

12. Základní grafické konstrukce

Z grafických nástrojů zde uvedeme pouze ty, které zde budeme nezbytně potřebovat. Grafické konstrukce lze v **Delphi** provádět na pozadí každého formuláře, a to do objektu **Canvas**. Na tomto pozadí je k dispozici matice bodů (**Pixels**). Na jednotlivé body odkazujeme dvojicí indexů, které však fungují jako kartézské souřadnice (tj. na rozdíl od matic je první x-ová (vodorovná) a druhá y – ová (svislá)). Jedná se o tzv. univerzální (světové) souřadnice, tj. levý horní roh formuláře má souřadnice [0,0], pravý dolní pak (v závislosti na velikosti formuláře [ClientWidth-1,ClientHeight-1]. Na pozadí jsou dále k dispozici metody **MoveTo**, **LineTo** pro kreslení úseček, objekty **Pen** (pro kreslení čar) a **Brush** (pro vyplňování ploch). Barvy jsou k dispozici jednak prostřednictvím předdeklarovaných konstant – **clRed** (**colorRed**), **clYellow** atd. nebo pomocí složek barevného systému RGB. Přitom každá složka obsazuje jeden byte paměti – složka Blue první, Green druhý a Red třetí. Chceme-li tedy např. obarvit bod o souřadnicích [50,100] barvou se složkami Red=255, Green=255, Blue=100, je třeba psát **Canvas.Pixels[50,100]:=255+256*255+256*256*100** (výsledkem je světle žlutá).

Příklad 1: V obrazu o daných rozměrech sestojte jednu periodu sinusovky. Obecný postup při sestavování grafu funkce $y = f(x)$ je patrný z připojeného obrázku: Definiční obor $\langle x_1; x_2 \rangle$ budeme procházet vhodným krokem hx a nad každým subintervalem graf nahradíme úsečkou. Nemáme-li však k dispozici žádné další prostředky, je třeba rovnici funkce (v našem případě $y = \sin x$) transformovat tak, abychom jednu periodu grafu umístili ne do obdélníka $\langle 0; 2\pi \rangle \times \langle -1; 1 \rangle$, ale do obdélníka $\langle 0; ClientWidth - 1 \rangle \times \langle 0; ClientHeight - 1 \rangle$. Navíc je třeba brát v úvahu opačnou orientaci osy y . Celou proceduru je tedy třeba navrhnout následujícím způsobem (zde navíc počítáme s desetinou výšky resp. šířky obrázku na prázdný okraj):

procedure TForm1.Sinusovka(Sender: TObject);

var x,hx: Integer;

A,B: **array** [1..2] **of** integer;

function sinus(x:Integer):Integer;

begin

sinus:=Round(ClientHeight/2-(0.9*ClientHeight/2)*sin((x-ClientWidth/20)*2*pi
/(0.9*ClientWidth)));

end;

begin

Canvas.MoveTo(0,0);Canvas.LineTo(ClientWidth-1,0);

Canvas.LineTo(ClientWidth-1,ClientHeight-1);

Canvas.LineTo(0,ClientHeight-1);Canvas.LineTo(0,0);

x:=Round(ClientWidth/20);hx:=5;

Repeat

A[1]:=x;A[2]:=sinus(x);

x:=x+hx;

B[1]:=x;B[2]:=sinus(x);

With Canvas **do**

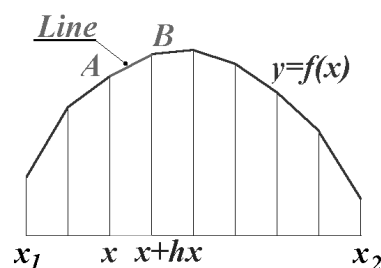
begin

MoveTo(A[1],A[2]);LineTo(B[1],B[2]);

end;

Until x>0.95*ClientWidth;

end;

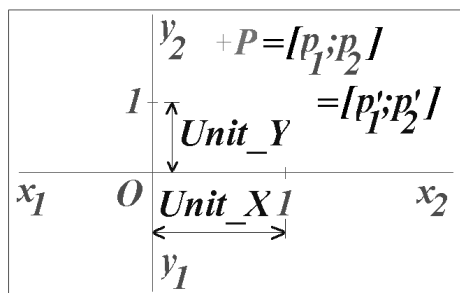


Vidíme, že lokálně deklarovaná funkce sinus je skutečně bizarní. Pokud bychom takto měli uvažovat vždy, když potřebujeme sestrojít nějaký graf, pak bychom toho asi mnoho neudělali. Tento způsob programování připomíná pověstného klauna pracně tlačícího klavír k daleko postavené židli. Pokusme se tedy naopak přisunout „židli ke klavíru“ – tj. transformovat nikoli rovnici funkce, ale souřadnice. Toto řešení spočívá v práci nikoli v souřadnicích světových, ale uživatelských. Budeme sice nyní nuceni zasahovat i do jiných míst zdrojového kódu, než do kterých nás zavedou události v Object Inspektoru, zato budou naše procedury deklarovány globálně a budou tudíž obecně použitelné

Další nevýhodou výše uvedeného kódu je, že objekt Canvas provádí své konstrukce pouze do VideoRAM. Znamená to, že je-li za běhu programu kreslicí plocha překreslena jiným oknem, zmizí i naše konstrukce. Proto je výhodnější provádět grafické konstrukce do jiných objektů, např. do objektu typu TImage (obraz), který najdeme v liště Additional (jeho jméno je implicitně nastaveno na **Image<číslo>**). Rozměry tohoto objektu jsou dány proměnnými Width resp. Height a mohou (dle dalších nastavených parametrů) překročit rozměry formuláře). Tento objekt obsahuje také objekt Canvas, kde jsou k dispozici všechny výše uvedené objekty a metody. Světlo žlutý bod v tomto objektu tedy sestojíme pomocí příkazu `Image1.Canvas.Pixels[50,100]:=255+256*255+256*256*100`.

Světové souřadnice, se kterými Delphi implicitně pracuje, jsou na připojeném obrázku znázorněny po obvodu. Práce v takové soustavě (jak jsme viděli v předchozím příkladu) je pro uživatele značně nepříjemná. Vytvoříme si proto tzv. uživatelskou souřadnou soustavu (uvnitř obdélníka). „Délku“ osy x (interval $\langle x_1; x_2 \rangle$) jakož i „délku“ osy y (interval $\langle y_1; y_2 \rangle$) budeme moci měnit. V obrázku je $Unit_X$ resp. $Unit_Y$ délka uživatelské jednotky na ose x resp. y ve světových souřadnicích. Zřejmě je

$[0;0]$ $[Width-1;0]$



$[0;Height-1]$ $[Width-1;Height-1]$

$$Unit_X = \frac{Width}{x_2 - x_1}; \quad Unit_Y = \frac{Height}{y_2 - y_1} \quad (1)$$

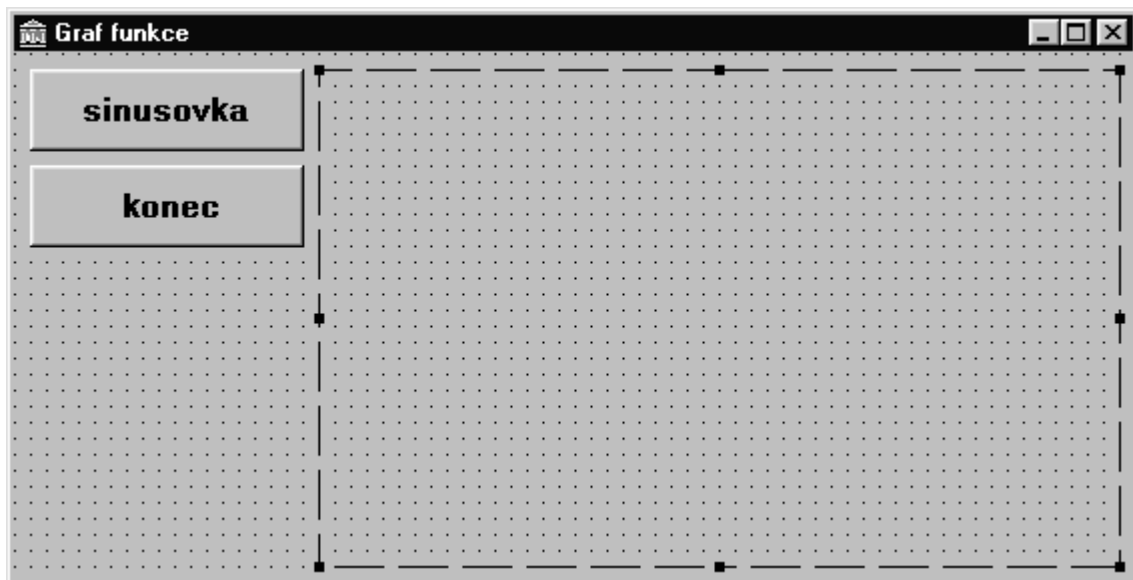
Souřadnice uživatelského počátku O ve světových souřadnicích jsou pak

$$O_1 = -x_1 \cdot Unit_X; \quad O_2 = y_2 \cdot Unit_Y \quad (2)$$

Je-li pak P libovolný bod na kreslicí ploše, $[p_1; p_2]$ jeho souřadnice v uživatelské soustavě a $[p'_1; p'_2]$ jeho souřadnice ve světové soustavě, pak zřejmě platí

$$p'_1 = O_1 + p_1 \cdot Unit_X; \quad p'_2 = O_2 - p_2 \cdot Unit_Y \quad (3)$$

Realizujme nyní uživatelskou souřadnou soustavu a vykresleme graf funkce jedné proměnné. Na formulář umístíme objekt **Image**, a dvě tlačítka – **Button1** a **Button2**, jejichž titulky (vlastnost **Caption**) jsme změnili na **sinusovka**, resp. **konec**. Proceduru, která funkci vykresluje, nazveme **Sinusovka** a budeme ji aktivovat **OnClick** na Button1. Tlačítko Button2 spojíme přes OnClick s procedurou **Konec**, která bude mít za úkol náš program ukončit. Po těchto základních úpravách za nás Delphi zapíše do unity, kterou jsme přejmenovali na **Graf_Funkce**, následující řádky:



```
unit Graf_Funkce;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls, ExtCtrls;

type
  TForm1 = class(TForm)
    Image1: TImage;
    Button1: TButton;
    Button2: TButton;
    procedure Sinusovka(Sender: TObject);
    procedure Konec(Sender: TObject);
  private
    {Private declarations }
  public
    {Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.Sinusovka(Sender: TObject);
begin
end;

procedure TForm1.Konec(Sender: TObject);
begin
end;

end.
```

Všimněme si nejdříve řádku **TForm1 = class(TForm)**. Ten se v unitě objeví automaticky, i když do formuláře nezařadíme žádnou komponentu. Deklaruje typ **TForm1** jako **dědice** typu **TForm**. Znamená to, že TForm1 automaticky přebírá vlastnosti typu TForm a navíc můžeme definovat jeho další vlastnosti. Něco za nás opět provede DELPHI. Zařadíme-li do formuláře komponentu **Image**, objeví se nová proměnná dědice – Image1, která je typu TImage. Spojíme-li s událostí OnClick proceduru, kterou v Object Inspectoru nazveme Triangle, pak, jak již víme, DELPHI tuto proceduru zapíše na konec zdrojového textu [procedure TForm1.Triangle (Sender: TObject);] a automaticky nás na ni navede, abychom ji mohli vypsát.

```
unit Graf_Funkce;
```

```
interface
```

```
uses
```

```
Windows,Messages,SysUtils,Classes,Graphics,Controls,Forms,Dialogs,StdCtrls,ExtCtrls;
```

```
type
```

```
    TPoint = array [1..2] of Double;
```

```
    TForm1 = class(TForm)
```

```
    Image1: TImage;
```

```
    Button1: TButton;
```

```
    Button2: TButton;
```

```
    procedure Sinusovka(Sender: TObject);
```

```
    private
```

```
    {Private declarations }
```

```
    public
```

```
    {Public declarations }
```

```
    O:    TPoint;
```

```
    Jx,Jy:Double;
```

```
    Procedure Scale(x1,x2,y1,y2:Double);
```

```
    Procedure MoveTo(P:TPoint);
```

```
    Procedure LineTo(P:TPoint;R,G,B:Byte);
```

```
    Procedure Line(P,Q:TPoint;R,G,B:Byte);
```

```
end;
```

```
var
```

```
    Form1: TForm1;
```

```
implementation
```

```
{ $R *.DFM }
```

```
Procedure TForm1.Scale(x1,x2,y1,y2:Double);
```

```
begin
```

```
    Jx:=Image1.Width/(x2-x1);Jy:=Image1.Height/(y2-y1);
```

```
    O[1]:= -x1*Jx;O[2]:= y2*Jy;
```

```
end;
```

```
Procedure TForm1.MoveTo(P:TPoint);
```

```
begin
```

```
    Image1.Canvas.MoveTo(Round(O[1]+P[1]*Jx),Round(O[2]-P[2]*Jy));
```

```
end;
```

```

Procedure TForm1.LineTo(P:TPoint;R,G,B:Byte);
begin
    Image1.Canvas.Pen.Color:=R+256*G+256*256*B;
    Image1.Canvas.LineTo(Round(O[1]+P[1]*Jx),Round(O[2]-P[2]*Jy));
end;

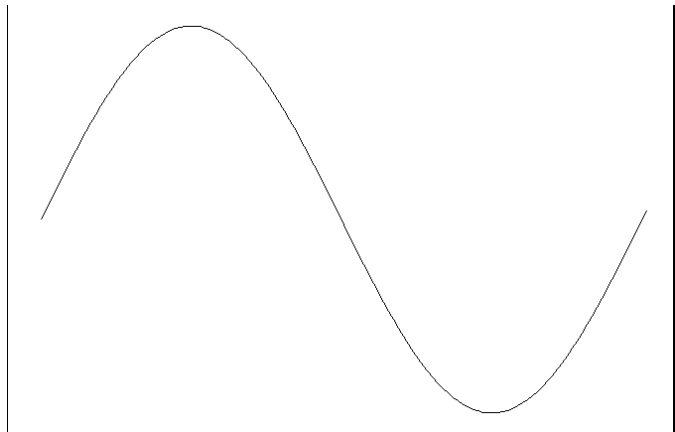
Procedure TForm1.Line(P,Q:TPoint;R,G,B:Byte);
begin
    MoveTo(P);LineTo(Q,R,G,B);
end;

procedure TForm1.Sinusovka(Sender: TObject);
var x,hx :Double;
    A,B: TPoint;

function f(x:Double):Double;
begin f:=sin(x); end;

begin
    Scale(-0.5,7,-1.2,1.2); hx:=0.1;x:=0;
    Repeat
        A[1]:=x;A[2]:=f(x);x:=x+hx;
        B[1]:=x;B[2]:=f(x);
        Line(A,B,200,0,0);
    Until x>2*pi;
end;
end.

```



Všimněte si, že námi definované procedury MoveTo a LineTo opět přesouvají kreslicí pero, resp. sestavují úsečku, nyní však již pracují v uživatelských souřadnicích. Jejich parametry již nejsou značně prkenné dvojice světových souřadnic typu integer, ale body s reálnými souřadnicemi tak, jak jsme na ně běžně zvyklí. Jsou sestaveny jako metody formuláře TForm1. Původní metody stejného jména zůstaly zachovány jako metody objektu TForm1.Image1.Canvas a jsou nám nadále k dispozici (využívají je i naše uživatelské metody TForm1.MoveTo a TForm1.LineTo). Navíc jsme si přidali jednoduchou metodu Line, která je vlastně zkratkou za dvojici metod MoveTo-LineTo. Umožňuje sestavení úsečky jedinou procedurou a ve „standardní výbavě“ Delphi z neznámého důvodu chybí.