1 Introduction

\[\text{EM} \text{X} \] has many advanced graphics packages now, the most extensive are \texttt{tikz} and \texttt{pstricks}. However, these are also large packages that take long to load and may not always work on all drivers. The standard \texttt{pict2e} package removes many of the previous limitations of the ‘old’ \texttt{EM} \text{X} \texttt{picture} environment and makes it a \textit{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 \texttt{pict2e} primitives (i.e. the \texttt{\cbezier} command).

2 Drawing ellipses

\begin{verbatim}
\ellipse \ellipse*
\end{verbatim}

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

\begin{verbatim}
\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}
\end{verbatim}

\begin{verbatim}
\earc \earc*
\end{verbatim}

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

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

\begin{verbatim}
\elliparc[(initial)]{(center-x)}{(center-y)}{(x-rad)}{(y-rad)}{(start-angle)}{(end-angle)}
\end{verbatim}

The core elliptical arc routine. These are to be used with path commands, like \texttt{\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 (\texttt{\lineto}) which draws a line to the arc starting point, the value 1 (\texttt{\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.

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

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 \texttt{\lineto} command to the starting point of the arc). Similarly, we use such initial line segment and a \texttt{\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 \texttt{\rotatebox} command from the \texttt{graphicx} package. For example:

\begin{verbatim}
\put(3,3){%
  \rotatebox[origin=c]{45}{\ellipse{3}{2}}%
}\end{verbatim}
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:

\begin{verbatim}
Boxed numbers: \ellipbox{1}, \ellipbox{123}.

We also used this command to draw the ellipse in the title of this article, and it is defined as:
\begin{verbatim}
\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}
\end{verbatim}

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

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 \(\alpha_1\) to \(\alpha_2\) 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 \verb|\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 \(\alpha_1\) and \(\alpha_2\) 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:

\[
E(t) = (c_x + a \cdot \cos(t), c_y + b \cdot \sin(t))
\]

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 \(\alpha\) 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 \(\alpha\) on the ellipse. From the definition of \(E\) it is straightforward to derive a parametric angle \(t_i\) for some \(\alpha_i\):

\[
t_i = \arctan_2\left(\frac{\sin(\alpha_i)}{b}, \frac{\cos(\alpha_i)}{a}\right)
\]

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

\[
p_1 = E(t_1)
p_2 = E(t_2)
\]
Figure 2: The relation between the parametric angle $t_1$ and the angle $\alpha_1$ 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:

$$q_1 = p_1 + \tan\left(\frac{t_2 - t_1}{2}\right) \cdot \mathcal{E}'(t_1)$$
$$= p_2 - \tan\left(\frac{t_2 - t_1}{2}\right) \cdot \mathcal{E}'(t_2)$$

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

$$q_1 = p_1 + \kappa \cdot \mathcal{E}'(t_1)$$
$$q_2 = p_2 - \kappa \cdot \mathcal{E}'(t_2)$$

$$\kappa = \sin(t_2 - t_1) \sqrt{\frac{4 + 3\tan^2\left(\frac{t_2 - t_1}{2}\right) - 1}{3}}$$

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 \texttt{cos/sin} operations, plus a $\sqrt{}$ and 2 \texttt{arctan} operations.
3.1 Optimizing elliptic arc equations

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

\[ t_i = \arctan\left(\frac{\sin(\alpha_i)}{b}, \frac{\cos(\alpha_i)}{a}\right) \]
\[ = \arctan\left(\frac{a}{b} \tan(\alpha_i)\right) \]
\[ = \arctan(\iota_i) \quad \text{(introducing } \iota_i \text{ for } \frac{a}{b} \tan(\alpha_i)) \]

where we write \( \iota_i \) for \( \frac{a}{b} \tan(\alpha_i) \).

Now,

\[ \cos t_i = \cos(t_i) \]
\[ = \cos(\arctan(\iota_i)) \quad \text{(geometry and pythagorean theorem)} \]
\[ = \pm \frac{1}{\sqrt{1 + \iota_i^2}} \]

with

\[ \pm = \text{if } \cos(\alpha_i) < 0 \quad \text{then } - \quad \text{else } + \]

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

\[ \sin t_i = \sin(t_i) \]
\[ = \sin(\arctan(\frac{a}{b} \tan(\alpha_i))) \]
\[ = \sin(\arctan(\iota_i)) \]
\[ = \pm \frac{\iota_i}{\sqrt{1 + \iota_i^2}} \]

Note that the interaction between the \( \sin \) and \( \iota_i \) term (whose sign is determined by \( \tan(\alpha_i) \)) allows us to reuse the sign function used for \( \cos t_i \).

Using the previous equalities we can restate the parametric equations in terms of \( \sin t_i \) and \( \cos t_i \):

\[ E_i = (c_x + a \cdot \cos t_i, c_y + b \cdot \sin t_i) \]
\[ E'_i = (-a \cdot \sin t_i, b \cdot \cos t_i) \]

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:

\[ \sin(t_2 - t_1) = \sin t_2 \cos t_1 - \cos t_2 \sin t_1 \quad \text{([1, page 72, 4.3.16])} \]
\[ \cos(t_2 - t_1) = \cos t_2 \cos t_1 + \sin t_2 \sin t_1 \quad \text{([1, page 72, 4.3.17])} \]
3.2 Circular square roots

Now, we only need two \( \tan \) operations to calculate the initial \( \iota_1 \) and \( \iota_2 \) terms but we still have three square roots: \( \sqrt{1 + \iota_1^2} \) and \( \sqrt{4 + 3\tan^2(\frac{t_2-t_1}{2})} \). 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:

\[
\begin{align*}
\text{csqrt}(x, y) &= \text{let } sqr = x^2 + y^2 \\
&= x_0 = \max(|x|, |y|, \frac{1}{\sqrt{2}}|x + y|) \\
x_1 &= (x_0 + \frac{sqr}{x_0})/2 \\
x_2 &= (x_1 + \frac{sqr}{x_1})/2
\end{align*}
\]

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 \( \alpha_1 \) and \( \alpha_2 \). 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:

\[
\begin{align*}
p_1 &= E_1 \\
p_2 &= E_2 \\
q_1 &= p_1 + \kappa \cdot E'_1 \\
q_2 &= p_2 - \kappa \cdot E'_2 \\
E_1 &= (c_x + a \cdot \cos t_1, c_y + b \cdot \sin t_1) \\
E'_1 &= (-a \cdot \sin t_1, b \cdot \cos t_1)
\end{align*}
\]
The $\cos_i$ and $\sin_i$ are calculated as:

$$\sin_i = \pm \frac{\iota_i}{\rho_i} \quad \cos_i = \pm \frac{1}{\rho_i}$$

with

$$\iota_i = \frac{a}{b} \tan(\alpha_i) \quad \rho_i = \text{csqrt}(1, \iota_i) \ (\approx \sqrt{1 + \iota_i^2})$$

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

And finally, the $\kappa$ term can be defined as:

$$\kappa = \sin_{t_21} \frac{\kappa_{\text{sqrt}} - 1}{3}$$

with

$$\sin_{t_21} = \sin(t_2 - t_1) \quad \cos_{t_21} = \cos(t_2 - t_1)$$

$$\kappa_{\text{tan}} = \frac{1 - \cos_{t_21}}{\sin_{t_21}} \quad (\text{note: divides by zero if } \alpha_1 = \alpha_2)$$

$$\kappa_{\text{sqrt}} = \text{csqrt}(\sqrt{4}, \sqrt{3} \cdot \kappa_{\text{tan}}) \ (\approx \sqrt{4 + 3\kappa_{\text{tan}}^2})$$

### 4 Implementation

Generally, we use e-T\TeX{} division to divide dimensions, where we divide $\langle \text{dim}_1 \rangle$ by $\langle \text{dim}_2 \rangle$ using: \texttt{\textbackslash dimexpr 1pt * (dim1)/(dim2)\textbackslash relax} since it keeps a 64-bit intermediate result for such ‘scaling’ expressions. Note that both $\langle \text{dim} \rangle$ expressions occur in an integer context and T\TeX{} will convert them to numbers automatically (i.e. in sp units).

#### 4.1 Generic math and trigonometry routines

\begin{verbatim}
\pIIe@csedef \{\csname\}\pattern\{\}\endcsedef
\pIIe@ellip@csqrt \{\dimen\} \{\dimen\} \{\dimen\} \{\dimen\} \{\dimen\}
\end{verbatim}

\begin{verbatim}
\pIIe@ellip@csqrt \{\dimen\} \{\dimen\} \{\dimen\} \{\dimen\} \{\dimen\}
\end{verbatim}
\newcommand*{\pIIe@ellip@csqrt@}{\pIIe@ellip@csqrt@}%

Internal routine: calculates $\text{dimen0} \approx \sqrt{x^2 + y^2}$, where $x \geq 0$ and $y \geq 0$, and $\@ovxx = x$ and $\@ovyy = y$.

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

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

First determine $\max(x, y, \frac{1}{\sqrt{2}}(x + y))$ in $\text{dimen0}$. Put the sum $x + y$ in $\@ovdx$.

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

Put initial guess in $\text{dimen0} = \max(|x|, |y|, \frac{1}{\sqrt{2}}(x + y))$.

\text{dimen0}.7071067\@ovdx
\ifdim\text{dimen0} < \@ovdy\text{dimen0} = \@ovyy\fi
\ifdim\text{dimen0} < \@ovxx\text{dimen0} = \@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\text{dimen0} by \dimexpr1pt * \@ovxx/\text{dimen0}\relax
\divide\text{dimen0} by 2\%
\advance\text{dimen0} by \dimexpr1pt * \@ovxx/\text{dimen0}\relax
\divide\text{dimen0} by 2\%
\fi

Result is $\text{dimen0}$.

\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 $x$. Set \@dimtmpb to $|x|$. \@tempdima\dimen@ \@tempdimb\@tempdima \ifdim\@tempdimb<\z@ \@tempdimb-\@tempdimb\fi \advance\dimen@ 0.0663\@tempdimb \advance\@tempdimb -1pt\relax \edef\@tempa{\strip@pt\@tempdimb}% \dimen@\@tempa\dimen@ \edef\@tempa{\strip@pt\@tempdima}% \dimen@\@tempa\dimen@ \dimen@-\dimen@ \add@{\frac{\pi}{4}} \approx 0.7853 \times x. \dimen@ 0.7853\@tempdima}
Save angle adjustment term in \@tempdimd.

\else\dimen@0\p@
\fi
\fi
\else
\@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 \( \arctan(x) = \pm \frac{1}{2} \pi - \arctan\left(\frac{1}{x}\right) \) to stay within the valid domain of \pIIe@atan\@. The sign \( \pm \) 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
\else
\#3\dimen@%
\}

4.2 Sub routines for drawing an elliptical arc

\pIIe@noneto \{(dimen_y)\}\{(dimen_x)\}
Ignores its arguments. Used as a no-op instead of \pIIe@lineto or \pIIe@moveto.
\newcommand*{\pIIe@noneto}[2]{}
\pIIe@ellip@sincost\{(\( \alpha_i \))\}\{(i = one or two)\}
Calculate \( \sin \alpha_i \) and \( \cos \alpha_i \) into the \@ellip(sin/cos)\@. Assumes \@ellipratio = \( \frac{a}{b} \).
\newcommand*{\pIIe@ellip@sincost}[2]{%}
Put the \( \sin(\alpha_i) \) and \( \cos(\alpha_i) \) into \@tempdimd and \@tempdimb.
\CalculateSin{\#1}
Check for extremes where $\tan\theta = \pm\infty$.
\begin{verbatim}
\ifdim\@tempdima=p0\relax
  \pIIe@csef{\ellipsin#2}{1}\%
  \pIIe@csef{\ellipcos#2}{0}\%
\else\ifdim\@tempdima=-p0\relax
  \pIIe@csef{\ellipsin#2}{-1}\%
  \pIIe@csef{\ellipcos#2}{0}\%
\else
  Calculate $\iota$ in $\@tempdimc$ and $\sqrt{1 + \iota^2}$ in $\@tempdimd$, and derive $\sin\iota$ and $\cos\iota$.
  \@tempdimc\@ellipratio\dimexpr1pt * \@tempdima/\@tempdimb\relax
  \ifnum\@tempdimb<0\relax
    \@tempdimd-\@tempdimd\fi
  \pIIe@csef{\ellipsin#2}{\strip@pt\dimexpr1pt * \@tempdimc/\@tempdimd\relax}
  \pIIe@csef{\ellipcos#2}{\strip@pt\dimexpr1pt * p0/\@tempdimd\relax}
\fi
\end{verbatim}

\pIIe@ellip@sincost\{\alpha_1\}\{\alpha_2\}
\begin{verbatim}
\pIIe@ellip@sincost\{\alpha_1\}\{\alpha_2\}
\end{verbatim}

\pIIe@ellip@sincost\{\alpha_1\}\{\alpha_2\}
\begin{verbatim}
\pIIe@ellip@sincost\{\alpha_1\}\{\alpha_2\}
\end{verbatim}

\pIIe@omega\{\iota = \text{one or two}\}
\begin{verbatim}
\pIIe@omega\{\iota = \text{one or two}\}
\end{verbatim}

\pIIe@omegain\{\iota = \text{one or two}\}
\begin{verbatim}
\pIIe@omegain\{\iota = \text{one or two}\}
\end{verbatim}
\newcommand*{pIIe@omegai}{% 
\@tempdimc\csname @ellipsin#1\endcsname\@ovro 
\@tempdimc-\@tempdimc
\@tempdimd\csname @ellipcos#1\endcsname\@ovri
\}

\pIIe@ellip@kappa
Calculates $\kappa$, expects $\ellip\left(\sin;\cos\right)\left(\text{one};\text{two}\right)$ to be defined.

\newcommand*{pIIe@ellip@kappa}{% 
Calculate $\sin_{21}$ and $\cos_{21}$ 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 \@ellipcostwo\@ovyy 

First test if $\sin_{21} = 0$ to prevent division by zero. In that case, it must have been 
that $\alpha_1 = \alpha_2$ and we set $\kappa$ to zero so it the control points become equal to the 
start and end point.
\ifdim\@tempdima=\z@\relax
\edef\@ellipkappa{0}\
\else
Calculate $\kappa_{\tan}$ in $\dimen@$
\dimen@\dimexpr1pt - \@tempdimb\relax 
\dimen@\dimexpr1pt \dimen@/\@tempdima\relax
Calculate $\kappa_{\sqrt}$ in $\dimen@$
\pIIe@ellip@csqrt{2\p@}{1.73205\dimen@}{\dimen@}\
Calculate $\kappa$ in $\dimen@$
\advance\dimen@ by \p@ 
\divide\dimen@ by 3\% 
\edef\@tempa{\strip@pt\@tempdima}\
\dimen@\@tempa\dimen@ 
\edef\@ellipkappa{\strip@pt\dimen@}\
\fi
%	ypeout{ calculated kappa: \@ellipkappa}\%
}

4.3 Core routines for drawing elliptical arcs

\pIIe@elliparc% 
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 ($\@ovro$, $\@ovri$),angle (#4, #5)}%
Define initial action: 0 (lineto), 1 (moveto), or 2 (nothing)
\begin{verbatim}
  \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 $\sin t_i$ and $\cos t_i$ 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}}%
  \fi
\end{verbatim}
\pIIe@elliparc@t
\[\langle c_x \rangle\{\langle c_y \rangle\{\langle t_1 \rangle\{\langle t_2 \rangle\}}\]
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 $\alpha_i$). This routine is mainly for other libraries that may already have computed the $t$ angles and need a bit more efficiency.

\begin{verbatim}
  \newcommand*{\@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
    \ifdim\@ovro=\z@\relax\@ovri\z@\fi
    \ifdim\@ovri=\z@\relax
      \@ellip@startto{#2}{#3}%
    \else
      Calculate $\sin t_i$ and $\cos t_i$ 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}}%
    \fi
  \end{verbatim}
And draw..
\pIIe@elliparc@draw{#2}{#3}\i
}\}
pIIe@elliparc@draw\{(c_{x})\} \{(c_{y})\}

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]{%
\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
\@ellip@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\@tempdima by \@tempdima
\advance\@tempdimb by \@tempdimb
\@ellip@add@nums\@tempdima\@tempdimb
\typeout{ control two: (\the\@tempdima,\the\@tempdimb)}%

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

These two macros check the arguments and normalize the angles.

\newcommand*{\pIIe@elliparc}[7][]{\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 $|\alpha_2 - \alpha_1| \leq 720$. 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}\
\whileilong\@tempsign\@arclen>720\p@ \do {\advance\@arclen -\@tempsign360\p@}
\@tempdima #6\p@
\edef\@angleend{\strip@pt\@tempdima}\
\pIIe@@elliparc[\#1]{\#2}{\#3}{\#6}{\@angleend}\
\else
\pIIe@@elliparc[\#1]{\#2}{\#3}{\#6}{\#7}\
\fi
\fi
\fi
\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]{%}
\begin{group}
\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%
\@tempsign #4\p@\advance\@tempsign by \@arclen
\edef\@anglemid{\strip@pt\@tempsign}\
\def\@tempa{\pIIe@elliparc[\#1]{\#2}{\#3}{\#4}}\
\expandafter\@tempa\expandafter{\@anglemid}\
\def\@tempa{\pIIe@elliparc[2]{\#2}{\#3}}\}
4.5 Drawing elliptical arcs

\elliparc [(start)]{(center-x)}{(center-y)}{(radius-x)}{(radius-y)}{(start-angle)}{(end-angle)}\]

\pIIeelliparc

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 [(\alpha_0), (\alpha_1)]{(radius-x)}{(radius-y)}\]

\pIIeearc

\earc*

The \earc* command generalizes the standard \arc with both a \textit{x}- and \textit{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 other packages.

\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
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}{\@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.

Symbols

\@angleend ................. 234, 235 \arctan ..................... 58
\@anglemid ............... 248, 250, 252
\@ellip@startto .......... 159–161, 167, 175–177, 183, 200
\@ellipcosone ............ 121, 136, 187
\@ellipcostwo .......... 121, 137, 140, 190
\@ellipkappa ............. 143, 152, 154, 202, 203, 208, 209
\@ellipratio ............ 108, 118
\@ellipsinone .......... 121, 135, 186
\@ellipsintwo .......... 121, 139, 141, 189
\@ifstar ............... 268, 284
\@killglue ............... 259
\@ovri .................. 117, 118, 126, 132, 157, 165, 166, 181, 182, 222, 223, 227
\@ovro ................. 117, 118, 124, 130, 157, 165, 181, 221, 223, 226
\@tempdim ................ 59
\@tempsign ............. 82, 85, 229, 230, 232, 245
\@whiledim ............. 232 \pIIe@elliparc ............. 283