**МІНІСТЕРСТВО ОСВІТИ І НАУКИ,
МОЛОДІ ТА СПОРТУ УКРАЇНИ**

**МІЖНАРОДНИЙ ЕКОНОМІКО-ГУМАНІТАРНИЙ
УНІВЕРСИТЕТ
ІМЕНІ АКАДЕМІКА СТЕПАНА ДЕМ'ЯНЧУКА**

С.О. Карпік

Сучасні системи візуалізації даних

Науковий керівник:
Р.М.Літнарович, канд. техн. наук, доцент

Рівне-2012

УДК 004.422.8
Карпік С.О. Сучасні системи візуалізації даних. Науковий керівник Р.М.Літнарович.. МЕГУ, Рівне, 2012.-84 с.

Рецензенти: В.Г.Бурачек, доктор технічних наук, професор
Є.С. Парняков, доктор технічних наук, професор
В.О.Боровий , доктор технічних наук, професор
Відповідальний за випуск: Й.В. Джунь, доктор фізико-математичних наук, професор.

Досліджується і аналізується створення програмного забезпечення для візуалізації даних по землетрусах. Монографія містить актуальний матеріал довідково-аналітичного характеру по наступних темах: створення програмного забезпечення, робота з графікою, створення графічного додатка, візуалізація даних, програмування, робота в середовищі розробки програм Borland C++ Builder 6.
Ключові слова: візуалізація даних, графічний додаток, програмне забезпечення, програмування.

Исследуется и анализируется создание программного обеспечения для визуализации данных по землетрясениям. Монография содержит актуальный материал справочно-аналитического характера по следующим темам: создание программного обеспечения, работа с графикой, создания графического приложения, визуализация данных, программирование, работа в среде разработки программ Borland C++ Builder 6.
Ключевые слова: визуализация данных, графическое приложение, программное обеспечение, программирование.

Investigates and analyzes the development of software for data visualization by earthquakes. The book contains material relevant reference and analytical nature in the following topics: software, work with graphics, create graphics applications, data visualization, programming, work in software development environment Borland C + + Builder 6.
Keywords: data visualization, graphic application, software, programming.

Сергій Олексійович Карпік,
магістрант інформаційних технологій

ЗМІСТ

1.4 Узагальнене програмування (generic programming).....51

2.1 Растрова, векторна і фрактальна графіка.....64
Під терміном "графіка" розуміють результат візуального подання реального або уявного об'єкта, отриманого традиційними методами – малюванням або друкуванням. Комп'ютерна графіка включає методи і засоби створення і обробки зображень за допомогою програмно-апаратних комплексів і охвачує всі види і форми подання зображень, доступних для сприйняття людиною на екрані монітору або в вигляді копії на певному носії. 64
В практиці комп'ютерної графіки виконання роботи по створенню, редагуванню та обробці зображень часто відокремлено від її графічного подання. Зображенням тут вважається об'єкт, відтворений пристроєм виводу, коли графічні дані візуалізуються. 64
В залежності від способу опису та формування зображення розрізняють растрову, векторну та фрактальну графіку.....64
Історично термін "растр" вказував на те, що пристрій при відтворенні зображення використовує набори пікселів (точок), організовані в вигляді послідовностей рядків розгортки. Растрові дані являють собою набори числових значень, які визначають кольори окремих пікселів, впорядкованих таким чином, щоб їх легко було відобразити на растрових пристроях. 64
Базовим елементом растрової графіки є піксель. Логічні пікселі подібні до математичних точок: вони мають місцеположення, але не займають фізичного простору. Фізичні пікселі – це реальні точки, що відображаються пристроєм виводу. Вони є найменшими фізичними елементами поверхні відображення і займають її певну площу. В зв'язку з цим на відстань між двома сусідніми пікселями вводяться обмеження. Якщо задати пристрою відображення надто високу роздільну здатність (кількість пікселів на одиницю довжини зображення), то якість зображення знизиться із-за накладення або злиття сусідніх пікселів. При надто низькій роздільній здатності пікселі можуть бути розкидані по всій площі пристрою відображення. Таким чином, при відображенні значень логічних пікселів із растрових даних в фізичні пікселі повинні враховуватись реальні розміри і розміщення фізичних пікселів.....65
Розрізняють роздільну здатність оригіналу, екранного та друкованого зображення. Роздільна здатність оригіналу вимірюється в точках на дюйм (dpi) і залежить від вимог до якості

зображення, розміру файла, способу оцифровування або методу створення початкової ілюстрації, вибраного формату файла. Підвищення вимог до якості зображення вимагає вищої роздільної здатності оригіналу. Для екранної копії достатньо роздільної здатності 72 dpi, для роздруковування на кольоровому принтері – 150-200 dpi, для виведення на фотоекспонуючий пристрій – 200-300 dpi. 65

Розмір точки растрового зображення залежить від методу і пара–метрів растрування оригіналу, коли на оригінал як би накладається сітка ліній, комірки якої утворюють елемент растра. Частота сітки растра вимірюється кількістю ліній на дюйм і називається лініатурою (lpi). Розмір точки растра розраховується для кожного елементу і залежить від інтенсивності тону в комірці. Для вищої інтенсивності щільніше заповнюється елемент растра. При раструванні з амплітудною модуляцією ілюзія більш темного тону створюється за рахунок збільшення розмірів точок при однаковій відстані між центрами елементів растра. При раструванні з частотною модуляцією інтенсивність тону регулюється зміною відстані між сусідніми точками однакового (найменшого) розміру. Інтенсивність тону прийнято розділяти на 256 рівнів. Для її відтворення достатньо мати розмір комірки растра 16x16 точок.....66

Растрова графіка використовується в випадках, коли потрібна висока точність в передачі кольорів і напівтонів. Однак при цьому розміри файлів суттєво збільшуються з ростом роздільної здатності (одиниці, десятки і сотні Мбайт). До недоліків растрової графіки, окрім великих розмірів файлів, слід віднести пікселізацію зображень при їх збільшенні та деформацію при зменшенні.....66

В векторній графіці базовим елементом зображення є лінія, яка описується математично як єдиний об’єкт, тому обсяг даних для відображення об’єкта засобами векторної графіки суттєво менший, ніж в растровій графіці. Лінія характеризується формою, товщиною, кольором, типом (суцільна, пунктирна і т.п.). Замкнуті лінії мають властивість заповнення простору, що ними охвачується, іншими об’єктами або кольором. Найпростіша незамкнута лінія обмежена двома точками, які називаються вузлами, котрі мають властивості, що впливають на форму кінця лінії і характер сполучення з іншими об’єктами. Всі інші об’єкти векторної графіки складаються з ліній. Найпростішими лініями є пряма (нескінченна), відрізок прямої, криві другого порядку (не мають точок згину – параболи, гіперболи, еліпси, кола), криві третього порядку (можуть мати точки згину), криві Безьє (основані на використанні пари дотичних, проведених до відрізка лінії в її кінцях, кути нахилу і довжина яких впливають на форму лінії).....66

Векторна графіка зручна для зберіганні і обробки зображень, що складаються з ліній, або можуть бути розкладені на прості геометричні об’єкти. Векторні дані легко масштабувати та виконувати над ними інші перетворення (наприклад, повертання зображення, додавання, видалення або зміну окремих елементів зображення). Поряд з цим векторні файли важко застосувати для зберігання складних фотореалістичних зображень. Векторні дані краще відображаються на векторних пристроях виводу (плотерах, дисплеях з довільним скануванням). Ефективно векторну графіку можна відобразити тільки на растрових дисплеях з високою роздільною здатністю. Візуалізація векторних даних може вимагати значно більше часу, ніж візуалізація растрових даних рівної складності.....67

Фрактальна графіка, як і векторна, основана на математичних обчисленнях. ЇЇ базовим елементом є математична формула, виключно на основі якої будується зображення. Таким способом будують як найпростіші регулярні структури, так і складні ілюстрації, що імітують природні ландшафти і тривимірні об’єкти.....67

2.2 VCL – надбудова над GDI.....67

GDI (Graphics Device Interface) – це та частина Windows, що забезпечує підтримку апаратно-незалежної графіки. C++Builder інкапсулює функції Windows GDI на різних рівнях. Найбільш важливим тут є спосіб, за допомогою якого графічні компоненти представляють свої зображення на екрані монітора. При прямому виклику функції GDI необхідно передавати їм дескриптор контексту пристрою (device context handle), що задає обрані інструменти малювання

3.4 Компоненти Borland C++ Builder 6 використані в програмі.....79

```
   {
    ifs>>A[i][j];
   }
  }
ifs.close();
 }
      Series1->Clear();
      Series2->Clear();
      Series3->Clear();
      Series4->Clear();
      Series5->Clear();
      Series6->Clear();
      Series7->Clear();
      Series8->Clear();
      Series9->Clear();
      Series10->Clear();
       int sum=0,r=0,k=0;
       for (int i=0;i<=1000;i+=100)
       {
        for(int j=0;j<=1000;j+=100)
          {
             sum=0;
    if(j<=900 and i<=900)
      { for(r=i;r<i+100;r++)
           {
            for(k=j;k<j+100;k++)
              {
               sum=sum+A[r][k];
              }
             }
            if(j==0){Series1->AddXY(i,sum);}
            if(j==100){Series2->AddXY(i,sum);}
            if(j==200){Series3->AddXY(i,sum);}
            if(j==300){Series4->AddXY(i,sum);}
            if(j==400){Series5->AddXY(i,sum);}
            if(j==500){Series6->AddXY(i,sum);}
            if(j==600){Series7->AddXY(i,sum);}
            if(j==700){Series8->AddXY(i,sum);}
            if(j==800){Series9->AddXY(i,sum);}
            if(j==900){Series10->AddXY(i,sum);}
        }
      }
  for (int i=0; i<1000; i++)
  { delete [] A[i]; }
  delete []A;
}
//---------------------------------------------------------------------------
void __fastcall TForm2::Button2Click(TObject *Sender)
{
  Series1->Clear();
  Series2->Clear();
```

```
  Series3->Clear();....102
  Series4->Clear();....102
  Series5->Clear();....102
  Series6->Clear();....102
  Series7->Clear();....102
  Series8->Clear();....102
  Series9->Clear();....102
  Series10->Clear();....102
}....102
//---------------------------------------------------------------------------....102
void __fastcall TForm2::Button3Click(TObject *Sender)....102
{....102
SavePictureDialog1->InitialDir;....102
  SavePictureDialog1->FileName="C:\\1.bmp";....102
 if (SavePictureDialog1->Execute())....102
 {....102
    Chart1->SaveToBitmapFile(SavePictureDialog1->FileName);....102
 }....102
}....102
//---------------------------------------------------------------------------....102
```

Вікно побудови діаграми....103

```
#include <vcl.h>....103
#pragma hdrstop....103
#include <fstream.h>....103
#include <iso646.h>....103
#include "Unit1.h"....103
#include "Unit3.h"....103
//---------------------------------------------------------------------------....103
#pragma package(smart_init)....103
#pragma resource "*.dfm"....103
TForm3 *Form3;....103
extern char *filename;....103
//---------------------------------------------------------------------------....103
__fastcall TForm3::TForm3(TComponent* Owner)....103
        : TForm(Owner)....103
{....103
}....103
//---------------------------------------------------------------------------....103
void __fastcall TForm3::Button1Click(TObject *Sender)....103
{....103
int **A=new int* [1000];....103
for(int col=0;col<1000;col++)....103
{ A[col]=new int [1000]; }....103
 ifstream ifs (filename,ifstream::in );....103
 {....103
 for (int i=0;i<1000;i++)....103
  {....103
   for (int j=0;j<1000;j++)....103
    {....103
     ifs>>A[i][j];....103
    }....104
   }....104
```

```
ifs.close();
 }
 int sum=0,r=0,k=0;
       Series1->Clear();
       Series2->Clear();
       Series3->Clear();
       Series4->Clear();
       Series5->Clear();
       Series6->Clear();
       Series7->Clear();
       Series8->Clear();
       Series9->Clear();
       Series10->Clear();
       
     for (int i=0;i<=1000;i+=100)
       {
        for(int j=0;j<=1000;j+=100)
          {
             sum=0;
     if(j<=900 and i<=900)
       { for(r=i;r<i+100;r++)
           {
            for(k=j;k<j+100;k++)
              {
               sum=sum+A[r][k];
              }
             }
            }
            if(j==0){Series1->AddXY(i,sum);}
            if(j==100){Series2->AddXY(i,sum);}
            if(j==200){Series3->AddXY(i,sum);}
            if(j==300){Series4->AddXY(i,sum);}
            if(j==400){Series5->AddXY(i,sum);}
            if(j==500){Series6->AddXY(i,sum);}
            if(j==600){Series7->AddXY(i,sum);}
            if(j==700){Series8->AddXY(i,sum);}
            if(j==800){Series9->AddXY(i,sum);}
            if(j==900){Series10->AddXY(i,sum);}
          }
        }
   for (int i=0; i<1000; i++)
   { delete [] A[i]; }
   delete []A;
}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button2Click(TObject *Sender)
{
 Series1->Clear();
 Series2->Clear();
 Series3->Clear();
 Series4->Clear();
 Series5->Clear();
```

```
 Series6->Clear();....105
 Series7->Clear();....105
 Series8->Clear();....106
 Series9->Clear();....106
 Series10->Clear();....106
}....106
//---------------------------------------------------------------------------....106
void __fastcall TForm3::Button3Click(TObject *Sender)....106
{....106
 SavePictureDialog1->InitialDir;....106
  SavePictureDialog1->FileName="C:\\2.bmp";....106
 if (SavePictureDialog1->Execute())....106
 {....106
   Chart1->SaveToBitmapFile(SavePictureDialog1->FileName);....106
 }....106
}....106
//---------------------------------------------------------------------------....106
Вікно відображення хвиль....106
#include <vcl.h>....106
#include <fstream.h>....106
#pragma hdrstop....106
#include "Unit1.h"....106
#include "Unit4.h"....106
//---------------------------------------------------------------------------....106
#pragma package(smart_init)....106
#pragma resource "*.dfm"....106
TForm4 *Form4;....106
extern char *filename;....106
//---------------------------------------------------------------------------....106
__fastcall TForm4::TForm4(TComponent* Owner)....106
      : TForm(Owner)....106
{....107
}....107
//---------------------------------------------------------------------------....107
void __fastcall TForm4::Button1Click(TObject *Sender)....107
{....107
int **A=new int* [1000];....107
 for(int col=0;col<1000;col++)....107
 { A[col]=new int [1000]; }....107
 ifstream ifs (filename, ifstream::in );....107
 {....107
 for (int i=0;i<1000;i++)....107
  {....107
   for (int j=0;j<1000;j++)....107
   {....107
    ifs>>A[i][j];....107
   }....107
  }....107
ifs.close();....107
 }....107
        for(int i = 0; i<1000; i++)....107
       {....107
```

```
          for(int j=0;j<1000;j++)
          {
            Image1->Canvas->Pixels[i][j]=0x00000000+30*(A[i][j]);
          }
        }
  for (int i=0; i<1000; i++)
  { delete [] A[i]; }
  delete []A;
}
//---------------------------------------------------------------------------
void __fastcall TForm4::BitBtn1Click(TObject *Sender)
{
Image1->Picture->Assign(0);
}
//---------------------------------------------------------------------------
void __fastcall TForm4::Button2Click(TObject *Sender)
{
 SavePictureDialog1->InitialDir;
  SavePictureDialog1->FileName="C:\\3.bmp";
 if (SavePictureDialog1->Execute())
 {
 Image1->Picture->SaveToFile(SavePictureDialog1->FileName);
 }
}
//---------------------------------------------------------------------------
void __fastcall TForm4::Button3Click(TObject *Sender)
{
  Image1->Canvas->Pen->Color = clGray;
    for(int x = 0; x < 1000; x += 100)
    {
       Image1->Canvas->MoveTo(x, 0);
       Image1->Canvas->LineTo(x, 1000);
    }
    for(int y = 0; y < 1000; y += 100)
    {
       Image1->Canvas->MoveTo(0, y);
       Image1->Canvas->LineTo(1000, y);
    }
}
//---------------------------------------------------------------------------
```

Вікно відображення ділянок

```
#include <vcl.h>
#include <fstream.h>
#include <iso646.h>
#pragma hdrstop
#include "Unit1.h"
#include "Unit5.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm5 *Form5;
extern char *filename;
```

```
//---------------------------------------------------------------------------109
__fastcall TForm5::TForm5(TComponent* Owner)109
        : TForm(Owner)109
{109
}109
//---------------------------------------------------------------------------109
void __fastcall TForm5::Button1Click(TObject *Sender)109
{109
 int StartX, StartY;109
 int sum=0, b=0, f=0;110
 int **A=new int* [1000];110
for(int col=0;col<1000;col++)110
{ A[col]=new int [1000]; }110
ifstream ifs (filename, ifstream::in );110
{110
for (int i=0;i<1000;i++)110
 {110
  for (int j=0;j<1000;j++)110
  {110
   ifs>>A[i][j];110
   }110
  }110
  ifs.close();110
 }110
       Graphics::TBitmap *DrawingBoard;110
       Boolean IsDrawing;110
       Form5->VertScrollBar->Visible= IsDrawing;110
       DrawingBoard = new Graphics::TBitmap;110
       int recev = 100;110
       int matrix;110
       matrix=recev*10;110
       DrawingBoard->Width = recev;110
       DrawingBoard->Height = recev;110
       DrawingBoard->Canvas->Pixels[0][0]=0x000000;110
       for (int i=0;i<=matrix;i+=recev)110
       {110
         for(int j=0;j<=matrix;j+=recev)110
           {111
              b=i;111
               if(i==0) {b=0;}111
              f=j;111
               if(j==0) {f=0;}111
              sum=0;111
           if(i<=900 and j<=900)111
            { for(int r=b;r<i+100;r++)111
            {111
            for(int k=f;k<j+100;k++)111
              {111
               sum=sum+A[r][k];111
               }111
            }111
           }111
```

```
            for(int x=0; x<recev; x++)111
            {111
              for(int y=0; y<recev; y++)111
                {111
                 DrawingBoard->Canvas->Pixels[x][y]=sum;111
                }111
              }111
              Form5->Image1->Canvas->Draw(j,i,DrawingBoard);111
          }111
        }111
  for (int i=0; i<1000; i++)111
  { delete [] A[i]; }111
delete []A;111
}111
//---------------------------------------------------------------------------112
void __fastcall TForm5::Button2Click(TObject *Sender)112
{112
Image1->Picture->Assign(0);112
}112
//---------------------------------------------------------------------------112
void __fastcall TForm5::Button3Click(TObject *Sender)112
{112
SavePictureDialog1->InitialDir;112
  SavePictureDialog1->FileName="C:\\4.bmp";112
 if (SavePictureDialog1->Execute())112
 {112
 Image1->Picture->SaveToFile(SavePictureDialog1->FileName);112
 }112
}112
//---------------------------------------------------------------------------112
void __fastcall TForm5::Button4Click(TObject *Sender)112
{112
Image1->Canvas->Pen->Color = clGray;112
    for(int x = 0; x < 1000; x += 100)112
    {112
       Image1->Canvas->MoveTo(x, 0);112
       Image1->Canvas->LineTo(x, 1000);112
    }112
    for(int y = 0; y < 1000; y += 100)112
    {112
       Image1->Canvas->MoveTo(0, y);112
       Image1->Canvas->LineTo(1000, y);112
    }112
}112
//---------------------------------------------------------------------------113
void __fastcall TForm5::Button5Click(TObject *Sender)113
{113
int sum=0, b=0, f=0;113
  int **A=new int* [1000];113
 for(int col=0;col<1000;col++)113
 { A[col]=new int [1000]; }113
 ifstream ifs (filename, ifstream::in );113
```

```
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm6::Button1Click(TObject *Sender)
{
 float  cx,cy;
 int level;
 WideString url;
  level=StrToInt(Edit4->Text);
  cx=StrToFloat(Edit1->Text);
  cy=StrToFloat(Edit2->Text);
 url="http://maps.google.com/staticmap?center="+IntToStr((int)cy)+'.'+
 IntToStr((int)(abs(cy-(int)cy)*1000000.))
 +","+
 IntToStr((int)cx)+'.'+
 IntToStr((int)(abs(cx-(int)cx)*1000000.))
 +"&&zoom="+IntToStr(level)+"&&size=600x600&key=MAPS_API_KEY";
   Edit3->Text=url;
}
//---------------------------------------------------------------------------
void __fastcall TForm6::Button2Click(TObject *Sender)
{
    wchar_t URL[150];
 Edit3->Text.WideChar(URL,150);
 CppWebBrowser1->Navigate(URL,0,NULL,NULL,NULL);
}
//---------------------------------------------------------------------------
void __fastcall TForm6::CppWebBrowser1DocumentComplete(TObject *Sender,
     LPDISPATCH pDisp, Variant *URL)
{
   CppWebBrowser1->Show();
   ShowMessage("   Карта завантажена!   ");
  int map_width=600,map_height=600;
   HDC hDc = CreateCompatibleDC(0);
 HBITMAP hBmp = CreateCompatibleBitmap(GetDC(0),map_width,map_height);
 SelectObject(hDc, hBmp);
       BitBlt(hDc, 0, 0, map_width, map_height, GetDC(CppWebBrowser1->Handle), 12, 16,
SRCCOPY);
       Graphics::TBitmap *bmp = new Graphics::TBitmap;
 bmp->Width = map_width;;
 bmp->Height =map_height;;
 bmp->Handle = hBmp;
 bmp->SaveToFile("mf.bmp");
 delete bmp;
CppWebBrowser1->Hide();
Image1->Visible=True;
Image1->Show();
Image1->Picture->LoadFromFile("mf.bmp");
       .
}
//---------------------------------------------------------------------------
```

```
void __fastcall TForm6::Button4Click(TObject *Sender)
{
int x,y;
 int sum=0, b=0, f=0;
 int **A=new int* [1000];
for(int col=0;col<1000;col++)
{ A[col]=new int [1000]; }
ifstream ifs (filename, ifstream::in );
{
for (int i=0;i<1000;i++)
 {
  for (int j=0;j<1000;j++)
  {
   ifs>>A[i][j];
   }
  }
 ifs.close();
 }
      Graphics::TBitmap *DrawingBoard;
      Boolean IsDrawing;
      Form6->VertScrollBar->Visible= IsDrawing;
      DrawingBoard = new Graphics::TBitmap;
      int recev = 100;
      int matrix;
      matrix=recev*10;
      DrawingBoard->Width = 60;
      DrawingBoard->Height = 60;
      DrawingBoard->Canvas->Pixels[0][0]=0x000000;
      for (int i=0;i<=matrix;i+=recev)
      {
       for(int j=0;j<=matrix;j+=recev)
        {
            b=i;
             if(i==0) {b=0;}
            f=j;
             if(j==0) {f=0;}
            sum=0;
         if(j<=900 and i<=900)
          { for(int r=b;r<i+100;r++)
          {
          for(int k=f;k<j+100;k++)
           {
            sum=sum+A[r][k];
            }
          }
         }
        .
          for( x=0; x<60; x++)
          {
           for( y=0; y<60; y++)
            {
             DrawingBoard->Canvas->Pixels[x][y]=+sum;
```

```
                }
              }
     if(sum>0) { Form6->Image1->Canvas->Draw(j-(j/100)*40,i-(i/100)*40,DrawingBoard); }
          }
        }
  for (int i=0; i<1000; i++)
  { delete [] A[i]; }
delete []A;
}
//---------------------------------------------------------------------------
void __fastcall TForm6::Button5Click(TObject *Sender)
{
Image1->Picture->Assign(0);
}
//---------------------------------------------------------------------------
void __fastcall TForm6::Button6Click(TObject *Sender)
{
SavePictureDialog1->InitialDir;
  SavePictureDialog1->FileName="C:\\5.bmp";
 if (SavePictureDialog1->Execute())
 {
 Image1->Picture->SaveToFile(SavePictureDialog1->FileName);
 }
}
//---------------------------------------------------------------------------
void __fastcall TForm6::Button3Click(TObject *Sender)
{
Image1->Canvas->Pen->Color = clGray;
    for(int x = 0; x < 600; x += 60)
    {
       Image1->Canvas->MoveTo(x, 0);
       Image1->Canvas->LineTo(x, 600);
    }
    for(int y = 0; y < 600; y += 60)
    {
       Image1->Canvas->MoveTo(0, y);
       Image1->Canvas->LineTo(600, y);
    }
}
//---------------------------------------------------------------------------
void __fastcall TForm6::Button7Click(TObject *Sender)
{
  int sum=0, b=0, f=0;
  int **A=new int* [1000];
 for(int col=0;col<1000;col++)
 { A[col]=new int [1000]; }
 ifstream ifs (filename, ifstream::in );
 {
 for (int i=0;i<1000;i++)
  {
   for (int j=0;j<1000;j++)
   {
```

```
    ifs>>A[i][j];
    }
   }
  ifs.close();
 }
      for (int i=0;i<1000;i+=100)
      {
       for(int j=0;j<1000;j+=100)
         {
            b=i;
             if(i==0) {b=0;}
            f=j;
             if(j==0) {f=0;}
            sum=0;
          if(j<=900 and i<=900)
          { for(int r=b;r<i+100;r++)
            {
             for(int k=f;k<j+100;k++)
               {
                sum=sum+A[r][k];
               }
             }
            }
           Image1->Canvas->TextOut((j-(j/100)*40)+20,(i-(i/100)*40)+20, sum);
         }
      }
 for (int i=0; i<1000; i++)
 { delete [] A[i]; }
delete []A;
}
//---------------------------------------------------------------------------
void __fastcall TForm6::Button8Click(TObject *Sender)
{
CppWebBrowser1->Hide();
Image1->Visible=True;
Image1->Show();
Image1->Picture->LoadFromFile("mf.bmp");
}
//---------------------------------------------------------------------------
void __fastcall TForm6::Button9Click(TObject *Sender)
{ int sum=0, b=0, f=0;
 int **A=new int* [1000];
for(int col=0;col<1000;col++)
{ A[col]=new int [1000]; }
ifstream ifs (filename, ifstream::in );
{
for (int i=0;i<1000;i++)
 {
  for (int j=0;j<1000;j++)
  {
   ifs>>A[i][j];
   }
```

```
  }
 ifs.close();
}
      for (int i=0;i<1000;i+=100)
      {
       for(int j=0;j<1000;j+=100)
         {
            b=i;
             if(i==0) {b=0;}
            f=j;
             if(j==0) {f=0;}
            sum=0;
          if(j<=900 and i<=900)
           { for(int r=b;r<i+100;r++)
             {
              for(int k=f;k<j+100;k++)
              {
               sum=sum+A[r][k];
               }
           }
          }
    if(sum>0) { Image1->Canvas->Pixels[(j-(j/100)*40)+31][(i-(i/100)*40)+33] = sum;
              Image1->Canvas->Pixels[(j-(j/100)*40)+31][(i-(i/100)*40)+34] = sum;
              Image1->Canvas->Pixels[(j-(j/100)*40)+31][(i-(i/100)*40)+35] = sum;
              Image1->Canvas->Pixels[(j-(j/100)*40)+29][(i-(i/100)*40)+33] = sum;
              Image1->Canvas->Pixels[(j-(j/100)*40)+29][(i-(i/100)*40)+34] = sum;
              Image1->Canvas->Pixels[(j-(j/100)*40)+29][(i-(i/100)*40)+35] = sum;
              Image1->Canvas->Pixels[(j-(j/100)*40)+30][(i-(i/100)*40)+33] = sum;
              Image1->Canvas->Pixels[(j-(j/100)*40)+30][(i-(i/100)*40)+34] = sum;
              Image1->Canvas->Pixels[(j-(j/100)*40)+30][(i-(i/100)*40)+35] = sum;
              }
         }
      }
  for (int i=0; i<1000; i++)
  { delete [] A[i]; }
delete []A;
}
//---------------------------------------------------------------------------
IH 21M
```

Вступ

Тема наукової роботи „Сучасні системі візуалізації даних” поєднує у собі програмування та комп’ютерну графіку. Метою даної роботи є створення програмного забезпечення для візуалізації даних по землетрусах з можливістю зберігати результати візуалізації у графічні файли. Предмет дослідження – методи автоматичного представлення даних у візуальному вигляді, перетворення даних у візуальний формат. Об'єкт дослідження – розробка програми для візуального виведення даних на екран та у графічний файл.

Візуальне представлення даних є набагато інформативнішим за інші методи отримання та сприйняття інформації, більш зручним і легшим для сприйняття та розуміння ніж наприклад представлення даних у таблицях, схемах, математичних матрицях, або просто в числах. У наш час поширюються засоби для візуального представлення інформації та даних і для таких цілей широко використовується комп’ютерна техніка, тому виникає необхідність створювати програмне забезпечення для візуалізації даних. З’явились також нові різновиди візуального представлення даних, деякі з них походять від графіків, діаграм, гістограм і не просто візуалізація на площині а візуалізація у 3D просторі.

Візуалізація даних - це наочне представлення великих масивів числової та іншої інформації, яке є можливим завдяки використанню комп'ютерної графіки. Продукти візуалізації даних можуть легко інтегруватися в інформаційні системи. Візуалізація даних може бути здійснена за допомогою таких засобів як інформаційні панелі. Візуалізація даних у своїй роботі завжди потрібна будь-яким дослідникам. До завдання візуалізації даних зводиться проблема подання в наочній формі даних експерименту або результатів теоретичного дослідження.

Створення такого програмного забезпечення це великі обсяги роботи, тому буде доречно використати середовище візуальної розробки, їх у наш час існує велика кількість і такі середовища широко використовуються програмістами.

Враховуючи вимоги до створення програмного забезпечення та актуальність сучасних мов програмування, для створення програмного забезпечення було обрано середовище візуальної розробки програм Borland C++ Builder 6.

Розділ 1 СУЧАСНІ ТЕХНОЛОГІЇ ПРОГРАМУВАННЯ

1.1 Об’єктно-орієнтоване програмування (ООП)

Об'єктно-орієнтоване програмування - результат природної еволюції більш ранніх методологій програмування. Воно виникло з процедурного програмування. В об'єктно-орієнтованому програмуванні ви відходите від ділення задач на під задачі. Ви намагаєтесь побачити вашу задачу, утворену з взаємодій між абстракціями – ідеалізованими об'єктами реального світу. Таким чином, ціль у тому, щоб використовувати в програмуванні повсякденний досвід і знання про поведінку звичайних об'єктів. Об’єкт – це абстракція, але з чітко визначеними властивостями чи ролями. Об'єктно-орієнтоване програмування є методом програмування, який імітує виконання людиною якої-небудь роботи. Воно більш структуроване і більш модульне і абстрактне, ніж традиційне програмування.

Об’єктно-орієнтоване програмування – це методика, що концентрує основну увагу програміста на зв'язках між об'єктами, а не на деталях їхньої реалізації. Класичні принципи ООП – інкапсуляція, наслідування, поліморфізм, створення класів і об'єктів, вони інтерпретуються й доповнюються новими поняттями й термінологією, прийнятими інтегрованими середовищами візуальної розробки.

Наслідування

Щоб найбільш ефективно повторно використовувати раніше створені класи, одного поєднання даних і методів в єдиній структурі недостатньо. Але повністю заново описувати новий тип даних, якщо потрібно змінити чи добавити декілька нових властивостей до старого типу, нераціонально. Це погано ще й тому, що якщо в метод, що є в двох класах, буде потрібно внести виправлення, то їх прийдеться вносити двічі, в двох однакових копіях. Щоб уникнути непотрібної роботи, в об'єктно–орієнтованому програмуванні був введений принцип наслідування властивостей і методів. Програмісту достатньо описати один базовий клас, а класи–послідовники - основувати на цьому базовому класі. При цьому будуть наслідуватися всі поля, властивості і методи базового класу, і додатково їх описувати не потрібно. Ланцюжки наслідування можуть бути необмеженої довжини. При цьому різні методи для кожного з послідовників дозволяється переписувати.

Поліморфізм

Коли буде відбуватися звернення до змінної, що відноситься до класу С, що є послідовником класу В, який в свою чергу наслідує класу А, то програмі доведеться вирішувати, який конкретно метод потрібно викликати: метод класу А, В чи С. У відповідності з принципом поліморфізму рішення приймається в залежності від типу змінної, що викликала цей метод.

Інкапсуляція

Інкапсуляція дозволяє розмежувати доступ розробників до різних полів і властивостей класу, приблизно так, як це зроблено в модулях C++Builder , коли з інших модулів видно лише інтерфейсну частину. Точно так і всередині класу деякі поля і методи можна зробити вільно доступними для використання в будь–якому місці програми, а інші поля і методи зробити доступними лише всередині поточного модуля і власних методів класу. Це дозволяє сховати в середині опис різних характеристик і можливостей класу, щоб зосередити увагу розробників, що повторно використовують цей клас, на його важливих якостях. Крім того, бажано не допускати безконтрольної зміни значень властивостей, так як це може призвести до порушення запланованого і збалансованого взаємозв'язку між цими якостями.

У середовищі C++Builder мова C++ розширена новими можливостями (компоненти, властивості, методи, обробники подій) і останніми доповненнями стандарту ANSI C++ (шаблони, простори імен, явні й непостійні оголошення, ідентифікація типів при виконанні програми RTTI, виключення, динамічні масиви, ключові словами explicit, mutable, typename, automated і ін.). У C++Builder введено нові типи даних: булевий тип даних bool, рядковий тип AnsiString (реалізований як клас, оголошений у заголовочному файлі vcl/dstring.h), тип "множина" (реалізований як шаблон класу, оголошений у заголовочному файлі vcl/sysdefs.h) та інші типи. Вважається що останні версії C++Builder найбільш повно відповідають стандарту ANSI/ISO серед всіх компіляторів на платформі Windows.

Компанія Borland пропонує потужний набір власних розширень мови C++, що забезпечує підтримку VCL бібліотеки (Visual Component Library). У C++Builder поряд із звичайними об'єктними класами мови C++, з’явилися нові компонентні класи (компоненти). Форми є основою додатків C++Builder. Створення користувацького інтерфейсу додатка полягає в додаванні на форму елементів (компонентів). Компоненти C++Builder розташовуються на палітрі компонентів, виконаної у вигляді багатосторінкового блокнота. Важлива особливість C++Builder полягає в тому, що він дозволяє створювати власні компоненти й настроювати палітру компонентів, а також створювати різні версії палітри компонентів для різних проектів.

Компонент (component) – спеціальний клас, властивості якого подають атрибути об'єкту, а його методи реалізують операції над відповідними екземплярами компонентних класів. Поняття властивість, метод, обробник подій розглянемо далі, а зараз перелічимо основні відмінності компонентних класів від об'єктних класів:

– всі компоненти є прямими або непрямими нащадками одногозагального класу-прародича (TComponent);

– компоненти звичайно використовуються безпосередньо,шляхом маніпуляції з їхніми властивостями;

– компоненти розміщаються тільки в динамічній пам'яті купи(heap) за допомогою оператора new, а не на стеку, як об'єкти звичайних класів;

– компоненти можна добавляти до палітри компонентів і далі маніпулювати з ними за допомогою редактора форм інтегрованого середовища візуальної розробки C++Builder;

– оголошення (інтерфейсна частина) компонентного класу обов’язково описується окремо (розширення .h) від реалізації(розширення .cpp).

Форма – це також компонент, і тому з кожною формою пов’язані файл оголошення та файл реалізації. Скелети обох файлів автоматично генеруються середовищем C++Builder при візуальному створенні нової форми, а програміст доповнює ці файли власним кодом. При додаванні у форму будь-якого компонента з палітри компонентів C++Builder автоматично формує програмний код для створення об'єкта (змінної) даного типу. Змінна додається як член класу даної форми.

1.2 Візуальне програмування інтерфейсів програмного забезпечення

Візуалізація – це процес графічного відображення складних процесів на екрані комп'ютера у вигляді графічних примітивів (фігур). Візуалізувати можна будь-які процеси: управління, побудови, малювання та інші. Приклад найпростішого варіанта візуалізації – лінійка прогресу (прямокутник, відсоток заповнення якого прямо пропорційний об'єму виконання якої-небудь операції). Дивлячись на лінійку, можна оцінити об'єм невиконаних операцій, який залишився.

Візуалізовувати можна інтерфейси програмного забезпечення. Це дозволяє спростити взаємодію програмного продукту з користувачем. Зображення на елементах інтерфейсу (зовнішнього вигляду програмного забезпечення) дозволяють користувачу інтуїтивно розбиратися в призначенні цих елементів.

Для візуалізації інтерфейсів програмного забезпечення існує цілий ряд спеціально розроблених елементів інтерфейсу – візуальних компонентів, що дозволяють відображати різну інформацію й здійснювати керування програмою в цілому. Найпростіший приклад – візуальна кнопка на екрані комп'ютера. Візуальна кнопка імітує поведінку звичайної кнопки на пульті керування будь-якого приладу. Її можна "натискати" як справжню.

Саме наявність візуальних засобів побудови інтерфейсів для Windows в C++Builder, а також створюване ними візуальне програмне забезпечення закріпили за ним термін "візуальне програмування".

Визначальними елементом процесу візуалізації є візуалізована модель, тобто модель, що піддається відображенню з метою можливості зміни її структури або її параметрів (або параметрів її окремих частин). Візуалізованою моделлю в C++Builder є вікно (форма, діалог) Windows, а не код програми. У C++Builder прийнято візуалізовувати тільки роботу з елементами інтерфейсу, коли об'єкти візуалізації розглядаються як візуальні компоненти, з яких складаються форми (вікна й діалоги) інтерфейсу програми.

Інструменти візуальної розробки в середовище C++Builder включають в себе дизайнер форм, інспектор об'єктів, палітру компонентів (вікно, що містить набір компонентів, з яких будується візуальна модель), менеджер проектів і редактор коду.

Ці інструменти включені в інтегроване середовище системи (Integrated Development Environment, IDE), яке забезпечує продуктивність багаторазового використання візуальних компонентів у поєднанні з удосконаленими інструментами й засобами доступу до баз даних.

Конструювання способом “перетягування” (drag-and-drop) дозволяє створювати додаток простим перетаскуванням захоплених мишею візуальних компонентів з палітри на форму додатка.

Механізми дво напрямної розробки (Two-Way-Tools) забезпечують контроль коду за допомогою гнучкої, інтегрованої й синхронізованої взаємодії між інструментами візуального проектування й редактором коду.

Інспектор об'єктів надає можливість оперувати властивостями (вікно властивостей – вікно, у якому відображаються параметри обраного елемента візуальної моделі) й подіями компонентів.

Компоненти можуть бути візуальні, видимі при роботі додатку, і не візуальні, виконуючі ті або інші службові функції. Візуальні компоненти відразу видні на екрані у процесі проектування у такому ж вигляді, в якому їх побачить користувач під час виконання додатку. Це дозволяє дуже легко вибрати місце їхнього розташування та їхній дизайн – форму, розмір, оформлення, текст, колір та інше. Не візуальні компоненти видимі на формі лише в процесі проектування у вигляді піктограм, але для користувача під час виконання вони не видимі, хоча й виконують для нього за кадром корисну роботу.

Візуальне програмування дозволяє звести проектування користувацького інтерфейсу до простих і наочних процедур. Але переваги візуального програмування не зводяться лише до цього.

Саме головне полягає в тому, що під час проектування форми й розміщення на ній компонентів C++Builder автоматично формує коди програми, включаючи в неї відповідні фрагменти, що описують даний компонент. А потім у відповідних діалогових вікнах користувач може змінити задані за замовчуванням значення властивостей цих компонентів і, при необхідності, написати обробники потрібних подій. Тобто проектування зводиться, фактично, до розміщення компонентів на формі, завдання деяких їхніх властивостей і написання, при необхідності, обробників подій. Результатом візуального проектування є скелет майбутньої програми, у яку вже внесені відповідні коди.

1.3 Технологія швидкої розробки додатків (RAD)

Завдяки візуальному об’єктно-орієнтованому програмуванню (ООП) була створена технологія, що одержала назву швидка розробка додатків (RAD — Rapid Application Development). Ця технологія характерна для нового покоління систем програмування, до якого відноситься й C++Builder.

C++Builder – це інструмент для швидкої розробки додатків (RAD) на C++ під Windows, який підтримує можливість програмування, що ґрунтується на компонентах.

Компоненти – це будівельні блоки для додатків. Тобто, використовуються об’єкти-компоненти зі своїми можливостями і об’єднуються в один додаток. C++Builder сам побудований на компонентах і робота з компонентами в C++Builder проста і надійна.

Швидка розробка додатків (RAD) означає підтримку властивостей, методів і подій компонентів у рамках об'єктно-орієнтовного програмування.

Властивості дозволяють легко встановлювати різноманітні характеристики компонентів, такі як назви, контекстні підказки або джерела даних. Методи (функції-члени) роблять певні операції над компонентним об'єктом. Події зв'язують впливи користувача на компоненти із кодами реакції на ці впливи.

Працюючи спільно, властивості, методи і події утворять середовище RAD інтуїтивного програмування надійних додатків для Windows.

Як було сказано вище, компонент – це спеціальний клас, властивості якого подають атрибути об'єкту, а його методи реалізують операції над відповідними екземплярами компонентних класів.

Якщо вибрати компонент із палітри й додати його до форми, інспектор об'єктів автоматично покаже властивості й події, які можуть бути використані із цим компонентом. У верхній частині інспектора об'єктів є список, що випадає, що дозволяє вибирати потрібний об'єкт із наявних на формі.

Поняття метод звичайно використовується в контексті компонентних класів і зовнішньо не відрізняється від терміна функція-член звичайного класу. Щоб викликати метод, треба вказати ім'я функції в контексті даного класу. Саме схований зв'язок методу з класом виділяє його з поняття простої функції. Під час виконання методу він має доступ до всіх даних свого класу. Це забезпечується передачею кожному методу схованого параметра – вказівника **this** на екземпляр класу. При будь-якому звертанні методу до членів даних класу, компілятор генерує спеціальний код, що використовує вказівник **this**. Метод є функцією, що пов'язана з компонентом і оголошується як частина об'єкта. Методи можна викликати, використовуючи операцію доступу через вказівник -> для цього компонента.

C++ Builder дозволяє маніпулювати виглядом і функціональною поведінкою компонентів не тільки за допомогою методів (як це роблять функції-члени звичайних класів), але й за допомогою властивостей і подій, властивих тільки класам компонентів. Маніпулювати з компонентним об'єктом можна як на стадії проектування додатка, так і під час його виконання.

Властивості (properties) компонентів являють собою розширення поняття членів даних і, хоча не бережуть дані як такі, проте забезпечують доступ до членів даних об'єкта. C++Builder використовує ключове слово **__property** для оголошення властивостей. Властивості є атрибутами компонента, що визначають його зовнішній вигляд і поведінку. Багато властивостей компонента мають значення, за замовчуванням (наприклад, висота кнопок). Властивості компонента відображаються на сторінці властивостей (Properties). Інспектор об'єктів відображає опубліковані (published) властивості компонентів. Крім published-властивостей, компоненти можуть і найчастіше мають загальні (public) властивості, які доступні тільки під час виконання додатка. Інспектор об'єктів використовується для встановлення властивостей під час проектування. Список властивостей розташовується на сторінці властивостей інспектора об'єктів. Можна визначити властивості під час проектування або написати код для видозміни властивостей компонента під час виконання додатка.

При визначенні властивостей компонента під час проектування потрібно вибрати компонент на формі, відкрити сторінку властивостей в інспекторі об'єктів, вибрати потрібну властивість і змінити її за допомогою редактора властивостей (це може бути просте поле для введення тексту або числа; список, що випадає; список, що розкривається діалогова панель та інше).

Сторінка подій (Events) інспектора об'єктів показує список подій, розпізнаваних компонентом. За допомогою подій (events) компонент повідомляє користувачу про те, що на нього зроблений деякий визначений вплив (програмування для операційних систем із графічним користувацьким інтерфейсом, зокрема, для Windows XP або Windows NT передбачає опис реакції додатка на ті або інші події, а сама операційна система займається постійним опитуванням комп'ютера з метою виявлення настання якої-небудь події). Кожний компонент має свій власний набір обробників подій. У C++Builder необхідно писати функції, які називаються обробниками подій, і зв'язувати події із цими функціями. Якщо подія відбудеться і обробник цієї події написаний, то програма виконає написану функцію.

Для того, щоб додати обробник подій, потрібно вибрати на формі за допомогою миші компонент, якому необхідний обробник подій, потім відкрити сторінку подій інспектора об'єктів і двічі клацнути лівою клавішею миші на колонку значень поруч із подією, щоб C++Builder згенерував прототип обробника події і показав його в редакторі коду. При цьому автоматично генерується текст порожньої функції, і редактор відкривається в тому місці, де треба вводити код. Курсор позиціюється у середині операторних дужок { ... }. Далі потрібно ввести код, який повинен виконуватися при настанні події. Обробник подій може мати параметри, які вказуються після імені функції в круглих дужках.

C++Builder використовує ключове слово **__closure** для оголошення подій. Основна сфера застосування методів – це обробники подій (event handlers), що реалізують реакцію програми на виникнення визначених подій. Типові прості події – натискання кнопки або клавіші на клавіатурі.

1.4 Узагальнене програмування (generic programming)

Поряд з ООП у C++Buider широко підтримується узагальнене програмування, яке спрямоване на спрощення повторного використання коду і на абстрагування загальних концепцій. Якщо в ООП акцент програмування ставиться на дані, то в узагальненому програмування – на алгоритми.

Узагальнене програмування має справу з абстрагуванням і класифікацією алгоритмів і структур даних. У багатьох випадках алгоритм можна виразити незалежно від деталей подання оброблюваних ним даних. Термін узагальнений приписується коду, тип якого є незалежним. В узагальненому програмуванні можна один раз написати функцію для узагальненого, тобто невизначеного типу і потім використовувати її для множини існуючих типів.

Концепція узагальненого програмування припускає використання типів даних як параметрів. При розробці алгоритму, що може працювати із множиною типів і структур даних, використовується якийсь абстрактний тип, що згодом параметризується. Такий підхід забезпечує простий спосіб введення різного роду загальних концепцій і позбавляє програміста від написання вручну спеціалізованого коду.

У мові програмування C++ узагальнене програмування реалізується за допомогою шаблонів. Шаблон являє собою параметризоване визначення деякого елемента програми. При цьому розроблювач усього лише вказує компілятору правило для генерації (породження) одного або декількох конкретних екземплярів цього елемента програми. У C++ допускається створення шаблонів для функцій, методів і класів. При цьому як параметр шаблона можна використовувати тип (клас), звичайний параметр відомого типу, інший шаблон. У шаблона може бути кілька параметрів.

Поряд із засобами визначення власних користувацьких шаблонів у розпорядження розробника надається стандартна бібліотека. У ній визначені шаблони, що представляють найбільш використовувані структури даних, а також алгоритми для їхньої обробки.

Отже, найбільш перспективним стає наступний (більш високий у порівнянні із класами) рівень абстрактного програмування – створення своїх і використання стандартних шаблонів і узагальнених алгоритмів стандартної бібліотеки, які вже розроблені й налагоджені професіоналами.

1.5 Огляд бібліотеки VCL

Бібліотека візуальних компонентів VCL (Visual Component Library) – об’єктно-орієнтована бібліотека для розробки програмного забезпечення, розроблена компанією Borland для підтримки візуального програмування. VCL входить у комплект поставки C++Builder, Delphi і Borland Developer Studio і є частиною середовища розробки, хоча розробка додатків у цих середовищах можлива й без використання VCL. VCL представляє величезну кількість (більше 360) готових до використання компонентів для роботи в самих різних областях програмування, таких, наприклад, як інтерфейс користувача (екранні форми, елементи керування та інші), робота з базами даних, взаємодія з операційною системою, програмування мережних додатків та інше.

Сукупність функцій, за допомогою яких здійснюється доступ до системних ресурсів, називається прикладним програмним інтерфейсом, або API (Application Programming Interface). Для взаємодії з Windows додаток використовує функції API, за допомогою яких реалізуються всі необхідні системні дії, такі як виділення пам'яті, вивід на екран, створення вікон і т.п. Бібліотека VCL є просто оболонкою для WinAPI. Класи VCL створені тільки для того, щоб полегшувати процес програмування. Наприклад, один з найважливіших класів TForm з бібліотеки VCL можна було б створити, не використовуючи класи VCL.

Компонент C++Builder – це особливий вид об'єктів – візуальний об'єкт (візуальний для проектування, а не для відображення користувача). Створювати й редагувати такий об'єкт можна як програмним шляхом, так і на етапі проектування.

При виконанні програми компоненти діляться на візуальні, які бачить користувач, і не візуальні, для яких немає можливості їхнього відображення, але доступ до властивостей яких дозволений.

Усі компоненти мають загального предка – клас TComponent. Усі компоненти C++Builder можуть бути доступні через палітру компонентів. Частина компонентів є елементами керування. В основному – це елементи керування Windows.

Елементи керування можна підрозділити на віконні й невіконні. Віконні елементи можуть одержувати фокус вводу і мають дескриптор вікна. Предком всіх віконних елементів керування є абстрактний клас TWinControl. Предком не віконних елементів керування є абстрактний клас TGraphicControl.

При додаванні у форму будь-якого компонента з палітри компонентів C++Builder автоматично формує програмний код для створення об'єкта (змінної) даного типу. Змінна додається як член класу даної форми.

Класи бібліотеки VCL використовують механізм простого успадкування: один клас може мати тільки одного предка. Коренем ієрархії класів є клас **TObject**. Далі на Рис. 1. наведено ієрархію ключових базових класів бібліотеки VCL.

Будь-який клас VCL-бібліотеки успадковується від класу TObject. Клас **TPersistent** пішов безпосередньо від класу TObject. Він забезпечує своїх нащадків можливістю взаємодіяти з іншими об'єктами і процесами на рівні даних. Його методи дозволяють передавати дані в потоки, а також забезпечують взаємодію об’єкта з інспектором об'єктів. Клас TPersistent має метод Assign, який дозволяє передавати поля і властивості одного об’єкту іншому.

Клас **TComponent** є предком всіх компонентів VCL-бібліотеки. Всі нащадки даного класу можуть бути розташовані в палітрі компонентів.

Клас **TComponent** дозволяє визначати батьківський елемент керування й власника компонента. Батьківським елементом керування називається той, у який безпосередньо поміщений даний компонент. Власником всіх компонентів, розташованих у формі, є сама форма. Власником всіх форм є додаток.

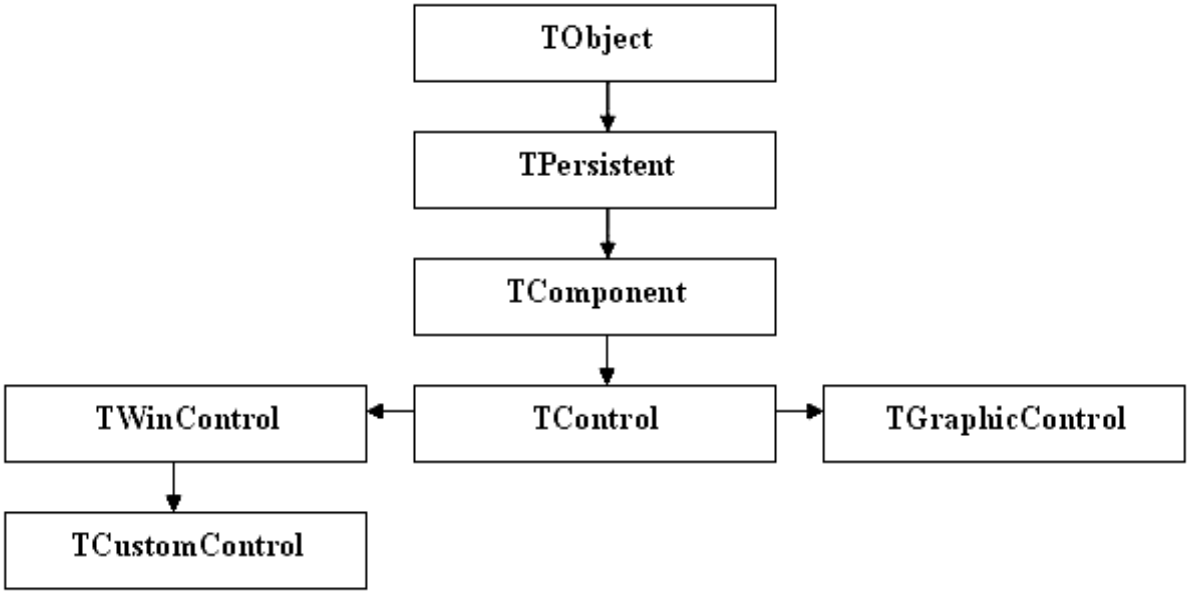

Рис. 1. Ієрархія ключових базових класів бібліотеки VCL

Якщо компонент розташований не безпосередньо у формі, а, наприклад, у компоненті типу TPanel, то власник і батьківський елемент керування в нього будуть різні.

TControl – це базовий клас всіх елементів керування (включаючи й вікно форми). Ці компоненти можуть бути видимі під час виконання. Для них визначені такі властивості, як позиція, курсор, підказка, що спливає, методи для малювання або переміщення елемента керування, події для маніпуляцій за допомогою миші.

Клас **TWinControl** є базовим класом всіх віконних елементів керування.

Клас **TApplication** інкапсулює об'єкт "windows-додаток". За допомогою цього класу визначається інтерфейс між розробником і середовищем Windows. У кожному додатку C++Builder завжди автоматично створюється один об’єкт Application як екземпляр класу додатка. Для більшості додатків цей об’єкт є екземпляром класу TApplication. Компонент TApplication не відображається в палітрі компонентів і не має властивостей, що публікуються. Для того, щоб мати можливість перехоплювати події для додатка, можна додати в будь-яку форму проекту компонент TApplicationEvents.

Кожний додаток C++Builder має глобальну змінну Screen типу **TScreen**. Компонент **TScreen**, так само як і компонент **TApplication**, недоступний з інспектора об'єктів. Цей компонент призначений для забезпечення доступу до пристрою виводу – екрану. Його властивості містять інформацію про використовуване розширення монітора, курсорах і шрифтах, доступних для додатка, списку форм додатка й активній формі.

TForm є базовим класом для створення вікна форми. За замовчуванням кожна нова створювана форма реалізується як нащадок класу **TForm**. Форма може бути головним вікном додатка, діалоговим вікном, дочірнім або MDI-вікном.

Клас форми є контейнером для всіх компонентів, розташовуваних на формі. Для доступу до властивостей форми або іменам компонентів можна використовувати вказівник **this**. Якщо перед іменем властивості відсутній який-небудь вказівник, то за замовчуванням вважається, що це властивість форми.

Форма є компонентом VCL, тому формі притаманні певні загальні властивості компонентів VCL.

1.6 Продуктивність візуальних компонентів VCL

Бібліотека візуальних Компонентів VCL придбала статус нового промислового стандарту і в цей час застосовується більш ніж півмільйоном користувачів, істотно прискорюючи розробку надійних додатків будь-якого ступеня складності. VCL містить близько 100 повторно використовуваних компонентів, які реалізують всі елементи користувальницького інтерфейсу операційної системи Windows. Крім того VCL надають у розпорядження програмістів такі оригінальні об'єкти, як записні книжки із закладками, табличні сітки для відображення вмісту баз даних і навіть органи керування пристроями мультимедіа. Перебуваючи в середовищі об‘єктно - орієнтованого Програмування C++Builder, компоненти можна використати безпосередньо, міняти їхні властивості, вигляд і поводження або породжувати похідні елементи, з потрібними відмітними характеристиками.

Сховище об'єктів є інструментом нової методики зберігання й повторного використання модулів даних, об'єктів, форм і програмної логіки. Оскільки побудова нового додатка на існуючому фундаменті значно заощаджує тимчасові витрати, сховище об'єктів надає для повторного використання готові структури: форми та закінчені програмні модулі. Створюючи прототип нового додатка, можна успадковувати, посилатися або просто копіювати існуючу структуру - точно так само архітектор приступає до проектування нового будинку.

Компонента Chart забезпечує швидку побудову на формі різноманітних графіків, діаграм, таблиць і передбачає перевірку правопису на багатьох мовах. У варіанті C++Builder Standard цей компонент є єдиним представником групи Active.

Інтеграція компонентів Active дозволяє розширити бібліотеку візуальних компонентів, включивши компоненти стандарту Active для розробки додатків у мережі Internet.

1.7 Огляд стандартної бібліотеки шаблонів STL

Стандартна бібліотека шаблонів (Standard Template Library - STL) об'єднує в собі контейнерні типи даних, алгоритми для їх обробки, ітератори для звернення до елементів або послідовностей в контейнері. STL також містить набір шаблонів для забезпечення стандартного введення-виведення.

Слово «стандартна» означає, що ця бібліотека є частиною стандарту мови C++ і повинна розглядатися, як перша альтернатива при виборі методів і засобів роботи з даними і потоками введення-виводу.

Слово «шаблонів» означає, що уся бібліотека побудована на шаблонах класів і функцій, що забезпечує можливість уніфікованої роботи з різними типами даних. Використання шаблонів в бібліотеці дозволяє не лише однаково обробляти вбудовані типи, але і працювати з призначеними для користувача типами даних, які не були відомі у момент розробки.

STL має ряд переваг:

Код бібліотеки написаний професійними програмістами, перевірений і відлагоджений. Вам не доведеться шукати помилки в реалізації контейнерів або алгоритмів STL. Швидше, помилки будуть пов'язані з невірним розумінням концепцій STL, але це питання досвіду використання.

Код бібліотеки написаний дуже ефективно з точки зору використання оперативної пам'яті і швидкодії для типових варіантів застосування.

Бібліотека пропонує уніфікований інтерфейс, одноманітний для усіх контейнерів і алгоритмів, що після отримання навички використання бібліотеки дозволяє значно підвищити читаність програми.

Використання бібліотеки дозволяє приступити відразу до рішення проектних завдань, не замислюючись про реалізацію низькорівневих контейнерів і алгоритмів. У разі роботи проектної групи це дозволяє уникнути дублювання коду у різних розробників.

• Бібліотека добре документована і описана в книгах. У разі розробки власних контейнерів і алгоритмів документація буде, швидше за все, значно менша, що підвищить вартість підготовки нового фахівця.

• Код, написаний з використанням STL легко переноситься на інші компілятори, операційні системи і платформи.

До недоліків STL можна віднести:

• Непристосованість до роботи із структурними типами даних.

• Низька ефективність (швидкодія, пам'ять) при рішенні приватних завдань, де можливі цільові оптимізації коду.

• Неадекватний інтерфейс шаблону для роботи з рядками

• Складність управління пулом пам'яті при роботі з контейнерами STL.

Вказані недоліки в основному торкаються рівня професійних програмістів, які здатні самостійно проаналізувати ефективність використання тієї або іншої бібліотеки. Для початкуючого програміста ця бібліотека є засобом, що дозволяє значно підвищити продуктивність праці і швидко перейти від рішення низькорівневих завдань до проблем предметної області.

Тому бібліотека STL в обов'язковому порядку має бути вивчена розробником і повинна всюди використовуватися до того часу, поки він не зможе самостійно переконливо аргументувати вибір іншої бібліотеки або розробку власних контейнерів і алгоритмів.

Перелічимо основні можливості стандартної бібліотеки C++, які з’явилися в ній з розвитком самої мови C++: потоки, числові методи (наприклад, операції з комплексними числами, множення масиву на константу та інші нескладні методи), рядки (класи для роботи з текстовою інформацією), множини, контейнери, алгоритми й об’єкти-функції, ітератори та інше.

STL – частина стандартної бібліотеки C++ SCL, що забезпечує загальноцільові стандартні класи й функції, які реалізують найбільш популярні й широко використовувані алгоритми й структури даних.

Бібліотека STL розроблена співробітником Hewllet-Packard Олександром Степановим. STL будується на основі шаблонів класів, і тому вхідні в неї алгоритми й структури застосовні майже до всіх типів даних. Ядро бібліотеки утворюють три складові: контейнери, алгоритми й ітератори.

Шаблони контейнерів, алгоритми та й взагалі вся бібліотека винесені в окремий простір імен, який одержав назву std. Існує кілька способів користуватися ключовими словами із простору імен std:

- після всіх файлів, що підключаються (наприклад, #include<set>), використати директиву

using namespace std;

- підключити бажані модулі STL окремими директивами using:

using std::stack;

using std::find;

- вказувати оператор прив’язки std:: перед кожним STL типом даних або алгоритмом.

Контейнери

Бібліотека STL представляє ряд контейнерів для зберігання даних, які умовно можна розділити на контейнери з довільним доступом, послідовні контейнери і асоціативні контейнери Контейнерами з довільним доступом є vector і deque. Обидва контейнери забезпечують довільний доступ до елементів (як до елементів звичайних масивів C). Основна відмінність vector і deque полягає в тому, що vector забезпечує додавання нових елементів у кінці, а deque - у кінці і на початку контейнера.

Приклад використання контейнера vector

```
#include <vector>
#include <iostream>
 int main( )
{
  using namespace std;
   vector <int> v1;
   v1.push_back( 10 );
   v1.push_back( 20 );
   cout << "The second integer of v1 is " << v1[1] << endl;
}
```

Контейнери з довільним доступом дозволяють звертатися до елементу за допомогою оператора []. Додавання елементів в vector проводиться за допомогою методу push_back, що додає новий елемент у вектор. Використання елементу до його додавання (чи резервування) неприпустимо.

Контейнером з послідовним доступом є list (пов'язаний список).

Також є контейнери, що визначають пріоритетну чергу (priority_queue) і стек (stack).

Асоціативні контейнери представлені словником (map, multimap) і набором (set, multiset), а також їх аналогами, реалізованим з використанням хеш-таблиць.

Асоціативні контейнери дозволяють зберігати дані у впорядкованому виді і забезпечують швидкий пошук потрібних значень по ключу. Приміром, set дозволяє зберігати набори унікальних (за значенням) об'єктів. А multiset дозволяє зберігати об'єкти, що повторюються. Контейнер map дозволяє зберігати словник пар ключ-значення і здійснювати швидкий пошук потрібного елементу по ключу.

Рядки

Шаблон sting реалізує функціонал, необхідний для роботи з рядками тексту. У відмінності від стандартного представлення рядка у вигляді масиву символів константною довгі шаблон string динамічно розподіляє пам'ять для рядка в кулі і гарантує достатній об'єм пам'яті при будь-яких операціях з рядками: введенні, злитті рядків і так далі

Шаблон забезпечує звільнення пам'яті, зайнятого строковим об'єктом, після того, як він стане не потрібний. Методи шаблону забезпечують усі необхідні алгоритми роботи з рядками: пошук, заміну, вставку, порівняння. Використання шаблону string значно спрощує роботу з рядками для програміста початківця.

Ітератори

Ітератори використовуються для доступу до елементів контейнерів також, як покажчики використовуються для доступу до елементів масивів в З . Ітератор є "розумним" покажчиком на елемент контейнера. На відміну від звичайного покажчика він пам'ятає повний тип даних на який посилається - з урахуванням типу контейнера, до елементу якого проводиться звернення.

Приклад використання ітератора:

```
 struct Circle {
 Circle() : radius(0) {}
 Circle(float radius) : radius(radius) {}
 float radius;
};
 int main()
{
 using namespace std;
```

```
 Circle c1(10),c2(20);
 map<string,Circle> circles;
 circles["one"]=c1;
 circles["two"]=c2;
 map<string,Circle>::iterator it = circles.begin();
 map<string,Circle>::const_iterator end = circles.end();
  for ( ; it != end; ++it)
{   cout << "NameIs: " << (it->first) << endl;
   cout << "Radius: " << (it->second).radius << endl;
 }
}
```

Більшість контейнерних типів надають два спеціальні методи - begin і end. Метод begin повертає ітератор, що вказує на перший елемент контейнера. Метод end повертає ітератор на елемент контейнера, що йде за останнім. Не можна звертатися до елементу, на який вказує ітератор, повертаний по end. Він служить лише для обмеження дії алгоритмів.

Ітератори бувають декількох типів - вхідні, вихідні, прямі, двонаправлені і довільного доступу. Тип ітератора визначає підмножину операторів, застосованих до нього. Наприклад, ітератор довільного доступу підтримує можливість додавання певного зміщення (i n), в той час, як інші ітератори дозволяють тільки операції зміщення ітератора на одну позицію і або і. Двонаправлені ітератори дозволяють переміщатися і у зворотному напрямі (i-- і --i).

Як і у випадку з покажчиками - основна операція, що застосовується до ітератора, - розіменування. Тобто звернення до об'єкту, на який вказує ітератор.

Алгоритми

Бібліотека STL надає основні види алгоритмів :

- Математичні (розрахунок сум, творів, генерація випадкових значень)

- Пошуку (мінімальне значення, пошук послідовності, підрахунок числа значень)

- Сортування

- Роботи з послідовностями (об'єднання послідовностей, порівняння, обробки послідовності типовою операцією)

Приклад використання алгоритму

```
#include <vector>
#include <iostream>
int main()
{
using namespace std;
const int n=4;
int a[n];
generate(a,a+n,rand);
copy(a,a+n,ostream_iterator<int>(cout," "));
}
```

Розділ 2 КОМП'ЮТЕРНА ГРАФІКА ТА РЕСУРСИ ДЛЯ СТВОРЕННЯ ГРАФІЧНИХ ДОДАТКІВ

2.1 Растрова, векторна і фрактальна графіка

Під терміном "графіка" розуміють результат візуального подання реального або уявного об'єкта, отриманого традиційними методами – малюванням або друкуванням. Комп'ютерна графіка включає методи і засоби створення і обробки зображень за допомогою програмно-апаратних комплексів і охвачує всі види і форми подання зображень, доступних для сприйняття людиною на екрані монітору або в вигляді копії на певному носії.

В практиці комп'ютерної графіки виконання роботи по створенню, редагуванню та обробці зображень часто відокремлено від її графічного подання. Зображенням тут вважається об'єкт, відтворений пристроєм виводу, коли графічні дані візуалізуються.

В залежності від способу опису та формування зображення розрізняють растрову, векторну та фрактальну графіку.

Історично термін "растр" вказував на те, що пристрій при відтворенні зображення використовує набори пікселів (точок), організовані в вигляді послідовностей рядків розгортки. Растрові дані являють собою набори числових значень, які визначають кольори окремих пікселів, впорядкованих таким чином, щоб їх легко було відобразити на растрових пристроях.

Базовим елементом растрової графіки є піксель. Логічні пікселі подібні до математичних точок: вони мають місцеположення, але не займають фізичного простору. Фізичні пікселі – це реальні точки, що відображаються пристроєм виводу. Вони є найменшими фізичними елементами поверхні відображення і займають її певну площу. В зв'язку з цим на відстань між двома сусідніми пікселями вводяться обмеження. Якщо задати пристрою відображення надто високу роздільну здатність (кількість пікселів на одиницю довжини зображення), то якість зображення знизиться із-за накладення або злиття сусідніх пікселів. При надто низькій роздільній здатності пікселі можуть бути розкидані по всій площі пристрою відображення. Таким чином, при відображенні значень логічних пікселів із растрових даних в фізичні пікселі повинні враховуватись реальні розміри і розміщення фізичних пікселів.

Розрізняють роздільну здатність оригіналу, екранного та друкованого зображення. Роздільна здатність оригіналу вимірюється в точках на дюйм (dpi) і залежить від вимог до якості зображення, розміру файла, способу оцифровування або методу створення початкової ілюстрації, вибраного формату файла. Підвищення вимог до якості зображення вимагає вищої роздільної здатності оригіналу. Для екранної копії достатньо роздільної здатності 72 dpi, для роздруковування на кольоровому принтері – 150-200 dpi, для виведення на фотоекспонуючий пристрій – 200-300 dpi.

Розмір точки растрового зображення залежить від методу і пара–метрів раструванння оригіналу, коли на оригінал як би накладається сітка ліній, комірки якої утворюють елемент растра. Частота сітки растра вимірюється кількістю ліній на дюйм і називається лініатурою (lpi). Розмір точки растра розраховується для кожного елементу і залежить від інтенсивності тону в комірці. Для вищої інтенсивності щільніше заповнюється елемент растра. При раструванні з амплітудною модуляцією ілюзія більш темного тону створюється за рахунок збільшення розмірів точок при однаковій відстані між центрами елементів растра. При раструванні з частотною модуляцією інтенсивність тону регулюється зміною відстані між сусідніми точками однакового (найменшого) розміру. Інтенсивність тону прийнято розділяти на 256 рівнів. Для її відтворення достатньо мати розмір комірки растра 16х16 точок.

Растрова графіка використовується в випадках, коли потрібна висока точність в передачі кольорів і напівтонів. Однак при цьому розміри файлів суттєво збільшуються з ростом роздільної здатності (одиниці, десятки і сотні Мбайт). До недоліків растрової графіки, окрім великих розмірів файлів, слід віднести пікселізацію зображень при їх збільшенні та деформацію при зменшенні.

В векторній графіці базовим елементом зображення є лінія, яка описується математично як єдиний об’єкт, тому обсяг даних для відображення об’єкта засобами векторної графіки суттєво менший, ніж в растровій графіці. Лінія характеризується формою, товщиною, кольором, типом (суцільна, пунктирна і т.п.). Замкнуті лінії мають властивість заповнення простору, що ними охвачується, іншими об’єктами або кольором. Найпростіша незамкнута лінія обмежена двома точками, які називаються вузлами, котрі мають властивості, що впливають на форму кінця лінії і характер сполучення з іншими об’єктами. Всі інші об’єкти векторної графіки складаються з ліній. Найпростішими лініями є пряма (нескінченна), відрізок прямої, криві другого порядку (не мають точок згину – параболи, гіперболи, еліпси, кола), криві третього порядку (можуть мати точки згину), криві Безьє (основані на використанні пари дотичних, проведених до відрізка лінії в її кінцях, кути нахилу і довжина яких впливають на форму лінії).

Векторна графіка зручна для зберіганні і обробки зображень, що складаються з ліній, або можуть бути розкладені на прості геометричні об’єкти. Векторні дані легко масштабувати та виконувати над ними інші перетворення (наприклад, повертання зображення, додавання, видалення або зміну окремих елементів зображення). Поряд з цим векторні файли важко застосувати для зберігання складних фотореалістичних зображень. Векторні дані краще відображаються на векторних пристроях виводу (плотерах, дисплеях з довільним скануванням). Ефективно векторну графіку можна відобразити тільки на растрових дисплеях з високою роздільною здатністю. Візуалізація векторних даних може вимагати значно більше часу, ніж візуалізація растрових даних рівної складності.

Фрактальна графіка, як і векторна, основана на математичних обчисленнях. ЇЇ базовим елементом є математична формула, виключно на основі якої будується зображення. Таким способом будують як найпростіші регулярні структури, так і складні ілюстрації, що імітують природні ландшафти і тривимірні об’єкти.

2.2 VCL – надбудова над GDI

GDI (Graphics Device Interface) – це та частина Windows, що забезпечує підтримку апаратно-незалежної графіки. C++Builder інкапсулює функції Windows GDI на різних рівнях. Найбільш важливим тут є спосіб, за допомогою якого графічні компоненти представляють свої зображення на екрані монітора. При прямому виклику функції GDI необхідно передавати їм дескриптор контексту пристрою (device context handle), що задає обрані інструменти малювання – перо, пензель і шрифт. Після завершення роботи із графічними зображеннями обов'язково треба привести контекст пристрою у вихідний стан і тільки потім звільнитися від нього.

Типовий приклад малювання із застосовуванням стандартних функцій GDI може виглядати так:

```
void __fastcall TForm1::FormPaint(TObject *Sender)
{   HDC dc = GetDC(Handle); // дескриптор контексту
```

```
HPEN PenHandle, OldPenHandle;
PenHandle = CreatePen(PS_SOLID, 1, RGB(255, 0, 0));
OldPenHandle = SelectObject(dc, PenHandle);
HBRUSH BrushHandle, OldBrushHandle;
BrushHandle = CreateSolidBrush(RGB(255, 255, 0));
OldBrushHandle = SelectObject(dc, BrushHandle);
// жовтий еліпс, обведений червоним контуром
Ellipse(dc, 10, 10, 120, 100);
SelectObject(dc,OldBrushHandle) ;
DeleteObject(BrushHandle) ;
SelectObject(dc,OldPenHandle);
DeleteObject(PenHandle) ;
}
```

Замість того, щоб працювати із графікою на такому рівні деталізації, C++Builder надає простий і завершений інтерфейс за допомогою властивості Canvas (канва) графічних компонентів бібліотеки VCL. Ця властивість ініціалізує правильний контекст пристрою й звільняє його в потрібний час, коли припиняється малювання. Дескриптор контексту пристрою, над яким "побудована" канва, може бути потрібним для різних низькорівневих операцій. Він задається властивістю канви Handle. Клас TCanvas є обгорткою для HDC (HDC доступний через властивість Handle) і представляє більш високорівневий інтерфейс для роботи із графікою.

За аналогією з функціями Windows GDI канва має вкладені властивості, що представляють характеристики пера (Pen), пензеля (Brush) й шрифту (Font).

Єдине, що повинен зробити користувач, працюючи із графічними компонентами, – це визначити характеристики використовуваних інструментів малювання. З використанням Canvas зникає необхідність стежити за системними ресурсами при створенні, виборі й звільненні інструментів. Канва сама дбає про це.

У ряду компонентів з бібліотеки візуальних компонентів VCL є властивість Canvas (канва), яка надає простий шлях для малювання на них. TCanvas – це об’єктний клас, який інкапсулює графічні функції Windows. Канва не є компонентом, але вона входить як властивість у ряд компонентів, які повинні вміти намалювати себе й відобразити яку-небудь інформацію. Canvas (канва) надає простий шлях для малювання, наприклад, на компонентах TImage, TPaintBox, TForm.

Канва містить методи-надбудови над всіма основними функціями малювання GDI Windows. Всі геометричні фігури малюються поточним пером. Ті з них, які можна зафарбовувати, зафарбовуються за допомогою поточного пензля. Пензель і перо при цьому мають поточний колір. Крім того, можна малювати й поточково, одержавши доступ до кожного пікселя.

Клас TCanvas інкапсулює графічні методи: Draw, TextOut, Arc, Rectangle та ін. Використовуючи властивість Canvas, можна відтворювати на формі будь-які графічні об'єкти – картинки, многокутники, текст і т.п. без використання компонентів TImage, TShape і TLabel (тобто без використання додаткових ресурсів), однак при цьому треба обробляти подію OnPaint того об'єкта, на канві якого відбувається малювання.

Розглянемо приклад коду з використанням Canvas, який знову, як і в попередньому прикладі, малює жовтий еліпс, обведений червоним контуром, і наочно ілюструє, наскільки C++Builder спрощує програмування графіки.

```
void __fastcall TForm1::FormPaint(TObject *Sender)
{   Canvas->Pen->Color = clRed; // колір контуру
Canvas->Brush->Color = clYellow; // колір заливки
// жовтий еліпс, обведений червоним контуром
Canvas->Ellipse(10, 10, 120, 100);   }
```

Тут відбувалось малювання прямо на формі (можна було це вказати явно Form1-> Canvas->Ellipse(10, 10, 120, 100);).

Зазначимо, що тільки частина компонентів має властивість Canvas. Якщо треба малювати на компоненті, який не має такої властивості (наприклад, TPanel, TButton та інші), то здавалося б, що немає іншого виходу, ніж із застосовуванням стандартних функцій GDI. Але, виявляється, що за допомогою класу TControlCanvas можна приєднати канву до компонента, у якого її немає, і далі просто малювати так, як завжди.

2.3 Об’єктний клас TCanvas

Canvas зв'язаний з усіма компонентами VCL, у яких є клієнтська частина, а також із класом TBitmap. Стандартні компоненти Windows такі як кнопки, списки й т.д. не мають властивості Canvas, тому що їх повністю малює Windows. Малювання на канві відбувається шляхом виклику відповідних функцій-членів (методів Draw, TextOut, Arc, Rectangle, Ellipse та ін.), а переключення режимів малювання відбувається шляхом модифікації властивостей класу TCanvas – Pen, Font, Brush, TextFlags та інших.

Основні властивості класу TCanvas

Brush – пензель, є об'єктом зі своїм набором властивостей:

Bitmap - картинка розміром строго 8x8,

використовується для заповнення (заливання) області на екрані;

Color - колір заливання;

Style - визначений стиль заливання; ця властивість конкурує з властивістю Bitmap (та властивість з них, яка встановлена останньою, і буде визначати вигляд заливання;

Handle - дана властивість дає можливість використовувати пензель у прямих викликах процедур Windows API .

ClipRect – (тільки читання) прямокутник, на якому відбувається графічний вивід.

CopyMode – властивість визначає, яким чином буде відбуватися копіювання (метод CopyRect) на дану канву зображення з іншого місця: один до одного, з інверсією зображення й ін.

Font – шрифт, яким виводиться текст (методом TextOut).

Handle – використовується для прямих викликів Windows API.

Pen – перо, визначає вид ліній; як і пензель (Brush) є об'єктом з набором властивостей:

Color - колір лінії;

Handle - для прямих викликів Windows API;

Mode - режим виводу: проста лінія, з інвертуванням, з виконанням виключаючого або та інші;

Style - стиль виводу: лінія, пунктир та інші;

Width - ширина лінії в точках;

PenPos - поточна позиція пера, перо рекомендується переміщати за допомогою методу MoveTo, а не прямою установкою даної властивості;

Pixels - двомірний масив елементів зображення (pixel), з його допомогою одержується доступ до кожної окремої точки зображення.

Основні методи класу TCanvas

Методи для малювання найпростішої графіки:

MoveTo – встановлює поточну позицію пера (PenPos);

LineTo – малює пряму до заданої точки;

Rectangle – малює прямокутник із заданою діагоналлю;

Ellipse – малює еліпс, вписаний в заданий діагоналлю прямокутник;

Arc – малює частину кривою еліпсу;

Chord– малює частину кривою еліпсу, з’єднану хордою;

Pie – малює сектор еліпсу;

Polygon –малює замкнуту ламану;

PolyLine –малює незамкнуту ламану;

RoundRect – малює заокруглений прямокутник.

При промальовуванні ліній у цих методах використовуються перо (Pen) канви, а для заповнення внутрішніх областей – пензель (Brush).

Методи для виводу тексту:

TextOut – виводить текстовий рядок (шрифтом (Font) канви);

TextRect – виводить текстовий рядок в прямокутнику (використовується шрифт (Font) канви);

TextHeight – задає його висоту;

TextWidth – задає його ширину.

Методи для виводу картинок на канву:

Draw. Як параметри вказуються прямокутник і графічний об'єкт для виводу (це може бути TBitmap, TIcon або TMetafile);

StretchDraw. Від Draw відрізняється тим, що розтягує або стискає картинку так, щоб вона заповнила весь зазначений прямокутник.

Методи для замальовування областей:

FillRect – замальовує прямокутник кольором і стилем пензля;

FloodFill – замальовує область довільної форми кольором і стилем пензля.

2.4 Подія OnPaint

При малюванні на канві форми або по PaintBox треба враховувати деякі особливості. Якщо вікно якогось іншого додатка перекриває вікно вашого додатка або, якщо малюнок тимчасово згортається, то зображення, намальоване на канві форми, псується. У компоненті Image цього не відбувається, оскільки в класі Tlmage уже передбачені всі необхідні дії, що здійснюють перемальовування зіпсованого зображення. А при малюванні на канві форми або інших віконних компонентів перемальовуванням повинен займатися сам розроблювач додатка. Якщо вікно було перекрито й зображення зіпсувалося, операційна система повідомляє додатку, що в оточенні щось змінилося й що додаток повинен почати відповідні дії. Якщо потрібне відновлення вікна, для нього генерується подія OnPaint.В обробнику цієї події потрібно перемалювати зображення. Перемальовування може відбуватися різними способами залежно від задачі.

Припустимо на формі треба розмістити малюнок з деякого графічного файлу. Це можна зробити, помістивши у секцію private інтерфейсної частини класу TForm1 (файл Unit1.h) рядок:

// вказівник на графічний об’єкт з класу Graphics

```
Graphics:: TBitmap *pBitmap;
```

а у тіло конструктора класу TForm1 (файл Unit1.cpp) – рядок:

```
pBitmap = new Graphics:: TBitmap;
// обробник події OnClick малювання картинки
// (файл Unit1.cpp)
void __fastcall TForm1::FormClick(TObject *Sender)
{
if (OpenPictureDialogl->Execute())
pBitmap->LoadFromFile(OpenPictureDialogl->FileName);
Canvas->Draw(0,0,pBitmap);
}
```

Оскільки графічний об’єкт створився динамічно (оператор new), треба його знищити за допомогою оператора delete. Краще всього це зробити за допомогою деструктора класу (секція public).

Файл Unit1.h:

```
__fastcall ~TForm1 (void);
```

Файл Unit1.cpp:

```
__fastcall ~TForm1::TForm1 (void)
{
delete pBitmap;
}
// обробник події OnPaint (файл Unit1.cpp)
void __fastcall TForm1::FormPaint(TObject *Sender)
{
if (pBitmap != NULL) Canvas->Draw(0,0,pBitmap);
}
```

Наведений вище обробник події OnPaint перемальовує все зображення, хоча, може бути, зіпсована тільки частина його. При великих зображеннях це може значно уповільнювати перемальовування. Перемальовування можна істотно прискорити, якщо перемальовувати тільки зіпсовану область канви.

У канви є властивість ClipRect типу TRect, що у момент обробки події OnPaint указує на область, що підлягає перемальовуванню. Тому більше ефективним буде обробник:

```
void_fastcall TForm1::FormPaint(TObject *Sender)
{
if (pBitmap != NULL)Canvas->CopyRect
(Canvas->ClipRect, pBitmap->Canvas, Canvas->ClipRect);
}
```

Він перемальовує тільки прямокутну область ClipRect, яка зіпсована.

Розділ 3 РЕАЛІЗАЦІЯ ПРОГРАМИ

3.1 Вибір мови програмування

Нам давно відомо про стрімкий розвиток інформаційних технологій. Такі темпи розвитку дійсно вражають а іноді й дивують. Не зважаючи на це в світі комп'ютерних технологій залишається одна дуже важлива область, зміни в якій відбуваються повільно – це програмування, кодування, створення вихідних текстів, кодів – це ключовий елемент в створені будь-якого програмного засобу. Сьогодні все відбувається так, як і раніше: розробник використовує обмежений набір логічних конструкцій, невелику кількість стандартних типів даних і деякі нові та сучасні алгоритми та технології. Причому такий підхід зовсім не змінився, хоча змінилося вже не одне покоління мов програмування. Наприклад, на зміну C та Pascal прийшла Java, C++, C#, та інші, але все ж таки багато програмістів досі підлагоджують складні програми, вручну проглядаючи протоколи роботи та ігноруючи зручні та комфортні засоби швидкої візуальної розробки.

Комп'ютерні видання, що претендують на звання професійних, нерідко пропагандують подібний підхід до створення програм. Складається своєрідний імідж програміста – одиночки здатного за пару безсонних ночей написати потрібну програму, яка героїчно поміститься в ста кілобайтах пам'яті. Програмуванню взагалі притаманний значний консерватизм, так як в принципі можна складати програми, обмежуючись знаннями багаторічної давнини. Але такі знання на сьогодні відходять на другий план. Сьогодні програмування безумовно перетворилося із мистецтва в ремесло. Звичайно, зараз навряд чи можна стати професіональним розробником, не вивчивши внутрішній лад Windows чи структуру компонентів VCL і принципи оптимізації програм. Роботодавців цікавить в першу чергу швидкість і властивість створення програм в колективі, а такі можливості може забезпечити лише середовище візуальної розробки, яке здатне взяти на себе значні об'єми рутинної роботи по підготовці та розробці прикладних програм, а також погодити діяльність групи постановників, кодувальників, тестерів та технічних письменників.

Мови програмування C/C++ є одними із найстаріших і що саме головне вони за свою історію розвитку стали чи не найбільш розвиненими та використовуваними. Зараз можна налічити десятки модифікацій цих мов, а також далеко не одна сучасна мова програмування взяла свої основи і не тільки основи з мов програмування C/C++, так наприклад була створена мова програмування Java. Тому я обираю мову C/C++. Також вивчивши цю мову можна із значною легкістю вивчити деякі модифікації цієї мови або ті мови які щось взяли для себе з мови C/C++, наприклад та ж мова Java або C#.

Враховуючи в загальному вимоги до сучасного програмного забезпечення яке використовується і створюється у наш час і вимоги до програмного забезпечення яке потрібно створити для візуалізації даних по землетрусах, одним з найкращих варіантів буде вибір візуального середовища розробки Borland C++ Builder 6.

На сьогоднішній день створення консольних програм які потім будуть використовуватись це одиничні випадки, консольні програми були актуальними багато років тому, а на сьогодні вони вже нажаль не перспективні, їх можна використовувати лише в навчальному процесі, а для реального використання вони вже не актуальні і крім того будуть незвичними, незрозумілими і незручними для користувачів які не пов’язані з програмуванням і нічого про нього не знають. У сучасному світі вже давно використовуються програми з графічним інтерфейсом які значно спрощують діалог програми з користувачем і як було сказано вище про бібліотеку візуальних компонентів VCL – вона значно спрощує розробку програмних засобів, завдяки готовим компонентам можна уникнути великого об’єму рутинної роботи при створенні програмного забезпечення. А обране середовище Borland C++ Builder 6 як раз дає змогу створювати програми з графічними інтерфейсами використовуючи бібліотеку VCL та багато інших можливостей.

3.2 Постановка задачі

Створити програмне забезпечення для візуалізації сейсмічних даних. Дані можуть бути по прогнозу землетрусів або по землетрусах які відбулися.

Програма повинна виконувати візуалізацію цих даних у 5 варіантах:

- Побудова 3D графіка по силах землетрусу у місцевості в якій відбувався або прогнозується землетрус розподіливши місцевість на 100 рівних частин, таким чином 1 частина рівна 1% місцевості;

- Побудова 3D діаграми по силах землетрусу у місцевості в якій відбувався або прогнозується землетрус теж розподіливши місцевість на 100 рівних частин;

- Відображення проходження хвиль землетрусу у місцевості;

- Шляхом поділу місцевості на ділянки, на 100 рівних частин, таким чином щоб 1 ділянка була рівна 1% місцевості визначити у яких ділянках відбувається землетрус, показати числами загальну суму сили хвиль землетрусу у цих ділянках і зафарбувати ділянки відповідним кольором залежно від сили хвиль землетрусу у ділянці;

- Візуалізація інформації про землетруси на картах пошукової системи Google, для цього необхідно завантажити карту з сайту пошукової системи Google попередньо задавши місцевість координатами Гринвіча і вказавши масштаб збільшення, або вставити в програму URL адресу карти і завантажити карту, після цього розподілити місцевість на 100 рівних частин та показати точками або зафарбувати ділянки у яких відбувається землетрус відповідним кольором залежно від сили землетрусу, показати числами суму сили хвиль землетрусу у цих ділянках, якщо у ділянках немає землетрусу то вивести нуль;

3.3 Опис алгоритмів

Вхідні дані:

Вхідними даними є матриця розмірністю 1000x1000 елементів у якій містяться цілі числа які характеризують силу хвиль землетрусу у місцевості і напрям проходження хвиль. Дані подаються у текстовому файлі. Вибравши відповідну команду меню у програмі цей текстовий файл відкривається і у програму записується шлях до цього файлу для подальшої роботи з файлом і обробки даних які містяться у файлі.

Вихідні дані:

Вихідними даними є візуалізація даних яку виконує програма в залежності від вибраного виду візуалізації вибраного в меню програми. Також, всі види візуалізації програми можна зберігати у графічні файли і використовувати ці файли в подальшому.

Алгоритм роботи програми

У програмі виконується візуальне представлення даних які задаються матрицею розмірністю 1000x1000 елементів і подаються у текстовому файлі.

На початку роботи програми потрібно запустити виконуваний файл програми і за допомогою відповідної команди меню завантажити текстовий файл з даними, у програму запишеться шлях до цього файлу та ім’я файлу у змінну.

Після завантаження файлу з даними і запису шляху та імені файлу у змінну необхідно вибрати потрібний вид або декілька видів візуалізації даних, після чого відкриється нове вікно програми яке відповідає вибраному виду візуалізації, у це вікно програми автоматично передасться ім’я та шлях до файлу з даними який був вибраний при запуску програми і матриця яка міститься у текстовому файлі зчитається з файлу, у новому вікні яке відкрилось потрібно буде вибрати пункт «Побудувати», після цього у вікні виконається вибраний вид візуалізації даних, результат візуалізації буде відображатись у вікні програми і цей результат можна буде зберегти у графічний файл.

Після виконання необхідних видів візуалізації можна повторити їх або завантажити наступний файл з даними або завершити роботу закривши програму.

Алгоритм роботи:

- Запуск програми;

- Завантаження файлу з даними і запис шляху до файлу та імені файлу у змінну;

- Вибір потрібного виду візуалізації у пункті меню або на панелі інструментів;

- Відкриття нового вікна при виборі виду візуалізації та передання до нього шляху до файлу з даними та імені файлу записаного у змінну;

- При візуалізації на картах пошукової системи Google, завантаження карти з сайту пошукової системи Google по заданих координатах Гринвіча або вставити у програму свою URL адресу карти і завантажити карту з сайту Google та вказати масштаб;

- Динамічне виділення пам’яті для матриці 1000x1000 елементів

- Зчитування матриці з файлу;

- Обробка даних матриці, в деяких пунктах розбиття на ділянки, обчислення сум сил хвиль землетрусу по ділянках

- Виконання візуалізації даних;

- Звільнення пам’яті яка була виділена динамічно;

- Збереження результату візуалізації у графічний файл або очищення вікна результатів візуалізації

- Завершення роботи програми

3.4 Компоненти Borland C++ Builder 6 використані в програмі

Програма розроблена у середовищі візуальної розробки Borland C++ Builder 6 у графічному варіанті з використанням вікон та компонентів бібліотеки VCL.

Form – форми, вікна програми, може бути одне або декілька;

MainMenu – компонент який використовується для створення головного меню програми;

OpenDialog – використовується для відкриття діалогу вибору файлів які потрібно завантажити у програму;

Image – використовується для відображення графічних файлів різного формату;

Panel – панель, використовується для поміщення на неї інших компонентів, зазвичай кнопок, полів для вводу даних і т. д.;

Button – використовується для створення кнопок у яких описується обробник подій, описані події виконуються при натисненні на кнопку;

Chart – використовується для побудови графіків та діаграм;

SavePictureDialog – використовується для збереження графічних файлів;

CppWebBrowser – використовується для завантаження веб сторінок;

Label – використовується для напису тексту;

Edit – використовується для створення полів через які будуть передаватися дані в програму або для виведення інформації в це поле;

ScrollBox – використовується для прокрутки зображень та інших даних які не помістились у вікні.

3.5 Опис роботи програми

При запуску програми з’являється вікно програми у якому розташоване меню програми що складається з трьох вкладок:

- Файл;
- Візуалізація даних;
- Про програму;

та панель інструментів яка складається з п’яти кнопок:

- Побудувати графік;
- Побудувати діаграму;
- Відображення хвиль;
- Відображення ділянок;
-Візуалізація на карті.

Працювати з програмою і виконувати обраний вид візуалізації можна за допомогою меню або панелі інструментів яка міститься у головному вікні програми.

Вкладка меню «Про програму» не виконує ніяких видів візуалізації, в ній міститься інформація про програму і про те для чого вона призначена. Всі види візуалізації даних, ті ж самі що на панелі інструментів, містяться у вкладці – «Візуалізація даних».

Вкладка «Файл» використовується для завантаження файлу з даними.

На Рис 2. зображене головне вікно програми яке з'являється при запуску програми.

Після запуску програми необхідно вибрати текстовий файл з даними за допомогою пункту меню Файл->Відкрити файл. І завантажити його у програму, для завантаження файлу відкриється діалогове вікно для вибору і завантаження файлу як зображено на Рис 3.

Рис 2. Головне вікно програми

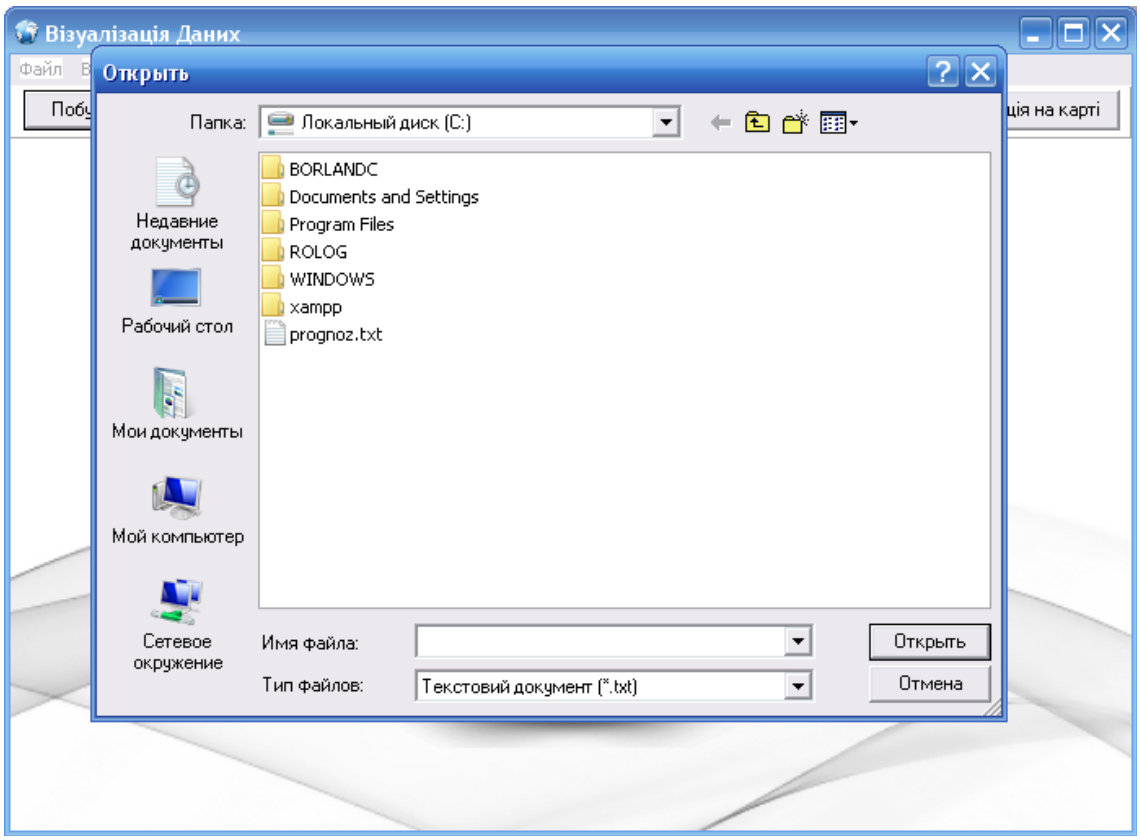

Рис 3. Відкриття і завантаження файлу у програму

Після того як обраний файл завантажено у програму, можна виконувати візуалізацію даних. Перший вид візуалізації - графік, для цього потрібно натиснути кнопку «Побудувати графік» на головному вікні програми або вибрати пункт меню Візуалізація даних-> Побудувати графік, після цього відкриється окреме вікно в якому будуть розміщені кнопки:

- Побудувати
- Очистити
- Зберегти у графічний файл

Якщо натиснути кнопку «Побудувати» у вікні програми побудується графік. Результат роботи програми показано на Рис 4.

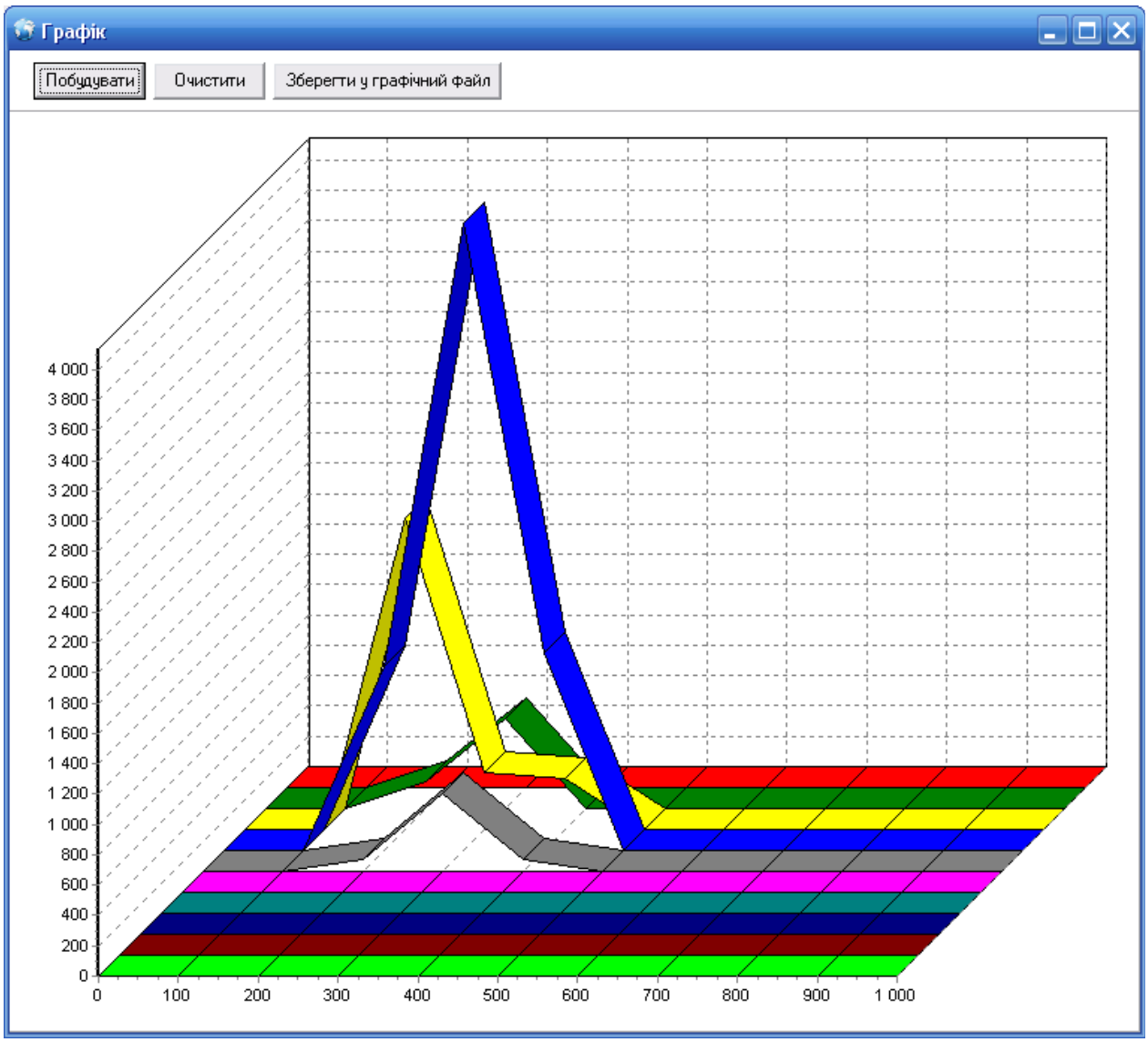

Рис 4. Побудова графіка

Результат візуалізації даних можна зберігати у графічний файл, якщо натиснути кнопку «Зберегти у графічний файл» то відкриється діалогове вікно для збереження файлу, в якому потрібно буде вказати імя файлу, розширення і вибрати місце для збереження файлу, як показано на Рис 5

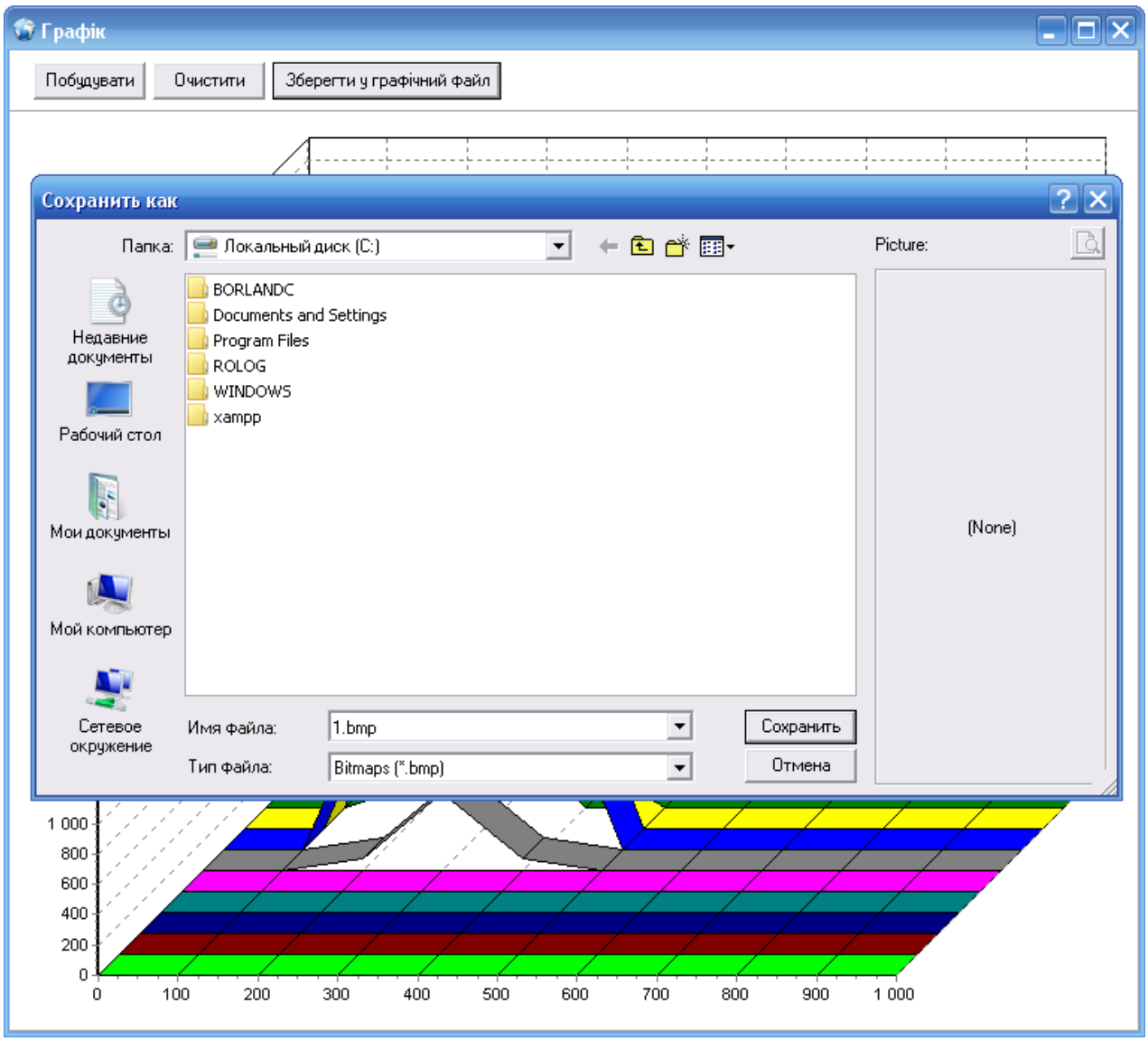

Рис 5. Відкриття діалогового вікна для збереження файлу

Після натиснення на кнопку «Сохранить» у діалоговому вікні, файл збережеться у вказаному місці з вказаним іменем при збереженні.

Результат збереженого графічного файлу у форматі BMP показаний на Рис 6.

Приблизно так само працюють і всі інші види візуалізації, користувачеві необхідно лише обрати потрібний вид візуалізації, виконати візуалізацію і при потребі зберегти у файл результати візуалізації.

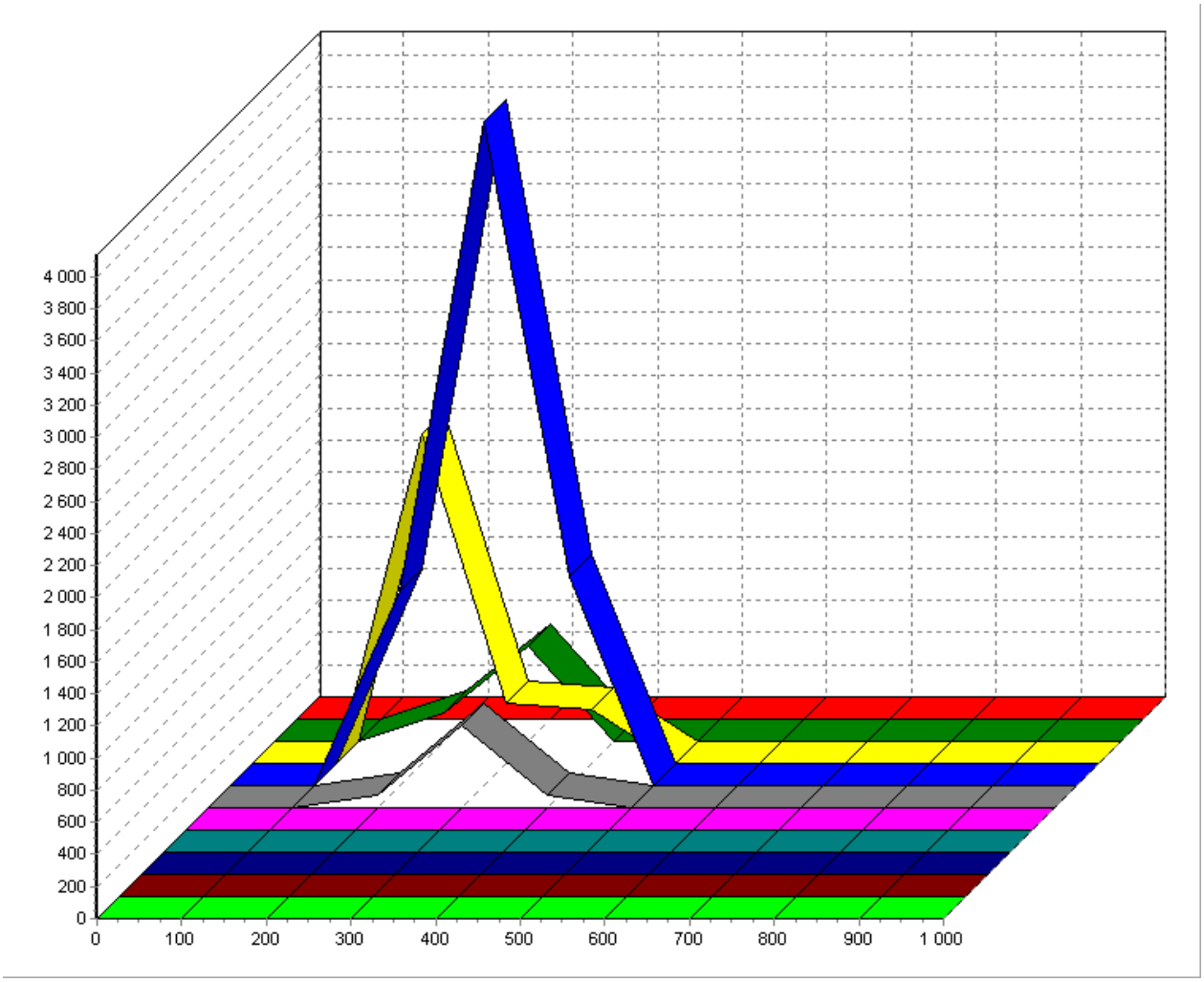

Рис 6. Збережений графічний файл

При натисненні на кнопку «Очистити» що показана на Рис 4, вікно з графіком очиститься.

Розглянемо решту видів візуалізації які виконуються в програмі.

Наступний вид візуалізації – діаграма. У нашій програмі діаграма будується по схожій схемі до побудови графіка, тільки діаграма дає менш точну візуалізацію ніж графік тому що вищі стовбці діаграми закривають собою нижчі стовбці, а в графіку стовбців немає, там все візуалізується у вигляді ліній які схожі на хвилі.

Для побудови діаграми потрібно натиснути кнопку «Побудувати діаграму» або вибрати пункт меню Візуалізація даних-> Побудувати діаграму. Результат побудови діаграми зображено на Рис 7. У вікні зображені ті ж кнопки що і в попередньому варіанті візуалізації, вони виконують ті ж функції тільки тут будується діаграма.

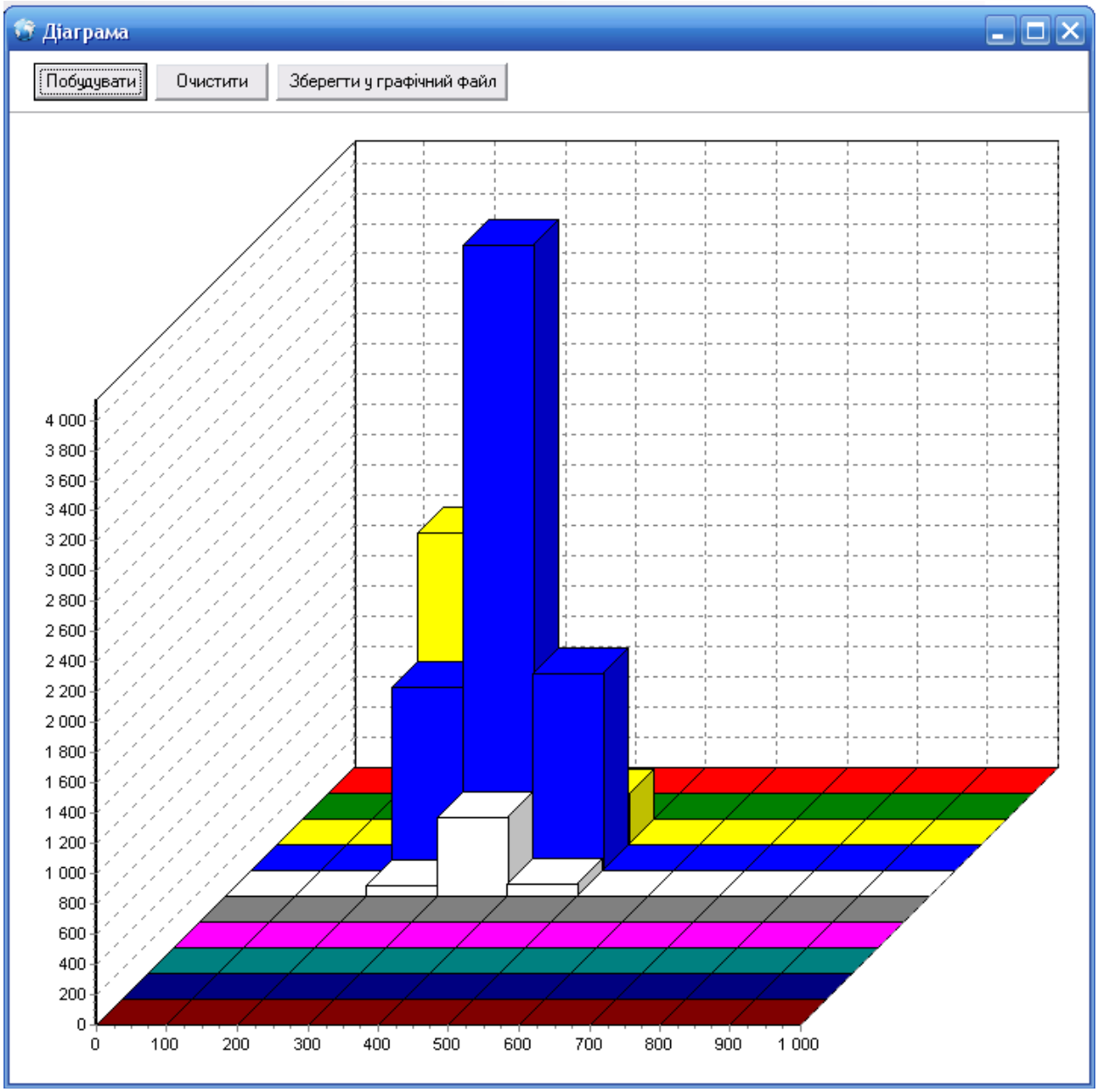

Рис 7. Побудова діаграми

При потребі результат візуалізації можна зберегти у графічний файл. При порівнянні цих двох зображень видно що у діаграмі за вищими стовбцями може бути не видно нижчих.

Наступний вид візуалізації – Відображення хвиль, цей спосіб відображає проходження хвиль у місцевості. Виконується командою меню Візуалізація даних-> Відображення хвиль, або натиснувши у головному вікні програми кнопку «Відображення хвиль». Колір хвиль зафарбовується залежно від сили хвиль землетрусу, використовується перехід від чорного до червоного та кольору, чим яскравіший червоний колір тим сильніша хвиля землетрусу.

Чорний колір означає що землетрусів немає. Результат відображення хвиль землетрусу показано на Рис 8.

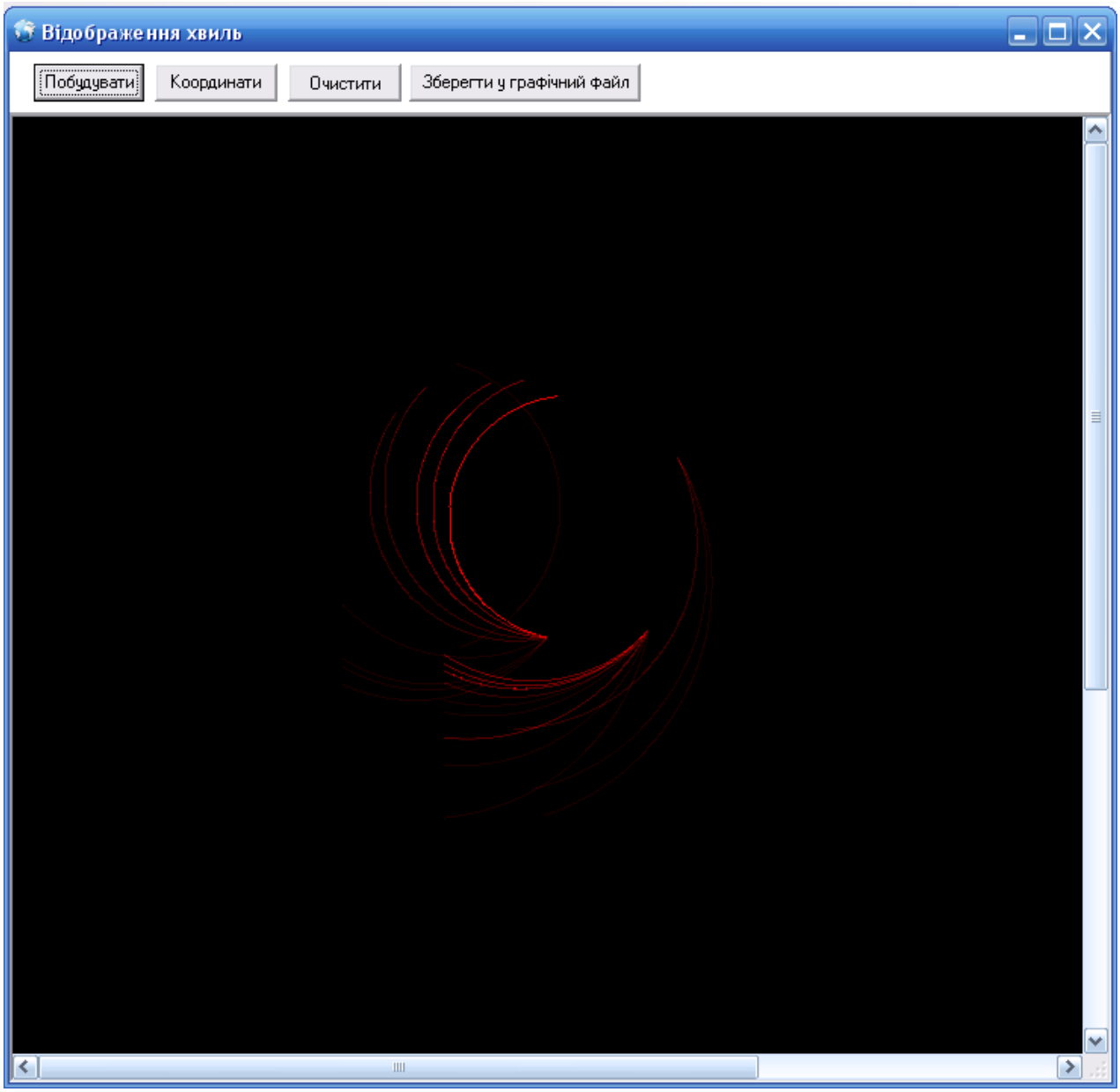

Рис 8. Відображення хвиль землетрусу

Якщо натиснути на кнопку координати то вікно розграфиться у сітку на 100 частин, таким чином що 1 клітинка це 1% місцевості, і приблизно буде видно в яку сторону під яким кутом рухаються хвилі землетрусу і порахувати скільки відсотків території підвернені землетрусу.

Наступний вид візуалізації – Відображення ділянок, цей вид візуалізації показує небезпечні ділянки в місцевості, ділянки зафарбовуються залежно від сили землетрусу, теж використовується перехід від чорного до червоного кольору.

Щоб виконати візуалізацію потрібно виконати команду меню Візуалізація даних-> Відображення ділянок або у головному вікні програми натиснути кнопку «Відображення ділянок». Результат відображення ділянок показаний на Рис 9.

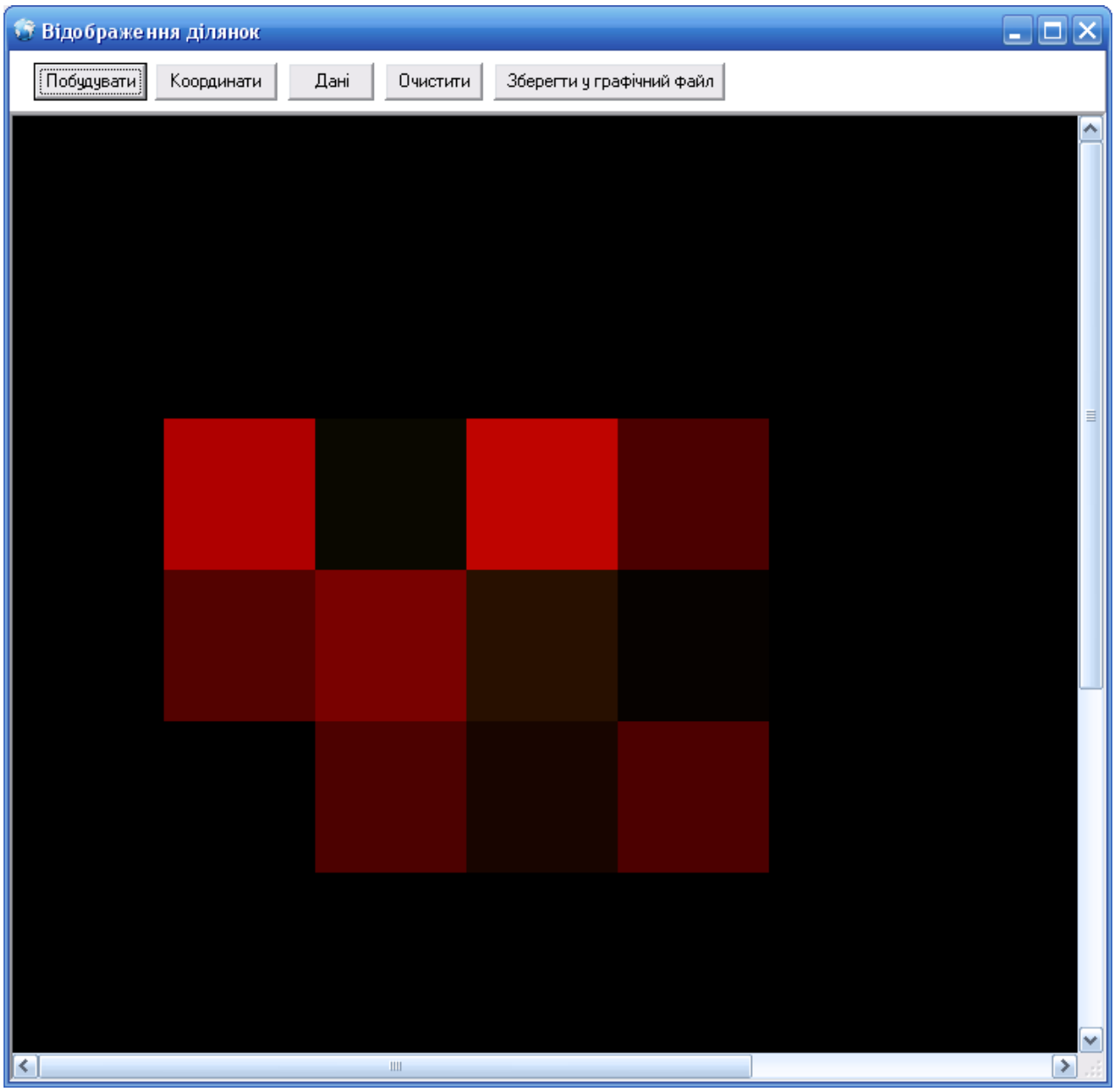

Рис 9. Відображення ділянок

Якщо натиснути на кнопку «Дані» то у квадратах які зафарбовані відтінками червоного та коричневого кольору, відобразяться числа які будуть означати силу хвиль землетрусу у ділянках, тут так само є кнопка «Координати», за допомогою неї можна поділили місцевість на 100 частин, таким чином що 1 частина це 1% місцевості. На Рис 10 показане вікно у якому місцевість ділиться на 100 частин і виводяться сили хвиль землетрусу у ділянках.

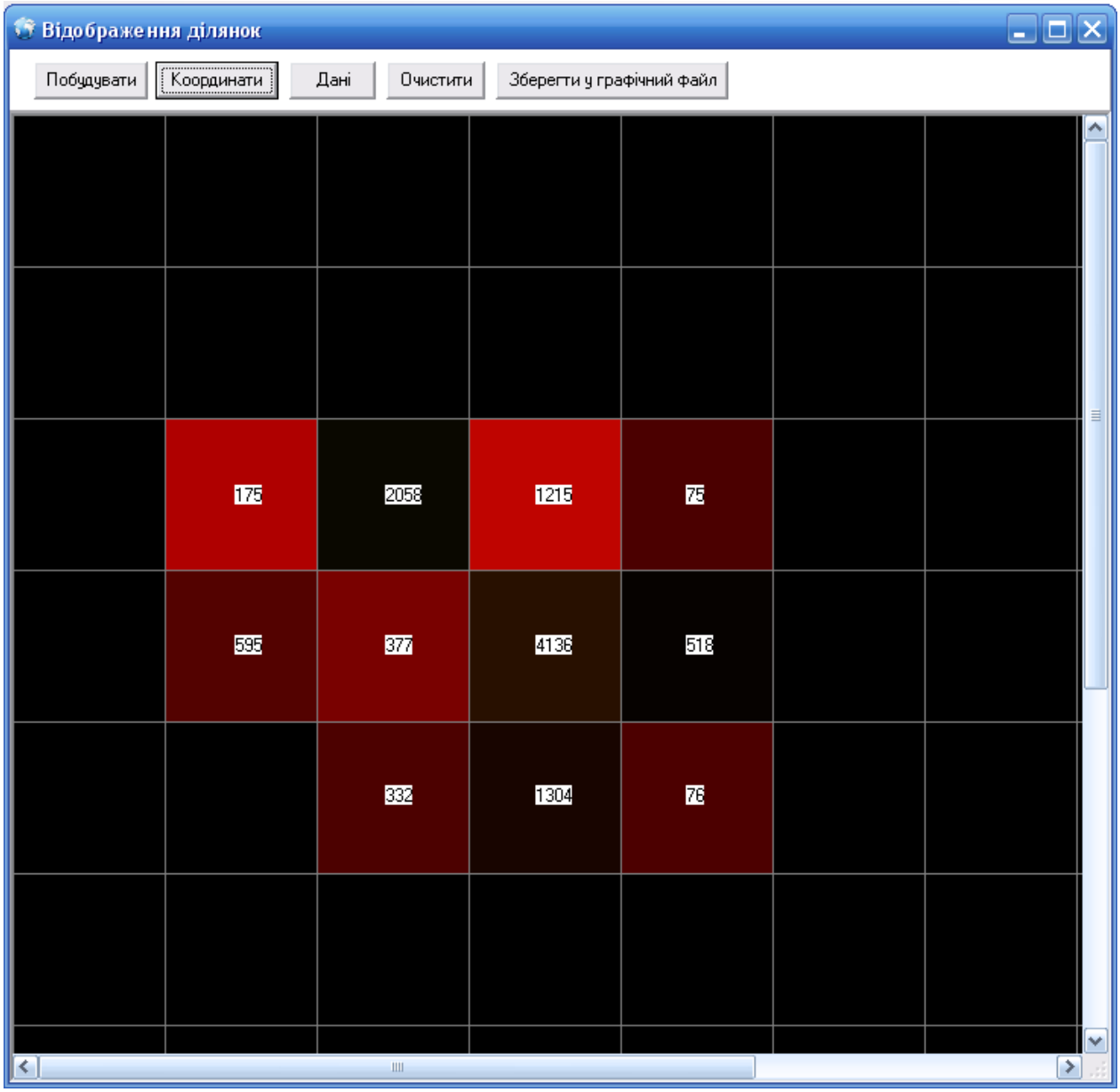

Рис 10. Відображення ділянок з даними і поділом місцевості

Останній вид візуалізації який виконується у програмі – Візуалізація на карті. Для використання цього виду візуалізації необхідно мати підключення до мережі Інтернет.

Для виконання візуалізації потрібно виконати команду меню Візуалізація даних-> Візуалізація на карті або у головному вікні програми натиснути кнопку «Візуалізація на карті».

При відкритті вікна візуалізації на карті необхідно завантажити карту з сайту пошукової системи Google, задавши у програму координати Гринвіча за якими буде задано яку карту завантажувати і натиснути кнопку «Сформувати URL», або вставити у програму URL адресу карти яку потрібно завантажити. Після цього натиснути на кнопку «Завантажити карту», якщо завантаження відбулось, буде показане повідомлення про завантаження карти на якому потрібно натиснути кнопку «ОК» як показано на Рис 11.

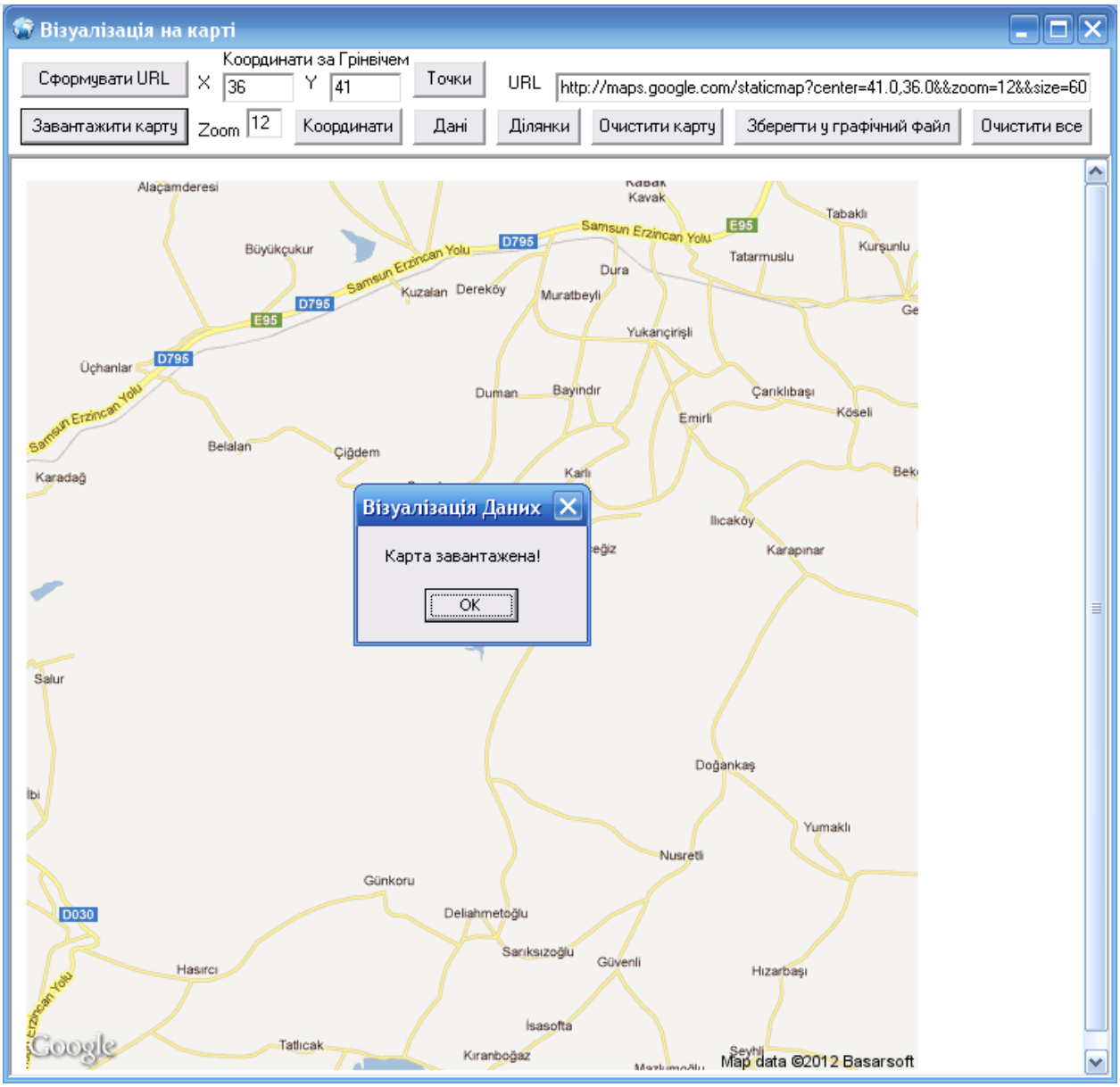

Рис 11. Завантаження карти

Після того як карта завантажена, можна виконувати візуалізацію даних і відображати ці дані на карті.

Керуючи кнопками «Точки», «Дані», «Ділянки», «Координати», можна виконувати відображення даних у різних варіантах, можна розграфити місцевість на 100 частин і вивести дані в ділянках або точками показати ділянки землетрусу і над точками вивести числа які відображають силу землетрусу у ділянках, або зафарбувати ділянки у яких є землетрус відповідним кольором залежно від землетрусу і показати числами силу землетрусу у ділянках. Результат візуалізації показано на рис 12.

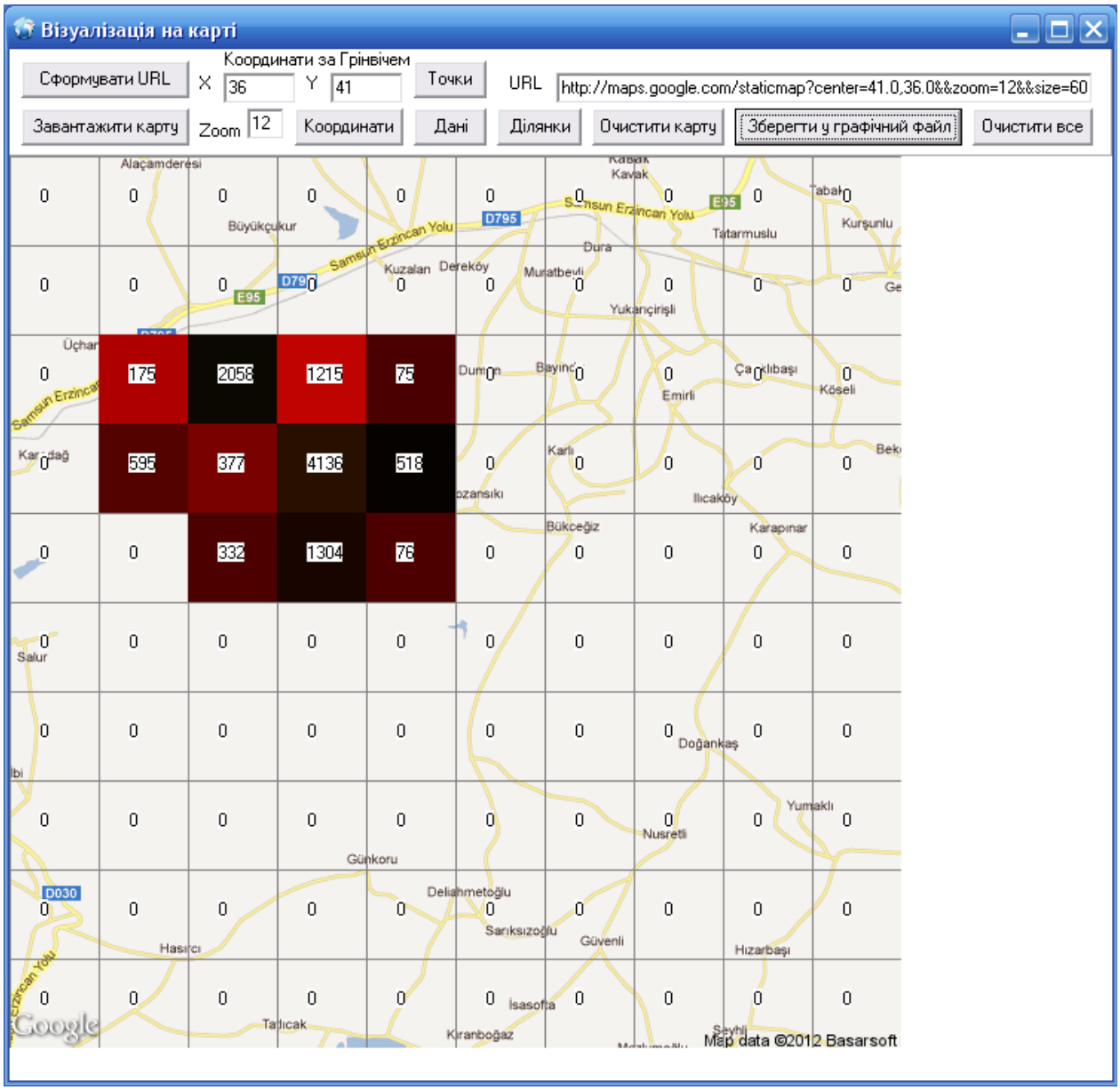

Рис 12. Візуалізація даних на карті

У тих ділянках де виведено число нуль, означає що землетрусу у цих ділянках немає, хвилі землетрусу там не проходять.

Так само можна показати ділянки точками не зафарбовуючи ділянки повністю і вивести числа які відображають сили землетрусу у ділянках, як показано на Рис 13. Іноді це більш зручно ніж зафарбовувати ділянку повністю.

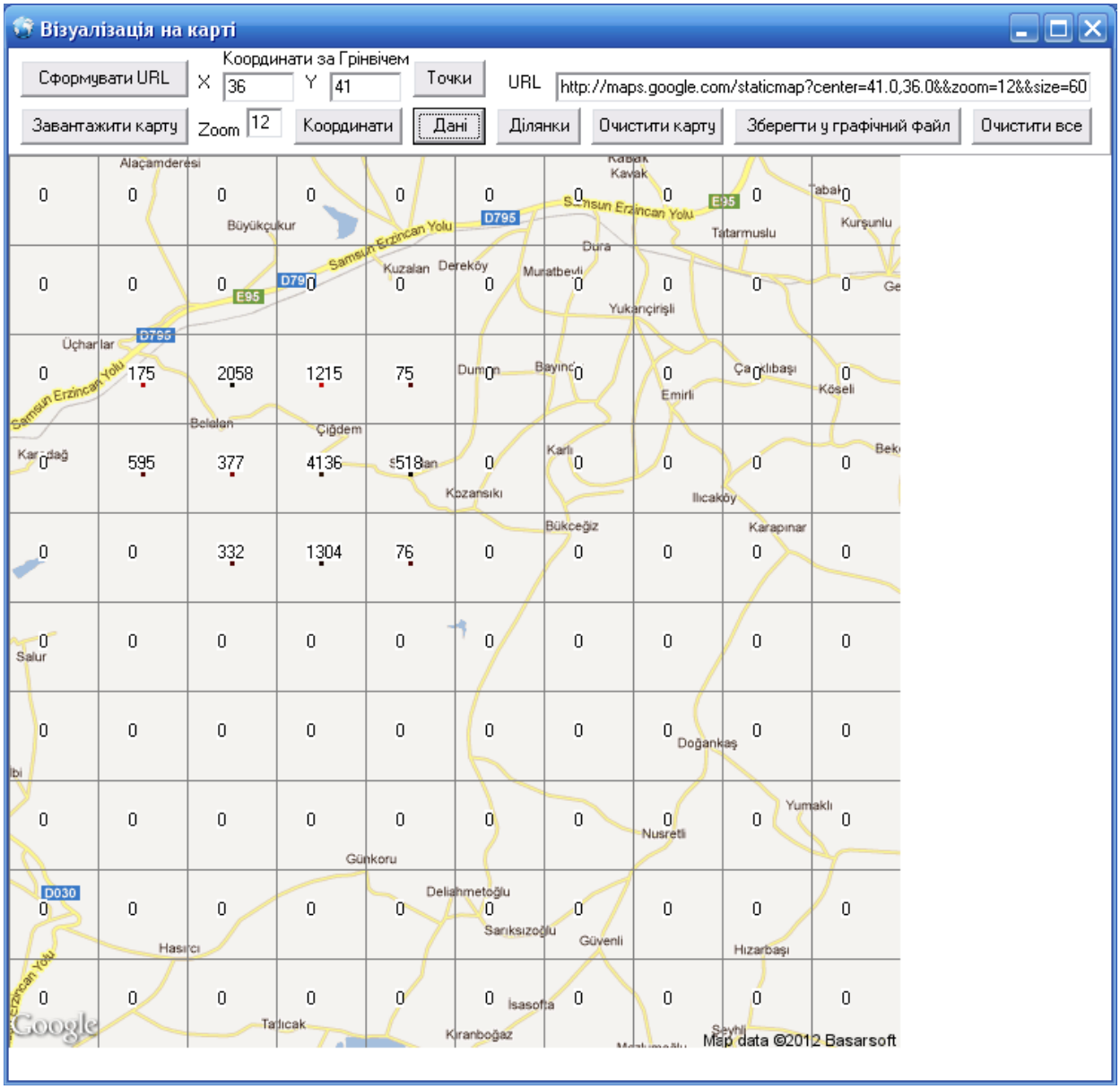

Рис 13. Візуалізація даних на карті з точками в ділянках

3.6. Інструкція користувача

Запуск програми відбувається з виконуваного файлу програми.

Після запуску програми з’являється головне вікно програми, за допомогою меню потрібно завантажити текстовий файл з даними і обирати потрібні види візуалізації даних по черзі за допомогою меню або у головному вікні програми. Кожна наступна дія буде відбуватись при виборі виду візуалізації та виконанні обраної візуалізації.

Для виходу з програми потрібно закрити програму.

3.7. Технічні вимоги

1. Персональний комп’ютер з мінімальними вимогами: Pentium 166 MHz, 64 Мбайт ОЗУ, відеоадаптер, що підтримує частоту екрану 85 Гц і 32-бітну глибину кольору, 14-15-дюймовий монітор, обсяг вільного місця на жорсткому диску 2 Мб.

2. Встановлена ОС Windows XP/ Windows 7/Vista.

Висновки

Під час виконання наукової роботи на тему „Сучасні системі візуалізації даних”, ознайомився з сучасними технологіями розробки та сучасними вимогами до програмного забезпечення, також навчився працювати з графікою в середовищі Borland C++ Builder 6 та створювати графічні додатки, вивчив основи візуального програмування, ознайомився з бібліотекою візуальних компонентів яка значно спрощує і полегшує розробку програмного забезпечення, використав деякі компоненти у своїй програмі та навчився працювати з ними, застосував раніше здобуті знання, покращив вміння працювати у середовищі Borland C++ Builder 6 та набув нових навиків роботи у сфері створення та розробки програмного забезпечення.

У науковій роботі створено програмний засіб для візуального представлення даних. За допомогою даного програмного забезпечення по сейсмічних даних можна будувати 3D графік, 3Dдіаграму, відображати проходження хвиль землетрусу у місцевості, небезпечні ділянки в місцевості та загальну силу хвиль в цих ділянках, виконувати візуалізацію на карті точками в яких ділянках є землетруси або зафарбовувати ділянки відповідним кольором в залежності від сили землетрусу, також показувати загальну силу хвиль в цих ділянках на карті, по відповідних представлених даних можна робити висновки про місцевість для якої виконувалось дослідження і візуалізація даних, які ділянки є найбільш небезпечними, які менш небезпечні, в яких є землетрус і в яких його немає. По результатах роботи програми можна отримати загальну характеристику місцевості у вигляді діаграми, графіка, візуалізації небезпечних ділянок, або більш точну у вигляді проходження хвиль землетрусу чи відображення небезпечних ділянок на карті в місцевості яка досліджується, результати візуалізації можна зберігати у графічні файли і використовувати їх у подальшому для дослідження місцевості та додавати ці файли до загальної інформації про досліджувану місцевість.

Література

1. Архангельский А.Я. Программирование в C++Builder 6.– М.: ЗАО ”Издательство БИНОМ”, 2002.–1152с.

2. Архангельский А.Я. C++ Builder 6. Справочное пособие. Книга 1. Язык C++. – М.: Бином-Пресс, 2004 г.– 544 с.

3. Мєтт Теллес. Borland C++Builder: библиотека программиста. – СПб: ПитерКом,1998. – 512с.

4. Кент Рейсдорф, Кен Хендерсон. Borland C++Builder: Освой самостоятельно. – Москва: Бином, 1998.–702с.

5. Буч Г. Объектно-ориентированное проектирование с примерами применения.– М.: Конкорд, 1992. – 367 с.

6. Шамис В.А. C++Bulder 4. Техника визуального программирования. Издание второе, переработанное и дополненное.– М.: Нолидж, 2000. – 656 с.

7. Шамис В. C++ Builder Borland Developer Studio 2006. – СПб: Издательство "Питер", 2007. – 784 с.

8. Алексанкин В. Г., Елманова Н. З. Среда разработки C++ Builder. – СПб: Издательство "Питер", 1999. – 312 с.

9. Тимофеев В.В. Язык C/C++. Программирование в C++Builder 5. – Москва: Бином, 2000. – 368 с.

10. Кошель С.П., Елманова Н.З. Введение в Borland C++ Builder. – М.: Диалог-МИФИ, 1997.– 252 с.

11. Страуструп Б. Язык программирования C++. – К.: ДиаСофт, 1993. – 256 с.

12. Страуструп Б. Язык программирования C++. – СПб.: ”Невский Диалект”, 2002.– 1099 с.

13. Страуструп Б. Дизайн и эволюция C++. – СПб: Издательство "Питер", 2006.– 448с.

14. Скотт Мейерс. Эффективное использование STL. Библиотека программиста . – СПб: Издательство "Питер", 2003. – 400 с.

15. Мэтью Г. Остерн. Обобщенное программирование и STL.Использование и наращивание стандартной библиотеки шаблонов C++. – СПб.: Невский Диалект, 2004. – 544 с.

Електронні ресурси

16. Программирование / C Builder ученик [Електронний ресурс]: Доступ до публікації:

http://www.ru-coding.com/c_1.php

17. Основы C++Builder [Електронний ресурс]: Доступ до публікації:

http://www.williamspublishing.com/PDF/5-8459-0499-4/part.pdf

18. Технологии C++Builder. Разработка приложений для бизнеса. Учебный курс [Електронний ресурс]: Бобровский С. И. Питер, 2007 г. 560 стр. Доступ до підручника:

http://www.piter.com/book.phtml?978591180213

19. Объектно-ориентированное программирование на языке C++. Лабораторный практикум [Електронний ресурс]: Ноткин А.М. Пермь: Перм. гос. техн. ун-т, 2001. - 92 с. Доступ до підручника:

http://window.edu.ru/window_catalog/files/r47642/pstu004.pdf

20. Публикации Бьерна Страуструпа. 2002-2012 г. [Електронний ресурс]: Доступ до публікацій:

http://www.research.att.com/~bs/papers.html

Додатки

Додаток А

Код програми

Головне вікно програми

```
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
#include "Unit2.h"
#include "Unit3.h"
#include "Unit4.h"
#include "Unit5.h"
#include "Unit6.h"
#include "Unit7.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
Form2->Show();
}
//---------------------------------------------------------------------------
```

```
void __fastcall TForm1::Button2Click(TObject *Sender)
{
Form3->Show();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button3Click(TObject *Sender)
{
  Form4->Show();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button4Click(TObject *Sender)
{
  Form5->Show();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button5Click(TObject *Sender)
{
  Form6->Show();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::N10Click(TObject *Sender)
{
AboutBox->Show();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::N7Click(TObject *Sender)
{
Form1->Close();
}
//---------------------------------------------------------------------------
```

```
  AnsiString path;
    char *filename=(char*) malloc(50);
void __fastcall TForm1::N12Click(TObject *Sender)
{
 OpenDialog1->Filter="Текстовий документ (*.txt)|*.txt";
 if(OpenDialog1->Execute())
 {
  path=OpenDialog1->FileName;
 }
  filename=path.c_str();
}
 //---------------------------------------------------------------------------
```

Вікно побудови графіка

```
#include <vcl.h>
#pragma hdrstop
#include <fstream.h>
#include <iso646.h>
#include "Unit1.h"
#include "Unit2.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm2 *Form2;
extern char *filename;
//---------------------------------------------------------------------------
__fastcall TForm2::TForm2(TComponent* Owner)
      : TForm(Owner)
{
}
```

```
//---------------------------------------------------------------------------
void __fastcall TForm2::Button1Click(TObject *Sender)
{
  int **A=new int* [1000];

 for(int col=0;col<1000;col++)
 { A[col]=new int [1000]; }
 ifstream ifs (filename,ifstream::in);
 {
 for (int i=0;i<1000;i++)
  {
   for (int j=0;j<1000;j++)
   {
     ifs>>A[i][j];
    }
   }
ifs.close();
 }
        Series1->Clear();
        Series2->Clear();
        Series3->Clear();
        Series4->Clear();
        Series5->Clear();
        Series6->Clear();
        Series7->Clear();
        Series8->Clear();
        Series9->Clear();
        Series10->Clear();

         int sum=0,r=0,k=0;
```

```
   for (int i=0;i<=1000;i+=100)
  {
    for(int j=0;j<=1000;j+=100)
      {
          sum=0;
if(j<=900 and i<=900)
  { for(r=i;r<i+100;r++)
        {
         for(k=j;k<j+100;k++)
           {
            sum=sum+A[r][k];
            }
           }
          }

         if(j==0){Series1->AddXY(i,sum);}
         if(j==100){Series2->AddXY(i,sum);}
         if(j==200){Series3->AddXY(i,sum);}
         if(j==300){Series4->AddXY(i,sum);}
         if(j==400){Series5->AddXY(i,sum);}
         if(j==500){Series6->AddXY(i,sum);}
         if(j==600){Series7->AddXY(i,sum);}
         if(j==700){Series8->AddXY(i,sum);}
         if(j==800){Series9->AddXY(i,sum);}
         if(j==900){Series10->AddXY(i,sum);}

    }
 }
```

```
   for (int i=0; i<1000; i++)
  { delete [] A[i]; }
   delete []A;
}
//---------------------------------------------------------------------------
void __fastcall TForm2::Button2Click(TObject *Sender)
{
  Series1->Clear();
  Series2->Clear();
  Series3->Clear();
  Series4->Clear();
  Series5->Clear();
  Series6->Clear();
  Series7->Clear();
  Series8->Clear();
  Series9->Clear();
  Series10->Clear();
}
//---------------------------------------------------------------------------
void __fastcall TForm2::Button3Click(TObject *Sender)
{
SavePictureDialog1->InitialDir;
  SavePictureDialog1->FileName="C:\\1.bmp";
 if (SavePictureDialog1->Execute())
 {
    Chart1->SaveToBitmapFile(SavePictureDialog1->FileName);
 }
}
//---------------------------------------------------------------------------
```

Вікно побудови діаграми

```
#include <vcl.h>
#pragma hdrstop
#include <fstream.h>
#include <iso646.h>
#include "Unit1.h"
#include "Unit3.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm3 *Form3;
extern char *filename;
//---------------------------------------------------------------------------
__fastcall TForm3::TForm3(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button1Click(TObject *Sender)
{
int **A=new int* [1000];
for(int col=0;col<1000;col++)
{ A[col]=new int [1000]; }
 ifstream ifs (filename,ifstream::in );
 {
 for (int i=0;i<1000;i++)
  {
   for (int j=0;j<1000;j++)
   {
    ifs>>A[i][j];
```

```
    }
   }
ifs.close();
 }

 int sum=0,r=0,k=0;
        Series1->Clear();
        Series2->Clear();
        Series3->Clear();
        Series4->Clear();
        Series5->Clear();
        Series6->Clear();
        Series7->Clear();
        Series8->Clear();
        Series9->Clear();
        Series10->Clear();

      for (int i=0;i<=1000;i+=100)
        {
          for(int j=0;j<=1000;j+=100)
            {
                sum=0;
      if(j<=900 and i<=900)
        { for(r=i;r<i+100;r++)
              {
               for(k=j;k<j+100;k++)
                 {
                  sum=sum+A[r][k];
                  }
                 }
```

```
            }
            if(j==0){Series1->AddXY(i,sum);}
            if(j==100){Series2->AddXY(i,sum);}
            if(j==200){Series3->AddXY(i,sum);}
            if(j==300){Series4->AddXY(i,sum);}
            if(j==400){Series5->AddXY(i,sum);}
            if(j==500){Series6->AddXY(i,sum);}
            if(j==600){Series7->AddXY(i,sum);}
            if(j==700){Series8->AddXY(i,sum);}
            if(j==800){Series9->AddXY(i,sum);}
            if(j==900){Series10->AddXY(i,sum);}

          }
        }

  for (int i=0; i<1000; i++)
  { delete [] A[i]; }
  delete []A;

}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button2Click(TObject *Sender)
{
 Series1->Clear();
 Series2->Clear();
 Series3->Clear();
 Series4->Clear();
 Series5->Clear();
 Series6->Clear();
 Series7->Clear();
```

```
 Series8->Clear();
 Series9->Clear();
 Series10->Clear();
}
//---------------------------------------------------------------------------
void __fastcall TForm3::Button3Click(TObject *Sender)
{
 SavePictureDialog1->InitialDir;
  SavePictureDialog1->FileName="C:\\2.bmp";
 if (SavePictureDialog1->Execute())
 {
   Chart1->SaveToBitmapFile(SavePictureDialog1->FileName);

 }
}
//---------------------------------------------------------------------------
```

Вікно відображення хвиль

```
#include <vcl.h>
#include <fstream.h>
#pragma hdrstop
#include "Unit1.h"
#include "Unit4.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm4 *Form4;
extern char *filename;
//---------------------------------------------------------------------------
__fastcall TForm4::TForm4(TComponent* Owner)
        : TForm(Owner)
```

```
{
}
//---------------------------------------------------------------------------

void __fastcall TForm4::Button1Click(TObject *Sender)
{
int **A=new int* [1000];
 for(int col=0;col<1000;col++)
 { A[col]=new int [1000]; }
 ifstream ifs (filename, ifstream::in );
 {
 for (int i=0;i<1000;i++)
  {
   for (int j=0;j<1000;j++)
   {
    ifs>>A[i][j];
   }
  }
ifs.close();
 }

        for(int i = 0; i<1000; i++)
       {
          for(int j=0;j<1000;j++)
          {
            Image1->Canvas->Pixels[i][j]=0x00000000+30*(A[i][j]);
          }
       }

  for (int i=0; i<1000; i++)
```

```
  { delete [] A[i]; }
  delete []A;
}
//---------------------------------------------------------------------------
void __fastcall TForm4::BitBtn1Click(TObject *Sender)
{
Image1->Picture->Assign(0);
}
//---------------------------------------------------------------------------
void __fastcall TForm4::Button2Click(TObject *Sender)
{
 SavePictureDialog1->InitialDir;

  SavePictureDialog1->FileName="C:\\3.bmp";

 if (SavePictureDialog1->Execute())
 {
 Image1->Picture->SaveToFile(SavePictureDialog1->FileName);

 }
}
//---------------------------------------------------------------------------
void __fastcall TForm4::Button3Click(TObject *Sender)
{
  Image1->Canvas->Pen->Color = clGray;

    for(int x = 0; x < 1000; x += 100)
    {
       Image1->Canvas->MoveTo(x, 0);
       Image1->Canvas->LineTo(x, 1000);
```

```
    }

    for(int y = 0; y < 1000; y += 100)
    {
        Image1->Canvas->MoveTo(0, y);
        Image1->Canvas->LineTo(1000, y);
    }
}
//---------------------------------------------------------------------------
```

Вікно відображення ділянок

```
#include <vcl.h>
#include <fstream.h>
#include <iso646.h>
#pragma hdrstop
#include "Unit1.h"
#include "Unit5.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm5 *Form5;
extern char *filename;
//---------------------------------------------------------------------------
__fastcall TForm5::TForm5(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm5::Button1Click(TObject *Sender)
{
  int StartX, StartY;
```

```
 int sum=0, b=0, f=0;
 int **A=new int* [1000];
for(int col=0;col<1000;col++)
{ A[col]=new int [1000]; }
ifstream ifs (filename, ifstream::in );
{
for (int i=0;i<1000;i++)
 {
  for (int j=0;j<1000;j++)
  {
    ifs>>A[i][j];
    }
   }
 ifs.close();
}

      Graphics::TBitmap *DrawingBoard;
      Boolean IsDrawing;
      Form5->VertScrollBar->Visible= IsDrawing;
      DrawingBoard = new Graphics::TBitmap;
      int recev = 100;
      int matrix;
      matrix=recev*10;
      DrawingBoard->Width = recev;
      DrawingBoard->Height = recev;
      DrawingBoard->Canvas->Pixels[0][0]=0x000000;
      for (int i=0;i<=matrix;i+=recev)
      {
        for(int j=0;j<=matrix;j+=recev)
```

```
        {
            b=i;
             if(i==0) {b=0;}
            f=j;
             if(j==0) {f=0;}
            sum=0;
          if(i<=900 and j<=900)
           { for(int r=b;r<i+100;r++)
            {
            for(int k=f;k<j+100;k++)
              {
               sum=sum+A[r][k];
              }
            }
           }
            for(int x=0; x<recev; x++)
            {
             for(int y=0; y<recev; y++)
               {
                DrawingBoard->Canvas->Pixels[x][y]=sum;
               }
            }
            Form5->Image1->Canvas->Draw(j,i,DrawingBoard);
        }
      }

  for (int i=0; i<1000; i++)
  { delete [] A[i]; }
delete []A;
}
```

```
//---------------------------------------------------------------------------
void __fastcall TForm5::Button2Click(TObject *Sender)
{
Image1->Picture->Assign(0);
}
//---------------------------------------------------------------------------
void __fastcall TForm5::Button3Click(TObject *Sender)
{
SavePictureDialog1->InitialDir;
  SavePictureDialog1->FileName="C:\\4.bmp";
 if (SavePictureDialog1->Execute())
 {
 Image1->Picture->SaveToFile(SavePictureDialog1->FileName);
 }
}
//---------------------------------------------------------------------------
void __fastcall TForm5::Button4Click(TObject *Sender)
{
Image1->Canvas->Pen->Color = clGray;
    for(int x = 0; x < 1000; x += 100)
    {
        Image1->Canvas->MoveTo(x, 0);
        Image1->Canvas->LineTo(x, 1000);
    }
    for(int y = 0; y < 1000; y += 100)
    {
        Image1->Canvas->MoveTo(0, y);
        Image1->Canvas->LineTo(1000, y);
    }
}
```

```
//---------------------------------------------------------------------------
void __fastcall TForm5::Button5Click(TObject *Sender)
{
int sum=0, b=0, f=0;
 int **A=new int* [1000];
for(int col=0;col<1000;col++)
{ A[col]=new int [1000]; }
ifstream ifs (filename, ifstream::in );
{
for (int i=0;i<1000;i++)
 {
  for (int j=0;j<1000;j++)
  {
   ifs>>A[i][j];
   }
  }
 ifs.close();
}
      for (int i=0;i<1000;i+=100)
      {
        for(int j=0;j<1000;j+=100)
          {
             b=i;
              if(i==0) {b=0;}
             f=j;
              if(j==0) {f=0;}
             sum=0;
          if(j<=900 and i<=900)
          { for(int r=b;r<i+100;r++)
            {
```

```
            for(int k=f;k<j+100;k++)
              {
               sum=sum+A[r][k];
               }
             }
            }

     if(sum>0) Image1->Canvas->TextOut((j-(j/100))+47,(i-(i/100))+46, sum);

         }
       }
  for (int i=0; i<1000; i++)
  { delete [] A[i]; }
delete []A;
}
//---------------------------------------------------------------------------
```

Вікно візуалізації на карті

```
#include <vcl.h>
#include <fstream.h>
#include <iso646.h>
#pragma hdrstop
#include "Unit1.h"
#include "Unit6.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "SHDocVw_OCX"
#pragma resource "*.dfm"
TForm6 *Form6;
extern char *filename;
//---------------------------------------------------------------------------
```

```
__fastcall TForm6::TForm6(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm6::Button1Click(TObject *Sender)
{
 float  cx,cy;
 int level;
 WideString url;
  level=StrToInt(Edit4->Text);
  cx=StrToFloat(Edit1->Text);
  cy=StrToFloat(Edit2->Text);
 url="http://maps.google.com/staticmap?center="+IntToStr((int)cy)+'.'+
 IntToStr((int)(abs(cy-(int)cy)*1000000.))
 +","+
 IntToStr((int)cx)+'.'+
 IntToStr((int)(abs(cx-(int)cx)*1000000.))
 +"&&zoom="+IntToStr(level)+"&&size=600x600&key=MAPS_API_KEY";
   Edit3->Text=url;

}
//---------------------------------------------------------------------------
void __fastcall TForm6::Button2Click(TObject *Sender)
{
    wchar_t URL[150];
 Edit3->Text.WideChar(URL,150);
 CppWebBrowser1->Navigate(URL,0,NULL,NULL,NULL);
}
//---------------------------------------------------------------------------
```

```
void __fastcall TForm6::CppWebBrowser1DocumentComplete(TObject *Sender,
      LPDISPATCH pDisp, Variant *URL)
{
  CppWebBrowser1->Show();
  ShowMessage("   Карта завантажена!   ");
 int map_width=600,map_height=600;
  HDC hDc = CreateCompatibleDC(0);
HBITMAP hBmp = CreateCompatibleBitmap(GetDC(0),map_width,map_height);
                                    SelectObject(hDc, hBmp);
       BitBlt(hDc, 0, 0, map_width, map_height, GetDC(CppWebBrowser1-
>Handle), 12, 16, SRCCOPY);
       Graphics::TBitmap *bmp = new Graphics::TBitmap;
                                    bmp->Width = map_width;;
                                   bmp->Height =map_height;;
                                      bmp->Handle = hBmp;
                                  bmp->SaveToFile("mf.bmp");
                                         delete bmp;
CppWebBrowser1->Hide();
Image1->Visible=True;
Image1->Show();
Image1->Picture->LoadFromFile("mf.bmp");

}
//---------------------------------------------------------------------------
void __fastcall TForm6::Button4Click(TObject *Sender)
{
int x,y;

 int sum=0, b=0, f=0;
```

```
 int **A=new int* [1000];
for(int col=0;col<1000;col++)
{ A[col]=new int [1000]; }
ifstream ifs (filename, ifstream::in );
{
for (int i=0;i<1000;i++)
 {
  for (int j=0;j<1000;j++)
  {
    ifs>>A[i][j];
   }
  }
 ifs.close();
 }
      Graphics::TBitmap *DrawingBoard;
      Boolean IsDrawing;
      Form6->VertScrollBar->Visible= IsDrawing;
      DrawingBoard = new Graphics::TBitmap;
      int recev = 100;
      int matrix;
      matrix=recev*10;
      DrawingBoard->Width = 60;
      DrawingBoard->Height = 60;
      DrawingBoard->Canvas->Pixels[0][0]=0x000000;

      for (int i=0;i<=matrix;i+=recev)
      {
        for(int j=0;j<=matrix;j+=recev)
          {
```

```
                b=i;
                 if(i==0) {b=0;}
                f=j;
                 if(j==0) {f=0;}
                sum=0;
              if(j<=900 and i<=900)
               { for(int r=b;r<i+100;r++)
               {
                for(int k=f;k<j+100;k++)
                  {
                   sum=sum+A[r][k];
                  }
                }
               }


                for( x=0; x<60; x++)
                {
                  for( y=0; y<60; y++)
                    {
                     DrawingBoard->Canvas->Pixels[x][y]=+sum;
                    }
                 }

          if(sum>0) { Form6->Image1->Canvas->Draw(j-(j/100)*40,i-
(i/100)*40,DrawingBoard); }
                }
              }

        for (int i=0; i<1000; i++)
```

```
  { delete [] A[i]; }
delete []A;
}
//---------------------------------------------------------------------------
void __fastcall TForm6::Button5Click(TObject *Sender)
{
Image1->Picture->Assign(0);
}
//---------------------------------------------------------------------------
void __fastcall TForm6::Button6Click(TObject *Sender)
{
SavePictureDialog1->InitialDir;
  SavePictureDialog1->FileName="C:\\5.bmp";
 if (SavePictureDialog1->Execute())
 {
 Image1->Picture->SaveToFile(SavePictureDialog1->FileName);
 }
}
//---------------------------------------------------------------------------
void __fastcall TForm6::Button3Click(TObject *Sender)
{
Image1->Canvas->Pen->Color = clGray;
    for(int x = 0; x < 600; x += 60)
    {
        Image1->Canvas->MoveTo(x, 0);
        Image1->Canvas->LineTo(x, 600);
    }

    for(int y = 0; y < 600; y += 60)
    {
```

```
        Image1->Canvas->MoveTo(0, y);
        Image1->Canvas->LineTo(600, y);
    }
}
//---------------------------------------------------------------------------
void __fastcall TForm6::Button7Click(TObject *Sender)
{
 int sum=0, b=0, f=0;
 int **A=new int* [1000];
for(int col=0;col<1000;col++)
{ A[col]=new int [1000]; }
ifstream ifs (filename, ifstream::in );
{
for (int i=0;i<1000;i++)
 {
  for (int j=0;j<1000;j++)
  {
   ifs>>A[i][j];
   }
  }
 ifs.close();
 }
      for (int i=0;i<1000;i+=100)
      {
        for(int j=0;j<1000;j+=100)
          {
             b=i;
              if(i==0) {b=0;}
             f=j;
              if(j==0) {f=0;}
```

```
            sum=0;
          if(j<=900 and i<=900)
          { for(int r=b;r<i+100;r++)
            {
             for(int k=f;k<j+100;k++)
               {
                sum=sum+A[r][k];
                }
             }
            }
           Image1->Canvas->TextOut((j-(j/100)*40)+20,(i-(i/100)*40)+20, sum);
         }
      }
  for (int i=0; i<1000; i++)
  { delete [] A[i]; }
delete []A;

}
//---------------------------------------------------------------------------
void __fastcall TForm6::Button8Click(TObject *Sender)
{
CppWebBrowser1->Hide();
Image1->Visible=True;
Image1->Show();
Image1->Picture->LoadFromFile("mf.bmp");
}
//---------------------------------------------------------------------------
void __fastcall TForm6::Button9Click(TObject *Sender)
{ int sum=0, b=0, f=0;
```

```
 int **A=new int* [1000];
for(int col=0;col<1000;col++)
{ A[col]=new int [1000]; }
ifstream ifs (filename, ifstream::in );
{
for (int i=0;i<1000;i++)
 {
  for (int j=0;j<1000;j++)
  {
    ifs>>A[i][j];
   }
  }
 ifs.close();
}
      for (int i=0;i<1000;i+=100)
      {
        for(int j=0;j<1000;j+=100)
          {
             b=i;
              if(i==0) {b=0;}
             f=j;
              if(j==0) {f=0;}
             sum=0;
           if(j<=900 and i<=900)
            { for(int r=b;r<i+100;r++)
              {
               for(int k=f;k<j+100;k++)
               {
                sum=sum+A[r][k];
                }
```

```
                }
              }
          if(sum>0) { Image1->Canvas->Pixels[(j-(j/100)*40)+31][(i-(i/100)*40)+33] =
sum;
                    Image1->Canvas->Pixels[(j-(j/100)*40)+31][(i-(i/100)*40)+34] = sum;
                    Image1->Canvas->Pixels[(j-(j/100)*40)+31][(i-(i/100)*40)+35] = sum;
                    Image1->Canvas->Pixels[(j-(j/100)*40)+29][(i-(i/100)*40)+33] = sum;
                    Image1->Canvas->Pixels[(j-(j/100)*40)+29][(i-(i/100)*40)+34] = sum;
                    Image1->Canvas->Pixels[(j-(j/100)*40)+29][(i-(i/100)*40)+35] = sum;
                    Image1->Canvas->Pixels[(j-(j/100)*40)+30][(i-(i/100)*40)+33] = sum;
                    Image1->Canvas->Pixels[(j-(j/100)*40)+30][(i-(i/100)*40)+34] = sum;
                    Image1->Canvas->Pixels[(j-(j/100)*40)+30][(i-(i/100)*40)+35] = sum;
                    }
              }
          }
   for (int i=0; i<1000; i++)
   { delete [] A[i]; }
  delete []A;
  }
  //---------------------------------------------------------------------------
```

Сергій Олексійович Карпік
магістрант інформаційних технологій

Сучасні системи візуалізації даних

ІН 21М

Комп'ютерний набір, верстка і макетування та дизайн в редакторі Microsoft®Office® Word 2003 С.О. Карпік

Науковий керівник: Р.М.Літнарович, канд. техн. наук, доцент

Міжнародний Економіко-Гуманітарний Університет ім. акад. Степана Дем'янчука

Кафедра математичного моделювання
33027,м.Рівне,Україна
Вул.акад. С.Дем'янчука,4, корпус 1
Телефон:(+00380) 362 23-73-09
Факс:(+00380) 362 23-01-86
E-mail:mail@regi.rovno.ua
E-mail: serhiy1990@yandex.ru