

Approximating an Ellipse by Circular Arcs

David Eberly
Magic Software, Inc.
<http://www.magic-software.com>

Created: January 13, 2002

1 Algorithm

The ellipses to be approximated are axis-aligned and centered at the origin,

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \quad (1)$$

where $a \neq b$. Only the portion of the ellipse in the first quadrant will be approximated by circular arcs. The approximations in the other quadrants can be obtained by reflections of those in the first quadrant.

The first part of the process involves selecting a set of ellipse points $\vec{P}_i = (x_i, y_i)$, $0 \leq i \leq n$, where $\vec{P}_0 = (a, 0)$, $\vec{P}_n = (0, b)$, and the points are counterclockwise ordered in the first quadrant. The selection is based on weighted averages of curvatures. Given a parameterized curve $(x(t), y(t))$, the curvature is

$$K(t) = \frac{x'(t)y''(t) - y'(t)x''(t)}{((x'(t))^2 + (y'(t))^2)^{3/2}}.$$

The ellipse is parameterized by $x(t) = a \cos t$ and $y(t) = b \sin t$ for $t \in [0, 2\pi)$. The derivatives are $x'(t) = -a \sin t$, $y'(t) = b \cos t$, $x''(t) = -a \cos t$, and $y''(t) = -b \sin t$. The curvature is $K(t) = ab/(a^2 \sin^2 t + b^2 \cos^2 t)^{3/2}$. As a function of (x, y) , the curvature is

$$K(x, y) = \frac{ab}{\left(\left(\frac{bx}{a}\right)^2 + \left(\frac{ay}{b}\right)^2\right)^{3/2}}, \quad (2)$$

where it is understood that (x, y) additionally satisfies equation (1). If the curvature K is specified, the corresponding (x, y) is obtained by solving equations (1) and (2). Setting $\lambda = (ab/K)^{2/3}$, the solution is

$$x = a \sqrt{\left| \frac{\lambda - a^2}{b^2 - a^2} \right|}, \quad y = b \sqrt{\left| \frac{\lambda - b^2}{a^2 - b^2} \right|}.$$

Define $K_0 = K(a, 0) = a/b^2$ and $K_n = K(0, b) = b/a^2$. The point \vec{P}_i for $1 \leq i \leq n-1$ is selected so that the curvature is $K_i = (1 - i/n)K_0 + (i/n)K_n$.

The second part of the process involves selecting a circular arc with center \vec{C}_i and radius r_i , $0 \leq i < n$, whose end points are \vec{P}_i and \vec{P}_{i+1} . In order to obtain a vertical tangent at $(a, 0)$, the circle (\vec{C}_0, r_0) is chosen that

contains \vec{P}_0 , \vec{P}_1 , and $(x_1, -y_1)$ where the last point is the reflection of \vec{P}_1 through the x -axis. Similarly, in order to obtain a horizontal tangent at $(0, b)$, the circle (\vec{C}_{n-1}, r_{n-1}) is chosen that contains \vec{P}_{n-1} , \vec{P}_n , and $(-x_{n-1}, y_{n-1})$ where the last point is the reflection of \vec{P}_{n-1} through the y -axis. The other circles (\vec{C}_i, r_i) for $1 \leq i \leq n-2$ are chosen to contain \vec{P}_{i-1} , \vec{P}_i , and \vec{P}_{i+1} .

2 Implementation

The source code for the approximation is in files `MgcAppr2DEllipseByArcs.{h,cpp}`. The cpp file is

```
// Magic Software, Inc.
// http://www.magic-software.com
// Copyright (c) 2000-2002. All Rights Reserved
//
// Source code from Magic Software is supplied under the terms of a license
// agreement and may not be copied or disclosed except in accordance with the
// terms of that agreement. The various license agreements may be found at
// the Magic Software web site. This file is subject to the license
//
// FREE SOURCE CODE
// http://www.magic-software.com/License/free.pdf

#include "MgcAppr2DEllipseByArcs.h"
#include "MgcCont2DCircleScribe.h"

//-----
void Mgc::ApproximateEllipseByArcs (float fA, float fB, int iNumArcs,
    Vector2*& rakPoint, Vector2*& rakCenter, float*& rafRadius)
{
    // allocate arrays
    assert( iNumArcs >= 2 );
    if ( iNumArcs < 2 )
    {
        rakPoint = NULL;
        rakCenter = NULL;
        rafRadius = NULL;
        return;
    }

    rakPoint = new Vector2[iNumArcs+1];
    rakCenter = new Vector2[iNumArcs];
    rafRadius = new float[iNumArcs];

    // intermediate ellipse quantities
    float fA2 = fA*fA, fB2 = fB*fB, fAB = fA*fB, fInvB2mA2 = 1.0f/(fB2-fA2);

    // End points of ellipse in first quadrant. Points are generated in
    // counterclockwise order.
    rakPoint[0] = Vector2(fA,0.0f);
    rakPoint[iNumArcs] = Vector2(0.0f,fB);
```

```

// curvature at end points, store curvature for computing arcs
float fK0 = fA/fB2;
float fK1 = fB/fA2;

// select ellipse points based on curvature properties
float fInvNumArcs = 1.0f/iNumArcs;
int i;
for (i = 1; i < iNumArcs; i++)
{
    // curvature at new point is weighted average of curvature at ends
    float fW1 = i*fInvNumArcs, fW0 = 1.0f - fW1;
    float fK = fW0*fK0 + fW1*fK1;

    // compute point having this curvature
    float fTmp = Math::Pow(fAB/fK,0.6666667f);
    rakPoint[i].x = fA*Math::Sqrt(Math::FAbs((fTmp-fA2)*fInvB2mA2));
    rakPoint[i].y = fB*Math::Sqrt(Math::FAbs((fTmp-fB2)*fInvB2mA2));
}

// compute arc at (a,0)
Circle2 kCircle;
Circumscribe(Vector2(rakPoint[1].x,-rakPoint[1].y),rakPoint[0],
    rakPoint[1],kCircle);
rakCenter[0] = kCircle.Center();
rafRadius[0] = kCircle.Radius();

// compute arc at (0,b)
int iLast = iNumArcs-1;
Circumscribe(Vector2(-rakPoint[iLast].x,rakPoint[iLast].y),
    rakPoint[iNumArcs],rakPoint[iLast],kCircle);
rakCenter[iLast] = kCircle.Center();
rafRadius[iLast] = kCircle.Radius();

// compute arcs at intermediate points between (a,0) and (0,b)
int iM, iP;
for (iM = 0, i = 1, iP = 2; i < iLast; iM++, i++, iP++)
{
    Circumscribe(rakPoint[iM],rakPoint[i],rakPoint[iP],kCircle);
    rakCenter[i] = kCircle.Center();
    rafRadius[i] = kCircle.Radius();
}
}
//-----

```

3 Test Program

The test program is

// Magic Software, Inc.

```

// http://www.magic-software.com
// Copyright (c) 2000-2002. All Rights Reserved
//
// Source code from Magic Software is supplied under the terms of a license
// agreement and may not be copied or disclosed except in accordance with the
// terms of that agreement. The various license agreements may be found at
// the Magic Software web site. This file is subject to the license
//
// FREE SOURCE CODE
// http://www.magic-software.com/License/free.pdf

#include "MgcAppr2DEllipseByArcs.h"
#include "MgcImages.h"
#include "MgcMatrix2.h"
#include "bresenhm.h"
using namespace Mgc;

static int gs_iB0 = 256, gs_iB1 = 256;
static int gs_iB0M = gs_iB0-1, gs_iB1M = gs_iB1-1;
static ImageRGB82D gs_kImage(gs_iB0,gs_iB0);
static int gs_iColor = 0;

//-----
static void SetPixel (int iX, int iY)
{
    if ( 0 <= iX && iX < gs_kImage.GetBound(0)
        && 0 <= iY && iY < gs_kImage.GetBound(1) )
    {
        gs_kImage(iX,iY) = gs_iColor;
    }
}
//-----
int main ()
{
    float fA = 2.0f, fB = 1.0f;
    int iNumArcs = 8;
    Vector2* akPoint = new Vector2[iNumArcs+1];
    Vector2* akCenter = new Vector2[iNumArcs];
    float* afRadius = new float[iNumArcs];

    ApproximateEllipseByArcs(fA,fB,iNumArcs,akPoint,akCenter,afRadius);

    // draw approximate ellipse to see how good it looks
    gs_kImage = 0x00FFFFFF;
    gs_iColor = 0x00FF0000;
    float fMax = 1.1f*fA, fMin = -fMax, fRange = fMax-fMin;
    int i;
    for (i = 0; i < iNumArcs; i++)
    {
        // inefficiently draw circle, I just want to see the results
        Vector2 kP0 = akPoint[i] - akCenter[i];
        float fR0 = kP0.Length();
    }
}

```

```

    Vector2 kP1 = akPoint[i+1] - akCenter[i];
    float fR1 = kP1.Length();
    float fAngle = Math::ACos(kP0.Dot(kP1)/(afRadius[i]*afRadius[i]));
    int jMax = 256;
    for (int j = 0; j <= jMax; j++)
    {
        Matrix2 kRot;
        kRot.FromAngle(j*fAngle/jMax);
        Vector2 kPm = akCenter[i] + kRot*kP0;

        int iX = int(gs_iBOM*(kPm.x-fMin)/fRange);
        int iY = int(gs_iB1M*(kPm.y-fMin)/fRange);
        SetPixel(iX,iY);
        SetPixel(iX,gs_iB1M-iY);
        SetPixel(gs_iBOM-iX,gs_iB1M-iY);
        SetPixel(gs_iBOM-iX,iY);
    }
}

// plot true ellipse
gs_iColor = 0x000000FF;
Ellipse2D(128,128,128/1.1f,64/1.1f,SetPixel);

// plot sample points
gs_iColor = 0x00000000;
for (i = 0; i <= iNumArcs; i++)
{
    Vector2 kPm = akPoint[i];
    int iX = int((gs_kImage.GetBound(0)-1)*(kPm.x-fMin)/fRange);
    int iY = int((gs_kImage.GetBound(1)-1)*(kPm.y-fMin)/fRange);
    for (int iDY = -1; iDY <= 1; iDY++)
    {
        for (int iDX = -1; iDX <= 1; iDX++)
        {
            SetPixel(iX+iDX,iY+iDY);
            SetPixel(iX+iDX,gs_iB1M-iY+iDY);
            SetPixel(gs_iBOM-iX+iDX,gs_iB1M-iY+iDY);
            SetPixel(gs_iBOM-iX+iDX,iY+iDY);
        }
    }
}

gs_kImage.Save("appr.im");

delete[] akPoint;
delete[] akCenter;
delete[] afRadius;

return 0;
}
//-----

```

The variable `iNumArcs` was set to 2, 4, and 8. The approximation by arcs is drawn in blue and the true ellipse, drawn with a Bresenham-style algorithm, is drawn in red. The arc end points are drawn as black dots. The following figures show the drawings, 2 arcs in the first quadrant in the top image, 4 arcs for the middle image, and 8 arcs for the bottom image.

