МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»



**ОТЧЁТ**

**по лабораторной работе №2**

**«*Построение многогранников»***

по дисциплине:

**«*Графические системы*»**

|  |  |
| --- | --- |
| Выполнил: *Астафьев В.Е.*  Группа  *АО-71* | Проверил:  Кухто А.В. |

Новосибирск

2018

**Цель работы:**

Изучить построение правильных многогранников с помощью параллельной или точечной проекции.

**Ход работы:**

1. Смоделировать и описать в программе полигональную модель правильных выпуклых многогранников: октаэдра, гексаэдра, тетраэдра, додекаэдра и икосаэдра.
2. Реализовать возможность выбора пользователем вышеперечисленных фигур в программе.
3. Вывести на экран проекцию многогранников с возможностью выбора отображения параллельной или точечной (центральной) проекции на экран.

**Программный код**:

**figures.h:**

#pragma once

#ifndef FIGURES\_H

#define FIGURES\_H

#include "windraw.h"

void init\_figures(figure\*\*figures, LPCTSTR\*\*names, int\*amount);

#endif

**vec3.h:**

#pragma once

#ifndef VEC3\_H

#define VEC3\_H

// Structure for storing point in space

typedef struct{

float x, y, z;

} vec3;

// Creates vec3

vec3 make\_vec(float x, float y, float z);

float dot\_product(vec3 a, vec3 b);

vec3 cross\_product(vec3 a, vec3 b);

// Multiply vector by matrix 3\*3

vec3 vec\_multmat3(vec3 v, float\*mat);

// Applies rotation to vector

vec3 vec\_rotate(vec3 v, vec3 rotation);

// Apply orthographic projection on vector

vec3 project\_ortho(vec3 v);

// Apply perspective projection on vector

vec3 project\_persp(vec3 v, float rx, float ry, float sx, float sy, float cam\_z);

#endif

**windraw.h:**

#pragma once

#ifndef WINDRAW\_H

#define WINDRAW\_H

#define WINDRAW\_SIZE 400

#define PROJECTION\_ORTHO 1

#define PROJECTION\_PERSPECTIVE 2

#include <Windows.h>

#include "vec3.h"

// Callback for handling WM\_PAINT event

typedef void(CALLBACK\*paint\_callback)(HWND, HDC);

// Callback for handling WM\_LBUTTONDOWN event

typedef void(CALLBACK\*click\_callback)(HWND, POINT);

// Callback for handling WM\_DESTROY event

typedef void(CALLBACK\*destroy\_callback)(HWND);

// Callback for handling WM\_COMMAND event

typedef void(CALLBACK\*command\_callback)(HWND, WORD, WORD, HWND);

// Structure for storing color in RGB [0,1] space

typedef struct {

float r, g, b;

} rgb;

// Structure for storing triangle in space

typedef struct {

vec3 vertices[3];

} face;

// Structure for storing figure with set amount of faces

typedef struct {

face\*faces;

int amount;

} figure;

// Structure for storing callbacks to use in graphics window events

typedef struct {

paint\_callback cb\_paint;

click\_callback cb\_click;

destroy\_callback cb\_destroy;

command\_callback cb\_command;

} callbacks;

// Creates rgb

rgb make\_rgb(float r, float g, float b);

// Creates face

face make\_face(vec3 a, vec3 b, vec3 c);

// Creates figure (faces are copied)

figure make\_fig(int amount, face\*faces);

// Creates figure from triangles list (count - amount of faces)

figure fig\_trilist(int count, ...);

// Creates figure from triangles strip (count - amount of faces)

figure fig\_tristrip(int count, ...);

// Creates figure from indexed triangles list (count - amount of faces)

figure fig\_trilist\_index(int count, vec3\*vertices, int\*indices);

// Creates figure from indexed triangles strip (count - amount of faces)

figure fig\_tristrip\_index(int count, vec3\*vertices, int\*indices);

// Initializes graphics window with specified callbacks

HWND init\_window(callbacks cb);

// Runs window event pump

WPARAM window\_loop(HWND hWnd);

// Creates HPEN from specified color and style (PS\_\* flags)

HPEN create\_pen(rgb color, int style);

// Creates HBRUSH from specified color

HBRUSH create\_brush(rgb color);

// Draws line between points using specified pen

void draw\_line(HDC hdc, vec3 from, vec3 to, HPEN pen);

// Draws face using specified pen and brush

void draw\_face(HDC hdc, face face, vec3 rotation, DWORD projection, HPEN pen, HBRUSH brush);

// Draws face wireframe using specified pen

void draw\_face\_wire(HDC hdc, face face, vec3 rotation, DWORD projection, HPEN pen);

// Draws figure using specified pen and brush

void draw\_figure(HDC hdc, figure fig, vec3 rotation, DWORD projection, HPEN pen, HBRUSH brush);

// Draws figure wireframe using specified pen

void draw\_figure\_wire(HDC hdc, figure fig, vec3 rotation, DWORD projection, HPEN pen);

// Fills canvas with default window color

void draw\_clear(HDC hdc);

// Draws text on specified point

void draw\_text(HDC hdc, float x, float y, LPCTSTR text);

#endif

**wintimer.h:**

#pragma once

#ifndef WINTIMER\_H

#define WINTIMER\_H

#include <Windows.h>

typedef void(CALLBACK\*timer\_callback)(LPVOID);

typedef struct WINTIMER WINTIMER,\*LPWINTIMER;

LPWINTIMER timer\_start(DWORD interval, timer\_callback callback, LPVOID lParam);

void timer\_stop(LPWINTIMER timer);

#endif

**figures.c:**

#include "figures.h"

#include <tchar.h>

void init\_figures(figure\*\*figures, LPCTSTR\*\*names, int\*amount){

\*amount = 5;

\*figures = malloc(sizeof(figure)\*(\*amount));

\*names = malloc(sizeof(LPCTSTR)\*(\*amount));

// Тетраэдр

int indices0[4 \* 3] = {

2,1,3,

1,2,0,

1,0,3,

3,0,2

};

// Куб

int indices1[12 \* 3] = {

7,4,3,

3,4,0,

7,3,6,

3,2,6,

6,2,5,

2,1,5,

4,5,0,

1,0,5,

6,5,7,

5,4,7,

1,3,0,

2,3,1

};

// Октаэдр

int indices2[8 \* 3] = {

3,5,0,

1,5,3,

1,3,4,

3,0,4,

2,4,0,

4,2,1,

5,1,2,

2,0,5

};

// Додекаэдр

int indices3[36 \* 3] = {

16,12,19,

16,1,12,

19,12,0,

12,1,8,

12,8,14,

8,3,14,

17,14,3,

17,18,14,

18,2,14,

14,2,9,

12,14,9,

9,0,12,

19,0,4,

9,11,4,

9,4,0,

18,6,2,

9,2,6,

9,6,11,

8,7,3,

10,7,8,

17,3,7,

16,5,1,

8,1,5,

8,5,10,

10,5,15,

13,10,15,

10,13,7,

17,7,13,

17,13,18,

18,13,6,

11,6,13,

13,15,11,

11,15,4,

19,4,15,

16,19,15,

16,15,5

};

// Икосаэдр

int indices4[20 \* 3] = {

10,2,1,

1,2,5,

2,10,6,

6,8,2,

5,2,8,

8,11,5,

0,8,6,

8,0,11,

5,4,1,

4,7,1,

1,7,10,

9,10,7,

10,9,6,

5,11,4,

3,7,4,

9,7,3,

4,11,3,

0,3,11,

3,0,9,

0,6,9

};

vec3 vertices0[4] = {

make\_vec(+0.000000,-0.500000,-1.000000),

make\_vec(-0.866030,-0.500000,+0.500000),

make\_vec(+0.000000,+1.000000,+0.000000),

make\_vec(+0.866030,-0.500000,+0.500000)

};

vec3 vertices1[8] = {

make\_vec(+0.707110,-0.707110,-0.707110),

make\_vec(+0.707110,-0.707110,+0.707110),

make\_vec(-0.707110,-0.707110,+0.707110),

make\_vec(-0.707110,-0.707110,-0.707110),

make\_vec(+0.707110,+0.707110,-0.707110),

make\_vec(+0.707110,+0.707110,+0.707110),

make\_vec(-0.707110,+0.707110,+0.707110),

make\_vec(-0.707110,+0.707110,-0.707110)

};

vec3 vertices2[6] = {

make\_vec(-0.707110,+0.000000,-0.707110),

make\_vec(+0.707110,+0.000000,+0.707110),

make\_vec(-0.707110,+0.000000,+0.707110),

make\_vec(+0.707110,+0.000000,-0.707110),

make\_vec(+0.000000,+1.000000,+0.000000),

make\_vec(+0.000000,-1.000000,+0.000000)

};

vec3 vertices3[20] = {

make\_vec(+0.577350,+0.577350,+0.577350),

make\_vec(+0.577350,+0.577350,-0.577350),

make\_vec(-0.577350,+0.577350,+0.577350),

make\_vec(-0.577350,+0.577350,-0.577350),

make\_vec(+0.577350,-0.577350,+0.577350),

make\_vec(+0.577350,-0.577350,-0.577350),

make\_vec(-0.577350,-0.577350,+0.577350),

make\_vec(-0.577350,-0.577350,-0.577350),

make\_vec(+0.000000,+0.356820,-0.934170),

make\_vec(+0.000000,+0.356820,+0.934170),

make\_vec(+0.000000,-0.356820,-0.934170),

make\_vec(+0.000000,-0.356820,+0.934170),

make\_vec(+0.356820,+0.934170,+0.000000),

make\_vec(-0.356820,-0.934170,+0.000000),

make\_vec(-0.356820,+0.934170,+0.000000),

make\_vec(+0.356820,-0.934170,+0.000000),

make\_vec(+0.934170,+0.000000,-0.356820),

make\_vec(-0.934170,+0.000000,-0.356820),

make\_vec(-0.934170,+0.000000,+0.356820),

make\_vec(+0.934170,+0.000000,+0.356820)

};

vec3 vertices4[12] = {

make\_vec(+0.527230,-0.850650,-0.000000),

make\_vec(-0.527230,+0.850650,+0.000000),

make\_vec(+0.527230,+0.850650,+0.000000),

make\_vec(-0.527230,-0.850650,+0.000000),

make\_vec(-0.850650,-0.000000,-0.527230),

make\_vec(+0.000000,+0.527230,-0.850650),

make\_vec(+0.850650,+0.000000,+0.527230),

make\_vec(-0.850650,-0.000000,+0.527230),

make\_vec(+0.850650,+0.000000,-0.527230),

make\_vec(+0.000000,-0.527230,+0.850650),

make\_vec(+0.000000,+0.527230,+0.850650),

make\_vec(+0.000000,-0.527230,-0.850650)

};

(\*figures)[0] = fig\_trilist\_index(4, vertices0, indices0);

(\*figures)[1] = fig\_trilist\_index(12, vertices1, indices1);

(\*figures)[2] = fig\_trilist\_index(8, vertices2, indices2);

(\*figures)[3] = fig\_trilist\_index(36, vertices3, indices3);

(\*figures)[4] = fig\_trilist\_index(20, vertices4, indices4);

(\*names)[0] = \_T("Тетраэдр");

(\*names)[1] = \_T("Гексаэдр");

(\*names)[2] = \_T("Октаэдр");

(\*names)[3] = \_T("Додекаэдр");

(\*names)[4] = \_T("Икосаэдр");

}

**main.c:**

#include "windraw.h"

#include "wintimer.h"

#include "figures.h"

#include <tchar.h>

#include <math.h>

#include <stdio.h>

#define \_tputs(str) \_tprintf(\_T("%s\n"), str);

// Makes sure angle is in [0,360] interval

float loop\_angle(float a){

while(a < 0)

a = 360.0 + a;

return fmod(a, 360);

}

// Pen for faces' borders

HPEN pen;

// Brush to fill faces

HBRUSH brush;

// Array of predefined figures

figure\*figures;

// Array of names of predefined figures

LPCTSTR\*fig\_names;

// Amount of predefined figures

int fig\_count;

// Index of currently selected predefined figure

int fig\_sel = 0;

// Array of buttons

HWND\*btn\_figures;

HWND buttons[9];

// Animation timer

LPWINTIMER timer;

// Rotation angle

vec3 rotation;

// Projection mode

DWORD projection\_mode = PROJECTION\_PERSPECTIVE;

// Temporary buffer for rotation angle string

LPTSTR rotation\_string;

// If true, object will be rotating on its own

BOOL animation = TRUE;

// If true, objects' faces will be visible

BOOL fill = TRUE;

// Callback for WM\_PAINT event

void CALLBACK on\_paint(HWND hWnd, HDC hdc){

draw\_clear(hdc); // Clear canvas

if(fill) // Draw selected figure

draw\_figure(hdc, figures[fig\_sel], rotation, projection\_mode, pen, brush);

else

draw\_figure\_wire(hdc, figures[fig\_sel], rotation, projection\_mode, pen);

SelectObject(hdc, GetStockObject(SYSTEM\_FIXED\_FONT));

SetTextAlign(hdc, TA\_BOTTOM | TA\_CENTER);

draw\_text(hdc, +0.0f, +1.0f, fig\_names[fig\_sel]); // Write name of selected figure

SetTextAlign(hdc, TA\_BOTTOM | TA\_LEFT);

draw\_text(hdc, -1.0f, +1.0f, rotation\_string); // Write name of selected figure

SetTextAlign(hdc, TA\_BOTTOM | TA\_RIGHT);

draw\_text(hdc, +1.0f, +1.0f, projection\_mode == PROJECTION\_ORTHO ? \_T("Параллельная") : \_T("Центральная")); // Write name of selected figure

}

// Callback for WM\_LBUTTONDOWN event

void CALLBACK on\_click(HWND hWnd, POINT pt){

if(++fig\_sel == fig\_count) // Loop selection

fig\_sel = 0;

RedrawWindow(hWnd, NULL, NULL, RDW\_INVALIDATE); // Call WM\_PAINT

}

// Callback for WM\_DESTROY event

void CALLBACK on\_destroy(HWND hWnd){

}

// Callback for WM\_COMMAND event

void CALLBACK on\_command(HWND hWnd, WORD wControl, WORD wCode, HWND hControl){

if(wCode == BN\_CLICKED){

for(int i = 0; i < fig\_count; ++i)

if(hControl == btn\_figures[i]){

fig\_sel = i;

RedrawWindow(hWnd, NULL, NULL, RDW\_INVALIDATE); // Call WM\_PAINT

return;

}

for(int i = 0; i < 10; ++i)

if(hControl == buttons[i]){

if(i == 0) rotation = make\_vec(0, 0, 320);

if(i == 1) rotation.x = loop\_angle(rotation.x + 10);

if(i == 2) rotation.x = loop\_angle(rotation.x - 10);

if(i == 3) rotation.y = loop\_angle(rotation.y + 10);

if(i == 4) rotation.y = loop\_angle(rotation.y - 10);

if(i == 5) rotation.z = loop\_angle(rotation.z + 10);

if(i == 6) rotation.z = loop\_angle(rotation.z - 10);

if(i == 7) animation = animation ? FALSE : TRUE;

if(i == 8) fill = fill ? FALSE : TRUE;

if(i == 9) projection\_mode = projection\_mode == PROJECTION\_ORTHO ? PROJECTION\_PERSPECTIVE : PROJECTION\_ORTHO;

RedrawWindow(hWnd, NULL, NULL, RDW\_INVALIDATE); // Call WM\_PAINT

return;

}

}

}

void CALLBACK on\_timer(LPARAM lParam){

HWND hWnd = lParam;

if(animation)

rotation.y = loop\_angle(rotation.y - 2);

\_stprintf\_s(rotation\_string, 32, \_T("%3.0f, %3.0f, %3.0f"), rotation.x, rotation.y, rotation.z);

InvalidateRect(hWnd, NULL, FALSE);

}

void init\_buttons(HWND window){

HINSTANCE hInstance = GetWindowLongPtr(window, GWLP\_HINSTANCE);

btn\_figures = malloc(sizeof(HWND)\*fig\_count);

for(int i = 0; i < fig\_count; ++i)

btn\_figures[i] = CreateWindow(

\_T("Button"),

fig\_names[i],

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON,

WINDRAW\_SIZE - 100, i \* 22,

100, 22,

window, NULL,

hInstance, NULL

);

buttons[0] = CreateWindow(

\_T("Button"),

\_T("Сброс"),

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON,

WINDRAW\_SIZE - 100, (fig\_count + 1) \* 22,

100, 22,

window, NULL,

hInstance, NULL

);

buttons[1] = CreateWindow(

\_T("Button"),

\_T("+X"),

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON,

WINDRAW\_SIZE - 100, (fig\_count + 2) \* 22,

50, 22,

window, NULL,

hInstance, NULL

);

buttons[2] = CreateWindow(

\_T("Button"),

\_T("-X"),

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON,

WINDRAW\_SIZE - 50, (fig\_count + 2) \* 22,

50, 22,

window, NULL,

hInstance, NULL

);

buttons[3] = CreateWindow(

\_T("Button"),

\_T("+Y"),

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON,

WINDRAW\_SIZE - 100, (fig\_count + 3) \* 22,

50, 22,

window, NULL,

hInstance, NULL

);

buttons[4] = CreateWindow(

\_T("Button"),

\_T("-Y"),

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON,

WINDRAW\_SIZE - 50, (fig\_count + 3) \* 22,

50, 22,

window, NULL,

hInstance, NULL

);

buttons[5] = CreateWindow(

\_T("Button"),

\_T("+Z"),

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON,

WINDRAW\_SIZE - 100, (fig\_count + 4) \* 22,

50, 22,

window, NULL,

hInstance, NULL

);

buttons[6] = CreateWindow(

\_T("Button"),

\_T("-Z"),

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON,

WINDRAW\_SIZE - 50, (fig\_count + 4) \* 22,

50, 22,

window, NULL,

hInstance, NULL

);

buttons[7] = CreateWindow(

\_T("Button"),

\_T("Анимация"),

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON,

WINDRAW\_SIZE - 100, (fig\_count + 6) \* 22,

100, 22,

window, NULL,

hInstance, NULL

);

buttons[8] = CreateWindow(

\_T("Button"),

\_T("Заливка"),

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON,

WINDRAW\_SIZE - 100, (fig\_count + 7) \* 22,

100, 22,

window, NULL,

hInstance, NULL

);

buttons[9] = CreateWindow(

\_T("Button"),

\_T("Проекция"),

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON,

WINDRAW\_SIZE - 100, (fig\_count + 8) \* 22,

100, 22,

window, NULL,

hInstance, NULL

);

}

void hide\_window(HWND hWnd){

LONG style = GetWindowLongPtr(hWnd, GWL\_STYLE);

style &= ~(WS\_VISIBLE);

style |= WS\_EX\_TOOLWINDOW;

style &= ~WS\_EX\_APPWINDOW;

ShowWindow(hWnd, SW\_HIDE);

SetWindowLongPtr(hWnd, GWL\_STYLE, style);

ShowWindow(hWnd, SW\_SHOW);

ShowWindow(hWnd, SW\_HIDE);

}

int main(int argc, char\*\*argv){

rotation = make\_vec(0, 0, 320);

rotation\_string = calloc(32, sizeof(TCHAR));

init\_figures(&figures, &fig\_names, &fig\_count); // Initialize figures

// Transparent pen to hide faces' borders

//pen = create\_pen(make\_rgb(0, 0, 0), PS\_NULL);

pen = create\_pen(make\_rgb(0.0f, 0.0f, 0.0f), PS\_SOLID);

// Brush to fill faces (#808080)

brush = create\_brush(make\_rgb(0.5f, 0.5f, 0.5f));

callbacks cb; // Setup callbacks

cb.cb\_paint = on\_paint;

cb.cb\_click = on\_click;

cb.cb\_destroy = on\_destroy;

cb.cb\_command = on\_command;

HWND hWnd = init\_window(cb); // Init graphics window

init\_buttons(hWnd); // Initialize buttons

timer\_start(100, on\_timer, hWnd);

UpdateWindow(hWnd);

hide\_window(GetConsoleWindow());

window\_loop(hWnd); // Start window event pump

timer\_stop(timer);

DeleteObject(pen); // Destroy pen and brush

DeleteObject(brush);

// free(btn\_figures);

for(int i = 0; i < fig\_count; ++i)

free(figures[i].faces);

free(figures);

free(fig\_names);

free(rotation\_string);

\_tputs(\_T("end"));

}

**vec3.c:**

#include "vec3.h"

#include <math.h>

// Creates vec3 struct

vec3 make\_vec(float x, float y, float z){

vec3 ret;

ret.x = x;

ret.y = y;

ret.z = z;

return ret;

}

float dot\_product(vec3 a, vec3 b){

return a.x\*b.x + a.y\*b.y + a.z\*b.z;

}

vec3 cross\_product(vec3 a, vec3 b){

return make\_vec(

a.y\*b.z - a.z\*b.y,

a.z\*b.x - a.x\*b.z,

a.x\*b.y - a.y\*b.x

);

}

// Multiply vector by matrix 3\*3

vec3 vec\_multmat3(vec3 v, float\*mat){

return make\_vec(

v.x\*mat[0] + v.y\*mat[1] + v.z\*mat[2],

v.x\*mat[3] + v.y\*mat[4] + v.z\*mat[5],

v.x\*mat[6] + v.y\*mat[7] + v.z\*mat[8]

);

}

// Applies rotation to vector

vec3 vec\_rotate(vec3 v, vec3 rotation){

const float deg2rad = 3.1415f / 180.0f;

vec3 rad = make\_vec(rotation.x\*deg2rad, rotation.y\*deg2rad, rotation.z\*deg2rad);

float matrix[9] = {

cos(rad.x)\*cos(rad.y) - sin(rad.x)\*sin(rad.z)\*sin(rad.y), -sin(rad.x)\*cos(rad.z), -cos(rad.x)\*sin(rad.y) - sin(rad.x)\*sin(rad.z)\*cos(rad.y),

cos(rad.x)\*sin(rad.z)\*sin(rad.y) + sin(rad.x)\*cos(rad.y), cos(rad.x)\*cos(rad.z), cos(rad.x)\*sin(rad.z)\*cos(rad.y) - sin(rad.x)\*sin(rad.y),

cos(rad.z)\*sin(rad.y), -sin(rad.z), cos(rad.z)\*cos(rad.y)

};

return vec\_multmat3(v, matrix);

}

// Apply orthographic projection on vector

vec3 project\_ortho(vec3 v){

return make\_vec(v.x, -v.y, 0);

}

// Apply perspective projection on vector

vec3 project\_persp(vec3 v, float rx, float ry, float sx, float sy, float cam\_z){

vec3 r = make\_vec(rx, ry, 1);

return make\_vec(

(v.x\*sx) / ((v.z - cam\_z) \* r.x) \* r.z,

(v.y\*sy) / ((v.z - cam\_z) \* r.y) \* r.z,

0

);

}

**wintimer.c:**

#include "wintimer.h"

struct WINTIMER {

timer\_callback callback;

LPVOID param;

DWORD interval;

BOOL run;

};

DWORD WINAPI timer\_func(LPWINTIMER timer) {

while (timer->run) {

timer->callback(timer->param);

Sleep(timer->interval);

}

HeapFree(GetProcessHeap(), 0, timer);

return 0;

}

LPWINTIMER timer\_start(DWORD interval, timer\_callback callback, LPVOID lParam) {

LPWINTIMER timer = HeapAlloc(GetProcessHeap(), HEAP\_ZERO\_MEMORY, sizeof(WINTIMER));

timer->interval = interval;

timer->callback = callback;

timer->param = lParam;

timer->run = TRUE;

HANDLE h = CreateThread(

NULL, 0,

timer\_func,

timer,

0, NULL

);

if (h == NULL) {

HeapFree(GetProcessHeap(), 0, timer);

return NULL;

}

return timer;

}

void timer\_stop(LPWINTIMER timer) {

if(timer)

timer->run = FALSE;

}

**windraw.c:**

#include "windraw.h"

#include <windowsx.h>

#include <tchar.h>

#include <stdlib.h>

#include <string.h>

#include <stdarg.h>

#include <math.h>

typedef struct {

HWND hWnd;

callbacks cb;

HBITMAP buffer;

} window\_data;

// Returns TRUE if face should not be rendered

BOOL should\_cull(face f, DWORD projection) {

vec3 a = f.vertices[0];

vec3 b = f.vertices[1];

vec3 c = f.vertices[2];

vec3 norm = cross\_product(

make\_vec(b.x - a.x, b.y - a.y, b.z - a.z),

make\_vec(c.x - a.x, c.y - a.y, c.z - a.z)

);

if(projection == PROJECTION\_ORTHO)

return norm.z < 0;

return dot\_product(

make\_vec(a.x, a.y, a.z - 2),

norm

) >= 0;

}

// Translates world vec3 [-1,1] to window POINT [0,WINDRAW\_SIZE]

POINT translate\_vec(vec3 v) {

POINT pt;

pt.x = (v.x\*WINDRAW\_SIZE + WINDRAW\_SIZE) / 2;

pt.y = (v.y\*WINDRAW\_SIZE + WINDRAW\_SIZE) / 2;

return pt;

}

// Applies rotation to face

face face\_rotate(face f, vec3 rotation) {

return make\_face(

vec\_rotate(f.vertices[0], rotation),

vec\_rotate(f.vertices[1], rotation),

vec\_rotate(f.vertices[2], rotation)

);

}

// Calculates face's normal vector

vec3 face\_normal(face f, vec3 angle) {

vec3 a = vec\_rotate(f.vertices[0], angle);

vec3 b = vec\_rotate(f.vertices[1], angle);

vec3 c = vec\_rotate(f.vertices[2], angle);

vec3 norm = cross\_product(

make\_vec(b.x - a.x, b.y - a.y, b.z - a.z),

make\_vec(c.x - a.x, c.y - a.y, c.z - a.z)

);

return norm;

}

// Applies projection on vector

vec3 vec\_project(vec3 v, DWORD persp) {

if(persp == PROJECTION\_PERSPECTIVE)

return project\_persp(v, WINDRAW\_SIZE, WINDRAW\_SIZE, WINDRAW\_SIZE, WINDRAW\_SIZE, 2);

return project\_ortho(v);

}

// Graphics window procedure

LRESULT CALLBACK wnd\_proc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam) {

window\_data\*wd = GetWindowLongPtr(hWnd, GWLP\_USERDATA); // Extracting window\_data from window userdata

if (uMsg == WM\_CREATE) { // Window has been created

LPCREATESTRUCT cs = lParam;

RECT client;

SIZE window;

wd = cs->lpCreateParams;

SetWindowLongPtr(hWnd, GWLP\_USERDATA, cs->lpCreateParams); // Store window\_data to window userdata

GetClientRect(hWnd, &client);

window.cx = WINDRAW\_SIZE + WINDRAW\_SIZE - client.right + client.left;

window.cy = WINDRAW\_SIZE + WINDRAW\_SIZE - client.bottom + client.top; // Resize window to match client size with WINDRAW\_SIZE

SetWindowPos(hWnd, NULL, NULL, NULL, window.cx, window.cy, SWP\_NOZORDER | SWP\_NOMOVE);

}

// Window repaint

if (uMsg == WM\_PAINT && wd->cb.cb\_paint) {

PAINTSTRUCT ps;

BeginPaint(hWnd, &ps);

if (wd->buffer == NULL) // Create buffer if this is the first time we draw

wd->buffer = CreateCompatibleBitmap(ps.hdc, WINDRAW\_SIZE \* 2, WINDRAW\_SIZE \* 2);

HDC buf\_hdc = CreateCompatibleDC(ps.hdc);

SelectBitmap(buf\_hdc, wd->buffer);

wd->cb.cb\_paint(hWnd, buf\_hdc); // Call paint callback on buffer HDC

BitBlt(ps.hdc, 0, 0, WINDRAW\_SIZE, WINDRAW\_SIZE, buf\_hdc, 0, 0, SRCCOPY);

DeleteDC(buf\_hdc);

EndPaint(hWnd, &ps);

return NULL;

}

if (uMsg == WM\_LBUTTONDOWN && wd->cb.cb\_click) {

POINT pt;

pt.x = LOWORD(lParam);

pt.y = HIWORD(lParam);

wd->cb.cb\_click(hWnd, pt); // Call callback with mouse coordinates

}

if (uMsg == WM\_DESTROY) { // Window is being destroyed. Free resources

DeleteBitmap(wd->buffer);

free(wd);

SetWindowLongPtr(hWnd, GWLP\_USERDATA, NULL);

PostQuitMessage(0);

return 0;

}

if (uMsg == WM\_COMMAND && wd->cb.cb\_command) { // Received notification from child control

WORD wControl = LOWORD(wParam);

WORD wCode = HIWORD(wParam);

HWND hControl = lParam;

wd->cb.cb\_command(hWnd, wControl, wCode, hControl);

}

return DefWindowProc(hWnd, uMsg, wParam, lParam);

}

// Initialize graphics window with passed callbacks and show it

HWND init\_window(callbacks cb) {

// Get current HINSTANCE from console window

HINSTANCE hInstance = GetWindowLongPtr(GetConsoleWindow(), GWLP\_HINSTANCE);

LPCTSTR class\_name = \_T("class\_name"); // WNDCLASS.lpszClassName

WNDCLASS class;

window\_data\*wd = malloc(sizeof(window\_data)); // Store callbacks struct in heap

wd->cb = cb;

wd->buffer = NULL;

class.style = CS\_HREDRAW | CS\_VREDRAW;

class.lpfnWndProc = wnd\_proc;

class.cbClsExtra = 0;

class.cbWndExtra = 0;

class.hInstance = hInstance;

class.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_APPLICATION));

class.hCursor = LoadCursor(NULL, IDC\_ARROW);

class.hbrBackground = COLOR\_WINDOW + 1;

class.lpszMenuName = NULL;

class.lpszClassName = class\_name;

RegisterClass(&class);

HWND hWnd = CreateWindow(

class\_name,

\_T(""),

WS\_SYSMENU | WS\_CAPTION | WS\_VISIBLE,

CW\_USEDEFAULT, CW\_USEDEFAULT,

WINDRAW\_SIZE, WINDRAW\_SIZE,

NULL, NULL,

hInstance, wd

);

wd->hWnd = hWnd;

return hWnd;

}

// Graphics window event pump

WPARAM window\_loop(HWND hWnd) {

MSG msg;

while (GetMessage(&msg, NULL, 0, 0)) {

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return msg.wParam;

}

// Creates rgb struct

rgb make\_rgb(float r, float g, float b) {

rgb ret;

ret.r = r;

ret.g = g;

ret.b = b;

return ret;

}

// Creates face struct

face make\_face(vec3 a, vec3 b, vec3 c) {

face ret;

ret.vertices[0] = a;

ret.vertices[1] = b;

ret.vertices[2] = c;

return ret;

}

// Creates figure struct (faces are copied)

figure make\_fig(int amount, face\*faces) {

figure ret;

ret.amount = amount;

ret.faces = malloc(sizeof(face)\*amount);

memcpy(ret.faces, faces, sizeof(face)\*amount);

return ret;

}

// Creates figure struct from triangles list

figure fig\_trilist(int count, ...) {

va\_list ap;

int vc = count \* 3;

vec3\*verts = malloc(sizeof(vec3)\*vc);

va\_start(ap, count);

for (int i = 0; i < vc; ++i)

verts[i] = va\_arg(ap, vec3);

va\_end(ap);

face\*faces = malloc(sizeof(face)\*count);

for (int i = 0; i < count; ++i) {

faces[i].vertices[0] = verts[i \* 3 + 0];

faces[i].vertices[1] = verts[i \* 3 + 1];

faces[i].vertices[2] = verts[i \* 3 + 2];

}

figure fig;

fig.amount = count;

fig.faces = faces;

free(verts);

return fig;

}

// Creates figure struct from triangles strip

figure fig\_tristrip(int count, ...) {

va\_list ap;

int vc = count + 2;

vec3\*verts = malloc(sizeof(vec3)\*vc);

va\_start(ap, count);

for (int i = 0; i < vc; ++i)

verts[i] = va\_arg(ap, vec3);

va\_end(ap);

face\*faces = malloc(sizeof(face)\*count);

for (int f = 0; f < count; ++f) {

faces[f].vertices[0] = verts[f + 0];

faces[f].vertices[1] = verts[f + 1];

faces[f].vertices[2] = verts[f + 2];

}

figure fig;

fig.amount = count;

fig.faces = faces;

free(verts);

return fig;

}

// Creates figure from indexed triangles list (count - amount of faces)

figure fig\_trilist\_index(int count, vec3\*vertices, int\*indices){

figure fig;

fig.amount = count;

fig.faces = malloc(sizeof(face)\*count);

for(int i = 0; i < count; ++i)

fig.faces[i] = make\_face(

vertices[indices[i \* 3 + 0]],

vertices[indices[i \* 3 + 1]],

vertices[indices[i \* 3 + 2]]

);

return fig;

}

// Creates figure from indexed triangles strip (count - amount of faces)

figure fig\_tristrip\_index(int count, vec3\*vertices, int\*indices){

figure fig;

fig.amount = count;

fig.faces = malloc(sizeof(face)\*count);

for(int i = 0; i < count; ++i)

fig.faces[i] = make\_face(

vertices[indices[i + 0]],

vertices[indices[i + 1]],

vertices[indices[i + 2]]

);

return fig;

}

// Creates HPEN from rgb struct (style is PS\_\* flags)

HPEN create\_pen(rgb color, int style) {

return CreatePen(style, 1, RGB(color.r \* 256, color.g \* 256, color.b \* 256));

}

// Creates solid color HBRUSH from rgb struct

HBRUSH create\_brush(rgb color) {

return CreateSolidBrush(RGB(color.r \* 255, color.g \* 255, color.b \* 255));

}

// Draws line between point using specified pen

void draw\_line(HDC hdc, vec3 from, vec3 to, HPEN pen) {

if (pen) SelectPen(hdc, pen);

POINT pt1 = translate\_vec(from);

POINT pt2 = translate\_vec(to);

MoveToEx(hdc, pt1.x, pt1.y, NULL);

LineTo(hdc, pt2.x, pt2.y);

}

// Draws face using specified pen and brush

void draw\_face(HDC hdc, face face, vec3 rotation, DWORD projection, HPEN pen, HBRUSH brush) {

POINT points[3] = {

translate\_vec(vec\_project(vec\_rotate(face.vertices[0], rotation), projection)),

translate\_vec(vec\_project(vec\_rotate(face.vertices[1], rotation), projection)),

translate\_vec(vec\_project(vec\_rotate(face.vertices[2], rotation), projection))

};

if (pen) SelectPen(hdc, pen);

if (brush) SelectBrush(hdc, brush);

Polygon(hdc, points, 3);

}

// Draws face wireframe using specified pen

void draw\_face\_wire(HDC hdc, face face, vec3 rotation, DWORD projection, HPEN pen) {

POINT points[4] = {

translate\_vec(vec\_project(vec\_rotate(face.vertices[0], rotation), projection)),

translate\_vec(vec\_project(vec\_rotate(face.vertices[1], rotation), projection)),

translate\_vec(vec\_project(vec\_rotate(face.vertices[2], rotation), projection)),

translate\_vec(vec\_project(vec\_rotate(face.vertices[0], rotation), projection))

};

if (pen) SelectPen(hdc, pen);

Polyline(hdc, points, 4);

}

// Draws figure using specified pen and brush

void draw\_figure(HDC hdc, figure fig, vec3 rotation, DWORD projection, HPEN pen, HBRUSH brush) {

if (pen) SelectPen(hdc, pen);

if (brush) SelectBrush(hdc, brush);

for (int i = 0; i < fig.amount; ++i)

if(!should\_cull(face\_rotate(fig.faces[i], rotation), projection))

draw\_face(hdc, fig.faces[i], rotation, projection, NULL, NULL);

}

// Draws figure wireframe using specified pen

void draw\_figure\_wire(HDC hdc, figure fig, vec3 rotation, DWORD projection, HPEN pen) {

if (pen) SelectPen(hdc, pen);

for (int i = 0; i < fig.amount; ++i)

draw\_face\_wire(hdc, fig.faces[i], rotation, projection, NULL, NULL);

}

// Fills whole canvas with default window color (COLOR\_WINDOW)

void draw\_clear(HDC hdc) {

RECT rect;

rect.right = WINDRAW\_SIZE;

rect.bottom = WINDRAW\_SIZE;

rect.left = -WINDRAW\_SIZE;

rect.top = -WINDRAW\_SIZE;

FillRect(hdc, &rect, COLOR\_WINDOW + 1);

}

// Draws text at specified point

void draw\_text(HDC hdc, float x, float y, LPCTSTR text) {

POINT pt = translate\_vec(make\_vec(x, y, 0));

int len = \_tcslen(text);

TextOut(hdc, pt.x, pt.y, text, len);

}

**Результат работы программы**:





