[Introduction](http://archive.gamedev.net/reference/articles/article748.asp" \l "INTRO)

1. li>[Z Order](http://archive.gamedev.net/reference/articles/article748.asp#ZORDER)
2. [Objects That Go Out of a Tile's Bounds](http://archive.gamedev.net/reference/articles/article748.asp#OUTOFBOUNDS)
3. [The Need For Speed](http://archive.gamedev.net/reference/articles/article748.asp#SPEED)
4. [Resolving Confusion from Part I](http://archive.gamedev.net/reference/articles/article748.asp#CONFUSION)

**Introduction**

Hi! Welcome back to Iso/Hex maps. In this tutorial, we will be discussing how to place objects (creatures, items, whatever) onto a Iso/Hex map. There won't be too much code in this tutorial, because most of it deals with the actual blitting of the images, and I'm trying to make these tutorials essentially language independent.

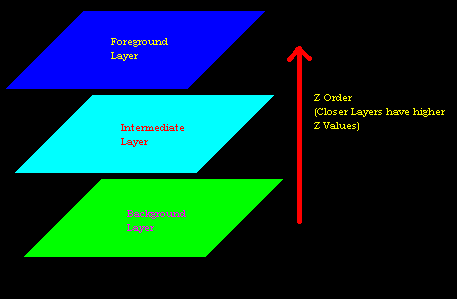
Again, I will be using 40x40 pixel bitmaps.

Also, I will explain by using Bitmasks. However, it has been brought to my attention that transparency bitmaps and RLE bitmaps will also work (and really, this only makes sense, because RLEs and Transparency were created as shortcuts to using bitmasks).

**Z Order**

The first topic that we need to discuss is that of Z Ordering, or Map Layering.

Here is my schematic for Z Ordering:



I've only put three layers into my example?you may require more for your game.

An explanation of the layers, and what goes into them:

**Background**: This layer consists of the tiles themselves, and any image that will always be BEHIND anything else.

**Intermediate**: This layer consists of the objects in the game?creatures, items, etc. Sometimes, it is a good idea to split this layer into two: an Item Layer, and a Creature Layer, with items below the creatures.

**Foreground**: This layer consists of the foreground images (walls, trees, etc). Sometimes, I split this layer into two: foreground obstacles (trees, pillars), and foreground walls.

The way that Z Ordered Iso/Hex tiles get blitted is as follows:

1. Bitmask for Background Layer is ANDed to the screen.
2. Bitmap for Background Layer is XORed to the screen.
3. Bitmask for Intermediate Layer is ANDed to the screen.
4. Bitmap for Intermediate Layer is XORed to the screen.
5. Bitmask for Foreground Layer is ANDed to the screen.
6. Bitmap for Foreground Layer is XORed to the screen.

This is applied to each TILE separately, not the entire layer. If you do the entire layer in one step instead of each tile, you will get a bizarre looking final picture (character's heads chopped off, etc.)

**Objects That Go Out of a Tile's Bounds**

This problem is more prevalent in Iso Maps than it is in Hex Maps, due to the uses of the two different types. Hex Maps are usually for top-down views, and Iso Maps are usually used for 3/4 views.

So, how to we get our "Dude": http://archive.gamedev.net/reference/articles/taniso/Dude.gif

Onto our Iso tile: http://archive.gamedev.net/reference/articles/taniso/GreenIso.gif

So that he looks like THIS: http://archive.gamedev.net/reference/articles/taniso/GreenIsoDude.gif

(See. He sticks out the top of the tile.)

Before we get to that, we need to set down a guideline, so that we don't draw funky maps.

**Isometric Forbidden Zones:**

http://archive.gamedev.net/reference/articles/taniso/IsoForbidden.gif

**Hexagonal Forbidden Zones:**

http://archive.gamedev.net/reference/articles/taniso/HexForbidden.gif

The white zones in these pictures are the Iso/Hex tiles themselves, and are ALWAYS okay to blit onto (in fact, if your images stay within the white--usually the case when Hex maps are used--then you can blit entire layers instead of each tile's layers separately).

The green zones can have image information in them, but you will have to Z Order by tile, not by layer.

The red zones can never have image information in them; if they do, they will either screw up your map, or never be seen.

Background images should take up at least all of the white region, and may spill into the green region. All other layers may take up as much or as little of the white and green regions as you wish.

Back to blitting our "Dude":

First, we blit the bitmask http://archive.gamedev.net/reference/articles/taniso/MaskIso.gifand the bitmaphttp://archive.gamedev.net/reference/articles/taniso/GreenIso.gif of our tile onto our map.

So that we have: http://archive.gamedev.net/reference/articles/taniso/GreenIso.gif.

Now, we AND the bitmaskhttp://archive.gamedev.net/reference/articles/taniso/DudeMask.gif of the dude onto the tile:http://archive.gamedev.net/reference/articles/taniso/GreenIsoDudeMask.gif.

Finally, we XOR the bitmaphttp://archive.gamedev.net/reference/articles/taniso/Dude.gif of the dude onto the tile: http://archive.gamedev.net/reference/articles/taniso/GreenIsoDude.gif.

And then we repeat these steps for each of the tiles in the map. (Additional Layers are blitted in the same way our dude was). The process is the same for Hexagonal Maps, but Dude doesn't stick out.

**The Need for Speed**

Okay, so you DON'T want to be blitting the entire map each game cycle. That would be too SLOW.

BUT! If you just re-blit the tiles in your map that change, your picture gets screwed up.

The solution is this: You update the tile in question, and a few surrounding tiles. While this is a little more work done to update the picture, at least your map will look right, and it beats the hell out of reblitting the ENTIRE map.

Here is my Update Map (for demonstrating methods of quick updating):

Iso: http://archive.gamedev.net/reference/articles/taniso/IsoUpdate.gif

Hex: http://archive.gamedev.net/reference/articles/taniso/HexUpdate.gif

The white zone represents the tile that we are updating. The other colored zones represent the tiles close enough to be affected by our tile.

So, we take a space on our double buffer (or hidden bitmap that we use for updating) that is 40x40, and to it we do the following:

**Iso Maps:**

This is the order in which we blit the tiles:

**Red, Yellow, Green, Cyan, Blue, White, Magenta, Grey.**

Here is how to find WHICH tiles these are (the X,Y locations), based off of the white tile.

(unfortunately, this depends on whether or not the white tile has an odd or even Y value)

|  |  |  |  |
| --- | --- | --- | --- |
| **Color** | **DX(Y is Even)** | **DX(Y is Odd)** | **DY** |
| **Red** | **-1** | **0** | **-3** |
| **Yellow** | **0** | **1** | **-3** |
| **Green** | **0** | **0** | **-2** |
| **Cyan** | **-1** | **0** | **-1** |
| **Blue** | **0** | **1** | **-1** |
| **White** | **0** | **0** | **0** |
| **Magenta** | **-1** | **0** | **1** |
| **Grey** | **0** | **1** | **1** |

Then, you plot them on your buffer with the following X Offsets, and Y Offsets:

|  |  |  |
| --- | --- | --- |
| **Color** | **X Offset** | **Y Offset** |
| **Red** | **-20** | **-33** |
| **Yellow** | **20** | **-33** |
| **Green** | **0** | **-22** |
| **Cyan** | **-20** | **-11** |
| **Blue** | **20** | **-11** |
| **White** | **0** | **0** |
| **Magenta** | **-20** | **+11** |
| **Grey** | **20** | **+11** |

Here's the X Offset/Y Offset values for **Hex maps** (Hex maps use the same DX/DY values as Iso maps):

|  |  |  |
| --- | --- | --- |
| **Color** | **X Offset** | **Y Offset** |
| **Cyan** | **-20** | **-30** |
| **Blue** | **20** | **-30** |
| **White** | **0** | **0** |
| **Magenta** | **-20** | **+30** |
| **Grey** | **20** | **+30** |

For each of the tiles in the Update Map, each layer (for added speed, you may want to clip the images that you are blitting onto the buffer).

Once you have blitted all of the necessary tiles onto the buffer, you can blit the buffered image onto the map using MOV (also called Copy).

There you go. You get better speed, and your picture doesn't get messed up.

(P.S. If you are using Visual Basic, I would like to suggest that you use the API function BitBlt rather than VB's PaintPicture Method. It's SOOOO much faster.)

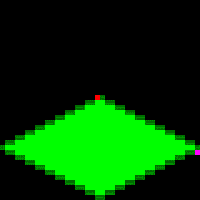
**Resolving Confusion from Part I**

Somebody sent me some E-Mail saying that I should explain how I arrived at the Y Offsets for my Iso/Hex tiles, because not all of you are going to be using 40x40 bitmaps. (although I know for sure that at least ONE of you has switched to using them).

So, here is the method I used to figure out the Y Offsets for my Iso and Hex maps:

(To more easily show this, I have blown up my Iso and Hex tiles)

Iso:

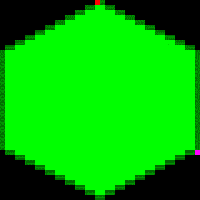


The determining factors for figuring out the X and Y offsets for a rhombus or a hex start with picking any point and figuring out where it will be when on an adjacent tile. In the above picture, the red dot corresponds to the Magenta dot on another tile. The coordinates of the red dot are (19,19), and the magenta dot is at (39,30). From this, calculating the X Offset and Y Offset are easy:

X Offset=39-19=20

Y Offset=30-19=11

Hex:



In this case, the red dot is at (19,0), and the magenta dot is at (39,30).

X Offset=39-19=20

Y Offset=30-0=30

I hope this clears up any confusion, and helps you when you go about designing your own, differently sized or shaped Iso/Hex tiles.

[**Discuss this article in the forums**](http://archive.gamedev.net/community/forums/topic.asp?key=featart&uid=748&forum_id=35&Topic_Title=Isometric+%27n%27+Hexagonal+Maps+Part+II)