Tiling

Jeff Rowley

Abstract

This document will provide some basic code to construct two dimensional diagrams that appear three dimensional for application to (mathematical) tiling problems.

Acknowledgements. *Considerable credit must be given to Tim Murphy for the many illustrative examples he has made available on the internet. The diagrams created in this document are adaptations of the code presented in his final example. Credit also to Alain Matthes for his excellent LaTeXpackage,* `tkz-euclide`*, which has been used to render the trigonometric diagrams. Thanks also to Professor D.A. Preece*[1] *for introducing the author to* tredoku *and motivating the problem as a (mathematical) tiling problem.*

1 *Tredoku*

Tredoku is essentially a game of *sudoku* played in three dimensions. Whereas *sudoku* is played on a single square in two dimensions, *tredoku* is played over many squares that are arranged in three dimensions. The rules of and procedures for solving the game are not considered herein; rather, the arrangements of the major squares (or *tiles*) and any recurring patterns in runs of various lengths is of interest. Producing diagrams of different arrangements in electronic format for inclusion in a LaTeXdocument is potentially tricky given that the problem is three dimensional but graphically can only be represented in two dimensions. Without breaking down the problem into a series of smaller problems, one could become quickly overwhelmed by the number of dimensions (each tile consists of four vectors with different origins, and there are many tiles, each of which can be at a different depth due to the three dimensions of arrangements). In the course of this document, we consider the problem of graphically presenting tiling arrangements in as simple a manner as possible.

2 Orientation

Before doing anything, we have to consider the orientation with which we wish to display our tiling arrangement. Let us define O to be the origin. Supposing that the diagram is to be orientated such that the y-axis is displayed along the $(0,1)'$ vector[2] (that is, displayed vertically), we must state the vectors along which the x-axis and z-axis are to lie.

$$\vec{Oy} = (0,1)'$$

[1] Emeritus Professor of Mathematics at Queen Mary, University of London.

[2] The diagram itself is in two dimensional space so we must map (Euclidean) vectors in $\mathbb{R}^3$ space into $\mathbb{R}^2$ space using some function, h.

Let us think about some particular orientations.

The most extreme orientation is where the z-dimension is projected perpendicular onto the xy-plane by the matrix $\mathbf{P}_0$, where $\mathbf{P}_0$ is defined below.

$$\mathbf{P}_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

In this specific example the z-axis effectively ceases to exist in the diagram since we remove all depth when we apply the projection matrix $\mathbf{P}_0$ to any coordinate vector $(x, y, z)'$.

$$\vec{O}x = (1, 0)'$$
$$\vec{O}z = (0, 0)'$$

The axes corresponding to this particular projection are shown in FIGURE 1.

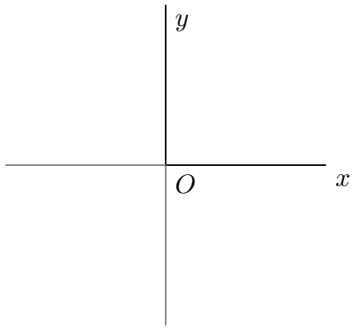

Figure 1: The orientation when the projection matrix $\mathbf{P}_0$ is applied. The z-dimension effectively ceases to exist.

For regression type problems where we are effectively projecting from $\mathbb{R}^{k+1}$ space onto $\mathbb{R}^k$ space (for models with a scalar unobservable and k observable covariates) this is satisfactory. In the context of the z-dimension being observable and containing some useful information, this is not satisfactory at all. We must consider only those projections that are not perpendicular to the xy-plane.

Consider instead some projection that makes the xz-plane and yz-plane visible as well as the xy-plane. That is, suppose that we rotate the plane that we are projecting onto such that it is no longer perpendicular to the xz-plane or the yz-plane. As an analogy, consider putting a *Rubik's cube* on the table. Looking at the *Rubik's cube* at the level of the table, we see only the leading face. This is the xy-plane as defined above. Now suppose that we turn the *Rubik's cube* clockwise a little. We can now see the right face of the *Rubik's cube*, or yz-plane. Finally, suppose that we tilt the *Rubik's cube* forward a little. We can now see the top face of the *Rubik's cube*, or xz-plane. Notice that the same result could be achieved by altering the point of observation.[3] Any point on or in the cube will now appear as if it is in a different position. We must now calculate the position of each point in $\mathbb{R}^2$ space when the *Rubik's cube* is rotated and tilted, and projected perpendicularly to the plane of observation.

A remark: both the rotation and tilt of the object occur along lines that intersect at the origin, O. Let θ be the angle of (clockwise) rotation; let ϕ be the angle of tilt (forwards). As a normalisation, both θ and ϕ are restricted to be less than a quarter turn. That is, $\theta, \phi \in (0°, 90°)$ where angles are given in degrees for convenience. Define $f : \mathbb{R}^3 \to \mathbb{R}^2$ to be a function that maps each point to its position on the projection. As the analogy shows, f can either be thought of as

[3]While a point of observation seems sensible in the analogy, a plane of observation (or projection) is more technically correct for the application.

a single function or as a composite function. We will think of f as a composite function in which rotation and tilt occur sequentially.

2.1 Rotation

What happens to an arbitrary vector, $\mathbf{a} = (a_1, a_2, a_3)'$, when we rotate the axes? Clearly, rotating the axes has no effect on the vertical position of any point in $\mathbb{R}^3$ space, either unprojected or on the projection. In contrast, both the x-coordinate and z-coordinate of any point will change. As the y-dimension is unaffected by the rotation, we restrict attention to the xz-plane.

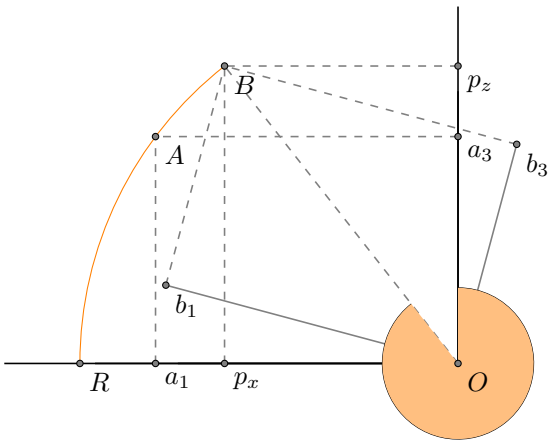

Figure 2: Restricting attention to the xz-plane, an arbitrary point, A, with x-component a_1 and z-component a_3 is rotated by θ degrees clockwise about the origin, O. This is the point B. Projected onto the original axes, B has x-component p_x and z-component p_z. The orange shaded angle is the angle ω, defined below.

The relevant points that we wish to calculate in Figure 2 are p_x and p_z. First, notice that if we know the length OB and some angle between O and B then we can calculate p_x and p_z. The angle we use is deliberately left unspecified at this point since the diagram could alternatively be drawn in any other quadrant or over two quadrants in which case we need to be careful which angle we specify since the sign of the points is important. We will discuss the choice of angle in greater detail in what follows.

OB is relatively straightforward to calculate, being the hypotenuse of AOa_1, since A and B lie on the circumference of a circle centred at O.

$$r \equiv OB = (a_1^2 + a_3^2)^{\frac{1}{2}} \tag{2.1.1}$$

As r is a length, it is restricted to be positive. Hence, only positive roots are considered.

Adopting $\omega \equiv \measuredangle Z^+OB$, which can be reflex, as our angle between O and B, we can derive expressions for p_x and p_z that will vary in sign according to the position of B.

$$p_x = r\sin(\omega) \tag{2.1.2}$$

$$p_z = r\cos(\omega) \tag{2.1.3}$$

Thus, when B is in the negative-negative quadrant, p_x and p_z are both negative, as is required. All that remains is to develop a means of calculating the angle ω using the information available to us.

$$\omega = \begin{cases} \theta + \tan^{-1}\left(\frac{|a_3|}{|a_1|}\right) + 90° \times \left(\frac{|\text{sgn}(a_1) - \text{sgn}(a_3)|}{2} + 2 \times \mathbb{1}[a_1 < 0] + \frac{1}{2} \times [a_3 = 0]\right) & \text{if } a_1 \neq 0, \\ \theta + 180° \times \mathbb{1}[a_3 < 0] & \text{if } a_1 = 0. \end{cases} \tag{2.1.4}$$

Having defined r in (2.1.1) and ω in (2.1.4), we have p_x and p_z in terms of known quantities, defined according to (2.1.2) and (2.1.3). Essentially, we have defined p_x and p_z to be functions that map θ, a_x, a_z to $\mathbb{R}$. For any vector A, there is a function $g : \mathbb{R}^3 \to \mathbb{R}^3$ that rotates A on the xz-plane about the origin.

$$g(x, y, z; \theta) \equiv (p_x(x, z; \theta), y, p_z(x, z; \theta))'$$

This completes our analysis of the effect of a horizontal rotation. We now switch our attention to the effect of a vertical tilt on the direction and length of an arbitrary vector.

2.2 Tilt

Notice that a horizontal rotation is the same as tilting the object if the orientation of the problem is changed. Hence, many of the results that we have obtained above are applicable. In this case, it is the x-component of the vector that is unaffected by tilt. Attention can be restricted to the yz-plane. Adapting FIGURE 2, the z-direction becomes the y-direction and the x-direction becomes the z-direction in the diagram. Suppose that we tilt (forwards) an arbitrary vector, $\mathbf{c} = (c_1, c_2, c_3)'$, by ϕ degrees.

A remark: tilting a vector forwards should be thought of as an anticlockwise rotation on the yz-plane (because the negative axis is closest to the point of observation). All results for the angle ϕ in this section will be negative so that ϕ remains within the interval $(0°, 90°)$.

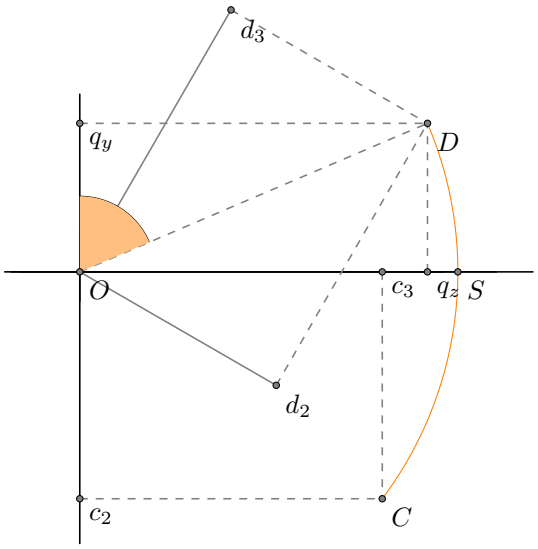

Figure 3: Restricting attention to the yz-plane, an arbitrary point, C, with y-component c_2 and z-component c_3 is tilted by ϕ degrees forwards about the origin, O. This is the point D. Projected onto the original axes, D has y-component q_y and z-component q_z. The orange shaded angle is the angle χ, defined below.

We have the following results. Again, calculating the length OD is straightforward.

$$s \equiv OD = (c_2^2 + c_3^2)^{\frac{1}{2}} \tag{2.2.1}$$

As for r, s is also restricted to be positive.

Then, adapting (2.1.4) to the problem, we define the angle $\chi \equiv \angle Y^+OD$, shaded orange in Figure 3.

$$\chi = \begin{cases} -\phi + \tan^{-1}\left(\frac{|c_2|}{|c_3|}\right) + 90° \times \left(\frac{|\text{sgn}(c_3) - \text{sgn}(c_2)|}{2} + 2 \times \mathbb{1}[c_3 < 0] + \frac{1}{2} \times [c_2 = 0]\right) & \text{if } c_3 \neq 0, \\ -\phi + 180° \times \mathbb{1}[c_2 < 0] & \text{if } c_3 = 0. \end{cases} \tag{2.2.2}$$

Accordingly, from (2.2.1) and (2.2.2), we obtain (2.2.3) and (2.2.4).

$$q_y = s\cos(\chi) \tag{2.2.3}$$

$$q_z = s\sin(\chi) \tag{2.2.4}$$

Again, q_y and q_z are functions of ϕ, c_2, c_3 that can be combined to make the vector valued function $\ell : \mathbb{R}^3 \to \mathbb{R}^3$.

$$\ell(x, y, z) \equiv (x, q_y(y, z; \phi), q_z(y, z; \phi))'$$

This completes our analysis of the effect of tilt. We have defined functions g and ℓ that independently map vectors to their new positions following rotation and tilt, respectively. We must now investigate their combined effect.

2.3 Rotation and tilt

As previously stated, the function f is a composite function. In fact, it is $f = \ell \circ g$. Suppose that we have any vector $(x, y, z)'$ and assign particular values to θ and ϕ. Then,

$$\begin{aligned} f(x, y, z; \theta, \phi) &= \ell(g(x, y, z; \theta); \phi) \\ &= \ell(p_x(x, z; \theta), y, p_z(x, z; \theta); \phi) \\ &= (p_x(x; \theta), q_y(y, p_z(x, z; \theta); \phi), q_z(y, p_z(z; \theta); \phi))' \end{aligned}$$

is the original vector having been rotated and tilted. But then recall from the initial discussion that we must apply the projection matrix $\mathbf{P}_0$ to this vector such that we map the vector from $\mathbb{R}^3$ space to $\mathbb{R}^2$ space.

$$\mathbf{P}_0 f(x, y, z; \theta, \phi) = (p_x(x; \theta), q_y(y, p_z(x, z; \theta); \phi))' \tag{2.3.1}$$

We can now use (2.3.1) to calculate the vectors along which the axes must lie. This is somewhat trivial since the axes originally lie on mutually orthogonal unit vectors. Let $\mathbf{x}_1, \mathbf{y}_1, \mathbf{z}_1$ be the directional vectors of the axes in $\mathbb{R}^2$ space following the change of orientation. The procedure for computing $\mathbf{x}_1$ will be shown in detail with results for $\mathbf{y}_1$ and $\mathbf{z}_1$ given.

Let us calculate $\mathbf{x}_1$. We know that the x-axis originally lies on the vector $(1, 0, 0)'$. As such, $r = 1$ and $\omega = \theta + 90°$ so we have that

$$\begin{aligned} p_x &= \sin(\theta + 90°) > 0 \text{ and} \\ p_z &= \cos(\theta + 90°) < 0, \end{aligned}$$

meaning that $s = |\cos(\theta + 90°)|$ and $\chi = -\phi + 270°$. Since we are projecting perpendicular to the original xy-plane, the z-component of the rotated and tilted vector is irrelevant.

$$q_y = |\cos(\theta + 90°)| \times \cos(-\phi + 270°)$$

Hence, $\mathbf{x}_1$ lies on the vector $(\sin(\theta + 90°), |\cos(\theta + 90°)| \times \cos(-\phi + 270°))'$.

$$(\mathbf{x}_1, \mathbf{y}_1, \mathbf{z}_1) = \begin{bmatrix} \sin(\theta + 90°) & 0 & \sin(\theta) \\ |\cos(\theta + 90°)| \times \cos(-\phi + 270°) & \cos(-\phi) & |\cos(\theta)| \times \cos(-\phi + 90°) \end{bmatrix} \tag{2.3.2}$$

The orientation of the axes after the change of orientation are shown in Figure 4, below.

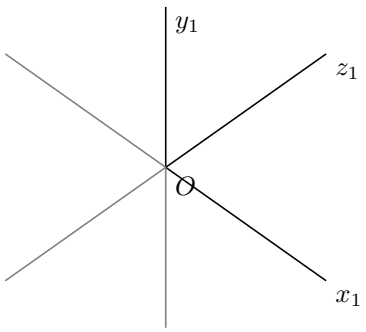

Figure 4: The orientation of the axes after rotation and tilt when projected from $\mathbb{R}^3$ space to $\mathbb{R}^2$ space for $\theta = \phi = 45°$.

3 Breaking down the problem

We return now to the crux of the problem: drawing tiles in LaTeX. As touched upon, drawing a single tile can be tricky enough, let alone drawing many tiles. We need to devise a means of breaking down the problem such that the addition of an extra tile amounts to an additional line of code, rather than many lines. To illustrate this point using the `tkz-euclide` package, suppose that we draw a simple unit square.

```
\begin{tikzpicture}
\tkzDefPoint(0,0){A}
\tkzDefPoint(1,0){B}
\tkzDefPoint(1,1){C}
\tkzDefPoint(0,1){D}
\tkzDrawSegments(A,B B,C C,D D,A)
\end{tikzpicture}
```

Despite the simplicity of the problem, we have had to define the coordinates of each of the four points and instruct the package to draw lines joining each of the points. Furthermore, we have had to input the exact coordinates of the four points. When the expressions determining these points are more complicated, manually computing each of these points is impractical.

Rather than attack the problem of drawing multiple tiles straight away we will consider ways to simplify the problem of drawing the unit square. The methods we develop to tackle this simple problem can then be adapted and applied to the tiling problem.

3.1 A simple problem

If we want to draw multiple unit squares at different points in $\mathbb{R}^2$ space, following the template above, will require that we write five additional lines for each unit square and compute four different points. However, each unit square shares a common feature. Namely, that the points B, C, D and their counterparts in other squares can be expressed as shifts from the point A.

```
\begin{tikzpicture}
\tkzDefPoint(0,0){A}
\tkzDefShiftPoint[A](1,0){B}
\tkzDefShiftPoint[A](1,1){C}
\tkzDefShiftPoint[A](0,1){D}
\tkzDrawSegments(A,B B,C C,D D,A)
\end{tikzpicture}
```

Although there are still five additional lines for each unit square we now only need to compute the location of one of the points, A in this example. Since A, B, C, D are generic labels for our points we could move to a more sensible labelling convention. For example, let a be our absolute point and a_1, a_2, a_3 be the relative points defined in terms of a. Our code now looks as though it could be repetitive: defining a sequence of absolute points $a, b, c, ...$ all other lines of code follow a common structure.

```
\begin{tikzpicture}
\tkzDefPoint(0,0){a}
\tkzDefShiftPoint[a](1,0){a_1}
\tkzDefShiftPoint[a](1,1){a_2}
\tkzDefShiftPoint[a](0,1){a_3}
\tkzDrawSegments(a,a_1 a_1,a_2 a_2,a_3 a_3,a)

\tkzDefPoint(2,2){b}
\tkzDefShiftPoint[b](1,0){b_1}
\tkzDefShiftPoint[b](1,1){b_2}
\tkzDefShiftPoint[b](0,1){b_3}
\tkzDrawSegments(b,b_1 b_1,b_2 b_2,b_3 b_3,b)
\end{tikzpicture}
```

As written, the only differences between the code for the unit square with absolute point a and the unit square with absolute point b are the labels a, b and the Cartesian coordinates of a, b.

Macros are incredibly useful tools that can be used to simplify repetitive tasks in which only the values of a small number of arguments vary. Here we could develop a macro that takes a point, say a, as its argument and then draws a unit square. Let us call this macro `unitsquare`.

```
\newcommand{\unitsquare}[1]{
\tkzDefShiftPoint[#1](1,0){#1_1}
\tkzDefShiftPoint[#1](1,1){#1_2}
\tkzDefShiftPoint[#1](0,1){#1_3}
\tkzDrawSegments(#1,#1_1 #1_1,#1_2 #1_2,#1_3 #1_3,#1)
}
```

Then the ten lines of code required to draw a unit square at $(0, 0)$ and another at $(2, 2)$ reduces to just four lines of code. Only the Cartesian coordinates of a and b need be calculated.

```
\begin{tikzpicture}
\tkzDefPoint(0,0){a}
\tkzDefPoint(2,2){b}
\unitsquare{a}
\unitsquare{b}
\end{tikzpicture}
```

In summary, we have noticed that the unit square has a common structure. That is, the length and direction of the vector joining any two points on the same unit square is common to all unit squares. All that changes across unit squares is the location of the square. This is the same as the location of any one point being different across unit squares. With this knowledge to hand, we built a macro that drew a unit square given a specified south-west corner.

3.2 Repetition in the tiling problem

If we can find common features across tiles then the problem of drawing multiple tiles will be considerably simpler. Notice that there are only three types of tiles. Returning to the example of the *Rubik's cube*, each face of the cube has a parallel face. For example, the leading face is parallel

to the face at the back, differing only in how far away the face is from us. We refer to the three types of tiles as xy, xz or yz tiles. Each type of tile is orthogonal to one of the axes; namely, each tile is orthogonal to the axis of the direction omitted from their name. Note that the location of any two tiles of the same type can differ in any direction.

Given three types of tiles, each with a common structure, we could theoretically build three macros that simplify the task of placing tiles. In fact, the macros will look very similar to the `unitsquare` macro that we built above.

Let us think about each type of tile in $\mathbb{R}^3$ space before we change the orientation of the space. Without loss of generality, let us consider a xy tile. Call A the south-west corner of the tile. Then the south-east corner of the tile is simply $A + (1, 0, 0)'$; the north-east corner of the tile is simply $A + (1, 1, 0)'$; the north-west corner of the tile is simply $A + (0, 1, 0)'$. Notice that the north-east corner can alternatively be expressed as $A + (1, 0, 0)' + (0, 1, 0)'$. We can derive similar expressions for the four corners of xz and yz tiles.

Changing the orientation of the space and projecting onto $\mathbb{R}^2$ space would appear, at first glance, to create some difficulty. However, notice that we have already computed the unit vectors for our projection when we calculated the vectors along which the axes lie, (2.3.2). The problem reduces to building the three macros and calculating a sequence of absolute points from which to plot our tiling arrangement.

Throughout this section we will set $\theta = 45°$ and $\phi = 35.26439°$. This is in order for the angles between each of the axes, as they appear on paper, to be 60°. For further simplicity, assume that tiles are restricted to lie on a Cartesian grid. This will restrict the location of tiles and, conveniently, means that any point can be reached from the origin using just the unit vectors, (2.3.2).

First, we need to compute the values of (2.3.2) given our choice of θ, ϕ.

$$\begin{bmatrix} 0.7071068 & 0.0000000 & 0.7071068 \\ -0.4082483 & 0.8164966 & 0.4082483 \end{bmatrix} \tag{3.2.1}$$

Now with these values, we can build the macros that we will rely on to construct any tiling arrangement of our choice.

```
%xy tile
\newcommand{\xytile}[1]{
\tkzDefShiftPoint[#1](0.70710678,-0.4082483){#1_1}
\tkzDefShiftPoint[#1_1](0,0.8164966){#1_2}
\tkzDefShiftPoint[#1](0,0.8164966){#1_3}
\tkzDrawSegments(#1,#1_1 #1_1,#1_2 #1_2,#1_3 #1_3,#1)
\tkzFillPolygon[color=gray!30](#1,#1_1,#1_2,#1_3)
}

%xz tile
\newcommand{\xztile}[1]{
\tkzDefShiftPoint[#1](0.70710678,-0.4082483){#1_1}
\tkzDefShiftPoint[#1_1](0.7071068,0.4082483){#1_2}
\tkzDefShiftPoint[#1](0.7071068,0.4082483){#1_3}
\tkzDrawSegments(#1,#1_1 #1_1,#1_2 #1_2,#1_3 #1_3,#1)
\tkzFillPolygon[color=gray!30](#1,#1_1,#1_2,#1_3)
}

%yz tile
\newcommand{\yztile}[1]{
\tkzDefShiftPoint[#1](0.7071068,0.4082483){#1_1}
\tkzDefShiftPoint[#1_1](0,0.8164966){#1_2}
```

```
\tkzDefShiftPoint[#1](0,0.8164966){#1_3}
\tkzDrawSegments(#1,#1_1 #1_1,#1_2 #1_2,#1_3 #1_3,#1)
\tkzFillPolygon[color=gray!30](#1,#1_1,#1_2,#1_3)
}
```

The `tkzFillPolygon` command is included so that tiles are shaded grey. It is not strictly necessary but is included for aesthetic purposes.

Suppose that we wish to construct the following arrangement, Figure 5.

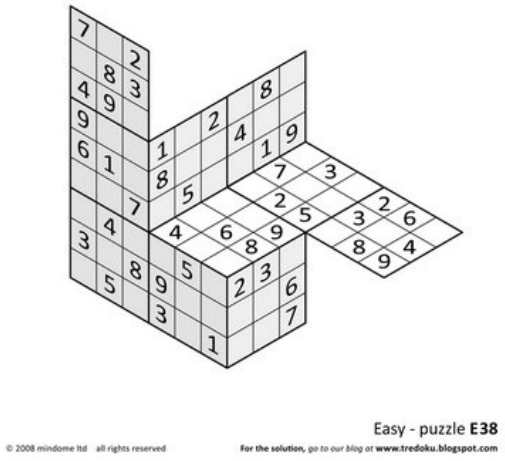

Figure 5: A *tredoku* puzzle found using a Google search, keyword: *tredoku*.

The code to replicate this arrangement is given below. Note that the lines before the first linebreak and after the second linebreak are necessary to draw in the axes and do not form part of the problem *per se*.

```
\begin{tikzpicture}
\tkzInit
\tkzDefPoint(0,0){O}
\tkzDefPoint(2.828427,-1.632993){x}
\tkzDefPoint(0,3.265986){y}
\tkzDefPoint(2.828427,1.632993){z}
\tkzDefPointBy[symmetry= center O](x) \tkzGetPoint{x_1}
\tkzDefPointBy[symmetry= center O](y) \tkzGetPoint{y_1}
\tkzDefPointBy[symmetry= center O](z) \tkzGetPoint{z_1}

\tkzSetUpLine[color=orange]
\tkzDefPoint(0,0.8164966){A}
\tkzDefPoint(0,1.632993){B}
\tkzDefPoint(0.7071068,-0.4082483){C}
\xytile{O}
\xytile{A}
\xytile{B}
\xytile{C}
\tkzDefPoint(0.7071068,0.4082483){D}
```

```
\tkzDefPoint(1.4142136,0.8164966){E}
\tkzDefPoint(2.1213204,0.4082483){F}
\xztile{D}
\xztile{E}
\xztile{F}
\tkzDefPoint(1.4142136,-0.8164966){G}
\yztile{D}
\yztile{E}
\yztile{G}

\tkzSetUpLine[color=black]
\tkzDrawSegments[delta=10](x,O y,O z,O)
\tkzSetUpLine[color=gray]
\tkzDrawSegments[delta=10](x_1,O y_1,O z_1,O)
\tkzLabelPoints(O,x,y,z)
\end{tikzpicture}
```

The code produces the graphic in FIGURE 6. This is identical to the tiling arrangement in the *tredoku* puzzle we sought to replicate. Particularly pleasing is the fact that all of the absolute points could be found within a minute simply by working out the coordinates of the south-west corners of each tile in $\mathbb{R}^3$ and premultiplying this coordinate vector by the matrix (3.2.1) using any statistical software package.

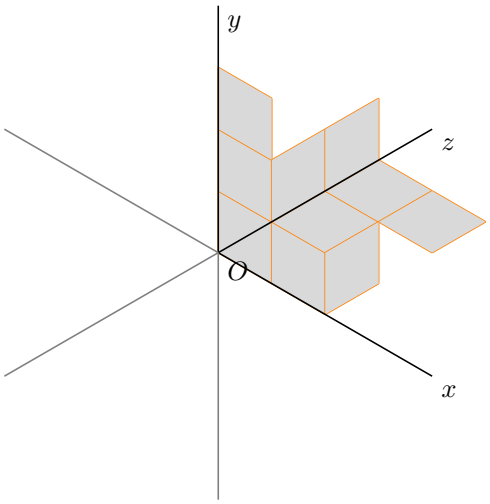

Figure 6: The *tredoku* puzzle in FIGURE 5 is replicated using the macros and tools built and derived above.