лекция №1

Тема: Компьютерные методы кодирования изображения. Графические примитивы.

цель: Написать программу вывода на экран рисунков состоящих из графических примитивов; определить цвет графических примитивов по цветовой схеме RGB32.

ПЛАН

1. Методы кодирования растрового изображения.

2. Задание цвета в формате RGB.

3. Графические примитивы. Зависимость набора графических примитивов от аппаратных и программных средств рисования.

4. Процедурный компонентное рисования.

1. Методы кодирования компьютерного изображения.

Все изображения, которые хранятся и обрабатываются на компьютерах разделяют на векторные и растровые. Рассмотрим отдельно каждый из них.

1.1. векторное изображение

Изображение состоящее из совокупности математических объектов называют векторным. Более часто в качестве таких объектов выступают графические примитивы. Для рисунков на плоскости графическими примитивами являются точки, отрезки прямых, прямоугольники со сторонами параллельными сторонам окна программы, эллипсы. Каждый графический примитив с помощью математических расчетов можно получить из произвольной точностью, поэтому изменение масштаба такого рисунке не ухудшит его качества и четкости контуров. Однако такие рисунки довольно редко используются, потому реалистичное изображение, например портрет реального человека, нельзя качественно изобразить графическими примитивами. Напротив, графическими примитивами достаточно просто изобразить мультипликационные рисунки, буквы, символы, гербы и другие подобные изображения. Рассмотрим способы задания графических примитивов.

Точка. Для задания точки нужно знать ее координаты, количество которых зависит от пространства в котором она задана: для плоскости это будет две координаты {x, y} и для пространства это

будет три координаты {x, y, z}. Иногда с точкой связывают атрибуты: форма (круг или прямоугольник), размер, цвет.

Рис. 1 - изображение точек

Отрезок или линия. Часть прямой, соединяющей заданные две точки. Таким образом, для задания линии нужно иметь координаты двух точек. Часто для линии используют атрибуты цвета, толщины и стиля (штрих, пунктир, пунктир с точкой и другие). Соответственно, на практике может встретиться реализация рисования линий в виде перечня координат точек, а непосредственно при рисовании линий используют порядковые номера перечисленных точек. Для плоской графики чаще координаты точек вставляют непосредственно в команду рисования линии. Различные методы восстановления изображения могут иметь более простую реализацию так первым так и вторым методами.

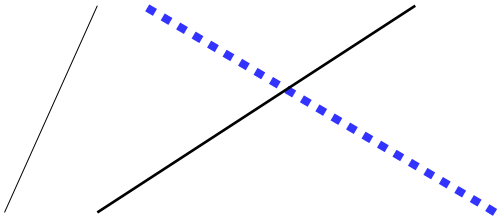

Рис. 2 - Примеры линий

Прямоугольник. Этот примитив может задаваться несколькими способами. Более часто используют координаты двух противоположных вершин, благодаря параллельности сторон прямоугольника сторонам окна программы такое задание является однозначным. В некоторых программных реализациях можно встретить метод задания прямоугольника положением его верхней левой вершины и его ширины и высоты. Для обоих методов задания прямоугольника есть возможность перейти от одного к другому в несколько операций сложения и вычитания. Прямоугольник может иметь атрибуты как и в линии, но также добавляются атрибуты метода покраски (сплошь, в полоску, и т.п.) и цвета окраски.

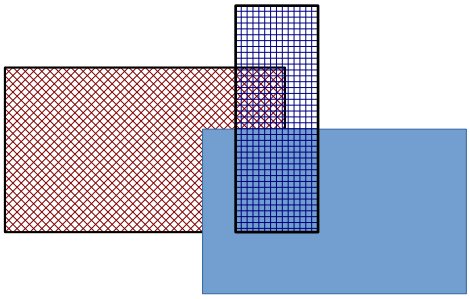

Рис. 3 - изображение прямоугольников

Эллипс. Как и прямоугольник может задаваться несколькими способами и имеет дополнительные атрибуты окраску. К способам задача относится задания прямоугольника, в который будет вписан эллипс, или центр эллипса и его радиусы в горизонтальном и вертикальном направлениях.

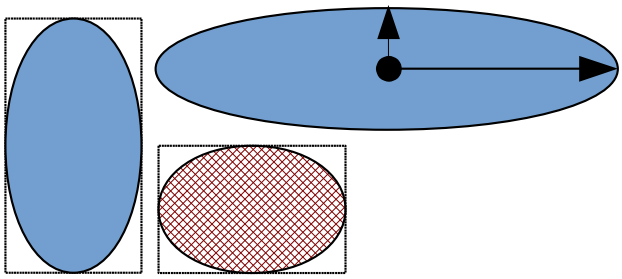

Рис. 4 - эллипс

Следующие графические примитивы не являются обязательными для систем отображения.

Ломаная линия. Есть расширением обычной линии. Задается массивом точек. Атрибуты совпадают с атрибутами отрезке. Часто в системах с ограниченными возможностями все примитивы реализуются с помощью ломаных линий, где прямые участки является малого размера. Но при таком упрощении изображение портится при значительном увеличении рисунка.

Полигон. Ломаная линия для которой первая точка совпадает с последней образует контур. Середина такого контура может быть закрашена, что образует многоугольники. В большинстве реализаций требуется чтобы ломаная линия контура не имела самопересечений, но в общем случае, задача построения многоугольника с контуром, который имеет самопересечений, разрешающая.

Кривые Безье и другие кривые. К таким примитивов относятся средства рисования кривых линий без изломов. В качестве таких линий используют сплайны, кривые Безье, частотные и вероятностные аппроксимации и интерполяции. Все эти методы сводятся к тому, чтобы по заданным (или около заданных) точкам провести плавную кривую.

1.2. растровое изображение

К растровых изображений относятся все изображения, состоящие из цветных точек - пикселей. При этом сами пиксели могут быть весьма различной формы и размеров:

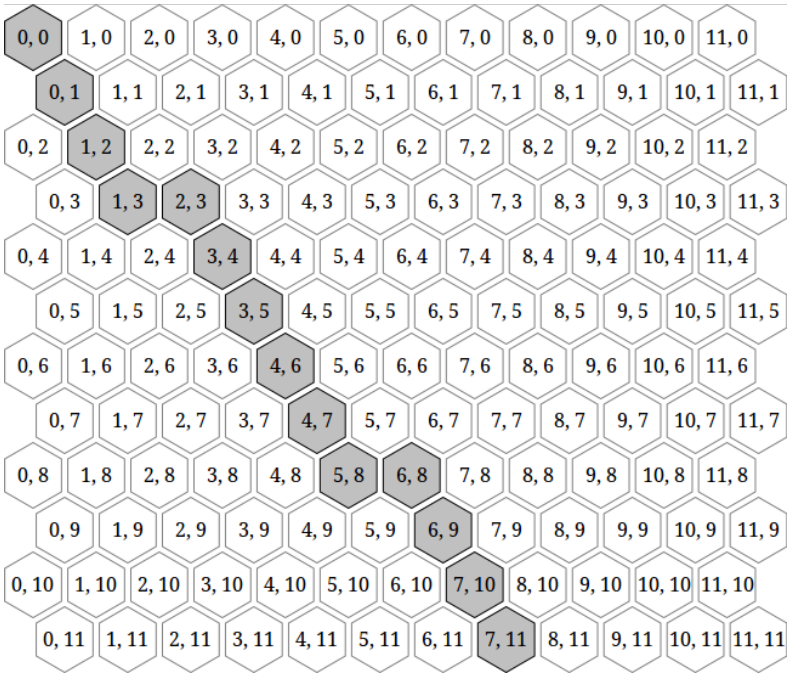

Рис. 5 - Гексагональный растр и изображение прямой на нем

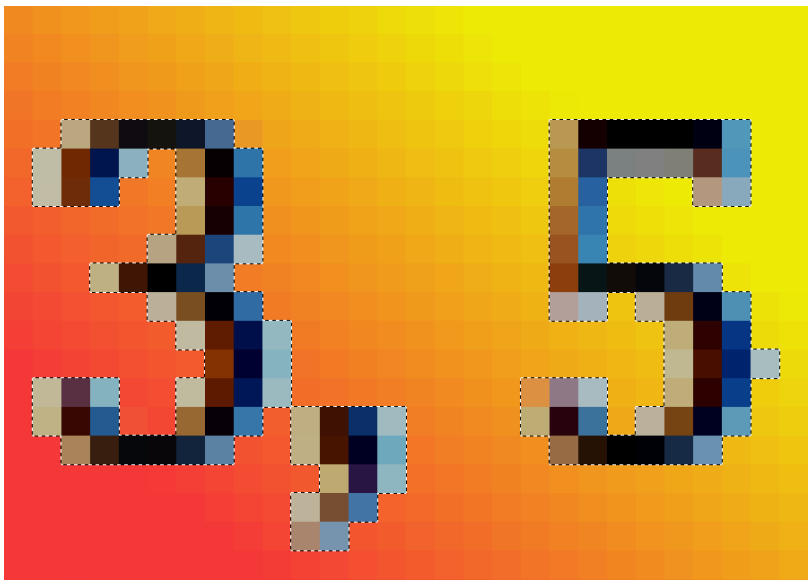

Рис. 6 - Квадратный растр и надпись "3,5" на нем

Фактически, каждое изображение, которое мы видим на экране монитора состоит из цветных квадратов (с некоторых пор квадратный пиксель стал стандартом на большинстве

устройств, хотя можно встретить до сих пор отклонения в "приплюснутые" пикселей с не стандарного отношением сторон экрана). При использовании векторной базы, программа или устройство приводит векторное изображение к растрового с помощью математических расчетов, и передает на мониторы в виде квадратного растра. Единственным отличием от растрового изображения, векторное изображение можно воспроизвести в растровый с заранее заданной точностью и использовать все возможности монитора на полную.

2. Задание цвета в формате RGB.

В процессе развития компьютерной техники постоянно менялись средства кодирования цвета. Так, при малых объемах доступной видеопамяти, использовали монохромное изображение, 4 цветов, 16 цветов и 256 цветов заданных заранее таблицей. И современное состояние устройств вывода графической информации привел к использованию полноцветной графики даже на мобильных устройствах, поэтому останавливаться на индексированных цветах в рамках общего курса не является целесообразным. Поэтому рассмотрим кодирования цвета целочисленными значениями 16, 24 и 32 разрядного формата.

24 разрядный формат представления цвета. Все системы передачи цветов используют свойство человеческого глаза трехкомпонентной чувствительности. По этим свойством удается имитировать восприятие большинства цветов с помощью только трех источников света: красного, зеленого и синего. По обозначениями этих цветов Red-Green-Blue такая схема воспроизведения цветов называется RGB формата. Для 24 разрядного формата RGB использовано представление цвета, как запись отдельных яркостей по компонентам 0..255 по 8 разрядов на каждый (по 1 байту, всего три байта), что вместе составляет 24 бита.

Достаточно часто цвет выражается четыре-байтный целочисленным неотъемлемым значением, которое на аппаратном уровне видеоустройства интерпретируется как четыре байта компоненты цвета. В таких случаях четвертая компонента или игнорируется, или понимается как прозрачность пикселей при наложении рисунков друг на друга. Составить четыре компонента цветов можно с помощью следующей схемы:

Color = R + G * 256 + B * 256 * 256 + A * 256 * 256 * 256, или Color = R
or (G shl 8) or (B shl 16) or (A shl 24),

где A есть компонентом прозрачности. Результат в шестнадцатеричном коде будет выглядеть $ AABBGGRR.

Таким образом, изменить рисунок можно записав в памяти, где находится цвет

пикселя, новое значение цвета. Другой проблемой является правильность выбора этого пикселя когда мы, например, рисуем круг.

3. Графические примитивы. Зависимость набора графических примитивов от аппаратных и программных средств рисования.

В первой главе определены графические примитивы и их методы задания, но не указано средств их изображения на окне программы. К сожалению универсального средства для их изображения не существует, но более близким к парадигме рисования Windows окон является рисование с помощью фреймворков Delphi, C ++ Builder, Lazarus FreePascal и некоторые другие. Последний от предыдущих отличается бесплатностью для любого использования, поэтому остановимся именно на нем. Дистрибутив Lazarus можно скачать по ссылке http://www.lazarus-ide.org/. Инсталляция является простаивает не требует ведущих знаний к операционной системе. В Debian производных операционных системах Lazarus лучше установить из официальных репозиториев.

Окно системы программирования имеет вид:

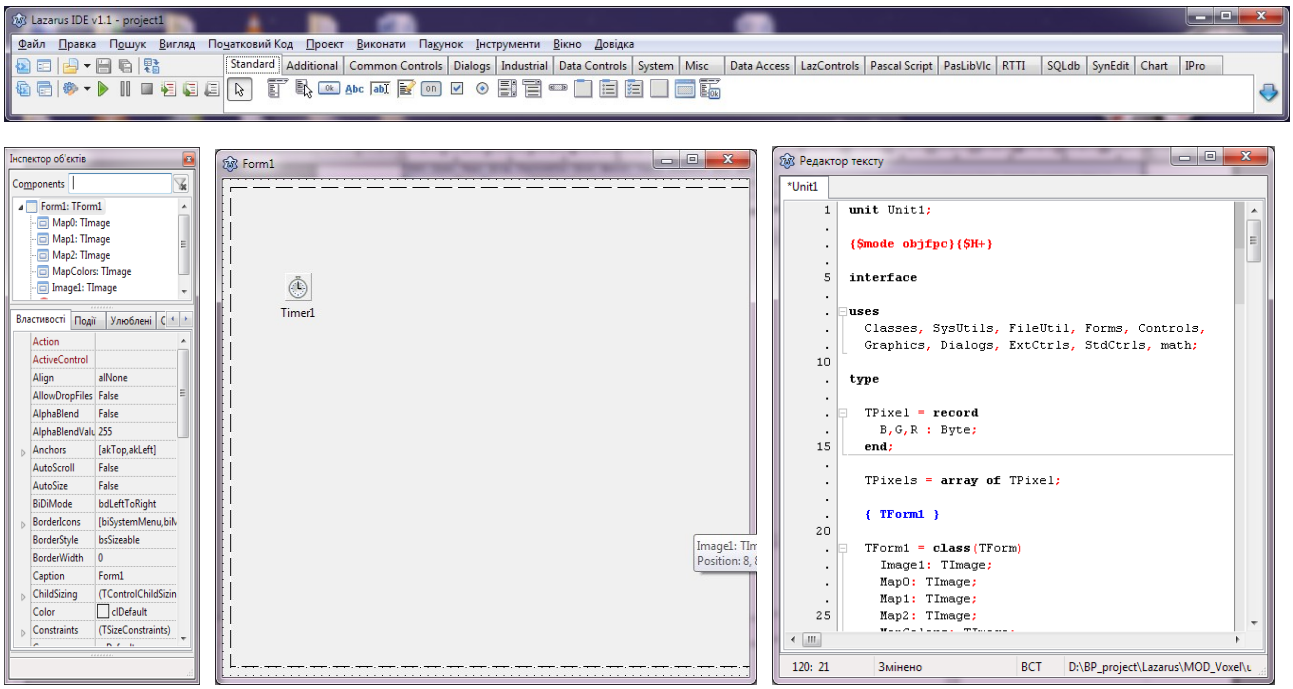

Рис. 7 - Вид среды программирования Lazarus

Далее будем считать что читатель уже известны особенности программирования на языке PASCAL или Free Pascal или Delphi. Если такого опыта нет, то можно ознакомиться с указанным

литературой:

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

Указанная литература доступна для использования на сервере кафедры ПЗИ. Для начала работы необходимо закрыть текущий проект и создать новый проект "Программа". После этого Вы увидите на экране четыре окна (рис. 7): сверху главное окно менеджера проекта с компонентами, слева менеджер компонентов проекта, по центру проект главного окна программы, справа текст программы. Первым действием меню выполняем "Сохранить все" и сохраняем составляющие проекта в отдельную папку.

Рисование будем проводить на компоненте TImage с вкладки компонентов Additional. При этом Вы сможете менять позицию и размер на проекте окна программы будущего рисунка с помощью мыши. Далее, двойным кликом мыши по свободной части окна переходим к процедуре, которая выполняется при создании окна. В этой процедуре будем выполнять команды рисования.

система координат в растровой графике несколько отличается от систем координат принятой в математике. В первую очередь координаты являются целочисленными. Во-вторых, вертикальная ось направлена вниз, то есть большее значение соответствует ниже пикселю. Точка (0; 0) соответствует верхнему левому углу рисунка.

Рисование точки-пикселя. Для рисования на компонентах используется свойство Canvas. Именно это свойство содержит все средства управления рисованием и графического изображения информации. Это свойство имеет подавляющее большинство компонентов, исключая те, за рисование которых отвечает непосредственно операционная система. Таким образом, чтобы поставить точку на рисунок с координатами (x; y) нужно записать код:

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

где x, y есть целые числа или целочисленные переменные.

Рисование отрезков. Для рисования линий нужно предварительно указать цвет этой линии и ее параметры. За эти свойства отвечает параметр Pen:

```
Image1.Canvas.Pen.Color = clRed; // Задание цвета линий Image1.Canvas.Pen.Width = 3; //
Задание толщины линий
```

Набор параметров линий значительно шире, но для начального рисования хватит только этих двух. С другими параметрами Вы можете ознакомиться с дополнительной литературы или по своим экспериментами. Рисование собственно отрезке производится с помощью команды:

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

Для дальнейших лабораторных работ, где будет проходить все более широкая растеризация векторных изображений, команд рисования линий и точек хватит. И знания дополнительных команд рисования значительно упростит создание собственных рисунков, поэтому продолжим.

Рисование прямоугольников. Для рисования прямоугольника нужно указывать не только свойства линии, но и еще параметры его окраску. Для этого свойство Canvas содержит составляющую свойство Brush - кисть. С помощью кисти можно указать не только цвет покраски но и нужно штриховки. Особенно полезным штриховки является при рисовании различных диаграмм. Следующий код демонстрирует рисования синего прямоугольника с зеленым обрамлением:

```
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)
```

Рисование эллипсов. Для рисования эллипсов используются аналогичные параметры, что и для прямоугольников. Задается эллипс прямоугольником, в который система рисования впишется эллипс, поэтому неокрашенные зоны на проектных макетах могут несколько не совпадать. Команды, которые рисуют эллипс поверх предыдущего прямоугольника показано следующим листингом:

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

Другие методы задания указанных графических примитивов или другие графические примитивы, при желании, обработайте самостоятельно. В дальнейшем умение выводить указанные примитиве позволит выполнить все обязательные задания.

Все указанные графические примитивы могут быть интерпретированы как элементы векторной графики, а их рисования - преобразование векторного изображения в растровое (растеризация). Пример рисунков составленных из графических примитивов можно посмотреть на следующем рисунке:

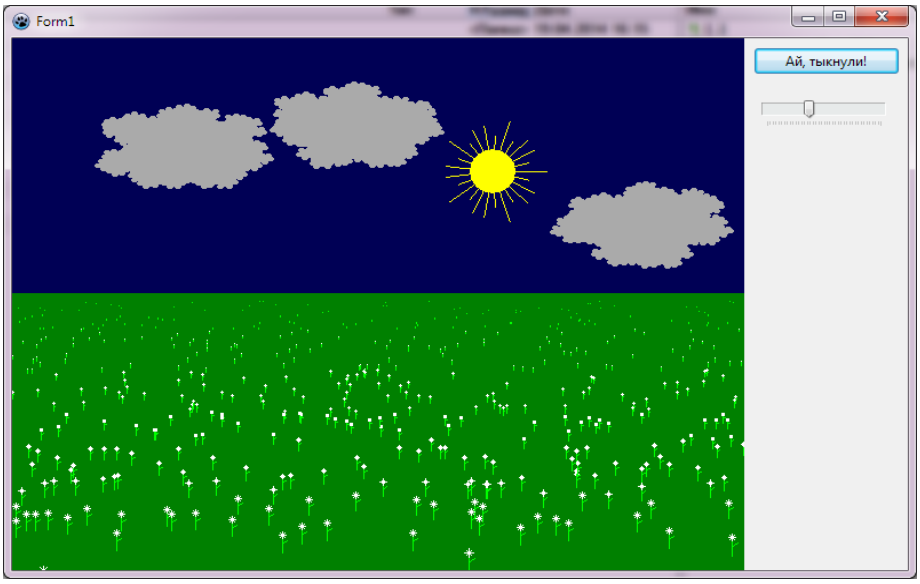

Рис. 8 - Использование графических примитивов

4. Процедурный компонентное рисования.

Часто при рисовании объекта из графических примитивов, комбинация "цветочек" используется неоднократно, поэтому боль существенно использовать функции для рисования определенной комбинации графических примитивов. Например рисования забора происходит рисованием штакетника, где есть жердь, которая повторяется до полусотни раз. Существенно, что такое рисование лучше сделать циклом, в котором жердь содержаться в теле цикла. Если же требуется рисовать нерегулярное расположение, то код рисования группы примитивов, например "птица", лучше вынести в отдельную процедуру. Эта процедура принимать желаемое положение птицы и рисовать его в указанном месте.

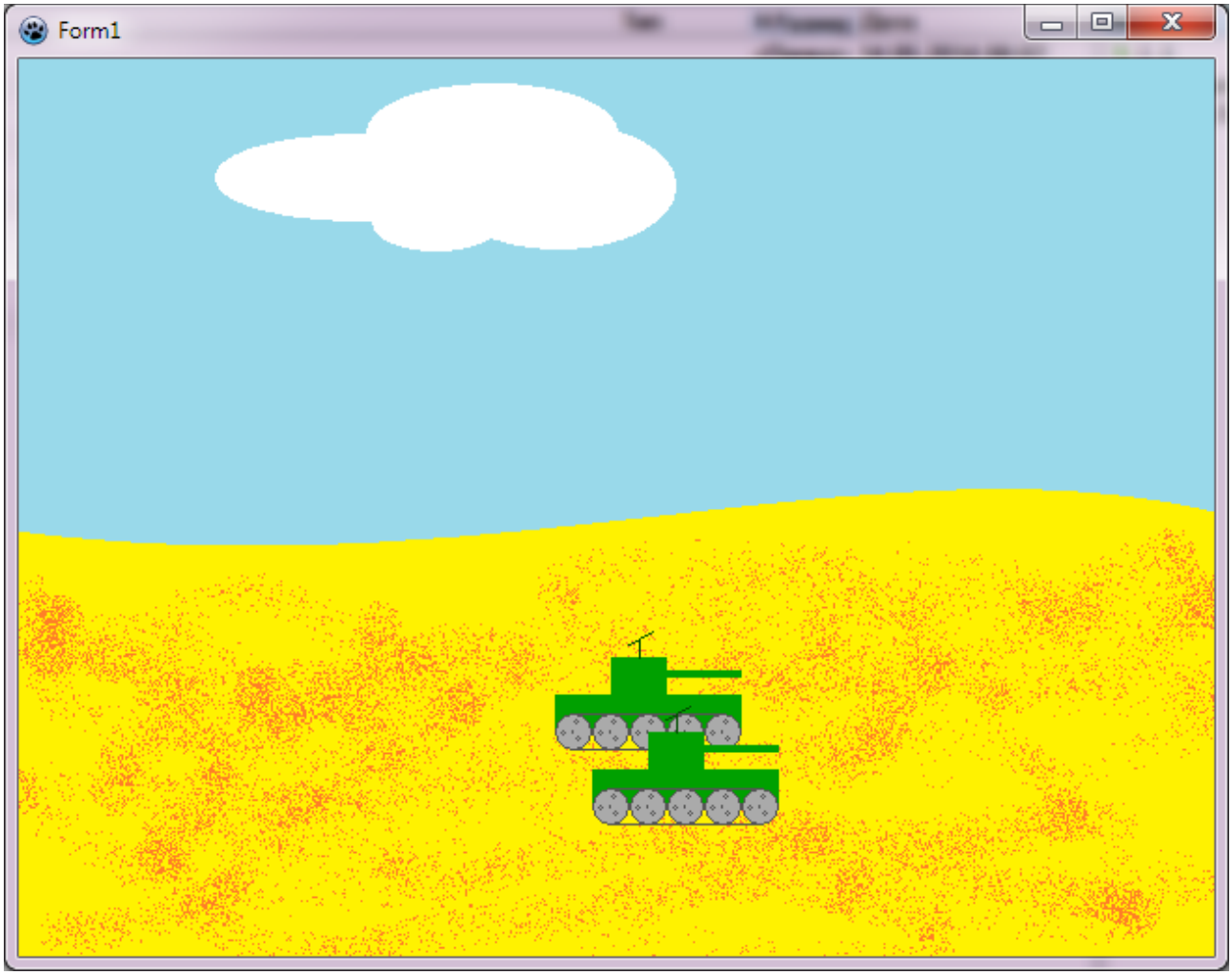

Рис. 9 - Процедурный вывод колес и корпусов танке

На рис. 9 показано рисования двух танков на разных местах. При этом каждый танк имеет по пять колес, которые рисовать отдельно вовсе не обязательно. Поэтому рисования танке реализовано отдельной процедурой, которая в свою очередь пользуется процедурой рисования колес (см. Следующий листинг):

```
    TForm1 = class (TForm)
        Image1: TImage;
. . .
    public
. . .
        kadr: Integer; // Менять для вращения колес 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 ($ 005 000)
    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 результате получен код, который использует опорные координаты, от которых начинается

строиться изображение танка. Соответственно, количество танков нарисованных на экране можно увеличить со значительно меньшими затратами на написание кода - достаточно лишь одной команды в указанной нахождением.