CMSC425 Fall 2019
Homework 2: More geometric exercises
Solutions

Part a. Warm up problems

1. Given this idea, show the value of q after multiplication, and then after normalization.

$$q = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 3 & 0 \\ 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 15 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 0.667 \\ 5.000 \\ 0.667 \\ 1 \end{bmatrix}$$

2. Assuming we have a 2 by 2 by 2 cube centered at the 3D homogenous point p=(1,3,1,1), give a sequence of homogeneous matrices to rotate the square 45 degrees clockwise around the X axis first, and then 60 degrees around the Z axis, both rotations around the center of the object.

$$M_T = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \ and \ {M_T}^{-1} = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -3 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

And

$$M_{Rx} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos -45 & -\sin -45 & 0 \\ 0 & \sin -45 & \cos -45 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \ and \ M_{Rz} = \begin{bmatrix} \cos 60 & -\sin 60 & 0 & 0 \\ \sin 60 & \cos 60 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The combined matrix is

$$M = M_T * M_{Ry} * \ M_{Rx} * {M_T}^{-1}$$

3. Do similar actions for a box in 3D centered at p=(1,3,1,1), but this time do the following. Scale the box by 2 in the y direction, so it has a preferred direction, and rotate it by 45 degrees around the x-axis and then 30 degrees around the z-axis.

$M_T \ and \ {M_T}^{-1}$ are the same as in Problem 2

And

$$M_{Rx} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos 45 & -\sin 45 & 0 \\ 0 & \sin 45 & \cos 45 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \ and \ M_{Rz} = \begin{bmatrix} \cos 30 & -\sin 30 & 0 & 0 \\ \sin 30 & \cos 30 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

And

$$M_s = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The combined matrix is

$$M = M_T * M_{Ry} * \ M_{Rx} * M_s * {M_T}^{-1}$$

4. Give Unity Vector3 methods to carry out problem ~~(4)~~ (3).

```
Transform M = Quaternion.identity;
M.translate(new Vector3(-1,-3,-1));
M.localScale += new Vector3(0, 1.0F, 0); // Original was 1,1,1
M.rotate(new Vector3(45,0,0), Space.self);
M.rotate(new Vector3(0,0,30), Space.self);
M.translate(new Vector3(1,3,1));
```

5. Give a Unity command to rotate 60 degrees around the vector <2,2,1,0>. Just one line …

```
Vector3 axis = new Vector3(2.0f, 2.0f, 1.0f);
transform.rotation = Quaternion.AngleAxis(60, axis);
```

6. Consider the problem of ray circle intersection (see the handout on the web page). Describe how you might simplify the problem by rotating the data so one of the vectors involved aligns with an axis.

Solution:

Version 1:

a) Compute the angle theta between v and the y-axis (or x, not critical which).

b) Rotate the entire system (most importantly the center point) by theta.

c) Now the distance from the center to the y-axis, and that's the distance from the point to the y-axis. If this is less

*But – which way do we rotate, cw or ccw? With dot product we know the magnitude of theta but not the sign. Need to account for this. You can embed 2d vectors in 3D as (x,y,0) and use the cross product to compute this.

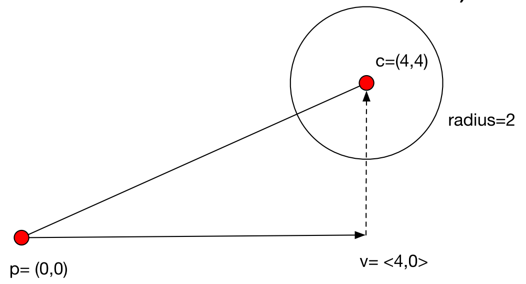
Version 2:

a) Create the rotation matrix directly by putting the vector v in the matrix

$$M_r = \begin{bmatrix} -v_y & v_x & 0 \\ v_x & v_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

b) Rotate the entire system (most importantly the center point) by this matrix

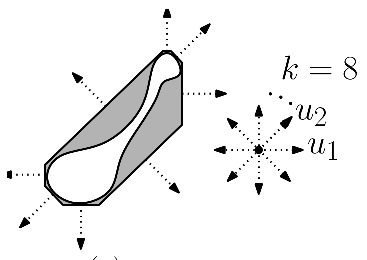
c) Now the distance from the center to the y-axis is the distance to the line, and if this is less than the radius, the line intersects the circle.



Part b. Application
These are intended as applications of the formulas to game design problems.

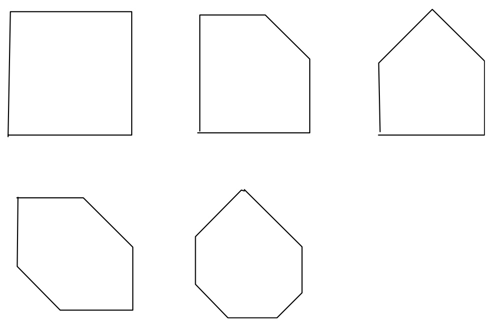
1. **8-DOP 2D collider**. A 4-DOP 2 collider is just an AABB box. An 8-DOP collider adds four more diagonal sides with slopes of -1 and 1, as below.

Assume an 8-DOP is represented by the polygon of 4 to 8 points (4 would be a square). We'll have the convention that the first point is the lower left most, and the polygon points go counterclockwise (see below).



(e)

a) Draw three examples of degenerate 8-DOPs – where they don't have 8 points. Use this an opportunity to explore the space of possible 8-DOPs. Here's cases with 4, 5, 5, 6 and 7 points.



b) Related to the Cohen Sutherland algorithm, given two lines, one horizontal or vertical, and the other with +1 or -1 slope, how would you compute the intersection?

There are four cases:

Horizontal with +1 slope	Vertical with +1 slope
Horizontal with -1 slope	Vertical with -1 slope

We'll do the first case, and assume the others are similar.
Given we're intersecting a vertical line with one that has a+1 slope, we need forms of the lines.

Assume the vertical line is represented by the x value xv.
Assume the other line is represented by y = mx + b with m = +1 so we have y = x + b
The intersection is then (xv, xp+b).

Or

Assume the vertical line is the same with x = xv.
Assume the other line is represented by a point and a vector, p(t) = <dx,dy>t+(px,py)
And we know the vector v=<dx,dy> = <1,1>
Then the intersection is (xp, (px-vx) + py)
The change in y is equal to the change in x.

c) Given a set of n points, how would you generate an 8-DOP? (This is a simplified version of convex hull). Consider the following issues:

How to find the extreme (eg, maximal) points in the 8 directions.
How to find the intersection points between the line segments in each direction.
How to determine if the 8-DOP is degenerate in any direction so that line is length 0.

To find the extremal points in every direction, the key is projecting each data point onto the vector in that direction, using the dot product. But, since our direction vectors are all relatively simple, the dot product is a simple sum of the elements of the vector

<vx,vy> • <1,0> = vx   <vx,yv> • <1,1> = vx + vy

Also, you only have to project once on a vector in each direction. If the result is positive you are could the positive in that direction; if the result is negative, you could be the minimum.

Finally, you don't need to take the sqrt to compute the actual distance from origin in a direction. Since a < b if sqrt(a) < sqrt(b), it's enough to store the dot product and not its sqrt.

For this 8-DOP, we have four fundamental directions and vectors:

Directions = [<1,0>, <1,1>, <0,1>,<-1,1>]

We have the maximum values in each direction, assume initialized to largest and smallest float.

```
float dopMax[4];
float dopMin[4];
```

The algorithm:

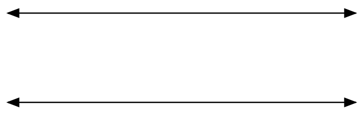
```
For each point p in the data set, do the following:
    // Project onto <1,0> vector with the dot product:
    dist = (px,py) • <1,0> = px
    if (dist > dopMax[0]) dopMax[0] = dist else if (dist < dopMin[0]) dopMin[0] = dist
    // Project onto <1,1> vector with the dot product:
    dist = (px,py) • <1,1> = px + py
    if (dist > dopMax[1]) dopMax[1] = dist else if (dist < dopMin[1]) dopMin[1] = dist
    // Project onto <0,1> vector with the dot product:
    dist = (px,py) • <1,1> = py
    if (dist > dopMax[2]) dopMax[2] = dist else if (dist < dopMin[2]) dopMin[2] = dist
    // Project onto <1,1> vector with the dot product:
    dist = (px,py) • <-1,1> = -px + py
    if (dist > dopMax[3]) dopMax[3] = dist else if (dist < dopMin[3]) dopMin[3] = dist
```

How to determine if the 8-DOP is degenerate in any direction so that line is length 0.
An 8DOP in 2D is degenerate at a corner point if the two adjacent maxs (mins) are px and py when the corner point is at px+py
An 8DOP in 2D is degenerate at top, bottom or side point if the two adjacent maxs (mins) are some gx and gy, and the max (min) in the side direction is gx+gy.

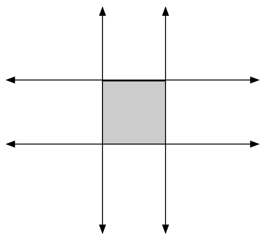
How to find the intersection points between the line segments in each direction.
To find the intersection points, you need to also find the point of max (min) in each direction. The point plus the direction gives you a line. Then you can intersect two adjacent lines to find

d) Finally, sketch how you'd intersect two 8-DOPs – what comparisons and computations would you make in which order?

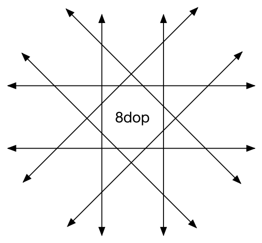
The key here is that a K-DOP can be considered the intersection of infinite slabs. Assume a slab is rectangle that is infinite in one direction



A 4DOP or square is the intersection of two slabs:



And an 8DOP is the intersection of four slabs.



So we just need to test if two KDOPs intersect for each slab

```
int TestKDOPKDOP(KDOP &a, KDOP &b)
{
   // Check if any intervals are non-overlapping, return if so
   for (int i = 0; i < 4; i++)
      if (a.min[i] > b.max[i] || a.max[i] < b.min[i])
         return 0;
   // All intervals are overlapping, so k-DOPs must intersect
   return 1;
}
```