|  |
| --- |
| Московский Энергетический Институт (ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ) |
| Лабораторная работа №5 |
| Удаление невидимых граней |
|  |
| **Студент Винников Алесандр группа А-14-07** |
| Преподаватель Князев Анатолий Васильевич |

|  |
| --- |
| Москва 2010 |

# Задание

Разработать программу визуализации трёхмерной сцены, реализующий метод Варнока (первый вариант) удаления невидимых поверхностей.

Определить время визуализации тестовых сцен

# Описание метода

Метод Варнока предназначен для визуализации сцен, заданных полигональными сетками. Метод работает в пространстве изображений.

Внимание уделяется тем частям сцены, которые наиболее насыщены объектами. Оцениваем, является ли содержимое окна простым для визуализации. Если окно простое, то производится отрисовка содержимого окна. Если нет, то окно разбивается на несколько подокон и к каждому применяется та же проверка.

В первой версии метода окно считается простым для визуализации, если оно пустое. Тогда оно закрашивается цветом фона. В противном случае оно разбивается до пикселей. Для определения, является ли окно пустым, окружим многоугольники прямоугольными объемлющими оболочками. Если объемлющие оболочки всех фигур находятся вне границ окна, то оно пустое.

При отрисовке отдельных пикселей нужно определить, лежит ли данная точка внутри многоугольника. Для этого посчитаем количество пересечений луча, исходящего из данной точки в направлении горизонтальной оси, со сторонами многоугольника. Если оно чётное, точка не принадлежит многоугольнику. Пусть луч направлен влево. Заметим, что такой способ работает для любых многоугольников, а не только для выпуклых. На рисунке луч пересекается с ребрами 3 раза – точка находится внутри фигуры.

# Алгоритм метода Варнока(первый вариант)

АЛГ окно\_пусто (левая, правая, верхняя, нижняя : граница окна)

НАЧ

ДЛЯ КАЖДОГО многоугольника

НАЧ

ЛОГИЧЕСКИЙ пересечение\_по\_х, пересечение\_по\_у;

ЕСЛИ(низ\_окна < низ\_оболочки)

пересеч\_по\_у = (верх\_окна > верх\_оболочки);

ИНАЧЕ

ЕСЛИ(верх\_окна > верх\_оболочки)

пересеч\_по\_у = (низ\_окна < верх\_оболочки);

ИНАЧЕ

пересеч\_по\_у = ИСТИНА;

ЕСЛИ(лев\_гр\_окна < лев\_гр\_оболочки)

пересеч\_по\_х = (прав\_гр\_окна > прав\_гр\_оболочки);

ИНАЧЕ

ЕСЛИ(прав\_гр\_окна > прав\_гр\_оболочки)

пересеч\_по\_x = (лев\_гр\_окна <прав\_гр\_оболочки);

ИНАЧЕ

пересеч\_по\_x = ИСТИНА;

ЕСЛИ (пересеч\_по\_x И пересеч\_по\_у) ВЕРНУТЬ ЛОЖЬ;

КОН

ВЕРНУТЬ ИСТИНА;

КОН

АЛГ отрисовка(левая, правая, верхняя, нижняя : граница окна)

НАЧ

ЕСЛИ (окно\_пусто())

НАЧ

Закрасить\_окно\_цветом\_фона();

КОН

ЕСЛИ (низ\_окна <= верх\_окна)

ЕСЛИ (лев\_гр\_окна >= прав\_гр\_окна)

НАЧ

ДЛЯ КАЖДОГО многоугольника

ЕСЛИ точка внутки многоугольника

найдём координату z и

ЕСЛИ она ближе остальных к наблюдателю

рассчитаем освещенность в этой точке и выведем её

КОН

ИНАЧЕ

НАЧ

отрисовка(лев\_гр, (лев\_гр+прав\_гр)/2, верх, низ);

отрисовка((лев\_гр+прав\_гр)/2, прав\_гр, верх, низ);

КОН

ИНАЧЕ

НАЧ

ЕСЛИ (лев\_гр\_окна >= прав\_гр\_окна)

НАЧ

отрисовка(лев\_гр, прав\_гр, верх, (верх+низ)/2);

отрисовка(лев\_гр, прав\_гр, (верх+низ)/2, низ);

КОН

НАЧ

отрисовка(лев\_гр, (лев\_гр+прав\_гр)/2, верх, (верх+низ)/2);

отрисовка(лев\_гр, (лев\_гр+прав\_гр)/2, (верх+низ)/2, низ);

отрисовка((лев\_гр+прав\_гр)/2, прав\_гр, верх, (верх+низ)/2);

отрисовка((лев\_гр+прав\_гр)/2, прав\_гр, (верх+низ)/2, низ);

КОН

КОН

КОН

# Описание структуры данных

Формат файла сцены:

1. Данные сцены
   1. координаты камеры в экранных пикселах (вещ x, y, z). Начало системы координат в центре экрана.
   2. поворот сцены вдоль оси y в радианах, считая от обратного напрвления оси z (вещ)
   3. поворот сцены вдоль оси z в радианах, считая от напрвления оси x (вещ)
   4. координаты источника света (вещ x,y,z)
2. Полигональная сетка - список вершин
   1. Количество вершин (цел n)
   2. Вершины, заданные вещественными координатами(вещ x0, y0, z0, x1, y1,… zn)
   3. Полигоны (P1, P2, P3, … до конца файла)
      1. количество вершин (цел mi)
      2. номера вершин(цел K1, K2, … Kmi). Нумерация начинается с 0
      3. цвет многоугольника(1, 2, или 3)

# Тесты и их результаты

1. Файл с данными сцены:

камера

0 0 -200

-0.18 -3.58

поворот сцены

0 0 -200

19

источник света

0 -100 0

50 0 50

-50 0 50

-50 0 -50

50 0 -50

60 -120 40

20 -120 40

60 -120 0

точки

20 -120 0

60 -80 40

20 -80 40

60 -80 0

20 -80 0

157.78 0 45.2

73.6 0 -32.5

80.3 0 14.5

157.78 -20 45.2

73.6 -20 -32.5

80.3 -20 14.5

3 0 1 2 1

3 0 2 3 1

3 0 3 4 1

3 0 4 1 1

4 1 2 3 4 1

4 5 6 8 7 2

4 9 10 12 11 2

4 5 6 10 9 2

полиномы

4 7 8 12 11 2

4 7 5 9 11 2

4 8 6 10 12 2

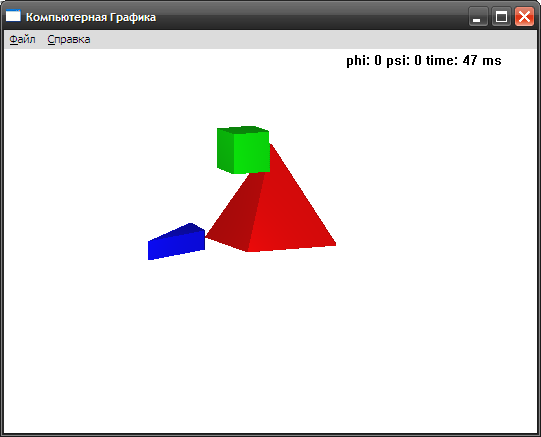
3 13 14 15 3

3 16 17 18 3

4 13 14 17 16 3

4 13 15 18 16 3

4 14 15 18 17 3



Сцена рисуется за 47 милисекунд. Это соответствует 21 кадрам в секунду

1. Зададим невыпуклый многоугольник:

0 0 -200 -0.18 -3.58

0 0 -200

5

-50 50 0

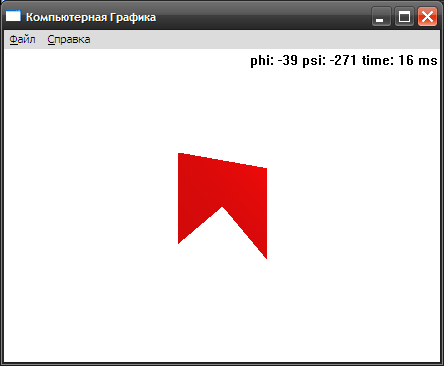
0 0 0

50 50 0

50 -50 0

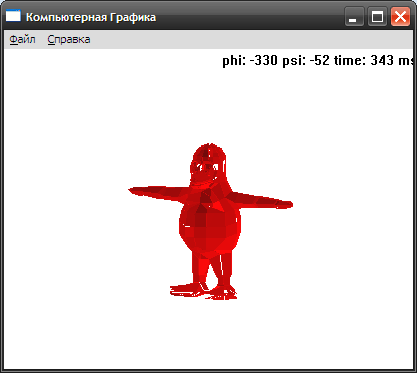
-50 -50 0

5 0 1 2 3 4 1



Многоугольник прорисовался за 16 милисекунд, 62,5 кадров в секунду.

Пингвин прорисовался за 343 мс (3 кадра в секунду):



В ходе тестирования я выяснил, что скорость прорисовки в большей степени зависит от размера изображаемых объектов, а также от количества точек и полигонов.

# Описнние функций

// Функция scene\_correct проставляет прямоугольные объемлющие оболочки проэкций фигур и осуществляет преобразование координат(поворот на углы phi и psi)

void scene\_correct(float phi, float psi)

// Функция point\_inside определяет, лежит ли точка внутри прямоугольника pol

/\* В основе алгоритма лежит идея подсчёта количества пересечений луча, исходящего из данной точки в направлении горизонтальной оси, со сторонами многоугольника. Если оно чётное, точка не принадлежит многоугольнику. В данном алгоритме луч направлен влево. \*/

bool point\_inside(int x, int y, int pol)

// Функция norma нормирует вектор

void norma(Scene::Mesh::Point& l)

// Функция draw\_point закрашивает точку экрана с координатами х, у цветом ближнего многоугольника

void draw\_point(HDC hdc, int x, int y)

// Функция is\_free возвращает, является ли прямоугольник r пустым.

bool is\_free(LPRECT r)

// Функция draw\_scene рисует сцену методом Варнока, r – окно. Функция рекурсивна.

void draw\_scene(HDC hdc, LPRECT r)

# Листинг программы

// computergraphics.cpp: определяет точку входа для приложения.

// scene.txt: определяет сцену

/\* Формат файла сцены:

\* координаты камеры в экранных пикселах (x, y, z). Начало системы координат в центре экрана.

\* поворот сцены в радианах, считая от обратного напрвления оси z (вдоль оси y, вдоль оси z)

\* координаты источника света (x,y,z)

(Полигональная сетка - список вершин)

\* Количество вершин

\* Список координат вершин (x,y,z). Нумерация с нуля

\* Список полигонов (количество вершин, номер первой вершины, номер второй ...)

\*/

#include "stdafx.h"

#include "computergraphics.h"

#include <fstream>

#include <conio.h>

#include <iostream>

#include <cmath>

#include <math.h>

#include <ctime>

#define MAX\_POLYS 1000

// Глобальные переменные:

HINSTANCE hInst; // текущий экземпляр

TCHAR szTitle[] = L"Компьютерная Графика"; // Текст строки заголовка

TCHAR szWindowClass[] = L"COMPUTERGRAPHICS"; // имя класса главного окна

// Отправить объявления функций, включенных в этот модуль кода:

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

HBRUSH bgr;

/\*

Данные сцены:

\* ox,oy,oz координаты наблюдателя(камеры)

\* phi, psi углы наблюдения

\* lx,ly,lz координаты источника света

\* Полигональная сетка: способ представления: списки вершин

\* npoints количество вершин

\* npolys количество многоугольников

\* points список вершин: x,y,z координаты

\* prjpoints список вершин после преобразования координат. Фактически x,y - координаты точки-проэкции на экране.

\* polys список многоугольников: n количество вершин, p номера вершин, maxx,maxy... координаты объемлющей оболочки фигуры на экране

\*/

struct Scene {

float ox, oy, oz, phi, psi, lx,ly, lz;

struct Mesh {

int npoints, npolys;

struct Point{

float x,y,z;

} \*points;

struct ProjectPoint

{

int x,y;

float z;

} \*prjpoints;

struct Poly{

int n;

int \*p;

int maxx, maxy, minx, miny;

int color;

} \*polys;

} mesh;

} scene;

int startx=0, starty=0;

float psi , phi;

void scene\_correct(float phi, float psi);

int APIENTRY \_tWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR lpCmdLine, int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

MSG msg;

HACCEL hAccelTable;

WNDCLASSEX wcex;

HWND hWnd;

std::ifstream fs("scene.txt");

int n;

fs>>scene.ox>>scene.oy>>scene.oz>>scene.phi>>scene.psi>>scene.lx>>scene.ly>>scene.lz>>n;

scene.mesh.points = new Scene::Mesh::Point[n];

scene.mesh.prjpoints = new Scene::Mesh::ProjectPoint[n];

scene.mesh.npoints = n;

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

fs>>scene.mesh.points[i].x>>scene.mesh.points[i].y>>scene.mesh.points[i].z;

int k=0;

scene.mesh.polys = new Scene::Mesh::Poly[MAX\_POLYS];

while (!fs.eof())

{

fs>>scene.mesh.polys[k].n;

scene.mesh.polys[k].p = new int[scene.mesh.polys[k].n];

for(int j=0;j<scene.mesh.polys[k].n;j++)

fs>>scene.mesh.polys[k].p[j];

fs>>scene.mesh.polys[k].color;

k++;

}

scene.mesh.npolys=k;

scene\_correct(scene.phi,scene.psi);

fs.close();

bgr = (HBRUSH)GetStockObject(WHITE\_BRUSH);

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\_COMPUTERGRAPHICS));

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

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW+1);

wcex.lpszMenuName = MAKEINTRESOURCE(IDC\_COMPUTERGRAPHICS);

wcex.lpszClassName = szWindowClass;

wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_SMALL));

RegisterClassEx(&wcex);

// Выполнить инициализацию приложения:

hInst = hInstance; // Сохранить дескриптор экземпляра в глобальной переменной

hWnd = CreateWindow(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT, 0, CW\_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

if (!hWnd) return FALSE;

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_COMPUTERGRAPHICS));

// Цикл основного сообщения:

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

{

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

delete[] scene.mesh.points;

delete[] scene.mesh.prjpoints;

for(int i=0; i<scene.mesh.npolys;i++) delete[] scene.mesh.polys[i].p;

delete[] scene.mesh.polys;

return (int) msg.wParam;

}

// Функция scene\_correct проставляет прямоугольные объемлющие оболочки проэкций фигур.

// Функция scene\_correct осуществляет преобразование координат

void scene\_correct(float phi, float psi)

{

for(int i=0; i<scene.mesh.npoints; ++i)

{

scene.mesh.prjpoints[i].x = cosf(psi)\*scene.mesh.points[i].x + sinf(psi)\*scene.mesh.points[i].z;

scene.mesh.prjpoints[i].y = cosf(phi)\*scene.mesh.points[i].y + sinf(phi)\*(cosf(psi)\*scene.mesh.points[i].z - sinf(psi)\*scene.mesh.points[i].x);

scene.mesh.prjpoints[i].z = cosf(phi)\*(cosf(psi)\*scene.mesh.points[i].z - sinf(psi)\*scene.mesh.points[i].x) - sinf(phi)\*scene.mesh.points[i].y;

}

for(int i=0; i<scene.mesh.npolys; ++i)

{

scene.mesh.polys[i].maxx=-0x7ffffffe;

scene.mesh.polys[i].maxy=-0x7ffffffe;

scene.mesh.polys[i].minx=0x7ffffffe;

scene.mesh.polys[i].miny=0x7ffffffe;

for(int j=0; j<scene.mesh.polys[i].n; ++j)

{

Scene::Mesh::ProjectPoint p = scene.mesh.prjpoints[scene.mesh.polys[i].p[j]];

if(scene.mesh.polys[i].maxx<p.x)scene.mesh.polys[i].maxx=p.x;

if(scene.mesh.polys[i].minx>p.x)scene.mesh.polys[i].minx=p.x;

if(scene.mesh.polys[i].maxy<p.y)scene.mesh.polys[i].maxy=p.y;

if(scene.mesh.polys[i].miny>p.y)scene.mesh.polys[i].miny=p.y;

}

}

}

// Функция point\_inside определяет, лежит ли точка внутри прямоугольника pol

/\* В основе алгоритма лежит идея подсчёта количества пересечений луча, исходящего из данной точки

в направлении горизонтальной оси, со сторонами многоугольника. Если оно чётное, точка не принадлежит

многоугольнику. В данном алгоритме луч направлен влево. \*/

bool point\_inside(int x, int y, int pol)

{

bool c = false;

int npol = scene.mesh.polys[pol].n;

for (int i = 0, j = npol - 1; i < npol; j = i++)

{

int ypi = scene.mesh.prjpoints[scene.mesh.polys[pol].p[i]].y;

int ypj = scene.mesh.prjpoints[scene.mesh.polys[pol].p[j]].y;

int xpi = scene.mesh.prjpoints[scene.mesh.polys[pol].p[i]].x;

int xpj = scene.mesh.prjpoints[scene.mesh.polys[pol].p[j]].x;

if ((((ypi<=y)&&(y<ypj))||((ypj<=y)&&(y<ypi))) && (x>(xpj - xpi)\*(y - ypi)/(ypj - ypi)+xpi))

c = !c;

}

return c;

}

// Функция norma нормирует вектор

void norma(Scene::Mesh::Point& l)

{

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

l.x = l.x / L;

l.y = l.y / L;

l.z = l.z / L;

}

// Функция draw\_point закрашивает точку экрана с координатами х, у цветом многоугольника

void draw\_point(HDC hdc, int x, int y)

{

// определим, какой многоугольник находится ближе. Перебирая полигоны, смотрим какой ближе.

float minz = 1000;

int mini = -1;

for(int i=0; i<scene.mesh.npolys; ++i)

{

if(!point\_inside(x,y,i)) continue;

// построем плоскость по 3м точкам и выразим z через x, y.

Scene::Mesh::ProjectPoint p1 = scene.mesh.prjpoints[scene.mesh.polys[i].p[0]];

Scene::Mesh::ProjectPoint p2 = scene.mesh.prjpoints[scene.mesh.polys[i].p[1]];

Scene::Mesh::ProjectPoint p3 = scene.mesh.prjpoints[scene.mesh.polys[i].p[2]];

float b = ((p3.z-p1.z)\*(p2.x-p1.x)-(p2.z-p1.z)\*(p3.x-p1.x))/((p2.y-p1.y)\*(p3.x-p1.x)-(p3.y-p1.y)\*(p2.x-p1.x));

float a = (b\*(p2.y-p1.y) + p2.z - p1.z)/(p1.x-p2.x);

float d = -p1.z - a\*p1.x - b\*p1.y;

float z = -d - a\*x - b\*y;

if ((scene.oz<z)&&(minz>z)) {minz = z; mini = i;}

}

if(mini==-1)

{

SetPixel(hdc,x,y,RGB(255,255,255));

return;

}

// определим освещенность пиксела

// l <- направление на источник

Scene::Mesh::Point l, n, r, e;

l.x = scene.lx - x;

l.y = scene.ly - y;

l.z = scene.lz - minz;

norma(l);

Scene::Mesh::ProjectPoint p1 = scene.mesh.prjpoints[scene.mesh.polys[mini].p[0]];

Scene::Mesh::ProjectPoint p2 = scene.mesh.prjpoints[scene.mesh.polys[mini].p[1]];

Scene::Mesh::ProjectPoint p3 = scene.mesh.prjpoints[scene.mesh.polys[mini].p[2]];

// a = p2 - p1, b = p3 - p1, n = [a \* b] <- направление нормали

n.x = (p2.y-p1.y)\*(p3.z-p1.z) - (p2.z-p1.z)\*(p3.y-p1.y);

n.y = (p2.z-p1.z)\*(p3.x-p1.x) - (p2.x-p1.x)\*(p3.z-p1.z);

n.z = (p2.x-p1.x)\*(p3.y-p1.y) - (p2.y-p1.y)\*(p3.x-p1.x);

norma(n);

// r = 2 (l \* n) \* n - l <- отраженный луч

float ln = l.x\*n.x + l.y\*n.y + l.z\*n.z; // (l,n) <- cos(угол между источником и нормалью)

r.x = 2\*ln\*n.x - l.x;

r.y = 2\*ln\*n.y - l.y;

r.z = 2\*ln\*n.z - l.z;

norma(r);

// e <- направление на зрителя = 0,0,-1

e.x = 0;

e.y = 0;

e.z = -1;

//norma(e);

float er = e.x\*r.x + e.y\*r.y + e.z\*r.z;

//if (er<0)er=0;

// I = If\*kf + Is\*(ks\*cos(l,n) + omega\*cos^n(e,r)), n=4;

float I = 200 + 15\*(ln+5\*er\*er\*er);

int i = (I<=255.0)?(int)I:255;

// закрасим пиксел

COLORREF cl = RGB(i,i,i);

if (scene.mesh.polys[mini].color==1) cl = RGB(i,10,10);

if (scene.mesh.polys[mini].color==2) cl = RGB(10,i,10);

if (scene.mesh.polys[mini].color==3) cl = RGB(10,10,i);

SetPixel(hdc,x,y,cl);

}

// Функция is\_free возвращает, является ли прямоугольник пустым.

bool is\_free(LPRECT r)

{

for(int i=0; i<scene.mesh.npolys; ++i)

{

// bottom - miny, top - maxy, right - maxx, left - minx

bool intersectx, intersecty;

if(r->bottom < scene.mesh.polys[i].miny) intersecty = r->top > scene.mesh.polys[i].miny;

else if(r->top > scene.mesh.polys[i].maxy) intersecty = r->bottom < scene.mesh.polys[i].maxy;

else intersecty = true;

if(r->left < scene.mesh.polys[i].minx) intersectx = r->right > scene.mesh.polys[i].minx;

else if(r->right > scene.mesh.polys[i].maxx) intersectx = r->left < scene.mesh.polys[i].maxx;

else intersectx = true;

if (intersectx && intersecty) return false;

}

return true;

}

// Функция draw\_scene рисует сцену методом Варнока

void draw\_scene(HDC hdc, LPRECT r)

{

if (is\_free(r)) { r->bottom+=1;r->right+=1; FillRect(hdc,r,bgr); r->bottom-=1;r->right-=1; return; }

LPRECT t = new RECT;

if (r->bottom<=r->top)

if (r->left>=r->right) draw\_point(hdc,r->left,r->bottom);

else // left, right

{

t->bottom=r->bottom;

t->left=r->left;

t->right=(r->right+r->left)>>1;

t->top=r->top;

draw\_scene(hdc,t);

t->left=t->right+1;

t->right=r->right;

draw\_scene(hdc,t);

}

else

{

if (r->left>=r->right)

{

t->bottom=(r->top+r->bottom)>>1;

t->left=r->left;

t->right=r->right;

t->top=r->top;

draw\_scene(hdc,t);

t->top=t->bottom+1;

t->bottom=r->bottom;

draw\_scene(hdc,t);

}

else

{

t->bottom=(r->top+r->bottom)>>1;

t->left=r->left;

t->right=(r->right+r->left)>>1;

t->top=r->top;

draw\_scene(hdc,t);

t->left=t->right+1;

t->right=r->right;

draw\_scene(hdc,t);

t->top=t->bottom+1;

t->bottom=r->bottom;

draw\_scene(hdc,t);

t->right=t->left-1;

t->left=r->left;

draw\_scene(hdc,t);

}

}

delete t;

}

//

// ФУНКЦИЯ WndProc обрабатывает сообщения в главном окне.

//

// WM\_COMMAND - обработка меню приложения

// WM\_PAINT - Закрасить главное окно

// WM\_DESTROY - ввести сообщение о выходе и вернуться.

//

//

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

int wmId, wmEvent;

PAINTSTRUCT ps;

HDC hdc;

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, DialogProc);

MessageBox(hWnd,L" Лабораторная работа №2\n \"Компьютерная графика\"\n студент Винников Александр\n группа А-14-07",L"О программе",MB\_OK);

break;

case IDM\_EXIT:

DestroyWindow(hWnd);

break;

default:

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

}

break;

case WM\_PAINT:

{

clock\_t startpaint = clock ();

hdc = BeginPaint(hWnd, &ps);

LPRECT r = new RECT;

GetClientRect(hWnd,r);

LPPOINT pt = new POINT;

int cxClient=(r->left+r->right)>>1;

int cyClient=(r->bottom+r->top)>>1;

SetWindowOrgEx(hdc,-cxClient,-cyClient,pt);

r->top=-cyClient;

r->left=-cxClient;

r->bottom=cyClient;

r->right=cxClient;

draw\_scene(hdc, r);

delete r;

delete pt;

wchar\_t ch[65];

wsprintf(ch,L"phi: %i psi: %i time: %u ms",(int)(phi\*100), (int)(psi\*100), clock()-startpaint);

TextOut(hdc,cxClient-190,-cyClient+2,ch,wcslen(ch));

EndPaint(hWnd, &ps);

break;

}

case WM\_DESTROY:

PostQuitMessage(0);

break;

case WM\_LBUTTONDOWN:

startx=HIWORD(lParam);

starty=LOWORD(lParam);

break;

case WM\_LBUTTONUP:

scene.phi+=(float)(HIWORD(lParam)-startx)/100;

scene.psi+=(float)(LOWORD(lParam)-starty)/100;

startx=0;

starty=0;

break;

case WM\_MOUSEMOVE:

if(startx==0&&starty==0)break;

scene\_correct(scene.phi+(float)(HIWORD(lParam)-startx)/100, scene.psi+(float)(LOWORD(lParam)-starty)/100);

phi = scene.phi+(float)(HIWORD(lParam)-startx)/100;

psi = scene.psi+(float)(LOWORD(lParam)-starty)/100;

InvalidateRect(hWnd,0,FALSE);

//UpdateWindow(hWnd);

//RedrawWindow(hWnd,0,0,0);

break;

default:

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

}

return 0;

}