

Re: Newbie looking for info on basic graphics with Delphi.

# Re: Newbie looking for info on basic graphics with Delphi.

---

*Source:* <http://coding.derkeiler.com/Archive/Delphi/comp.lang.pascal.delphi.misc/2006-10/msg00142.html>

---

- *From:* "Maarten Wiltink" <[maarten@xxxxxxxxxxxxxxxxxxxxx](mailto:maarten@xxxxxxxxxxxxxxxxxxxxx)>
  - *Date:* Tue, 31 Oct 2006 10:37:39 +0100
- 

"CC" <[dontbother@xxxxxxxxxxxxxxxxx](mailto:dontbother@xxxxxxxxxxxxxxxxx)> wrote in message  
[news:r1mck2plt1earbrrunbira5irceqam8ge4@xxxxxxxxxxx](mailto:news:r1mck2plt1earbrrunbira5irceqam8ge4@xxxxxxxxxxx)

On Mon, 30 Oct 2006 10:03:34 +0100, "Maarten Wiltink"  
<[maarten@xxxxxxxxxxxxxxxxxxxxx](mailto:maarten@xxxxxxxxxxxxxxxxxxxxx)> wrote:

[...]

Where can I find some basic algorithms for wireframe  
drawing?

I have some source. I'll see if I can put it on my website.

Would it be asking too much of you to post an example of what you  
describe here? Please and thanks!

No, I was rather hoping you'd ask. We all like to show off. (—:

The wireframe object is derived from TPersistent because I like  
Assign as a hook. I overrode Assign to check for TStrings, so I  
could assign a TStrings to the object and it would parse them,  
and I overrode AssignTo to check for TCanvas, so I could assign  
the object to a TCanvas and it would render itself onto it.

Actually, just now I see it's derived from TComponent so I could  
register it and drop it on a form.

The file format is simple: a text file with first three reals  
(x, y, z) on a line that say where the origin is, then lines with  
six reals (x0, y0, z0, x1, y1, z1) that describe lines in the  
wireframe. This can be made more complex as desired.

The wireframe itself is of course three-dimensional; for rendering  
a projection onto the x-y plane is performed (a fancy way of saying  
that the z-coordinate is thrown away) and all lines are simply drawn

Re: Newbie looking for info on basic graphics with Delphi.

Re: Newbie looking for info on basic graphics with Delphi.

with MoveTo/LineTo.

The interesting stuff is what you do with your x-, y-, and z-coordinates before you draw them.

The object has three properties that are angles, one that is a 3D point (the origin), one that is a 2D point (the location on the canvas where the origin will be), and a zoom factor. Strictly speaking, most of those properties are only needed for rendering which is a function outside the model itself, so they should be in a 'scene' object which contains wireframes, but this is how it's now. It works.

The translations are easy. Just simple scalar math. (Vector math, really, but even I just wrote out the two or three lines because they both happen just once.)

(These paragraphs to be read in a monospaced font)

Rotations are matrix multiplications. Going back to 2D for a moment, a rotation of a vector  $w = \begin{pmatrix} x \\ y \end{pmatrix}$

over an angle  $\phi$  is done by (left) multiplication with  $T = \begin{pmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{pmatrix}$ .

The result  $T*w$  is equal to  $\begin{pmatrix} x*\cos \phi - y*\sin \phi \\ x*\sin \phi + y*\cos \phi \end{pmatrix}$

and is  $w$  rotated by

the angle  $\phi$ . Get thee to a pencil and paper, and try it out. It's

magic. Even knowing \*why\* it can't really do anything else than rotate (or mirror), I still find it mystifying and beautiful.

It scales to 3D by fleshing out the 2x2 matrix to a 3x3 matrix and putting ones and zeroes in the right place to transport one coordinate over unchanged. To rotate x and y (therefore, about the z axis), blow up

$T$  to  $T_z = \begin{pmatrix} \cos \phi & -\sin \phi & 0 \\ \sin \phi & \cos \phi & 0 \\ 0 & 0 & 1 \end{pmatrix}$ .

To rotate about the y axis (and therefore,

x and z), use one of the other two angles (every axis gets its own angle in the object), and expand  $T$  to  $T_y = \begin{pmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{pmatrix}$ .

When rendering a bunch of lines, all the points have to be transformed the same way, say  $w' = T_x*T_y*T_z*w$ . The order of the T's is important but I can't tell you which is the right one to make it behave as you want.

Unfortunately, matrix multiplication is expensive; fortunately, it is also associative and you can compute  $T = T_x*T_y*T_z$  ahead of time, doing only a single (rotational) transformation on every point during rendering. There are tricks to do the translations and the projection with matrix multiplications, too, but I kept those apart.

Rendering, then, comes out as the following steps in turn:

1. Translate against model origin. (This might be done during loading,

Re: Newbie looking for info on basic graphics with Delphi.

Re: Newbie looking for info on basic graphics with Delphi.

but I wanted to be able to shift the origin afterwards to rotate around different points in the model.)

2. Rotate. Multiply with the compound transformation matrix.
3. Project. Throw away the z-coordinate.
4. Zoom. Multiply all coordinates with the zoom factor. (Might be done a step earlier, too; I took it as part of the transformation from 'world' (model) to 'window' (canvas) coordinates.)
5. Translate against canvas centre. (0, 0) on a canvas is in the top left corner; since rotation is about the model's origin, that comes out better in the middle of the canvas.

All these steps are performed for both points in every line. The compound rotation is precomputed.

All that, in 395 lines. If it's rocket science, it's little more than a distress flare. A budget model, and without much distress.

You can read up on matrix multiplication and rotational transformations at MathWorld ([.wolfram.com](http://www.wolfram.com)). I heartily endorse that product and/or service.

I have a library unit that implements the TmwWireframe object and two small programs that use it: a simple viewer that rotates, shifts, and re-centers (an important function) the model according to keypresses, and an animated viewer that rotates them on a timer (think Elite, but without the Strauss waltz). I'll show you mine if you show me yours. (By which I mean: \*try\* first.)

Groetjes,  
Maarten Wiltink

.