Ministerul Educaţiei, Culturii și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Departamentul Ingineria Software și Automatică

**RAPORT**

Lucrare de laborator Nr.1

Disciplina: GC

Tema: Studierea primitivelor grafice simple 2D

A efectuat: Белоскурник Дан st.gr. TI-186,

A verificat : [Osovschi Mariana](https://else.fcim.utm.md/user/view.php?id=1030&course=1) lect.univ.

Chișinău 2020

Цель работы: получение практических знаний по синтезу двумерных графических сцен с использованием простых графических примитивов библиотеки Graphics.h. Получение практических навыков по настройке цветов графических фигур, линий.

Рабочая задача:

1. Разработать программу для синтеза статической 2D сцены, используя не менее 6 графических примитивов разных цветов и отображая их на экране.

Контрольные вопросы:

2. Драйверы и графические режимы

3. Графические примитивы

4. Коррекция форм синтезированных изображений

5. Отображение атрибутов графических примитивов

6. Текст графически

7. Запишите числовые значения графически

8. Смотровые ворота

9. Стандартные форматы изображений

Ответы на теоретические вопросы

1. Независимо от операционных систем и платформ, обычная графическая система может быть представлена в виде разных модульных уровней

Разные уровни это:

▪Уровень 1 является графическим оборудованием и оборудованием ввода, основными аппаратными компонентами в любой системе графики. Например, банкомат имеет сенсорный дисплей, что является сразу и его интерфейсом ввода и оборудованием отображения, DVD плеер имеет видеовыход на телевизор, а передняя панель ЖК дисплея имеет свое оборудование отображения и дистанционное управление в качестве входного интерфейса.

▪Уровень 2 представляет собой уровень драйверов, который обеспечивает взаимодействие с операционной системой. Каждая операционная система имеет свой собственный механизм взаимодействия, и производители устройств стараются убедиться, что они обеспечивают драйверы для всех популярных операционных систем. Например, карты с чипом NVIDIA® поставляются с драйверами для Linux и Windows.

▪Уровень 3 состоит из оконной среды, которая представляет собой механизм отрисовки, отвечающий за создание графики и механизм шрифтов, ответственный за отрисовку шрифтов. Например, механизм отрисовки обеспечивает линии, прямоугольники и функциональные возможности для отрисовки других геометрических форм.

▪Уровень 4 является инструментальным уровнем. Набор инструментальных средств строится поверх определённой оконной среды и предоставляет API для использования приложением. Некоторые инструменты доступны поверх нескольких оконных сред и тем самым обеспечивают переносимость приложений. Инструментарии обеспечивают функции для рисования сложных элементов управления, таких как кнопки, поля ввода, списки, и так далее.

▪Самый верхний уровень - это графическое приложение. Приложение не всегда использует инструментарий и оконную среду. С помощью некоторой минимальной абстракции или промежуточного уровня можно было бы написать приложение, которое непосредственно взаимодействует с оборудованием через интерфейс драйвера. Кроме того, некоторые приложения, такие как видеоплеер, требуют интерфейс ускорителя для обхода графического уровня и прямое взаимодействие с драйвером. Для таких случаев графическая система предусматривает специальное взаимодействие, такое как знаменитый Direct-X в Windows.

1. Windows поддерживает пять графических режимов, которые дают возможность прикладной программе устанавливать, как смешивать цвета, где показать выводимые данные, как масштабировать вывод данных и так далее. Эти режимы, которые хранятся в контексте устройства (DC), описаны в следующей ниже таблице.

|  |  |
| --- | --- |
| Графический режим | Описание |
| Background (Фон) | Определяет, как цвет фона смешивается с существующими цветами окна или экрана в ходе операций с текстом и точечным рисунком. |
| Drawing (Рисование) | Определяет, как цвет изображения смешивается с существующими цветами окна или  экрана для операций пера, кисти, точечного рисунка и текста. |
| Mapping (Отображение) | Определяет, как графический вывод данных отображается из логического (или мирового) пространства в окне, на экране или на бумаге принтера. |
| Polygon-fill (Закрашивание многоугольника) | Определяет, как  использовать шаблон кисти, чтобы закрасить внутреннюю часть сложных регионов. |
| Stretching (Растяжение) | Определяет, как цвета точечного рисунка смешиваются с существующими цветами окна или экрана, когда точечный рисунок сжимается (или уменьшается в масштабе). |

1. Под *графическими примитивами* понимаются минимальные графические объекты, которые составляют векторный рисунок. К графическим примитивам в OpenDraw относятся: линии и стрелки; прямоугольники; окружности, эллипсы, дуги, сегменты и секторы; кривые; соединительные линии; трёхмерные объекты (куб, шар, цилиндр и т. д.); текст. Из графических примитивов могут быть составлены более сложные объекты при помощи функции комбинирования и логических операций над формами;
2. **Прямая линия**

Canvas->LineTo(100,200)  
рисует линию в точку с координатами (100, 200), после чего текущей становится точка с координатами (100, 200).

1. **Ломаная линия**

Метод Polyline вычерчивает ломаную линию. В качестве параметров методу передается массив типа TPoint, содержащий координаты узловых точек линии, и количество звеньев линии. Метод Polyline вычерчивает ломаную линию, последовательно соединяя точки, координаты которых находятся в массиве: первую со второй, вторую с третьей, третью с четвертой и т. Д

1. **Прямоугольник**

Метод Rectangle вычерчивает прямоугольник. В инструкции вызова метода надо указать координаты двух точек — углов прямоугольника. Например, оператор

Canvas->Rectangle(10,10,50,50)

1. **Многоугольник**

Метод Polygon вычерчивает многоугольник. Инструкция вызова метода в общем виде выглядит так:

Canvas->Polygon(p,n)

где р — массив записей типа TPoint, который содержит координаты вершин многоугольника; n — количество вершин.

1. **Эллипс окружность**

Нарисовать эллипс или окружность (частный случай эллипса) можно при помощи метода Ellipse. Инструкция вызова метода в общем виде выглядит следующим образом:

Canvas->Ellipse(xl,yl,x2,у2)

Параметры x1, y1, x2, y2 определяют координаты прямоугольника, внутри которого вычерчивается эллипс или, если прямоугольник является квадратом, — окружность Метод Arc рисует дугу — часть эллипса (окружности).

1. **Дуга**

Canvas->Arc(xl,yl,х2,у 2,х3,у3, х4,у4)

Параметры x1, y1, х2, у2 определяют эллипс (окружность), частью которого является дуга. Параметры х3 и у3 задают начальную, а х4 и у4 — конечную точку дуги. Начальная (конечная) точка дуги — это точка пересечения границы эллипса и прямой, проведенной из центра эллипса в точку с координатами х3 и у3(х4, у4). Метод Arc вычерчивает дугу против часовой стрелки от начальной точки к конечной

1. **Текст**

Вывод текста (строк типа Ansistring) на поверхность графического объекта обеспечивает метод TextOutA. Инструкция вызова метода TextoutA в общем виде выглядит следующим образом:

Canvas->TextOutA(x,y,TeKCT)

Параметр текст задает выводимый текст. Параметры х и у определяют координаты точки графической поверхности, от которой выполняется вывод текста

1. Алгоритм Брезенхе́ма — это алгоритм, определяющий, какие точки [двумерного растра нужно закрасить, чтобы получить близкое приближение прямой линии между двумя заданными точками](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%8B_%D0%BF%D0%BE%D1%81%D1%82%D1%80%D0%BE%D0%B5%D0%BD%D0%B8%D1%8F_%D0%BE%D1%82%D1%80%D0%B5%D0%B7%D0%BA%D0%B0).

Использует в своей реализации инкрементирование по основной координате осуществляет множество проверок и математических операций над целыми числами. Для упрощения наклона линий выбирается диапазон от 0 до 1.

Общая формула линии между двумя точками:

(

Также существует алгоритм Брезенхема для рисования окружностей. По методу построения он похож на рисование линии. В этом алгоритме строится дуга окружности для первого квадранта, а координаты точек окружности для остальных квадрантов получаются симметрично. На каждом шаге алгоритма рассматриваются три пикселя, и из них выбирается наиболее подходящий путём сравнения расстояний от центра до выбранного пикселя с радиусом окружности.

1. Объект Pen является свойством объекта Canvas определяющий цвет стиль и толщину линий

Для отображения атрибутов существует два объекта:

1. Pen рисует контуры объекта
2. Brush внутренняя поверхность графических примитивов

Атрибут Pen:

1. Цвет смешивание 255 x 255 x 255

* clBlack
* clBlue
* clRed

1. Вид линий

* psSolid
* psDosh
* psDot

1. Толщина линий. Толщина линий задаётся в пикселях

Атрибуты Brush

1. Стиль

* bsBold
* bsHorizontal

1. Цвет те же значения как и в Pen

Атрибуты текста

1. Название (вид шрифта)
2. Размер задается в пунктах. Пункт- это единица измерения шрифта используемый в полиграфии самый маленький шрифт 1/72 дюйма-один пункт
3. Стиль

* fsBold
* fsItalic

1. Цвет Tcolor=cl…
2. .
3. . Формат графического файла определяет способ хранения графической информации в файле (растровый или векторный), а также форму хранения информации (используемый алгоритм сжатия для уменьшения объёма файла).

|  |  |
| --- | --- |
| Название формата | Описание |
| BMP   (Windows Device Independent Bitmap) | — формат хранения растровых изображений, разработанный компанией Microsoft.  С форматом BMP работает огромное количество программ, так как его поддержка интегрирована в операционные системы Windows и OS/2.  Формат BMP поддерживается всеми графическими редакторами, работающими под ее управлением, способен хранить как индексированный (до 256 цветов), так и RGB-цвет (16,7 млн. оттенков).  Имена файлов BMP используют расширения \*.bmp, \*.dib и \*.rle |
| GIF   (Graphic Interchange Format) | — стандартный растровый формат представления изображений в WWW. Формат GIF позволяет хорошо сжимать файлы, в которых много однородных заливок (логотипы, надписи, схемы), записывать изображение "через строчку" (Interlaced mode), благодаря чему, имея только часть файла, можно увидеть изображение целиком, но с меньшим разрешением. Применяется для хранения рисунков и анимации в Интернете.  Имена файлов GIF используют расширение \*.gif . |
| TIFF   (Tagged Image File Format) | — формат хранения растровых графических изображений.  TIFF используется при сканировании, отправке факсов, распознавании текста, в полиграфии, широко поддерживается графическими приложениями.  TIFF может сохранять векторную графику программы Photoshop, Alpha-каналы для создания масок в видеоклипах Adobe Premiere и др. Имена файлов TIFF используют расширение \*.tiff и \*.tif. |
| JPEG   (Joint Photographic Experts Group) | — один из популярных графических форматов, применяемый для хранения фотоизображений.  Алгоритм JPEG в наибольшей степени *пригоден* для сжатия фотографий и картин, содержащих реалистичные сцены с плавными переходами яркости и цвета. Наибольшее распространение JPEG получил в цифровой фотографии и для хранения и передачи изображений с использованием сети Интернет.  JPEG *не подходит* для сжатия изображений при многоступенчатой обработке, так как искажения в изображения будут вноситься каждый раз при сохранении промежуточных результатов обработки.  Имена файлов JPEG используют расширения:  .jpeg, .jfif, .jpg, .JPG, или .JPE. |

Практическая часть:

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

#include <vcl.h>

#include <math.h>

#pragma hdrstop

#include "Unit1.h"

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

#pragma package(smart\_init)

#pragma resource "\*.dfm"

TForm1 \*Form1;

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

\_\_fastcall TForm1::TForm1(TComponent\* Owner)

: TForm(Owner)

{

}

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

void \_\_fastcall TForm1::Button1Click(TObject \*Sender)

{

Canvas->Brush->Color = clBlack;

Canvas->Pen->Color = clBlue;

Canvas->Pen->Width=2;

Canvas->MoveTo(40,135);

Canvas->LineTo(40,225);

Canvas->Arc(40,210,80,240,40,225,60,240);

Canvas->MoveTo(60,240);

Canvas->LineTo(120,240);

Canvas->Arc(100,240,140,270,140,255,120,240);

Canvas->MoveTo(140,255);

Canvas->LineTo(140,285);

Canvas->Arc(100,270,140,300,120,300, 140,285);

Canvas->MoveTo(120,300);

Canvas->LineTo(40,300);

Canvas->MoveTo(40,330);

Canvas->LineTo(120,330);

Canvas->Arc(100,330,140,360,140,345,120,330);

Canvas->MoveTo(140,345);

Canvas->LineTo(140,375);

Canvas->Arc(100,360,140,390,120,390,140,375);

Canvas->MoveTo(120,390);

Canvas->LineTo(60,390);

Canvas->Arc(40,390,80,420,60,390,40,405);

Canvas->MoveTo(40,405);

Canvas->LineTo(40,495);

Canvas->Arc(40,480,80,510,40,495,60,510);

Canvas->MoveTo(60,510);

Canvas->LineTo(740,510);

Canvas->Arc(720,480,760,510,740,510,760,495);

Canvas->MoveTo(760,495);

Canvas->LineTo(760,405);

Canvas->Arc(720,390,760,420,760,405,740,390);

Canvas->MoveTo(740,390);

Canvas->LineTo(680,390);

Canvas->Arc(660,360,700,390,660,375,680,390);

Canvas->MoveTo(660,375);

Canvas->LineTo(660,345);

Canvas->Arc(660,330,700,360,680,330,660,345);

Canvas->MoveTo(680,330);

Canvas->LineTo(760,330);

Canvas->MoveTo(760,300);

Canvas->LineTo(680,300);

Canvas->Arc(660,270,700,300,660,285,680,300);

Canvas->MoveTo(660,285);

Canvas->LineTo(660,255);

Canvas->Arc(660,240,700,270,680,240,660,255);

Canvas->MoveTo(680,240);

Canvas->LineTo(740,240);

Canvas->Arc(720,210,760,240,740,240,760,225);

Canvas->MoveTo(760,225);

Canvas->LineTo(760,135);

Canvas->Arc(720,120,760,150,760,135,740,120);

Canvas->MoveTo(740,120);

Canvas->LineTo(60,120);

Canvas->Arc(40,120,80,150,60,120,40,135);

Canvas->Rectangle(80,150,140,210);

Canvas->Rectangle(180,150,340,210);

Canvas->Rectangle(380,120,420,180);

Canvas->Rectangle(460,150,540,180);

TPoint tPoints[6];

tPoints[0].x = 580;

tPoints[0].y = 150;

tPoints[1].x = 720;

tPoints[1].y = 150;

tPoints[2].x = 720;

tPoints[2].y = 210;

tPoints[3].x = 660;

tPoints[3].y = 210;

tPoints[4].x = 660;

tPoints[4].y = 180;

tPoints[5].x = 580;

tPoints[5].y = 180;

Canvas->Polygon(tPoints,5);

Canvas->Rectangle(540,210,620,240);

TPoint tPoints1[9];

tPoints1[0].x = 500;

tPoints1[0].y = 210;

tPoints1[1].x = 500;

tPoints1[1].y = 270;

tPoints1[2].x = 380;

tPoints1[2].y = 270;

tPoints1[3].x = 380;

tPoints1[3].y = 210;

tPoints1[4].x = 420;

tPoints1[4].y = 210;

tPoints1[5].x = 420;

tPoints1[5].y = 240;

tPoints1[6].x = 460;

tPoints1[6].y = 240;

tPoints1[7].x = 460;

tPoints1[7].y = 210;

tPoints1[8].x = 500;

tPoints1[8].y = 210;

Canvas->Polyline(tPoints1,8);

Canvas->Rectangle(260,240,340,270);

TPoint tPoints2[8];

tPoints2[0].x = 220;

tPoints2[0].y = 240;

tPoints2[1].x = 220;

tPoints2[1].y = 300;

tPoints2[2].x = 300;

tPoints2[2].y = 300;

tPoints2[3].x = 300;

tPoints2[3].y = 330;

tPoints2[4].x = 220;

tPoints2[4].y = 330;

tPoints2[5].x = 220;

tPoints2[5].y = 390;

tPoints2[6].x = 180;

tPoints2[6].y = 390;

tPoints2[7].x = 180;

tPoints2[7].y = 240;

Canvas->Polygon(tPoints2,7);

TPoint tPoints3[6];

tPoints3[0].x = 620;

tPoints3[0].y = 270;

tPoints3[1].x = 620;

tPoints3[1].y = 360;

tPoints3[2].x = 500;

tPoints3[2].y = 360;

tPoints3[3].x = 500;

tPoints3[3].y = 300;

tPoints3[4].x = 540;

tPoints3[4].y = 300;

tPoints3[5].x = 540;

tPoints3[5].y = 270;

Canvas->Polygon(tPoints3,5);

TPoint tPoints4[8];

tPoints4[0].x = 250;

tPoints4[0].y = 360;

tPoints4[1].x = 250;

tPoints4[1].y = 480;

tPoints4[2].x = 340;

tPoints4[2].y = 480;

tPoints4[3].x = 340;

tPoints4[3].y = 420;

tPoints4[4].x = 460;

tPoints4[4].y = 420;

tPoints4[5].x = 460;

tPoints4[5].y = 390;

tPoints4[6].x = 300;

tPoints4[6].y = 390;

tPoints4[7].x = 300;

tPoints4[7].y = 360;

Canvas->Polygon(tPoints4,7);

TPoint tPoints5[6];

tPoints5[0].x = 500;

tPoints5[0].y = 390;

tPoints5[1].x = 500;

tPoints5[1].y = 450;

tPoints5[2].x = 460;

tPoints5[2].y = 450;

tPoints5[3].x = 460;

tPoints5[3].y = 480;

tPoints5[4].x = 620;

tPoints5[4].y = 480;

tPoints5[5].x = 620;

tPoints5[5].y = 390;

Canvas->Polygon(tPoints5,5);

Canvas->Rectangle(80,420,140,480);

Canvas->Rectangle(180,420,220,480);

Canvas->Rectangle(660,420,720,480);

Canvas->Rectangle(380,450,420,510);

// Canvas->Pen->Color = clBlue;

Canvas->Pen->Width=4;

Canvas->MoveTo(380,300);

Canvas->LineTo(340,300);

Canvas->LineTo(340,360);

Canvas->LineTo(460,360);

Canvas->LineTo(460,300);

Canvas->LineTo(420,300);

Canvas->Pen->Color = clFuchsia;

Canvas->Pen->Width=2;

Canvas->LineTo(380,300);

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

Canvas->Pen->Color = clBlack;

Canvas->Pen->Width=1;

Canvas->Brush->Color = clRed;

Canvas->Ellipse(51,131,71,143);

Canvas->Ellipse(151,219,169,231);

Canvas->Brush->Color = clYellow;

int i;

int x1=92;

int x2=108;

int d;

Canvas->Ellipse(x1,129,x2,141);

for(i=1;i<7;i++)

{

d=43;

x1=x1+d;

x2=x2+d;

Canvas->Ellipse(x1,129,x2,141);

}

Canvas->Ellipse(352,249,368,261);

Canvas->Ellipse(152,159,168,171);

Canvas->Ellipse(152,159+30,168,171+30);

int xx1=154;

int xx2=168;

int y1=219;

int y2=231;

int l;

for(i=1;i<4;i++)

{

d=33;

xx1=xx1-d;

xx2=xx2-d;

Canvas->Ellipse(xx1,219,xx2,231);

}

Canvas->Ellipse(54,189,69,201);

Canvas->Ellipse(54,189-27,69,201-27);

TPoint tPoints6[8];

tPoints6[0].x = 146;

tPoints6[0].y = 405;

tPoints6[1].x = 150;

tPoints6[1].y = 398;

tPoints6[2].x = 160;

tPoints6[2].y = 394;

tPoints6[3].x = 172;

tPoints6[3].y = 400;

tPoints6[4].x = 158;

tPoints6[4].y = 405;

tPoints6[5].x = 172;

tPoints6[5].y = 410;

tPoints6[6].x = 153;

tPoints6[6].y = 414;

tPoints6[7].x = 158;

tPoints6[7].y = 415;

Canvas->Polygon(tPoints6,7);

}

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

