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 «Вертолет»

Исполнитель

студент III курса группы 7  Г.С. Бобрович

Руководитель

к. т.н., ассистент \_\_\_\_\_\_\_\_\_\_ Р. Ю. Скепко

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

Руководитель\_\_\_\_\_\_\_\_\_\_ Р. Ю. Скепко

Минск 2022

ОГЛАВЛЕНИЕ

[ЗАДАНИЕ 3](#_Toc121870300)

[ВВЕДЕНИЕ 4](#_Toc121870301)

[ОПИСАНИЕ АЛГОРИТМОВ РЕШЕНИЯ ЗАДАЧИ 5](#_Toc121870302)

[МАТЕМАТИЧЕСКОЕ ОПИСАНИЕ АЛГОРИТМОВ 10](#_Toc121870306)

[ПОЯСНЕНИЯ К ЛИСТИНГАМ ПРИЛОЖЕНИЯ 19](#_Toc121870307)

[ЗАКЛЮЧЕНИЕ 21](#_Toc121870308)

[ЛИСТИНГ ПРИЛОЖЕНИЯ 26](#_Toc121870309)

[ПРИЛОЖЕНИЕ А 26](#_Toc121870310)

[ПРИЛОЖЕНИЕ Б 28](#_Toc121870311)

[ПРИЛОЖЕНИЕ В 30](#_Toc121870312)

[ПРИЛОЖЕНИЕ Г 31](#_Toc121870313)

[ПРИЛОЖЕНИЕ Д 34](#_Toc121870314)

[ПРИЛОЖЕНИЕ Е 35](#_Toc121870315)

[СПИСОК ЛИТЕРАТУРЫ 36](#_Toc121870316)

# ЗАДАНИЕ

Создать приложение Windows для графического 2D -изображения летящего вертолета (вид сверху), рисунок 1.

Изменяемые параметры вертолета:

* период вращения одной лопасти вертолета ;
* скорость перемещения вертолета (при большем T больше скорость и наоборот);
* цвет лопастей;
* цвет корпуса.

Реализовать режим зависания (нет перемещения при вращающихся лопастях)

Реализовать режим вращения машины вокруг оси винта при нажатии определенной клавиши. При повторном нажатии вертолет должен лететь в том направлении, в котором было прекращено вращение.

Обеспечить перемещение вертолета вправо относительно вектора скорости.

Для установки параметров модели использовать окно диалога.

Рисунок 1 – схематичный вид вертолета сверху

# ВВЕДЕНИЕ

Компьютерная графика (также машинная графика) — область деятельности, в которой компьютеры наряду со специальным программным обеспечением используются в качестве инструмента как для создания (синтеза) и редактирования изображений, так и для оцифровки визуальной информации, полученной из реального мира, с целью дальнейшей её обработки и хранения.

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. Основными задачами машинной графики являются ввод (считывание) графической информации в ЭВМ, вывод ее из ЭВМ (формирование изображений), а также определенного рода переработка информации в компьютере. Таким образом, основные задачи машинной геометрии или, как говорят, автоматизированного геометрического моделирования и конструирования — синтез в ЭВМ и анализ геометрических объектов, решение задач геометрического характера.

Главной задачей данной курсовой работы является разработка приложения для построения 3D изображения и перемещения фигур, представляющих собой вертолет.

Для отображения цвета поверхностей будет использована цветовая модель RGB (рисунок 1.1). За основу этой цветовой модели принято три основных цвета R(Red), G(Green), B(Blue). В модели RGB любой цвет получается в результате сложения этих трех основных цветов.

RGB — это аддитивная цветовая модель. Другими словами, чтобы получить другие цвета, вы смешиваете основной красный, зеленый и синий цвета. Если вы смешаете все три цвета с максимальной интенсивностью (100%), вы получите белый цвет. С другой стороны, если вы смешаете их все с минимальной интенсивностью (0%), вы получите черный цвет.

Для модели RGB каждая из компонент может представляться числами, с ограниченным диапазоном от [0; 255;] так называемым - True Color в котором каждая компонента представлена в виде байта, что даёт 256 градаций для каждой компоненты. Количество возможных сочетаний цветов составляет 256\*256\*256 = 16.7 млн цветов ().

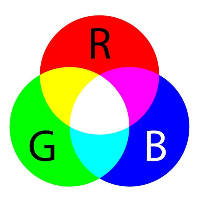


Рисунок 1.1 - Цветовая модель RGB

# ОПИСАНИЕ АЛГОРИТМОВ РЕШЕНИЯ ЗАДАЧИ

## Общая схема приложения

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

Начало

Установка параметров в начальные значения, создание окна, запуск прослушивателей 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 прослушивателей сообщений выполняются по стандарту платформы MFC для приложений без использования архитектуры «Документ/Вид». Для хранения и передачи параметров был доработан класс CChildView (приложение А). Завершение работы приложения выполняется также по стандарту платформы MFC.

## Получение и обработка команд

Цикл получения и обработки команд можно изобразить следующим образом:

Начало

Конец

Регуляция положения вертолета нажатием клавиш

Установка параметров в диалоге

(рисунок 2)

Зависание

Движение вперед

нет

Вращение винта и перерисовка вертолета

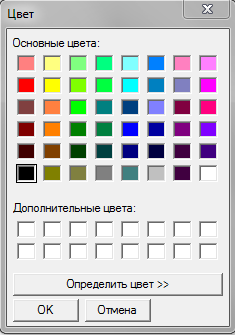
Вызов справки из меню

Команда меню «Выход»

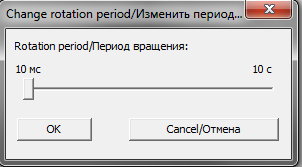
Отображение справочного окна

При помощи обработчика сообщений от клавиатуры реализованы движение в боковых направлениях и режим зависания.

Через пункты меню «Правка» доступен вызов диалогов настройки цвета винта, цвета лопастей, периода вращения лопастей (рисунок 2). Обработка движения слайдера в диалоге настройки периода вращения реализована при помощи механизма DoDataeXchange, являющегося полезным инструментом взаимодействия приложения на платформе MFC с пользователем (приложение Б).



а)



б)

Рисунок 2 – окна диалогов настройки цвета (а) и периода вращения (б)

## 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. 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 всего вертолета и поворота лопастей в методе прорисовки вертолета вызываются реализованные аффинные преобразования в соответствии с математическим аппаратом компьютерной геометрии и графики (приложение Г).

Для решения задачи согласования моделируемого процесса с изображением на экране использовались матрицы пересчета из мировых координат в оконные. Расчет данных матриц выполнялся в соответствии с математическим аппаратом компьютерной геометрии и графики. Матрица пересчета из видовых координат в оконные формируется функцией SpaceToWindow. Определения функций представлены в приложении Д.

Процесс прорисовки вертолета в окне приложения можно продемонстрировать следующей блок-схемой:

Конец

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

Перемещение точек корпуса и винта на текущие координаты

Поворот винта на текущий угол

Прорисовка корпуса и винта при помощи функции контекста окна DrawPolygon

Начало

Получение точек, соответствующих корпусу и винту в контексте окна

# МАТЕМАТИЧЕСКОЕ ОПИСАНИЕ АЛГОРИТМОВ

Основным алгоритмом для решения этой задачи будет являться преобразование координат из мировой системы в оконную с дальнейшими преобразованиями. В случае данной задачи необходимо изменять координаты лопастей путём аффинных преобразований. С их помощью мы будем поворачивать лопасти на нужный нам угол относительно той точки, где эти лопасти крепятся.

Исходя из вышеперечисленного следует, что вращение лопастей будет осуществляться за счёт вращения точек (угловое преобразование) относительно определённой точки, которая в дальнейшем послужит центром для оси вращения точек, из которых состоят лопасти. В таком случае нам придётся использовать преобразование сдвига совместно с угловым преобразованием.

Все преобразования будут проводиться в мировой системе координат, и только потом приводится в оконную. Все точки располагаются в одной мировой системе координат.

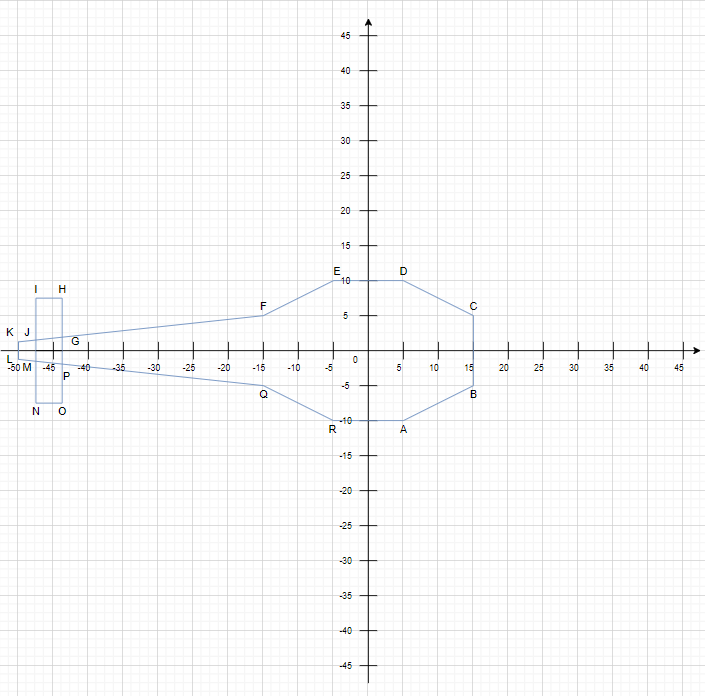
**Построение вертолета**

1. **Построение корпуса**

Корпус будет построен с помощью отрезков:

**AB,BC,CD,DE,EF,**

**FG,GH,HI,IL,LM,MN,NO,OP,PQ,QR,RA**

****

**Координаты точек:**

A=(5,-10) L=(-50,-1)

B=(15,-5) M=(-47,-1.5)

C=(15,5) N=(-47,-7.5)

D=(5,10) O=(-44,-7.5)

E=(-5,10) P=(-44,-2)

F=(-15,5) Q=(-15,-5)

G=(-44,2) R=(-5,-10)

H=(-44,7.5) A=(5,-10)

I=(-47,7.5)

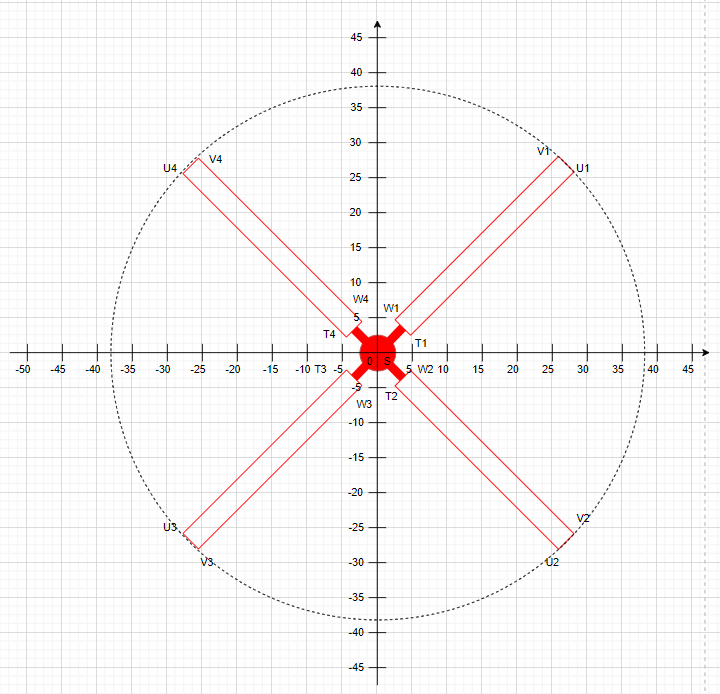
J=(-47,1.5)

K=(-50,1)

1. **Построение винтов**

Винты будут построены с помощью отрезков: **T1U1,U1V1,V1W1,W1T1,T2U2,U2V2,V2W2,W2T2,T3U3,U3,V3,V3W3,W3T3,T4U4,U4V4,V4W4,W4T4.**

Точка **S-**находится на 0-координатах и является пересечением креплений винтов.

****

**Координаты точек:**

S=0 R(S)=37.5

T1=(5,2.5) T3=(-5,-2.5)

U1=(27.5,26) U3=(-27.5,-26)

V1=(26,27.5) V3=(-26,-27.5)

W1=(2.5,5) W3=(-2.5,-5)

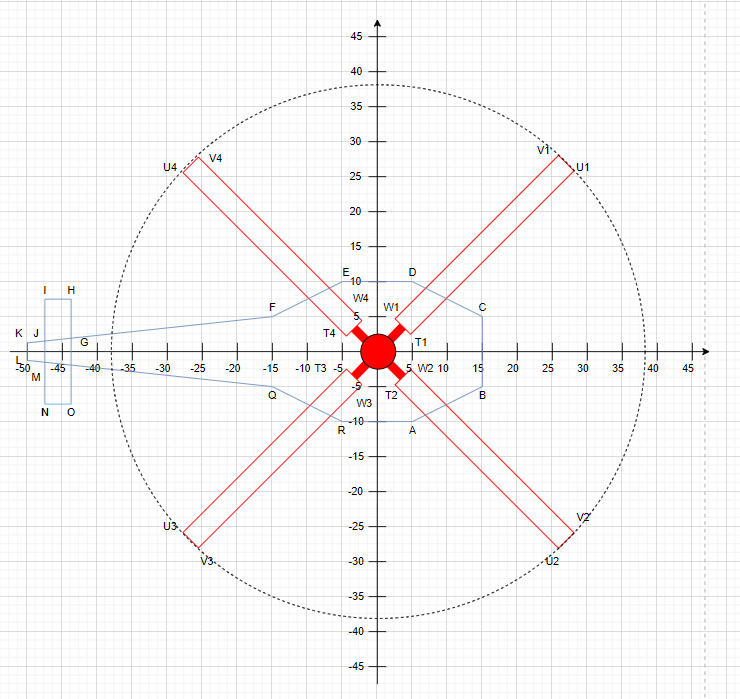
T2=(2.5,-5) T4=(-2.5,5)

U2=(26,-27.5) U4=(-26,27.5)

V2=(27.5,-26) V4=(27.5,-26)

W2=(5,-2.5) W4=(-5,2.5)

1. **Общий вид вертолета**

****

**Сдвиг объекта**

Преобразования сдвига мы будем использовать для того, чтобы сместить систему координат к точке, относительно которой будет происходить поворот. При данном преобразовании координаты точки на оси изменяться, но своё местоположение в пространстве оно не изменит. Данное преобразование имеет следующий вид:

Здесь x,y – начальные координаты точки в системе X0Y. - новые координаты точки, куда будет смещена система координат. –новые координаты точки. Значения представлены в следующей системе:

В матричном виде преобразование сдвига будет иметь вид:

Здесь и далее индекс «s» означает, что изменению подвергается система координат.

После преобразования по формуле (4) нужно повернуть точку относительно нового положения центра системы координат.

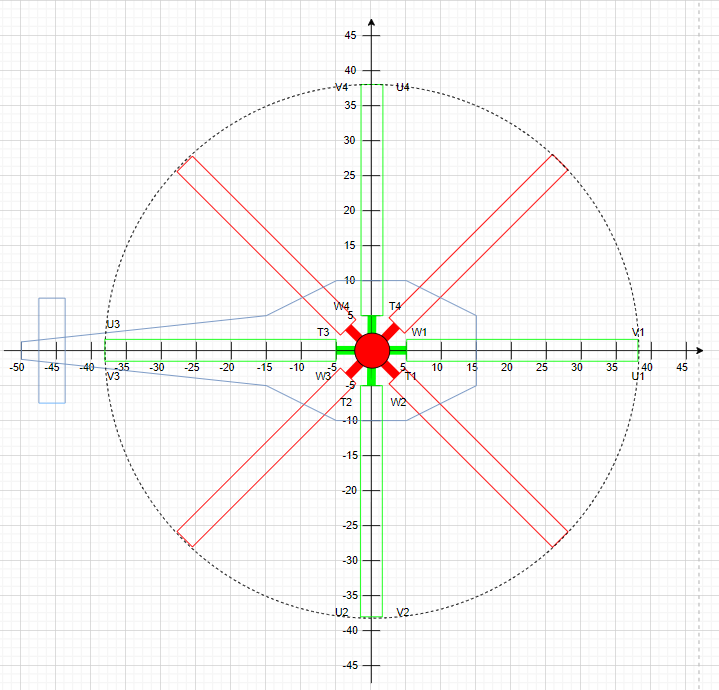
**Смещение точек**

Для смещения точек будет применена следующая формула:



Где a – угол смещения, X1 и Y1 – текущие координаты точек, X и Y – новые координаты точек. При этом чем меньше угол поворота, тем плавнее будут двигаться лопасти. Для примера сместим точки лопастей на a = 45o. Получившиеся лопасти помечены зеленым цветом.

**Смещение точек винтов на 45°**

****

**Точки винтов после смещения на 45° по часовой стрелке:**

S=0 R(S)=37.5

T1=(5,-1) T3=(-5,1)

U1=(38,-1) U3=(-38,1)

V1=(38,1) V3=(-38,-1)

W1=(5,1) W3=(-5,-1)

T2=(-1,-5) T4=(1,5)

U2=(-1,-38) U4=(1,38)

V2=(1,-38) V4=(-1,38)

W2=(1,-5) W4=(-1,5)

**Обратное преобразование**

После того, как мы повернули точку, нужно будет вернуть систему координат X’0Y’ в прежнее положение. Для этого нужно будет выполнить обратное преобразование сдвига, которое будет иметь вид:

**Преобразование из мировой в оконную систему координат**

Теперь после проведения преобразований в мировой системе координат их нужно привести в оконную. По условию задачи при изменении размера окна изображение соответствующим образом масштабируется, значит, нужно использовать анизотропный режим. Данный режим позволяет не только менять направление осей, но также задавать каждой оси свой масштаб. Для начала запишем общее уравнение перевода мировых координат в оконные для точки M = M(x,y). В оконной системе координат уравнение точки будет иметь вид Тогда уравнение для перевода в оконные координаты будет иметь вид:

(12)

где - ширина области отображения в оконных координатах,

**-** ширина области отображения в мировых координатах,

- высота области отображения в оконных координатах,

- высота области отображения в мировых координатах.

Эту систему можно переписать в виде

(13)

где .

Равенство (13) можно представить в матричной форме, тогда оно будет иметь вид

(14)

или

(15)

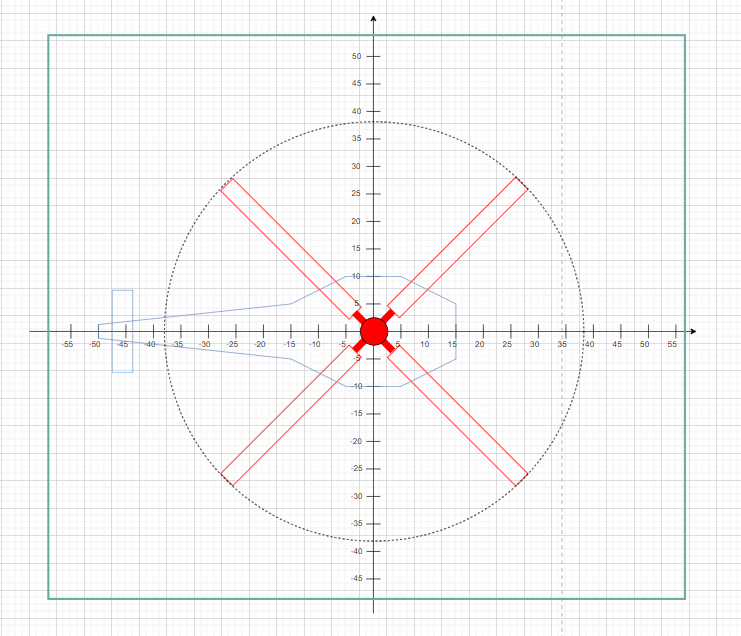
где

Раскрыв скобки в равенстве (13) и переставив элементы, мы можем привести к более удобной форме для вычисления в матричной форме:

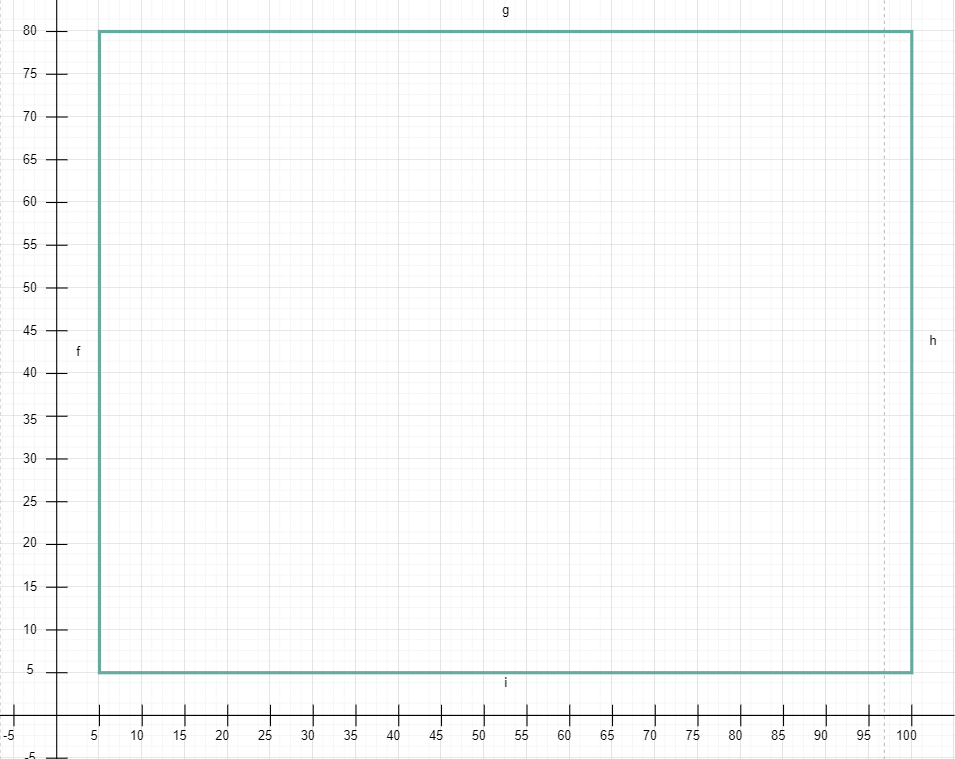
или

где

Далее зададим область отображения в мировых координатах

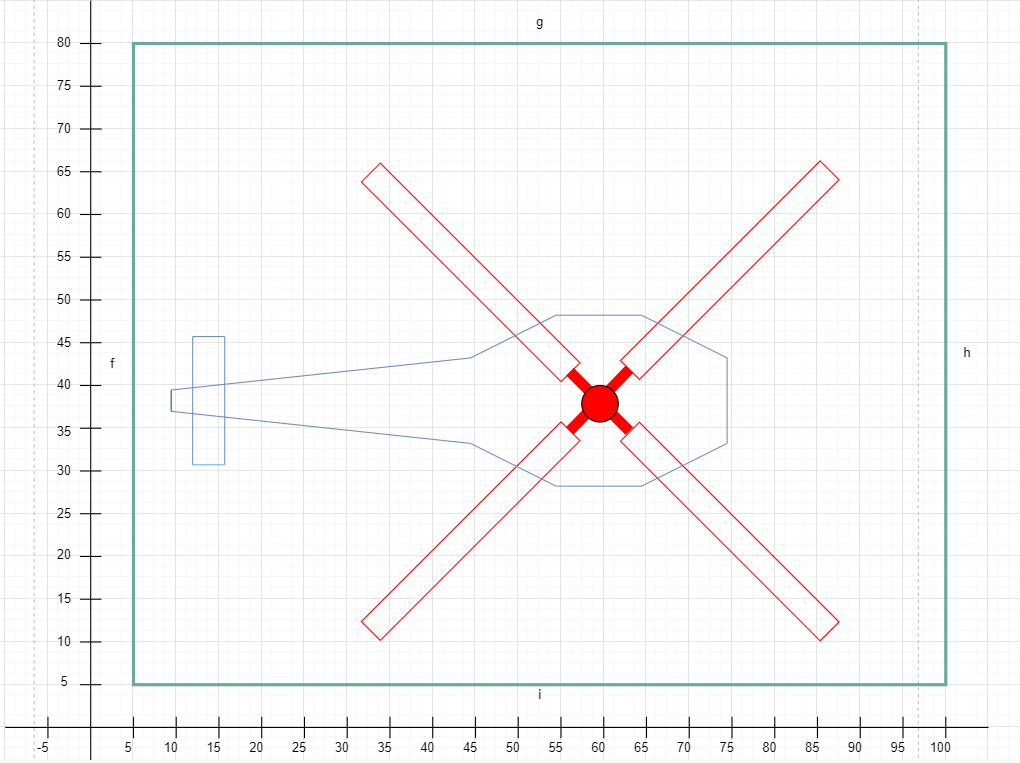


И аналогичную для оконных координат



Подставим значения в формулу перевода:

После расчетов итоговое изображение будет выглядеть следующим образом:



**Отрисовка полученного результата**

Отрисовка происходит с помощью программных функций, с учетом полученных в предыдущих пунктах результатов.

# ПОЯСНЕНИЯ К ЛИСТИНГАМ ПРИЛОЖЕНИЯ

**ChildView.cpp**

В данной части кода задаются изначальные характеристики вертолета, такие как: время зависания, режим зависания(в данном случае равно true, что ставит вертолет в положении зависания), координаты изначальной позиции вертолета, цвет лопастей и цвет лопасти в формате RGB, стартовая скорость вращения лопастей.

// CChildView

CChildView::CChildView()

{

Timer=false; //время зависания

Hovering=true; //режим зависания

RW.SetRect(300,300,400,400);

ColorOfHelicopterBlade=RGB(255,255,255); //изначальный цвет лопастей

ColorOfHelicopterBody=RGB(255,255,255); //изначальный цвет корпуса

Т=3000; //стартовая скорость вращения лопастей

}

В этом участке кода происходит обработка сообщений из основного окна приложения.

// обработчики сообщений CChildView

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;

}

В этом участке кода осуществлена обработка нажатия кнопок на клавиатуре, которые отвечают за перемещения вертолета.

// обработчик нажатия кнопок

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

if(nChar==38){ //вверх

HP.Move(0,-5);

Invalidate();

}

else if(nChar==40){ //вниз

HP.Move(0,5);

Invalidate();

}

else if(nChar==32){ //вперед

Hovering=!Hovering;

}

}

//вызов диалога установки цвета лопастей

void CChildView::On32771()

{

CColorDialog DiagColor;

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

ColorOfHelicopterBlade=DiagColor.GetColor();

Invalidate();

}

}

//вызов диалога установки цвета корпуса

void CChildView::On32772()

{

CColorDialog DiagColor;

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

ColorOfHelicopterBody=DiagColor.GetColor();

Invalidate();

}

}

//вызов диалога установки периода вращения

void CChildView::On32773()

{

Settings MyDialog;

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

Т=MyDialog.GetT();

Invalidate();

}

}

**CMatrix.h**

В данном классе объявляются переменные и конструкторы, которые нам пригодятся в дальнейших математических расчетах и отрисовки вертолета.

class CMatrix

{

double \*\*array;

int n\_rows; **//Число строк**

int n\_cols; **//Число столбцов**

public:

CMatrix(); **// Конструктор по умолчанию (1 на 1)**

CMatrix(int,int); **//Конструктор**

CMatrix(int); **//Конструктор -вектора (один столбец)**

CMatrix(const CMatrix&); **// Конструктор копирования**

~CMatrix();

double &operator()(int,int); **//Выбор элемента матрицы по индексу**

double &operator()(int); **//Выбор элемента вектора по индексу**

CMatrix operator-(); **//Оператор "-"**

CMatrix operator=(const CMatrix&); **//Оператор "Присвоить":M1=M2**

CMatrix operator\*(CMatrix&); **//Оператор "Произведение": М1\*М**

CMatrix operator\*(double x); **//Оператор "Произведение": М1\*a**

CMatrix operator/(double x); **//Оператор "Деление": М1/a**

CMatrix operator+(CMatrix&); **//Оператор "+": M1+M2**

CMatrix operator-(CMatrix&); **//Оператор "-": M1-M2**

CMatrix operator+(double); **//Оператор "+": M+a**

CMatrix operator-(double); **//Оператор "-": M-a**

int rows()const{return n\_rows;}; **//Возвращает число строк**

int cols()const{return n\_cols;}; **//Возвращает число строк**

CMatrix Transp(); **//Возвращает матрицу,транспонированную к текущей**

CMatrix GetRow(int); **//Возвращает строку по номеру**

CMatrix GetRow(int,int,int);

CMatrix GetCol(int); **//Возвращает столбец по номеру**

CMatrix GetCol(int,int,int);

CMatrix RedimMatrix(int,int); **//Изменяет размер матрицы с уничтожением данных**

CMatrix RedimData(int,int); **//Изменяет размер матрицы с сохранением данных, которые можно сохранить**

CMatrix RedimMatrix(int); **//Изменяет размер матрицы с уничтожением данных**

CMatrix RedimData(int); **//Изменяет размер матрицы с сохранением данных, которые можно сохранить**

double MaxElement(); **//Максимальный элемент матрицы**

double MinElement(); **//Минимальный элемент матрицы**

double Abs(); **//Модуль матрицы**

};

**HelicopterPoints.cpp**

В данном участке кода определяются координаты, по которым будет отрисовываться корпус и винты вертолета, а именно координаты точек корпуса и координаты точек всех четырех винтов

HelicopterPoints::HelicopterPoints(void)

{

//угол

Corner = 0; X = 0; Y = 0;

//устаналиваем начальное положение корпуса и винта

HelicopterBodyPoints.RedimMatrix(3,18);

HelicopterRotorPoints.RedimMatrix(3,28);

//точки корпуса

for(int i=0;i<18;i++)

{

HelicopterBodyPoints(2,i)=1;

}

//точки винта

for(int i=0;i<28;i++)

{

HelicopterRotorPoints(2,i)=1;

}

HelicopterBodyPoints(0,0)=-10;

HelicopterBodyPoints(1,0)=20;

HelicopterBodyPoints(0,1)=10;

HelicopterBodyPoints(1,1)=20;

HelicopterBodyPoints(0,2)=20;

HelicopterBodyPoints(1,2)=10;

HelicopterBodyPoints(0,3)=20;

HelicopterBodyPoints(1,3)=-10;

HelicopterBodyPoints(0,4)=10;

HelicopterBodyPoints(1,4)=-20;

HelicopterBodyPoints(0,5)=-10;

HelicopterBodyPoints(1,5)=-20;

HelicopterBodyPoints(0,6)=-30;

HelicopterBodyPoints(1,6)=-15;

HelicopterBodyPoints(0,7)=-130;

HelicopterBodyPoints(1,7)=-5;

HelicopterBodyPoints(0,8)=-130;

HelicopterBodyPoints(1,8)=-15;

HelicopterBodyPoints(0,9)=-135;

HelicopterBodyPoints(1,9)=-15;

HelicopterBodyPoints(0,10)=-135;

HelicopterBodyPoints(1,10)=-5;

HelicopterBodyPoints(0,11)=-140;

HelicopterBodyPoints(1,11)=-5;

HelicopterBodyPoints(0,12)=-140;

HelicopterBodyPoints(1,12)=5;

HelicopterBodyPoints(0,13)=-135;

HelicopterBodyPoints(1,13)=5;

HelicopterBodyPoints(0,14)=-135;

HelicopterBodyPoints(1,14)=15;

HelicopterBodyPoints(0,15)=-130;

HelicopterBodyPoints(1,15)=15;

HelicopterBodyPoints(0,16)=-130;

HelicopterBodyPoints(1,16)=5;

HelicopterBodyPoints(0,17)=-30;

HelicopterBodyPoints(1,17)=15;

HelicopterRotorPoints(0,0)=-5;

HelicopterRotorPoints(1,0)=5;

HelicopterRotorPoints(0,1)=-5;

HelicopterRotorPoints(1,1)=15;

HelicopterRotorPoints(0,2)=-10;

HelicopterRotorPoints(1,2)=15;

HelicopterRotorPoints(0,3)=-10;

HelicopterRotorPoints(1,3)=100;

HelicopterRotorPoints(0,4)=10;

HelicopterRotorPoints(1,4)=100;

HelicopterRotorPoints(0,5)=10;

HelicopterRotorPoints(1,5)=15;

HelicopterRotorPoints(0,6)=5;

HelicopterRotorPoints(1,6)=15;

HelicopterRotorPoints(0,7)=5;

HelicopterRotorPoints(1,7)=5;

HelicopterRotorPoints(0,8)=15;

HelicopterRotorPoints(1,8)=5;

HelicopterRotorPoints(0,9)=15;

HelicopterRotorPoints(1,9)=10;

HelicopterRotorPoints(0,10)=100;

HelicopterRotorPoints(1,10)=10;

HelicopterRotorPoints(0,11)=100;

HelicopterRotorPoints(1,11)=-10;

HelicopterRotorPoints(0,12)=15;

HelicopterRotorPoints(1,12)=-10;

HelicopterRotorPoints(0,13)=15;

HelicopterRotorPoints(1,13)=-5;

HelicopterRotorPoints(0,14)=5;

HelicopterRotorPoints(1,14)=-5;

HelicopterRotorPoints(0,15)=5;

HelicopterRotorPoints(1,15)=-15;

HelicopterRotorPoints(0,16)=10;

HelicopterRotorPoints(1,16)=-15;

HelicopterRotorPoints(0,17)=10;

HelicopterRotorPoints(1,17)=-100;

HelicopterRotorPoints(0,18)=-10;

HelicopterRotorPoints(1,18)=-100;

HelicopterRotorPoints(0,19)=-10;

HelicopterRotorPoints(1,19)=-15;

HelicopterRotorPoints(0,20)=-5;

HelicopterRotorPoints(1,20)=-15;

HelicopterRotorPoints(0,21)=-5;

HelicopterRotorPoints(1,21)=-5;

HelicopterRotorPoints(0,22)=-15;

HelicopterRotorPoints(1,22)=-5;

HelicopterRotorPoints(0,23)=-15;

HelicopterRotorPoints(1,23)=-10;

HelicopterRotorPoints(0,24)=-100;

HelicopterRotorPoints(1,24)=-10;

HelicopterRotorPoints(0,25)=-100;

HelicopterRotorPoints(1,25)=10;

HelicopterRotorPoints(0,26)=-15;

HelicopterRotorPoints(1,26)=10;

HelicopterRotorPoints(0,27)=-15;

HelicopterRotorPoints(1,27)=5;

rs.SetRectD(-50,-50,50,50);

}

void HelicopterPoints::Draw(CDC& dc,CRect& RW,COLORREF ColorOfHelicopterBlade,COLORREF ColorOfHelicopterBody){

CBrush\* br=new CBrush(ColorOfHelicopterBody);//кисть для рисования корпуса

//матрица пересчета из мировой системы координат (МСК) в оконную (ОСК)

//с учетом смещения вертолета в МСК

CMatrix K=SpaceToWindow(rs,RW)\*Translate2D(X,Y);

CMatrix KV=Rotate2D(Corner);//матрица пересчета для поворота лопастей

CPoint\* Points=new CPoint[18];//точки корпуса в ОСК

for(int i=0;i<18;i++){

CMatrix W(3);

W=K\*HelicopterBodyPoints.GetCol(i);//пересчет координат точек корпуса из МСК в ОСК

Points[i].x=W(0);

Points[i].y=W(1);

}

dc.SelectObject(br);//выбор кисти

dc.Polygon(Points,18);//прорисовка корпуса

Points=new CPoint[28];

for(int i=0;i<28;i++){

CMatrix W(3);

W=K\*KV\*HelicopterRotorPoints.GetCol(i);//пересчет коорцинат точек лопастей из МСК в ОСК

Points[i].x=W(0);

Points[i].y=W(1);

}

br=new CBrush(ColorOfHelicopterBlade);//кисть цвета лопастей

dc.SelectObject(br);//выбор кисти

dc.Polygon(Points,28);//прорисовка лопастей

//кружочек - центр вращения лопастей

CMatrix E(3,2);

E(0,0)=-10;E(1,0)=-10;E(2,0)=1;

E(0,1)=10;E(1,1)=10;E(2,1)=1;

for(int i=0;i<2;i++){

CMatrix W(3);

W=K\*E.GetCol(i);//пересчет координат кружочка из МСК в ОСК

Points[i].x=W(0);

Points[i].y=W(1);

}

dc.Ellipse(Points[0].x,Points[0].y,Points[1].x,Points[1].y);

delete Points;

delete br;

}

void HelicopterPoints::RotorTurn(double Crnr){

Corner+=Crnr;

}

//движение вертолета с проверкой вылета за предел экрана

void HelicopterPoints::Move(double iX,double iY){

X+=iX;

if(X>700)X-=1400;

Y+=iY;

if(Y>700)Y-=1400;

else if(Y<0)Y+=1400; }

# ЗАКЛЮЧЕНИЕ

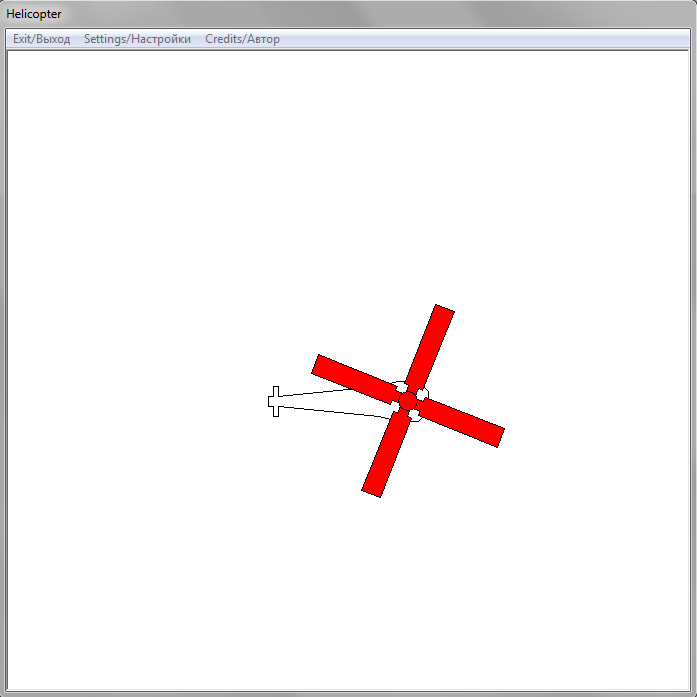
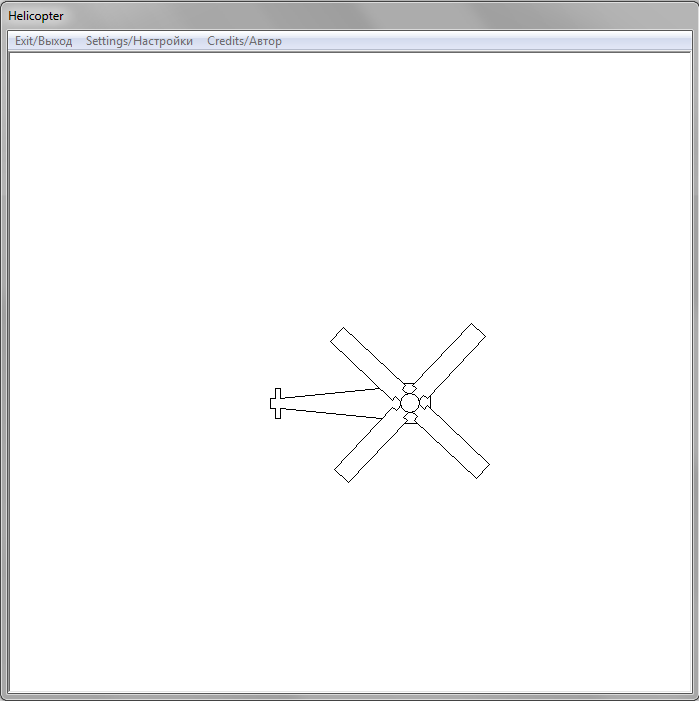
Как итог, разработанное приложение моделирует заданный процесс движения вертолета с использованием математического аппарата геометрии и графики. Поддерживается режим зависания, а также движение перпендикулярно направлению основного движения. В дальнейшем возможно усовершенствование приложения путем использования аппаратных средств пересчета координат и алгоритмов из оптимизированных библиотек. Получены навыки практического использования пакета MFC, применения математического аппарата компьютерной геометрии и графики.

Демонстрация работы приложения:

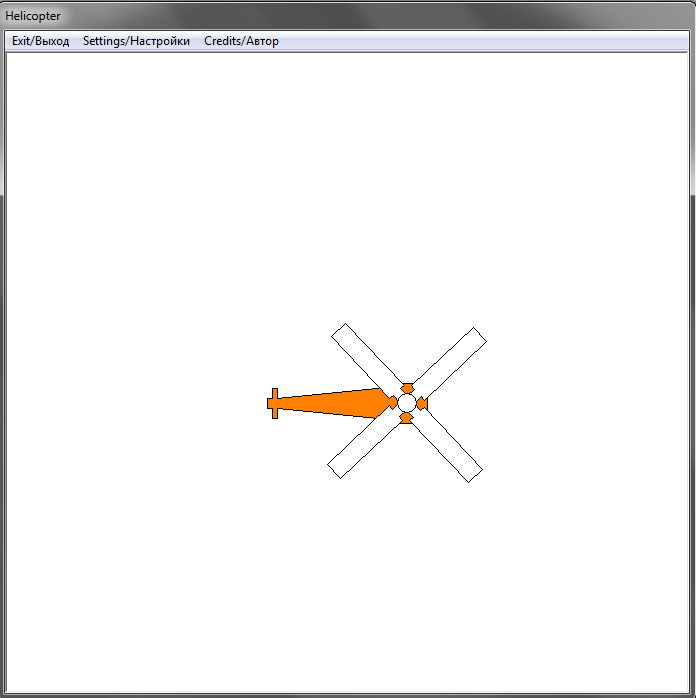
а) движение вертолета,

б) смена цветов лопастей,

в) смена цветов корпуса.



а) б)



в)

Рисунок 3а,3б,3в – демонстрация работы приложения

# ЛИСТИНГ ПРИЛОЖЕНИЯ

**Приложение А. Класс CChildView.**

// ChildView.cpp : реализация класса CChildView

//

#include "stdafx.h"

#include "Helicopter.h"

#include "ChildView.h"

#include "Settings.h"

#ifdef \_DEBUG

#define new DEBUG\_NEW

#endif

// CChildView

CChildView::CChildView()

{

Timer=false; //время зависания

Hovering=true; //режим зависания

RW.SetRect(300,300,400,400);

ColorOfHelicopterBlade=RGB(255,255,255); //изначальный цвет лопастей

ColorOfHelicopterBody=RGB(255,255,255); //изначальный цвет корпуса

Т=3000; //стартовая скорость вращения лопастей

}

CChildView::~CChildView()

{

}

BEGIN\_MESSAGE\_MAP(CChildView, CWnd)

ON\_WM\_PAINT()

ON\_WM\_TIMER()

ON\_WM\_KEYDOWN()

ON\_COMMAND(ID\_32771, &CChildView::On32771)

ON\_COMMAND(ID\_32772, &CChildView::On32772)

ON\_COMMAND(ID\_32773, &CChildView::On32773)

END\_MESSAGE\_MAP()

// обработчики сообщений CChildView

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); // контекст устройства для рисования

if(Timer==false){

SetTimer(1,30,NULL);

Timer=true;

}

HP.Draw(dc,RW,ColorOfHelicopterBlade,ColorOfHelicopterBody);

}

// обработчик нажатия кнопок

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

if(nChar==38){ //вверх

HP.Move(0,-5);

Invalidate();

}

else if(nChar==40){ //вниз

HP.Move(0,5);

Invalidate();

}

else if(nChar==32){ //вперед

Hovering=!Hovering;

}

}

void CChildView::OnTimer(UINT flag){

if(!Hovering){

HP.Move(Т/300,0);

}

HP.RotorTurn(360/Т\*30);

Invalidate();

}

//вызов диалога установки цвета лопастей

void CChildView::On32771()

{

CColorDialog DiagColor;

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

ColorOfHelicopterBlade=DiagColor.GetColor();

Invalidate();

}

}

//вызов диалога установки цвета корпуса

void CChildView::On32772()

{

CColorDialog DiagColor;

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

ColorOfHelicopterBody=DiagColor.GetColor();

Invalidate();

}

}

//вызов диалога установки периода вращения

void CChildView::On32773()

{

Settings MyDialog;

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

Т=MyDialog.GetT();

Invalidate();

}

}

**Приложение Б. Класс Settings**.**cpp**

// Settings.cpp: файл реализации

//

#include "stdafx.h"

#include "Helicopter.h"

#include "Settings.h"

#include "afxdialogex.h"

// диалоговое окно Settings

IMPLEMENT\_DYNAMIC(Settings, CDialogEx)

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

: CDialogEx(Settings::IDD, pParent)

{

EnableAutomation();

Т=10;

}

Settings::~Settings()

{

}

void Settings::OnFinalRelease()

{

// Когда будет освобождена последняя ссылка на объект автоматизации,

// вызывается OnFinalRelease. Базовый класс автоматически

// удалит объект. Перед вызовом базового класса добавьте

// дополнительную очистку, необходимую вашему объекту.

CDialogEx::OnFinalRelease();

}

void Settings::DoDataExchange(CDataExchange\* pDX)

{

CDialogEx::DoDataExchange(pDX);

DDX\_Control(pDX, IDC\_SLIDER1, Slider);

}

BEGIN\_MESSAGE\_MAP(Settings, CDialogEx)

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

ON\_BN\_CLICKED(IDCANCEL, &Settings::OnBnClickedCancel)

ON\_NOTIFY(NM\_CUSTOMDRAW, IDC\_SLIDER1, &Settings::OnNMCustomdrawSlider1)

END\_MESSAGE\_MAP()

BEGIN\_DISPATCH\_MAP(Settings, CDialogEx)

END\_DISPATCH\_MAP()

// Примечание: мы добавили поддержку для IID\_ISettings, чтобы обеспечить безопасную с точки зрения типов привязку

// из VBA. Этот IID должен соответствовать GUID, связанному с

// disp-интерфейсом в файле .IDL.

// {B19C8FA6-0A8B-4961-BEC4-D4D9E596EA8A}

static const IID IID\_ISettings =

{ 0xB19C8FA6, 0xA8B, 0x4961, { 0xBE, 0xC4, 0xD4, 0xD9, 0xE5, 0x96, 0xEA, 0x8A } };

BEGIN\_INTERFACE\_MAP(Settings, CDialogEx)

INTERFACE\_PART(Settings, IID\_ISettings, Dispatch)

END\_INTERFACE\_MAP()

// обработчики сообщений Settings

void Settings::OnBnClickedOk()

{

// TODO: добавьте свой код обработчика уведомлений

CDialogEx::OnOK();

}

void Settings::OnBnClickedCancel()

{

// TODO: добавьте свой код обработчика уведомлений

CDialogEx::OnCancel();

}

void Settings::OnNMCustomdrawSlider1(NMHDR \*pNMHDR, LRESULT \*pResult)

{

LPNMCUSTOMDRAW pNMCD = reinterpret\_cast<LPNMCUSTOMDRAW>(pNMHDR);

Т=Slider.GetPos()\*99.9+10;

\*pResult = 0;

}

double Settings::GetT(){

return Т;

}

**Приложение В. Описание класса CMatrix.h**

class CMatrix

{

double \*\*array;

int n\_rows; // Число строк

int n\_cols; // Число столбцов

public:

CMatrix(); // Конструктор по умолчанию (1 на 1)

CMatrix(int,int); // Конструктор

CMatrix(int); // Конструктор -вектора (один столбец)

CMatrix(const CMatrix&); // Конструктор копирования

~CMatrix();

double &operator()(int,int); // Выбор элемента матрицы по индексу

double &operator()(int); // Выбор элемента вектора по индексу

CMatrix operator-(); // Оператор "-"

CMatrix operator=(const CMatrix&); // Оператор "Присвоить": M1=M2

CMatrix operator\*(CMatrix&); // Оператор "Произведение": М1\*М2

CMatrix operator\*(double x); // Оператор "Произведение": М1\*a

CMatrix operator/(double x); // Оператор "Деление": М1/a

CMatrix operator+(CMatrix&); // Оператор "+": M1+M2

CMatrix operator-(CMatrix&); // Оператор "-": M1-M2

CMatrix operator+(double); // Оператор "+": M+a

CMatrix operator-(double); // Оператор "-": M-a

int rows()const{return n\_rows;} ; // Возвращает число строк

int cols()const{return n\_cols;}; // Возвращает число строк

CMatrix Transp(); // Возвращает матрицу,транспонированную к текущей

CMatrix GetRow(int); // Возвращает строку по номеру

CMatrix GetRow(int,int,int);

CMatrix GetCol(int); // Возвращает столбец по номеру

CMatrix GetCol(int,int,int);

CMatrix RedimMatrix(int,int); // Изменяет размер матрицы с уничтожением данных

CMatrix RedimData(int,int); // Изменяет размер матрицы с сохранением данных,

//которые можно сохранить

CMatrix RedimMatrix(int); // Изменяет размер матрицы с уничтожением данных

CMatrix RedimData(int); // Изменяет размер матрицы с сохранением данных,

//которые можно сохранить

double MaxElement(); // Максимальный элемент матрицы

double MinElement(); // Минимальный элемент матрицы

double Abs(); // Модуль матрицы

};

**Приложение Г. Методы класса HelicopterPoints.cpp**

HelicopterPoints::HelicopterPoints(void)

{

//угол

Corner = 0; X = 0; Y = 0;

//устаналиваем начальное положение корпуса и винта

HelicopterBodyPoints.RedimMatrix(3,18);

HelicopterRotorPoints.RedimMatrix(3,28);

//точки корпуса

for(int i=0;i<18;i++)

{

HelicopterBodyPoints(2,i)=1;

}

//точки винта

for(int i=0;i<28;i++)

{

HelicopterRotorPoints(2,i)=1;

}

HelicopterBodyPoints(0,0)=-10;

HelicopterBodyPoints(1,0)=20;

HelicopterBodyPoints(0,1)=10;

HelicopterBodyPoints(1,1)=20;

HelicopterBodyPoints(0,2)=20;

HelicopterBodyPoints(1,2)=10;

HelicopterBodyPoints(0,3)=20;

HelicopterBodyPoints(1,3)=-10;

HelicopterBodyPoints(0,4)=10;

HelicopterBodyPoints(1,4)=-20;

HelicopterBodyPoints(0,5)=-10;

HelicopterBodyPoints(1,5)=-20;

HelicopterBodyPoints(0,6)=-30;

HelicopterBodyPoints(1,6)=-15;

HelicopterBodyPoints(0,7)=-130;

HelicopterBodyPoints(1,7)=-5;

HelicopterBodyPoints(0,8)=-130;

HelicopterBodyPoints(1,8)=-15;

HelicopterBodyPoints(0,9)=-135;

HelicopterBodyPoints(1,9)=-15;

HelicopterBodyPoints(0,10)=-135;

HelicopterBodyPoints(1,10)=-5;

HelicopterBodyPoints(0,11)=-140;

HelicopterBodyPoints(1,11)=-5;

HelicopterBodyPoints(0,12)=-140;

HelicopterBodyPoints(1,12)=5;

HelicopterBodyPoints(0,13)=-135;

HelicopterBodyPoints(1,13)=5;

HelicopterBodyPoints(0,14)=-135;

HelicopterBodyPoints(1,14)=15;

HelicopterBodyPoints(0,15)=-130;

HelicopterBodyPoints(1,15)=15;

HelicopterBodyPoints(0,16)=-130;

HelicopterBodyPoints(1,16)=5;

HelicopterBodyPoints(0,17)=-30;

HelicopterBodyPoints(1,17)=15;

HelicopterRotorPoints(0,0)=-5;

HelicopterRotorPoints(1,0)=5;

HelicopterRotorPoints(0,1)=-5;

HelicopterRotorPoints(1,1)=15;

HelicopterRotorPoints(0,2)=-10;

HelicopterRotorPoints(1,2)=15;

HelicopterRotorPoints(0,3)=-10;

HelicopterRotorPoints(1,3)=100;

HelicopterRotorPoints(0,4)=10;

HelicopterRotorPoints(1,4)=100;

HelicopterRotorPoints(0,5)=10;

HelicopterRotorPoints(1,5)=15;

HelicopterRotorPoints(0,6)=5;

HelicopterRotorPoints(1,6)=15;

HelicopterRotorPoints(0,7)=5;

HelicopterRotorPoints(1,7)=5;

HelicopterRotorPoints(0,8)=15;

HelicopterRotorPoints(1,8)=5;

HelicopterRotorPoints(0,9)=15;

HelicopterRotorPoints(1,9)=10;

HelicopterRotorPoints(0,10)=100;

HelicopterRotorPoints(1,10)=10;

HelicopterRotorPoints(0,11)=100;

HelicopterRotorPoints(1,11)=-10;

HelicopterRotorPoints(0,12)=15;

HelicopterRotorPoints(1,12)=-10;

HelicopterRotorPoints(0,13)=15;

HelicopterRotorPoints(1,13)=-5;

HelicopterRotorPoints(0,14)=5;

HelicopterRotorPoints(1,14)=-5;

HelicopterRotorPoints(0,15)=5;

HelicopterRotorPoints(1,15)=-15;

HelicopterRotorPoints(0,16)=10;

HelicopterRotorPoints(1,16)=-15;

HelicopterRotorPoints(0,17)=10;

HelicopterRotorPoints(1,17)=-100;

HelicopterRotorPoints(0,18)=-10;

HelicopterRotorPoints(1,18)=-100;

HelicopterRotorPoints(0,19)=-10;

HelicopterRotorPoints(1,19)=-15;

HelicopterRotorPoints(0,20)=-5;

HelicopterRotorPoints(1,20)=-15;

HelicopterRotorPoints(0,21)=-5;

HelicopterRotorPoints(1,21)=-5;

HelicopterRotorPoints(0,22)=-15;

HelicopterRotorPoints(1,22)=-5;

HelicopterRotorPoints(0,23)=-15;

HelicopterRotorPoints(1,23)=-10;

HelicopterRotorPoints(0,24)=-100;

HelicopterRotorPoints(1,24)=-10;

HelicopterRotorPoints(0,25)=-100;

HelicopterRotorPoints(1,25)=10;

HelicopterRotorPoints(0,26)=-15;

HelicopterRotorPoints(1,26)=10;

HelicopterRotorPoints(0,27)=-15;

HelicopterRotorPoints(1,27)=5;

rs.SetRectD(-50,-50,50,50);

}

void HelicopterPoints::Draw(CDC& dc,CRect& RW,COLORREF ColorOfHelicopterBlade,COLORREF ColorOfHelicopterBody){

CBrush\* br=new CBrush(ColorOfHelicopterBody);//кисть для рисования корпуса

//матрица пересчета из мировой системы координат (МСК) в оконную (ОСК)

//с учетом смещения вертолета в МСК

CMatrix K=SpaceToWindow(rs,RW)\*Translate2D(X,Y);

CMatrix KV=Rotate2D(Corner);//матрица пересчета для поворота лопастей

CPoint\* Points=new CPoint[18];//точки корпуса в ОСК

for(int i=0;i<18;i++){

CMatrix W(3);

W=K\*HelicopterBodyPoints.GetCol(i);//пересчет координат точек корпуса из МСК в ОСК

Points[i].x=W(0);

Points[i].y=W(1);

}

dc.SelectObject(br);//выбор кисти

dc.Polygon(Points,18);//прорисовка корпуса

Points=new CPoint[28];

for(int i=0;i<28;i++){

CMatrix W(3);

W=K\*KV\*HelicopterRotorPoints.GetCol(i);//пересчет коорцинат точек лопастей из МСК в ОСК

Points[i].x=W(0);

Points[i].y=W(1);

}

br=new CBrush(ColorOfHelicopterBlade);//кисть цвета лопастей

dc.SelectObject(br);//выбор кисти

dc.Polygon(Points,28);//прорисовка лопастей

//кружочек - центр вращения лопастей

CMatrix E(3,2);

E(0,0)=-10;E(1,0)=-10;E(2,0)=1;

E(0,1)=10;E(1,1)=10;E(2,1)=1;

for(int i=0;i<2;i++){

CMatrix W(3);

W=K\*E.GetCol(i);//пересчет координат кружочка из МСК в ОСК

Points[i].x=W(0);

Points[i].y=W(1);

}

dc.Ellipse(Points[0].x,Points[0].y,Points[1].x,Points[1].y);

delete Points;

delete br;

}

void HelicopterPoints::RotorTurn(double Crnr){

Corner+=Crnr;

}

//движение вертолета с проверкой вылета за предел экрана

void HelicopterPoints::Move(double iX,double iY){

X+=iX;

if(X>700)X-=1400;

Y+=iY;

if(Y>700)Y-=1400;

else if(Y<0)Y+=1400;

}

**Приложение Д. Функции аффинных преобразований.**

CMatrix Translate2D(double x,double y){ //смещение в двухмерной системе координат

CMatrix m(3,3);

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

m(0,2)=x;

m(1,2)=y;

return m;

}

CMatrix Rotate2D(double fi){ //поворот в двухмерной системе координат

CMatrix m(3,3);

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

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

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

m(2,2)=1;

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

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

return m;

}

//смещение в трехмерной системе координат

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 вокруг оси ОZ

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 вокруг оси ОХ

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 вокруг оси ОY

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;}

**Приложение Е. Вспомогательные функции.**

#pragma once

#include "stdafx.h"

#include "MyGDI.h"

//получение оконных координат по видовым

CMatrix SpaceToWindow(CRectD& rs, CRect& rw){

CMatrix m(3,3);

double kx=(rw.right-rw.left)/(rs.right-rs.left);

double ky=(rw.bottom-rw.top)/(rs.top-rs.bottom);

m(2,2)=1;

m(0,0)=kx;

m(1,1)=-ky;

m(0,2)=rw.left-kx\*rs.left;

m(1,2)=rw.top-ky\*rs.bottom;

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 V1(0)\*V2(0)+V1(1)\*V2(1)+V1(2)\*V2(2);

}

//косинус угла между векторами

double cosViV2(CMatrix V1,CMatrix V2){

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

}

//получение декартовых координат по сферическим

CMatrix SphereToCart(CMatrix DView){

CMatrix Ve(3);

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

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

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

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

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

return Ve;

}

//получение видовых координат по положению наблюдателя

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.