A graphical ellipse envelope construction with GNU 3DLDF

Laurence Finston

Abstract

This article demonstrates the use of GNU 3DLDF for a graphical solution of the problem of constructing the envelope of an ellipse and an approximation to the curve itself.

Introduction

Before the universal availability of computers and graphics software, technical drawings had to be made by hand by draftsmen and -women, who were skilled professionals. Making a technical drawing of even moderate complexity was time-consuming, painstaking and error-prone work, requiring much knowledge and patience and the ability to endure frustration. Erasures were difficult and a single error of conception or execution could render useless the work of many hours.

Well into the 20th century, graphical methods for constructing curves were of importance for the creation of technical drawings. In addition, such constructions were of central importance in the mathematics of the ancient Greeks, especially those constructions that only made use of a straight edge and dividers, with the added restriction that the use of the dividers for measuring lengths was forbidden.

Even today, with computers and 3D graphic software, graphical methods of constructing geometric figures retain their fascination and continue to provide insight and diversion to those who appreciate elegant and ingenious solutions to mathematical puzzles. In fact, the use of computers greatly increases the pleasure of creating technical drawings, due to their speed, accuracy and the ease of making corrections.

This article demonstrates the use of GNU 3DLDF for a graphical solution of the problem of constructing an ellipse found in Lockwood’s A Book of Curves, p. 13 [2]. GNU 3DLDF is a package for three-dimensional drawing with METAPOST and METAFONT output. It implements a language based on the METAFONT language with many additional data types and operations. More information can be found on the GNU 3DLDF website: https://www.gnu.org/software/3dldf/LDF.html

The following figures are two-dimensional, so it would have been possible to create them using METAPOST alone. They do use several features of 3DLD

that are not present in METAPOST, but it could easily be adapted. For example, rotation about the z-axis could be replaced by calls to reflectedabout.

Constructing an ellipse envelope

Draw a circle c with center C (fig. 1). \( \overline{AA'} \) is a diameter of c and S a point on \( \overline{AA'} \). The distance \( CS \) should be \( \geq \frac{3}{5}CA \). \( Q_5 \), \( Q_{10} \) and \( Q_{20} \) are points on the perimeter of \( c \) such that \( \angle ASC_5 = 25^\circ \), \( \angle A'SQ_{10} = 50^\circ \) and \( \angle A'SQ_{20} = 100^\circ \). \( R_5 \), \( R_{10} \) and \( R_{20} \) are points on the perimeter of \( c \) such that \( \angle SQ_5R_5 = \angle SQ_{10}R_{10} = \angle SQ_{20}R_{20} = 90^\circ \). \( S' \) is \( S \) rotated around \( C \) by 180°. That is, if \( x_S = -k \), \( x_S' = k \). \( S \) and \( S' \) are the foci of the ellipse.

With fixed \( S \), as points \( Q_x \), \( R_x \) for \( 1 \leq x \leq N - 1 \), \( N = 72 \) are added, whereby \( Q_x \) and \( R_x \) are on the same side of \( \overline{AA'} \), their intersections will form an envelope describing an ellipse \( e \) with foci \( S \) and \( S' \) and major axis \( \overline{AA'} \).

As points \( Q_x \) and \( R_x \) are added, it becomes clear that the intersections of the lines \( Q_xR_x \) quickly form an envelope revealing the outline of the ellipse. (see fig. 2).

If this figure were to be drawn by hand, two lines would have to be drawn for each of the \( Q_xR_x \) pairs and the right angles \( \angle SQ_xR_x \) would have to be obtained. Placing a set square accurately so that \( Q_x \) and \( R_x \) were both squarely on the perimeter thirty times would be quite a challenge.

A graphical ellipse envelope construction with GNU 3DLDF

doi.org/10.47397/tb/43-3/tb135finston-ellipse
Assuming this feat was accomplished, the next step would be to trace the curve of the ellipse. This could be done with a flexible curve or a French curve. I personally have never had good results with either of these tools, especially where the curvature was small.

On the other hand, when tracing a curve approximating the ellipse using points on the envelope using a computer, it doesn’t suffice to intuitively recognize the rough shape of the ellipse. While it is not necessary to find all of the intersection points that are closest to the ellipse, it is necessary to find a sufficient number of them to trace a good approximation to the latter and to ensure that all of the points chosen are as close as possible to the ellipse.

Figure 2 shows the intersection points \( p_{29} = Q_{29}R_{29} \cap Q_{28}R_{28} \) and \( p_{30} = Q_{30}R_{30} \cap Q_{29}R_{29} \). The intersection points used are thus \( p_x = Q_xR_x \cap Q_{x-1}R_{x-1} \) for \( x > 1 \).

In figure 2, \( p_{29} \) lies in the first quadrant of the ellipse, while \( p_{30} \) lies in the second. It is not strictly speaking necessary to find the intersection points \( p_x \) for \( x > 29 \), since the intersection points in the first quadrant may simply be rotated about the x- and y-axes in order to obtain points close to the ellipse in the remaining three quadrants of \( e \). Unless by chance, they will not, however, be the same points that would be found by continuing to find the intersection points of the lines \( Q_xR_x \).

By hand, this would be no less work than finding \( p_1 \ldots p_{29} \) in the first place, but with the computer it is the work of a moment.

\( Q_{35} \) is already very close to \( A \) and \( Q_{35}R_{35} \) is not too far from being vertical. \( Q_{36} \), in fact, coincides with \( A \), \( Q_{36}R_{36} \) is vertical and the intersection point \( p_{36} = Q_{36}R_{36} \cap Q_{35}R_{35} \) doesn’t exist.

\( S' \) is the reflection of \( S \) about the y-axis. \( S \) and \( S' \) are the foci of the ellipse. The major axis is \( \overline{AA'} \) but the minor axis is not so easy to determine. It is twice the distance from \( C \) to a point \( w \) on the ellipse on the line through \( C \) perpendicular to \( \overline{AA'} \), to the left of \( p_{29} \) and to the right of \( p_{30} \), which are both close to the ellipse, but not actually on it. I will have

Laurence Finston
to give some thought to how to determine \( w \) without “cheating”.

To execute this drawing in pen-and-ink would be a nightmare.

\[ Q_{a+1} R_{a+1} \] are closer to \( C \) than its intersection points with other lines \( Q_x R_x \) and would hence produce the closest approximation to an ellipse. However, I would have to think about whether I could prove this with my limited mathematical skills.

As examples of the positions of other intersection points, \( g_{43} \) is the intersection point \( Q_{10} R_{10} \cap Q_{25} R_{25} \) and lies close to the perimeter of the ellipse and \( g_{44} \) is the intersection point \( Q_5 R_5 \cap Q_{30} R_{30} \) and lies outside \( c \).

\( Q_3 R_3 \) intersects with \( Q_{40} R_{40} \) and is closer to \( S \) than its intersection points with other lines and would hence produce the closest approximation to an ellipse. However, I would have to think about whether I could prove this with my limited mathematical skills.

It would nevertheless be somewhat unsatisfying to have such different arc lengths depending on the position on the circle of the points used for generating the ellipse and using points that were only close to the ellipse inside of actually on it for the path. Instead, 3DLDF uses the parametric equation for an ellipse to generate the path for objects of type ellipse, i.e.,

\[
(x, y) = (a \cos(t), b \sin(t)) \text{ for } 0 \leq t \leq 2\pi.
\]

Using the intersection points \( p_x = Q_x R_x \cap Q_{x-1} R_{x-1} \) appears to produce nearly correct results. It would seem that the intersection point of a line \( Q_x R_x \) with its adjacent lines \( Q_{x-1} R_{x-1} \) and \( Q_{x+1} R_{x+1} \) are closer to \( C \) than its intersection points with other lines \( Q_x R_x \) and would hence produce the closest approximation to an ellipse. However, I would have to think about whether I could prove this with my limited mathematical skills.

As examples of the positions of other intersection points, \( g_{43} \) is the intersection point \( Q_{10} R_{10} \cap Q_{25} R_{25} \) and lies close to the perimeter of the ellipse and \( g_{44} \) is the intersection point \( Q_5 R_5 \cap Q_{30} R_{30} \) and lies outside \( c \).

\( Q_3 R_3 \) intersects with \( Q_{40} R_{40} \) and is closer to \( S \) than its intersection points with other lines and would hence produce the closest approximation to an ellipse. However, I would have to think about whether I could prove this with my limited mathematical skills.

It would nevertheless be somewhat unsatisfying to have such different arc lengths depending on the position on the circle of the points used for generating the ellipse and using points that were only close to the ellipse inside of actually on it for the path. Instead, 3DLDF uses the parametric equation for an ellipse to generate the path for objects of type ellipse, i.e.,

\[
(x, y) = (a \cos(t), b \sin(t)) \text{ for } 0 \leq t \leq 2\pi.
\]

Using the intersection points \( p_x = Q_x R_x \cap Q_{x-1} R_{x-1} \) appears to produce nearly correct results. It would seem that the intersection point of a line \( Q_x R_x \) with its adjacent lines \( Q_{x-1} R_{x-1} \) and \( Q_{x+1} R_{x+1} \) are closer to \( C \) than its intersection points with other lines \( Q_x R_x \) and would hence produce the closest approximation to an ellipse. However, I would have to think about whether I could prove this with my limited mathematical skills.

As examples of the positions of other intersection points, \( g_{43} \) is the intersection point \( Q_{10} R_{10} \cap Q_{25} R_{25} \) and lies close to the perimeter of the ellipse and \( g_{44} \) is the intersection point \( Q_5 R_5 \cap Q_{30} R_{30} \) and lies outside \( c \).
Figure 6 shows the quarter ellipse $q_0$ containing points $A$ and the intersection points $p_1 \ldots p_{29}$. This figure also shows that $Q_{29}R_{29}$ and $Q_{30}R_{30}$ intersect at $p_{30}$.

Figure 7 shows the completed approximation to an ellipse $e_a$ consisting of the combination of the paths $q_0$, $q_1'$ and $q_3'$ where $q_1'$ is the reflection of $q_0$ about the y-axis with the order of the points reversed and $q_3'$ is $q_0q_1'$ reflected about the x-axis and with the order of the points reversed.

The annotated GNU 3DLDF code

The following listings contain only the parts of the 3DLDF program for the figures in this article that are of particular interest. Labels, “bookkeeping chores” and other items have been left out in the interest of comprehensibility. The full code may be found here: https://www.gnu.org/software/3dldf/ellipses.html#Constructions

Let’s start with some basic declarations:

```plaintext
point p[];
point R[];
point Q[];
point a[];
point d[];
path q[];
circle c[];
numeric n[];
boolean b[];
bool_point_vector bpv;
picture v[];
```

Everything here is just the same as it would be in METAFONT except for point p[] and the other point array declarations, circle c[] and bool_point_vector bpv. point is the 3D equivalent of pair and circle is a type in 3DLDF similar to a path, except its radius is stored as part of the object and there are special operations that apply to circles that don’t apply to paths, such as get_center and the predicate is_circular.

Of course, as mentioned above, the figures in this article do not require any 3D calculations and could
as easily have been created using METAPOST with a few changes. Nonetheless, in 3DLDF, points in space, whether two-dimensional or three-dimensional, are represented by objects of type point and the type pair doesn’t exist.

bool_point is a type in 3DLDF that combines a boolean and a point in a single object. bool_point objects may be returned as the result of operations, such as intersection_point, whereby the boolean part indicates whether a particular condition is true or false, e.g., whether the point lies on one or both of the paths.

bool_point_vector is a 3DLDF type containing multiple bool_point objects. It is a so-called “vector-type”. The latter are similar to arrays, e.g., bool_point[], except that a vector-type object may be returned as the result of an operation or operated upon as a single object, whereas these things aren’t possible for arrays.

\[ c_0 := \text{unit\_circle scaled (3cm, 0, 3cm)} \]
rotated (90, 0);
draw c0;
point C;
C := \text{origin};
point A;
A := \text{get\_point 16 c0};
point Aprime;
Aprime := \text{get\_point 0 c0};
draw A -- Aprime;
point S;
S := mediate(A, C, .2);

unit\_circle is a predefined constant of type circle. Unlike METAFONT, where the “canonical” unit is the pixel and METAPOST, where it is the PostScript point or big point (1bp = 1/72in), in 3DLDF, the canonical unit is the centimeter and thus unit\_circle has radius (not diameter!) 1cm. And unlike fullcircle, which has 8 points in METAFONT, unit\_circle has 32, thus point 16 of \( c_0 \) is at the halfway point around the circle.

32 points is normally enough to prevent a circle from “going out of shape” when it is projected with a moderate amount of foreshortening. See “An Introduction to GNU 3DLDF” in this issue.

There are three other differences with respect to METAFONT in this section of the code:

- All of the assignments are actual assignments using := rather than equations using =. Unfortunately, 3DLDF doesn’t (yet?) share METAFONT’s “amazing ability to deduce explicit values from implicit statements” [1, p. 83].

- get\_point is the equivalent in 3DLDF of point in METAFONT, as in (pair primary) \(\rightarrow\) point (numeric expression) of (path primary) [1, p. 73].

In 3DLDF, as previously mentioned, the symbol point is a “declarator” used to declare point objects and its use as an operator would have caused conflicts in the grammar generated by the parser generator GNU Bison.

- Nor did Bison allow METAFONT’s syntax for the mediation operation, e.g., .5[p0, p1] because it would have conflicted with the other uses of brackets. Therefore, the operator mediate must be used instead.

In 3DLDF, as in METAFONT, \( A' \) would have been a valid name for a variable. However, I generally don’t use such variable names and I thought it would be potentially confusing, so I used Aprime instead.

\[ \text{numeric j; j := 0;} \]

As in METAFONT, \( j \) could just have been used without explicitly declaring it as a numeric. Doing so is considered good programming style, although it may be overkill here and I have some doubts about whether every member of the “programming style police” actually has practical experience writing computer programs.

\[ \text{numeric N, k;} \]
N := 72;
K := 360/N;
for i = 0 upto 100:
Q[i] := Aprime rotated (0, 0, 0 + (i * K));
d[j] := Q[i] shifted (0, 0, 1);
d[j+1] := S rotatedaround (Q[i], d[j]) 90;
bpv := (Q[i] -- d[j+1])
intersection_points c0;
a0 := bpv0;
a1 := bpv1;
if xpart a0 > xpart a1:
a2 := a0;
else:
a2 := a1;
fi
R[i] := a2;
j += 2;
endfor;

The loop in this section of the code is where the real action of the program begins. \( Q_x \) is found by rotating \( A' \) about \( C \), the perpendicular to \( SQ_x \) through \( Q_x \) is found and \( R_x \) is found as the intersection point of the perpendicular with \( c_0 \) with the greatest x-coordinate. A circle and a line, that is, a path with two points and a simple connector, that is, one without any modifiers that would cause it to diverge from a straight line, can have 0 to 2 intersections.
Please note that this loop finds the points \( Q_x \) in a different way than in the description in section “Constructing an ellipse envelope” on page 333. However, it is completely arbitrary how these points are found.

Again, in most ways, the 3DLDF code would mostly be valid in METAFONT, but there are several important differences:

In METAFONT, rotation is about a 2D point, either the origin, with plain \texttt{rotated}, or about an arbitrary point with \texttt{rotatedaround}. In 3DLDF, rotation is about the \( x \), \( y \), and \( z \)-axes with \texttt{rotated} and about an axis specified as two \texttt{points} with the operator \texttt{rotated\_around}.

METAFONT provides the primitive operation \texttt{intersectiontimes} and a related macro named \texttt{intersectionpoint}, both of which return a single \texttt{pair} as their result. Therefore, if two \texttt{paths} intersect more than once, information about only one of the intersections is returned.

For 3DLDF, where geometric figures play a much greater role than in METAFONT, this behavior would not be acceptable, so \texttt{bool\_point\_vectors} are used as the type of the return values for the various operations that return intersection points or times.

The \texttt{a0 := \texttt{bpv0}} and \texttt{a1 := \texttt{bpv1}} statements show that \texttt{bool\_points} can be assigned to \texttt{points}, whereby the \texttt{boolean} component of the \texttt{bool\_point} is discarded. For many but not all operations in 3DLDF, \texttt{bool\_points} may be used in place of \texttt{points}.

It is worth noting that the intersection points of lines with each other and with lines with circles in this program are not found as in METAFONT. There, all \texttt{paths} are implemented as Bézier curves and intersection times and points are found with a routine that applies to all Bézier curves, irrespective of shape.

In 3DLDF, the intersections of lines with each other and with other geometric figures, such as circles, are used by combining and solving the implicit equations of the figures. However, since there are no restrictions on the transformations that can be applied to objects in 3DLDF, they must be tested to ensure that the equations still apply. For this purpose, 3DLDF implements the \texttt{predicate} operations \texttt{is\_linear}, \texttt{is\_circular}, \texttt{is\_elliptical}, etc.

Here is the next portion of code we’ll consider:

\begin{verbatim}
for i = 1 upto \( N - 1 \):
  if i <> 36:
    p[i] := \( (Q[i] -- R[i]) \)
    intersection_point \( (Q[i-1] -- R[i-1]) \);
    b[i] := p[i] rotated \( (0, 180) \);
    if i < 30:
      q0 += p[i];
      q1 += b[i];
    fi
  else:
    message "Skipping p36."
  fi
endfor;
\end{verbatim}

A second loop finds the \( N - 2 = 70 \) intersection points \( p_x = Q_x R_x \cap Q_{x-1} R_{x-1} \) for \( 1 \leq x < N \), \( x \neq 36 \). \( Q_{36} R_{36} \) is skipped, because \( Q_{36} \) coincides with \( A \) and thus \( Q_{36} R_{36} \cap Q_{35} R_{35} \) doesn’t exist.

The intersection points are appended to \( q_0 \). In addition, they are rotated \( 180^\circ \) about the \( y \)-axis and appended to \( q_1 \). Since ellipses are symmetrical about their major and minor axes, I only have to find the intersection points in the upper right quadrant of \( c_0 \) and can rotate them into the other quadrants instead of finding the intersection points in the latter, although that would certainly be possible with the construction described by Lockwood, whereby the points would be different.

Unlike the convention in \TeX{} and METAFONT, where the names of macros, variables, etc., are run together, I favor the use of the underline character in variable names. However, \texttt{currentpicture} may be used as a synonym for \texttt{current\_picture}, \texttt{rotated\_around} for \texttt{rotated\_around}, \texttt{with\_pen} for \texttt{with\_pen} and similarly for many other names of operators and predefined variables and constants.

Thus far, I have left out most of the drawing and labelling commands. However, the following are of special interest:

\begin{verbatim}
undrawdot Sprime with\_pen pensquare
  scaled (.65cm, .65cm, .65cm);

dotlabelbot("$S$", Sprime);
undrawdot S with\_pen pencircle
  scaled (.25cm, .25cm, .25cm);
drawdot S with\_pen dot\_pen;
\end{verbatim}

Laurence F CString
About $S$ and $S'$: \texttt{undrawdot} uses a large circular or square pen, respectively, to clear out a space so that the labels may be seen and, the case of $S$, to avoid a large splotch of black ink. In addition, a selection of lines is drawn over the white space to show that the lines $\overrightarrow{Q_xR_x}$ converge at $S$. 

\begin{verbatim}
q2 := q0 .. reversed q1;
q3 := q2 rotated (180, 0);
q4 := q2 .. reversed q3;
q4 += cycle;
drawarrow q0 with_color red
    with_pen medium_pen;
draw q0 .. reversed q1 .. reversed q3 .. Aprime .. cycle
    with_pen pencircle
    scaled (2.5mm, 2.5mm, 2.5mm);
undraw q0 .. reversed q1 .. reversed q3 .. Aprime .. cycle
    with_pen pencircle
    scaled (1.5mm, 1.5mm, 1.5mm);
drawarrow q0 with_color red;
drawarrow reversed q1 with_color dark_green;
drawarrow reversed q3 with_color blue;
\end{verbatim}

This is the code that draws the constructed approximation to an ellipse $e_a$. I’ve included the drawing commands here because, together with the erasures above, this is a good example of technical drawing tasks that are trivial with the computer but would be difficult to execute by hand and likely to cause much wailing and gnashing of teeth.

To create the black outlines of $e_a$, it is first drawn using a large \texttt{pencircle} of 2.5mm diameter. To compare, in technical drawings, 0.5mm is the size used most. For example, acrylic templates for drawing shapes are designed for use with technical pens with 0.5mm nibs. (Other commonly available sizes are 0.25mm, 0.7mm and 1mm.) In this article, the “standard” pen size is 0.333mm. Then, the middle of the curve is cleared out by undrawing it with a \texttt{pencircle} of diameter 1.5mm. Finally, the paths $q_0$, $q_1'$ and $q_3'$, whereby the latter two are simply $q_1$ and $q_3$ reversed, are drawn in color and with arrows using a \texttt{pencircle} of diameter 0.5mm.

In \texttt{METAFONT}, \texttt{pencircle} would be scaled using a single numeric value while \texttt{pensquare} would be scaled using two. In 3DLDF, a drawing command copies an object such as a \texttt{path}, associates the copy with any items such as pens or colors that are specified in the command and puts them all together onto a \texttt{picture}, \texttt{current_picture} by default. The pens are only used when \texttt{endfig} or \texttt{output} causes \texttt{METAFONT} or \texttt{METAPOST} code to be written to an output file. They are therefore purely 2D objects. While they may be scaled using three numerical values, in fact only the x- and y-coordinates are used and the z-coordinate is ignored, even when the object is projected using the parallel projection onto the x-z plane.

Since this may change in the future, it is safest to specify all three dimensions when scaling a pen.

\textbf{Acknowledgements.} Many thanks to Denis Roegel and Boguslaw Jackowski for improving this article with their corrections and suggestions.

\textbf{References}


Ł Laurence Finston
Germany
Laurence.Finston@gmx.de

A graphical ellipse envelope construction with GNU 3DLDF