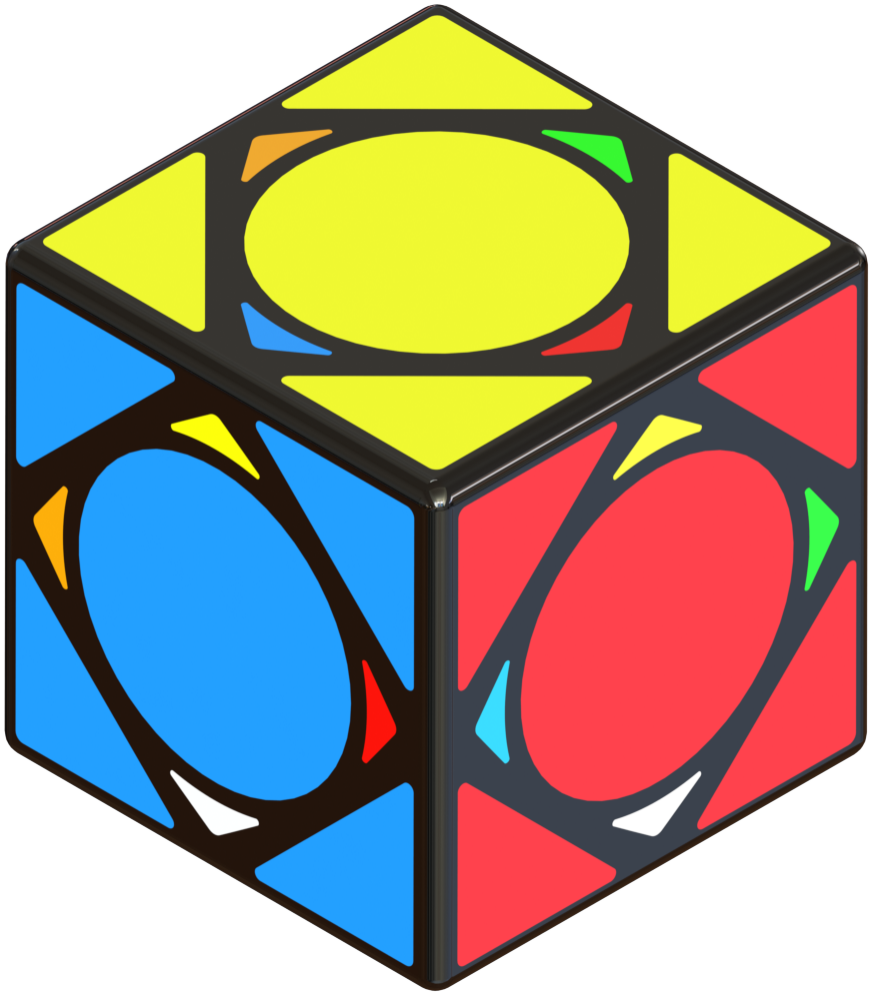
Flat Non-Polygonal 2D Stickers Embedded in 3D Space

Parametrizing segments of conic sections in 3D space

Written for usage in the k3DW Puzzle Simulator



Braden Ganetsky

Version 1.0

Contents

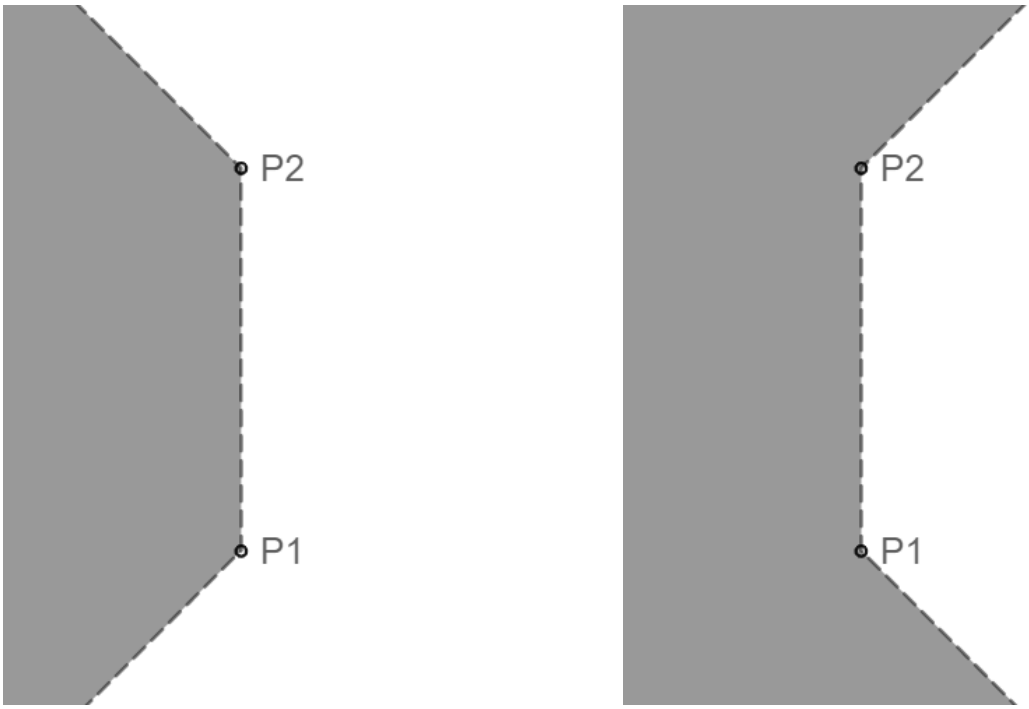
1 Introduction

This report is written to document the process of parametrizing segments of conic sections between 2 points in 3D space. This is written for use in the k3DW Puzzle Simulator, which makes use of non-polygonal stickers. Instead of storing various curves as polygons with very small edges, it is more advantageous to store the curves as parametric equations, and only convert to polygons for rendering purposes.

1.1 Visualizing the Curve

In the simulator, every sticker is defined by a list of vertices ordered in a counter-clockwise direction when viewed head on, which means that the normal vector points out of the paper. This is how the program determines in which direction to calculate the normal vector to a given sticker. Every sticker must be 2-dimensional, meaning that all vertices must be co-planar, within floating point error. However, stickers may be concave, meaning that a subset of the vertices may in fact wind in a clockwise direction, such as in the case of a crescent shape. Therefore, the idea of vertices being in a counter-clockwise direction refers to the global winding of the sticker, not for any proper subset of vertices in particular.

Given this information, the following visualization holds true. If a given sticker is viewed head on, with a pair of vertices lined up vertically in the field of view, with the earlier-defined vertex below the later-defined vertex, then the sticker sits to the left of the line connecting the two vertices. This holds true for both convex and concave subsets, illustrated below in figure 1.1. This is an important baseline when defining the curves later on.



(a) Convex subset of vertices (b) Concave subset of vertices

Figure 1.1: Demonstration that the sticker is always on the left of the line connecting two vertices

1.2 Parametrizing the Curve

When specifying stickers in a puzzle file, each edge has a starting point, $\mathbf{P}_1$, and a type, as well as the other data that is required to define that specific type of curve. At this time, all the types are conic sections, which likely allows for all the variation ever needed for this simulator. Regardless, it is fairly simple to add a new type if it is ever needed.

1.2.1 General Parametric Form

Each type of curve has different input data required to properly define it, including the normal $\mathbf{k}$, the starting point $\mathbf{P}_1$, and the ending point $\mathbf{P}_2$. These three are already stored in the `Sticker` object. Interestingly, a parametrization was found such that the stored data for each curve will have the same form for all curve types. All types have a parametric equation in the form of

$$\mathbf{C}(t) = \mathbf{P}_1 + \mathbf{v}f(t_0 \cdot t) + \mathbf{w}g(t_0 \cdot t) \tag{1.1}$$

where $\mathbf{P}_1$ is the starting point, $\mathbf{v}$ and $\mathbf{w}$ are vectors to be determined by the constructor of a given type of curve, and f and g are functions which are the same for all instances of a type of curve. The value t_0 is a constant such that the entire curve segment is traced out for $t \in [0, 1]$, to standardize the equations. This means that $\mathbf{C}(0) = \mathbf{P}_1$ and $\mathbf{C}(1) = \mathbf{P}_2$ for all curves.

1.2.2 New Reference Frame

Because every curve is defined in 3D space, not in the 2D plane, the curve must be translated and rotated in space. The simplest way to do this is to define a new reference frame and then represent the curve in this reference frame. The definition of a `Line` is simple enough that it does not require defining this new reference frame, however all the other curves use this. First, the origin is translated to the input point $\mathbf{P}_1$. A new vector $\mathbf{L}$ is used in place of $\mathbf{P}_2$, with the following definition.

$$\mathbf{L} = \mathbf{P}_2 - \mathbf{P}_1 \tag{1.2}$$

The `Circle`, `Parabola`, `Ellipse`, and `Hyperbola` require a third point, which is defined by using the input variable d. Start with the midpoint between $\mathbf{P}_1$ and $\mathbf{P}_2$, and travel perpendicular to this line for a distance of d. This new point is called $\mathbf{P}_0$ in the original reference frame, and $\mathbf{Q}$ in the new reference frame. The direction of travel can be mathematically defined as the cross product of $\mathbf{L}$ and $\mathbf{k}$, shown in 1.3 below. The point $\mathbf{Q}$ can be defined from $\mathbf{P}_0$, or directly from the base values. Both are shown below. If d is negative, then $\mathbf{Q}$ is offset from the $\mathbf{L}$ vector in the opposite direction.

$$\begin{aligned} \mathbf{P}_0 &= \frac{\mathbf{P}_1 + \mathbf{P}_2}{2} + (\mathbf{L} \times \mathbf{k}) \cdot d \\ \mathbf{Q} = \mathbf{P}_0 - \mathbf{P}_1 &= \frac{\mathbf{L}}{2} + (\mathbf{L} \times \mathbf{k}) \cdot d \end{aligned} \tag{1.3}$$

The new basis vectors for this reference frame are defined next, which are called $\mathbf{i}$, $\mathbf{j}$, and $\mathbf{k}$. The $\mathbf{k}$ vector is the unit normal vector given in the `Sticker` object. The $\mathbf{i}$ vector is constructed by taking the unit

vector of $\mathbf{L}$ and rotating it by a given angle about $\mathbf{k}$. For the `Parabola`, `Ellipse`, and `Hyperbola`, this angle is an input, called α. For the `Circle` this value is always 0 and no rotation is required. The $\mathbf{j}$ vector is then constructed as the perpendicular vector to both $\mathbf{i}$ and $\mathbf{k}$. This way, all three basis vectors have unit length and are mutually perpendicular. They are given in equation 1.4 below. When d is negative, the $\mathbf{i}$ vector rotates in the opposite direction from $\mathbf{L}$, and $\mathbf{j}$ is the negative cross product.

$$\begin{aligned} \mathbf{i} &= \mathrm{rot}_{\mathbf{k}}\left(\frac{\mathbf{L}}{|\mathbf{L}|}, \alpha\,\mathrm{sign}(d)\right) \\ \mathbf{j} &= \mathrm{sign}(d)\cdot(\mathbf{i}\times\mathbf{k}) \end{aligned} \tag{1.4}$$

If d is given as a negative value, then the curve should be parametrized in the opposite direction. The simplest way to deal with a negative distance is to incorporate the change into the basis vectors, and then proceed as usual. This is what the definitions above have done. From here on out, d is assumed to be strictly positive. Any negative values are already handled in these definitions, so then any negative d can be turned positive.

With these new basis vectors, the values of $\mathbf{P}_0$ and $\mathbf{Q}$ can take another form, shown below.

$$\begin{aligned} \mathbf{P}_0 &= \frac{\mathbf{P}_1+\mathbf{P}_2}{2} - \mathbf{i}\cdot d\sin(\alpha) + \mathbf{j}\cdot d\cos(\alpha) \\ \mathbf{Q} &= \frac{\mathbf{L}}{2} - \mathbf{i}\cdot d\sin(\alpha) + \mathbf{j}\cdot d\cos(\alpha) \end{aligned} \tag{1.5}$$

The projections of $\mathbf{L}$ and $\mathbf{Q}$ onto the $\mathbf{i}$ and $\mathbf{j}$ axes are defined as follows. They are used very often in the construction of the curves. The forms of the equations using the dot product are used in the code, and the forms using the trigonometric functions of α are used in Desmos for testing.

$$\begin{aligned} L_i &= \mathbf{L}\cdot\mathbf{i} = |\mathbf{L}|\cos(\alpha) \\ L_j &= \mathbf{L}\cdot\mathbf{j} = |\mathbf{L}|\sin(\alpha) \\ Q_i &= \mathbf{Q}\cdot\mathbf{i} = \frac{L_i}{2} - d\sin(\alpha) \\ Q_j &= \mathbf{Q}\cdot\mathbf{j} = \frac{L_j}{2} + d\cos(\alpha) \end{aligned} \tag{1.6}$$

1.3 Coding the Curve

The curves are implemented as functors in C++, with `operator()` overloaded to compute $\mathbf{C}(t)$. Each type of curve has its own derived class with a constructor to specifically set the data for that type, and taking in its own set of required inputs. Each derived class defines the **`virtual`** base functions to for its own requirements. The data stored in `Curve` is implemented in the seemingly strange way as a **`union`** for usage in the `split` function.

```
class CurveData {
public:
    Point p1;
    Vec v, w;
    double t0;
};

class Curve {
public:
    Curve() = default;
    Curve(const Point& p1, const Vec& v, const Vec& w, double t0);
    Curve(const CurveData& data);
    Curve(const Curve& curve);
    virtual Point operator()(double t);
    virtual std::pair<Curve, Curve> split(double s);

protected:
    union {
        struct {
            Point p1;
            Vec v, w;
            double t0;
        };
        CurveData data;
    };
};
```

1.4 Splitting the Curve

A curve may need to be split at a particular point $\mathbf{C}(t = s)$ when a sticker is split. This will result in two subsequent curves, `first` and `second`. Each of these two new curves must be given their member values. The following is common for all curves. This is found within the `std::pair<Curve, Curve> split(double)` member function of `Curve`. The values of $\mathbf{v}$, $\mathbf{w}$, and t_0 are specific to each curve type and are determined by computing $\mathbf{C}_1(t) = \mathbf{C}(st)$ and $\mathbf{C}_2(t) = \mathbf{C}(t(1-s)+s)$.

2 Line

2.1 Parametrizing the Line

The line is the base case for connecting 2 vertices in space. It is the simplest possible curve. The parametric form of a line requires a starting point and a vector, and given in equation 2.1 below.

$$\mathbf{C}(t) = \mathbf{P}_1 + \mathbf{v}t \tag{2.1}$$

In this case, as compared to equation 1.1, $t_0 = 1$, $\mathbf{w} = \mathbf{0}$, $f(t) = t$, and $g(t)$ is irrelevant.

The validity of this equation can be tested. It linear in t, so it certainly draws a line. Values of t can be plugged in to determine if it draws the correct line.

$$\begin{aligned} \mathbf{C}(0) &= \mathbf{P}_1 + \mathbf{v}(0) \\ &= \mathbf{P}_1 \end{aligned} \tag{2.2}$$

$$\begin{aligned} \mathbf{C}(1) &= \mathbf{P}_1 + \mathbf{v}(1) \\ &= \mathbf{P}_1 + (\mathbf{P}_2 - \mathbf{P}_1) \\ &= \mathbf{P}_2 \end{aligned} \tag{2.3}$$

Therefore this is the parametric equation for the given line segment. This type of curve requires very little thinking or derivation. The same cannot be said for the other types of curves.

2.2 Coding the Line

The code for the `Line` object is as follows. It is a simple implementation of the explanation above. Instead of comments in the code to explain how it works, this report acts as the documentation.

```
class Line : public Curve {
public:
    Line(const Point& p1, const Point& p2);
    Point operator()(double t);
    std::pair<Curve, Curve> split(double s);
};
```

```
Line::Line(const Point& p1, const Point& p2) {
    this->p1 = p1;
    this->v = p2 - p1;
    this->w = Vec::zero;
    this->t0 = 1;
}
Point Line::operator()(double t) {
    return p1 + v * t;
}
```

2.3 Splitting the Line

For all lines, $\mathbf{w}$ is the zero vector and $t_0 = 1$, therefore only new values for $\mathbf{v}$ need to be found for `first` and `second`, respectively.

For a line split at $t = s$, the first and second new values of $\mathbf{v}$ can be found by the method shown in section 1.4. For $\mathbf{C}_1(t)$, t must range from 0 to s instead of from 0 to 1. The new values can be found by modifying the argument in the previously derived equation for $\mathbf{C}(t)$, equation 2.1.

$$t \to st$$

$$\begin{aligned} \mathbf{C}_1(t) &= \mathbf{P}_1 + \mathbf{v}(st) \\ &= \mathbf{P}_1 + (\mathbf{v} \cdot s)t \\ &= \mathbf{P}_1 + \mathbf{v}_1 t \end{aligned} \tag{2.4}$$

Therefore,

$$\mathbf{v}_1 = \mathbf{v} \cdot s \tag{2.5}$$

For $\mathbf{C}_2(t)$, t must range from s to 1. The new values can be found by modifying the argument in the previously derived equation for $\mathbf{C}(t)$, equation 2.1.

$$t \to t(1-s) + s$$

$$\begin{aligned} \mathbf{C}_2(t) &= \mathbf{P}_1 + \mathbf{v}(t(1-s) + s) \\ &= (\mathbf{P}_1 + \mathbf{v} \cdot s) + (\mathbf{v}(1-s))\, t \\ &= \mathbf{C}(s) + \mathbf{v}_2 t \\ &= (\mathbf{P}_1)_2 + \mathbf{v}_2 t \end{aligned} \tag{2.6}$$

Therefore,

$$\mathbf{v}_2 = \mathbf{v} \cdot (1-s) \tag{2.7}$$

To summarize this splitting operation in code, continuing on from section 1.4, the following is implemented for a line.

```
std::pair<Curve, Curve> Line::split(double s) {
    CurveData first, second;
    first.p1 = p1;          second.p1 = operator()(s);
    first.v = v*s;          second.v = v*(1-s);
    first.w = Vec::zero;    second.w = Vec::zero;
    first.t0 = 1;           second.t0 = 1;
    return std::make_pair(Curve(first), Curve(second));
}
```

3 Circle

3.1 Visualizing the Circle

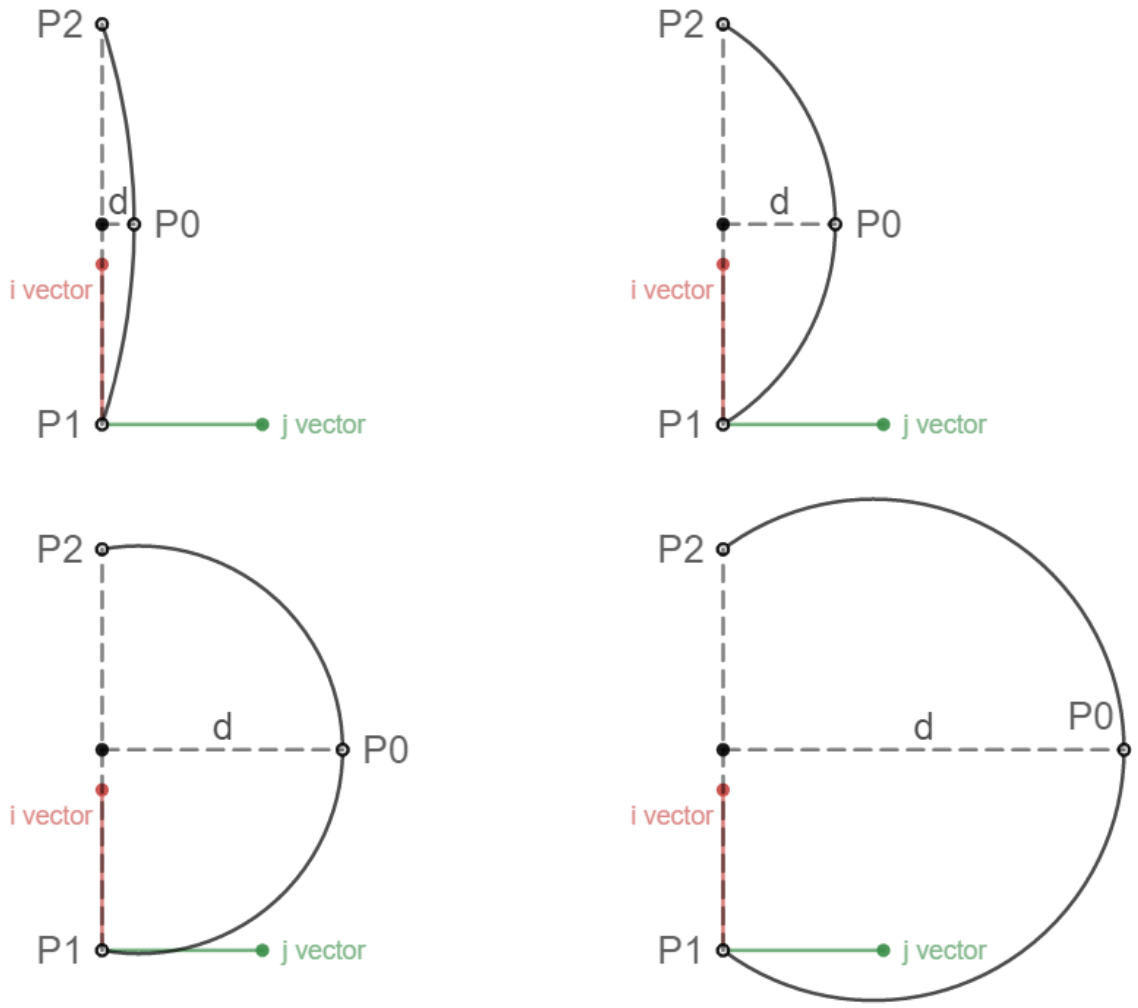


Figure 3.1: Demonstration of the various depths of a circle segment

As shown above in figure 3.1, a circle can be any level of depth. As well, not shown above, this can apply to a convex or concave subset of vertices on a sticker, but that differentiation is not relevant, as explained in section 1.2. The specific depth of the circle is irrelevant to the math involved, and the direction of the is normalized when creating the **i-j** axis system. Therefore, once the direction of the circle is taken into account, every circle can be considered identically.

Regardless of the circle, it is parametrized from $\mathbf{P}_1$ to $\mathbf{P}_2$ in a negative direction, if the **i-j** reference frame were viewed as though it was a right-handed coordinate system like the standard **x-y** plane. This is the basis for parametrizing any given circle.

3.2 Parametrizing the Circle

A circle is fully defined by 3 points. For this circle, these points are $\mathbf{P}_1$, $\mathbf{P}_2$, and $\mathbf{P}_0$, which are translated to become $\mathbf{0}$, $\mathbf{L}$, and $\mathbf{Q}$. In a 2D place, the analytical equation of a general circle is as follows.

$$(x-a)^2 + (y-b)^2 = r^2 \tag{3.1}$$

3.2.1 Determining the Analytical Constants

The projections of $\mathbf{L}$ and $\mathbf{Q}$ onto the $\mathbf{i}$ and $\mathbf{j}$ axes can be simplified for the case of a circle. This is because $\alpha = 0$, so it is omitted.

$$\begin{aligned} L_i &= \mathbf{L} \cdot \mathbf{i} = |\mathbf{L}| \\ L_j &= \mathbf{L} \cdot \mathbf{j} = 0 \\ Q_i &= \mathbf{Q} \cdot \mathbf{i} = \frac{1}{2}|\mathbf{L}| \\ Q_j &= \mathbf{Q} \cdot \mathbf{j} = d \end{aligned} \tag{3.2}$$

Plugging the origin into equation 3.1, the value of r can be found.

$$\begin{aligned} r^2 &= (0-a)^2 + (0-b)^2 \\ &= a^2 + b^2 \\ r &= \sqrt{a^2 + b^2} \end{aligned} \tag{3.3}$$

Plugging both the origin and $\mathbf{L}$ into equation 3.1, then subtracting one from the other, the value of a can be found.

$$\begin{cases} (0-a)^2 + (0-b)^2 & = r^2 \\ (L_i - a)^2 + (L_j - b)^2 & = r^2 \end{cases}$$

$$\begin{cases} a^2 + (0-b)^2 & = r^2 \\ L_i^2 - 2L_i a + a^2 + (0-b)^2 & = r^2 \end{cases}$$

$$\begin{aligned} L_i^2 - 2L_i a &= 0 \\ \Rightarrow a &= \frac{1}{2} L_i \end{aligned} \tag{3.4}$$

Plugging $\mathbf{Q}$ into equation 3.1, the value of b can be found.

$$\begin{aligned}(Q_i - a)^2 + (Q_j - b)^2 &= r^2 \\ (\frac{1}{2}L_i - \frac{1}{2}L_i)^2 + (d - b)^2 &= \sqrt{a^2 + b^2}^2 \\ d^2 - 2db + b^2 &= \left(\frac{1}{2}L_i\right)^2 + b^2 \\ \Rightarrow b &= \frac{d}{2} - \frac{L_i^2}{8d}\end{aligned} \tag{3.5}$$

3.2.2 Determining the Parametric Constants

Equation 3.1 gives the analytical equation of the circle, but it defines the entire circle. For this use case, a segment of this circle must be defined parametrically. The parametric equation for this circle is as follows.

$$\mathbf{C}(t) = \mathbf{P}_1 + (r\cos(t_0 \cdot t - c) + a)\mathbf{i} + (r\sin(t_0 \cdot t - c) + b)\mathbf{j} \tag{3.6}$$

The values of t_0 and c must be found such that the boundary conditions are met. To find expressions for $\cos(c)$ and $\sin(c)$, plug in $t = 0$. These equations are separated by their $\mathbf{i}$ and $\mathbf{j}$ components.

$$\mathbf{C}(0) - \mathbf{P}_1 = \mathbf{0} \tag{3.7}$$

$$\begin{aligned}&\begin{cases} r\cos(0 - c) + a = 0 \\ r\sin(0 - c) + b = 0 \end{cases} \\ &\Rightarrow \begin{cases} \cos(c) = -\frac{a}{r} \\ \sin(c) = \frac{b}{r} \end{cases}\end{aligned} \tag{3.8}$$

To find t_0, plug in $t = 1$. Substitutions for $\cos(c)$ and $\sin(c)$ are taken from equation 3.8. Equation 3.10 below uses the $\mathbf{j}$ component of equation 3.9 because $L_j = 0$, which makes it easier.

$$\mathbf{C}(1) - \mathbf{P}_1 = \mathbf{L} \tag{3.9}$$

$$\begin{aligned} r\sin(t_0 - c) + b = L_j &= 0 \\ \sin(t_0)\cos(c) - \cos(t_0)\sin(c) &= -\frac{b}{r} \\ -\sin(t_0)\frac{a}{r} - \cos(t_0)\frac{b}{r} &= -\frac{b}{r} \\ \frac{\sin(t_0)}{\cos(t_0) - 1} + \frac{b}{a} &= 0 \\ \cot\left(\frac{t_0}{2}\right) &= \frac{b}{a} \\ \rightarrow t_0 = 2\,\mathrm{arccot}\left(\frac{b}{a}\right) + 2\pi n & \end{aligned} \tag{3.10}$$

The circle is parametrized in a negative direction as compared to the **i-j** coordinate system viewed as a right-hand x-y plane, therefore the value of t_0 must lie between -2π and 0. For $n = -1$, the expression of t_0 above yields the correct range. However, there is no arccot function in C++, therefore the following rearranging is necessary in equation 3.11. This gives the final expression for t_0.

$$\begin{aligned} t_0 &= 2\,\text{arccot}\left(\frac{b}{a}\right) - 2\pi \\ &= 2\left(-\arctan\left(\frac{b}{a}\right) + \frac{\pi}{2}\right) - 2\pi \\ \Rightarrow t_0 &= -2\arctan\left(\frac{b}{a}\right) - \pi \end{aligned} \tag{3.11}$$

3.2.3 General Form of the Circle

Equation 3.6 gives the general parametric equation of the circle. This can be reduced down so that is has the form of equation 1.1.

$$\begin{aligned} \mathbf{C}(t) &= \mathbf{P}_1 + (r\cos(t_0 \cdot t - c) + a)\mathbf{i} + (r\sin(t_0 \cdot t - c) + b)\mathbf{j} \\ &= \mathbf{P}_1 + r\mathbf{i}\cos(t_0 \cdot t - c) + a\mathbf{i} + r\mathbf{j}\sin(t_0 \cdot t - c) + b\mathbf{j} \\ &= \mathbf{P}_1 + \mathbf{i}a + \mathbf{j}b + r\mathbf{i}(\cos(t_0 \cdot t)\cos(c) + \sin(t_0 \cdot t)\sin(c)) + r\mathbf{j}(\sin(t_0 \cdot t)\cos(c) - \cos(t_0 \cdot t)\sin(c)) \\ &= \mathbf{P}_1 + \mathbf{i}a + \mathbf{j}b + r\mathbf{i}\left(-\cos(t_0 \cdot t)\frac{a}{r} + \sin(t_0 \cdot t)\frac{b}{r}\right) + r\mathbf{j}\left(-\sin(t_0 \cdot t)\frac{a}{r} - \cos(t_0 \cdot t)\frac{b}{r}\right) \\ &= \mathbf{P}_1 + \mathbf{i}a + \mathbf{j}b - \mathbf{i}\cos(t_0 \cdot t)a - \mathbf{j}\cos(t_0 \cdot t)b + \mathbf{i}\sin(t_0 \cdot t)b - \mathbf{j}\sin(t_0 \cdot t)a \\ &= \mathbf{P}_1 + (\mathbf{i}a + \mathbf{j}b)(1 - \cos(t_0 \cdot t)) + (\mathbf{i}b - \mathbf{j}a)\sin(t_0 \cdot t) \end{aligned}$$

Two new vectors can be defined to simplify this equation.

$$\begin{aligned} \mathbf{v} &= \mathbf{i}a + \mathbf{j}b \\ \mathbf{w} &= \mathbf{i}b - \mathbf{j}a \end{aligned} \tag{3.12}$$

Note that the **v** vector is the center point of the circle. This means that the final parametric equation for the circle is as follows.

$$\mathbf{C}(t) = \mathbf{P}_1 + \mathbf{v}(1 - \cos(t_0 \cdot t)) + \mathbf{w}\sin(t_0 \cdot t) \tag{3.13}$$

This equation has the same form as equation 1.1. This means that, once the necessary values are calculated, this equation generates the desired circular section when t ranges from 0 to 1.

3.3 Coding the Circle

The constructor for the `Circle` derived object is as follows. It is a simple implementation of the math derived above. The `Circle` is actually implemented as a special case of an `Ellipse` object, as the **`operator()`** and `split` functions will be identical, therefore they are implemented for the `Ellipse`. There is no need to calculate many of the intermediary values, such as $\mathbf{P}_0$, r, or c. They can be bypassed by calculating the final values directly. Instead of comments in the code to explain how it works, this report acts as the documentation.

```
class Circle : public Ellipse {
public:
    Circle(const Point& p1, const Point& p2, const Vec& k, double dist);
};
```

```
Circle::Circle(const Point& p1, const Point& p2, const Vec& k, double dist) {
    assert(dist != 0);
    this->p1 = p1;

    Vec l = p2 - p1;
    double li = abs(l);

    Vec i = l / li;
    Vec j = (dist > 0 ? 1 : -1)* (i * k);
    if (dist < 0)
        dist = -dist;

    double a = li / 2;
    double b = dist / 2 - (li * li) / (8 * dist);

    this->v = i * a + j * b;
    this->w = i * b - j * a;
    this->t0 = -2 * std::atan(b / a) - PI;
}
```

This constructor could be omitted and instead the arguments could be passed directly into the `Ellipse` constructor, but this involved more computations on the numbers, allowing more room for floating point error.

3.4 Splitting the Circle

For a circle split at $t = s$, the first and second new circles can be found by the method shown in section 1.4. For $\mathbf{C}_1(t)$, t must range from 0 to s instead of from 0 to 1. The new values can be found by modifying the argument in the previously derived equation for $\mathbf{C}(t)$, equation 3.13.

$$t \to st$$

$$\begin{aligned} \mathbf{C}_1(t) &= \mathbf{P}_1 + \mathbf{v}(1 - \cos(t_0 \cdot (st)) + \mathbf{w}\sin(t_0 \cdot (st)) \\ &= \mathbf{P}_1 + \mathbf{v}(1 - \cos((st_0) \cdot t) + \mathbf{w}\sin((st_0) \cdot t) \end{aligned} \tag{3.14}$$

Only the value of t_0 is different, as this equation has exactly the same form as equation 3.13, therefore the final values are as follows.

$$\begin{aligned} \mathbf{v}_1 &= \mathbf{v} \\ \mathbf{w}_1 &= \mathbf{w} \\ t_{01} &= st_0 \end{aligned} \tag{3.15}$$

For $\mathbf{C}_2(t)$, t must range from s to 1. The new values can be found by modifying the argument in the previously derived equation for $\mathbf{C}(t)$, equation 3.13.

$$t \to t(1 - s) + s$$

$$\begin{aligned} \mathbf{C}_2(t) &= \mathbf{P}_1 + \mathbf{v}(1 - \cos(t_0 \cdot (t(1 - s) + s))) + \mathbf{w}\sin(t_0 \cdot (t(1 - s) + s)) \\ &= \mathbf{P}_1 + \mathbf{v} - \mathbf{v}\cos(t \cdot t_0(1 - s) + st_0) + \mathbf{w}\sin(t \cdot t_0(1 - s) + st_0)) \end{aligned}$$

Because the point at which the circle is split becomes the starting point for the second circle, then

$$\mathbf{C}(s) = \mathbf{P}_1 + \mathbf{v}(1 - \cos(st_0)) + \mathbf{w}\sin(st_0)$$

The final equation for the second circle can be simplified as follows, by expanding the cos and sin terms using additive identities, and then rearranging in the right way.

$$\begin{aligned} \mathbf{C}_2(t) &= \mathbf{C}(s) + (\mathbf{v}\cos(st_0) - \mathbf{w}\sin(st_0))(1 - \cos((1 - s)t_0 \cdot t)) + (\mathbf{v}\sin(st_0) + \mathbf{w}\cos(st_0))\sin((1 - s)t_0 \cdot t) \\ &= \mathbf{C}(s) + \mathbf{v}_2(1 - \cos(t_{02} \cdot t)) + \mathbf{w}_2\sin(t_{02} \cdot t) \end{aligned} \tag{3.16}$$

Equation 3.16 is in the same form as equation 3.13, therefore the final values are as follows.

$$\begin{aligned} \mathbf{v}_2 &= \mathbf{v}\cos(st_0) - \mathbf{w}\sin(st_0) \\ \mathbf{w}_2 &= \mathbf{v}\sin(st_0) + \mathbf{w}\cos(st_0) \\ t_{02} &= (1 - s)t_0 \end{aligned} \tag{3.17}$$

The `split` function for a circle is identical to the `Ellipse`, so it will be implemented only for the `Ellipse`.

4 Parabola

4.1 Visualizing the Parabola

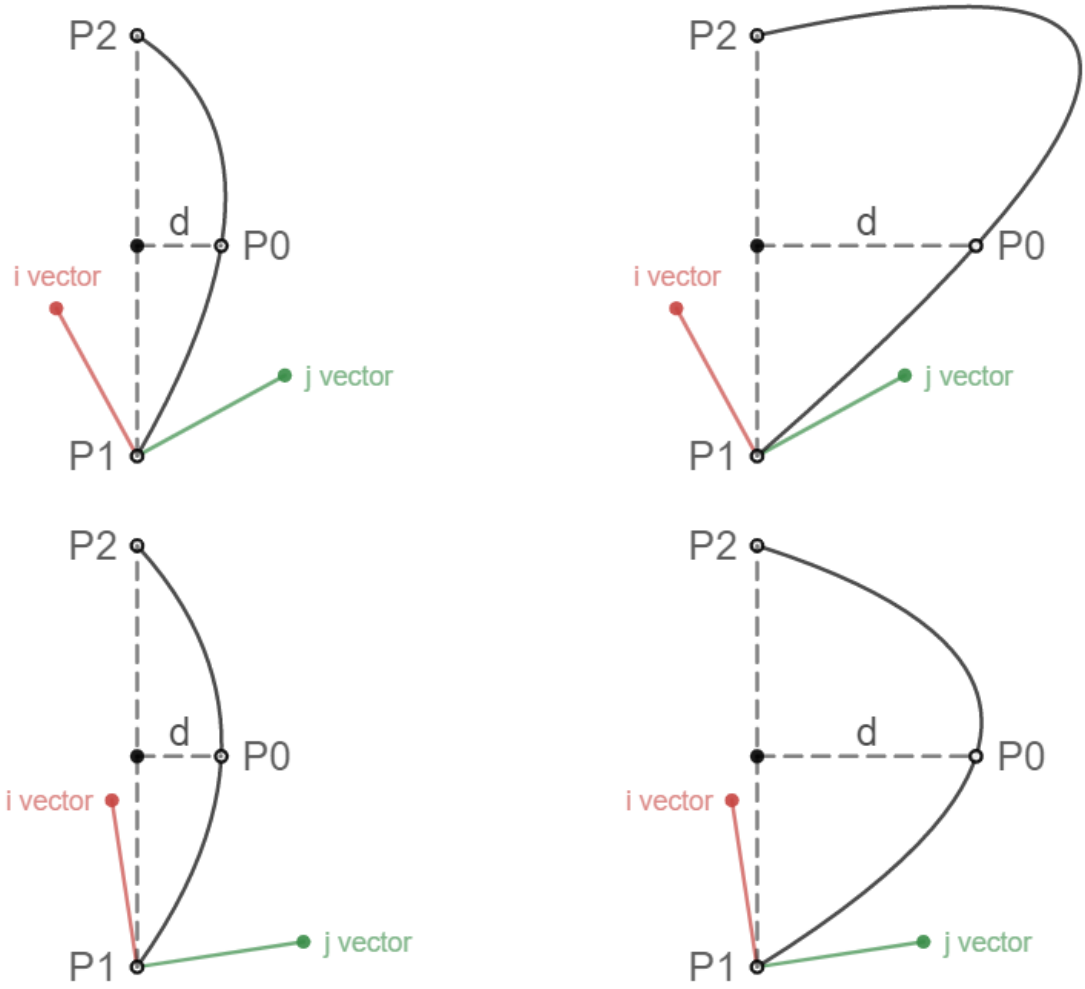


Figure 4.1: Demonstration of various depths and angles of a parabola segment

As shown above in figure 4.1, a parabola can be any level of depth up until the parabola intersects the **j** axis, and can be any angle within $\pm 90°$ of zero. As well, this can apply to a convex or concave subset of vertices on a sticker, but that differentiation is not relevant, as explained in section 1.2. The specific depth and angle of the parabola are irrelevant to the math involved, and the direction is normalized when creating the **i-j** axis system. Therefore, once the direction of the parabola is taken into account, every parabola can be considered identically.

Regardless of the parabola, it is parametrized from $\mathbf{P}_1$ to $\mathbf{P}_2$ as t varies from 0 to 1. This is the basis for parametrizing any given parabola.

4.2 Parametrizing the Parabola

A parabola, like a circle, is defined by 3 points, but only if a reference frame is specified. For this parabola, these points are $\mathbf{P}_1$, $\mathbf{P}_2$, and $\mathbf{P}_0$, which are translated to become $\mathbf{0}$, $\mathbf{L}$, and $\mathbf{Q}$. However, changing the reference frame changes the parabola, which means that there is an additional degree of freedom as compared to a circle. The reference frame is defined by the angle $\alpha \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$, which is the angular offset of the basis vector $\mathbf{i}$, as explained in section 1.2.2.

In a 2D plane, the analytical equation of a general parabola is as follows.

$$y = ax^2 + bx + c \tag{4.1}$$

For a given angle, the distance d must be sufficiently small such that the parabola remains downward-facing in the **i-j** plane, otherwise the parabolic section will not intersect all 3 required points. At the upper limit of d, the parabola becomes 2 parallel lines, as shown in figure 4.2 below.

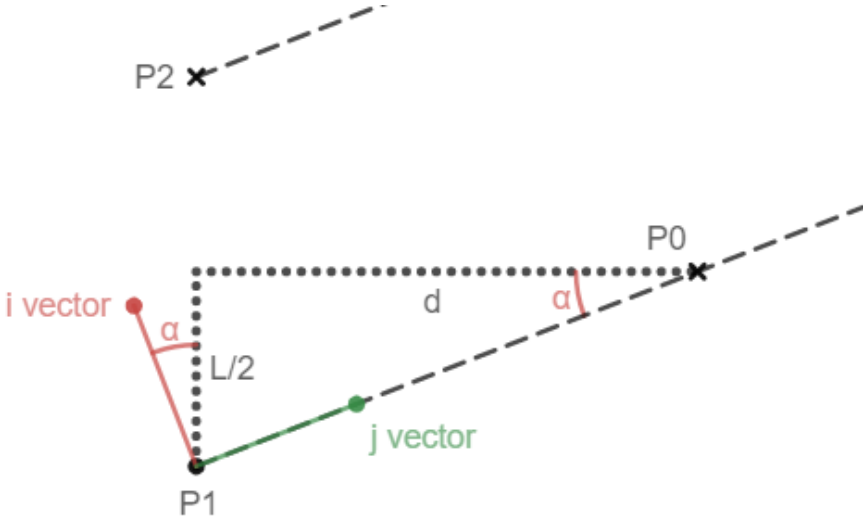


Figure 4.2: Limit of the distance input for a parabola

From figure 4.2 above, a bound on d can be derived as follows. This is a geometric limit on the inputs.

$$\begin{aligned} \tan|\alpha| &= \frac{\frac{|\mathbf{L}|}{2}}{d_{max}} \\ \Rightarrow d_{max} &= \frac{|\mathbf{L}|}{2\tan|\alpha|} \\ \Rightarrow d &\leq \left|\frac{\mathbf{L}}{2\tan(\alpha)}\right| \end{aligned} \tag{4.2}$$

4.2.1 Determining the Constants

Plugging the origin into equation 4.1, the value of c can be found.

$$\begin{aligned} 0 &= a(0)^2 + b(0) + c \\ \Rightarrow c &= 0 \end{aligned} \tag{4.3}$$

Plugging $\mathbf{L}$ into equation 4.1, an expression for b can be found in terms of a.

$$\begin{aligned} L_j &= aL_i^2 + bL_i \\ \Rightarrow b &= \frac{L_j}{L_i} - aL_i \end{aligned} \tag{4.4}$$

Plugging $\mathbf{Q}$ into equation 4.1, and then substituting the expression for b, the value of a can be found.

$$\begin{aligned} Q_j &= aQ_i^2 + bQ_i \\ Q_j &= aQ_i^2 + \left(\frac{L_j}{L_i} - aL_i\right) Q_i \\ \Rightarrow a &= \frac{\frac{Q_j}{Q_i} - \frac{L_j}{L_i}}{Q_i - L_i} \end{aligned} \tag{4.5}$$

Using the analytical equation for the parabola, equation 4.1, this can be changed into a parametric equation, with $\mathbf{P}_1$ at the origin. Let $x = t_0 \cdot t$.

$$\begin{aligned} \mathbf{C}(t) - \mathbf{P}_1 &= \mathbf{i}x + \mathbf{j}y \\ \mathbf{C}(t) - \mathbf{P}_1 &= \mathbf{i}x + \mathbf{j}(ax^2 + bx) \\ \mathbf{C}(t) - \mathbf{P}_1 &= \mathbf{i}(t_0 \cdot t) + \mathbf{j}(a(t_0 \cdot t)^2 + b(t_0 \cdot t)) \end{aligned} \tag{4.6}$$

The starting condition of $\mathbf{C}(0) = \mathbf{P}_1$ is already met, regardless of the choice of t_0. Now, a value of t_0 must be determined such that $\mathbf{C}(1) = \mathbf{P}_2$, that is, $\mathbf{C}(1) - \mathbf{P}_1 = \mathbf{L}$.

$$\mathbf{C}(1) - \mathbf{P}_1 = \mathbf{L} = \mathbf{i}t_0 + \mathbf{j}(at_0^2 + bt_0) \tag{4.7}$$

Writing out the $\mathbf{i}$ and $\mathbf{j}$ components of this equation separately, the value of t_0 is easily found to be $t_0 = L_i$.

$$\Rightarrow \begin{cases} L_i = t_0 \\ L_j = at_0^2 + bt_0 \end{cases} \tag{4.8}$$

4.2.2 General Form of the Parabola

Equation 4.6 gives the general parametric equation of the parabola. This can be reduced down so that is has the form of equation 1.1.

$$\begin{aligned}\mathbf{C}(t) &= \mathbf{P}_1 + \mathbf{i}(t_0 \cdot t) + \mathbf{j}(a(t_0 \cdot t)^2 + b(t_0 \cdot t)) \\ &= \mathbf{P}_1 + \mathbf{i}(L_i \cdot t) + \mathbf{j}(a(L_i \cdot t)^2 + b(L_i \cdot t)) \\ &= \mathbf{P}_1 + (\mathbf{i}L_i)t + (\mathbf{j}aL_i^2)t^2 + (\mathbf{j}bL_i)t \\ &= \mathbf{P}_1 + (\mathbf{i}L_i + \mathbf{j}bL_i)t + (\mathbf{j}aL_i^2)t^2\end{aligned} \tag{4.9}$$

Two new vectors can be defined to simplify this equation.

$$\begin{aligned}\mathbf{v} &= L_i(\mathbf{i} + \mathbf{j}b) \\ \mathbf{w} &= L_i^2(\mathbf{j}a)\end{aligned} \tag{4.10}$$

In this case, because the value of $t_0 = L_i$ can be incorporated into $\mathbf{v}$ and $\mathbf{w}$, then set $t_0 = 1$ for all parabolic sections in this program. This way, t_0 is not even used for the parabola. The final parametric equation for the parabola is as follows.

$$\mathbf{C}(t) = \mathbf{P}_1 + \mathbf{v}t + \mathbf{w}t^2 \tag{4.11}$$

This equation has the same form as equation 1.1. This means that, once the necessary values are calculated, this equation generates the desired parabolic section when t ranges from 0 to 1.

4.3 Coding the Parabola

The code for the `Parabola` object is as follows. It is a simple implementation of the math derived above. Instead of comments in the code to explain how it works, this report acts as the documentation.

```
class Parabola : public Curve {
public:
    Parabola(const Point& p1, const Point& p2, const Vec& k, double dist, double alpha);
    Point operator()(double t);
    std::pair<Curve, Curve> split(double s);
};
```

```
Parabola::Parabola(const Point& p1, const Point& p2, const Vec& k, double dist, double alpha) {
    assert(dist != 0 && alpha >= -90 && alpha <= 90);
    this->p1 = p1;

    Vec l = p2 - p1;
    Vec q = l/2 + (l*k)*dist;
    if (alpha != 0)
        assert(std::abs(dist) <= abs(l / (2*std::tan(alpha * PI/180))));

    Vec i = ArbiRot(k)(alpha * (dist > 0 ? 1 : -1))(unit(l));
    Vec j = (dist > 0 ? 1 : -1) * (i*k);
    if(dist < 0)
        dist = -dist;

    double li = l | i, lj = l | j,
           qi = q | i, qj = q | j;

    double a = (qj/qi - lj/li) / (qi - li);
    double b = lj/li - a*li;

    this->v = li * (i + j*b);
    this->w = li*li * (j*a);
    this->t0 = 1;
}
Point Parabola::operator()(double t) {
    return p1 + v*t + w*t*t;
}
```

The class `ArbiRot`, which stands for *arbitrary rotation*, facilitates rotation about an arbitrary axis in 3D space. The `ArbiRot` constructor takes in a vector, in this case `k`, and outputs an `ArbiRot` object. An `ArbiRot` object has the **`operator()`** overloaded such that it takes in an angle in degrees, and returns the rotation matrix that performs the rotation by the provided angle about the vector given in the constructor, using the right hand rule. In this case, this rotation matrix is given by `ArbiRot(k)(alpha * (dist > 0 ? 1 : -1))`.

A `Matrix` object has the **`operator()`** overloaded such that it takes in either a `Vec` (alias for `Point`) or another `Matrix` and returns the result of the matrix multiplication between the original `Matrix` and the input. In this case `unit(l)` is a `Vec` object, so the **`operator()`** returns another `Vec` object. This is how **i** is calculated.

4.4 Splitting the Parabola

For all parabolic sections in this program, $t_0 = 1$, therefore only new values for $\mathbf{v}$ and $\mathbf{w}$ need to be found for `first` and `second`, respectively.

For a parabola split at $t = s$, the first and second new parabolas can be found by the method shown in section 1.4. For $\mathbf{C}_1(t)$, t must range from 0 to s instead of from 0 to 1. The new values can be found by modifying the argument in the previously derived equation for $\mathbf{C}(t)$, equation 4.11.

$$t \to st$$

$$\begin{aligned} \mathbf{C}_1(t) &= \mathbf{P}_1 + \mathbf{v}(st) + \mathbf{w}(st)^2 \\ &= \mathbf{P}_1 + (\mathbf{v}s)t + (\mathbf{w}s^2)t^2 \end{aligned} \tag{4.12}$$

This equation has exactly the same form as equation 4.11, therefore the final values are as follows.

$$\begin{aligned} \mathbf{v}_1 &= \mathbf{v}s \\ \mathbf{w}_1 &= \mathbf{w}s^2 \end{aligned} \tag{4.13}$$

For $\mathbf{C}_2(t)$, t must range from s to 1. The new values can be found by modifying the argument in the previously derived equation for $\mathbf{C}(t)$, equation 4.11.

$$t \to t(1-s) + s$$

$$\begin{aligned} \mathbf{C}_2(t) &= \mathbf{P}_1 + \mathbf{v}(t(1-s)+s) + \mathbf{w}(t(1-s)+s)^2 \\ &= \mathbf{P}_1 + \mathbf{v}s + \mathbf{v}(1-s)t + \mathbf{w}(1-s)^2t^2 + 2\mathbf{w}(1-s)st + \mathbf{w}s^2 \\ &= (\mathbf{P}_1 + \mathbf{v}s + \mathbf{w}s^2) + (\mathbf{v}(1-s) + 2\mathbf{w}(1-s)s)t + (\mathbf{w}(1-s)^2)t^2 \\ &= \mathbf{C}(s) + \mathbf{v}_2 t + \mathbf{w}_2 t^2 \end{aligned} \tag{4.14}$$

This equation is in the same form as equation 4.11, therefore the final values are as follows.

$$\begin{aligned} \mathbf{v}_2 &= \mathbf{v}(1-s) + 2\mathbf{w}(1-s)s \\ \mathbf{w}_2 &= \mathbf{w}(1-s)^2 \end{aligned} \tag{4.15}$$

To summarize this splitting operation in code, continuing on from section 1.4, the following is implemented for a parabola.

```
std::pair<Curve, Curve> Parabola::split(double s) {
    CurveData first, second;
    first.p1 = p1;          second.p1 = operator()(s);
    first.v = v*s;          second.v = v*(1-s) + w*(2*s*(1-s));
    first.w = w*(s*s);      second.w = w*((1-s)*(1-s));
    first.t0 = 1;           second.t0 = 1;
    return std::make_pair(Curve(first), Curve(second));
}
```

5 Ellipse

5.1 Visualizing the Ellipse

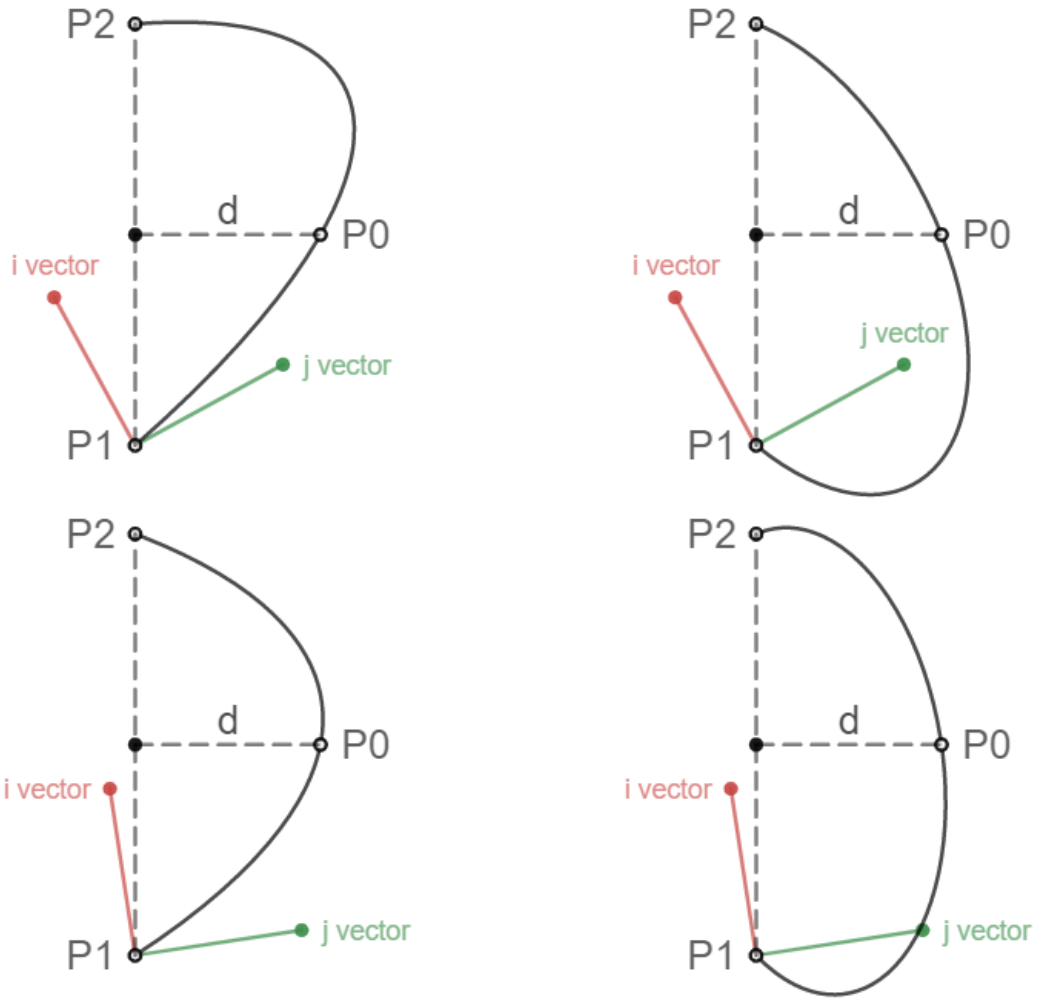


Figure 5.1: Demonstration of various angles and ratios of an ellipse segment

As shown above in figure 5.1, an ellipse and can be any level of depth, can be any angle within $\pm 90°$ of zero, and can be any ratio of the major and minor axes. As well, this can apply to a convex or concave subset of vertices on a sticker, but that differentiation is not relevant, as explained in section 1.2. The specific depth, angle, and ratio of the ellipse are irrelevant to the math involved, and the direction is normalized when creating the **i-j** axis system. Therefore, once the direction of the ellipse is taken into account, every ellipse can be considered identically.

Regardless of the ellipse, it is parametrized from $\mathbf{P}_1$ to $\mathbf{P}_2$ in a negative direction, if the **i-j** reference frame were viewed as though it was a right-handed coordinate system like the standard **x-y** plane. This is the basis for parametrizing any given ellipse.

5.2 Parametrizing the Ellipse

An ellipse is defined by 4 points if a reference frame is specified, and if the major and minor axes of the ellipse are parallel to the coordinate vectors. For this ellipse, the first three points are $\mathbf{P}_1$, $\mathbf{P}_2$, and $\mathbf{P}_0$, which are translated to become $\mathbf{0}$, $\mathbf{L}$, and $\mathbf{Q}$. Like a parabola, the reference frame is defined by the angle $\alpha \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$, which is the angular offset of the basis vector $\mathbf{i}$, as explained in section 1.2.2.

For the final degree of freedom, instead of specifying an additional point for the ellipse to pass through, the ratio of one axis to the other axis is specified. This effectively supplies the shape of the ellipse, as the variable r.

The analytical equation of a general ellipse is as follows, in a 2D plane.

$$\left(\frac{x-h}{a}\right)^2 + \left(\frac{y-k}{b}\right)^2 = 1 \tag{5.1}$$

5.2.1 Determining the Analytical Constants

The constants a and b are the lengths of the axes of the ellipse. Therefore, from the definition of r, b can be found in terms of a as shown in the following equation.

$$\begin{aligned} r &= \frac{b}{a} \\ \Rightarrow b &= ar \end{aligned} \tag{5.2}$$

Plugging the origin into equation 5.1, the value of a can be found in terms of h and k as shown in the following equation.

$$\begin{aligned} \left(\frac{0-h}{a}\right)^2 + \left(\frac{0-k}{b}\right)^2 &= 1 \\ \left(\frac{h}{a}\right)^2 + \left(\frac{k}{ar}\right)^2 &= 1 \\ \Rightarrow a &= \sqrt{h^2 + \left(\frac{k}{r}\right)^2} \end{aligned} \tag{5.3}$$

Next, the values of h and k must be determined. First, plugging $\mathbf{L}$ into equation 5.1, h can be found in terms of k.

$$\begin{aligned} \left(\frac{L_i-h}{a}\right)^2 + \left(\frac{L_j-k}{b}\right)^2 &= 1 \\ (L_i-h)^2 + \left(\frac{L_j-k}{r}\right)^2 &= a^2 \end{aligned}$$

$$L_i^2 - 2L_i h + h^2 + \left(\frac{L_j}{r}\right)^2 - \frac{2L_j k}{r^2} + \left(\frac{k}{r}\right)^2 = h^2 + \left(\frac{k}{r}\right)^2$$

$$L_i^2 - 2L_i h + \left(\frac{L_j}{r}\right)^2 - \frac{2L_j k}{r^2} = 0$$

$$\Rightarrow h = \frac{L_j(L_j - 2k)}{2L_i r^2} + \frac{L_i}{2} \tag{5.4}$$

Finally, plugging $\mathbf{Q}$ into equation 5.1 can yield an expression for k only in terms of inputs.

$$\left(\frac{Q_i - h}{a}\right)^2 + \left(\frac{Q_j - k}{b}\right)^2 = 1$$

$$(Q_i - h)^2 + \left(\frac{Q_j - k}{r}\right)^2 = a^2$$

$$Q_i^2 - 2Q_i h + h^2 + \left(\frac{Q_j}{r}\right)^2 - \frac{2Q_j k}{r^2} + \left(\frac{k}{r}\right)^2 = h^2 + \left(\frac{k}{r}\right)^2$$

$$Q_i^2 - 2Q_i h + \left(\frac{Q_j}{r}\right)^2 - \frac{2Q_j k}{r^2} = 0$$

$$Q_i^2 - 2Q_i \left(\frac{L_j^2 - 2L_j k}{2L_i r^2} + \frac{L_i}{2}\right) + \left(\frac{Q_j}{r}\right)^2 - \frac{2Q_j k}{r^2} = 0$$

$$Q_i^2 L_i r^2 - Q_i L_j^2 + 2Q_i L_j k - Q_i L_i^2 r^2 + Q_j^2 L_i - 2Q_j L_i k = 0$$

$$\Rightarrow k = \frac{Q_i L_i r^2 (Q_i - L_i) - Q_i L_j^2 + Q_j^2 L_i}{2(Q_j L_i - Q_i L_j)} \tag{5.5}$$

5.2.2 Determining the Parametric Constants

Equation 5.1 gives the analytical equation of the ellipse, but it defines the entire ellipse. For this use case, a segment of this ellipse must be defined parametrically. The parametric equation for this ellipse is as follows.

$$\mathbf{C}(t) = \mathbf{P}_1 + (a\cos(t_0 \cdot t - c) + h)\mathbf{i} + (b\sin(t_0 \cdot t - c) + k)\mathbf{j} \tag{5.6}$$

The values of t_0 and c must be found such that the boundary conditions are met. To find expressions for $\cos(c)$ and $\sin(c)$, plug in $t = 0$. These equations are separated by their $\mathbf{i}$ and $\mathbf{j}$ components.

$$\mathbf{C}(0) - \mathbf{P}_1 = \mathbf{0} \tag{5.7}$$

$$\begin{cases} a\cos(0 - c) + h = 0 \\ b\sin(0 - c) + k = 0 \end{cases}$$
$$\Rightarrow \begin{cases} \cos(c) = -\frac{h}{a} \\ \sin(c) = \frac{k}{b} = \frac{k}{ar} \end{cases} \tag{5.8}$$

To find t_0, plug in $t = 1$. Substitutions for $\cos(c)$ and $\sin(c)$ are taken from equation 5.8. This derivation below uses the **i** component of equation 5.9, but using the **j** component would yield an equivalent expression for t_0, which is shown later on. This derivation is a reverse engineering of a simpler equation plugged into Wolfram Alpha.

$$\mathbf{C}(1) - \mathbf{P}_1 = \mathbf{L} \tag{5.9}$$

$$\begin{aligned}
a\cos(t_0 - c) + h &= L_i \\
\cos(t_0)\cos(c) + \sin(t_0)\sin(c) &= \frac{L_i - h}{a} \\
-\cos(t_0)\frac{h}{a} + \sin(t_0)\frac{k}{ar} &= \frac{L_i - h}{a} \\
-hr\cos(t_0) + k\sin(t_0) &= L_i r - hr
\end{aligned}$$

$$\begin{aligned}
-2hr\cos(t_0) + 2hr + L_i r\cos(t_0) - L_i r + 2k\sin(t_0) - L_i r\cos(t_0) - L_i r &= 0 \\
(2hr - L_i r)(1 - \cos(t_0)) + 2k\sin(t_0) - L_i r(\cos(t_0) + 1) &= 0
\end{aligned}$$

$$\begin{aligned}
(2hr - L_i r)\frac{1 - \cos(t_0)}{\cos(t_0) + 1} + 2k\frac{\sin(t_0)}{\cos(t_0) + 1} - L_i r &= 0 \\
(2hr - L_i r)\tan^2\left(\frac{t_0}{2}\right) + 2k\tan\left(\frac{t_0}{2}\right) - L_i r &= 0
\end{aligned}$$

$$\begin{aligned}
\rightarrow \tan\left(\frac{t_0}{2}\right) &= \frac{-2k \pm \sqrt{4k^2 - 4(2hr - L_i r)(-L_i r)}}{2(2hr - L_i r)} \\
&= \frac{-k \pm \sqrt{k^2 + 2L_i\left(\frac{L_j(L_j - 2k)}{2L_i r^2} + \frac{L_i}{2}\right)r^2 - L_i^2 r^2}}{2hr - L_i r} \\
&= \frac{-k \pm \sqrt{k^2 - 2kL_j + L_j^2 + L_i^2 r^2 - L_i^2 r^2}}{2hr - L_i r} \\
&= \frac{-k \pm \sqrt{(k - L_j)^2}}{2hr - L_i r} \\
&= \frac{-k \pm (k - L_j)}{2hr - L_i r} \\
\rightarrow t_0 &= 2\arctan\left(\frac{-k \pm (k - L_j)}{2hr - L_i r}\right) + 2\pi n
\end{aligned}$$

The ellipse is parametrized in a negative direction as compared to the **i-j** coordinate system viewed as a right-hand x-y plane, therefore the value of t_0 must lie between -2π and 0. For $n = 0$, the expression of t_0 above yields values between $-\pi$ and π, and for $n = -1$, this expression yields values between -3π and $-\pi$. This means that the value of n changes depending on the other variables. Instead, this expression will be rearranged to have a constant value of n.

$$
\begin{aligned}
t_0 &= 2\arctan\left(\frac{-k \pm (k - L_j)}{2hr - L_i r}\right) + 2\pi n \\
&= 2\left(-\arctan\left(\frac{2hr - L_i r}{-k \pm (k - L_j)}\right) - \frac{\pi}{2}\right) + 2\pi n \\
&= 2\arctan\left(\frac{L_i r - 2hr}{-k \pm (k - L_j)}\right) + \pi(2n - 1)
\end{aligned}
$$

This new expression has the correct range for $n = -1$. Therefore the expression for t_0 can be expressed as the following.

$$
t_0 = 2\arctan\left(\frac{L_i r - 2hr}{-k \pm (k - L_j)}\right) - \pi \tag{5.10}
$$

A very similar derivation can be done using the $\mathbf{j}$ component of $\mathbf{C}(1)$ instead. This is shown below for the sake of completeness.

$$
\begin{aligned}
b\sin(t_0 - c) + k &= L_j \\
\sin(t_0)\cos(c) - \cos(t_0)\sin(c) &= \frac{L_j - k}{b} \\
-\sin(t_0)\frac{h}{a} - \cos(t_0)\frac{k}{ar} &= \frac{L_j - k}{ar} \\
k - L_j - k\cos(t_0) - hr\sin(t_0) &= 0
\end{aligned}
$$

$$
\begin{aligned}
2k - L_j - 2k\cos(t_0) + L_j\cos(t_0) - 2hr\sin(t_0) - L_j\cos(t_0) - L_j &= 0 \\
(2k - L_j)(1 - \cos(t_0)) - 2hr\sin(t_0) - (\cos(t_0) + 1)L_j &= 0 \\
(2k - L_j)\frac{1 - \cos(t_0)}{\cos(t_0) + 1} - 2hr\frac{\sin(t_0)}{\cos(t_0) + 1} - L_j &= 0 \\
(2k - L_j)\tan^2\left(\frac{t_0}{2}\right) - 2hr\tan\left(\frac{t_0}{2}\right) - L_j &= 0 \\
\rightarrow \tan\left(\frac{t_0}{2}\right) &= \frac{2hr \pm \sqrt{4h^2r^2 - 4(2k - L_j)(-L_j)}}{2(2k - L_j)} \\
&= \frac{hr \pm \sqrt{h^2r^2 + 2kL_j - L_j^2}}{2k - L_j}
\end{aligned}
$$

Rearranging equation 5.4, the following expression is found.

$$
2hL_i r^2 - L_i^2 r^2 = L_j(L_j - 2k)
$$

Then the previous expression can be rewritten as the following.

$$\begin{aligned}\tan\left(\frac{t_0}{2}\right) &= \frac{hr \pm \sqrt{h^2r^2 - 2hL_ir^2 + L_i^2r^2}}{2k - L_j} \\ &= \frac{hr \pm r\sqrt{(h-L_i)^2}}{2k - L_j} \\ &= \frac{hr \pm r(h-L_i)}{2k - L_j}\end{aligned}$$

$$\begin{aligned}\rightarrow t_0 &= 2\arctan\left(\frac{hr \pm r(h-L_i)}{2k - L_j}\right) + 2\pi n \\ &= 2\left(-\arctan\left(\frac{2k - L_j}{hr \pm r(h-L_i)}\right) - \frac{\pi}{2}\right) + 2\pi n \\ &= 2\arctan\left(\frac{L_j - 2k}{hr \pm r(h-L_i)}\right) + \pi(2n-1) \\ \Rightarrow t_0 &= 2\arctan\left(\frac{L_j - 2k}{hr \pm r(h-L_i)}\right) - \pi\end{aligned} \tag{5.11}$$

There are now two separate expressions for t_0 which are equivalent, from equations 5.10 and 5.11. These are shown below.

$$\begin{aligned}t_0 &= 2\arctan\left(\frac{L_ir - 2hr}{-k \pm (k - L_j)}\right) - \pi \\ t_0 &= 2\arctan\left(\frac{L_j - 2k}{hr \pm r(h-L_i)}\right) - \pi\end{aligned} \tag{5.12}$$

The correct signs in the denominator of the arctan argument were determined empirically through Desmos, in order to make the arguments given to each of the arctan functions equal. This could also be done analytically if required. The final equivalent expressions for t_0 are as follows.

$$\begin{aligned}\Rightarrow t_0 &= 2\arctan\left(\frac{2hr - L_ir}{L_j}\right) - \pi \\ \Rightarrow t_0 &= 2\arctan\left(\frac{L_j - 2k}{L_ir}\right) - \pi\end{aligned} \tag{5.13}$$

Either of these two expressions in equation 5.13 will suffice in order to give the correct result for t_0. In order to use fewer arithmetic operations, the second one was chosen.

5.2.3 General Form of the Ellipse

Equation 5.6 gives the general parametric equation of the ellipse. This can be reduced down so that is has the form of equation 1.1.

$$\begin{aligned}
\mathbf{C}(t) &= \mathbf{P}_1 + (a\cos(t_0 \cdot t - c) + h)\mathbf{i} + (b\sin(t_0 \cdot t - c) + k)\mathbf{j} \\
&= \mathbf{P}_1 + a\mathbf{i}\cos(t_0 \cdot t - c) + h\mathbf{i} + b\mathbf{j}\sin(t_0 \cdot t - c) + k\mathbf{j} \\
&= \mathbf{P}_1 + \mathbf{i}h + \mathbf{j}k + a\mathbf{i}(\cos(t_0 \cdot t)\cos(c) + \sin(t_0 \cdot t)\sin(c)) + b\mathbf{j}(\sin(t_0 \cdot t)\cos(c) - \cos(t_0 \cdot t)\sin(c)) \\
&= \mathbf{P}_1 + \mathbf{i}h + \mathbf{j}k + a\mathbf{i}\left(-\cos(t_0 \cdot t)\frac{h}{a} + \sin(t_0 \cdot t)\frac{k}{ar}\right) + b\mathbf{j}\left(-\sin(t_0 \cdot t)\frac{h}{a} - \cos(t_0 \cdot t)\frac{k}{ar}\right) \\
&= \mathbf{P}_1 + \mathbf{i}h + \mathbf{j}k - \mathbf{i}\cos(t_0 \cdot t)h - \mathbf{j}\cos(t_0 \cdot t)k + \mathbf{i}\sin(t_0 \cdot t)\frac{k}{r} - \mathbf{j}\sin(t_0 \cdot t)hr \\
&= \mathbf{P}_1 + (\mathbf{i}h + \mathbf{j}k)(1 - \cos(t_0 \cdot t)) + \left(\mathbf{i}\frac{k}{r} - \mathbf{j}hr\right)\sin(t_0 \cdot t)
\end{aligned}$$

Two new vectors can be defined to simplify this equation.

$$\begin{aligned}
\mathbf{v} &= \mathbf{i}h + \mathbf{j}k \\
\mathbf{w} &= \mathbf{i}\frac{k}{r} - \mathbf{j}hr
\end{aligned} \tag{5.14}$$

Note that the $\mathbf{v}$ vector is the center point of the ellipse. This means that the final parametric equation for the ellipse is as follows.

$$\mathbf{C}(t) = \mathbf{P}_1 + \mathbf{v}(1 - \cos(t_0 \cdot t)) + \mathbf{w}\sin(t_0 \cdot t) \tag{5.15}$$

This equation has the same form as equation 1.1, and is identical to the final equation for the circle, equation 3.13. This means that, once the necessary values are calculated, this equation generates the desired circular section when t ranges from 0 to 1.

5.3 Implementing the Ellipse

The code for the `Ellipse` object is as follows. It is a simple implementation of the math derived above. This is the superclass for `Circle`, instead of `Curve` itself. There is no need to calculate many of the intermediary values, such as a, b, or c. They can be bypassed by calculating the final values directly. Instead of comments in the code to explain how it works, this report acts as the documentation.

```
class Ellipse : public Curve {
public:
    Ellipse() = default;
    Ellipse(const Point& p1, const Point& p2, const Vec& normal, double dist, double alpha,
        double ratio);
    Point operator()(double t);
    std::pair<Curve, Curve> split(double s);
};
```

```
Ellipse::Ellipse(const Point& p1, const Point& p2, const Vec& normal, double dist, double alpha,
    double ratio) {
    assert(dist != 0 && alpha >= -90 && alpha <= 90 && ratio != 0);
    this->p1 = p1;

    Vec l = p2 - p1;
    Vec q = l/2 + (l*normal)*dist;

    Vec i = ArbiRot(normal)(alpha * (dist > 0 ? 1 : -1))(unit(l));
    Vec j = (dist > 0 ? 1 : -1) * (i * normal);
    if (dist < 0)
        dist = -dist;
    if (ratio < 0)
        ratio = -ratio;

    double li = l | i, lj = l | j,
           qi = q | i, qj = q | j;

    double k = (qi*li*ratio*ratio*(qi - li) - qi*lj*lj + qj*qj*li) / (2 * (qj*li - qi*lj));
    double h = (lj*(lj - 2*k)) / (2*li*ratio*ratio) + li/2;

    this->v = i*h + j*k;
    this->w = i*k/ratio - j*h*ratio;
    this->t0 = 2 * std::atan((lj-2*k)/(li*ratio)) - PI;
}
Point Ellipse::operator()(double t) {
    t *= t0;
    return p1 + v*(1-std::cos(t)) + w*std::sin(t);
}
```

The creation and usage of an `ArbiRot` object, which stands for *arbitrary rotation*, is explained in section 4.3. Essentially, it facilitates rotation about an arbitrary axis in 3D space.

5.4 Splitting the Ellipse

Because the general form of a circular section is identical to the general form of an elliptical section, the derivation for the split is identical to that of the circle, shown in section 3.4. The full derivation can be seen in that section. A brief summary is shown below.

$$\begin{aligned} \mathbf{C}_1(t) &= \mathbf{C}(st) \\ \Rightarrow \mathbf{P}_1 + \mathbf{v}(1 - \cos((st_0) \cdot t) + \mathbf{w}\sin((st_0) \cdot t) \end{aligned} \tag{5.16}$$

$$\begin{aligned} \mathbf{C}_2(t) &= \mathbf{C}(t(1-s)+s) \\ \Rightarrow \mathbf{C}(s) + (\mathbf{v}\cos(st_0) - \mathbf{w}\sin(st_0))(1 - \cos((1-s)t_0 \cdot t)) + (\mathbf{v}\sin(st_0) + \mathbf{w}\cos(st_0))\sin((1-s)t_0 \cdot t) \end{aligned} \tag{5.17}$$

Although the math was derived for the `Circle` case, the implementation is done in the `Ellipse` object. To summarize this splitting operation in code, continuing on from section 1.4, the following is implemented for an ellipse and, by extension, a circle.

```
std::pair<Curve, Curve> Ellipse::split(double s) {
    CurveData first, second;
    first.p1 = p1;          second.p1 = operator()(s);
    first.v = v;            second.v = v*std::cos(s*t0) - w*std::sin(s*t0);
    first.w = w;            second.w = v*std::sin(s*t0) + w*std::cos(s*t0);
    first.t0 = s*t0;        second.t0 = (1-s)*t0;
    return std::make_pair(Curve(first), Curve(second));
}
```

6 Hyperbola

6.1 Visualizing the Hyperbola

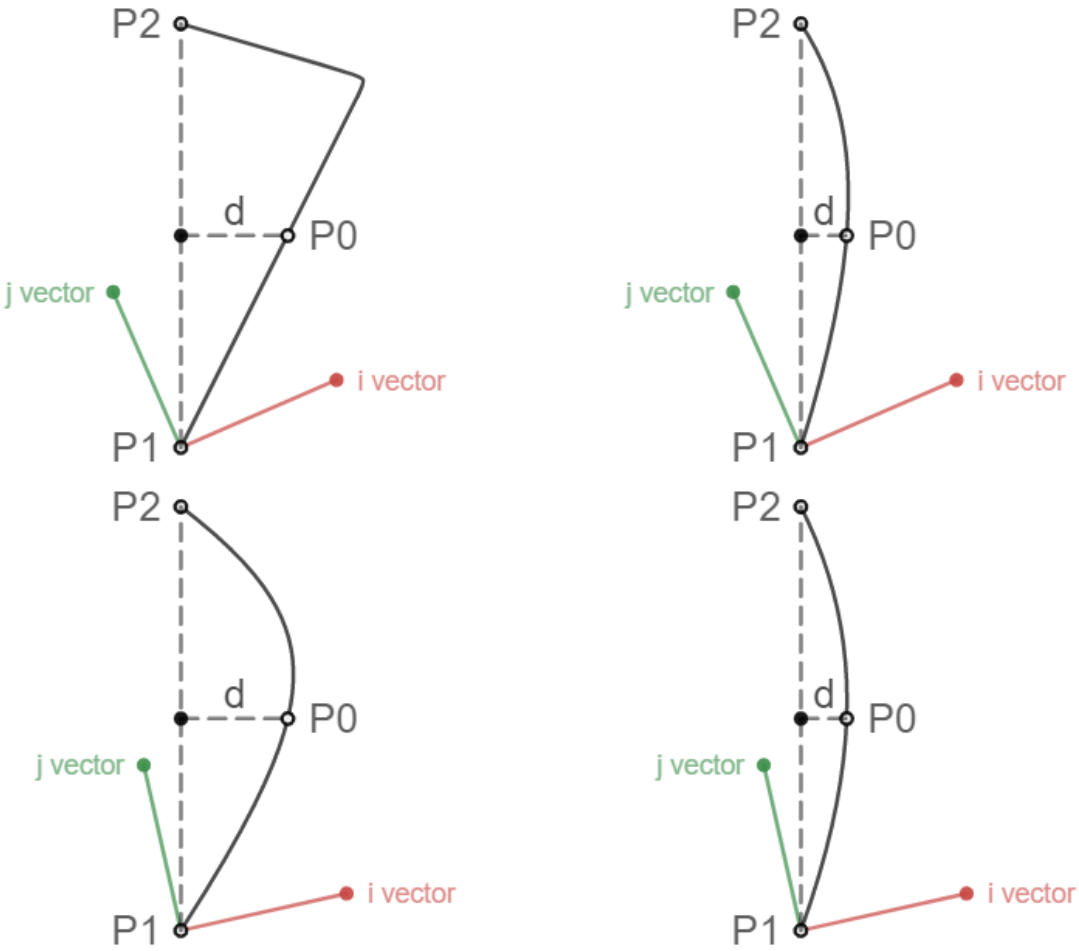


Figure 6.1: Demonstration of various angles and ratios of an ellipse segment

As shown above in figure 6.1, a hyperbola and can be any level of depth, can be any angle within $\pm 90°$ of zero, and can be any ratio of the major and minor axes, as long as it is a valid finite segment as explained in section 6.2.3. As well, this can apply to a convex or concave subset of vertices on a sticker, but that differentiation is not relevant, as explained in section 1.2. The specific depth, angle, and ratio of the hyperbola are irrelevant to the math involved, and the direction is normalized when creating the **i-j** axis system. As it turns out, all valid hyperbolas have their **i** and **j** vectors swapped, which is a process described below. Therefore, once the direction of the ellipse is taken into account, every ellipse can be considered identically.

Regardless of the hyperbola, it is parametrized from $\mathbf{P}_1$ to $\mathbf{P}_2$ as t varies from 0 to 1. This is the basis for parametrizing any given hyperbola.

6.2 Parametrizing the Hyperbola

Like an ellipse, a hyperbola is defined by 4 points if a reference frame is specified, and if the major and minor axes of the hyperbola are parallel to the coordinate vectors. For this hyperbola, the first three points are $\mathbf{P}_1$, $\mathbf{P}_2$, and $\mathbf{P}_0$, which are translated to become $\mathbf{0}$, $\mathbf{L}$, and $\mathbf{Q}$. Like a parabola, the reference frame is defined by the angle $\alpha \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$, which is the angular offset of the basis vector $\mathbf{i}$, as explained in section 1.2.2.

For the final degree of freedom, instead of specifying an additional point for the hyperbola to pass through, the ratio of one axis to the other axis is specified. This effectively supplies the shape of the hyperbola, as the variable r.

For a hyperbola, however, this is not the final conceptual framework. There are two general analytical equations of a hyperbola in a 2D plane. One of them opens in the $\mathbf{x}$ direction, while the other one opens in the $\mathbf{y}$ direction. Both of these must be considered, and they are as follows, respectively.

$$\begin{aligned} \left(\frac{x-h}{a}\right)^2 - \left(\frac{y-k}{b}\right)^2 &= 1 \\ \left(\frac{y-k}{b}\right)^2 - \left(\frac{x-h}{a}\right)^2 &= 1 \end{aligned} \tag{6.1}$$

Even though these two equations are nearly identical, one of them will be the correct equation to use for a specific case while the other will yield the complement hyperbola. For ease of calculation, only the first of these equations will be used. That is, the hyperbola will be assumed to open up in the $\mathbf{x}$ direction. If it is discovered that the second equation should be used instead, that is the hyperbola should instead open up in the $\mathbf{y}$ direction instead, then some pairs of variables must be swapped to construct the complement hyperbola. If this happens, then the following swaps will be made: $\mathbf{i} \leftrightarrow \mathbf{j}; a \leftrightarrow b; h \leftrightarrow k; L_i \leftrightarrow L_j; Q_i \leftrightarrow Q_j; r \leftrightarrow \frac{1}{r}$. This way, in the equations, the hyperbola will open up along the $\mathbf{i}$ axis in all cases. However, the relevant values and the axes themselves are swapped if the complement hyperbola is required, so that it appears to open up along the $\mathbf{j}$ axis. The condition for this swapping to occur will be outlined further below.

6.2.1 Determining the Analytical Constants

The constants a and b are the lengths of the axes of the hyperbola. Therefore, from the definition of r, b can be found in terms of a as shown in the following equation.

$$\begin{aligned} r &= \frac{b}{a} \\ \Rightarrow b &= ar \end{aligned} \tag{6.2}$$

Plugging the origin into the first line of equation 6.1, the value of a can be defined in terms of h and k

as shown in the following equation.

$$\left(\frac{0-h}{a}\right)^2 - \left(\frac{0-k}{b}\right)^2 = 1$$
$$\left(\frac{h}{a}\right)^2 - \left(\frac{k}{ar}\right)^2 = 1$$
$$\Rightarrow a = \sqrt{h^2 - \left(\frac{k}{r}\right)^2} \tag{6.3}$$

However, there are cases where the argument under the square root is negative. Therefore, this is the condition for which the variable swapping should occur. If the expression under the square root undergoes the variable swapping process, it becomes its negative. Therefore, the above expression will be used in all substitutions of a. All possible issues will be fixed by swapping the variables.

Next, the values of h and k must be determined. First, plugging $\mathbf{L}$ into equation 6.1, h can be found in terms of k.

$$\left(\frac{L_i-h}{a}\right)^2 - \left(\frac{L_j-k}{b}\right)^2 = 1$$
$$(L_i-h)^2 - \left(\frac{L_j-k}{r}\right)^2 = a^2$$
$$L_i^2 - 2L_ih + h^2 - \left(\frac{L_j}{r}\right)^2 + \frac{2L_jk}{r^2} - \left(\frac{k}{r}\right)^2 = h^2 - \left(\frac{k}{r}\right)^2$$
$$L_i^2 - 2L_ih - \left(\frac{L_j}{r}\right)^2 + \frac{2L_jk}{r^2} = 0$$
$$\Rightarrow h = \frac{L_j(2k-L_j)}{2L_ir^2} + \frac{L_i}{2} \tag{6.4}$$

Finally, plugging $\mathbf{Q}$ into equation 6.1 can yield an expression of k defined only in terms of inputs.

$$\left(\frac{Q_i-h}{a}\right)^2 - \left(\frac{Q_j-k}{b}\right)^2 = 1$$
$$(Q_i-h)^2 - \left(\frac{Q_j-k}{r}\right)^2 = a^2$$
$$Q_i^2 - 2Q_ih + h^2 - \left(\frac{Q_j}{r}\right)^2 + \frac{2Q_jk}{r^2} - \left(\frac{k}{r}\right)^2 = h^2 - \left(\frac{k}{r}\right)^2$$
$$Q_i^2 - 2Q_ih - \left(\frac{Q_j}{r}\right)^2 + \frac{2Q_jk}{r^2} = 0$$
$$Q_i^2 - 2Q_i\left(\frac{2L_jk-L_j^2}{2L_ir^2} + \frac{L_i}{2}\right) - \left(\frac{Q_j}{r}\right)^2 + \frac{2Q_jk}{r^2} = 0$$
$$Q_i^2L_ir^2 - 2Q_iL_jk + Q_iL_j^2 - Q_iL_i^2r^2 - Q_j^2L_i + 2Q_jL_ik = 0$$

$$\Rightarrow k = \frac{Q_i L_i r^2 (Q_i - L_i) + Q_i L_j^2 - Q_j^2 L_i}{2(Q_i L_j - Q_j L_i)} \tag{6.5}$$

At this stage, the necessary variables will be swapped if needed, since all the variables that require swapping have been defined. In the subsequent sections, the variables are taken to be the values after the swapping process has been performed

6.2.2 General Form of the Hyperbola

Equation 6.1 gives the analytical equation of the hyperbola, but it defines the entire hyperbola. For this use case, a segment of this hyperbola must be defined parametrically. The parametric equation for this hyperbola is as follows.

$$\mathbf{C}(t) = \mathbf{P}_1 + (a\sec(t_0 \cdot t - c) + h)\mathbf{i} + (b\tan(t_0 \cdot t - c) + k)\mathbf{j} \tag{6.6}$$

At this time, however, there has been no method discovered to translate this form of the equation into the final goal, which would look very similar to the general form of the ellipse found in section 5.2.3. The goal with equation 6.6 is to eventually rearrange it to look like the following equation.

$$\mathbf{C}(t) = \mathbf{P}_1 + \mathbf{v}(1 - \sec(t_0 \cdot t)) + \mathbf{w}\tan(t_0 \cdot t) \tag{6.7}$$

From experimentation in Desmos, it was discovered that a pair of valid $\mathbf{v}$ and $\mathbf{w}$ vectors would be almost identical to those from equation 5.14. The only difference is a sign in the $\mathbf{w}$ term, which is consistent with the differences between the analytical ellipse and hyperbola equations. These are not rigorously derived, but it is correct. These vectors are as follows.

$$\begin{aligned} \mathbf{v} &= \mathbf{i}h + \mathbf{j}k \\ \mathbf{w} &= \mathbf{i}\frac{k}{r} + \mathbf{j}hr \end{aligned} \tag{6.8}$$

Even though equations 6.6 and 6.7 trace identical hyperbolas for t between 0 and 2π, they are traversed at different rates and have different starting locations. This means that a correct value for t_0 in equation 6.6 will be incorrect for equation 6.7. Therefore, for the hyperbola, the value of t_0 will be found from the general equation (6.7) instead of the base parametric equation (6.6).

Finding t_0 will be done in a similar manner to finding t_0 for the ellipse. In equation 6.7, $\mathbf{C}(0) = \mathbf{P}_1$ for any value of t_0. Therefore the $t = 1$ end condition will be used, broken up into its $\mathbf{i}$ and $\mathbf{j}$ components.

$$\mathbf{C}(1) - \mathbf{P}_1 = \mathbf{v}(1 - \sec(t_0)) + \mathbf{w}\tan(t_0) = \mathbf{L}$$

$$\begin{cases} h(1 - \sec(t_0)) + \frac{k}{r}\tan(t_0) = L_i \\ k(1 - \sec(t_0)) + hr\tan(t_0) = L_j \end{cases} \tag{6.9}$$

Using the $\mathbf{i}$ equation, the following derivation for t_0 is valid. A similar derivation can be done for the L_j

equation. This is very similar to the derivation done for an ellipse starting in equation 5.9.

$$h(1-\sec(t_0)) + \frac{k}{r}\tan(t_0) = L_i$$
$$hr\cos(t_0) - hr + k\sin(t_0) = L_i r\cos(t_0)$$

$$2hr\cos(t_0) - 2hr - L_i r\cos(t_0) + L_i r - L_i r - L_i r\cos(t_0) + 2k\sin(t_0) = 0$$
$$(L_i r - 2hr)(1-\cos(t_0)) + 2k\sin(t_0) - L_i r(\cos(t_0)+1) = 0$$

$$(L_i r - 2hr)\frac{1-\cos(t_0)}{\cos(t_0)+1} + 2k\frac{\sin(t_0)}{\cos(t_0)+1} - L_i r = 0$$
$$(L_i r - 2hr)\tan^2\left(\frac{t_0}{2}\right) + 2k\tan\left(\frac{t_0}{2}\right) - L_i r = 0$$

$$\rightarrow \tan\left(\frac{t_0}{2}\right) = \frac{-2k \pm \sqrt{(2k)^2 - 4(L_i r - 2hr)(-L_i r)}}{2(L_i r - 2hr)}$$
$$= \frac{-k \pm \sqrt{k^2 + L_i^2 r^2 - 2L_i\left(\frac{L_j(2k-L_j)}{2L_i r^2} + \frac{L_i}{2}\right)r^2}}{L_i r - 2hr}$$
$$= \frac{-k \pm \sqrt{k^2 + L_i^2 r^2 - 2kL_j + L_j^2 - L_i^2 r^2}}{L_i r - 2hr}$$
$$= \frac{-k \pm \sqrt{(k-L_j)^2}}{L_i r - 2hr}$$
$$= \frac{-k \pm (k - L_j)}{L_i r - 2hr}$$

$$\rightarrow t_0 = 2\arctan\left(\frac{-k \pm (k-L_j)}{L_i r - 2hr}\right) + 2\pi n \tag{6.10}$$

The hyperbola is parametrized in a negative direction as compared to the **i-j** coordinate system viewed as a right-hand x-y plane, therefore the value of t_0 must lie between -2π and 0. For $n = 0$, the expression of t_0 above yields values between $-\pi$ and π, and for $n = -1$, this expression yields values between -3π and $-\pi$. This means that the value of n changes depending on the other variables. Instead, this expression will be rearranged to have a constant value of n.

$$t_0 = 2\arctan\left(\frac{-k \pm (k-L_j)}{L_i r - 2hr}\right) + 2\pi n$$
$$= 2\left(-\arctan\left(\frac{L_i r - 2hr}{-k \pm (k-L_j)}\right) - \frac{\pi}{2}\right) + 2\pi n$$
$$= 2\arctan\left(\frac{2hr - L_i r}{-k \pm (k-L_j)}\right) + \pi(2n-1)$$

This new expression has the correct range for $n = -1$. Therefore the expression for t_0 can be expressed as the following first expression. Similarly to the derivation ending in equation 5.11, the following second expression is exactly equivalent to the first, which could be found by doing the derivation in equation 6.2.2

for the **j** component instead. Equation 6.11 is similar to equation 5.12.

$$\begin{aligned} t_0 &= 2\arctan\left(\frac{2hr - L_i r}{-k \pm (k - L_j)}\right) - \pi \\ t_0 &= 2\arctan\left(\frac{2k - L_j}{hr \pm r(h - L_i)}\right) - \pi \end{aligned} \tag{6.11}$$

The correct signs in the denominator of the arctan argument were determined empirically through Desmos, in order to make the arguments given to each of the arctan functions equal. This could also be done analytically if required. Sometimes the curve must be parametrized in the positive direction to ensure that it intersects **Q**, which means that t_0 will need to be positive. To do this, add 2π to the value sometimes, meaning that the sign on the π will change. The final equivalent expressions for t_0 are as follows, with a clarification later on which sign to use on the π term.

$$\begin{aligned} \Rightarrow t_0 &= 2\arctan\left(\frac{L_i r - 2hr}{L_j}\right) \pm \pi \\ \Rightarrow t_0 &= 2\arctan\left(\frac{2k - L_j}{L_i r}\right) \pm \pi \end{aligned} \tag{6.12}$$

Either of these two expressions in equation 5.13 will suffice in order to give the correct result for t_0. In order to use fewer arithmetic operations, the second one was chosen.

Interestingly, the variables do not need to be swapped when using the general parametric equation (6.7) instead of the base parametric equation. For the value of **v** and **w** this is obvious, and is shown below.

$$\begin{aligned} \mathbf{v} = \mathbf{i}h + \mathbf{j}k &\leftrightarrow \mathbf{v} = \mathbf{j}k + \mathbf{i}h \\ \mathbf{w} = \mathbf{i}\frac{k}{r} + \mathbf{j}hr &\leftrightarrow \mathbf{w} = \mathbf{j}\frac{h}{\frac{1}{r}} + \mathbf{i}k\frac{1}{r} = \mathbf{j}hr + \mathbf{i}\frac{k}{r} \end{aligned}$$

For t_0 this is less obvious, but it is true that the absolute value of the expression inside the arctan function remains constant after a swap. That is, the following is true. It can be shown by plugging in the expression for h in terms of k and simplifying. The direction of parametrization also switches, meaning that the sign of the entire expression would change. This means that t_0 remains constant after a swap.

$$\frac{L_i r - 2hr}{L_j} = \frac{L_j - 2k}{L_i r}$$

6.2.3 Restrictions on the Hyperbola to Finite Segments

A complete hyperbola has an infinite arc length, meaning that certain combinations of inputs yield an asymptotic curve segment. Two conditions on the hyperbola will eliminate the possibility of any hyperbola segments being infinite in size.

First, a curve is only valid when all three points are on the same half of the hyperbola, which is comprised of two disjoint curves in its totality. Symbolically, this can be expressed as the following tripartite condition.

$$\operatorname{sign}(h) = \operatorname{sign}(h - L_i) = \operatorname{sign}(h - Q_i) \tag{6.13}$$

Secondly, in the **j** direction, **Q** must sit between the origin and **L**, otherwise the hyperbola will be parametrized in the wrong direction and will have an infinite length. This condition can be expressed as the following.

$$|Q_j| < |L_j| \tag{6.14}$$

All hyperbola segments begin at the origin and are initially tangent to the **w** vector. This means that for any finite hyperbola segment, the sign on the π term in the expression for t_0 must reflect the sign discrepancy of Q with respect to the **w** vector. That is, the sign on the π term will be the sign of the projection of **Q** onto **w**. Therefore, the final expression for t_0 is as follows.

$$\Rightarrow t_0 = 2\arctan\left(\frac{2k - L_j}{L_i r}\right) \pm \pi \operatorname{sign}(\mathbf{Q} \cdot \mathbf{w}) \tag{6.15}$$

6.3 Coding the Hyperbola

The code for the `Hyperbola` object is as follows. It is a simple implementation of the math derived above. There is no need to calculate many of the intermediary values, such as a or b. They can be bypassed by calculating the final values directly. Instead of comments in the code to explain how it works, this report acts as the documentation.

```
class Hyperbola : public Curve {
public:
    Hyperbola() = default;
    Hyperbola(const Point& p1, const Point& p2, const Vec& normal, double dist, double alpha,
        double ratio);
    Point operator()(double t);
    std::pair<Curve, Curve> split(double s);
};
```

```
Hyperbola::Hyperbola(const Point& p1, const Point& p2, const Vec& normal, double dist, double
    alpha, double ratio) {
    assert(dist != 0 && alpha >= -90 && alpha <= 90 && ratio != 0);
    this->p1 = p1;

    Vec l = p2 - p1;
    Vec q = l/2 + (l*normal)*dist;

    Vec i = ArbiRot(normal)(alpha * (dist > 0 ? 1 : -1))(unit(l));
    Vec j = (dist > 0 ? 1 : -1) * (i * normal);
    if (dist < 0)
        dist = -dist;
    if (ratio < 0)
        ratio = -ratio;

    double li = l | i, lj = l | j,
           qi = q | i, qj = q | j;

    double k = (qi*li*ratio*ratio*(qi-li) + qi*lj*lj - qj*qj*li) / (2 * (qi*lj - qj*li));
    double h = (lj*(2*k - lj)) / (2*li*ratio*ratio) + li/2;
```

```
    if (h*h - (k*k)/(ratio*ratio) < 0) {
        std::swap(i,j);
        std::swap(h,k);
        std::swap(li,lj);
        std::swap(qi,qj);
        ratio = 1/ratio;
    }

    assert((h>0) == (h-li>0) && (h>0) == (h-qi>0));
    assert(std::abs(qj) < std::abs(lj));

    this->v = i*h + j*k;
    this->w = i*k/ratio + j*h*ratio;
    this->t0 = 2 * std::atan((2*k-lj)/(li*ratio)) + PI * ((q|w) > 0 ? 1 : -1);
}
Point Hyperbola::operator()(double t) {
    t *= t0;
    return p1 + v*(1-std::sec(t)) + w*std::tan(t);
}
```

The creation and usage of an `ArbiRot` object, which stands for *arbitrary rotation*, is explained in section 4.3. Essentially, it facilitates rotation about an arbitrary axis in 3D space.

6.4 Splitting the Hyperbola

6.4.1 First Segment

For a hyperbola split at $t = s$, the first new hyperbola segments can be found by the method shown in section 1.4, but the second segment must be found some other way. For $\mathbf{C}_1(t)$, t must range from 0 to s instead of from 0 to 1. The new values can be found by modifying the argument in the previously derived equation for $\mathbf{C}(t)$, equation 6.7.

$$t \to st$$

$$\begin{aligned} \mathbf{C}_1(t) &= \mathbf{P}_1 + \mathbf{v}(1 - \sec(t_0 \cdot (st)) + \mathbf{w}\tan(t_0 \cdot (st)) \\ &= \mathbf{P}_1 + \mathbf{v}(1 - \sec((st_0) \cdot t) + \mathbf{w}\tan((st_0) \cdot t) \end{aligned} \tag{6.16}$$

Only the value of t_0 is different, as this equation has exactly the same form as equation 6.7, therefore the final values are as follows.

$$\begin{aligned} \mathbf{v}_1 &= \mathbf{v} \\ \mathbf{w}_1 &= \mathbf{w} \\ t_{01} &= st_0 \end{aligned} \tag{6.17}$$

6.4.2 Second Segment

For $\mathbf{C}_2(t)$, t must range from s to 1. However, unlike the other types of curves, the new values cannot be found by modifying the argument in the previously derived equation for $\mathbf{C}(t)$. Instead, some creativity is needed.

In a 2D plane, any hyperbola centered at the origin, $\mathbf{C}_o(t)$, can be represented by the following expression, where $\mathbf{v}_o$ is a point on the hyperbola and $\mathbf{w}_o$ is a tangent vector at that point.

$$\mathbf{C}_o(t) = \mathbf{v}_o \sec(t) + \mathbf{w}_o \tan(t) \tag{6.18}$$

In the case of this paper, the following substitutions are made so that $\mathbf{C}(t)$ is translated to be centered at the origin.

$$\begin{aligned} \mathbf{C}_o(t) &= \mathbf{C}(t) - \mathbf{P}_1 - \mathbf{v} \\ &= -\mathbf{v}\sec(t_0 \cdot t) + \mathbf{w}\tan(t_0 \cdot t) \end{aligned} \tag{6.19}$$

The value of $\mathbf{v}_o$ can be any point on the curve, and $\mathbf{w}_o$ must be a corresponding tangent vector, which means that any given hyperbola has an infinite number of valid parametrizations in this form. The current chosen point on the curve is $-\mathbf{C}_o(0) = \mathbf{v}$. For the second hyperbola segment, however, the starting point must be moved to $t = s$. Because this hyperbola is centered at the origin, any point $\mathbf{P}$ on the curve has a corresponding point $-\mathbf{P}$ that is also on the curve. Therefore $\mathbf{v}_2$ can be simply defined as the following expression.

$$\Rightarrow \mathbf{v}_2 = -\mathbf{C}_o(s) = \mathbf{v}\sec(st_0) - \mathbf{w}\tan(st_0) \tag{6.20}$$

All corresponding tangent vectors for $\mathbf{v}_2$ are of the form shown below, where n is a real number.

$$\begin{aligned}\mathbf{w}_2 &= n \cdot (-\mathbf{C}_o'(s)) \\ &= n \cdot \left.\frac{\mathrm{d}\,(-\mathbf{C}_o(t))}{\mathrm{d}t}\right|_{t=s} \\ &= n \cdot t_0 \sec(st_0) \cdot (\mathbf{v}\tan(st_0) - \mathbf{w}\sec(st_0))\end{aligned} \tag{6.21}$$

From experimentation, the expression for n was found, which gives a final expression for $\mathbf{w}_2$. There is no mathematical rigour regarding this expression for n. It was simply found through trial and error. This value cancels terms in the above expression for $\mathbf{w}_2$.

$$\rightarrow n = \frac{-1}{t_0 \sec(st_0)}$$

$$\Rightarrow \mathbf{w}_2 = -\mathbf{v}\tan(st_0) + \mathbf{w}\sec(st_0) \tag{6.22}$$

Interestingly, the expressions for $\mathbf{v}_2$ and $\mathbf{w}_2$ are identical except that the positions of $\mathbf{v}$ and $\mathbf{w}$ are swapped. The value of $\mathbf{v}_2$ is taken to be a point on the hyperbola, so the value of $\mathbf{w}_2$ can then be viewed as the corresponding point on the complement hyperbola, where both hyperbolas share their center point, major and minor axes, and asymptotes. Also interestingly, the expressions for $\mathbf{v}_2$ for $\mathbf{w}_2$ are very similar to the expressions found for the ellipse.

This hyperbola must now be translated back from being centered at the origin. First, the following observation is made. If equation 6.19 is valid, then the following equation must also be true. This equation has the same form as equation 6.7, which has the same form as equation 1.1.

$$\begin{aligned}\mathbf{C}_{2o}(t) &= \mathbf{C}_2(t) - \mathbf{C}(s) - \mathbf{v}_2 \\ &= -\mathbf{v}_2 \sec(t_{02} \cdot t) + \mathbf{w}_2 \tan(t_{02} \cdot t)\end{aligned}$$

$$\Rightarrow \mathbf{C}_2(t) = \mathbf{C}(s) + \mathbf{v}_2(1 - \sec(t_{02} \cdot t)) + \mathbf{w}_2 \tan(t_{02} \cdot t) \tag{6.23}$$

Lastly, the expression for t_{02} must be found. This can be derived by plugging in $t = 1$ to equation 6.23 above.

$$\begin{aligned}\mathbf{C}_2(1) = \mathbf{C}(1) &= \mathbf{C}(s) + \mathbf{v}_2(1 - \sec(t_{02})) + \mathbf{w}_2 \tan(t_{02}) \\ \mathbf{C}(1) - \mathbf{C}(s) &= \mathbf{v}_2(1 - \sec(t_{02})) + \mathbf{w}_2 \tan(t_{02})\end{aligned}$$

$$\begin{aligned}LHS \rightarrow \mathbf{C}(1) - \mathbf{C}(s) &= -\mathbf{v}\sec(t_0) + \mathbf{w}\tan(t_0) + \mathbf{v}\sec(st_0) - \mathbf{w}\tan(st_0) \\ &= -\mathbf{v}\sec(t_0) + \mathbf{w}\tan(t_0) + \mathbf{v}_2\end{aligned}$$

$$\begin{aligned}RHS \rightarrow \mathbf{v}_2(1 - \sec(t_{02})) &+ \mathbf{w}_2 \tan(t_{02}) \\ = \mathbf{v}_2 + (\mathbf{v}\sec(st_0) - \mathbf{w}\tan(st_0))(-\sec(t_{02})) &+ (-\mathbf{v}\tan(st_0) + \mathbf{w}\sec(st_0))\tan(t_{02})\end{aligned}$$

$$\rightarrow -\mathbf{v}\sec(t_0) + \mathbf{w}\tan(t_0) = (\mathbf{v}\sec(st_0) - \mathbf{w}\tan(st_0))(-\sec(t_{02})) + (-\mathbf{v}\tan(st_0) + \mathbf{w}\sec(st_0))\tan(t_{02})$$

This equation can be separated into its **v** and **w** components, as shown below, as though they are basis vectors.

$$\begin{cases} \sec(t_0) = \sec(st_0)\sec(t_{02}) + \tan(st_0)\tan(t_{02}) \\ \tan(t_0) = \tan(st_0)\sec(t_{02}) + \sec(st_0)\tan(t_{02}) \end{cases} \tag{6.24}$$

Using the **v** equation, the derivation is performed similarly to the derivation ending in equation 6.10.

$$\begin{aligned} \sec(t_0) &= \sec(st_0)\sec(t_{02}) + \tan(st_0)\tan(t_{02}) \\ \sec(t_0)\cos(t_{02}) &= \sec(st_0) + \tan(st_0)\sin(t_{02}) \\ 2\sec(t_0)\cos(t_{02}) + \sec(t_0) - \sec(t_0) &= 2\sec(st_0) + \sec(st_0)\cos(t_{02}) - \sec(st_0)\cos(t_{02}) + 2\tan(st_0)\sin(t_{02}) \end{aligned}$$

$$\begin{aligned} 0 &= (\sec(t_0) + \sec(st_0))(1 - \cos(t_{02})) + 2\tan(st_0)\sin(t_{02}) + (\sec(st_0) - \sec(t_0))(\cos(t_{02}) + 1) \\ 0 &= (\sec(t_0) + \sec(st_0))\frac{1 - \cos(t_{02})}{\cos(t_{02}) + 1} + 2\tan(st_0)\frac{\sin(t_{02})}{\cos(t_{02}) + 1} + (\sec(st_0) - \sec(t_0)) \\ 0 &= (\sec(t_0) + \sec(st_0))\tan^2\left(\frac{t_{02}}{2}\right) + 2\tan(st_0)\tan\left(\frac{t_{02}}{2}\right) + (\sec(st_0) - \sec(t_0)) \\ \rightarrow \tan\left(\frac{t_{02}}{2}\right) &= \frac{-2\tan(st_0) \pm \sqrt{(2\tan(st_0))^2 - 4(\sec(t_0) + \sec(st_0))(\sec(st_0) - \sec(t_0))}}{2(\sec(t_0) + \sec(st_0))} \\ &= \frac{-\tan(st_0) \pm \sqrt{\tan^2(st_0) - \sec^2(st_0) + \sec^2(t_0)}}{\sec(t_0) + \sec(st_0)} \\ &= \frac{-\tan(st_0) \pm \sqrt{-1 + \sec^2(t_0)}}{\sec(t_0) + \sec(st_0)} \\ &= \frac{-\tan(st_0) \pm \tan(t_0)}{\sec(t_0) + \sec(st_0)} \end{aligned}$$

From experimentation, the plus sign is required in the above expression. Therefore the final expression for t_{02} is as follows. The extra integer constant is not needed, that is the principle arctan value is used.

$$\Rightarrow t_{02} = 2\arctan\left(\frac{\tan(t_0) - \tan(st_0)}{\sec(t_0) + \sec(st_0)}\right) \tag{6.25}$$

6.4.3 Implementing in Code

To summarize this splitting operation in code, continuing on from section 1.4, the following is implemented for a hyperbola.

```
std::pair<Curve, Curve> Hyperbola::split(double s) {
    CurveData first, second;
    first.p1 = p1;          second.p1 = operator()(s);
    first.v = v;            second.v =  v*std::sec(s*t0) - w*std::tan(s*t0);
    first.w = w;            second.w = -v*std::tan(s*t0) + w*std::sec(s*t0);
    first.t0 = s*t0;        second.t0 = 2 * std::atan( (std::tan(t0) - std::tan(s*t0))
                            / (std::sec(t0) + std::sec(s*t0)) );
    return std::make_pair(Curve(first), Curve(second));
```

7 All Code

This section presents all the code from this report in one spot.

7.1 Curve Class

```
class CurveData {
public:
    Point p1;
    Vec v, w;
    double t0;
};

class Curve {
public:
    Curve() = default;
    Curve(const Point& p1, const Vec& v, const Vec& w, double t0);
    Curve(const CurveData& data);
    Curve(const Curve& curve);
    virtual Point operator()(double t);
    virtual std::pair<Curve, Curve> split(double s);

protected:
    union {
        struct {
            Point p1;
            Vec v, w;
            double t0;
        };
        CurveData data;
    };
};
```

```
Curve::Curve(const Point& p1, const Vec& v, const Vec& w, double t0) {
    this->p1 = p1;
    this->v = v;
    this->w = w;
    this->t0 = t0;
}
Curve::Curve(const CurveData& data) {
    this->data = data;
}
Curve::Curve(const Curve& curve) {
    this->data = curve.data;
}
Point Curve::operator()(double t) {
    return Point::zero;
}
std::pair<Curve, Curve> Curve::split(double s) {
    return std::make_pair(Curve(), Curve());
}
```

7.2 Line Class

```
class Line : public Curve {
public:
    Line(const Point& p1, const Point& p2);
    Point operator()(double t);
    std::pair<Curve, Curve> split(double s);
};
```

```
Line::Line(const Point& p1, const Point& p2) {
    this->p1 = p1;
    this->v = p2 - p1;
    this->w = Vec::zero;
    this->t0 = 1;
}
Point Line::operator()(double t) {
    return p1 + v * t;
}
std::pair<Curve, Curve> Line::split(double s) {
    CurveData first, second;
    first.p1 = p1;          second.p1 = operator()(s);
    first.v = v*s;          second.v = v*(1-s);
    first.w = Vec::zero;    second.w = Vec::zero;
    first.t0 = 1;           second.t0 = 1;
    return std::make_pair(Curve(first), Curve(second));
}
```

7.3 Circle Class

```
class Circle : public Ellipse {
public:
    Circle(const Point& p1, const Point& p2, const Vec& k, double dist);
};
```

```
Circle::Circle(const Point& p1, const Point& p2, const Vec& k, double dist) {
    assert(dist != 0);
    this->p1 = p1;

    Vec l = p2 - p1;
    double li = abs(l);

    Vec i = l / li;
    Vec j = (dist > 0 ? 1 : -1)* (i * k);
    if (dist < 0)
        dist = -dist;

    double a = li / 2;
    double b = dist / 2 - (li * li) / (8 * dist);

    this->v = i * a + j * b;
    this->w = i * b - j * a;
    this->t0 = -2 * std::atan(b / a) - PI;
}
```

7.4 Parabola Class

```
class Parabola : public Curve {
public:
    Parabola(const Point& p1, const Point& p2, const Vec& k, double dist, double alpha);
    Point operator()(double t);
    std::pair<Curve, Curve> split(double s);
};
```

```
Parabola::Parabola(const Point& p1, const Point& p2, const Vec& k, double dist, double alpha) {
    assert(dist != 0 && alpha >= -90 && alpha <= 90);
    this->p1 = p1;

    Vec l = p2 - p1;
    Vec q = l/2 + (l*k)*dist;
    if (alpha != 0)
        assert(std::abs(dist) <= abs(l / (2*std::tan(alpha * PI/180))));

    Vec i = ArbiRot(k)(alpha * (dist > 0 ? 1 : -1))(unit(l));
    Vec j = (dist > 0 ? 1 : -1) * (i*k);
    if(dist < 0)
        dist = -dist;

    double li = l | i, lj = l | j,
           qi = q | i, qj = q | j;

    double a = (qj/qi - lj/li) / (qi - li);
    double b = lj/li - a*li;

    this->v = li * (i + j*b);
    this->w = li*li * (j*a);
    this->t0 = 1;
}
Point Parabola::operator()(double t) {
    return p1 + v*t + w*t*t;
}
std::pair<Curve, Curve> Parabola::split(double s) {
    CurveData first, second;
    first.p1 = p1;          second.p1 = operator()(s);
    first.v = v*s;          second.v = v*(1-s) + w*(2*s*(1-s));
    first.w = w*(s*s);      second.w = w*((1-s)*(1-s));
    first.t0 = 1;           second.t0 = 1;
    return std::make_pair(Curve(first), Curve(second));
}
```

7.5 Ellipse Class

```
class Ellipse : public Curve {
public:
    Ellipse() = default;
    Ellipse(const Point& p1, const Point& p2, const Vec& normal, double dist, double alpha,
        double ratio);
    Point operator()(double t);
    std::pair<Curve, Curve> split(double s);
};
```

```
Ellipse::Ellipse(const Point& p1, const Point& p2, const Vec& normal, double dist, double alpha,
    double ratio) {
    assert(dist != 0 && alpha >= -90 && alpha <= 90 && ratio != 0);
    this->p1 = p1;

    Vec l = p2 - p1;
    Vec q = l/2 + (l*normal)*dist;

    Vec i = ArbiRot(normal)(alpha * (dist > 0 ? 1 : -1))(unit(l));
    Vec j = (dist > 0 ? 1 : -1) * (i * normal);
    if (dist < 0)
        dist = -dist;
    if (ratio < 0)
        ratio = -ratio;

    double li = l | i, lj = l | j,
           qi = q | i, qj = q | j;

    double k = (qi*li*ratio*ratio*(qi - li) - qi*lj*lj + qj*qj*li) / (2 * (qj*li - qi*lj));
    double h = (lj*(lj - 2*k)) / (2*li*ratio*ratio) + li/2;

    this->v = i*h + j*k;
    this->w = i*k/ratio - j*h*ratio;
    this->t0 = 2 * std::atan((lj-2*k)/(li*ratio)) - PI;
}
Point Ellipse::operator()(double t) {
    t *= t0;
    return p1 + v*(1-std::cos(t)) + w*std::sin(t);
}
std::pair<Curve, Curve> Ellipse::split(double s) {
    CurveData first, second;
    first.p1 = p1;          second.p1 = operator()(s);
    first.v = v;            second.v = v*std::cos(s*t0) - w*std::sin(s*t0);
    first.w = w;            second.w = v*std::sin(s*t0) + w*std::cos(s*t0);
    first.t0 = s*t0;        second.t0 = (1-s)*t0;
    return std::make_pair(Curve(first), Curve(second));
}
```

7.6 Hyperbola Class

Note that `std::sec()` does not exist in the STL. This function was defined separately as the following.

```
namespace std{
    inline double sec(double x) {
        return 1 / cos(x);
    }
}
```

```
class Hyperbola : public Curve {
public:
    Hyperbola() = default;
    Hyperbola(const Point& p1, const Point& p2, const Vec& normal, double dist, double alpha,
        double ratio);
    Point operator()(double t);
    std::pair<Curve, Curve> split(double s);
};
```

```
Hyperbola::Hyperbola(const Point& p1, const Point& p2, const Vec& normal, double dist, double
    alpha, double ratio) {
    assert(dist != 0 && alpha >= -90 && alpha <= 90 && ratio != 0);
    this->p1 = p1;

    Vec l = p2 - p1;
    Vec q = l/2 + (l*normal)*dist;

    Vec i = ArbiRot(normal)(alpha * (dist > 0 ? 1 : -1))(unit(l));
    Vec j = (dist > 0 ? 1 : -1) * (i * normal);
    if (dist < 0)
        dist = -dist;
    if (ratio < 0)
        ratio = -ratio;

    double li = l | i, lj = l | j,
           qi = q | i, qj = q | j;

    double k = (qi*li*ratio*ratio*(qi-li) + qi*lj*lj - qj*qj*li) / (2 * (qi*lj - qj*li));
    double h = (lj*(2*k - lj)) / (2*li*ratio*ratio) + li/2;

    if (h*h - (k*k)/(ratio*ratio) < 0) {
        std::swap(i,j);
        std::swap(h,k);
        std::swap(li,lj);
        std::swap(qi,qj);
        ratio = 1/ratio;
    }

    assert((h>0) == (h-li>0) && (h>0) == (h-qi>0));
    assert(std::abs(qj) < std::abs(lj));

    this->v = i*h + j*k;
    this->w = i*k/ratio + j*h*ratio;
    this->t0 = 2 * std::atan((2*k-lj)/(li*ratio)) + PI * ((q|w) > 0 ? 1 : -1);
}
Point Hyperbola::operator()(double t) {
    t *= t0;
    return p1 + v*(1-std::sec(t)) + w*std::tan(t);
}
std::pair<Curve, Curve> Hyperbola::split(double s) {
    CurveData first, second;
    first.p1 = p1;          second.p1 = operator()(s);
    first.v = v;            second.v =  v*std::sec(s*t0) - w*std::tan(s*t0);
    first.w = w;            second.w = -v*std::tan(s*t0) + w*std::sec(s*t0);
    first.t0 = s*t0;        second.t0 = 2 * std::atan( (std::tan(t0) - std::tan(s*t0))
                            / (std::sec(t0) + std::sec(s*t0)) );
    return std::make_pair(Curve(first), Curve(second));
}
```