Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Системы автоматизации проектирования»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовая работа

по курсу «Компьютерная графика»

на тему:

«Разработка программы построения и визуализации 3D объекта»

**Выполнили:**

Трубаненко А.Г.

Ланцов А.С.

**Принял:**

Финогеев А. А.

Пенза 2021

Оглавление

[Введение 3](#_Toc74128864)

[Постановка задачи 4](#_Toc74128865)

[1 Расчетная часть 5](#_Toc74128866)

[2 Декомпозиция на графические примитивы и аппроксимация гранями 10](#_Toc74128867)

[3 Описание расположения объекта/сцены в системе координат 12](#_Toc74128868)

[4 Импортирование созданных моделей и ландшафта в программную среду 12](#_Toc74128869)

[5 Разработка программы визуализации объекта 12](#_Toc74128870)

[6 Реализация интерфейса в отношении управления камерой 12](#_Toc74128871)

[7 Отладка и тестирование программы 12](#_Toc74128872)

[Заключение 20](#_Toc74128873)

[Список литературы 21](#_Toc74128874)

[Приложение А 22](#_Toc74128875)

[Приложение В 23](#_Toc74128876)

# **Введение**

**Трехмерная графика**или **3D-моделирование** – компьютерная графика, сочетающая в себе приемы и инструменты, необходимые для создания объемных объектов в трехмерном пространстве.

Под приемами стоит понимать способы формирования трехмерного графического объекта – расчет его параметров, черчение «скелета» или объемной не детализированной формы; выдавливание, наращивание и вырезание деталей и т.д.

А под инструментами - профессиональные программы для 3D-моделирования. В первую очередь – SolidWork, Blender, 3DMAX, а также некоторые другие программы для объемной визуализации предметов и пространства.

Области применения 3D- моделирования:

* **Реклама и маркетинг**
* **Городское планирование**
* **Промышленность**
* **Компьютерные игры**
* **Кинематограф**
* **Архитектура и дизайн интерьеров**
* **Анимация**

# **Постановка задачи**

Требуется определить модель трехмерного объекта и синтезировать ее в среде 3D-моделирования. Далее, разработать программу, реализующую построение и визуализацию готового объекта.

Устройства управления – клавиатура

В качестве среды моделирования был выбран программный продукт Blender.

В качестве языка разработки был выбран С++.

В качестве среды программирования был выбран продукт JetBrains CLion.

# **1 Расчетная часть**

Анализ и описание процесса синтеза 3D объекта методом твердотельного моделирования

Шаг 1. Добавили цилиндр.

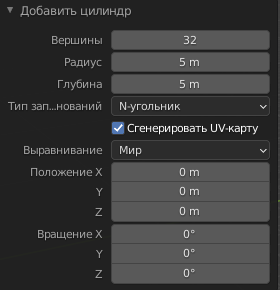


Рисунок 1 Параметры цилиндра

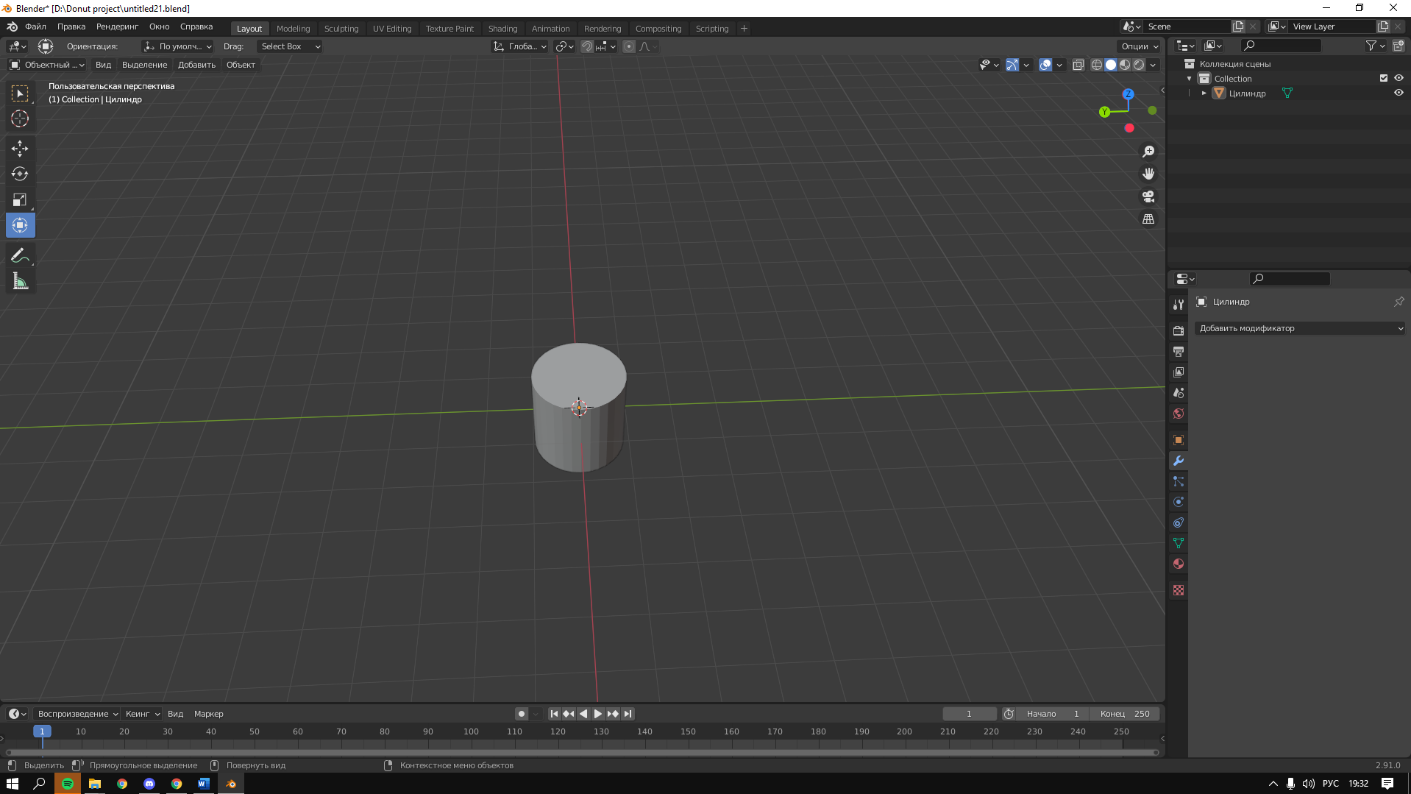


Рисунок 2 Добавление цилиндра

Шаг 2. Произвели некоторые операции над фигурой. Сузили низ цилиндра, чтобы образовалась форма вафельного стаканчика.

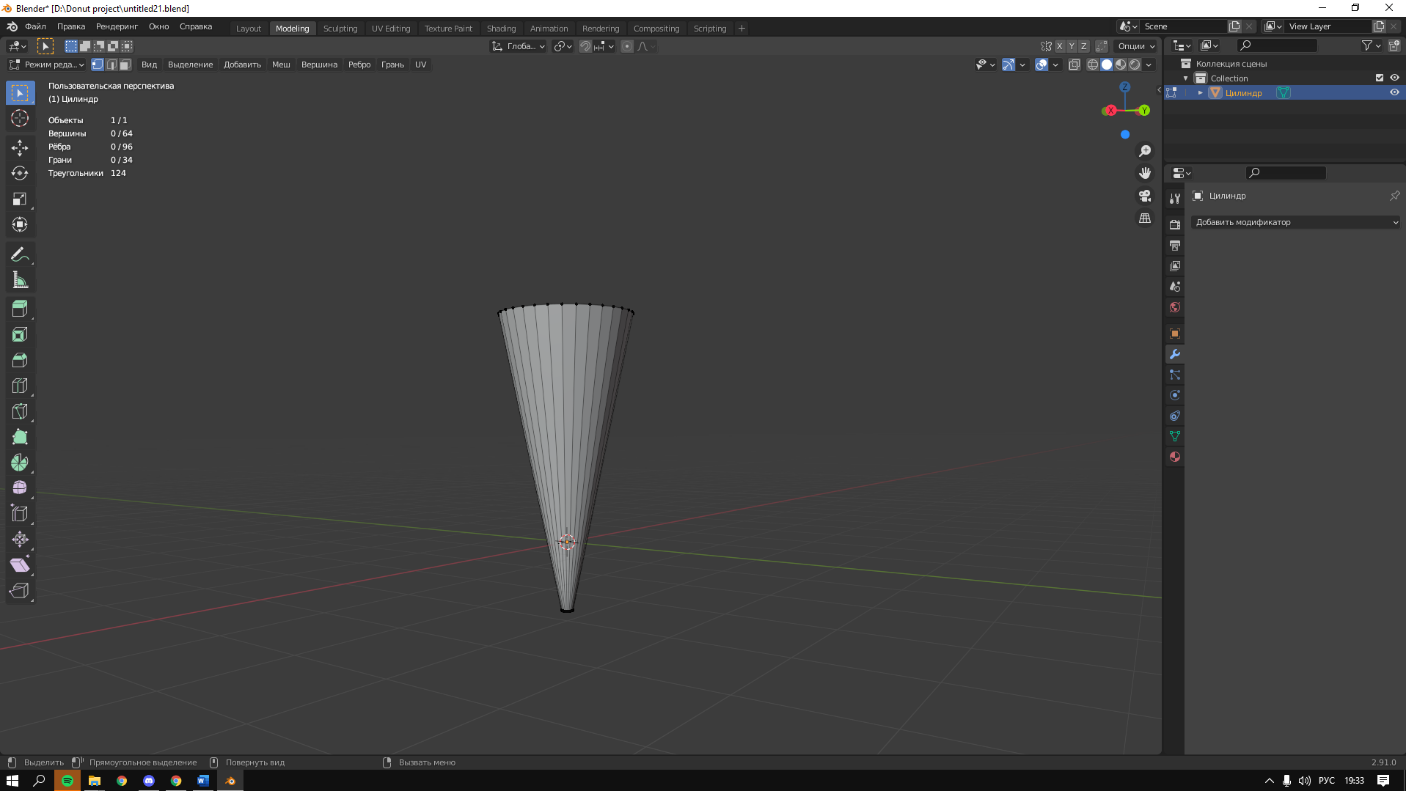


Рисунок 3 Преобразования над цилиндром

Шаг 3. Разрезали цилиндр по контуру.

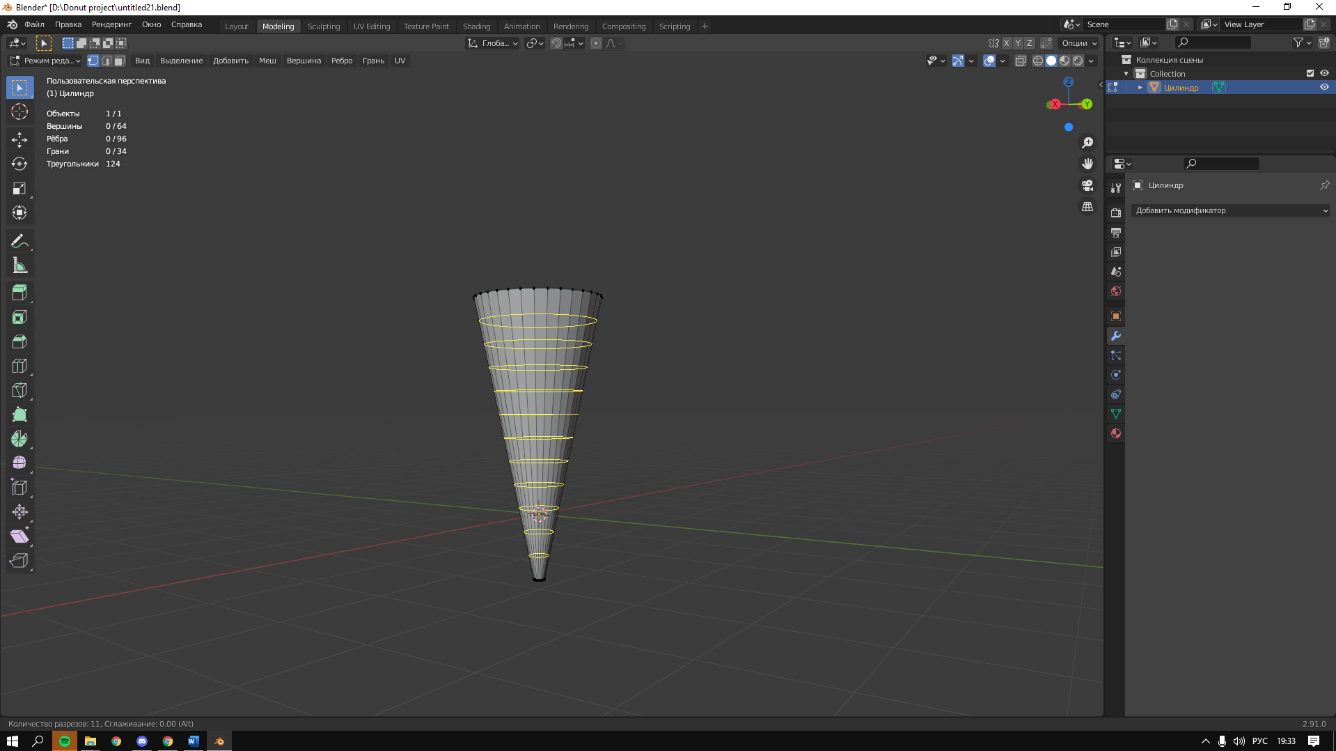


Рисунок 4 Разрезание цилиндра по контуру

Шаг 4. Продублировали рожок и не много увеличили в размере относительно первого.

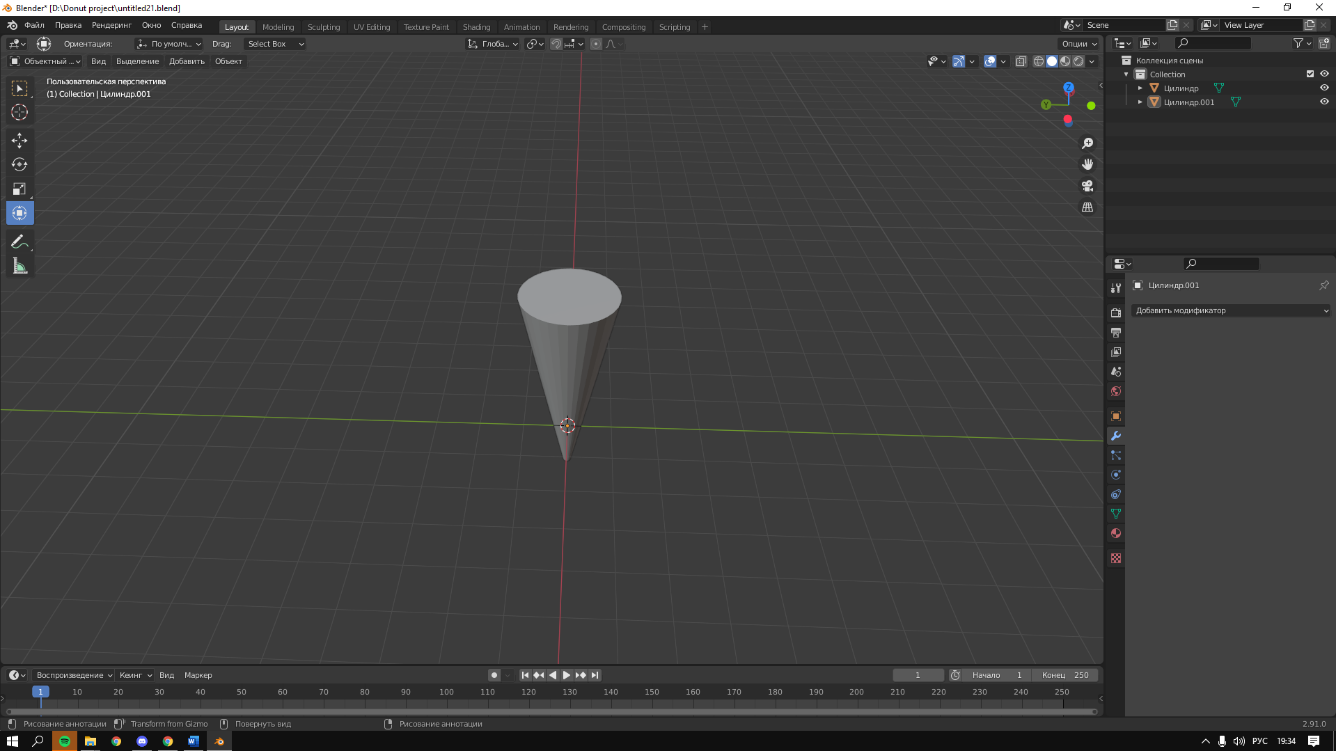


Рисунок 5 Дублирование и преобразование рожка

Шаг 5. Добавили на Цилиндр001 “модификатор сетка” и “простая деформация”.

Во-втором модификаторе изменили угол кручения на 120 градусов по оси Z.

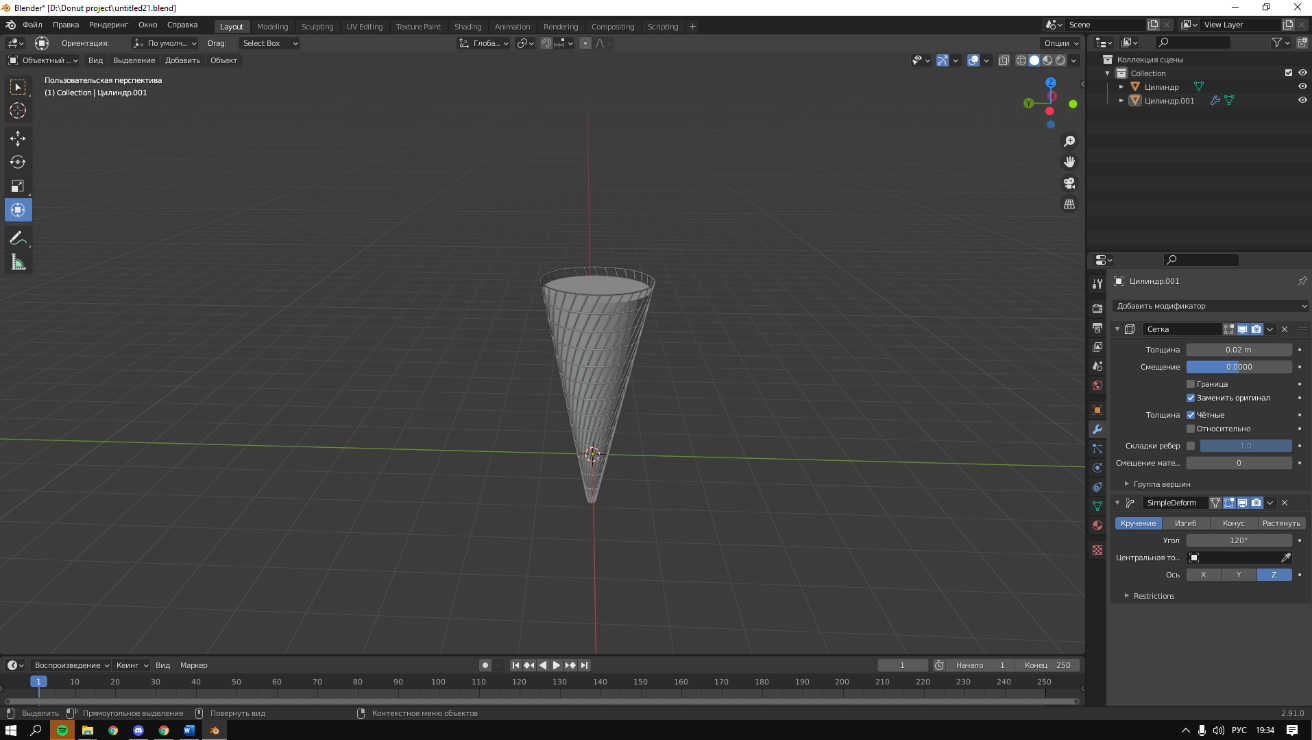


Рисунок 6 Добавление модификаторов на Цилиндр001

Шаг 6. Добавили плоскость для создания основы мороженного.

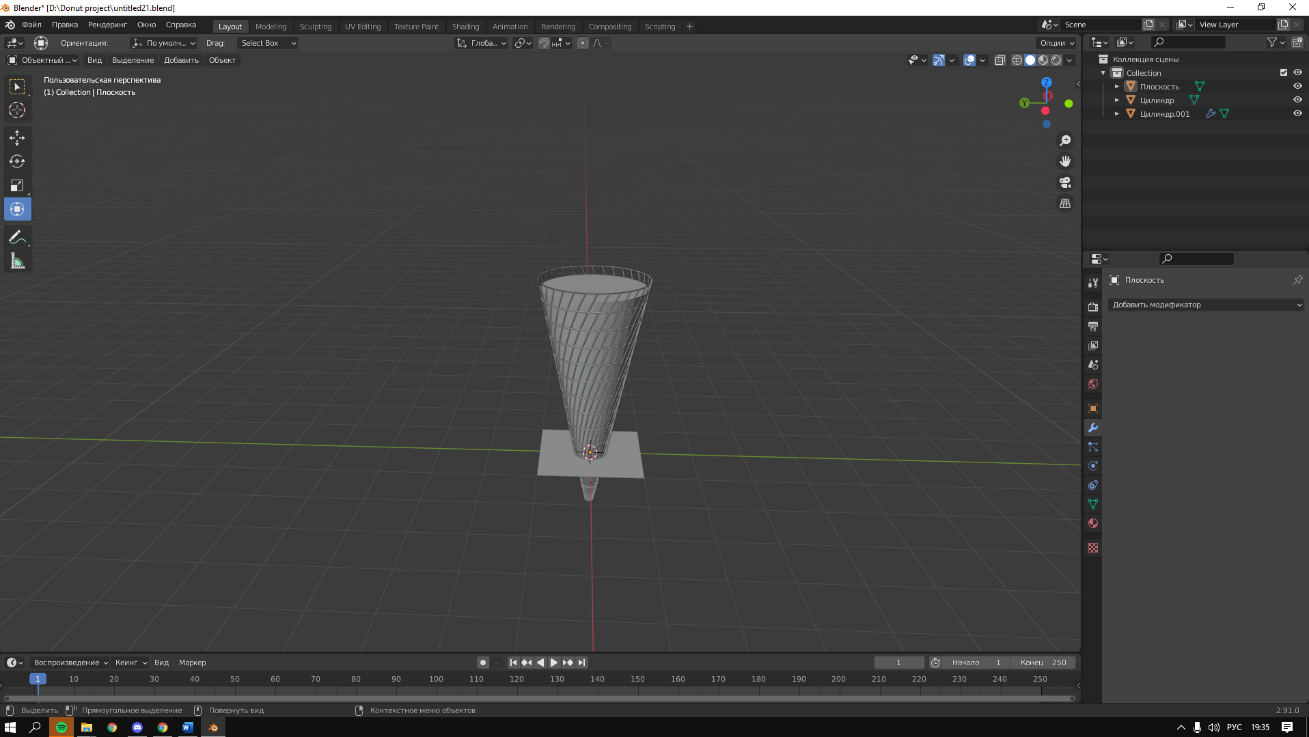


Рисунок 7 Добавление плоскости

Шаг 7. Переместили плоскость на верх рожка.

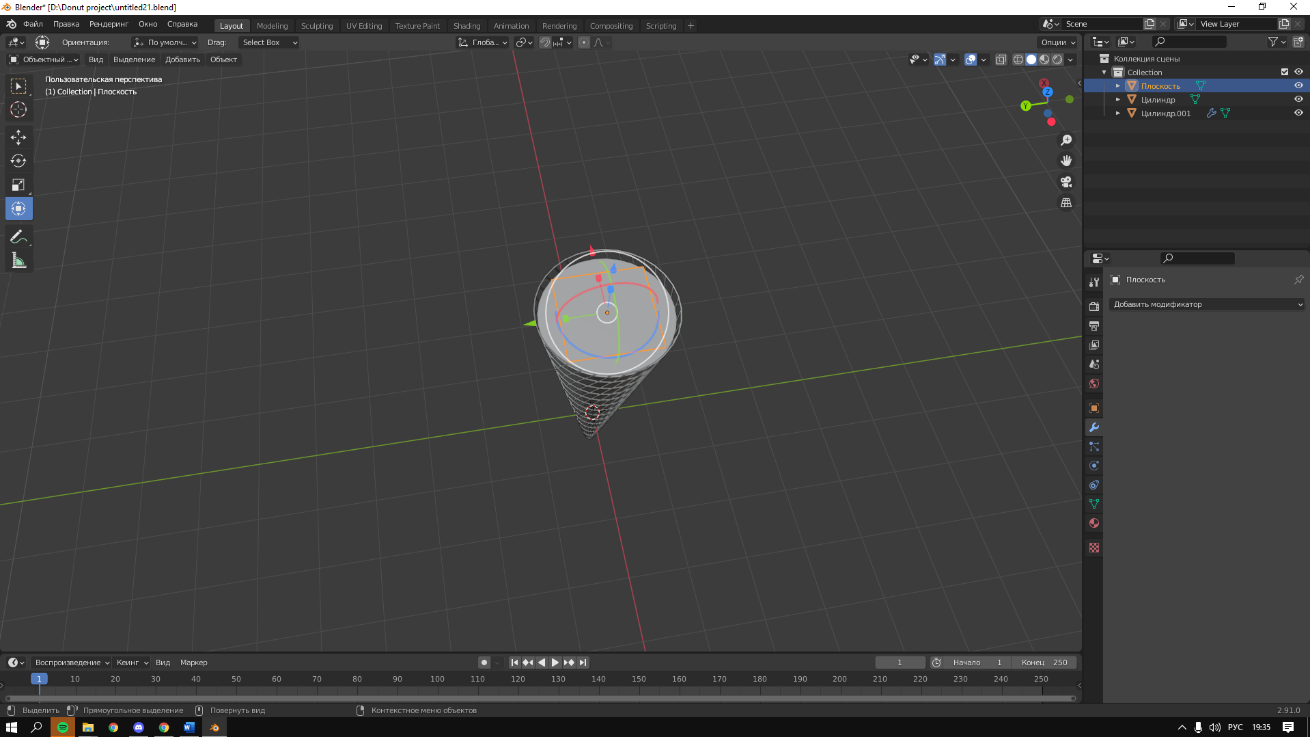


Рисунок 8 Перемещение плоскости

Шаг 8. На плоскость наложили модификаторы “Простая деформация” “Подразделение поверхности ” “Винт”.

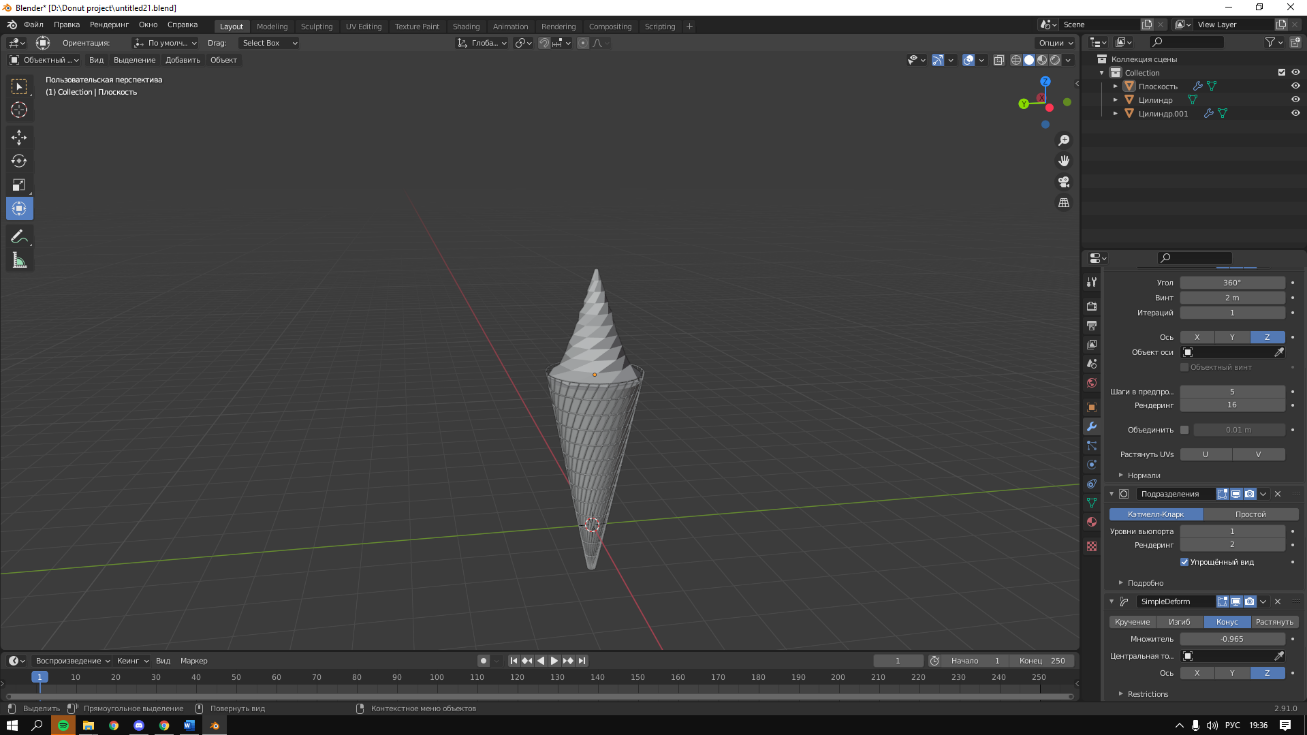


Рисунок 9 Добавление модификаторов на плоскость

Шаг 9. Итог.

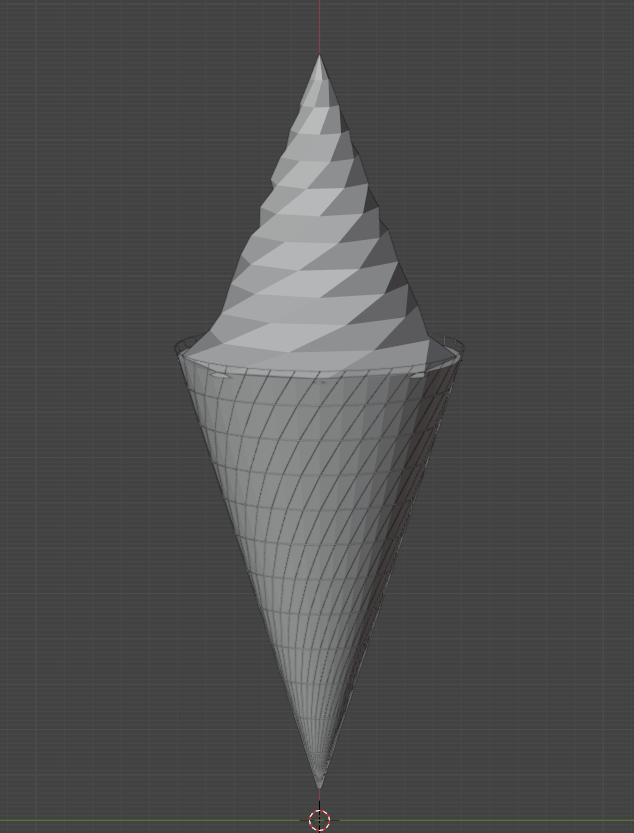


Рисунок 10 Итог

# **2 Декомпозиция на графические примитивы и аппроксимация гранями**

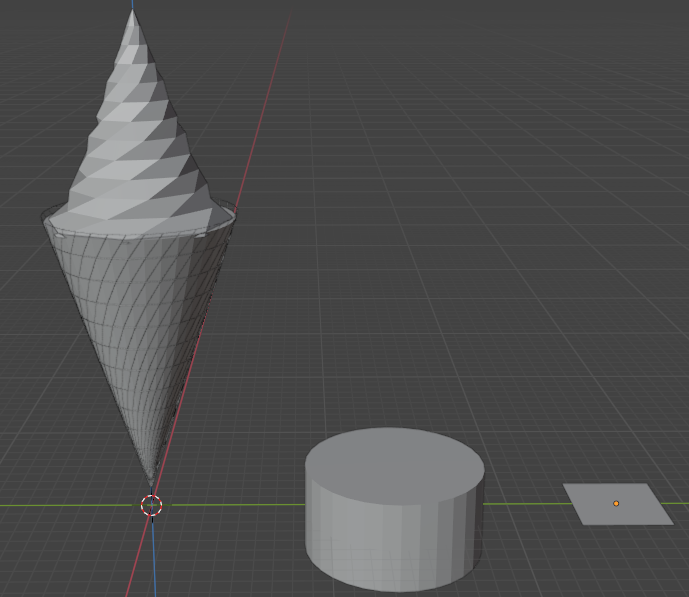


Рисунок 11 Декомпозиция

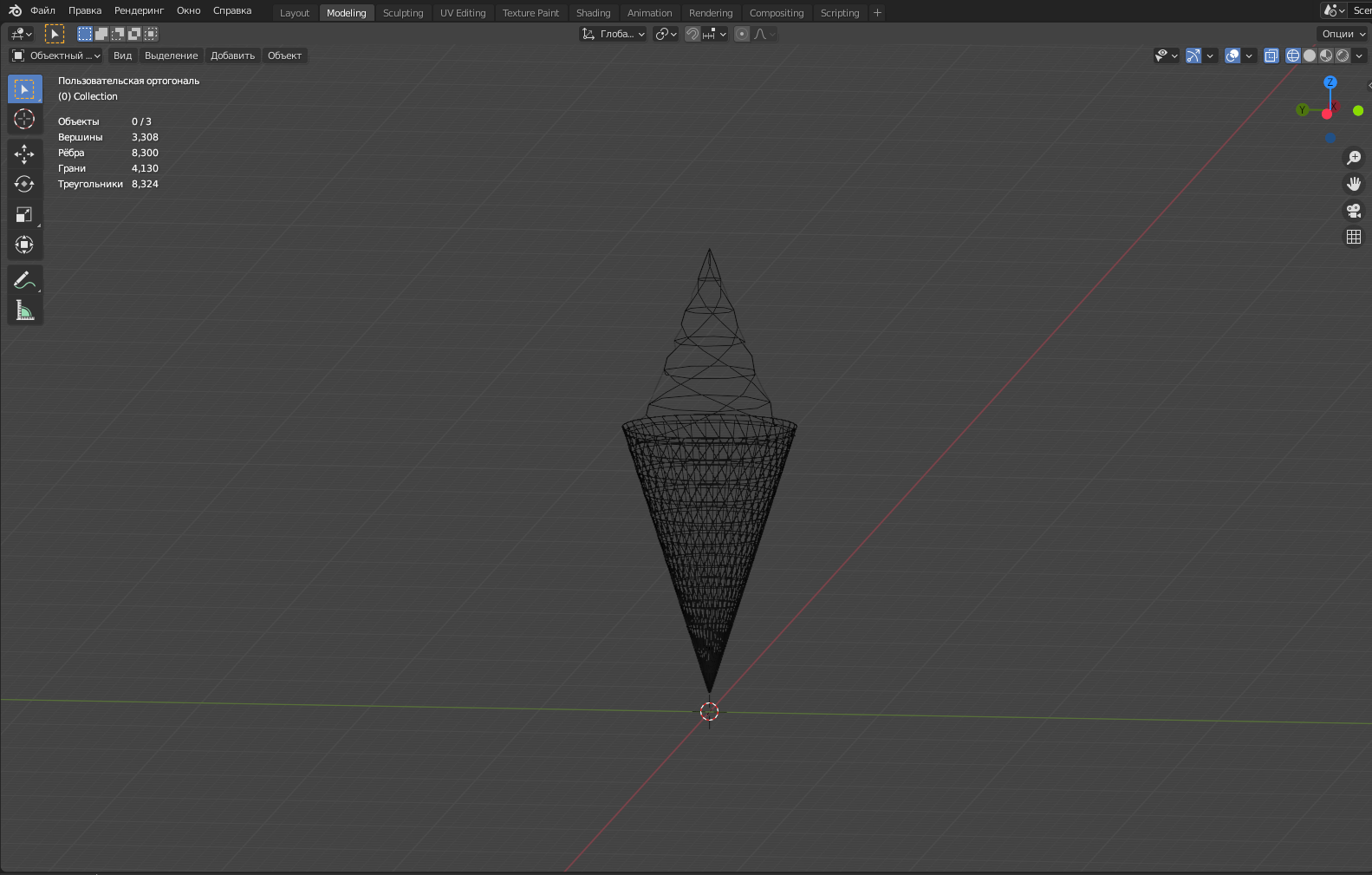


Рисунок 12 Аппроксимация гранями

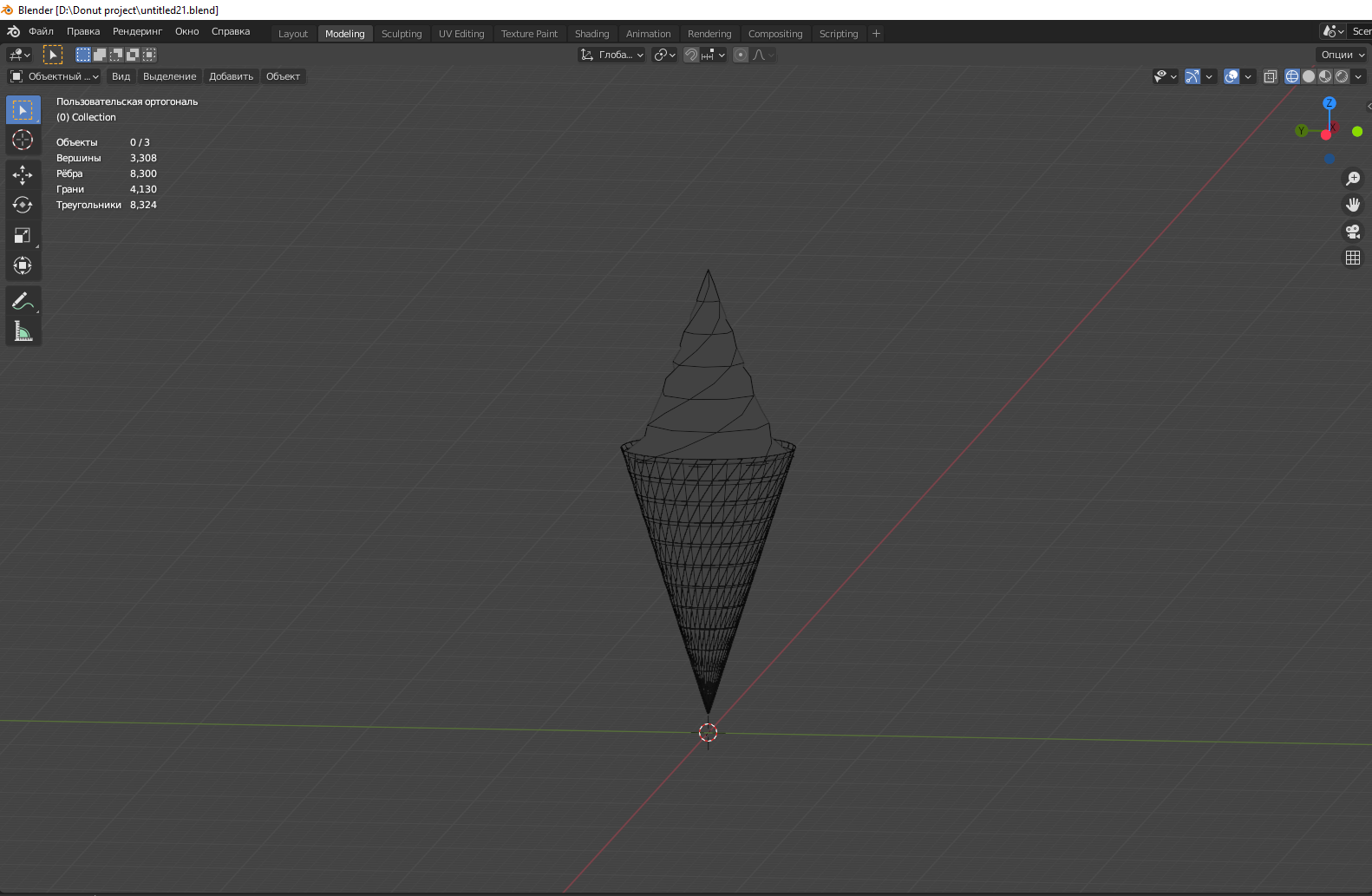


Рисунок 13 Аппроксимация гранями

# **3 Описание расположения объекта/сцены в системе координат**

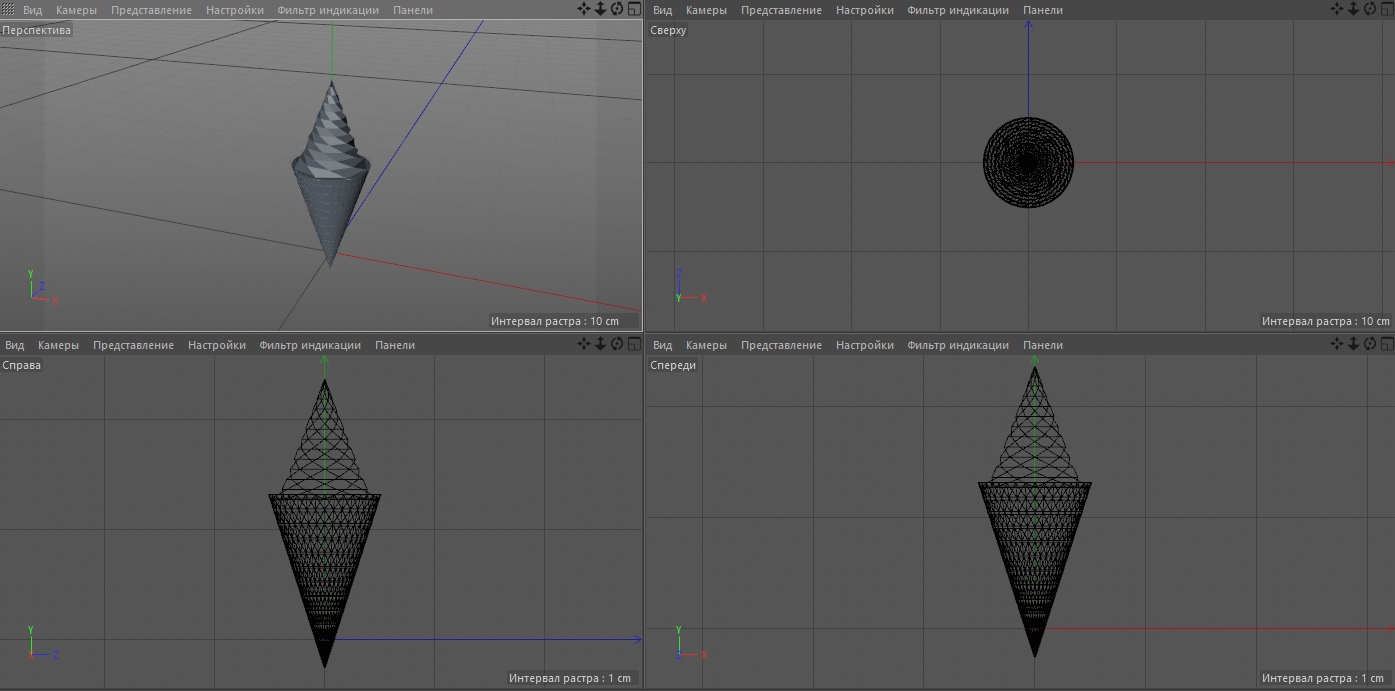


Рисунок 14 Расположения объекта

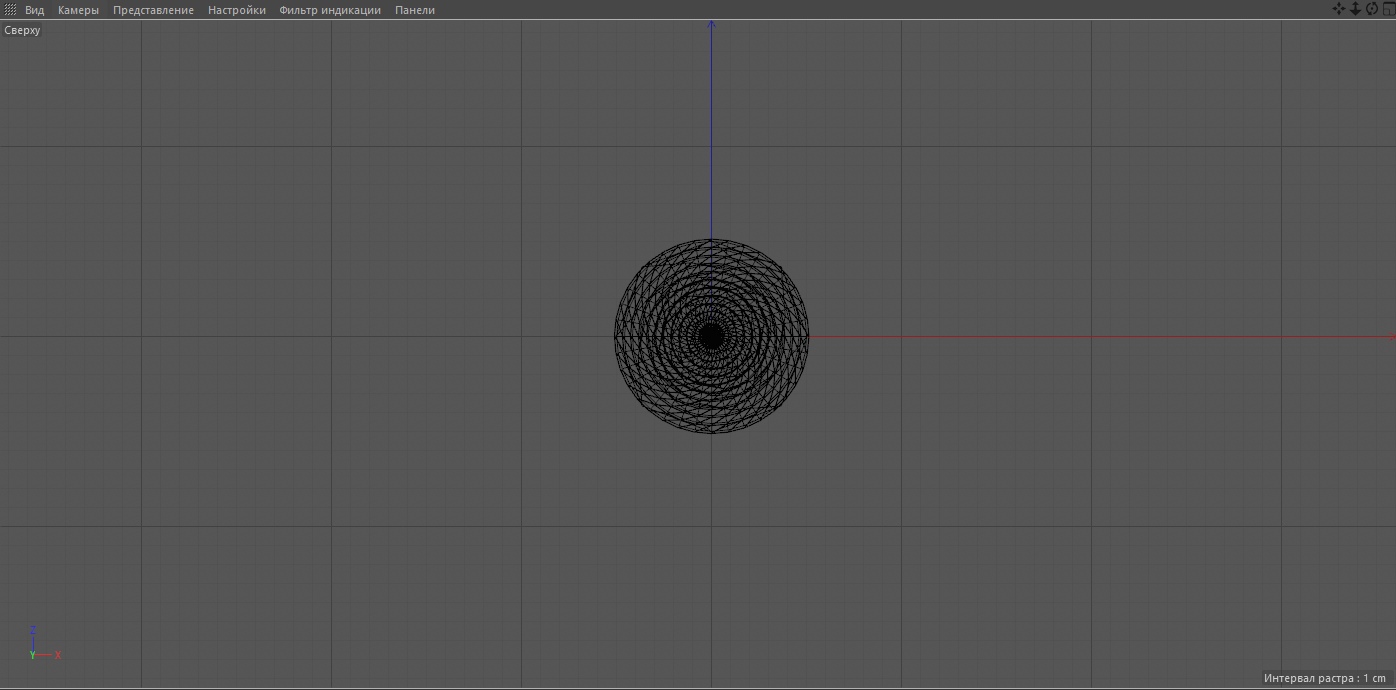


Рисунок 15 Расположение сверху

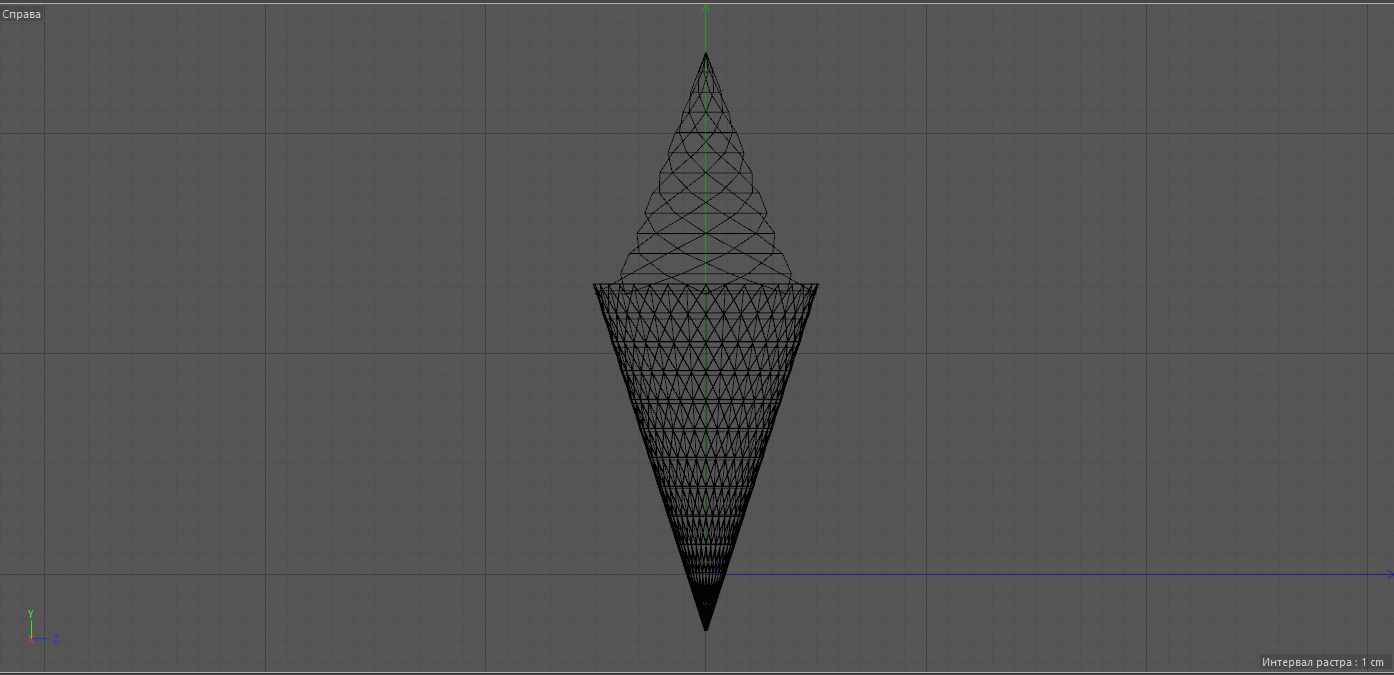


Рисунок 16 Расположение сбоку

# **4 Импортирование созданных моделей и ландшафта в программную среду**

Экспортировать файл модели будем в формате obj. Данный формат содержит информацию о вершинах модели и гранях.

Формат obj файла:

#вершины

v 0.123 0.234 0.345

v -0.123 0.234 0.345

v 0.123 -0.234 0.345

#грани (индексы вершин)

f 1 2 3

f 3 4 5

f 6 3 7

f 7 8 9

Алгоритм чтения из файла:

Токен - это строка с присвоенным и, таким образом, идентифицированным значением. В нашем случае токенами будут “v”, “t”, “#”, числовые значения, а также некоторые другие лексемы obj формата, неиспользуемые нашей программой, но которые могут быть сгенерированы программой 3D моделирования. Их мы будем игнорировать.

Объявление токена:

enum TokenKind {  
 *TOKEN\_FLOAT* = 128,*//0-128 reserved for symbols  
 TOKEN\_INT*,  
 *TOKEN\_NAME*,  
 *TOKEN\_COMMENT*,  
 *TOKEN\_VERTEX*,  
 *TOKEN\_FACE*,  
 *TOKEN\_OBJECT\_NAME*,  
 *// ...*};

struct Token {  
 TokenKind kind;  
 const char \*start;  
 const char \*end;  
 union {  
 double float\_val;  
 uint32\_t int\_val;  
 const char\* name;  
 };  
};

Функция *next\_token()* обрабатывает информацию о следующем токене. Так же в ней, считываются числовые значения координат и индексов вершин.

Загрузка файла в память:

const char\* stream\_start;  
const char\* stream;

void init\_stream(const char\* str) {  
 stream\_start = stream = str;  
 char\_to\_digit['0'] = 0;  
 char\_to\_digit['1'] = 1;  
 char\_to\_digit['2'] = 2;  
 char\_to\_digit['3'] = 3;  
 char\_to\_digit['4'] = 4;  
 char\_to\_digit['5'] = 5;  
 char\_to\_digit['6'] = 6;  
 char\_to\_digit['7'] = 7;  
 char\_to\_digit['8'] = 8;  
 char\_to\_digit['9'] = 9;  
  
 next\_token();  
}  
  
void init\_stream\_from\_file(FILE\* from){  
 size\_t grow\_size = 1024;  
 char\* buffer = (char\*)malloc(grow\_size);  
 int len = 0,grow\_len = 0;  
  
 int end = 0;  
 for(;end!=**EOF**;){  
 end = fscanf(from,"%c",buffer+len);  
 len++;  
 grow\_len++;  
 if (grow\_len == grow\_size){  
 buffer = (char\*)realloc(buffer,len+grow\_size);  
 grow\_len = 1;  
 }  
 }  
 buffer[len-1] = '\0';  
 init\_stream(buffer);  
}

Функция *parse\_vertex*, будет вызываться каждый раз, когда находится токен вершины. Функция последовательно считывает координаты вершин и записывает из в вектор.

Функция *parse\_face*, будет вызываться каждый раз, когда находится токен грани. Функция последовательно считывает индексы вершин и записывает новую грань в вектор.

В функции *parse\_object* реализован цикл обработки файла. Функция возвращает объект типа *Object*. Выделение памяти под данные модели, происходит из приватной кучи для объектов. Это структурирует данные, и увеличивает локальность кэша.

# **5 Разработка программы визуализации объекта**

Для визуализации объекта были объявлены следующие классы:

struct POINT3{  
 double x;  
 double y;  
 double z;  
};

struct Face{  
 uint32\_t\* order;  
 uint32\_t vertices\_count;  
};  
  
class Object{  
private:  
 POINT3\* world\_points;  
 POINT\* screen\_points;  
 Face\* faces;  
 uint32\_t edges\_count;  
 uint32\_t vertices\_count;

void draw(HDC hdc,COLORREF color);  
};

Для динамики камеры были объявлены следующие классы:

struct SphericalCoords{  
 Radius r;  
 Angle phi;  
 Angle theta;  
  
 POINT3 to\_cartesian();  
};  
  
struct Camera{  
 SphericalCoords center\_position;   
 POINT offsets;  
};

Структуры *Angle* и *Radius*, были созданы для предотвращения переполнения значений сферических координат.

Основной класс Object содержит информацию необходимую для рендеринга модели. Метод *update* реализует обновление экранных координат объекта *screen\_points*, путем преобразования мировых координат в экранные, в зависимости от расположения камеры и экранной плоскости.

void Object::update(Camera& cam, int w\_height, int w\_width) {  
 for (int i = 0; i < vertices\_count; ++i) {  
 screen\_points[i]=RenderPoint(world\_points[i],cam,w\_height,w\_width);  
 }  
}

Функция *RenderPoint* последовательно производит следующие преобразования над каждой мировой координатой модели:

Видовые преобразования:

Значений координат выведены из формулы видовых преобразований:

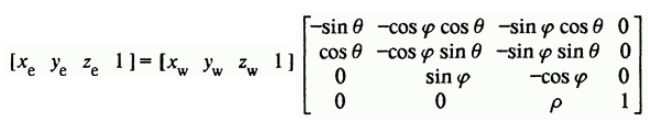


Рисунок 17 Формула видовых преобразований

double M3[3] = {Y\*cos\_A - X\*sin\_A,  
 Z\*sin\_B - X\*cos\_A\*cos\_B - Y\*sin\_A\*cos\_B,  
 -X\*cos\_A\*cos\_B - Y\*sin\_A\*sin\_B - Z\*cos\_B + cam.center\_position.r.value};  
  
POINT3 screen\_dot = PointEx(M3[0], M3[1], M3[2]);

Перспективные преобразования:  
X = screen\_dot.x,Y = screen\_dot.y, Z = screen\_dot.z;  
  
POINT projected = Point(Round(X/Z\*cam.dc),  
 Round(Y/Z\*cam.dc));

Экранные:

return Point(projected.x + w\_width/2 + cam.offsets.x,-projected.y + w\_height/1.5 + cam.offsets.y);

Метод *draw* рисует ребра модели по координатам *screen\_points* и их индексам в *faces*.

void Object::draw(HDC hdc,COLORREF color) {  
 SelectObject(hdc,CreatePen(**PS\_SOLID**,0,color));  
  
 POINT p1,p2;  
 for (int i = 0; i < edges\_count; ++i) {  
 p1 = screen\_points[faces[i].order[0]];  
 MoveToEx(hdc,p1.x,p1.y, nullptr);  
 for (int j = 1; j < faces[i].vertices\_count; ++j) {  
 p2 = screen\_points[faces[i].order[j]];  
 LineTo(hdc,p2.x,p2.y);  
 }  
 LineTo(hdc,p1.x,p1.y);  
 }  
}

Функции рисования земли и осей абсцисс, ординат и аппликат были реализованы отдельно, т.к. координаты необходимые для их рендеринга легко рассчитываются и нет необходимости хранить объекты под них.

# **6 Реализация интерфейса в отношении управления камерой**

Управления камерой производится с помощью клавиатуры

W,A,S,D - поворот камеры

R,T - приблизиться к объекту, отдалиться от объекта.

F,G - увеличить, уменьшить влияние перспективы (поле зрения)

Стрелки - перемещение экранной плоскости

Дополнительный функционал:

F5 – вывод положения камеры в консоль

# **7 Отладка и тестирование программы**

Таблица 1 – Описание поведения программы при тестировании

|  |  |  |
| --- | --- | --- |
| Выполнение действия | Ожидаемый результат | Полученный результат |
| Загрузка существующего файла | Импорт файла, рисование модели | Импорт файла, рисование модели |
| Загрузка несуществующего файла | Аварийное завершение программы | Аварийное завершение программы |
| Нажатие клавиш поворота камера | Поворот проекции | Поворот проекции |
| Нажатие клавиш приближения, удаления камеры | Приближение, удаление проекции | Приближение, удаление проекции |
| Нажатия клавиш перемещения экранной плоскости | Перемещение экранной плоскости | Перемещение экранной плоскости |
| Нажатие клавиш изменения поля зрения | Увеличение, уменьшение поля зрения | Увеличение, уменьшение поля зрения только до некоторого значения, |
| Нажатие клавиши, для вывода информации о положении камеры | Вывод информации о положении камеры | Вывод информации о положении камеры |

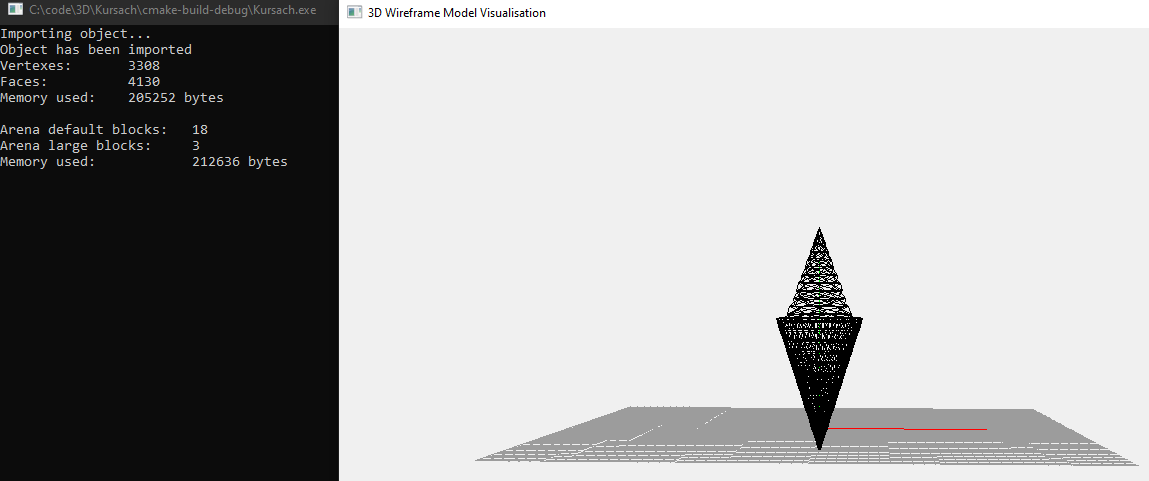


Рисунок 18 Успешная загрузка файла

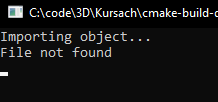


Рисунок 19 Файл не найден

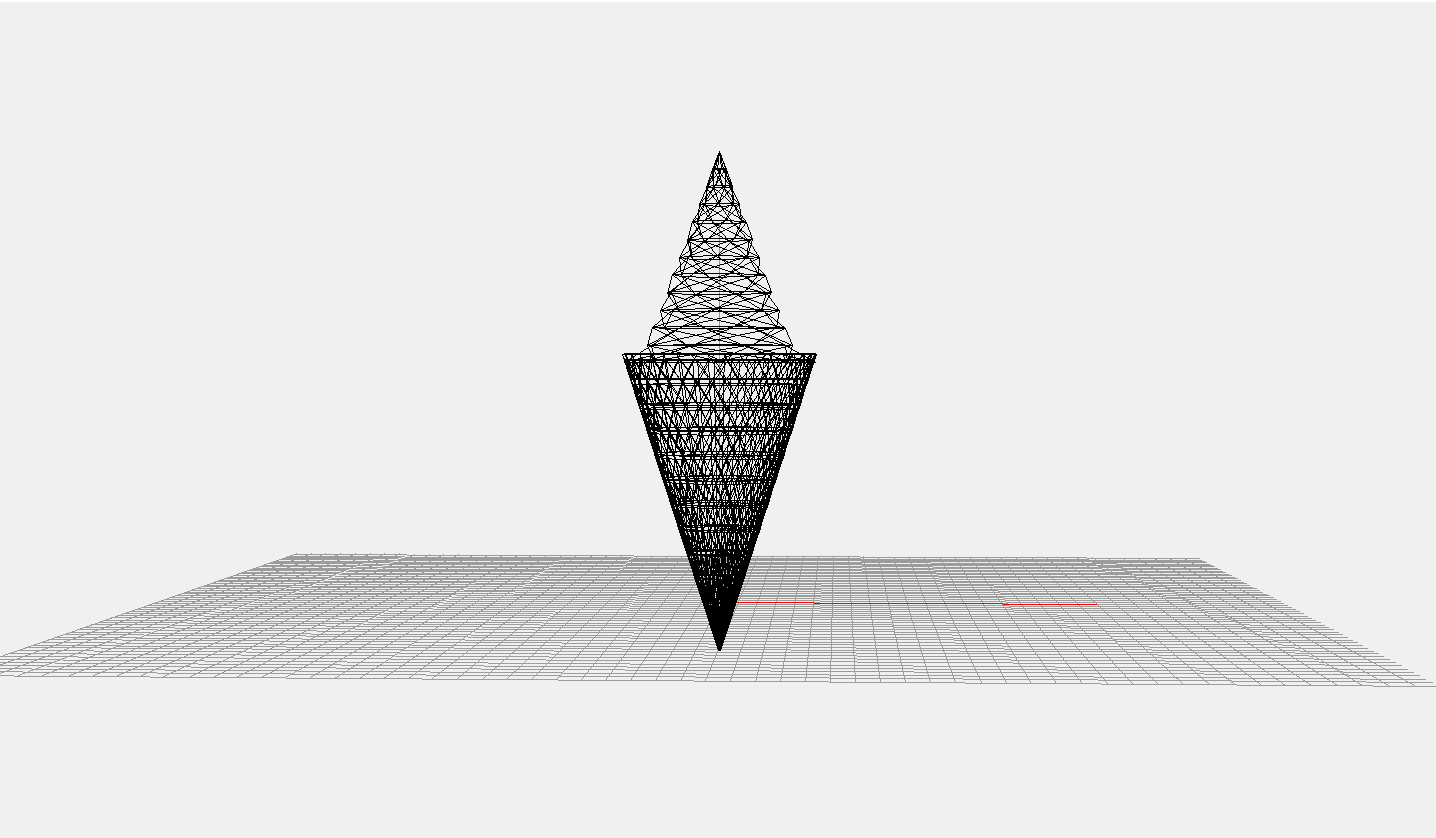


Рисунок 20 Перемещение камеры 1

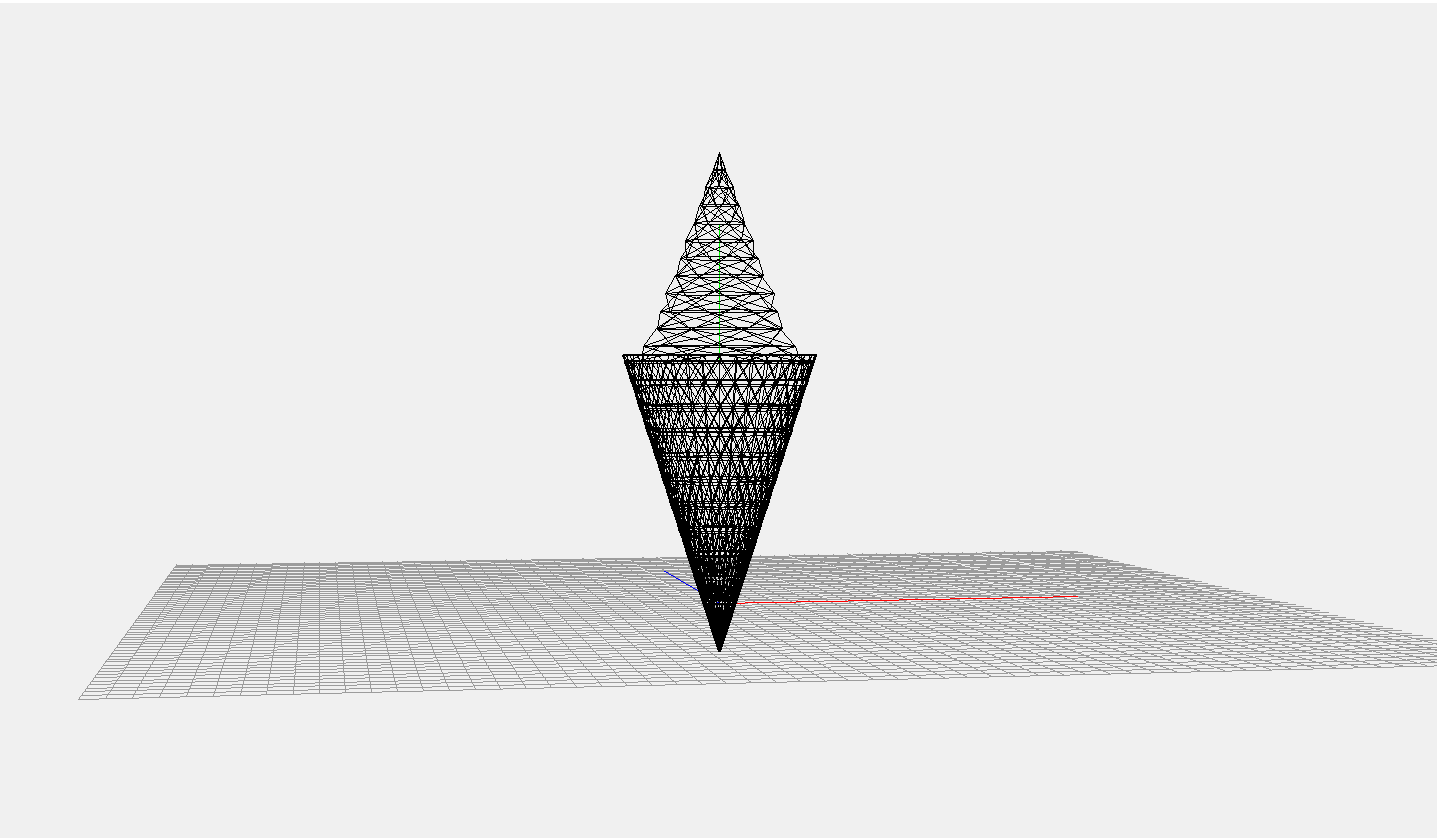


Рисунок 21 Перемещение камеры 2

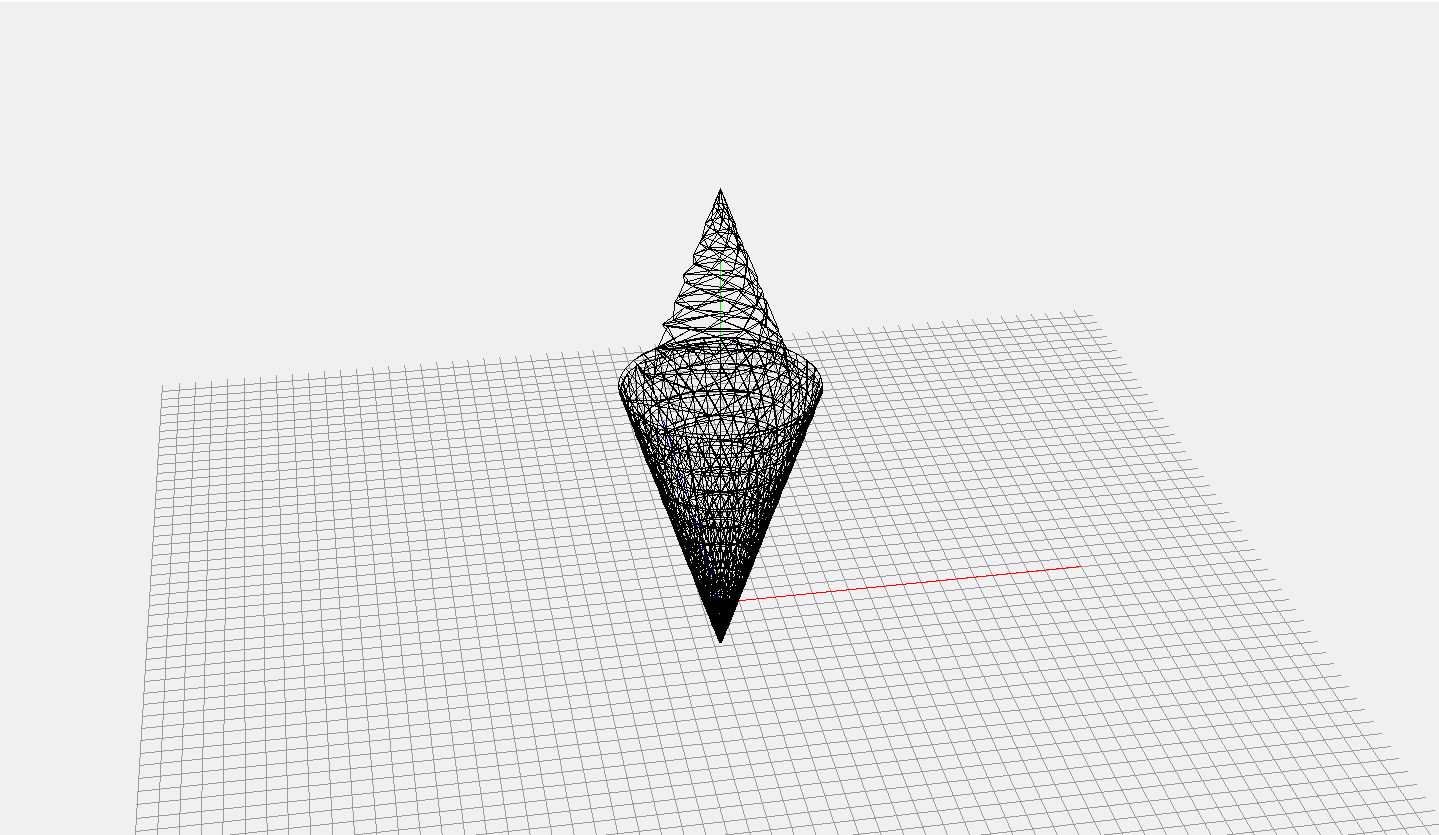


Рисунок 22 Перемещение камеры 3

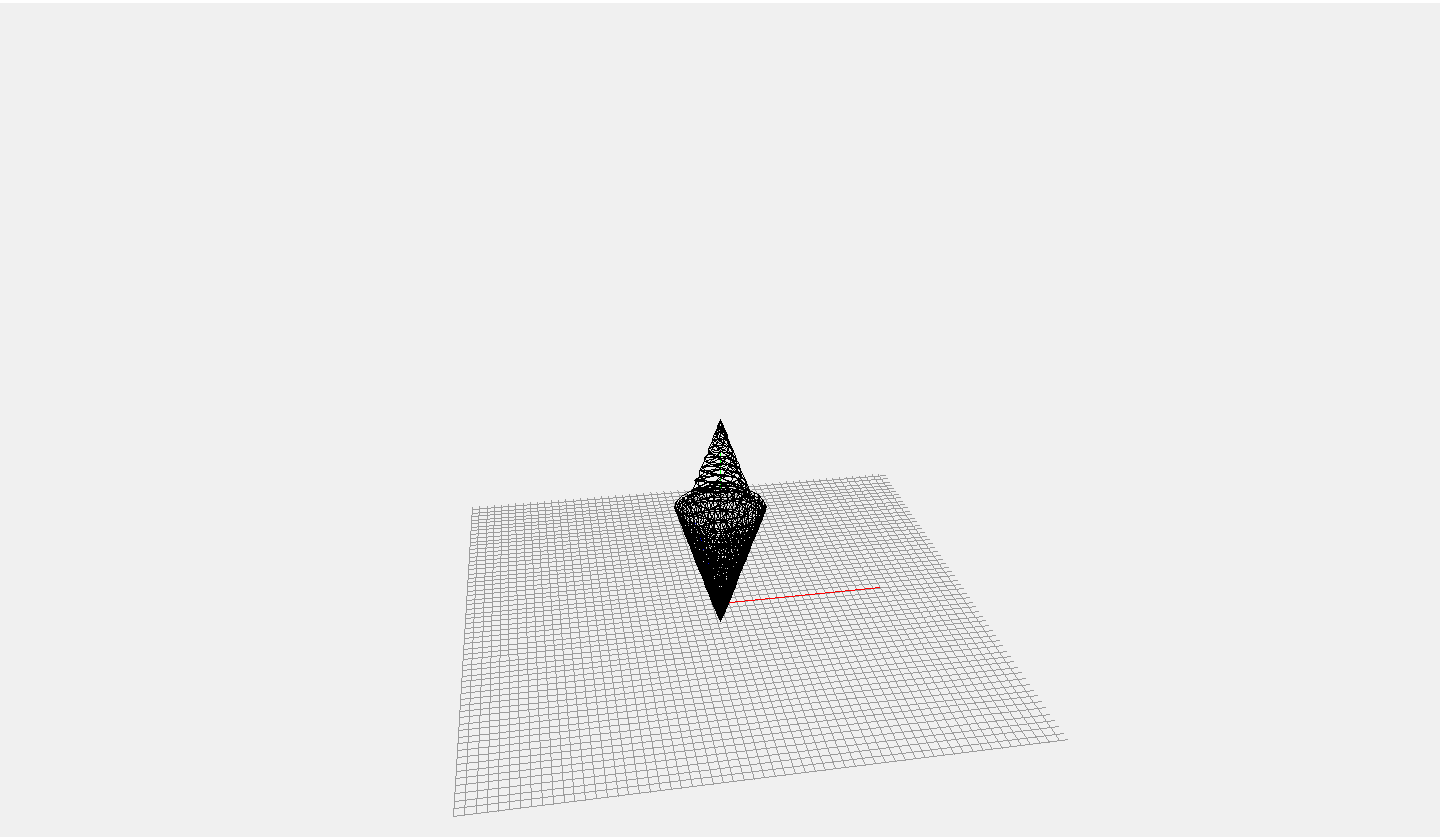


Рисунок 23 Перемещение камеры 4

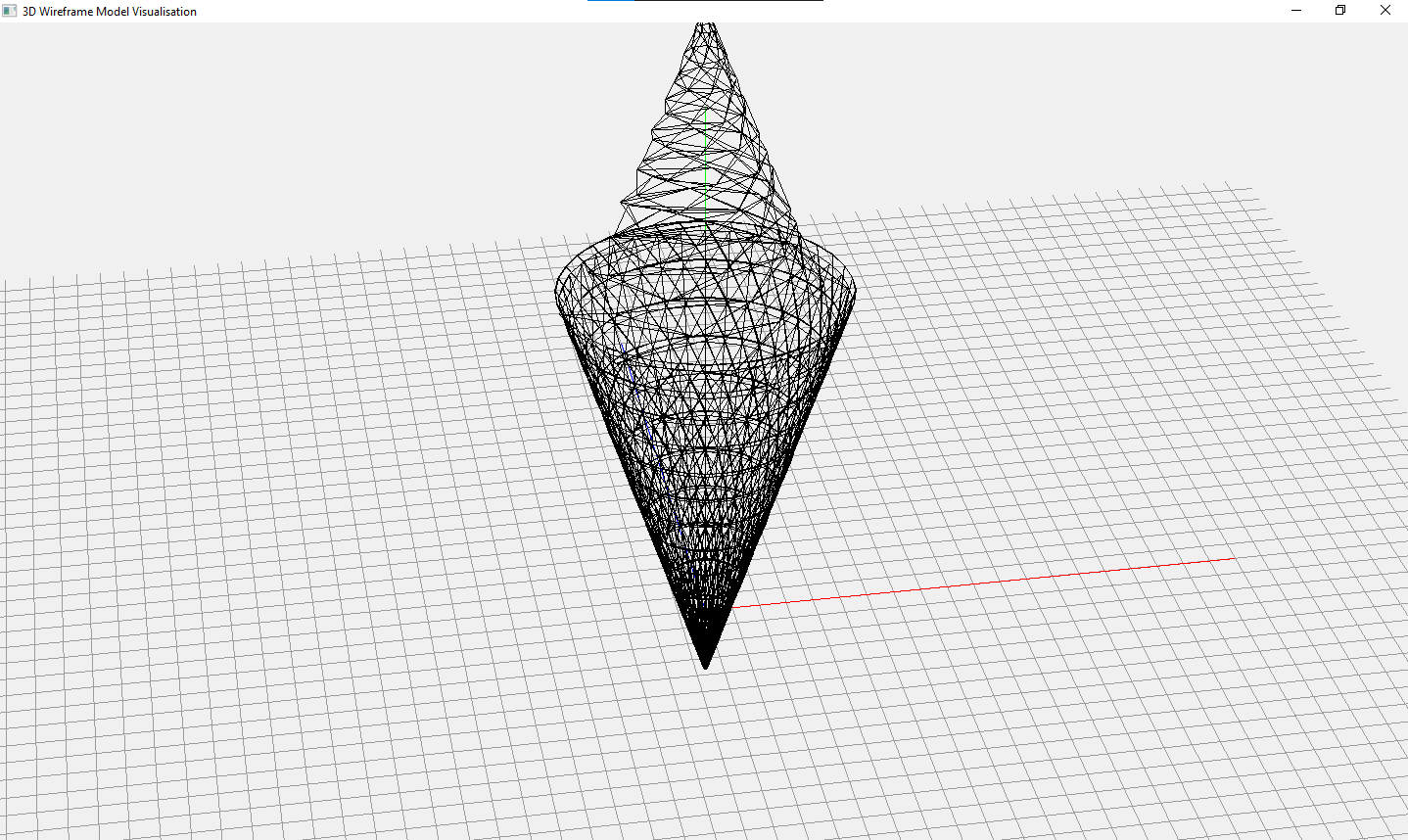


Рисунок 24 Изменение поля зрения 1

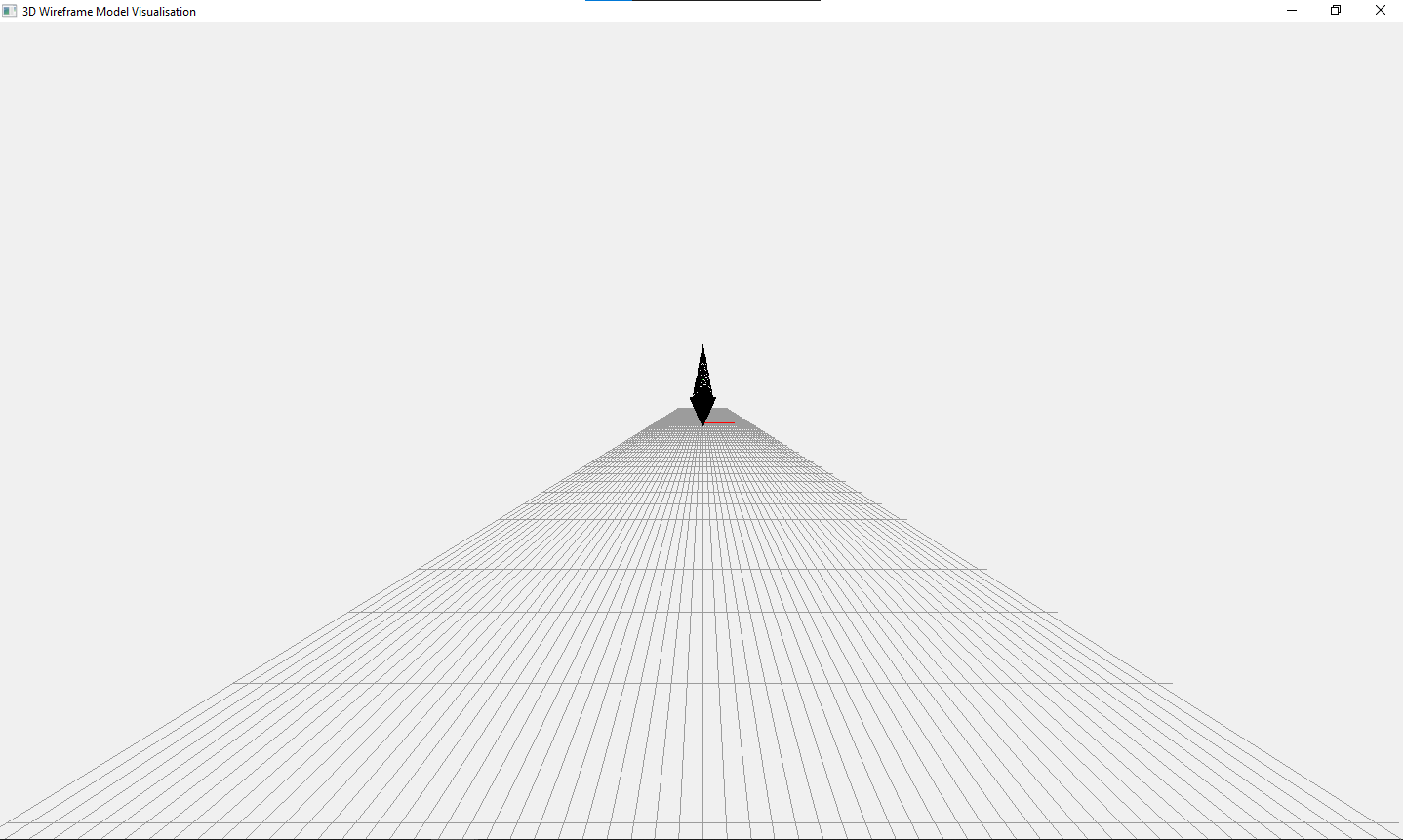


Рисунок 25 Изменение поля зрения 2

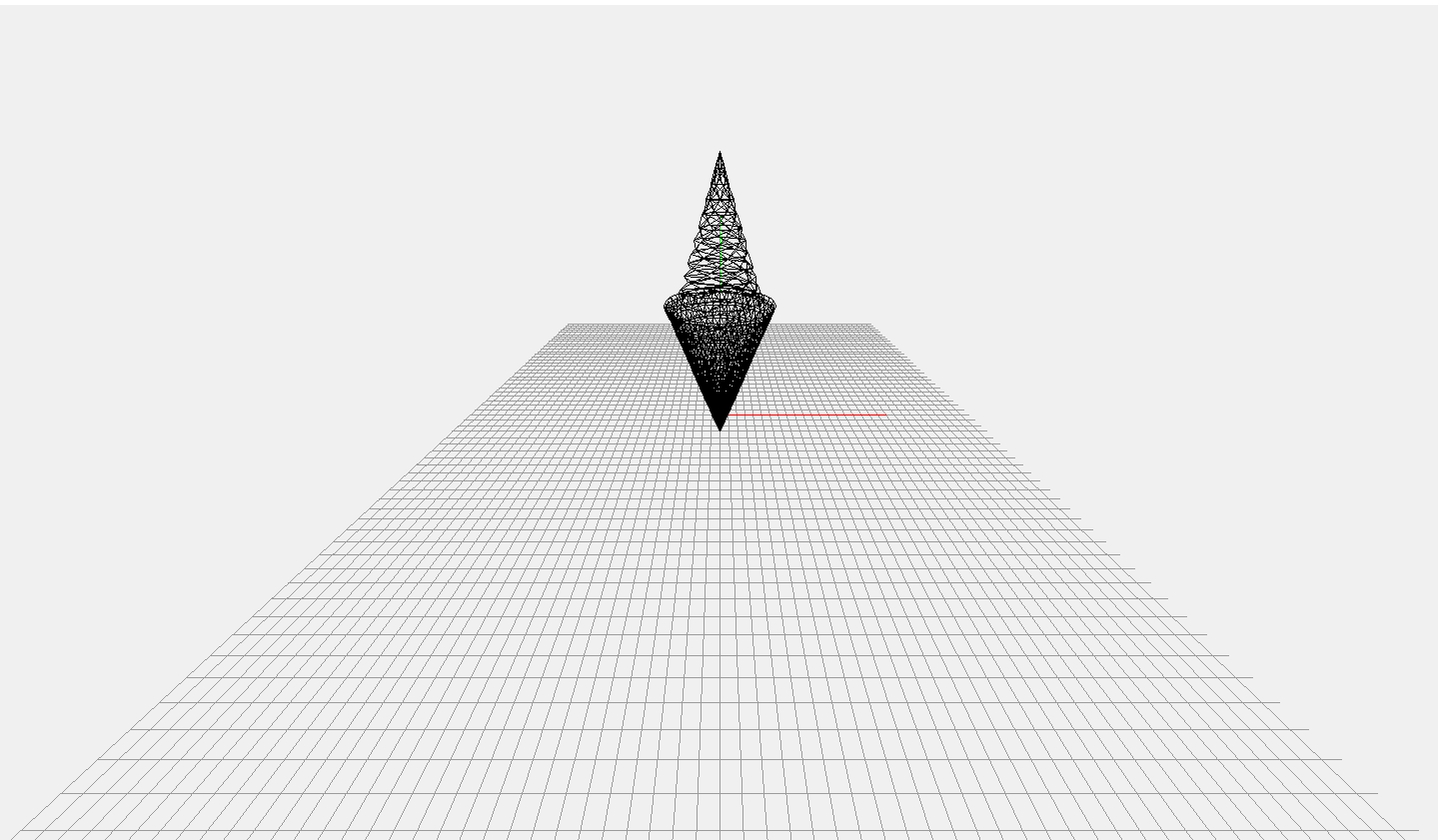


Рисунок 26 Изменение поля зрения 3

Дефекты, вызванные большим увеличением поля зрения. Возможная причина – переполнение значений координат.



Рисунок 27 Дефекты рендеринга

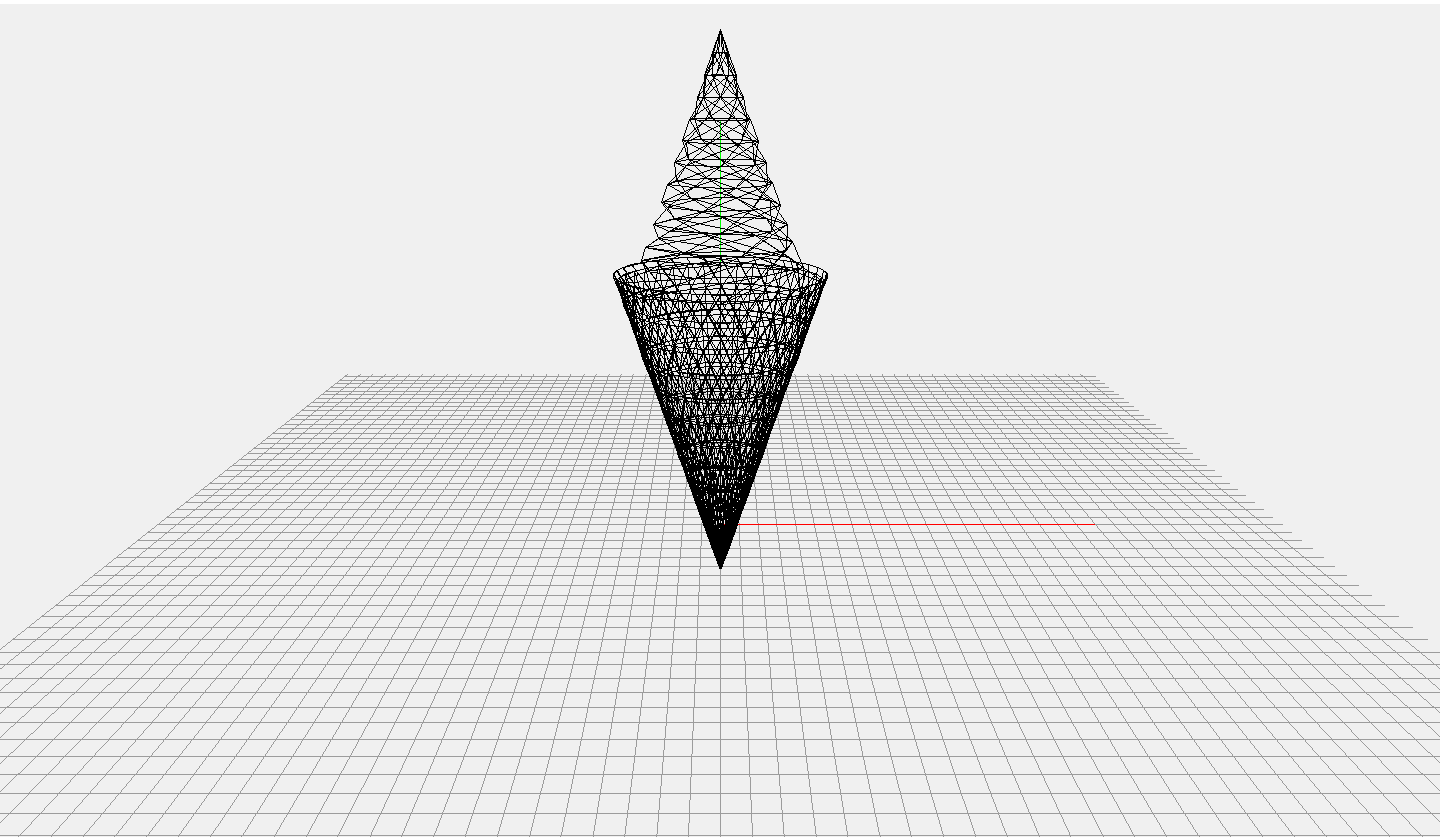


Рисунок 28 Перемещение экранной плоскости 1

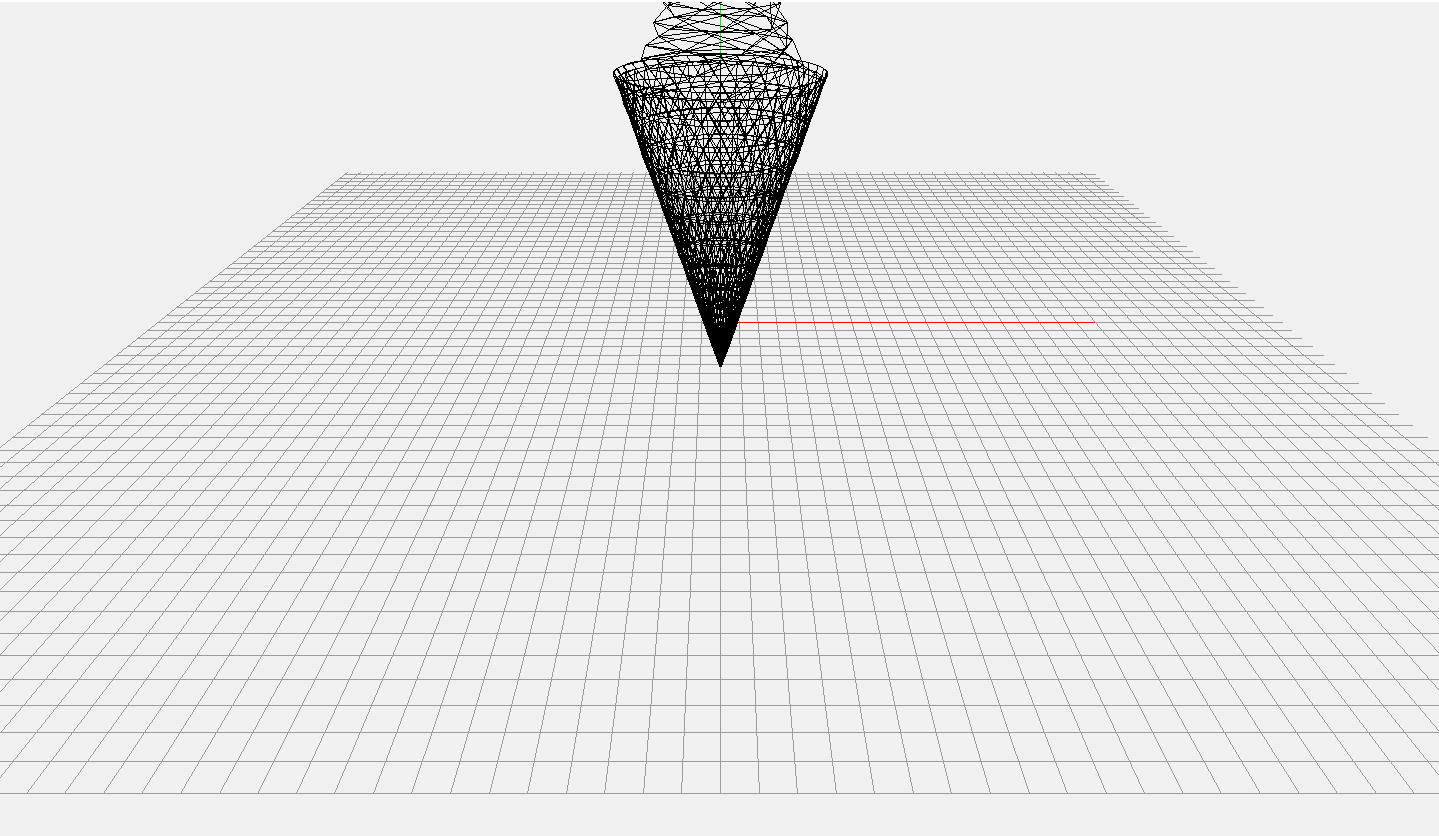


Рисунок 29Перемещение экранной плоскости 2

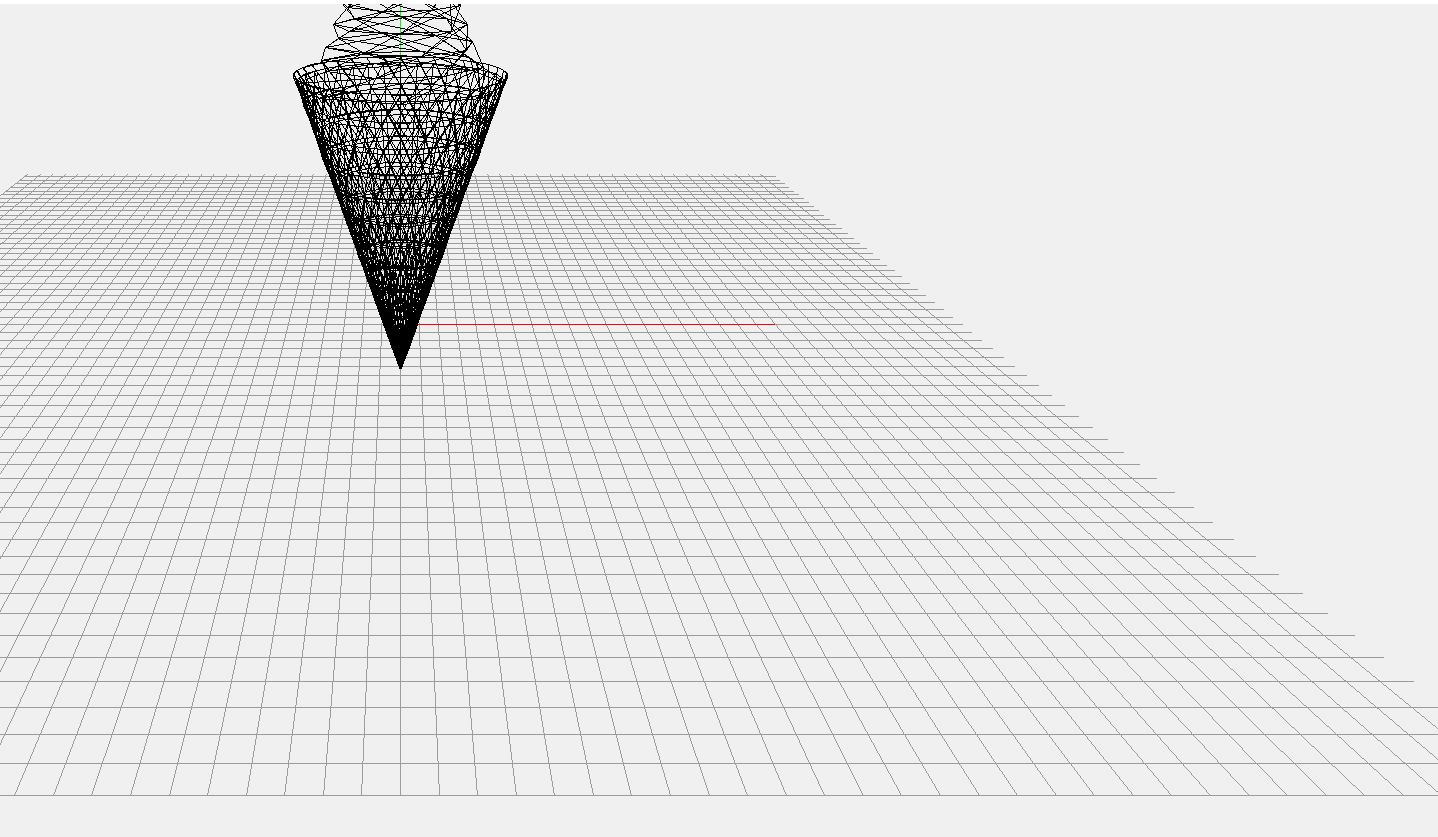


Рисунок 30 Перемещение экранной плоскости 3

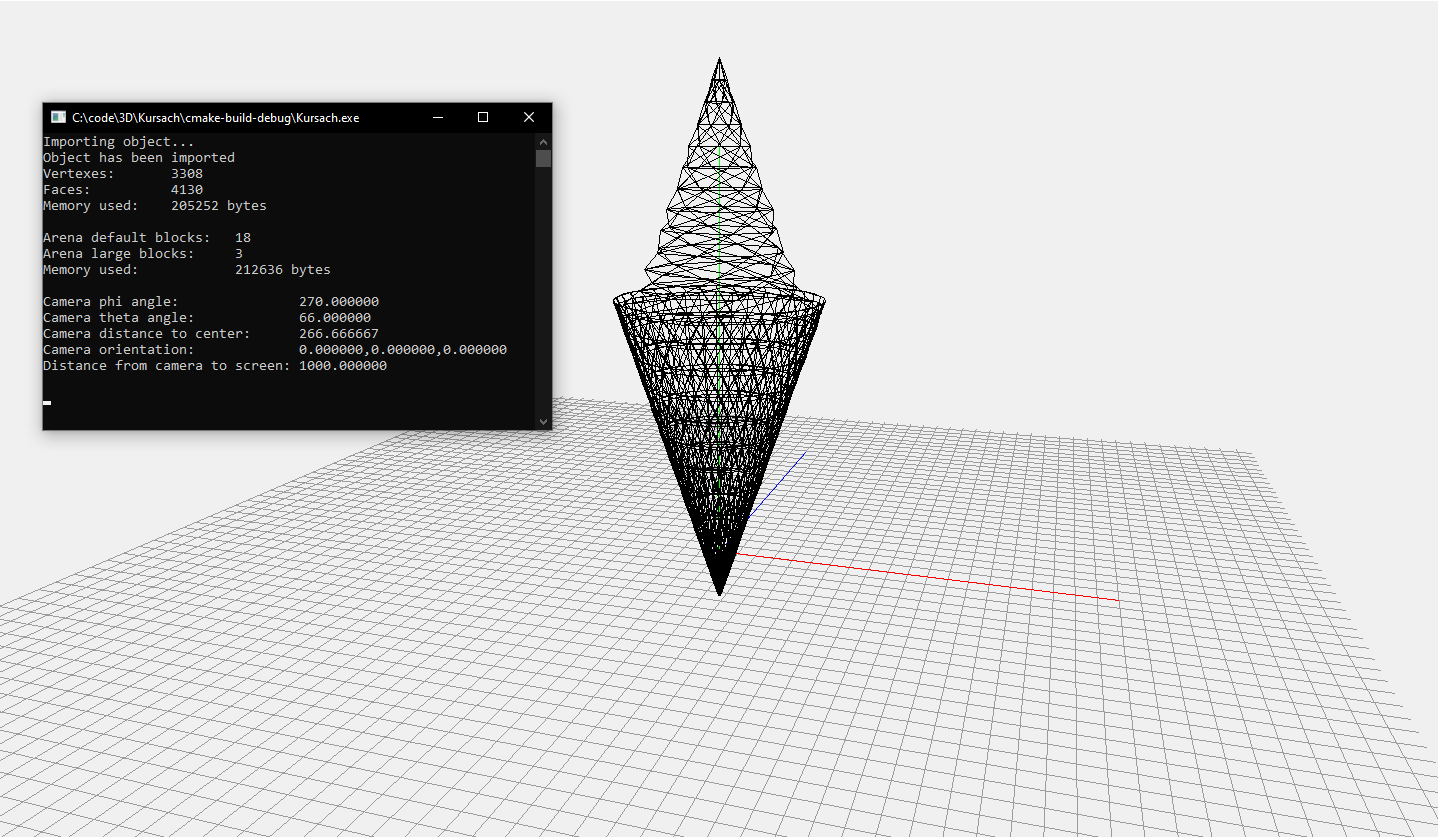


Рисунок 31 Информация о положении камеры 1

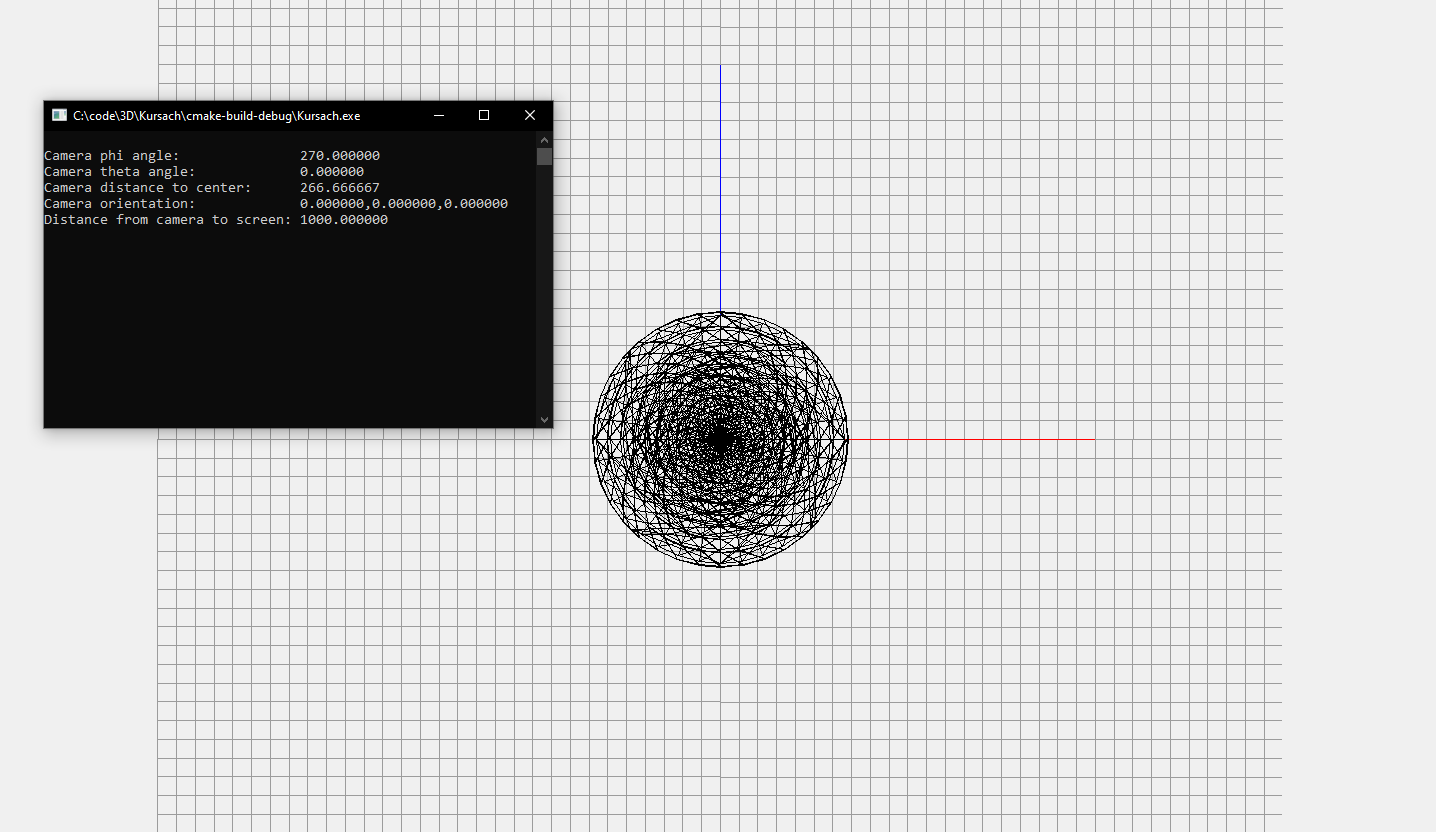


Рисунок 32 Информация о положении камеры 2

Заключение

В процессе создании этого проекта были разработаны трехмерная модель и программа, реализующая двухмерную проекцию этой модели.

При выполнении данной курсовой работы были получены навыки создания объектов в среде 3D моделирования, разработки программ на языке С++, с использованием Win32 API.

Программа работает корректно, за исключением случая, когда сильно увеличивается поле зрение, при наблюдении за объектом.

Программа имеет достаточный для использования функционал. Функционал программы можно расширить, добавив управление

мышью, возможность импортировать несколько объектов и взаимодействовать с каждым из них.

# **Список литературы**

1. Аммерал, Л., 1992. Принципы программирования в машинной графике. М.: Сол Систем.

2. Windows GDI. Просмотрено 29.05.2021 docs.microsoft.com/en-us/windows/win32/gdi/windows-gdi.

3. 3D projection. Просмотрено 29.05.2021 en.wikipedia.org/wiki/3D\_projection.

4. Blender 2.92 Руководство Пользователя Просмотрено 29.05.2021 docs.blender.org/manual/ru/2.92/.

# **Приложение А**

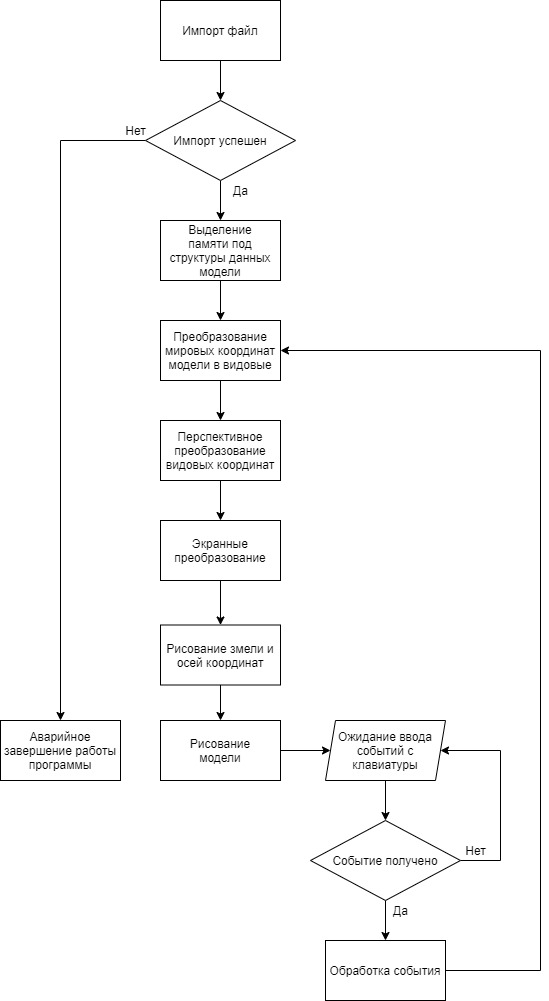


Рисунок 33 Структурная схема алгоритма

# Приложение В

Листинг программы:

main.cpp

#include "resource.h"  
#include "3D.h"  
  
namespace env{  
 int ScreenWidth = GetSystemMetrics(**SM\_CXSCREEN**)/1.5; *//Разрешение экрана по вертикали* int ScreenHeight = GetSystemMetrics(**SM\_CYSCREEN**)/1.5; *//Разрешение экрана по горизонтали* double window\_size = 1;  
  
 Camera main\_camera = {{600, 273, 84},  
 {0,0,0},  
 {0,0},1000};  
  
 arena\_t objects\_arena = **EMPTY\_ARENA**;  
 Object\* main\_object = **NULL**;  
}  
  
using namespace env;  
  
#define **MAX\_LOADSTRING** 100  
HINSTANCE hInst;  
  
LRESULT **CALLBACK** WndProc(HWND, UINT, WPARAM, LPARAM);  
INT\_PTR **CALLBACK** About(HWND, UINT, WPARAM, LPARAM);  
  
int **WINAPI** WinMain(HINSTANCE currentInstance, HINSTANCE previousInstance, PSTR cmdLine, INT cmdCount){  
 AllocConsole();  
 freopen("CONOUT$", "w", **stdout**);  
  
 printf("Importing object...\n");  
 main\_object = parse\_obj(&objects\_arena,"./ice.obj");  
  
 *// Register the window class* const char \*CLASS\_NAME = "myWin32WindowClass";  
 WNDCLASS wc{};  
 wc.hInstance = currentInstance;  
 wc.lpszClassName = CLASS\_NAME;  
 wc.hCursor = **LoadCursor**(nullptr, **IDC\_ARROW**);  
 wc.hbrBackground = (HBRUSH)**COLOR\_WINDOW**;  
 wc.lpfnWndProc = WndProc;  
 **RegisterClass**(&wc);  
  
 *// Create the window* **CreateWindow**(CLASS\_NAME, "3D Wireframe Model Visualisation",  
 WS\_OVERLAPPEDWINDOW | WS\_VISIBLE, *// Window style* CW\_USEDEFAULT, CW\_USEDEFAULT, *// Window initial position* ScreenWidth, ScreenHeight, *// Window size* nullptr, nullptr, nullptr, nullptr);  
  
  
 *// Цикл основного сообщения:* HACCEL hAccelTable = **LoadAccelerators**(currentInstance, **MAKEINTRESOURCE**(IDC\_LAB1));  
  
 MSG msg;  
  
 while (**GetMessage**(&msg, nullptr, 0, 0))  
 {  
 if (!**TranslateAccelerator**(msg.hwnd, hAccelTable, &msg))  
 {  
 TranslateMessage(&msg);  
 **DispatchMessage**(&msg);  
 }  
 }  
  
 return 0;  
}  
  
LRESULT **CALLBACK** WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam){  
 int wmId, wmEvent;  
 PAINTSTRUCT ps;  
 HDC hdc,hdc1;  
  
 const double RotateAngle = 1;  
  
 switch (message)  
 {  
 case **WM\_COMMAND**:  
 wmId = **LOWORD**(wParam);  
 wmEvent = **HIWORD**(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\_KEYDOWN**:  
 {  
 switch (wParam)  
 {  
 case 'W':  
 main\_camera.center\_position.theta += RotateAngle;  
 InvalidateRect(hWnd, **NULL**, **TRUE**);  
 break;  
 case 'S':  
 main\_camera.center\_position.theta -= RotateAngle;  
 InvalidateRect(hWnd, **NULL**, **TRUE**);  
 break;  
 case 'A':  
 main\_camera.center\_position.phi += RotateAngle;  
 InvalidateRect(hWnd, **NULL**, **TRUE**);  
 break;  
 case 'D':  
 main\_camera.center\_position.phi -= RotateAngle;  
 InvalidateRect(hWnd, **NULL**, **TRUE**);  
 break;  
 case 'X': {  
 window\_size \*= 1.5;  
 ScreenWidth \*= 1.5;  
 ScreenHeight \*= 1.5;  
 SetWindowPos(  
 hWnd,  
 **HWND\_TOP**,  
 0,  
 0,  
 ScreenWidth,  
 ScreenHeight,  
 **SWP\_NOMOVE** );  
 InvalidateRect(hWnd, **NULL**, **TRUE**);  
 UpdateWindow(hWnd);  
 break;  
 }  
 case 'Z': {  
 window\_size /= 1.5;  
 ScreenWidth /= 1.5;  
 ScreenHeight /= 1.5;  
 SetWindowPos(  
 hWnd,  
 **HWND\_TOP**,  
 0,  
 0,  
 ScreenWidth,  
 ScreenHeight,  
 **SWP\_NOMOVE** );  
 InvalidateRect(hWnd, **NULL**, **TRUE**);  
 UpdateWindow(hWnd);  
 break;  
 }  
 case **VK\_F5**:{  
 main\_camera.log(**stdout**);  
 break;  
 }  
 case **VK\_UP**:  
 main\_camera.offsets.y -= 10;  
 InvalidateRect(hWnd, **NULL**, **TRUE**);  
 break;  
 case **VK\_DOWN**:  
 main\_camera.offsets.y += 10;  
 InvalidateRect(hWnd, **NULL**, **TRUE**);  
 break;  
 case **VK\_LEFT**:  
 main\_camera.offsets.x -= 10;  
 InvalidateRect(hWnd, **NULL**, **TRUE**);  
 break;  
 case **VK\_RIGHT**:  
 main\_camera.offsets.x += 10;  
 InvalidateRect(hWnd, **NULL**, **TRUE**);  
 break;  
  
 case 'R':  
 main\_camera.dc \*= 1.5;  
 InvalidateRect(hWnd, **NULL**, **TRUE**);  
 break;  
 case 'T':  
 main\_camera.dc /= 1.5;  
 InvalidateRect(hWnd, **NULL**, **TRUE**);  
 break;  
 case 'F':  
 main\_camera.center\_position.r /= 1.5;  
 InvalidateRect(hWnd, **NULL**, **TRUE**);  
 break;  
 case 'G':  
 main\_camera.center\_position.r \*= 1.5;  
 InvalidateRect(hWnd, **NULL**, **TRUE**);  
 break;  
 }  
 } break;  
 case **WM\_PAINT**:{  
 hdc = BeginPaint(hWnd, &ps);  
  
 main\_object->update(main\_camera,ScreenHeight,ScreenWidth);  
 draw\_ground(hdc,main\_camera,ScreenHeight,ScreenWidth);  
 draw\_coordinate\_lines(hdc, main\_camera, ScreenHeight, ScreenWidth);  
 main\_object->draw(hdc,0);  
  
 EndPaint(hWnd, &ps);  
 break;  
 }  
 case **WM\_DESTROY**:  
 PostQuitMessage(0);  
 break;  
 default:  
 return **DefWindowProc**(hWnd, message, wParam, lParam);  
 }  
 return 0;  
}

3D.h

#ifndef **FUNCTIONS\_H**#define **FUNCTIONS\_H**#include "framework.h"  
#include "arena.h"  
  
#define **PI** ((double)3.14159265359)  
  
struct POINT3{  
 double x;  
 double y;  
 double z;  
};  
  
inline POINT Point(int x, int y){  
 return { x,y };  
}  
  
inline POINT3 PointEx(double x, double y, double z) {  
 return { x,y,z };  
}  
  
inline int Round(double x){  
 if (x < 0) x -= 0.5; else x += 0.5;  
 return (int)x;  
}  
  
inline double DegreeToRad(double Angle) {  
 return (((double)Angle \* **PI**) / 180);  
}  
  
#define **ANGLE\_UPPER\_LIMIT**(deg) ((deg) < 360 ? (deg) : (deg)-360)  
#define **ANGLE\_LOWER\_LIMIT**(deg) ((deg) > 0 ? (deg) : (deg)+360)  
  
struct Radius{  
 double value;  
  
 inline double operator =(double r){  
 value = (r > 0 ? r : 0);  
 return value;  
 }  
  
 inline void operator +=(double r){  
 value += r;  
 }  
 inline void operator -=(double r){  
 value = (value - r > 0 ? value - r : 0);  
 }  
 inline void operator \*=(double r){  
 value \*= r;  
 }  
 inline void operator /=(double r){  
 value /= r;  
 }  
 inline bool operator ==(double r){  
 return value == r;  
 }  
 inline bool operator !=(double r){  
 return value != r;  
 }  
};  
  
struct Angle{  
 double value;  
  
 void operator +=(double deg){  
 value = **ANGLE\_UPPER\_LIMIT**(value + deg);  
 }  
 void operator -=(double deg){  
 value = **ANGLE\_LOWER\_LIMIT**(value - deg);  
 }  
 void operator \*=(double deg){  
 value = **ANGLE\_UPPER\_LIMIT**(value \* deg);  
 }  
 void operator /=(double deg){  
 value /= deg;  
 }  
 inline bool operator ==(double deg){  
 return value == deg;  
 }  
 inline bool operator !=(double deg){  
 return value != deg;  
 }  
};  
  
struct SphericalCoords{  
 Radius r;  
 Angle phi;  
 Angle theta;  
  
 POINT3 to\_cartesian();  
};  
  
struct TaitBryanAngles{  
 Angle x;  
 Angle y;  
 Angle z;  
};  
  
struct Camera{  
 SphericalCoords center\_position;  
 TaitBryanAngles orientation;  
 POINT offsets;  
 double dc = 1000;*//distance from screen to camera centre* void log(FILE\* stream){  
 fprintf(stream,"Camera phi angle:\t\t%f\n",center\_position.phi.value);  
 fprintf(stream,"Camera theta angle:\t\t%f\n",center\_position.theta.value);  
 fprintf(stream,"Camera distance to center:\t%f\n",center\_position.r.value);  
 TaitBryanAngles C = orientation;  
 fprintf(stream,"Camera orientation:\t\t%f,%f,%f\n",C.x,C.y,C.z);  
 fprintf(stream,"Distance from camera to screen:\t%f\n\n",dc);  
 }  
};  
  
struct Face{  
 uint32\_t\* order;  
 uint32\_t vertices\_count;  
};  
  
class Object{  
private:  
 POINT3\* world\_points;  
 POINT\* screen\_points;  
 Face\* faces;  
 uint32\_t edges\_count;  
 uint32\_t vertices\_count;  
public:  
 Object(){  
 world\_points = nullptr;  
 screen\_points = nullptr;  
 faces = nullptr;  
 edges\_count = 0;  
 vertices\_count = 0;  
 }  
 Object(POINT3\* world\_points, POINT\* screen\_points, Face\* edges, uint32\_t vertices\_count, uint32\_t edges\_count){  
 this->world\_points = world\_points;  
 this->screen\_points = screen\_points;  
 this->faces = edges;  
 this->edges\_count = edges\_count;  
 this->vertices\_count = vertices\_count;  
 }  
 void update(Camera& cam, int w\_height, int w\_width);  
 void draw(HDC hdc,COLORREF color);  
};  
  
void draw\_ground(HDC hdc,Camera& cam,int w\_height, int w\_width);  
void draw\_coordinate\_lines(HDC hdc, Camera& cam, int w\_height, int w\_width);  
  
Object\* parse\_obj(arena\_t\* arena,const char\* fname);  
  
#endif **FUNCTIONS\_H**

3D.cpp

#define **RENDER**(x,y,z) RenderPoint(PointEx(x,y,z),cam,w\_height,w\_width)  
void draw\_ground(HDC hdc,Camera& cam,int w\_height, int w\_width){  
 SelectObject(hdc,CreatePen(**PS\_SOLID**,0,**RGB**(156, 156, 156)));  
  
 int range[2] = {-150,150};  
 int step = 5;  
 int lines\_count = (-range[0]+range[1])/step;  
  
 POINT p1,p2;  
 *//vertical* for (int i = 0; i < lines\_count; ++i) {  
 p1 = **RENDER**(range[0] + step \* i, range[0], 0);  
 p2 = **RENDER**(range[0] + step \* i, range[1], 0);  
  
 MoveToEx(hdc,p1.x,p1.y, nullptr);  
 LineTo(hdc,p2.x,p2.y);  
 }  
  
 *//horizontal* for (int i = 0; i < lines\_count; ++i) {  
 p1 = **RENDER**(range[0], range[0] + step \* i, 0);  
 p2 = **RENDER**(range[1], range[0] + step \* i, 0);  
  
 MoveToEx(hdc,p1.x,p1.y, nullptr);  
 LineTo(hdc,p2.x,p2.y);  
 }  
}  
  
void draw\_coordinate\_lines(HDC hdc, Camera& cam, int w\_height, int w\_width){  
 int len = 100;  
  
 POINT p1,p2;  
 p1 = **RENDER**(0, 0, 0);  
  
 *//x* SelectObject(hdc,CreatePen(**PS\_SOLID**,1,**RGB**(255, 0, 0)));  
 p2 = **RENDER**(len, 0, 0);  
  
 MoveToEx(hdc,p1.x,p1.y,**NULL**);  
 LineTo(hdc,p2.x,p2.y);  
  
 *//y* SelectObject(hdc,CreatePen(**PS\_SOLID**,1,**RGB**(0, 0, 255)));  
 p2 = **RENDER**(0, len, 0);  
 MoveToEx(hdc,p1.x,p1.y,**NULL**);  
 LineTo(hdc,p2.x,p2.y);  
  
 *//z* SelectObject(hdc,CreatePen(**PS\_SOLID**,1,**RGB**(0, 255, 0)));  
 p2 = **RENDER**(0, 0, len);  
 MoveToEx(hdc,p1.x,p1.y,**NULL**);  
 LineTo(hdc,p2.x,p2.y);  
}

obj\_parser.cpp

void next\_token() {  
 switch (\*stream) {  
 case '-':  
 case '0': case '1': case '2': case '3': case '4': case '5': case '6': case '7': case '8': case '9': {  
 bool negative;  
 if (\*stream == '-'){  
 negative = true;  
 stream++;  
 }  
 else negative = false;  
  
 const char\* num\_start = stream;  
 while (isdigit(\*stream)) {  
 stream++;  
 }  
 char c = \*stream;  
 stream = num\_start;  
 if (c == '.') {  
 scan\_float();  
 } else {  
 scan\_int();  
 }  
 if (negative){  
 if(token.kind == *TOKEN\_FLOAT*){  
 token.float\_val = -token.float\_val;  
 }else if(token.kind == *TOKEN\_INT*){  
 token.int\_val = ~(token.int\_val -1);  
 }  
 }  
 break;  
 }  
 case 'a': case 'b': case 'c': case 'd':case 'e': case 'f':case 'g': case 'h':case 'i':case 'j':  
 case 'k': case 'l': case 'm': case 'n':case 'o': case 'p':case 'q': case 'r':case 's':case 't':  
 case 'u': case 'v': case 'w': case 'x':case 'y': case 'z':  
 case 'A': case 'B': case 'C': case 'D':case 'E': case 'F':case 'G': case 'H':case 'I':case 'J':  
 case 'K': case 'L': case 'M': case 'N':case 'O': case 'P':case 'Q': case 'R': case 'S': case 'T':  
 case 'U': case 'V': case 'W': case 'X':case 'Y': case 'Z':  
 case '\_': {  
 token.start = stream;  
 while (isalnum(\*stream) || \*stream == '\_') {  
 stream++;  
 }  
 token.kind = *TOKEN\_NAME*;  
  
 if ((stream - token.start) == 1){  
 if(token.start[0] == 'v'){  
 token.kind = *TOKEN\_VERTEX*;  
 goto end;  
 }else if(token.start[0] == 'f'){  
 token.kind = *TOKEN\_FACE*;  
 goto end;  
 }else if(token.start[0] == 'o'){  
 token.kind = *TOKEN\_OBJECT\_NAME*;  
 goto ignore\_line;  
 }  
 }else if(!strncmp(token.start,"vt",2) || !strncmp(token.start,"vn",2)){  
 goto ignore\_line;  
 }else{  
 goto end;  
 }  
 ignore\_line:  
 while (\*stream != '\n') {  
 stream++;  
 }  
 end:  
  
 break;  
 }  
 case '#':{  
 token.kind = *TOKEN\_COMMENT*;  
 while (\*stream != '\n') {  
 stream++;  
 }  
 break;  
 }  
 default:  
 token.kind = (TokenKind)(\*stream++);  
 break;  
 }  
 token.end = stream;  
}  
  
inline bool is\_token(TokenKind kind){  
 return token.kind == kind;  
}  
  
inline bool expect\_token(TokenKind kind){  
 if(is\_token(kind)){  
 next\_token();  
 return true;  
 } else{  
 fatal("unexpected token");  
 return false;  
 }  
}  
  
POINT3 vertex;  
Face face;  
std::vector<uint32\_t> indexes;  
  
static bool parse\_vertex(){  
 next\_token();  
 expect\_token(TokenKind(' '));  
 if(is\_token(*TOKEN\_FLOAT*)){  
 vertex.x = token.float\_val;  
 }else if(is\_token(*TOKEN\_INT*)){  
 vertex.x = token.int\_val;  
 }else{  
 fatal("unexpected token");  
 }  
  
 next\_token();  
 expect\_token(TokenKind(' '));  
 if(is\_token(*TOKEN\_FLOAT*)){  
 vertex.y = token.float\_val;  
 }else if(is\_token(*TOKEN\_INT*)){  
 vertex.y = token.int\_val;  
 }else{  
 fatal("unexpected token");  
 }  
  
 next\_token();  
 expect\_token(TokenKind(' '));  
 if(is\_token(*TOKEN\_FLOAT*)){  
 vertex.z = token.float\_val;  
 }else if(is\_token(*TOKEN\_INT*)){  
 vertex.z = token.int\_val;  
 }else{  
 fatal("unexpected token");  
 }  
  
 next\_token();  
 expect\_token(TokenKind('\n'));  
 if (is\_token(*TOKEN\_VERTEX*)){  
 return false;  
 }else{  
 return true;  
 }  
}  
  
static bool parse\_face(){  
 next\_token();  
 int count = 0;  
 while(token.kind != '\n' && \*(stream+1) !='\0'){  
 expect\_token(TokenKind(' '));  
 if (is\_token(*TOKEN\_INT*)){  
 indexes.push\_back(token.int\_val-1);  
 count++;  
 }else{  
 fatal("unexpected token");  
 }  
 next\_token();  
 if(is\_token(TokenKind('/'))){  
 while(token.kind != TokenKind(' ') && token.kind != TokenKind('\n') && \*(stream+1) !='\0'){  
 next\_token();  
 }  
 }  
 }  
 face.vertices\_count = count;  
  
 next\_token();  
 if (is\_token(*TOKEN\_FACE*)){  
 return false;  
 }else{  
 return true;  
 }  
}  
  
Object\* parse\_obj(arena\_t\* arena,const char\* fname){  
 FILE\* file = fopen(fname,"r");  
  
 if(!file){  
 printf("File not found\n");  
 \_getch();  
 exit(5);  
 }  
  
 init\_stream\_from\_file(file);  
  
 std::vector<POINT3> vertexes;  
 std::vector<Face> faces;  
  
 bool last = false;  
 while(!is\_token(TokenKind('\0'))) {  
 switch (token.kind) {  
 case *TOKEN\_VERTEX*:  
 while(!last){  
 last = parse\_vertex();  
 vertexes.push\_back(vertex);  
 }  
 last = false;  
 break;  
 case *TOKEN\_FACE*:  
 while(!last){  
 last = parse\_face();  
 faces.push\_back(face);  
 }  
 last = false;  
 break;  
 default:  
 next\_token();  
 break;  
 }  
 }  
 Object\* obj = (Object\*)arena\_alloc(arena,sizeof(Object));  
  
 POINT3\* WPT = (POINT3\*)arena\_dup(arena,vertexes.data(),vertexes.size()\*sizeof(POINT3));  
 POINT\* SPT = (POINT\*)arena\_alloc(arena,vertexes.size()\*sizeof(POINT));  
 Face\* \_edges = (Face\*)arena\_dup(arena,faces.data(), faces.size() \* sizeof(Face));  
  
 size\_t object\_size = sizeof(Object)+  
 sizeof(POINT3)\*vertexes.size()+  
 sizeof(POINT)\*vertexes.size()+  
 sizeof(Face)\*faces.size()+  
 sizeof(uint32\_t)\*indexes.size();  
  
 int offset = 0;  
 for (int i = 0; i < faces.size(); ++i) {  
 \_edges[i].order = (uint32\_t\*)arena\_alloc(arena,sizeof(uint32\_t)\*\_edges[i].vertices\_count);  
 for (int j = 0; j < \_edges[i].vertices\_count; ++j) {  
 \_edges[i].order[j] = indexes[j+offset];  
 }  
 offset+=\_edges[i].vertices\_count;  
 \_edges[i].vertices\_count--;  
 }  
  
 free((void\*)stream\_start);  
 \*obj = Object(WPT, SPT, \_edges, vertexes.size(), faces.size());  
  
 printf("Object has been imported\nVertexes:\t%d\nFaces:\t\t%d\nMemory used:\t%u bytes\n\n",  
 vertexes.size(), faces.size(), object\_size);  
 arena\_log(**stdout**,arena);  
  
 return obj;  
}