LaTeX, pict2e and complex numbers

Tug 2006

The 27th Annual meeting & International Conference of the TeX Users group

Claudio Beccari

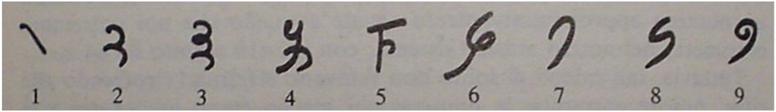
TUG & GuIT

Marrakesh, 9–11 november 2006

Summary

A short historical perspective

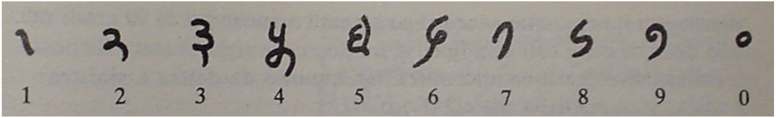
In the beginning there were the Indians who, by the end of the VII century, had developed a system of digits and had conceived the idea of positional numbering but did not yet use the "zero"[1]



[1] All images in these initial slides are taken from the book by Georges Ifrah, *Les chifres ou l'histoire d'une grande invention.*

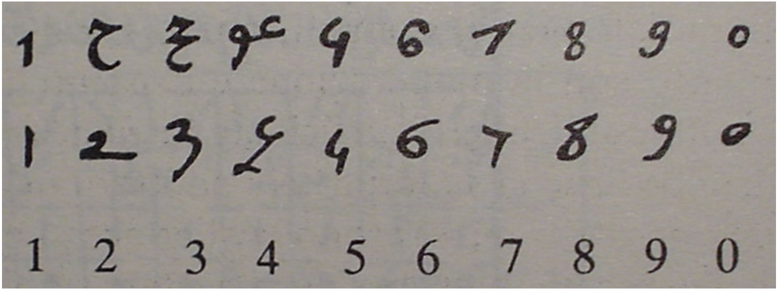
A short historical perspective

The Indian digits were picked up by the Arabs and through the latter they were brought to the rest of the Mediterranean world. In the IX century the Arab digits were still almost identical to the Indian ones, but the zero was finally there.



A short historical perspective

In the following centuries two different writing systems developed in the Eastern Arab territories compared to the Western Arab ones; in the Maghreb area the following digits were used and passed on into the Hiberic peninsula from where they spread out in the rest of the Western European countries.

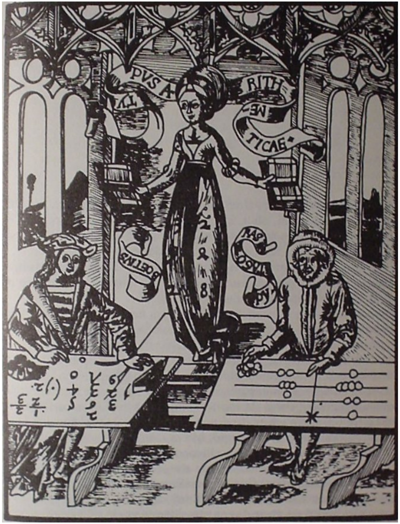


A short historical perspective

In the Hiberic peninsula and in the rest of Europe the digits evolved into what we use today; in the XVI century the situation was almost stabilized, but it was not the end of it.

A short historical perspective

The book *Margherita Philosophica* by Gregorius Reish, published in Fribourg in 1503, still had a frontispiece with an allegory of *Arithmetica* who is overlooking a couple of scholars, one (apparently Boetius) making calculations with the new Arabic numerals, and the other (apparently Pytagoras) making calculations with the abacus.



A short historical perspective

Fractional decimal numbers were also introduced in the XVI century;

- in 1582 Simon Stévin started using fractional decimal numbers without reference to fractions, but using “decimals”
- in 1592 Jost Bürgi simplified the redundant Stévin’s notation with a small circle separating the integer from the fractional part
- in 1592 Magini introduced the point as the decimal separator
- at the beginning of the XVII century Willbord Snellius introduced the comma as the decimal separator

A short historical perspective

Fractional decimal numbers were also introduced in the XVI century;

- in 1582 Simon Stévin started using fractional decimal numbers without reference to fractions, but using “decimals”
- in 1592 Jost Bürgi simplified the redundant Stévin’s notation with a small circle separating the integer from the fractional part
- in 1592 Magini introduced the point as the decimal separator
- at the beginning of the XVII century Willbord Snellius introduced the comma as the decimal separator

A short historical perspective

Fractional decimal numbers were also introduced in the XVI century;

- in 1582 Simon Stévin started using fractional decimal numbers without reference to fractions, but using “decimals”
- in 1592 Jost Bürgi simplified the redundant Stévin’s notation with a small circle separating the integer from the fractional part
- in 1592 Magini introduced the point as the decimal separator
- at the beginning of the XVII century Willbord Snellius introduced the comma as the decimal separator

A short historical perspective

Fractional decimal numbers were also introduced in the XVI century;

- in 1582 Simon Stévin started using fractional decimal numbers without reference to fractions, but using “decimals”
- in 1592 Jost Bürgi simplified the redundant Stévin’s notation with a small circle separating the integer from the fractional part
- in 1592 Magini introduced the point as the decimal separator
- at the beginning of the XVII century Willbord Snellius introduced the comma as the decimal separator

A short historical perspective

During the XVI century Tartaglia and Cardano (1545) introduced the square root of −1 but the name of the *imaginary unit* was introduced in 1637 by Réné Descartes.
Gauss (1799) highly contributed to the diffusion of complex numbers; but a solid theory was published by Hamilton in 1833.

Conclusion of the historical sketch

Decimal fractional numbers are at least four centuries old, so we are certainly in the position to state that decimal fractional numbers can be part of our preferred program TeX and `pdftex` with their variants LaTeX and `pdflatex`.
Even complex numbers are just as old; why not using both kinds of number for extending the potential capabilities of TeX?

The `picture` environment of LaTeX

As everybody knows LaTeX has a built in mechanism for drawing pictures: the `picture` environment.

As everybody knows that environment is very essential and is good for drawing elementary pictures, such as very simple block diagrams.

When in 1994 LaTeX 2_ε was published and Leslie Lamport wrote his manual second edition, he established a “syntax” for a new, extended `picture` environment but until 2003 nobody wrote the code.

Packages that circumvented such limitations

Meanwhile people needed to draw line art, diagrams, histograms, et-cetera, and needed alternative solutions.

Possibly the oldest and most widely known is the **PSTricks** package that, through a typical LaTeX interface, writes to the DVI file raw PostScript code and requires processing by **dvips** that converts the file into the PostScript language.

Typically **ghostscript** and its graphical interface **Gview** are used for printing and viewing on screen.

Packages that avoided such limitations

More recently Till Tantau wrote a beautiful package **pgf**, *portable graphics format*, that attempts to bring to `pdflaltex` the functionality that **PSTricks** gives to LaTeX.

Since the page description language of the PDF format is a simple subset of the PostScript language, **pgf** must operate in a smarter way so as to supply the output file with solid PDF code.

At the moment **pgf** does not supply the same functionality as **PSTricks** but it is performing very, very well and no doubt it will arrive to the same functionality of his older brother.

Packages that avoided such limitations

On a simpler basis the packages **curves** and **curveslm** have been on the scene for several years. The first was capable of designing arbitrary curves even with LaTeX 209, but used only second degree Bézier splines so that sometimes the algorithm produced strange loops in proximity of the inflection points if the nodes were not properly chosen.
The second package had the same functionality, but resorted to the printer driver facilities, more or less as the new package **pict2e**.

ConTEXt and METAPOST

In 2002 Hans Hagen published **METAFUN** with its manual; the program is intended to mix TEX code and METAPOST code and to use a special extension of ConTEXt that allows to run alternatively TEX and METAPOST so as to embed a picture produced in PostScript by METAPOST into a PDF file.
The idea of **METAFUN** is to produce serious PostScript code by means of METAPOST, which uses a simplified PostScript language, so simplified that even `pdflatex` can understand it and produce the desired graphical result.
The manual itself is typeset by means of Hagen's program and the graphical effects obtainable are of superior quality.

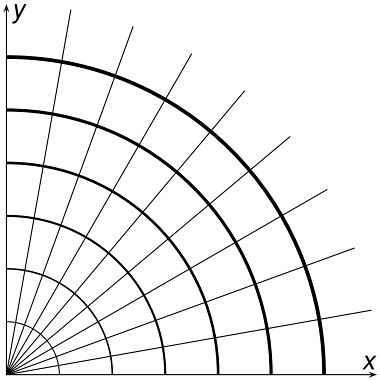
The extended `picture` environment of LaTeX

In 2003 the long expected extension to the `picture` environment was published; it was realized by H.Gäßlein and R.Niepraschk by means of the extension package `pict2e`.

The segment and vector slope limitations are eliminated; the circle radii can be chosen at will; the ovals corners may have radii chosen at will; the line thickness may be chosen at will also for sloping or curved lines; cubic Bézier splines are added to the quadratic ones, but both draw their lines with continuous strokes, rather than with the superposition of a multitude of dots.

Example picture

This figure is composed with the facilities offered by `pict2e` for slopes, circle radii, and curve thickness.



Limitations of the pict2e extension package

Lamport in 1994, and Gäßlein & Niepraschk maintained some limitations to the picture functionality the most noticeable of which is the fact that the slope parameters of segments and vectors must still be integers within a limited although wide range: 0–999.

There is no reason to keep these slope parameters in the set of integers, but this is what Lamport prescribed in his specifications published well before the software was realized.

Division between fractional numbers

Gäßlein & Niepraschk implemented a division routine where numerator and denominator are integers; they got inspiration from other software; their `\pIIe@@divide` macro is similar to the one used in the **trig** package.

Their routine actually divides two lengths, so there would not be any problem if these lengths corresponded to fractional, instead of integer, numbers of points.

Decimal long division

I realized a decimal long division macro that, may be, is a little slower than `\pIIe@@divide`, but has been working for me in the past ten years and never gave me any problem: no overflows, no divisions by zero, et-cetera. It receives two lengths and outputs their dimensionless ratio.
The terms of the division may be two generic lengths, but in general they are two fractional decimal numbers used as scale factors for the internal dimension `\p@` that equals 1pt.

First extensions to `pict2e`

With this new long division macro the line and vector commands are redefined so as to avoid the limitation imposed by Lamport that requires their slope parameters to be integer numbers whose absolute value does not exceed 999.
Not only, but without this restriction it is possible to define new macros that greatly ease the drawing of polygonal lines.

\Line, \Vector and \polyline macros

If the slope parameters can be any (reasonable) decimal fractional number, why can't they be directly the horizontal and the vertical projections of the segment or vector? Why should the composer still keep track of the horizontal component only?

```
\put(23.7,35.2){\Line(12.5,32.3)}
```

simply puts a segment starting at the picture coordinates 23.7 and 35.2 and reaches 12.5 units further right and 32.3 units further up.

\polyline

Even better: a polyline macro joins an arbitrary number of points with the suitable segments:

```
\polyline(23.7,35.2)(32.7,71.4)(10,71.4)(10,0)
```

starts out at the picture coordinate 23.7 and 35.2 and sequentially joins the subsequent points ending up at point 10 and 0.

Actually there is no need to compute the segment slopes because **pict2e** already contains the `moveto`, `lineto` and `stroke` macros that make the task much easier.

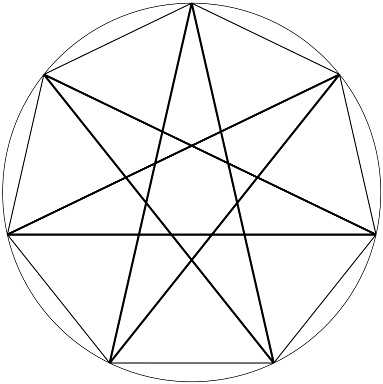
Line terminators

Another small improvement may be realized with macros that set the line terminators and the shape of the curve joins; all these imply passing the output file the proper PostScript or PDF commands such that the curves end with a square or a semicircular terminator, or such that curves are joined by means of round or bevel joins.

These are simple declarations that should be valid for an entire picture environment or until a different declaration is used.

A heptagon and a heptagonal star

Another picture drawn with the last suggested extensions



The complex number extension

As recalled in the historical introduction, complex numbers have been in use for at least four centuries.
Besides many scientific disciplines, if the code is correctly read, it can even be realized that complex numbers are the base of graphical languages such as PostScript, METAFONT, METAPOST, et-cetera.

The scale rotate operator

Complex numbers are not that complex!
They are simply operators that perform at the same time a scaling and a plane rotation.
Everybody is familiar with scaling; everybody is familiar with rotation.
Their form allows them to be treated as if they were numbers, but numbers of a special kind, so special that in common usage they are called *complex*, but…

Forget about complexity!

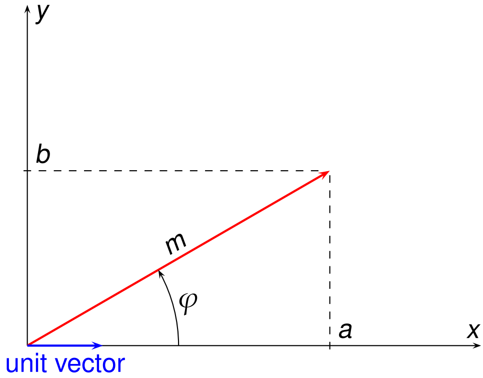
Writing complex numbers

Any written form of a complex number must contain explicitly or implicitly two pieces of information:

1. the scaling factor, and
2. the rotation angle

These pieces of information can be assembled in a variety of ways that emphasize this or that peculiarity of the operator.

Writing complex numbers



As an operator, the complex number acts on a vector, in this figure on the *unit horizontal vector*; it scales it by the factor m and rotates it by the angle φ producing the red vector.

Writing complex numbers

The complex number may be written in these equivalent ways:

$$(a, b) = a + \mathrm{i}\, b = m\,\mathrm{e}^{\mathrm{i}\varphi} = m\angle\varphi$$

The first notation is preferred by the mathematicians, the last by the engineers; i is the so called *imaginary unit*.
For TEX inner workings it is better the form (a, b), but the last form makes it more understandable the scaling–rotating performance of a complex number.

The direction

The angle φ determines the direction of the operator acting on the unit horizontal vector. Since any vector may be thought of a unit horizontal vector scaled and rotated by a complex number, the resulting vector direction is identified by φ.

Any unit vector with angle φ identifies the direction of all vectors with any magnitude but with the same angle φ.

The possibility of identifying directions is a precious feature of complex numbers.

Advanced features of **pict2e**

The extension package **pict2e** contains a variety of advanced features that are already usable or that may be usefully exploited.
Complex numbers help a lot to dig out the tremendous possibilities offered by **pict2e**.
I will not describe all the inner workings, the package and the various complex number arithmetical operations that may be implemented.
I will present only some final results, that are relatively small samples of what can be done with **pict2e**.

Arcs of any amplitude

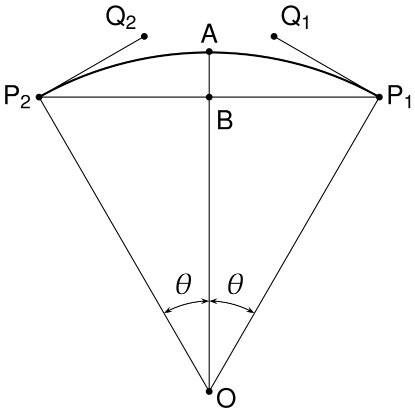
The macro

`\Arc(⟨center⟩)(⟨starting point⟩){⟨amplitude⟩}`

draws an arc centered in ⟨*center*⟩, starting at the indicated ⟨*starting point*⟩ where the ⟨*amplitude*⟩ is expressed in sexagesimal degrees; the positive direction of rotation is *counterclockwise*.

Arc geometrical parameters

The geometrical construction of the arc parameters is an interesting application of what has been exposed up to now.

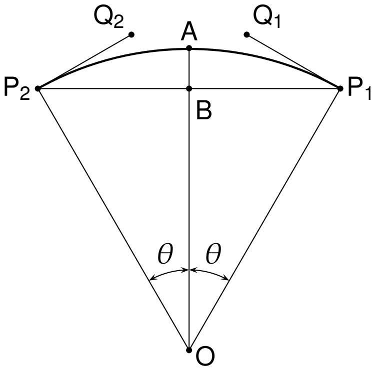


Arc geometrical parameters

The given information are the coordinates of the center point O, the starting point P_1 and the aperture 2θ.

Point P_2, the ending point, is determined by rotating vector $\overrightarrow{OP_1}$ by the angle 2θ.

The problem is to find the control points Q_1 and Q_2 of the third order Bézier spline joining P_1 and P_2 so that the arc is the best approximation to a circular arc.

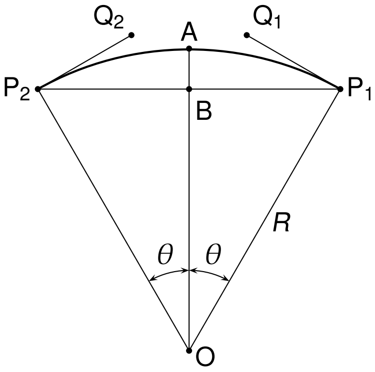


Arc geometrical parameters

The segments $\overline{P_1Q_1}$ and $\overline{P_2Q_2}$ must be tangent to the arc, so they must lie on the perpendiculars to $\overrightarrow{OP_1}$ and $\overrightarrow{OP_2}$; the only question is: how long are these segments?

Taking into account the Bézier spline equation and the fact that $\overline{OA}$ must equal $\overline{OP_1} = R$, the unknown distance K may be found and it turns to be

$$K = \frac{4}{3}\frac{(1-\cos\theta)R}{\sin\theta}$$



Arc geometrical parameters

At this point the unit direction vectors $\overrightarrow{P_1Q_1}$ and $\overrightarrow{P_2Q_2}$ may be scaled by K and the four parameters for the third degree Bézier spline are completely known.

The **pict2e** cubic spline available in this package may be invoked and the circular arc may be drawn.

Notice that a 180° degree cubic spline approximates so well a semicircle that the difference is barely noticeable by naked eye.

A 90° degree cubic spline is such a good approximation that the small difference may be computed but it is absolutely invisible.

Therefore a wider arc may be split in a succession of 90° arcs followed by the remaining small part and the various pieces my be easily concatenated so as to show a single curve.

Arched vectors

If we know how to draw circular arcs, we can affix an arrow tip at the end point or at both ends and have arched vectors; the macros are

$$\backslash VectorArc(\langle center\rangle)(\langle start\_point\rangle)\{\langle amplitude\rangle\}$$
$$\backslash VectorARC(\langle center\rangle)(\langle start\_point\rangle)\{\langle amplitude\rangle\}$$

Examples of both arched vectors have been used throughout this presentation.

What about a generic arc?

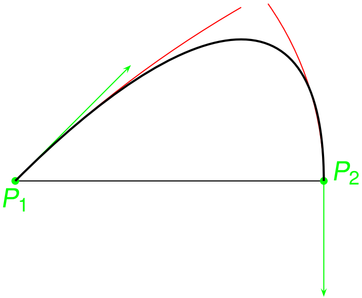
If one desires to draw a generic arc knowing only the starting and ending points and the tangents at these points, **pict2e** and the above calculations for circular arcs are sufficient to make up a macro such as this:

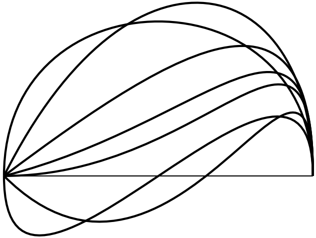
\CurveBetween⟨*start_point*⟩and⟨*end_point*⟩WithDirs⟨*start_dir*⟩and⟨*end_dir*⟩

The delimited argument macro is not in the LaTeX tradition but it seems to be clearer when actual parameters are to be inserted.

Generic arc parameters

By knowing just the directions at the end points, one may assume, among the infinity of possible solutions, that the control points along the directions are at distances K_1 and K_2 such that these distances define the parameters of the osculating circles to the desired generic arc.





Multi-arc curves

Since we know now how to draw a generic arc, we want to concatenate several arcs such that the end point tangent direction of the $(n-1)$-th arc coincides with the starting point tangent of the n-th arc.

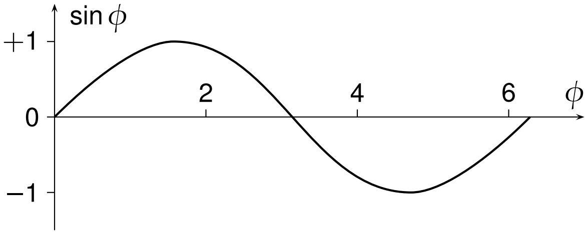
To this end I designed a macro with the following syntax

$$\texttt{\textbackslash Curve(}\langle P_0\rangle\texttt{)<}\langle dir_0\rangle\texttt{>(}\langle P_1\rangle\texttt{)<}\langle dir_1\rangle\texttt{>...}$$
$$\texttt{(}\langle P_k\rangle\texttt{)<}\langle dir_k\rangle\texttt{>[}\langle dir'_k\rangle\texttt{]}\texttt{(}\langle P_{k+1}\rangle\texttt{)<}\langle dir_{k+1}\rangle\texttt{>...}$$
$$\texttt{(}\langle P_N\rangle\texttt{)<}\langle dir_N\rangle\texttt{>}$$

in order to draw a complicated curve made up of N generic arcs.

The optional argument enclosed in square brackets allows to change direction across a node so as to draw a cusp.

A smooth multi-arc curve: a sine wave

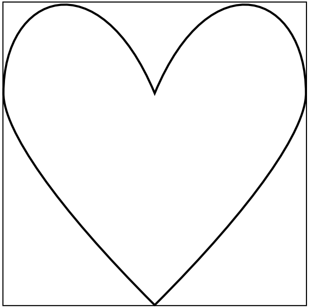


The relevant code is the following

```
\Curve(0,0)<1,1>%       0 deg
(1.570796,1)<1,0>%    90 deg
(4.712389,-1)<1,0>% 270 deg
(6.283185,0)<1,1>%  360 deg
```

Note: the angle ϕ is in radians.

A curve with cusps



The relevant code is the following

```
\Curve(2.5,0)<1,1>(5,3.5)<0,1>%
(2.5,3.5)<-.5,-1.2>[-.5,1.2]%
(0,3.5)<0,-1>(2.5,0)<1,-1>
```

pict2e is more powerful than expected

The inner commands of **pict2e** may be brought to the composer with no effort so as to extend its usefulness.
The typical higher level graphic language commands such as `moveto`, `lineto`, `curveto`, `stroke`, `fill` are there but they are well hidden behind internal commands protected with the prefix `\pIIe@` so that a user interface is required.
Its division routine has more or less the functionalities as the one I used; it simply needs to be cleaned up from the excessive constraints imposed by Lamport's syntax.

Woe to integers!

If the predilection for integers is removed, even with the modest capabilities of TeX in calculating with fractional numbers, it is possible to achieve a pretty good functionality.
Of course if my plea for the extension of e-TeX so as to natively deal with fractional numbers (floating point numbers) was accepted, all the graphic packages could obtain great benefits; the benefits would help very much also some other LaTeX internals, for example the New Font Selection Scheme.

Complex numbers and scale-rotate operators

Of course it is not necessary to call complex numbers the scale-rotate operators; their functionality is hard coded into programs such as METAFONT and METAPOST, but they are at the base of any graphical software; why should just TeX keep away from them?

Well, yes! I agree: TeX is not a graphical software, it is the best typesetter. But it is such a good piece of software that even graphics, up to a pretty good level, are within its reach.

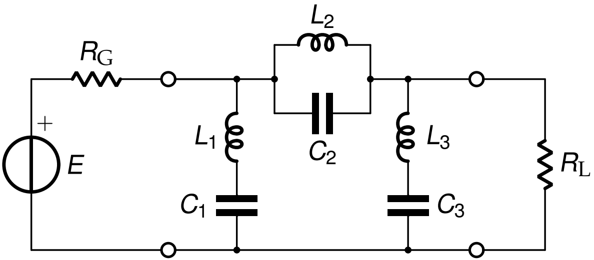
The **curve2e** package

The macros I described and the functionality they produce have been uploaded in CTAN about a year ago with file **curve2e**.
You can examine the code, spot the errors, extend the functionality, may be even produce a new version of **pict2e** that makes use of what I presented so as to exploit TEX to its maximum graphics capacities.

Extension of other packages

I am confident that the ideas I presented, in particular the use of scale-rotate operators, may benefit also other graphical packages. After I made my extensions for generic curves built up with a concatenation of cubic Bézier splines, I did not touch any other graphical software.
OK, I do not make generic drawings and I don't need all the functionality of, say, **PSTricks**, but with the modest extension I made I can avoid the long path `latex` + `dvips` + `pstopdf` in order to produce PDF files with decent graphics.

One of my drawings



Happy TeXing!