Eva Fan-Yi Kung
998137573, g3ekungg

CSC418 Assignment #1 (Part 1)

Q1)

$$Let\ f(t) = (x(t), y(t)),\ where\ x(t)$$
$$= 4cos(2\pi t) + 1/16cos(32\pi t),\ y(t) = 2sin(2\pi t) + 1/16sin(32\pi t)$$

Tangent Vector,

$$f'(t) = (x'(t), y'(t))$$
$$= (-8\pi sin(2\pi t) - 2\pi sin(32\pi t),\ 4\pi cos(2\pi t) + 2\pi cos(32\pi t))$$

Normal Vector,

The dot product must equal to zero because normal vector is perpendicular,, therefore,

$$(-y'(t),\ x'(t))$$
$$= (-4\pi cos(2\pi t) - 2\pi cos(32\pi t),\ -8\pi sin(2\pi t) - 2\pi sin(32\pi t))$$

Symmetric along x-axis (y=0),

By definition, the wiggly ellipse in 2D will be symmetric along the x-axis if (x,y) = (x,-y), that is, when all y values are replaced with its negation, the equation still holds (in equivalence)

For all arbitrary points,
f(t) = (x(t), **y(t)**) = (x(t), **-y(t)**) for all t from 0 to 2pi should satisfy.

Along y-axis (x=0) ,

Same idea as previous but with all x values replaced such that (x,y) = (-x, y), However, in this case, the wiggly ellipse is not symmetrical along the y axis, so the value you get is not the same.

ie) Counter example
Along x=0, y remains the same.
At t = 0, the x coordinate is x(0) = 4.25,
however, when the value on the other end of the y-axis,
at t=0.5, x(π /2)= - 3.88 , since x(0) != x(0.5) , they are not symmetrical

The curve's perimeter is also the same as computing the arclength of the curve from 0 to 2 π

$$Length = \int_0^{2pi} \sqrt{(x'(t)^2 + y'(t)^2)}dt = \int_0^{2pi} \sqrt{((-8\pi sin(2\pi t) - 2\pi sin(32\pi t))^2 + (4\pi cos(2\pi t) + 2\pi cos(32\pi t))^2)}dt$$

Since the function f(x) = (x(t), y(t)) is a graph of a wiggly ellipse, piecewise approximation can be done through calculation/collecting each segments line integral (length of each piece) and take the total sum. For instance, it can be calculated through taking the length of each quadrant.

Q2)

$$Area\ of\ a\ circle\ =\ \pi r^2$$

$$Area\ of\ donut\ =\ \pi r_2^2 - \pi r_1^2\ =\ \pi(r_2^2 - r_1^2)\ ,\ since\ r2 > r1$$

There are up to four intersections between a line and the donut boundary.

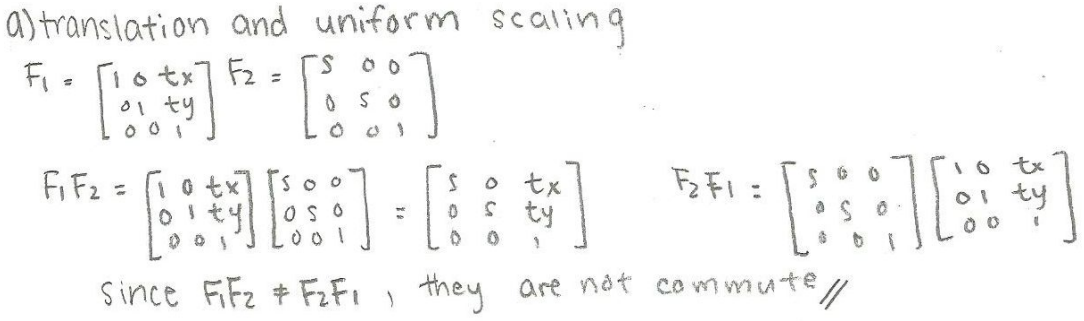
Algorithm to compute # of location(s) and intersections:

```
#pseudocode
Initialize array a to store pairs of locations
Let r1, r2 = radius of both circles in donut
Let p(lambda) = equation of line
Let d be the direction of the line

Substitute p(lambda) into the equation of the circle to find intersection
#||q-p(lambda)|| = r^2 for r1
Expand the equation
Apply the quadratic formula (to ensure all possible values are solvable)
#this gives us the values of the x coordinates crossing the circle(s)
Sub the lambda back into line equation and get y
Store value pairs into array a

Repeat the same steps for other circle, r2,
Store value pairs into array a

Store the founded x and y pair into array
loop array and output(return) each pair and size of array
```

If the graph transformed only on the donut, the number of locations may vary, since the donut boundaries might largen wide enough for the line to not reach them or vice versa.Also, the values of the actual locations will be changed, like the other case, the values of the new location of the intersections not be the same but the difference is that it will not be a direct scalar of the given scaling transformation.

If the graph transformed non uniformly on the donut and the line, the the number of locations remains the same, however, the location will be scaled by a factor of sx or sy from their original location.

Q3)



b) translation and non-uniform scaling

$$F_1 = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \quad F_2 = \begin{bmatrix} Sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$F_1F_2 = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} Sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} sx & 0 & tx \\ 0 & sy & ty \\ 0 & 0 & 1 \end{bmatrix} \quad F_2F_1 = \begin{bmatrix} Sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} sx & 0 & sxtx \\ 0 & sy & syty \\ 0 & 0 & 1 \end{bmatrix}$$

Since $F_1F_2 \neq F_2F_1$, they not commute //

c) ~~Scale~~ Scaling and rotation

$$F_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad F_2 = \begin{bmatrix} \cos t & -\sin t & 0 \\ \sin t & \cos t & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$F_1F_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} \cos t & -\sin t & 0 \\ \sin t & \cos t & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos t & -\sin t & 0 \\ \sin t & \cos t & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

since $F_1F_2 = F_2F_1$, they are commute

$$F_2F_1 = \begin{bmatrix} \cos t & -\sin t & 0 \\ \sin t & \cos t & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos t & -\sin t & 0 \\ \sin t & \cos t & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

//

d) scaling and scaling (assuming ~~it~~ can be / ~~can~~ is non-uniform)

$$F_1 = \begin{bmatrix} Sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad F_2 = \begin{bmatrix} rx & 0 & 0 \\ 0 & ry & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$F_1F_2 = \begin{bmatrix} Sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} rx & 0 & 0 \\ 0 & ry & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} SxRx & 0 & 0 \\ 0 & syry & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad F_2F_1 = \begin{bmatrix} rx & 0 & 0 \\ 0 & ry & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} Sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} rxSx & 0 & 0 \\ 0 & rysy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

since $F_1F_2 = F_2F_1$, they are commute //

e) Translation and Shear

$$F_1 = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \quad F_2 = \begin{bmatrix} 1 & hx & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$F_1F_2 = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & hx & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & hx & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \quad F_2F_1 = \begin{bmatrix} 1 & hx & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & hx & tx+hxty \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix}$$

Since $F_1F_2 \neq F_2F_1$, they are not commute. //

Q4)

(Working under the assumption that we do not need to worry about the optimal time constraint/limit in terms of efficiency)

Note: Procedures will be explained in pseudocode/english writing

One way to determine whether a point q is inside, outside, or on the edge of the triangle, is to check which side of the plane the edge/two points create:

Let E1, E2, and E3 be the corresponding edge pairs of (v0, v1), (v1, v2), and (v2, v0) in order.

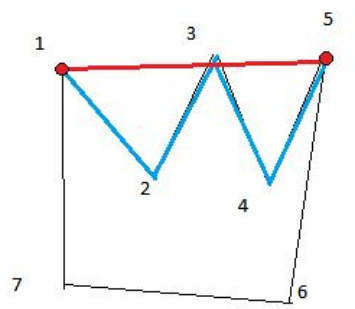
Let q=(q1, q2) as their coordinates

If the point is on a triangle's edge, calculate the directional vector of each edge, and substitute into the equation of a line , and substitute q1,and q2 into x and y coordinates correspondingly.

For condition of whether the point is outside or inside the triangle, we must check wither the points is within the boundaries. If the x coordinate is in between the lowest x and highest value of x within all three edges, and if the y coordinate is in between the low and highest value of y, then the point is within the triangle, otherwise, it is not (vice versa).

To compose a quadrilateral from two triangles, one edge from each triangle must be equal in order to join them together. First we would check each edge, and find out the length, and then look for a corresponding edge on the second triangle to union them together. (can be any leg from each shape as long as both triangles contain the same edge with the same length)

The procedure to create a n-sided convex polygon, is to split the given n -sided polygon into nonoverlapping mini polygons through straight diagonal lines connecting each point together.
Such that every four vertices, calculate the distance between the first and last end point, if it does not cross/is a reachable point to for a 4 sided polygon, connect the edges. Continue until finish. Afterwards, split each 4 sided polygon into two triangles, since two points in between will share the same edge in length and will make a valid triangle. Repeat until the end.



This procedure will not work for concave polygons, here is an example:

To connect edge (1,2), (2,3), (3,4), (4,5) , at point 1 and 5 when you try to connect the edges it will not work because it is not within bounds. Especially since concave polygons contain interior angles greater than 180 degrees, the two end edges (1,2) and (4,5) will not be able to connect without a problem.

In order to perform the test on a convex polygon, it can be applied from using the procedure in part 1, onto individual triangles. Using the procedure for triangulating a quadrilateral previously, each edge created will be tracked and also every triangle that is formed (overlapping edges are ok). After splitting, the triangulation can be applied for each triangle relative to the point we want to check. However, we must take into consideration additionally for the cases of edges. Normally, if the point exists outside one triangle, it is still within the polygon, or would be inside another triangle unless it is on an edge(since it is just split into individual triangles), we know it cannot be outside the shape as a whole. However for edge cases, we have to check whether there is a triangle on the other side of its edge, if there isnt, then if a point is checked to be existing outside its edge, then it is out of bounds of the entire polygon returning false as a result.