МИНОБРНАУКИ РФ

ФГБОУ ВО «ИжГТУ имени М.Т. Калашникова»

Кафедра «Программное обеспечение»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по дисциплине «Компьютерная графика»

Выполнил студент группы Б20-191-2: Широбоков А.А.

Приняла: Левицкая Л.Н.

Ижевск 2022

1. Цель работы

Цель курсового проекта – визуализация 3D-модели в формате OBJ, относительно положения наблюдателя и направления его взгляда.

2. Идея решения задачи

Вариант 12. Мельница.

Для начала создадим основные структуры для хранения вершин и поверхностей:

struct point3d

{

float x, y, z;

};

struct rectangle

{

point3d point[4];

wchar\_t sym;

short col;

};

struct triangle

{

point3d point[3];

wchar\_t sym;

short col;

};

Опишем структуру, содержащую все поверхности, считанные из файла, также добавим метод чтения нужной информации из файла:

struct mesh

{

vector<rectangle> rect;

vector<triangle> tri;

bool ReadObjectFile(string sFilename)

{

ifstream f(sFilename);

if (!f.is\_open())

return false;

vector<point3d> verts;

while (!f.eof())

{

char line[128];

f.getline(line, 128);

strstream s;

s << line;

char junk;

if (line[0] == 'v')

{

point3d v;

s >> junk >> v.x >> v.y >> v.z;

verts.push\_back(v);

}

if (line[0] == 'f')

{

string f[4];

string res = "";

vector<int> resvec;

s >> junk >> f[0] >> f[1] >> f[2] >> f[3];

for (const auto& fo : f)

{

res.clear();

for (const auto& fi : fo)

{

if (fi != '/')

{

res += fi;

}

else

{

resvec.push\_back(stoi(res) - 1);

break;

}

}

}

if (resvec.size() == 4)

{

rect.push\_back({ verts[resvec[0]], verts[resvec[1]], verts[resvec[2]], verts[resvec[3]] });

}

else if (resvec.size() == 3)

{

tri.push\_back({ verts[resvec[0]], verts[resvec[1]], verts[resvec[2]] });

}

resvec.clear();

}

}

return true;

}

};

Структура для хранения любой 4х4 матрицы:

struct matrix4x4

{

float m[4][4] = { 0 };

};

Метод умножения вектора на матрицу:

void VectorXMatrix(point3d &p, point3d &outp, matrix4x4 &mat)

{

outp.x = p.x \* mat.m[0][0] + p.y \* mat.m[1][0] + p.z \* mat.m[2][0] + mat.m[3][0];

outp.y = p.x \* mat.m[0][1] + p.y \* mat.m[1][1] + p.z \* mat.m[2][1] + mat.m[3][1];

outp.z = p.x \* mat.m[0][2] + p.y \* mat.m[1][2] + p.z \* mat.m[2][2] + mat.m[3][2];

float w = p.x \* mat.m[0][3] + p.y \* mat.m[1][3] + p.z \* mat.m[2][3] + mat.m[3][3];

if (w != 0.0f)

{

outp.x /= w; outp.y /= w; outp.z /= w;

}

}

Матрица проекции:

matproj.m[0][0] = (float)ScreenHeight() / (float)ScreenWidth() \* 1.0f / tanf(90.0f \* 0.5f / 180.0f \* 3.14159f);

matproj.m[1][1] = 1.0f / tanf(90.0f \* 0.5f / 180.0f \* 3.14159f);

matproj.m[2][2] = 1000.0f / (1000.0f - 0.1f);

matproj.m[3][2] = (-1000.0f \* 0.1f) / (1000.0f - 0.1f);

matproj.m[2][3] = 1.0f;

matproj.m[3][3] = 0.0f;

Матрица поворота по Y и X:

matRotY.m[0][0] = cosf(fTheta);

matRotY.m[0][2] = -sinf(fTheta);

matRotY.m[1][1] = 1;

matRotY.m[2][0] = sinf(fTheta);

matRotY.m[2][2] = cosf(fTheta);

matRotY.m[3][3] = 1;

matRotX.m[0][0] = 1;

matRotX.m[1][1] = cosf(fThetaX \* 0.5f);

matRotX.m[1][2] = sinf(fThetaX \* 0.5f);

matRotX.m[2][1] = -sinf(fThetaX \* 0.5f);

matRotX.m[2][2] = cosf(fThetaX \* 0.5f);

matRotX.m[3][3] = 1;

Нормаль к поверхности:

point3d normal;

normal.x = (recttrans.point[2].y - recttrans.point[1].y) \* (recttrans.point[0].z - recttrans.point[1].z) - (recttrans.point[2].z - recttrans.point[1].z) \* (recttrans.point[0].y - recttrans.point[1].y);

normal.y = (recttrans.point[2].z - recttrans.point[1].z) \* (recttrans.point[0].x - recttrans.point[1].x) - (recttrans.point[2].x - recttrans.point[1].x) \* (recttrans.point[0].z - recttrans.point[1].z);

normal.z = (recttrans.point[2].x - recttrans.point[1].x) \* (recttrans.point[0].y - recttrans.point[1].y) - (recttrans.point[2].y - recttrans.point[1].y) \* (recttrans.point[0].x - recttrans.point[1].x);

float l = sqrtf(normal.x \* normal.x + normal.y \* normal.y + normal.z \* normal.z);

normal.x /= l; normal.y /= l; normal.z /= l;

Задаем освещение и вычисляем насколько похоже направление нормали к направлению света:

point3d light\_direction = { 0.0f, 0.0f, -1.0f };

float l = sqrtf(light\_direction.x \* light\_direction.x + light\_direction.y \* light\_direction.y + light\_direction.z \* light\_direction.z);

light\_direction.x /= l; light\_direction.y /= l; light\_direction.z /= l;

float dp = normal.x \* light\_direction.x + normal.y \* light\_direction.y + normal.z \* light\_direction.z;

CHAR\_INFO c = GetColour(dp);

recttrans.col = c.Attributes;

recttrans.sym = c.Char.UnicodeChar;

Проецируем из 2D в 3D:

VectorXMatrix(recttrans.point[0], rectproj.point[0], matproj);

VectorXMatrix(recttrans.point[1], rectproj.point[1], matproj);

VectorXMatrix(recttrans.point[2], rectproj.point[2], matproj);

VectorXMatrix(recttrans.point[3], rectproj.point[3], matproj);

Масштабирование в поле зрения камеры:

rectproj.point[0].x += 1.0f; rectproj.point[0].y += 1.0f;

rectproj.point[1].x += 1.0f; rectproj.point[1].y += 1.0f;

rectproj.point[2].x += 1.0f; rectproj.point[2].y += 1.0f;

rectproj.point[3].x += 1.0f; rectproj.point[3].y += 1.0f;

rectproj.point[0].x \*= 0.5f \* (float)ScreenWidth();

rectproj.point[0].y \*= 0.5f \* (float)ScreenHeight();

rectproj.point[1].x \*= 0.5f \* (float)ScreenWidth();

rectproj.point[1].y \*= 0.5f \* (float)ScreenHeight();

rectproj.point[2].x \*= 0.5f \* (float)ScreenWidth();

rectproj.point[2].y \*= 0.5f \* (float)ScreenHeight();

rectproj.point[3].x \*= 0.5f \* (float)ScreenWidth();

rectproj.point[3].y \*= 0.5f \* (float)ScreenHeight();

Имитация z-буфера; сортировка поверхностей и их заполнение:

sort(rectsort.begin(), rectsort.end(), [](rectangle& rect1, rectangle& rect2)

{

float z1 = (rect1.point[0].z + rect1.point[1].z + rect1.point[2].z + rect1.point[3].z) / 4.0f;

float z2 = (rect2.point[0].z + rect2.point[1].z + rect2.point[2].z + rect2.point[3].z) / 4.0f;

return z1 > z2;

});

for (auto& rectproj : rectsort)

{

FillTriangle(rectproj.point[0].x, rectproj.point[0].y,

rectproj.point[1].x, rectproj.point[1].y,

rectproj.point[2].x, rectproj.point[2].y,

rectproj.sym, rectproj.col);

FillTriangle(rectproj.point[3].x, rectproj.point[3].y,

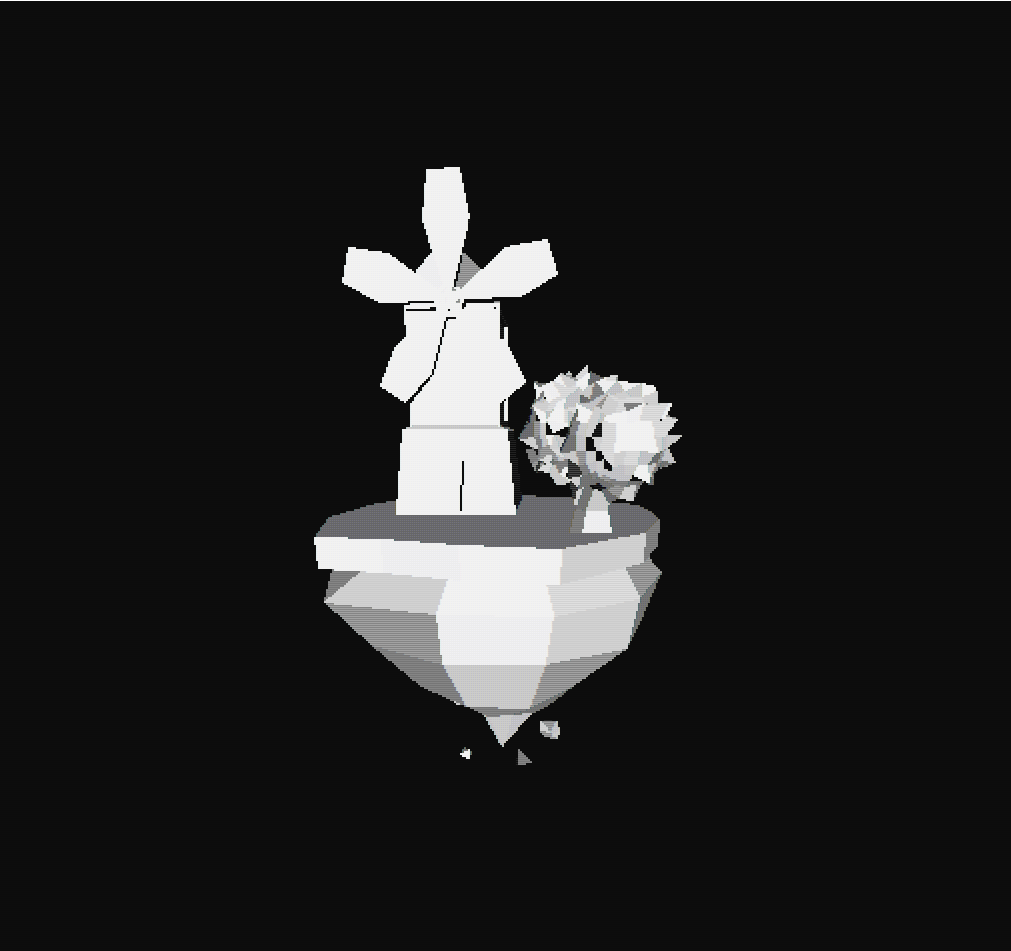
rectproj.point[0].x, rectproj.point[0].y,

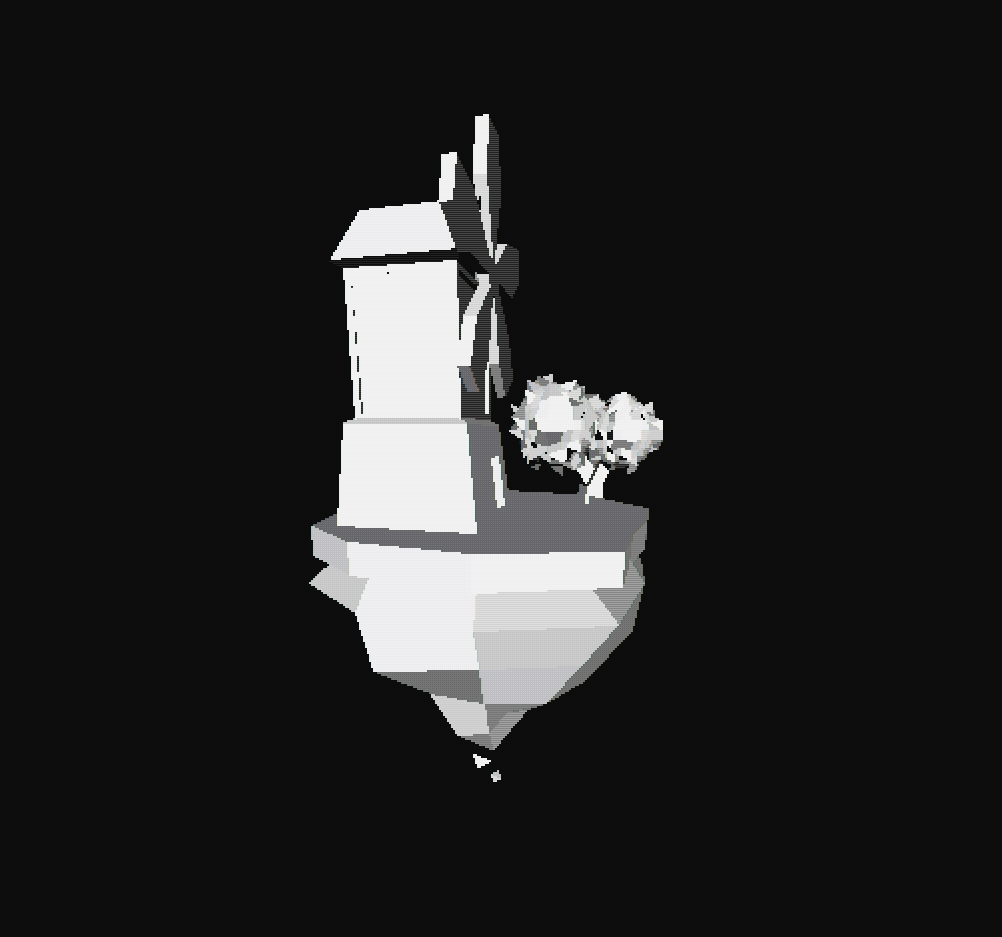
rectproj.point[2].x, rectproj.point[2].y,

rectproj.sym, rectproj.col);

}

3. Результат работы





4. Код программы

#include "olcConsoleGameEngine.h"

#include <fstream>

#include <strstream>

#include <algorithm>

#include <string>

using namespace std;

struct point3d

{

float x, y, z;

};

struct rectangle

{

point3d point[4]; //4 points x,y,z

wchar\_t sym;

short col;

};

struct triangle

{

point3d point[3]; //3 points x,y,z

wchar\_t sym;

short col;

};

struct mesh

{

vector<rectangle> rect;

vector<triangle> tri;

bool ReadObjectFile(string sFilename)

{

ifstream f(sFilename);

if (!f.is\_open())

return false;

vector<point3d> verts;

while (!f.eof())

{

char line[128];

f.getline(line, 128);

strstream s;

s << line;

char junk;

if (line[0] == 'v')

{

point3d v;

s >> junk >> v.x >> v.y >> v.z;

verts.push\_back(v);

}

if (line[0] == 'f')

{

string f[4];

string res = "";

vector<int> resvec;

s >> junk >> f[0] >> f[1] >> f[2] >> f[3];

for (const auto& fo : f)

{

res.clear();

for (const auto& fi : fo)

{

if (fi != '/')

{

res += fi;

}

else

{

resvec.push\_back(stoi(res) - 1);

break;

}

}

}

if (resvec.size() == 4)

{

rect.push\_back({ verts[resvec[0]], verts[resvec[1]], verts[resvec[2]], verts[resvec[3]] });

}

else if (resvec.size() == 3)

{

tri.push\_back({ verts[resvec[0]], verts[resvec[1]], verts[resvec[2]] });

}

resvec.clear();

}

}

return true;

}

};

struct matrix4x4

{

float m[4][4] = { 0 };

};

class d3 : public Engine3d

{

public:

d3()

{

m\_sAppName = L"3dmill";

}

private:

mesh Mill;

matrix4x4 matproj;

point3d pos;

float fTheta;

float fThetaX;

void VectorXMatrix(point3d &p, point3d &outp, matrix4x4 &mat)

{

outp.x = p.x \* mat.m[0][0] + p.y \* mat.m[1][0] + p.z \* mat.m[2][0] + mat.m[3][0];

outp.y = p.x \* mat.m[0][1] + p.y \* mat.m[1][1] + p.z \* mat.m[2][1] + mat.m[3][1];

outp.z = p.x \* mat.m[0][2] + p.y \* mat.m[1][2] + p.z \* mat.m[2][2] + mat.m[3][2];

float w = p.x \* mat.m[0][3] + p.y \* mat.m[1][3] + p.z \* mat.m[2][3] + mat.m[3][3];

if (w != 0.0f)

{

outp.x /= w; outp.y /= w; outp.z /= w;

}

}

CHAR\_INFO GetColour(float lum)

{

short bg\_col, fg\_col;

wchar\_t sym;

int pixel\_bw = (int)(13.0f \* lum);

switch (pixel\_bw)

{

case 0: bg\_col = BG\_BLACK; fg\_col = FG\_BLACK; sym = PIXEL\_SOLID; break;

case 1: bg\_col = BG\_BLACK; fg\_col = FG\_DARK\_GREY; sym = PIXEL\_QUARTER; break;

case 2: bg\_col = BG\_BLACK; fg\_col = FG\_DARK\_GREY; sym = PIXEL\_HALF; break;

case 3: bg\_col = BG\_BLACK; fg\_col = FG\_DARK\_GREY; sym = PIXEL\_THREEQUARTERS; break;

case 4: bg\_col = BG\_BLACK; fg\_col = FG\_DARK\_GREY; sym = PIXEL\_SOLID; break;

case 5: bg\_col = BG\_DARK\_GREY; fg\_col = FG\_GREY; sym = PIXEL\_QUARTER; break;

case 6: bg\_col = BG\_DARK\_GREY; fg\_col = FG\_GREY; sym = PIXEL\_HALF; break;

case 7: bg\_col = BG\_DARK\_GREY; fg\_col = FG\_GREY; sym = PIXEL\_THREEQUARTERS; break;

case 8: bg\_col = BG\_DARK\_GREY; fg\_col = FG\_GREY; sym = PIXEL\_SOLID; break;

case 9: bg\_col = BG\_GREY; fg\_col = FG\_WHITE; sym = PIXEL\_QUARTER; break;

case 10: bg\_col = BG\_GREY; fg\_col = FG\_WHITE; sym = PIXEL\_HALF; break;

case 11: bg\_col = BG\_GREY; fg\_col = FG\_WHITE; sym = PIXEL\_THREEQUARTERS; break;

case 12: bg\_col = BG\_GREY; fg\_col = FG\_WHITE; sym = PIXEL\_SOLID; break;

default:

bg\_col = BG\_BLACK; fg\_col = FG\_BLACK; sym = PIXEL\_SOLID;

}

CHAR\_INFO c;

c.Attributes = bg\_col | fg\_col;

c.Char.UnicodeChar = sym;

return c;

}

public:

bool OnUserCreate() override

{

Mill.ReadObjectFile("prop.obj");

matproj.m[0][0] = (float)ScreenHeight() / (float)ScreenWidth() \* 1.0f / tanf(90.0f \* 0.5f / 180.0f \* 3.14159f);

matproj.m[1][1] = 1.0f / tanf(90.0f \* 0.5f / 180.0f \* 3.14159f);

matproj.m[2][2] = 1000.0f / (1000.0f - 0.1f);

matproj.m[3][2] = (-1000.0f \* 0.1f) / (1000.0f - 0.1f);

matproj.m[2][3] = 1.0f;

matproj.m[3][3] = 0.0f;

return true;

}

bool OnUserUpdate(float fElapsedTime) override

{

Fill(0, 0, ScreenWidth(), ScreenHeight(), PIXEL\_SOLID, FG\_BLACK);

matrix4x4 matRotZ, matRotX, matRotY;

vector<triangle> trisort;

vector<rectangle> rectsort;

fTheta += 0.5f \* fElapsedTime;

fThetaX = 5.0;

matRotY.m[0][0] = cosf(fTheta);

matRotY.m[0][2] = -sinf(fTheta);

matRotY.m[1][1] = 1;

matRotY.m[2][0] = sinf(fTheta);

matRotY.m[2][2] = cosf(fTheta);

matRotY.m[3][3] = 1;

matRotX.m[0][0] = 1;

matRotX.m[1][1] = cosf(fThetaX \* 0.5f);

matRotX.m[1][2] = sinf(fThetaX \* 0.5f);

matRotX.m[2][1] = -sinf(fThetaX \* 0.5f);

matRotX.m[2][2] = cosf(fThetaX \* 0.5f);

matRotX.m[3][3] = 1;

for (auto rect : Mill.rect)

{

rectangle rectproj, recttrans, rectrotZ, rectrotY, rectrotX,

rectrotYX, rectrotZX, rectrotZY;

VectorXMatrix(rect.point[0], rectrotY.point[0], matRotY);

VectorXMatrix(rect.point[1], rectrotY.point[1], matRotY);

VectorXMatrix(rect.point[2], rectrotY.point[2], matRotY);

VectorXMatrix(rect.point[3], rectrotY.point[3], matRotY);

VectorXMatrix(rectrotY.point[0], rectrotYX.point[0], matRotX);

VectorXMatrix(rectrotY.point[1], rectrotYX.point[1], matRotX);

VectorXMatrix(rectrotY.point[2], rectrotYX.point[2], matRotX);

VectorXMatrix(rectrotY.point[3], rectrotYX.point[3], matRotX);

recttrans = rectrotYX;

recttrans.point[0].z = rectrotYX.point[0].z + 100.0f;

recttrans.point[1].z = rectrotYX.point[1].z + 100.0f;

recttrans.point[2].z = rectrotYX.point[2].z + 100.0f;

recttrans.point[3].z = rectrotYX.point[3].z + 100.0f;

point3d normal;

normal.x = (recttrans.point[2].y - recttrans.point[1].y) \* (recttrans.point[0].z - recttrans.point[1].z) - (recttrans.point[2].z - recttrans.point[1].z) \* (recttrans.point[0].y - recttrans.point[1].y);

normal.y = (recttrans.point[2].z - recttrans.point[1].z) \* (recttrans.point[0].x - recttrans.point[1].x) - (recttrans.point[2].x - recttrans.point[1].x) \* (recttrans.point[0].z - recttrans.point[1].z);

normal.z = (recttrans.point[2].x - recttrans.point[1].x) \* (recttrans.point[0].y - recttrans.point[1].y) - (recttrans.point[2].y - recttrans.point[1].y) \* (recttrans.point[0].x - recttrans.point[1].x);

float l = sqrtf(normal.x \* normal.x + normal.y \* normal.y + normal.z \* normal.z);

normal.x /= l; normal.y /= l; normal.z /= l;

if (normal.x \* (recttrans.point[0].x - pos.x) +

normal.y \* (recttrans.point[0].y - pos.y) +

normal.z \* (recttrans.point[0].z - pos.z) < 0.0f)

{

point3d light\_direction = { 0.0f, 0.0f, -1.0f };

float l = sqrtf(light\_direction.x \* light\_direction.x + light\_direction.y \* light\_direction.y + light\_direction.z \* light\_direction.z);

light\_direction.x /= l; light\_direction.y /= l; light\_direction.z /= l;

float dp = normal.x \* light\_direction.x + normal.y \* light\_direction.y + normal.z \* light\_direction.z;

CHAR\_INFO c = GetColour(dp);

recttrans.col = c.Attributes;

recttrans.sym = c.Char.UnicodeChar;

VectorXMatrix(recttrans.point[0], rectproj.point[0], matproj);

VectorXMatrix(recttrans.point[1], rectproj.point[1], matproj);

VectorXMatrix(recttrans.point[2], rectproj.point[2], matproj);

VectorXMatrix(recttrans.point[3], rectproj.point[3], matproj);

rectproj.col = recttrans.col;

rectproj.sym = recttrans.sym;

rectproj.point[0].x += 1.0f; rectproj.point[0].y += 1.0f;

rectproj.point[1].x += 1.0f; rectproj.point[1].y += 1.0f;

rectproj.point[2].x += 1.0f; rectproj.point[2].y += 1.0f;

rectproj.point[3].x += 1.0f; rectproj.point[3].y += 1.0f;

rectproj.point[0].x \*= 0.5f \* (float)ScreenWidth();

rectproj.point[0].y \*= 0.5f \* (float)ScreenHeight();

rectproj.point[1].x \*= 0.5f \* (float)ScreenWidth();

rectproj.point[1].y \*= 0.5f \* (float)ScreenHeight();

rectproj.point[2].x \*= 0.5f \* (float)ScreenWidth();

rectproj.point[2].y \*= 0.5f \* (float)ScreenHeight();

rectproj.point[3].x \*= 0.5f \* (float)ScreenWidth();

rectproj.point[3].y \*= 0.5f \* (float)ScreenHeight();

rectsort.push\_back(rectproj);

}

}

for (auto tri : Mill.tri)

{

triangle triproj, tritrans, trirotZ, trirotY, trirotX,

trirotYX, trirotZX, trirotZY;

VectorXMatrix(tri.point[0], trirotY.point[0], matRotY);

VectorXMatrix(tri.point[1], trirotY.point[1], matRotY);

VectorXMatrix(tri.point[2], trirotY.point[2], matRotY);

VectorXMatrix(trirotY.point[0], trirotYX.point[0], matRotX);

VectorXMatrix(trirotY.point[1], trirotYX.point[1], matRotX);

VectorXMatrix(trirotY.point[2], trirotYX.point[2], matRotX);

tritrans = trirotYX;

tritrans.point[0].z = trirotYX.point[0].z + 10.0f;

tritrans.point[1].z = trirotYX.point[1].z + 10.0f;

tritrans.point[2].z = trirotYX.point[2].z + 10.0f;

point3d normal, line1, line2;

line1.x = tritrans.point[1].x - tritrans.point[0].x;

line1.y = tritrans.point[1].y - tritrans.point[0].y;

line1.z = tritrans.point[1].z - tritrans.point[0].z;

line2.x = tritrans.point[2].x - tritrans.point[0].x;

line2.y = tritrans.point[2].y - tritrans.point[0].y;

line2.z = tritrans.point[2].z - tritrans.point[0].z;

normal.x = line1.y \* line2.z - line1.z \* line2.y;

normal.y = line1.z \* line2.x - line1.x \* line2.z;

normal.z = line1.x \* line2.y - line1.y \* line2.x;

float l = sqrtf(normal.x \* normal.x + normal.y \* normal.y + normal.z \* normal.z);

normal.x /= l; normal.y /= l; normal.z /= l;

if (normal.x \* (tritrans.point[0].x - pos.x) +

normal.y \* (tritrans.point[0].y - pos.y) +

normal.z \* (tritrans.point[0].z - pos.z) < 0.0f)

{

point3d light\_direction = { 0.0f, 0.0f, -1.0f };

float l = sqrtf(light\_direction.x \* light\_direction.x + light\_direction.y \* light\_direction.y + light\_direction.z \* light\_direction.z);

light\_direction.x /= l; light\_direction.y /= l; light\_direction.z /= l;

float dp = normal.x \* light\_direction.x + normal.y \* light\_direction.y + normal.z \* light\_direction.z;

CHAR\_INFO c = GetColour(dp);

tritrans.col = c.Attributes;

tritrans.sym = c.Char.UnicodeChar;

VectorXMatrix(tritrans.point[0], triproj.point[0], matproj);

VectorXMatrix(tritrans.point[1], triproj.point[1], matproj);

VectorXMatrix(tritrans.point[2], triproj.point[2], matproj);

triproj.col = tritrans.col;

triproj.sym = tritrans.sym;

triproj.point[0].x += 1.0f; triproj.point[0].y += 1.0f;

triproj.point[1].x += 1.0f; triproj.point[1].y += 1.0f;

triproj.point[2].x += 1.0f; triproj.point[2].y += 1.0f;

triproj.point[0].x \*= 0.5f \* (float)ScreenWidth();

triproj.point[0].y \*= 0.5f \* (float)ScreenHeight();

triproj.point[1].x \*= 0.5f \* (float)ScreenWidth();

triproj.point[1].y \*= 0.5f \* (float)ScreenHeight();

triproj.point[2].x \*= 0.5f \* (float)ScreenWidth();

triproj.point[2].y \*= 0.5f \* (float)ScreenHeight();

trisort.push\_back(triproj);

}

}

// z buf

sort(trisort.begin(), trisort.end(), [](triangle& tri1, triangle& tri2)

{

float z1 = (tri1.point[0].z + tri1.point[1].z + tri1.point[2].z) / 3.0f;

float z2 = (tri2.point[0].z + tri2.point[1].z + tri2.point[2].z) / 3.0f;

return z1 > z2;

});

for (auto& triproj : trisort)

{

FillTriangle(triproj.point[0].x, triproj.point[0].y,

triproj.point[1].x, triproj.point[1].y,

triproj.point[2].x, triproj.point[2].y,

triproj.sym, triproj.col);

}

sort(rectsort.begin(), rectsort.end(), [](rectangle& rect1, rectangle& rect2)

{

float z1 = (rect1.point[0].z + rect1.point[1].z + rect1.point[2].z + rect1.point[3].z) / 4.0f;

float z2 = (rect2.point[0].z + rect2.point[1].z + rect2.point[2].z + rect2.point[3].z) / 4.0f;

return z1 > z2;

});

for (auto& rectproj : rectsort)

{

FillTriangle(rectproj.point[0].x, rectproj.point[0].y,

rectproj.point[1].x, rectproj.point[1].y,

rectproj.point[2].x, rectproj.point[2].y,

rectproj.sym, rectproj.col);

FillTriangle(rectproj.point[3].x, rectproj.point[3].y,

rectproj.point[0].x, rectproj.point[0].y,

rectproj.point[2].x, rectproj.point[2].y,

rectproj.sym, rectproj.col);

}

return true;

}

};

int main()

{

d3 millproj;

if (millproj.ConstructConsole(512, 480, 2, 2))

millproj.Start();

return 0;

}