Geometria Computacional 2D

Ole Peter Smith, IME, UFG, ole@mat.ufg.br

IX Semana da Educação Matemática
Cidade de Goiás-GO
21/05/2011

TI**A**MTOWTDI:
There is **Always** More Than One Way To Do It!
Larry Wall

Introdução

- Quero desenhar Geometria
- Geometria Analítica
- Álgebra Linear
- Geometria Diferencial
- 2D - caminhando pra 3D
- Matemáticos e Programação

Uma figura diz mais do que milhares de palavras...

Ferramentas - Software Livre

- Livre para: Usar - Ver - Modificar - Redistribuir o código
- Ferramentas:
  Biblioteca Gráfica: GD
  http://www.libgd.org
  Linguagem do Programação:
  PHP, C/C++, Perl,...
- Formatos Gráficos: GIF, PNG
- Animações: GIF animado
- Usando PHP: Apache, CGI, PHP

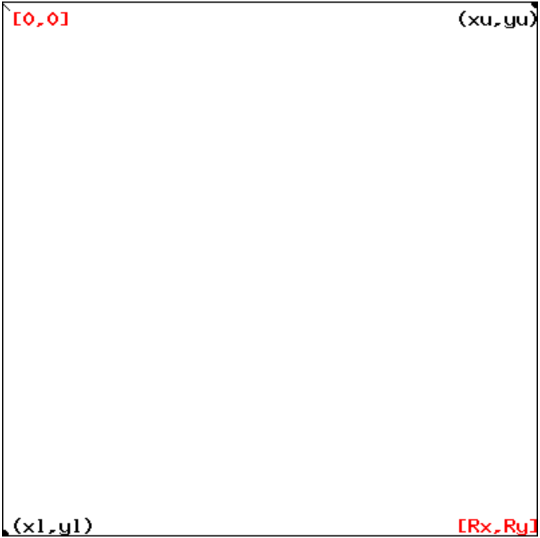
Simplicidade: Economia de Pensamento

Canvas

- Coordenadas Geométricas: (x_{LL}, y_{LL}) à $(x_{UU}, y_{UU}$.
- Resuloção: (R_x, R_y), ex. 400x400
- Abra-cadabras:

```
<?php
//Tell we are sending an image
header("Content-type:  image/png");
$image = imagecreate (400,400)
            or die ("Cannot Create image");
header("Content-type:  image/png");
$bgcolor = imagecolorallocate ($image,255, 255, 255);
$textcolor = imagecolorallocate ($image,0,0,0);
//Draw something and write image
imageline($image,0,1,5,6,$textcolor);
imagepng ($image);
?>
```

$$y = x + 1; \quad 0 \leq x \leq 5$$



Scaling

- Problema: Escalonamento
- Problema: Origem em UR
- Remédio: Escalonamento

$$(x_l, y_l) \mapsto (0, R_Y) \qquad (x_u, y_u) \mapsto (R_x, 0)$$

- Aplicação *Afim*:

$$\underline{\mathbf{X}} = \begin{pmatrix} X \\ Y \end{pmatrix} = \underline{\underline{\mathbf{A}}}\ \underline{\mathbf{x}} + \underline{\mathbf{b}} = \begin{pmatrix} a_x & 0 \\ 0 & a_y \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} b_x \\ b_y \end{pmatrix}$$

- Então:

$$\begin{pmatrix} 0 \\ R_y \end{pmatrix} = \begin{pmatrix} a_x x_l + b_x \\ a_y y_l + b_y \end{pmatrix} \quad \begin{pmatrix} R_x \\ 0 \end{pmatrix} = \begin{pmatrix} a_x x_u + b_x \\ a_y y_u + b_y \end{pmatrix}$$

UFG

Scaling

- Subtraindo:

$$\begin{pmatrix} R_x \\ -R_y \end{pmatrix} = \begin{pmatrix} a_x(x_u - x_l) \\ a_y(y_u - y_l) \end{pmatrix}$$

- $\Leftrightarrow$

$$\begin{pmatrix} a_x \\ a_y \end{pmatrix} = \begin{pmatrix} \frac{R_x}{x_u - x_l} \\ \frac{-R_y}{y_u - y_l} \end{pmatrix}$$

- Inserindo:

$$\begin{pmatrix} b_x \\ b_y \end{pmatrix} = \begin{pmatrix} R_x - a_x x_u \\ a_y y_u \end{pmatrix} = \begin{pmatrix} R_x - \frac{R_x}{x_u - x_l} x_u \\ \frac{-R_y}{y_u - y_l} y_u \end{pmatrix}$$

-

$$\begin{pmatrix} b_x \\ b_y \end{pmatrix} = \begin{pmatrix} \frac{-R_x}{x_u - x_l} x_l \\ \frac{-R_y}{y_u - y_l} y_u \end{pmatrix}$$

Globals

- ```
  global $a,$b;
  $a=array(0.0,0.0);
  $b=array(1.0,1.0);

  global $R;
  $R=array(400,400);

  global $xl,$xu;
  $xl=array(0.0,0.0);
  $xu=array(1.0,1.0);

  global $fact,$delta;
  $fact=1.0/40.0;
  $delta=array($fact*$R[0],$fact*$R[1]);
  ```
  ```

Init Geo

-
```
function InitGeo()
{
 global $a,$b,$xl,$xu,$R;
 $a=array
 (
    (1.0*$R[0])/($xu[0]-$xl[0]),
    (-1.0*$R[1])/($xu[1]-$xl[1])
 );
 $b=array
 (
    1.0*$R[0]-$a[0]*$xu[0],
    -$a[1]*$xu[1]
 );
}
```

Init IMage Object

- ```
  function InitImage()
  {
   //Produce the image
   header("Content-type:  image/png");

   global $R;
   $image = imagecreate ($R[0]+1,$R[1]+1)
            or die ("Cannot Create image");

   //First color allocated is background (white)
   imagecolorallocate ($image,255, 255, 255);

   return $image;
  }
  ```
  ```

Scale Point

- ```
  function ScalePoint($p)
  {
   global $a,$b;

   $pp=array();
   for ($i=0;$i<2;$i++)
   {
      $pp[$i]=$a[$i]*$p[$i]+$b[$i];
   }

   return $pp;
  }
  ```
  ```

Draw Point

- ```
  function DrawPoint($image,$p,$color,$r=10)
  {
   $pp=ScalePoint($p);
   imagefilledarc($image,$pp[0],$pp[1],
                  $r,$r,0,360,$color,0);
  }
  ```
  ```

First Image

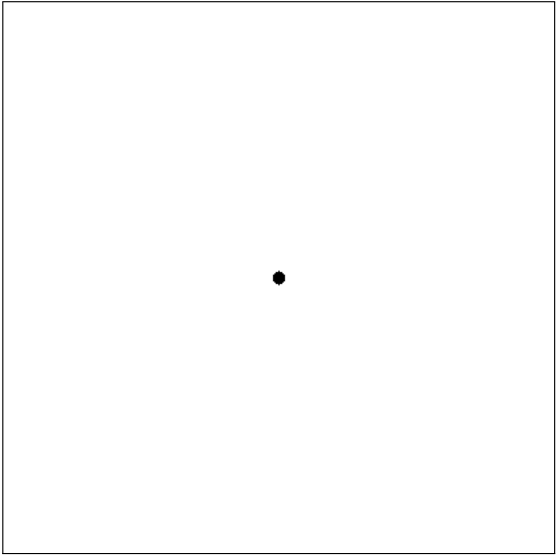
- ```
  function Fig2()
  {
   $image = InitImage();
   //Black
   $textcolor = imagecolorallocate ($image,0,0,0);

   InitGeo();

   $pc=array(0.5,0.5);
   DrawPoint($image,$pc,$textcolor);

   imagepng ($image);
  }
  ```
  ```

First Image

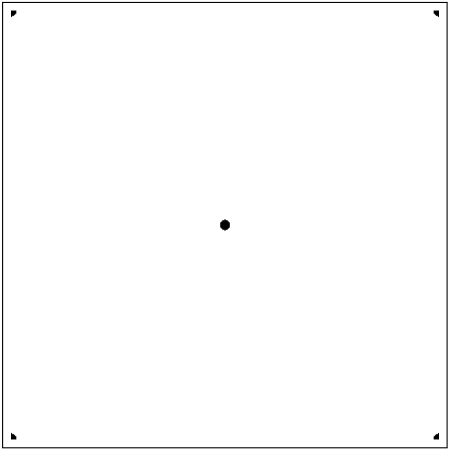


Second Image

```
$p0=array(0.0,0.0);
$p1=array(1.0,0.0);
$p2=array(1.0,1.0);
$p3=array(0.0,1.0);
$pc=array(0.5,0.5);

DrawPoint($image,$p0,$textcolor);
DrawPoint($image,$p1,$textcolor);
DrawPoint($image,$p2,$textcolor);
DrawPoint($image,$p3,$textcolor);
DrawPoint($image,$pc,$textcolor);
```

Second Image



Criando uma Margen

- Problema: Pontos perdo das bordas
- Solução:

$$\begin{pmatrix} \delta_x \\ R_y \end{pmatrix} = \begin{pmatrix} a_x x_l + b_x \\ a_y y_l + b_y \end{pmatrix} \quad \begin{pmatrix} R_x \\ \delta_y \end{pmatrix} = \begin{pmatrix} a_x x_u + b_x \\ a_y y_u + b_y \end{pmatrix}$$

- Resultado:

$$\begin{pmatrix} a_x \\ a_y \end{pmatrix} = \begin{pmatrix} \frac{R_x - 2\delta_x}{x_u - x_l} \\ \frac{2\delta_y - R_y}{y_u - y_l} \end{pmatrix}$$

-

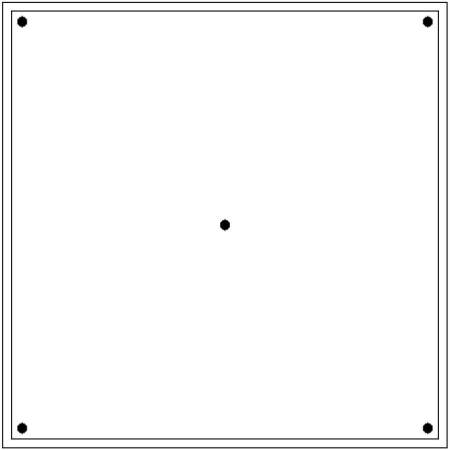
$$\begin{pmatrix} b_x \\ b_y \end{pmatrix} = \begin{pmatrix} R_x - \delta_x - a_x x_u \\ \delta_y - a_y y_u \end{pmatrix}$$

Criando uma Margen

- ```
  function InitGeo()
  {
   global $a,$b,$xl,$xu,$R;

   $delta=array(1.0/40.0*$R[0],1.0/40.0*$R[1]);
   $a=array
   (
      (1.0*$R[0]-2.0*$delta[0])/($xu[0]-$xl[0]),
      (2.0*$delta[1]-1.0*$R[1])/($xu[1]-$xl[1])
   );
   $b=array
   (
      1.0*$R[0]-1.0*$delta[0]-$a[0]*$xu[0],
      1.0*$delta[1]-$a[1]*$xu[1]
   );
  }
  ```
  ```

Second Image - again



Segmento da Reta

- Equação de uma reta, l: $ax + by = c$; $a, b, c \in \mathbb{R}$
- Vetor normal
$$\underline{\mathbf{n}} = \widehat{\underline{\mathbf{v}}} = \begin{pmatrix} a \\ b \end{pmatrix}$$
- Vetor direcional
$$\underline{\mathbf{v}} = \begin{pmatrix} -b \\ a \end{pmatrix}$$
- Parametrização, $\underline{\mathbf{r}}_0 \in l$:
$$\underline{\mathbf{r}} = \underline{\mathbf{r}}_0 + t\underline{\mathbf{v}}, \quad t \in \mathbb{R}$$
-
$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + t \begin{pmatrix} -b \\ a \end{pmatrix}$$

DrawLine

- ```
  function DrawLine($image,$p1,$p2,$color,$r=0)
  {
   $pp1=ScalePoint($p1);
   $pp2=ScalePoint($p2);

   if ($r>0)
   {
      DrawPoint($image,$p1,$color,$r);
      DrawPoint($image,$p2,$color,$r);
   }

  imageline($image,$pp1[0],$pp1[1],$pp2[0],$pp2[1],$color)
  }
  ```
  ```

Segmento da Reta

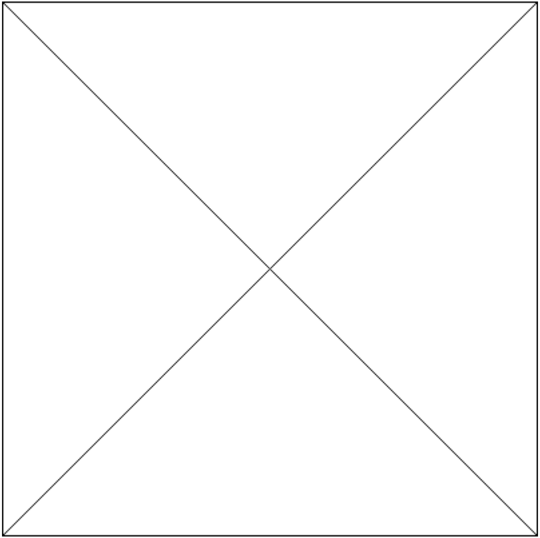
- ```
  function DrawFrameDiags($image,$color)
  {
   global $xl,$xu;

   $xul=array($xu[0],$xl[1]);
   $xlu=array($x1[0],$xu[1]);

   DrawLine($image,$xl,$xul,$color);
   DrawLine($image,$xul,$xu,$color);
   DrawLine($image,$xu,$xlu,$color);
   DrawLine($image,$xlu,$xl,$color);

   DrawLine($image,$xl,$xu,$color);
   DrawLine($image,$xul,$xlu,$color);
  }
  ```
  ```

Retas



Círculo - Elipse

- Equação:

$$(x - x_c)^2 + (y - y_0)^2 = r^2$$

- Parametrização:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_c + r\cos t \\ y_c + r\sin t \end{pmatrix}; \quad t \in [0, 2\pi[$$

- Equação:

$$\left(\frac{x - x_c}{a}\right)^2 + \left(\frac{y - y_c}{b}\right)^2 = 1$$

- Parametrização:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_c + a\cos t \\ y_c + b\sin t \end{pmatrix}; \quad t \in [0, 2\pi[$$

UFG

Círculo - Elipse

- Círculo em $\underline{\mathbf{r}}_c = (\frac{1}{2}, \frac{1}{2})$, $r = \frac{1}{4}$
-
```
imageellipse($image,int $xc,int $yc,
               int $a,int $b,$color)
```
- ScalePoint($\underline{\mathbf{r}}_c$) - E r? Hm!
-
```
function ScaleVector($p)
{
 global $a,$b;
 $pp=array();
 for ($i=0;$i<2;$i++)
 {
    $pp[$i]=$a[$i]*$p[$i];
 }

 return $pp;
}
```

Vector

```
function Vector($p1,$p2)
{
 $p=array();
 for ($i=0;$i<2;$i++)
 {
   $p[$i]=$p2[$i]-$p1[$i];
 }
 return $p;
}
```

Comprimento - 2-Norm

- ```
  function Length($p1,$p2)
  {
   $p=Vector($p1,$p2);
   $len=0.0;
   for ($i=0;$i<2;$i++)
   {
     $len+=$p[$i]*$p[$i];
   }
   return sqrt($len);
  }
  ```
  ```

Círculo

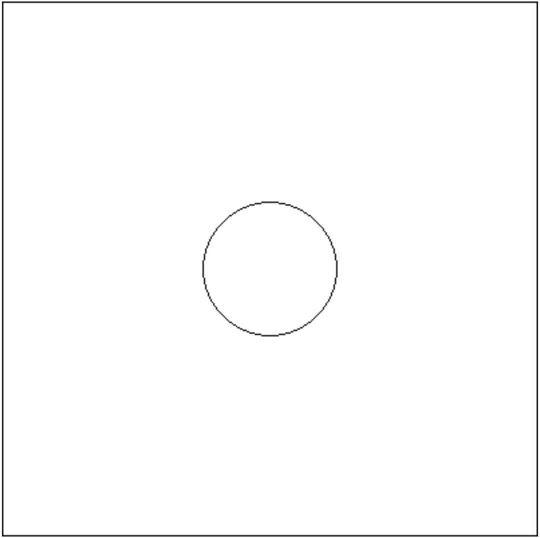
- ```
  $pc=array(0.5,0.5);
  $p1=array(0.75,0.5);
  $p2=array(0.5,0.75);

  $v1=ScaleVector( Vector($pc,$p1) );
  $v2=ScaleVector( Vector($pc,$p2) );

  $a=Length($v1);
  $b=Length($v2);
  $ppc=ScalePoint($pc);

  imageellipse($image,$ppc[0],$ppc[1],$a,$b,$textcolor);
  ```
- Ugly - Feio!
  ```

Círculo



Curva Paramétrica

- $$\underline{\mathbf{r}}(\tau) = \begin{pmatrix} x(\tau) \\ y(\tau) \end{pmatrix}; \quad \tau \in I = [t, T]$$
- Divide I em n intervalos:
$$\tau_0, \tau_1, \ldots, \tau_n$$
- $$\Delta = \frac{T - t}{n}$$
- $\tau_0 = \tau$
  $i = 1, \ldots, n$:
$$\tau_i = \tau_{i-1} + \Delta$$
- Segmentos: $\underline{\mathbf{r}}(\tau_{i-1})$ à $\underline{\mathbf{r}}(\tau_i)$

Elipse

- ```
  function Ellipse($n,$pc,$a,$b)
  {
   $dt=2*pi()/(1.0*($n-1));

   $r=array();
   for ($t=0.0,$m=0;$m<=$n;$m++)
   {
      $r[$m]=array
      (
         $pc[0]+$a*cos($t),
         $pc[1]+$b*sin($t)
      );
   }

   return $r;
  }
  ```
  ```

Poligone

- ```
  function
  DrawPolygon($image,$ps,$color,$close=FALSE,$r=0)
  {
   for ($n=0;$n<count($ps)-1;$n++)
   {
     DrawLine($image,$ps[$n],$ps[$n+1],$color,$r);
   }

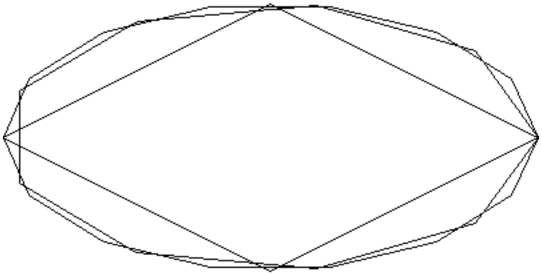
   if ($close)
   {
     DrawLine($image,$ps[count($ps)-1],$ps[0],
              $color,$r);
   }
  }
  ```
  ```

Elipse

- Elipse em $(\frac{1}{2}, \frac{1}{2})$ - semi-eixos: $(\frac{1}{2}, \frac{1}{4})$
- $$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{1}{2} + \frac{1}{2}\cos t \\ \frac{1}{2} + \frac{1}{4}\sin t \end{pmatrix}; \quad t \in [0, 2\pi[$$
- ```
  $n=5;
  for ($k=1;$k<4;$k++)
  {
    $ps=Ellipse($n*$k,array(0.5,0.5),0.5,0.25);

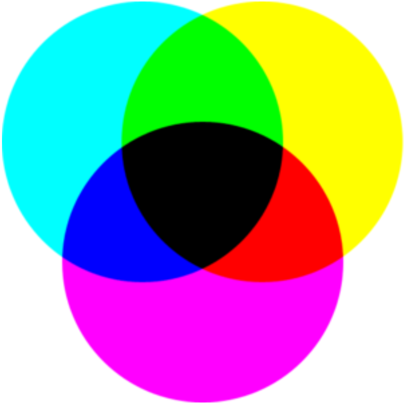
    $pps=ScalePoints($ps);
    DrawPolygon($image,$ps,$textcolor,10);
  }
  ```
  ```

Elipse



Colors

- RGB: Red - Green - Blue
- CMYK: Cyan - Magenta- Yellow - Key (Black)



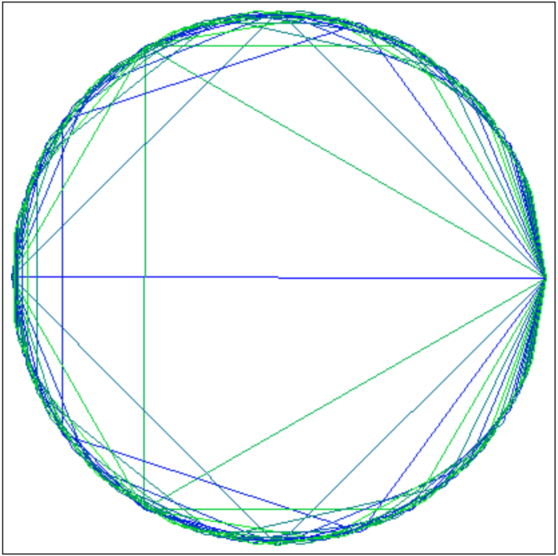
RGB

- Cubo: $[0, 255]x[0, 255]x[0, 255]$
- Black: $(0, 0, 0)$
- White: $(255, 255, 255)$
- Red: $(255, 0, 0)$
- Green: $(0, 255, 0)$
- Blue: $(0, 0, 255)$

Playing

- ```
  $colors=array
  (
     imagecolorallocate ($image,0,0,0),//Black
     imagecolorallocate ($image,255,0,0),//Red
     imagecolorallocate ($image,0,255,0),//Green
     imagecolorallocate ($image,0,0,255)//Blue
  );
  $n=2;
  for ($k=1;$k<=20;$k++)
  {
     $ps=Ellipse($n*$k,array(0.5,0.5),0.5,0.5);
     $pps=ScalePoints($ps);
     DrawPolygon($image,$ps,
                    $colors[ ($k%3)+1 ]); // RGB
  }
  ```
  ```

Poligones



Combinação Linear

- ```
  function
  LinearCombination($dim,$alpha,$v1,$beta,$v2)
  {
   $v=array();
   for ($i=0;$i<$dim;$i++)
   {
     $v[$i]=$alpha*$v1[$i]+$beta*$v2[$i];
   }

   return $v;
  }
  ```
  ```

Combinação Convexa

- ```
  function ConvexCombination($dim,$alpha,$v1,$v2)
  {
   return LinearCombination($dim,
                $alpha,$v1,1.0-$alpha,$v2);
  }
  ```
  ```

Combinação Convexa de Cores

- ```
  $colors=array
  (
  array(255,0,0),
  array(0,255,0),
  array(0,0,255),
  array(0,255,255)
  );

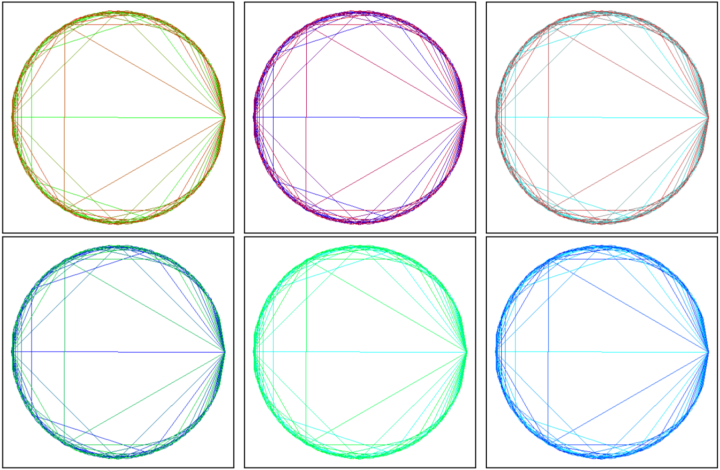
  $nfig=1;
  for ($i=0;$i<count($colors);$i++)
  {
     for ($j=$i+1;$j<count($colors);$j++)
     {
        $n=20;
        for ($k=3;$k<=$n;$k++)
        {
  ```
  ```

Combinação Convexa de Cores

- ```
              $alpha=255.0*($k-3)/(1.0*$n);
              $ps=Ellipse($k,array(0.5,0.5),0.5,0.5);
              $pps=ScalePoints($ps);
              $color=ConvexCombination(3,
              $alpha,$colors[$i],$colors[$j]);
              $color=imagecolorallocate ($image,
              $color[0],$color[1],$color[2]);
              DrawPolygon($image,$ps,$color,TRUE,10);
            }

            ShowImage ($image,"Fig9_$nfig.png");
            $nfig++;
         }
     }
  ```
  ```

Poligones



Transformações

- Linear

$$f(\underline{\mathbf{x}}) = \underline{\underline{\mathbf{A}}}\ \underline{\mathbf{x}}$$

$$f(\underline{\mathbf{x}} + \underline{\mathbf{y}}) = f(\underline{\mathbf{x}}) + f(\underline{\mathbf{y}})$$

$$f(\underline{\lambda\mathbf{x}}) = \lambda f(\underline{\mathbf{x}})$$

- $\underline{\underline{\mathbf{A}}}$ *regular*, $\det \underline{\underline{\mathbf{A}}} \neq 0$:

$$f^{-1}(\underline{\mathbf{x}}) = \underline{\underline{\mathbf{A}}}^{-1}\ \underline{\mathbf{x}}$$

- Afim $\subset$ Linear

$$f(\underline{\mathbf{x}}) = \underline{\underline{\mathbf{A}}}\ \underline{\mathbf{x}} + \underline{\mathbf{b}}$$

$$f(\underline{\mathbf{x}} + \underline{\mathbf{y}}) \neq f(\underline{\mathbf{x}}) + f(\underline{\mathbf{y}})$$

$$f(\underline{\lambda\mathbf{x}}) \neq \lambda f(\underline{\mathbf{x}})$$

- $\underline{\underline{\mathbf{A}}}$ *regular*:

$$f^{-1}(\underline{\mathbf{x}}) = \underline{\underline{\mathbf{A}}}^{-1}\ \underline{\mathbf{x}} - \underline{\mathbf{b}}$$

Translação

- Translação $\underline{\mathbf{t}}$:

$$T(\underline{\mathbf{p}}) = \underline{\mathbf{p}} + \underline{\mathbf{t}}$$

$$T^{(n)}(\underline{\mathbf{p}}) = \underline{\mathbf{p}} + n\underline{\mathbf{t}}, \quad n \in \mathbb{Z}$$

  Ñ linear, mas afim! Merda!!!

- function Translate(t,p)

```
{
 $pp=array();
 for ($i=0;$i<2;$i++)
 {
    $pp[$i]=$p[$i]+$t[$i];
 }

 return $pp;
}
```

Escalonamento

- Fatores $\lambda_x \neq 0, \lambda_y \neq 0$:

$$S(\underline{\mathbf{p}}) = \begin{pmatrix} \lambda_x x & 0 \\ 0 & \lambda_y y \end{pmatrix} \underline{\mathbf{p}}$$

$$S^{(n)}(\underline{\mathbf{p}}) = \begin{pmatrix} \lambda_x^n x & 0 \\ 0 & \lambda_y^n y \end{pmatrix} \underline{\mathbf{p}}, \quad n \in \mathbb{Z}$$

  Linear!

-
```
function Scale($lambdax,$lambday,$p)
{
 return array($lambdax*$p[0],$lambday*$p[1]);
}
```

Rotação

- Ângulo θ:

$$R(\underline{\mathbf{p}}) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \underline{\mathbf{p}}$$

$$R^{(n)}(\underline{\mathbf{p}}) = \begin{pmatrix} \cos n\theta & -\sin n\theta \\ \sin n\theta & \cos n\theta \end{pmatrix} \underline{\mathbf{p}}, \quad n \in \mathbb{Z}$$

Linear!

-
```
function Rotate($theta,$p)
{
 return array
 (
   cos($theta)*$p[0]-sin($theta)*$p[1],
   sin($theta)*$p[0]+cos($theta)*$p[1]
 );
}
```

Projeção

- Ângulo θ, $\underline{\mathbf{e}} = (\cos\theta, \sin\theta)$:

$$P(\underline{\mathbf{p}}) = (\underline{\mathbf{e}} \cdot \underline{\mathbf{p}})\underline{\mathbf{e}} = (p_x \cos\theta + p_y \sin\theta) \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix} =$$

$$\begin{pmatrix} \cos^2\theta & \cos\theta\sin\theta \\ \cos\theta\sin\theta & \sin^2\theta \end{pmatrix} \underline{\mathbf{p}} = \frac{1}{2} \begin{pmatrix} 1+\cos 2\theta & \sin 2\theta \\ \sin 2\theta & 1-\cos 2\theta \end{pmatrix} \underline{\mathbf{p}}$$

  Linear, singular

- *Idempotente*:

$$P^{(2)}(\underline{\mathbf{p}}) = P\left((\underline{\mathbf{e}} \cdot \underline{\mathbf{p}})\underline{\mathbf{e}}\right) = \underline{\mathbf{e}} \cdot ((\underline{\mathbf{e}} \cdot \underline{\mathbf{p}})\underline{\mathbf{e}})\underline{\mathbf{e}} = (\underline{\mathbf{e}} \cdot \underline{\mathbf{p}})\underline{\mathbf{e}} = P(\underline{\mathbf{p}})$$

- Por indução:

$$P^{(n)}(\underline{\mathbf{p}}) = P(\underline{\mathbf{p}}) \quad n \in \mathbb{N}$$

Projeção

- Corolário:

$$\begin{pmatrix} \cos^2\theta & \cos\theta\sin\theta \\ \cos\theta\sin\theta & \sin^2\theta \end{pmatrix}^n = \begin{pmatrix} \cos^2\theta & \cos\theta\sin\theta \\ \cos\theta\sin\theta & \sin^2\theta \end{pmatrix}$$

- ```
function Project($theta,$p)
{
 return array
 (
    cos($theta)*cos($theta)*$p[0]+,
    sin($theta)*cos($theta)*$p[1],
    sin($theta)*cos($theta)*$p[0]+
    sin($theta)*sin($theta)*$p[1]
 );
}
```

UFG
```

Reflexão

- Ângulo θ, $\underline{\mathbf{e}} = (\cos\theta, \sin\theta)$:

$$\underline{\mathbf{p}} = \underline{\mathbf{p}}_{\parallel} + \underline{\mathbf{p}}_{\perp}$$

$$\underline{\mathbf{p}}_{\parallel} = (\underline{\mathbf{e}} \cdot \underline{\mathbf{p}})\underline{\mathbf{e}}$$

$$\underline{\mathbf{p}}_{\perp} = \underline{\mathbf{p}} - (\underline{\mathbf{e}} \cdot \underline{\mathbf{p}})\underline{\mathbf{e}}$$

$$R(\underline{\mathbf{p}}) = \underline{\mathbf{p}}_{\parallel} - \underline{\mathbf{p}}_{\perp} = 2(\underline{\mathbf{e}} \cdot \underline{\mathbf{p}})\underline{\mathbf{e}} - \underline{\mathbf{p}} =$$

$$2(p_x \cos\theta + p_y \sin\theta) \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix} - \begin{pmatrix} p_x \\ p_y \end{pmatrix} =$$

$$\begin{pmatrix} 2\cos^2\theta - 1 & 2\sin\theta\cos\theta \\ 2\sin\theta\cos\theta & 2\sin^2\theta - 1 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \end{pmatrix} =$$

$$\begin{pmatrix} \cos 2\theta & \sin 2\theta \\ \sin 2\theta & -\cos 2\theta \end{pmatrix} \begin{pmatrix} p_x \\ p_y \end{pmatrix}$$

Reflexão

- Corolário:

$$\left(\begin{array}{cc} \cos 2\theta & \sin 2\theta \\ \sin 2\theta & -\cos 2\theta \end{array}\right)^2 = \underline{\underline{I}}$$

-
```
function Relect($theta,$p)
{
 return array
 (
    cos(2$theta)*$p[0]+,
    sin(2$theta)*$p[1],
    sin(2$theta)2*$p[0]+
    -cos(2$theta)*$p[1]
 );
}
```

Hipérbole

- Equação:

$$\left(\frac{x-x_c}{a}\right)^2-\left(\frac{y-y_c}{b}\right)^2=1 \qquad \left(\frac{y-y_c}{a}\right)^2-\left(\frac{x-x_c}{b}\right)^2=1$$

- Hiperbólicas

$$\cosh t=\frac{e^t+e^{-t}}{2} \qquad \sinh t=\frac{e^t-e^{-t}}{2}$$

- $$\cosh^2 t-\sinh^2 t=1$$

$$\cosh 2t=\cosh^2 t+\sinh^2 t=2\cosh^2 t-1=2\sinh^2 t+1$$

- Parametrização:

$$\begin{pmatrix} x \\ y \end{pmatrix}=\begin{pmatrix} x_c+a\cosh t \\ y_c+b\sinh t \end{pmatrix}; \qquad t\in\mathbb{R}$$

UFG

Hipérbole

- ```
  function Hiperbola($n,$pc,$a,$b)
  {
   $wdt=4.0;
   $dt=$wdt/(1.0*($n-1));
   $r=array();
   for ($t=-$wdt/2.0,$m=0;$m<=$n;$m++)
   {
     $r[$m]=array
     (
       $pc[0]+$a*cosh($t),
       $pc[1]+$b*sinh($t)
     );
     $t+=$dt;
   }
   return $r;
  }
  ```
  ```

Hipérbole

```
global $xl,$xu;
$xl=array(-2.0,-2.0);
$xu=array(2.0,2.0);
$image = InitImage();
InitGeo();
$color=imagecolorallocate ($image,0,0,0);
$ps=Hiperbola(50,array(0.0,0.0),1.0,1.0);
$pps=array();
for ($n=0;$n<count($ps);$n++)
{
   $pps[$n]=Reflect(pi()/2.0,$ps[$n]);
}
DrawPolygon($image,$ps,$color,FALSE,10);
DrawPolygon($image,$pps,$color,FALSE,10);
```

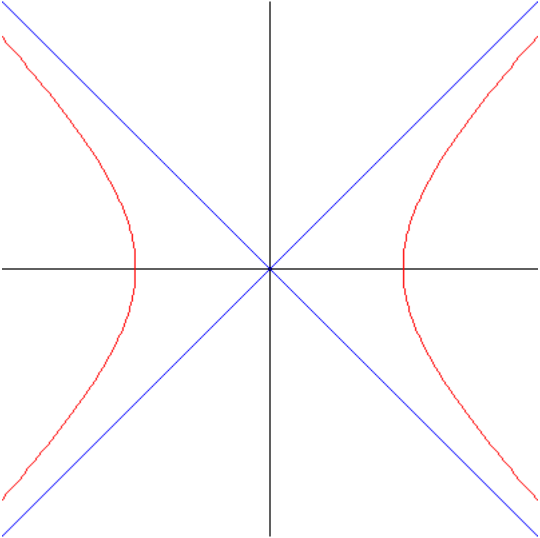
Hipérbole

- ```
  DrawLine($image,array(-2.0,0.0),array(2.0,0.0),$black);
  DrawLine($image,array(0.0,-2.0),array(0.0,2.0),$black);

  DrawLine($image,array(-2.0,-2.0),array(2.0,2.0),$blue);
  DrawLine($image,array(2.0,-2.0),array(-2.0,2.0),$blue);

  DrawPolygon($image,$ps,$red,FALSE,10);
  DrawPolygon($image,$pps,$red,FALSE,10);
  ```
  ```

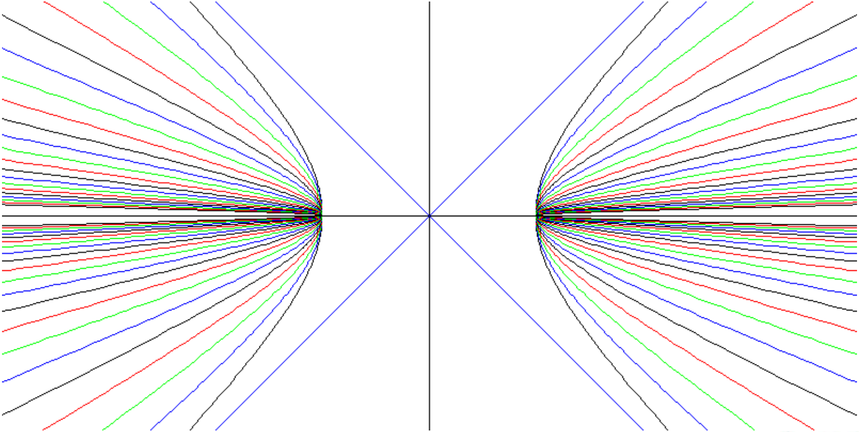
Hipérbole



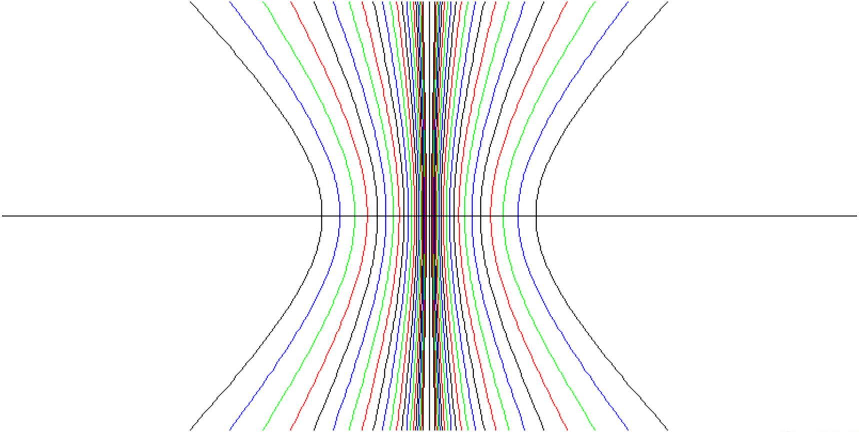
Hipérbole Escalonado

```
$v=1.0;$a=1.0/1.2;
$pss=array();
$ppss=array();
for ($k=0;$k<=20;$k++)
{
   for ($n=0;$n<count($ps);$n++)
   {
      $pss[$n]=Scale(1.0,$v,$ps[$n]);
      $ppss[$n]=Scale(1.0,$v,$pps[$n]);
   }
   DrawPolygon($image,$pss,
               $colors[$k%4],FALSE,10);
   DrawPolygon($image,$ppss,
               $colors[$k%4],FALSE,10);
   $v*=$a;
}
```

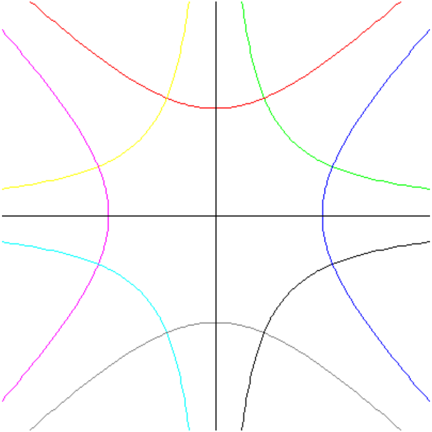
Hipérbole Escalonado



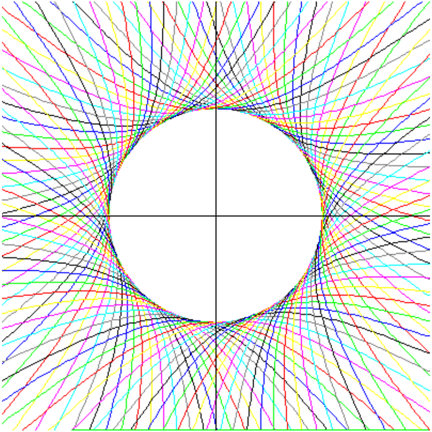
Hipérbole Escalonado



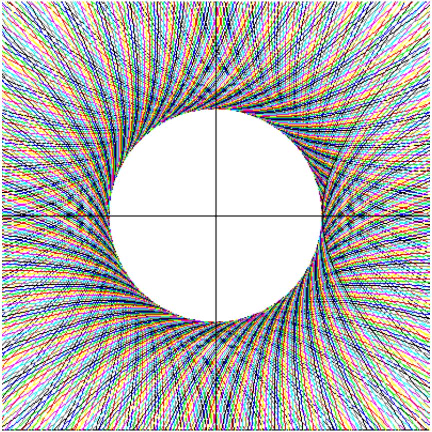
Hipérbole Rotacionado



Hipérbole Rotacionado



Hipérbole Rotacionado



Rotacão pelo $\underline{\mathbf{p}}_c$

- Translar centro $\underline{\mathbf{p}}_c$ até origem
- Rotacione
- Translar de volta
- Translação não linear...
- Esdrúxula...

Rotationar pelo $\underline{\mathbf{p}}_c$

- Geometria Projetiva
- $$\begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} \mapsto \begin{pmatrix} \frac{x'}{z'} \\ \frac{y'}{z'} \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} \frac{x'}{z'} \\ \frac{y'}{z'} \end{pmatrix}$$

  $z' \neq 0$. $z' = 0$: Ponto no infinito
- Olhar o plano: $z = 1$ do origem
- $$\begin{pmatrix} 1 & 0 & t_1 \\ 0 & 1 & t_2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x + t_1 \\ y + t_2 \\ 1 \end{pmatrix}$$

  Transla!

Geometria Projetiva

- $$\begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \\ 1 \end{pmatrix}$$

  Rotaciona!

- $$\begin{pmatrix} \lambda_x & 0 & 0 \\ 0 & \lambda_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} \lambda_x x \\ \lambda_y y \\ 1 \end{pmatrix}$$

  Escalona!

Rotação pelo $\underline{\mathbf{p}}_c$

- Translar centro $\underline{\mathbf{p}}_c$ até origem:

$$\underline{\underline{\mathbf{T}}} = \begin{pmatrix} 1 & 0 & -p_x \\ 0 & 1 & -p_y \\ 0 & 0 & 1 \end{pmatrix}$$

- Rotacione:

$$\underline{\underline{\mathbf{R}}} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- Translar de volta:

$$\underline{\underline{\mathbf{T}}}^{-1} = \begin{pmatrix} 1 & 0 & p_x \\ 0 & 1 & p_y \\ 0 & 0 & 1 \end{pmatrix}$$

- Composição pela esquerda:

$$\underline{\underline{\mathbf{R}}}' = \underline{\underline{\mathbf{T}}}^{-1}\underline{\underline{\mathbf{R}}}\,\underline{\underline{\mathbf{T}}}$$

Translação

- ```
  function TransMatrix($t)
  {
   return array
   (
      array(1.0,0.0,$t[0]),
      array(0.0,1.0,$t[1]),
      array(0.0,0.0,1.0),
   );
  }
  ```
  ```

Rotação

- ```
  function RotationMatrix($theta)
  {
   return array
   (
     array(cos($theta),-sin($theta),$t[0]),
     array(sin($theta),cos($theta),$t[1]),
     array(0.0,0.0,1.0),
   );
  }
  ```
  ```

Escalonamento

- ```
  function ScalingMatrix($lambdax,$lambday)
  {
   return array
   (
     array($lambdax,0.0,$t[0]),
     array(0.0,$lambday,$t[1]),
     array(0.0,0.0,1.0),
   );
  }
  ```
  ```

Transformação

- ```
  function Transform($A,$p)
  {
   $pp=array();
   for ($i=0;$i<count($A);$i++)
   {
     $pp[$i]=0.0;
     for ($j=0;$j<count($A[$i]);$j++)
     {
       $pp[$i]+=$A[$i][$j]*$p[$j];
     }
   }
   return $pp;
  }
  ```
  ```

Composição - Multiplicação

```
function MatrixMult($A,$B)
{
 $C=array();
 for ($i=0;$i<count($A);$i++)
 {
    $C[$i]=array();
    for ($j=0;$j<count($A);$j++)
    {
       $C[$i][$j]=0.0;
       for ($k=0;$k<count($A);$k++)
       {
          $C[$i][$j]+=$A[$i][$k]*$B[$k][$j];
       }
    }
 }
 return $C;
```

Elipse roteado

- Elipse em $(\frac{1}{2}, \frac{1}{2})$, semieixos $(\frac{1}{2}$ e $\frac{1}{4})$, roteado $\frac{\pi}{3}$ graus pelo centro
- ```
  $ps=Ellipse(200,array(0.5,0.5),0.5,0.25);
  for ($m=0;$m<count($ps);$m++)
  {
     $ps[$m][2]=1.0;
  }

  $T=TransMatrix(array(-0.5,-0.5));
  $R=RotationMatrix(pi()/3.0);
  $TT=TransMatrix(array(0.5,0.5));

  $R1=MatrixMult($R,$T);
  $R2=MatrixMult($TT,$R1);
  ```
  ```

Elipse roteado

- ```
  $pss=array();
  $psss=array();
  $pssss=array();
  for ($m=0;$m<count($ps);$m++)
  {
    $pss[$m]=Transform($T,$ps[$m]);
    $psss[$m]=Transform($R1,$ps[$m]);
    $pssss[$m]=Transform($R2,$ps[$m]);
  }

  DrawPolygon($image,$ps,$colors[0],FALSE,10);
  DrawPolygon($image,$pss,$colors[1],FALSE,10);
  DrawPolygon($image,$psss,$colors[2],FALSE,10);
  DrawPolygon($image,$pssss,$colors[3],FALSE,10);
  ```
  ```

Elipse roteado