

Министерство науки и высшего 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 интерактивной программы построения и визуализации
каркасной модели 3D объекта»

Выполнил:

студент группы 21ВВС1

Нагорная Д.А.

Принял:

к.т.н., доцент Финогеев А. А.

Пенза 2023

Содержание

Введение.....	5
1 Расчетная часть.....	6
1.1 Анализ и описание процесса синтеза объекта методом твердотельного моделирования....	6
1.2 Декомпозиция на графические примитивы, аппроксимация гранями.....	11
1.3 Описание расположения объекта/сцены в системе координат.....	12
1.4 Импортирование созданных моделей и ландшафта в программную среду	13
1.5 Разработка программы визуализации объекта/сцены.....	17
1.6 Программная реализация динамики объекта.....	18
1.7 Реализация интерфейса в отношении управления объектом или камерой в сцене	20
1.8 Отладка и тестирование программы	20
2 Графическая часть.....	25
2.1. Чертежи трех проекций	25
2.2. Структурная схема алгоритма.....	26
3 Экспериментальная часть.....	27
3.1. Тестирование и верификация программного обеспечения	27
Заключение	28
Список используемых источников.....	29
ПРИЛОЖЕНИЕ А.....	30
Листинг программы	30

Введение

На данный момент времени компьютерная графика стала основным средством связи между человеком и компьютером, постоянно расширяющим сферы своего применения, так как в графическом виде результаты становятся более наглядными и понятными. Без компьютерной графики не обходится ни одна современная программа.

Трехмерная графика является разделом компьютерной графики, в которой применяется представление объектов в трех измерениях, призванная обеспечить пространственно-временную непрерывность получаемых изображений.

Программные пакеты, позволяющие производить трехмерную графику, то есть моделировать объекты виртуальной реальности и создавать на основе этих моделей изображения, очень разнообразны, многофункциональны и постоянно дорабатываются, совершенствуются и видоизменяются. В последние годы устойчивыми лидерами в этой области являются коммерческие продукты, такие как Autodesk 3ds Max, Maya, Newtek Lightware 3D, SoftImage XSI, и сравнительно новые, например, Rhinoceros 3D, Cinema 4D. Кроме того, уверенно набирают популярность и открытые продукты, распространяемые свободно, а именно полнофункциональный пакет Blender, который позволяет выполнить и производство моделей, и последующий рендеринг.

Компьютерная графика и 3D моделирование являются активно развивающимися технологиями, используемыми для визуализации или представления идей и изображений в трехмерном пространстве. Несмотря на то, что исследования в области классической компьютерной графики связаны с проектами рендеринга фильмов и игр на персональных компьютерах, исследования в области 3D моделирования направлены на инструменты и процессы для визуализации реальных сцен.

1 Расчетная часть

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

Разработка 3D-модели осуществлялась в среде разработки трехмерной графики Blender.

В качестве основы вертолета был создан удлиненный прямоугольный параллелепипед (рисунок 1).

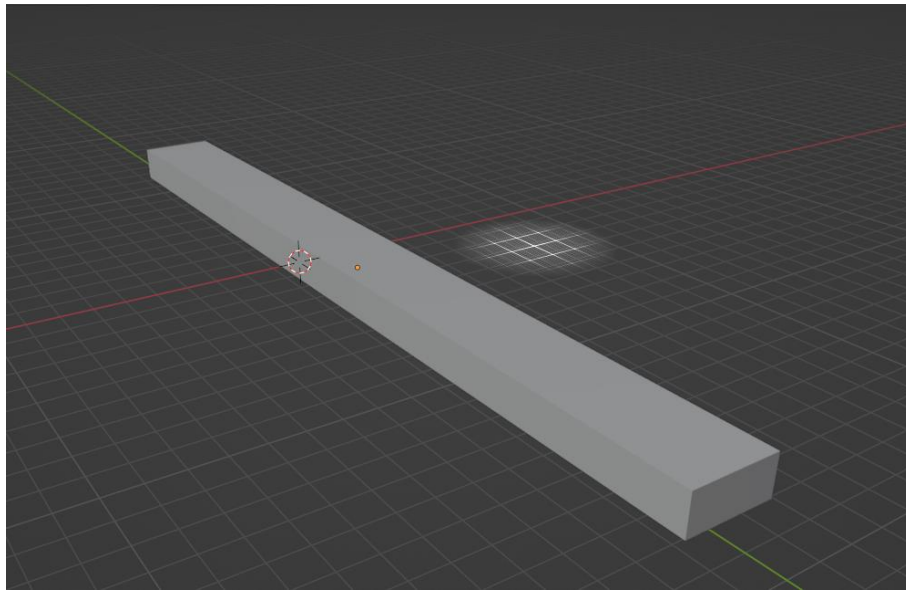


Рисунок 1 – Создание прямоугольного параллелепипеда

В режиме редактирования с помощью инструмента Loop Cut выполнен петлевой разрез (рисунок 2).

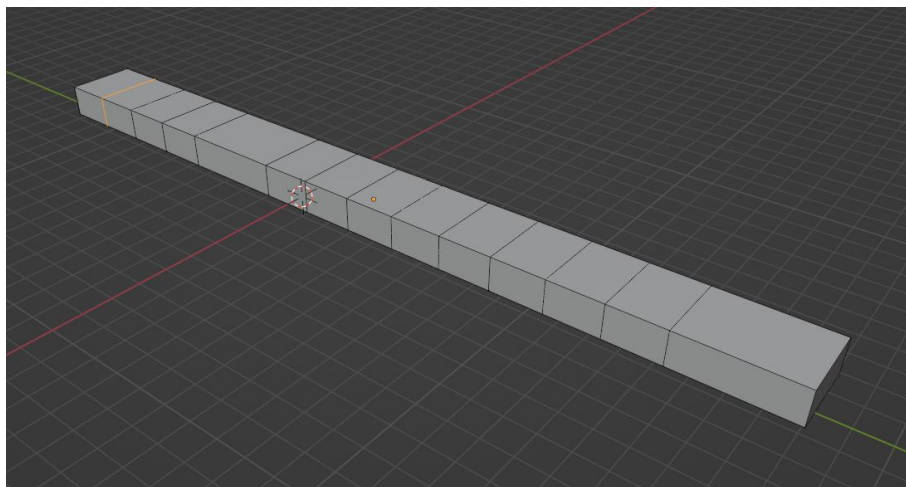


Рисунок 2 – Петлевой разрез модели

В том же режиме посредством инструмента Extrude Region и механизма Move было произведено экструдирование граней и перемещение ребер (рисунок 3).

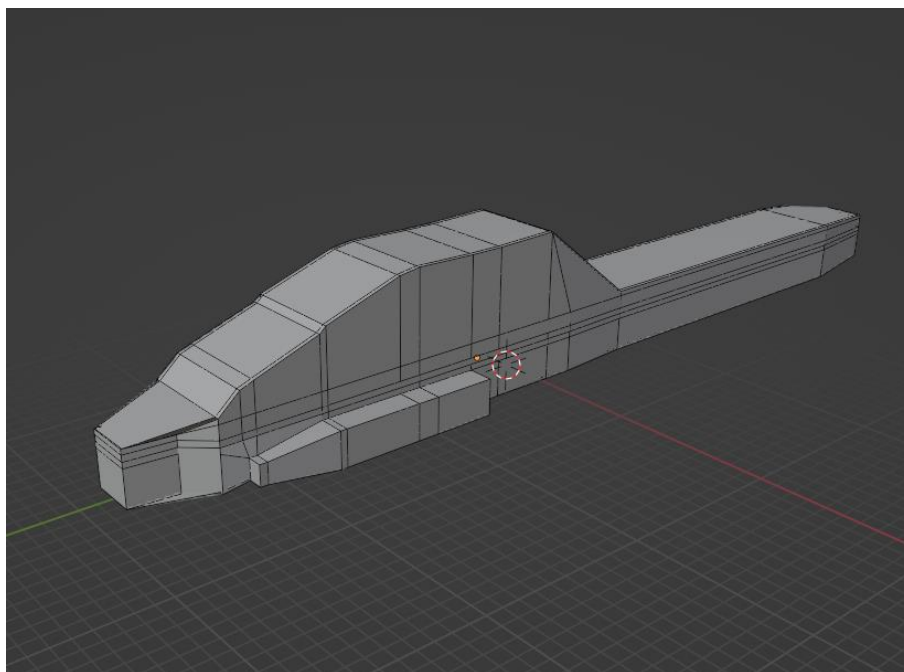


Рисунок 3 – Представление корпуса вертолета

Путем простых операций были построены хвостовое оперение и рулевой винт вертолета (рисунок 4).

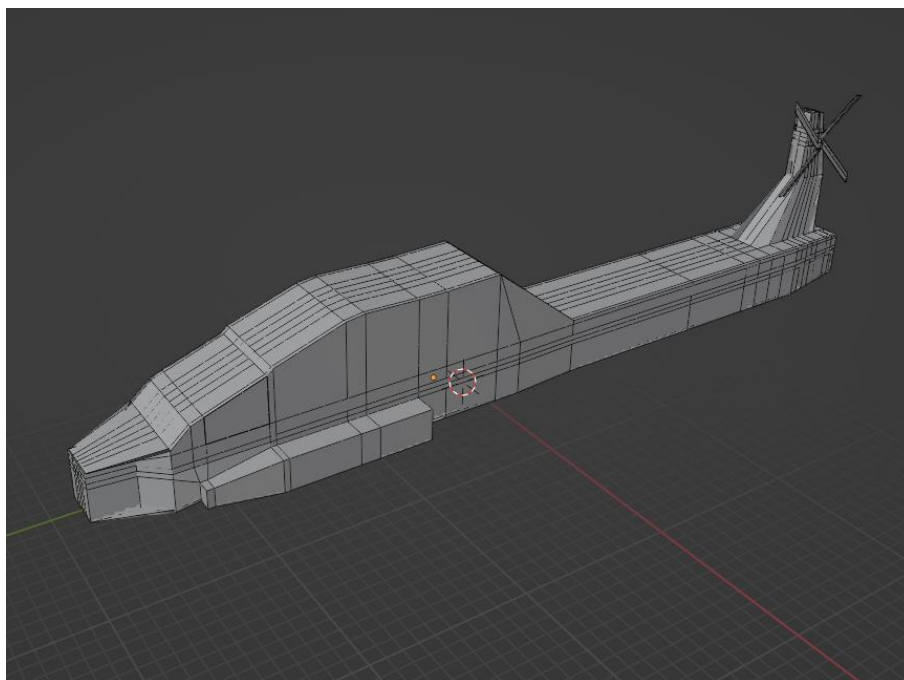


Рисунок 4 – Построение рулевого винта

С помощью добавления нового объекта на вкладке Modifier Properties был выбран инструмент Boolean для создания отверстия для окон (рисунок 5).

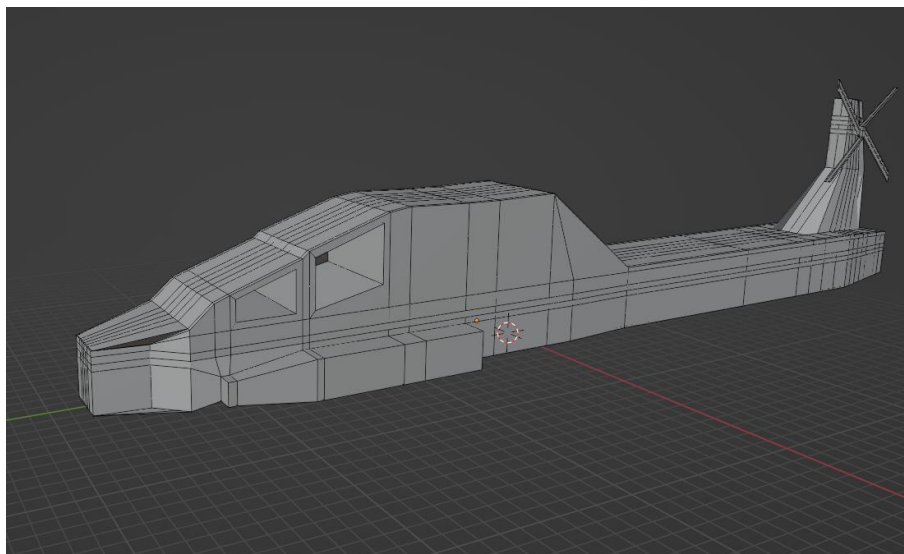


Рисунок 5 – Создание окон

Для нашей 3D-модели было осуществлено построение двигателя и спонсона (рисунок 6).

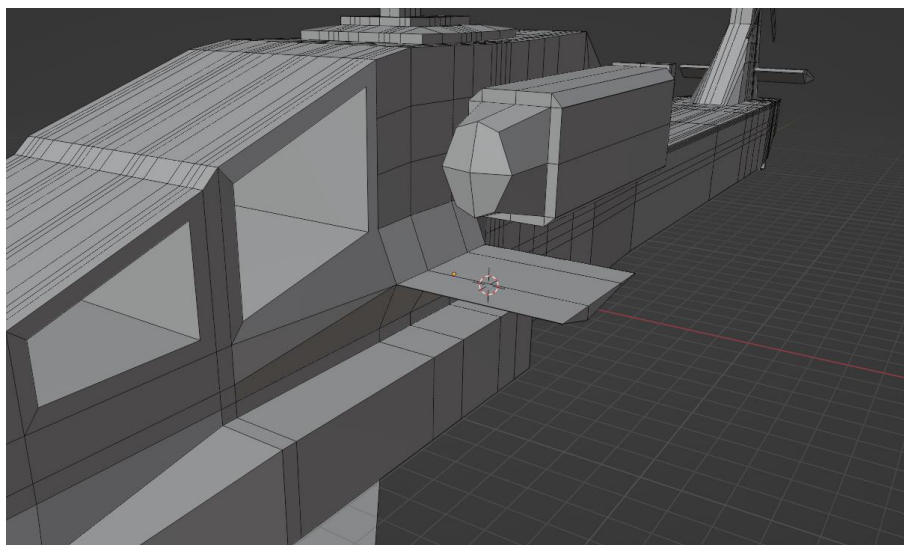


Рисунок 6 – Построение двигателя и спонсона

Далее в модель был возведен несущий винт (рисунок 7).

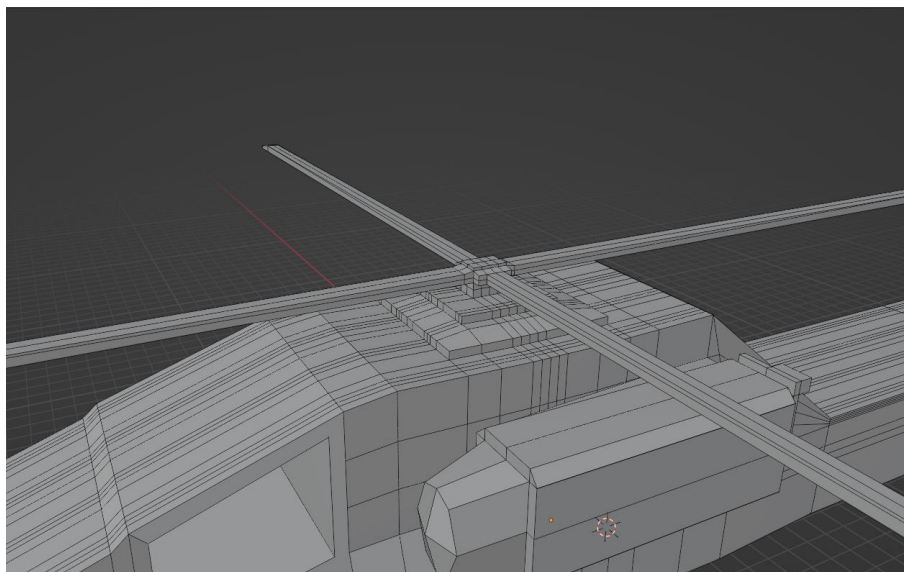


Рисунок 7 – Построение несущего винта

Напоследок были смоделированы колеса передней и основной опоры шасси (рисунок 8).

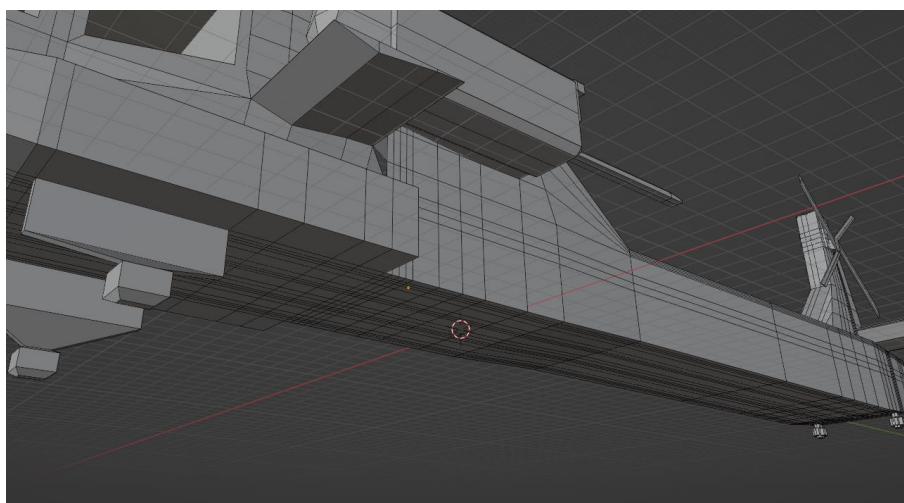


Рисунок 8 – Построение колес

В результате проделанных действий была спроектирована 3D-модель, представленная на рисунке 9.

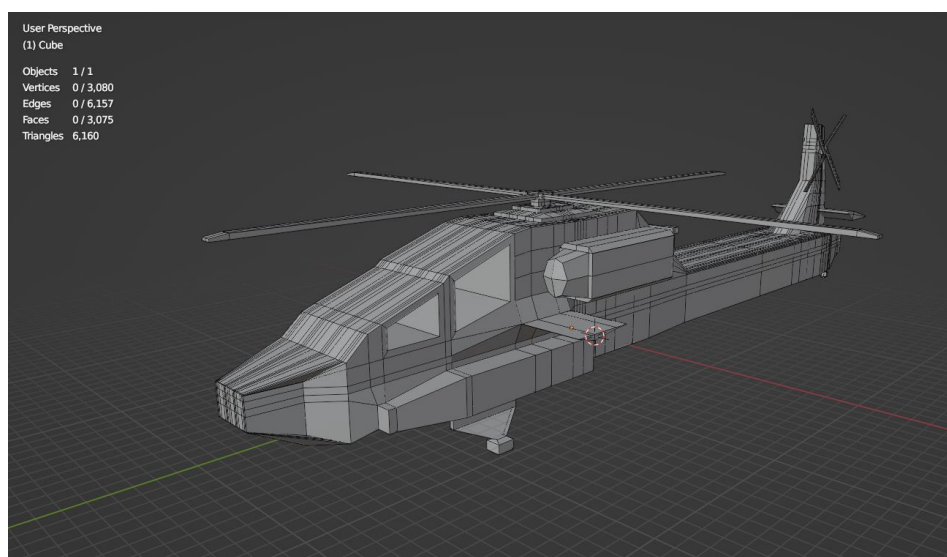


Рисунок 9 – Предварительный результат

Из рисунка 9 видно, что полученная модель имеет более трех тысяч вершин. Для того, чтобы сократить данное число, были произведены некоторые действия. С помощью инструмента Limited Dissolve были удалены выбранные ребра и, соответственно, вершины (рисунок 10).

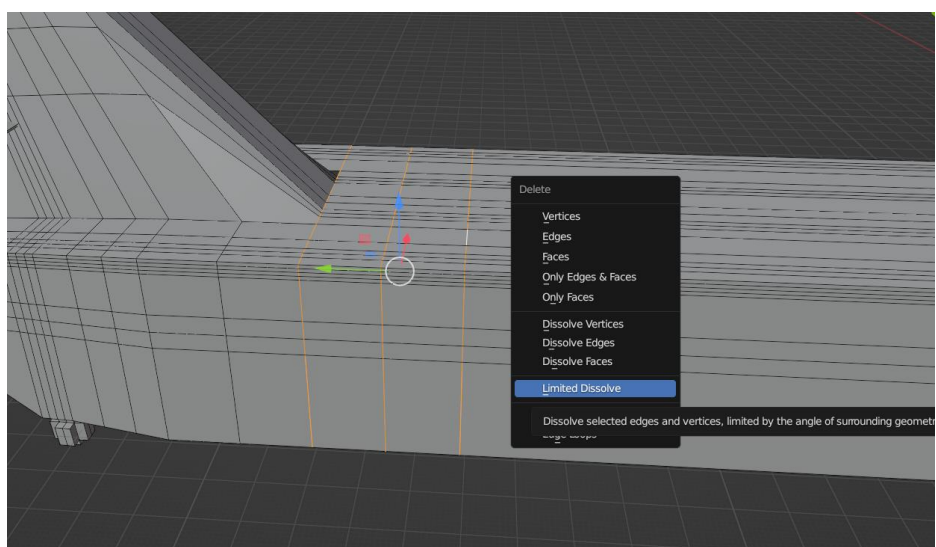


Рисунок 10 – Сокращение вершин

На рисунке 11 был показан окончательный результат. Представленная модель имеет 1327 вершин.

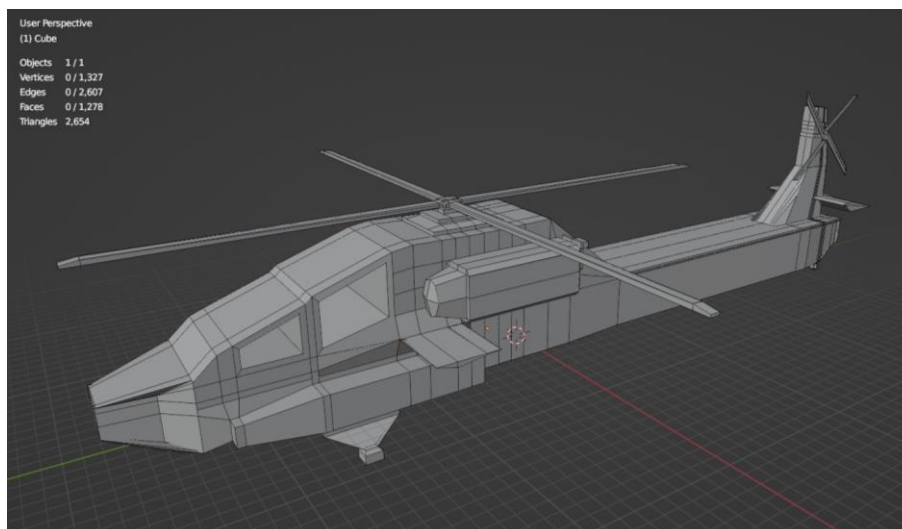


Рисунок 11 – Результат

1.2 Декомпозиция на графические примитивы, аппроксимация гранями

На рисунке 12 представлена декомпозиция модели на графические примитивы.



Рисунок 12 – Декомпозиция

На рисунке 13 представлена аппроксимация гранями.

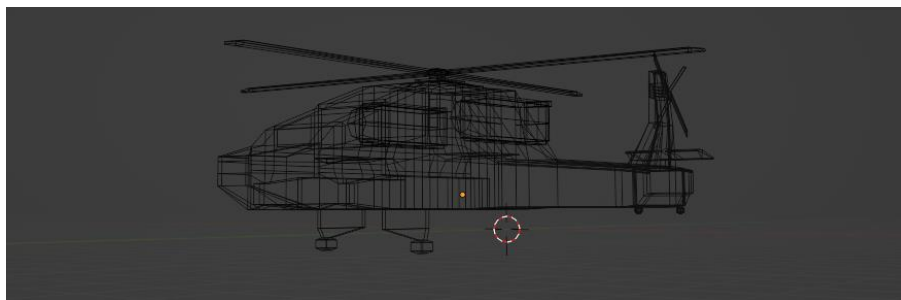


Рисунок 13 – Аппроксимация гранями

1.3 Описание расположения объекта/сцены в системе координат

В данном разделе были описаны расположения объекта в системе координат.

На рисунке 14 представлена проекция объекта на плоскость (x, y) .

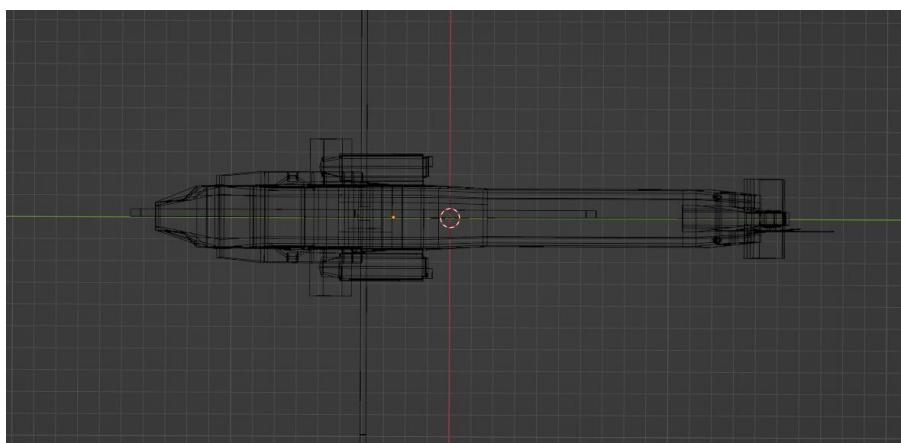


Рисунок 14 – Расположение сверху

На рисунке 15 представлена проекция объекта на плоскость (y, z) .

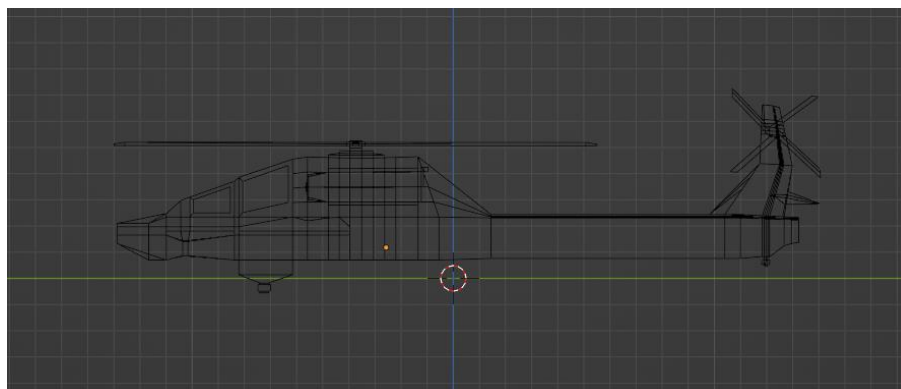


Рисунок 15 – Расположение сбоку

На рисунке 16 представлена проекция объекта на плоскость (x, z).

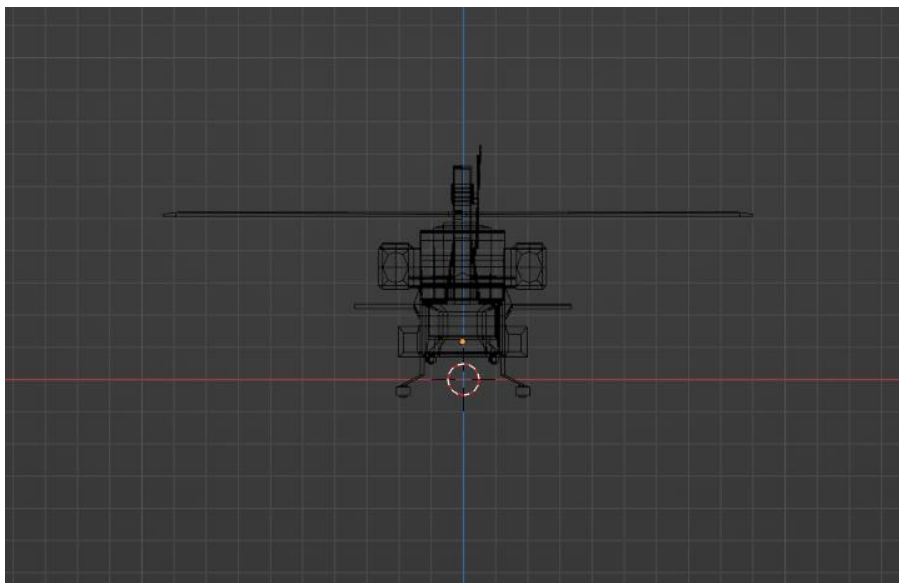


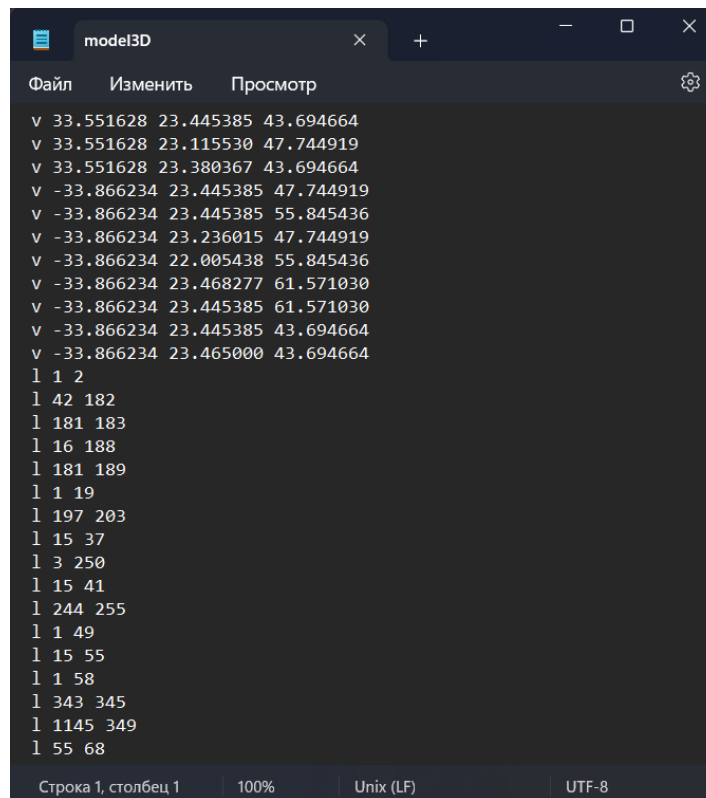
Рисунок 16 – Расположение спереди

1.4 Импортирование созданных моделей и ландшафта в программную среду

Перед импортированием в программную среду, созданную модель экспортировали в формате obj. Данный формат содержит информацию о геометрии объекта: координаты вершин, ребер, текстуры, нормали и параметры одной поверхности объекта.

Так как каркасная модель является моделью объекта в трехмерной графике, представляющая совокупность вершин и ребер, необходимо в нашем файле представить только их. Перед экспортированием в режиме редактирования была выделена вся модель и нажатием на клавишу X выбран инструмент Limited Dissolve, чтобы отобразить каркасную модель объекта. Только после этого модель сохранена в obj файл, в котором содержатся информация о вершинах (v) и порядке соединения этих вершин, то есть ребрах (l).

На рисунке 17 было представлено содержимое obj файла.



```
v 33.551628 23.445385 43.694664
v 33.551628 23.115530 47.744919
v 33.551628 23.380367 43.694664
v -33.866234 23.445385 47.744919
v -33.866234 23.445385 55.845436
v -33.866234 23.236015 47.744919
v -33.866234 22.005438 55.845436
v -33.866234 23.468277 61.571030
v -33.866234 23.445385 61.571030
v -33.866234 23.445385 43.694664
v -33.866234 23.465000 43.694664
l 1 2
l 42 182
l 181 183
l 16 188
l 181 189
l 1 19
l 197 203
l 15 37
l 3 250
l 15 41
l 244 255
l 1 49
l 15 55
l 1 58
l 343 345
l 1145 349
l 55 68
```

Рисунок 17 – Содержимое файла формата .obj

Импортирование модели в программную среду реализовывалось через алгоритм чтения файла и записи результата в динамические массивы.

Для начала была написана функция `quantity_vert_edg()` для подсчета количества вершин и ребер, необходимая для создания динамических массивов. Данная функция построчно проходит файл, пока не будет достигнут его конец. Сначала проверяется первый элемент строки. Если этим элементом является вершина (v), то считается их количество и записывается в переменную `quantity_vertexes`, а если первый элемент – информация о ребре модели (l), то количество записывается в переменную `quantity_edges`. Алгоритм подсчета количества вершин и ребер представлен на рисунке 18.

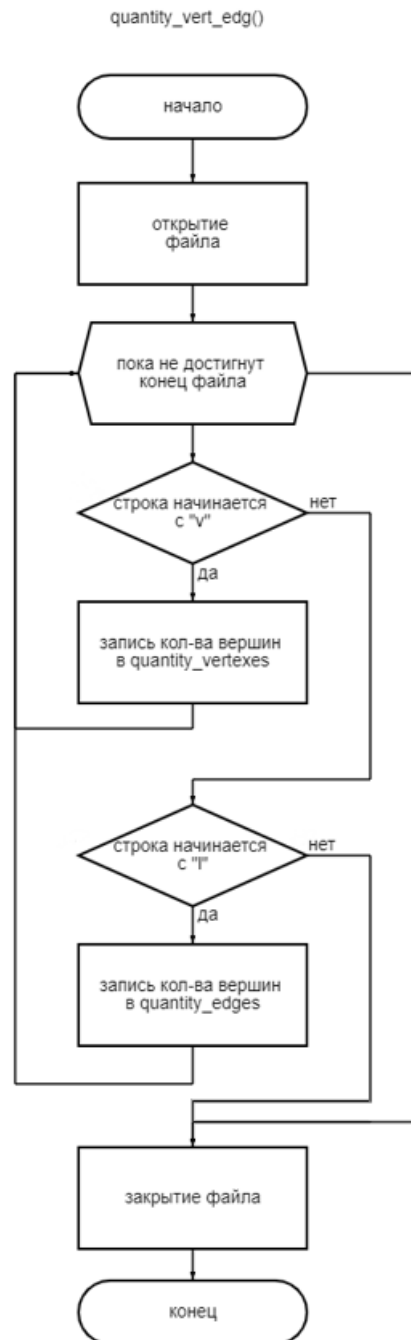


Рисунок 18 – Схема чтения из файла

Далее были реализованы функции `read_vertexes_from_file()` и `read_edges_from_file()`, которые снова же построчно проходят файл, из которого в зависимости от первого символа (`v` или `l`) происходит запись значений в динамические массивы `mass_vertexes` и `mass_edges` (рисунок 19, рисунок 20).

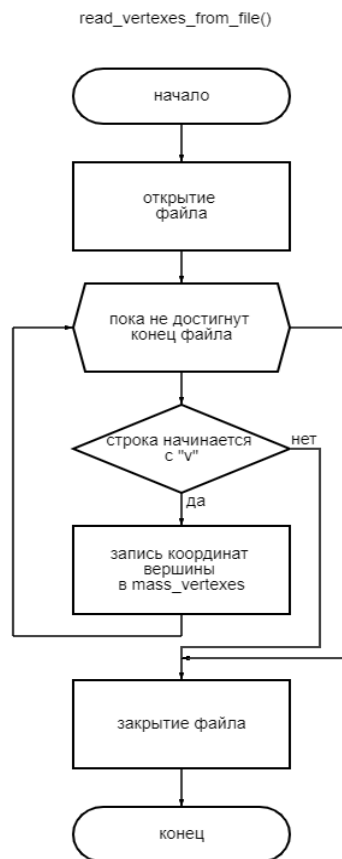


Рисунок 19 – Схема чтения из файла



Рисунок 20 – Схема чтения из файла

1.5 Разработка программы визуализации объекта/сцены

Отрисовка разработанной 3D модели осуществляется с помощью функции drawing, которая использует встроенные функции MoveToEx и LineTo. Данная функция проходит по каждой строке массива mass_edges, хранящего порядок соединения вершин. Затем положение (позиция) пера ставится на вершину со значением первого j-го элемента i-ой строки. Далее с помощью функции LineTo соединяем эту вершину со значением второго j-го элемента i-ой строки. Алгоритм отрисовки модели представлен на рисунке 21.

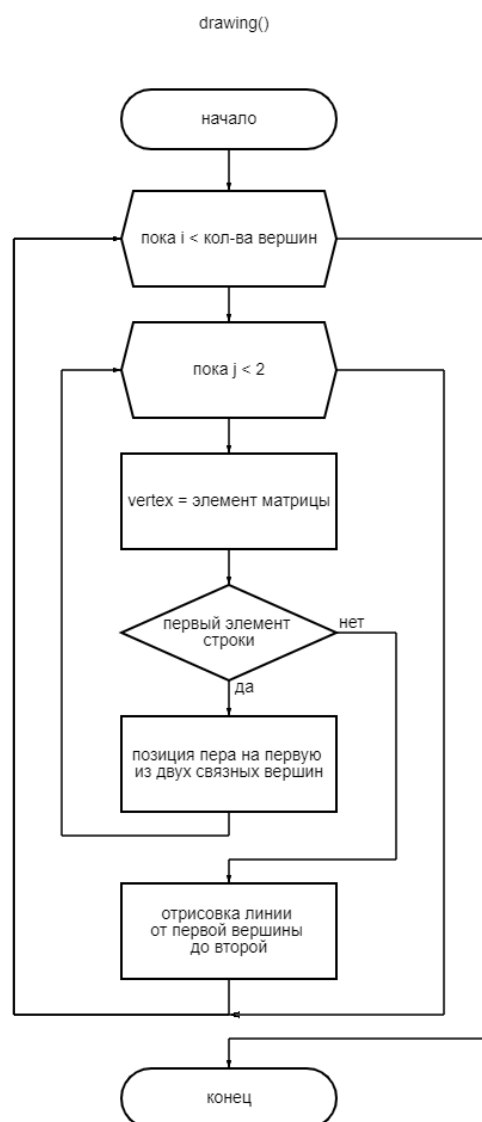


Рисунок 21 – Отрисовка изображения

1.6 Программная реализация динамики объекта

При проецировании объемного объекта на экранную плоскость для каждой точки объекта производится выполнение последовательности преобразований, переводящих точку из трехмерного пространства в двухмерное. Первоначально предполагается, что точка задана в мировых координатах (X_w, Y_w, Z_w) и требуется найти ее отображение на экране (X_s, Y_s).

Процесс визуализации будет проходить в три этапа:

- а) видовое;
- б) перспективное;
- в) экранное.

Для видового преобразования была написана функция `view_transformation()`. В математическом выражении, представляющем собой произведение вектора однородных координат точки объекта на матрицу преобразования, используется величина R (расстояние от точки наблюдения до начала мировой системы координат), полярные углы θ и φ в горизонтальной и вертикальной плоскости соответственно. Функция выполняет перемножение матрицы видовых преобразований (рисунок 22) на координаты вершин объекта и записывает результат в массив `new_vertexes`.

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

Рисунок 22 – Матрица видовых преобразований

Данная матрица получена в результате смены мировой системы координат на пользовательскую с началом координат в точке наблюдения.

Перспективные преобразования учитывают некоторые эффекты, связанные с особенностями зрительной системы человека, и позволяют сделать изображение более реалистичными. При применении ортогональной проекции, в которой проекторами являются параллельные линии, просто отбрасывается координата Z_e , а (X_e, Y_e) представляет точку в двухмерном пространстве. На рисунке 23 представлены выражения для перспективных преобразований.

$$\frac{X}{d} = \frac{x}{z} \quad X = \frac{dx}{z} \quad \frac{Y}{d} = \frac{y}{z} \quad Y = \frac{dy}{z}$$

Рисунок 23 – Формулы для перспективных преобразований

Экранное преобразование осуществляет коррекцию координат (X, Y) в (X_s, Y_s) с учетом таких параметров, как текущее разрешение графического дисплея, сдвига по осям, направление оси координат (рисунок 24).

$$X = \frac{dx}{z} + c_1 \quad Y = \frac{dy}{z} + c_2$$

Рисунок 24 – Формулы для экранных преобразований

Для данного преобразования была составлена функция `screen_transformation()`, выполняющая смещение координаты на определенное значение переменной, чтобы объект отображался ближе к центру окна приложения.

Построение каркасного отображения трехмерного объекта, или так называемой «проволочной» (wireframe) модели, заключается в выполнении описанных выше преобразований для каждой точки – вершины объекта и последующей прорисовки ребер в виде отрезков, соединяющих найденные вершин.

1.7 Реализация интерфейса в отношении управления объектом или камерой в сцене

Управление объекта осуществляется с помощью клавиш клавиатуры.

Клавиши управления для изменения углов поворота:

вверх – поднимает камеру наблюдателя вверх.

вниз – поднимает камеру наблюдателя вниз.

вправо – вращение объекта вокруг оси Z против часовой стрелки.

влево – вращение объекта вокруг оси Z по часовой стрелке.

Клавиши F1 и F2 – уменьшение и увеличение коэффициента масштабирования соответственно.

Клавиши F3 и F4 – уменьшение и увеличение расстояния от точки наблюдения до начала мировых координат соответственно.

Клавиши F5 и F6 – уменьшение и увеличение расстояния от наблюдателя до экранной плоскости соответственно.

1.8 Отладка и тестирование программы

В таблице 1 представлен процесс тестирования программы.

Таблица 1 – Описание поведения программы при тестировании

Выполнение действия	Ожидаемый результат	Полученный результат
загрузка существующего файла, отрисовка модели	импорт файла	выполнено
нажатие клавиш изменения углов поворота	поворот проекции	выполнено
нажатие клавиш изменения коэффициента масштабирования	изменение объекта	выполнено
нажатие клавиш изменения расстояния от точки наблюдения до начала координат	изменение расстояния от точки наблюдения до начала координат	выполнено
нажатия клавиш изменения расстояния до экранной плоскости	изменение расстояния экранной плоскости	выполнено

В результате тестирования было выявлено, что программа успешно проверяет на соответствие необходимым требованиям. Ниже представлены результаты работы:



Рисунок 25 – Загрузка файла, отрисовка модели

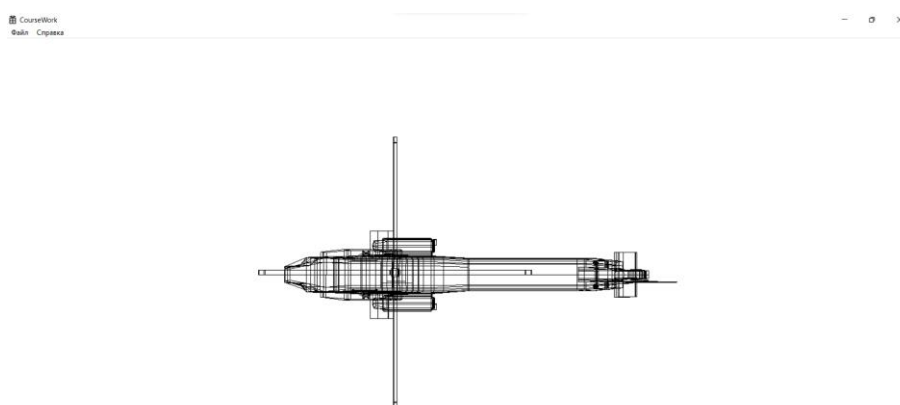


Рисунок 26 – Вид сверху

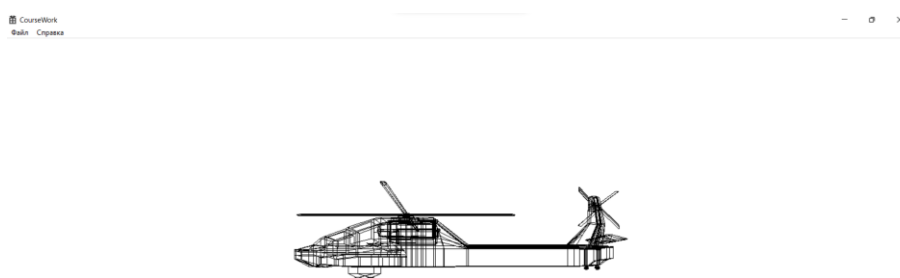


Рисунок 27 – Вид сбоку



Рисунок 28 – Вид спереди

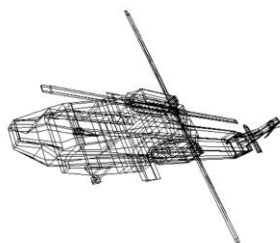


Рисунок 29 – Произвольный угол

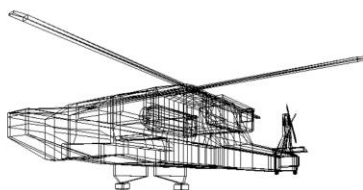


Рисунок 30 – Увеличение коэффициента масштабирования



Рисунок 31 – Уменьшение коэффициента масштабирования

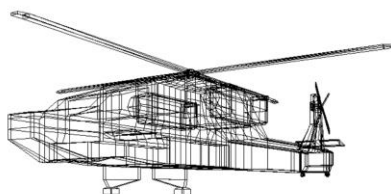


Рисунок 32 – Увеличение расстояния



Рисунок 33 – Уменьшение расстояния



Рисунок 34 – Увеличение расстояния до экранной плоскости

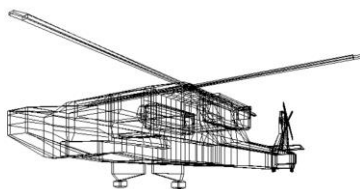


Рисунок 35 – Уменьшение расстояния до экранной плоскости

2 Графическая часть

2.1. Чертежи трех проекций

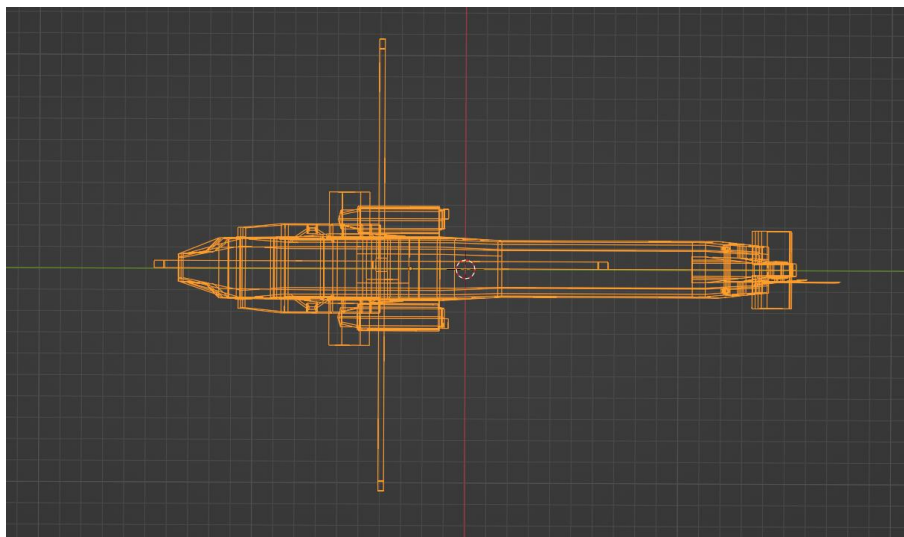


Рисунок 36 – Проекция на ось Z

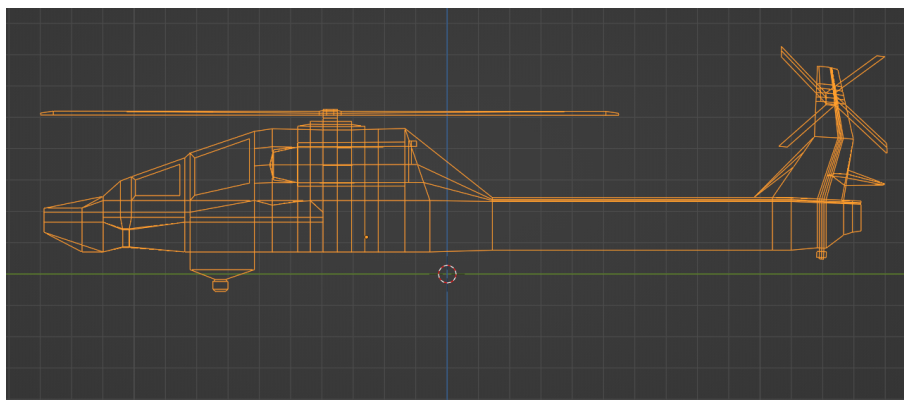


Рисунок 37 – Проекция на ось X

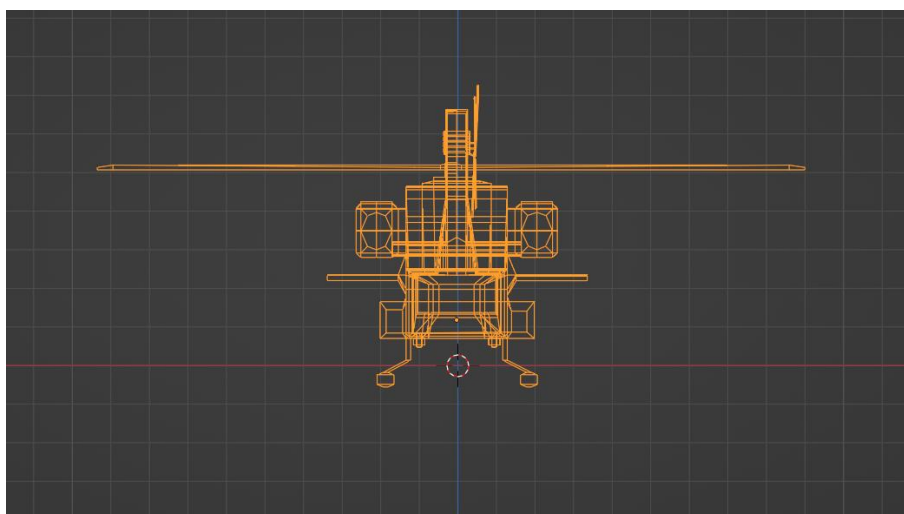


Рисунок 38 – Проекция на ось Y

2.2. Структурная схема алгоритма

Все схемы алгоритма для разработки программы построения и визуализации 3D объекта приведены в расчетной части. На рисунке 40 представлена схема последовательного вызова функций.

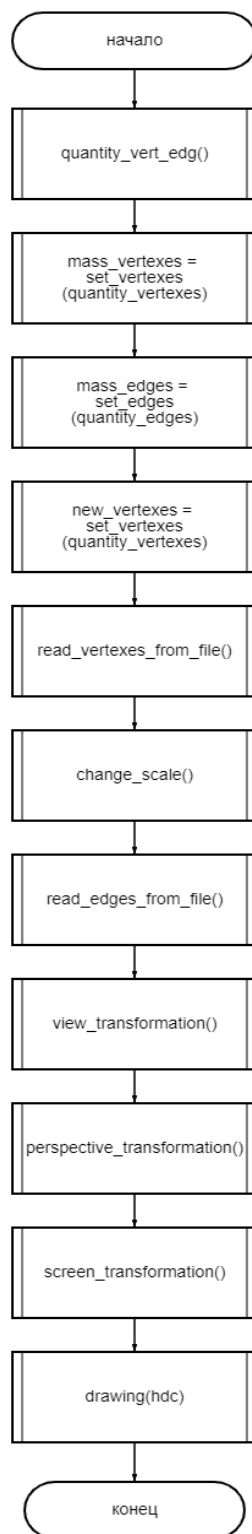


Рисунок 39 – Схема вызова функций

3 Экспериментальная часть

3.1. Тестирование и верификация программного обеспечения

Среда разработки Microsoft Visual Studio 2019 предоставляет все средства, необходимые при разработке и отладки программы.

Тестирование проводилось в рабочем порядке, а именно в процессе разработки программы. В ходе тестирования было выявлены требуемые исключения, связанные с алгоритмом программы. Результаты тестирования были приведены в пункте 1.8.

Заключение

В результате выполнения курсового проектирования была разработана 3D-модель вертолета в среде Blender, а также программа, реализующая построение каркасной модели объекта в Microsoft Visual Studio на языке C++.

Созданная программа позволяет выполнять изменение модели, расстояния от точки наблюдения до начала координат, расстояния до экранной плоскости.

В данной документации представлены этапы моделирования объекта в программной среде, алгоритм решения поставленной задачи, пояснение алгоритма с предоставленными блок-схемами, описание и результаты тестирования программы, выводы о проделанной работе.

В процессе выполнения данной курсовой работы был получен практический опыт в 3D-моделировании.

Список используемых источников

1. Технология программирования на C++. Win32 API-приложения. – СПб.: БХВ-Петербург, 2010. – 288 с.: ил. – (Учебное пособие).
2. Методы и алгоритмы компьютерной графики в примерах на Visual C++. – СПб.: БХВ-Петербург, 2002. – 416 с.: ил.
3. Сучкова, Л. И. Win32 API: основы программирования: учебное пособие/ Л. И. Сучкова; АлтГТУ им. И. И. Ползунова. – Барнаул, АлтГТУ, 2010. – 138 с., ил.
4. Компьютерная графика: учеб. /М. А. Кудрина, К. Е. Климентьев. – Самара: Изд-во Самар. гос. аэрокосм. ун-та, 2013. – 138 с.
5. Безруков В. А. Win32 API. Программирование/ учебное пособие. – СПб: СПбГУ ИТМО, 2009. – 90

ПРИЛОЖЕНИЕ А.

Листинг программы

```
#include "framework.h"
#include "CourseWork.h"
#include "math.h"
#include "stdio.h"
#include "conio.h"
#include "stdlib.h"
#include "windows.h"

#define _CRT_SECURE_NO_WARNINGS

#define M_PI 3.14159265358979323846

#define MAX_LOADSTRING 100

// 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);

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, IDC_COURSEWORK, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);

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

    HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_COURSEWORK));

    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;
}
```

```

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(IDI_COURSEWORK));
    wcex.hCursor        = LoadCursor(nullptr, IDC_ARROW);
    wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
    wcex.lpszMenuName    = MAKEINTRESOURCEW(IDC_COURSEWORK);
    wcex.lpszClassName  = szWindowClass;
    wcex.hIconSm        = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));

    return RegisterClassExW(&wcex);
}

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, SW_SHOWMAXIMIZED); // показываем окно на весь экран
    UpdateWindow(hWnd);

    return TRUE;
}

const char name_file[] = "model3D.obj";
char str[20];
FILE* file;

int quantity_vertexes = 0;
int quantity_edges = 0;

float** mass_vertexes;
float** mass_edges;

float** new_vertexes;

float angleA = 330.0;
float angleB = 180.0;

float R = 1024.0;
float d = 512.0;

float screen_width = 750;
float screen_height = 400;

int ratio_scale = 4;

void quantity_vert_edg()
{
    file = fopen(name_file, "r");
    while ((fscanf(file, "%s", str) != EOF))
    {
        if (!file)

```

```

        {
            break;
        }
        if (str[0] == 'v')
        {
            quantity_vertexes++;
        }
        if (str[0] == 'l')
        {
            quantity_edges++;
        }
    }
    fclose(file);
};

float** set_vertexes(int size)
{
    float** arr_vertexes;
    arr_vertexes = (float**)malloc(size * sizeof(float*));
    for (int i = 0; i < size; i++)
    {
        arr_vertexes[i] = (float*)malloc(4 * sizeof(float));
        for (int j = 0; j < 4; j++)
        {
            arr_vertexes[i][j] = 1;
        }
    }
    return arr_vertexes;
};

float** set_edges(int size)
{
    float** arr_edges;
    arr_edges = (float**)malloc(size * sizeof(float*));
    for (int i = 0; i < size; i++)
    {
        arr_edges[i] = (float*)malloc(4 * sizeof(float));
    }
    return arr_edges;
};

void read_vertexes_from_file()
{
    int i = 0;
    file = fopen(name_file, "r");
    while ((fscanf(file, "%s", str) != EOF))
    {
        if (!file)
        {
            break;
        }
        if (str[0] == 'v')
        {
            for (i; i < quantity_vertexes; i++)
            {
                for (int j = 0; j < 3; j++)
                {
                    fscanf(file, "%f", &mass_vertexes[i][j]);
                }
                break;
            }
            i++;
        }
    }
    fclose(file);
}

```



```

void read_edges_from_file()
{
    int i = 0;
    file = fopen(name_file, "r");
    while ((fscanf(file, "%s", str) != EOF))
    {
        if (!file)
        {
            break;
        }
        if (str[0] == 'l')
        {
            for (i; i < quantity_edges; i++)
            {
                for (int j = 0; j < 2; j++)
                {
                    fscanf(file, "%f", &mass_edges[i][j]);
                }
                break;
            }
            i++;
        }
    }
    fclose(file);
}

void view_transformation() {
    float fi = angleA * M_PI / 180;
    float TETA = angleB * M_PI / 180;

    float matrix_vid[4][4]
    {
        {-sin(TETA) , -cos(fi) * cos(TETA) , -sin(fi) * cos(TETA) , 0},
        { cos(TETA) , -cos(fi) * sin(TETA) , -sin(fi) * sin(TETA) , 0},
        {      0 , sin(fi) , -cos(fi) , 0},
        {      0 ,      0 ,      R , 1}
    };

    for (int i = 0; i < quantity_vertexes; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            new_vertexes[i][j] = 0;
            for (int k = 0; k < 4; k++)
            {
                new_vertexes[i][j] = new_vertexes[i][j] + mass_vertexes[i][k] *
matrix_vid[k][j];
            }
            new_vertexes[i][3] = mass_vertexes[i][3];
        }
    }
}

void perspective_transformation()
{
    for (int i = 0; i < quantity_vertexes; i++)
    {
        new_vertexes[i][0] = (d * new_vertexes[i][0] / new_vertexes[i][2]);
        new_vertexes[i][1] = (d * new_vertexes[i][1] / new_vertexes[i][2]);
    }
}

void screen_transformation()
{
    for (int i = 0; i < quantity_vertexes; i++)
    {
        new_vertexes[i][0] = new_vertexes[i][0] + screen_height;
        new_vertexes[i][1] = new_vertexes[i][1] + screen_width;
    }
}

```

```

    }
}

void change_scale()
{
    for (int i = 0; i < quantity_vertexes; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            mass_vertexes[i][j] = mass_vertexes[i][j] * ratio_scale;
        }
    }
}

void drawing(HDC hdc)
{
    int vertex;
    for (int i = 0; i < quantity_edges; i++)
    {
        for (int j = 0; j < 2; j++)
        {
            vertex = mass_edges[i][j];
            if (vertex > 0)
            {
                if (j == 0)
                {
                    MoveToEx(hdc,    new_vertexes[vertex][1],    new_vertexes[vertex][0],
NULL);

                }
                else
                {
                    LineTo(hdc, new_vertexes[vertex][1], new_vertexes[vertex][0]);
                }
            }
        }
    }
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_KEYDOWN:
        {
            switch (LOWORD(wParam))
            {
                case VK_RIGHT:
                {
                    angleA = angleA + 10.0;
                    InvalidateRect(hWnd, NULL, TRUE);
                    break;
                }
                case VK_LEFT:
                {
                    angleA = angleA - 10.0;
                    InvalidateRect(hWnd, NULL, TRUE);
                    break;
                }
                case VK_UP:
                {
                    angleB = angleB + 10.0;
                    InvalidateRect(hWnd, NULL, TRUE);
                    break;
                }
                case VK_DOWN:
                {
                    angleB = angleB - 10.0;

```

```

        InvalidateRect(hWnd, NULL, TRUE);
        break;
    }

    case VK_F1:
    {
        ratio_scale = ratio_scale + 1.0;
        InvalidateRect(hWnd, NULL, TRUE);
        break;
    }
    case VK_F2:
    {
        ratio_scale = ratio_scale - 1.0;
        InvalidateRect(hWnd, NULL, TRUE);
        break;
    }

    case VK_F3:
    {
        d = d + 100.0;
        InvalidateRect(hWnd, NULL, TRUE);
        break;
    }
    case VK_F4:
    {
        d = d - 100.0;
        InvalidateRect(hWnd, NULL, TRUE);
        break;
    }

    case VK_F5:
    {
        R = R + 50.0;
        InvalidateRect(hWnd, NULL, TRUE);
        break;
    }
    case VK_F6:
    {
        R = R - 50.0;
        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...
    }

```

```

        quantity_vert_edg();

        mass_vertexes = set_vertexes(quantity_vertexes);
        mass_edges = set_edges(quantity_edges);

        new_vertexes = set_vertexes(quantity_vertexes);
        read_vertexes_from_file();
        change_scale();
        read_edges_from_file();
        view_transformation();
        perspective_transformation();
        screen_transformation();
        drawing(hdc);

        EndPaint(hWnd, &ps);

        free(mass_vertexes);
        free(mass_edges);
        free(new_vertexes);
    }
    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;
}

```