Jon Ritman's Isometric Tutorial

Jon Ritman is the author of the filmation isometric games "Batman" and "Head Over Heels" back in the eighties. As I stucked in programming my isometric game, I contacted him for some tips. He was very kind and he gave me some answers. Thanks, Jon!

I will just copy and paste these e-mails here (my questions are marked blue).

Danko, DKOZAR.COM, December 10th 2004.

Hi, John!

First of all I'd like to say that I'm a fan of Head over Heels - for me it's the best game of all the time. And I think that isometric games are the best games ever made on 8-bit computers.

Since today's web games (Flash, javascript) are mostly on that 20-year-ago level, I decided to make isometric one. And I didn't realize how hard it is and what the problems are until I stuck with creating the isometric engine.

http://dkozar.com/flash/izometrija.swf (space + arrows to move)

I use 1) map coordinates, 2) isometric coordinates and finally 3) Flash coordinates.

My problem is how to make a perfect sprite-sorting algorithm. I can say that I turned the web upside-down to find something like that, but it was the waste of time, because there is no deeper analysis of isometric sprite sorting.

In my engine there is a weighting "sprite" algorithm based on assigning weight to every brick from top corner on the lowest brick level to the lower corner, than level up and so on. It works well for the static conditions. Problems come up when something is moving.
I am aware that the walls and the floor (and one half of the door) have to be pure graphics, since they are always "beyond", but in my example they are made of bricks - I'll change that later. I also came up that in just one layer the sorting algorithm is calculated with x+z condition..

I think that only you and some people at "Ultimate" salved all problems so...
If you would like to help me, please write back! I'd be grateful.

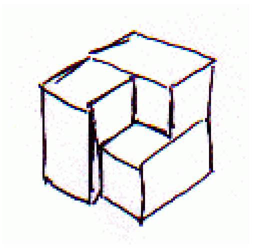
Danko

First the bad news, there is no perfect sorting method using isometric sprites!

There are tricks to make it work most of the time though:

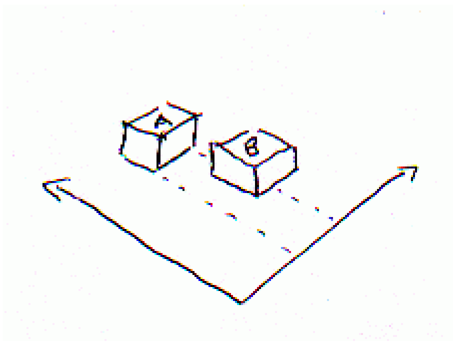
I used a linked list that had the object furthest back appearing first in the list, to redraw any section of screen just run through the list checking if the sprite overlaps the area you want to draw and if it does, draw the overlapping section.

For tall objects, split them in half and have two entries in the list, if you don't do this you will definitely have problems.



Note the example above, no one block is fully in front of another, the only way to sort them is to have the tall object split into two parts.

To place an object in the list you start at the beginning of the list (the back of the room) and work your way through it till you find an object that is in front on one of the axis, then you insert the new object in behind it.



In this example object B is in front of object A because object B overlaps on one axis.

To move an object during game play:
1) calculate the new draw window by taking the moving objects current screen area and it's new screen area and create a box covering both areas.
2) remove the moving object from the linked list
3) Searching from the start of the list look for another object that is in front of the moving object (or the end of the list)
4) place the moving object behind the found object (e.g. before it in the list)
5) draw every object in the list that overlaps the window into a buffer
6) copy the buffer to the correct screen position

It's that simple

I did collision detection on a plane detection, when an object moves only 1 or 2 edges are advancing (two if diagonal movement)
First I check if there is an edge (of each object in the linked list) that is touching the moving edge, if there isn't then the object moves, if there is then the object doesn't move but does set a flag (a 6 bit flag, one for each direction - either 1,2 or 3 bits are set) in the touched object. This flag is known as the 'push' flag. When each object is processed it checks the push flag, some objects have distinct actions for any push direction and some just to one or more directions - objects that can move use the push flag to tell them what, if any, direction they have been pushed - this push may be in opposition to their own natural movement, therefore stopping them. Stationary objects that receive a push make an attempt to move in the appropriate direction, they may in turn push another object.

Because this is a very simple physics engine you will notice that if you push a row of brick towards the camera it has a different action to a row pushed away from the camera, this is to do with the processing order (object to the back of the room are earlier in the list and are processed first). Pushing toward the camera will look reasonably smooth as the entire row will be pushed each cycle, pushing away from the camera the bricks that are pushed will have already been processed that cycle and will therefore not move until the following cycle, if there are 4 bricks being pushed then it can take 4 game cycles before the one on the end of the row moves creating a gap between it and the 3rd, the 3rd brick then moves 3 cycles later, the 2nd brick 2 cycles after that etc.

The push flag is zeroed after the object has been processed

Doors are handled separately, they are mostly outside the room and can only be touched on one edge, if they are touched by the main characters they actually set the push flags of the characters rather than the other way around, this has the effect of making it easier to get through doors as they push the character into the middle of the doorway. A door is drawn as two separate parts, one being the far door post and the other being the near post and the top of the door frame, this allows the main character to go between them

Outside walls are just strips of wallpaper and not included in the linked list, they are the first edges checked when an object moves and if the object is against the wall then movement cant happen unless it is a main character and they are lined up with the doorway.

Conveyor belts are a little like doors in that they set the push flags of objects on top of them

Object that can fall keep a link to the object they were last on top of, when they are processed they check if that object is still beneath them, if it is then fine, if not then they search the list for another and if none is found they move down

Floors are simple, if the objects base is at floor level (and in a room all floors are the same height) then it cant move down

Both walls and floors are draw first into the buffer as simple graphic strips, the reason the rear door appear to be behind the walls is that there is a special version of the near doorpost for the rear walls that has a cut away to let you see the wall

---

For the floors, I thought that they are made of bricks because there are rooms with no floors (with link to another room). This probably means that each room has a one bit flag for floor/no floor.

almost correct, in practice in the compressed map the floor was stored as 3 bits (8 combinations) with 7 combos available to describe the floor pattern and one combo to say it wasn't there at all

---

Does it mean that all of the objects in the room (fixed and moving objects) are of the same kind - some with "move" flags which enable them to move, some with this flags zeroed? Does it mean that the map coordinates (e.g. (1,1,2), (3,3,2)) are considered only at the initialization of the room and objects (first drawing)? Because later some objects can be moved. You don't use map coordinates later in moving the objects in the room? Or for walls only? Or: are the walls something like the floor, I mean, do you specify the x, y, z size for every room at the initialization and then check does the character touch the walls?

In HoH the map was stored in an extremely compressed form as there was little memory available (301 rooms in 5k) and then expanded into memory as each room was initialized - there were only 8 different room sizes (3 bits), 8 door types (3 bits) etc. The wall positions were stored at room initialization and checked by the collision routine. All objects within the room had a link to a processing routine, many didn't change in any way and the link pointed to a null program, others could be pushed but did little else so the linked program simply checked the push flag and moved the object if any bits were set.

BTW there was also a flag similar to the push flag that indicated what way the object was moving so that when another object hit it, it could use the flag to OR into it's own push flag, this is how the conveyor and doorposts did their stuff.

You said that every tall object must be created of two objects (for sorting purposes). Is it the same with the far door post which can be in front of/behind the main character? You said also that far door post is drawn of only one part.

Because only the main character could go between the doorposts the impossible situation described yesterday could not occur so there was no need to split the door up. As I remember I did something pretty similar using half the width but when I did the height axis I didn't store the base point half way up the object, I used the bottom of the object as it needed less messing about to check if standing on something