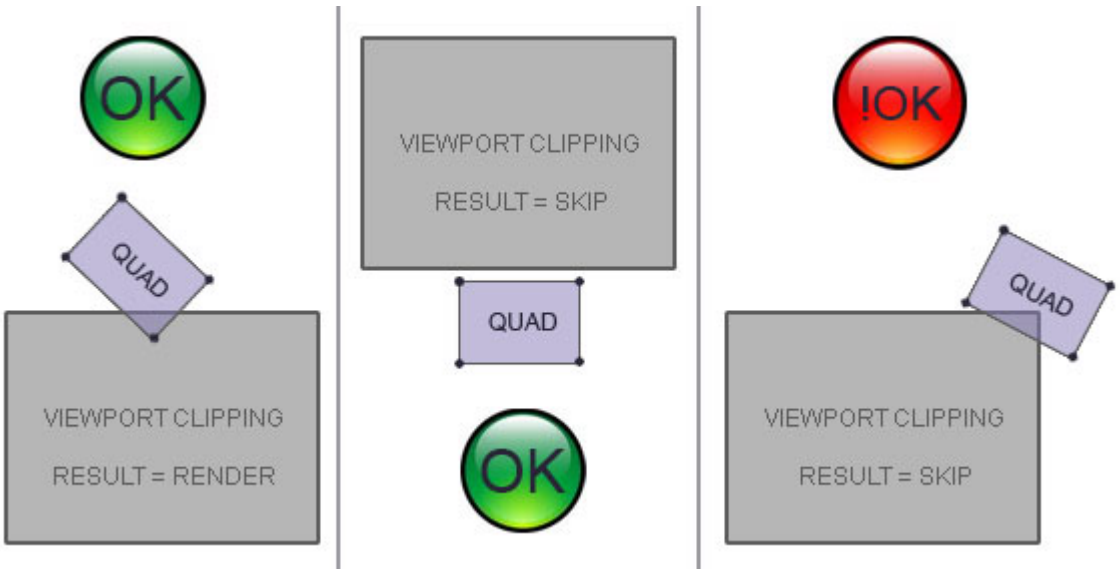
Part Ib – Intermezzo Improved Quad Clipping

You may or may not have noticed, but the clipping procedure in the DistortedPlane class we've been using can be improved, since it allows for false negatives. This is demonstrated when turning around fast in the panorama from the previous post: white triangles (or whatever your background color is) will pop up.

What is going on?
Basically our clipping method tests whether one of the four corner points of the quad-to-render is within the clipped region. If true, render, if false, skip. That test is very easy to understand: "hey part of me is within the visible region, so render me!" Unfortunately it is also very wrong.

Take a look at the following image:



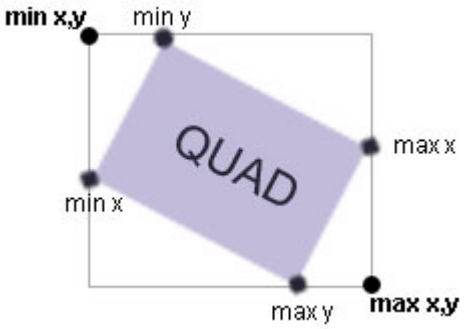
You see the first two cases are fine, however the last case is wrong. It is a false negative, our clipping test tells us we can skip it, but visually we can see it should have been drawn.

Logical, since we tested for the corner points only.
What we should have done is test whether any part of the quad-to-draw (referred to as quad in the rest of this post) is intersecting our clipping rectangle. One way to do that is to write an intersection test for the quad with the non distorted clipping rectangle, for example through a separating axis test.

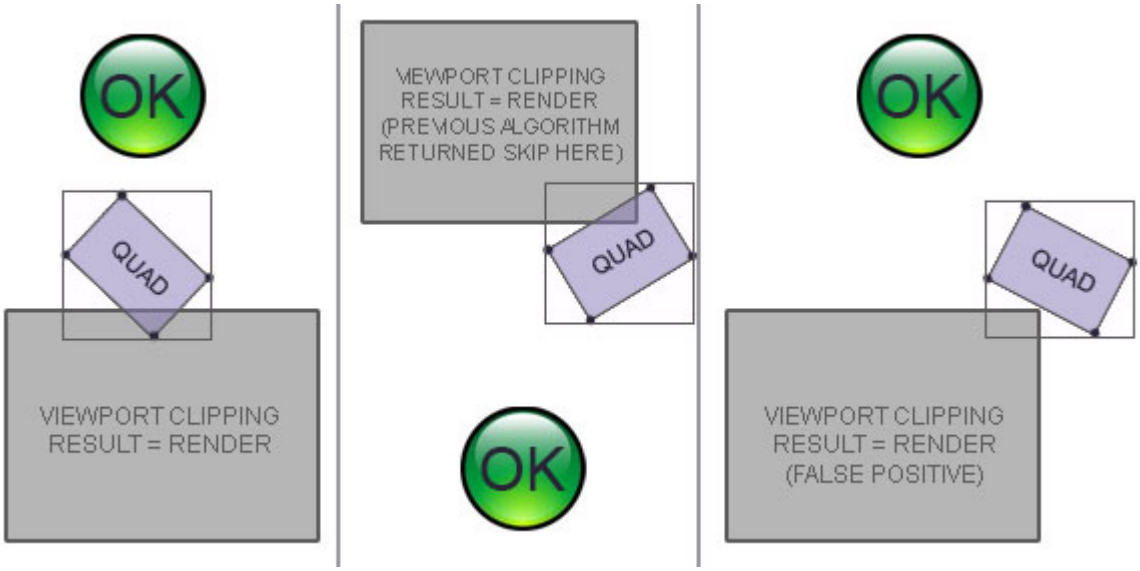
Problem is that we don't want to do that for each quad since our panorama will slow to a crawl. So what we will be doing instead is a bounding box test.

Bounding box test
What is the bounding box for a quad? Assuming we have the four points UL, UR, LL, LR that we've been using in each post, the upperleft and lowerright coordinates of the bounding box around these points are given by (in actionscript pseudocode):

Visually:



If we replace our clipping test with a bounding box intersection test between the clipping rectangle and a quad's bounding box we will get the following results:



So we see that with a bounding box test we get the opposite from what we had with the ‘clipping-rectangle-contains-at-least-one-quad-vertex’ test: we might get false positives. In other words now and then a quad might be rendered which is outside the clipping rectangle. We don’t care. It won’t happen a lot and it’s better than not rendering false negatives.

But how do we implement a bounding box or in other words a rectangle to rectangle intersection test?
There are different options, one simple option would be to create two rectangle instances representing our rectangles and call the intersect method on them. This is bad for two reasons:

a) we would have to create new objects (although we could reuse these objects)
b) we would have to sort our points so that we know which point is upperleft and which point is lowerright in order to create these rectangles right (which is sloooow).

So assuming we roll our own intersection test, how does the intersection test work without sorting the points?
Intersection tests might be hard to understand (or not depending on your math skill), a lot of the explanations I’ve found online talk about overlapping coordinates, which gets hard to visualize mentally fast if we can’t even be sure which

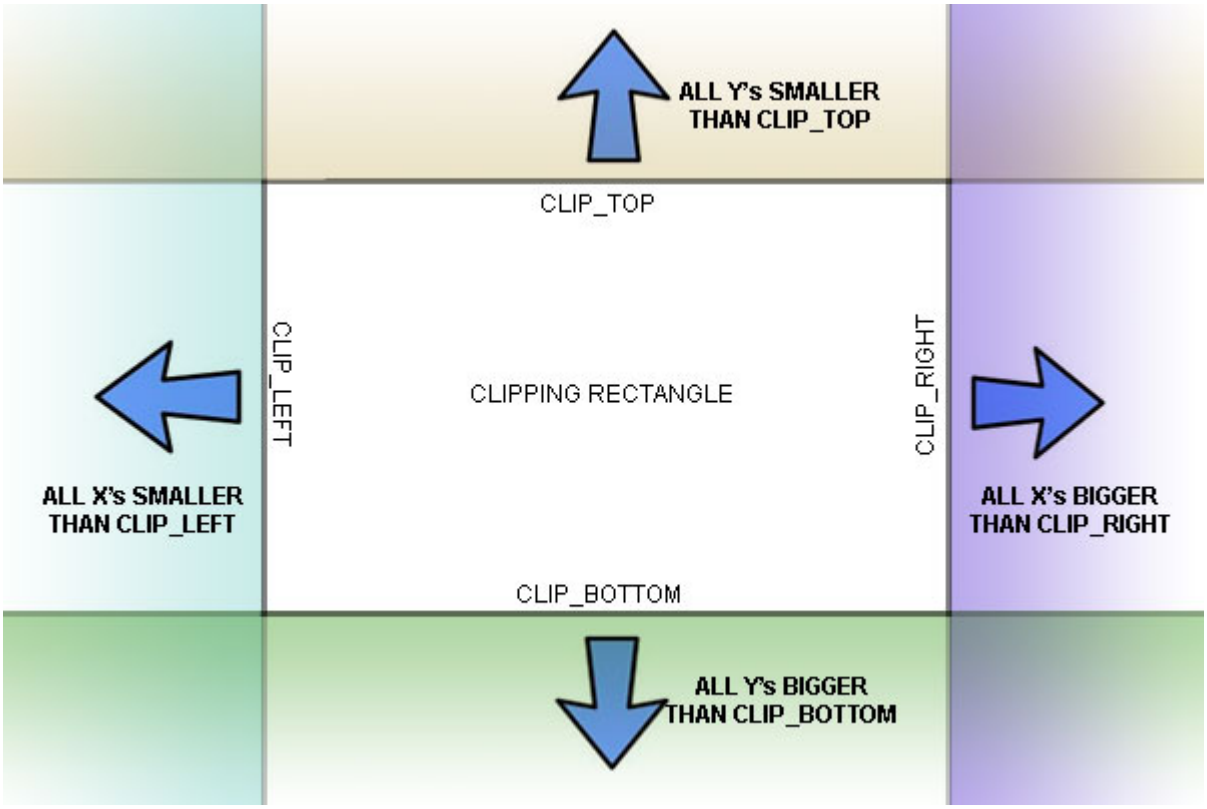
point is which.

I find this the easiest way to look at the problem: when is there no intersection?

Seeing our clipping plane has x boundaries clipLeft and clipRight and y boundaries clipTop and clipBottom, we can safely say there is absolutely no possibility of an intersection, if:

- all the points of our quad have an x lower than clipLeft OR
- all the points of our quad have an x higher than clipRight OR
- all the points of our quad have an y lower than clipTop OR
- all the points of our quad have an y higher than clipBottom (clipTop being smaller than clipBottom)

Take a look at the following image for a visual explanation, this image shows all the equations where the quad does not intersect the clipping rectangle:



So when DOES the quad intersect the clipping rectangle? Answer: in all other cases: not all x’s are to the left or right of the clipping rectangle and neither are all y’s: the quad’s bounding box must be hitting some part of the clipping rectangle!

For our DistortedPlane we replace our faulty clipping test with:

Note that if we had sorted our points this test would reduce to a much simpler rectangle-rectangle intersection test which can be found all over google. The same principle applies but since the coordinates are sorted, we no longer have to test ALL coordinates but only specified coordinates based on which side the intersection test is being performed:

but all the min/max operations are killing so we’ll do it the other way.
Next time an updated 3d Panorama in Actionscript 2 with a lot of issues fixed!