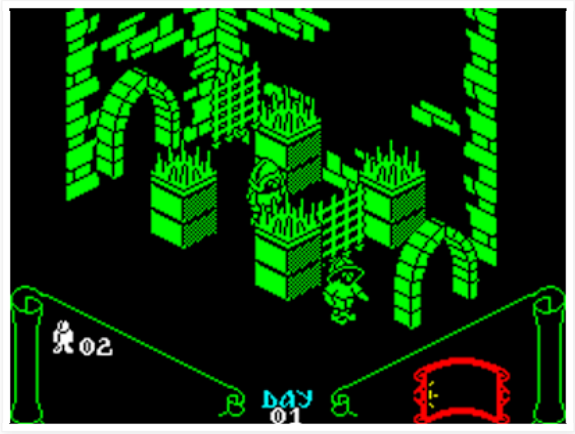
Bannalia: trivial notes on themes diverse

Wednesday, February 20, 2008

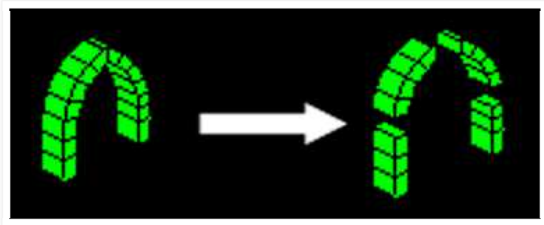
Filmation math

In 1984, European teenagers were thunderstruck by the appearance of the ZX Spectrum hit Knight Lore from a firm called Ultimate Play the Game. This adventure game featured a rendering technique like we hadn't ever dreamed of before.



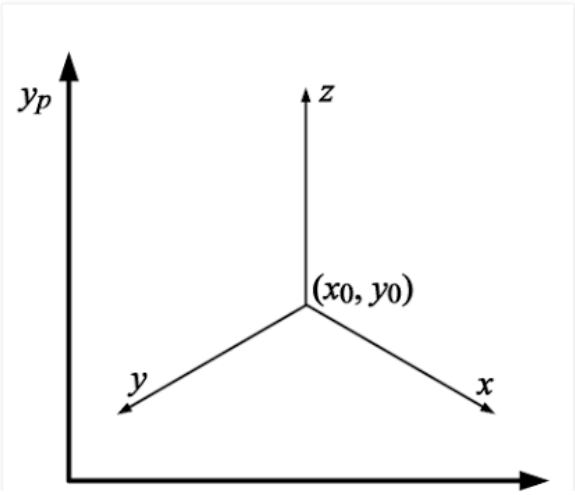
In a world of 48 KB, 2D platform games, Knight Lore represented a quantum leap forward with its strikingly realistic 3D world based on isometric perspective. Ultimate called this rendering technique Filmation. I used to wonder how they could implement Filmation in such a limited computer as the ZX Spectrum, but the math behind it turns out to be simpler than it might seem.

A given room consists of a background set (the stone walls in the example above) and a bunch of objects. We can assume that the objects are roughly convex: if an object is not convex, for instance if it has holes like the arched gateways, we can just decompose it into a number of convex elements.



Since isometric perspective does not introduce any scaling effect (distant objects are drawn the same size as closer ones), rendering a frame boils down to drawing the background set and then all the room objects, applying a simple projection rule:

$$(x,y,z) \rightarrow (x_0 + \sqrt{3}\,x/2 - \sqrt{3}\,y/2,\ y_0 - x/2 - y/2 + z)$$



About Me

Joaquín M López Muñoz

View my complete profile

Blog Archive

Subscribe via email

Your email address:

Go

Subscribe

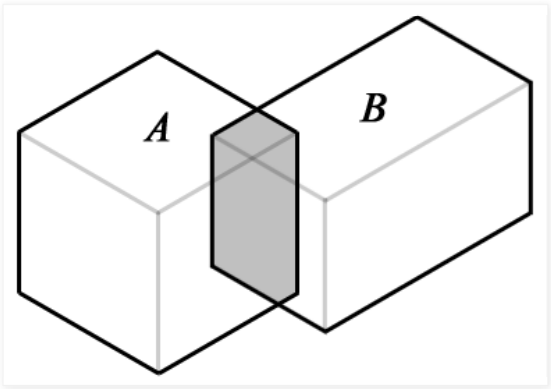
Subscribe to Bannalia

x_p

which is generally approximated by the following:

$$(x,y,z) \to (x_0 + x - y, y_0 - x/2 - y/2 + z)$$

so as to simplify computations as well as graphic design (the x and y axes are projected to easily pixelizable lines wih slope 1/2). But there is one detail left: if an object A appears behind B, we must draw A before we render B. So, room objects have to be ordered for drawing purposes according to the "is behind" relation. In which situations is A behind B? As objects are assumed to be convex and they do not overlap (i.e. $A \cap B = \emptyset$), we can approximate them by prisms with opposite vertices (x_A,y_A,z_A), (x'_A,y'_A,z'_A) and (x_B,y_B,z_B), (x'_B,y'_B,z'_B), with $x_A<x'_A$, $y_A<y'_A$, $z_A<z'_A$, and similarly for B. These prisms are projected to hexagons as depicted in the figure:



A little calculation shows that the hexagons for A and B overlap if and only if:

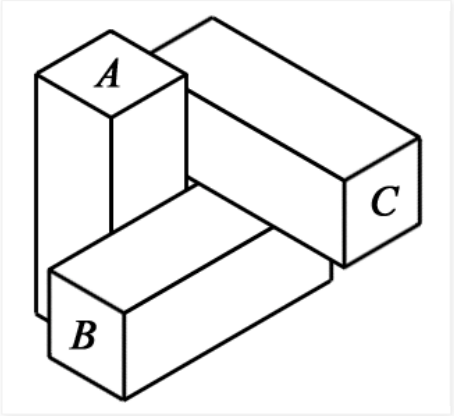
1. $[x_A - y'_A, x'_A - y_A]$ and $[x_B - y'_B, x'_B - y_B]$ overlap, and
2. $[x_A - z'_A, x'_A - z_A]$ and $[x_B - z'_B, x'_B - z_B]$ overlap, and
3. $[-y'_A + z_A, -y_A + z'_A]$ and $[-y'_B + z_B, -y_B + z'_B]$ overlap.

If the hexagons do not overlap, it is immaterial which object we draw first; if they do overlap, we can apply the following easily provable properties:

1. $x'_A < x_B \to A$ is behind B,
2. $x'_B < x_A \to B$ is behind A,
3. $y'_A < y_B \to A$ is behind B,
4. $y'_B < y_A \to B$ is behind A,
5. $z'_A < z_B \to A$ is behind B,
6. $z'_B < z_A \to B$ is behind A.

Since the prisms A and B do not overlap (in 3D), it can be seen that at least one of the clauses 1-6 can be applied, so we can use these properties to sort all the room objects and then proceed to drawing. Note that the relation "is behind" defined by the algorithm above is not total: if the projections of two blocks do not intersect, then neither block is behind the other; thus, in general blocks can be sorted in more than one acceptable manner.

Are we done then? It looks so, and actually a rendering algorithm can be implemented along these lines and seemingly works fine. But there is a fundamental flaw in our analysis. The relation "is behind" is not transitive:



In the figure, A is behind B, B is behind C and C is behind A! There is no way that these objects can be sorted so that they are rendered properly; if we wish to keep our basic

algorithm, at least one of the objects should be split in two so as to restore transitivity, but knowing when and how to do this is not trivial. How did Ultimate programmers solve this problem? Well, the answer is that they did not solve it. Consider for instance the following snapshot, taken from Nigel Barford's Java Spectrum Emulator:

One table is on top of the other and slightly displaced with respect to it. If we position our hero so that he stands behind the bottom table and in front of the top one, something unexpected happens:

The top table is suddenly rendered as if it were behind the bottom table: if you examine this configuration you will see that we have indeed an antitransitive cycle composed of the two tables and the main character. As these situations are uncommon, the programmers of the Filmation engine decided to just ignore them. So, next time you play a Filmation game and observe one of these rendering quirks, you will know the math behind them.

Posted by Joaquín M López Muñoz at 9:49 AM

39 comments :

Anonymous September 11, 2008 at 10:30 PM

I am trying to apply your overlap formula to a project I am working on and am having an issue with understand some of the logic/vernacular used in your post.

Could you explain a bit more about what you mean by prisms? Also by opposite vertices are you meaning something like [Ax, Ay, Az] and [Ax + Awidth, Ay + Alength, Az + Aheight]. Or are you referring to actual screen coordinates of certain vertices on the objects?

This is a great article. Thank you so much for shedding some practical math on topic.

Reply

Think Tank Dev September 11, 2008 at 10:34 PM

This comment has been removed by the author.

Reply

Anonymous September 11, 2008 at 11:51 PM

I think I understand by what you mean prisms now (those being the actual psuedo-3D objects we're drawing).

What I don't understand what you mean by overlap:

1. [xA−y'A,x'A−yA] and [xB−y'B,x'B−yB] overlap, and
2. [xA−z'A,x'A−zA] and [xB−z'B,x'B−zB] overlap, and
3. [−y'A+zA,−yA+z'A] and [−y'B+zB,−yB+z'B] overlap.

I have expressed these as two dimensional points (6 total) but I am not sure how you compare them to determine if an overlap exists.

Please advise.

Thanks.

Reply

Joaquín M López Muñoz September 12, 2008 at 8:04 AM

What I don't understand what you mean by overlap:

1. [xA−y'A,x'A−yA] and [xB−y'B,x'B−yB] overlap, and
2. [xA−z'A,x'A−zA] and [xB−z'B,x'B−zB] overlap, and
3. [−y'A+zA,−yA+z'A] and [−y'B+zB,−yB+z'B] overlap.

What I mean by "[a,b] and [c,d] overlap" is:

($a \le c \le b$) or ($c \le a \le d$)

Hope this helps.

Reply

Anonymous September 12, 2008 at 8:33 PM

That makes alot more sense to me. I am doing this in actionScript3 so the explanation was hard to put into context of AS3.

Thanks again for the great posting. I have now gotten my isometric rendering engine doing depth sorting correctly.

Reply

Joaquín M López Muñoz September 12, 2008 at 9:16 PM

Glad to be helpful. Just in case you haven't found it yet, please take a look at a JavaScript implementation of these ideas here.

Reply

jwopitz October 2, 2008 at 6:31 PM

Hi. Wanted to say thanks to your post I have gotten past a major hurdle in developing my isometric library built in actionscript 3. You can check it out here - http://code.google.com /p/as3isolib/

Reply

Joaquín M López Muñoz October 2, 2008 at 6:49 PM

Hi jwopitz, I'm glad the stuff's been helpful! When will you provide a demo app for your library?

Reply

jwopitz October 15, 2008 at 10:07 PM

Tho there isn't a live demo of the lib in use, there are some tutorials posted as well as a users' group.

Reply

Brian H February 26, 2009 at 12:18 AM

Hi, great post. I've developed an Actionscript library that uses more or less the Filmation approach, although this implementation is O(n) so if the original Filmation algorithm was O(n^3) then maybe this is an improvement? Check it out here:

http://www.fenstalker.com/content/isoworld

Your thoughts?

Reply

Joaquín M López Muñoz February 26, 2009 at 7:57 AM

Hi Brian,

Very nice demo! Can you describe your O(n) algorithm?

Best regards,

Reply

Satan Transexual December 1, 2010 at 5:25 PM

I think the sorting error that you show in the snapshot is not because of the case you mention.

Is because the Filmation algorithm is not transitive for the set { all cases except intersection and “ABC overlap”}. It´s because:

“If the hexagons do not overlap, it is immaterial which object we draw first”

This is right, but the algorithm assuming that because “it is immaterial which object we draw first”, it´s not necessary to sort it, and this is wrong:

Let´s name A the table at the back, B the character and C the table at the front, and X the array in which we are going to sort the objects by depth. Because you don´t know in which order you are going to compare the objects, it´s possible to end in this situation:

X is empty

Add C to X

Is C behind B? Because they don´t overlap, it doesn´t matter, so add B to X.

Now you have the X=(C, B)

Is B behind A? Because they don´t overlap, it doesn´t matter, so add A to X.

Now you have the X=(C, B, A)

So you display the objects in that order and because A is displayed after C, so the table at the back appears over the table at the front.

To solve this problem you need a transitive algorithm for the set { all cases except intersection and “ABC overlap”}

I think this can easily achieved by making some extra comparisons if the objects don´t intersect.

Reply

Joaquín M López Muñoz December 2, 2010 at 8:37 AM

Hi Satan,

I think your analysis is not correct: A, B and C all overlap each other, so the algorithm does not assign them arbitrary rendering orders as your post suggests. Please take a look again at the figure with three prisms and the text above:

A is behind B, B is behind C and C is behind A.

There is no way that one can order A, B and C so that if X is behind Y then X is ordered after Y. It's just impossible, not a defect of the algorithm we happen to be using.

Reply

Bartosz December 21, 2010 at 4:39 PM

Hi,

first of all thanks for a great explanation of filmation math! I've managed to implement it in ActionScript 3. However I'm experiencing one bug which I can't fix.

For some reason detecting if items overlaps fails in one case. See screenshot below:

http://deluxe.pl/iso/glitch1.png

If I move the block in the top little bit close to the bigger one then it works:

http://deluxe.pl/iso/glitch2.png

Did you ever met this problem?

Best Regards,
Bartek

Reply

Joaquín M López Muñoz December 23, 2010 at 8:03 AM

Hi Bartosz,

I don't know why you're getting the glitch you show, since there are no transitive cycles in your example. I think a correct implementation of the ordering algorithm should not fail here. Can you post or provide a link to the relevant portion of your code?

Reply

Joaquín M López Muñoz December 23, 2010 at 8:08 AM

Bartosz,

Additionally, have a look at the function *depth_sort* in this article. Comparing with your code might give you some clue.

Reply

Max January 10, 2012 at 12:17 PM

Hi Joaquin,
first, many thanks for writing this nice article! It really gave me some insight to isometric rendering. I'm currently working on an iso engine and thinking about the occlusion computations. It is not using the classic "diamond" map, but a rectangular map with arbitrary shaped convex boundary polygons for the objects.

I assume that objects are shaped like "towers", i.e. they never overlap with their x and y coordinates in 3D space (unlike the tables / prisms in your example, which is the reason for the circle in the draw order).

While Satan T., in his comment, was wrong about the point of occlusion between the example objects A, B and C, he mentions a real problem which makes the whole thing complicated:

"If the hexagons do not overlap, it is immaterial which object we draw first."

This is, one the one hand, true, because we can not in general decide over the order if two objects don't overlap.

On the other hand, like S.T. mentioned, the case of "equality" of two objects breaks the transitivity of the relation: if there are three objects A, B and C, and A and B do not overlap as well as B and C do not overlap, but A occludes C, this causes a problem:

A sort algorithm might find that
A == B and B == C
and therefore do nothing. But instead, because A occludes C, it should actually swap A and C!!

I have been using the (C++) STL sort algorithm and encountered that problem.

Now, with an own algorithm that always compares every object with every other object, I have a O(n^2) complexity or sth. like that, which is much too slow - it brought my application from ~300 frames (with the std::sort) down to ~40 frames x-D

I know this is due to the more complex case of comparing two arbitrary polygons, and I wish I could somehow introduce the transitivity into my isBehind relation... but I fear it's not possible. What do you think?

Best Regards, Max

Reply

Joaquín M López Muñoz January 10, 2012 at 2:57 PM

Hi Max,

First of all, using a sorting algorithm can't work because the "is behind" relation is not a strict weak order (a partial ≤-type order): for instance, if A->B (-> meaning "is behind") and B->C it does not follow that A->C --the offset between A and C can be so large that they not overlap. So, using std::sort might seem to work but will produce glitches and spurious results. You should need (transitive cycles aside) to replace -> with its transitive

closure ->*, defined as A->*B if there's a path A->X1->...->Xn->B. ->* is a strict weak order (transitive cycles aside.)

In graph theory's terms, the problem is: we have a bunch of objects related by -> and need to

1. detect and eliminate -> cycles
2. do a topological sorting of the remaining graph.

This can be done in O(|V|+|E|), where V and E are the vertices and efges of the graph. The problem is that we don't have E given as a set of edges: we can only determine if there's an edge between A and B (i.e. if A->B), which is not the same thing --constructing the E set is O(|V|^2).

I'm sure this algorithm can be improved to reach lineal or near-linear time, but I haven't done the anlasysis. Let me think of it for a while.

Reply

Max January 10, 2012 at 6:19 PM

Hi Joaquin,
thanks for your fast reply! The thingy with the transitive closure sounds worth checking!
I already got the clue that it can be solved with a topological sort, so I spent the last hours on using boost for that. It turned out that, as one might expect, the performance leak is not the topological sort of the graph, but its actual creation of the graph, i.e. finding out which object is occluding which object.
For now, this is done by checking each object against all others.
I think investing the possibility to turn the relation into a strict weak ordering relation might help at this point, I'll check that!

Reply

Suminsky August 15, 2014 at 7:48 PM

Pardon my ignorance, but I dont get what you mean by prism with opposite vertices, nor I know what vertices are you talking about with [xa, ya, za], [xa', ya', za']. Those are only 2 vertices..I dont get that representation.
With makes the info following that hard to grasp.
If you would put the xa, xa', etc. on the figure (showing the overlapping of A and B), where those would be?

Reply

Joaquín M López Muñoz August 15, 2014 at 8:21 PM

Hi Suminsky

In the figure you refer to, [xa, ya, za] would be the vertex right at the back of the prism, that is, the only one that is not visible, while [xa', ya', za'] would be the vertex opposite that, that is, the one that lies at the intersection of the three light gray edges. You will note that these two vertices univocally determine the whole prism.

Reply

Suminsky August 15, 2014 at 9:19 PM

So you're checking if 2 hexagons overlap using 3d coordinates of the 'rectangular cube' they represent?
Jeez..How is that possible? To me the only way to do this would be making a full collision test between the 2 hexagons (with 2d coords).
How did you get to this man?

Reply

Joaquín M López Muñoz August 15, 2014 at 11:40 PM

Precisely, the conditions for the two hexagons to overlap (or not) are laid out in the article, equations 1. to 3. As it happens, the 2D coordinates of the hexagons can be expressed as functions of the 3D coordinates of the associated primsms, hence hexagon overlapping can be checked using the latter.

Reply

Vincent Williams July 15, 2015 at 7:09 AM

Hi I am making an isometric game but I am having some trouble following this tutorial. Can you explain where these points are on the diagram: "(xA,yA,zA), (x'A,y'A,z'A) and (xB,yB,zB), (x'B,y'B,z'B)" I don't understand what all those letters refer to. Or could you post the is_behind method? Sorry if this is a stupid question.

Reply

Replies

Joaquín M López Muñoz July 15, 2015 at 9:03 AM

Hi Vincent:

I explained this very issue to Suminsky some posts ago:

In the figure you refer to, [xa, ya, za] would be the vertex right at the back of the prism, that is, the only one that is not visible, while [xa', ya', za'] would be the vertex opposite that, that is, the one that lies at the intersection of the three light gray edges. You will note that these two vertices univocally determine the whole prism.

As for is_behind, you might want to take a look at the code in this entry.

Vincent Williams July 15, 2015 at 3:43 PM

First of all thank you for responding so fast! That explanation makes more sense but how do I get a prism from a 2d image? If I just have 2d images I know how to get X and Y but not Z. :(

Vincent Williams July 15, 2015 at 3:44 PM

Oh also I saw that article but it did not show code for is_behind, It just showed how to sort objects using the is_behind method.

Joaquín M López Muñoz July 16, 2015 at 12:44 AM

How do I get a prism from a 2d image? If I just have 2d images I know how to get X and Y but not Z.

If you're writing an isometric game your characters, objects etc. move in a 3D world with X and Y (tipically ground coordinates) and Z (typically height above the ground). You can approximate these blocks as 3D prisms described by the coordinates (xA,yA,zA), (x'A,y'A,z'A) and project to the 2D screen using the formulas explained at the article.

Joaquín M López Muñoz July 16, 2015 at 12:46 AM

Oh also I saw that article but it did not show code for is_behind [...]

Look at the source code of the frame where the game is embedded (this URL).

Reply

Vincent Williams July 16, 2015 at 6:24 AM

Thank you so much I think I can figure it out from here (although I dont know javascript so its a bit hard to read). I have been searching for the answer for so long. Im so glad I happened to find this article!

Reply

Vincent Williams July 22, 2015 at 6:16 PM

Sorry one more question. I am using a map system where the point (0, 0) is at the left not the top and I cant figure out how to get it to work with that system. (imagine that the y line is between the x and the z in your picture). I have tried figuring it out myself but I am a bit confused on how you came up with these calculations: "
A little calculation shows that the hexagons for A and B overlap if and only if:

[xA−y'A,x'A−yA] and [xB−y'B,x'B−yB] overlap, and
[xA−z'A,x'A−zA] and [xB−z'B,x'B−zB] overlap, and
[−y'A+zA,−yA+z'A] and [−y'B+zB,−yB+z'B] overlap."

Reply

Replies

Joaquín M López Muñoz July 22, 2015 at 8:32 PM

Sorry but I don't get how your coordinates are laid out. Anyway, the formulas for checking if two hexagons overlap will work just the same (technically, they're invariant on axis naming and sign).

Vincent Williams July 22, 2015 at 8:46 PM

The coordinate system works like this: http://www.alcove-games.com /wp-content/uploads/2013/03/iso_coord.png. And since the x and y values start at the left instead of the top, when I try to use your equations they aren't drawn in the right order.

Reply

Joaquín M López Muñoz July 22, 2015 at 9:15 PM

I see.

xc = 0.5 + 0.5 xi + 0.5 yi
yc = 0.25 - 0.25 xi + 0.25 yi + 0.5 zi

Reply

Replies

Vincent Williams July 22, 2015 at 9:37 PM

Dont the equations in this method need to be altered as well?
function block_projection_overlap(obj)
{
function interval_overlap(a1,a2,b1,b2)
{
return a1>=b1&&a1=a1&&b1<a2;
}

return interval_overlap(
this.x-this.y-this.yy,this.x+this.xx-this.y,
obj.x- obj.y- obj.yy, obj.x+ obj.xx- obj.y)&&
interval_overlap(
this.x-this.z-this.zz,this.x+this.xx-this.z,
obj.x- obj.z- obj.zz, obj.x+ obj.xx- obj.z)&&
interval_overlap(
-this.y-this.yy+this.z,-this.y+this.z+this.zz,
- obj.y- obj.yy+ obj.z,- obj.y+ obj.z+ obj.zz);
}

Joaquín M López Muñoz July 23, 2015 at 7:55 AM

No, these equations stay the same regardless of the orientation of 3D axes.

Reply

Unknown October 19, 2015 at 5:29 PM

I think I found a bug in your "isBehind" heuristic or I missed out on something here.
Consider the blocks as defined by:
[xA,yA,zA] = [0,0,0] (furthest vertex from camera)
[x'A,y'A,z'A] = [1,1,1] (closest vertex to camera)

and
[xB,yB,zB] = [0.5,0,0] furthest vertex from camera of block B
[x'B,y'B,z'B] = [1.5,1,1] closest vertex to camera of block B

Now they clearly intersect, but none of your 6 cases does apply.
How do I proceed then?

Reply

Replies

Joaquín M López Muñoz October 19, 2015 at 6:11 PM

As you correctly point out, the projections of these two blocks overlap yet none of the conditions 1...6 apply. The reason is that the blocks intersect **in 3D** (for instance, the point [0.75,0.75,0.75] is inside both A and B), and isBehind requires that this is not the case:

As objects are assumed to be convex and they do not overlap (i.e. A∩B = ∅)...

Marvin Neurath October 19, 2015 at 7:41 PM

> [...] if they do overlap, we can apply the following easily provable properties [...] Since the prisms A and B do not overlap (in 3D)[...]

I think it would be better here to use "intersect in 3D" rather than overlap. This was pretty confusing for me. And it would be great if you could mention what [x,y,z] and [x',y',z'] refers to. Took me a while to figure it out in the comments.

Anyways thanks for this article, it helped a lot!

Reply