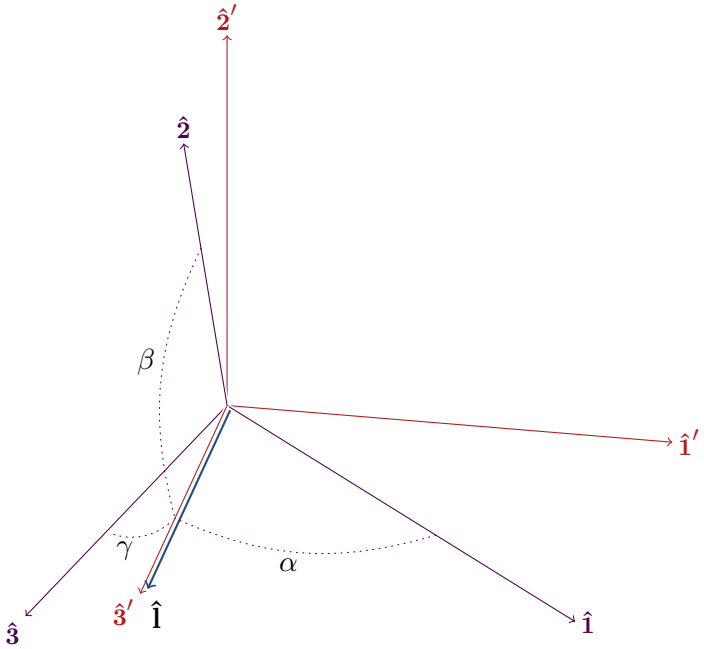
Line-of-Sight Vector and the Viewing Plane

Paul Kotschy

31 May 2016

Compiled on July 10, 2025



Abstract

In my work,[1] I often need to conceptualise three-dimensional landscapes, and then to render them on paper or screen in two dimensions. Given the impressive Ti*k*Z system for typesetting graphical content directly in a TEX document, I don't want to have to look elsewhere. As a typesetting system, TEX and Ti*k*Z make for a powerful and feature-rich pair, and Ti*k*Z has a consistent and intuitive user-interface. But it is ostensibly a two-dimensional drawing capability. I would like to typeset three-dimensional graphical landscapes and diagrams using Ti*k*Z.

By working with the notion of a viewing plane perpendicular to a line-of-sight, I describe how a three-dimensional landscape may be rendered by projecting it onto the viewing plane. And by way of an example, I demonstrate how I now use Ti*k*Z, TEX and my `PKREALVECTOR` C object class to typeset such three-dimensional landscapes with relative ease.

[1] I declare this to be my own work, entirely. In particular, no AI was used in any research, analysis, synthesis, writing, nor typesetting of this work. In short, AI was not recruited at any time in this work. Errors and inaccuracies are therefore proudly my own.

Contents

1 A three-dimensional landscape

Suppose we are given a set of points $\{\mathbf{x} = x\hat{\mathbf{1}} + y\hat{\mathbf{2}} + z\hat{\mathbf{3}}\}$ which comprises a three-dimensional "landscape". As objective observers of this landscape, we are in fact unable to view at once this landscape in its entirety. Instead, we pick a "line of sight" (normally subconsciously) relative to some orientation of the landscape, and view a projection of the landscape onto the plane perpendicular to this line of sight. Strictly speaking, we pick two lines of sight, one for each eye, and view two projections, using the subtle differences between the two to infer depth in the landscape. But this analysis considers only one of the projections.

To record graphically on paper or on screen what our one eye sees, we must represent the three-dimensional landscape on paper. The landscape must first be specified (obviously). A line of sight must then be chosen. And a projection must be calculated which can then be drawn on the paper.

The landscape shown in Figure 1 is simple. It consists of a single position vector

$$\mathbf{x} = x\hat{\mathbf{1}} + y\hat{\mathbf{2}} + z\hat{\mathbf{3}} \tag{1}$$

That is, the landscape consists of the quantities x, y, z, and the orthonormal vector basis $\{\hat{\mathbf{1}}, \hat{\mathbf{2}}, \hat{\mathbf{3}}\}$. If the landscape is expressed in terms of an alternative basis, say $\{\hat{\mathbf{1}}', \hat{\mathbf{2}}', \hat{\mathbf{3}}'\}$, as

$$\mathbf{x} = x'\hat{\mathbf{1}}' + y'\hat{\mathbf{2}}' + z'\hat{\mathbf{3}}' \tag{2}$$

and if the basis is oriented such that its $\hat{\mathbf{3}}'$ basis vector is parallel to our desired line of sight, then $\hat{\mathbf{3}}'$ cannot be rendered, because it would be oriented perpendicular to our page. But the projection of $\mathbf{x}$ onto the $\hat{\mathbf{2}}'\hat{\mathbf{3}}'$ plane can be. It is simply $\mathbf{x_p} = x'\hat{\mathbf{1}}' + y'\hat{\mathbf{2}}'$, and x' and y' can happily be drawn on the paper to represent $\mathbf{x}$. I shall call the $\hat{\mathbf{2}}'\hat{\mathbf{3}}'$ plane the *viewing plane*. This analysis, then, concerns the calculation of $\mathbf{x_p}$ in terms of $\hat{\mathbf{1}}'$ and $\hat{\mathbf{2}}'$ to be located in the viewing plane.

2 Line-of-sight

The line of sight can be arbitrarily specified in the landscape. In this analysis I choose to specify a *line-of-sight vector* $\hat{\mathbf{l}}$ as

$$\hat{\mathbf{l}}(\alpha, \beta, \gamma) = \hat{\mathbf{3}}'(\alpha, \beta, \gamma) = \cos\alpha\,\hat{\mathbf{1}} + \cos\beta\,\hat{\mathbf{2}} + \cos\gamma\,\hat{\mathbf{3}} \tag{3}$$

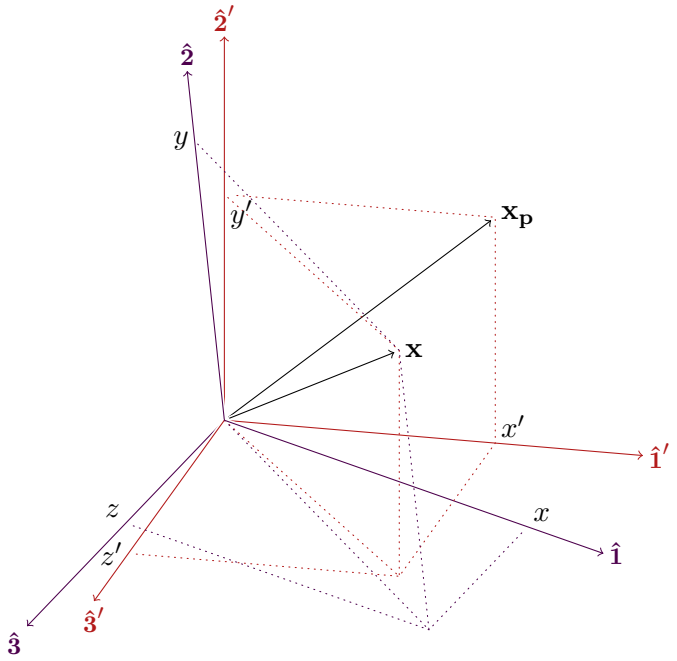


Figure 1: A three-dimensional landscape specified as a single position vector $\mathbf{x} = x\hat{\mathbf{1}} + y\hat{\mathbf{2}} + z\hat{\mathbf{3}}$. The projection vector $\mathbf{x_p}$ of $\mathbf{x}$ is calculated to lie in the $\hat{\mathbf{1}}'\hat{\mathbf{2}}'$ viewing plane which lies perpendicular to the chosen "line-of-sight" vector $\hat{\mathbf{l}} = \hat{\mathbf{3}}'$.

for some angles α, β and γ, as shown in Figure 2. The vector is of unit length provided that

$$\cos^2\gamma = 1 - \cos^2\alpha - \cos^2\beta = \sin^2\beta - \cos^2\alpha \tag{4}$$

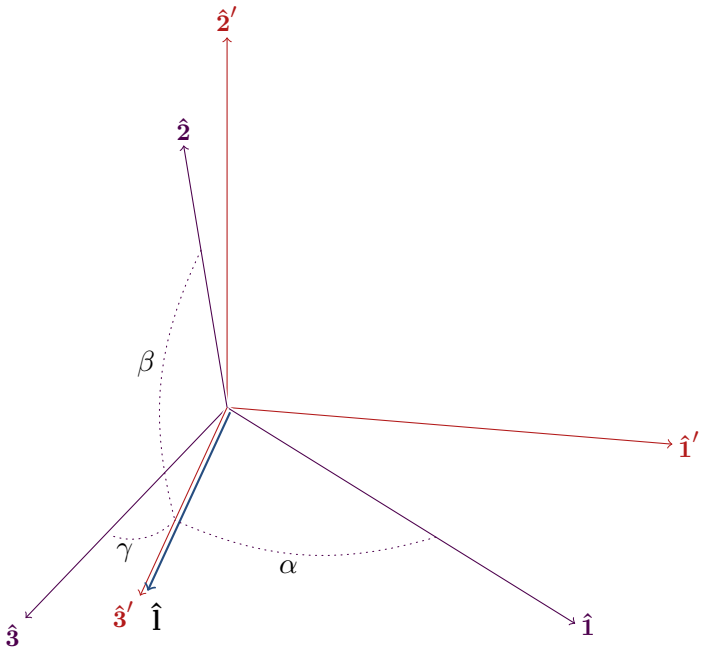


Figure 2: Specification of the line-of-sight vector, $\hat{\mathbf{l}} = \hat{\mathbf{3}}' = \cos\alpha\hat{\mathbf{1}} + \cos\beta\hat{\mathbf{2}} + \cos\gamma\hat{\mathbf{3}}$, subject to $\cos^2\alpha + \cos^2\beta + \cos^2\gamma = 1$.

So we proceed using the three angles α, β and γ, but remembering that only two of the three are independent and can therefore be used as input parameters. Furthermore, if α and β are used as the input parameters, to ensure that γ is a real quantity, α and β must satisfy

$$\sin\beta > \cos\alpha$$

By specifying a line-of-sight vector $\hat{\mathbf{l}}$ in the $\hat{\mathbf{1}}\hat{\mathbf{2}}\hat{\mathbf{3}}$ space (Figure 2), we can think of $\hat{\mathbf{l}}$ helping to fix $\hat{\mathbf{1}}$, $\hat{\mathbf{2}}$ and $\hat{\mathbf{3}}$ in the $\hat{\mathbf{1}}'\hat{\mathbf{2}}'\hat{\mathbf{3}}'$ space.

3 Angle of tilt

But the fixing is not complete. Indeed, $\hat{\mathbf{1}}$ can be rotated onto $\hat{\mathbf{2}}$, $\hat{\mathbf{2}}$ onto $\hat{\mathbf{3}}$, and $\hat{\mathbf{3}}$ onto $\hat{\mathbf{1}}$ without α, β and γ changing. An additional constraint is therefore needed. I choose to fix the *angle of tilt* of the $\hat{\mathbf{2}}$ basis vector relative to the $\hat{\mathbf{2}}'\hat{\mathbf{3}}'$ plane, naming the angle θ, as shown in Figure 3.

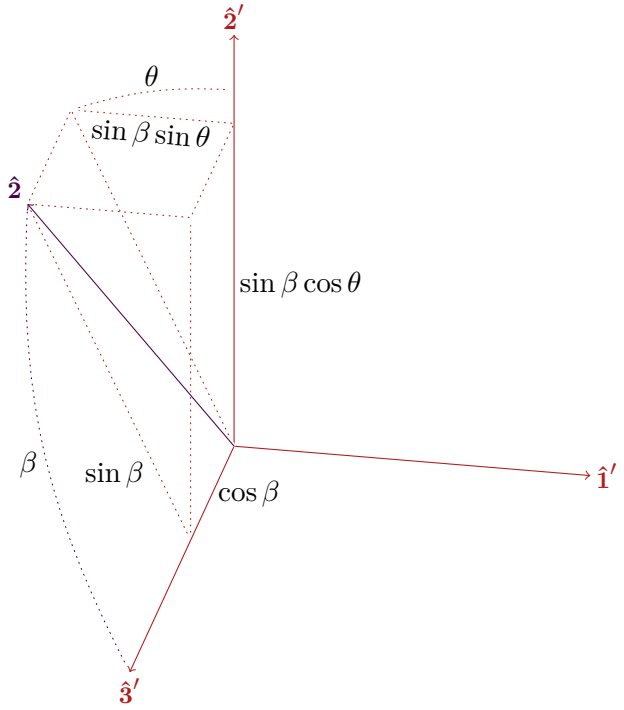


Figure 3: Specification of the angle of tilt, θ, as the angle subtended between the $\hat{\mathbf{2}}$ basis vector and the $\hat{\mathbf{2}}'\hat{\mathbf{3}}'$ plane in a direction perpendicular to the $\hat{\mathbf{2}}'\hat{\mathbf{3}}'$ plane.

By inspecting the various distances derived from the definitions of β and θ, the $\hat{\mathbf{2}}$ basis vector may at once be expressed in terms of $\{\hat{\mathbf{1}}', \hat{\mathbf{2}}', \hat{\mathbf{3}}'\}$ as

$$\hat{\mathbf{2}} = -\sin\beta\sin\theta\,\hat{\mathbf{1}}' + \sin\beta\cos\theta\,\hat{\mathbf{2}}' + \cos\beta\,\hat{\mathbf{3}}' \tag{5}$$

4 The landscape's projection

If the $\hat{\mathbf{1}}$, $\hat{\mathbf{2}}$ and $\hat{\mathbf{3}}$ basis vectors which span our given landscape can be expressed in terms of $\hat{\mathbf{1}}'$, $\hat{\mathbf{2}}'$ and $\hat{\mathbf{3}}'$, then so can any vector, such as $\mathbf{x}$, also be. So we begin by writing

$$\begin{aligned}
\hat{\mathbf{1}} &= a_1\hat{\mathbf{1}}' + a_2\hat{\mathbf{2}}' + a_3\hat{\mathbf{3}}' \\
\hat{\mathbf{2}} &= b_1\hat{\mathbf{1}}' + b_2\hat{\mathbf{2}}' + b_3\hat{\mathbf{3}}' \\
\hat{\mathbf{3}} &= c_1\hat{\mathbf{1}}' + c_2\hat{\mathbf{2}}' + c_3\hat{\mathbf{3}}'
\end{aligned} \tag{6}$$

with the objective of solving for a_1, a_2, …, c_3, subject to the conditions (3), (4), (5), and the orthonormality of the $\{\hat{\mathbf{1}}, \hat{\mathbf{2}}, \hat{\mathbf{3}}\}$ vector basis:

$$\begin{aligned}
\hat{\mathbf{1}}\cdot\hat{\mathbf{2}} = \hat{\mathbf{2}}\cdot\hat{\mathbf{3}} = 0 \\
\hat{\mathbf{1}}\cdot\hat{\mathbf{1}} = \hat{\mathbf{3}}\cdot\hat{\mathbf{3}} = 1
\end{aligned} \tag{7}$$

We know immediately from (5) that

$$\begin{aligned} b_1 &= -\sin\beta\sin\theta \\ b_2 &= \sin\beta\cos\theta \\ b_3 &= \cos\beta \end{aligned} \tag{8}$$

Substituting (5), (6) and (8) into (3) gives

$$\begin{aligned} \cos\alpha\,(a_1\,\hat{\mathbf{1}}' + a_2\,\hat{\mathbf{2}}' + a_3\,\hat{\mathbf{3}}') \;+\; &\cos\beta\,(-\sin\beta\sin\theta\,\hat{\mathbf{1}}' + \sin\beta\cos\theta\,\hat{\mathbf{2}}' + \cos\beta\,\hat{\mathbf{3}}') \\ &+\; \cos\gamma\,(c_1\,\hat{\mathbf{1}}' + c_2\,\hat{\mathbf{2}}' + c_3\,\hat{\mathbf{3}}') = \hat{\mathbf{3}}' \end{aligned}$$

And rearranging,

$$\begin{aligned} \hat{\mathbf{3}}' \;=\; &(\cos\alpha\,a_1 - \sin\beta\cos\beta\sin\theta + \cos\gamma\,c_1)\,\hat{\mathbf{1}}' \\ &+\; (\cos\alpha\,a_2 + \sin\beta\cos\beta\cos\theta + \cos\gamma\,c_2)\,\hat{\mathbf{2}}' \\ &+\; (\cos\alpha\,a_3 + \cos^2\beta + \cos\gamma\,c_3)\,\hat{\mathbf{3}}' \end{aligned}$$

And since $\hat{\mathbf{1}}'$, $\hat{\mathbf{2}}'$ and $\hat{\mathbf{3}}'$ are linearly independent,

$$\begin{aligned} \cos\alpha\,a_1 + \cos\gamma\,c_1 \;&=\; \sin\beta\cos\beta\sin\theta \\ \cos\alpha\,a_2 + \cos\gamma\,c_2 \;&=\; -\sin\beta\cos\beta\cos\theta \\ \cos\alpha\,a_3 + \cos\gamma\,c_3 \;&=\; 1 - \cos^2\beta = \sin^2\beta \end{aligned}$$

from which

$$\begin{aligned} c_1 \;&=\; -\frac{\cos\alpha}{\cos\gamma}\,a_1 \;+\; \frac{\sin\beta\cos\beta\sin\theta}{\cos\gamma} \\ c_2 \;&=\; -\frac{\cos\alpha}{\cos\gamma}\,a_2 \;-\; \frac{\sin\beta\cos\beta\cos\theta}{\cos\gamma} \\ c_3 \;&=\; -\frac{\cos\alpha}{\cos\gamma}\,a_3 \;+\; \frac{\sin^2\beta}{\cos\gamma} \end{aligned} \tag{9}$$

The four orthonormality conditions in (7) give

$$\begin{aligned} \hat{\mathbf{1}}\cdot\hat{\mathbf{2}} \;&=\; -\sin\beta\sin\theta\,a_1 + \sin\beta\cos\theta\,a_2 + \cos\beta\,a_3 \;=\; 0 \\ \hat{\mathbf{2}}\cdot\hat{\mathbf{3}} \;&=\; -\sin\beta\sin\theta\,c_1 + \sin\beta\cos\theta\,c_2 + \cos\beta\,c_3 \;=\; 0 \\ \hat{\mathbf{1}}\cdot\hat{\mathbf{1}} \;&=\; a_1^2 + a_2^2 + a_3^2 \;=\; 1 \\ \hat{\mathbf{3}}\cdot\hat{\mathbf{3}} \;&=\; c_1^2 + c_2^2 + c_3^2 \;=\; 1 \end{aligned} \tag{10}$$

Substituting (9) into the fourth condition in (10) gives

$$(-\cos\alpha\,a_1 + \sin\beta\cos\beta\sin\theta)^2 \;+\; (-\cos\alpha\,a_2 - \sin\beta\cos\beta\cos\theta)^2 \;+\; \left(-\cos\alpha\,a_3 + \sin^2\beta\right)^2 \;=\; \cos^2\gamma$$

$$\begin{aligned} \cos^2\alpha(a_1^2 + a_2^2 + a_3^2) \;+\; &(\sin\beta\cos\beta)^2(\sin^2\theta + \cos^2\theta) \;+\; (\sin^2\beta)^2 \\ &+\; 2\left[\sin\beta\cos\beta\cos\alpha(\cos\theta\,a_2 - \sin\theta\,a_1) - \sin^2\beta\cos\alpha\,a_3\right] = \cos^2\gamma \end{aligned}$$

Using the third condition in (10), this may by simplified to

$$\sin\beta\,[\cos\beta(\cos\theta\,a_2 - \sin\theta\,a_1) - \sin\beta\,a_3] + \cos\alpha \;=\; 0$$

But from the first condition in (10),

$$\cos\theta\,a_2 - \sin\theta\,a_1 \;=\; -\frac{\cos\beta}{\sin\beta}\,a_3$$

So

$$\sin\beta\left[\cos\beta\left(-\frac{\cos\beta}{\sin\beta}\,a_3\right) - \sin\beta\,a_3\right] + \cos\alpha \;=\; 0$$

Solving for a_3 gives

$$a_3 \;=\; \cos\alpha$$

This result for a_3 may now be substituted into the first and third conditions in (10) to give

$$\begin{aligned} \sin\beta\,(\sin\theta\,a_1 - \cos\theta\,a_2) \;&=\; \cos\alpha\cos\beta \\ a_1^2 + a_2^2 \;&=\; 1 - \cos^2\alpha \;=\; \sin^2\alpha \end{aligned} \tag{11}$$

4.1 Case $\beta = 0$

If $\beta = 0$, then the first condition in (11) requires that $\alpha = (2n+1)\frac{\pi}{2}$ and $\gamma = (2m+1)\frac{\pi}{2}$ for any integers n and m. Using the second condition in (11), together with the further orthonormality condition $\hat{\mathbf{1}} \cdot \hat{\mathbf{3}} = 0$, gives

$$\begin{aligned}
\hat{\mathbf{1}} &= a_1 \hat{\mathbf{1}}' + \sqrt{1 - a_1^2}\, \hat{\mathbf{2}}' \\
\hat{\mathbf{2}} &= \hat{\mathbf{3}}' \\
\hat{\mathbf{3}} &= -\sqrt{1 - a_1^2}\, \hat{\mathbf{1}}' + a_1 \hat{\mathbf{2}}'
\end{aligned}$$

In this form, $\hat{\mathbf{1}}$, $\hat{\mathbf{2}}$ and $\hat{\mathbf{3}}$ satisfy all the orthonormality constraints (Eq. (7)), which means that a_1 is a free parameter. I shall choose to use the specified value of θ to fix a_1 by setting the angle between $\hat{\mathbf{1}}$ and $\hat{\mathbf{1}}'$ equal to θ, so that $\hat{\mathbf{1}} \cdot \hat{\mathbf{1}}' = \cos\theta$. In this case $\beta = 0$ then, our basis vectors are

$$\boxed{\begin{aligned}
\hat{\mathbf{1}} &= \cos\theta\, \hat{\mathbf{1}}' + \sin\theta\, \hat{\mathbf{2}}' \\
\hat{\mathbf{2}} &= \hat{\mathbf{3}}' \\
\hat{\mathbf{3}} &= -\sin\theta\, \hat{\mathbf{1}}' + \cos\theta\, \hat{\mathbf{2}}'
\end{aligned}} \tag{12}$$

4.2 Case $\beta \neq 0$

When $\beta \neq 0$, the two simultaneous equations in (11) may easily be solved for a_1 and a_2, giving

$$\begin{aligned}
a_1 &= \frac{\cos\alpha\cos\beta}{\sin\beta}\sin\theta + \sqrt{1 - \frac{\cos^2\alpha}{\sin^2\beta}}\,\cos\theta \\
a_2 &= \sqrt{1 - \frac{\cos^2\alpha}{\sin^2\beta}}\,\sin\theta - \frac{\cos\alpha\cos\beta}{\sin\beta}\cos\theta
\end{aligned}$$

Or using (4),

$$\begin{aligned}
a_1 &= \frac{1}{\sin\beta}\left(\cos\alpha\cos\beta\sin\theta + \cos\gamma\cos\theta\right) \\
a_2 &= \frac{1}{\sin\beta}\left(\cos\gamma\sin\theta - \cos\alpha\cos\beta\cos\theta\right)
\end{aligned}$$

Direct substituting into (9), and after some simplification, gives

$$\begin{aligned}
c_1 &= \frac{1}{\sin\beta}\left(\cos\gamma\cos\beta\sin\theta - \cos\alpha\cos\theta\right) \\
c_2 &= \frac{1}{\sin\beta}\left(-\cos\alpha\sin\theta - \cos\gamma\cos\beta\cos\theta\right) \\
c_3 &= \cos\gamma
\end{aligned}$$

With the solutions to a_1, a_2, …, c_3, we are now able to express the $\hat{\mathbf{1}}$, $\hat{\mathbf{2}}$ and $\hat{\mathbf{3}}$ basis vectors in the $\hat{\mathbf{1}}'\hat{\mathbf{2}}'\hat{\mathbf{3}}'$ space ((5) and (6)) as

$$\begin{aligned}
\hat{\mathbf{1}} &= \frac{1}{\sin\beta}\left(\cos\alpha\cos\beta\sin\theta + \cos\gamma\cos\theta\right)\hat{\mathbf{1}}' + \frac{1}{\sin\beta}\left(\cos\gamma\sin\theta - \cos\alpha\cos\beta\cos\theta\right)\hat{\mathbf{2}}' + \cos\alpha\,\hat{\mathbf{3}}' \\
\hat{\mathbf{2}} &= -\sin\beta\sin\theta\hat{\mathbf{1}}' + \sin\beta\cos\theta\hat{\mathbf{2}}' + \cos\beta\,\hat{\mathbf{3}}' \\
\hat{\mathbf{3}} &= \frac{1}{\sin\beta}\left(\cos\gamma\cos\beta\sin\theta - \cos\alpha\cos\theta\right)\hat{\mathbf{1}}' + \frac{1}{\sin\beta}\left(-\cos\alpha\sin\theta - \cos\gamma\cos\beta\cos\theta\right)\hat{\mathbf{2}}' + \cos\gamma\,\hat{\mathbf{3}}'
\end{aligned} \tag{13}$$

provided that $\cos\gamma = \sqrt{1 - \cos^2\alpha - \cos^2\beta}$.

And consequently, any vector $\mathbf{x}$ embedded in the $\hat{\mathbf{1}}\hat{\mathbf{2}}\hat{\mathbf{3}}$ space (Equation (1)) is transformed into the same vector in the $\hat{\mathbf{1}}'\hat{\mathbf{2}}'\hat{\mathbf{3}}'$ space (Equation (2)) as:

$$\begin{aligned}\mathbf{x} &= x'\hat{\mathbf{1}}' + y'\hat{\mathbf{2}}' + z'\hat{\mathbf{3}}' \\ &= \left[\frac{1}{\sin\beta}\left(\cos\alpha\cos\beta\sin\theta + \cos\gamma\cos\theta\right)x - \sin\beta\sin\theta\, y + \frac{1}{\sin\beta}\left(\cos\gamma\cos\beta\sin\theta - \cos\alpha\cos\theta\right)z\right]\hat{\mathbf{1}}' \\ &+ \left[\frac{1}{\sin\beta}\left(\cos\gamma\sin\theta - \cos\alpha\cos\beta\cos\theta\right)x + \sin\beta\cos\theta\, y + \frac{1}{\sin\beta}\left(-\cos\alpha\sin\theta - \cos\gamma\cos\beta\cos\theta\right)z\right]\hat{\mathbf{2}}' \\ &+ \left[\cos\alpha\, x + \cos\beta\, y + \cos\gamma\, z\right]\hat{\mathbf{3}}'\end{aligned} \tag{14}$$

4.3 Vector-centric solution for the case $\beta \neq 0$.

The task is essentially complete because the projection vector $\mathbf{x_p}$ is obtained by simply discarding $\mathbf{x}$'s $\hat{\mathbf{3}}'$ component in (14). But it would be interesting to recast (14) in a more vector–centric way. Consider, firstly, $\mathbf{x}$'s $\hat{\mathbf{1}}'$ component. Rearranging the terms,

$$\begin{aligned}\sin\beta x' &= (x\cos\alpha + y\cos\beta + z\cos\gamma)\cos\beta\sin\theta - y\cos^2\beta\sin\theta + (x\cos\gamma - z\cos\alpha)\cos\theta \\ &\quad - y\sin^2\beta\sin\theta \\ &= (x\cos\alpha + y\cos\beta + z\cos\gamma)\cos\beta\sin\theta - y\sin\theta - (z\cos\alpha - x\cos\gamma)\cos\theta \\ &= \cos\beta\sin\theta\,\mathbf{x}\cdot\hat{\mathbf{3}}' - \sin\theta\,\mathbf{x}\cdot\hat{\mathbf{2}} - \cos\theta\,(\mathbf{x}\times\hat{\mathbf{3}}')\cdot\hat{\mathbf{2}} \\ &= \cos\beta\sin\theta\,\mathbf{x}\cdot\hat{\mathbf{3}}' - \sin\theta\,\mathbf{x}\cdot\hat{\mathbf{2}} - \cos\theta\,(\mathbf{x}\times\hat{\mathbf{3}}')\cdot\hat{\mathbf{2}} + \cos\theta\cos\beta\,(\mathbf{x}\times\hat{\mathbf{3}}')\cdot\hat{\mathbf{3}}' \\ &\qquad\qquad \text{because } (\mathbf{x}\times\hat{\mathbf{3}}')\cdot\hat{\mathbf{3}}' \equiv 0\end{aligned}$$

Factorising gives

$$x' = \frac{1}{\sin\beta}\left(\sin\theta\,\mathbf{x} + \cos\theta\,\mathbf{x}\times\hat{\mathbf{l}}\right)\cdot\left(\cos\beta\,\hat{\mathbf{l}} - \hat{\mathbf{2}}\right)$$

The $\hat{\mathbf{2}}'$ and $\hat{\mathbf{3}}'$ components may be calculated similarly, giving

$$\boxed{\begin{aligned}\mathbf{x} &= x'\hat{\mathbf{1}}' + y'\hat{\mathbf{2}}' + z'\hat{\mathbf{3}}' \\ &= \frac{1}{\sin\beta}\left(\sin\theta\,\mathbf{x} + \cos\theta\,\mathbf{x}\times\hat{\mathbf{l}}\right)\cdot\left(\cos\beta\,\hat{\mathbf{l}} - \hat{\mathbf{2}}\right)\hat{\mathbf{1}}' \\ &\quad + \frac{1}{\sin\beta}\left(-\cos\theta\,\mathbf{x} + \sin\theta\,\mathbf{x}\times\hat{\mathbf{l}}\right)\cdot\left(\cos\beta\,\hat{\mathbf{l}} - \hat{\mathbf{2}}\right)\hat{\mathbf{2}}' \\ &\quad + (\mathbf{x}\cdot\hat{\mathbf{l}})\,\hat{\mathbf{3}}'\end{aligned}} \tag{15}$$

The result gives a prescription for the transformed landscape (namely, $\mathbf{x}$ in the $\hat{\mathbf{1}}'\hat{\mathbf{2}}'\hat{\mathbf{3}}'$ space) in terms of the original landscape ($\mathbf{x}$ in the $\hat{\mathbf{1}}\hat{\mathbf{2}}\hat{\mathbf{3}}$ space), the specified angles β and θ, and the specified line-of-sight vector $\hat{\mathbf{l}}$ (Eq. (3)).

5 Example

Suppose that we wish our line-of-sight vector $\hat{\mathbf{l}}$ to lie on the $\hat{\mathbf{1}}\hat{\mathbf{3}}$ plane. For this we would need to specify $\beta = \frac{\pi}{2}$ (or 90°). From (3), our line-of-sight vector becomes $\hat{\mathbf{l}} = \cos\alpha\,\hat{\mathbf{1}} + \sin\alpha\hat{\mathbf{3}}$, so $\mathbf{x}\times\hat{\mathbf{l}} = y\sin\alpha\,\hat{\mathbf{1}} + (z\cos\alpha - x\sin\alpha)\hat{\mathbf{2}} - y\cos\alpha\,\hat{\mathbf{3}}$. And from (15), our transformed landscape becomes

$$\begin{aligned}\mathbf{x} &= ((x\sin\alpha - z\cos\alpha)\cos\theta - y\sin\theta)\hat{\mathbf{1}}' \\ &\quad + ((x\sin\alpha - z\cos\alpha)\sin\theta + y\cos\theta)\hat{\mathbf{2}}' \\ &\quad + (x\cos\alpha + z\sin\alpha)\hat{\mathbf{3}}'\end{aligned}$$

If we decide now to not impose any angle of tilt, so $\theta = 0$:

$$\mathbf{x} \;=\; (x\sin\alpha - z\cos\alpha)\hat{\mathbf{1}}' + y\,\hat{\mathbf{2}}' + (x\cos\alpha + z\sin\alpha)\hat{\mathbf{3}}'$$

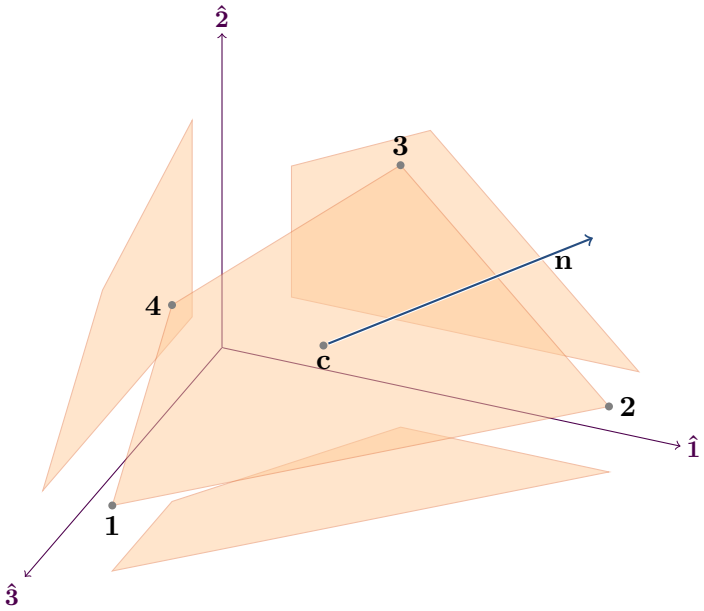
Finally, if we wish for our line-of-sight vector to coincide with the $\hat{\mathbf{3}}$ basis vector, so $\alpha = \dfrac{\pi}{2}$:

$$\mathbf{x} \;=\; x\hat{\mathbf{1}}' + y\hat{\mathbf{2}}' + z\hat{\mathbf{3}}'$$

as expected.

6 Drawing with TEX, Ti*k*Z and PKREALVECTOR

In this section I demonstrate the combined use of TEX, Ti*k*Z and my PKREALVECTOR C object class to produce three-dimensional schematic diagrams in TEX, such as this one:



It works as follows. Suppose we wish to typeset a PDF-formatted document containing a single diagram of three planar projections of a surface embedded in $\mathbb{R}^3$, as shown above. We'll begin with a UNIX `Makefile`. UNIX's MAKE system is an excellent tool to help manage TEX-typesetting projects. Prepare a MAKE configuration file named `Makefile` with the content:

```
#------------------------------------------------------------
# Generic Make targets.
#
all: projections.pdf

clean:
        @rm -f projections.pdf
        @rm -f *.o
        @rm -f *.run three-projections.tex

#------------------------------------------------------------
# File based Make targets.
#
projections.pdf: projections.tex three-projections.tex

#------------------------------------------------------------
# Implicit rule targets.
#
```

```
19  .SUFFIXES: .c .o .run .tex .pdf
20  .c.o:
21          clang -c -DDEBUG=2 -I/usr/local/pklib/include -DFreeBSD -o ${@} ${<}
22  .o.run:
23          clang -DDEBUG=2 -I/usr/local/pklib/include -DFreeBSD -o ${@} \pkShellSlash
24                          ${<} \pkShellSlash
25                          /usr/local/pklib/lib/libpk.a \pkShellSlash
26                          /usr/local/pklib/lib/libpkmath.a \pkShellSlash
27                          -lm
28  .run.tex:
29          ./${<} > ${@}
30  .tex.pdf:
31          pdflatex ${<}
```

This simple "Makefile" for MAKE captures all the necessary file dependencies, and will trigger the required actions in accordance with these file dependencies. MAKE's `make` command will read this `Makefile` and use it to typeset a TEX input file named `projections.tex`. The output will be a PDF-formatted file named `projections.pdf`. Prepare the `projections.tex` with the content:

```
1  \documentclass[a4paper,11pt]{article}
2  \usepackage[T1]{fontenc}
3  \usepackage{lmodern}
4  \usepackage{pktikz}
5
6  \begin{document}
7  \pagestyle{empty}
8  \begin{center}
9     \input{three-projections.tex}
10 \end{center}
11 \end{document}
```

This `projections.tex` TEX source file `\input`'s another external TEX source file named `three-projections.tex`, expecting that file to contain the Ti*k*Z source code instructions for typesetting the diagram. But that file should not exist yet. Instead it will be generated dynamically as the output of the execution of the `three-projections.run` executable program, which in turn will be created by compiling the C code located in a file named `three-projections.c`. Actually, by virtue of the abovementioned `Makefile`, you simply needed to type `make` (or `make all`) to create the final document file, `projections.pdf`

Prepare the `three-projections.c` C source file as follows. Note that the listing below includes typesetted annotations. Just follow the source line numbers.

```
1  /*
2   * This C source file has been primed to be typeset using my
3   * pkTechDoc literate programming system.  PJ Kotschy. 18May16.
4   */
5  #include <pkfeatures.h>
6
7  #include <stddef.h>
8  #include <stdlib.h>
9  #include <stdio.h>
10 #include <unistd.h>
11 #include <stdarg.h>
12 #include <string.h>
13 #include <math.h>
14
15 #include <pkmemdebug.h>
16 #include <pkerror.h>
17 #include <pktypes.h>
```

```
18  #include <pkstring.h>
19  #include <pkmath.h>
20  #include <pkrealvector.h>
21
22  const char *LOGFNAME = "/tmp/diagram.log";
23
24  void _printVector( const PKREALVECTOR *v )
25  {
26     printf( "Vector %s = ( ", pkRealVectorGetName(v) );
27     pkRealVectorPrintf( v, "__COMPONENTVALUE__", ", " );
28     puts(" )");
29     return;
30  }
```

The `_diagram()` private function below specifies the diagram's three-dimensional landscape. It does this primarily using the `PKREALVECTOR` object class. The function prints to standard output a body of Ti*k*Z source code which may be used to typeset the landscape in TEX.

But before this function can do so, it must transform the landscape in such a way that what Ti*k*Z typesets is a two-dimensional projection of the three-dimensional landscape. The function rotationally transforms the landscape onto the space spanned by the $\{\hat{\mathbf{1}}', \hat{\mathbf{2}}', \hat{\mathbf{3}}'\}$ orthonormal vector basis set, where the $\hat{\mathbf{1}}'$ and $\hat{\mathbf{2}}'$ basis vectors lie in the plane of the page and $\hat{\mathbf{3}}'$ is perpendicular to the page, i.e., lying parallel to the reader's line of sight.

In the function, the `alpha` (α) and `beta` (β) are required for the transformation. The angle α lies between between the landscape's $\hat{\mathbf{1}}$ basis vector and $\hat{\mathbf{3}}'$; β is the angle between the landscape's $\hat{\mathbf{2}}$ basis vector and $\hat{\mathbf{3}}'$; and θ is the tilt angle between $\hat{\mathbf{2}}$ and the $\hat{\mathbf{2}}'\hat{\mathbf{3}}'$ plane. The actual transformation is affected by the call

```
pkRealVectorsUnderLineOfSightBasis2( alpha, beta, theta
                                     e1, e2, e3,
                                     ...
                                     NULL )
```

```
31  static void _diagram(void)
32  {
33     const PKMATHREAL alpha = 70.0 / 180.0 * M_PI,
34                      beta  = 60.0 / 180.0 * M_PI,
35                      theta = 00.0 / 180.0 * M_PI;
36     PKREALVECTOR *e1, *e2, *e3,  /* Orthonormal basis vectors. */
37                  *v[4][4],       /* Four planes, four vertices per plane. */
38                  *c,             /* Geometric centre of the plane. */
39                  *n,             /* Average normal vector. */
40                  *p;             /* Point at 'c+n'. */
41     int i, j;
42
```

Here we specify the three-dimensional landscape.

```
43     e1 = pkRealVectorAlloc1( "\\one",   3, 6.0, 0.0, 0.0 );
44     e2 = pkRealVectorAlloc1( "\\two",   3, 0.0, 6.0, 0.0 );
45     e3 = pkRealVectorAlloc1( "\\three", 3, 0.0, 0.0, 6.0 );
46     pkRealVectorScale(e1,1.1);
47     pkRealVectorScale(e2,0.8);
48     pkRealVectorScale(e3,1.1);
49
50     v[0][0] = pkRealVectorAlloc1( "\\vecVone",   3, 1.0, 1.0, 6.0 );
51     v[0][1] = pkRealVectorAlloc1( "\\vecVtwo",   3, 6.0, 1.0, 1.0 );
```

```
52      v[0][2] = pkRealVectorAlloc1( "\\vecVthree", 3, 3.0, 4.0, 1.0 );
53      v[0][3] = pkRealVectorAlloc1( "\\vecVfour",  3, 1.0, 3.0, 4.0 );
54      for ( i = 1; i < 4; i++ ) {
55         for ( j = 0; j < 4; j++ )
56            v[i][j] = pkRealVectorAlloc0( "", 3, pkRealVectorGetComponent(v[0][j]) );
57      }
58      for ( j = 0; j < 4; j++ ) {
59         pkRealVectorGetComponent(v[1][j])[0] = 0.0;
60         pkRealVectorGetComponent(v[2][j])[1] = 0.0;
61         pkRealVectorGetComponent(v[3][j])[2] = 0.0;
62      }
63
64      c = pkRealVectorAlloc1( "\\vecC", 3, 0.0, 0.0, 0.0 );
65      for ( j = 0; j < 4; j++ )
66         pkRealVectorIncrease( c, v[0][j] );
67      pkRealVectorScale(c,0.25);
68      {
69         PKREALVECTOR *d1 = pkRealVectorAllocDiff( "d1", v[0][2], v[0][0] ),
70                      *d2 = pkRealVectorAllocDiff( "d2", v[0][3], v[0][1] );
71         n = pkRealVectorAllocCrossProduct( "\\vecN", d1, d2 );
72         pkRealVectorFreeDiff(d1);
73         pkRealVectorFreeDiff(d2);
74      }
75      pkRealVectorNormalise(n);
76      pkRealVectorScale(n,12.0);
77      p = pkRealVectorAllocSum( "p", c, n );
```

Rotationally transform the landscape onto the space spanned by the abovementioned “line-of-site” basis.

```
78      if ( 0 == pkRealVectorsUnderLineOfSightBasis2( alpha, beta, theta,
79                                                     e1, e2, e3,
80                                                     v[0][0], v[0][1], v[0][2], v[0][3],
81                                                     v[1][0], v[1][1], v[1][2], v[1][3],
82                                                     v[2][0], v[2][1], v[2][2], v[2][3],
83                                                     v[3][0], v[3][1], v[3][2], v[3][3],
84                                                     c, n, p,
85                                                     NULL ) ) {
86
```

Output TikZ source code to typeset a projection of the transformed landscape. The projection is easy because, under the transformation, the 3-component of all points and vectors now points out of the page, parallel to $\hat{\mathbf{3}}'$.

```
87         puts(   "\\begin{PkTikzpicture}[scale=1.0]" );
88         puts(   "   %" );
89         puts(   "   %\\draw[help lines] (-0.2,-0.2) grid (7.1,5.1);");
90         puts(   "   %");
91         puts(   "   \\coordinate (origin) at (0,0);");
92         printf( "   \\coordinate (v0) at (%g,%g);\n"
93                 "   \\coordinate (v1) at (%g,%g);\n"
94                 "   \\coordinate (v2) at (%g,%g);\n"
95                 "   \\coordinate (v3) at (%g,%g);\n"
96                 "   \\coordinate (c) at (%g,%g);\n",
97                 pkRealVectorGetComponent(v[0][0])[0],
98                 pkRealVectorGetComponent(v[0][0])[1],
99                 pkRealVectorGetComponent(v[0][1])[0],
100                pkRealVectorGetComponent(v[0][1])[1],
101                pkRealVectorGetComponent(v[0][2])[0],
102                pkRealVectorGetComponent(v[0][2])[1],
```

```
                pkRealVectorGetComponent(v[0][3])[0],
                pkRealVectorGetComponent(v[0][3])[1],
                pkRealVectorGetComponent(c)[0],
                pkRealVectorGetComponent(c)[1] );
        puts(   "   %" );
        printf( "   \\draw[pktikzbasisvector,<->]\n"
                "         (%g,%g) node[right]{$%s$} --\n"
                "         (origin) --\n"
                "         (%g,%g) node[above]{$%s$};\n"
                "   \\draw[pktikzbasisvector,->]\n"
                "         (origin) --\n"
                "         (%g,%g) node[below left]{$%s$};\n",
                pkRealVectorGetComponent(e1)[0], pkRealVectorGetComponent(e1)[1],
                pkRealVectorGetName(e1),
                pkRealVectorGetComponent(e2)[0], pkRealVectorGetComponent(e2)[1],
                pkRealVectorGetName(e2),
                pkRealVectorGetComponent(e3)[0], pkRealVectorGetComponent(e3)[1],
                pkRealVectorGetName(e3) );
        puts(   "   %" );
        for ( i = 1; i < 4; i++ ) {
           printf( "   \\draw[pktikztranslucentsurface]\n"
                   "         (%g,%g) --\n"
                   "         (%g,%g) --\n"
                   "         (%g,%g) --\n"
                   "         (%g,%g) -- cycle;\n",
                   pkRealVectorGetComponent(v[i][0])[0],
                   pkRealVectorGetComponent(v[i][0])[1],
                   pkRealVectorGetComponent(v[i][1])[0],
                   pkRealVectorGetComponent(v[i][1])[1],
                   pkRealVectorGetComponent(v[i][2])[0],
                   pkRealVectorGetComponent(v[i][2])[1],
                   pkRealVectorGetComponent(v[i][3])[0],
                   pkRealVectorGetComponent(v[i][3])[1] );
        }
        puts(   "   %" );
        printf( "   \\draw[pktikztranslucentsurface] (v0) -- (v1) -- (v2) -- (v3) -- cycle;\n" );
        printf( "   \\path (v0) coordinate[pktikzpoint,label=below:$%s$] --\n"
                "          (v1) coordinate[pktikzpoint,label=right:$%s$] --\n"
                "          (v2) coordinate[pktikzpoint,label=above:$%s$] --\n"
                "          (v3) coordinate[pktikzpoint,label=left:$%s$];\n",
                pkRealVectorGetName(v[0][0]),
                pkRealVectorGetName(v[0][1]),
                pkRealVectorGetName(v[0][2]),
                pkRealVectorGetName(v[0][3]) );
        puts(   "   %" );
        printf( "   \\path (c) coordinate[pktikzpoint,label=below:$%s$];\n",
                pkRealVectorGetName(c) );
        printf( "   \\draw[pktikzemphvector]\n"
                "         (c) --\n"
                "         (%g,%g) node[black,very near end,below]{$%s$};\n",
                pkRealVectorGetComponent(p)[0],
                pkRealVectorGetComponent(p)[1],
                pkRealVectorGetName(n) );
        puts(   "\\end{PkTikzpicture}" );

    } else {

        puts("ERROR: 'pkRealVectorsUnderLineOfSightBasis2()' failed.");

    }
```

Clean up.

```
163      pkRealVectorFree1(e1);
164      pkRealVectorFree1(e2);
165      pkRealVectorFree1(e3);
166      for ( i = 0; i < 4; i++ ) {
167         for ( j = 0; j < 4; j++ )
168            pkRealVectorFree1(v[i][j]);
169      }
170      pkRealVectorFree1(c);
171      pkRealVectorFreeCrossProduct(n);
172      pkRealVectorFreeSum(p);
173
174      return;
175   }
176
177   int main( const int argc, const char *argv[] )
178   {
179      _diagram();
180      exit(0);
181   }
```