[5. Понятие об инкрементных алгоритмах растеризации. Растровое представление отрезка 2](#_Toc294988720)

[6. Понятие об инкрементных алгоритмах растеризации. Построение окружности методом средней точки 5](#_Toc294988721)

[11. Построение кривых. Интерполяционный полином Лагранжа. Достоинства и недостатки 6](#_Toc294988722)

[12. Построение кривых. Интерполяционный кубический сплайн. Реализация 8](#_Toc294988723)

[13. Построение кривых. Геометрические сплайны. Кривая Безье. Геометрический алгоритм построения кривой Безье 12](#_Toc294988724)

[14. Мировые и экранные координаты. Алгоритм пересчёта мировых 2D-координат в оконные 14](#_Toc294988725)

[16. Режимы отображения и настройка их параметров (функции класса CDC MFC). Функция SetMyMode 15](#_Toc294988726)

[24. Каркасная визуализация трёхмерных изображений. Принцип удаления невидимых граней для выпуклого многогранника 16](#_Toc294988727)

[27. Закрашивание поверхностей. Вычисление нормалей и углов отражения 16](#_Toc294988728)

[25. Изображение поверхности z = f(x,y). Метод сортировки граней по глубине (алгоритм художника) 24](#_Toc294988729)

[30. Графическая библиотека OpenGL. Структура GLUT-приложения. Пример 28](#_Toc294988730)

[31. Графическая библиотека OpenGL. Рисование графических примитивов. Дисплейные списки. Удаление нелицевых граней. Вывод текста 28](#_Toc294988731)

[32. Графическая библиотека OpenGL. Системы координат. Модельно-видовые преобразования. Примеры 28](#_Toc294988732)

[33. Графическая библиотека OpenGL. Проекции. Область вывода. Создание анимации. Примеры 28](#_Toc294988733)

[34. Графическая библиотека OpenGL. Материалы и освещение. Примеры 33](#_Toc294988734)

[35. Графическая библиотека OpenGL. Структура приложения с использованием библиотеки MFC 38](#_Toc294988735)

## 5. Понятие об инкрементных алгоритмах растеризации. Растровое представление отрезка

// Отрисовка линии по алгоритму Брезенхема, координаты которой подчиняются условию

// 0 <= y2 - y1 <= x2 - x1

void DrawLine0(CPaintDC &dc, int x1, int y1, int x2, int y2, COLORREF color)

{

int

dx = x2 - x1,

dy = y2 - y1,

d = 2 \* dy - dx,

d1 = 2 \* dy,

d2 = 2 \* (dy - dx);

dc.SetPixel(x1, y1, color);

for (int x = x1 + 1, y = y1; x <= x2; ++x) // Цикл слева направо

{

if (d > 0)

{

y += y2 < y1 ? -1 : 1;

d += d2;

}

else { d += d1; }

dc.SetPixel(x, y, color);

}

}

// Рисовать любую линию по алгоритму Брезенхема

void DrawLine1(CPaintDC &dc, int x1, int y1, int x2, int y2, COLORREF color)

{

int dx = x2 > x1 ? x2 - x1 : x1 - x2, // модуль dx

dy = y2 > y1 ? y2 - y1 : y1 - y2, // модуль dy

t = 0; // временная переменная для обмена

dc.SetPixel(x1, y1, color);

if (dy <= dx) // Цикл по x

{

int d = 2 \* dy - dx,

d1 = 2 \* dy,

d2 = 2 \* (dy - dx);

if (x2 < x1) // x2 - x1 должно быть >= 0

{

t = x2; x2 = x1; x1 = t;

t = y2; y2 = y1; y1 = t;

}

for (int x = x1 + 1, y = y1; x <= x2; ++x)

{

if (d > 0)

{

y += y2 < y1 ? -1 : 1;

d += d2;

}

else { d += d1; }

dc.SetPixel(x, y, color);

}

}

else // Цикл по y

{

int d = 2 \* dx - dy,

d1 = 2 \* dx,

d2 = 2 \* (dx - dy);

if (y2 < y1) // y2 - y1 должно быть >= 0

{

t = x2; x2 = x1; x1 = t;

t = y2; y2 = y1; y1 = t;

}

for (int x = x1, y = y1 + 1; y <= y2; ++y)

{

if (d > 0)

{

x += x2 < x1 ? -1 : 1;

d += d2;

}

else { d += d1; }

dc.SetPixel(x, y, color);

}

}

}

// Рисовать линию (инкрементный восьмисвязный алгоритм Брезенхэма)

void DrawLine2(CPaintDC &dc, int x1, int y1, int x2, int y2, COLORREF color)

{

int dx = x2 - x1, dy = y2 - y1, d;

int incX = (dx >= 0) ? ((dx > 0) ? (1) : (0)) : (-1);

int incY = (dy >= 0) ? ((dy > 0) ? (1) : (0)) : (-1);

if (dx < 0) { dx = -dx; }

if (dy < 0) { dy = -dy; }

d = (dx > dy) ? (dx) : (dy);

dc.SetPixel(x1, y1, color);

for (int i = 0, x = x1, y = y1, xerr = 0, yerr = 0; i < d; ++i)

{

xerr += dx;

yerr += dy;

if (xerr > d)

{

xerr -= d;

x += incX;

}

if (yerr > d)

{

yerr -= d;

y += incY;

}

dc.SetPixel(x, y, color);

}

}

## 6. Понятие об инкрементных алгоритмах растеризации. Построение окружности методом средней точки

POINT circleCenter; // центр окружности

// Отрисовка "симметричных" точек окружности

void DrawCirclePoints(CPaintDC &dc, int x, int y, COLORREF color)

{

dc.SetPixel(circleCenter.x + x, circleCenter.y + y, color);

dc.SetPixel(circleCenter.x + y, circleCenter.y + x, color);

dc.SetPixel(circleCenter.x + y, circleCenter.y - x, color);

dc.SetPixel(circleCenter.x + x, circleCenter.y - y, color);

dc.SetPixel(circleCenter.x - x, circleCenter.y - y, color);

dc.SetPixel(circleCenter.x - y, circleCenter.y - x, color);

dc.SetPixel(circleCenter.x - y, circleCenter.y + x, color);

dc.SetPixel(circleCenter.x - x, circleCenter.y + y, color);

}

// Рисовать окружность

void DrawCircle(CPaintDC &dc, int xc, int yc, int radius, COLORREF color)

{

int x = 0,

y = radius,

d = 1 - radius,

d1 = 3,

d2 = -2 \* radius + 5;

circleCenter.x = xc;

circleCenter.y = yc;

DrawCirclePoints(dc, x, y, color);

for (; y > x; ++x, d1 += 2)

{

if (d < 0)

{

d += d1;

d2 += 2;

}

else

{

d += d2;

d2 += 4;

--y;

}

DrawCirclePoints(dc, x, y, color);

}

}

## 11. Построение кривых. Интерполяционный полином Лагранжа. Достоинства и недостатки

Lagrange.h:

#pragma once

#include "CMatrix.h"

class Lagrange

{

private:

double \*m\_x, \*m\_y; // Массивы координат интерполируемой функции

unsigned int m\_n; // Размерность x, y

public:

Lagrange(const double \*x, const double \*y, unsigned int n);

Lagrange(CMatrix & x, CMatrix & y);

~Lagrange();

double F(double x) const; // Интерполированная функция y = f(x)

void BuildLagrange(

CMatrix & x, CMatrix & y, double xFrom, double xTo, double step);

};

Lagrange.cpp:

#include "stdafx.h"

#include "Lagrange.h"

Lagrange::Lagrange(const double \*x, const double \*y, unsigned int n) : m\_n(n)

{

::memcpy(m\_x, x, n);

::memcpy(m\_y, y, n);

}

Lagrange::Lagrange(CMatrix & x, CMatrix & y)

{

m\_n = x.cols() \* x.rows();

m\_x = new double[m\_n];

m\_y = new double[m\_n];

for (unsigned int i = 0; i < m\_n; ++i)

{

m\_x[i] = x(i);

m\_y[i] = y(i);

}

}

Lagrange::~Lagrange()

{

delete[] m\_x, m\_y;

}

double Lagrange::F(double x) const

{

double c, y = 0;

for (unsigned int i = 0, j; i < m\_n; ++i)

{

for (j = 0, c = 1; j < m\_n; ++j)

{

if (i != j)

{

c \*= (x - m\_x[j]) / (m\_x[i] - m\_x[j]);

}

}

y += c \* m\_y[i];

}

return y;

}

void Lagrange::BuildLagrange(CMatrix & x, CMatrix & y, double xLeft, double xRight, double step)

{

int k = (int)((xRight - xLeft) / step + 1);

x.RedimMatrix(k);

y.RedimMatrix(k);

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

{

x(i) = xLeft + i \* step;

y(i) = this->F(x(i));

}

}

## 12. Построение кривых. Интерполяционный кубический сплайн. Реализация

CubicSpline.h:

#pragma once

#include "CMatrix.h"

#include <limits>

class CubicSpline

{

private:

struct spline\_tuple // Структура, описывающая сплайн на каждом сегменте сетки

{

double a, b, c, d, x;

};

spline\_tuple \*m\_splines; // Сплайн

unsigned int m\_n; // Количество узлов сетки

public:

CubicSpline(const double \*x, const double \*y, unsigned int n);

CubicSpline(CMatrix & x, CMatrix & y);

~CubicSpline();

double F(double x) const; // Интерполированная функция y = f(x)

void BuildSpline(CMatrix & x, CMatrix & y, double xFrom, double xTo, double step);

private:

void x\_BuildSpline( // Построение сплайна

const double \*x, // узлы сетки, должны быть упорядочены по возрастанию, кратные узлы запрещены

const double \*y, // значения функции в узлах сетки

unsigned int n // количество узлов сетки

);

void x\_FreeMemory();

};

CubicSpline.cpp:

#include "stdafx.h"

#include "CubicSpline.h"

#include <cmath>

CubicSpline::CubicSpline(const double \*x, const double \*y, unsigned int n) :

m\_splines(NULL)

{

x\_BuildSpline(x, y, n);

}

CubicSpline::CubicSpline(CMatrix & x, CMatrix & y) : m\_splines(NULL)

{

unsigned int n = x.cols() \* x.rows();

double

\*xTmp = new double[n],

\*yTmp = new double[n];

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

{

xTmp[i] = x(i);

yTmp[i] = y(i);

}

x\_BuildSpline(xTmp, yTmp, n);

delete[] xTmp, yTmp;

}

CubicSpline::~CubicSpline()

{

x\_FreeMemory();

}

double CubicSpline::F(double x) const

{

if (!m\_splines)

return std::numeric\_limits<double>::quiet\_NaN(); // Если сплайны ещё не построены - возвращаем NaN

spline\_tuple \*s;

if (x <= m\_splines[0].x) // Если x меньше точки сетки x[0] - пользуемся первым эл-тов массива

s = m\_splines + 1;

else if (x >= m\_splines[m\_n - 1].x) // Если x больше точки сетки x[n - 1] - пользуемся последним эл-том массива

s = m\_splines + m\_n - 1;

else // Иначе x лежит между граничными точками сетки - производим бинарный поиск нужного эл-та массива

{

unsigned int i = 0, j = m\_n - 1;

while (i + 1 < j)

{

unsigned int k = i + (j - i) / 2;

if (x <= m\_splines[k].x)

j = k;

else

i = k;

}

s = m\_splines + j;

}

double dx = (x - s->x);

return s->a + (s->b + (s->c / 2. + s->d \* dx / 6.) \* dx) \* dx; // Вычисляем значение сплайна в заданной точке по схеме Горнера (в принципе, "умный" компилятор применил бы схему Горнера сам, но ведь не все так умны, как кажутся)

}

void CubicSpline::BuildSpline(CMatrix & x, CMatrix & y, double xLeft, double xRight, double step)

{

int k = (int)((xRight - xLeft) / step + 1);

x.RedimMatrix(k);

y.RedimMatrix(k);

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

{

x(i) = xLeft + i \* step;

y(i) = this->F(x(i));

}

}

void CubicSpline::x\_BuildSpline(const double \*x, const double \*y, unsigned int n)

{

x\_FreeMemory();

m\_n = n;

// Инициализация массива сплайнов

m\_splines = new spline\_tuple[n];

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

{

m\_splines[i].x = x[i];

m\_splines[i].a = y[i];

}

m\_splines[0].c = m\_splines[n - 1].c = 0.;

// Решение СЛАУ относительно коэффициентов сплайнов c[i] методом прогонки для трехдиагональных матриц

// Вычисление прогоночных коэффициентов - прямой ход метода прогонки

double \*alpha = new double[n - 1];

double \*beta = new double[n - 1];

alpha[0] = beta[0] = 0.;

for (unsigned int i = 1; i < n - 1; ++i)

{

double h\_i = x[i] - x[i - 1], h\_i1 = x[i + 1] - x[i];

double A = h\_i;

double C = 2. \* (h\_i + h\_i1);

double B = h\_i1;

double F = 6. \* ((y[i + 1] - y[i]) / h\_i1 - (y[i] - y[i - 1]) / h\_i);

double z = (A \* alpha[i - 1] + C);

alpha[i] = -B / z;

beta[i] = (F - A \* beta[i - 1]) / z;

}

// Нахождение решения - обратный ход метода прогонки

for (unsigned int i = n - 2; i > 0; --i)

m\_splines[i].c = alpha[i] \* m\_splines[i + 1].c + beta[i];

// Освобождение памяти, занимаемой прогоночными коэффициентами

delete[] beta;

delete[] alpha;

// По известным коэффициентам c[i] находим значения b[i] и d[i]

for (unsigned int i = n - 1; i > 0; --i)

{

double h\_i = x[i] - x[i - 1];

m\_splines[i].d = (m\_splines[i].c - m\_splines[i - 1].c) / h\_i;

m\_splines[i].b = h\_i \* (2. \* m\_splines[i].c + m\_splines[i - 1].c) / 6. + (y[i] - y[i - 1]) / h\_i;

}

}

void CubicSpline::x\_FreeMemory()

{

if (m\_splines)

{

delete[] m\_splines;

m\_splines = NULL;

}

}

## 13. Построение кривых. Геометрические сплайны. Кривая Безье. Геометрический алгоритм построения кривой Безье

Bezier.h:

#pragma once

#include "CMatrix.h"

class Bezier

{

private:

double \*m\_x, \*m\_y; // Массивы

unsigned int m\_n; // Размерность x, y

public:

Bezier(const double \*x, const double \*y, unsigned int n);

Bezier(CMatrix & x, CMatrix & y);

~Bezier();

void BuildBezier(CMatrix & x, CMatrix & y, double step);

};

Bezier.cpp:

#include "stdafx.h"

#include "Bezier.h"

Bezier::Bezier(const double \*x, const double \*y, unsigned int n) : m\_n(n)

{

::memcpy(m\_x, x, n);

::memcpy(m\_y, y, n);

}

Bezier::Bezier(CMatrix & x, CMatrix & y)

{

m\_n = x.cols() \* x.rows();

m\_x = new double[m\_n];

m\_y = new double[m\_n];

for (unsigned int i = 0; i < m\_n; ++i)

{

m\_x[i] = x(i);

m\_y[i] = y(i);

}

}

Bezier::~Bezier()

{

delete[] m\_x, m\_y;

}

void Bezier::BuildBezier(CMatrix & x, CMatrix & y, double step)

{

unsigned int sz = 1 / step + 1, g = 0, i;

x.RedimMatrix(sz);

y.RedimMatrix(sz);

for(double t = 0; t < 1.0001; t += step, ++g)

{

double\*\* mas = new double\*[2];

for(i = 0; i < m\_n; ++i)

{

mas[i] = new double[2];

}

for(i = 0; i < m\_n; ++i)

{

mas[i][0] = m\_x[i];

mas[i][1] = m\_y[i];

}

for(unsigned int k = m\_n, j; k > 0; --k)

{

for(j = 0; j < k - 1; ++j)

{

mas[j][0] = mas[j][0] \* (1 - t) + mas[j + 1][0] \* t;

mas[j][1] = mas[j][1] \* (1 - t) + mas[j + 1][1] \* t;

}

}

x(g) = mas[0][0];

y(g) = mas[0][1];

for(i = 0; i < m\_n; ++i) { delete[] mas[i]; }

}

}

## 14. Мировые и экранные координаты. Алгоритм пересчёта мировых 2D-координат в оконные

CMatrix SpaceToWindow(CRectD & rectSpace, CRect& rectWindow)

{

CMatrix matr(3,3);

//Размер области в окне

int dwx = rectWindow.Width();

int dwy = rectWindow.Height();

// Размер области в мировых координатах

double dsx = rectSpace.GetWidth();

double dsy = rectSpace.GetHeight();

double kx = (double)dwx / dsx; // Масштаб по x

double ky = (double)dwy / dsy; // Масштаб по y

matr(0,0) = kx;

matr(1,0) = 0;

matr(2,0) = 0;

matr(0,1) = 0;

matr(1,1) = -ky;

matr(2,1) = 0;

matr(0,2) = (double)rectWindow.left - kx \* rectSpace.GetLeft();

matr(1,2) = (double)rectWindow.bottom + ky \* rectSpace.GetBottom();

matr(2,2) = 1;

return matr;

}

## 16. Режимы отображения и настройка их параметров (функции класса CDC MFC). Функция SetMyMode

void CPlot2D::SetMyMode(CDC & dc, CRectD & rSpace, CRect & rWindow)

{

int dsx = (int)rSpace.GetWidth(),

dsy = (int)rSpace.GetHeight(),

xsl = (int)rSpace.GetLeft(),

ysl = (int)rSpace.GetBottom(),

dwx = (int)rWindow.Width(),

dwy = (int)rWindow.Height(),

xwl = (int)rWindow.left,

ywl = (int)rWindow.bottom;

dc.SetMapMode(MM\_ANISOTROPIC);

dc.SetWindowExt(dsx, dsy);

dc.SetViewportExt(dwx, -dwy);

dc.SetWindowOrg(xsl, ysl);

dc.SetViewportOrg(xwl, ywl);

}

## 24. Каркасная визуализация трёхмерных изображений. Принцип удаления невидимых граней для выпуклого многогранника

## 27. Закрашивание поверхностей. Вычисление нормалей и углов отражения

LibPyramid.h:

class CPyramid

{

CMatrix Vertices; // Координаты вершин

void GetRect(CMatrix& Vert,CRectD& RectView);

public:

CPyramid();

void Draw(CDC& dc,CMatrix& P,CRect& RW);

void Draw1(CDC& dc,CMatrix& P,CRect& RW);

void Draw2(CDC& dc,CMatrix& P,CRect& RW);

void ColorDraw(CDC& dc,CMatrix& PView,CRect& RW,COLORREF Color);

};

LibPyramid.cpp:

CPyramid::CPyramid()

{

int rDown = 6, rUp = 3, height = 15;

Vertices.RedimMatrix(4,8);

// Координаты вершин - столбцы

Vertices(0,0)=rDown; // A: x=10,y=0,z=0

Vertices(1,1)=-rDown; // B: x=0,y=-10,z=0

Vertices(0,2)=-rDown; // C: x=-10,y=0,z=0

Vertices(1,3)=rDown; // D: x=0,y=10,z=0

Vertices(0,4)=rUp;

Vertices(1,5)=-rUp;

Vertices(0,6)=-rUp;

Vertices(1,7)=rUp;

Vertices(2,4)=height;

Vertices(2,5)=height;

Vertices(2,6)=height;

Vertices(2,7)=height;

//for(int i=0;i<8;i++)Vertices(3,i)=0;

}

void CPyramid::Draw1(CDC& dc,CMatrix& PView,CRect& RW)

// Рисует пирамиду С УДАЛЕНИЕМ невидимых ребер

// Самостоятельный пересчет координат из мировых в оконные (MM\_TEXT)

// dc - ссылка на класс CDC MFC

// PView - координаты точки наблюдения в мировой сферической системе координат

// (r,fi(град.), q(град.))

// RW - область в окне для отображения

{

CMatrix ViewCart=SphereToCart(PView); // Декартовы координаты точки наблюдения

//double zz=ViewCart(2);

CMatrix MV=CreateViewCoord(PView(0),PView(1),PView(2)); //Матрица(4x4)

//для пересчета в видовую СК

CMatrix ViewVert=MV\*Vertices; // Координаты вершин пирамиды в видовой СК

CRectD RectView;

GetRect(ViewVert,RectView); // Получаем охватывающий прямоугольник

CMatrix MW=SpaceToWindow(RectView,RW); // Матрица (3x3) для пересчета

// в оконную систему координат

// Готовим массив оконных координат для рисования

CPoint MasVert[8]; // Массив оконных координат вершин

CMatrix V(3); //V0(3);

V(2)=1;

//V0(0)=RectView.left;

//V0(1)=RectView.bottom;

// Цикл по количеству вершин - вычисляем оконные координаты вершин

for(int i=0;i<8;i++)

{

V(0)=ViewVert(0,i); // x

V(1)=ViewVert(1,i); // y

//V=V-V0; // x-xL, y-yL, 1

V=MW\*V; // Оконные координаты точки

MasVert[i].x=(int)V(0);

MasVert[i].y=(int)V(1);

}

// Рисуем

CPen Pen(PS\_SOLID, 2, RGB(100, 255, 100));

CPen\* pOldPen =dc.SelectObject(&Pen);

CBrush Brus(RGB(255,100 , 100));

CBrush\* pOldBrush =dc.SelectObject(&Brus);

CMatrix VE=Vertices.GetCol(4,0,2); // Вершина E

CMatrix R1(3),R2(3), R3(3), VN(3);

double sm;

for(int i=0;i<4;i++)

{

int k;

if(i==3) k=0;

else k=i+1;

R1=Vertices.GetCol(i,0,2);

R2=Vertices.GetCol(k,0,2);

R3=Vertices.GetCol(i+4,0,2);

//R2=Vertices.GetCol(i+5,0,2);

CMatrix V1=R2-R1; // Вектор - ребро в основании

CMatrix V2=R3-R2; // Вектор - ребро сверху

VN=VectorMult(V2,V1); // Вектор ВНЕШНЕЙ(!) нормали к грани

sm=ScalarMult(VN,ViewCart); // Скалярное произведение вектора

//нормали к грани и вектора точки наблюдения

if(sm>=0) // Грань видима - рисуем боковую грань

{

dc.MoveTo(MasVert[i]);

dc.LineTo(MasVert[i+4]);

dc.LineTo(MasVert[i == 3 ? 4 : (i+5)]);

dc.LineTo(MasVert[i == 3 ? 0 : (i+1)]);

dc.LineTo(MasVert[i]);

}

}

dc.SelectObject(&pOldBrush);

VN=VectorMult(R1,R2);

sm=ScalarMult(VN,ViewCart);

if(sm>=0)dc.Polygon(MasVert, 4); // Основание

else { // Верхушка

dc.MoveTo(MasVert[4]);

dc.LineTo(MasVert[5]);

dc.LineTo(MasVert[6]);

dc.LineTo(MasVert[7]);

dc.LineTo(MasVert[4]);

}

dc.SelectObject(pOldPen);

dc.SelectObject(pOldBrush);

}

////////////////////////// Draw(...) ///////////////////////////////

void CPyramid::Draw(CDC& dc,CMatrix& PView,CRect& RW)

// Рисует пирамиду БЕЗ удаления невидимых ребер

// Самостоятельный пересчет координат из мировых в оконные (MM\_TEXT)

// dc - ссылка на класс CDC MFC

// PView - координаты точки наблюдения в мировой сферической системе координат

// (r,fi(град.), q(град.))

// RW - область в окне для отображения

{

CMatrix ViewCart=SphereToCart(PView); // Декартовы координаты точки наблюдения

//double zz=ViewCart(2);

CMatrix MV=CreateViewCoord(PView(0),PView(1),PView(2)); //Матрица(4x4)

//для пересчета в видовую СК

CMatrix ViewVert=MV\*Vertices; // Координаты вершин пирамиды в видовой СК

CRectD RectView;

GetRect(ViewVert,RectView); // Получаем охватывающий прямоугольник

CMatrix MW=SpaceToWindow(RectView,RW); // Матрица (3x3) для пересчета

// в оконную систему координат

// Готовим массив оконных координат для рисования

CPoint MasVert[8]; // Массив оконных координат вершин

CMatrix V(3); //V0(3);

V(2)=1;

//V0(0)=RectView.left;

//V0(1)=RectView.bottom;

// Цикл по количеству вершин - вычисляем оконные координаты вершин

for(int i=0;i<8;i++)

{

V(0)=ViewVert(0,i); // x

V(1)=ViewVert(1,i); // y

//V=V-V0; // x-xL, y-yL, 1

V=MW\*V; // Оконные координаты точки

MasVert[i].x=(int)V(0);

MasVert[i].y=(int)V(1);

}

// Рисуем

CPen Pen(PS\_SOLID, 2, RGB(0, 0, 0));

CPen\* pOldPen =dc.SelectObject(&Pen);

dc.Polygon(MasVert, 4); // Основание

//dc.Polygon(MasVert + 4, 4); // Верхушка

dc.MoveTo(MasVert[4]);

dc.LineTo(MasVert[5]);

dc.LineTo(MasVert[6]);

dc.LineTo(MasVert[7]);

dc.LineTo(MasVert[4]);

for(int i=0;i<4;i++)

{

dc.MoveTo(MasVert[i+4]); // Перо на вершину E

dc.LineTo(MasVert[i]);

}

// Координаты центра O пересечения диагоналей основания

int x0=(MasVert[0].x+MasVert[2].x)/2;

int y0=(MasVert[0].y+MasVert[2].y)/2;

int x1=(MasVert[4].x+MasVert[6].x)/2;

int y1=(MasVert[4].y+MasVert[6].y)/2;

}

void CPyramid::Draw2(CDC& dc,CMatrix& PView,CRect& RW)

// Рисует пирамиду БЕЗ удаления невидимых ребер

// Пересчет координат выполняет Windows, для этого

// должен быть установлен режим отображения MM\_ANISOTROPIC

// dc - ссылка на класс CDC MFC

// PView - координаты точки наблюдения в мировой сферической системе координат

// (r,fi(град.), q(град.))

// RW - область в окне для отображения

{

CMatrix ViewCart=SphereToCart(PView); // Декартовы координаты точки наблюдения

double zz=ViewCart(2);

CMatrix MV=CreateViewCoord(PView(0),PView(1),PView(2)); //Матрица(4x4)

//для пересчета в видовую СК

CMatrix ViewVert=MV\*Vertices; // Координаты вершин пирамиды в видовой СК

CRectD RectView;

GetRect(ViewVert,RectView); // Получаем охватывающий прямоугольник

CPoint MasVert[8];

for(int i=0;i<8;i++)

{

MasVert[i].x=(int)ViewVert(0,i);

MasVert[i].y=(int)ViewVert(1,i);

}

CRect IRV((int)RectView.left,(int)RectView.top,(int)RectView.right,(int)RectView.bottom);

SetMyMode(dc,IRV,RW);

// Рисуем

CPen Pen(PS\_SOLID, 0, RGB(255, 0, 0));

CPen\* pOldPen =dc.SelectObject(&Pen);

dc.Polygon(MasVert, 4); // Основание

dc.Polygon(MasVert + 4, 4); // Верхушка

for(int i=0;i<8;i++)

{

dc.MoveTo(MasVert[i+4]);

dc.LineTo(MasVert[i]);

}

// Координаты центра O пересечения диагоналей основания

int x0=(MasVert[0].x+MasVert[2].x)/2;

int y0=(MasVert[0].y+MasVert[2].y)/2;

int x1=(MasVert[4].x+MasVert[6].x)/2;

int y1=(MasVert[4].y+MasVert[6].y)/2;

// Рисуем диагонали и линию из E в O

CPen Pen1(PS\_DASH, 0, RGB(0, 255, 0));

dc.SelectObject(&Pen1);

dc.MoveTo(x1,y1);

dc.LineTo(x0,y0);

dc.MoveTo(MasVert[0]); // Перо на вершину A

dc.LineTo(MasVert[2]); // Диагональ

dc.MoveTo(MasVert[1]); // Перо на вершину B

dc.LineTo(MasVert[3]); // Диагональ

dc.MoveTo(MasVert[4]); // Перо на вершину A1

dc.LineTo(MasVert[6]); // Диагональ

dc.MoveTo(MasVert[5]); // Перо на вершину B1

dc.LineTo(MasVert[7]); // Диагональ

dc.SelectObject(pOldPen);

dc.SetMapMode(MM\_TEXT);

}

void CPyramid::ColorDraw(CDC& dc,CMatrix& PView,CRect& RW,COLORREF Color)

// Рисует пирамиду С УДАЛЕНИЕМ невидимых ребер + заливка и освещение

// Самостоятельный пересчет координат из мировых в оконные (MM\_TEXT)

// dc - ссылка на класс CDC MFC

// PView - координаты точки наблюдения в мировой сферической системе координат

// (r,fi(град.), q(град.))

// RW - область в окне для отображения

{

BYTE red=GetRValue(Color);

BYTE green=GetGValue(Color);

BYTE blue=GetBValue(Color);

CMatrix ViewCart=SphereToCart(PView); // Декартовы координаты точки наблюдения

CMatrix MV=CreateViewCoord(PView(0),PView(1),PView(2)); //Матрица(4x4) для пересчета в видовую СК

CMatrix ViewVert=MV\*Vertices; // Координаты вершин пирамиды в видовой СК

CRectD RectView;

GetRect(ViewVert,RectView); // Получаем охватывающий прямоугольник

CMatrix MW=SpaceToWindow(RectView,RW); // Матрица (3x3) для пересчета

// в оконную систему координат

// Готовим массив оконных координат для рисования

CPoint MasVert[8]; // Массив оконных координат вершин

CMatrix V(3); //V0(3);

V(2)=1;

// Цикл по количеству вершин - вычисляем оконные координаты вершин

for(int i=0;i<8;i++)

{

V(0)=ViewVert(0,i); // x

V(1)=ViewVert(1,i); // y

//V=V-V0; // x-xL, y-yL, 1

V=MW\*V; // Оконные координаты точки

MasVert[i].x=(int)V(0);

MasVert[i].y=(int)V(1);

}

// Рисуем

CMatrix VE=Vertices.GetCol(4,0,2); // Вершина E

CMatrix R1(3),R2(3), R3(3), VN(3);

double sm;

for(int i=0;i<4;i++)

{

int k;

if(i==3) k=0;

else k=i+1;

R1=Vertices.GetCol(i,0,2);

R2=Vertices.GetCol(k,0,2);

R3=Vertices.GetCol(i+4,0,2);

CMatrix V1=R2-R1; // Вектор - ребро в основании

CMatrix V2=R3-R2; // Вектор - ребро в верхушке

VN=VectorMult(V2,V1); // Вектор ВНЕШНЕЙ(!) нормали к грани

sm=CosV1V2(VN,ViewCart); // Косинус угла между вектором нормали

// к грани и вектором точки наблюдения

if(sm>=0) // Грань видима - рисуем боковую грань

{

CPen Pen(PS\_SOLID, 2, RGB(sm\*sm\*red, sm\*sm\*green, sm\*sm\*blue));

CPen\* pOldPen =dc.SelectObject(&Pen);

CBrush Brus(RGB(sm\*sm\*red, sm\*sm\*green, sm\*sm\*blue));

CBrush\* pOldBrush =dc.SelectObject(&Brus);

CPoint MasVertR[4]={MasVert[i+4], MasVert[i == 3 ? 4 : (i+5)], MasVert[i == 3 ? 0 : (i+1)], MasVert[i]};

dc.Polygon(MasVertR, 4); // Ребро

dc.SelectObject(pOldBrush);

dc.SelectObject(pOldPen);

}

}

VN=VectorMult(R1,R2);

sm=CosV1V2(VN,ViewCart);

CPen Pen(PS\_SOLID, 2, RGB(sm\*sm\*red, sm\*sm\*green, sm\*sm\*blue));

CPen\* pOldPen =dc.SelectObject(&Pen);

CBrush Brus(RGB(sm\*sm\*red, sm\*sm\*green, sm\*sm\*blue));

CBrush\* pOldBrush =dc.SelectObject(&Brus);

if(sm>=0)

{

dc.Polygon(MasVert, 4); // Основание

}

else

{

dc.Polygon(MasVert+4, 4); // Верхушка

}

dc.SelectObject(pOldBrush);

dc.SelectObject(pOldPen);

}

void CPyramid::GetRect(CMatrix& Vert,CRectD& RectView)

// Вычисляет координаты прямоугольника,охватывающего проекцию

// пирамиды на плоскость XY в ВИДОВОЙ системе координат

// Ver - координаты вершин (в столбцах)

// RectView - проекция - охватывающий прямоугольник

{

CMatrix V=Vert.GetRow(0); // x - координаты

double xMin=V.MinElement();

double xMax=V.MaxElement();

V=Vert.GetRow(1); // y - координаты

double yMin=V.MinElement();

double yMax=V.MaxElement();

RectView.SetRectD(xMin,yMax,xMax,yMin);

}

## 25. Изображение поверхности z = f(x,y). Метод сортировки граней по глубине (алгоритм художника)

Surface.h:

#pragma once

#include "CRectD.h"

#include "CMatrix.h"

class Surface

{

public:

Surface(void);~Surface(void);

typedef double (\*SurfaceFunction)(double, double); // функция рассчитывает набор точек на поверхности

SurfaceFunction Sf;

CRectD RD;

CRect RW;

CRectD RV;

CMatrix PView;

int Nx, Ny;

POINT \*\*MasPoint;

//pfunc2 PF;

CMatrix MW;

bool st;

// (формирует матрицу для отображения точек на поверхности)

POINT \*\*CreateSurfaceMatrix(SurfaceFunction Func, CRectD &RS, int Nx, int Ny, CRect &RW, CMatrix &PView);

void GetProjection(CRectD &RS, CMatrix Data, CMatrix PView, CRectD &PR);

void DrawSurface(CDC &dc);// функция прорисовки поверхности

void PreDraw(double fi , double teta,SurfaceFunction sf);

void PreDraw(double fi , double teta){PreDraw(fi,teta,Sf);}

};

Surface.cpp:

#include "Surface.h"

#include "GraphicFunctions.h"

using namespace GraphicFunctions;

Surface::Surface(void)

{

Nx=100, Ny=100;

MasPoint=NULL;

PView.RedimMatrix(3);

PView(0)=1; PView(1)=300; PView(2)=80;

RD.SetRectD(-10,-10,10,10);

RW.SetRect(100,100,500,500);

st=true;

MW.RedimMatrix(3,3);

}

void Surface::PreDraw(double fi , double teta,SurfaceFunction sf)

{

if (MasPoint!=NULL) {for (int i=0;i<Nx;i++){delete MasPoint[i];} delete MasPoint;}

PView(0)=1; PView(1)=fi; PView(2)=teta;

MasPoint=CreateSurfaceMatrix(sf, RD, Nx, Ny, RW, PView);

}

Surface::~Surface(void)

{

}

POINT\*\* Surface::CreateSurfaceMatrix(SurfaceFunction MyF, CRectD &RS, int Nx, int Ny, CRect &RW, CMatrix &PView)

{

// MyF - функция поверхности, RS - область в мировых координтах на которую опирается поверхность,

// Nx, Ny - число точек на осях x - y для вычисления значений,

// RW - область отображения на экране, PView - параметры точки наблюдения

// определение дальнего угла

double xL, yL, ddx;

CMatrix RR = SphereToCart(PView);

double xv=RR(0), yv=RR(1), zv=RR(2);

// определим шаги по x и y (вычисление точек поверхности) - нужно реальное расстояние, поэтому - нужен модуль

double dx= (RS.right - RS.left)/(Nx-1);

double dy= (RS.top - RS.bottom)/(Ny-1);

// точка наблюдения (непосредственное вычисление точки наблюдения)

double y=RS.bottom+(RS.top-RS.bottom)\*(xv-RS.left)/(RS.right-RS.left);

// определить дальнюю точку

yL=RS.bottom;

if (yv<y) {xL=RS.left;ddx=dx;}

else {xL=RS.right;ddx=-dx;}

// матрица заполнения значений функции

CMatrix F(Nx,Ny);

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

{

double xi=xL+i\*ddx;

for (int j=0;j<Ny;j++)

{

double yj=yL+j\*dy;

double Zij=MyF(xi,yj);

F(i,j)=Zij;

}

}

if (st)

{

// определим охватывающий прямоугольник по крайним точкам фигуры

GetProjection(RS, F, PView, RV);

// пересчет в оконную систему координат

MW=SpaceToWindow(RV, RW);st=false;

}

//CMatrix MW=SpaceToWindow(RV, RW);

CMatrix V(3); V(2)=1;

// матрица пересчета в видовую

CMatrix MV=CreateViewCoord(PView(0), PView(1), PView(2));

CMatrix PS(4); PS(3)=1;

POINT \*\*MasPoint=NULL;

MasPoint=new POINT\*[Nx];

for (int i=0;i<Nx;i++) {MasPoint[i]=new POINT [Ny];}

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

{

double xi=xL+i\*ddx;

PS(0)=xi;

for (int j=0;j<Ny;j++)

{

double yi=yL+j\*dy;

double Zij=F(i,j);

PS(1)=yi;

PS(2)=Zij;

CMatrix PV=MV\*PS;

V(0)=PV(0); V(1)=PV(1); V=MW\*V;

MasPoint[i][j].x=(int)V(0);

MasPoint[i][j].y=(int)V(1);

}

}

return MasPoint;

}

void Surface::DrawSurface(CDC &dc)//, POINT\*\* MasPoint, int Nx, int Ny)

{

POINT pt[4];

for(int i=0;i<Nx-1;i++)

{

for (int j=0;j<Ny-1;j++)

{

pt[0].x=MasPoint[i][j].x;

pt[0].y=MasPoint[i][j].y;

pt[1].x=MasPoint[i][j+1].x;

pt[1].y=MasPoint[i][j+1].y;

pt[2].x=MasPoint[i+1][j+1].x;

pt[2].y=MasPoint[i+1][j+1].y;

pt[3].x=MasPoint[i+1][j].x;

pt[3].y=MasPoint[i+1][j].y;

dc.Polygon(pt,4);

}

}

}

// получить проекции

void Surface::GetProjection(CRectD &RS, CMatrix Data, CMatrix PView, CRectD &PR)

{

double Zmax=Data.MaxElement();

double Zmin=Data.MinElement();

CMatrix PS(4,4);

PS(3,0)=1; PS(3,1)=1; PS(3,2)=1; PS(3,3)=1;

CMatrix MV=CreateViewCoord(PView(0), PView(1), PView(2));

PS(0,0)=RS.left; PS(0,1)=RS.left; PS(0,2)=RS.right; PS(0,3)=RS.right;

PS(1,0)=RS.top; PS(1,1)=RS.bottom; PS(1,2)=RS.top; PS(1,3)=RS.bottom;

PS(2,0)=Zmax; PS(2,1)=Zmax; PS(2,2)=Zmax; PS(2,3)=Zmax;

CMatrix Q=MV\*PS;

CMatrix V=Q.GetRow(0);

double Xmin=V.MinElement();

double Xmax=V.MaxElement();

V=Q.GetRow(1);

double Ymax=V.MaxElement();

PS(2,0)=Zmin; PS(2,1)=Zmin; PS(2,2)=Zmin; PS(2,3)=Zmin;

Q=MV\*PS; V=Q.GetRow(1);

double Ymin=V.MinElement();

PR.SetRectD(Xmin, Ymax, Xmax, Ymin);

## 30. Графическая библиотека OpenGL. Структура GLUT-приложения. Пример

## 31. Графическая библиотека OpenGL. Рисование графических примитивов. Дисплейные списки. Удаление нелицевых граней. Вывод текста

## 32. Графическая библиотека OpenGL. Системы координат. Модельно-видовые преобразования. Примеры

## 33. Графическая библиотека OpenGL. Проекции. Область вывода. Создание анимации. Примеры

#include <windows.h>

#include <gl/glut.h>

#pragma comment(linker, "/ENTRY:main /SUBSYSTEM:WINDOWS")

struct COLOR

{

GLclampf Red, Green, Blue, Alpha;

COLOR(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha = 0)

: Red(red), Green(green), Blue(blue), Alpha(alpha) {}

};

enum DRAW\_STATE { DRAW\_STATE\_PYRAMID, DRAW\_STATE\_PYRAMID\_PLUS };

DRAW\_STATE g\_DrawState = DRAW\_STATE\_PYRAMID;

GLint

g\_InitWidth = 800, // Стартовая ширина окна

g\_InitHeight = 600; // Стартовая высота окна

GLdouble // Размерности ортографической проекции

g\_Left = -10., g\_Right = 10., g\_Bottom = -10., g\_Top = 10., g\_Znear = -10., g\_Zfar = 10.;

GLfloat // Размерности пирамиды (внутри орто-проекции)

g\_X0 = -5., g\_X1 = 5., g\_Y0 = -5., g\_Y1 = 5., g\_Z0 = 0., g\_Z1 = 9.;

GLfloat

g\_AngleX = -10., // Угол для поворота вокруг оси X, град

g\_AngleY = 30., // Угол для поворота вокруг оси Y, град

g\_AngelStep = 5., // Шаг изменеия угла, град

g\_LineWidth = 1.; // Толщина линий

bool g\_IsPress = false;

GLfloat

g\_RotateAnimationStep = 0.2;

void OnDisplay();

void OnWindowsResize(GLint newWidth, GLint newHeight);

void OnKeyPress(unsigned char key, int x, int y);

void OnSpecialKeyPress(int key, int x, int y);

void OnMouse(int button, int state, int x, int y);

void OnIdle();

// Рисовать пирамиду (E)ABCD, где A(x0, y0, z0),

// B(x1, y0, z0),

// C(x1, y1, z0),

// D(x0, y1, z0),

// E( 0, 0, z1)

void DrawPyramid(GLfloat x0, GLfloat x1, GLfloat y0, GLfloat y1, GLfloat z0, GLfloat z1);

void main() // Entry point

{

glutInitDisplayMode(GLUT\_RGB | GLUT\_SINGLE | GLUT\_DEPTH);

glutInitWindowSize(g\_InitWidth, g\_InitHeight);

glutInitWindowPosition(

(GetSystemMetrics(SM\_CXSCREEN) - g\_InitWidth ) / 2,

(GetSystemMetrics(SM\_CYSCREEN) - g\_InitHeight) / 2);

glutCreateWindow("Удаление нелицевых граней (F2), анимация (LeftMouseButton) на OpenGL");

glutDisplayFunc(OnDisplay);

glutReshapeFunc(OnWindowsResize);

glutKeyboardFunc(OnKeyPress);

glutSpecialFunc(OnSpecialKeyPress);

glutMouseFunc(OnMouse);

glutIdleFunc(OnIdle);

glutMainLoop();

}

void OnIdle()

{

if (g\_IsPress)

{

g\_AngleY += g\_RotateAnimationStep;

if (g\_AngleY > 359.99)

g\_AngleY = g\_RotateAnimationStep;

}

glutPostRedisplay();

}

void OnMouse(int button, int state, int x, int y)

{

switch (button)

{

case 0:

g\_IsPress = true;

break;

case 2:

g\_IsPress = false;

break;

}

glutPostRedisplay();

}

void OnDisplay()

{

COLOR

backColor(1., 1., 1.), // Цвет фона RGB

lineColor(0., 0., 0.); // Цвет линий RGB

glClearColor(backColor.Red, backColor.Green, backColor.Blue, backColor.Alpha);

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3ub(lineColor.Red, lineColor.Green, lineColor.Blue);

glLineWidth(g\_LineWidth);

glMatrixMode(GL\_MODELVIEW); // Видовая матрица

glLoadIdentity(); // Видовая матрица M = I

glRotatef(g\_AngleY, 0.0, 1.0, 0.0); // Вокруг оси Y

glRotatef(g\_AngleX, 1.0, 0.0, 0.0); // Вокруг оси X

glMatrixMode(GL\_PROJECTION); // Ортографическая проекция

glLoadIdentity();

glOrtho(g\_Left, g\_Right, g\_Bottom, g\_Top, g\_Znear, g\_Zfar);

switch(g\_DrawState)

{

case DRAW\_STATE\_PYRAMID:

glDisable(GL\_CULL\_FACE);

break;

case DRAW\_STATE\_PYRAMID\_PLUS:

glEnable(GL\_CULL\_FACE);

glCullFace(GL\_BACK);

break;

}

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);

DrawPyramid(g\_X0, g\_X1, g\_Y0, g\_Y1, g\_Z0, g\_Z1);

glFinish();

}

void OnWindowsResize(GLint newWidth, GLint newHeight)

{

glViewport(0, 0, newWidth, newHeight);

}

void OnKeyPress(unsigned char key, int x, int y)

{

static const unsigned char esc\_key = '\033';

if (key == esc\_key)

exit(0);

}

void OnSpecialKeyPress(int key, int x, int y)

{

switch(key)

{

case GLUT\_KEY\_F1:

g\_DrawState = DRAW\_STATE\_PYRAMID;

break;

case GLUT\_KEY\_F2:

g\_DrawState = DRAW\_STATE\_PYRAMID\_PLUS;

break;

case GLUT\_KEY\_UP:

g\_AngleX -= g\_AngelStep;

break;

case GLUT\_KEY\_DOWN:

g\_AngleX += g\_AngelStep;

break;

case GLUT\_KEY\_LEFT:

g\_AngleY -= g\_AngelStep;

break;

case GLUT\_KEY\_RIGHT:

g\_AngleY += g\_AngelStep;

break;

}

glutPostRedisplay();

}

void DrawPyramid(GLfloat x0, GLfloat x1, GLfloat y0, GLfloat y1, GLfloat z0, GLfloat z1)

{

glBegin(GL\_POLYGON); // ABCD

glVertex3f(x1, y0, z0); // D

glVertex3f(x0, y0, z0); // A

glVertex3f(x0, y1, z0); // B

glVertex3f(x1, y1, z0); // C

glEnd();

glBegin(GL\_POLYGON); // EDC

glVertex3f(0., 0., z1); // E

glVertex3f(x1, y0, z0); // D

glVertex3f(x1, y1, z0); // C

glEnd();

glBegin(GL\_POLYGON); // ECB

glVertex3f(0., 0., z1); // E

glVertex3f(x1, y1, z0); // C

glVertex3f(x0, y1, z0); // B

glEnd();

glBegin(GL\_POLYGON); // EBA

glVertex3f(0., 0., z1); // E

glVertex3f(x0, y1, z0); // B

glVertex3f(x0, y0, z0); // A

glEnd();

glBegin(GL\_POLYGON); // EAD

glVertex3f(0., 0., z1); // E

glVertex3f(x0, y0, z0); // A

glVertex3f(x1, y0, z0); // D

glEnd();

}

## 34. Графическая библиотека OpenGL. Материалы и освещение. Примеры

#include <cmath>

#include <cstring>

#include <gl/glut.h>

struct POINT

{

double X;

double Y;

double Z;

POINT(double x, double y, double z) : X(x), Y(y), Z(z) {}

};

GLint g\_InitWidth = 800; // Начальная ширина окна

GLint g\_InitHeight = 600; // Начальная высота окна

GLdouble // Размерности ортографической проекции

g\_Left = -10., g\_Right = 10., g\_Bottom = -10., g\_Top = 10., g\_Znear = -10., g\_Zfar = 10.;

GLfloat // Размерности пирамиды (внутри орто-проекции)

g\_X0 = -5., g\_X1 = 5., g\_Y0 = -5., g\_Y1 = 5., g\_Z0 = 0., g\_Z1 = 9.;

GLint left, right, top, bottom, znear, zfar; // Для определения области видимости графических

// данных после проектирования в СКН

float g\_AngleX = 0; // Угол для поворота вокруг оси X, град

float g\_AngleY = 0; // Угол для поворота вокруг оси Y, град

float d\_angle = 10; // Шаг изменеия угла, град

// Параметры материала пирамиды

float mat3\_dif[] = {0.1f, 0.5f, 0.0f};

float mat3\_amb[] = {0.2f, 0.2f, 0.2f};

float mat3\_spec[] = {0.6f, 0.6f, 0.6f};

float mat3\_shininess = 0.1f \* 128;

void GetNormal(const POINT &P1, const POINT &P2, const POINT &P3, GLfloat \*VN)

// Вычисляет единичный вектор нормали к плоскости,определяемой

// точками P1,P2,P3(P=(P.x,P.y,P.z)

// Вектора P1P2, P1P3 и VN образуют правую тройку

// Результат записывается в вектор VN[3]

{

double V1[3] = { P2.X - P1.X, P2.Y - P1.Y, P2.Z - P1.Z };

double V2[3] = { P3.X - P1.X, P3.Y - P1.Y, P3.Z - P1.Z };

double x = V1[1] \* V2[2] - V1[2] \* V2[1];

double y = -(V1[0] \* V2[2] - V1[2] \* V2[0]);

double z = V1[0] \* V2[1] - V1[1] \* V2[0];

double r = sqrt(x\*x + y\*y + z\*z);

VN[0] = x/r;

VN[1] = y/r;

VN[2] = z/r;

}

void Pyramid(GLfloat x1, GLfloat x2, GLfloat y1, GLfloat y2, GLfloat z1, GLfloat z2)

{

// Основание ABCD: A(x1,y1,z1), B(x1,y2,z1), C(x2,y2,z1), D(x2,y1,z1). Вершина E(0,0,z2)

GLfloat VN[3]; // Вектор нормали

POINT A(x1, y1, z1), B(x1, y2, z1), C(x2, y2, z1), D(x2, y1, z1), E(0, 0, z2);

GetNormal(A,B,D,VN); // Внешняяя нормаль к основанию

glBegin(GL\_POLYGON); // Основание

glNormal3fv(VN);

glVertex3f(x2, y1, z1); // D

glVertex3f(x1, y1, z1); // A

glVertex3f(x1, y2, z1); // B

glVertex3f(x2, y2, z1); // C

glEnd();

GetNormal(E, D, C,VN); // Внешняяя нормаль к EDC

glBegin(GL\_POLYGON); // EDC

glNormal3fv(VN);

glVertex3f(0., 0., z2); // E

glVertex3f(x2, y1, z1); // D

glVertex3f(x2, y2, z1); // C

glEnd ();

GetNormal(E, C, B, VN); // Внешняяя нормаль к ECB

glBegin(GL\_POLYGON); // ECB

glNormal3fv(VN);

glVertex3f(0., 0., z2); // E

glVertex3f(x2, y2, z1); // C

glVertex3f(x1, y2, z1); // B

glEnd();

GetNormal(E, B, A, VN); // Внешняяя нормаль к EBA

glBegin(GL\_POLYGON); // EBA

glNormal3fv(VN);

glVertex3f(0., 0., z2); // E

glVertex3f(x1, y2, z1); // B

glVertex3f(x1, y1, z1); // A

glEnd();

GetNormal(E, A, D, VN); // Внешняяя нормаль к EAD

glBegin(GL\_POLYGON); // EAD

glNormal3fv(VN);

glVertex3f(0., 0., z2); // E

glVertex3f(x1, y1, z1); // A

glVertex3f(x2, y1, z1); // D

glEnd();

}

void OnDraw(void)

{

glClearColor(1.0, 1.0, 1.0, 0.0); // Фон

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT ); // Заливка

glMatrixMode(GL\_MODELVIEW); // Видовая матрица

glLoadIdentity();

glRotatef(g\_AngleY, 0.0, 1.0, 0.0); // Вокруг оси Y, M = Ry(angleY)

glRotatef(g\_AngleX, 1.0, 0.0, 0.0); // Вокруг оси X, M = Ry(angleY) \* Rx(angleX)

// Результат V2 = Ry\*Rx\*V1

glMatrixMode(GL\_PROJECTION); // Ортографическая проекция

glLoadIdentity();

glOrtho(g\_Left, g\_Right, g\_Bottom, g\_Top, g\_Znear, g\_Zfar);

glEnable(GL\_CULL\_FACE);

glCullFace(GL\_BACK);

glMaterialfv(GL\_FRONT,GL\_AMBIENT,mat3\_amb);

glMaterialfv(GL\_FRONT,GL\_DIFFUSE,mat3\_dif);

glMaterialfv(GL\_FRONT,GL\_SPECULAR,mat3\_spec);

glMaterialf(GL\_FRONT,GL\_SHININESS,mat3\_shininess);

Pyramid(g\_X0, g\_X1, g\_Y0, g\_Y1, g\_Z0, g\_Z1);

glutSwapBuffers(); // Переключение буферов (обязательно при использовании двойной буферизации GLUT\_DOUBLE)

glFinish();

}

void OnWindowsResize(GLint newWidth, GLint newHeight)

{

glViewport(0, 0, newWidth, newHeight);

}

void OnKeyPress(unsigned char key, int x, int y)

{

static const unsigned char esc\_key = '\033';

if (key == esc\_key)

exit(0);

}

void OnSpecialKeyPress(int key, int x, int y)

{

switch(key)

{

case GLUT\_KEY\_RIGHT:

g\_AngleY += d\_angle;

break;

case GLUT\_KEY\_LEFT:

g\_AngleY -= d\_angle;

break;

case GLUT\_KEY\_UP:

g\_AngleX -= d\_angle;

break;

case GLUT\_KEY\_DOWN:

g\_AngleX += d\_angle;

break;

}

glutPostRedisplay();

}

void Init() // Инициализация параметров материалов и источника света

{

GLfloat light\_ambient[] = { 0., 0., 0., 1. };

GLfloat light\_diffuse[] = { 1., 1., 1., 1. };

GLfloat light\_specular[] = { 1., 1., 1., 1. };

GLfloat light\_position[] = { 0., 0., 1., 0. };

glLightfv(GL\_LIGHT0, GL\_AMBIENT, light\_ambient);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, light\_diffuse);

glLightfv(GL\_LIGHT0, GL\_SPECULAR, light\_specular);

glLightfv(GL\_LIGHT0, GL\_POSITION, light\_position);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

}

void main() // Entry point

{

glutInitDisplayMode(GLUT\_RGB | GLUT\_SINGLE | GLUT\_DEPTH);

glutInitWindowSize(g\_InitWidth, g\_InitHeight);

glutCreateWindow("Пирамида с освещением");

Init();

glutDisplayFunc(OnDraw);

glutReshapeFunc(OnWindowsResize);

glutKeyboardFunc(OnKeyPress);

glutSpecialFunc(OnSpecialKeyPress);

glutMainLoop();

}

## 35. Графическая библиотека OpenGL. Структура приложения с использованием библиотеки MFC

#include "gl/gl.h"

#include "gl/glu.h"

unsigned char threeto8[8] = { 0, 0111>>1, 0222>>1, 0333>>1, 0444>>1, 0555>>1, 0666>>1, 0377 };

unsigned char twoto8[4] = { 0, 0x55, 0xaa, 0xff };

unsigned char oneto8[2] = { 0, 255 };

IMPLEMENT\_DYNCREATE(CCubeView, CView)

BEGIN\_MESSAGE\_MAP(CCubeView, CView)

//{{AFX\_MSG\_MAP(CCubeView)

ON\_COMMAND(ID\_FILE\_PLAY, OnFilePlay)

ON\_UPDATE\_COMMAND\_UI(ID\_FILE\_PLAY, OnUpdateFilePlay)

ON\_WM\_CREATE()

ON\_WM\_DESTROY()

ON\_WM\_SIZE()

ON\_WM\_TIMER()

ON\_WM\_ERASEBKGND()

//}}AFX\_MSG\_MAP

END\_MESSAGE\_MAP()

void CCubeView::OnDraw(CDC\* /\*pDC\*/)

{

CCubeDoc\* pDoc = GetDocument();

ASSERT\_VALID(pDoc);

DrawScene();

}

void CCubeView::OnFilePlay()

{

m\_play = m\_play ? FALSE : TRUE;

if (m\_play)

SetTimer(1, 15, NULL);

else

KillTimer(1);

}

void CCubeView::OnUpdateFilePlay(CCmdUI\* pCmdUI)

{

pCmdUI->SetCheck(m\_play);

}

BOOL CCubeView::PreCreateWindow(CREATESTRUCT& cs)

{

// An OpenGL window must be created with the following flags and must not

// include CS\_PARENTDC for the class style. Refer to SetPixelFormat

// documentation in the "Comments" section for further information.

cs.style |= WS\_CLIPSIBLINGS | WS\_CLIPCHILDREN;

return CView::PreCreateWindow(cs);

}

int CCubeView::OnCreate(LPCREATESTRUCT lpCreateStruct)

{

if (CView::OnCreate(lpCreateStruct) == -1)

return -1;

Init(); // initialize OpenGL

return 0;

}

void CCubeView::OnDestroy()

{

HGLRC hrc;

KillTimer(1);

hrc = ::wglGetCurrentContext();

::wglMakeCurrent(NULL, NULL);

if (hrc)

::wglDeleteContext(hrc);

if (m\_pOldPalette)

m\_pDC->SelectPalette(m\_pOldPalette, FALSE);

if (m\_pDC)

delete m\_pDC;

CView::OnDestroy();

}

void CCubeView::OnSize(UINT nType, int cx, int cy)

{

CView::OnSize(nType, cx, cy);

if(cy > 0)

{

glViewport(0, 0, cx, cy);

if((m\_oldRect.right > cx) || (m\_oldRect.bottom > cy))

RedrawWindow();

m\_oldRect.right = cx;

m\_oldRect.bottom = cy;

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(45.0f, (GLdouble)cx/cy, 3.0f, 7.0f);

glMatrixMode(GL\_MODELVIEW);

}

}

void CCubeView::OnTimer(UINT\_PTR nIDEvent)

{

DrawScene();

CView::OnTimer(nIDEvent);

// Eat spurious WM\_TIMER messages

MSG msg;

while(::PeekMessage(&msg, m\_hWnd, WM\_TIMER, WM\_TIMER, PM\_REMOVE));

}

/////////////////////////////////////////////////////////////////////////////

// GL helper functions

void CCubeView::Init()

{

PIXELFORMATDESCRIPTOR pfd;

int n;

HGLRC hrc;

GLfloat fMaxObjSize, fAspect;

GLfloat fNearPlane, fFarPlane;

m\_pDC = new CClientDC(this);

ASSERT(m\_pDC != NULL);

if (!bSetupPixelFormat())

return;

n = ::GetPixelFormat(m\_pDC->GetSafeHdc());

::DescribePixelFormat(m\_pDC->GetSafeHdc(), n, sizeof(pfd), &pfd);

hrc = wglCreateContext(m\_pDC->GetSafeHdc());

wglMakeCurrent(m\_pDC->GetSafeHdc(), hrc);

GetClientRect(&m\_oldRect);

glClearDepth(1.0f);

glEnable(GL\_DEPTH\_TEST);

if (m\_oldRect.bottom)

fAspect = (GLfloat)m\_oldRect.right/m\_oldRect.bottom;

else // don't divide by zero, not that we should ever run into that...

fAspect = 1.0f;

fNearPlane = 3.0f;

fFarPlane = 7.0f;

fMaxObjSize = 3.0f;

m\_fRadius = fNearPlane + fMaxObjSize / 2.0f;

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(45.0f, fAspect, fNearPlane, fFarPlane);

glMatrixMode(GL\_MODELVIEW);

}

BOOL CCubeView::bSetupPixelFormat()

{

static PIXELFORMATDESCRIPTOR pfd =

{

sizeof(PIXELFORMATDESCRIPTOR), // size of this pfd

1, // version number

PFD\_DRAW\_TO\_WINDOW | // support window

PFD\_SUPPORT\_OPENGL | // support OpenGL

PFD\_DOUBLEBUFFER, // double buffered

PFD\_TYPE\_RGBA, // RGBA type

24, // 24-bit color depth

0, 0, 0, 0, 0, 0, // color bits ignored

0, // no alpha buffer

0, // shift bit ignored

0, // no accumulation buffer

0, 0, 0, 0, // accum bits ignored

32, // 32-bit z-buffer

0, // no stencil buffer

0, // no auxiliary buffer

PFD\_MAIN\_PLANE, // main layer

0, // reserved

0, 0, 0 // layer masks ignored

};

int pixelformat;

if ( (pixelformat = ChoosePixelFormat(m\_pDC->GetSafeHdc(), &pfd)) == 0 )

{

MessageBox("ChoosePixelFormat failed");

return FALSE;

}

if (SetPixelFormat(m\_pDC->GetSafeHdc(), pixelformat, &pfd) == FALSE)

{

MessageBox("SetPixelFormat failed");

return FALSE;

}

return TRUE;

}

unsigned char CCubeView::ComponentFromIndex(int i, UINT nbits, UINT shift)

{

unsigned char val;

val = (unsigned char) (i >> shift);

switch (nbits)

{

case 1:

val &= 0x1;

return oneto8[val];

case 2:

val &= 0x3;

return twoto8[val];

case 3:

val &= 0x7;

return threeto8[val];

default:

return 0;

}

}

void CCubeView::DrawScene(void)

{

static BOOL bBusy = FALSE;

static GLfloat wAngleY = 10.0f;

static GLfloat wAngleX = 1.0f;

static GLfloat wAngleZ = 5.0f;

if(bBusy)

return;

bBusy = TRUE;

glClearColor(1.0f, 1.0f, 1.0f, 1.0f);

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glPushMatrix();

glTranslatef(0.0f, 0.0f, -m\_fRadius);

glRotatef(wAngleX, 1.0f, 0.0f, 0.0f);

glRotatef(wAngleY, 0.0f, 1.0f, 0.0f);

glRotatef(wAngleZ, 0.0f, 0.0f, 1.0f);

wAngleX += 1.0f;

wAngleY += 10.0f;

wAngleZ += 5.0f;

GLfloat // Размерности пирамиды (внутри орто-проекции)

x0 = -1., x1 = 1., y0 = -1., y1 = 1., z0 = 0., z1 = 1.5;

glColor3f(0.5f, 0.0f, 1.0f);

glBegin(GL\_POLYGON); // ABCD

glVertex3f(x1, y0, z0); // D

glVertex3f(x0, y0, z0); // A

glVertex3f(x0, y1, z0); // B

glVertex3f(x1, y1, z0); // C

glEnd();

glColor3f(0.5f, 0.0f, 1.0f);

glBegin(GL\_POLYGON); // EDC

glVertex3f(0., 0., z1); // E

glVertex3f(x1, y0, z0); // D

glVertex3f(x1, y1, z0); // C

glEnd();

glColor3f(0.0f, 0.5f, 1.0f);

glBegin(GL\_POLYGON); // ECB

glVertex3f(0., 0., z1); // E

glVertex3f(x1, y1, z0); // C

glVertex3f(x0, y1, z0); // B

glEnd();

glColor3f(0.0f, 0.0f, 0.5f);

glBegin(GL\_POLYGON); // EBA

glVertex3f(0., 0., z1); // E

glVertex3f(x0, y1, z0); // B

glVertex3f(x0, y0, z0); // A

glEnd();

glColor3f(0.0f, 0.5f, 1.0f);

glBegin(GL\_POLYGON); // EAD

glVertex3f(0., 0., z1); // E

glVertex3f(x0, y0, z0); // A

glVertex3f(x1, y0, z0); // D

glEnd();

glPopMatrix();

glFinish();

SwapBuffers(wglGetCurrentDC());

bBusy = FALSE;

}

BOOL CCubeView::OnEraseBkgnd(CDC\*)

{

return TRUE;

}