МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий   
Кафедра Информатики и веб-дизайна   
Специальность 1-98 01 03 Программное обеспечение информационной   
безопасности мобильных систем

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТА:**

по дисциплине «Компьютерная геометрия и графика»

Тема «Шар\_2»

Исполнитель

студент 3 курса 7 группы Шичко Владислав Сергеевич \_

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

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

Руководитель Дятко А.А. \_

# Содержание

[Содержание 2](#_Toc57832689)

[Задание 3](#_Toc57832690)

[Введение 4](#_Toc57832691)

[1. Математическое описание алгоритмов решения задачи 5](#_Toc57832692)

[2. Описание алгоритмов приложения 11](#_Toc57832693)

[2.1. Общий алгоритм приложения 11](#_Toc57832694)

[2.2. Алгоритм обработки команд 12](#_Toc57832695)

[2.2.1. Сохранение изображения 12](#_Toc57832696)

[2.2.2. Изменение параметров 12](#_Toc57832697)

[2.3. Алгоритм отображения модели 13](#_Toc57832698)

[3. Описание интерфейсов приложения 15](#_Toc57832699)

[Заключение 18](#_Toc57832700)

[Приложения 19](#_Toc57832701)

[Приложение А. Блок-схема алгоритма обработки команд 19](#_Toc57832702)

[Приложение Б. Блок-схема алгоритма отображения модели 20](#_Toc57832703)

[Приложение В. Файл Sphere.cpp 21](#_Toc57832704)

# Задание

Приложение Windows  **«Шар\_2»**

Создать приложение Windows для графического 3D - изображения шара при его движении в плоскости XY в СК XYZ (Рис.5, вид сверху – положение наблюдателя на оси Z).

Траектория движения шара образуется из двух независимых движений:

* 1. гармоническое колебание вдоль радиус – вектора центра шара c частотой  и амплитудой A;
  2. вращение радиус – вектора центра шара вокруг оси Z (по или против часовой стрелки – по выбору пользователя) с частой .

Изменяемые параметры модели:

* амплитуда колебания A;
* частота колебания ;
* частота колебания ;
* длина  радиуса – вектора .
* положение наблюдателя , , в сферической СК с началом в точке O;
* положение источника света , , в сферической СК с началом в точке O;
* цвет источника света;
* тип модели отражения света от поверхности шара;

Использовать диффузную модель отражения света от шара.

При изменении размера окна изображение соответствующим образом масштабируется.

Показывать или не показывать траекторию движения шара (*по выбору пользователя*).

Для установки модели использовать окно диалога.

Обеспечить запись изображения в графический файл \*.bmp

X

Y

R

A

Z

# Введение

В современном мире часто возникают задачи наглядного представления каких-либо процессов. Данное представление может быть достигнуто при помощи использования технологий компьютерной графики.

В данном курсовом проекте будет модулироваться движение шара по сложной траектории в трехмерном пространстве. У пользователя должна быть возможность управлять камерой для наблюдения за процессом с разных сторон. Так как процесс движения будет отображаться на двухмерном дисплее, возникает необходимость пересчета из мировых трехмерных координат в двухмерные оконные с учетом положения камеры. Также необходимо учесть освещение и другие параметры, вводимые пользователем.

Во время выполнения программы у пользователя должна быть возможность сохранить изображение в файл.

Для реализации поставленной задачи будет использоваться библиотека MFC языка программирования С++, предоставляющая набор классов для реализации приложения, эффективно взаимодействующего с пользователем при помощи периферийных устройств, меню, диалоговых окон.

# Математическое описание алгоритмов решения задачи

Параметры, заданные для расчетов:

* А = 1.3;
* f = 20;
* F = 1;
* R = 10;
* PView – координаты камеры в сферической СК;
* PLight( – координаты источника света в сферической СК;
* (,координаты границ в мировой СК (максимальные и минимальные значения осей координат);
* , – координаты границ оконной СК.

Для начала пользователь вводит все необходимые данные. Пусть положение камеры в сферической СК задано вектором PView, и задаются в градусах. Переводим и в радианы . Тогда вектор PView будет содержать координаты (10, 1.0472, 0.5235). Далее необходимо получить матрицу пересчета из мировой системы координат в видовые с учетом положения камеры. Она является результатом перемножения матриц аффиных преобразований:

=

где – матрица преобразования правосторонней СК в левостороннюю, – матрица поворота вокруг оси X на угол ,– матрица поворота вокруг оси Z на угол , – матрица смещения начала системы координат.

Далее найдем матрицу преобразования из видовых координат в оконные.

,

где , , = 250-(-250) = 500 – ширина области отображения в мировых координатах, 800 – 0 = 800 – ширина области отображения в оконных координатах, – высота области отображения в мировых координатах, – 0 = 600 – высота области отображения в оконных координатах. Тогда

.

Задаем координаты для осей координат как начало координат и максимальные значения для каждого направления, а также от начала координат до минимального значения. выполняем пересчет координат из мировой в видовую и из видовой в оконную СК при помощи умножения на матрицы пересчета A и T и отображаем линии, соединяющие соответствующие точки, на экране (Рис.1).

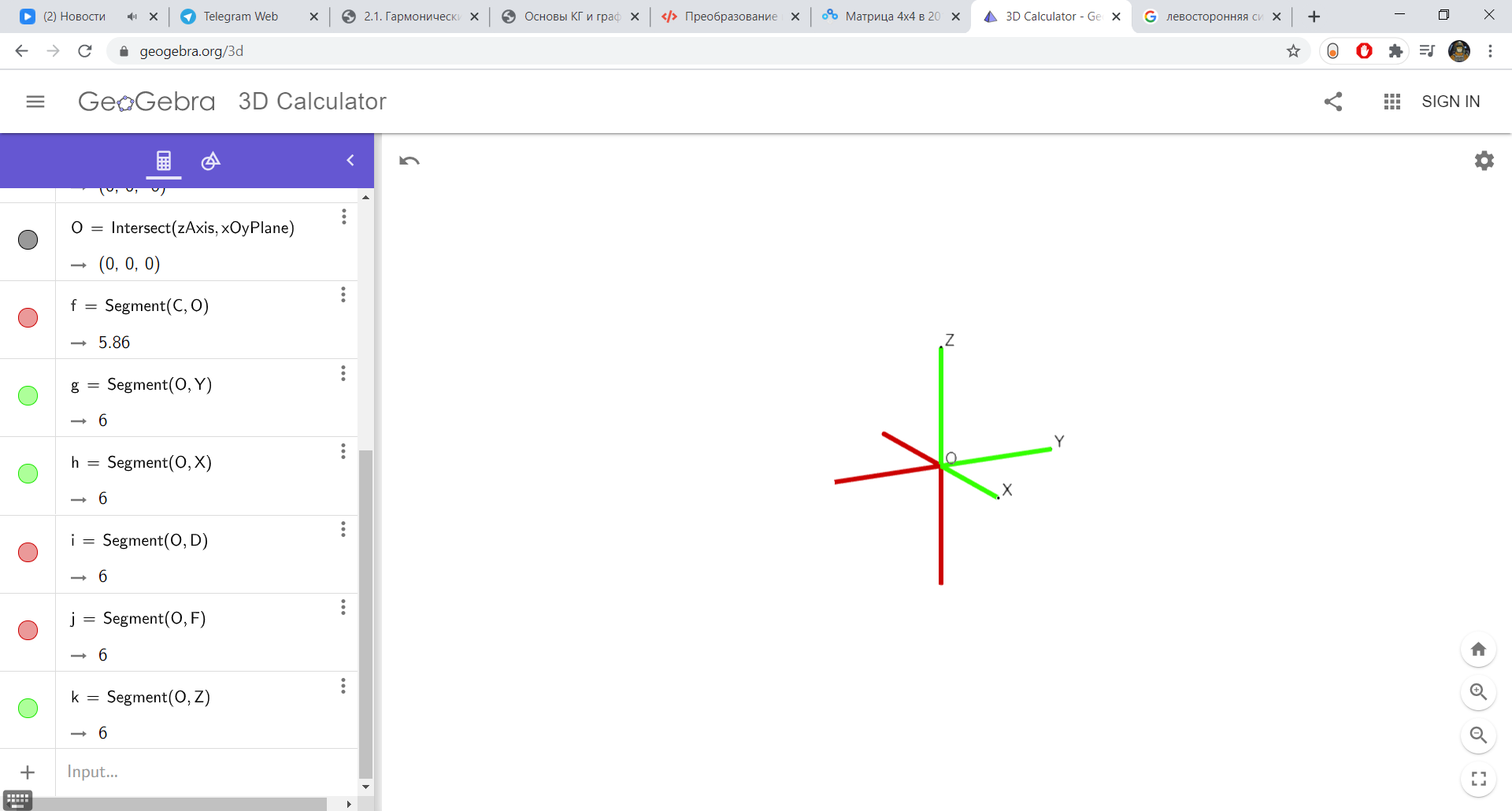


Рисунок 1 – Оси координат

Пример расчета координат точки Z на в оконной СК. В мировой СК координаты точки имеют вид:

,

При помощи матрицы A, полученной выше, выполняем расчет координат точки в видовой системе координат с учетом положения камеры:

Для пересчета из видовой в оконную СК убираем составляющую координаты z и получаем

Таким образом, координаты точки Z в оконной СК будут равны (400, 150). Повторяем вычисления для остальных максимальных и минимальных точек осей координат, а также для начала координат и отображаем оси на экране, соединяя соответствующие точки.

Для вывода траектории движения шара можно использовать матрицу M координат точек. Каждая точка будет представлена в виде

,

где x, y – координаты точек, z = 0, т.к. траектория располагается в плоскости XY.

График движения по окружности может быть задан параметрически, как

,

где F – частота вращения радиус-вектора вокруг оси Z. Тогда траектория вращения может иметь следующий вид:

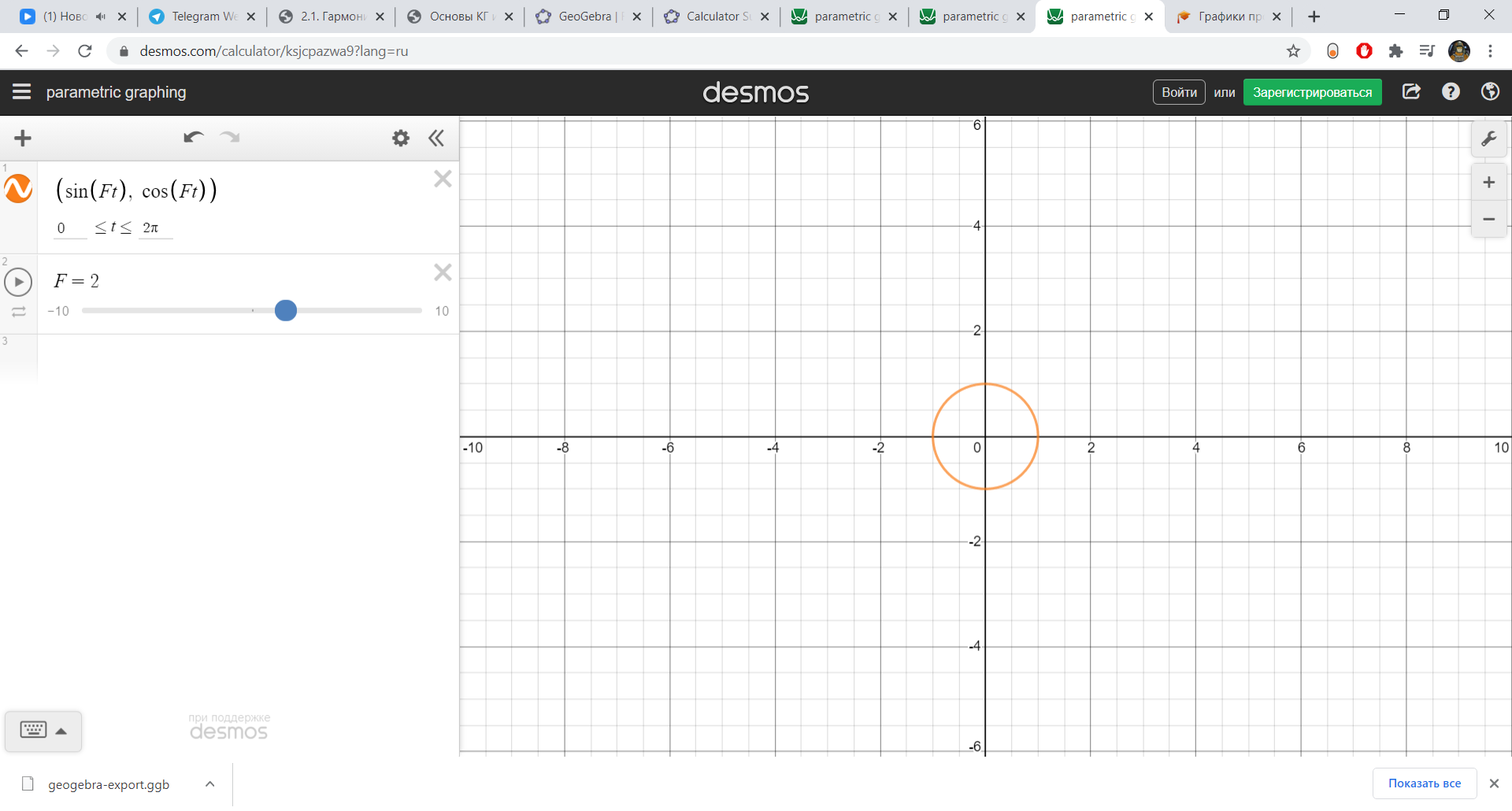


Рисунок 2 – Пример траектории вращения

График колебания около точки может быть задан параметрически, как

,

где R – радиус окружности (смещение точки от начала координат), A – амплитуда колебания, f – частота колебания. Траектория колебания имеет вид, изображенный на рисунке 3.

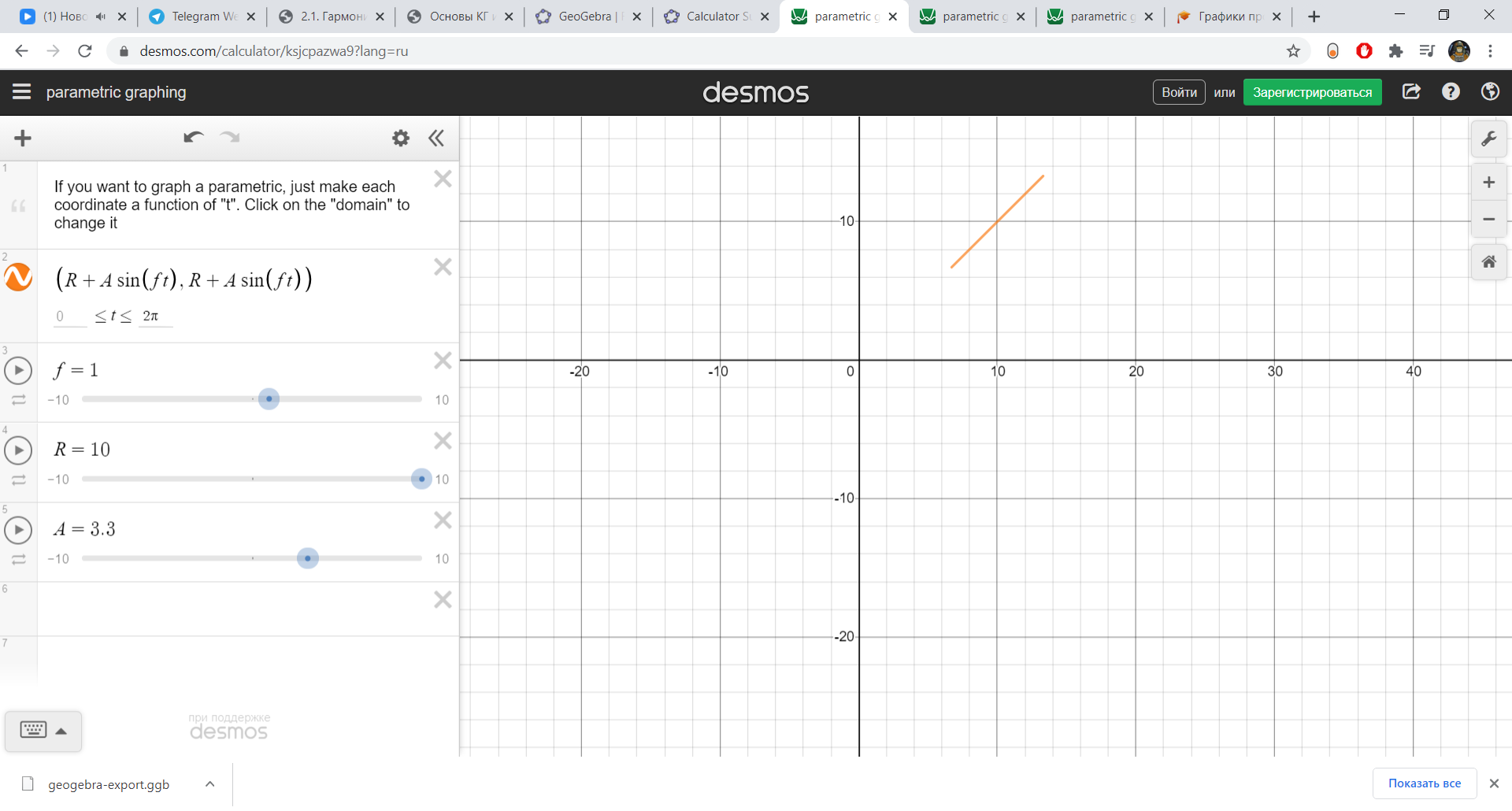


Рисунок 3 – Траектория колебания около точки

Произведение двух параметрических функций для x и y будет задавать необходимое по условию движение (Рис. 4):

,

где R – радиус окружности (смещение точки от начала координат), A – амплитуда колебания, f – частота колебания длины радиус-вектора, F – частота вращения радиуса-вектора.

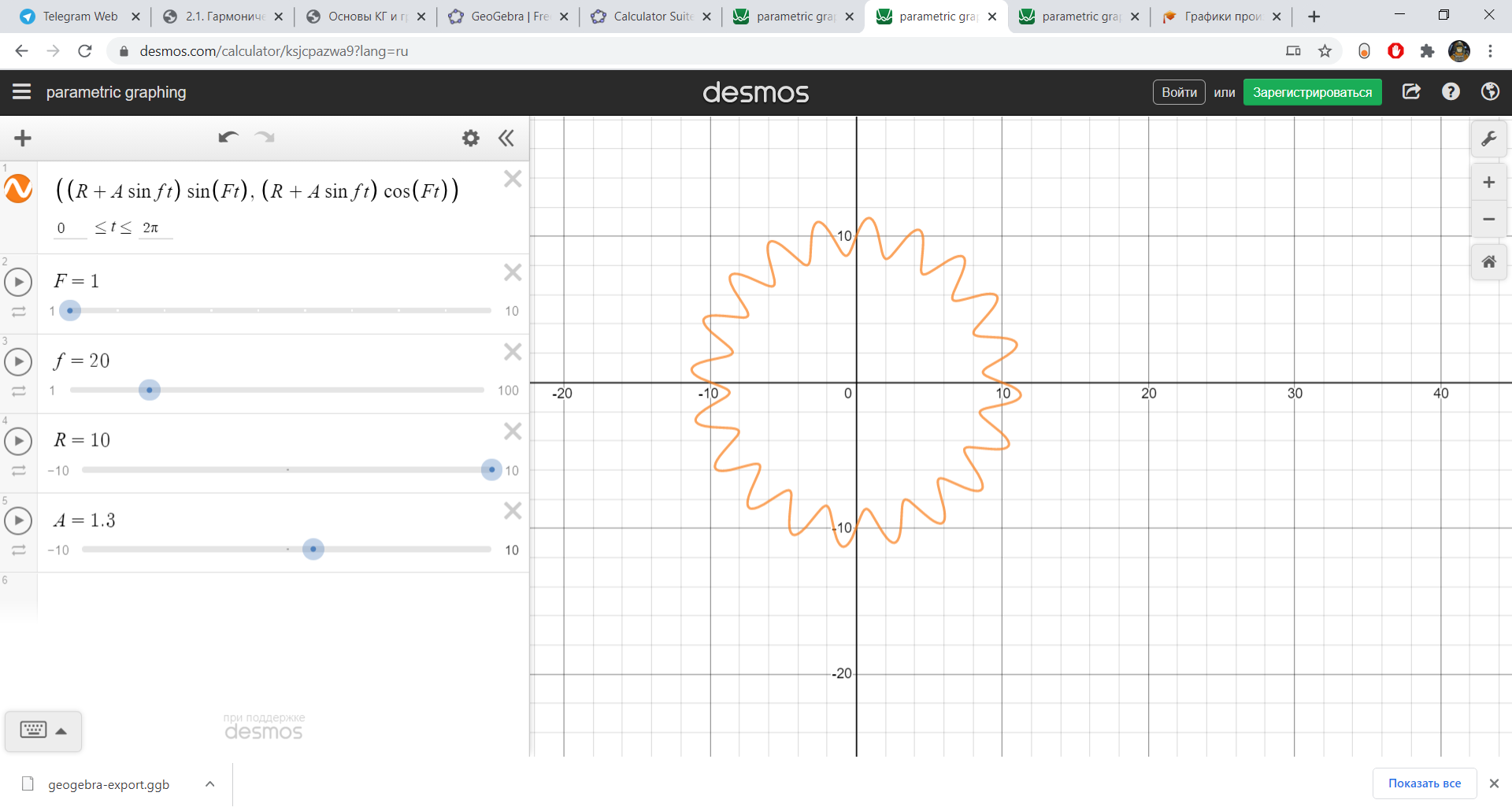


Рисунок 4 – Требуемая по заданию траектория движения.

Далее при помощи формулы, изменяя параметр t, заполняем матрицу M значениями координат x и y для точек на траектории (с заданной плотностью). После этого переводим координаты из мировой СК в видовую систему:

,

где – матрица координат точек в видовой системе координат. Переводим из видовой в оконную по формуле

,

где W – матрица координат точек траектории в оконной СК.

Пример расчета оконных координат конкретной точки на траектории.

Пусть заданы координаты точки на траектории, при параметре t = π/4. Подставим значение параметра t, а также остальные параметры в параметрические функции:

Таким образом координаты точки (координата z=0, т.к. траектория лежит в плоскости XY):

Выполняем пересчет из мировой в видовую СК для данной точки:

Для пересчета из видовой в оконную СК убираем составляющую координаты z и получаем

Таким образом, координаты точки Z в оконной СК будут равны (395, 310.038). Далее необходимо получить координаты для остальных точек на траектории, изменяя параметр t с заданным шагом.

При выводе сферы на экран берем точку на траектории из матрицы M и сдвигаем аффинным преобразованием начало системы координат в эту точку. Далее получаем координаты точек поверхности шара начиная с наиболее удаленных от наблюдателя, переводим координаты из мировой СК в видовую, из видовой в оконную. Выводим пиксель в соответствующие координаты оконной СК, и с цветом, полученным в зависимости от положения источника света и типа отражения.

При диффузном отражении цвет пикселя рассчитывается по следующей формуле (Рис. 5):

,

где – интенсивность светового потока, которую фиксировал бы приемник, находящийся в «зените», если бы площадка облучалась то же с зенита, – коэффициент, который учитывает свойства материала поверхности. Значение находится в диапазоне от 0 до 1. – угол между нормалью к поверхности в точке и наблюдателем, – угол падающего света относительно нормали.

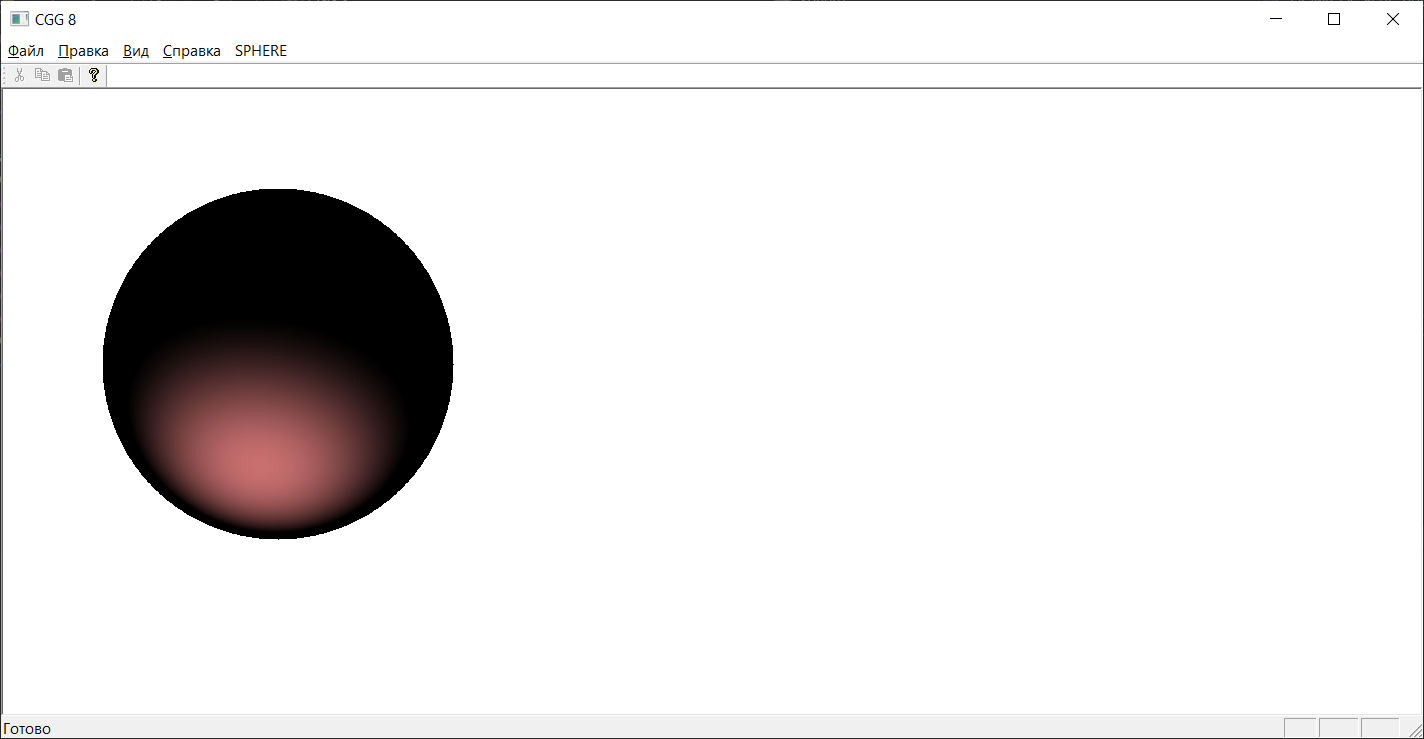


Рисунок 5 – Диффузное отражение поверхности шара.

При зеркальном отражении цвет пикселя рассчитывается по следующей формуле (Рис. 6):

,

где – интенсивность светового потока, – коэффициент пропорциональности, – угол отклонения наблюдателя от линии идеально отраженного луча, показатель p находится в диапазоне от 1 до 200 и зависит от качества полировки.

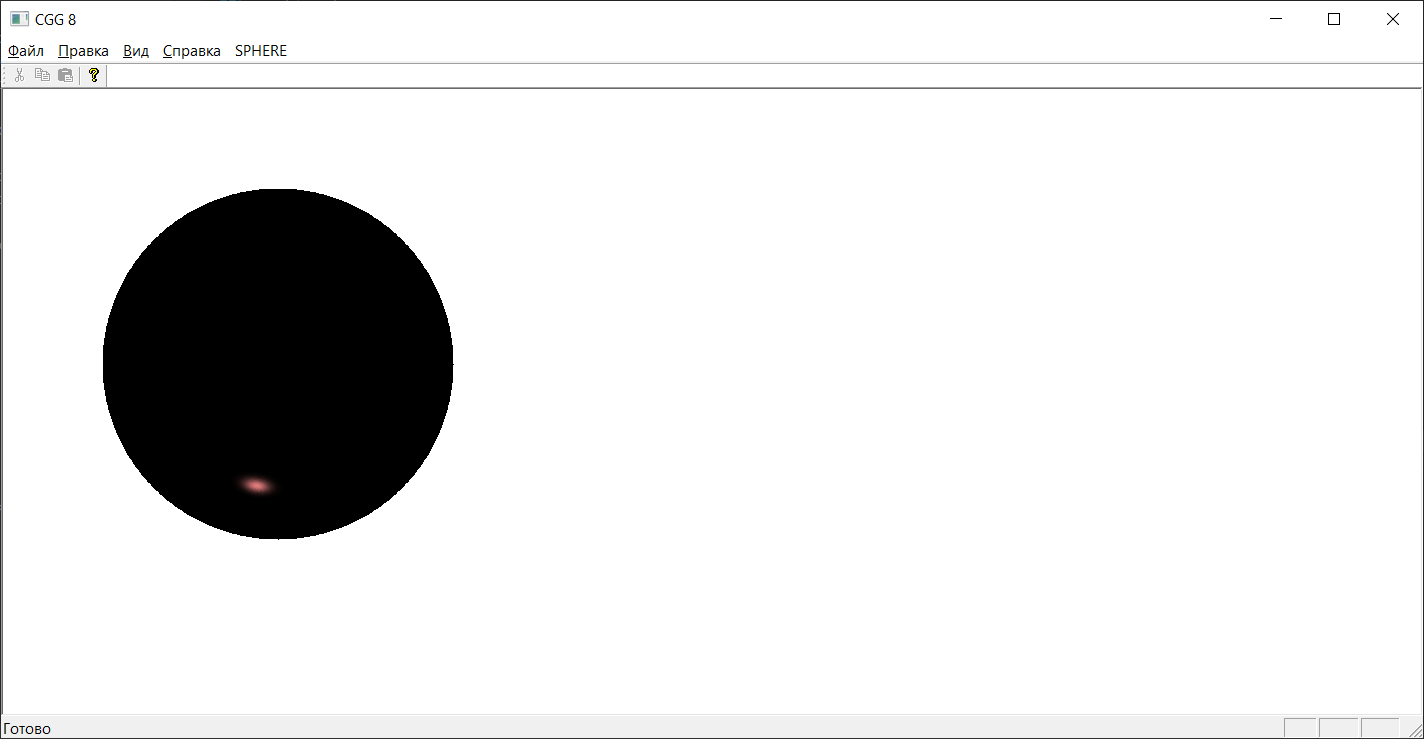


Рисунок 6 – Зеркальное отражение поверхности шара.

В программе интенсивность будет рассчитываться для каждого базисного цвета модели RGB.

# Описание алгоритмов приложения

## Общий алгоритм приложения

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

Начало

Установка начальных значений параметров, создание окна, запуск обработчиков сообщений

Получение и обработка команд, прорисовка изображения с учетом заданных пользователем параметров

Поступила команда «Выход»

Завершение обработки сообщений, закрытие окон, освобождение памяти

Конец

нет

да

Рисунок 7 – Общая блок-схема приложения.

Создание окна и запуск обработчиков сообщений выполняются по стандарту MFC приложений без использования архитектуры «Документ/Вид». Для хранения параметров была выработана соответствующая структура (приложение А). Завершение работы приложения выполняется также по стандарту MFC приложений.

## Алгоритм обработки команд

Приложение дает возможность пользователю использовать несколько команд, таких как: «Сохранить изображение», «Изменить параметры», «Получить справку», а также «Выход». Блок-схема алгоритма обработки команд представлена в приложении А.

Команда «Выход» вызывается, когда пользователь нажимает на кнопку закрытия основного окна приложения и приводит к закрытию приложения.

Команда «Получить справку» отображает окно «О проекте…» при нажатии пользователем на соответствующую кнопку меню.

Разработанные команды находятся в меню приложения и могут быть вызваны нажатием пользователя на соответствующий пункт меню/подменю.

### Сохранение изображения

Сохранение изображения осуществляется в функции SaveImage() (рисунок 8), которая находится в файле ChildView.cpp. В ее теле определяется переменная типа CFileDialog, которая представляет собой стандартный объект диалогового окна Windows для работы с файлами. Далее вызывается функция ClientToBmp, определенная в файле imagelib.cpp, которая осуществляет сохранение изображения рабочей области приложения в файл .bmp, путь к которому задает пользователь посредством диалогового окна.

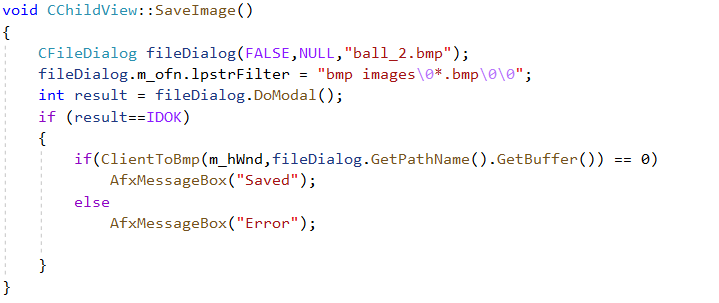


Рисунок 8 – Код функции SaveImage.

### Изменение параметров

Изменение параметров происходит в функции DialogMode() (рисунок 9), которая определена в файле ChildView.cpp. При ее вызове останавливается таймер функцией KillTimer(1), а также создается разработанное диалоговое окно ball\_2Dialog для установки параметров модели. Если в диалоговом окне была нажата кнопка «Запуск», то таймер запускается функцией SetTimer и шар начинает движение по траектории.

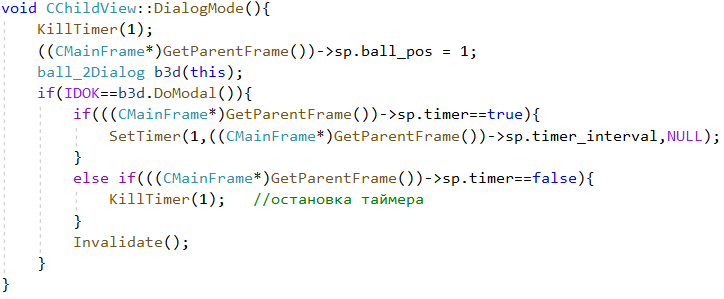


Рисунок 9 – Код функции DialogMode.

## Алгоритм отображения модели

Отображение модели движения шара по сложной траектории осуществляется в функции OnPaint() (рисунок 10), определенной в файле ChildView.cpp. Блок-схема алгоритма приведена в приложении Б.

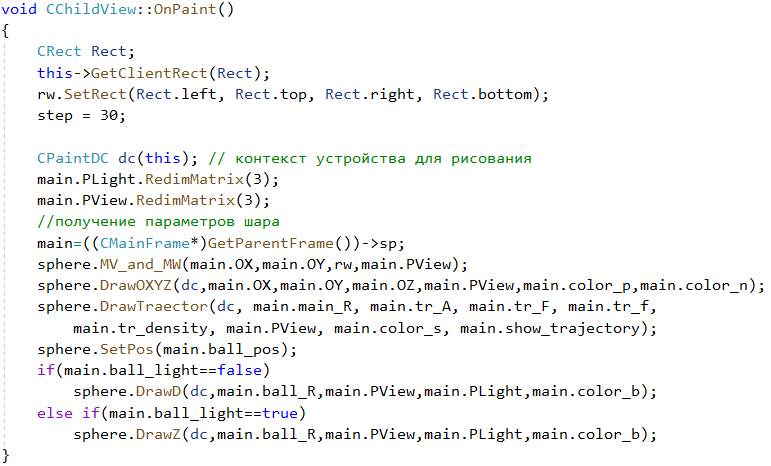


Рисунок 9 – Код функции DialogMode.

Сначала определяются координаты оконной системы координат, полученные при помощи функции GetClientRect(). Далее происходит инициализация переменной step, отвечающей за шаг поворота в градусах камеры и источника света.

После этого инициализируем значениями по умолчанию переменную main типа sphere\_params, описанного в файле sphere\_params.h (рисунок 10). В этой переменной хранятся параметры модели, такие как радиус траектории движения, амплитуда колебания шара, частота и т.д.

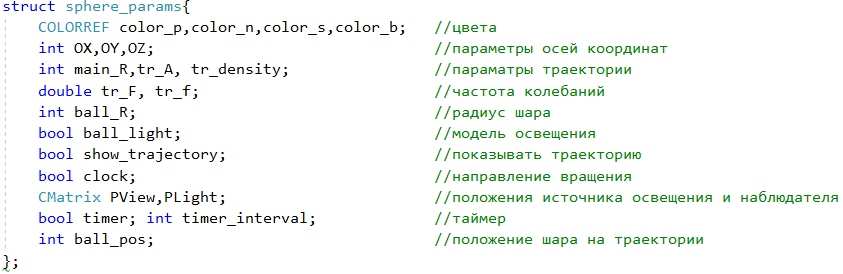


Рисунок 10 – Структура sphere\_params.

После инициализации параметров происходят вызовы функций, определенных для класса Sphere в файле Sphere.cpp. Код функций представлен в приложении В.

Сначала вызывается функция MV\_and\_MW, которая генерирует матрицы пересчета из мировой в видовую и из видовой в оконную системы координат, с учетом заданных параметров.

Далее вызывается функция DrawOXYZ, которая отображает на экране оси координат.

Следующим шагом происходит вызов функции DrawTraector, которая в соответствии с заданными параметрами траектории заполняет массив координат лежащих на траектории точек, для последующего использования в качестве координат центра шара. Также в этой функции осуществляется отображение траектории на экране, если необходимо. Расчет координат точек происходит в соответствии с формулами, полученными в разделе 1.

Функция SetPos устанавливает новую позицию отрисовки шара соответствующим элементом массива координат точек траектории.

Далее, в зависимости от выбранной модели отражения вызывается одна из двух функций DrawD, DrawZ. Их основное отличие в том, что DrawD осуществляет отображение шара в соответствии с диффузионной моделью отражения, а DrawZ – в соответствии с зеркальной. Отрисовка шара происходит следующим образом: сначала выводятся точки, наиболее удаленные от наблюдателя, потом те, что ближе, при этом в каждой итерации цикла отображаются точки как левой половины, так и правой.

# Описание интерфейсов приложения

При запуске приложения открывается основное окно приложения (Рисунок 11). В нем можно увидеть элементы модели движущегося шара, а также меню.

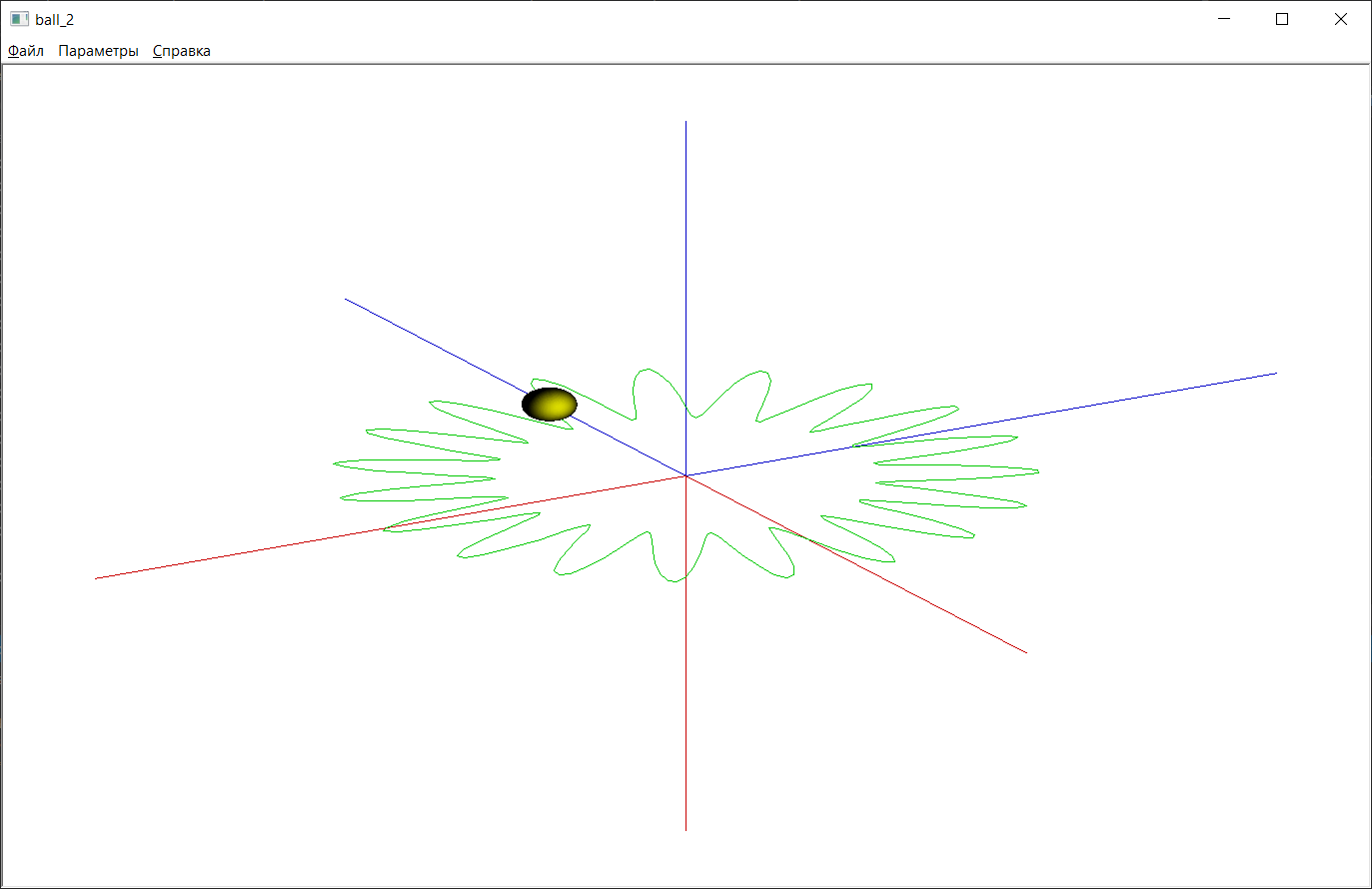


Рисунок 11 – Основное окно приложения.

Меню содержит такие элементы, как «Файл», «Параметры», «Справка». «Файл» имеет подпункты «Сохранить изображение» для открытия диалога сохранения изображения (Рисунок 12), а также подпункт «Выход» для закрытия приложения.

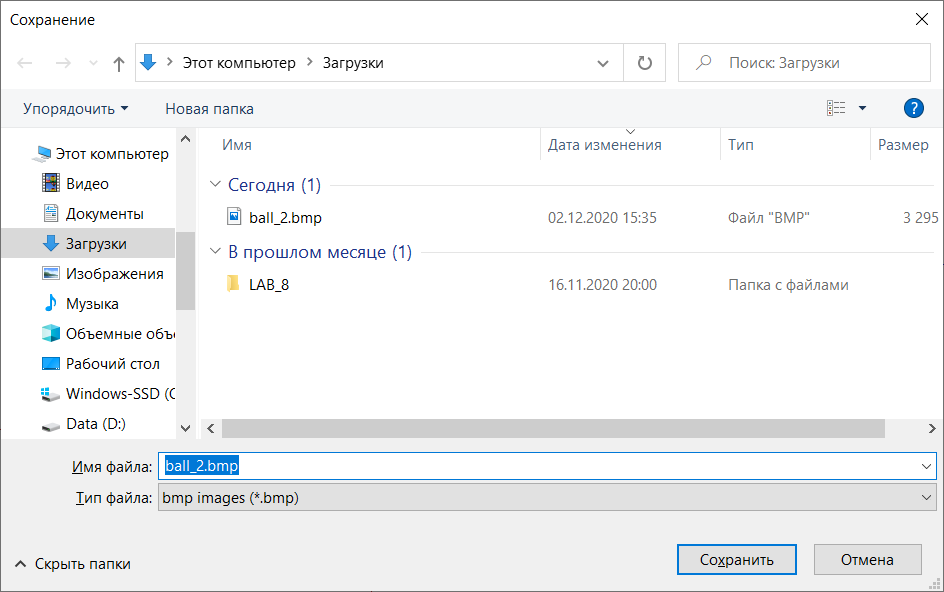


Рисунок 12 – Диалог сохранения изображения.

Пункт меню «Параметры» имеет подпункт «Изменить», при нажатии на который открывается диалоговое окно изменения параметров (рисунок 13).

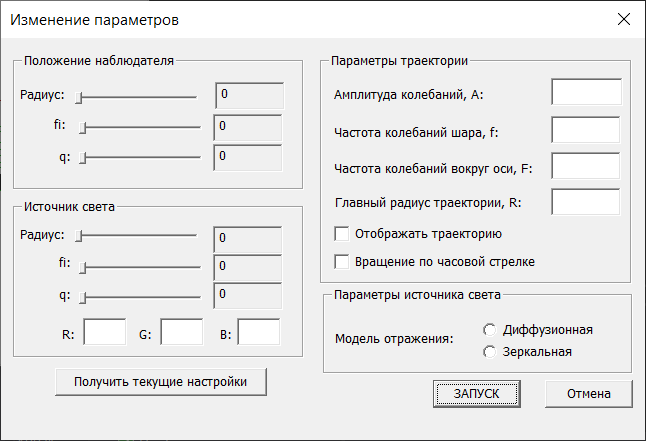


Рисунок 13 – Диалог изменения параметров.

В этом диалоговом окне можно установить все необходимые параметры модели: координаты камеры, координаты источника света, параметры траектории, а также модель отражения.

Поля ввода текста имеют валидацию, так как ввод некорректных параметров недопустим.

При нажатии на кнопку «Получить текущие настройки» форма заполняется текущими настройками (Рисунок 14).

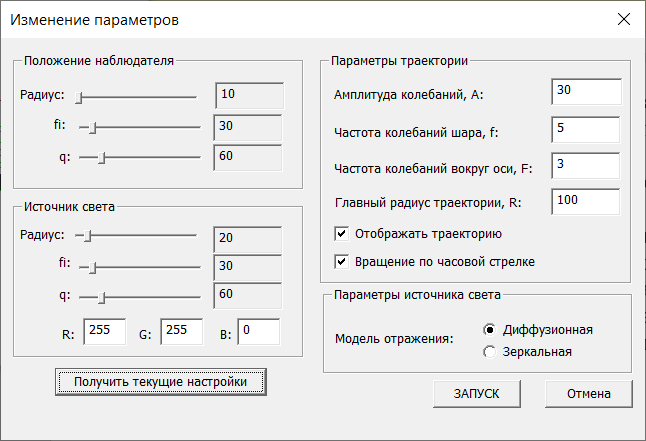


Рисунок 14 – Получение текущих настроек.

При нажатии на кнопку «Отмена» или «Х» в верхней правой части диалогового окна, окно закрывается без применения новых параметров. При нажатии на кнопку «Запуск» окно закрывается, применяя параметры и запуская модель движения шара на выполнение (рисунок 15).

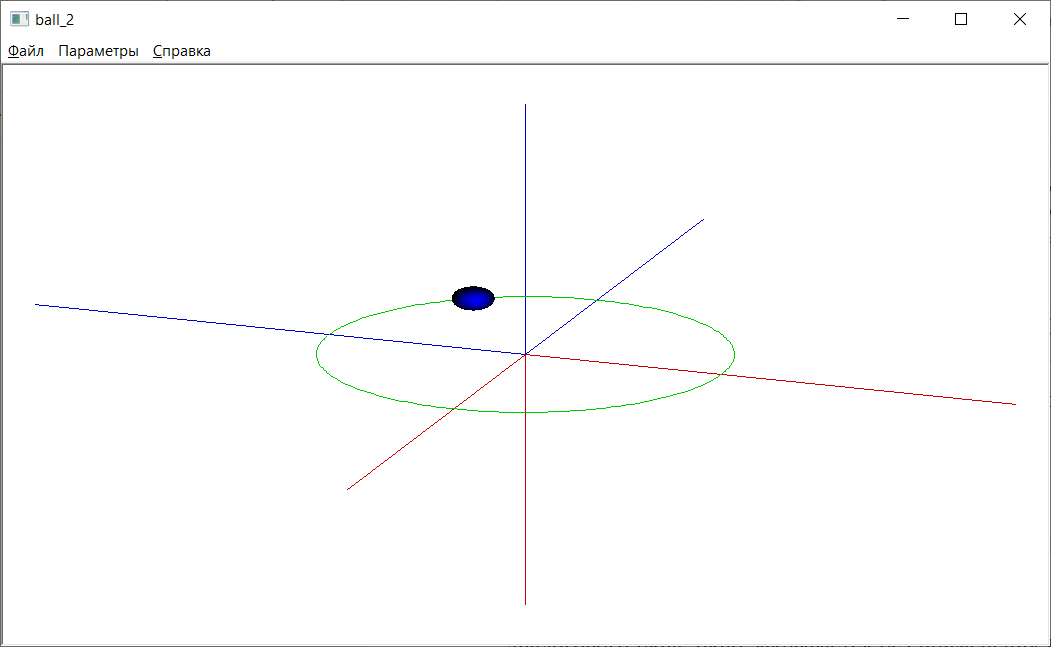


Рисунок 15 – Модель с новыми параметрами.

Также в приложении определены некоторые клавиши для изменения параметров и управления:

* Пробел – остановка таймера, открытие диалогового окна изменения параметров;
* Стрелки клавиатуры – управление положением камеры;
* W, A, S, D – управление положением источника света;
* O – зеркальная модель отражения;
* P – диффузионная модель отражения.

При выборе подпункта меню «Справка» «О проекте ball\_2…» открывается окно справки с краткой информацией о приложении (рисунок 16).

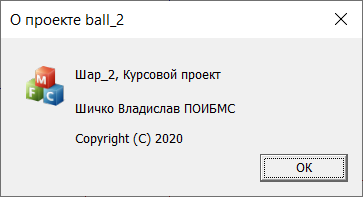


Рисунок 16 – Окно о проекте.

# Заключение

Разработанное приложение моделирует процесс движения шара по сложной траектории, которая состоит из двух независимых движений, а также моделирует освещение с разными типами отражения. Так как процесс происходил в трехмерном пространстве, была необходимость преобразования координат из мировых в оконные. Приложение позволяет пользователю задавать параметры модели, а также сохранять ее изображения в файлы формата bmp.

В ходе разработки приложения были приобретены навыки работы с библиотекой MFC C++.

# Приложения

## Приложение А. Блок-схема алгоритма обработки команд

Начало

Конец

Перерисовка модели

Ввод новых параметров

Отображение окна параметров

Поступила команда «Сохранить изображение»

Поступила команда «Изменить параметры»

Поступила команда «Получить справку»

Поступила команда «Выход» (Закрытие окна)

Ввод параметров сохранения

Отображение окна сохранения изображения

Отображение окна справки

да

да

да

да

нет

нет

нет

нет

## Приложение Б. Блок-схема алгоритма отображения модели

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

Отображение линий осей координат на экране

Получение массива точек траектории движения в соответствии с заданными параметрами

Начало

Отображение траектории на экране

Конец

Отображение шара в соответствии с диффузионной моделью отражения

Необходимо отображать траекторию?

Установка координат центра шара на следующую позицию на траектории

Какая модель отражения используется?

Отображение шара в соответствии с зеркальной моделью отражения

зеркальная

диффузионная

## Приложение В. Файл Sphere.cpp

#include "StdAfx.h"

#include "Sphere.h"

CSphere::CSphere(void)

{

current\_pos=0;

MV.RedimMatrix(4,4);

MW.RedimMatrix(3,3);

}

CSphere::~CSphere(void)

{

}

void CSphere::MV\_and\_MW(double OX,double OY,CRect& RW,CMatrix& PView){

MV=CreateViewCoord(PView(0),PView(1),PView(2)); //матрица пересчета из мировой в виддовую СК

CRectD RV(-OX,OY,OX,-OY);

MW=SpaceToWindow(RV,RW); //матрица пересчета из видовой в оконную СК

}

void CSphere::DrawOXYZ(CDC& dc,double OX,double OY,double OZ,CMatrix& PView,COLORREF color1,COLORREF color2){

CPen\* pen=new CPen(PS\_SOLID,1,color1);

CPen\* old\_pen=dc.SelectObject(pen);

CMatrix S0(4),SX(4),SY(4),SZ(4); //в мировой СК

CMatrix V0(4),VX(4),VY(4),VZ(4); //в видовой СК

CMatrix W0(3),WX(3),WY(3),WZ(3); //в оконной СК

S0(3)=1;

SX(0)=OX;SX(3)=1;

SY(1)=OY;SY(3)=1;

SZ(2)=-OZ;SZ(3)=1;

V0=MV\*S0;VX=MV\*SX;VY=MV\*SY;VZ=MV\*SZ; //пересчет из мировой в видовую СК

V0(2)=1;VX(2)=1;VY(2)=1;VZ(2)=1;

W0=MW\*V0.GetCol(0,0,2); //пересчет из видовой в оконную СК

WX=MW\*VX.GetCol(0,0,2);

WY=MW\*VY.GetCol(0,0,2);

WZ=MW\*VZ.GetCol(0,0,2);

CPoint p0(W0(0),W0(1)),pX(WX(0),WX(1)),pY(WY(0),WY(1)),pZ(WZ(0),WZ(1));

dc.MoveTo(p0);dc.LineTo(pX); //прорисовка положительных направлений осей координат

dc.MoveTo(p0);dc.LineTo(pY);

dc.MoveTo(p0);dc.LineTo(pZ);

pen=new CPen(PS\_SOLID,1,color2);

dc.SelectObject(pen);

dc.MoveTo(p0);dc.LineTo(p0.x-(pX.x-p0.x),p0.y-(pX.y-p0.y)); //прорисовка отрицательных направлений осей координат

dc.MoveTo(p0);dc.LineTo(p0.x-(pY.x-p0.x),p0.y-(pY.y-p0.y));

dc.MoveTo(p0);dc.LineTo(p0.x-(pZ.x-p0.x),p0.y-(pZ.y-p0.y));

dc.SelectObject(old\_pen);

}

void CSphere::DrawTraector(CDC& dc,double Radius,double tr\_A,double tr\_F,double tr\_f,double density, CMatrix& PView,COLORREF color, bool show\_trajectory){

CPen\* pen=new CPen(PS\_SOLID,1,color);

CPen\* old\_pen = dc.SelectObject(pen);

double step = pi/density;

int trajectory\_point\_amount = 2\*pi/(step\*abs(tr\_F));

S.RedimMatrix(4,trajectory\_point\_amount); //в мировой СК

CMatrix V(4,trajectory\_point\_amount); //в видовой СК

CMatrix W(3,trajectory\_point\_amount); //в оконной СК

for(int i=0;i<trajectory\_point\_amount;i++) {

S(0,i)=(Radius+tr\_A\*sin((double)i\*tr\_f))\*sin(i\*step\*tr\_F);

S(1,i)=(Radius+tr\_A\*sin((double)i\*tr\_f))\*cos(i\*step\*tr\_F);

S(2,i)=0;

S(3,i)=1;

}

for(int i=0;i<trajectory\_point\_amount;i++){

CMatrix m=MV\*S.GetCol(i); //пересчет из мировой в видовую СК

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

V(j,i)=m(j);

V(2,i)=1;

}

for(int i=0;i<trajectory\_point\_amount;i++){

CMatrix m=MW\*V.GetCol(i,0,2); //пересчет из видовой в оконную СК

for(int j=0;j<3;j++)

W(j,i)=m(j);

}

if (show\_trajectory) {

CPoint p0(W(0, 0), W(1, 0));

dc.MoveTo(p0);

for (int i = 1; i < trajectory\_point\_amount; i++) {

CPoint p(W(0, i), W(1, i));

dc.LineTo(p); //прорисовка линиями от точки к точке

}

dc.SelectObject(old\_pen);

}

}

void CSphere::DrawD(CDC& dc,double Radius,CMatrix& PView,CMatrix& PLight,COLORREF color){

BYTE R=GetRValue(color),G=GetGValue(color),B=GetBValue(color);

CMatrix VSphere(3),VSphereNorm(3),PV(4),VCart(3);

COLORREF col=color;

double df=(double)45/Radius/MW(0,0),

dq=(double)-45/Radius/MW(1,1),kLight=0;

double cos\_PN=0,cos\_RN=0;

CMatrix m=(MV\*S.GetCol(current\_pos)).GetCol(0,0,2); //положение центра шара в видовой СК

m(2)=1;

m=MW\*m; //переход к оконным координатам

CMatrix m2(4);m2(3)=1;

m2=(MV\*m2); //координаты начала мировых координат в видовой СК

m2.RedimData(3);

m2(2)=1;

m2=MW\*m2; //переход к оконным координатам

m=m-m2; //смещение центра шара от начала мировых координат в оконной СК

VSphere(0)=Radius;

//получение декартовых координат наблюдателя и источника освещения

CMatrix VR=SphereToCart(PView),VS=SphereToCart(PLight);

//перенос начала координат в центр шара, находящийся в заданной точке траектории

//т.е. перемещение наблюдателя и источника света

VR.RedimData(4);VR(3)=1;

/\*VR=Translate3D(S(0,current\_pos),S(1,current\_pos),S(2,current\_pos))\*VR;\*/

VS.RedimData(4);VS(3)=1;

VS=Translate3D(S(0,current\_pos),S(1,current\_pos),S(2,current\_pos))\*VS;

for(double fi=0;fi<=180.0;fi+=df){ //обход точек поверхности шара

for(double q=0;q<=180.0;q+=dq){

//обход левой половины

VSphere(1)=PView(1)+fi;

VSphere(2)=q;

VCart=SphereToCart(VSphere); //переход от сферических к декартовым координатам точки на поверхности шара

VSphereNorm=VCart;

PV(0)=VCart(0);

PV(1)=VCart(1);

PV(2)=VCart(2);

PV(3)=1;

PV=MV\*PV; //переход от мировой к видовой системе координат

VCart(0)=PV(0);

VCart(1)=PV(1);

VCart(2)=1;

VCart=MW\*VCart; //переход от видовой к оконной СК

cos\_PN=cosViV2(VS,VSphereNorm);

if(cos\_PN<=0){

//установка освещенности точки в зависимости от угла между

//нормалью к поверхности сферы и вектором к источнику освещения

kLight=-cos\_PN;

}

else kLight=0;

col=RGB(R\*kLight,G\*kLight,B\*kLight);

VCart=VCart+m; //смещение в оконной СК относительно начала координат

dc.SetPixel(VCart(0),VCart(1),col); //закрашивание точки на поверхности шара

//обход правой половины

VSphere(1)=PView(1)-fi;

VSphere(2)=q;

VCart=SphereToCart(VSphere); //переход от сферических к декартовым координатам точки на поверхности шара

VSphereNorm=VCart;

PV(0)=VCart(0);

PV(1)=VCart(1);

PV(2)=VCart(2);

PV(3)=1;

PV=MV\*PV; //переход от мировой к видовой системе координат

VCart(0)=PV(0);

VCart(1)=PV(1);

VCart(2)=1;

VCart=MW\*VCart; //переход от видовой к оконной СК

cos\_PN=cosViV2(VS,VSphereNorm);

if(cos\_PN<=0){

//установка освещенности точки в зависимости от угла между

//нормалью к поверхности сферы и вектором к источнику освещения

kLight=-cos\_PN;

}

else kLight=0;

col=RGB(R\*kLight,G\*kLight,B\*kLight);

VCart=VCart+m; //смещение в оконной СК относительно начала координат

dc.SetPixel(VCart(0),VCart(1),col); //закрашивание точки на поверхности шара

}

}

}

void CSphere::DrawZ(CDC& dc,double Radius,CMatrix& PView,CMatrix& PLight,COLORREF color){

BYTE R=GetRValue(color),G=GetGValue(color),B=GetBValue(color);

CMatrix VSphere(3),VSphereNorm(3),PV(4),VCart(3);

COLORREF col=color;

double df=(double)45/Radius/MW(0,0),dq=(double)-45/Radius/MW(1,1),kLight=0;

double cos\_RN,cos\_PN,xx,cos\_A;

CMatrix m=(MV\*S.GetCol(current\_pos)).GetCol(0,0,2); //положение центра шара в видовой СК

m(2)=1;

m=MW\*m; //переход к оконным координатам

CMatrix m2(4);m2(3)=1;

m2=(MV\*m2); //координаты начала мировых координат в видовой СК

m2.RedimData(3);

m2(2)=1;

m2=MW\*m2; //переход к оконным координатам

m=m-m2; //смещение центра шара от начала мировых координат в оконной СК

VSphere(0)=Radius;

//получение декартовых координат наблюдателя и источника освещения

CMatrix VR=SphereToCart(PView),VS=SphereToCart(PLight);

//перенос начала координат в центр шара, находящийся в заданной точке траектории

//т.е. перемещение наблюдателя и источника света

VR.RedimData(4);VR(3)=1;

VR=Translate3D(S(0,current\_pos),S(1,current\_pos),S(2,current\_pos))\*VR;

VS.RedimData(4);VS(3)=1;

VS=Translate3D(S(0,current\_pos),S(1,current\_pos),S(2,current\_pos))\*VS;

for(double fi=0;fi<=180.0;fi+=df){ //обход точек сферы

for(double q=0;q<=180.0;q+=dq){

//левая половина

VSphere(1)=PView(1)+fi;

VSphere(2)=q;

VCart=SphereToCart(VSphere);

VSphereNorm=VCart;

PV(0)=VCart(0);

PV(1)=VCart(1);

PV(2)=VCart(2);

PV(3)=1;

PV=MV\*PV; //переход от мировой к видовой системе координат

VCart(0)=PV(0);

VCart(1)=PV(1);

VCart(2)=1;

VCart=MW\*VCart; //переход от видовой к оконной системе координат

cos\_PN=cosViV2(VS,VSphereNorm);

if(cos\_PN<0){

xx=2\*VS.Abs()\*cos\_PN/VSphereNorm.Abs();

VS.RedimData(3);

CMatrix RX=-VSphereNorm\*xx+VS;

cos\_A=cosViV2(RX,VR);

//установка освещенности точки в зависимости от угла между

//вектором к наблюдателю и вектором к источнику освещения

if(cos\_A<0)

kLight=cos\_A\*cos\_A;

else kLight=0;

}

else kLight=0;

col=RGB(R\*kLight,G\*kLight,B\*kLight);

VCart=VCart+m; //смещение в оконной СК относительно начала координат

dc.SetPixel(VCart(0),VCart(1),col);

//правая половина

VSphere(1)=PView(1)-fi;

VSphere(2)=q;

VCart=SphereToCart(VSphere);

VSphereNorm=VCart;

PV(0)=VCart(0);

PV(1)=VCart(1);

PV(2)=VCart(2);

PV(3)=1;

PV=MV\*PV; //переход от мировой к видовой системе координат

VCart(0)=PV(0);

VCart(1)=PV(1);

VCart(2)=1;

VCart=MW\*VCart; //переход от видовой к оконной системе координат

cos\_PN=cosViV2(VS,VSphereNorm);

if(cos\_PN<0){

xx=2\*VS.Abs()\*cos\_PN/VSphereNorm.Abs();

VS.RedimData(3);

CMatrix RX=-VSphereNorm\*xx+VS;

cos\_A=cosViV2(RX,VR);

//установка освещенности точки в зависимости от угла между

//вектором к наблюдателю и вектором к источнику освещения

if(cos\_A<0)

kLight=cos\_A\*cos\_A;

else kLight=0;

}

else kLight=0;

col=RGB(R\*kLight,G\*kLight,B\*kLight);

VCart=VCart+m; //смещение в оконной СК относительно начала координат

dc.SetPixel(VCart(0),VCart(1),col);

}

}

}

void CSphere::SetPos(int pos){

current\_pos=pos;

}

int CSphere::GetPos(){

return current\_pos;

}

# Список литературы

1. Давыдов В. - Visual C++. Разработка Windows-приложений с помощью MFC и API-функций. – СПб.: БХВ-Петербург, 2008. – 576с.: ил.
2. MFC шаг за шагом [Электронный ресурс]. – Электронные данные. – Режим доступа – http://www.firststeps.ru/mfc/steps/   
   Дата доступа: 27.11.2020
3. Порев В. – «Компьютерная графика. Учебное пособие» – СПб.: БХВ-Петербург, 2002. – 432с.: ил.