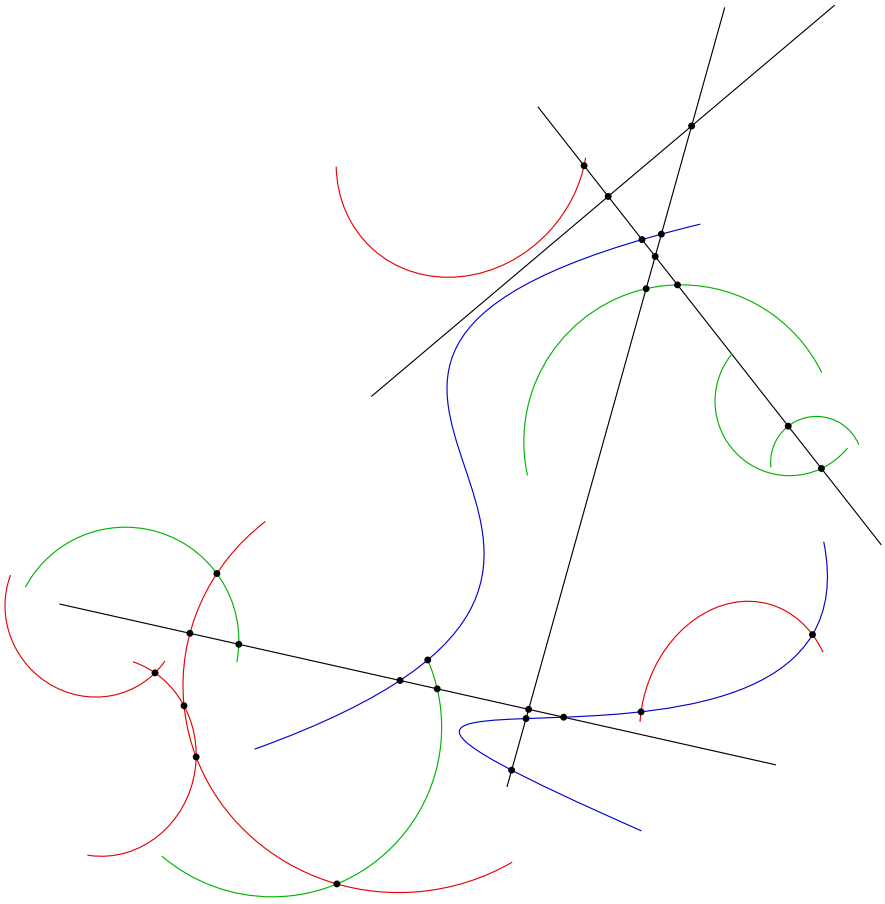
2D graphics primitives

Indrek Mandre <indrek@mare.ee>
http://www.mare.ee/indrek/misc/

July 11, 2009

Abstract

As I started writing my own little 2D graphics library, I realized that with very little extra effort I could write all the techniques and formulae down into this little document. This here is written to be between theory and practical application, so I don't quite write raw source code but rather concentrate on the equations and how they come. Sample code is provided to calculate the more complex polynomial coefficients as writing them down as formulae is neither useful nor interesting. All in all I hope you find this document useful, at least in finding the right path.



Contents

1 2D objects

1.1 Line, ray, line segment

Lines can be defined through two points or better through a point $\mathbf{p}_0$ and a normalized vector $\hat{\mathbf{n}}$ on the line.

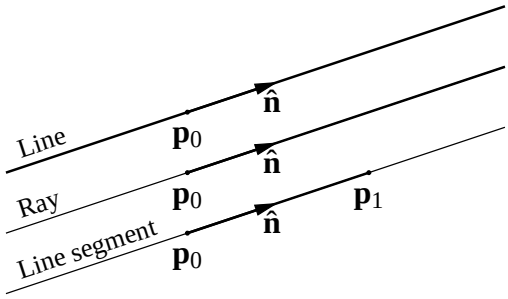


Figure 1: Line, ray, line segment.

The parametric equation for a line is

$$\mathbf{x} = \mathbf{p}_0 + t\hat{\mathbf{n}}, \qquad t = (-\infty, \infty).$$

The parametric equation for a ray is

$$\mathbf{x} = \mathbf{p}_0 + t\hat{\mathbf{n}}, \qquad t = [0, \infty).$$

The parametric equation for a line segment is

$$\mathbf{x} = \mathbf{p}_0 + t\hat{\mathbf{n}}, \qquad t = [0, |\mathbf{p}_1 - \mathbf{p}_0|].$$

1.1.1 Line segment length

$$l = |\mathbf{p}_1 - \mathbf{p}_0|.$$

1.1.2 Distance of a point from a line

Given point $\mathbf{x}$ the distance from the line is:

$$d = |\mathbf{p}_0 + ((\mathbf{x} - \mathbf{p}_0) \cdot \hat{\mathbf{n}})\hat{\mathbf{n}} - \mathbf{x}|$$

1.1.3 Determing whether a point is on the line

Testing by the parameter t:

$$\begin{aligned} \text{for a ray:} \qquad & t \geq 0, \\ \text{for a line segment:} \qquad & t \geq 0 \wedge t \leq l. \end{aligned}$$

Testing by a given point $\mathbf{x}$ we first find the parameter t by

$$t = (\mathbf{x} - \mathbf{p}_0) \cdot \hat{\mathbf{n}}.$$

If the point is on the line then the following must hold:

$$|\mathbf{p}_0 + t\hat{\mathbf{n}} - \mathbf{x}| = 0.$$

Additionally for a ray or a line segment we need to make sure that t is in the proper range.

1.2 Circle, arc

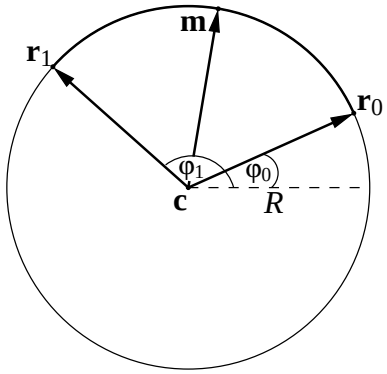


Figure 2: Circle, arc.

Circle can be defined through a center $\mathbf{c} = (x_0, y_0)$ and radius R. The implicit equation for a circle is

$$(x - x_0)^2 + (y - y_0)^2 = R^2.$$

Arc on the circle can be specified by two angles φ_0 and φ_1.

1.2.1 Determing whether a point is on the circle

Given point $\mathbf{x}$ the following must hold true:

$$|\mathbf{x} - \mathbf{c}| = R.$$

Given an arc segment between angles φ_0 and φ_1, we can find the normalized vectors towards the ends of the arc segment:

$$\hat{\mathbf{r}}_0 = (\cos\varphi_0, \sin\varphi_0),$$
$$\hat{\mathbf{r}}_1 = (\cos\varphi_1, \sin\varphi_1)$$

and the vector $\hat{\mathbf{m}}$ splitting the arc:

$$\hat{\mathbf{m}} = \left(\cos\frac{\varphi_0 + \varphi_1}{2}, \sin\frac{\varphi_0 + \varphi_1}{2}\right).$$

If the point is on the circle, then it is also on the specified arc if

$$(\mathbf{x} - \mathbf{c}) \cdot \hat{\mathbf{m}} \geq R\hat{\mathbf{r}}_0 \cdot \hat{\mathbf{m}}.$$

The vector $\hat{\mathbf{m}}$ and the value on the right side can be precalculated for fast evaluation.

1.2.2 Length of an arc

$$l = |\varphi_1 - \varphi_0|R.$$

1.3 Ellipse, elliptic arc

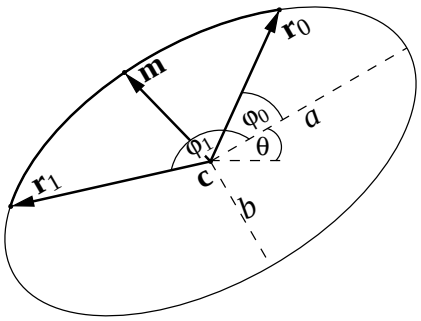


Figure 3: Ellipse, elliptic arc.

Canonical ellipse (one that is at the center of the coordinate system and has not been rotated) has the equation

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1. \tag{1}$$

1.3.1 Implicit equation of the ellipse

Suppose we have an ellipse with radiuses a and b, rotated counter-clockwise by θ and then translated to location $\mathbf{r} = (r_x, r_y)$. The reverse transformation will take a point $\mathbf{p} = (x, y)$ to

$$\mathbf{p}' = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} (\mathbf{p} - \mathbf{r}) = \begin{pmatrix} (x - r_x)\cos\theta + (y - r_y)\sin\theta \\ -(x - r_x)\sin\theta + (y - r_y)\cos\theta \end{pmatrix}.$$

Replacing this into the canonical ellipse equation (1) gets us

$$\frac{((x - r_x)\cos\theta + (y - r_y)\sin\theta)^2}{a^2} + \frac{(-(x - r_x)\sin\theta + (y - r_y)\cos\theta)^2}{b^2} = 1.$$

Expanding this leads to

$$u_{xx}x^2 + u_{xy}xy + u_{yy}y^2 + u_x x + u_y y + u_0 = 0 \tag{2}$$

where the coefficients come as

```
u_xx = a*a*sin(theta)*sin(theta) + b*b*cos(theta)*cos(theta);
u_xy = -2*a*a*cos(theta)*sin(theta)
  + 2*b*b*cos(theta)*sin(theta);
u_yy = a*a*cos(theta)*cos(theta) + b*b*sin(theta)*sin(theta);
u_x = -2*r_y*b*b*cos(theta)*sin(theta)
  + 2*r_y*a*a*cos(theta)*sin(theta)
  - 2*r_x*a*a*sin(theta)*sin(theta)
  - 2*r_x*b*b*cos(theta)*cos(theta);
u_y = -2*r_x*b*b*cos(theta)*sin(theta)
```

```
    + 2*r_x*a*a*cos(theta)*sin(theta)
    - 2*r_y*a*a*cos(theta)*cos(theta)
    - 2*r_y*b*b*sin(theta)*sin(theta);
  u_0 = -2*r_x*r_y*a*a*cos(theta)*sin(theta)
    + 2*r_x*r_y*b*b*cos(theta)*sin(theta) - a*a*b*b
    + a*a*r_x*r_x*sin(theta)*sin(theta)
    + a*a*r_y*r_y*cos(theta)*cos(theta)
    + b*b*r_x*r_x*cos(theta)*cos(theta)
    + b*b*r_y*r_y*sin(theta)*sin(theta);
```

We can also rewrite equation (2) in the following way:

$$u_{xx}x^2 + (u_{xy}y + u_x)x + (u_{yy}y^2 + u_u y + u_0) = 0. \tag{3}$$

1.3.2 Determing whether a point is on the ellipse

Given point $\mathbf{x} = (x, y)$ we can verify that it is on the ellipse by evaluating the implicit ellipse equation (2).

If we are given an arc on the ellipse through angles φ_0 and φ_1, then we can again find the normalized vectors towards the ends of the arc segment:

$$\hat{\mathbf{r}}_0 = (\cos\varphi_0 + \theta, \sin\varphi_0 + \theta),$$
$$\hat{\mathbf{r}}_1 = (\cos\varphi_1 + \theta, \sin\varphi_1 + \theta)$$

and the vector $\hat{\mathbf{m}}$ splitting the arc:

$$\hat{\mathbf{m}} = \left(\cos\frac{\varphi_0 + \varphi_1 + 2\theta}{2}, \sin\frac{\varphi_0 + \varphi_1 + 2\theta}{2}\right).$$

If the point is on the circle, then it is also on the specified arc if

$$(\mathbf{x} - \mathbf{c}) \cdot \hat{\mathbf{m}} \geq |\mathbf{x} - \mathbf{c}| \hat{\mathbf{r}}_0 \cdot \hat{\mathbf{m}}.$$

1.3.3 Finding the vectors $\mathbf{r}_0$ and $\mathbf{r}_1$

Given a canonical ellipse, the endpoints are located towards normalized vectors $\hat{\mathbf{r}}_0 = (r_{0x}, r_{0y})$ and $\hat{\mathbf{r}}_1 = (r_{1x}, r_{1y})$ given by

$$\hat{\mathbf{r}}_0 = (\cos\varphi_0, \sin\varphi_0),$$
$$\hat{\mathbf{r}}_1 = (\cos\varphi_1, \sin\varphi_1).$$

We can get vectors $\mathbf{r}_0$ and $\mathbf{r}_1$ by intersecting a line in the direction of an endpont **($\mathbf{x} = \hat{\mathbf{r}}_0 t$)** with the ellipse. This can be done by solving the following formula:

$$\frac{(r_{0x}t)^2}{a^2} + \frac{(r_{0y}t)^2}{b^2} = 1,$$

$$r_{0x}^2 b^2 t^2 + r_{0y}^2 a^2 t^2 = a^2 b^2$$

From here we only have use for the positive t

$$t = \sqrt{\frac{a^2 b^2}{a^2 r_{0y}^2 + b^2 r_{0x}^2}}$$

and the point is

$$\mathbf{r}_0 = t\hat{\mathbf{r}}_0 = \sqrt{\frac{a^2b^2}{a^2r_{0y}^2 + b^2r_{0x}^2}}\hat{\mathbf{r}}_0.$$

For an arbitrary ellipse with rotation θ the formulae are

$$\mathbf{r}_0 = \sqrt{\frac{a^2b^2}{a^2r_{0y}^2 + b^2r_{0x}^2}} \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \hat{\mathbf{r}}_0,$$
$$\mathbf{r}_1 = \sqrt{\frac{a^2b^2}{a^2r_{1y}^2 + b^2r_{1x}^2}} \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \hat{\mathbf{r}}_1.$$

1.3.4 Length of an elliptic arc

Given ellipse with a paramentric equation in canonical form is

$$x(t) = a\cos t,$$
$$y(t) = b\sin t.$$

Then the length is

$$\begin{aligned} l &= \int ds \\ &= \int \sqrt{dx^2 + dy^2} \\ &= \int_{t_0}^{t_1} \sqrt{a^2\sin^2 t + b^2\cos^2 t}\,dt. \end{aligned}$$

While this is an ellipse and that's an integral - we have one of the famous unsolvable elliptic integrals. So we must use numeric integration again. What is left to determine are the integration limits. As we can calculate the points at our arc ends, the value t is simply

$$t_0 = \cos^{-1}\frac{x_{\varphi_0}}{a} \quad \text{if} \quad y_{\varphi_0} \geq 0,$$

or

$$t_0 = 2\pi - \cos^{-1}\frac{x_{\varphi_0}}{a} \quad \text{if} \quad y_{\varphi_0} < 0.$$

1.3.5 Normal along the line

We again find the parameter t corresponding to position $\mathbf{x}$ just as in 1.3.4 and then just use the parametric equation:

$$\hat{\mathbf{n}} = \frac{\left(\frac{\mathbf{dx}}{\mathbf{dt}}, \frac{\mathbf{dy}}{\mathbf{dt}}\right)}{\left|\left(\frac{\mathbf{dx}}{\mathbf{dt}}, \frac{\mathbf{dy}}{\mathbf{dt}}\right)\right|} = \frac{(-a\sin t, b\cos t)}{\sqrt{a^2\sin^2 t + b^2\cos^2 t}}.$$

1.4 Cubic bezier curve

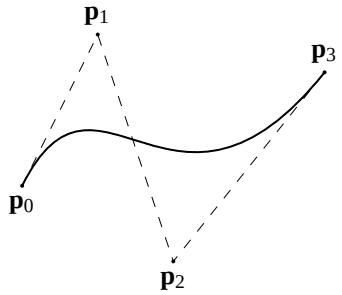


Figure 4: Cubic bezier curve.

Cubic bezier curve is parametrically defined through four points $\mathbf{p}_k = (p_{xk}, p_{yk})$ as:

$$\mathbf{x}(t) = (1-t)^3\mathbf{p}_0 + 3(1-t)^2 t\mathbf{p}_1 + 3(1-t)t^2\mathbf{p}_2 + t^3\mathbf{p}_3, \quad t \in [0,1]. \tag{4}$$

Sometimes it is useful to employ this form instead

$$\begin{cases} x = a_3t^3 + a_2t^2 + a_1t + a_0, \\ y = b_3t^3 + b_2t^2 + b_1t + b_0 \end{cases}$$

where the coefficients are

```
a3 = p_x3 - p_x0 - 3*p_x2 + 3*p_x1;
a2 = -6*p_x1 + 3*p_x0 + 3*p_x2;
a1 = -3*p_x0 + 3*p_x1;
a0 = p_x0;
b3 = p_y3 - p_y0 - 3*p_y2 + 3*p_y1;
b2 = -6*p_y1 + 3*p_y0 + 3*p_y2;
b1 = -3*p_y0 + 3*p_y1;
b0 = p_y0;
```

1.4.1 Determing whether a point is on the cubic bezier curve

If the point is given parametrically by t, then it just has to be in the interval $[0,1]$. If we are given a points $\mathbf{x} = (x,y)$ then we need to solve the equation

$$(x - a_3t^3 + a_2t^2 + a_1t + a_0)^2 + (y - b_3t^3 + b_2t^2 + b_1t + b_0)^2 = 0$$

for t. If the t is real and within the interval $[0,1]$, then the point is on the line. This equation expands to

$$u_6t^6 + u_5t^5 + u_4t^4 + u_3t^3 + u_2t^2 + u_1t + u_0 = 0$$

where the coefficients are

```
u6 = a3*a3 + b3*b3;
u5 = 2*a2*a3 + 2*b2*b3;
u4 = 2*a1*a3 + 2*b1*b3 + a2*a2 + b2*b2;
u3 = -2*a3*x - 2*b3*y + 2*a0*a3 + 2*a1*a2 + 2*b0*b3 + 2*b1*b2;
u2 = -2*a2*x - 2*b2*y + 2*a0*a2 + 2*b0*b2 + a1*a1 + b1*b1;
u1 = -2*a1*x - 2*b1*y + 2*a0*a1 + 2*b0*b1;
u0 = -2*a0*x - 2*b0*y + a0*a0 + b0*b0 + x*x + y*y;
```

1.4.2 Implicitization of the cubic bezier curve

We can give the parameterized cubic bezier curve the following form:

$$\begin{cases} a_3t^3 + a_2t^2 + a_1t + a_0 - x = 0, \\ b_3t^3 + b_2t^2 + b_1t + b_0 - y = 0. \end{cases}$$

We want to turn this into an implicit curve equation, that is one where we have no dependance on the parameter t:

$$f(x,y) = 0.$$

This can be done using the resultant (or the Bezout's determinant) to eliminate variable t. As a result we get the following equation:

$$v_{xxx}x^3 + v_{xxy}x^2y + v_{xyy}xy^2 + v_{yyy}y^3 + v_{xx}x^2 + v_{xy}xy + v_{yy}y^2 + v_xx + v_yy + v_0 = 0$$

where

```
v_xxx = b3*b3*b3;
v_xxy = -3*a3*b3*b3;
v_xyy = 3*b3*a3*a3;
v_yyy = -a3*a3*a3;
v_xx = -3*a3*b1*b2*b3 + a1*b2*b3*b3 - a2*b3*b2*b2
  + 2*a2*b1*b3*b3 + 3*a3*b0*b3*b3 + a3*b2*b2*b2 - 3*a0*b3*b3*b3;
v_xy = a1*a3*b2*b3 - a2*a3*b1*b3 - 6*b0*b3*a3*a3
  - 3*a1*a2*b3*b3 - 2*a2*a3*b2*b2 + 2*b2*b3*a2*a2
  + 3*b1*b2*a3*a3 + 6*a0*a3*b3*b3;
v_yy = 3*a1*a2*a3*b3 + a3*b2*a2*a2 - a2*b1*a3*a3
  - 3*a0*b3*a3*a3 - 2*a1*b2*a3*a3 - b3*a2*a2*a2 + 3*b0*a3*a3*a3;
v_x = a2*a3*b0*b1*b3 - a1*a2*b1*b2*b3 - a1*a3*b0*b2*b3
  + 6*a0*a3*b1*b2*b3 + b1*a1*a1*b3*b3 + b3*a2*a2*b1*b1
  + 3*b3*a3*a3*b0*b0 + a1*a3*b1*b2*b2 - a2*a3*b2*b1*b1
  - 6*a0*a3*b0*b3*b3 - 4*a0*a2*b1*b3*b3 - 3*b0*b1*b2*a3*a3
  - 2*a0*a1*b2*b3*b3 - 2*a1*a3*b3*b1*b1 - 2*b0*b2*b3*a2*a2
  + 2*a0*a2*b3*b2*b2 + 2*a2*a3*b0*b2*b2 + 3*a1*a2*b0*b3*b3
  + a3*a3*b1*b1*b1 + 3*a0*a0*b3*b3*b3 - 2*a0*a3*b2*b2*b2;
v_y = a0*a2*a3*b1*b3 + a1*a2*a3*b1*b2 - a0*a1*a3*b2*b3
  - 6*a1*a2*a3*b0*b3 - a1*a1*a1*b3*b3 - 3*a3*a3*a3*b0*b0
  - a1*a3*a3*b1*b1 - a3*a1*a1*b2*b2 - 3*a3*a0*a0*b3*b3
  + a2*b2*b3*a1*a1 - a1*b1*b3*a2*a2 - 3*a0*b1*b2*a3*a3
  - 2*a0*b2*b3*a2*a2 - 2*a3*b0*b2*a2*a2 + 2*a0*a2*a3*b2*b2
  + 2*a2*b0*b1*a3*a3 + 2*a3*b1*b3*a1*a1 + 3*a0*a1*a2*b3*b3
  + 4*a1*b0*b2*a3*a3 + 6*a0*b0*b3*a3*a3 + 2*b0*b3*a2*a2*a2;
v_0 = a0*a1*a2*b1*b2*b3 + a0*a1*a3*b0*b2*b3 - a0*a2*a3*b0*b1*b3
  - a1*a2*a3*b0*b1*b2 + b0*a1*a1*a1*b3*b3 - b3*a2*a2*a2*b0*b0
  + a1*b0*a3*a3*b1*b1 + a1*b2*a0*a0*b3*b3 + a3*b0*a1*a1*b2*b2
  + a3*b2*a2*a2*b0*b0 - a0*b1*a1*a1*b3*b3 - a0*b3*a2*a2*b1*b1
  - a2*b1*a3*a3*b0*b0 - a2*b3*a0*a0*b2*b2 - 3*a0*b3*a3*a3*b0*b0
  - 2*a1*b2*a3*a3*b0*b0 + 2*a2*b1*a0*a0*b3*b3
  + 3*a3*b0*a0*a0*b3*b3 + a0*a2*a3*b2*b1*b1 + a1*b0*b1*b3*a2*a2
  - a0*a1*a3*b1*b2*b2 - a2*b0*b2*b3*a1*a1 - 3*a0*a1*a2*b0*b3*b3
  - 3*a3*b1*b2*b3*a0*a0 - 2*a0*a2*a3*b0*b2*b2
  - 2*a3*b0*b1*b3*a1*a1 + 2*a0*a1*a3*b3*b1*b1
  + 2*a0*b0*b2*b3*a2*a2 + 3*a0*b0*b1*b2*a3*a3
  + 3*a1*a2*a3*b3*b0*b0 + a3*a3*a3*b0*b0*b0 - a0*a0*a0*b3*b3*b3
  + a3*a0*a0*b2*b2*b2 - a0*a3*a3*b1*b1*b1;
```

1.4.3 Splitting the cubic bezier curve

We want to split the bezier curve into two curves at parametric point t. We apply the de Casteljau algorithm:

$$\begin{aligned}
\mathbf{p}_0^1 &= (1-t)\mathbf{p}_0 + t\mathbf{p}_1,\\
\mathbf{p}_1^1 &= (1-t)\mathbf{p}_1 + t\mathbf{p}_2,\\
\mathbf{p}_2^1 &= (1-t)\mathbf{p}_2 + t\mathbf{p}_3,\\
\mathbf{p}_0^2 &= (1-t)\mathbf{p}_0^1 + t\mathbf{p}_1^1,\\
\mathbf{p}_1^2 &= (1-t)\mathbf{p}_1^1 + t\mathbf{p}_2^1,\\
\mathbf{p}_0^3 &= (1-t)\mathbf{p}_0^2 + t\mathbf{p}_1^2
\end{aligned}$$

Here the splitting point is $\mathbf{p}_0^3$ and the two new cubic bezier curves are:

$$\begin{aligned}
\mathbf{x}_1(t) &= (1-t)^3\mathbf{p}_0 + 3(1-t)^2 t\mathbf{p}_0^1 + 3(1-t)t^2\mathbf{p}_0^2 + t^3\mathbf{p}_0^3, \quad t \in [0,1],\\
\mathbf{x}_2(t) &= (1-t)^3\mathbf{p}_0^3 + 3(1-t)^2 t\mathbf{p}_1^2 + 3(1-t)t^2\mathbf{p}_2^1 + t^3\mathbf{p}_3, \quad t \in [0,1].
\end{aligned}$$

1.4.4 Is a cubic bezier curve self-intersecting

We get a pair of equations

$$\begin{cases}
a_3t^3 + a_2t^2 + a_1t + a_0 = a_3s^3 + a_2s^2 + a_1s + a_0,\\
b_3t^3 + b_2t^2 + b_1t + b_0 = b_3s^3 + b_2s^2 + b_1s + b_0.
\end{cases}$$

Simplifying this gets us

$$\begin{cases}
a_3(t^3 - s^3) + a_2(t^2 - s^2) + a_1(t-s) = 0,\\
b_3(t^3 - s^3) + b_2(t^2 - s^2) + b_1(t-s) = 0.
\end{cases}$$

Assuming $t \neq s$ we can divide this by $(t-s)$. This yields us

$$\begin{cases}
a_3(t^2 + st + s^2) + a_2(t+s) + a_1 = 0,\\
b_3(t^2 + st + s^2) + b_2(t+s) + b_1 = 0.
\end{cases}$$

We can reduce this to a polynomial of single variable by using the resultant, so that we get:

$$u_2t^2 + u_1t + u_0 = 0$$

where the coefficients are

```
u2 = -2*a2*a3*b2*b3 + a2*a2*b3*b3 + a3*a3*b2*b2;
u1 = -a1*a3*b2*b3 - a2*a3*b1*b3 + a1*a2*b3*b3 + b1*b2*a3*a3;
u0 = -a1*a2*b2*b3 - a2*a3*b1*b2 - 2*a1*a3*b1*b3 + a1*a1*b3*b3
  + a3*a3*b1*b1 + a1*a3*b2*b2 + b1*b3*a2*a2;
```

We solve this equation and if there are real values in the range $[0,1]$, then the cubic bezier curve is self-intersecting. The intersection points are indicated by the values of t.

1.4.5 Length of a cubic bezier curve

$$\begin{aligned} l &= \int ds \\ &= \int \sqrt{dx^2 + dy^2} \\ &= \int_0^1 \sqrt{(3a_3t^2 + 2a_2t + a_1)^2 + (3b_3t^2 + 2b_2t + b_1)^2}dt. \end{aligned}$$

However, it seems that this integral does not have a closed form solution so one has to integrate it numerically, for example using the Simpson's rule.

1.4.6 Normal along the line

At every point in the line there is a normal vector tangent to the line. It is

$$\hat{\mathbf{n}} = \frac{\left(\frac{\mathbf{dx}}{\mathbf{dt}}, \frac{\mathbf{dy}}{\mathbf{dt}}\right)}{\left|\left(\frac{\mathbf{dx}}{\mathbf{dt}}, \frac{\mathbf{dy}}{\mathbf{dt}}\right)\right|} = \frac{(3a_3t^2 + 2a_2t + a_1, 3b_3t^2 + 2b_2t + b_1)}{\sqrt{(3a_3t^2 + 2a_2t + a_1)^2 + (3b_3t^2 + 2b_2t + b_1)^2}}.$$

2 2D intersections

2.1 Line, line segment and ray intersections

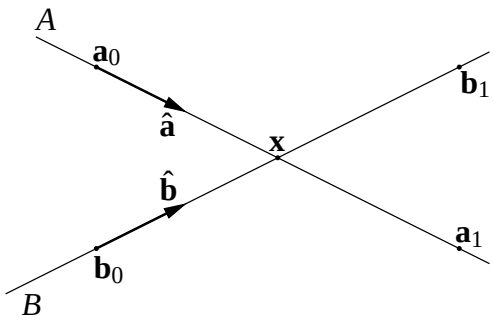


Figure 5: 2D line intersection.

We are given two lines A and B (figure 5) by the parametric equations

$$\begin{aligned} \mathbf{a} &= \mathbf{a}_0 + s\hat{\mathbf{a}}, \\ \mathbf{b} &= \mathbf{b}_0 + t\hat{\mathbf{b}}, \end{aligned}$$

where $\mathbf{a}_0 = (a_{0x}, a_{0y})$ and $\mathbf{b}_0 = (b_{0x}, b_{0y})$ are points on the lines A and B, $\hat{\mathbf{a}} = (a_{nx}, a_{ny})$ and $\hat{\mathbf{b}} = (b_{nx}, b_{ny})$ are normalized vectors along the lines and s and t are the independent parameters. We are interested in the point $\mathbf{x} = (x, y)$ where those two intersect, that is where $\mathbf{a} = \mathbf{b}$. For this the following must hold true:

$$\mathbf{a}_0 + s\hat{\mathbf{a}} = \mathbf{b}_0 + t\hat{\mathbf{b}} \tag{5}$$

and after finding the parameters t or s satisfying this condition we can write

$$\mathbf{x} = \mathbf{a}_0 + s\hat{\mathbf{a}} = \mathbf{b}_0 + s\hat{\mathbf{b}}. \tag{6}$$

Expanding the vectors in (5) we get a system of two equations in two unknowns:

$$\begin{cases} a_{nx}s - b_{nx}t = b_{0x} - a_{0x}, \\ a_{ny}s - b_{ny}t = b_{0y} - a_{0y}. \end{cases}$$

From here

$$s = \frac{\begin{vmatrix} b_{0x} - a_{0x} & -b_{nx} \\ b_{0y} - a_{0y} & -b_{ny} \end{vmatrix}}{\begin{vmatrix} a_{nx} & -b_{nx} \\ a_{ny} & -b_{ny} \end{vmatrix}} = \frac{(b_{0y} - a_{0y})\, b_{nx} - (b_{0x} - a_{0x})\, b_{ny}}{a_{ny}b_{nx} - a_{nx}b_{ny}},$$

$$t = \frac{\begin{vmatrix} a_{nx} & b_{0x} - a_{0x} \\ a_{ny} & b_{0y} - a_{0y} \end{vmatrix}}{\begin{vmatrix} a_{nx} & -b_{nx} \\ a_{ny} & -b_{ny} \end{vmatrix}} = \frac{a_{nx}\,(b_{0y} - a_{0y}) - a_{ny}\,(b_{0x} - a_{0x})}{a_{ny}b_{nx} - a_{nx}b_{ny}}.$$

As $\mathbf{x}$ in equation (6) can be calculated from only one parameter, only one of these is required. However, if we are dealing with rays or line segments then these parameters are necessary to determine if the intersection is on the segment (see section 1.1.3).

2.2 Line-arc intersection

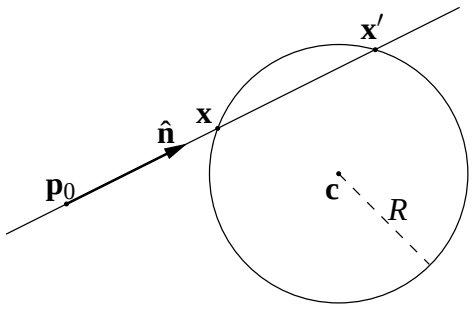


Figure 6: 2D line-arc intersection.

The equation of the circle centered at $\mathbf{c} = (x_0, y_0)$ with radius R is

$$(x - x_0)^2 + (y - y_0)^2 = R^2$$

and the equation of the line is

$$\mathbf{x} = \mathbf{p}_0 + s\hat{\mathbf{n}}.$$

where $\mathbf{p}_0 = (a, b)$ is a point on the line, $\hat{\mathbf{n}} = (n_x, n_y)$ is a normalized vector along the line and s is the parametric variable. Putting the line equation into the circle we get

$$(a + sn_x - x_0)^2 + (b + sn_y - y_0)^2 = R^2.$$

This is a quadratic equation that we can solve for s:

$$((a-x_0)+sn_x)^2+((b-y_0)+sn_y)^2=R^2,$$

$$(A+sn_x)^2+(B+sn_y)^2=R^2,$$

$$A^2+2Asn_x+s^2n_x^2+B^2+2Bsn_y+s^2n_y^2-R^2=0,$$

$$(n_x^2+n_y^2)s^2+s(2An_x+2Bn_y)+A^2+B^2-R^2=0,$$

$$s^2+sC+D=0,$$

$$s=\frac{-C\pm\sqrt{C^2-4D}}{2},$$

$$s=-(An_x+Bn_y)\pm\sqrt{(An_x+Bn_y)^2-A^2-B^2+R^2}.$$

So the points on the circle are

$$\mathbf{x}=\mathbf{p}_0+s\hat{\mathbf{n}}$$

where s is

$$s=-(An_x+Bn_y)\pm\sqrt{(An_x+Bn_y)^2-A^2-B^2+R^2},$$

$$A=(a-x_0),\ B=(b-y_0).$$

In case the expression under the square root is negative, there is no intersection.

2.3 Arc-arc intersections

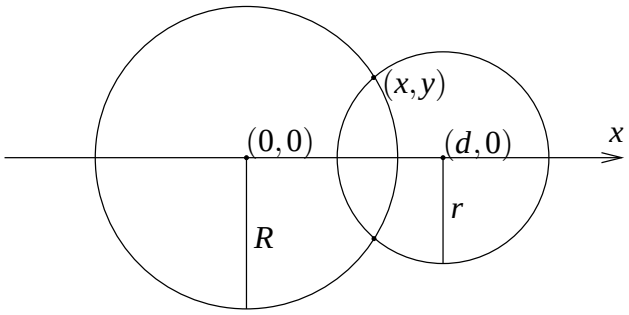


Figure 7: 2D arc-arc intersection.

To simplify things lets put one of the circles to the middle of the coordinate system and another with center on the x-axis. In this case the two circle equations are

$$x^2+y^2=R^2,$$

$$(x-d)^2+y^2=r^2$$

where d is the distance between the centers of the two circles. Combining these two we get

$$(x-d)^2+R^2-x^2-r^2=0.$$

From here we can find x:

$$x^2 - 2xd + d^2 + R^2 - x^2 - r^2 = 0,$$

$$x = \frac{d^2 + R^2 - r^2}{2d}.$$

Finding y is now trivial:

$$y = \pm\sqrt{R^2 - x^2}.$$

Translating these equations for arbitrarily placed circles centered at $\mathbf{c}_1$ and $\mathbf{c}_2$ we get:

$$\mathbf{d} = \mathbf{c}_2 - \mathbf{c}_1,$$

$$d = |\mathbf{d}|,$$

$$x = \frac{d^2 + R^2 - r^2}{2d},$$

$$y = \pm\sqrt{R^2 - x^2},$$

$$\mathbf{x} = \mathbf{c}_1 + \frac{x}{d}\mathbf{d} \pm y \begin{bmatrix} \cos\frac{\pi}{2} & \sin\frac{\pi}{2} \\ -\sin\frac{\pi}{2} & \cos\frac{\pi}{2} \end{bmatrix} \hat{\mathbf{d}},$$

$$\mathbf{x} = \mathbf{c}_1 + \frac{d^2 + R^2 - r^2}{2d^2}\mathbf{d} \pm \sqrt{\frac{R^2 - x^2}{d^2}} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \mathbf{d},$$

$$\mathbf{x} = \mathbf{c}_1 + \frac{d^2 + R^2 - r^2}{2d^2}\mathbf{d} \pm \sqrt{\frac{R^2 - \frac{(d^2+R^2-r^2)^2}{4d^2}}{d^2}} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \mathbf{d}.$$

This can be simplified for calculations such as

$$\mathbf{x} = \mathbf{c}_1 + B\mathbf{d} \pm \sqrt{\frac{R^2 - 0.5AB}{d^2}} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \mathbf{d},$$

$$A = d^2 + R^2 - r^2,$$

$$B = \frac{A}{2d^2}.$$

In case the expression under the square root is negative the circles don't intersect.

2.4 Ellipse-line intersection

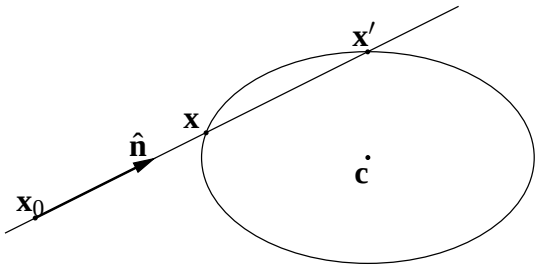


Figure 8: 2D line-ellipse intersection.

The canonical form for an ellipse is

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

where a is the horizontal and b is the vertical radius. Any ellipse can be translated and rotated to take this form. The parametric equation for a line is

$$\mathbf{x}=\mathbf{x}_0+s\hat{\mathbf{n}}$$

where $\mathbf{x}_0=(x_0,y_0)$ is a point on the line and $\hat{\mathbf{n}}=(n_x,n_y)$ is a normalized vector along the line. These two equations can be combined so that we get

$$\frac{(x_0+sn_x)^2}{a^2}+\frac{(y_0+sn_y)^2}{b^2}=1.$$

This is a quadratic equation that we can solve for s:

$$\frac{x_0^2}{a^2}+\frac{2x_0n_x}{a^2}s+\frac{n_x^2}{a^2}s^2+\frac{y_0^2}{b^2}+\frac{2y_0n_y}{b^2}s+\frac{n_y^2}{b^2}s^2-1=0,$$

$$\left(\frac{n_x^2}{a^2}+\frac{n_y^2}{b^2}\right)s^2+\left(\frac{2x_0n_x}{a^2}+\frac{2y_0n_y}{b^2}\right)s+\frac{x_0^2}{a^2}+\frac{y_0^2}{b^2}-1=0,$$

$$(n_x^2b^2+n_y^2a^2)s^2+(2x_0n_xb^2+2y_0n_ya^2)s+x_0^2b^2+y_0^2a^2-a^2b^2=0,$$

$$As^2+Bs+C=0,$$

$$s=\frac{-B\pm\sqrt{B^2-4AC}}{2A}.$$

In case the expression under the square root is negative, there are no intersections.

2.5 Ellipse-ellipse intersection

We are given two ellipses through implicit equations

$$\begin{aligned}u_{xx}x^2+(u_{xy}y+u_x)x+(u_{yy}y^2+u_uy+u_0) &= 0,\\ v_{xx}x^2+(v_{xy}y+v_x)x+(v_{yy}y^2+v_uy+v_0) &= 0.\end{aligned}$$

These are two multivariate polynomials and we want to find their common roots. To do this we eliminate one of the variables using the resultant. The result is the following equation:

$$a_4y^4+a_3y^3+a_2y^2+a_1y+a_0=0$$

where the coefficients come as

```
a4 = -u_xx*u_xy*v_xy*v_yy - u_xy*u_yy*v_xx*v_xy
  - 2*u_xx*u_yy*v_xx*v_yy + u_xx*u_xx*v_yy*v_yy
  + u_yy*u_yy*v_xx*v_xx + u_xx*u_yy*v_xy*v_xy
  + v_xx*v_yy*u_xy*u_xy;
a3 = -u_x*u_xx*v_xy*v_yy - u_x*u_yy*v_xx*v_xy
  - u_xx*u_xy*v_x*v_yy - u_xx*u_xy*v_xy*v_y
  - u_xy*u_y*v_xx*v_xy - u_xy*u_yy*v_x*v_xx
  - 2*u_xx*u_y*v_xx*v_yy - 2*u_xx*u_yy*v_xx*v_y
  + 2*u_x*u_xy*v_xx*v_yy + 2*u_xx*u_yy*v_x*v_xy
```

```
    + u_xx*u_y*v_xy*v_xy + v_xx*v_y*u_xy*u_xy
    + 2*u_y*u_yy*v_xx*v_xx + 2*v_y*v_yy*u_xx*u_xx;
  a2 = -u_0*u_xy*v_xx*v_xy - u_x*u_xx*v_x*v_yy
    - u_x*u_xx*v_xy*v_y - u_x*u_y*v_xx*v_xy - u_x*u_yy*v_x*v_xx
    - u_xx*u_xy*v_0*v_xy - u_xx*u_xy*v_x*v_y - u_xy*u_y*v_x*v_xx
    - 2*u_0*u_xx*v_xx*v_yy - 2*u_xx*u_y*v_xx*v_y
    - 2*u_xx*u_yy*v_0*v_xx + 2*u_x*u_xy*v_xx*v_y
    + 2*u_xx*u_y*v_x*v_xy + u_xx*u_xx*v_y*v_y + u_y*u_y*v_xx*v_xx
    + u_0*u_xx*v_xy*v_xy + u_xx*u_yy*v_x*v_x + v_0*v_xx*u_xy*u_xy
    + v_xx*v_yy*u_x*u_x + 2*u_0*u_yy*v_xx*v_xx
    + 2*v_0*v_yy*u_xx*u_xx;
  a1 = -u_0*u_x*v_xx*v_xy - u_0*u_xy*v_x*v_xx - u_x*u_xx*v_0*v_xy
    - u_x*u_xx*v_x*v_y - u_x*u_y*v_x*v_xx - u_xx*u_xy*v_0*v_x
    - 2*u_0*u_xx*v_xx*v_y - 2*u_xx*u_y*v_0*v_xx
    + 2*u_0*u_xx*v_x*v_xy + 2*u_x*u_xy*v_0*v_xx
    + u_xx*u_y*v_x*v_x + v_xx*v_y*u_x*u_x + 2*u_0*u_y*v_xx*v_xx
    + 2*v_0*v_y*u_xx*u_xx;
  a0 = -u_0*u_x*v_x*v_xx - u_x*u_xx*v_0*v_x - 2*u_0*u_xx*v_0*v_xx
    + u_0*u_0*v_xx*v_xx + u_xx*u_xx*v_0*v_0 + u_0*u_xx*v_x*v_x
    + v_0*v_xx*u_x*u_x;
```

We solve the equation, discard all complex values and get a series of possible y. We replace these into

$$u_{xx}x^2 + (u_{xy}y + u_x)x + (u_{yy}y^2 + u_u y + u_0) = 0$$

and solve it to get matching x-s. Finally we check that the points (x,y) we get satisfy

$$v_{xx}x^2 + (v_{xy}y + v_x)x + (v_{yy}y^2 + v_u y + v_0) = 0.$$

Once this has passed we have arrived to our intersection points (x,y). Ellipses can have up to four different intersection points.

Note that for circle arc-elliptic arc intersection we can just convert the circle into an ellipse and apply this method.

2.6 Cubic bezier-line intersection

This can be tackled the following way. We translate and rotate the line and the bezier curve so that the line is on the x-axis. This gives us a simple equation for now the y-component of the bezier parametric equation must be 0 at the intersecting points.

Suppose we have a line defined through a point $\mathbf{x}_0$ as

$$\mathbf{x} = \mathbf{x}_0 + s\hat{\mathbf{n}}$$

where $\hat{\mathbf{n}} = (n_x, n_y)$ is a normalized vector along the line. The transformation matrix to rotate the cubic bezier's control points would be

$$A = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} n_x & -n_y \\ n_y & n_x \end{bmatrix}.$$

We would transform the bezier curve control points like this:

$$\mathbf{p}'_k = A(\mathbf{p}_k - \mathbf{x}_0)$$

where $\mathbf{p}'_k = (a_k, b_k)$.

From the control points we only have use for the y-component and form the following equation:

$$(1-t)^3 b_0 + 3(1-t)^2 t b_1 + 3(1-t)t^2 b_2 + t^3 b_3 = 0$$

Expanding this gets us

$$(b_3 + 3b_1 - 3b_2 - b_0)t^3 + (3b_0 - 6b_1 + 3b_2)t^2 + (3b_1 - 3b_0)t + b_0 = 0$$

which is a polynomial in t. Solving this, and limiting for real cases in the interval $[0,1]$, gets us the locations of the intersecting points.

2.7 Cubic bezier-ellipse intersection

The implicit equation of the ellipse is

$$u_{xx}x^2 + u_{xy}xy + u_{yy}y^2 + u_x x + u_y y + u_0 = 0.$$

The bezier curve is defined parametrically as

$$\begin{cases} x = a_3 t^3 + a_2 t^2 + a_1 t + a_0, \\ y = b_3 t^3 + b_2 t^2 + b_1 t + b_0. \end{cases}$$

What we can do is substitute the x and y in the ellipse's implicit equation with the ones defined by the bezier curve. We get a degree 6 polynomial:

$$v_6 t^6 + v_5 t^5 + v_4 t^4 + v_3 t^3 + v_2 t^2 + v_1 t + v_0 = 0$$

where the coefficients are

```
v6 = a3*b3*u_xy + u_xx*a3*a3 + u_yy*b3*b3;
v5 = a2*b3*u_xy + a3*b2*u_xy + 2*a2*a3*u_xx + 2*b2*b3*u_yy;
v4 = a1*b3*u_xy + a2*b2*u_xy + a3*b1*u_xy + 2*a1*a3*u_xx
  + 2*b1*b3*u_yy + u_xx*a2*a2 + u_yy*b2*b2;
v3 = a3*u_x + b3*u_y + a0*b3*u_xy + a1*b2*u_xy + a2*b1*u_xy
  + a3*b0*u_xy + 2*a0*a3*u_xx + 2*a1*a2*u_xx + 2*b0*b3*u_yy
  + 2*b1*b2*u_yy;
v2 = a2*u_x + b2*u_y + a0*b2*u_xy + a1*b1*u_xy + a2*b0*u_xy
  + 2*a0*a2*u_xx + 2*b0*b2*u_yy + u_xx*a1*a1 + u_yy*b1*b1;
v1 = a1*u_x + b1*u_y + a0*b1*u_xy + a1*b0*u_xy + 2*a0*a1*u_xx
  + 2*b0*b1*u_yy;
v0 = u_0 + a0*u_x + b0*u_y + a0*b0*u_xy + u_xx*a0*a0
  + u_yy*b0*b0;
```

We solve the polynomial for roots, discarding all complex values and values outside $[0,1]$.

2.8 Cubic bezier-cubic bezier intersection

This can be done by using an implicitized form for one of the bezier curves:

$$u_{xxx}x^3 + u_{xxy}x^2 y + u_{xyy}xy^2 + u_{yyy}y^3 + u_{xx}x^2 + u_{xy}xy + u_{yy}y^2 + u_x x + u_y y + u_0 = 0$$

and substituting x and y from the other bezier curve:

$$\begin{cases} x = a_3 t^3 + a_2 t^2 + a_1 t + a_0, \\ y = b_3 t^3 + b_2 t^2 + b_1 t + b_0. \end{cases}$$

We'll get a polynomial in t that we can solve for roots:

$$v_9 t^9 + v_8 t^8 + ... + v_0 = 0$$

where the coefficients are

```
v9 = a3*u_xyy*b3*b3 + b3*u_xxy*a3*a3 + u_xxx*a3*a3*a3
  + u_yyy*b3*b3*b3;
v8 = 2*a2*a3*b3*u_xxy + 2*a3*b2*b3*u_xyy + a2*u_xyy*b3*b3
  + b2*u_xxy*a3*a3 + 3*a2*u_xxx*a3*a3 + 3*b2*u_yyy*b3*b3;
v7 = 2*a1*a3*b3*u_xxy + 2*a2*a3*b2*u_xxy + 2*a2*b2*b3*u_xyy
  + 2*a3*b1*b3*u_xyy + a1*u_xyy*b3*b3 + a3*u_xyy*b2*b2
  + b1*u_xxy*a3*a3 + b3*u_xxy*a2*a2 + 3*a1*u_xxx*a3*a3
  + 3*a3*u_xxx*a2*a2 + 3*b1*u_yyy*b3*b3 + 3*b3*u_yyy*b2*b2;
v6 = a3*b3*u_xy + 2*a0*a3*b3*u_xxy + 2*a1*a2*b3*u_xxy
  + 2*a1*a3*b2*u_xxy + 2*a1*b2*b3*u_xyy + 2*a2*a3*b1*u_xxy
  + 2*a2*b1*b3*u_xyy + 2*a3*b0*b3*u_xyy + 2*a3*b1*b2*u_xyy
  + 6*a1*a2*a3*u_xxx + 6*b1*b2*b3*u_yyy + u_xx*a3*a3
  + u_yy*b3*b3 + a0*u_xyy*b3*b3 + a2*u_xyy*b2*b2
  + b0*u_xxy*a3*a3 + b2*u_xxy*a2*a2 + 3*a0*u_xxx*a3*a3
  + 3*b0*u_yyy*b3*b3 + u_xxx*a2*a2*a2 + u_yyy*b2*b2*b2;
v5 = a2*b3*u_xy + a3*b2*u_xy + 2*a2*a3*u_xx + 2*b2*b3*u_yy
  + 2*a0*a2*b3*u_xxy + 2*a0*a3*b2*u_xxy + 2*a0*b2*b3*u_xyy
  + 2*a1*a2*b2*u_xxy + 2*a1*a3*b1*u_xxy + 2*a1*b1*b3*u_xyy
  + 2*a2*a3*b0*u_xxy + 2*a2*b0*b3*u_xyy + 2*a2*b1*b2*u_xyy
  + 2*a3*b0*b2*u_xyy + 6*a0*a2*a3*u_xxx + 6*b0*b2*b3*u_yyy
  + a1*u_xyy*b2*b2 + a3*u_xyy*b1*b1 + b1*u_xxy*a2*a2
  + b3*u_xxy*a1*a1 + 3*a1*u_xxx*a2*a2 + 3*a3*u_xxx*a1*a1
  + 3*b1*u_yyy*b2*b2 + 3*b3*u_yyy*b1*b1;
v4 = a1*b3*u_xy + a2*b2*u_xy + a3*b1*u_xy + 2*a1*a3*u_xx
  + 2*b1*b3*u_yy + 2*a0*a1*b3*u_xxy + 2*a0*a2*b2*u_xxy
  + 2*a0*a3*b1*u_xxy + 2*a0*b1*b3*u_xyy + 2*a1*a2*b1*u_xxy
  + 2*a1*a3*b0*u_xxy + 2*a1*b0*b3*u_xyy + 2*a1*b1*b2*u_xyy
  + 2*a2*b0*b2*u_xyy + 2*a3*b0*b1*u_xyy + 6*a0*a1*a3*u_xxx
  + 6*b0*b1*b3*u_yyy + u_xx*a2*a2 + u_yy*b2*b2 + a0*u_xyy*b2*b2
  + a2*u_xyy*b1*b1 + b0*u_xxy*a2*a2 + b2*u_xxy*a1*a1
  + 3*a0*u_xxx*a2*a2 + 3*a2*u_xxx*a1*a1 + 3*b0*u_yyy*b2*b2
  + 3*b2*u_yyy*b1*b1;
v3 = a3*u_x + b3*u_y + a0*b3*u_xy + a1*b2*u_xy + a2*b1*u_xy
  + a3*b0*u_xy + 2*a0*a3*u_xx + 2*a1*a2*u_xx + 2*b0*b3*u_yy
  + 2*b1*b2*u_yy + 2*a0*a1*b2*u_xxy + 2*a0*a2*b1*u_xxy
  + 2*a0*a3*b0*u_xxy + 2*a0*b0*b3*u_xyy + 2*a0*b1*b2*u_xyy
  + 2*a1*a2*b0*u_xxy + 2*a1*b0*b2*u_xyy + 2*a2*b0*b1*u_xyy
  + 6*a0*a1*a2*u_xxx + 6*b0*b1*b2*u_yyy + a1*u_xyy*b1*b1
  + a3*u_xyy*b0*b0 + b1*u_xxy*a1*a1 + b3*u_xxy*a0*a0
  + 3*a3*u_xxx*a0*a0 + 3*b3*u_yyy*b0*b0 + u_xxx*a1*a1*a1
  + u_yyy*b1*b1*b1;
v2 = a2*u_x + b2*u_y + a0*b2*u_xy + a1*b1*u_xy + a2*b0*u_xy
  + 2*a0*a2*u_xx + 2*b0*b2*u_yy + 2*a0*a1*b1*u_xxy
  + 2*a0*a2*b0*u_xxy + 2*a0*b0*b2*u_xyy + 2*a1*b0*b1*u_xyy
  + u_xx*a1*a1 + u_yy*b1*b1 + a0*u_xyy*b1*b1 + a2*u_xyy*b0*b0
  + b0*u_xxy*a1*a1 + b2*u_xxy*a0*a0 + 3*a0*u_xxx*a1*a1
  + 3*a2*u_xxx*a0*a0 + 3*b0*u_yyy*b1*b1 + 3*b2*u_yyy*b0*b0;
v1 = a1*u_x + b1*u_y + a0*b1*u_xy + a1*b0*u_xy + 2*a0*a1*u_xx
  + 2*b0*b1*u_yy + 2*a0*a1*b0*u_xxy + 2*a0*b0*b1*u_xyy
  + a1*u_xyy*b0*b0 + b1*u_xxy*a0*a0 + 3*a1*u_xxx*a0*a0
  + 3*b1*u_yyy*b0*b0;
v0 = u_0 + a0*u_x + b0*u_y + a0*b0*u_xy + u_xx*a0*a0
  + u_yy*b0*b0 + a0*u_xyy*b0*b0 + b0*u_xxy*a0*a0
  + u_xxx*a0*a0*a0 + u_yyy*b0*b0*b0;
```

We solve this for roots, discard the complex roots and values outside the region $[0,1]$ and then calculate the point using the parametric form. After that we need to make sure that the point is actually on the first curve. This can be done as specified in section 1.4.1.

2.9 Refining the intersections

Intersections involve calculations that can be numerically unstable. This comes out very clearly for the cubic bezier curve-cubic bezier curve intersections. If we require the cubic bezier curve to be sufficiently smooth, we can use a simple trick to refine the intersection point.

Having an intersection point we can take its parametric locations on the two curves. If the curves are smooth, then zooming in at the intersection point one can see that they behave very closely to straight lines there. We can calculate the normals and do a simple line-line intersection to refine the intersection point.

This simple technique can improve the precision by several orders of magnitude in one step.

A Polynomials and resultants

I'm not going to go into much detail on how to solve polynomials or calculate resultants. If you need to look for a nice algorithm with example code please find "Numeric Recipes: the Art of Scientific Computing". For example the Laguerre's method is nice for finding the roots and you can use the Newton-Raphson to polish the real roots.

Calculating resultants can be done using a CAS (computer algebra system).

B Code generation routines

Most of the indexes in the equations above were generated using python and Sympy. Sympy is a pretty nice CAS (computer algebra system). Here is the code that was used:

```
from __future__ import division
from sympy import *
import re

def spliteqstr(s,n):
  idx = max (s.rfind('+', 0, n), s.rfind('-', 0, n))
  if idx == -1: idx = len(s)
  return (s[0:idx], s[idx:])

def printpolycode(varn, P, n):
  el = []
  for i in range(n,-1,-1):
    el.append ((varn + str(i), P.coeff(i)))
  printeqlist(el)

def printeqlist(el):
  for (varn, eq) in el:
    s = str(varn) + '␣=␣' + str(eq) + ';'
    s = re.sub (r'([a-z]+[a-z0-9\(\)␣]*)\*\*3', r'\1*\1*\1', s)
    s = re.sub (r'([a-z]+[a-z0-9\(\)␣]*)\*\*2', r'\1*\1', s)
    while len(s) > 65:
      (s, rest) = spliteqstr (s, 65)
      print s
      s = '␣␣' + rest
    print s

def implicit_el():
  x, y, theta, a, b = map(Symbol, ['x', 'y', 'theta', 'a', 'b'])
  r_x, r_y = map(Symbol, ['r_x','r_y'])
  u_xx, u_xy, u_yy, u_x, u_y, u_0 = map(Symbol, ['u_xx', 'u_xy', 'u_yy', 'u_x', 'u_y', 'u_0'])
  P = Poly (expand(
      (((x-r_x)*cos(theta)+(y-r_y)*sin(theta))**2/a**2+
      (-(x-r_x)*sin(theta)+(y-r_y)*cos(theta))**2/b**2-1)*a**2*b**2),x,y)
  printeqlist([
```

```
    [u_xx, P.coeff(2,0)],
    [u_xy, P.coeff(1,1)],
    [u_yy, P.coeff(0,2)],
    [u_x, P.coeff(1,0)],
    [u_y, P.coeff(0,1)],
    [u_0, P.coeff(0,0)]
  ])

def parametric_bz():
  t = Symbol('t')
  P = Poly(sympify('(1-t)**3*p_x0+3*(1-t)**2*t*p_x1+3*(1-t)*t**2*p_x2+t**3*p_x3'), t)
  printpolycode('a', P, 3)
  P = Poly(sympify('(1-t)**3*p_y0+3*(1-t)**2*t*p_y1+3*(1-t)*t**2*p_y2+t**3*p_y3'), t)
  printpolycode('b', P, 3)

def bz_bz_check():
  x, y, t = symbols('xyt')
  X = sympify('a3*t**3 + a2*t**2 + a1*t + a0')
  Y = sympify('b3*t**3 + b2*t**2 + b1*t + b0')
  P = Poly((X-x)**2+(Y-y)**2,t)
  printpolycode('u', P, 6)

def implicit_bz():
  x, y, t = symbols('xyt')
  a3, a2, a1, a0 = map(Symbol, ['a3','a2','a1','a0'])
  b3, b2, b1, b0 = map(Symbol, ['b3','b2','b1','b0'])
  P1 = Poly(a3*t**3+a2*t**2+a1*t+a0-x,t)
  P2 = Poly(b3*t**3+b2*t**2+b1*t+b0-y,t)
  P = Poly(resultant(P1,P2), x, y)
  printeqlist([
    ['v_xxx', P.coeff(3,0)],
    ['v_xxy', P.coeff(2,1)],
    ['v_xyy', P.coeff(1,2)],
    ['v_yyy', P.coeff(0,3)],
    ['v_xx', P.coeff(2,0)],
    ['v_xy', P.coeff(1,1)],
    ['v_yy', P.coeff(0,2)],
    ['v_x', P.coeff(1,0)],
    ['v_y', P.coeff(0,1)],
    ['v_0', P.coeff(0,0)]
  ])

def bz_bz():
  x, y, t = symbols('xyt')
  P = sympify('u_xxx*x**3+u_xxy*x**2*y+u_xyy*x*y**2+u_yyy*y**3+u_xx*x**2+u_xy*x*y+u_yy*y**2+u_x*x+u_y*y+u_0')
  X = sympify('a3*t**3 + a2*t**2 + a1*t + a0')
  Y = sympify('b3*t**3 + b2*t**2 + b1*t + b0')
  P = Poly(P.subs(x,X).subs(y,Y), t)
  printpolycode('v', P, 9)

def el_el():
  x, y = symbols('xy')
  u_xx, u_xy, u_yy, u_x, u_y, u_0 = map(
      Symbol, ['u_xx', 'u_xy', 'u_yy', 'u_x', 'u_y', 'u_0'])
  v_xx, v_xy, v_yy, v_x, v_y, v_0 = map(
      Symbol, ['v_xx', 'v_xy', 'v_yy', 'v_x', 'v_y', 'v_0'])
  P1 = Poly(u_xx*x**2+u_xy*x*y+u_yy*y**2+u_x*x+u_y*y+u_0, x);
  P2 = Poly(v_xx*x**2+v_xy*x*y+v_yy*y**2+v_x*x+v_y*y+v_0, x);
  P = Poly(resultant(P1,P2), y)
  printpolycode('a', P, 4)

def bz_el():
  x, y, t = symbols('xyt')
  P = sympify('u_xx*x**2+u_xy*x*y+u_yy*y**2+u_x*x+u_y*y+u_0');
  X = sympify('a3*t**3 + a2*t**2 + a1*t + a0')
  Y = sympify('b3*t**3 + b2*t**2 + b1*t + b0')
  P = Poly(P.subs(x,X).subs(y,Y), t)
  printpolycode('v', P, 6)

def bz_self_inters():
  s, t = symbols('st')
  P1 = Poly(sympify('a3*(t**2+s*t+s**2)+a2*(s+t)+a1'), s)
  P2 = Poly(sympify('b3*(t**2+s*t+s**2)+b2*(s+t)+b1'), s)
  P = Poly(resultant(P1,P2), t)
  printpolycode('u', P, 2)

implicit_el()
parametric_bz()
bz_bz_check()
implicit_bz()
el_el()
bz_el();
bz_bz();
bz_self_inters()
```