Coordinates in Hexagon-Based Tile Maps   
by [Thomas Jahn](mailto:lithander@gmx.de)   
28 Feb. 2002

**Introduction**

Tile Maps are often used for the graphics in two-dimensional games. Using tiles means that all graphics are combinations of smaller graphics, similar to mosaics. The advantage is a smaller need for memory. The tilesets are seldom bigger than a few hundred kilobytes and a map needs far less memory than a pre-rendered image; it needs only a pointer per field that identifies which tile is used.

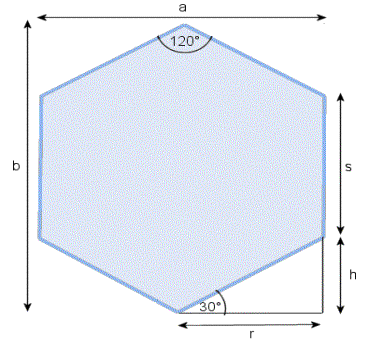
The simplest tiling system has plain squares. It's used in games like Warcraft II or the first C&C games. For isometric views a la Diablo or Ulitma 7 you usually use diamond shaped tiles. These Tiles allow you to achieve a pseudo 3D look. Both tiling systems have a major disadvantage. All movements are restricted to four directions (vertical and horizontal for squares and diagonal for diamonds) or you have to cope with different step distances. (The distance between two tiles that touch only at the edges is bigger, then the distance of tiles that touch at the sides.)

A hexagon based tile system offers a very nice solution. It allows movements in 6 directions with equal distances, so it's used especially by tactical games like Panzer General. A second advantage is that those hexagon-tiled maps often just look better then simple square maps.

On the other side hexagon tiles are harder to work with, both for artists as well as for programmers. This article solves some problems you might have when trying to implement hexagon based tile maps. I will explain how to plot the tiles correctly and how to define which pixel is on which hexagon (i.e. for mouse event handling).

**The mathematical structure of Hexagons**

First of all let's have a look at the mathematical characteristics of a hexagon.

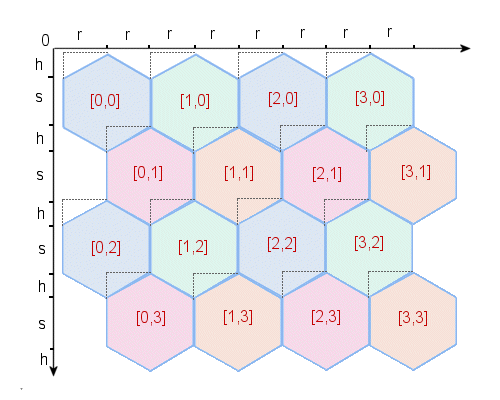


In a hexagon all six sides have the same length **s** which meet at an angle of 120�. All other values depend on **s**, so if you set **s** you can easily calculate the missing information:

|  |  |
| --- | --- |
| The height **h**: | **h = sin( 30�) \* s** |
| The distance **r**: | **r = cos( 30�) \* s** |
| The height of the surrounding rectangle **b**: | **b = s + 2 \* h** |
| The width of the surrounding rectangle **a**: | **a = 2 \* r** |

**Converting array coordinates to pixel coordinates**

As in all other tile maps the information for hexagon based tile maps is stored in a two dimensional array. To plot hexagonal tiles we need to know the pixel coordinate of the top left edge of the bounding rectangle of the hexagon. As we only know the array coordinate (the position in the array) we have to find a way to convert these coordinates to pixel coordinates.



If you have a look at the above picture you'll probably realize that every second row of tiles is indented by **r**.

For even rows (blue-green) we have to use this calculation:

**x**pixel = **x**field \* **2 r**   
**y**pixel = **y**field \* **( h + s )**

For odd rows (red-orange) we have to add **r** to the horizontal pixel coordinate:

**x**pixel = **x**field \* **2 r + r**   
**y**pixel = **y**field \* **( h + s )**

It's not hard to write that as program code:

//pseudo Code!!!

PixelX := ArrayX \* 2 \* TileR + (FieldX AND 1) \* TileR

PixelY := ArrayY \* (HexagonH + HexagonS);

AND is a boolean And-Operator. The result of the brackets is always one if the last bit of ArrayX is on, which means that ArrayX is odd.

**Converting pixel coordinates to array coordinates**

Now it should be quite easy to plot our map. But that's not enough! Things become complicated again if we want to know which tile (the array coordinate) the mouse cursor is over. To solve this we have to find a way to convert pixel coordinates to array coordinates. A relatively simple solution is using a mousemap (like CivII does) but there is (in my opinion) a more elegant and faster way.

The first thing we do is to get the pixel coordinate. If we have a scrollable map we have to add the position of the left top edge of the viewable sector to the cursor position.

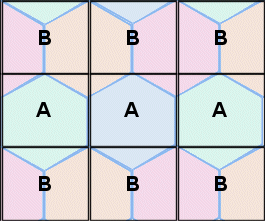
Now we divide our map into rectangular sections. Have a look at the illustration below. It's quite easy to calculate the section in which our pixel coordinate lies. A section has a height of **h + s** and a width of **2 \* r** (or simply **a**). Therefore we can use these simple equations:

**x**section = **x**pixel / **( 2 r )**   
**y**section = **y**pixel / **( h + s )**

Keep in mind that you have to round off when doing these calculations.

It's not enough to know in which section we are. We also need to know on which pixel we're on within the section. To determine that we subtract the position of the left top edge of our section from the global pixel coordinates.

**x**sectpxl = **x**pixel - **x**section \* **( 2 r )**   
**y**sectpxl = **y**pixel - **y**section \* **( h + s )**



There are two different types of section as you can see. It's very important to know of what type our section is. Looking at the above illustration you might realize that sections in even rows (those with an even **y**section value) are type A while sections in odd rows are of type B.

Let's see how all these calculations are coded:

//pseudo Code!!!

SectX := PixelX div (2 \* TileR);

SectY := PixelY div (HexagonH + HexagonS);

SectPxlX := PixelX mod (2 \* TileR);

SectPxlY := PixelY mod (HexagonH + HexagonS);

If (SectY AND 1 = 0) then SectTyp := A else SectTyp := B;

Now we have all the information we need to determine the array coordinate of the tile a pixel is in. Depending on what section we have the remaining calculations are different.

**A-Sections**

A-Sections consist of three areas.

* If the pixel position in question lies within the big bottom area the array coordinate of the tile is the same as the coordinate of our section.
* If the position lies within the top left edge we have to subtract one from the horizontal (x) and the vertical (y) component of our section coordinate.
* If the position lies within the top right edge we reduce only the vertical component.

It's not hard to determine which area we are in if we know the gradient of the diagonal edges. Every edge has a gradient of either **m = HexagonH / HexagonR** or **m' = - HexagonH / HexagonR**. The following code shows how you can code the area distinction:

//pseudo Code!!!

// middle

ArrayY := SectY;

ArrayX := SectX;

// left Edge

if SectPxlY < (HexagonH - SectPxlX \* m) then begin

ArrayX := SectY - 1;

ArrayX := SectX - 1;

end;

// right Edge

if SectPxlY < (- HexagonH + SectPxlX \* m) then begin

ArrayY := SectY - 1;

ArrayX := SectX;

end;

**B-Sections**

B-Sections consist of three areas, too. But they are shaped differently!

* If the pixel position in question lies within the right area the array coordinate of the tile is the same as the coordinate of our section.
* If the position lies within the left area we have to subtract one from the horizontal (x) component of our section coordinate.
* If the position lies within the top area we have to subtract one from the vertical (y) component.

We will again use the gradient of the edges to determine which area the pixel is in. The following code shows how:

//pseudo Code!!!

// right side

if SectPxlX >= HexagonR then begin

if sectPxlY < (2 \* HexagonH - SectPxlX \* m) then begin

ArrayY := SectY - 1;

ArrayX := SectX - 1;

end else begin

ArrayY := SectY;

ArrayX := SectX;

end;

end;

// left side

if SectPxlX < HexagonR then begin

if SectPxlY < (SectPxlX \* m) then begin

ArrayY := SectY - 1;

ArrayX := SectX;

end else begin

ArrayY := SectY;

ArrayX := SectX - 1;

end;

end;

Well, now you should be able to handle mouse events and cope with every other problem involved with the conversion of pixel coordinates to array coordinates.

Last but not least I want to thank David House for reviewing this article!

Please visit my website at <http://pixelpracht.flipcode.com> to get more articles and other stuff. I'd welcome some comments or inspirational ideas. Here's my mail address: [lithander@gmx.de](mailto:lithander@gmx.de)!

Happy Coding!   
Thomas

[Related Posts Plugin for WordPress, Blogger...](http://www.linkwithin.com/)