Лекція №1

Тема: Комп'ютерні методи кодування зображення. Графічні примітиви.

Мета: Написати програму виведення на екран малюнків які складаються з графічних примітивів; визначити колір графічних примітивів за кольоровою схемою RGB32.

ПЛАН

1. Методи кодування растрового зображення.

2. Задання кольору в форматі RGB.

3. Графічні примітиви. Залежність набору графічних примітивів від апаратних та програмних засобів малювання.

4. Процедурне компонентне малювання.

1. Методи кодування комп'ютерного зображення.

Всі зображення, які зберігаються та оброблюються на комп'ютерах поділяють на векторні та растрові. Розглянемо окремо кожен з них.

1.1. Векторне зображення

Зображення яке складається з сукупності математичних об'єктів називають векторним. Більш часто в якості таких об'єктів виступають графічні примітиви. Для малюнків на площині графічними примітивами є точки, відрізки прямих, прямокутники з сторонами паралельними сторонам вікна програми, еліпси. Кожен графічний примітив за допомогою математичних розрахунків можна отримати з довільною точністю, тому зміна масштабу такого малюнку не погіршить його якості та чіткості контурів. Однак такі малюнки досить рідко використовуються, бо реалістичне зображення, наприклад портрет реальної людини, не можна якісно зобразити графічними примітивами. Напроти, графічними примітивами досить просто зобразити мультиплікаційні малюнки, літери, символи, герби та інші подібні зображення. Розглянемо способи задання графічних примітивів.

Точка. Для задання точки потрібно знати її координати, кількість яких залежить від простору в якому вона задана: для площини це буде дві координати {x,y} та для простору це

буде три координати {x,y,z}. Іноді з точкою зв’язують атрибути: форма (коло чи прямокутник), розмір, колір.

Рис. 1 — Зображення точок

Відрізок або лінія. Частина прямої, яка сполучає задані дві точки. Таким чином для задання лінії потрібно мати координати двох точок. Часто для лінії використовують атрибути кольору, товщини та стилю (штрих, пунктир, пунктир з точкою та інші). Відповідно, на практиці може зустрітися реалізація малювання ліній у вигляді переліку координат точок, а безпосередньо при малюванні ліній використовують порядкові номера перелічених точок. Для пласкої графіки частіше координати точок вставляють безпосередньо в команду малювання лінії. Різні методи відновлення зображення можуть мати більш просту реалізацію так першим так і другим методами.

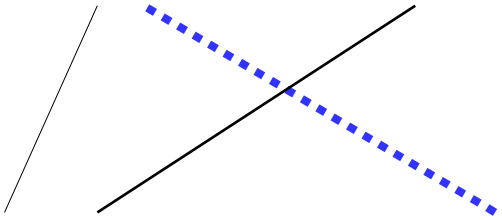

Рис. 2 — Приклади ліній

Прямокутник. Цей примітив може задаватися декількома способами. Більш часто використовують координати двох протилежних вершин, завдяки паралельності сторін прямокутника сторонам вікна програми таке задання є однозначним. В деяких програмних реалізаціях можна зустріти метод задання прямокутника положенням його верхньої лівої вершини та його ширини та висоти. Для обох методів задання прямокутника є можливість перейти від одного до іншого в декілька операцій додавання та віднімання. Прямокутник може мати атрибути як і у лінії, але також додаються атрибути методу фарбування (суцільно, в смужку, й т.п.) та кольору фарбування.

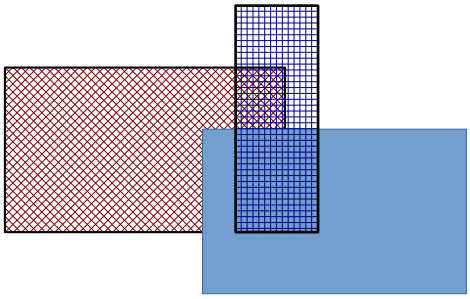

Рис. 3 — Зображення прямокутників

Еліпс. Як і прямокутник може задаватися декількома способами та має додаткові атрибути фарбування. До способів завдання відноситься задання прямокутника, в який буде вписаний еліпс, або центр еліпсу та його радіуси в горизонтальному та вертикальному напрямках.

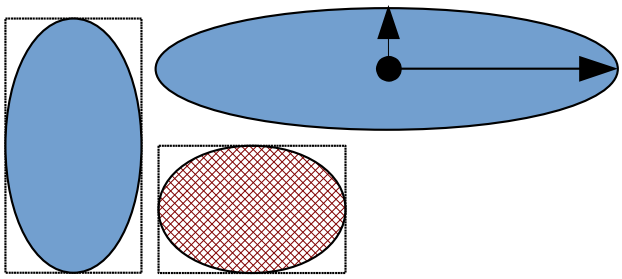

Рис. 4 — Еліпси

Наступні графічні примітиви не є обов’язковими для систем відтворення зображення.

Ламана лінія. Є розширенням звичайної лінії. Задається масивом точок. Атрибути збігаються з атрибутами відрізку. Часто в системах з обмеженими можливостями всі примітиви реалізуються за допомогою ламаних ліній, де прямі ділянки є малого розміру. Але при такому спрощенні зображення псується при значному збільшенні малюнка.

Полігон. Ламана лінія для якої перша точка збігається з останньою утворює контур. Середина такого контура може бути зафарбована, що утворює багатокутники. В більшості реалізацій вимагається щоб ламана лінія контуру не мала самоперетинів, але в загальному випадку, задача побудови багатокутника з контуром, який має самоперетини, розв’язна.

Криві Безьє та інші криві. До таких примітивів відносяться засоби малювання кривих ліній без зламів. В якості таких ліній використовують сплайни, криві Безьє, частотні та ймовірнісні апроксимації та інтерполяції. Всі ці методи зводяться до того, щоб по заданим (або біля заданих) точкам провести плавну криву.

1.2. Растрове зображення

До растрових зображень відносяться всі зображення, які складаються з кольорових точок — пікселів. При цьому самі пікселі можуть бути досить різної форми та розмірів:

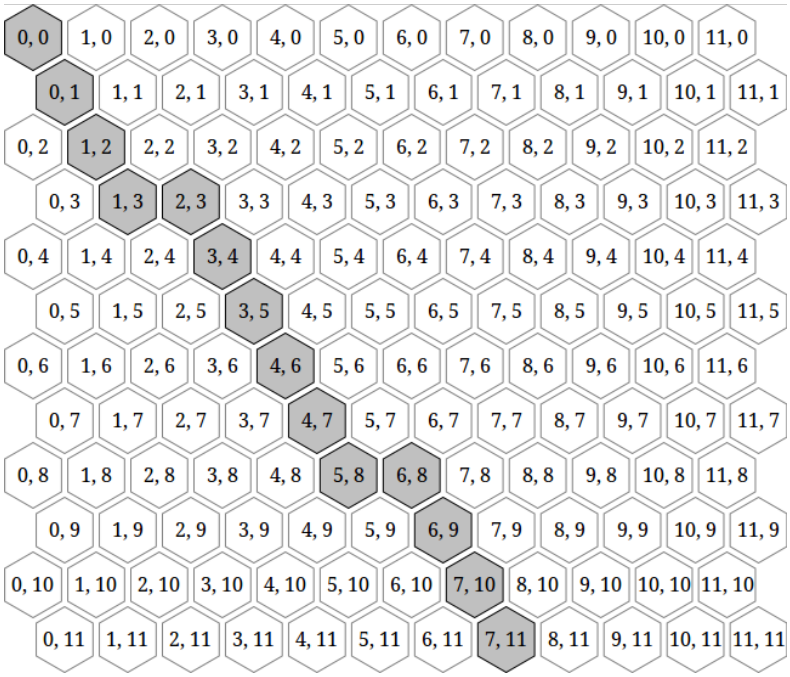

Рис. 5 — Гексагональний растр та зображення прямої на ньому

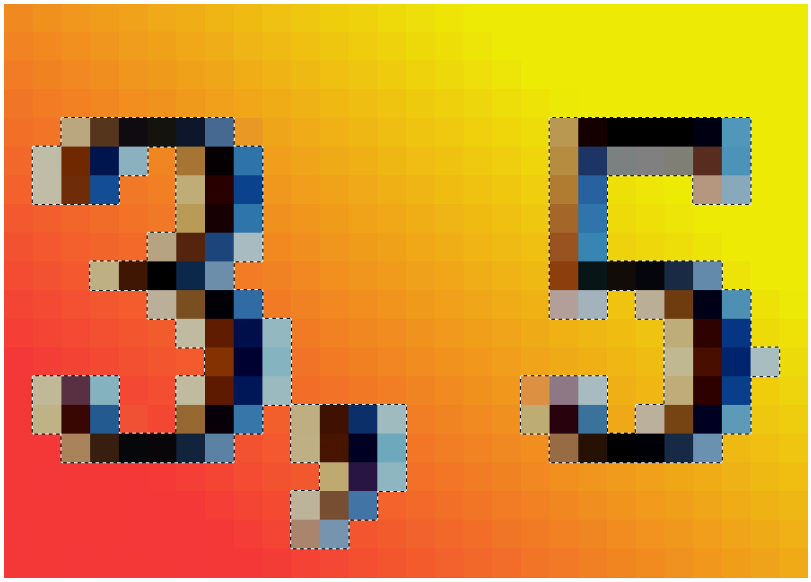

Рис. 6 — Квадратний растр та надпис “3,5” на ньому

Фактично, кожне зображення, яке ми бачимо на екрані монітора складається з кольорових квадратів (з деякого часу квадратний піксель став стандартом на більшості

пристроїв, хоча можна зустріти й досі відхилення до “приплюснутих” пікселів з не стандарним відношенням сторін екрану). В разі використання векторної бази, програма або пристрій призводить векторне зображення до растрового за допомогою математичних розрахунків, і передає на монітори у вигляді квадратного растру. Єдиною відмінністю від растрового зображення, векторне зображення можна відтворити у растрове з наперед заданою точністю і використовувати всі можливості монітора на повну.

2. Задання кольору в форматі RGB.

В процесі розвитку комп’ютерної техніки постійно змінювалися засоби кодування кольору. Так, при малих об’ємах доступної відеопам’яті, використовували монохромне зображення, 4 кольорів, 16 кольорів та 256 кольорів заданих наперед таблицею. Та сучасний стан розвитку пристроїв виводу графічної інформації призвів до використання повнокольорової графіки навіть на мобільних пристроях, тому зупинятися на індексованих кольорах в рамках загального курсу не є доцільним. Тому розглянемо кодування кольору цілочисловими значеннями 16, 24 та 32 розрядного формату.

24 розрядний формат представлення кольору. Всі системи передачі кольорів використовують властивість людського ока трикомпонентної чутливості. За цією властивістю вдається імітувати сприйняття більшості кольорів за допомогою лише трьох джерел світла: червоного, зеленого та синього. За позначеннями цих кольорів Red-Green-Blue така схема відтворення кольорів має назву RGB формату. Для 24 розрядного формату RGB використано представлення кольору, як запис окремих яскравостей по компонентам 0..255 по 8 розрядів на кожен (по 1 байту, всього три байти), що разом складає 24 біти.

Досить часто колір виражається чотири-байтним цілочисловим невід'ємним значенням, яке на апаратному рівні відеопристрою інтерпретується як чотири байтові компоненти кольору. В таких випадках четверта компонента або ігнорується, або розуміється як прозорість пікселів при накладанні малюнків одне на одне. Скласти чотири компоненти кольорів можна за допомогою такої схеми:

Color = R+G*256+B*256*256+A*256*256*256, або

Color := R or (G shl 8) or (B shl 16) or (A shl 24),

де A є компонентою прозорості. Результат в шістнадцятковому коді матиме вигляд $AABBGGRR.

Таким чином, змінити малюнок можна записавши до пам’яті, де знаходиться колір

пікселя, нове значення кольору. Іншою проблемою є правильність вибору цього пікселя коли ми, наприклад, малюємо коло.

3. Графічні примітиви. Залежність набору графічних примітивів від апаратних та програмних засобів малювання.

В першій главі визначено графічні примітиви та їх методи задання, але не вказано засобів їх зображення на вікні програми. На жаль універсального засобу для їх зображення не існує, але більш близьким до парадигми малювання Windows вікон є малювання за допомогою фреймворків Delphi, C++ Builder, Lazarus FreePascal та деякі інші. Останній від попередніх відрізняється безкоштовністю для будь-якого використання, тому зупинимося саме на ньому. Дистрибутив Lazarus можна скачати за посиланням http://www.lazarus-ide.org/. Інсталяція є простою не вимагає провідних знань до операційної системи. В Debian похідних операційних системах Lazarus краще встановити з офіційних репозитаріїв.

Вікно системи програмування має вигляд:

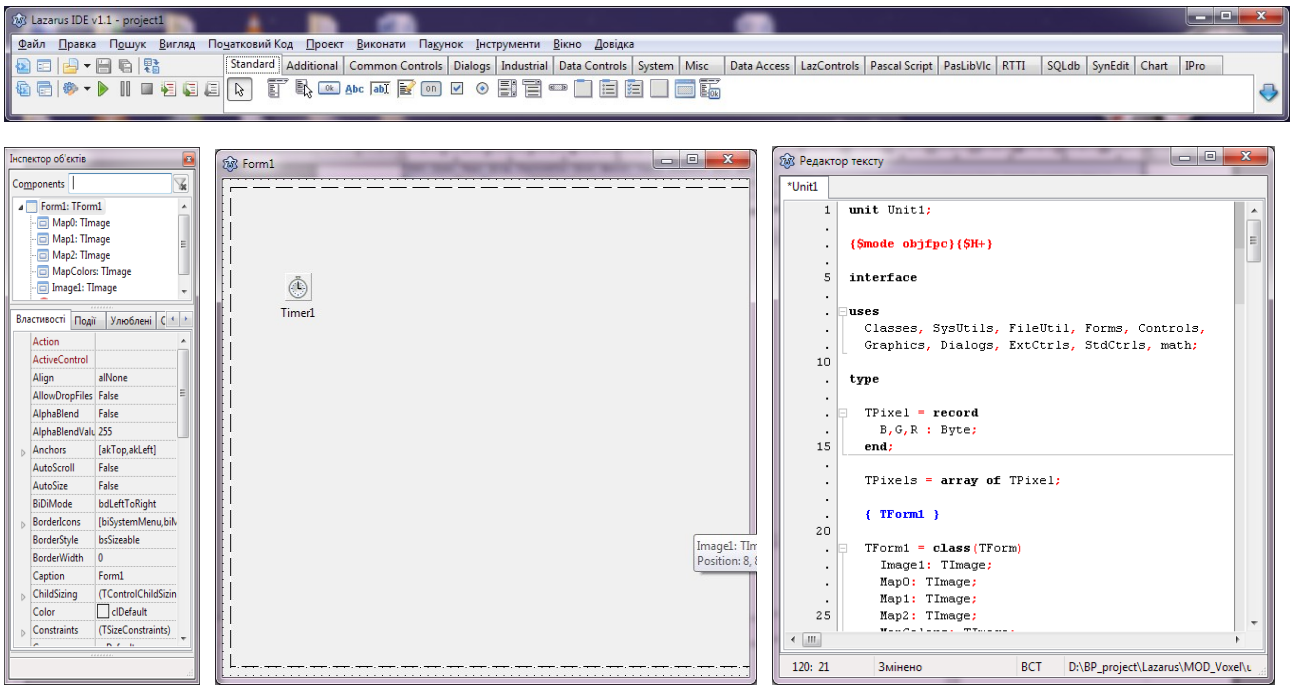

Рис. 7 — Вигляд середовища програмування Lazarus

Далі вважатимемо що читачу вже відомі особливості програмування мовою PASCAL або Free Pascal чи Delphi. Якщо такого досвіду немає, то можна ознайомитися з зазначеною

літературою:

1. Кетков, Ю. Л. Свободное программное обеспечение. FREE PASCAL для студентов и школьников / Ю. Л. Кетков, А. Ю. Кетков. — СПб.: БХВ-Петербург, 2011. — 384 с.

2. Мансуров К.Т. Основы программирования в среде Lazarus, 2010. – 772 с.: ISBN 978-9967-03-646-8

Вказана література є доступною для використання на сервері кафедри ПЗІ.

Для початку роботи необхідно закрити поточний проект та створити новий проект “Програма”. Після цього Ви побачите на екрані чотири вікна (рис. 7): зверху головне вікно менеджера проекту з компонентами, ліворуч менеджер компонентів проекту, по центру проект головного вікна програми, праворуч текст програми. Першою дією з меню виконуємо “Зберегти все” і зберігаємо складові проекту в окрему теку.

Малювання будемо проводити на компоненті TImage з вкладки компонентів Additional. При цьому Ви матимете можливість змінювати позицію й розмір на проекті вікна програми майбутнього малюнка за допомогою миші. Далі, подвійним кліком миші по вільній частині вікна переходимо до процедури, яка виконується при створені цього вікна. В цій процедурі будемо виконувати команди малювання.

Система координат в растровій графіці дещо відрізняється від систем координат прийнятої в математиці. В першу чергу координати є цілочисловими. По-друге, вертикальна вісь направлена вниз, тобто більше значення відповідає нижчому пікселю. Точка (0;0) відповідає верхньому лівому куту малюнка.

Малювання точки-пікселя. Для малювання на компонентах використовується властивість Canvas. Саме ця властивість містить всі засоби керування малюванням та графічного зображення інформації. Цю властивість має переважна більшість компонентів, виключаючи ті, за малювання яких відповідає безпосередньо операційна система. Таким чином, щоб поставити точку на малюнок за координатами (x;y) потрібно записати код:

```
Image1.Canvas.Pixels[x,y]:=MyColor;
```

де x, y є цілі числа або цілочислові змінні.

Малювання відрізків. Для малювання ліній потрібно попередньо вказати колір цієї лінії та її параметри. За ці властивості відповідає параметр Pen:

```
Image1.Canvas.Pen.Color := clRed; //Задання кольору ліній
Image1.Canvas.Pen.Width := 3; //Задання товщини ліній
```

Набір параметрів ліній є значно ширшим, але для початкового малювання вистачить лише цих двох. З іншими параметрами Ви можете ознайомитись з додаткової літератури або за власними експериментами. Малювання власне відрізку проводиться за допомогою команди:

```
Image1.Canvas.Line(50,50,150,150);
Image1.Canvas.Line(x1,y1,x2,y2);
```

Для подальших лабораторних робіт, де буде проходити все більш широка растеризація векторних зображень, команд малювання ліній та точок вистачить. Та знання додаткових команд малювання значно спростить створення власних малюнків, тому продовжимо.

Малювання прямокутників. Для малювання прямокутника потрібно вказувати не лише властивості лінії, а й ще параметри його фарбування. Для цього властивість Canvas містить складову властивість Brush – пензель. За допомогою пензля можна вказати не лише колір фарбування але й потрібне штрихування. Особливо корисним штрихування є при малюванні різноманітних діаграм. Наступний код демонструє малювання синього прямокутника з зеленим обрамленням:

```
Image1.Canvas.Pen.Color := clGreen; // $00FF00
Image1.Canvas.Brush.Color := clBlue; // $FF0000
Image1.Canvas.FillRect(150,50,200,150);
Image1.Canvas.FillRect(x1,y1,x2,y2);
```

Малювання еліпсів. Для малювання еліпсів використовуються аналогічні параметри, що й для прямокутників. Задається еліпс прямокутником, в який система малювання впише еліпс, тому незафарбовані зони на проектних макетах можуть дещо не збігатися. Команди, які малюють еліпс поверх попереднього прямокутника показано наступним лістингом:

```
Image1.Canvas.Pen.Color := clRed; // $0000FF
Image1.Canvas.Brush.Color := clYellow; // $00FFFF
Image1.Canvas.Ellipse(150,50,200,150);
```

Інші методи задання зазначених графічних примітивів або інші графічні примітиви, при бажанні, опрацюйте самостійно. Надалі вміння виводити вказані примітиві дозволить виконати всі обов'язкові завдання.

Всі зазначені графічні примітиви можуть бути інтерпретовані як елементи векторної графіки, а їх малювання — перетворення векторного зображення у растрове (растеризація). Приклад малюнків складених з графічних примітивів можна переглянути на наступному рисунку:

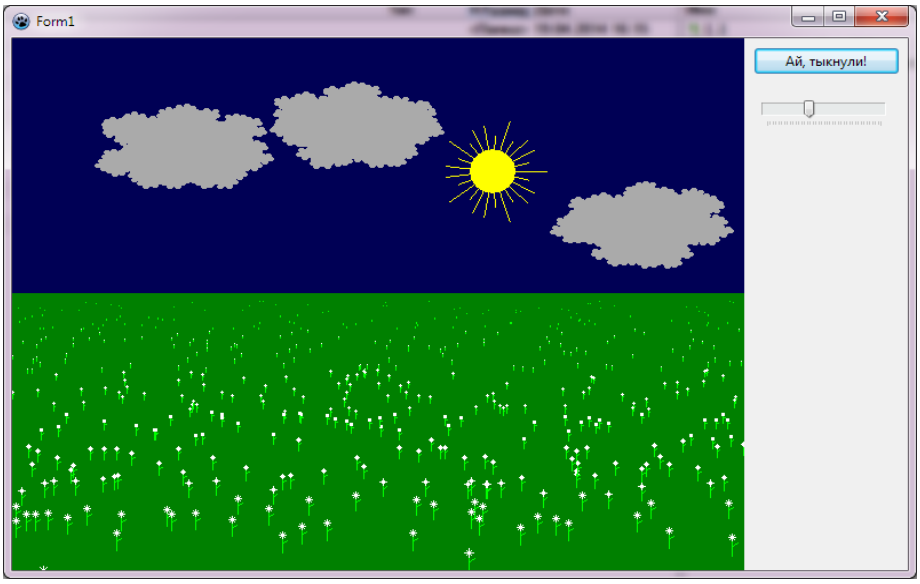

Рис. 8 — Використання графічних примітивів

4. Процедурне компонентне малювання.

Часто при малюванні об'єкту з графічних примітивів, комбінація “квіточка” використовується неодноразово, тому біль істотно використовувати функції для малювання певної комбінації графічних примітивів. Наприклад малювання паркану відбувається малюванням штахетнику, де є штахетина, яка повторюється до півсотні разів. Істотно, що таке малювання краще зробити циклом, в якому штахетина міститиметься в тілі циклу. Якщо ж вимагається малювати нерегулярне розташування, то код малювання групи примітивів, наприклад “птах”, краще винести в окрему процедуру. Ця процедура прийматиме бажане положення птаха й малювати його в вказаному місці.

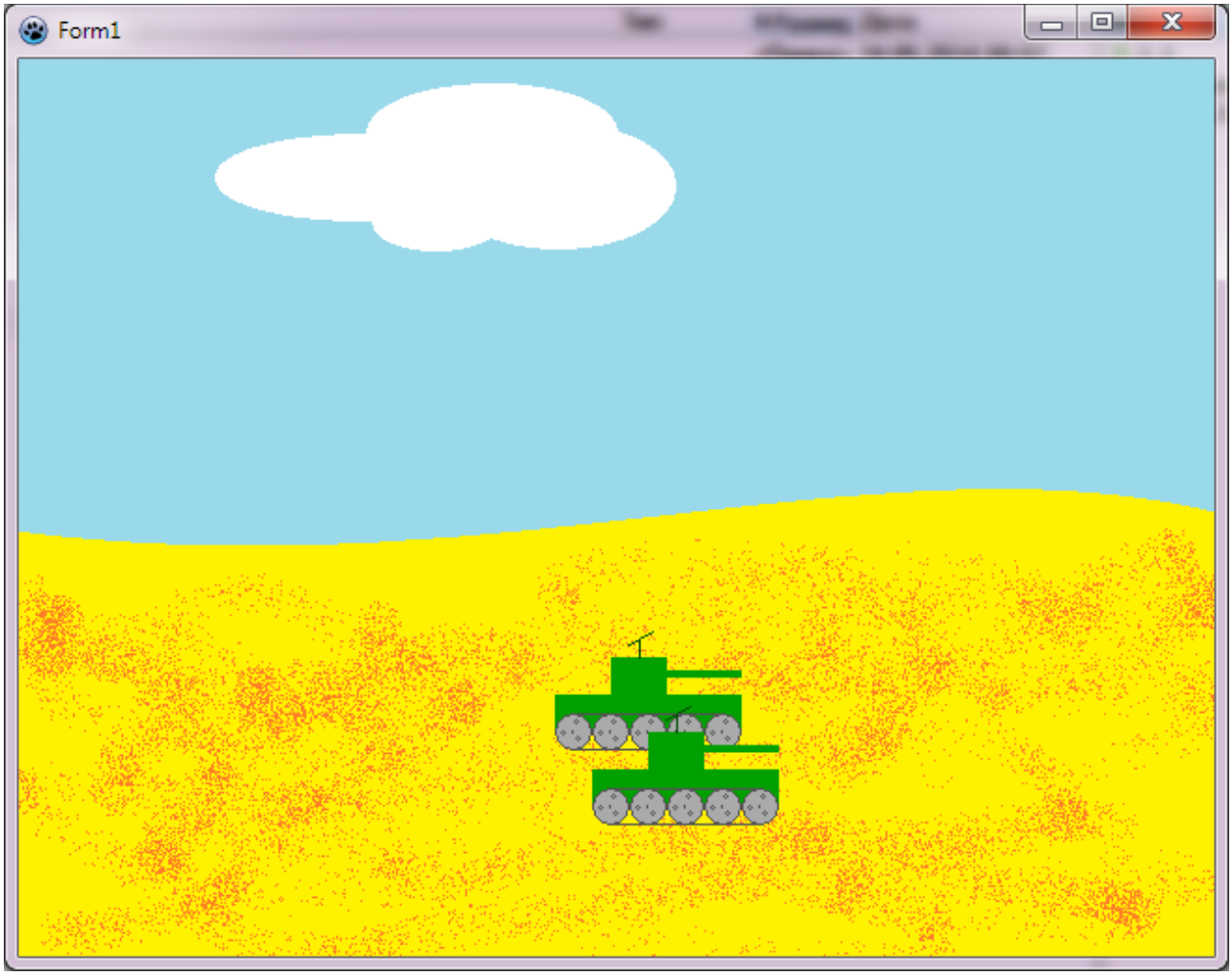

Рис. 9 — Процедурний вивід коліс та корпусів танку

На рис. 9 показано малювання двох танків на різних місцях. При цьому кожен танк має по п'ять коліс, які малювати окремо зовсім не обов'язково. Тому малювання танку реалізовано окремою процедурою, яка в свою чергу користується процедурою малювання коліс (див. наступний лістинг):

```
  TForm1 = class(TForm)
    Image1: TImage;
...
  public
...
    kadr: Integer; // Змінювати для обертання коліс
    procedure DrawKoleso(x, y: integer);
    procedure DrawTank(x, y: integer);
  end;
```

```
procedure TForm1.DrawKoleso(x, y: integer);
var dx,dy:integer;
begin
  Image1.Canvas.Pen.Color := TColor($555555);
  Image1.Canvas.Brush.Color:= TColor($AAAAAA);
  Image1.Canvas.Ellipse(x,y,x+20,y+20);
  Image1.Canvas.Ellipse(x+9,y+9,x+11,y+11);
  dx := 8+round(5.0*cos(0.2*kadr));
  dy := 8+round(5.0*sin(0.2*kadr));
  Image1.Canvas.Ellipse(x+dx,y+dy,x+3+dx,y+3+dy);
  dx := 8+round(5.0*cos(0.2*kadr+PI*2.0/3.0));
  dy := 8+round(5.0*sin(0.2*kadr+PI*2.0/3.0));
  Image1.Canvas.Ellipse(x+dx,y+dy,x+3+dx,y+3+dy);
  dx := 8+round(5.0*cos(0.2*kadr+PI*4.0/3.0));
  dy := 8+round(5.0*sin(0.2*kadr+PI*4.0/3.0));
  Image1.Canvas.Ellipse(x+dx,y+dy,x+3+dx,y+3+dy);
end;

procedure TForm1.DrawTank(x, y: integer);
begin
  Image1.Canvas.Pen.Color := TColor($005000);
  Image1.Canvas.Brush.Color:= TColor($00A000);
  Image1.Canvas.FillRect(x,y,x+100,y+20);
  for i:=0 to 4 do
  begin
    nx:=x+20*i;
    DrawKoleso(nx,y+10);
  end;
  DrawTrack(x+10,y+10);
  DrawTrack(x+10,y+29);
  DrawBashta(x+30,y+20);
end;
```

В результаті отримано код, який використовує опорні координати, від яких починається

будуватися зображення танку. Відповідно, кількість танків намальованих на екрані можна збільшити зі значно меншими затратами на написання коду — достатньо лише однієї команди в вказаним знаходженням.