Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет ИТ

Кафедра ИиВД

Специальность: 1-98 01 03 – «Программное обеспечение информационной безопасности мобильных систем»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

КУРСОВОЙ РАБОТЫ

по дисциплине Компьютерная геометрия и графика

Тема: Приложение Windows «Эллипсоид»

Исполнитель

студент III курса группы 7 Халипов Е. С. (Ф.И.О)

Руководитель

доцент, к. т.н. \_\_\_\_\_\_\_\_\_\_ А. А. Дятко

Курсовой проект защищен с оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель\_\_\_\_\_\_\_\_\_\_ А. А. Дятко

Минск 2019

Оглавление

[Оглавление 2](#_Toc27510966)

[Задание на курсовую работу 3](#_Toc27510967)

[Введение 4](#_Toc27510968)

[Описание алгоритмов решения задачи 5](#_Toc27510969)

[1.Общая схема приложения 5](#_Toc27510970)

[2. Получение и обработка команд 6](#_Toc27510971)

[3.Алгоритмы построения изображения 8](#_Toc27510972)

[Заключение 10](#_Toc27510973)

[Приложения 12](#_Toc27510974)

[Приложение А. Класс ChildView. 12](#_Toc27510975)

[Приложение Б. Класс SizeDialog 16](#_Toc27510976)

[Приложение В. Описание класса CMatrix. 18](#_Toc27510977)

[Приложение Г. Описание класса Ellipsoid. 25](#_Toc27510978)

[Приложение Д. Функции аффинных пеобразованй. 29](#_Toc27510979)

[Список литературы 31](#_Toc27510980)

# Задание на курсовую работу

Создать приложение Windows «Эллипсоид»

Построить 3D-изображение эллипсоида (рис. 1.1).

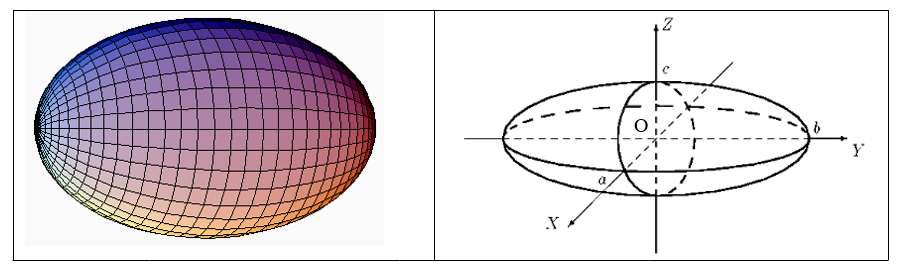


Рисунок 1.1 – 3D-изображение эллипсоида

Уравнение эллипсоида в системе координат (СК) XYZ.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Таблица 1.1 – Уравнение эллипсоида

Для описания поверхности эллипсоида использовать полигональную модель.

Изменяемые параметры модели:

* значение полуосей эллипсоида ;
* положение наблюдателя , в сферической СК с началом в точке О;
* положение источника света , в сферической СК с началом в точке О;
* цвет источника света;
* модель отражения света от поверхности фигуры;

Для установки параметров модели использовать окно диалога.

При изменении размера окна изображения соответствующим образом масштабируется.

Обеспечить запись изображения в графический файл \*.bmp.

# Введение

Компьютерная или машинная графика — это вполне самостоятельная область человеческой деятельности со своими проблемами и спецификой. Компьютерная графика — это и новые эффективные технические средства для проектировщиков, конструкторов и исследователей, и программные системы, и машинные языки, и новые научные, учебные дисциплины, родившиеся на базе синтеза таких наук как аналитическая, прикладная, начертательная геометрия, программирование для ЭВМ, методы вычислительной математики и т. п.

В настоящее время задачи компьютерной графика и других областей требуют применения наглядных мультимедийных средств. Наиболее актуальным их применение является в сфере образования, где такие средства могут значительно улучшить уровень усвоения учениками изучаемого материала.

Неотъемлемым компонентом мультимедийных средств является графическое изображение моделируемого процесса. Для современного процесса обучения уже недостаточно просто изобразить модель в виде двумерного изображения (например, в задаче из учебника по математике). Для реализации возможности динамического редактирования 3D-моделей можно использовать математический аппарат компьютерной геометрии и графики.

Ключевыми элементами компьютерных моделей являются точки. Они используются для построения всех моделируемых объектов. Важной задачей является перемещение, поворот и растяжение отдельных объектов относительно друг друга. В математическом аппарате компьютерной геометрии и графики это выполняется при помощи перемещения точек объектов.

Полигональная сетка — это совокупность вершин, рёбер и граней, которые определяют форму многогранного объекта в трёхмерной компьютерной графике и объёмном моделировании. Гранями обычно являются треугольники, четырёхугольники или другие простые выпуклые многоугольники (полигоны), так как это упрощает рендеринг, но сетки могут также состоять и из наиболее общих вогнутых многоугольников, или многоугольников с отверстиями.

Решение данных задач предстоит и при выполнении поставленного задания. Для этого будет использоваться платформа MFC, предоставляющая набор классов для реализации оконного приложения, эффективно взаимодействующего с пользователем при помощи периферийных устройств, оконного интерфейса, меню.

# Описание алгоритмов решения задачи

# 1.Общая схема приложения

Для решения поставленных задач было создано приложение MFC, построенное по следующей схеме:

Начало

Установка параметров в начальные значения, создание окна, запуск обработчиков сообщений

Получение и обработка команд, прорисовка изображения с учетом поступающих параметров

Поступила команда «Выход»

Завершение обработки сообщений, закрытие окон, освобождение памяти

Конец

нет

да

Создание окна и запуск обработчиков сообщений выполняются по стандарту платформы MFC для приложений без использования архитектуры «Документ/Вид». Для хранения и передачи параметров был доработан класс ChildView (приложение А). Завершение работы приложения выполняется также по стандарту платформы MFC.

# 2. Получение и обработка команд

Цикл получения и обработки команд можно изобразить следующим образом:

Начало

Конец

Прорисовка эллипсоида

Вызов справки из меню

Команда меню «Выход»

Отображение справочного окна

Изменение положения наблюдателя и источника света нажатием клавиш

Установка цвета света источника и размеров в эллипсоиде в диалоге

Изменение типа освещения нажатием клавиш

При помощи обработчика сообщений от клавиатуры реализованы следующие реакции:

* поворот наблюдателя при помощи стрелок;
* поворот источника света при помощи клавиш w, a, s, d;
* изменение типа освещения при помощи клавиш o, p;

Кроме того, реакции вызова диалога настройки параметров, выхода, сохранения файла доступны в меню.

Через пункты меню «Commands» доступен вызов диалогов настройки цвета освещения, размеров эллипсоида. Обработка действий отдельных элементов управления реализована при помощи механизма DoDataExchange, являющегося полезным инструментом взаимодействия приложения на платформе MFC с пользователем (приложение Б). Он служит для реализации механизмов автоматического обмена данными и автоматической проверки данных.

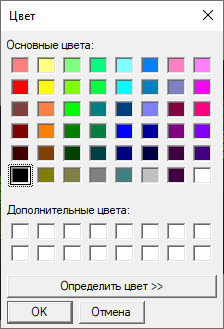


Рисунок 3.1 – диалог выбора цвета освещения

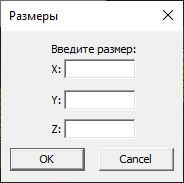


Рисунок 3.2 – диалог установки размера эллипсоида

# 3.Алгоритмы построения изображения

Одним из ключевых моментов решения поставленных задач является реализация координат точек в виде векторов в соответствии с математическим аппаратом компьютерной геометрии и графики. Для двухмерного пространства точка представляется в виде вектора из 3-ех элементов: двух координат и единицы; для трехмерного – трех координат и единицы. Это позволяет свести преобразования координат к умножению исходного вектора на соответствующую матрицу. Для работы с векторами и матрицами использовался класс CMatrix (приложение В).

Для решения задачи перемещения и поворота объектов в методах прорисовки эллипсоида с диффузионной и зеркальной моделью освещения были реализованы аффинные преобразования в соответствии с математическим аппаратом компьютерной геометрии и графики (приложение Г).

Для решения задачи изображения трехмерного процесса на двухмерном дисплее с учетом положения наблюдателя использовались матрицы пересчета из мировых координат в видовые и из видовых – в оконные. Расчет данных матриц выполнялся в соответствии с математическим аппаратом компьютерной геометрии и графики. Матрица пересчета из мировых координат в оконные формируется функцией CreateViewCoord, а матрица пересчета из видовых координат в оконные – функцией SpaceToWindow. Определения функций представлены в приложении Д.

Положение наблюдателя и источника света удобнее изменять в сферической системе координат при помощи стрелок и клавиш WASD. Но в расчетах в соответствии с математическим аппаратом компьютерной геометрии и графики требуется использовать координаты в декартовой системе. Поэтому был реализован метод перевода из сферических координат в декартовы, математического аппарата компьютерной геометрии и графики. В диалогах регуляция положения наблюдателя и источника света не дублировалась.

Процесс прорисовки эллипсоида в окне приложения демонстрируется следующей блок-схемой:

Конец

Расчет матриц пересчета из мировой системы координат (МСК) в видовую и из видовой в оконную (ОСК)

Расположение точек эллипсоида в соответствии с параметрическими уравнениями

Порядок прорисовки полигонов эллипсоида «на наблюдателя»

Получение точек эллипсоида в контексте окна

Начало

Вычисление положения источника света по заданным координатам в сферической системе координат, пересчет в оконные координаты, изображение источника света на экране

Прорисовка эллипсоида

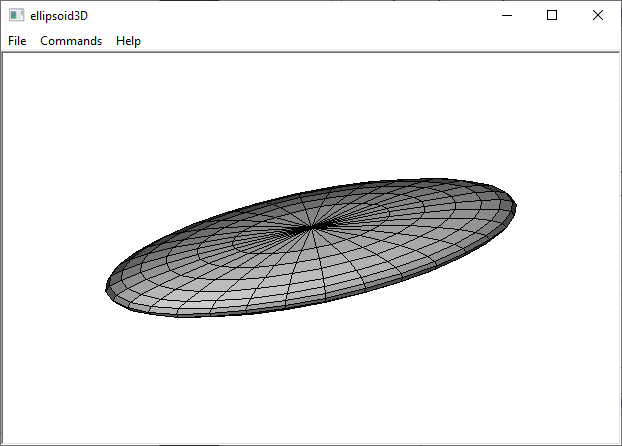
# Заключение

Разработанное приложение позволяет моделировать трехмерный эллипсоид с использованием математического аппарата геометрии и графики. Поддерживается смена типа освещения (диффузное или зеркальное) и его цвета.

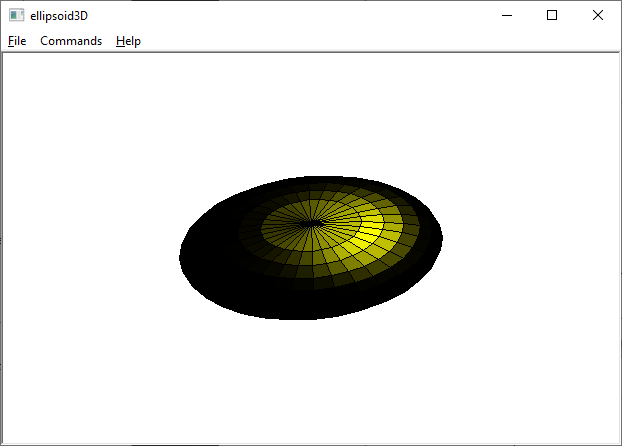
В дальнейшем возможно усовершенствование приложения путем использования аппаратных средств пересчета координат и алгоритмов из оптимизированных библиотек, движения в пространстве.

Получены навыки практического использования пакета MFC, применения математического аппарата компьютерной геометрии и графики.

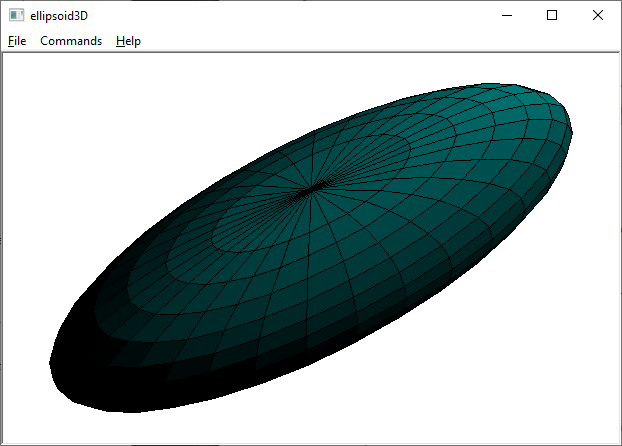
Демонстрация работы приложения:



а)



б)



в)

Рисунок 4 – демонстрация работы приложения

# Приложения

Приложение А. Класс ChildView.

// ChildView.cpp : implementation of the CChildView class

#include "stdafx.h"

#include "ellipsoid3D.h"

#include "ChildVie.h"

#include "SizeDialog.h"

#ifdef \_DEBUG

#define new DEBUG\_NEW

#endif

// CChildView

CChildView::CChildView( ) {

step = 10;

m = 0;

PView.RedimMatrix( 3 );

//начальное положение наблюдателя

PView( 0 ) = 1; PView( 1 ) = 315; PView( 2 ) = 45;

PLight.RedimMatrix( 3 );

//начальное положение источника света

PLight( 0 ) = 1; PLight( 1 ) = 315; PLight( 2 ) = 45;

pp = new CEllipsoid( );

//флаги нажатия мыши и выделения рамкой

m\_IsMousePressed = 0;

m\_IsSelectedRect = 0;

m\_SelectedRect = new CRect( );

//перо для прорисовки рамки выделения

pen.CreatePen( PS\_DOT, 1, RGB( 255, 0, 0 ) );

}

CChildView::~CChildView( ) { }

BEGIN\_MESSAGE\_MAP( CChildView, CWnd )

ON\_WM\_PAINT( )

ON\_WM\_KEYDOWN( )

ON\_WM\_LBUTTONDOWN( )

ON\_WM\_LBUTTONUP( )

ON\_WM\_MOUSEMOVE( )

ON\_COMMAND( ID\_COMMANDS\_SETCOLOR, &CChildView::OnSetcolor )

ON\_COMMAND( ID\_COMMANDS\_SETSIZE, &CChildView::OnSetsize )

ON\_COMMAND( ID\_FILE\_SAVE\_AS, &CChildView::OnSave1 )

ON\_WM\_SIZE( )

END\_MESSAGE\_MAP( )

// CChildView message handlers

BOOL CChildView::PreCreateWindow( CREATESTRUCT& cs ) {

if ( !CWnd::PreCreateWindow( cs ) )

return FALSE;

cs.dwExStyle |= WS\_EX\_CLIENTEDGE;

cs.style &= ~WS\_BORDER;

cs.lpszClass = AfxRegisterWndClass( CS\_HREDRAW | CS\_VREDRAW | CS\_DBLCLKS,

::LoadCursor( NULL, IDC\_ARROW ), reinterpret\_cast<HBRUSH>( COLOR\_WINDOW + 1 ), NULL );

return TRUE;

}

void CChildView::OnPaint( ) {

CPaintDC dc( this );

//прорисовка эллипсоида

pp->DrawEnlighted( dc, PView, PLight, rw, col, m );

//прорисовки рамки выделения

if ( m\_IsSelectedRect ) {

dc.SelectObject( pen );

dc.MoveTo( m\_SelectedRect->left, m\_SelectedRect->top );

dc.LineTo( m\_SelectedRect->right, m\_SelectedRect->top );

dc.LineTo( m\_SelectedRect->right, m\_SelectedRect->bottom );

dc.LineTo( m\_SelectedRect->left, m\_SelectedRect->bottom );

dc.LineTo( m\_SelectedRect->left, m\_SelectedRect->top );

}

}

void CChildView::OnLButtonDown( UINT var, CPoint point ) {

//начало выделения области мышью

p1 = point;

m\_IsSelectedRect = m\_IsMousePressed = true;

}

void CChildView::OnLButtonUp( UINT var, CPoint p2 ) {

//окончание выделения области мышью

m\_IsMousePressed = 0;

m\_SelectedRect = new CRect( min( p1.x, p2.x ), min( p1.y, p2.y ), max( p1.x, p2.x ), max( p1.y, p2.y ) );

}

void CChildView::OnMouseMove( UINT var, CPoint p2 ) {

//прорисовка рамки при выделении

if ( m\_IsSelectedRect && m\_IsMousePressed ) {

m\_SelectedRect = new CRect( min( p1.x, p2.x ), min( p1.y, p2.y ), max( p1.x, p2.x ), max( p1.y, p2.y ) );

Invalidate( 1 );

}

}

void CChildView::OnKeyDown( UINT nChar, UINT nRepCount, UINT nFlags ) {

//поворот наблюдателя

if ( nChar == 37 ) { //влево

PView( 1 ) += step;

if ( PView( 1 ) > 360 )

PView( 1 ) -= 360;

Invalidate( );

}

else if ( nChar == 39 ) { //вправо

PView( 1 ) -= step;

if ( PView( 1 ) < 0 )

PView( 1 ) += 360;

Invalidate( );

}

else if ( nChar == 38 ) { //вверх

PView( 2 ) += step;

if ( PView( 2 ) > 360 )

PView( 2 ) -= 360;

Invalidate( );

}

else if ( nChar == 40 ) { //вниз

PView( 2 ) -= step;

if ( PView( 2 ) < 0 )

PView( 2 ) += 360;

Invalidate( );

}

//поворот источника света

else if ( nChar == 65 ) { //A

PLight( 1 ) -= step;

if ( PLight( 1 ) < 0 )

PLight( 1 ) += 360;

Invalidate( );

}

else if ( nChar == 68 ) { //D

PLight( 1 ) += step;

if ( PLight( 1 ) > 360 )

PLight( 1 ) -= 360;

Invalidate( );

}

else if ( nChar == 87 ) { //W

PLight( 2 ) -= step;

if ( PLight( 2 ) < 0 )

PLight( 2 ) += 360;

Invalidate( );

}

else if ( nChar == 83 ) { //S

PLight( 2 ) += step;

if ( PLight( 2 ) > 360 )

PLight( 2 ) -= 360;

Invalidate( );

}

//переключение модели освещения

else if ( nChar == 79 ) { //O

m = 1;

Invalidate( );

}

else if ( nChar == 80 ) { //P

m = 0;

Invalidate( );

}

}

//диалог установки цвета эллипсоида

void CChildView::OnSetcolor( ) {

CColorDialog ccd;

if ( ccd.DoModal( ) == IDOK ) {

col = ccd.GetColor( );

Invalidate( );

}

}

//диалоги установки размеров эллипсоида

void CChildView::OnSetsize( ) {

SizeDialog sd;

if ( sd.DoModal( ) == IDOK ) {

int a = sd.GetX( ), b = sd.GetY( ), c = sd.GetZ( );

pp = new CEllipsoid( a, b, c );

Invalidate( );

}

}

//диалог сохранения выделенной области в файл

void CChildView::OnSave1( ) {

CFileDialog fOpenDlg( FALSE, \_T( "fif" ), 0, NULL, \_T( "Image File (\*.bmp)|\*.bmp|" ), this );

// заголовок диалога

fOpenDlg.m\_pOFN->lpstrTitle = \_T( "Select BMP File" );

// отображение диалога

if ( fOpenDlg.DoModal( ) == IDOK ) {

m\_saved\_file = fOpenDlg.GetPathName( );

Invalidate( );

Sleep( 1000 );

ClientToBmp( GetSafeHwnd( ), m\_saved\_file, \*m\_SelectedRect );

}

m\_IsSelectedRect = false;

}

//сохранение выделенной области в файл

int ClientToBmp( HWND hWnd, const char\* filepath, CRect r ) {

//открытие фвйла для записи

HANDLE fh = CreateFile( filepath, GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL | FILE\_FLAG\_SEQUENTIAL\_SCAN, NULL );

if ( fh == INVALID\_HANDLE\_VALUE )

return 2;

//информационный заголовок

BITMAPINFOHEADER bi;

ZeroMemory( &bi, sizeof( BITMAPINFOHEADER ) );

bi.biSize = sizeof( BITMAPINFOHEADER );

bi.biWidth = r.right - r.left;

bi.biHeight = r.bottom - r.top;

bi.biPlanes = 1;

bi.biBitCount = 24; //16;

bi.biSizeImage = ( bi.biWidth \* bi.biBitCount + 31 ) / 32 \* 4 \* bi.biHeight;

//файловый заголовок

BITMAPFILEHEADER bmfHdr;

ZeroMemory( &bmfHdr, sizeof( BITMAPFILEHEADER ) );

bmfHdr.bfType = ( 'M' << 8 ) | 'B'; //0x4D42; //BM

bmfHdr.bfSize = bi.biSizeImage + sizeof( BITMAPFILEHEADER ) + bi.biSize;

bmfHdr.bfReserved1 = bmfHdr.bfReserved2 = 0;

bmfHdr.bfOffBits = ( DWORD )sizeof( BITMAPFILEHEADER ) + (DWORD)bi.biSize;

//получение в hBitmap выделенной области изображения

HDC hDC = GetDC( hWnd );

HDC hDCMem = CreateCompatibleDC( hDC );

HBITMAP hBitmap = CreateCompatibleBitmap( hDC, bi.biWidth, bi.biHeight );

HBITMAP oldBitmap = (HBITMAP)SelectObject( hDCMem, hBitmap );

BitBlt( hDCMem, 0, 0, bi.biWidth, bi.biHeight, hDC, r.left, r.top, SRCCOPY );

hBitmap = (HBITMAP)SelectObject( hDCMem, oldBitmap );

HANDLE hDIB = GlobalAlloc( GHND, bi.biSizeImage );

char\* lp = (char\*)GlobalLock( hDIB );

//получение изображения в аппаратно-независимом формате

GetDIBits( hDC, hBitmap, 0, bi.biHeight, lp, (LPBITMAPINFO)& bi, DIB\_RGB\_COLORS );

DWORD dwWritten = sizeof( BITMAPFILEHEADER );

//запись файлового заголовка

WriteFile( fh, (LPSTR)& bmfHdr, sizeof( BITMAPFILEHEADER ), &dwWritten, NULL );

dwWritten = sizeof( BITMAPINFOHEADER );

//запись информационного заголовка

WriteFile( fh, (LPSTR)& bi, sizeof( BITMAPINFOHEADER ), &dwWritten, NULL );

dwWritten = bi.biSizeImage;

//запись изображения в аппаратно-независимом формате

WriteFile( fh, lp, bi.biSizeImage, &dwWritten, NULL );

GlobalUnlock( hDIB );

GlobalFree( hDIB );

DeleteObject( hBitmap );

lp = NULL;

CloseHandle( fh );

ReleaseDC( hWnd, hDCMem );

ReleaseDC( hWnd, hDC );

DeleteDC( hDCMem );

DeleteDC( hDC );

if ( dwWritten == 2 ) return 2;

return 0;

}

void CChildView::OnSize( UINT nType, int cx, int cy ) {

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

rw.SetRect( 100, 100, cx - 100, cy - 100 );

}

Приложение Б. Класс SizeDialog

// SizeDialog.cpp : implementation file

#include "stdafx.h"

#include "SizeDialog.h"

// SizeDialog dialog

IMPLEMENT\_DYNAMIC( SizeDialog, CDialog )

SizeDialog::SizeDialog( CWnd\* pParent /\*=NULL\*/ )

: CDialog( SizeDialog::IDD, pParent ) {

EnableAutomation( );

}

SizeDialog::~SizeDialog( ) { }

void SizeDialog::OnFinalRelease( ) {

// When the last reference for an automation object is released

// OnFinalRelease is called. The base class will automatically

// deletes the object. Add additional cleanup required for your

// object before calling the base class.

CDialog::OnFinalRelease( );

}

void SizeDialog::DoDataExchange( CDataExchange\* pDX ) {

CDialog::DoDataExchange( pDX );

DDX\_Control( pDX, IDC\_EDIT1, xedit );

DDX\_Control( pDX, IDC\_EDIT3, yedit );

DDX\_Control( pDX, IDC\_EDIT2, zedit );

}

BEGIN\_MESSAGE\_MAP( SizeDialog, CDialog )

ON\_BN\_CLICKED( IDOK, &SizeDialog::OnBnClickedOk )

ON\_EN\_CHANGE( IDC\_EDIT1, &SizeDialog::OnEnChangeEdit1 )

ON\_EN\_CHANGE( IDC\_EDIT3, &SizeDialog::OnEnChangeEdit3 )

ON\_EN\_CHANGE( IDC\_EDIT2, &SizeDialog::OnEnChangeEdit2 )

END\_MESSAGE\_MAP( )

BEGIN\_DISPATCH\_MAP( SizeDialog, CDialog )

END\_DISPATCH\_MAP( )

// Note: we add support for IID\_ISizeDialog to support typesafe binding

// from VBA. This IID must match the GUID that is attached to the

// dispinterface in the .IDL file.

// {140C0F47-65E4-4C40-9F5E-2B44453B3970}

static const IID IID\_ISizeDialog =

{ 0x140C0F47, 0x65E4, 0x4C40, { 0x9F, 0x5E, 0x2B, 0x44, 0x45, 0x3B, 0x39, 0x70 } };

BEGIN\_INTERFACE\_MAP( SizeDialog, CDialog )

INTERFACE\_PART( SizeDialog, IID\_ISizeDialog, Dispatch )

END\_INTERFACE\_MAP( )

// SizeDialog message handlers

void SizeDialog::OnBnClickedOk( ) {

// TODO: Add your control notification handler code here

OnOK( );

}

void SizeDialog::OnEnChangeEdit1( ) {

CString\* s = new CString( );

xedit.GetWindowText( \*s );

a = atoi( s->GetBuffer( ) );

}

void SizeDialog::OnEnChangeEdit3( ) {

CString\* s = new CString( );

yedit.GetWindowText( \*s );

b = atoi( s->GetBuffer( ) );

}

void SizeDialog::OnEnChangeEdit2( ) {

CString\* s = new CString( );

zedit.GetWindowText( \*s );

c = atoi( s->GetBuffer( ) );

}

int SizeDialog::GetX( ) {

return a;

}

int SizeDialog::GetY( ) {

return b;

}

int SizeDialog::GetZ( ) {

return c;

}

Приложение В. Описание класса CMatrix.

#include "stdafx.h"

#include "CMatrix.h"

CMatrix::CMatrix( ) {

n\_rows = 1;

n\_cols = 1;

array = new double\* [ n\_rows ];

for ( int i = 0; i < n\_rows; i++ ) array[ i ] = new double[ n\_cols ];

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

for ( int j = 0; j < n\_cols; j++ ) array[ i ][ j ] = 0;

}

//-------------------------------------------------------------------------------

CMatrix::CMatrix( int Nrow, int Ncol )

// Nrow - число строк

// Ncol - число столбцов

{

n\_rows = Nrow;

n\_cols = Ncol;

array = new double\* [ n\_rows ];

for ( int i = 0; i < n\_rows; i++ ) array[ i ] = new double[ n\_cols ];

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

for ( int j = 0; j < n\_cols; j++ ) array[ i ][ j ] = 0;

}

//---------------------------------------------------------------------------------

CMatrix::CMatrix( int Nrow ) //Вектор

// Nrow - число строк

{

n\_rows = Nrow;

n\_cols = 1;

array = new double\* [ n\_rows ];

for ( int i = 0; i < n\_rows; i++ ) array[ i ] = new double[ n\_cols ];

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

for ( int j = 0; j < n\_cols; j++ ) array[ i ][ j ] = 0;

}

//---------------------------------------------------------------------------------

CMatrix::~CMatrix( ) {

for ( int i = 0; i < n\_rows; i++ ) delete array[ i ];

delete array;

}

//---------------------------------------------------------------------------------

double& CMatrix::operator()( int i, int j )

// i - номер строки

// j - номер столбца

{

if ( ( i > n\_rows - 1 ) || ( j > n\_cols - 1 ) ) // проверка выхода за диапазон

{

TCHAR\* error = \_T( "CMatrix::operator(int,int): выход индекса за границу диапазона " );

MessageBox( NULL, error, \_T( "Ошибка" ), MB\_ICONSTOP );

exit( 1 );

}

return array[ i ][ j ];

}

//---------------------------------------------------------------------------------

double& CMatrix::operator()( int i )

// i - номер строки для вектора

{

if ( n\_cols > 1 ) // Число столбцов больше одного

{

char\* error = "CMatrix::operator(int): объект не вектор - число столбцов больше 1 ";

MessageBox( NULL, error, "Ошибка", MB\_ICONSTOP );

exit( 1 );

}

if ( i > n\_rows - 1 ) // проверка выхода за диапазон

{

TCHAR\* error = TEXT( "CMatrix::operator(int): выход индекса за границу диапазона " );

MessageBox( NULL, error, TEXT( "Ошибка" ), MB\_ICONSTOP );

exit( 1 );

}

return array[ i ][ 0 ];

}

//---------------------------------------------------------------------------------

CMatrix CMatrix::operator-( )

// Оператор -M

{

CMatrix Temp( n\_rows, n\_cols );

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

for ( int j = 0; j < n\_cols; j++ ) Temp( i, j ) = -array[ i ][ j ];

return Temp;

}

//---------------------------------------------------------------------------------

CMatrix CMatrix::operator+( CMatrix& M )

// Оператор M1+M2

{

int bb = ( n\_rows == M.rows( ) ) && ( n\_cols == M.cols( ) );

if ( !bb ) {

char\* error = "CMatrix::operator(+): несоответствие размерностей матриц ";

MessageBox( NULL, error, "Ошибка", MB\_ICONSTOP );

exit( 1 );

}

CMatrix Temp( \*this );

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

for ( int j = 0; j < n\_cols; j++ ) Temp( i, j ) += M( i, j );

return Temp;

}

//---------------------------------------------------------------------------------

CMatrix CMatrix::operator-( CMatrix& M )

// Оператор M1-M2

{

int bb = ( n\_rows == M.rows( ) ) && ( n\_cols == M.cols( ) );

if ( !bb ) {

char\* error = "CMatrix::operator(-): несоответствие размерностей матриц ";

MessageBox( NULL, error, "Ошибка", MB\_ICONSTOP );

exit( 1 );

}

CMatrix Temp( \*this );

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

for ( int j = 0; j < n\_cols; j++ ) Temp( i, j ) -= M( i, j );

return Temp;

}

//---------------------------------------------------------------------------------

CMatrix CMatrix::operator\*( CMatrix& M )

// Умножение на матрицу M

{

double sum;

int nn = M.rows( );

int mm = M.cols( );

CMatrix Temp( n\_rows, mm );

if ( n\_cols == nn ) {

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

for ( int j = 0; j < mm; j++ ) {

sum = 0;

for ( int k = 0; k < n\_cols; k++ ) sum += ( \*this )( i, k ) \* M( k, j );

Temp( i, j ) = sum;

}

}

else {

TCHAR\* error = TEXT( "CMatrix::operator\*: несоответствие размерностей матриц " );

MessageBox( NULL, error, TEXT( "Ошибка" ), MB\_ICONSTOP );

exit( 1 );

}

return Temp;

}

//---------------------------------------------------------------------------------

CMatrix CMatrix::operator\*( double x )

// Умножение на число

{

CMatrix Temp( \*this );

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

for ( int j = 0; j < n\_cols; j++ ) Temp( i, j ) = Temp( i, j ) \* x;

return Temp;

}

//---------------------------------------------------------------------------------

CMatrix CMatrix::operator/( double x )

// Деление на число

{

CMatrix Temp( \*this );

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

for ( int j = 0; j < n\_cols; j++ ) Temp( i, j ) = Temp( i, j ) / x;

return Temp;

}

//---------------------------------------------------------------------------------

CMatrix CMatrix::operator=( const CMatrix& M )

// Оператор присваивания M1=M

{

if ( this == &M ) return \*this;

int nn = M.rows( );

int mm = M.cols( );

if ( ( n\_rows == nn ) && ( n\_cols == mm ) ) {

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

for ( int j = 0; j < n\_cols; j++ ) array[ i ][ j ] = M.array[ i ][ j ];

}

else // для ошибки размерностей

{

TCHAR\* error = TEXT( "CMatrix::operator=: несоответствие размерностей матриц" );

MessageBox( NULL, error, TEXT( "Ошибка" ), MB\_ICONSTOP );

exit( 1 );

}

return \*this;

}

//---------------------------------------------------------------------------------

CMatrix::CMatrix( const CMatrix& M ) // Конструктор копирования

{

n\_rows = M.n\_rows;

n\_cols = M.n\_cols;

array = new double\* [ n\_rows ];

for ( int i = 0; i < n\_rows; i++ ) array[ i ] = new double[ n\_cols ];

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

for ( int j = 0; j < n\_cols; j++ ) array[ i ][ j ] = M.array[ i ][ j ];

}

//---------------------------------------------------------------------------------

CMatrix CMatrix::operator+( double x )

// Оператор M+x, где M - матрица, x - число

{

CMatrix Temp( \*this );

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

for ( int j = 0; j < n\_cols; j++ ) Temp( i, j ) += x;

return Temp;

}

//---------------------------------------------------------------------------------

CMatrix CMatrix::operator-( double x )

// Оператор M+x, где M - матрица, x - число

{

CMatrix Temp( \*this );

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

for ( int j = 0; j < n\_cols; j++ ) Temp( i, j ) -= x;

return Temp;

}

//---------------------------------------------------------------------------------

CMatrix CMatrix::Transp( )

// Возвращает матрицу,транспонированную к (\*this)

{

CMatrix Temp( n\_cols, n\_rows );

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

for ( int j = 0; j < n\_rows; j++ ) Temp( i, j ) = array[ j ][ i ];

return Temp;

}

//---------------------------------------------------------------------------------

CMatrix CMatrix::GetRow( int k )

// Возвращает строку матрицы по номеру k

{

if ( k > n\_rows - 1 ) {

char\* error = "CMatrix::GetRow(int k): параметр k превышает число строк ";

MessageBox( NULL, error, "Ошибка", MB\_ICONSTOP );

exit( 1 );

}

CMatrix M( 1, n\_cols );

for ( int i = 0; i < n\_cols; i++ )M( 0, i ) = ( \*this )( k, i );

return M;

}

//---------------------------------------------------------------------------------

CMatrix CMatrix::GetRow( int k, int n, int m )

// Возвращает подстроку из строки матрицы с номером k

// n - номер первого элемента в строке

// m - номер последнего элемента в строке

{

int b1 = ( k >= 0 ) && ( k < n\_rows );

int b2 = ( n >= 0 ) && ( n <= m );

int b3 = ( m >= 0 ) && ( m < n\_cols );

int b4 = b1 && b2 && b3;

if ( !b4 ) {

char\* error = "CMatrix::GetRow(int k,int n, int m):ошибка в параметрах ";

MessageBox( NULL, error, "Ошибка", MB\_ICONSTOP );

exit( 1 );

}

int nCols = m - n + 1;

CMatrix M( 1, nCols );

for ( int i = n; i <= m; i++ )M( 0, i - n ) = ( \*this )( k, i );

return M;

}

//---------------------------------------------------------------------------------

CMatrix CMatrix::GetCol( int k )

// Возвращает столбец матрицы по номеру k

{

if ( k > n\_cols - 1 ) {

char\* error = "CMatrix::GetCol(int k): параметр k превышает число столбцов ";

MessageBox( NULL, error, "Ошибка", MB\_ICONSTOP );

exit( 1 );

}

CMatrix M( n\_rows, 1 );

for ( int i = 0; i < n\_rows; i++ )M( i, 0 ) = ( \*this )( i, k );

return M;

}

//---------------------------------------------------------------------------------

CMatrix CMatrix::GetCol( int k, int n, int m )

// Возвращает подстолбец из столбца матрицы с номером k

// n - номер первого элемента в столбце

// m - номер последнего элемента в столбце

{

int b1 = ( k >= 0 ) && ( k < n\_cols );

int b2 = ( n >= 0 ) && ( n <= m );

int b3 = ( m >= 0 ) && ( m < n\_rows );

int b4 = b1 && b2 && b3;

if ( !b4 ) {

char\* error = "CMatrix::GetCol(int k,int n, int m):ошибка в параметрах ";

MessageBox( NULL, error, "Ошибка", MB\_ICONSTOP );

exit( 1 );

}

int nRows = m - n + 1;

CMatrix M( nRows, 1 );

for ( int i = n; i <= m; i++ )M( i - n, 0 ) = ( \*this )( i, k );

return M;

}

//---------------------------------------------------------------------------------

CMatrix CMatrix::RedimMatrix( int NewRow, int NewCol )

// Изменяет размер матрицы с уничтожением данных

// NewRow - новое число строк

// NewCol - новое число столбцов

{

for ( int i = 0; i < n\_rows; i++ ) delete array[ i ];

delete array;

n\_rows = NewRow;

n\_cols = NewCol;

array = new double\* [ n\_rows ];

for ( int i = 0; i < n\_rows; i++ ) array[ i ] = new double[ n\_cols ];

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

for ( int j = 0; j < n\_cols; j++ ) array[ i ][ j ] = i \* j;

return ( \*this );

}

//---------------------------------------------------------------------------------

CMatrix CMatrix::RedimData( int NewRow, int NewCol )

// Изменяет размер матрицы с сохранением данных, которые можно сохранить

// NewRow - новое число строк

// NewCol - новое число столбцов

{

CMatrix Temp = ( \*this );

this->RedimMatrix( NewRow, NewCol );

int min\_rows = Temp.rows( ) < ( \*this ).rows( ) ? Temp.rows( ) : ( \*this ).rows( );

int min\_cols = Temp.cols( ) < ( \*this ).cols( ) ? Temp.cols( ) : ( \*this ).cols( );

for ( int i = 0; i < min\_rows; i++ )

for ( int j = 0; j < min\_cols; j++ ) ( \*this )( i, j ) = Temp( i, j );

return ( \*this );

}

//---------------------------------------------------------------------------------

CMatrix CMatrix::RedimMatrix( int NewRow )

// Изменяет размер матрицы с уничтожением данных

// NewRow - новое число строк

// NewCol=1

{

for ( int i = 0; i < n\_rows; i++ ) delete array[ i ];

delete array;

n\_rows = NewRow;

n\_cols = 1;

array = new double\* [ n\_rows ];

for ( int i = 0; i < n\_rows; i++ ) array[ i ] = new double[ n\_cols ];

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

for ( int j = 0; j < n\_cols; j++ ) array[ i ][ j ] = i \* j;

return ( \*this );

}

//---------------------------------------------------------------------------------

CMatrix CMatrix::RedimData( int NewRow )

// Изменяет размер матрицы с сохранением данных, которые можно сохранить

// NewRow - новое число строк

// NewCol=1

{

CMatrix Temp = ( \*this );

this->RedimMatrix( NewRow );

int min\_rows = Temp.rows( ) < ( \*this ).rows( ) ? Temp.rows( ) : ( \*this ).rows( );

for ( int i = 0; i < min\_rows; i++ )( \*this )( i ) = Temp( i );

return ( \*this );

}

//----------------------------------------------------------------------------------

double CMatrix::MaxElement( )

// Максимальное значение элементов матрицы

{

double max = ( \*this )( 0, 0 );

for ( int i = 0; i < ( this->rows( ) ); i++ )

for ( int j = 0; j < ( this->cols( ) ); j++ ) if ( ( \*this )( i, j ) > max ) max = ( \*this )( i, j );

return max;

}

//----------------------------------------------------------------------------------

double CMatrix::MinElement( )

// Минимальное значение элементов матрицы

{

double min = ( \*this )( 0, 0 );

for ( int i = 0; i < ( this->rows( ) ); i++ )

for ( int j = 0; j < ( this->cols( ) ); j++ ) if ( ( \*this )( i, j ) < min ) min = ( \*this )( i, j );

return min;

}

double CMatrix::Abs( )

// Модуль матрицы

{

return sqrt( ( ( \*this )( 0 ) \* ( \*this )( 0 ) + ( \*this )( 1 ) \* ( \*this )( 1 ) + ( \*this )( 2 ) \* ( \*this )( 2 ) ) );

}

Приложение Г. Описание класса Ellipsoid.

// ChildView.cpp : implementation of the CChildView class

#include "stdafx.h"

#include "ellipsoid3D.h"

#include "ChildVie.h"

#include "SizeDialog.h"

#ifdef \_DEBUG

#define new DEBUG\_NEW

#endif

// CChildView

CChildView::CChildView( ) {

step = 10;

m = 0;

PView.RedimMatrix( 3 );

//начальное положение наблюдателя

PView( 0 ) = 1; PView( 1 ) = 315; PView( 2 ) = 45;

PLight.RedimMatrix( 3 );

//начальное положение источника света

PLight( 0 ) = 1; PLight( 1 ) = 315; PLight( 2 ) = 45;

pp = new CEllipsoid( );

//флаги нажатия мыши и выделения рамкой

m\_IsMousePressed = 0;

m\_IsSelectedRect = 0;

m\_SelectedRect = new CRect( );

//перо для прорисовки рамки выделения

pen.CreatePen( PS\_DOT, 1, RGB( 255, 0, 0 ) );

}

CChildView::~CChildView( ) { }

BEGIN\_MESSAGE\_MAP( CChildView, CWnd )

ON\_WM\_PAINT( )

ON\_WM\_KEYDOWN( )

ON\_WM\_LBUTTONDOWN( )

ON\_WM\_LBUTTONUP( )

ON\_WM\_MOUSEMOVE( )

ON\_COMMAND( ID\_COMMANDS\_SETCOLOR, &CChildView::OnSetcolor )

ON\_COMMAND( ID\_COMMANDS\_SETSIZE, &CChildView::OnSetsize )

ON\_COMMAND( ID\_FILE\_SAVE\_AS, &CChildView::OnSave1 )

ON\_WM\_SIZE( )

END\_MESSAGE\_MAP( )

// CChildView message handlers

BOOL CChildView::PreCreateWindow( CREATESTRUCT& cs ) {

if ( !CWnd::PreCreateWindow( cs ) )

return FALSE;

cs.dwExStyle |= WS\_EX\_CLIENTEDGE;

cs.style &= ~WS\_BORDER;

cs.lpszClass = AfxRegisterWndClass( CS\_HREDRAW | CS\_VREDRAW | CS\_DBLCLKS,

::LoadCursor( NULL, IDC\_ARROW ), reinterpret\_cast<HBRUSH>( COLOR\_WINDOW + 1 ), NULL );

return TRUE;

}

void CChildView::OnPaint( ) {

CPaintDC dc( this );

//прорисовка эллипсоида

pp->DrawEnlighted( dc, PView, PLight, rw, col, m );

//прорисовки рамки выделения

if ( m\_IsSelectedRect ) {

dc.SelectObject( pen );

dc.MoveTo( m\_SelectedRect->left, m\_SelectedRect->top );

dc.LineTo( m\_SelectedRect->right, m\_SelectedRect->top );

dc.LineTo( m\_SelectedRect->right, m\_SelectedRect->bottom );

dc.LineTo( m\_SelectedRect->left, m\_SelectedRect->bottom );

dc.LineTo( m\_SelectedRect->left, m\_SelectedRect->top );

}

}

void CChildView::OnLButtonDown( UINT var, CPoint point ) {

//начало выделения области мышью

p1 = point;

m\_IsSelectedRect = m\_IsMousePressed = true;

}

void CChildView::OnLButtonUp( UINT var, CPoint p2 ) {

//окончание выделения области мышью

m\_IsMousePressed = 0;

m\_SelectedRect = new CRect( min( p1.x, p2.x ), min( p1.y, p2.y ), max( p1.x, p2.x ), max( p1.y, p2.y ) );

}

void CChildView::OnMouseMove( UINT var, CPoint p2 ) {

//прорисовка рамки при выделении

if ( m\_IsSelectedRect && m\_IsMousePressed ) {

m\_SelectedRect = new CRect( min( p1.x, p2.x ), min( p1.y, p2.y ), max( p1.x, p2.x ), max( p1.y, p2.y ) );

Invalidate( 1 );

}

}

void CChildView::OnKeyDown( UINT nChar, UINT nRepCount, UINT nFlags ) {

//поворот наблюдателя

if ( nChar == 37 ) { //влево

PView( 1 ) += step;

if ( PView( 1 ) > 360 )

PView( 1 ) -= 360;

Invalidate( );

}

else if ( nChar == 39 ) { //вправо

PView( 1 ) -= step;

if ( PView( 1 ) < 0 )

PView( 1 ) += 360;

Invalidate( );

}

else if ( nChar == 38 ) { //вверх

PView( 2 ) += step;

if ( PView( 2 ) > 360 )

PView( 2 ) -= 360;

Invalidate( );

}

else if ( nChar == 40 ) { //вниз

PView( 2 ) -= step;

if ( PView( 2 ) < 0 )

PView( 2 ) += 360;

Invalidate( );

}

//поворот источника света

else if ( nChar == 65 ) { //A

PLight( 1 ) -= step;

if ( PLight( 1 ) < 0 )

PLight( 1 ) += 360;

Invalidate( );

}

else if ( nChar == 68 ) { //D

PLight( 1 ) += step;

if ( PLight( 1 ) > 360 )

PLight( 1 ) -= 360;

Invalidate( );

}

else if ( nChar == 87 ) { //W

PLight( 2 ) -= step;

if ( PLight( 2 ) < 0 )

PLight( 2 ) += 360;

Invalidate( );

}

else if ( nChar == 83 ) { //S

PLight( 2 ) += step;

if ( PLight( 2 ) > 360 )

PLight( 2 ) -= 360;

Invalidate( );

}

//переключение модели освещения

else if ( nChar == 79 ) { //O

m = 1;

Invalidate( );

}

else if ( nChar == 80 ) { //P

m = 0;

Invalidate( );

}

}

//диалог установки цвета эллипсоида

void CChildView::OnSetcolor( ) {

CColorDialog ccd;

if ( ccd.DoModal( ) == IDOK ) {

col = ccd.GetColor( );

Invalidate( );

}

}

//диалоги установки размеров эллипсоида

void CChildView::OnSetsize( ) {

SizeDialog sd;

if ( sd.DoModal( ) == IDOK ) {

int a = sd.GetX( ), b = sd.GetY( ), c = sd.GetZ( );

pp = new CEllipsoid( a, b, c );

Invalidate( );

}

}

//диалог сохранения выделенной области в файл

void CChildView::OnSave1( ) {

CFileDialog fOpenDlg( FALSE, \_T( "fif" ), 0, NULL, \_T( "Image File (\*.bmp)|\*.bmp|" ), this );

// заголовок диалога

fOpenDlg.m\_pOFN->lpstrTitle = \_T( "Select BMP File" );

// отображение диалога

if ( fOpenDlg.DoModal( ) == IDOK ) {

m\_saved\_file = fOpenDlg.GetPathName( );

Invalidate( );

Sleep( 1000 );

ClientToBmp( GetSafeHwnd( ), m\_saved\_file, \*m\_SelectedRect );

}

m\_IsSelectedRect = false;

}

//сохранение выделенной области в файл

int ClientToBmp( HWND hWnd, const char\* filepath, CRect r ) {

//открытие фвйла для записи

HANDLE fh = CreateFile( filepath, GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL | FILE\_FLAG\_SEQUENTIAL\_SCAN, NULL );

if ( fh == INVALID\_HANDLE\_VALUE )

return 2;

//информационный заголовок

BITMAPINFOHEADER bi;

ZeroMemory( &bi, sizeof( BITMAPINFOHEADER ) );

bi.biSize = sizeof( BITMAPINFOHEADER );

bi.biWidth = r.right - r.left;

bi.biHeight = r.bottom - r.top;

bi.biPlanes = 1;

bi.biBitCount = 24; //16;

bi.biSizeImage = ( bi.biWidth \* bi.biBitCount + 31 ) / 32 \* 4 \* bi.biHeight;

//файловый заголовок

BITMAPFILEHEADER bmfHdr;

ZeroMemory( &bmfHdr, sizeof( BITMAPFILEHEADER ) );

bmfHdr.bfType = ( 'M' << 8 ) | 'B'; //0x4D42; //BM

bmfHdr.bfSize = bi.biSizeImage + sizeof( BITMAPFILEHEADER ) + bi.biSize;

bmfHdr.bfReserved1 = bmfHdr.bfReserved2 = 0;

bmfHdr.bfOffBits = ( DWORD )sizeof( BITMAPFILEHEADER ) + (DWORD)bi.biSize;

//получение в hBitmap выделенной области изображения

HDC hDC = GetDC( hWnd );

HDC hDCMem = CreateCompatibleDC( hDC );

HBITMAP hBitmap = CreateCompatibleBitmap( hDC, bi.biWidth, bi.biHeight );

HBITMAP oldBitmap = (HBITMAP)SelectObject( hDCMem, hBitmap );

BitBlt( hDCMem, 0, 0, bi.biWidth, bi.biHeight, hDC, r.left, r.top, SRCCOPY );

hBitmap = (HBITMAP)SelectObject( hDCMem, oldBitmap );

HANDLE hDIB = GlobalAlloc( GHND, bi.biSizeImage );

char\* lp = (char\*)GlobalLock( hDIB );

//получение изображения в аппаратно-независимом формате

GetDIBits( hDC, hBitmap, 0, bi.biHeight, lp, (LPBITMAPINFO)& bi, DIB\_RGB\_COLORS );

DWORD dwWritten = sizeof( BITMAPFILEHEADER );

//запись файлового заголовка

WriteFile( fh, (LPSTR)& bmfHdr, sizeof( BITMAPFILEHEADER ), &dwWritten, NULL );

dwWritten = sizeof( BITMAPINFOHEADER );

//запись информационного заголовка

WriteFile( fh, (LPSTR)& bi, sizeof( BITMAPINFOHEADER ), &dwWritten, NULL );

dwWritten = bi.biSizeImage;

//запись изображения в аппаратно-независимом формате

WriteFile( fh, lp, bi.biSizeImage, &dwWritten, NULL );

GlobalUnlock( hDIB );

GlobalFree( hDIB );

DeleteObject( hBitmap );

lp = NULL;

CloseHandle( fh );

ReleaseDC( hWnd, hDCMem );

ReleaseDC( hWnd, hDC );

DeleteDC( hDCMem );

DeleteDC( hDC );

if ( dwWritten == 2 ) return 2;

return 0;

}

void CChildView::OnSize( UINT nType, int cx, int cy ) {

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

rw.SetRect( 100, 100, cx - 100, cy - 100 );

}

Приложение Д. Функции аффинных пеобразованй.

#include "StdAfx.h"

#include "Athena2D.h"

//перемещение объекта

CMatrix Translate3D( double x, double y, double z ) {

CMatrix m( 4, 4 );

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

m( 0, 3 ) = x;

m( 1, 3 ) = y;

m( 2, 3 ) = z;

return m;

}

//поворот объекта вокруг оси OZ

CMatrix Rotate3Dz( double fi ) {

CMatrix m( 4, 4 );

double fi\_r = fi \* pi / 180.0;

m( 0, 0 ) = cos( fi\_r );

m( 1, 1 ) = cos( fi\_r );

m( 2, 2 ) = 1; m( 3, 3 ) = 1;

m( 0, 1 ) = -sin( fi\_r );

m( 1, 0 ) = sin( fi\_r );

return m;

}

//поворот объекта вокруг оси OX

CMatrix Rotate3Dx( double fi ) {

CMatrix m( 4, 4 );

double fi\_r = fi \* pi / 180.0;

m( 1, 1 ) = cos( fi\_r );

m( 2, 2 ) = cos( fi\_r );

m( 0, 0 ) = 1; m( 3, 3 ) = 1;

m( 1, 2 ) = -sin( fi\_r );

m( 2, 1 ) = sin( fi\_r );

return m;

}

//поворот объекта вокруг оси OY

CMatrix Rotate3Dy( double fi ) {

CMatrix m( 4, 4 );

double fi\_r = fi \* pi / 180.0;

m( 0, 0 ) = cos( fi\_r );

m( 2, 2 ) = cos( fi\_r );

m( 1, 1 ) = 1; m( 3, 3 ) = 1;

m( 0, 2 ) = -sin( fi\_r );

m( 2, 0 ) = sin( fi\_r );

return m;

}

//векторное произведение векторов

CMatrix VectorMult( CMatrix V1, CMatrix V2 ) {

CMatrix m( 3 ), a( 3, 3 );

a( 0, 1 ) = -V1( 2 );

a( 0, 2 ) = V1( 1 );

a( 1, 2 ) = -V1( 0 );

a( 1, 0 ) = V1( 2 );

a( 2, 0 ) = -V1( 1 );

a( 2, 1 ) = V1( 0 );

m = a \* V2;

return m;

}

//скалярное произведение векторов

double ScalarMult( CMatrix V1, CMatrix V2 ) {

return sqrt( V1( 0 ) \* V1( 0 ) + V1( 1 ) \* V1( 1 ) + V1( 2 ) \* V1( 2 ) ) \* sqrt( V2( 0 ) \* V2( 0 ) + V2( 1 ) \* V2( 1 ) + V2( 2 ) \* V2( 2 ) ) \* ( V1( 0 ) \* V2( 0 ) + V1( 1 ) \* V2( 1 ) + V1( 2 ) \* V2( 2 ) ) / sqrt( ( V1( 0 ) \* V1( 0 ) + V1( 1 ) \* V1( 1 ) + V1( 2 ) \* V1( 2 ) ) \* ( V2( 0 ) \* V2( 0 ) + V2( 1 ) \* V2( 1 ) + V2( 2 ) \* V2( 2 ) ) );

}

//косинус угла между векторами

double cosViV2( CMatrix V1, CMatrix V2 ) {

return ScalarMult( V1, V2 ) / ( V1.Abs( ) \* V2.Abs( ) );

}

//преобразование сферических координат в декартовые

CMatrix SphereToCart( CMatrix DView ) {

CMatrix Ve( 3 );

double fi\_r = ( 270 - DView( 1 ) ) \* pi / 180;

double theta\_r = DView( 2 ) \* pi / 180;

Ve( 0 ) = DView( 0 ) \* sin( theta\_r ) \* sin( fi\_r );

Ve( 1 ) = DView( 0 ) \* sin( theta\_r ) \* cos( fi\_r );

Ve( 2 ) = DView( 0 ) \* cos( theta\_r );

return Ve;

}

//создание матрицы пересчета координат из мировой системы координат в видовую (МСК->ВСК)

CMatrix CreateViewCoord( double R, double fi, double theta ) {

CMatrix Ve( 4, 4 ), Mz( 4, 4 );

Mz( 0, 0 ) = -1; Mz( 1, 1 ) = 1; Mz( 2, 2 ) = 1; Mz( 3, 3 ) = 1;

double fi\_r = fi \* pi / 180;

double theta\_r = theta \* pi / 180;

Ve = Mz \* Rotate3Dx( 180 - theta ) \* Rotate3Dz( 90 - fi );

return Ve;

}

# Список литературы

1. Поляков А., Брусенцев В. - Методы и алгоритмы компьютерной графики в примерах на Visual C++ - СПб.: БХВ-Петербург, 2003.
2. Давыдов В. - Visual C++. Разработка Windows-приложений с помощью MFC и API-функций.
3. Шикин Е.В., Боресков А.В. - Компьютерная графика. Динамика, реалистические изображения. - М.: Диалог-МИФИ, 1995.
4. Павлидис Т. - Алгоритмы машинной графики и обработка изображений. - М.: Радио и связь, 1988.