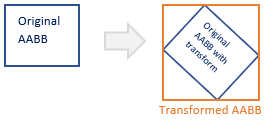
# COMP270: 3D Computational Geometry Worksheet 1 - **Answers**

1. Consider the set of five points (7, 11, -5), (2, 3, 8), (-3, 3, 1), (-5, -7, 0) and (6, 3, 4).  
   An *axis aligned bounding box (AABB)* is the smallest box whose edges are aligned with the coordinate axes that contains all the points, defined by its minimum and maximum vertices **p**min and **p**max.
   1. What are **p**min and **p**max for the above five points?  
      The minimum and maximum points are simply the ones made up of all the smallest/largest x, y and z values out of all the points:  
      **p**min = (-5, -7, -5), **p**max = (7, 11, 8)
   2. List all eight vertices of the AABB.  
      These are just the various combinations of the min and max x, y and z values:  
      (-5, -7, -5), (-5, -7, 8), (-5, 11, 8), (-5, 11, -5), (7, -7, -5), (7, -7, 8), (7, 11, 8), (7, 11, -5)
   3. Determine the centre point **c** of the AABB.  
      For a general set of points, the centre point is just their average position. As our AABB is symmetrical, its centre is half-way between **p**min and **p**max:
   4. Multiply the five points by the following matrix (a 45° rotation about the z-axis):
   5. What is the AABB of these transformed points?  
      **p**min = (-4.242, -8.484, -5), **p**max = (2.121, 12.726, 8)
   6. What is the AABB we get by transforming the original AABB? (i.e. the bounding box of the transformed corner points).  
      Before looking at how to find this, note how *not* to:

* if we just transformed the original AABB, we would get a box of the same dimensions but that is *not* axis aligned;
* we can’t simply transform the original **p**min and **p**max, as you can hopefully see from the diagram below

To get a new ‘transformed’ AABB, we need to find the AABB of the transformed original AABB:  
We could do this by transforming all eight points of the original AABB, and then finding *their* **p**min and **p**max, but a more efficient method is to calculate the new min/max points directly, using the fact that we know what effect the transform will have on each of the input points – i.e. we can find formulae for each transformed point in terms of a linear combination of the original points, as described by the matrix:  
Since we would find the new min/max values by choosing the components from each transformed point, we just have to decide which combination of the min/max values of x, y and z will give the min/max values of x’, y’ and z’, which in this case gives us:  
increased efficiency of not having to transform all the points may or may not compensate for the less-than-optimal AABB size. (For more details on this approach, please see section 9.4 of “3D Math Primer for Graphics and Game Development”, by Fletcher Dunn and Ian Parberry).

1. A robot is at the position (1, 10, 3) and her right, up and forward vectors (expressed in world space) are , and respectively (note that these vectors form an orthonormal basis).  
   One way to interpret this is to think of the right, up and forward vectors as the axes of the robot’s local space, so that a point **p**l = (, , ) in that space is a linear combination of distances along the local axes, i.e. the world-space vector from the robot’s position **r** to the point in question, **p**w, is given by:  
   Expressed in this way, it’s easy to see that, to get the full world-space coordinates, we need to add the robot’s position to the linear combination:  
   … which shouldn’t really be a surprise, as this is exactly what the matrix “is” :D (You may even recognise it as a rotation of 30° about the y-axis, followed by the translation).  
     
   From the above, we can obtain the following functions to convert local coordinates to world-space ones:  
   The following points are expressed in object space; calculate their coordinates in world space:
   1. (-1, 2, 0)  
      (0.866 x (-1) + 0.5 x 0 + 1, 2 + 10, -0.5 x (-1) + 0.866 x 0 + 3) = (0.134, 12, 3.5)
   2. (1, 2, 0)  
      (0.866 x 1 + 0.5 x 0 + 1, 2 + 10, -0.5 x 1 + 0.866 x 0 + 3) = (1.866, 12, 2.5)
   3. (0, 0, 0)  
      (0.866 x 0 + 0.5 x 0 + 1, 0 + 10, -0.5 x 0 + 0.866 x 0 + 3) = (1, 10, 3)  
      … The robot’s position is the origin of its local space!
   4. (1, 5, 0.5)  
      (0.866 x 1 + 0.5 x 0.5 + 1, 5 + 10, -0.5 x 1 + 0.866 x 0.5 + 3) = (2.116, 15, 2.933)
   5. (0, 5, 10)  
      (0.866 x 0 + 0.5 x 10 + 1, 5 + 10, -0.5 x 0 + 0.866 x 10 + 3) = (6, 15, 11.66)  
        
      The coordinates below are in world space; find their positions relative to the robot:  
      We now need to convert the coordinates that are a linear combination of our standard basis vectors (the usual x, y and z axes) into a linear combination of the robot’s axes by finding the local coordinates/coefficients.  
        
      One way to do this is to rearrange the functions we were using to do the object-to-world conversion to isolate the world coordinate values:  
      Substituting this into the formula for gives in terms of world coordinates:  
      Now putting this back into the expression for :  
        
      Finally, the y coordinates aren’t affected by the others, so our final functions are:  
      ourse, we could also achieve the same result by finding the inverse matrix, either using numerical methods or by constructing the matrices for the reverse transformations – rotating by -30° about the y-axis and translating by (-1, 10, 3) – and multiplying them, remembering that the translation must be applied first in this case:  
      , but let’s use the formulae again this time:
   6. (1, 10, 3)  
      (0.866 x 1 - 0.5 x 3 + 0.634, 10 - 10, 0.5 x 1 + 0.866 x 3 - 3.098) = (0, 0, 0)  
      … As we would expect, since this is the robot’s position in world space.
   7. (0, 0, 0)  
      (0.866 x 0 - 0.5 x 0 + 0.634, 0 - 10, 0.5 x 0 + 0.866 x 0 - 3.098) = (0.634, -10, -3.098)  
      If we view this transformation as one that moves an object around in world space, then rotating the origin has no effect (since rotations are about the origin) and the net result is the translation values.
   8. (2.732, 10, 2)  
      (0.866 x 2.732 - 0.5 x 2 + 0.634, 10 - 10, 0.5 x 2.732 + 0.866 x 2 - 3.098) = (2, 0, 0)
   9. (2, 11, 4)  
      (0.866 x 2 - 0.5 x 4 + 0.634, 11 - 10, 0.5 x 2 + 0.866 x 4 - 3.098) = (0.366, 1, 1.366)
   10. (1, 20, 3)  
       (0.866 x 1 - 0.5 x 3 + 0.634, 20 - 10, 0.5 x 1 + 0.866 x 3 - 3.098) = (0, 10, 0)