



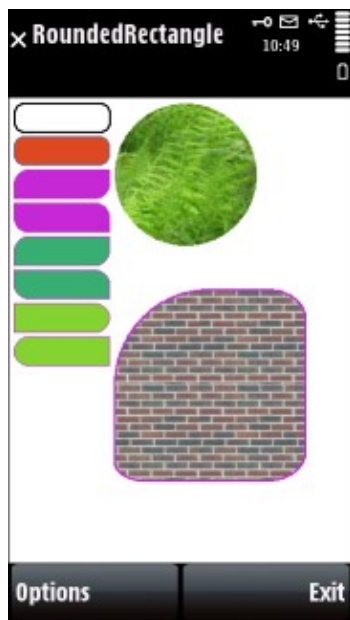
<b>ID</b>		<b>Creation date</b>	May 7, 2009
<b>Platform</b>	S60 3rd Edition, S60 5th Edition	<b>Tested on devices</b>	Nokia 5800 XpressMusic, Nokia E90
<b>Category</b>	Symbian C++	<b>Subcategory</b>	UI

**Keywords (APIs, classes, methods, functions):** CWindowGc::UseBrushPattern(), CWindowGc::DrawPolygon()

## Overview

In many cases a rounded rectangle would look better than just a plain rectangle with sharp corners. You can use `CWindowGc::DrawRoundRect()` to draw a rounded rectangle, but it only allows you to have four symmetric corners.

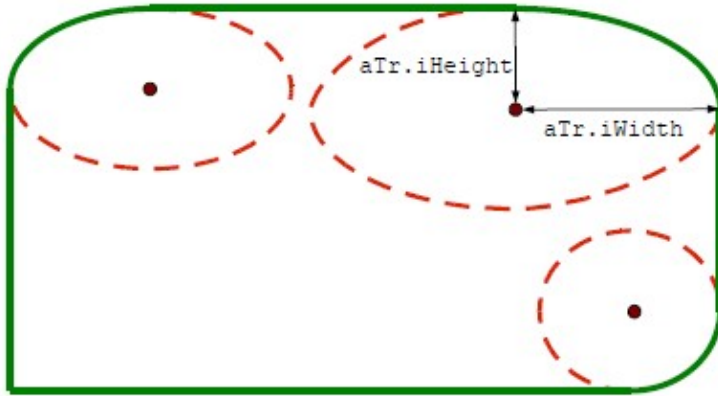
This short article provides you a one solution to create a more complicated rounded rectangles, which have different properties for each of the corners.



## How it is done

Since drawing of advanced rounded rectangles is not directly supported, one must do some hacking. You can think rounded rectangle to consist of four ellipses (or less if there is sharp corners) as presented in the picture below.

## How\_to\_create\_a\_rounded\_rectangle



The outermost points of each ellipse are part of the border of the rounded rectangle. There is different ways to calculate ellipse points, but I used one that can be found from the end of the ellipse article in the wikipedia.

CWindowGc:: DrawPolygon() is used to draw the rectangle. It is quite handy as you can use gc.UseBrushPattern() to set background image and gc.SetPenXXX() properties to define how the border will look. Please note that it would be more efficient to calculate and store the rectangle vertice values before entering the draw function. In this example, calculation is done "on the fly".

## Source code

### Header file

```
/*
 * Draws a rectangle. Each corner radius is limited to be half of rectangle height or width
 *
 * gc graphics context
 * aX upper left corner x coordinate
 * aY upper left corner y coordinate
 * width rectangle width
 * height rectangle height
 * aBl bottom left corner properties TSize(ellipse horizontal radius, ellipse vertical radius)
 * aTl top left corner properties
 * aTr top right corner properties
 * aBr bottom right corner properties
 * aBodrderThickness thickness of the border
 * aBorderColor border color
 * aBackgroundColor color of the background (fill color)
 * */
void drawRoundedRectangle(CWindowGc &gc, TInt aX, TInt aY, TInt width, TInt height,
TSize aBl, TSize aTl, TSize aTr, TSize aBr, TSize aBodrderThickness, TRgb aBorderColor, TRgb aBack

/*
 * gc graphics context
 * aX upper left corner x coordinate
 * aY upper left corner y coordinate
 * width rectangle width
 * height rectangle height
 * aBl bottom left corner properties TSize(ellipse horizontal radius, ellipse vertical radius)
```

## How\_to\_create\_a\_rounded\_rectangle

```
* aTl top left corner properties
* aTr top right corner properties
* aBr bottom right corner properties
* aBodrderThickness thickness of the border
* aBorderColor border color
* aBackgroundImage Pointer to the background image
*/
void drawRoundedRectangle(CWindowGc &gc, TInt aX, TInt aY, TInt width, TInt height,
TSize aBl, TSize aTl, TSize aTr, TSize aBr, TSize aBodrderThickness, TRgb aBorderColor, CFbsBitmap

CArrayFix<TPoint>* calculateCornerPixels(TInt aX, TInt aY, TInt width, TInt height, TSize aBl, T

/*
* From CCoeControl, Draw
* Draw this CRoundedRectangleAppView to the screen.
* @param aRect the rectangle of this view that needs updating
*/
void Draw(const TRect& aRect) const;
```

## Source file

```
void CRoundedRectangleAppView::drawRoundedRectangle(CWindowGc &gc, TInt aX, TInt aY, TInt aWidth,
TSize aBl, TSize aTl, TSize aTr, TSize aBr, TSize aBorderThickness, TRgb aBorderColor, TRgb aBack
{
    SetBrushStyle(CGraphicsContext::ESolidBrush);
    SetPenStyle(CGraphicsContext::ESolidPen);

    SetPenSize(aBorderThickness);
    SetPenColor(aBorderColor);

    SetBrushColor(aBackgroundColor);

    CArrayFix<TPoint>* points = calculateCornerPixels(aX, aY, aWidth, aHeight, aBl, aTl, aTr, aBr);
    DrawPolygon(points);
    delete points;
}

void CRoundedRectangleAppView::drawRoundedRectangle(CWindowGc &gc, TInt aX, TInt aY, TInt aWidth,
TSize aBl, TSize aTl, TSize aTr, TSize aBr, TSize aBorderThickness, TRgb aBorderColor, CFbsBitmap
{
    SetPenStyle(CGraphicsContext::ESolidPen);
    SetPenSize(aBorderThickness);
    SetPenColor(aBorderColor);

    UseBrushPattern(aBackgroundImage);
    SetBrushStyle(CGraphicsContext::EPatternedBrush);

    CArrayFix<TPoint>* points = calculateCornerPixels(aX, aY, aWidth, aHeight, aBl, aTl, aTr, aBr);

    DrawPolygon(points);

    SetBrushStyle(CGraphicsContext::ESolidBrush);
    DiscardBrushPattern();

    delete points;
}
```

## How\_to\_create\_a\_rounded\_rectangle

```

CArrayFix<TPoint>* CRoundedRectangleAppView::calculateCornerPixels(TInt aX, TInt aY, TInt width,
{
    TReal s0,alpha
    TReal e0,alpha
    TReal alpha
    =0,alpha;X
    =0,alpha;X
    const TReal piRad = KPi/180;

    //simple sanity checks
    TReal widthPerTwo = width /2;
    TReal heightPerTwo = height /2;

    aBl.iWidth = Min(aBl.iWidth, widthPerTwo);
    aTl.iWidth = Min(aTl.iWidth, widthPerTwo);
    aTr.iWidth = Min(aTr.iWidth, widthPerTwo);
    aBr.iWidth = Min(aBr.iWidth, widthPerTwo);

    aBl.iHeight = Min(aBl.iHeight, heightPerTwo);
    aTl.iHeight = Min(aTl.iHeight, heightPerTwo);
    aTr.iHeight = Min(aTr.iHeight, heightPerTwo);
    aBr.iHeight = Min(aBr.iHeight, heightPerTwo);

    //calculate ellipse middle points
    TPoint bottomLeft(aX+aBl.iWidth, aY+height-aBl.iHeight);
    TPoint topLeft(aX+aTl.iWidth, aY+aTl.iHeight);
    TPoint topRight(aX+width-aTr.iWidth, aY+aTr.iHeight);
    TPoint bottomRight(aX+width-aBr.iWidth, aY+height-aBr.iHeight);

    //container for the calculated points
    CArrayFixFlat<TPoint> rectanglePoints(12);
    CleanUp(rectanglePoints);

    //bottom left corner
    if(aBl.iHeight==0 || aBl.iWidth==0)
    {
        ->AppendL(TPoint(aX,aY+height)); //sharp corner
    }
    else
    {
        //let's reduce points needed based on the corner properties
        //no need for calculating fixed amount of points for every corner e.g. 2x2 corner
        =0; TReal stepBl
        ::Sqrt(stepBl*stepBl+aBl.iWidth*aBl.iWidth+aBl.iHeight*aBl.iHeight);
        = ((90/stepBl < 1 ? 1: 90/stepBl); //limit the minium to be 1 --> max 90 points per corner

    for (TReal i = 90; i <= 180 ; i += stepBl)
    {
        = i * piRad ;          alpha
        ::Sin(sinalpha,alpha);   Math
        ::Cos(cosalpha,alpha);   Math

        ::Round(X,bottomLeft.iX + (aBl.iWidth * cosalpha),0);
        ::Round(Y,bottomLeft.iY + (aBl.iHeight * sinalpha),0);

        ->AppendL(TPoint(aX,aY+height));
    }
}

// top left corner

```

## How\_to\_create\_a\_rounded\_rectangle

```
if(aTl.iHeight==0 || aTl.iWidth==0)
{
    ->AppendL@P@oins(aX,aY);
}
else
{
    =0; TReal stepTl
    ::Sqrt(stepTl^2+aTl.iWidth*aTl.iWidth+aTl.iHeight*aTl.iHeight);
    = ((90/stepTl) ? 1: 90/stepTl);

for (TReal i = 180; i <= 270; i += stepTl)
{
    = i * piRad ;    alpha
    ::Sin(sinalpha,alphaMath
    ::Cos(cosalpha,alphaMath

    ::Round(X,topLeft.iXMath(aTl.iWidth * cosalpha),0);
    ::Round(Y,topLeft.iYMath(aTl.iHeight * sinalpha),0);

    ->AppendL@P@oins(aX,aY);
}
}

// top right corner
if(aTr.iHeight==0 || aTr.iWidth==0)
{
    ->AppendL@P@oins(aX+width,aY);
}
else
{
    =0; TReal stepTr
    ::Sqrt(stepTr^2+aTr.iWidth*aTr.iWidth+aTr.iHeight*aTr.iHeight);
    = ((90/stepTr) ? 1: 90/stepTr);

for (TReal i = 270; i <= 360; i += stepTr)
{
    = i * piRad ;    alpha
    ::Sin(sinalpha,alphaMath
    ::Cos(cosalpha,alphaMath

    ::Round(X,topRight.iXMath(aTr.iWidth * cosalpha),0);
    ::Round(Y,topRight.iYMath(aTr.iHeight * sinalpha),0);

    ->AppendL@P@oins(aX,aY);
}
}

// bottom right corner
if(aBr.iHeight==0 || aBr.iWidth==0)
{
    ->AppendL@P@oins(aX+width,aY+height);
}
else{
    =0; TReal stepBr
    ::Sqrt(stepBr^2+aBr.iWidth*aBr.iWidth+aBr.iHeight*aBr.iHeight);
    = ((90/stepBr) ? 1: 90/stepBr);

for (TReal i = 0; i <= 90; i += stepBr)
{
```

## How\_to\_create\_a\_rounded\_rectangle

```
        = i * piRad ;                alpha
        ::Sin(sinalpha, alpha);      Math
        ::Cos(cosalpha, alpha);      Math

        ::Round(X, bottomRight.iX + (MBrhiWidth * cosalpha), 0);
        ::Round(Y, bottomRight.iY + (MBrhiHeight * sinalpha), 0);

        ->AppendL(TPoint(X, Y), rectanglePoints
    }
}

    CleanupSpace // rectanglePoints
return rectanglePoints;

}

// -----
// CRoundedRectangleAppView::Draw()
// Draws the display.
// -----
//
void CRoundedRectangleAppView::Draw(const TRect& /*aRect*/) const
{
    // Get the standard graphics context
    CWindowGSystemGc();

    TRect drawRect;
    Clear(drawRect);

    drawRoundedRectangle(gc, 5, 100, 30, TSize(10, 10), TSize(10, 10), TSize(10, 10), TSize(10, 10), TSize(2, 2), TSize(2, 2));
    drawRoundedRectangle(gc, 10, 100, 30, TSize(10, 10), TSize(10, 10), TSize(10, 10), TSize(10, 10), TSize(1, 1), TSize(1, 1));

    drawRoundedRectangle(gc, 15, 100, 30, TSize(0, 0), TSize(16, 21), TSize(0, 0), TSize(16, 21), TSize(1, 1), TSize(1, 1), TSize(1, 1), TSize(1, 1));
    drawRoundedRectangle(gc, 10, 100, 30, TSize(14, 19), TSize(0, 0), TSize(14, 19), TSize(0, 0), TSize(1, 1), TSize(1, 1), TSize(1, 1), TSize(1, 1));

    drawRoundedRectangle(gc, 15, 100, 30, TSize(0, 0), TSize(15, 20), TSize(0, 0), TSize(15, 20), TSize(1, 1), TSize(1, 1), TSize(1, 1), TSize(1, 1));
    drawRoundedRectangle(gc, 10, 100, 30, TSize(15, 20), TSize(0, 0), TSize(15, 20), TSize(0, 0), TSize(1, 1), TSize(1, 1), TSize(1, 1), TSize(1, 1));

    drawRoundedRectangle(gc, 25, 100, 30, TSize(0, 0), TSize(0, 0), TSize(15, 15), TSize(15, 15), TSize(1, 1), TSize(1, 1), TSize(1, 1), TSize(1, 1));
    drawRoundedRectangle(gc, 25, 100, 30, TSize(15, 15), TSize(15, 15), TSize(0, 0), TSize(0, 0), TSize(1, 1), TSize(1, 1), TSize(1, 1), TSize(1, 1));

    //commented out as these use images that are not loaded in this example
    //drawRoundedRectangle(gc, 110, 5, 150, 150, TSize(300, 200), TSize(400, 200), TSize(400, 300), TSize(400, 300), TSize(1, 1), TSize(1, 1), TSize(1, 1), TSize(1, 1));
    //drawRoundedRectangle(gc, 110, 200, 200, 200, TSize(30, 20), TSize(150, 150), TSize(40, 30), TSize(40, 30), TSize(1, 1), TSize(1, 1), TSize(1, 1), TSize(1, 1));
}
}
```

## Considerations

- Code is not fully tested.
- Converting from calculated TReal values to TInt may cause some graphic bugs.
- Since we are talking about graphics there is always room for optimization