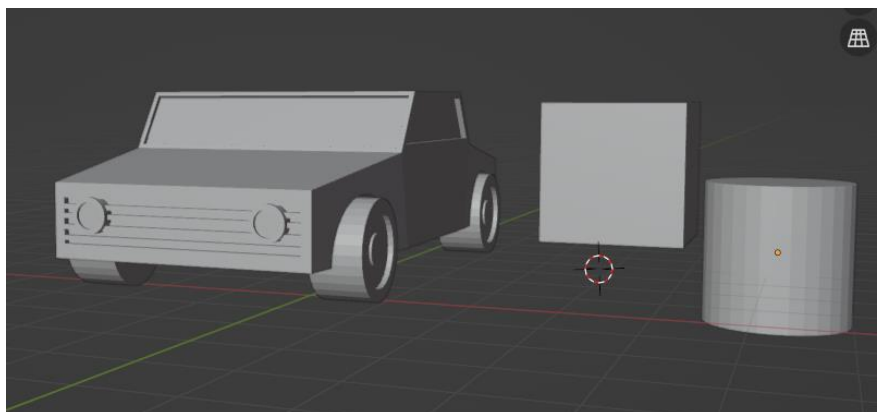


Рисунок 6 – Декомпозиция

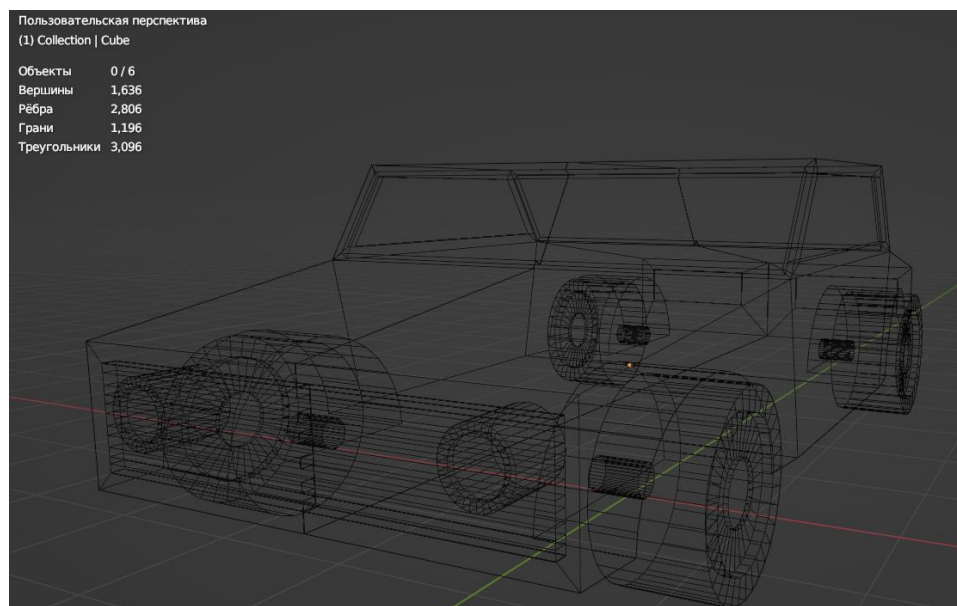


Рисунок 7 – Аппроксимация гранями

1.3 Описание расположения объекта в системе координат

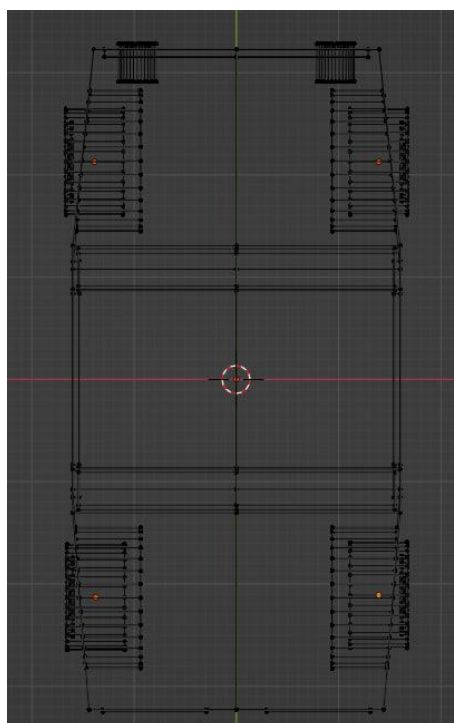


Рисунок 8 – Расположение сверху

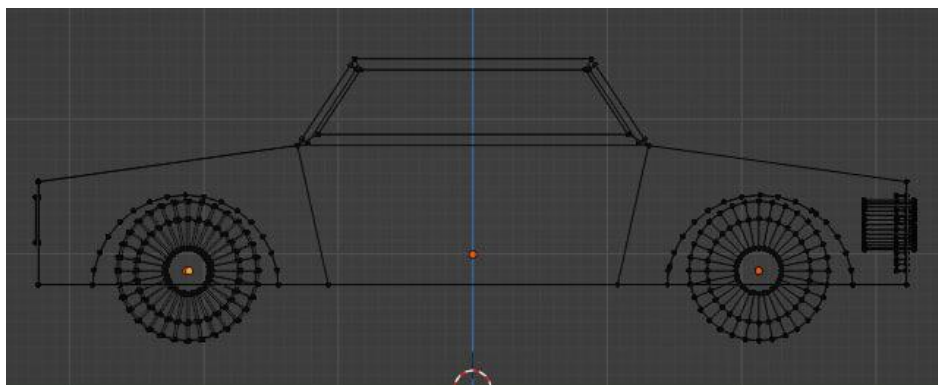


Рисунок 9 – Расположение сбоку

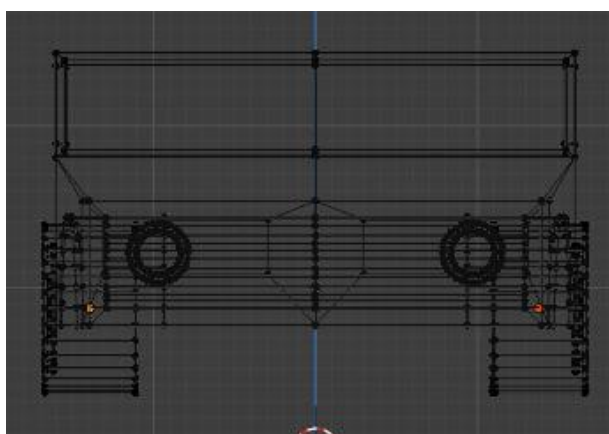


Рисунок 10 – Расположение спереди

1.4 Импортирование созданной модели в программную среду

Импортированный файл в OBJ формате содержит информацию о координатах вершин модели и её гранях, что удобно использовать для чтения из файла при реализации программы на C++.

OBJ — это формат файла, содержащий информацию о геометрии объекта. Он используется для хранения 3D-моделей, созданных путем комбинирования 3D-объектов, сеток, форм, метаданных, текстур и многого другого.

По заданию данной курсовой работы готовый продукт должен быть представлен в программной среде в виде сетчатой модели, соответственно, для отрисовки необходимо, чтобы в файле OBJ хранились данные лишь о вершинах – v , и рёбрах – l объекта. Чтобы в OBJ файле хранилась информация только о координатах вершин и порядке их соединения, в среде 3D моделирования выделяем весь объект в режиме редактирования, нажимаем X

и во всплывающем окне выбираем «Только грани» (рис 11). Также необходимо удалить все материалы и окружающую среду в файлах данного проекта Blender при их наличии. Таким образом, при экспорте в OBJ останутся данные лишь о вершинах и рёбрах.

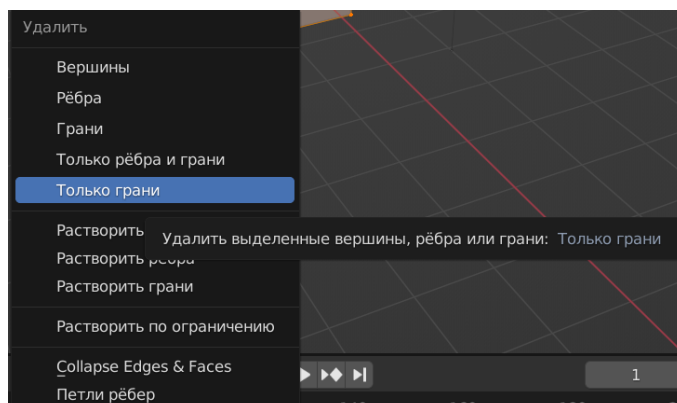


Рисунок 11 – Удаление граней объекта

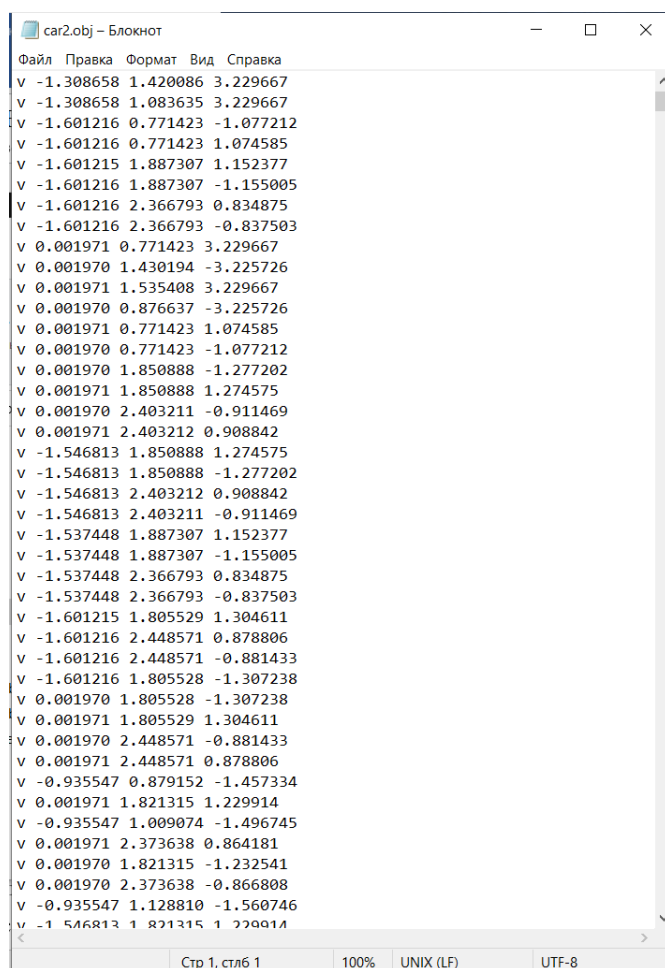


Рисунок 12 – Содержимое файла car2.obj

Импортирование модели в программную среду осуществляется через

алгоритм чтения OBJ файла с сохранением результата в динамический массив.

Функция `void calculateVertexAndLineCount` проходит файл построчно, пока не достигнет конца. Если первым элементом строки стоит либо символ «v», либо «l», то будет считаться количество таких строк. Количество «v» и «l» нужно для создания динамических массивов, в которых будет содержаться информация о вершинах и рёбрах. После этого функции `void readVertex` и `void readEdges` вновь построчно проходят файл и в зависимости от первого символа (v или l) записывают значения в соответствующие динамические массивы.

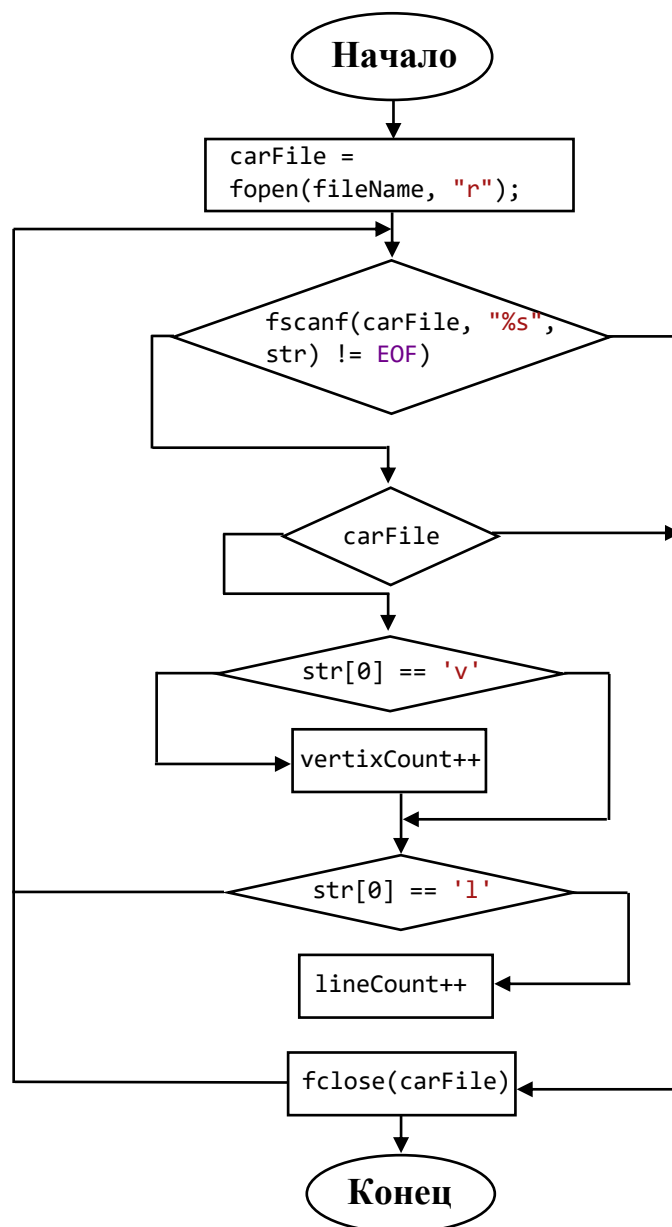


Рисунок 13 – Блок-схема алгоритма счёта вершин и рёбер

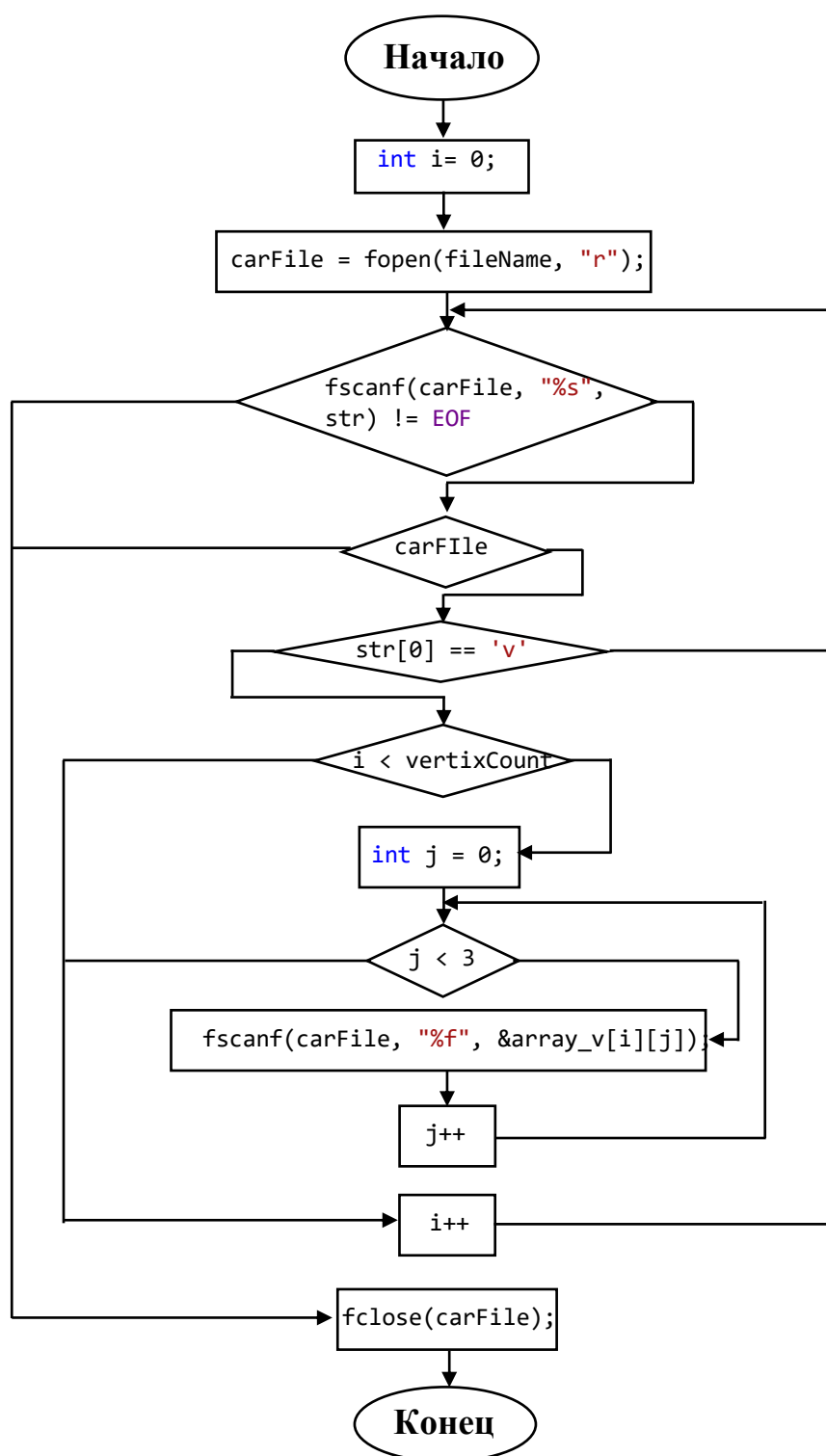


Рисунок 14 – Блок-схема функции void readVertex()

1.5 Разработка программы визуализации объекта

Отрисовка 3D модели осуществляется через функцию `void paint (HDC hdc)`, в которой прописаны встроенные в WinAPI функции `MoveToEx` и `LineTo`.

Функция `BOOL MoveToEx(HDC hdc, int X, int Y, LPPOINT lpPoint)` изменяет текущее положение (позицию) пера. В структуру `lpPoint` заносятся старые координаты пера, в общем случае можно указывать константу `NULL`.

Функция `BOOL LineTo(HDC hdc, int XEnd, int YEnd)` строит прямую линию, начало соответствует текущему положению пера. По окончании отрисовки, текущее положение пера изменится на `(XEnd, YEnd)`.

Функция работает следующим образом: итеративно пробегает по каждой строке массива `array_1`, хранящей порядок соединения вершин. Далее перо отрисовки ставится на вершину со значением первого `j`-го элемента `i`-ой строки, после чего с помощью функции `LineTo` соединяем линии в порядке, описанном в `i`-ой строке массива `array_1`. В конце последняя вершина соединяется с первой вершиной.

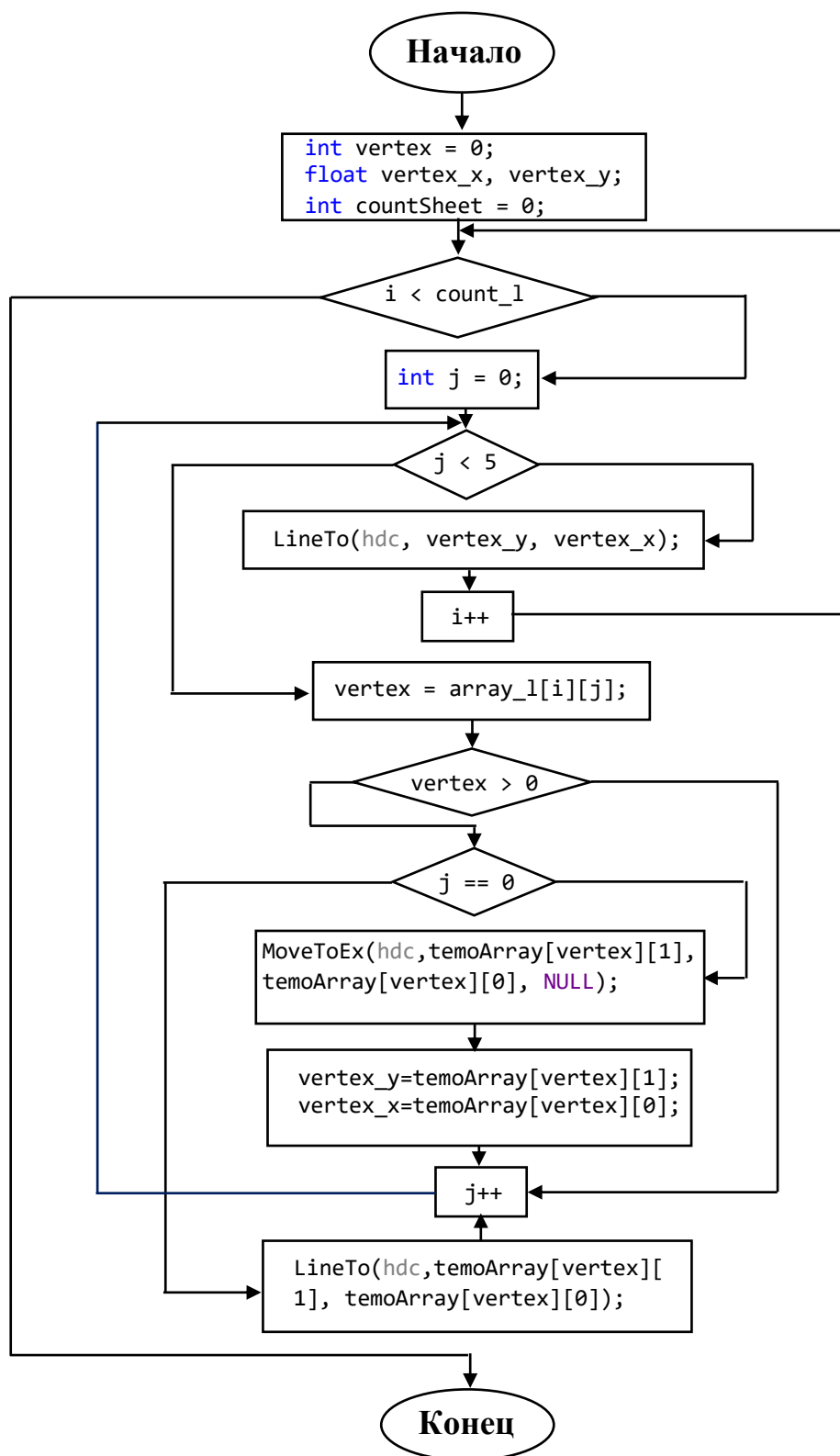


Рисунок 15 – Блок-схема функции void paint ()

1.6 Программная реализация динамики объекта

Для реализации динамики объекта преобразуем мировые координаты вершин. Оно будет проводиться поэтапно:

Первый этап – видовое преобразование, второй – перспективное, третий – экранное.

Для видового преобразования необходима величина R (расстояние от точки наблюдения до начала мировой системы координат) и полярные углы $\text{float } \text{fiAngle}$ и $\text{float } \text{tettaAngle}$.

Угол phi (φ) – это угол от положительной оси y до ортогональной проекции вектора на плоскость yz . Угол положителен по отношению к положительной оси z . Тета-угол (θ) – это угол от оси x к самому вектору. Угол положителен к плоскости yz .

Для реализации видового преобразования была написана функция `void viewTransformation()`. Функция выполняет перемножение матрицы видовых преобразований на координаты вершин объекта и записывает результат в двумерный динамический массив `temoArray`.

Матрица видовых преобразований представлена в виде формулы (1):

$$[x_e \ y_e \ z_e \ 1] = [x_w \ y_w \ z_w \ 1] * \begin{bmatrix} -\sin \theta & -\cos \varphi \cos \theta & -\sin \varphi \cos \theta & 0 \\ \cos \theta & -\cos \varphi \sin \theta & -\sin \varphi \sin \theta & 0 \\ 0 & \sin \varphi & -\cos \varphi & 0 \\ 0 & 0 & \rho & 1 \end{bmatrix} \quad (1)$$

Для перспективных преобразований используется параметр d , который задаёт расстояние между экраном и точкой наблюдения. Чтобы реализовать данное преобразование была написана функция `void createPerspectiveView()`.

Перспективные преобразования представлены формулой (2):

$$X = d \frac{x}{z} \quad (2.1) \quad Y = d \cdot \frac{y}{z} \quad (2.2)$$

Экранный преобразование осуществляет коррекцию координат с учётом таких параметров, как текущее разрешение графического дисплея, сдвига по осям, направления оси ординат. Функция `void moveScreen()` реализует экранное преобразование, смещает координаты на определенное значение переменной, чтобы объект отображался ближе к центру окна приложения.

Экранные преобразования представлены формулой (3):

$$X = d \frac{x}{z} + c_1 \quad (3.1) \quad Y = d \cdot \frac{y}{z} + c_2 \quad (3.2)$$

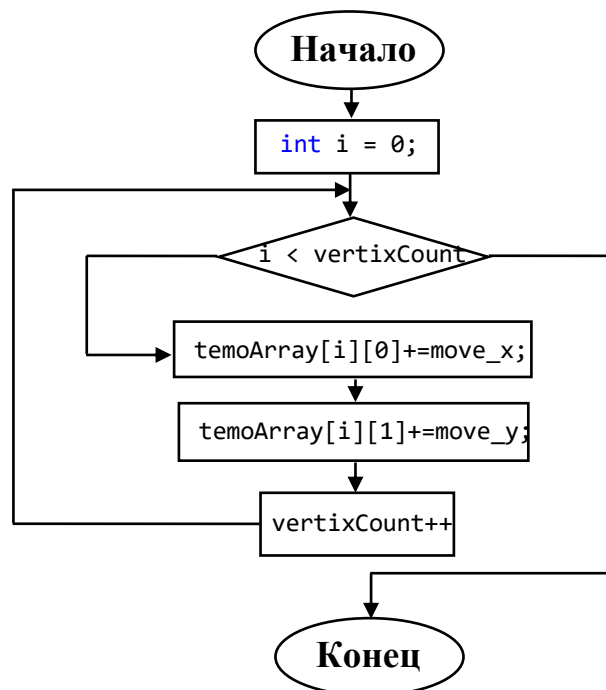


Рисунок 16 – Блок-схема функции `void moveScreen()`

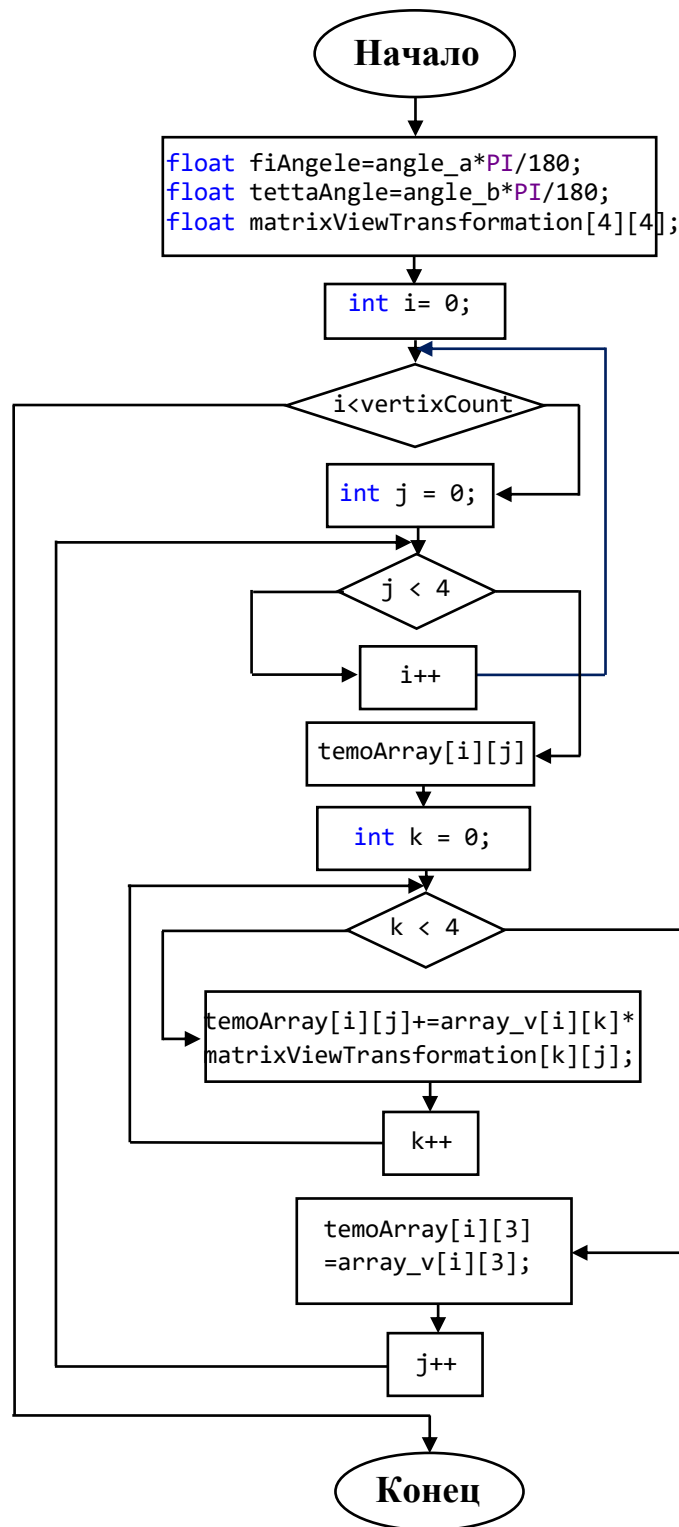


Рисунок 17 – Блок-схема функции void viewTransformation()

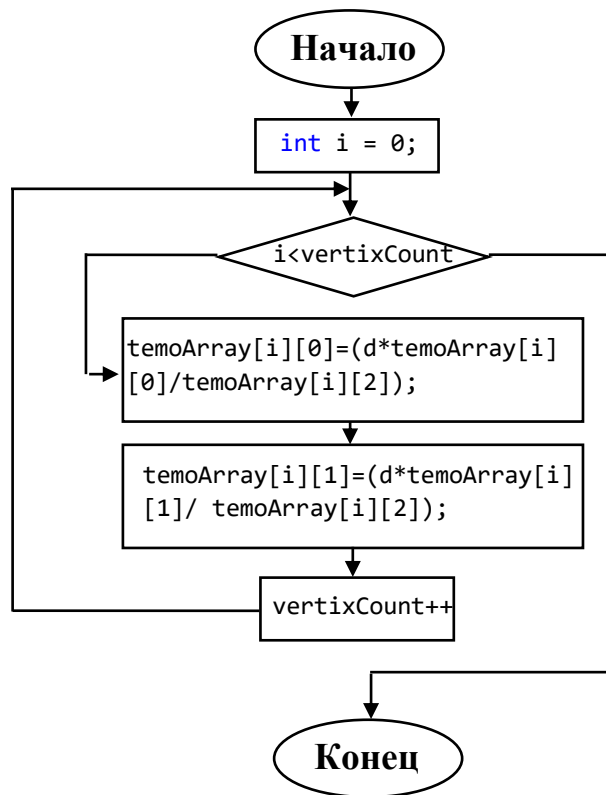


Рисунок 18 – Блок-схема функции void createPerspectiveView() Реализация интерфейса в отношении управления объектом или камерой в сцене

Управление параметрами динамики объекта производится с помощью клавиатуры:

↑, ↓, ←, → – уменьшение/увеличение углов поворота.

F1, F2 – уменьшение/увеличение расстояния от точки наблюдения до начала мировых координат.

F3, F4 – уменьшение/увеличение расстояния до экранной плоскости.

W, E – уменьшение/увеличение коэффициента масштабирования.

1.7 Отладка и тестирование программы

Таблица 1 – Описание поведения программы при тестировании

Описание теста	Ожидаемый результат	Полученный результат
Загрузка существующего файла	Импорт файла, рисование модели	Верно
Нажатие клавиш изменения углов поворота.	Поворот проекции.	Верно
Нажатие клавиш изменения расстояния от точки наблюдения до начала координат.	Увеличение, уменьшение расстояния от точки наблюдения до начала координат.	Верно
Нажатия клавиш изменения расстояния до экранной плоскости.	Увеличение, уменьшение расстояния экранной плоскости.	Верно
Нажатие клавиш изменения коэффициента масштабирования.	Увеличение, уменьшение объекта.	Верно

В результате тестирования было выявлено, что программа успешно выполняет свои функции.

ЗАКЛЮЧЕНИЕ

В результате выполнения работы была разработана программа построения каркасной модели 3D объекта в Microsoft Visual Studio на языке C++.

Созданная программа позволяет выполнять следующие действия:

- Увеличение и уменьшение расстояния от точки наблюдения до начала координат.
- Увеличение и уменьшение расстояния экранной плоскости.
- Увеличение и уменьшение модели.

Работа по созданию программы позволила приобрести практические навыки программирования по работе с компьютерной графикой и 3D моделированием.

На основе контрольных примеров получились верные результаты, что позволяет сделать вывод о правильной реализации алгоритма программы.

СПИСОК ЛИТЕРАТУРЫ

1. Объектно-ориентированное программирование в C ++ 4 -е изд. Лафоре Роберт
Издательство Санкт-Петербург, 2016. – 928 с.
2. Л. Аммерал \ 62 Принципы программирования в машинной графике. Пер. с
англ. —М.: "Сол Систем", 1992.— 224 с

Электронные ресурсы:

3. [Студопедия // Рисование точки \[Электронный ресурс\]. – URL.:
https://studopedia.ru/16_70419_risovanie-tochki.html](https://studopedia.ru/16_70419_risovanie-tochki.html) (дата обращения
25.04.2023).
4. [MathWorks // фи/тета-углов в координаты u/v – \[Электронный ресурс\]. –URL.:
https://www.mathworks.com/help/phased/ref/phitheta2uv.html](https://www.mathworks.com/help/phased/ref/phitheta2uv.html) (дата обращения
27.04.2023).

ПРИЛОЖЕНИЕ А. ЛИСТИНГ ПРОГРАММЫ

```
#define _CRT_SECURE_NO_WARNINGS
#include "framework.h"
#include "math.h"
#include "stdio.h"
#include "tchar.h"
#include "conio.h"
#include "stdlib.h"
#include "windows.h"
#include "string.h"
#include "Resource.h"
#include <iostream>

#define MAX_LOADSTRING 100
#define PI 3.1415

// Глобальные переменные
HINSTANCE hInst; // текущий экземпляр
WCHAR szTitle[MAX_LOADSTRING]; // Текст строки заголовка
WCHAR szWindowClass[MAX_LOADSTRING]; // имя класса главного окна

// Отправить объявления функций, включенных в этот модуль кода
ATOM MyRegisterClass(HINSTANCE hInstance);
BOOL InitInstance(HINSTANCE, int);
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

//Создаем форму приложения
int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
                     _In_opt_ HINSTANCE hPrevInstance,
                     _In_ LPWSTR lpCmdLine,
                     _In_ int nCmdShow)
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);

    // TODO: Разместите код здесь.
    // Инициализация глобальных строк
    LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadStringW(hInstance, IDS_CAR, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);

    // Выполнить инициализацию приложения
    if (!InitInstance(hInstance, nCmdShow))
    {
        return FALSE;
    }

    HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDS_CAR));
    MSG msg;
    // Цикл основного сообщения
    while (GetMessage(&msg, nullptr, 0, 0))
    {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }
}
```

```

        return (int)msg.wParam;
    }
    //Инициализируем форму приложения
    ATOM MyRegisterClass(HINSTANCE hInstance)
    {
        WNDCLASSEXW wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);
        wcex.style = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc = WndProc;
        wcex.cbClsExtra = 0;
        wcex.cbWndExtra = 0;
        wcex.hInstance = hInstance;
        wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDS_CAR));
        wcex.hCursor = LoadCursor(nullptr, IDC_ARROW);
        wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
        wcex.lpszMenuName = MAKEINTRESOURCEW(IDS_CAR);
        wcex.lpszClassName = szWindowClass;
        wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));
        return RegisterClassExW(&wcex);
    }

    //Стандартная функция, для отображения формы в с++
    BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
    {
        hInst = hInstance; // Сохранить маркер экземпляра в глобальной переменной

        HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
            CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);
        if (!hWnd)
        {
            return FALSE;
        }

        ShowWindow(hWnd, nCmdShow);
        UpdateWindow(hWnd);

        return TRUE;
    }

    const char fileName[10] = "Car.obj"; //название файла, которого загружаем
    int vertixCount = 0; //Количество вершин, которое есть в файле
    int lineCount = 0; //Количество линий
    float angleY = 0.0; // угол вращения по X. Полярные углы Альфа и Бетта. Горизонтальной и
    вертикальной оси
    float angleX = 180.0;
    float R = 450.0; //расстояние от точки наблюдения до начала мировой системы координат
    float d = 500.0; //расстояние от точки до центральной плоскости

    int scale = 50; //размер объекта

    float** array_v; // двумерный массив всех вершин
    float** array_l; //двумерный массив всех граней
    float** temoArray; //все координаты, над которыми производим преобразования

    char str[100];
    FILE* carFile; //стандартный файл, чтобы открыть .obj

    float move_y = 300; // расстояние по y
    float move_x = 300; //расстояние по x

```

```

//считываем количество вершин и линий
void calculateVertexAndLineCount() {
    carFile = fopen(fileName, "r");
    while ((fscanf(carFile, "%s", str) != EOF)) { //EOF -End of file
        if (!carFile) break;
        if (str[0] == 'v') { //строка начинается на v - вершина
            vertexCount++;
        }
        if (str[0] == 'l') {
            lineCount++;
        }
    }
    fclose(carFile);
};

//отвод памяти для матрицы. Возвращаем указатель на матрицу.
float** allocateDynamicMemory(int SIZE) {
    float** mass;
    mass = (float**)malloc(SIZE * sizeof(float*));
    for (int i = 0; i < SIZE; i++) {
        mass[i] = (float*)malloc(4 * sizeof(float));
    };
    return mass;
};

//читаем вершины
void readVertex() {
    int i = 0;
    carFile = fopen(fileName, "r");
    while ((fscanf(carFile, "%s", str) != EOF)) {
        if (!carFile) break;
        if (str[0] == 'v') {
            for (i; i < vertexCount; i++) {
                for (int j = 0; j < 3; j++) {
                    fscanf(carFile, "%f", &array_v[i][j]);
                }
                break;
            }
            i++;
        }
    }
    fclose(carFile);
}

//Читаем количество граней
void readEdges() {
    int i = 0;
    carFile = fopen(fileName, "r");
    while ((fscanf(carFile, "%s", str) != EOF)) {
        if (!carFile) break;
        if (str[0] == 'l') {
            for (i; i < lineCount; i++) {
                for (int j = 0; j < 2; j++) {
                    fscanf(carFile, "%f", &array_l[i][j]);
                }
                break;
            }
            i++;
        }
    }
}

```

```

        fclose(carFile);
    }

    //для указания изначального scale
    void createOneArray() {
        for (int i = 0; i < vertexCount; i++) {
            for (int j = 0; j < 4; j++) {
                array_v[i][j] = 1;
            }
        }
    }

    //Функция переноса из 3д-мира в новую систему координат
    void viewTransformation() {
        //перевод в радианы для функций cos и sin
        float fiAngle = angleY * PI / 180;
        float tettaAngle = angleX * PI / 180;

        //формула видового преобразования объекта в пользовательскую систему координат.
        //Где R - является центром
        float matrixViewTransformation[4][4]{
            {-sin(tettaAngle) , -cos(fiAngle) * cos(tettaAngle) , -sin(fiAngle) *
            cos(tettaAngle) , 0},
            { cos(tettaAngle) , -cos(fiAngle) * sin(tettaAngle) , -sin(fiAngle) *
            sin(tettaAngle) , 0},
            { 0 , sin(fiAngle) , -cos(fiAngle) , 0},
            { 0 , 0 , R , 1}
        };

        for (int i = 0; i < vertexCount; i++) {
            for (int j = 0; j < 4; j++) {
                temoArray[i][j] = 0; //умножение матриц. Одна является строкой из 3
                //координат и 1. вторая - наша матрица
                //пример
                // x = 1 y = 2 z = 4 W = 1 (W - один, чтобы при умножении
                //на последнюю координату матрицы matrixViewTransformation эта
                //координата не занулялась)
                for (int k = 0; k < 4; k++) {
                    temoArray[i][j] += array_v[i][k] *
                    matrixViewTransformation[k][j]; //умножаем на каждую координату
                }

                temoArray[i][3] = array_v[i][3]; // Возвращаем W в 1 , чтобы при
                //следующих трансформациях последний элемент матрицы учитывался
            }
        }

        //В цикле for, который выполняется для каждой вершины объекта(где vertexCount - количество
        //вершин), происходит следующее :
        //
        //Для каждой вершины объекта выполняется перспективное преобразование по осям x и y.
        //
        //Координаты x и y вершины преобразуются, делением на глубину z(temoArray[i][2]), и
        //умножением на коэффициент d.
        //
        //Результат преобразования сохраняется обратно в temoArray[i][0] и temoArray[i][1].

        void createPerspectiveView() {
            for (int i = 0; i < vertexCount; i++) {
                temoArray[i][0] = (d * temoArray[i][0] / temoArray[i][2]);
            }
        }
    }

```

```

        temoArray[i][1] = (d * temoArray[i][1] / temoArray[i][2]);
    }
}

void moveScreen() {
    for (int i = 0; i < vertexCount; i++) {
        temoArray[i][0] += move_x; // смещение по X для каждой координаты
        temoArray[i][1] += move_y; // смещение по Y для каждой координаты
    }
}

//меняем scale объекта. Каждую координату умножаем на коэффициент scale.
void changeScale() {
    for (int i = 0; i < vertexCount; i++) {
        for (int j = 0; j < 3; j++) {
            array_v[i][j] *= scale;
        }
    }
}

//Отрисовываем объект
//Номер вершины vertex берется из массива array_1, который хранит информацию о гранях и
их вершинах.
//
//Если vertex положительный, это означает, что грань связана с вершиной, иначе она
игнорируется.
//
//Если j(индекс внутреннего цикла) равен 0, это означает, что текущая итерация отвечает
за первую точку грани. В этом случае функция MoveToEx используется для установки начальной точки
рисования(перемещение к указанным координатам).
//
//Значения координат x и y текущей вершины(temoArray[vertex][1] и temoArray[vertex][0])
// сохраняются в переменные vertexY и vertexX соответственно.//
//Если j не равно 0, то это означает, что текущая итерация отвечает
// за вторую точку грани. В этом случае функция LineTo используется
// для рисования линии от предыдущей точки к текущей точке грани(temoArray[vertex][1] и
temoArray[vertex][0]).
//
//После завершения внутреннего цикла, используется функция
//LineTo для проведения линии от последней точки грани к начальной точке грани(vertexY и
vertexX).
//Это позволяет замкнуть контур грани.
void paint(HDC hdc) { // hdc - контекст девайса. Позволяет обращаться к Win Api
    int vertex = 0;
    float vertexX, vertexY;
    int countSheet = 0;
    for (int i = 0; i < lineCount; i++) { //проходим по количеству граней.
        for (int j = 0; j < 2; j++) { //2 потому что у грани есть только 2 точки.
            vertex = array_1[i][j]; // берем номер вершины

            if (vertex > 0) {
                if (j == 0) {
                    MoveToEx(hdc, temoArray[vertex][1],
temoArray[vertex][0], NULL); // ставим перо, для начала рисования
                    vertexY = temoArray[vertex][1]; // запоминаем
координаты начала
                    vertexX = temoArray[vertex][0];
                }
                else {
                    LineTo(hdc, temoArray[vertex][1],
temoArray[vertex][0]); // рисуем линию до 2 точки
                }
            }
        }
    }
}

```

```

    }
    }
    LineTo(hdc, vertexY, vertexX); // рисуем в обратную
}

//Win API для обработки нажатия клавиш
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
    case WM_KEYDOWN:
    {
        switch (LOWORD(wParam))
        {
        case VK_RIGHT:
        {
            angleY = angleY + 15.0;
            InvalidateRect(hWnd, NULL, TRUE); // запрос на обновления экрана
            break;
        }
        case VK_LEFT:
        {
            angleY = angleY - 15.0;
            InvalidateRect(hWnd, NULL, TRUE);
            break;
        }
        case VK_UP:
        {
            angleX = angleX + 5.0;
            InvalidateRect(hWnd, NULL, TRUE);
            break;
        }
        case VK_DOWN:
        {
            angleX = angleX - 5.0;
            InvalidateRect(hWnd, NULL, TRUE);
            break;
        }
        case VK_F1:
        {
            R = R + 15.0; // отдаляем камеру
            InvalidateRect(hWnd, NULL, TRUE); //
            break;
        }
        case VK_F2:
        {
            R = R - 15.0; //приближаем
            InvalidateRect(hWnd, NULL, TRUE);
            break;
        }
        case VK_F3:
        {
            d = d + 100.0; // Коэффициент d из формулы
            InvalidateRect(hWnd, NULL, TRUE);
            break;
        }
        case VK_F4:
        {
            d = d - 100.0;
            InvalidateRect(hWnd, NULL, TRUE);
        }
    }
    }
}

```

```

        break;
    }
    case 'S':
    {
        scale = scale + 5.0; //увеличиваем объект
        InvalidateRect(hWnd, NULL, TRUE);
        break;
    }

    case 'W':
    {
        scale = scale - 5.0; //уменьшаем объект
        InvalidateRect(hWnd, NULL, TRUE);
        break;
    }
}
case WM_COMMAND:
{
    int wmId = LOWORD(wParam);
    // Разобрать выбор в меню:
    switch (wmId)
    {
        case IDM_ABOUT:
            DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
            break;
        case IDM_EXIT:
            DestroyWindow(hWnd);
            break;
        default:
            return DefWindowProc(hWnd, message, wParam, lParam);
    }
}
break;
case WM_PAINT:
{
    PAINTSTRUCT ps;
    HDC hdc = BeginPaint(hWnd, &ps);
    // TODO: Добавьте сюда любой код прорисовки, использующий HDC...
    calculateVertexAndLineCount();
    array_v = allocateDynamicMemory(vertexCount);
    array_l = allocateDynamicMemory(lineCount);
    createOneArray();
    temoArray = allocateDynamicMemory(vertexCount);
    readVertex();
    changeScale();
    readEdges();
    viewTransformation();
    createPerspectiveView();
    moveScreen();
    paint(hdc);
    EndPaint(hWnd, &ps);
    free(array_v);
    free(array_l);
    free(temoArray);
}
break;
case WM_DESTROY:
    PostQuitMessage(0);
    break;
default:

```

```

        return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
// Обработчик сообщений для окна "О программе".
INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
    UNREFERENCED_PARAMETER(lParam);
    switch (message)
    {
    case WM_INITDIALOG:
        return (INT_PTR)TRUE;

    case WM_COMMAND:
        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
        {
            EndDialog(hDlg, LOWORD(wParam));
            return (INT_PTR)TRUE;
        }
        break;
    }
    return (INT_PTR)FALSE;
}

```

ПРИЛОЖЕНИЕ Б. ГРАФИЧЕСКАЯ ЧАСТЬ. ЧЕРТЕЖИ ТРЁХ ПРОЕКЦИЙ ОБЪЕКТА.

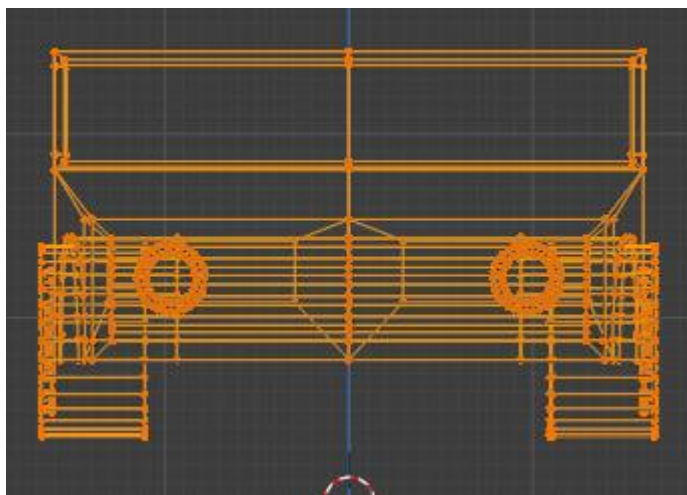


Рисунок 19 – Чертёж на проекции Y

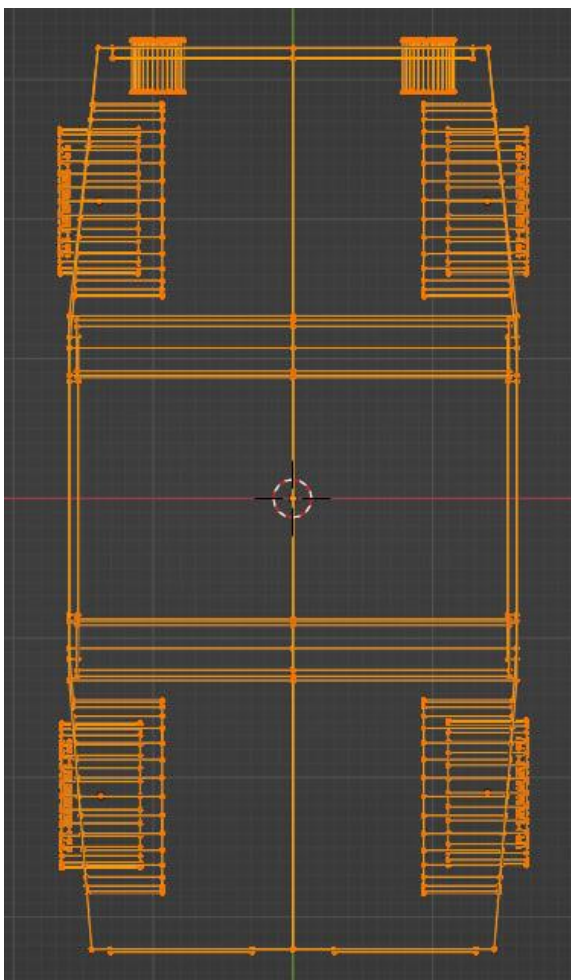


Рисунок 20 – Чертёж на проекции Z

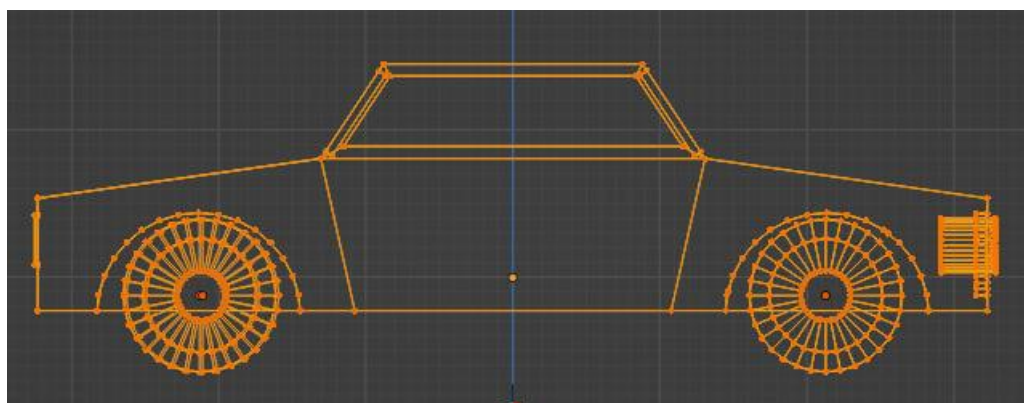


Рисунок 21 – Чертёж на проекции X

ПРИЛОЖЕНИЕ В. ОКНО ПРОГРАММЫ.ВИЗУАЛИЗАЦИЯ ОБЪЕКТА И КАМЕРЫ.

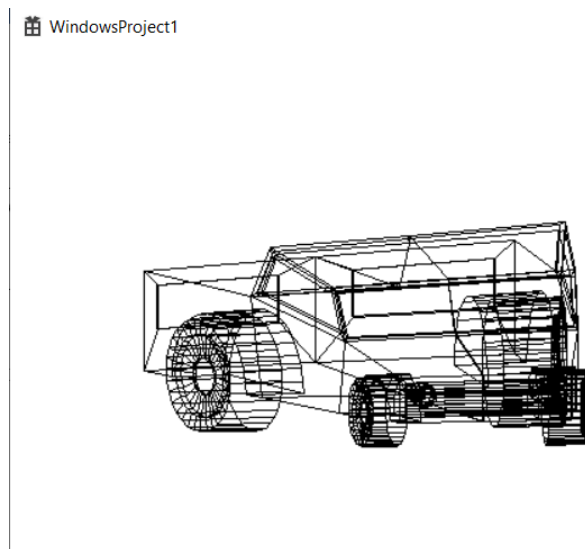


Рисунок 22 – Загрузка файла

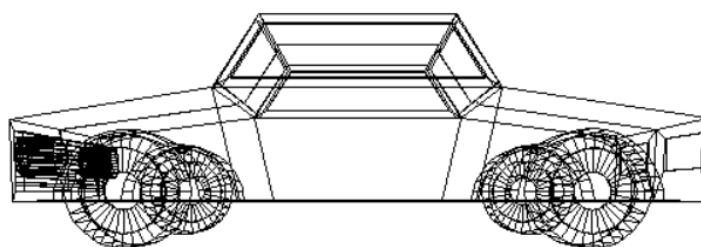


Рисунок 23 – Вид сбоку

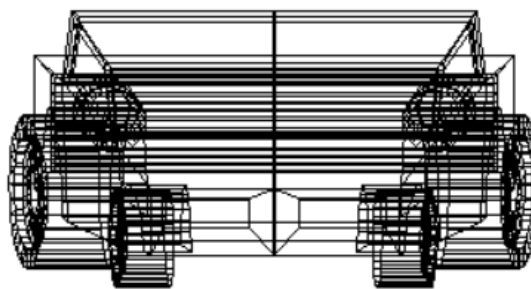


Рисунок 24 – Вид спереди

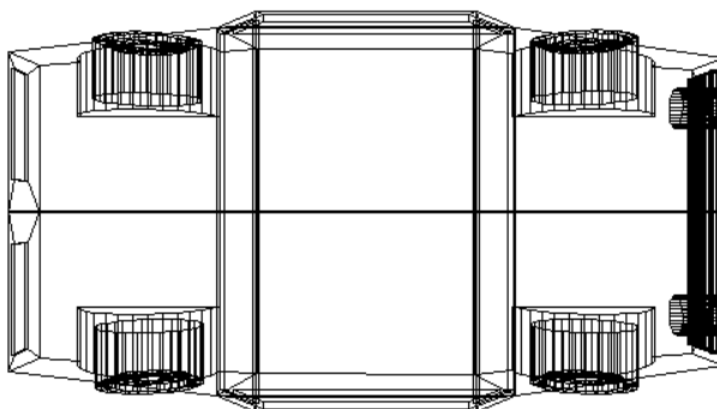


Рисунок 25 – Вид сверху

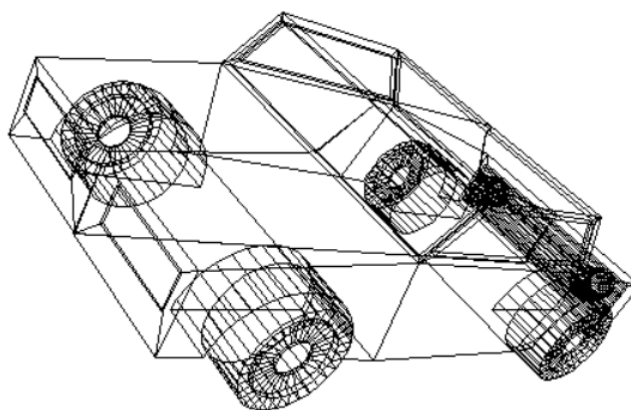


Рисунок 26 – Произвольный угол

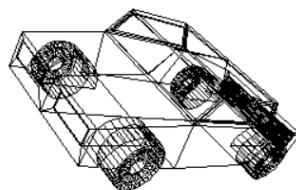


Рисунок 27 – Увеличение расстояния

WindowsProject1

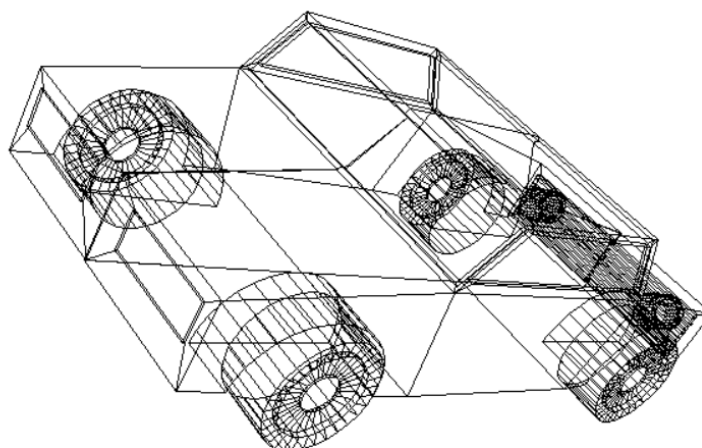


Рисунок 28 – Уменьшение расстояния

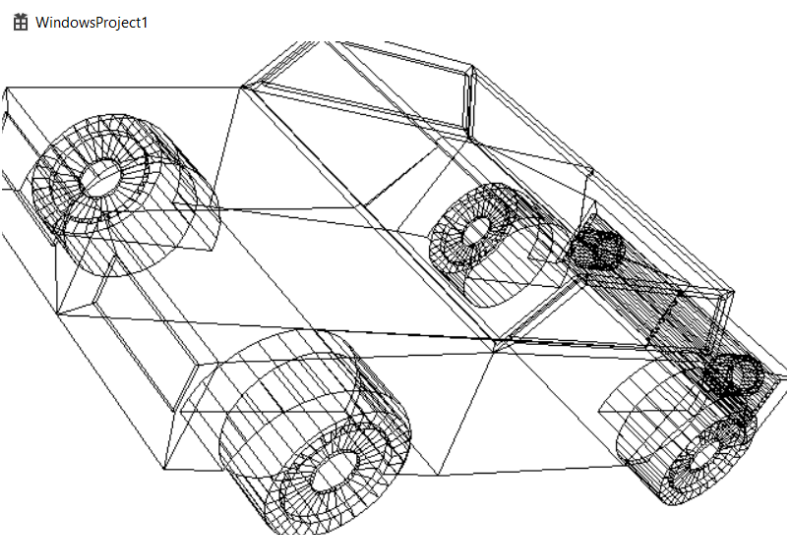


Рисунок 29 – Увеличение расстояния до экранной плоскости

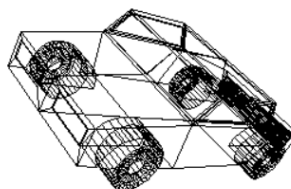


Рисунок 30 – Уменьшение расстояния до экранной плоскости

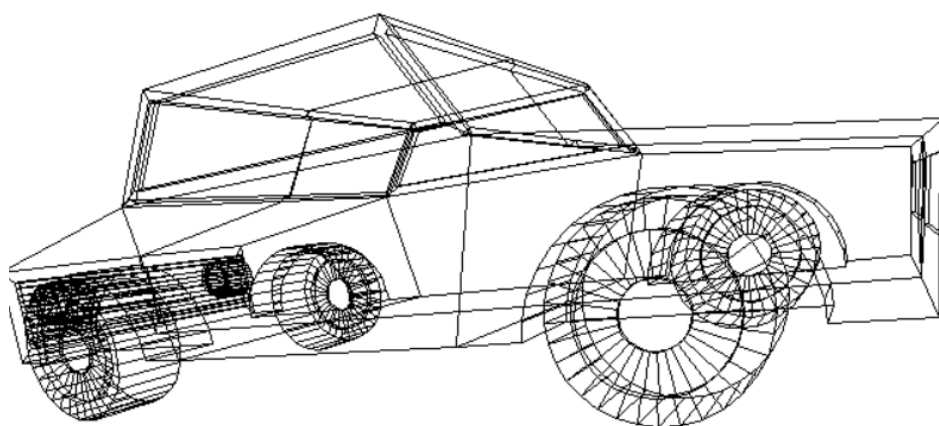


Рисунок 31 – Увеличение коэффициента

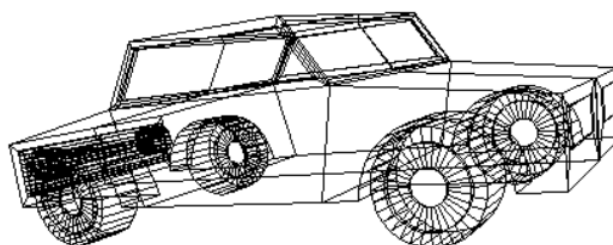


Рисунок 32 – Уменьшение коэффициента