

Утверждаю

Ректор университета

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О.Н. Федонин

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020г.

**ПРОГРАММИРОВАНИЕ WINDOWS-ПРИЛОЖЕНИЙ**

Вывод графики в окно с использованием GDI

**Методические указания**

**к выполнению лабораторной работы №7**

**для студентов очной формы обучения**

**по направлению подготовки**

**02.03.03 «Математическое обеспечение и администрирование информационных систем», профиль «Технология программирования»**

**Брянск 2020**

**УДК 004.01**

Программирование Windows-приложений. Вывод графики в окно с использованием GDI [Электронный ресурс]: методические указания к выполнению лабораторной работы №7 для студентов очной формы обучения по направлению подготовки 02.03.03 «Математическое обеспечение и администрирование информационных систем», профиль «Технология программирования» – Брянск: БГТУ, 2020. – 18 с.

Разработал:

Е.А. Белов,

канд. техн. наук, доц.

Рекомендовано кафедрой «Информатика и программное обеспечение» БГТУ (протокол № 1 от 31.01.2020г.)

**Методические издания публикуются в авторской редакции**

**1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

GDI, с точки зрения программиста - это контекст отображения. Контекст отображения можно сравнить с *листом бумаги* на котором приложение рисует, а также *инструментом для рисования.* Инструменты для рисования - это перья, кисти (а также шрифты и целые графические изображения), с помощью которых приложение создаст необходимый рисунок. Кроме контекста отображения (листа бумаги) и инструмента для рисования, приложению доступны десятки функций программного интерфейса GDI.

Контекст отображения является структурой данных, описывающих устройство отображения. В этой структуре хранятся различные характеристики устройства (бумаги) и набор инструментов для рисования, выбранные по умолчанию. Приложение может выбрать в контекст отображения различные инструменты (например, перья различной толщины и цвета). Поэтому, если необходимо нарисовать линию красного цвета, то перед выполнением операции рисования, необходимо *выбрать в контекст отображения* соответствующее перо.

Функции рисования не имеют параметров, указывающих цвет и толщину линий. Такие параметры хранятсяв контексте отображения.

Приложение может создать контекст отображения не только для экрана монитора, но и для любого другого *графического* устройства, например, для принтера. В этом случае, оно может рисовать на принтере различные изображения, используя те же функции, что и при рисовании на экране. Можно создать контекст отображения для метафайла. Метафайл - это обычный файл на диске или в памяти, в котором хранятся последовательности команд интерфейса GDI. Приложение может выполнить вывод графической информации в метафайл, как в обычное устройство вывода, а затем «проиграть» метафайл на реальном устройстве вывода (экран монитора или принтер).

**3. ЗАДАНИЕ**

На основе примера простейшего приложения создать приложение, обеспечивающее при получении сообщения WM\_PAINT вывод некоторого изображения в окно.

Требования к выполнению:

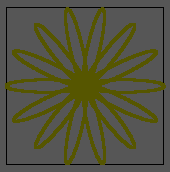
* При каждом получении сообщения WM\_PAINT изображение должно выводиться в случайном месте окна и иметь случайные значения своих характеристик.
* Псевдокод для рисования изображения и структурный тип, описывающий его переменные характеристики, приводятся **для каждого варианта задания**. Следующие характеристики являются обязательными для всех вариантов: ***местоположение, размер*** и ***цвет*** изображения.
* Список сообщений, обязательных для обработки функцией главного окна, WM\_PAINT, WM\_DESTROY. Остальные сообщения передать на обработку стандартной оконной функции.
* Исходные тексты программ обязательно должны быть **отформатированы** при помощи отступов и табуляции для подчеркивания логической структуры программы.

Отчет должен содержать исходный текст приложения, снабженный комментариями и объяснениями поведения окон того или иного стиля.

**4. ВАРИАНТЫ**

**Вариант 1**

В качестве изображения предлагается изображение цветка с N лепестками. Каждый i-й лепесток рисуется так же, как и первый (точнее, нулевой) - в виде эллипса, но является повернутым на угол i⋅df, где df=2⋅PI/N, а i изменяется от 0 до N-1.



1. Структурный тип, описывающий характеристики изображения:

struct FLOWER

{

int N; // количество лепестков (вид изображения)

int x,y; // координаты центра цветка (**местоположение**)

int r; // радиус цветка, т.е. длина лепестка (**размер**)

COLORREF color; // цвет линии (**цвет** изображения)

};

2. Псевдокод, предназначенный для модификации и использования в качестве функции рисования изображения. Контекст отображения передается через параметр hDC, а характеристики изображения - при помощи структуры, содержащей значения этих характеристик:

#define PI 3.14156

void flower(HDC hDC,struct FLOWER flw)

// функция вывода цветка с N лепестками

{

double rx,ry,x1,x2,x3,x4,x5,x6,y1,y2,y3,y4,y5,y6,f,df;

rx=flw.r/2.; // r - длина лепестка, rx - ее половина

x1=flw.x+rx; y1=flw.y;// x1, y1 - центр нулевого лепестка

ry=2.\*PI\*rx\*0.7/(float)flw.N; ry=ry/2.;

// ry - половина ширины лепестка

df=2.\*PI/(float)flw.N;// df - угол, занимаемый лепестком

// создание объектов GDI (функция CreatePen)

**… создание сплошного пера цвета flw.color толщиной 3 пикселя**

// установка атрибутов контекста отображения (функция SelectObject)

**… установка созданного пера в контекст hDC и сохранение "старого" пера**

for(int i=0; i<flw.N; i++)

// рисуем очередной лепесток цветка с N лепестками ( 0<=i<N)

{

// сдвинуть начальную позицию вывода в центр

// цветка (функция MoveToEx)

**… передвинуть текущую позицию пера в точку с координатами (flw.x, flw.y)**

f=(float)i\*df; // f - угол поворота для i-того лепестка

// вывод лепестка, повернутого относительного

// нулевого на f радиан

for(x2=-rx; x2<rx; x2+=0.5)

// вывод первой половины лепестка

{

y2=sqrt((1.-(x2\*x2)/(rx\*rx))\*(ry\*ry));

x3=x1+x2; y3=y1-y2;

x4=x3-flw.x; y4=y3-flw.y;

x5=x4\*cos(f)-y4\*sin(f);

5=x4\*sin(f)+y4\*cos(f);

x6=x5+flw.x; y6=y5+flw.y;

// нарисуем очередной фрагмент контура

// лепестка (функция LineTo)

**… провести линию от текущей позиции пера до точки (x6, y6)**

}

for(x2=rx; x2>-rx; x2-=0.5)

// вывод второй половины лепестка

{

y2=sqrt((1.-(x2\*x2)/(rx\*rx))\*(ry\*ry));

x3=x1+x2; y3=y1+y2;

x4=x3-flw.x; y4=y3-flw.y;

x5=x4\*cos(f)-y4\*sin(f);

y5=x4\*sin(f)+y4\*cos(f);

x6=x5+flw.x; y6=y5+flw.y;

// нарисуем очередной фрагмент контура

// лепестка (функция LineTo)

**… провести линию от текущей позиции пера до точки (x6, y6)**

}

// нарисуем очередной фрагмент контура лепестка (функция LineTo)

**провести линию от текущей позиции пера до точки (flw.x, flw.y)**

}

// восстановление атрибутов контекста отображения

**… установка "старого" пера в контекст hDC**

// удаление созданых объектов GDI

**… удаление созданного пера**

}

3. В оконной функции главного окна приложения объявить статическую переменную flw типа struct FLOWER для хранения значений характеристик изображения. Значения характеристик будут случайным образом задаваться при получении сообщения WM\_PAINT.

4. При обработке сообщения WM\_PAINT присвоить значения характеристикам изображения случайные значения (функция rand) и перерисовать его полностью:

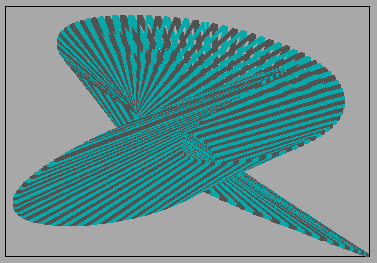
* получить контекст отображения;
* получить размер рабочей области окна (функция GetClientRect);
* присвоить:
  + - * flw.N - случайное значение в диапазоне от 5 до 25;
      * flw.r - случайное значение в диапазоне от 20 до 220;
      * flw.x - случайное значение в диапазоне от flw.r до разности ширины окна и flw.r;
      * flw.у - случайное значение в диапазоне от flw.r до разности высоты окна и flw.r;
      * flw.color - случайное значение цвета: RGB(rand()%255,rand()%255,rand()%255);
* вызывать функцию flower, передав контекст отображения и структуру с характеристиками;
* освободить контекст отображения.

*Значения величин могут быть изменены экспериментально автором работы.*

5. Создать исполняемый модуль и проверить его работу. Для проверки правильности работы приложения необходимо, например, после запуска приложения сначала минимизировать его главное окно, а затем опять восстановить размер окна. Изображение должно появиться в другом месте окна и иметь новые значения характеристик.

**Вариант 2**

В качестве изображения предлагается изображение узора, состоящего из линий, координаты начала и конца которых задаются тригонометрическими функциями. Аргументы используемых тригонометрических функций ставятся в зависимость от некоторых коэффициентов и величины t, которая изменяется от 0 до 5 с шагом 0,01.



1. Структурный тип, описывающий характеристики изображения цветка:

struct UZOR

{

unsigned C1,A3,A4,B3,B4; // коэффициенты

int x,y; // координаты лев. верх. угла изображения

unsigned w,h; // ширина и высота рисунка (**размер**)

COLORREF color; // цвет линии (**цвет изображения**)

};

2. Псевдокод, предназначенный для модификации и использования в качестве функции рисования изображения. Контекст отображения передается через параметр hDC, а характеристики изображения - при помощи структуры, содержащей значения этих характеристик:

void uzor(HDC hDC,struct UZOR uzr) // функция вывода узора

{

int dx=uzr.w/2,dy=uzr.h/2,ddx=dx/4,ddy=dy/4,x1,x2,y1,y2;

double D,T =5;

COLORREF grey=RGB(127,127,127);

// серый цвет чередуется с цветом изображения.

int step=1;

// создание объектов GDI (функция CreatePen)

**… создание сплошного пера цвета uzr.color толщиной 3 пикселя**

**… создание сплошного пера цвета grey толщиной 3 пикселя**

for(double t=0; t<T; t+=0.01) // рисуем очередной фрагмент узора t ( 0<=T<5)

{

// установка атрибутов контекста отображения (функция SelectObject)

if(step)

{

**… установка пера цвета изображения в контекст hDC**

**и сохранение "старого" пера**

}

else

{

**… установка пера серого цвета в контекст hDC**

**и сохранение "старого" пера**

}

step=!step;

// определение координат начала и конца очередной линии узора

D=t+uzr.C1;

x1=(int)((2\*cos(uzr.A4\*t)+2\*cos(uzr.A3\*2\*t))\*ddx);

y1=(int)((2\*cos(uzr.B4\*t)+2\*cos(uzr.B3\*2\*t))\*ddy);

x2=(int)((2\*cos(uzr.A4\*D)+2\*cos(uzr.A3\*2\*D))\*ddx);

y2=(int)((2\*cos(uzr.B4\*D)+2\*cos(uzr.B3\*2\*D))\*ddy);

x1= uzr.x+(x1+dx); y1= uzr.y+(y1+dy);

x2= uzr.x+(x2+dx); y2= uzr.y+(y2+dy);

// вывод очередной линии от (x1,y1) до (x2,y2)

// (функции MoveToEx и LineTo)

**… передвинуть текущую позицию пера в точку с координатами (x1, y1)**

**… провести линию от текущей позиции пера до точки (x2, y2)**

// восстановление атрибутов контекста отображения

**… установка "старого" пера в контекст hDC**

}

// удаление созданных объектов GDI

**… удаление двух созданных перьев**

}

3. В оконной функции главного окна приложения объявить статическую переменную uzr типа struct UZOR для хранения значений характеристик изображения. Значения характеристик будут случайным образом задаваться при получении сообщения WM\_PAINT.

4. При обработке сообщения WM\_PAINT присвоить значения характеристикам изображения случайные значения (функция rand) и перерисовать его полностью:

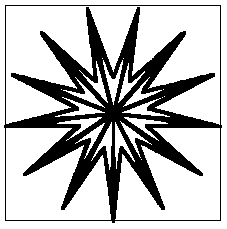
* получить контекст отображения;
* получить размер рабочей области окна (функция GetClientRect);
* присвоить:
  + - * uzr.A4, uzr.A3, uzr.B4, uzr.B3, uzr.C1- случайное значение в диапазоне от 0 до 5;
      * uzr.w - случайное значение в диапазоне от 20 до разности ширины окна и 20;
      * uzr.h - случайное значение в диапазоне от 20 до разности высоты окна и 20;
      * uzr.x - случайное значение в диапазоне от 0 до разности ширины окна и uzr.w;
      * uzr.у - случайное значение в диапазоне от 0 до разности высоты окна и uzr.h;
      * uzr.color - случайное значение цвета: RGB(rand()%255,rand()%255,rand()%255);
* вызывать функцию uzor, передав контекст отображения и структуру с характеристиками;
* освободить контекст отображения.

*Значения величин могут быть изменены экспериментально автором работы.*

5. Создать исполняемый модуль и проверить его работу. Для проверки правильности работы приложения необходимо, например, после запуска приложения сначала минимизировать его главное окно, а затем опять восстановить размер окна. Изображение должно появиться в другом месте окна и иметь новые значения характеристик.

**Вариант 3**

В качестве изображения предлагается изображение N-конечной звезды.



1. Структурный тип, описывающий характеристики изображения:

struct STAR

{

unsigned N; // количество фрагментов (**вид изображения**)

int x,y; // координаты центра звезды (**местоположение**)

unsigned r; // радиус звезды (**размер**)

COLORREF color; // цвет линии (**цвет изображения**)

};

2. Псевдокод, предназначенный для модификации и использования в качестве функции рисования изображения. Контекст отображения передается через параметр hDC, а характеристики изображения - при помощи структуры, содержащей значения этих характеристик:

#define PI 3.14156

void ugol(HDC hDC,int x1,int y1,int x2,int y2,int x3,int y3,int n);

// рабочая функция

void star(HDC hDC,struct STAR str)

{

int rc=str.r/3; // радиус центра звезды

double a1,a2,a3,d=2.\*PI/(str.N\*2);

// создание объектов GDI (функция CreatePen)

**… создание сплошного пера цвета str.color толщиной 3 пикселя**

// установка атрибутов контекста отображения (функция SelectObject)

**… установка созданного пера в контекст hDC и сохранение "старого" пера**

for(int i=1; i<=str.N; i++)

// рисуем очередной фрагмент звезды ( 1<=i<=N)

{

// определение значений углов

a1=2\*PI\*(i-1)/str.N; a2=a1+d; a3=a1-d;

// вывод угла, заштрихованного отрезками

ugol(hDC,str.x,str.y,str.x+str.r\*sin(a1),

str.y+str.r\*cos(a1),str.x+rc\*sin(a2),str.y+rc\*cos(a2),25);

ugol(hDC,str.x,str.y,str.x+str.r\*sin(a1),str.y+str.r\*cos(a1),

str.x+rc\*sin(a3),str.y+rc\*cos(a3),25);

}

// восстановление атрибутов контекста отображения

**… установка "старого" пера в контекст hDC**

// удаление созданых объектов GDI

**… удаление созданного пера**

}

// вывод угла, заштрихованного отрезками

void ugol(HDC hDC,int x1,int y1,int x2,int y2,int x3,int y3,int n)

{

int n1=n+1; double a; int xx1,yy1,xx2,yy2;

for(int i=1; i<=n1; i++)

{

a=(i-1.)/n;

xx1= x1+(x2-x1)\*a; yy1= y1+(y2-y1)\*a;

xx2= x2+(x3-x2)\*a; yy2= y2+(y3-y2)\*a;

// вывод очередной линии от (xx1,yy1) до (xx2,yy2)

// (функции MoveToEx и LineTo)

**… передвинуть текущую позицию пера в точку с координатами (xx1, yy1)**

**… провести линию от текущей позиции пера до точки (xx2, yy2)**

}

}

3. В оконной функции главного окна приложения объявить статическую переменную str типа struct STAR для хранения значений характеристик изображения. Значения характеристик будут случайным образом задаваться при получении сообщения WM\_PAINT.

4. При обработке сообщения WM\_PAINT присвоить значения характеристикам изображения случайные значения (функция rand) и перерисовать его полностью:

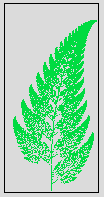
* получить контекст отображения;
* получить размер рабочей области окна (функция GetClientRect);
* присвоить:
  + - * str.N - случайное значение в диапазоне от 5 до 20;
      * str.r - случайное значение в диапазоне от 20 до 220;
      * str.x - случайное значение в диапазоне от str.r до разности ширины окна и str.r;
      * str.у - случайное значение в диапазоне от str.r до разности высоты окна и str.r;
      * str.color - случайное значение цвета: RGB(rand()%255,rand()%255,rand()%255);
* вызывать функцию star, передав контекст отображения и структуру с характеристиками;
* освободить контекст отображения.

*Значения величин могут быть изменены экспериментально автором работы.*

5. Создать исполняемый модуль и проверить его работу. Для проверки правильности работы приложения необходимо, например, после запуска приложения сначала минимизировать его главное окно, а затем опять восстановить размер окна. Изображение должно появиться в другом месте окна и иметь новые значения характеристик.

**Вариант 4**

В качестве изображения предлагается фрактальное изображение листа папоротника. Изображение формируется из MaxIteration пикселей. Ширина получаемого изображения в два раза меньше заданной высоты.



1. Структурный тип, описывающий характеристики изображения:

struct FERN

{

int x,y; // координаты лев. верх. угла (**местоположение**)

int h; // высота листа (**размер**)

COLORREF color; // цвет линии (**цвет изображения**)

};

2. Псевдокод, предназначенный для модификации и использования в качестве функции рисования изображения. Контекст отображения передается через параметр hDC, а характеристики изображения - при помощи структуры, содержащей значения этих характеристик:

void fern(HDC hDC,struct FERN frn)

// функция вывода листа папоротника

{

long MaxIteration=frn.h\*100l; // число итераций

static double x0=0,y0=0,xx,yy, x,y;

// таблица коэффициентов для данного фрактального изображения

double a[4]={ 0.00, 0.85, 0.20,-0.15};

double b[4]={ 0.00, 0.04,-0.26, 0.28};

double c[4]={ 0.00,-0.04, 0.23, 0.26};

double d[4]={ 0.16, 0.85, 0.22, 0.24};

double e[4]={ 0.00, 0.00, 0.00, 0.00};

double f[4]={ 0.00, 1.60, 1.60, 0.44};

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

// рисуем очередной пиксель (0<=i<MaxIteration)

{

// определение координат

int k=rand()%100+1;

if(k<=1) k=0;

else if(k>1&&k<=86) k=1;

else if(k>86&&k<=93) k=2;

else if(k>93&&k<=100)

k=3;

xx=a[k]\*x0+b[k]\*y0+e[k];

yy=c[k]\*x0+d[k]\*y0+f[k];

x0=xx; y0=yy;

x= frn.x+(int)(x0\*frn.h/11+frn.h/4);

y= frn.y+(int)(y0\*(-frn.h)/11+ frn.h);

// вывод очередного пикселя (функция SetPixel)

**… нарисовать цветом frn.color пиксель с координатами (x, y)**

}

}

3. В оконной функции главного окна приложения объявить статическую переменную frn типа struct FERN для хранения значений характеристик изображения. Значения характеристик будут случайным образом задаваться при получении сообщения WM\_PAINT.

4. При обработке сообщения WM\_PAINT присвоить значения характеристикам изображения случайные значения (функция rand) и перерисовать его полностью:

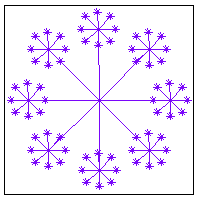
* получить контекст отображения;
* получить размер рабочей области окна (функция GetClientRect);
* присвоить:
  + - * frn.h - случайное значение в диапазоне от 20 до разности высоты окна и 20;
      * frn.x - случайное значение в диапазоне от 0 до разности ширины окна и frn.h/2;
      * frn.у - случайное значение в диапазоне от 0 до разности высоты окна и frn.h;
      * frn.color - случайное значение цвета: RGB(rand()%255,rand()%255,rand()%255);
* вызывать функцию fern, передав контекст отображения и структуру с характеристиками;
* освободить контекст отображения.

*Значения величин могут быть изменены экспериментально автором работы.*

5. Создать исполняемый модуль и проверить его работу. Для проверки правильности работы приложения необходимо, например, после запуска приложения сначала минимизировать его главное окно, а затем опять восстановить размер окна. Изображение должно появиться в другом месте окна и иметь новые значения характеристик.

**Вариант 5**

В качестве изображения предлагается изображение снежинки из N звеньев (уровней), P ветвей (лучей). Длина отрезка каждого очередного звена в четыре раза короче предыдущего.



1. Структурный тип, описывающий характеристики изображения:

struct SNOWFLAKE

{

unsigned N; // кол- во звеньев (уровней) (**вид изображения**)

unsigned P; // кол-во ветвей в каждом звене (**вид изображения**)

int x,y; // координаты центра (**местоположение**)

unsigned r; // радиус снежинки (**размер**)

COLORREF color; // цвет линии (**цвет изображения**)

};

2. Псевдокод, предназначенный для модификации и использования в качестве функции рисования изображения. Контекст отображения передается через параметр hDC, а характеристики изображения - при помощи структуры, содержащей значения этих характеристик:

#define PI 3.14156

void snow(HDC hDC,struct SNOWFLAKE snow)

{

if(snow.N>0) // рисуем очередной фрагмент ( 1<=k<=N)

{

// длина ветви (луча)

int L=(int)(snow.r\*3\*exp((snow.N-1)\*log(4.))/(exp(snow.N\*log(4.))-1));

for(int i=0; i<snow.P; i++)

// вывод луча и на его конце - лучей меньшего уровня

{

// создание объектов GDI (функция CreatePen)

**… создание сплошного пера цвета snow.color толщиной 1 пиксель**

// установка атрибутов контекста отображения (функция SelectObject)

**… установка созданного пера в контекст hDC и сохранение "старого" пера**

// определение координат очередной развилки

int x1=(int)(snow.x+L\*cos(2\*PI\*i/snow.P));

int y1=(int)(snow.y+L\*sin(2\*PI\*i/snow.P));

// вывод линии от (snow.x,snow.y) до (x1,y1)

// (функции MoveToEx и LineTo)

**… передвинуть текущую позицию пера в точку с координатами (snow.x, snow.y)**

**… провести линию от текущей позиции пера до точки (x1, y1)**

// восстановление атрибутов контекста отображения

**… установка "старого" пера в контекст hDC**

// удаление созданых объектов GDI

**… удаление созданного пера**

// вывод P ветвей звена с номером N-1 на i-той

// ветви звена с номером N

struct SNOWFLAKE snowTmp={snow.N-

1,snow.P,x1,y1,snow.r/4,snow.color};

snowflake(hDC,snowTmp); // вывод при помощи рекурсии

}

}

}

3. В оконной функции главного окна приложения объявить статическую переменную snow типа struct SNOWFLAKE для хранения значений характеристик изображения. Значения характеристик будут случайным образом задаваться при получении сообщения WM\_PAINT.

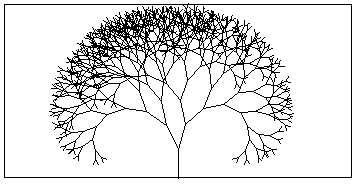
4. При обработке сообщения WM\_PAINT присвоить значения характеристикам изображения случайные значения (функция rand) и перерисовать его полностью:

* получить контекст отображения;
* получить размер рабочей области окна (функция GetClientRect);
* присвоить:
  + - snow.N - значение 4, а snow.P - значение 8;
    - snow.r - случайное значение в диапазоне от 20 до 220;
    - snow.x - случайное значение в диапазоне от snow.r до разности ширины окна и snow.r;
    - snow.у - случайное значение в диапазоне от snow.r до разности высоты окна и snow.r;
    - snow.color - случайное значение цвета: RGB(rand()%255,rand()%255,rand()%255);
* вызывать функцию snow, передав контекст отображения и структуру с характеристиками;
* освободить контекст отображения.

5. Создать исполняемый модуль и проверить его работу. Для проверки правильности работы приложения необходимо, например, после запуска приложения сначала минимизировать его главное окно, а затем опять восстановить размер окна. Изображение должно появиться в другом месте окна и иметь новые значения характеристик.

**Вариант 6**

В качестве изображения предлагается фрактальное изображение дерева. Изображение формируется из отрезков. Примерная высота итогового дерева 6 высот основного ствола, примерная ширина - по 6 высот основного ствола в обе стороны.



1. Структурный тип, описывающий характеристики изображения:

struct TREE

{

int f; // определяет угол наклона (**вид изображения**)

int x,y; //координаты корня дерева (**местоположение**)

int h; // высота основного ствола дерева (**размер**)

COLORREF color; // цвет изображения

};

2. Псевдокод, предназначенный для модификации и использования в качестве функции рисования изображения. Контекст отображения передается через параметр hDC, а характеристики изображения - при помощи структуры, содержащей значения этих характеристик:

#define PI 3.14156

void ugolok(HDC hDC,int \*X[],int \*Y[],int \*UG[],

int \*N,int dl,int ug0,int ug,

int \*X0,int \*Y0,int \*i); // рабочая функция

void tree(HDC hDC,struct TREE tr) // функция вывода дерева

{

int x0,y0,\*X,\*Y,ug,i,k,\*UG,ug0,m,n,kvet,hvet,N;

i=0; k=0; m=1; ug0=28+tr.f; // инициализации

N=(k+1);

// текущее число элементов массивов - хранилищ координат и углов

X=(int\*)malloc(N\*sizeof(int));

Y=(int\*)malloc(N\*sizeof(int));

UG=(int\*)malloc(N\*sizeof(int));

// создание объектов GDI (функция CreatePen)

**… создание сплошного пера цвета tr.color толщиной 1 пиксел**

// установка атрибутов контекста отображения (функция SelectObject)

**… установка созданного пера в контекст hDC и сохранение "старого" пера**

// ствол дерева

ug=UG[k]=ug0\*3.2;

x0=X[k]=tr.x; y0=Y[k]=tr.y;

ugolok(hDC,&X,&Y,&UG,&N,tr.h,ug0,ug,&x0,&y0,&i);

do // рисуем само дерево

{

hvet=rand()%2+2; //параметр уменьшения ветки

if(hvet==2) kvet=128\*2; else kvet=64\*2;

if (tr.h>hvet) tr.h-=hvet; else break;

for(n=1;n<=m;n++)

{

k++;

ug=UG[k]+tr.f;

x0=X[k]; y0=Y[k];

ugolok(hDC,&X,&Y,&UG,&N,tr.h,ug0,ug,&x0,&y0,&i);

}

m=m\*2;

} while(m<=kvet);

// восстановление атрибутов контекста отображения

**… установка "старого" пера в контекст hDC**

// удаление созданых объектов GDI

**… удаление созданного пера**

free(X); free(Y); free(UG);

}

// вывод веток, выходящих из вершины

void ugolok(HDC hDC,int \*X[],int \*Y[],int \*UG[], int \*N, int dl,int ug0, int ug,int \*X0,int \*Y0,int \*i)

{

int x0=\*X0,y0=\*Y0,x1,y1,w; double grad=PI/180.;

if(\*i==0) w=1;else w=2;

for(int a=1;a<=w;a++)

{

(\*i)++;

if((\*i)+1>\*N)

// необходимо выделить дополнительную память

{

\*N=(\*i)+1;

\*X=(int\*)realloc(\*X,(\*N)\*sizeof(int));

\*Y=(int\*)realloc(\*Y,(\*N)\*sizeof(int));

\*UG=(int\*)realloc(\*UG,(\*N)\*sizeof(int));

}

(\*UG)[\*i]=ug+ug0;

(\*X)[\*i]=x1=x0+cos(ug\*grad)\*dl;

(\*Y)[\*i]=y1=y0-sin(ug\*grad)\*dl;

// вывод очередной линии от (x0,y0) до (x1,y1)

// (функции MoveToEx и LineTo)

**… передвинуть текущую позицию пера в точку с координатами (x0, y0)**

**… провести линию от текущей позиции пера до точки (x1, y1)**

ug-=ug0\*2+(rand()%26-15);

}

}

3. В оконной функции главного окна приложения объявить статическую переменную tr типа struct TREE для хранения значений характеристик изображения. Значения характеристик будут случайным образом задаваться при получении сообщения WM\_PAINT.

4. При обработке сообщения WM\_PAINT присвоить характеристикам изображения случайные значения (функция rand) и перерисовать его полностью:

* получить контекст отображения;
* получить размер рабочей области окна (функция GetClientRect);
* присвоить:
  + - tr.h - случайное значение в диапазоне от 20 до 40;
    - tr.x - случайное значение в диапазоне от 6\*tr.h до разности ширины окна и 6\*tr.h;
    - tr.у - случайное значение в диапазоне от 6\*tr.h до высоты окна
    - tr.color - случайное значение цвета: RGB(rand()%255,rand()%255,rand()%255);
* вызывать функцию tree, передав контекст отображения и структуру с характеристиками;
* освободить контекст отображения.

5. Создать исполняемый модуль и проверить его работу. Для проверки правильности работы приложения необходимо, например, после запуска приложения сначала минимизировать его главное окно, а затем опять восстановить размер окна. Изображение должно появиться в другом месте окна и иметь новые значения характеристик

**5. КОНТРОЛЬНЫЕ ВОПРОСЫ**

1. О чем уведомляется окно, получившее сообщение WM\_PAINT?
2. При возникновении каких причин окно получает сообщение WM\_PAINT?
3. В каких случаях Windows самостоятельно восстанавливает изображение в рабочей области?
4. Какое сообщение обязательно посылается окну перед отправкой ему сообщения WM\_PAINT? Как реагирует на это сообщение оконная функция обработки сообщений по умолчанию?
5. Что произойдет после первого же прихода сообщения WM\_PAINT, если функция окна нарисует что-либо в окне во время обработки сообщений, отличных от WM\_PAINT?
6. Что такое недействительная область (или область обновления)?
7. Что происходит при появлении недействительной области окна?
8. Может ли в очереди сообщений находиться несколько сообщений WM\_PAINT?
9. Как приложение может удалить из очереди сообщение WM\_PAINT?
10. При помощи чего приложения могут рисовать на устройствах вывода?
11. Когда приложение может использовать функции BeginPaint и EndPaint?
12. Обязательно или нет передавать необрабатываемые сообщения WM\_PAINT в функцию обработки сообщений по умолчанию? Что при этом происходит?
13. Как получить контекст устройства, если необходимо рисовать в рабочей области при обработке отличных от WM\_PAINT сообщений?
14. Каким образом приложение может явно потребовать перерисовку всего окна или его части?
15. Что делает функция InvalidateRect?
16. Что является итогом выполнения функции UpdateWindow?
17. Какая функция является “противоположной” для функции InvalidateRect?
18. Что определяет контекст устройства?
19. В каких ситуациях приложение может не создавать контекст устройства вывода?
20. Что такое общий контекст? Как приложение может его получить?
21. Когда для общего контекста следует выполнять настройку атрибутов?
22. Для чего служит контекст класса окна?
23. Следует ли освобождать контекст класса окна после его получения?
24. Когда желательно использовать контекст класса окна?
25. Можно ли для контекста класса окна выполнять настройку большинства атрибутов всего один раз?
26. Какие атрибуты контекста класса окна обязательно следует настраивать после его получения?
27. Когда используется личный контекст? В преимущества и недостатки его использования?
28. Кем используется и что позволяет делать родительский контекст?
29. Какой контекст позволяет осуществлять вывод в нерабочую область окна? Как его получить?
30. Как можно получить информацию об устройстве вывода? Контекст какого типа следует для этого использовать?
31. Какой контекст применятся для представления растрового изображения в памяти? С кем должен быть совместим этот контекст? Как создается такой контекст?
32. Что такое метафайл? Что он позволяет делать? Как получить его контекст?
33. При помощи какого контекста осуществляется вывод изображений на такое устройство, как, например, принтер?
34. При помощи какого контекста приложение может рисовать на всей поверхности экрана?
35. Какие объекты GDI приложение может создавать и использовать в процессе своей работы?
36. Какова последовательность действий, выполняемых приложением, использующим объекты GDI?
37. Что необходимо сделать для того, чтобы функции рисования могли использовать тот или иной объект GDI?
38. Что такое предопределенные системой объекты GDI? Как приложение может получить к ним доступ?
39. Для чего применяются перья? Каким образом их можно создавать и использовать?
40. Что такое режим смешивания переднего плана, на что он влияет?
41. Что такое режим фона и цвет фона, что они определяют?
42. Для чего используются кисти? Какие кисти может создать приложение?
43. Что такое начало отсчета кисти? Для чего используется этот атрибут?
44. Как приложение может создать свой шрифт, что оно для этого должно указать функции создания шрифта?
45. Каковы основные моменты работы с растровыми изображениями?
46. Для чего используется технология отсечения?
47. Какую форму может иметь область отсечения?

**6. СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ**

1. Рихтер, Д. Windows для профессионалов: создание эффективных Win32-приложений (с CD-ROM)./Д. Рихтер. – М.: Русская редакция, 2000. – 752с.
2. Шилдт, Г. "MFC основы программирования". /Г. Шилдт. - Киев: BHV, 1997. – 452с.
3. Румянцев, П.В. Азбука программирования в Win32 API. / П.В. Румянцев. – М.: Радио и связь, 1999. – 242с.
4. Соломон, Г. Внутреннее устройство Microsoft Windows 2000. / Г. Соломон, М. Руссинович. – 4-е изд. – Спб.: Питер, 2001. - 752с.
5. Роббинс, Д. Отладка Windows-приложений (+ CD-ROM)./ Д. Роббинс. - Спб.: BHV, 2001. - 512с.
6. Финогенов, К. Г. Win32. Основы пpогpаммиpования/К.Г. Финогенов. – М.:Диалог - МИФИ, 2002.-416 с.
7. MSDN. <http://msdn.microsoft.com>
8. RSDN. <http://www.rsdn.ru>

Программирование Windows-приложений. Вывод графики в окно с использованием GDI [Электронный ресурс]: методические указания к выполнению лабораторной работы №7 для студентов очной формы обучения по направлению подготовки 02.03.03 «Математическое обеспечение и администрирование информационных систем», профиль «Технология программирования».

БЕЛОВ ЕВГЕНИЙ АЛЕКСАНДРОВИЧ

Научный редактор Д.И. Булатицкий

Компьютерный набор Е.А. Белов

Иллюстрации Е.А. Белов

Подписано в печать \_\_.\_\_.2020г. Формат 60х84 1/16 Бумага офсетная. Офсетная печать. Усл.печ.л. 1,63 Уч.-изд.л. 1,52 Тираж 1 экз.

Брянский государственный технический университет

Кафедра «Информатика и программное обеспечение», тел. 56-09-84

241035, Брянск, бульвар 50 лет Октября, 7 БГТУ