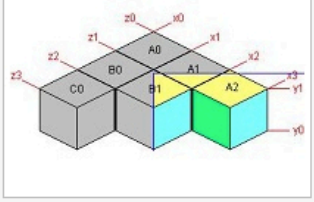


Isometric Projection

Abstract. This page provides the theoretical and technical background of projecting obtained elevation data isometrically.

Series. This article is part of a series. To navigate within the series, see the heading *Series Navigation* in the menubox to the left.

This Site

Index to locate pages by Keyword.

Sitemap to access pages by their hierarchical organization.

This Document

Path

Home ▸ Articles ▸ Digital Terrain Analysis ▸ Isometric Projection

Series Navigation

Digital Elevation Models
Isometric Tiling

Author

Herbert Glarner

Published

2007-Mar-19: Original HTML version
2011-Jan-24: Recoded in XLM

Further Reading

- W Isometric Projection on Wikipedia
- Axonometric Projections : a Technical Overview by Thiadmer Riemersma
- Articles about isometric projections on GameDev.net

Copyright and License

Introduction

Isometric Projection

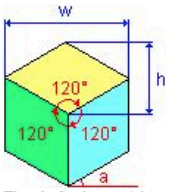


Fig. 1: An isometric cube

An isometric projection is defined by the property, that all the axes have the same metric (isometric, Greek: "equal measure"). Let's draw a cube to visualize what this means (fig. 1).

All the cube's edges have the same length. Furthermore, because the angles between sides all are `120°`, all sides are symmetric W rhombi, meaning that the surface of each side is equal. Also note, that the perimeter of the shown cube is a perfect W hexagon.

It is easy to see, that the angles at the other side of the edges measure `180°-120°=60°`. Also, such an angle plus `a` make a right angle, `a=90°-60°=30°`.

But the angle `a` also is defined by `a=arctan(h/w)=30°` (or, more accurately, to reflect the involved triangle, by `a=arctan((h/2)/(w/2))=30°`).

For the relationship between `h` and `w` therefore must be true, that `h/w=sin(30°)/cos(30°)=tan(30°)=0.57735`, or probably more conveniently, `h/w=1/sqrt(3)`.

Dimetric Projection


Fig. 2: A magnified 30° line

However, in computer graphics this real isometry is not truely liked. The reason for this becomes apparent, when we magnify one of the surface edges in the x/z-plane (fig. 2): The lines just don't look good.

30°-lines appear blocky and unsmooth. Mainly for this reason, it is usual in computer graphics to not use real isometric projections and apply lines which truely look straight instead (fig. 3), even if that means to depart from the nice 30°, 60° and 120° angles.


Fig. 3: A magnified straight line

This line grows twice as fast horizontally than it does vertically. Therefore its angle `a` can be determined by `a=arctan(1/2)=26.565°`.

Because of this, our `120°` angles of the original cube need to be adjusted as well. They become `90°+26.565°=116.565°` and `360°-2×26.565°=126.87°` (fig. 4).

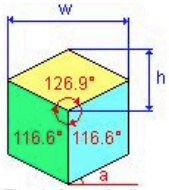


Fig. 4: An optically more appealing cube

Technically, the departure from a representation with three equal 120° angles makes our rendering no longer an isometric projection: the accurate term for this kind of drawings is W dimetric projection. However, for convenience, we will continue to refer to the term "isometric projection" in this article.

Because the angles changed, `h` changed as well. For the relationship between `h` and `w` now is true, that `h/w=sin(26.565°)/cos(26.565°)=tan(26.565°)=0.5`, which is not surprising because we defined the line that way.

There's another and more important reason to apply the 2:1 ratio besides the one dealing with the optical appearance, though.

It makes the design of structures parallel to the ground plane a lot easier and oftentimes saves much time in calculating the proper lengths without really affecting the optical impression too much.

Surfaces

Applying Isometry

Imagine a small landscape subdivided into 4×4 regular squares. Let's label its rows A to D and its columns 1 to 4. When we look at this small landscape of ours from above it has the following appearance (fig. 5):

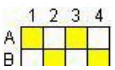


Fig. 5: Landscape looked at from above

After applying the principles of isometric rendering as outlined above, our landscape will look like this (fig. 6):

Fig. 6: The same landscape rendered isometrically

Spotlight on Tiles

This type of rendering has some issues, though. They can be revealed if we look at a magnified tile. Fig. 7 shows an individual tile from our landscape:

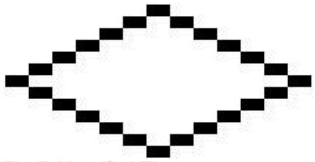


Fig. 7: Magnified Tile

The problem becomes apparent when we attempt to connect such tiles. We can not simply place them next to each other: it is absolutely impossible to tile them, no matter how hard we try (fig. 8).

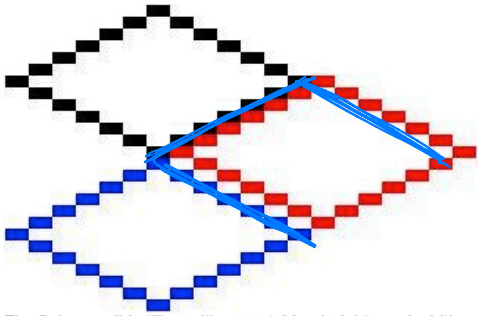


Fig. 8: Impossible tiling with unmatching heights and widths

Notice, however, what happens when we shift the tiles two pixels to the left and one pixel upwards: the two adjacent edges of any two tiles are shared by each other (fig. 9).

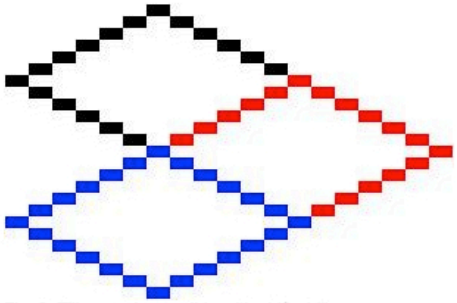


Fig. 9: Tiling made possible with shifted tiles

Alas, it is not really possible to share an edge between different tiles: even if we drew the tiles this way, it still would be true that a pixel can only be displayed once (although it could be drawn multiple times in the same screen location). For this reason we declare, that a tile's lower edges (i.e. the SW edge and the SE edge) end already with the pixel just above the so far common edge (fig. 10). Notice, that the green edges then do not any longer form part of our 3 tiles: they indicate the top (NW and NE) edges of the next adjacent tiles.

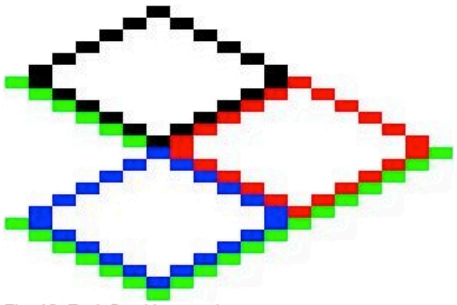


Fig. 10: Redefined lower edges

That said, it is easy to see what our isometric tiles eventually must look like (fig. 11): whatever edge length you ultimately need to represent your grid, just reduce it by a two pixel wide element and duplicate the Western and Eastern corner heights.

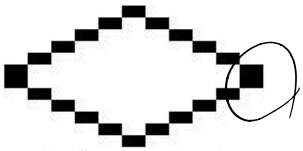


Fig. 11: Effective tile shape to allow tiling

Irregular Tile Properties

There are a few more things to keep in mind when working with tiles.

First of all, remember that the lower edges of the tiles are not treated as a part of that tile any longer: although they still do exist, they are rendered by the adjacent SW and SE tiles, which share those edges with our tile in question. Therefore, when it is desired to draw outlines as we did in the initial example, then only the two upper edges should be drawn (the black ones in fig. 12):

Fig. 12: Outlining edges

Secondly, since the perceived lower edges in fact are not yet the real lower edges, care must be taken when the center point of a tile is of importance: the middle of the edge's length is not just the middle element of the two pixel wide elements which we see (in our example there are 6 such elements, see fig. 13), but the middle element of those elements plus 1, because the shared lower edge must be included (i.e. 7 elements in our example, making the 4th the middle element). Note, that the lower end points of the middle lines do not fall on a 2-pixel-element of the (apparent) lower edges, simply because they are not the lower edges yet.

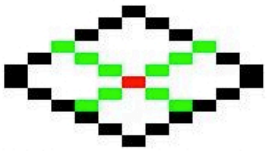


Fig. 13: Finding a tile's center

Deriving the Coordinates

Before we go one step further and seriously deal with the third dimension (elevation), let's have a look at our cube again and properly define the edges' coordinates for the representation of its surface area:

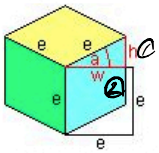


Fig. 14: The dimensions of the cube

All the edges labeled `e` have the same length.

Now, we can let `e` be whatever we like: 1 centimeter, 1 meter or 1 mile. The important thing to remember is just, that our isometric cube has equal lengths at any edge. This is important, because it also says, that the effective horizontal and vertical coordinates determining the edges' corners are *not* equal to `e`. In fact, the position of the edges' corners in the x/z plane are calculated by foreshortening them: for `w` is valid, that `w=cos(a)=cos(26.565°)=0.8944e`, and for `h` is true, that `h=sin(a)=sin(26.565°)=0.4472e`.

Notice, however, that `2h=w` is still valid, and also notice, that the upright edge denoting the elevation retains its length `e`: no need to shorten it or to modify it in any way.

Now imagine several such cubes attached to each other. Let's define, that the x axis runs along the NE edges and the z axis along the NW edges. Labeling the "rows" with capital letters and the "columns" with Roman numbers, we are able to name the individual cubes. Fig. 15 shows the identifiable coordinates:

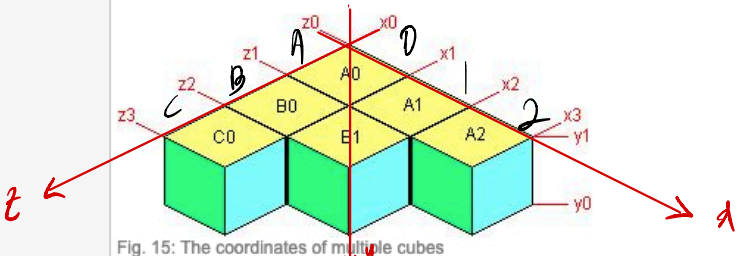


Fig. 15: The coordinates of multiple cubes

Still assuming an edge length of `e`, we can now attempt to localize the 2D coordinates for our cubes, i.e. the coordinates on the monitor's screen when those cubes would be rendered onto a such. To avoid a terminological confusion, let's refer to the screen axes simply as the horizontal and vertical 2D

axes for now.

Let's have a look at the cubes' x axis (the NE edges) first. The first observation is, that the more to the right a cube is located, the more increases its value on the horizontal 2D axis. Specifically does the next cube to the right increase its horizontal value by $0.8944e$ (the cosine of 26.565°) relative to its left neighbor. And there's a second observation to make: also the vertical 2D axis is affected, even when we stay within the same row. Each neighbor to the right increases the vertical value by $0.4472e$ (the sine of 26.565°) relative to its left neighbor.

The y axis (NW edges) is very similar in its behavior: both the horizontal and vertical 2D axes change when we move one cube downward. For each row we go down, the vertical 2D value increases $0.4472e$, and simultaneously the horizontal 2D value decreases by $0.8944e$.

There's a third axis, though, which we silently ignored until now. In fact, so far it is not at all an interesting axis, as it stays the same all the time, namely e. Still, it can be observed, that it does have an influence on the coordinates which are renderened to our screen: the (yellow) surface is "elevated" by e, i.e. the screen's vertical axis decreases by e. It does not require much imagination to see, that this value influences only the vertical 2D axis: the horizontal 2D axis is not affected at all.

Let's try to fill in the two-dimensional screen coordinates into a table. Let's refer to the screen coordinates as x_{Sn} for the horizontal axis, and and y_{Sn} for the vertical axis from now on (the subscript S stands for *Screen*, and n simply enumerates the coordinate from left to right, and top down, resp.) Accordingly we will refer to the three-dimensional "world coordinates" as x_{Wn} ("NE edges"), z_{Wn} ("NW edges"), and y_{Wn} (the elevation). The subscript W stands for *World* and distinguishes this set of coordinates from the 2D-coordinates. Be careful to not to confuse the two y-coordinates: they signify different things!

We recall the following observations:

- A neighbor to the right (increasing x_W):
  - increases x_S by $\cos(a)$
  - increases y_S by $\sin(a)$
- A neighbor below (increasing z_W):
  - *decreases* x_S by $\cos(a)$
  - increases y_S by $\sin(a)$
- The elevation y_W
  - *decreases* y_S by its true value
  - does not affect x_S

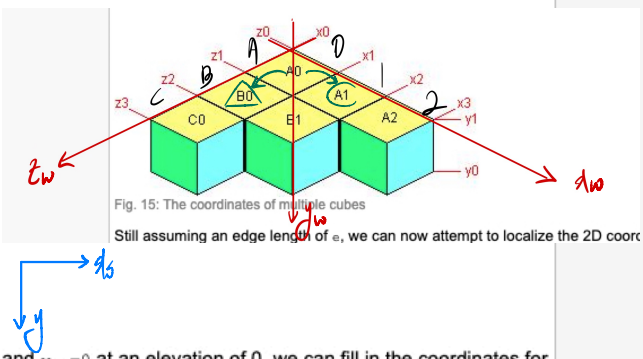


Fig. 15: The coordinates of multiple cubes

Still assuming an edge length of e, we can now attempt to localize the 2D coord

Assuming a coordinate origin of $x_{S0}=0$ and $y_{S0}=0$ at an elevation of 0, we can fill in the coordinates for fig. 15 into the table as follows: (Hint: observe the coloring of the coefficients.)

	x_{W0}	x_{W1}	x_{W2}	x_{W3}
z_{W0}	$x_S=0\times\cos(a)s-0\times\cos(a)s$ $y_S=0\times\sin(a)s+0\times\sin(a)s-1\times s$	$x_S=1\times\cos(a)s-0\times\cos(a)s$ $y_S=1\times\sin(a)s+0\times\sin(a)s-1\times s$	$x_S=2\times\cos(a)s-0\times\cos(a)s$ $y_S=2\times\sin(a)s+0\times\sin(a)s-1\times s$	$x_S=3\times\cos(a)s-0\times\cos(a)s$ $y_S=3\times\sin(a)s+0\times\sin(a)s-1\times s$
z_{W1}	$x_S=0\times\cos(a)s-1\times\cos(a)s$ $y_S=0\times\sin(a)s+1\times\sin(a)s-1\times s$	$x_S=1\times\cos(a)s-1\times\cos(a)s$ $y_S=1\times\sin(a)s+1\times\sin(a)s-1\times s$	$x_S=2\times\cos(a)s-1\times\cos(a)s$ $y_S=2\times\sin(a)s+1\times\sin(a)s-1\times s$	
z_{W2}	$x_S=0\times\cos(a)s-2\times\cos(a)s$ $y_S=0\times\sin(a)s+2\times\sin(a)s-1\times s$	$x_S=1\times\cos(a)s-2\times\cos(a)s$ $y_S=1\times\sin(a)s+2\times\sin(a)s-1\times s$		
z_{W3}	$x_S=0\times\cos(a)s-3\times\cos(a)s$ $y_S=0\times\sin(a)s+3\times\sin(a)s-1\times s$			

It can be seen, that the red coefficients repeat the index of x_{Wn} and the green coefficients the index of z_{Wn}. The blue coefficients are just the elevation at that coordinate: it is 1 everywhere, because our cubes all have an elevation of 1×s.

Generalizing the Coordinates

The above exercise immediately leads to a generalization on how to obtain screen formulas from world coordinates:

$$x_S = x_W\times\cos(a)s - z_W\times\cos(a)s = (x_W-z_W)\cos(a)s$$

$$y_S = x_W\times\sin(a)s + z_W\times\sin(a)s - y_W\times s = (x_W+z_W)\sin(a)s - y_W\times s = ((x_W+z_W)\sin(a)-y_W)s$$

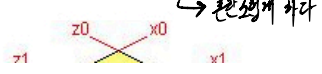
Examples

As a first example let's calculate the screen coordinates of the topmost corner of the cube labeled "C0" in fig. 15. The topmost corner of C0 is at world coordinates $x_W=0$ and $z_W=2$. Its elevation is $y_W=1$ (expressed in s). Let's assume, that $s=1$ (meters, if you like).

$$x_S = (x_W-z_W)\cos(a)s = (0-2)\times0.8944\times1 = -2\times0.8944\times1 = -1.7888$$

$$y_S = ((x_W+z_W)\sin(a)-y_W)s = ((0+2)\times0.4472-1)\times1 = (2\times0.4472-1)\times1 = -0.1056$$

Whereas the value x_S immediately seems to be plausible, the value y_S might call for some additional elaboration: why is the value negative, when it clearly is more South than x_{W0}, z_{W0}? Well, we need to recall, that our origin is not at an elevation of $1\times s$, but at the base elevation 0. The following fig. 16 (in which we leave out the cubes obfuscating the whole picture and additionally render the cube A0 as a wire-frame model) helps to clarify the concept:



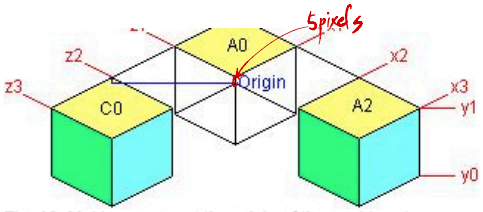


Fig. 16: Making apparent the origin of the world system

With the origin pointed out (labeled "Origin", at x_{W0}, z_{W0}, but at base level 0) it immediately becomes obvious (by following the blue horizontal to the left and then upward to our target corner), that our example point indeed lies above (i.e. to the "North") of the origin. Not much above, though, but after all, `-0.1056` is not that much either.

As a second example let's look at the coordinate $x_W=1$ and $z_W=1$. Its elevation is $y_W=1$ (expressed in s). This time, we set `s=47` (length of an edge in pixels).

$x_S = (x_W-z_W)\cos(a)s = (1-1)\times0.8944\times47 = 0\times0.8944\times47 =$ `0 pixels`

$y_S = ((x_W+z_W)\sin(a)-y_W)s = ((1+1)\times0.4472-1)\times47 = (2\times0.4472-1)\times47 =$ `-4.9632 pixels`

5 pixels above the origin seems about right.

Rendering Flat Surfaces

Screen Center and World Focus

Our formulas work well, but in our previous examples we had to deal with negative numbers. Of course, there is nothing wrong with negative numbers per se, but because we were claiming to calculate screen coordinates, we nevertheless would face some difficulties in applying our results. After all, we can not render a pixel at the coordinate `x=0` and `y=-4.9632`, because such a point would lie above the screen's upper edge.

The problem is due to choice of the origin. In our previous examples, this was the base elevation of the topmost point of the cube A0. Being the origin inherently has the notion of being located at *screen* coordinates `x=0` and `y=0`.

This is fine, but it has the drawback, that only a quarter of all world coordinates (more precisely, a quarter of the points at base level) can be transformed into screen coordinates.

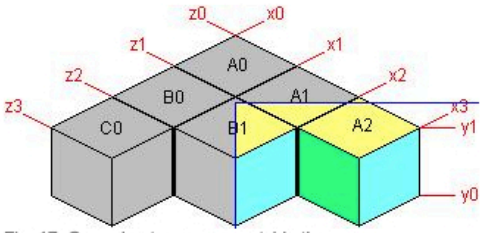


Fig. 17: Greyed out areas are outside the screen

What we need to do is to define a center point in the world system. That center point is to become the center of the screen, and relative to that point all the other transformations from the world coordinate system to the screen coordinate system are going to be calculated.

Whereas it is no problem to pick such a point in the world coordinate system (it can be virtually any arbitrarily chosen point which we declare to be "the center point"), the definition of the center of the screen (to which the center point needs to be transformed) requires some elaboration. The complication which arises is due to the fact, that the screen has two dimensions with *precisely defined lenghts* (which is a good thing - otherwise we would not be able to calculate a center).

So let's focus to the "screen" for a moment. What exactly is the screen? The most appropriate answer might be: it depends what you want it to be. In essence it is just a two-dimensional rectangle into which you want to represent three-dimensional points applying our found formulae. This may be the whole display area which is surrounded by the physical ends of your device, or it may just be a rectangle of arbitrary size which you dedicate to your product (in most operating systems called a "Window"). All in all, however, it does not really matter: the only thing which matters is the *length of the two dimensions*: we need to know those in order to calculate a center point within your screen.

So let's, somewhat arbitrarily, define an example screen. Let's say, it has a width of $W_S=250$ `pixels` and a height of $H_S=100$ `pixels` (the subscript `S` stands for "Screen" again, `W` and `H` denote the Height and Width, resp.) Then it is easy to find the center of the screen: just divide the two lengths by 2 and you're done:

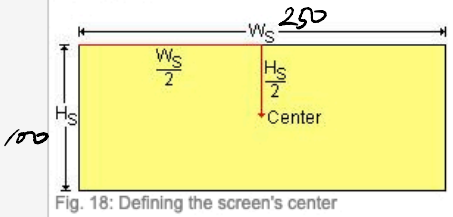
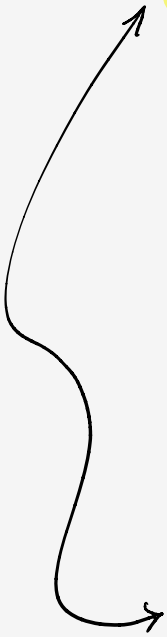


Fig. 18: Defining the screen's center

We will need these coordinates later on, therefore let's give them a dedicated name (whereby the subscript `C` refers to the screen's *center*):

x_C = ScreenWidth / 2

y_C = ScreenHeight / 2



To this Center we want to render an arbitrary point from our world coordinate system. Now, "arbitrary" does not mean, that it does not matter what point we choose: it does matter, because this is the point which an observer of the screen focusses instinctively, virtually perceiving it as the "center of the world" or, maybe better, the "relative center of his world around which the whole world revolves". Or, in yet other terms, the point, where "the action happens".

Therefore it is typical for isometric projections to associate this center with the observer (most prominently featured in computer games, where it is this point at which the player's avatar is located). This is an important property, because it implies, that although the screen's center is fix (at least as long as the dimensions of the screen themselves don't change), the world's center of interest is dynamic and may change (for example will our aforementioned avatar not stand still all the time, but move in one way or the other, thus relocating the world's center of interest while still being displayed at the screen's center).

The quintessence of this is, that it is the world which seems to move. This directly implies, that we need to manipulate our calculated x_S and y_S in one way or the other, such that the chosen point of interest lies directly on the screen's center.

Let's say, that we wanted the center of the top surface of cube B1 to be the world's center (for example because a player's avatar is standing on this particular spot). Let's call this point *Focus* from now on to clearly distinguish it from the term *Center*, which we will exclusively use when we mean the screen's center.

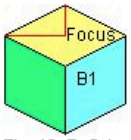


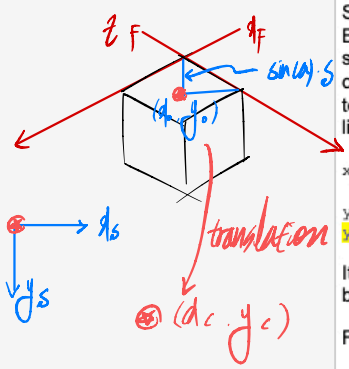
Fig. 19: Defining the world's focus

To match the (world's) Focus with the (screen's) Center is a 2-step process. Firstly, we need to calculate the focus' coordinates within the world coordinate system (this is, because the center of the top surface area is not a given point already: only the four corners of the surface are known à priori). Secondly, we need to "move" this point in some way, so that its coordinates eventually match the Center (this second operation, known as a Translation, will turn out to consist of two substeps, so that we might as well speak of a 3-way process).

First things first. The calculation of the Focus might appear to be a trivial operation at first glance, but I can assure you, that this impression is false. It looks such trivial just because the surface of our cube is "flat", i.e., all corners have the same elevation and thus the plane is parallel to "ground zero". Things will turn out to be slightly more complicated, when the corners do not all have this facilitating property. In particular will we not be able anymore to treat the surface as a rhombus like we are doing for now, but need to look at it as compound triangles. More of that later, though.

When the surface is flat, the Focus' coordinates can easily be calculated: just take the topmost corner's coordinates and go downwards half the distance toward its opposite corner, or take the leftmost corner and go halfway toward that one's opposite corner. Putting these two options together, using one property of each, we could easily achieve our goal: just use the x_W coordinate of the topmost corner and the y_W coordinate of the leftmost corner and we have the desired coordinates of our Focus. Note, however, that doing so we would leave the isolated perspective of viewing at a single point for the first time: this operation involves two points (the two mentioned corners). This is necessary and unavoidable in many instances, but for the moment being we'd like to stick with the single-point view of things.

Well, of course, we know h already: we were doing calculations with it all the time: $h=\sin(a)s=\sin(26.565°)s=0.4472s$ (with s being the cube's edge length).



So, for the moment being, it suffices to add $\sin(a)s$ to the y_S coordinate of our generalized formula. Because the Focus' coordinates are important, as we will see, it makes sense to assign the resulting screen coordinates an own subscript O for *Origin*. Remember, however, that O is in the screen coordinate system, otherwise denoted with the subscript S. We also will define a subscript F for *Focus* to indicate, that we mean a special point within the world coordinate system. Then the formulae look like this:

$x_O = (x_F-z_F)\cos(a)s$ (unchanged)

$y_O = ((x_F+z_F)\sin(a)-y_F)s + \sin(a)s = ((x_F+z_F)\sin(a)-y_F+\sin(a))s = ((x_F+z_F+1)\sin(a)-y_F)s$

It turns out, that after factoring in the sine the formula is not really much more time-consuming than before: we just need to add 1 at the right place.

For the Focus of our cube's surface this results in:

$x_O = (x_F-z_F)\cos(a)s = (1-1)\times0.8944\times47 = 0$

$y_O = ((x_F+z_F+1)\sin(a)-y_F)s = ((1+1+1)\times0.4472-1)\times47 = (3\times0.4472-1)\times47 = 16.0552$

Now compare this with the coordinates we got in an earlier example for the topmost corner of our cube: they were $x_S=0$ and $y_S=-4.9632$: the Focus is $16.0552-(-4.9632)=21.0184$ pixels lower now, which unsurprisingly happens to be $h=\sin(a)\times s=0.4472\times47=21.0184$, exactly the component we plugged in.

Recall, that so far our formulas for x_S and y_S were relative to the base elevation of the uppermost corner of the first cube A0. Now we want them to be relative to the Focus at x_O and y_O. To make them relative to the Focus, we will need to subtract the Focus' coordinates (which are relative to A0 themselves) from the other points' screen coordinates. It should be clear immediately, that if we

subtract x_O and y_O from any world-to-screen calculation, we will end up with screen coordinates relative to the screen's origin. For example would the calculation of the Focus itself result in $x_O-x_O=0$ and $y_O-y_O=0$, which is the screen's origin.

And by now also the final step is fairly obvious: we don't want the Focus to be displayed at the screen's origin, but at its center, and so we need to add the screen coordinates x_C and y_C of the center on top of this all.

Therefore our final formulas will look like this (of course, here we did not factor in the sine as was done for the Focus, because we want to refer to the really mentioned points, and not to the center of an area surface; hence, the term $+1$ does not appear in y_S):

$$x_S = (x_W-z_W)\cos(a)s-x_O+x_C$$

$$y_S = ((x_W+z_W)\sin(a)-y_W)s-y_O+y_C$$

Translating World to Screen

Let's check how this all works out for the topmost corner of our singled out cube B1. The corner's coordinates are $x_W=1$ and $z_W=1$. Its elevation is $y_W=1$. First we do the preliminary tasks and calculate the (screen) *Center* (the screen still being assumed to have a width of 250 pixels and a height of 100 pixels) and the *Origin* as per our cube's *Focus* point:

$$x_C = ScreenWidth / 2 = 250 / 2 = 125$$

$$y_C = ScreenHeight / 2 = 100 / 2 = 50$$

$$x_O = (x_F-z_F)\cos(a)s = (1-1)\times0.8944\times47 = 0\times0.8944\times47 = 0$$

$$y_O = ((x_F+z_F+1)\sin(a)-y_F)s = ((1+1+1)\times0.4472-1)\times47 = (3\times0.4472-1)\times47 = 16.0552$$

Then we can calculate any desired point within the world coordinate system, for example the top corner of cube B1, by applying the formulas for x_S and y_S:

$$x_S = (x_W-z_W)\cos(a)s-x_O+x_C = (1-1)\times0.8944\times47-0+125 = 0\times0.8944\times47-0+125 = 0-0+125 = 125$$

$$y_S = ((x_W+z_W)\sin(a)-y_W)s-y_O+y_C = ((1+1)\times0.4472-1)\times47-16.0552+50 = (2\times0.4472-1)\times47-16.0552+50 = -4.9632-16.0552+50 = 28.9816$$

Because the topmost corner of the surface of cube B1 also is the bottommost corner of the surface of cube A0 (which latter happens to be our "Focus cube"), we expect this point to be $h = \sin(a)s$ above the screen's *Center*. Since $\sin(a)s = 0.4472\times47 = 21.0184$, and $100/2 - 21.0184 = 28.9816$, we can see that the formula holds.

Now let's apply the final formulas x_S and y_S to the coordinates of our 6 cubes once again, using the precalculated values x_C and y_C as well as x_O and y_O. The values were rounded to the next integer, because we calculate pixel positions:

	x_{W0}	x_{W1}	x_{W2}	x_{W3}
z_{W0}	x_S=125 y_S=-13	x_S=167 y_S=8	x_S=209 y_S=29	x_S=251 y_S=50
z_{W1}	x_S=83 y_S=8	x_S=125 y_S=29	x_S=167 y_S=50	
z_{W2}	x_S=41 y_S=29	x_S=83 y_S=50		
z_{W3}	x_S=-1 y_S=50			

Eventually, let's plot the calculated coordinates to our screen of dimensions 250×100 pixels. To see the whole picture, we connect the calculated points with straight lines (even those points which would lie outside the screen area, i.e. have a negative x or y value, and also those which are larger than the screen's dimensions). Additionally we single out the *Center* (representing the *Focus*):

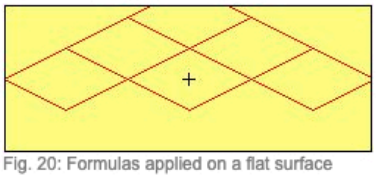


Fig. 20: Formulas applied on a flat surface

Redefining the Focus

By now we are able to move the "action center" from one spot to another simply by redefining the Focus, i.e. by recalculating the coordinates x_O and y_O. Let's say, that your application requires to jump to cube A2 from B1. The topmost corner of A2 has the world coordinates $x_W=2$=2 and $z_W=0$, and its elevation is $y_W=1$. Plugging in the according values we get:

$$x_O = (x_F-z_F)\cos(a)s = (2-0)\times0.8944\times47 = 2\times0.8944\times47 = 84.0736$$

$$y_O = (x_F+z_F+1)\sin(a)-y_F)s = ((2+0+1)\times0.4472-1)\times47 = (3\times0.4472-1)\times47 = 16.0552$$

It can be observed, that y_O did not change when compared with cube A1. This makes sense, because that cube's *Focus* optically is on the same height as A1's *Focus*. But, x_O did change by $84.0736-0 = 84.0736$ pixels, i.e. the *Focus* is 84 pixels more to the right now (which means, that the "world moves" to the left by that amount). Note, that $2\times w = 2\times\cos(a)s = 2\times0.8944\times47 = 84.0736$, which is exactly what we got.

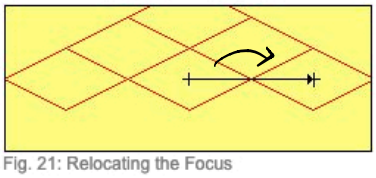


Fig. 21: Relocating the Focus

The screen's *Center* does not change, of course (unless we also changed the screen's dimensions). Therefore the only remaining task is to recalculate the new coordinates x_S and y_S for all world points in order to get the following representation:

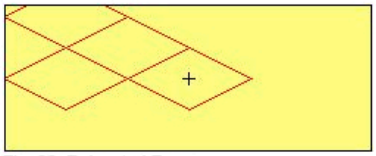


Fig. 22: Relocated Focus

Note, that in a real application we probably would not want to just "jump" to a new *Focus*, but scroll smoothly from one location to another. Also, we wouldn't recalculate points which are both visible before and after the transition, we would merely move them to their new position, only calculating new points as they are "moved in" at the according edges.

Redefining the Center

From time to time the need may arise to redefine the screen's *Center* as well. This is needed, when the screen's dimensions are changed (for example, when the user resizes the window in which your application renders the world).

Let's assume, that your user wishes to make the window taller and resizes its height from 100 to 150 pixels. Although he most likely will do so by modifying either the window's top or bottom edge (but not both at the same time), the impact is such, that the height difference is applied to both the top and bottom edges simultaneously, half the height difference at each side: this behaviour keeps the *Center* centered.

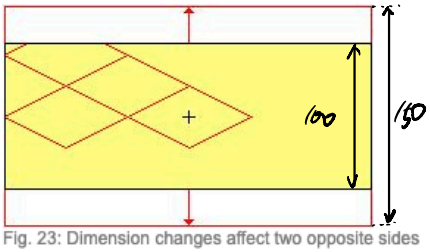


Fig. 23: Dimension changes affect two opposite sides

The changed dimensions lead to a recalculation of the x_C and y_C coordinates:

x_C = `ScreenWidth / 2 = 250 / 2 = 125`

y_C = `ScreenHeight / 2 = 150 / 2 = 75`

Recalculating all coordinates x_S and y_S for all world points based on fig. 22 then leads to the following output:

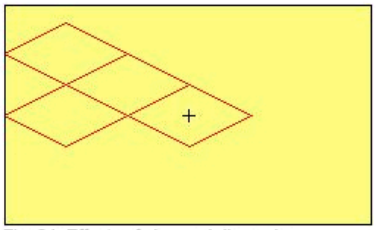


Fig. 24: Effects of changed dimensions

Also note, that oftentimes it is not necessary to really recalculate all the world coordinates: the points which are already present usually would simply be moved to the new center by shifting them horizontally or vertically as required (in our case 25 pixels downward). Then we only need to calculate the points which are in the newly exposed window parts (and even this part can be omitted if the dimension change results in a shrinking window).

Working with Elevations

Number of Elevation Points

So far we were working with flat surfaces, i.e., elevations of all points having the same height (namely $1 \times s$ in all our examples). Let's have a look, what will change when we apply some different elevations. "Applying elevation" is a somewhat fuzzy term, though: where exactly is "elevation" applied? Let's examine this some more in depth.

First of all there is a point of view that each point on the grid has a dedicated elevation. Let's call such

a point a grid point. Grid points are shared by usually 4 surrounding tiles (2 only if the grid point is along the whole landscape's edge, 1 if it's one of the landscape's corners). And then there's the possibility to define an individual elevation for each of all the tiles' 4 corners. Let's call those tile corners. The following pictures visualize the two points of view:

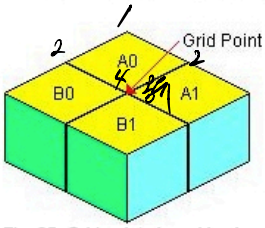


Fig. 25: Grid point shared by 4 tiles

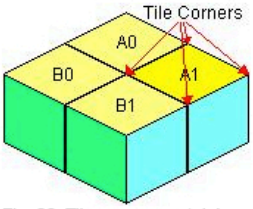


Fig. 26: Tile corners pertaining to a dedicated tile

These two different point of views both have their advantages and disadvantages. If we do work with grid points, the most obvious advantage is, that we only need to store a minimum amount of elevation data: in particular, storing 1 elevation per tile suffices, as the other 3 corners can be derived from the elevations of the 3 neighboring tiles (since they are shared points). The downside is, that with grid points we can not handle true vertical structures (e.g. cliffs): to define a true vertical structure at a given point in the x/z plane, we in one way or the other need to provide 2 different heights for the y dimension. The tile corners approach is one such way.

Catching up the previous statement that we "need to provide 2 different heights for the y dimension" could attempt one to think, that we do not really need to store an elevation for each corner of every tile, because this only would duplicate information available anyway from elsewhere. In fact, at first glance it seems to suffice to take the grid point approach, but to store 2 elevations for a single corner of each tile: a lower and an upper elevation. When there is no vertical structure at that point, then the two values will be identical, else their difference tells about the height of the vertical structure. Let's call those two elevations upper elevation and lower elevation. The following picture shows the two elevations for the topmost corner of cube B1 (and with that implicitely also the elevations of the according corners of the surrounding cubes):

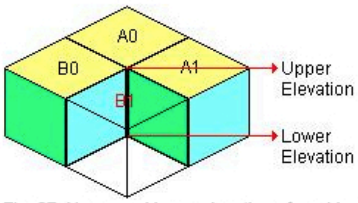


Fig. 27: Upper and lower elevation of a grid point

There's a major flaw with this approach, though: the interpretation is ambigeous. Let's define some elevations to make this clear:

	Topmost corner of		
	B0	B1	B2
Upper elevation	1	2	2
Lower elevation	1	1	2

It is clear, that B0 has an elevation of 1 and B2 a such of 2. But how shall B1 be interpreted? There are two possibilities. Let's connect the lines along the x axis:

Fig. 28: Ambigeous interpretation of elevations

The same ambiguity occurs along the `z` axis.

Therefore, if one needs to represent true vertical structures (such as cliffs in a landscape), there is no way around the tile corners approach (fig. 26).

However, note that there is not always the need to represent true vertical structures. This is particularly the case, when our DEM delivers just a single elevation for any point. Among the DEMs having this property belong all techniques, which obtain their data by measuring "as far as they can see" (i.e. the closest obstacle defines the elevation), but not beyond that point. A prominent example is the NASA's External SiteShuttle Radar Topography Mission. In such cases it is more convenient to use the Grid Point approach (fig. 25): it simply saves on storage amount and calculation time.

Elevation Base

So far, when working with elevations in a flat landscape, we always implied that the base of the elevation was the bottom of our cubes. More precisely: we assumed that there was a base elevation of 0, upon which we erected structures (cubes). This assumption won't change in the further discussion, but it might be worth to point out some properties of the elevation base to have it defined properly:

- The elevation base is a plane defined by the `x` and `z` axes.
- Every point in the elevation base plane has an elevation of 0.
- It is from the elevation base that elevations are defined on the vertical `y` axis, orthogonal to the `x/z` plane.
- Elevations may be defined in both directions: if they point upward along the positive direction of the `y` axis we call it a positive elevation, otherwise a negative elevation.

Probably the most interesting single aspect of these definitions is, that an elevation can be negative. If, however, this property is not desired (for instance because our data structure only allows for positive values), then it is trivial to redefine the base such, that its lowest point translates to 0. Of course, this translation comes at a cost, as we have to find the lowest point first, usually requiring to scan the whole DEM in a preparatory step.

Revisiting the Formulae

Now that we have aquired a clearer understanding about the term elevation, let's have another look at the formulae to represent any given DEM coordinate.

$$x_S = (x_W - z_W)\cos(a)s - x_O + x_C$$

$$y_S = ((x_W + z_W)\sin(a) - y_W)s - y_O + y_C$$

The blue term y_W is the only one with any relevance to elevation. It transforms the 3D component into a 2D representation by using its true elevation. But didn't we say, that we wanted the elevation to be relative to the elevation of the *Focus*? That elevation is hidden in the y_O term (subscript O for *Origin*):

$$x_O = (x_F - z_F)\cos(a)s$$

$$y_O = ((x_F + z_F + 1)\sin(a) - y_F)s$$

In y_S, let's substitute the term y_O with the according formula:

$$y_S = ((x_W + z_W)\sin(a) - y_W)s - ((x_F + z_F + 1)\sin(a) - y_F)s + y_C$$

We want to concentrate on the two blue terms. Simplifying the formula by highlighting the for the moment irrelevant parts, and then substituting them by capital letters:

$$y_S = (\mathbf{(x_W + z_W)\sin(a)} - y_W)s - (\mathbf{(x_F + z_F + 1)\sin(a)} - y_F)s + \mathbf{y_C}$$

$$y_S = (A - y_W)s - (B - y_F)s + C$$

Expanding the formula

$$y_S = (A - y_W)s - (B - y_F)s + C = A \times s - y_W \times s - (B \times s - y_F \times s) + C = A \times s - y_W \times s - B \times s + y_F \times s + C$$

shows, that because of $(-y_W + y_F) \times s$ the term y_W indeed is relative to y_F: we correct any movement towards the upper screen edge ($-y_W$) by a movement into the other direction ($+y_F$), based on the Focus' elevation.

This is what we wanted to verify. So, are we done? Well, recall the procedure to find the focus (fig. 19): there we assumed, that the surface area of our focus tile was flat (i.e. parallel to the elevation base plane). Because of this, the vertical center was just $h = \sin(a) \times s$ further below, i.e. towards the screen's bottom edge. This was fed into our formula as highlighted:

$$y_O = ((x_F + z_F \mathbf{+1})\sin(a) - y_F)s$$

(Note, that x_O is affected in no way: the focus' elevation impacts only the vertical screen position, never the horizontal one.)

Unfortunately, the surface of the focus tile does not necessarily need to be flat. So, how do we find out the true value in order to redefine our simplified formula? The next section deals with this last aspect in our quest of deriving the final formula.

Averaged Center Elevation

Let's go back to our cube and examine its surface appearance when given different elevations to its 4 corners. We examine 3 cases with the following elevations:

	Case A	Case B	Case C
Topmost corner	1	2	2
Rightmost corner	1	1.5	1
Bottommost corner	1	1	1
Leftmost corner	1	1.5	1

Case A. There's really not much to point out here: fig. 29 shows the standard case which we dealt with all the time, the surface of the cube being a plane parallel to the base elevation plane:

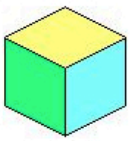


Fig. 29: Case A. Flat surface parallel to the base elevation plane

Case B. Things start to get somewhat more interesting in fig. 30. Note, however, that the surface still is a perfect plane, although it is not parallel to the base elevation plane any longer.

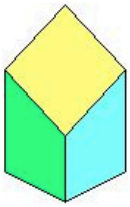


Fig. 30: Case B. Flat surface, not parallel to the base elevation plane

Case C. When we were under the impression, that we easily could derive the surface's center for all cases, then fig. 31 and 32 demonstrate that it's not trivial. This is due to the fact, that the given elevations impose an ambiguity, because they can be interpreted in (at least) 2 ways. Both the following interpretations split the rectangular surface area into 2 triangles:

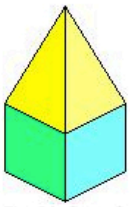


Fig. 31: Case C (a). Any of the diagonals can be interpreted to be connected, in this case forming a ridge line

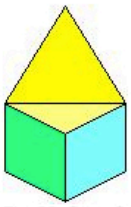


Fig. 32: Case C (b). It is also possible to connect the other diagonal, in this case forming a valley line

Although we currently just are looking for a generalization of the focus point, it can safely be assumed, that any non-flat landscape features a multitude of tiles which can not be interpreted unambigeously (i.e. are "type C" tiles). Hence it is imperative to solve the problem not only for the focus tile, but for all tiles which need to be rendered.

We already mentioned, that "Case C" can be interpreted in at least two ways. This is because it is possible to take better approaches than just guessing which one of the two diagonals to consider. For example could we define, that the elevation at the center of the surface is the arithmetic average of the elevations at the 4 corners. In our case, this would result in an elevation of `(2+1+1+1)/4 = 5/4 = 1.25`. We then would be able to connect each corner individually with the calculated center point, resulting in a total of 4 triangles instead of the former 2 only:

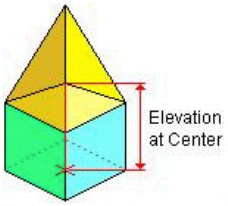


Fig. 33: Introducing averaged elevation at the center

The Final Formulae

What exactly does this mean now for our formula calculating the focus' center? Let's look at its vertical coordinate component again, the part in question still highlighted:

$$y_O = ((x_F+z_F+1)\sin(a)-y_F)s$$

One could argue now, that because the tile was flat and parallel to the base elevation plane, each corner being at an elevation of $1\times s$ above the base elevation, the term $+1$ apparently is nothing else than the average of the 4 corners' elevations already, and because the vertical y axis can be represented 1:1 with the effective elevation, it suffices to replace the term $+1$ with the effective elevation average. For our calculation above, this was $+1.25$. Right?

Well, no. First of all, the term $+1$ is positive, and since y_O desribes the vertical screen coordinate, it goes downward towards the lower edge. A higher elevation (as is the case in our example with $+1.25$) thus must lie further up, towards the upper edge. Furthermore, the term $+1$ was not multiplied with s, but with $\sin(a)$, which is half the "vertical diagonal" on the surface. So, how comes that we have a value of $+1$ there? Well, we based the value on the only coordinate we considered back then, and that was the one of the cube's topmost corner. Recall how we derived the center based on the topmost corner. We said: "just take the topmost corner's coordinates and go downward half the distance toward its opposite corner". In fact, the expressions x_F, z_F and y_F referred to the coordinates of the topmost corner of our focus tile.

$$x_O = (x_F-z_F)\cos(a)s$$

$$y_O = ((x_F+z_F+1)\sin(a)-y_F)s$$

Of particular interest is not only the expression $+1$, but also the expression $-y_F$, as this is the elevation of the topmost corner. And so we arrive at the big picture of it all: from the topmost corner at the base elevation we go up to that corner's real altitude, hence $-y_F\times s$, and then down again half the diagonal with $+1\times\sin(a)\times s$.

Because we want to use the average height of all 4 corners now, we will replace the elevation y_F of the topmost corner with that average. And what about the term $+1$? Well, it still remains there, as it is only the viewpoint which changes. Rather than going downwards on the top surface, we do this on the base elevation now: so to speak, we go to the tile's center of its base area first, before applying the averaged elevation there (however, mathematically an order does not matter in translation series).

The quintessence is, that nothing changes but the interpretation of what y_F signifies. To make this clear, we shall denote the term with $\overline{y_F}$ from now on, the overline standing for "averaged elevation of all 4 corners of the tile".

What about x_F and z_F, do they change as well? No: they don't have any elevation component and merely state the coordinates of a tile's topmost corner. There is no need to average anything here.

So, our final formulas for the *Focus* read:

$x_O = (x_F - z_F)\cos(a)s$

$y_O = ((x_F + z_F + 1)\sin(a) - y_F)s$

Note, that x_F and z_F still indicate the topmost corner of the tile which shall act as the focus.

Also note, that the formulae for x_S and y_S do not change: they always were referring to a tile's corner, which is assumed to have a dedicated elevation anyway.

The only new element regarding tiles is the fact, that also for them a center point exists now, acting as the common corner of the 4 triangles constituting that tile's surface. The elevation of that point is the average of the elevations of all 4 points.

Thus for the 4 corners of any tile (even the focus tile!) remains valid:

$x_S = (x_W - z_W)\cos(a)s - x_O + x_C$

$y_S = ((x_W + z_W)\sin(a) - y_W)s - y_O + y_C$

And since this section is titled "The Final Formulae", let's repeat the (unchanged) formulas for x_C and y_C as well:

x_C = `ScreenWidth / 2`

y_C = `ScreenHeight / 2`

Feedback and Questions

You are welcome to submit ✉feedback regarding this page. I'll also try to answer your ✉questions.