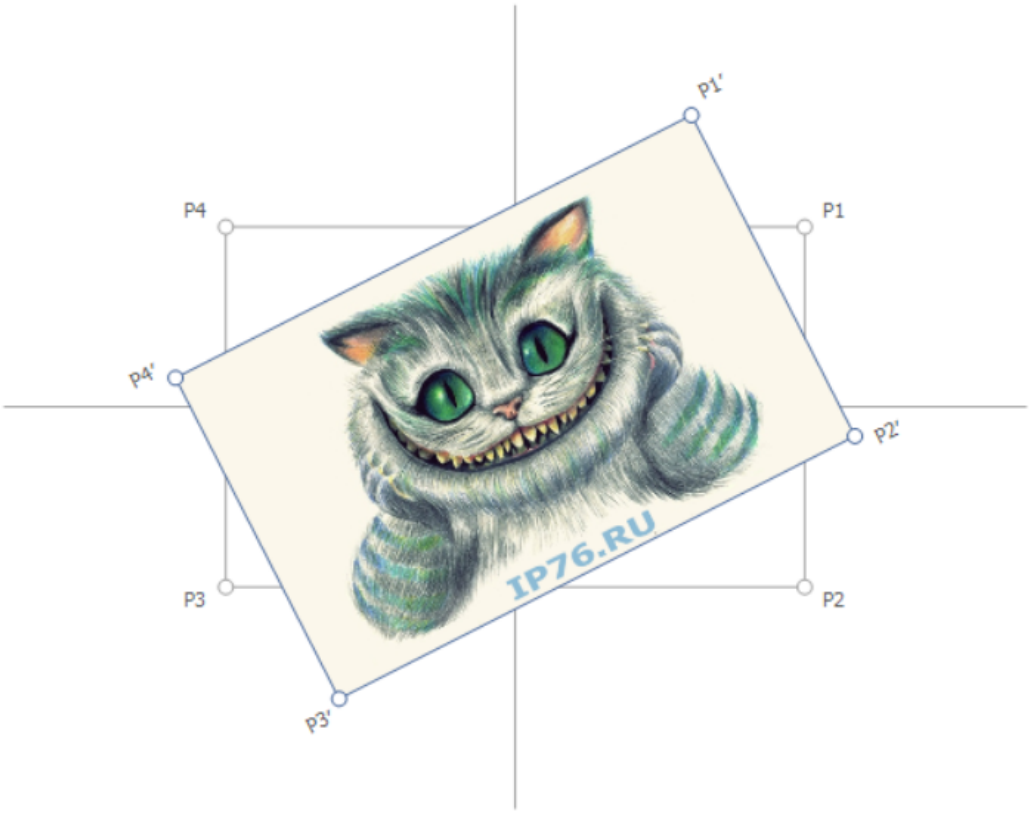
Аффинные преобразования на плоскости

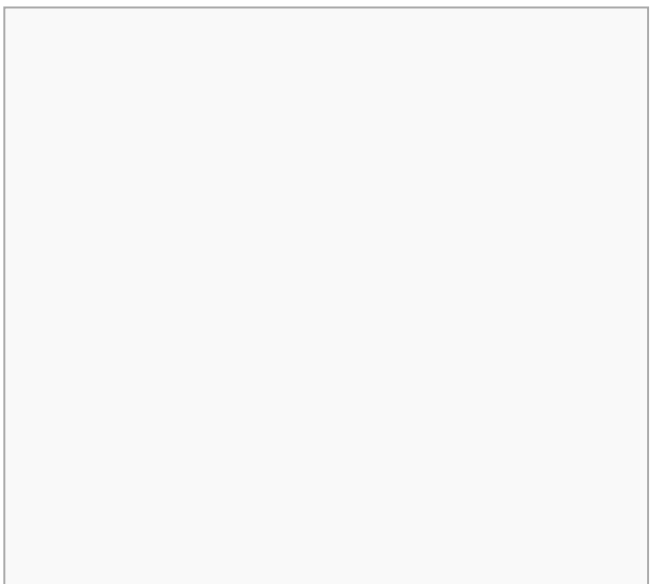


В интернете полно теории по аффинным преобразованиям. Повторяться смысла нет. Теоремы, доказательства, утверждения — это безусловно интересно, но сайт заточен на практическое применение. Поэтому больше интересует метод, а не теория.

В основе метода лежит изменение единичной матрицы с учетом матриц масштабирования, перемещения, сдвига и поворота. Эти матрицы являются **справочными** и также легко находятся в интернете.

Не смотря на простоту метода, с его применением возникают какие-то сложности, связанные, видимо, с непониманием, почему именно такие матрицы, почему важен порядок, почему это должно работать.

Хотя, как мне кажется, больше отпугивает само понятие матрицы. Давайте немного поботаним.



Содержание [скрыть]

Поворот

Предположим, стоит задача повернуть прямоугольник на некоторый угол относительно его центра. Очевидно, надо рассчитать 4 угловые точки и построить по ним полигон.

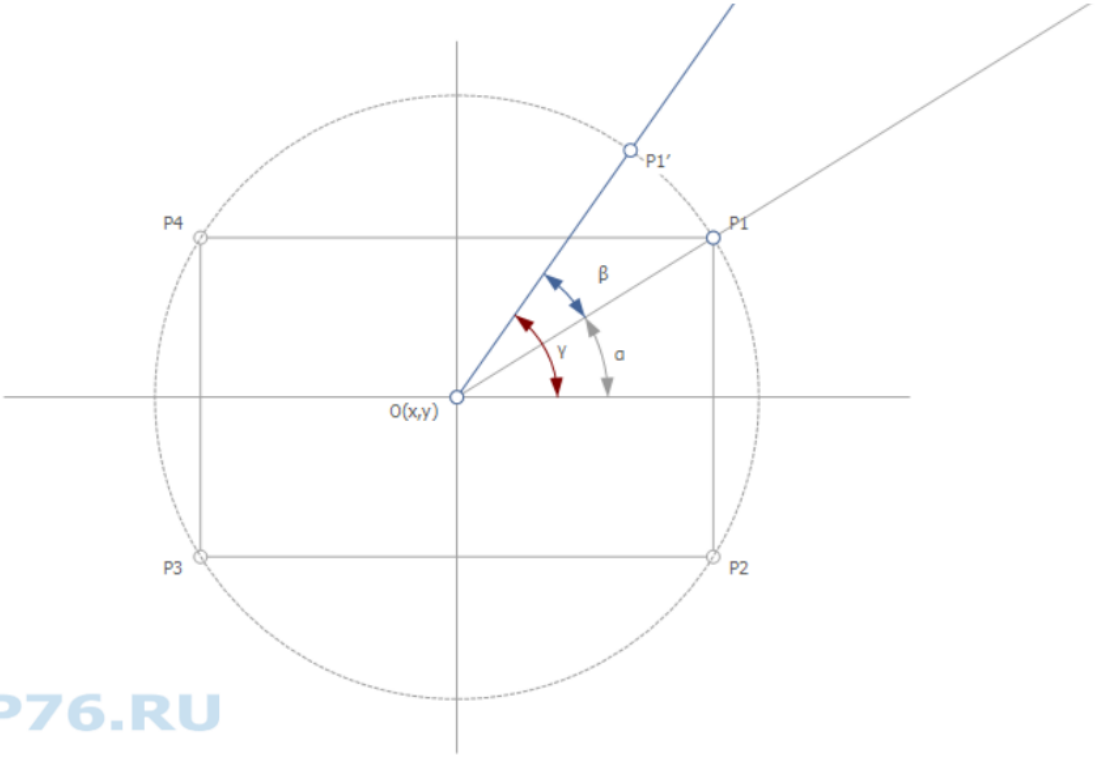


Рис.1. Поворот точки P1 на угол β

Имеем некий прямоугольник с вершинами в точках P1, P2, P3, P4. Рассмотрим точку **P1**(x,y). Она отстоит от оси абсцисс на угол **α**. Повернем ее на угол **β**. Очевидно, что вращение происходит по окружности с центром, находящимся в центре заданного прямоугольника O(x,y).

Рассчитаем координаты новой точки P1′(x′, y′).

$$x' = cos(\gamma) \times R = cos(\alpha + \beta) \times R$$
$$y' = sin(\gamma) \times R = sin(\alpha + \beta) \times R$$

Где R – радиус окружности на которой расположена точка P1, и равен **(O, P1)**. Воспользуемся формулами сложения углов (1.1 и 2.1) из **справочника**:

$$x' = (cos(\alpha) \times cos(\beta) - sin(\alpha) \times sin(\beta)) \times R$$
$$y' = (sin(\alpha) \times cos(\beta) + cos(\alpha) \times sin(\beta)) \times R$$

или

$$x' = R \times cos(\alpha) \times cos(\beta) - R \times sin(\alpha) \times sin(\beta)$$
$$y' = R \times sin(\alpha) \times cos(\beta) + R \times cos(\alpha) \times sin(\beta)$$

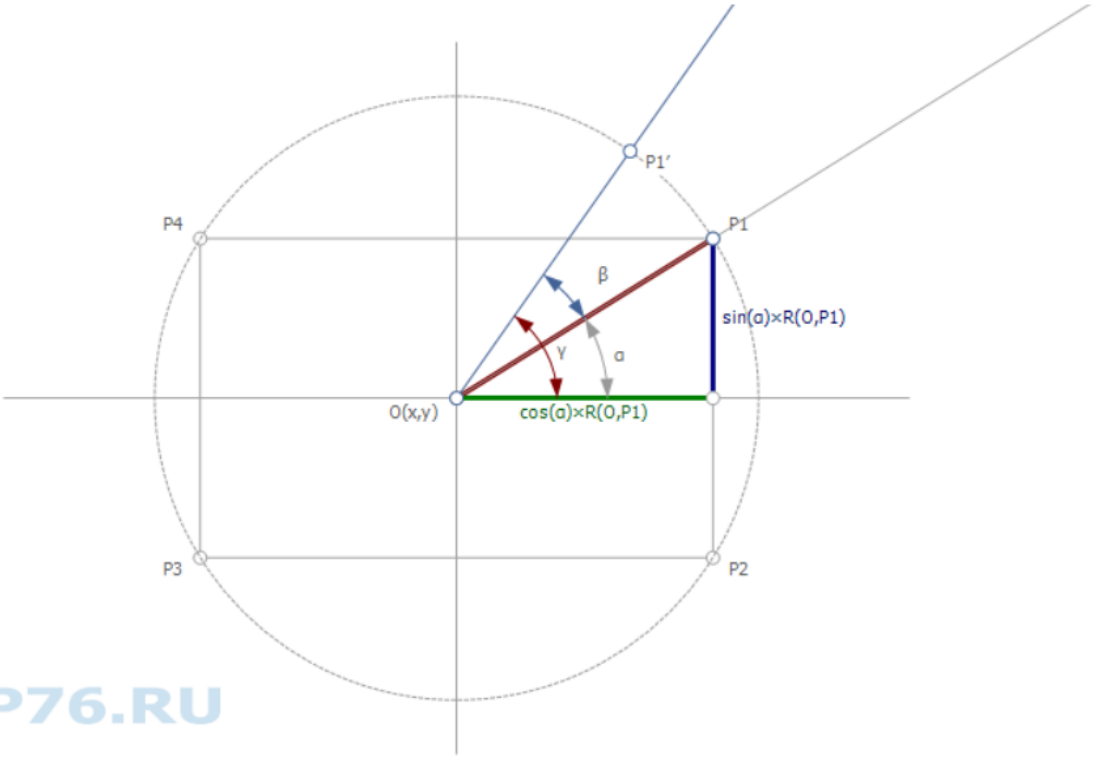


Рис.2. Координаты точки P1 через угол α

Замечаем, что R × cos(α) это не что иное, как координата **X** точки P1, а R × sin(α) – координата **Y**. Таким образом, формулы расчета координат новой точки P1′ приобретают вид:

$$x' = x \times cos(\beta) - y \times sin(\beta)$$
$$y' = x \times sin(\beta) + y \times cos(\beta)$$

Чтобы выйти на правильную позицию на холсте, прибавим к полученным значениям координаты центра прямоугольника:

$$x' = x \times cos(\beta) - y \times sin(\beta) + O(x)$$
$$y' = x \times sin(\beta) + y \times cos(\beta) + O(y)$$

И мы только что получили матрицу поворота аффинного преобразования.

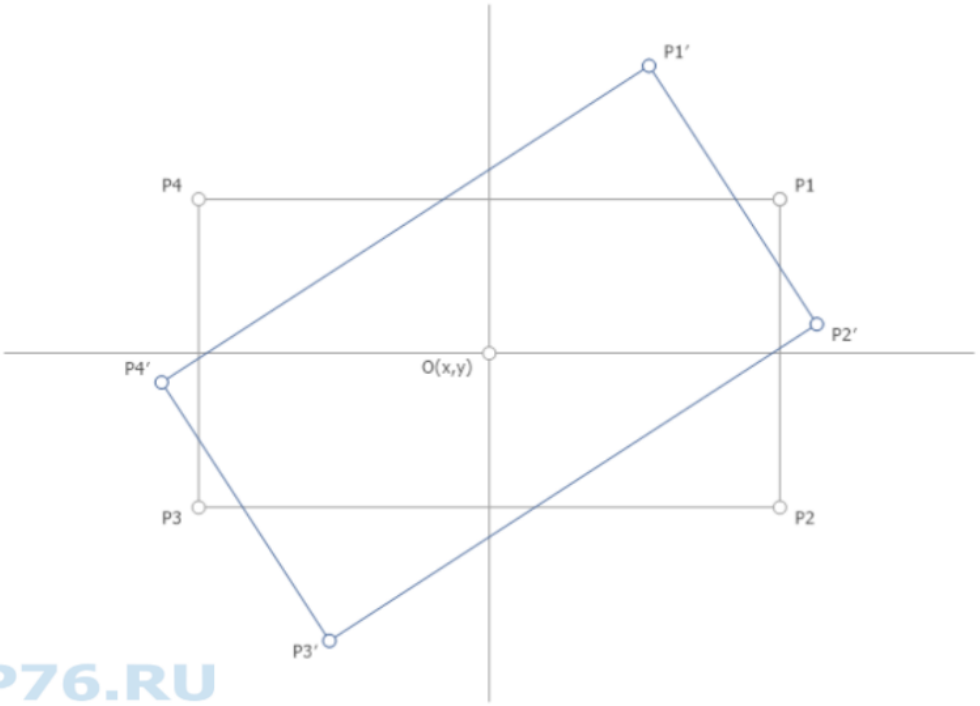


Рис.3. Поворот прямоугольника на угол β относительно его центра

В общем виде формулы можно записать как:

$$x' = x \times M11 + y \times M21 + Dxy' = x \times M12 + y \times M22 + Dy$$

В матричном виде:

$$\begin{pmatrix} x' & y' & 1 \end{pmatrix} = \begin{pmatrix} x & y & 1 \end{pmatrix} \times \begin{pmatrix} M11 & M12 & 0 \\ M21 & M22 & 0 \\ Dx & Dy & 1 \end{pmatrix}$$

Где: M11, M12, M21, M22, Dx, Dy – коэффициенты, определяющие преобразование.

Теперь представим себе, что прямоугольник – это на самом деле бесконечная плоскость. И к каждой точке этой плоскости применены одна и те же формула трансформации. Вот это и будет называться аффинным преобразованием на плоскости.

Таким образом, матрицей поворота будет следующая:

$$\begin{pmatrix} cos\ \alpha & sin\ \alpha & 0 \\ -sin\ \alpha & cos\ \alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Немного кода:

```
type
  TxPoint = packed record
```

```
    X : single;
    Y : single;
  end;

//*****************************************************************
//   Посчитать координаты точки повернутой на Angle радиан
//*****************************************************************
function CalcAnglePoint(const ACenter, APoint: TxPoint;
  const Angle: Single): TxPoint;
var
  sn,cs: single;
begin
  SinCos(Angle, sn, cs);
  Result.X := (APoint.X-ACenter.X) * cs - (APoint.Y-ACenter.Y) * sn +
    ACenter.X;
  Result.Y := (APoint.X-ACenter.X) * sn + (APoint.Y-ACenter.Y) * cs +
    ACenter.Y;
end;


```

Сдвиг

Еще одно интересное преобразование. Состоит из вертикального сдвига, когда меняется только координата Y, и горизонтального, когда меняется только X.

Для определения величин сдвигов будем использовать углы отклонения от горизонтали или вертикали, т.к. это наиболее часто встречающаяся ситуация.

Снова рассматриваем прямоугольник, помня, что это на самом деле плоскость.

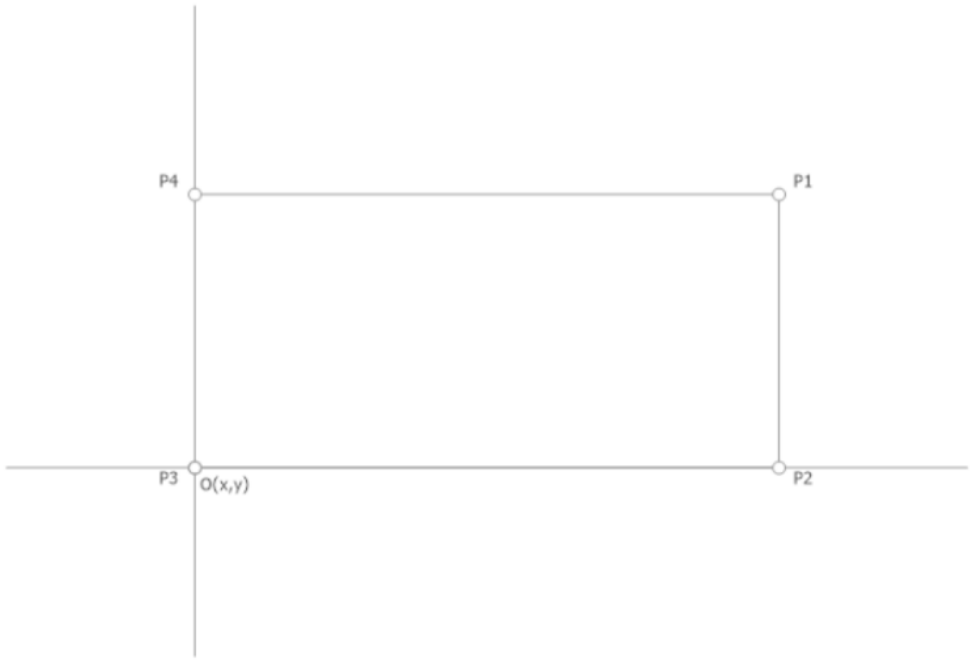


Рис.4. Плоскость с центром трансформации в левой нижней точке P3

Для наглядности деформация будет происходить относительно левой нижней точки P3.

Вначале деформируем следующим образом:

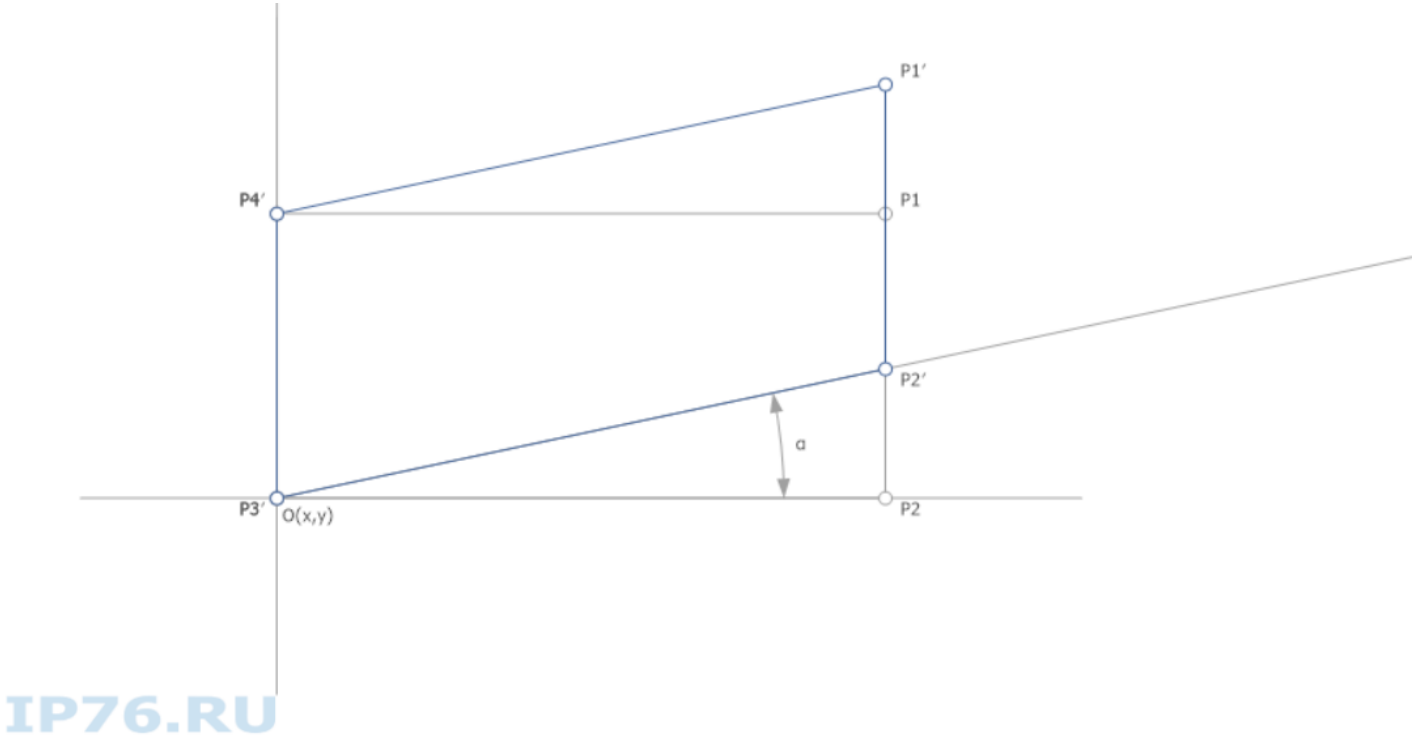


Рис.5. Вертикальный сдвиг

При вертикальном сдвиге координаты X не меняются. Изменяются координаты Y. В данном случае координата **Y** точки **P1** изменилась на расстояние **(P2,P2′)**.

$$tg(\alpha) = \frac{(P2, P2')}{(P2, O)}$$

$$(P2, P2') = (P2, O) \times tg(\alpha) = x \times tg(\alpha)$$

Из чего получим формулу преобразования для Y:

$$y' = y + x \times tg(\alpha)$$

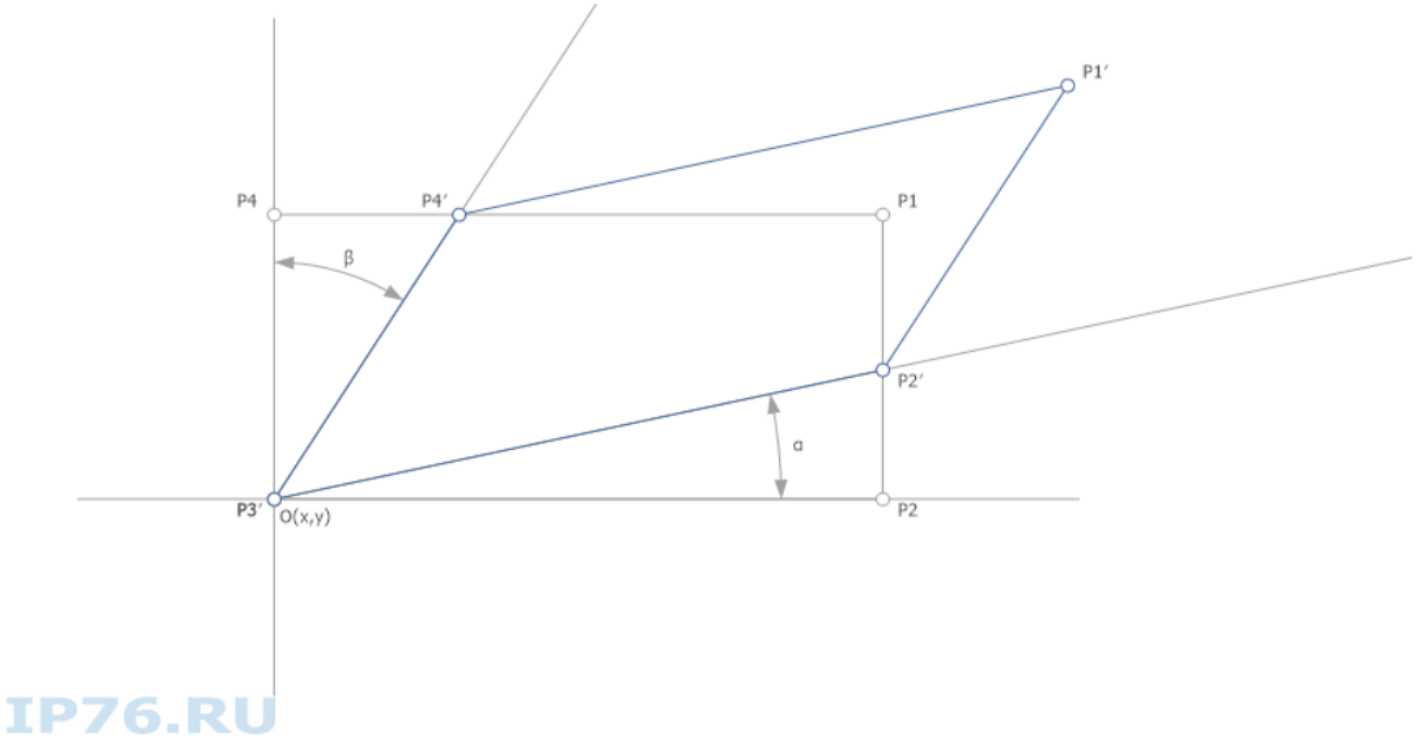


Рис.6. Вертикальный + горизонтальный сдвиги

Теперь добавим горизонтальный сдвиг. Координата **X** точки P1 изменилась на расстояние **(P4,P4')**, которое рассчитывается аналогичным образом.

В итоге, формулы для трансформации сдвига выглядят следующим образом:

$$x' = x + y \times tg(\beta)$$
$$y' = x \times tg(\alpha) + y$$

Или матрицей:

$$\begin{pmatrix} 1 & tg\ \alpha & 0 \\ tg\ \beta & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

В позиции M12 находится коэффициент, на который нужно умножить расстояние X, чтобы получить величину вертикального смещения по Y. Аналогично, в позиции M21 находится коэффициент, на который умножается расстояние Y для определения горизонтального сдвига. Это на тот случай, если не нужны углы и заранее знаем, на какие величины хотим деформировать.

Немного кода:

```
//*********************************************************************
//  Посчитать координаты точки при сдвиге с углами A и B
//*********************************************************************
function CalcShearPoint(const ACenter, APoint: TxPoint;
  const A,B: single): TxPoint;
var
  tA, tB: single;
begin
  tA := Tan(A);
  tB := Tan(B);
  Result.X := (APoint.X - ACenter.X) + (APoint.Y - ACenter.Y)*tB +
    ACenter.X;
```

```
13    Result.Y := (APoint.X - ACenter.X)*tA + (APoint.Y - ACenter.Y) +
14      ACenter.Y;
15  end;
```

Перемещение и масштабирование

Это очень простые матрицы и в свете всего вышесказанного, думаю, понятные и останавливаться на них особого смысла нет. Поэтому, для полноты картины, просто приведу.

Пермещение	Масштабирование
$x' = x + Dx$ $y' = y + Dy$	$x' = x \times Sx$ $y' = y \times Sy$
$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Dx & Dy & 1 \end{pmatrix}$	$\begin{pmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Композиция

Сразу извинюсь перед хранителями традиций. Композиция, строго говоря, не перемножение матриц, хотя бы потому, что производится в порядке, обратном стандартному перемножению. Но давайте будем честными, как ни назови, и в каком порядке производится — это умножение матриц. И у нас это будет стандартно — слева направо.

Предположим, у нас есть следующее преобразование:

$$x' = A \times x + B \times y$$
$$y' = C \times x + D \times y$$

На которое следом идет такое преобразование:

$$x'' = A' \times x' + B' \times y'$$
$$y'' = C' \times x' + D' \times y'$$

$$x'' = A' \times (A \times x + B \times y) + B' \times (C \times x + D \times y)$$
$$y'' = C' \times (A \times x + B \times y) + D' \times (C \times x + D \times y)$$

Раскроем скобки и преобразуем:

$$x'' = A' \times A \times x + A' \times B \times y + B' \times C \times x + B' \times D \times y$$
$$y'' = C' \times A \times x + C' \times B \times y + D' \times C \times x + D' \times D \times y$$

$$x'' = x \times (A' \times A + B' \times C) + y \times (A' \times B + B' \times D)$$
$$y'' = x \times (C' \times A + D' \times C) + y \times (C' \times B + D' \times D)$$

или

$$x'' = A'' \times x + B'' \times y$$
$$y'' = C'' \times x + D'' \times y$$

где

$$
\begin{aligned}
A'' &= A \times A' + C \times B' \\
B'' &= B \times A' + D \times B' \\
C'' &= A \times C' + C \times D' \\
D'' &= B \times C' + D \times D'
\end{aligned}
$$

Это ничто иное, как перемножение матриц коэффициентов:

$$
\begin{pmatrix} A'' & C'' \\ B'' & D'' \end{pmatrix} = \begin{pmatrix} A & C \\ B & D \end{pmatrix} \times \begin{pmatrix} A' & C' \\ B' & D' \end{pmatrix}
$$

Соответственно, расчет новых координат выглядит как:

$$
\begin{pmatrix} x'' & y'' \end{pmatrix} = \begin{pmatrix} x & y \end{pmatrix} \times \begin{pmatrix} A'' & C'' \\ B'' & D'' \end{pmatrix}
$$

Понятно, что можно умножить получившуюся матрицу на матрицу третьего преобразования, четвертого и т.д. Таким образом, чтобы получить коэффициенты сложного аффинного преобразования, нужно перемножить матрицы коэффициентов в порядке применения трансформаций.

Утверждение, прямо скажем, смелое, но правильное.

Проверим. Допустим, я хочу вначале применить деформацию, а потом повернуть на угол.

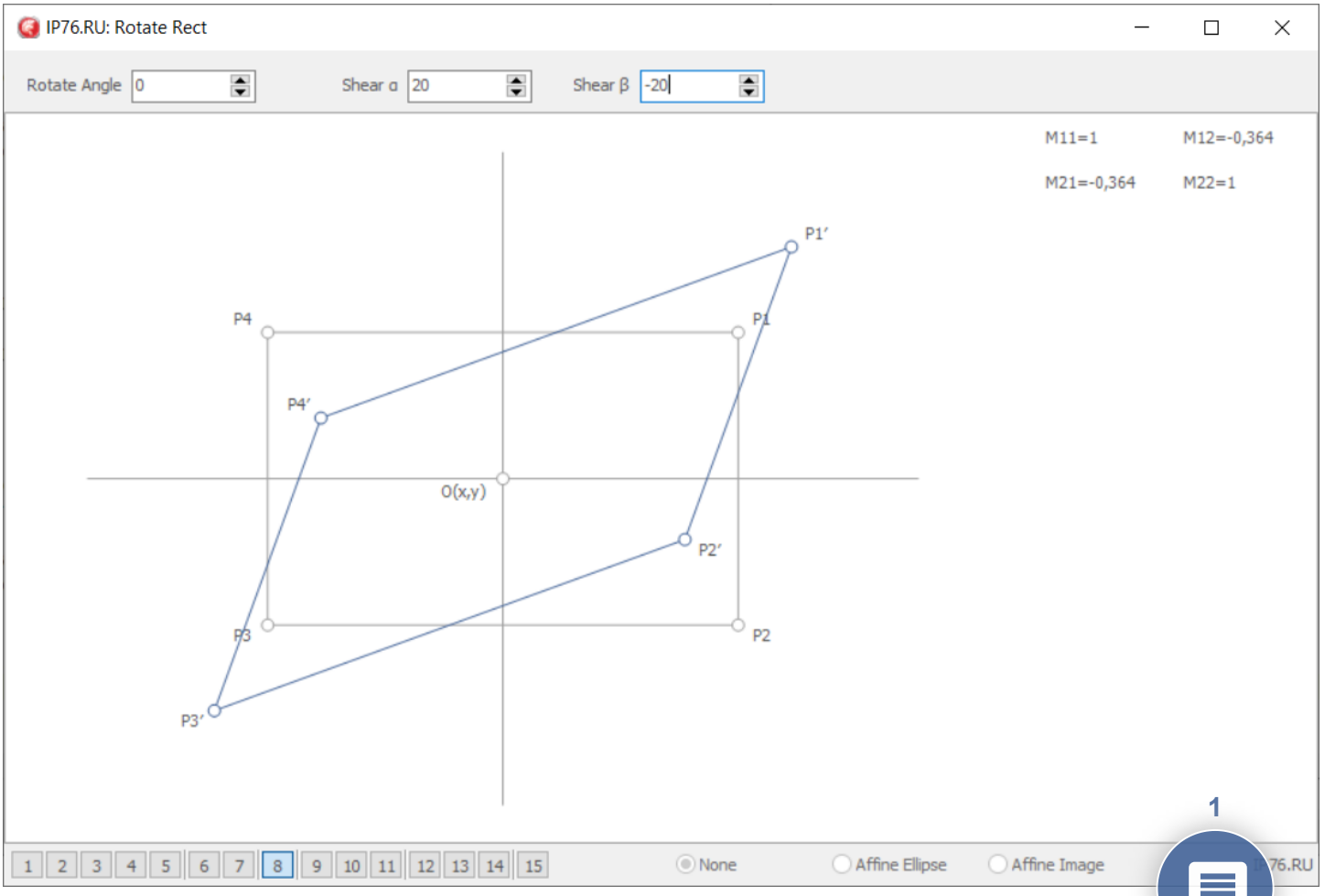


Рис.7. Деформация без поворота

На рисунке 7 пока только деформация. Теперь к деформированному прямоугольнику (плоскости) применяю поворот на 80 градусов.

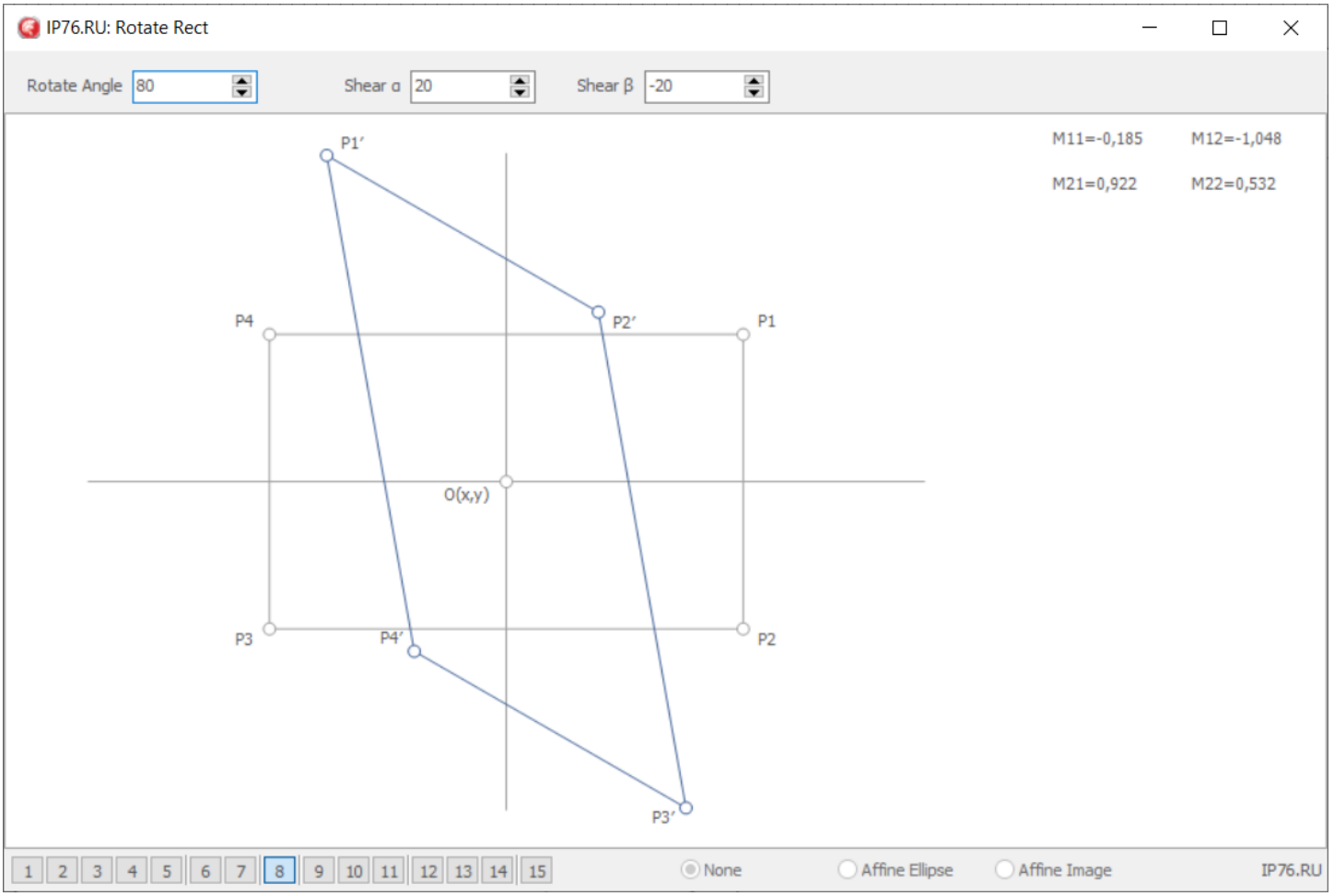


Рис.8. Деформация + поворот

Расчет координат в точности воспроизводит приведенные выше выкладки:

```
type
  T4Point = array [0..3] of TxPoint;
  T4Matrix = array [0..3] of Single;

//*********************************************************************
//  Комбинированный расчет деформации и поворота
//*********************************************************************
function CalcComboPoints (const APoints: T4Point;
  const ACenter: TxPoint; const Angle, A, B: single;
  var M: T4Matrix): T4Point;
var
  A0,B0,C0,D0: Single; // матрица деформации M0
  A1,B1,C1,D1: Single; // матрица поворота M1
  A2,B2,C2,D2: Single; // матрица M0 * M1
  sn, cs: Single;      // синус косинус Angle
  tA, tB: Single;      // тангенсы углов A, B
  i: Integer;
begin
  SinCos(Angle,sn,cs);
  tA := Tan(A);
  tB := Tan(B);
  // Деформация
  A0 := 1; B0 := tB; C0 := tA; D0 := 1;
  // Поворот
  A1 := cs; B1 := -sn; C1 :=sn; D1 :=cs;
  // Перемножение матриц M0 * M1
  A2 := A0 * A1  + C0 * B1;
  B2 := B0 * A1  + D0 * B1;
  C2 := A0 * C1  + C0 * D1;
  D2 := B0 * C1  + D0 * D1;
  M[0] := A2; M[1] := B2; M[2] := C2; M[3] := D2;
  // Расчет
```

```
33    for i := 0 to Length(APoints)-1 do
```

Применение

Аффинные преобразования изменяют геометрию плоскости, при этом сохраняя параллельность линий и соотношение расстояний.

Безусловно, это один из основных методов обработки изображений. Аффинные преобразования используются для исправления искажений и деформаций, возникающих при не самых идеальных ракурсах камеры. Широко используется в машинном обучении и компьютерном зрении.

Лично мне чаще всего попадались проблемы, которые обобщенно можно выразить двумя вопросами:

- как повернуть изображение?
- как нарисовать эллипс под углом?

Эти два вопроса объединяет одно. Изображение – набор точек, эллипс — геометрическое место точек. Решая вопрос в лоб, надо к каждой точке применить функции поворота. И если для изображения – ну, может быть, то к эллипсу, как фигуре векторной графики, это выглядит топорней некуда.

Аффинное преобразование – это преобразование плоскости. Применяя к плоскости некую трансформацию, мы понимаем, что эта трансформация применяется к каждой точке плоскости. Чтобы плоскость не содержала, это вытянется, сожмется, деформируется, повернется вместе с плоскостью.

Центр трансформации

Все что мы делали ранее, происходило относительно либо центра прямоугольника, либо одной из его вершин. Это центр трансформации. По сути – это начало координат той координатной системы, в которой происходит трансформация. Поэтому, перед всеми преобразованиями необходимо установить центр системы координат.

В Windows этим занимается функция:

```
1  function SetViewportOrgEx(DC: HDC; X, Y: Integer;
2    Point: PPoint): BOOL;
```

С этой функцией обычно идет еще набор функций, типа изменения окна вывода. Сейчас пока это все не нужно. После применения этой функции начало координат из верхнего левого угла переместиться в указанную точку. Все геометрические построения должны это учитывать. Поэтому лучше изначально писать код так, чтобы все рисовки были в относительных координатах.

Аффинное преобразование Windows

В основе работы с аффинным преобразованием лежит изменение матрицы с учетом масштабирования, перемещения, деформации или поворота. В Windows GDI за матрицу отвечает класс **TXForm**.


```
1   tagXFORM = record
2     eM11: Single;
3     eM12: Single;
4     eM21: Single;
5     eM22: Single;
6     eDx: Single;
7     eDy: Single;
8   end;
```

```
  TXForm = tagXFORM;

```

Знакомые обозначения. Поля имеют смысл ровно тот, который описан и в справочнике, и выше в статье. Это коэффициенты матрицы преобразования.

Необходимо проинициализировать поля записи и применить преобразование функцией Windows GDI:

```
function <strong>SetWorldTransform</strong>(DC: HDC; const p2: TXForm): BOOL;
```

Однако, работать с мировыми координатами и аффинными преобразованиями в Windows GDI — удовольствие так себе. Поэтому рекомендуется это делать в GDI+.

В GDI+ есть масса вариантов, сильно облегчающих жизнь при трансформации плоскости. Но сейчас в плане статьи стоит задача работать именно через матрицы преобразования, поэтому работаем следующим образом.

Необходимо создать экземпляр класса **TGPMatrix**. Затем проинициализировать его методом:

```
function TGPMatrix.<strong>SetElements</strong>(m11, m12, m21, m22, dx, dy: Single): TStatus;
```

Снова знакомые имена аргументов. Аргументы имеют тот же смысл, что и поля записи **TXForm**.

После инициализации применяем трансформацию методом **TGPGraphics**:

```
function TGPGraphics.<strong>SetTransform</strong>(matrix: TGPMatrix): TStatus;
```

После этих манипуляций рисуем все, что хотим нарисовать, не заботясь о расчетах, углах и синусах.

Представим, что один человек рисует портрет, а второй стоит сбоку и смотрит на холст. Для него портрет выглядит деформированным, а для художника, который смотрит прямо, портрет вполне себе правильный, не трансформированный и пока нравится. Что-то похожее происходит при аффинных преобразованиях. Правда, в приведенном примере трансформация перспективная. Но сути дела это не меняет.

Рекомендован следующий порядок при работе с аффинными преобразованиями:

1. Назначение нового центра координат (спорное утверждение)
2. Инициализация матрицы преобразования
3. Применение матрицы
4. Рисуем эллипс

Композиция

Как выше говорилось, при сложном преобразовании, состоящим из нескольких трансформаций, необходимо перемножить матрицы в порядке применения преобразований. За это отвечает функция Windows GDI:

```
function <strong>CombineTransform</strong>(var p1: TXForm; const p2, p3: TXForm): BOOL;
```

Здесь p2 и p3 — первая и вторая структуры TXForm, результат их комбинирования будет находиться в параметре p1. Т.е. надо понимать, что p3 – это преобразование, накладываемое на p2, т.е. идущее следом за ним. Порядок применения матриц очень важен.

Практика

Весь исходный код можно скачать по ссылке ниже. Чтобы не загромождать буду фрагментировать. Если приводить весь код в статье, до конца можно никогда не добраться.

Перед вызовом этой функции точка начала координат уже установлена в центр прямоугольника.

```
//*************************************************************
//  Рисование с использованием аффинных преобразований
//*************************************************************
procedure TFmRRMain.DrawAffine(const ACanvas: TxIPCanvas;
  const ARect: TxRect; const ACenter: TxPoint);
var
  val: TxRect;       // отображаемый прямоугольник
  pnt: TxPoint;      // центр вселенной
  mode: Integer;     // текущий режим отображения
  Mx: TGPMatrix;     // матрица GDI+ (аналог TXForm)
  RotaMx: TXForm;    // матрица поворота
  SkewMx: TXForm;    // матрица сдвига
  sn,cs: Single;     // синус и косинус угла поворота
  tA,tB: Single;     // тангенсы углов сдвига
begin
  mode := GetCurrMode;
  if mode < 9 then exit;

  SinCos(-FAngle, sn, cs);
  tA := tan(FAngleA);
  tB := tan(FAngleB);
  Mx := TGPMatrix.Create;

  try
    with ACanvas do
    begin
      // прямоугольник фигуры
      val := ARect;

      // установили центр трансформации
      pnt := xPoint(0,0);

      // рисуем координатные оси
```

Аффинный поворот

Итак, у нас есть демонстрационной приложение, написанное для статьи. Внизу есть ряд кнопок 1..15. Это разные режимы демонстрации. Режимы 9..11 отвечают за демонстрацию аффинных преобразований с предварительным назначением нового центра координат . В режиме 9 видим такую картину:

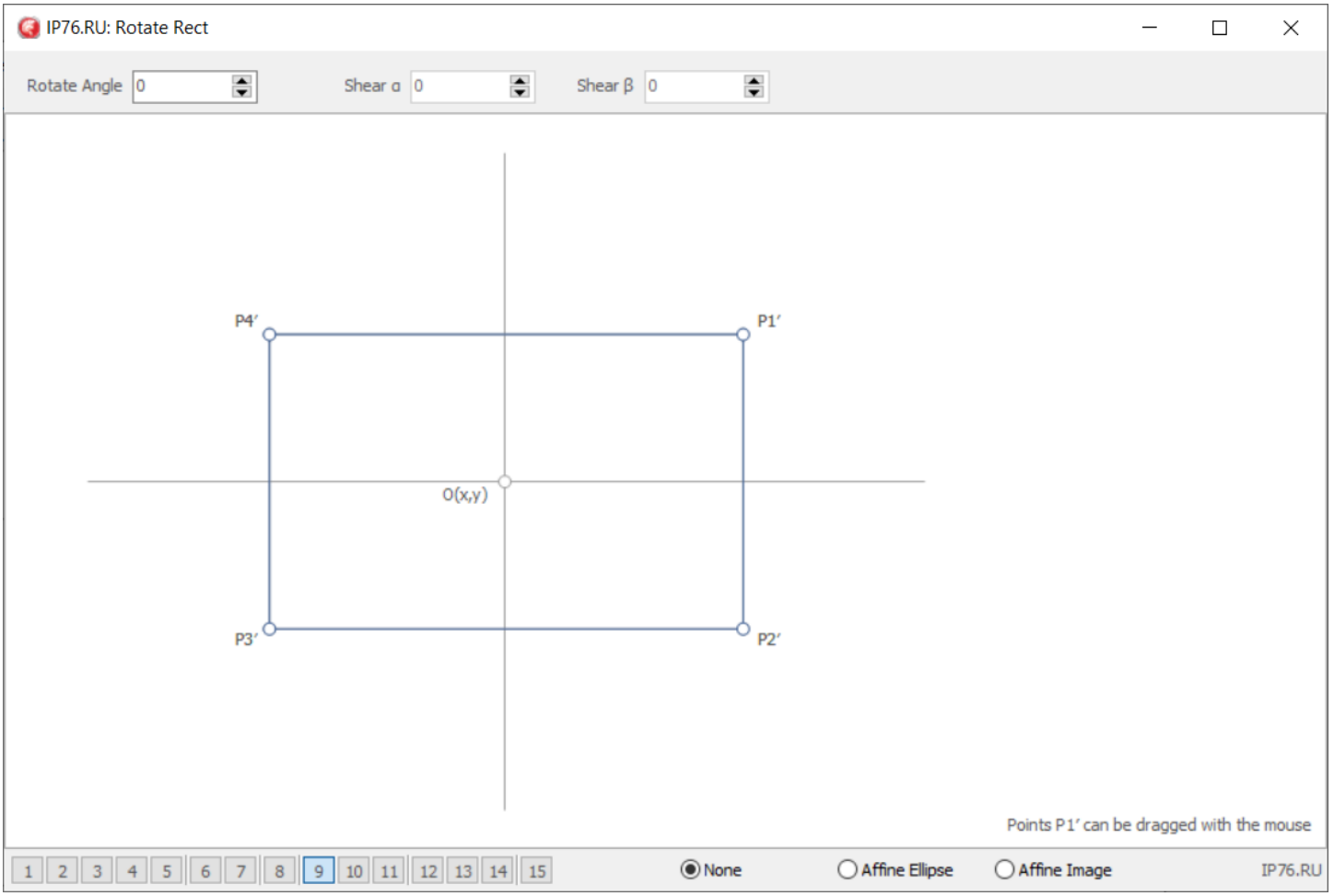


Рис.9. «Спокойное» аффинное преобразование

Ну, это мы уже видели. Однако, если повернуть прямоугольник, за вершину P1 или в поле «Rotate Angle» увидим интересное изменение:

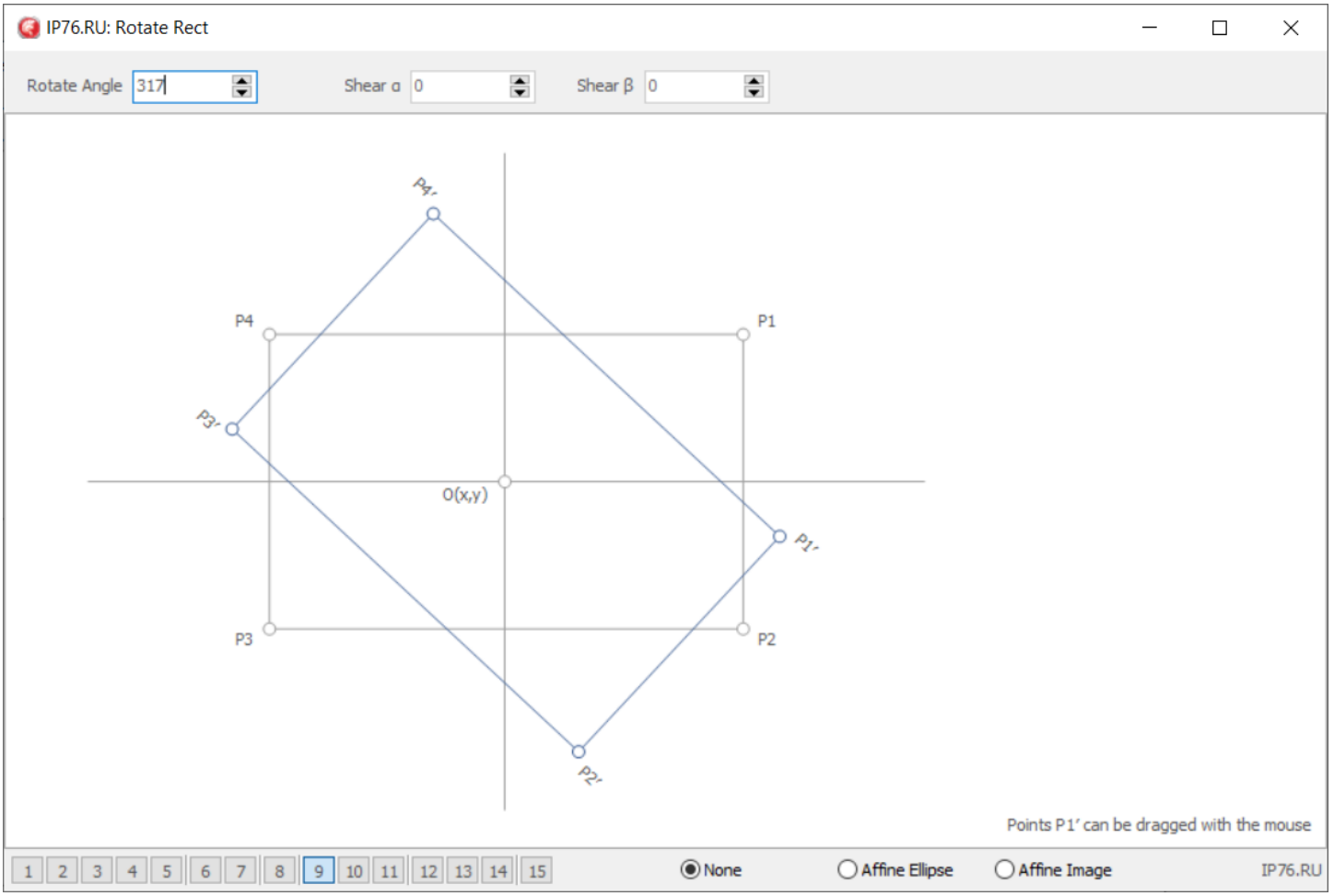


Рис.10. Подписи к вершинам тоже под углом

Помимо поворота непосредственно прямоугольника, повернулись и подписи вершин.

Преобразование применятся к плоскости, к каждой точке плоскости. Грубо говоря, преобразование применяется ко всему, что на плоскости.

Поэтому, можно предположить, что такой же фокус пройдет и с эллипсом:

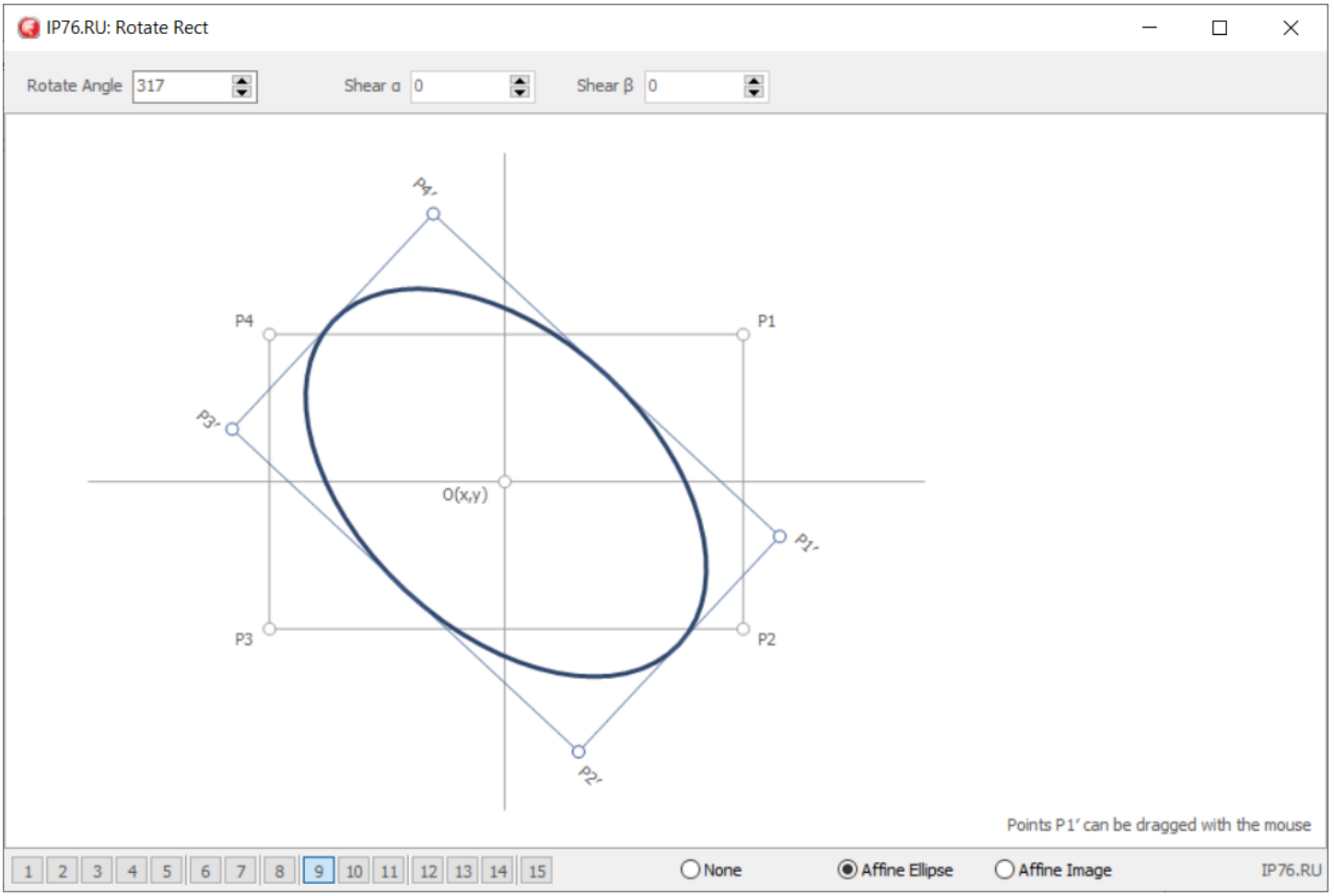


Рис.11. Эллипс под углом

То же самое касается и изображения:

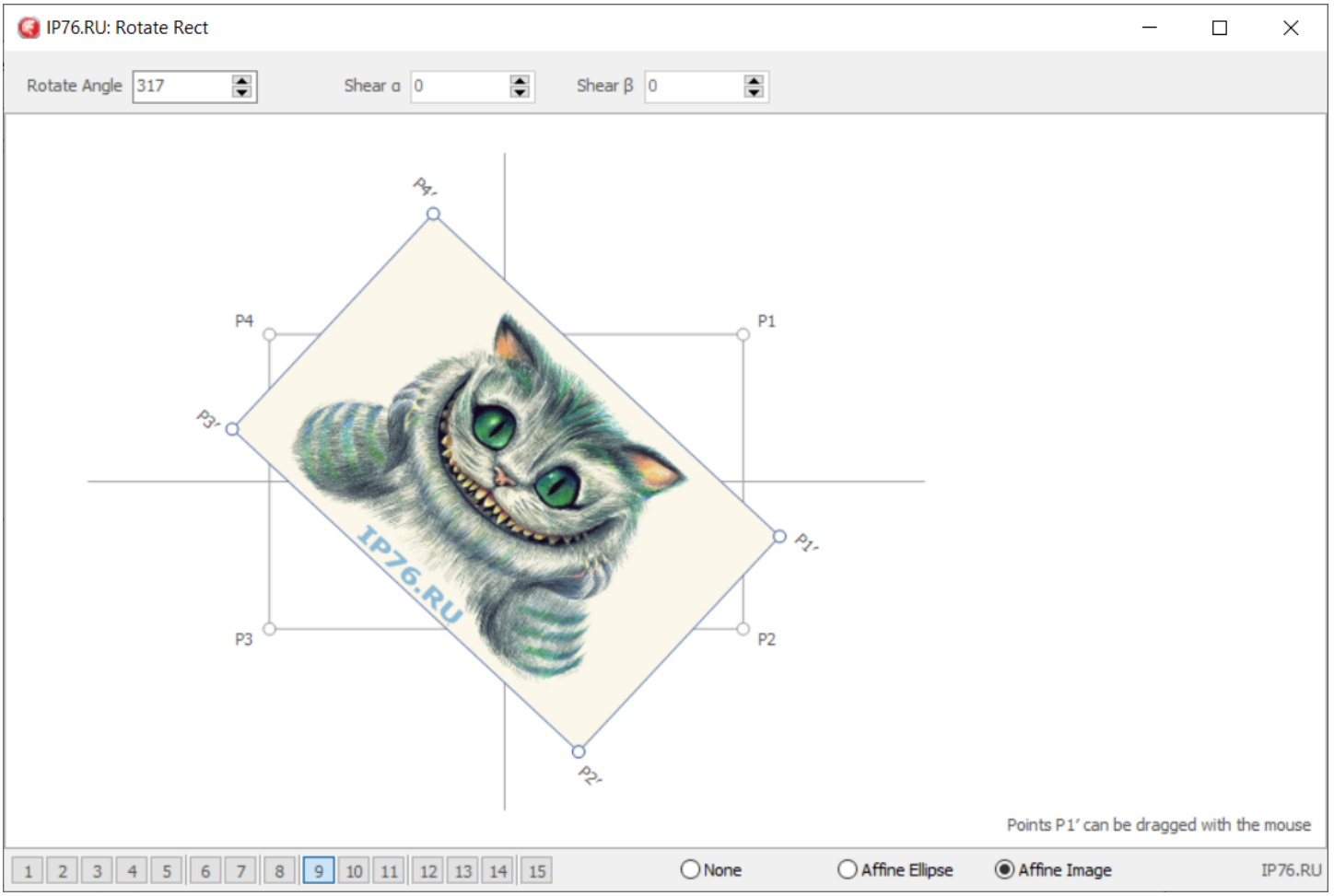


Рис.12. Изображение под углом

В чем, собственно, прелесть аффинных преобразований и состоит. Не надо считать каждую точку, писать километры кода. Мы просто рисуем, как будто и нет никаких преобразований, все также прямоугольно, параллельно и перпендикулярно. Все расчеты берет на себя плоскость, к которой применили преобразование.

Фрагмент из вышеприведенного кода:

```
      // Если есть галка - рисуем кота
      if chbAffineImage.Checked then
        StretchDraw(GPRectF(val),FImage);

      // Если есть галка - рисуем эллипс
      if chbAffineEllipse.Checked then
        DrawEllipse(ACanvas, val);

```

В функции DrawEllipse нет ничего особенного, вынесена, чтобы не загромождать код:

```
//*****************************************************************
//  Просто нарисовать "толстый" синий эллипс
//*****************************************************************
procedure DrawEllipse(const ACanvas: TxIPCanvas; const ARect: TxRect);
begin
  with Acanvas do
  begin
    Pen.Color := Darker(clIP76Color,30);
    Pen.Width := 3;
    Brush.Style := bsClear;
    Ellipse(GPRectF(ARect));
    Pen.Width := 1;
  end;
end;
```

```
15
```

T.е. не взирая на все требуемые повороты, деформации и прочие издевательства, спокойно рисуем все в том же прямоугольнике val, все теми же обычными методами.

Аффинный сдвиг

Аналогичные чудеса в режиме деформации сдвига (кнопка 10):

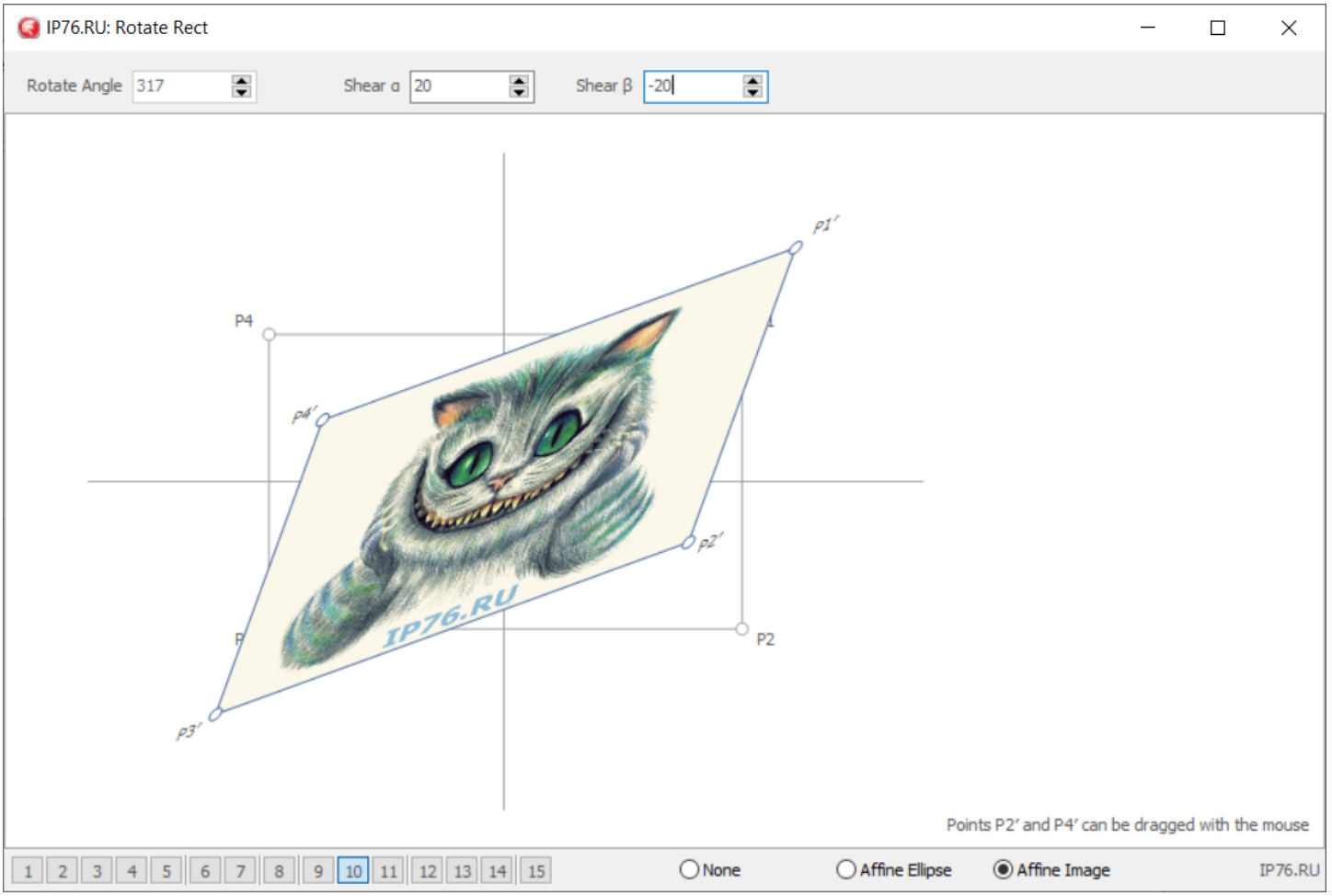


Рис.13. Поперечная деформация (сдвиг) кота

Перед рисованием инициализируем матрицы TXForm и в зависимости, от того что требуется, применяем ту или иную матрицу. Фрагмент из кода выше:

```
      // матрица поворота
      RotaMx.eM11 := cs;  RotaMx.eM12 := sn;
      RotaMx.eM21 := -sn; RotaMx.eM22 := cs;
      RotaMx.eDx  := 0;   RotaMx.eDy  := 0;

      // матрица сдвига
      SkewMx.eM11 := 1;  SkewMx.eM12 := tA;
      SkewMx.eM21 := tB; SkewMx.eM22 := 1;
      SkewMx.eDx  := 0;  SkewMx.eDy  := 0;

      // композиция матриц
      if mode in [11] then
        <strong>CombineTransform</strong>(RotaMx, SkewMx, RotaMx);

      // если поворот, либо сдвиг + поворот
      if mode in [9,11] then
        MX.SetElements(RotaMx.eM11, RotaMx.eM12,
          RotaMx.eM21, RotaMx.eM22, RotaMx.eDx, RotaMx.eDy);
```

```
19
20        // если сдвиг
21        if mode in [10] then
22          MX.SetElements(SkewMx.eM11, SkewMx.eM12,
23            SkewMx.eM21, SkewMx.eM22, SkewMx.eDx, SkewMx.eDy);
24
25        // Установить преобразование
26        GPGraphics.SetTransform(MX);
27
```

Аффинная композиция

В коде выше присутствует композиция матриц — **CombineTransform**. Последовательность преобразований такая — вначале деформация, потом поворот. Проинспектируем кнопкой 11.

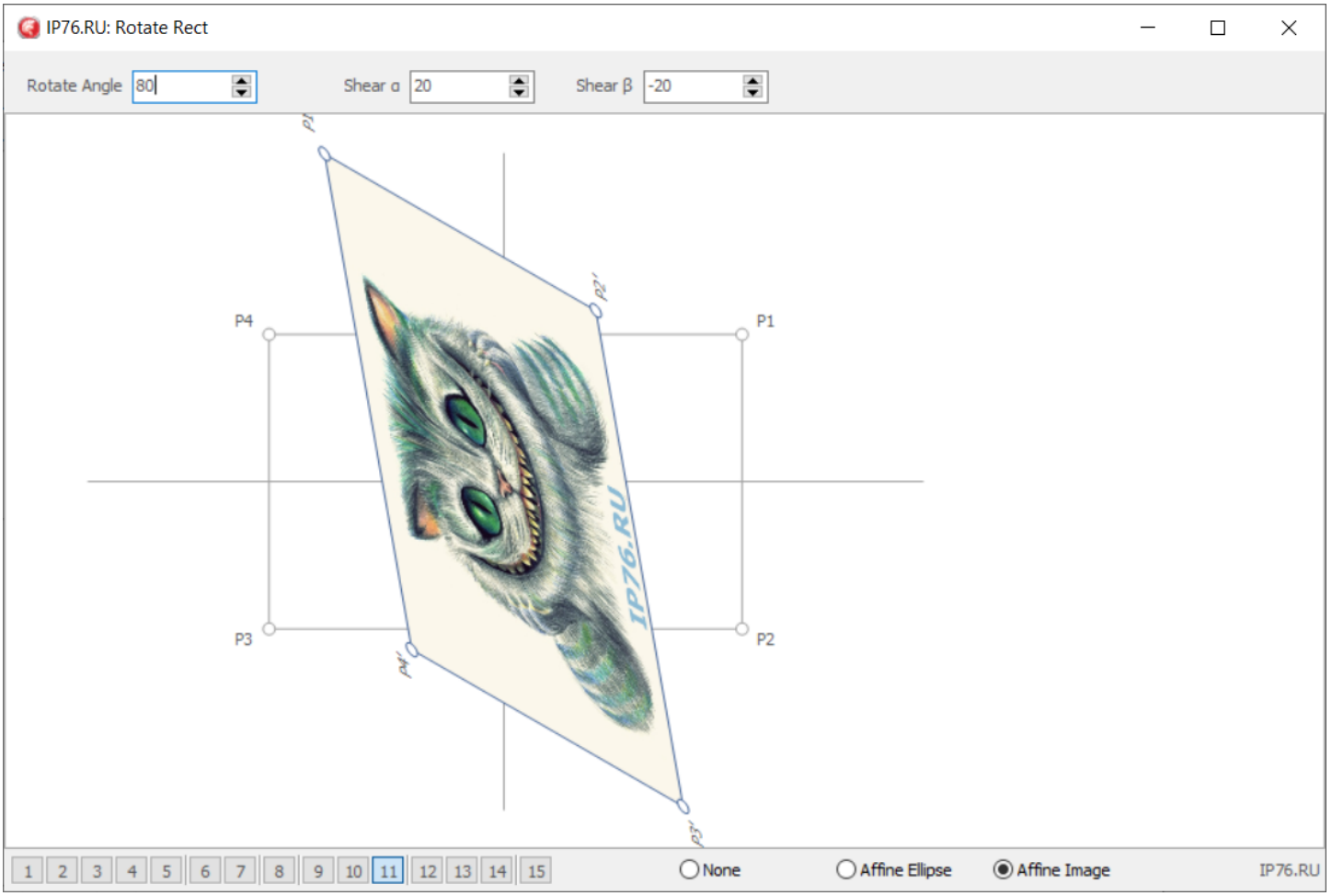


Рис.14. Деформированный, повернутый, замученный кот

Сравним с деформацией на рис.8. Там мы делали композицию матриц руками (в интерфейсе – режим 8), теперь этим занимается Windows GDI.

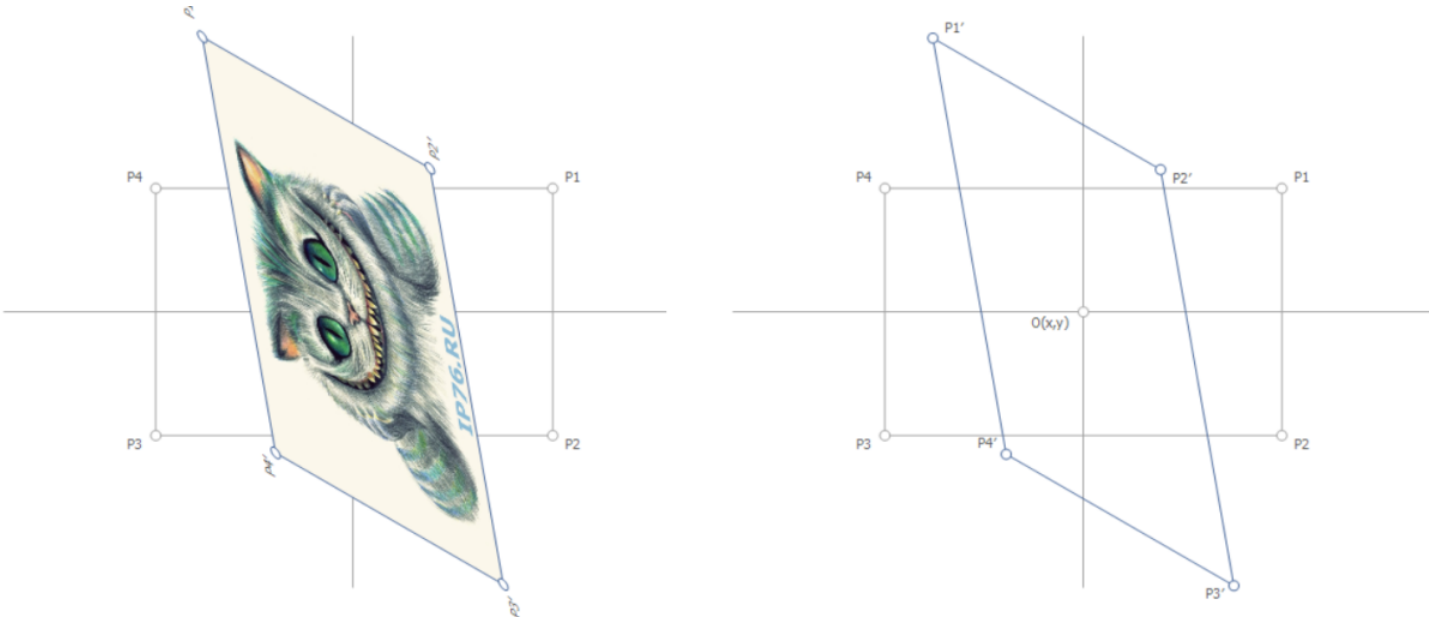


Рис.15. Все правильно делаем

Перед вызовом функции **DrawAffine**, необходимо перевести центр системы координат в центр прямоугольника. Напомню, начало системы координат – это центр трансформации. Именно вокруг него вертится вселенная.

```
try
  // находим прямоугольник пропорциональный картинке
  val := xRect(0,0,FImage.Width,FImage.Height);
  val := GetProportRect(val, rct.Width * (FRect.Right-FRect.Left),
           rct.Height * (FRect.Bottom-FRect.Top));
  // смещаем его, согласно настройкам FRect
  xOffsetRect(val, rct.Width*FRect.Left,rct.Height*FRect.Top);
  // находим центр прямоугольника - центр трансформации
  pnt := CenterRect(val);

  // работаем с аффинным преобразованием
  // устанавливаем центр прямоугольника как начало координат
  if mode in [9,10,11]  then
    <strong>SetViewportOrgEx</strong>(bmp.Canvas.Handle,
      Trunc(pnt.X), Trunc(pnt.Y), nil);

  with cnv do
  begin
    ...
    rct := ClipRect;
    // Если центр посредине, надо сместить пространство
    if mode in [9,10,11]  then
      <strong>OffsetRect</strong>(rct, -Trunc(pnt.X), -Trunc(pnt.y));
    ...
    // Инициализация точек для мыши
    FZeroPoint := CalcAnglePoint (pnt,
      xPoint(val.Right, val.Top),  -FAngle);
    FPointA := CalcShearPoint(pnt,
      xPoint(val.Right, val.Bottom),FAngleA,FAngleB);
    FPointB := CalcShearPoint(pnt,
      xPoint(val.Left,  val.Top),   FAngleA, FAngleB);

    FCenterPoint := pnt;
```

Это фрагмент обработчика OnPaint того PaintBox, на котором рисуем . Вначале идет вызов **SetViewportOrgEx**, переводит начало координат в центр прямоугольника. Потом происходит коррекция прямоугольника области рисования **OffsetRect**(rct, -Trunc(pnt.X), -Trunc(pnt.y)), иначе все координаты «уплывут» вправо на величину (pnt.x, pnt.y), затем вызов **DrawAffine.**

Если б мышь знала… Координаты вершин

В приложении есть возможность таскать за некоторые вершины. Тут возникает проблема. Если применяем аффинное преобразование, координаты точек, за которые можно «ухватиться» сильно отличаются от координат в «нормальном» состоянии координатной системы. Поэтому, перед тем как плоскость начнет изменяться, необходимо рассчитать нужные точки, т.е. координаты, понятные мышке.

```
      // Инициализация точек для мыши
      FZeroPoint := CalcAnglePoint (pnt,
        xPoint(val.Right, val.Top),  -FAngle);
      FPointA := CalcShearPoint(pnt,
        xPoint(val.Right, val.Bottom),FAngleA,FAngleB);
      FPointB := CalcShearPoint(pnt,
        xPoint(val.Left,  val.Top),   FAngleA, FAngleB);

      FCenterPoint := pnt;

      if mode in [10,13] then
        // Режим, когда прямоугольник по центру и таскаем за p2 и p4
        FPointC := CalcShearPoint(pnt,
          xPoint(val.Left, val.Bottom),FAngleA, FAngleB)
      else
        // Во всех остальных случаях, опорная точка - центр
        FPointC := FCenterPoint;

```

И только потом

```
xOffsetRect(val, -Trunc(pnt.X), -Trunc(pnt.y));
```

И только после этого

```
DrawAffine(cnv, FCurrRect, FCenterPoint)
```

Функции расчета достались в наследство от предыдущих этапов, когда экспериментировали над прямоугольником без применения аффинных преобразования. Не зря ж в конечном счете экспериментировали.

Конечно, можно сформировать и скомбинировать матрицы, взять коэффициенты и посчитать по формуле. Почти как в CalcComboPoints.

В GDI+ в TGPMatrix для этих целей существуют методы:

```
function TGPMatrix.TransformPoints(pts: PGPPointF;
  count: Integer = 1): TStatus;

function TGPMatrix.TransformPoints(pts: PGPPoint;
  count: Integer = 1): TStatus;

```

Здесь они не используются. Потом как-нибудь обязательно вернемся к ним.

Смещать ли начало координат?

Теоретически рекомендуется делать именно так, смещать и потом трансформировать. И, возможно, это правильно. Но можно обойтись и без этого. Просто привычней работать в «обычной» системе координат, когда точка (0,0) системы находится в левом верхнем углу.

По сути, как это делается, мы уже рассмотрели в самом начале статьи, когда вращали прямоугольник, считали координаты вершин и смещали в нужное место. Давайте это сделаем с помощью аффинных преобразований.

Порядок действий таков:

1. Определяемся с прямоугольником, в котором что-то хотим нарисовать
2. Определяемся с центром трансформации (pnt), т.е. точкой, вокруг которой будет происходить трансформация
3. Смещаем прямоугольник на -pnt.x, -pnt.y, тем самым совмещая центр трансформации с началом координат
4. Формируем матрицу трансформации
5. Добавляем матрицу перемещения с параметрами (или просто инициализируем в текущей матрице) eDx=pnt.x, eDy=pnt.y
6. Рисуем в получившемся смещенном прямоугольнике

```
//*************************************************************
// Нарисовать прямоугольник под углом, без перевода начала координат
// Если указано, рисовать AImage или эллипс
//*************************************************************
procedure DrawRectAffineRotate(const ACanvas: TxIPCanvas;
  const ARect: TxRect; const ACenter: TxPoint; const Angle: Single;
  const ADrawEllipse, ADrawPicture: Boolean;
  const AImage: TGPImage = nil);
var
  MX: TGPMatrix;  // матрица преобразования
  rct: TxRect;    // прямоугольник вывода
  sn, cs: Single; // синус косинус заданного угла
begin
  SinCos(Angle, sn, cs);
  Mx := TGPMatrix.Create;
  rct := ARect;

  try
    with ACanvas do
    begin
      // сместим прямоугольник, чтобы его центр совпал с началом
      xOffsetRect(rct, <strong>-ACenter.X, -ACenter.Y</strong>);
      // инициализация преобразования
      MX.SetElements(cs, sn, -sn, cs, <strong>ACenter.x, ACenter.y</strong>);
      // Установить преобразование
      GPGraphics.SetTransform(MX);
      // Если указано - рисуем изображение
      if ADrawPicture and Assigned(AImage) then
        StretchDraw(GPRectF(rct), AImage);
      // Если указано - рисуем эллипс
      if ADrawEllipse then
        DrawEllipse(ACanvas, rct);
    end;
```

Сильно сократил код за счет того, что не использую TXForm, а инициализирую сразу TGPMatrix. Что тут изменилось. В матрицу поворота добавил еще смещение в точку (**ACenter.x, ACenter.y**) – центр прямоугольника, или центр трансформации. Перед рисованием сместил прямоугольник отрисовки так, чтобы он оказался центрирован по началу координат, т.е. точки (0,0). Аффинное преобразование отработает поворот и сместиться на указанные величины по x и y.

Т.е. произойдет ровно тоже самое, что мы делали руками, когда прибавляли координаты центра к расчетным значениям вершин.

В демонстрационном примере это режимы 12..14, которые визуально будут совпадать с 9..11, с той лишь разницей, что перевода центра координат не происходит.

Чтобы продемонстрировать, работоспособность метода, существует последний режим – 15. Помимо основного прямоугольника, рисуется звезда и текст под углом, обратным текущему. Текст в дополнение к повороту также подвергается масштабированию.

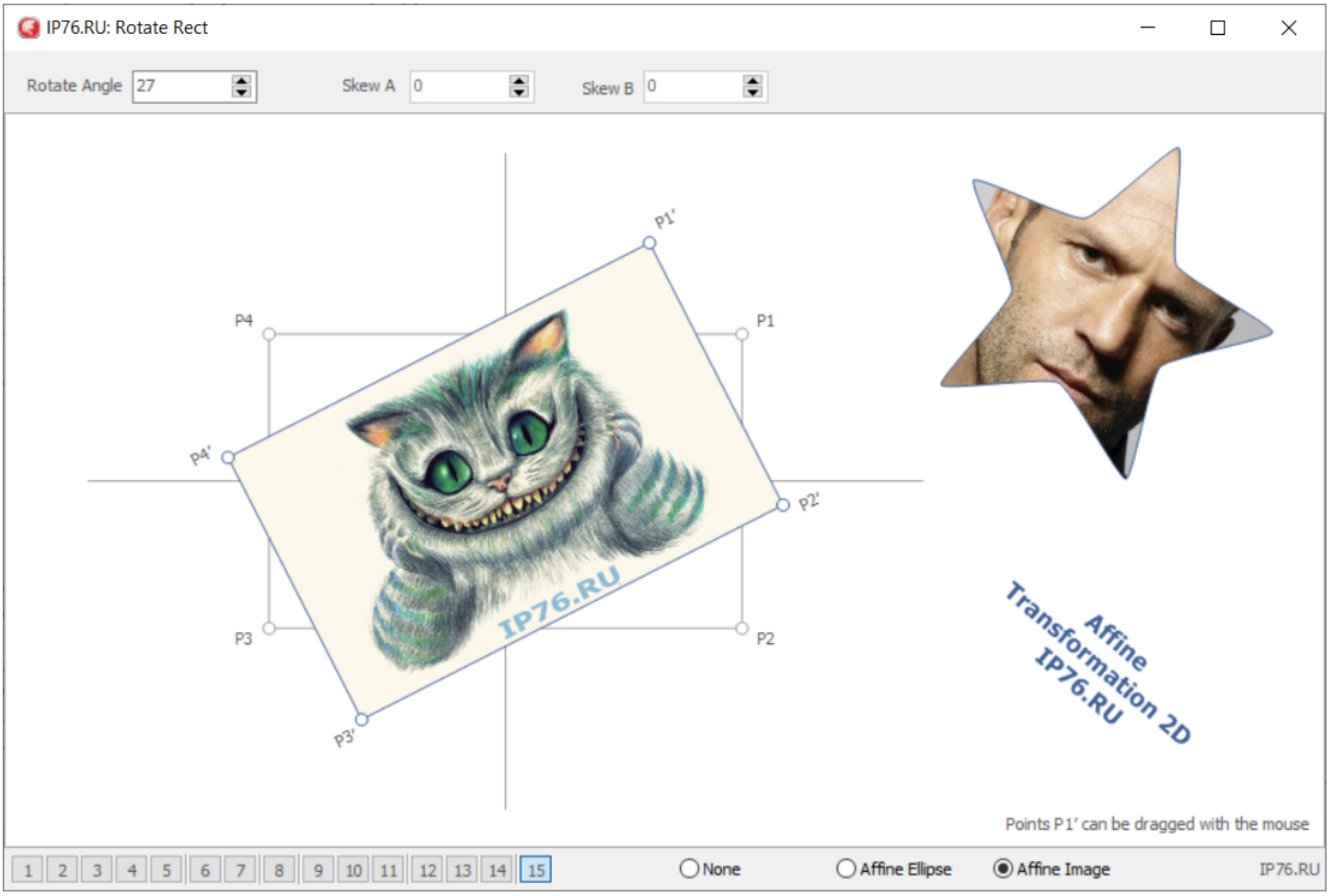


Рис.16. Аффинные преобразования в любой требуемой точке плоскости

Я грозился вернуться к теме поворота звезд. Говорил, что есть более правильные методы. Брутальная звезда имени Стетхама, помните? Так вот, наиболее правильным будет вращать звезды, и не только, через аффинное преобразование.

Процедура, отвечающая за рисовку всего этого хозяйства:

```
procedure TFmRRMain.DrawAffineStars(const ACanvas: TxIPCanvas;
  const ARect: TxRect; const ACenter: TxPoint);
var
  val: TxRect;   // отображаемый прямоугольник
  pnt: TxPoint;  // центр вселенной
begin
  with ACanvas do
  begin
    // центр координат для прямоугольника
    val := ARect;
    pnt := ACenter;
    // рисуем координатные оси
    DrawCoordSystem(ACanvas, val, pnt);
    // рисуем исходный прямоугольник с точками
    DrawRectPoints(ACanvas, val, True, False, clBtnShadow);

    // Рисуем основной прямоугольник (картинка, эллипс)
    DrawRectAffineRotate(ACanvas, val, pnt, -FAngle,
      chbAffineEllipse.Checked, chbAffineImage.Checked,
      FImage.GPImage);
    // сместить прямоугольник на начало сист.координат
    xOffsetRect(val, -pnt.X, -pnt.Y);
    // рисуем описывающий прямоугольник
    DrawRectPoints(ACanvas, val, True, True, clIP76Color);

    // прямоугольник под звезду
```

```
    val := ARect;
    xOffsetRect(val, ARect.Right - 90, 0);
    val.Top := ClipRect.Top + 10;
    val.Right := ClipRect.Right;
    val.Bottom := ARect.Top + xHeightRect(ARect) / 2;
    // Рисуем звезду
    DrawStarAffineRotate(ACanvas, val, FAngle, FStarImage.GPImage);
```

Как видно, не очень много, и в принципе, просто.

Процедура для рисовки текста, чуть посложнее. Добавил к повороту еще и преобразование масштаба.

```
//*****************************************************************
// Нарисовать текст под углом, без перевода начала координат
//*****************************************************************
procedure DrawTextAffineRotate(const ACanvas: TxIPCanvas;
  const ARect: TxRect; const Angle, Scale: Single);
const
  ROTA_TEXT = 'Affine'+#13#10+'Transformation 2D'+#13#10+'IP76.RU';
var
  MX: TGPMatrix;  // матрица преобразования
  MXS: TGPMatrix; // матрица преобразования - масштаб
  rct: TxRect;    // прямоугольник вывода
  pnt: TxPoint;   // центр
  sn, cs: Single; // синус косинус заданного угла
begin
  SinCos(Angle, sn, cs);
  MX := TGPMatrix.Create;
  MXS := TGPMatrix.Create;

  rct := ARect;
  pnt := CenterRect(rct);

  try
    // инициализация вращения
    MX.SetElements(cs, sn, -sn, cs, pnt.x,pnt.y);
    // инициализация масштаба
    MXS.SetElements(Scale, 0, 0, Scale, 0, 0);
    // добавить вращение после масштаба
    MXS.<strong>Multiply</strong>(MX, <strong>MatrixOrderAppend</strong>);

    with ACanvas do
    begin
      // инициализация шрифта
      Font.Size := 18;
```

Как видим, матрицы TXForm не используются. Чтобы добавить преобразование в GDI+ используется метод TGPMatrix:

```
function TGPMatrix.Multiply(matrix: TGPMatrix;
  order: TMatrixOrder = MatrixOrderPrepend): TStatus;
```

Суть такая же, как **CombineTransform** для Windows GDI. В листинге вначале происходит масштаб, потом поворот.

Видео + скачать

На этом пока все.

Друзья, спасибо за внимание!

Подписывайтесь на **телеграм-канал**.

Оцените мой труд, жду ваших комментариев.

Скачать (1 036 Кб): **Исходники** (Delphi XE 7-10)

Скачать (1 843 Кб): **Исполняемый файл**

Скачать (4 205 Кб): **Affine2D**

Небольшой комментарий к Affine2D. Если подвести курсор к верхнему правому углу поля отрисовки, появится знак меню. Нажмите его и настройте надписи, картинки, и т.д., по своему усмотрению.

Присоединиться к обсуждению

1 КОММЕНТАРИЙ

Старые

Елена

4 месяцев назад

Понравилась статья- ничего лишнего, все по делу. Спасибо большое за ваш труд. С нетерпением жду вашу следующую статью.

4 Ответить