The ellipse package

Daan Leijen
daan@microsoft.com

October 10, 2015

1 Introduction

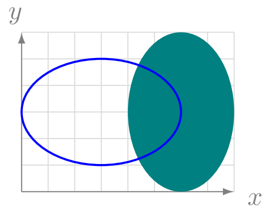
LaTeX has many advanced graphics packages now, the most extensive are tikz and pstricks. However, these are also large packages that take long to load and may not always work on all drivers. The standard pict2e package removes many of the previous limitations of the ‘old’ LaTeX `picture` environment and makes it a *lean and portable* alternative to the more full featured packages. However, even though it can draw circles and circle arcs well, it lacks the ability to draw ellipses and elliptical arcs. This package adds these functions on top of the standard pict2e primitives (i.e. the `\cbezier` command).

2 Drawing ellipses

`\ellipse` `{`⟨*x-radius*⟩`}{`⟨*y-radius*⟩`}`
`\ellipse*`

These commands draw an ellipse with the specified radiï. The `\ellipse` command draws a stroked ellipse with the current `\linethickness` while `\ellipse*` draws a filled ellipse with the current `\color`. For example:

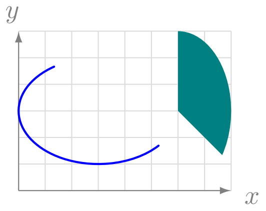
```
\setlength{\unitlength}{10pt}%
\begin{picture}(6,8)
  \linethickness{0.8pt}%
  \put(6,3){\color{teal}\ellipse*{2}{3}}%
  \put(3,3){\color{blue}\ellipse{3}{2}}%
\end{picture}
```



`\earc` `[`⟨*start-angle*⟩`,`⟨*end-angle*⟩`]{`⟨*x-radius*⟩`}{`⟨*y-radius*⟩`}`
`\earc*`

These commands draw part of an ellipse with the specified radiï. The `\earc` command draws a stroked elliptical arc with the current `\linethickness` while `\earc*` draws a filled elliptical ‘pie slice’ with the current `\color`. The optional argument specifies a start and end-angle in degrees which must be between −720 and 720 (but can be fractional). The endings of the arcs are determined by the *cap* setting: `\buttcap` (default), `\roundcap` (add half disc), or `\squarecap` (add half square).

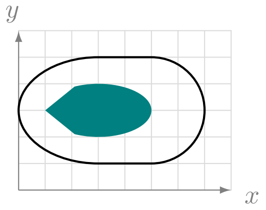
```
\put(3,3){%
  \color{blue}\roundcap\earc[135,330]{3}{2}}%
\put(6,3){%
  \color{teal}\earc*[-45,90]{2}{3}}%
```



`\elliparc` [⟨*initial*⟩]{⟨*center-x*⟩}{⟨*center-y*⟩}{⟨*x-rad*⟩}{⟨*y-rad*⟩}{⟨*start-angle*⟩}{⟨*end-angle*⟩}

The core elliptical arc routine. These are to be used with path commands, like `\lineto`, `\moveto`, `\strokepath`, etc, and can draw an elliptical arc at any center point. The optional argument specifies the initial drawing action: the default is 0 (`\lineto`) which draws a line to the arc starting point, the value 1 (`\moveto`) just moves to the starting point, and 2 does nothing as an initial action. If the start angle is larger than the end angle, the arc is drawn clockwise, and otherwise anti-clockwise.

```
\elliparc[1]{3}{3}{3}{2}{90}{270}%
\elliparc{5}{3}{2}{2}{-90}{90}%
\closepath\strokepath
\color{teal}%
\moveto(1,3)
\elliparc{3}{3}{2}{1}{-135}{135}%
\closepath
\fillpath
```

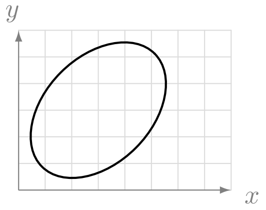


Note how the two initial arcs are automatically connected by a line segment from $(3, 1)$ to $(5, 1)$ (due to the default optional argument of 0 that uses a `\lineto` command to the starting point of the arc). Similarly, we use such initial line segment and a `\closepath` to draw the triangular side of the inner ellipse.

2.1 Rotated ellipses

There is no direct command to rotate an ellipse but you can use the standard `\rotatebox` command from the graphicx package. For example:

```
\put(3,3){%
  \rotatebox[origin=c]{45}{\ellipse{3}{2}}%
}%
```



2.2 Using the picture environment inline

The standard LaTeX `picture` environment is nowadays quite powerful and convenient. Read the latest pict2e documentation and "The unknown *picture* environment" [2] for more information. One particularly nice feature is that we can create a picture as `\begin{picture}(0,0)` to give it zero space. This can be used for example to define an `\ellipbox` command like:

```
Boxed numbers:
  \ellipbox{1}, \ellipbox{123}.
```

Boxed numbers: ①, (123).

We also used this command to draw the ellipse in the title of this article, and it is defined as:

```
\newsavebox{\@ebox}
\newcommand*\@unit[1]{\strip@pt\dimexpr#1\relax}%
\newcommand*\ellipbox[1]{%
  \begingroup
  \savebox{\@ebox}{#1}%
  \setlength{\unitlength}{1pt}%
  \hspace*{0.8ex}%
  \begin{picture}(0,0)%
   \put(\@unit{0.5\wd\@ebox},\@unit{0.5\ht\@ebox - 0.5\dp\@ebox}){%
     \ellipse{\@unit{0.8ex + 0.5\wd\@ebox}}{\@unit{0.8ex + 0.5\ht\@ebox}}%
  }%
  \end{picture}%
  \usebox{\@ebox}\hspace{0.25ex}\endgroup}
```

This is not the best code possible but it hopefully gives you a good idea on how to implement your own boxes. Note the use of the `\@unit` macro to convert dimensions to units, which is also why we need to set the `\unitlength` to `1pt` here.

References

[1] M. Abramowitz and I.A. Stegun: *Handbook of Mathematical Functions*, `people.math.sfu.ca/~cbm/aands`, 1964

[2] Claudio Beccari: *The unknown* picture *environment*, TUGBoat, vol. 33(1), 2012. `tug.org/TUGboat/tb33-1/tb103becc-picture.pdf`

[3] Luc Maisonobe: *Drawing an elliptical arc using polylines, quadratic or cubic Bézier lines.* `www.spaceroots.org/documents/ellipse/elliptical-arc.pdf`, 2003

[4] S. Rajan, Sichun Wang, R. Inkol, and A. Joyal: *Efficient approximations for the arctangent function.* In Signal Processing Magazine, vol. 23(3), pages 108–111, May 2006

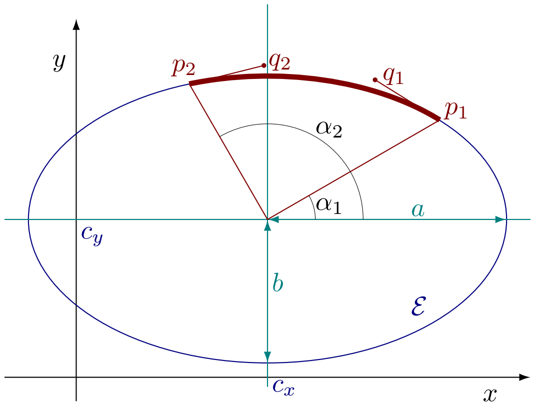


Figure 1: Approximating an elliptical arc with a cubic Bézier curve. The center of the ellipse is at (c_x, c_y) with a horizontal radius of a and a vertical one b. The elliptical arc goes from α_1 to α_2 and is approximated with a thick red cubic Bézier curve. The curve starts at p_1 and ends in p_2 with two control points q_1 and q_2. The curve was drawn using the command `\elliparc{4}{3.3}{5}{3}{30}{120}`.

3 Elliptical arcs as Bézier curves

Drawing an ellipse or part of an ellipse (*elliptical arc*) using Bézier curves requires some math to determine the right control points of the Bézier curve. Figure 1 establishes some notation. We do not consider rotated ellipses here and always use a for the x-radius and b for the y-radius. We are interested in finding the Bézier curve between the α_1 and α_2 angles, which implies finding the starting point p_1, the end point p_2 and the control points q_1 and q_2.

Each point on an ellipse is determined by the following parametric equation:

$$\mathcal{E}(t) = (c_x + a \cdot cos(t), c_y + b \cdot sin(t))$$

where t is the parametric angle. The parametric angle t is just a property of the ellipse and has no 'real' counterpart. Figure 2 gives some helpful intuition how the α angles and t angles are related: we can imagine drawing a unit circle inside an ellipse where for every t angle on the unit circle we have a corresponding point and angle α on the ellipse. From the definition of $\mathcal{E}$ it is straightforward to derive a parametric angle t_i for some α_i:

$$t_i = arctan_2(\frac{sin(\alpha_i)}{b}, \frac{cos(\alpha_i)}{a})$$

Given this relation, the start and end points of our curve are simply:

$$p_1 = \mathcal{E}(t_1)$$
$$p_2 = \mathcal{E}(t_2)$$

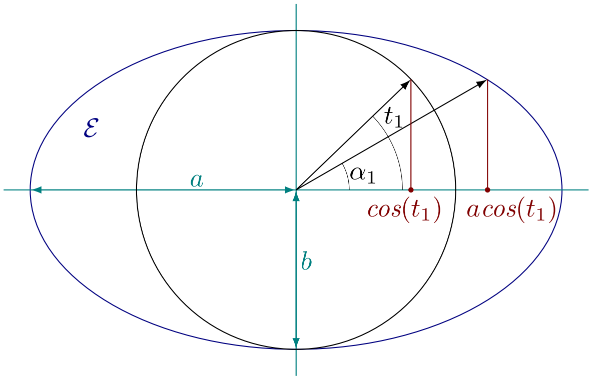


Figure 2: The relation between the parametric angle t_1 and the angle α_1 to the point on the ellipse. All points on the ellipse are defined by the parametric equation $\mathcal{E}(t) = (c_x + a \cdot cos(t), c_y + b \cdot sin(t))$

To be able to calculate optimal control points q we need to also determine the tangent of each point on the ellipse, which is given by the derivative of $\mathcal{E}$:

$$\mathcal{E}'(t) = (-a \cdot sin(t), b \cdot cos(t))$$

The derivation of the optimal Bézier control points for an ellipse is quite involved, see [3] for a nice overview. For a quadratic Bézier curve, it turns out the optimal control points are determined as:

$$\begin{aligned} q_1 &= p_1 + tan(\frac{t_2 - t_1}{2}) \cdot \mathcal{E}'(t_1) \\ &= p_2 - tan(\frac{t_2 - t_1}{2}) \cdot \mathcal{E}'(t_2) \end{aligned}$$

while for a cubic Bézier curve, one solution for optimal control points is:

$$\begin{aligned} q_1 &= p_1 + \kappa \cdot \mathcal{E}'(t_1) \\ q_2 &= p_2 - \kappa \cdot \mathcal{E}'(t_2) \\ \kappa &= sin(t_2 - t_1)\frac{\sqrt{4 + 3tan^2(\frac{t_2 - t_1}{2})} - 1}{3} \end{aligned}$$

We will use cubic bezier curves since they look best. However, a naïve implementation may be too expensive in LaTeX: if we count the expensive operations, we need about 11 cos/sin operations, plus a $\sqrt{}$ and 2 $arctan$ operations.

3.1 Optimizing elliptic arc equations

Fortunately, we can improve upon this. First we note:

$$\begin{aligned}
t_i &= arctan_2(\frac{sin(\alpha_i)}{b}, \frac{cos(\alpha_i)}{a}) \\
&= arctan(\frac{a}{b} tan(\alpha_i)) \\
&= arctan(\iota_i) && \text{(introducing } \iota_i \text{ for } \tfrac{a}{b} tan(\alpha_i))
\end{aligned}$$

where we write ι_i for $\frac{a}{b} tan(\alpha_i)$. Now,

$$\begin{aligned}
cost_i &= cos(t_i) \\
&= cos(arctan(\iota_i)) && \text{(geometry and pythagorean theorem)} \\
&= \pm_i \frac{1}{\sqrt{1+\iota_i^2}}
\end{aligned}$$

with

$$\pm_i = \mathsf{if}\ cos(\alpha_i) < 0\ \mathsf{then}\ -\ \mathsf{else}\ +$$

Later we will see how we can efficiently calculate the square root term, but first do the same derivation for the *sin* function:

$$\begin{aligned}
sint_i &= sin(t_i) \\
&= sin(arctan(\frac{a}{b} tan(\alpha_i))) \\
&= sin(arctan(\iota_i)) \\
&= \pm_i \frac{\iota_i}{\sqrt{1+\iota_i^2}}
\end{aligned}$$

Note that the interaction between the *sin* and ι_i term (whose sign is determined by $tan(\alpha_i)$) allows us to reuse the sign function used for $cost_i$.

Using the previous equalities we can restate the parametric equations in terms of $sint_i$ and $cost_i$:

$$\begin{aligned}
\mathcal{E}_i &= (c_x + a \cdot cost_i), c_y + b \cdot sint_i) \\
\mathcal{E}'_i &= (-a \cdot sint_i, b \cdot cost_i)
\end{aligned}$$

This takes care of p_1 and p_2. The control points q still need $sin(t_2 - t_1)$ and $tan(\frac{t_2-t_1}{2})$. The halving rule on *tan* gives us:

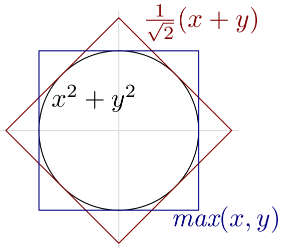
$$tan(\frac{t_2 - t_1}{2}) = \frac{1 - cos(t_2 - t_1)}{sin(t_2 - t_1)} \quad \text{([1, page 71, 4.3.20])}$$

So that leaves $sin(t_2 - t_1)$ and $cos(t_2 - t_1)$. Using the addition laws it follows:

$$\begin{aligned}
sin(t_2 - t_1) &= sint_2 cost_1 - cost_2 sint_1 \quad \text{([1, page 72, 4.3.16])} \\
cos(t_2 - t_1) &= cost_2 cost_1 + sint_2 sint_1 \quad \text{([1, page 72, 4.3.17])}
\end{aligned}$$

3.2 Circular square roots

Now, we only need two tan operations to calculate the initial ι_1 and ι_2 terms but we still have three square roots: $\sqrt{1+\iota_i^2}$ and $\sqrt{4+3tan^2(\frac{t_2-t_1}{2})}$. Fortunately, both have the form $\sqrt{x^2+y^2}$. For this form, we can make a very good initial guess for the square root, since this is the parametric equation for a circle. The two good initial guesses form a 'square' and 'diamond' around this circle, namely $max(|x|,|y|)$ and $\frac{1}{\sqrt{2}}|x+y|$. Each one can be superior depending if x and y are close or not, but it can be shown that the best choice is always the largest of these. Using this guess as an initial seed, we can do a standard Newton-Raphson iteration to find a the square root where we only need 2 or 3 steps to achieve the desired precision. Let's define a 'circular square root' function $csqrt$ such that $csqrt(x,y) \approx \sqrt{x^2+y^2}$ as:



$$
\begin{aligned}
csqrt(x,y) = \text{let}\quad & sqr = x^2+y^2 \\
& x_0 = max(|x|,|y|,\frac{1}{\sqrt{2}}|x+y|) \\
& x_1 = (x_0 + \frac{sqr}{x_0})/2 \\
& x_2 = (x_1 + \frac{sqr}{x_1})/2 \\
\text{in}\quad & x_2
\end{aligned}
$$

3.3 The optimized elliptical Bézier equations

Taking it all together, we get the following equations for a cubic Bézier curve approximation of an elliptical arc, where we assume as input the center point (c_x, c_y), the x- and y-radius (a, b), and a start and end angle α_1 and α_2. It is assumed that $\alpha_1 \neq \alpha_2$ and $a \geq 0, b \geq 0$. Of course, with bezier curves one should build a full ellipse of parts where for each part $|\alpha_1 - \alpha_2| \leq 90$. Given these parameters, the start and end point p_1 and p_2, and the control points q_1 and q_2 are defined as:

$$
\begin{aligned}
& p_1 = \mathcal{E}_1 \\
& p_2 = \mathcal{E}_2 \\
& q_1 = p_1 + \kappa \cdot \mathcal{E}_1' \\
& q_2 = p_2 - \kappa \cdot \mathcal{E}_2' \\
& \mathcal{E}_i = (c_x + a \cdot cost_i, c_y + b \cdot sint_i) \\
& \mathcal{E}_i' = (-a \cdot sint_i, b \cdot cost_i)
\end{aligned}
$$

The $cost_i$ and $sint_i$ are calculated as:

$$sint_i = \pm_i \frac{\iota_i}{\rho_i}$$

$$cost_i = \pm_i \frac{1}{\rho_i}$$

with

$$\iota_i = \frac{a}{b} tan(\alpha_i)$$

$$\rho_i = csqrt(1, \iota_i) \;\; (\approx \sqrt{1 + \iota_i^2})$$

$$\pm_i = \textsf{if } cos(\alpha_i) < 0 \textsf{ then } - \textsf{ else } +$$

And finally, the κ term can be defined as:

$$\kappa = sint_{21} \frac{\kappa_{sqrt} - 1}{3}$$

with

$$\begin{aligned} sint_{21} &= sint_2 \, cost_1 - cost_2 \, sint_1 \;\; (= sin(t_2 - t_1)) \\ cost_{21} &= cost_2 \, cost_1 + sint_2 \, sint_1 \;\; (= cos(t_2 - t_1)) \\ \kappa_{tan} &= \frac{1 - cost_{21}}{sint_{21}} \;\; (\text{note: divides by zero if } \alpha_1 = \alpha_2) \\ \kappa_{sqrt} &= csqrt(\sqrt{4}, \sqrt{3} \cdot \kappa_{tan}) \;\; (\approx \sqrt{4 + 3\kappa_{tan}^2}) \end{aligned}$$

4 Implementation

Generally, we use e-TEX division to divide dimensions, where we divide $\langle dim_1 \rangle$ by $\langle dim_2 \rangle$ using: `\dimexpr 1pt *` $\langle dim_1 \rangle$`/`$\langle dim_2 \rangle$`\relax` since it keeps a 64-bit intermediate result for such 'scaling' expressions. Note that both $\langle dim \rangle$ expressions occur in an integer context and TEX will convert them to numbers automatically (i.e. in `sp` units).

4.1 Generic math and trigonometry routines

`\pIIe@csedef` `{`$\langle csname \rangle$`}`*pattern*`{`$\langle body \rangle$`}`

Define a macro by a *csname*. Just like the `\csedef` function from etoolbox package

```
\providecommand*\pIIe@csedef[1]{\expandafter\edef\csname #1\endcsname}
```

`\pIIe@ellip@csqrt` `{`$\langle dimen_x \rangle$`}{`$\langle dimen_y \rangle$`}{`$\langle dimreg_{res} \rangle$`}`

Calculates $res \approx \sqrt{x^2 + y^2}$ and caches previous results for efficiency.

Overwrites `\@ovxx`,`\@ovyy`,`\@ovdx`,`\@ovdy`,`\@tempa`, and `\dimen@`.

```
\newcommand*\pIIe@ellip@csqrt[3]{%
  \@ovxx=#1\relax
  \ifdim\@ovxx<\z@\@ovxx-\@ovxx\fi
  \@ovyy=#2\relax
  \ifdim\@ovyy<\z@\@ovyy-\@ovyy\fi
  \edef\pIIe@csname{@csqrt(\number\@ovxx,\number\@ovyy)}%
  \expandafter\ifx\csname\pIIe@csname\endcsname\relax
    \pIIe@ellip@csqrt@%
    \pIIe@csedef{\pIIe@csname}{\the\dimen@}%
    #3\dimen@
  \else
    #3\dimexpr\csname\pIIe@csname\endcsname\relax
  \fi
}
```

`\pIIe@ellip@csqrt@` Internal routine: calculates `\dimen@` $\approx \sqrt{x^2+y^2}$. where $x \geq 0$ and $y \geq 0$, and `\@ovxx` $= x$ and `\@ovyy` $= y$.

Overwrites `\@ovdx`,`\@ovdy`, and `\@tempa`.

```
\newcommand*\pIIe@ellip@csqrt@{%
```

First determine $max(x, y, \frac{1}{\sqrt{2}}(x+y))$ in `\dimen@`. Put the sum $x+y$ in `\@ovdx`.

```
  \@ovdx\@ovxx
  \advance\@ovdx by \@ovyy
```

Put initial guess in `\dimen@` $= max(|x|, |y|, \frac{1}{\sqrt{2}}(x+y))$.

```
  \dimen@0.7071067\@ovdx
  \ifdim\dimen@<\@ovyy\dimen@\@ovyy\fi
  \ifdim\dimen@<\@ovxx\dimen@\@ovxx\fi
```

To prevent overflowing TEX dimensions we only do a further Newton-Raphson approximation if the sum $x+y$ is less than 128pt. Otherwise, for our application, the initial guess is still very precise since $x \ll y$ in that case.

```
  \ifdim\@ovdx<128\p@
```

Set `\@ovxx` to x^2+y^2

```
    \edef\@tempa{\strip@pt\@ovxx}%
    \@ovxx\@tempa\@ovxx
    \edef\@tempa{\strip@pt\@ovyy}%
    \@ovyy\@tempa\@ovyy
    \advance\@ovxx by \@ovyy
```

Do two steps of Newton-Raphson (should we do three?)

```
    \advance\dimen@ by \dimexpr1pt * \@ovxx/\dimen@\relax
    \divide\dimen@ by 2%
    \advance\dimen@ by \dimexpr1pt * \@ovxx/\dimen@\relax
    \divide\dimen@ by 2%
  \fi
```

Result is `\dimen@`.

```
}
```

`\pIIe@atan@` Approximate the *arctan* using

$$x \cdot \frac{\pi}{4} - x \cdot (|x|-1) \cdot (0.2447 + 0.0663 \cdot |x|)$$

This approximation was described by Rajan et al. [4].

The `\IIe@atan@` computes the arctan of `\dimen@` which must be between -1 and 1, and stores it in `\dimen@` again. Overwrites `\@tempdim`(a,b,c,d),`\@tempa`, and `\dimen@`.

```
\newcommand*\pIIe@atan@{%
```

`\dimen@` contains x.

```
  \@tempdima\dimen@
```

Set `\@dimtmpb` to $|x|$

```
  \@tempdimb\@tempdima
  \ifdim\@tempdimb<\z@\@tempdimb-\@tempdimb\fi
  \dimen@0.0663\@tempdimb
  \advance\dimen@ 0.2447pt\relax
  \advance\@tempdimb -1pt\relax
  \edef\@tempa{\strip@pt\@tempdimb}%
  \dimen@\@tempa\dimen@
  \edef\@tempa{\strip@pt\@tempdima}%
  \dimen@\@tempa\dimen@
  \dimen@-\dimen@
```

Add $x \cdot \frac{\pi}{4}$ ($\approx 0.7853 \cdot x$).

```
  \advance\dimen@ 0.7853\@tempdima
}
```

`\pIIe@atantwo` $\{\langle \mathit{dimen}_y\rangle\}\{\langle \mathit{dimen}_x\rangle\}\{\langle \mathit{dimreg}_{res}\rangle\}$

Calculate $\langle \mathit{res}\rangle = \mathit{arctan}_2(y, x)$ and caches the result for later use.

Overwrites `\@tempdim`(a,b,c,d),`\@tempa`, and `\dimen@`. Both y and x must be dimensions.

```
\newcommand*\pIIe@atantwo[3]{%
  \edef\pIIe@csname{@atan2(\number\dimexpr#1\relax,\number\dimexpr#2\relax)}%
  \expandafter\ifx\csname\pIIe@csname\endcsname\relax
    \pIIe@atantwo@{#1}{#2}{#3}%
    \pIIe@csedef{\pIIe@csname}{\the\dimexpr#3\relax}%
  \else
    #3\dimexpr\csname\pIIe@csname\endcsname\relax
  \fi
}
```

`\pIIe@atantwo@` $\{\langle \mathit{dimen}_y\rangle\}\{\langle \mathit{dimen}_x\rangle\}\{\langle \mathit{dimreg}_{res}\rangle\}$

Calculate $\langle \mathit{res}\rangle = \mathit{arctan}_2(y, x)$. Overwrites `\@tempdim`(a,b,c,d),`\@tempa`, and `\dimen@`. Both y and x must be dimensions.

```
\newcommand*\pIIe@atantwo@[3]{%
  \@tempdima\dimexpr#2\relax
  \@tempdimb\dimexpr#1\relax
```

Handle extremes

```
  \ifdim\@tempdima=\z@\relax
    \ifdim\@tempdimb>\z@\relax\dimen@90\p@
    \else\ifdim\@tempdimb<\z@\relax\dimen@-90\p@
    \else\dimen@0\p@
    \fi\fi
  \else
```

Save angle adjustment term in `\@tempdimd`.

```
    \@tempdimd\z@
    \ifdim\@tempdima<\z@\relax
      \ifdim\@tempdimb<\z@\relax\@tempdimd-180\p@
      \else\@tempdimd180\p@
      \fi
    \fi
```

Divide $\frac{y}{x}$ and check if $-1 \leq \frac{y}{x} \leq 1$.

```
    \dimen@\dimexpr1pt * \@tempdimb/\@tempdima\relax
    \@tempdimc\dimen@
    \ifdim\@tempdimc<\z@\relax\@tempdimc-\@tempdimc\fi
    \ifdim\@tempdimc>\p@\relax
```

Use the equality $arctan(x) = \pm\frac{1}{2}\pi - arctan(\frac{1}{x})$ to stay within the valid domain of `\pIIe@atan@`. The sign ± is positive when $x \geq 0$ and negative otherwise.

```
      \dimen@\dimexpr1pt * \@tempdima/\@tempdimb\relax
      \ifdim\dimen@<\z@\relax\def\@tempsign{-}\else\def\@tempsign{}\fi
      \pIIe@atan@
      \dimen@-\dimen@
      \advance\dimen@ by \@tempsign1.5707pt\relax
    \else
      \pIIe@atan@
    \fi
```

And convert back to degrees ($\frac{180}{\pi} \approx 57.29578$)

```
    \dimen@57.29578\dimen@
```

Apply angle adjustment

```
    \advance\dimen@ by \@tempdimd
  \fi
  #3\dimen@%
}
```

4.2 Sub routines for drawing an elliptical arc

`\pIIe@noneto` {$\langle dimen_x \rangle$}{$\langle dimen_y \rangle$}

Ignores its arguments. Used as a no-op instead of `\pIIe@lineto` or `pIIe@moveto`.

```
\newcommand*\pIIe@noneto[2]{}
```

`\pIIe@ellip@sincost@` {$\langle \alpha_i \rangle$}{$\langle i =$ *one or two* $\rangle$}

Calculate $sint_i$ and $cost_i$ into the `\@ellip(sin/cos)`i. Assumes `\@ellipratio` $= \frac{a}{b}$.

```
\newcommand*\pIIe@ellip@sincost@[2]{%
```

Put the $sin(\alpha_i)$ and $cos(\alpha_i)$ into `\@tempdima` and `\@tempdimb`.

```
  \CalculateSin{#1}%
  \CalculateCos{#1}%
  \@tempdima\UseSin{#1}\p@
  \@tempdimb\UseCos{#1}\p@
```

Check for extremes where $tan = \pm\infty$.

```
  \ifdim\@tempdima=\p@\relax
    \pIIe@csedef{@ellipsin#2}{1}%
    \pIIe@csedef{@ellipcos#2}{0}%
  \else\ifdim\@tempdima=-\p@\relax
    \pIIe@csedef{@ellipsin#2}{-1}%
    \pIIe@csedef{@ellipcos#2}{0}%
  \else
```

Calculate ι_i in `\@tempdimc` and $\sqrt{1+\iota_i^2}$ in `\@tempdimd`, and derive $sint_i$ and $cost_i$.

```
    \@tempdimc\@ellipratio\dimexpr1pt * \@tempdima/\@tempdimb\relax
    %\typeout{ i#2=\the\@tempdimc, sin(#1)=\the\@tempdima}%
    \pIIe@ellip@csqrt{\p@}{\@tempdimc}\@tempdimd
    \ifdim\@tempdimb<\z@\relax\@tempdimd-\@tempdimd\fi
    \pIIe@csedef{@ellipsin#2}{\strip@pt\dimexpr1pt * \@tempdimc/\@tempdimd\relax}%
    \pIIe@csedef{@ellipcos#2}{\strip@pt\dimexpr1pt * \p@/\@tempdimd\relax}%
  \fi\fi
}
```

`\pIIe@ellip@sincost` {$\langle\alpha_1\rangle$}{$\langle\alpha_2\rangle$}

Calculate $sint_i$ and $cost_i$ into the `\@ellip(sin/cos)(one/two)`. Assumes `\@ovro`$= a$ and `\@ovri`$= b$ with $b \neq 0$.

```
\newcommand*\pIIe@ellip@sincost[2]{%
```

Set `\@ellipratio` to the ratio $\frac{a}{b}$.

```
  %\typeout{ calc sin cos: angles (#1,#2), radii: (\the\@ovro,\the\@ovri)}%
  \edef\@ellipratio{\strip@pt\dimexpr1pt * \@ovro/\@ovri\relax}%
```

And calculate $sint_i$ and $cost_i$

```
  \pIIe@ellip@sincost@{#1}{one}%
  \pIIe@ellip@sincost@{#2}{two}%
  %\typeout{ sincos(a=#1)=(\@ellipsinone,\@ellipcosone), sincos(a=#2)=(\@ellipsintwo,\@ellipcos
}
```

`\pIIe@omega` {$\langle i =$ *one or two*$\rangle$}

Calculates $\mathcal{E}_i$ into `\@tempdima` and `\@tempdimb`. Assumes `\@ovro`$= a$ and `\@ovri`$= b$.

```
\newcommand*\pIIe@omega[3]{%
  \@tempdima\csname @ellipcos#3\endcsname\@ovro
  \advance\@tempdima by #1\relax
  \@tempdimb\csname @ellipsin#3\endcsname\@ovri
  \advance\@tempdimb by #2\relax
}
```

`\pIIe@omegai` {$\langle i =$ *one or two*$\rangle$}

Calculates $\mathcal{E}'_i$ into `\@tempdimc` and `\@tempdimd`. Assumes `\@ovro`$= a$ and `\@ovri`$= b$.

```
\newcommand*\pIIe@omegai[1]{%
  \@tempdimc\csname @ellipsin#1\endcsname\@ovro
  \@tempdimc-\@tempdimc
  \@tempdimd\csname @ellipcos#1\endcsname\@ovri
}
```

`\pIIe@ellip@kappa` Calculates κ, expects `\@ellip(sin/cos)(one/two)` to be defined.

```
\newcommand*\pIIe@ellip@kappa{%
```

Calculate $sint_{21}$ and $cost_{21}$ in `\@tempdima` and `\@tempdimb`.

```
  \@ovyy\@ellipsinone\p@
  \@ovxx\@ellipcosone\p@
  \@tempdima\@ellipcostwo\@ovyy
  \@tempdima-\@tempdima
  \advance\@tempdima by \@ellipsintwo\@ovxx
  \@tempdimb\@ellipcostwo\@ovxx
  \advance\@tempdimb by \@ellipsintwo\@ovyy
```

First test if $sint_{21} = 0$ to prevent division by zero. In that case, it must have been that $\alpha_1 = \alpha_2$ and we set κ to zero so it the control points become equal to the start and end point.

```
  \ifdim\@tempdima=\z@\relax
    \edef\@ellipkappa{0}%
  \else
```

Calculate κ_{tan} in `\dimen@`

```
    \dimen@\dimexpr1pt - \@tempdimb\relax
    \dimen@\dimexpr1pt * \dimen@/\@tempdima\relax
```

Calculate κ_{sqrt} in `\dimen@`

```
    \pIIe@ellip@csqrt{2\p@}{1.73205\dimen@}{\dimen@}%
```

Calculate κ in `\dimen@`

```
    \advance\dimen@ by -\p@
    \divide\dimen@ by 3%
    \edef\@tempa{\strip@pt\@tempdima}%
    \dimen@\@tempa\dimen@
    \edef\@ellipkappa{\strip@pt\dimen@}%
  \fi
  %\typeout{ calculated kappa: \@ellipkappa}%
}
```

4.3 Core routines for drawing elliptical arcs

`\pIIe@elliparc@` `[`$\langle start\rangle$`]{`$\langle c_x\rangle$`}{`$\langle c_y\rangle$`}{`$\langle \alpha_1\rangle$`}{`$\langle \alpha_2\rangle$`}`
Assumes that the radii are set as `\@ovro`$= a$ and `\@ovri`$= b$. This is the main routine for drawing an elliptic arc, where $|\alpha_2 - \alpha_1| \leq 90$.

```
\newcommand*\pIIe@elliparc@[5]{%
  %\typeout{elliparc: #1, center: (#2, #3), radius (\the\@ovro, \the\@ovri),angle (#4, #5)}%
```

Define initial action: 0 (lineto), 1(moveto), or 2 (nothing)

```
\ifcase #1\relax
    \let\@ellip@startto\pIIe@lineto
\or \let\@ellip@startto\pIIe@moveto
\or \let\@ellip@startto\pIIe@noneto%
\else\PackageWarning{ellipse}{Illegal initial action in \protect\elliparc: %
        must be one of 0 (lineto), 1 (moveto) or 2 (do nothing) but I got: #1}%
\fi
```

Perform just the start action if the radii are zero

```
\ifdim\@ovro=\z@\relax\@ovri\z@\fi
\ifdim\@ovri=\z@\relax
  \@ellip@startto{#2}{#3}%
\else
```

Calculate $sint_i$ and $cost_i$ first into the `\@ellip(sin/cos)(one/two)` registers.

```
  \pIIe@ellip@sincost{#4}{#5}%
```

And draw..

```
  \pIIe@elliparc@draw{#2}{#3}%
\fi
}
```

`\pIIe@elliparc@t` `[`⟨*start*⟩`]{`⟨c_x⟩`}{`⟨c_y⟩`}{`⟨t_1⟩`}{`⟨t_2⟩`}`

Assumes that the radii are set as `\@ovro`= a and `\@ovri`= b. Moreover, this routine take t_1 and t_2 as the angles of the ellipse equation (instead of real angles α_i). This routine is mainly for other libraries that may already have computed the t angles and need a bit more efficiency.

```
\newcommand*\pIIe@elliparc@t[5]{%
```

Define initial action: 0 (lineto), 1(moveto), or 2 (nothing)

```
\ifcase #1\relax
    \let\@ellip@startto\pIIe@lineto
\or \let\@ellip@startto\pIIe@moveto
\or \let\@ellip@startto\pIIe@noneto%
\else\PackageWarning{ellipse}{Illegal initial action in \protect\elliparc: %
        must be one of 0 (lineto), 1 (moveto) or 2 (do nothing) but I got: #1}%
\fi
```

Perform just the start action if the radii are zero

```
\ifdim\@ovro=\z@\relax\@ovri\z@\fi
\ifdim\@ovri=\z@\relax
  \@ellip@startto{#2}{#3}%
\else
```

Calculate $sint_i$ and $cost_i$ first into the `\@ellip(sin/cos)(one/two)` registers.

```
  \CalculateSin{#4}\CalculateCos{#4}%
  \edef\@ellipsinone{\UseSin{#4}}%
  \edef\@ellipcosone{\UseCos{#4}}%
  \CalculateSin{#5}\CalculateCos{#5}%
  \edef\@ellipsintwo{\UseSin{#5}}%
  \edef\@ellipcostwo{\UseCos{#5}}%
```

And draw..

```
    \pIIe@elliparc@draw{#2}{#3}%
  \fi
}
```

`pIIe@elliparc@draw` `{`$\langle c_x\rangle$`}{`$\langle c_y\rangle$`}`

Expects a =`\@ovro`, b = `\@ovri`, and `\@ellip(sin/cos)(one/two)` defined. `\@ellipstarto` should contain the initial drawing action and is called with an initial x and y coordinate (usually equal to `\pIIe@lineto`,`\pIIe@moveto`, or `pIIe@noneto`).

```
\newcommand*\pIIe@elliparc@draw[2]{%
% Calculate $\kappa$.
%    \begin{macrocode}
  \pIIe@ellip@kappa%
```

Now we are ready to compute the control points. First p_1.

```
  \pIIe@omega{#1}{#2}{one}%
  %\typeout{ point one: (\the\@tempdima,\the\@tempdimb)}%
```

The coordinates are added to the path if and how necessary:

```
  \@ellip@startto\@tempdima\@tempdimb
```

Add control point q_1

```
  \pIIe@omegai{one}%
  \advance\@tempdima by \@ellipkappa\@tempdimc
  \advance\@tempdimb by \@ellipkappa\@tempdimd
  \pIIe@add@nums\@tempdima\@tempdimb
  %\typeout{ control one: (\the\@tempdima,\the\@tempdimb)}%
```

Calculate p_2

```
  \pIIe@omega{#1}{#2}{two}%
```

Add control point q_1

```
  \pIIe@omegai{two}%
  \@tempdimc\@ellipkappa\@tempdimc
  \@tempdimd\@ellipkappa\@tempdimd
  \@tempdimc-\@tempdimc
  \@tempdimd-\@tempdimd
  \advance\@tempdimc by \@tempdima
  \advance\@tempdimd by \@tempdimb
  \pIIe@add@nums\@tempdimc\@tempdimd
  %\typeout{ control two: (\the\@tempdimc,\the\@tempdimd)}%
```

And finally add p_2 to the path

```
  \pIIe@add@CP\@tempdima\@tempdimb
  %\typeout{ point two: (\the\@tempdima,\the\@tempdimb)}%
  \pIIe@addtoGraph\pIIe@curveto@op
}
```

4.4 Normalizing elliptical arcs

`pIIe@elliparc` [$\langle start\rangle$]{$\langle c_x\rangle$}{$\langle c_y\rangle$}{$\langle a\rangle$}{$\langle b\rangle$}{$\langle \alpha_1\rangle$}{$\langle \alpha_2\rangle$}

`pIIe@@elliparc`

These two macros check the arguments and normalize the angles.

```
\newcommand*\pIIe@elliparc[7][0]{%
```

Store the radii in registers, where `\@ovro`$= a$ and `\@ovri`$= b$.

```
  \@ovro #4\relax
  \@ovri #5\relax
  \iffalse%dim\@ovro=\@ovri
```

Call the circular arc routine if the x- and y-radius are equal

```
    \pIIe@arc[#1]{#2}{#3}{#4}{#6}{#7}
  \else
```

Normalize angles such that the arc angle $|\alpha_2 - \alpha_1| \le 720$. Store the arc angle in `\@arclen`.

```
    \ifdim \@ovro<\z@ \pIIe@badcircarg\else
      \ifdim \@ovri<\z@ \pIIe@badcircarg\else
        \@arclen #7\p@ \advance\@arclen -#6\p@
        \ifdim \@arclen<\z@ \def\@tempsign{-}\else\def\@tempsign{}\fi
        \ifdim \@tempsign\@arclen>720\p@
          \PackageWarning {ellipse}{The arc angle is reduced to -720..720}%
          \@whiledim \@tempsign\@arclen>720\p@ \do {\advance\@arclen-\@tempsign360\p@}%
          \@tempdima #6\p@ \advance\@tempdima \@arclen
          \edef\@angleend{\strip@pt\@tempdima}%
          \pIIe@@elliparc{#1}{#2}{#3}{#6}{\@angleend}%
        \else
          \pIIe@@elliparc{#1}{#2}{#3}{#6}{#7}%
        \fi
      \fi
    \fi
  \fi
}
```

`\pIIe@@elliparc` divides the total angle in parts of at most 90 degrees. Assumes `\@ovro`$= a$ and `\@ovri`$= b$, and `\@arclen` the arc angle, with `\@tempsign` sign of the arc angle.

```
\newcommand*\pIIe@@elliparc[5]{%
  \begingroup
  \ifdim \@tempsign\@arclen>90\p@
```

If the arc angle is too large, the arc is recursively divided into 2 parts until the arc angle is at most 90 degrees.

```
    \divide\@arclen 2%
    \@tempdima #4\p@\advance\@tempdima by \@arclen
    \edef\@anglemid{\strip@pt\@tempdima}%
    \def\@tempa{\pIIe@@elliparc{#1}{#2}{#3}{#4}}%
    \expandafter\@tempa\expandafter{\@anglemid}%
    \def\@tempa{\pIIe@@elliparc{2}{#2}{#3}}%
    \expandafter\@tempa\expandafter{\@anglemid}{#5}%
  \else
```

The arc angle is smaller than 90 degrees.

```
    \pIIe@elliparc@{#1}{#2}{#3}{#4}{#5}%
  \fi
  \endgroup
}%
```

4.5 Drawing elliptical arcs

`\elliparc`
`\pIIeelliparc`

[⟨*start*⟩]{⟨*center-x*⟩}{⟨*center-y*⟩}{⟨*radius-x*⟩}{⟨*radius-y*⟩}{⟨*start-angle*⟩}{⟨*end-angle*⟩}

The main elliptical arc drawing routine. We start with `\pIIeelliparc` to avoid conflicts with other packages.

```
\newcommand*\pIIeelliparc[7][0]{%
  \@killglue
  \pIIe@elliparc[#1]{#2\unitlength}{#3\unitlength}{#4\unitlength}{#5\unitlength}{#6}{#7}%
  \ignorespaces%
}
\ifx\undefined\elliparc\else
  \PackageWarning{ellipse}{\protect\elliparc\space is redefined}%
\fi
\let\elliparc\pIIeelliparc
```

`\earc`
`\earc*`

[⟨α_0⟩,⟨α_1⟩]{⟨*radius-x*⟩}{⟨*radius-y*⟩}

The `\earc` command generalizes the standard `\arc` with both a x- and y-radius. The `\earc*` version draws a filled elliptical arc while `\earc` only strokes the elliptical arc. Both take an optional comma separated pair of angles which specify the initial and final angle (0 and 360 by default). We start with `\pIIeearc` to avoid conflicts with otherpackages.

```
\newcommand*\pIIeearc
  {\@ifstar{\@tempswatrue\pIIe@earc@}{\@tempswafalse\pIIe@earc@}}
\newcommand*\pIIe@earc@[3][0,360]{\pIIe@earc@@(#1){#2}{#3}}
\def\pIIe@earc@@(#1,#2)#3#4{%
  \if@tempswa
    \pIIe@moveto\z@\z@
    \pIIe@elliparc{\z@}{\z@}{#3\unitlength}{#4\unitlength}{#1}{#2}%
    \pIIe@closepath\pIIe@fillGraph
  \else
    \pIIe@elliparc[1]{\z@}{\z@}{#3\unitlength}{#4\unitlength}{#1}{#2}%
    \pIIe@strokeGraph
  \fi}
\ifx\undefined\earc\else
  \PackageWarning{ellipse}{\protect\earc\space is redefined}%
\fi
\let\earc\pIIeearc
```

`\ellipse` `\ellipse*` `{`⟨*radius-x*⟩`}{`⟨*radius-y*⟩`}`

The `\ellipse` draws an ellipse with the specified x- and y-radius. The `\ellipse*` version draws a filled ellipse. We start with `\pIIeellipse` to avoid conflicts with other packages. The implementation redirects immediately to `earc` which generalized this command.

```
277 \newcommand*\pIIeellipse
278   {\@ifstar{\@tempswatrue\pIIe@earc@}{\@tempswafalse\pIIe@earc@}}
279 \let\ellipse\pIIeellipse
```

Change History

v1.0
  General: Initial version 1

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.