UDC 004.92

Beata Pańczyk
Institute of Computer Science, Lublin University of Technology, Poland
beatap@pluton.pol.lublin.pl

Basics of Computer Graphics in C++ Builder Environment

Computer graphics refers to any computer device or program that makes a computer capable of displaying and manipulating pictures. Many software applications include graphics components. Graphics support all CAD/CAM systems, some database management systems and spreadsheet programs. Computer graphics is generation of (possibly realistic) images of virtual scenes using computer hardware. In this context it refers to the algorithms, conceptual constructions and mathematical background needed to render virtual scenes either 2D or 3D. Computer Graphics is also unit for MSc students of computer science; it is the base for practical using the calculus, geometry, numerical methods and programming at all. This article presents the way of teaching computer graphics using object programming in C++ Builder environment.

Introduction

Students are learning the "Basics of Computer Graphics" unit on VIII semester (15 hours lectures and 15 hours laboratory), paralleling with "C++ programming" (30 hours lectures and 30 hours laboratory). They have no practice in visual and events programming yet. Computer graphics is the best base for teaching objective techniques of programming and C++ Builder [1] is chosen as visual programming environment.

Theoretical lectures are concerned on:

- description and basic transformations (translation, scaling and rotation) of two dimension (2D) objects;
- projection (parallel, central) and basic transformation of three dimensions (3D) objects [2], [3];
- animations techniques;

The proposed way of descriptions 2D and 3D object may be adequate class definition.

1 Graphics in C++Builder

The integrated development environment (IDE) of C++Builder Borland Software Corporation system provides all the tools needed to design, develop, test, debug, and deploy applications, allowing rapid prototyping and a shorter development time. The IDE includes all the tools necessary to start designing applications [4]:

- Form Designer, or form, a blank window on which to design the user interface UI for application;
- Component palette for displaying visual and nonvisual components for designing your user interface;
- Object Inspector for examining and changing an object's properties and events;
- Object TreeView for displaying and changing components' logical relationships;
- Code editor for writing and editing the underlying program logic;
- Project Manager for managing the files that makes up one or more projects;
- Integrated debugger for finding and fixing errors in code;
- Many other tools such as property editors to change the values for an object's property;
- Command-line tools including compilers, linkers, and other utilities;
- Extensive class libraries with many reusable objects. Many of the objects provided in the class library are accessible in the IDE from the Component palette. By convention, the names of objects in the class library begin with a T, such as TStatusBar.

C++ Builder leverages the Rapid Application Development (RAD) capabilities of the Visual Component Library (VCL) and Component Library for Cross-Platform (CLX).

1.1 Graphics programming – TCanvas class

TCanvas class from **VCL** library [4] is used for a drawing surface for objects that draw an image of them. Standard window controls such as edit controls or list boxes do not require a canvas, as they are drawn by the system. **TCanvas** provides properties, events and methods that assist in creating an image by specifying the type of brush, pen and font to use.

- Drawing and filling a variety of shapes and lines.
- Writing text.
- Rendering graphic images.
- Enabling a response to changes in the current image.

TCanvas properties:

- **Pen** – specifying the pen to use for drawing lines and outlining shapes in the image; properties: **Width**, **Style** (psDash, psDot, psClear, psSolid) and **Color** (clBlue, clRed, clYellow);
- **Brush –** the color and pattern to use when drawing the background or filling in graphical shapes; properties: **Color, Style** (bsSolid, bsClear, bsHorizontal, bsVertical, bsDiagonal);
- **Font** – specifying the font (face, color, size, style) to use for writing text on the image.

TCanvas methods (for example):

- **MoveTo** – setting the value of PenPos (pen position) before calling LineTo;
- **LineTo** – drawing a line from PenPos up to (x,y), but not including the point (x, y);
- **Rectangle** – drawing a rectangle using Pen and fill it with Brush;

- **Polyline** – connecting a set of points on the canvas;
- **Polygon** – drawing a closed, many-sided shape on the canvas, using the value of Pen. After drawing the complete shape, Polygon fills the shape using the value of Brush.

Example 1 *– Setting the value of TCanvas properties*

```
Canvas->Pen->Color=clRed;
Canvas->Pen->Style=psDot;
Canvas->Pen->Width=1;
Canvas->MoveTo(100,100);
Canvas->LineTo(100,100);
Canvas->Brush->Color=clYellow;
Canvas->Brush->Style=bsDiagonal;
Canvas->Rectangle(10,10,100,100);
```

2 Graphics 2D

Next step may be class definition for drawing and transformation the object described as array of *n* points with (x, *y*) coordinates. The class methods should contain constructor, function for drawing our object and functions for basic graphics transformations (i.e. translation, rotation, scaling).

Graphical objects are describing as set of points, so it is enough to define transformation functions only for one point. The 2D point is represented by couple of Cartesian coordinates P(*x*, *y*). After transformation the point *P(x,y)* is transformed to the point *P′(x′, y′)*.

2.1 Transformations

The basic 2D transformations of point *P(x, y)* are described as:

- translation $T\,[dx,\ dy]$: $\begin{cases} x' = x + dx \\ y' = y + dy \end{cases}$;
- scaling $S(sx,\ sy)$ about point $O(0,\ 0)$: $\begin{cases} x' = sx \cdot x \\ y' = sy \cdot y \end{cases}$;
- rotation $O(\varphi)$ about point $O(0,0)$: $\begin{cases} x' = x\cos\varphi - y\sin\varphi \\ y' = x\sin\varphi + y\cos\varphi \end{cases}$.

Rotation about point *(x0, y0)* is composition of basic transformation: translation $T\,[-x0, -y0]$, rotation $O(\varphi)$ and translation $T\,[x0, y0]$.

Example 2

The example presents the class ***object_2D*** definition. This class is defining the graphical 2D object described using the array *t* of *n* points coordinates. The class has an

interface: constructor and methods: **draw** (drawing the polyline), **translate**, **scale** and **rotate**. The constructors arguments are two integer arrays of coordinates *x, y* and number of points. The constructor creates the dynamic array of points *t***.**

```
class object_2D    //polyline object
{  protected:
  int n;//number of points
  TPoint *tx; //array of points(x1,y1)
   public:
  object_2D(int *x,int *y,int ile);
  void draw(TCanvas *k);
  void translate(int dx, int dy);
  void scale(float sx,float sy);
  void rotate(int x0,int y0,float fi);
  };
object_2D::object_2D(int *x,int * y,int ile)
{n=ile;
  tx=new TPoint[n];
  for (int i=0;i<n;i++)  tx[i]=Point(x[i],y[i]);
}
void object_2D::draw(TCanvas *k)
{    k->Polyline(tx,n-1);
void object_2D::translate(int dx,int dy)
{     for (int i=0;i<n;i++)  { tx[i].x+=dx;        tx[i].y+=dy; }      }
```

The methods **scale** and **rotate** should be defined the same as the **rotate** method. The **object_2D** class was used to define the boat and the car:

```
int          //boat
  x1[]={350,390,350,350,370,358,315,310,350},
  y1[]={148,148,80,156,167,182,162,140,156},
            //car
  x[]={10,20,20,22,30,32,32,78,78,82,88,90,90,100,100,80,75,10,10};
  y[]={90,90,93,97,97,93,90,90,93,97,97,93,90,90 ,70 ,70,55,55,90};
```

Now it is possible to create the objects of **object_2D** class and call for them its methods. For drawing the sample object (car or boat or something else) choosing our form menu, it is necessary to call the events function [3] for example:

```
void __fastcall TForm1::Samochd2Click(TObject *Sender)
{
  Refresh();                          //clearing the form window
  object_2D c(x,y,19);                //creating the car object
  c.rysuj(Form1->Canvas);             //drawing the car on the form canvas
}
```

```
void __fastcall TForm1::Samochd4Click(TObject *Sender)
{
  object_2D c(x,y,19);
  c.translate(100,10);                  //car translation
  c.draw(Form1->Canvas);                //drawing car after translation
}
```

Calling the **draw** methods the polyline is drawing. It is possible to draw the filling polygon by changing the **polyline** method (used in the **draw** method) on the **polygon** method. But it is possible to realise the same in another way showing the properties of the object programming. It is possible to define the **object_2Df** class inheritances the **object_2D** class with predefined **draw** method:

```
class object_2Df: public object_2D        //public inheritance
{  public:
  void draw(TCanvas *k);
  object_2Df(int *x,int * y,int ile):object_2D(x,y,ile){}
};
void object_2Df::draw(TCanvas *k)
{k->Polygon(tx,n-1);          }
```

Now it is possible to draw the polyline or polygon 2D objects. The example is presented on fig. 1.

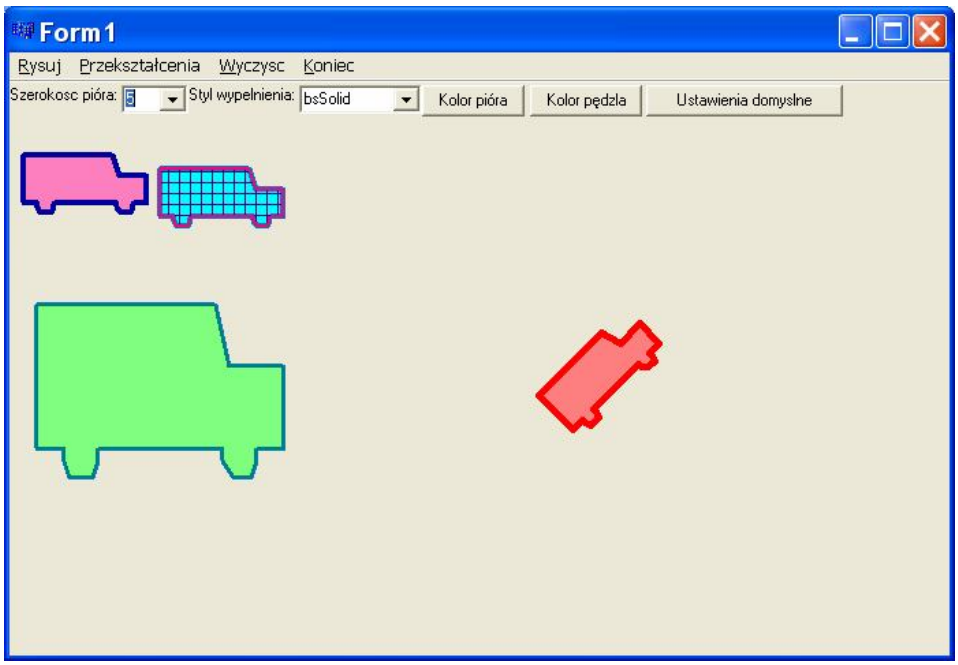

Figure 1 – The basic transformations of the 2D object

Using the simple methods transformation of object_2D and object_2Df classes it is possible to create interesting graphical 2D compositions and animations.

2.2. Animation in C++ Builder

A bitmap is a powerful graphics object used to create, manipulate (scale, scroll, rotate, and paint), and store images in memory and as files on a disk. TBitmap contains an internal image of the bitmap graphic and automatically manages realization of the palette when drawn. To draw a bitmap on a canvas it is necessary to call the Draw method of a TCanvas object, passing a TBitmap as a parameter.

Creating copies of a TBitmap is very fast since the handle is copied rather than the image. If the image is modified and the handle is shared by more than one TBitmap object, the image is copied before the modification is performed (that is, copy on write).

Example 3

The example presents the animation of the boat swimming to the sun using the TBitmap object.

```
object_2Df ob(x1,y1,9);                 //object for animation
                                        //creation the bitmap
Graphics::TBitmap *bm=new Graphics::TBitmap;
                                        // establishing the bitmap dimensions
bm->Width=ClientWidth; bm->Height=ClientHeight;
                                        // the basic animation loop:
                                        // step 1: drawing background
                                        // step 2: calculating the new objects
                                        // coordinates (object transformation)
                                        // step 3: drawing the object after transformation
for (int i=1;i<150;i++)
{ draw_background(bm->Canvas);          //function for drawing the background on the bitmap
                                        //(should be defined)
                                        //calculate the new object coordinates
  ob.draw(bm->Canvas);                  //drawing the object (after transformation)
                                        //on the bitmap
 Canvas->Draw(0,0,bm);                  // drawing the bitmap on the form canvas
  Sleep(100);                           //sleeping program on 100ms
}                                       //end of animation loop
delete bm;                              //free memory
}
```

3 The student's projects

Below there are presented the several students projects (fig. 2-4).

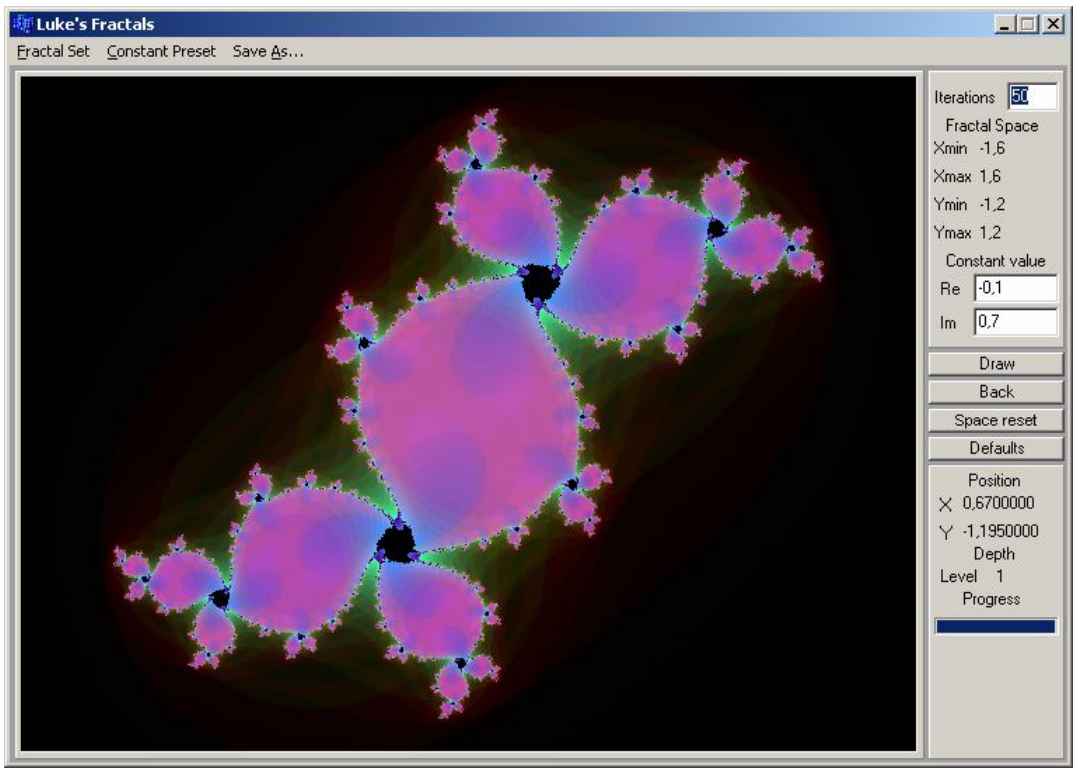

Figure 2 – Project “Fractals”

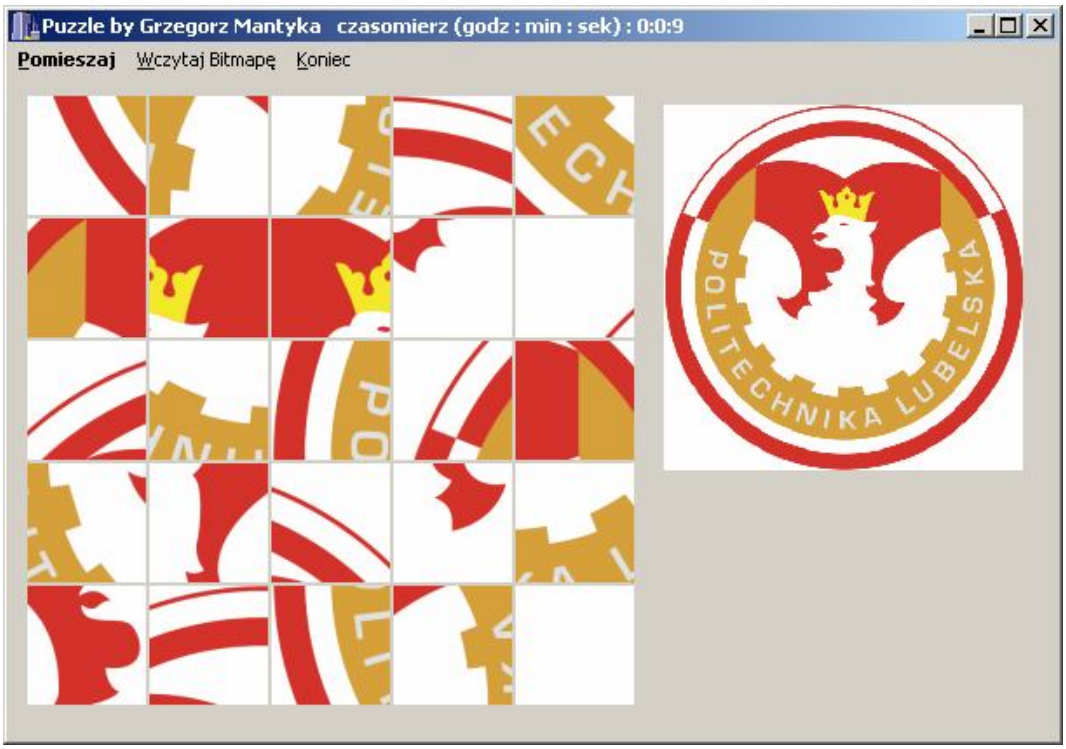

Figure 3 – Project “Puzzle”

Figure 4 – Project "3D graphics"

Conclusions

The projects realised on computer graphics unit are the wonderful base for inspiration of the students. Building the virtual world students are able to learn and to solve by them the complicate, clearly programming problems.

Literature

1. Reisdorph K. C++ Builder 6. – Helion. – Gliwice, 2003.
2. Angell I.O. Wprowadzenie do grafiki komputerowej. – Warszawa: WNT, 1988.
3. Hearn D., Baker M.P. Grafika mikrokomputerowa. – Warszawa: WNT, 1988.
4. C++ Builder Help.

Беата Панчик

Основи комп'ютерної графіки середовища програмування Builder C++

Стаття представляє шлях навчання комп'ютерній графіці, з використанням об'єктного програмування в середовищі програмування Builder C++. У статті показано, як комп'ютерна графіка спрощує форму маніпулювання користувачем програмним і апаратним забезпеченням комп'ютера.

Беата Панчик

Основы компьютерной графики среды программирования Builder C++

Статья представляет путь обучения компьютерной графике с использованием объектного программирования в среде программирования Builder C++. В статье показано, как компьютерная графика упрощает форму манипулирования пользователем программным и аппаратным обеспечением компьютера.

Статья поступила в редакцию 15.04.2005.