CS174A Lecture 4

SIGGRAPH trailers from 2016

Going backwards,

https://www.youtube.com/watch?v=l1KO-lnHfps

And

https://www.youtube.com/watch?v=dQBJ0r5Pj5s

Today on https://www.dwitter.net/

Today on https://www.shadertoy.com/slideshow

Announcements & Reminders

- *10/16/22: A2 due; will be discussed during this week’s TA session*
- *10/27/22: Midterm Exam: 6:00 – 7:30 PM PST, in person, in class*
- *12/06/22: Final Exam: 6:30 – 9:30 PM PST, in person, in class*
- *Updated syllabus on Canvas*

Team Project

- ***General Info***
  - Team sizes: 3-4
  - Expectations scale with size, e.g., we expect advanced graphics like shadows, reflections, physics, picking, scene graphs, etc.
  - For example, 3 members = 1 advanced feature, 4 members = 2, etc.
  - Project must include basic topics of syllabus at least through 7th week of quarter; it should have interactive graphics
  - You can use tinygraphics, but no external libraries or frameworks are allowed (no Three.js)
  - Project assignments 1-4 should provide you the background needed for your project
  - Project discussions will occur during Friday TA sessions

Team Project

- ***Due Dates***
  - 11/08/22: initial draft of project proposals and team members
  - 11/10/22: midway demo
  - 11/22/22: final version of project proposals
  - 12/02/22: project presentations during Friday TA sessions
  - 12/02/22: team project code due
- ***Grading (total: 150 points)***
  - Prelim proposal: 5%
  - Final proposal + midway evaluation: 5%
  - Final demo + code + readme: 20%

Last Lecture Recap

- ***Primitives:*** ***points, vectors***
- ***Vectors:*** ***dot and cross products***
- ***Coordinate Systems:*** ***LH CS, RH CS***
- ***Matrices:*** ***square, zero, identity, symmetric, matrix operations, matrix properties***

Next Up

- ***Homogeneous Representation of Points & Vectors***
- ***Spaces: Vector & Affine***
- ***Shapes: lines, circles, polygons (triangles), polyhedrons***
- ***Transformations: translation, scaling, rotation, shear***
- ***Spaces:***
  - Model space
  - Object/world space
  - Eye/camera space
  - Screen space

Points vs Vectors

What is the difference?

Points have location, but no size or direction

Vectors have size and direction, but no location

Problem: We represent 3D points/vectors both as 3-tuples

Homogeneous Representation

Convention: Vectors and Points are represented as 4x1 column matrices, as follows:

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \boxed{0} \end{bmatrix} \qquad \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ \boxed{1} \end{bmatrix}$$

Switching Representations

Normal to homogeneous:

- Vector: append as fourth coordinate 0

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \rightarrow \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ 0 \end{bmatrix}$$

- Point: append as fourth coordinate 1

$$P = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} \rightarrow \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ 1 \end{bmatrix}$$

Switching Representations

Homogeneous to normal:

- Vector: remove fourth coordinate (0)

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

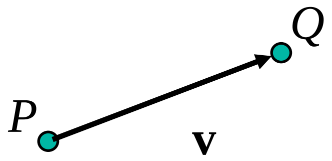
- Point: remove fourth coordinate (1)

$$P = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

Relationship Between Points and Vectors

A difference between two points is a vector:

$$Q - P = \mathbf{v}$$

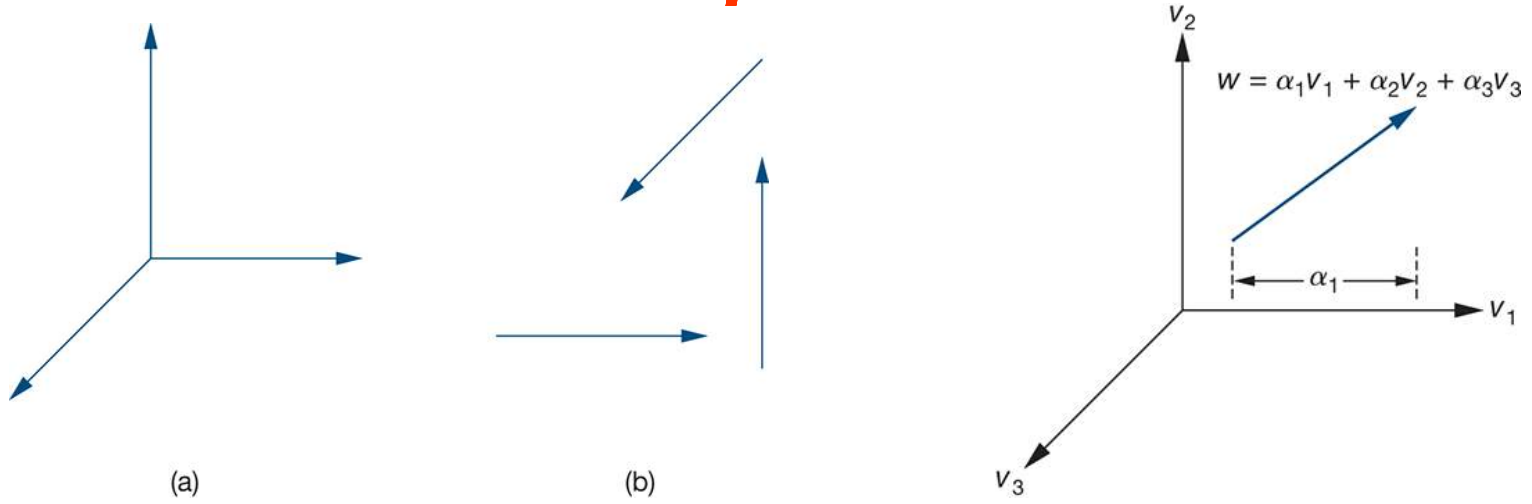


We can consider a point as a base point plus a vector offset:

$$Q = P + \mathbf{v}$$

Spaces & Frames

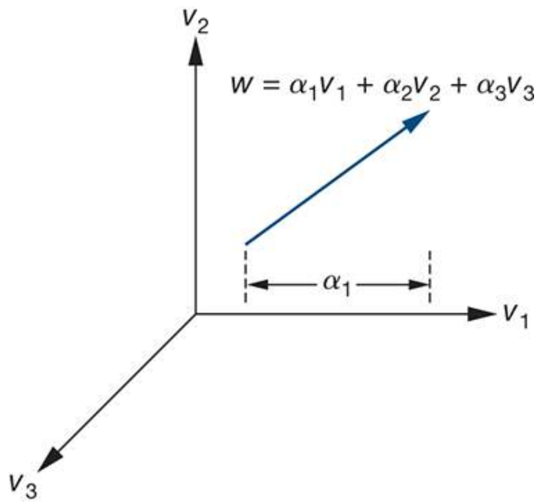
- ***Vector & Affine Spaces***

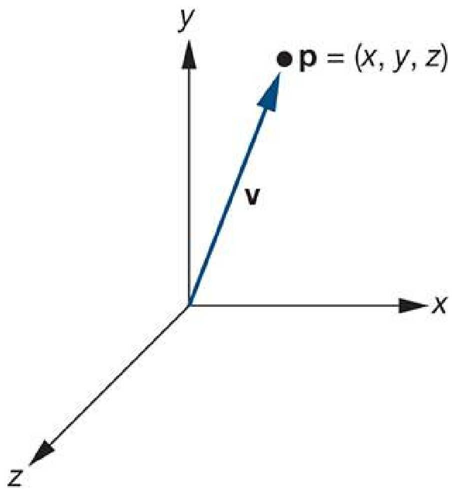


Basis = **Vector** Space: support only vectors, not points
Frame = Basis + PoR (origin) = **Affine** Space: support vectors & points
Basis defined by v_1,v_2,v_3; Frame defined by v_1,v_2,v_3,P_0

Coordinate System

- ***Vectors & Points***





$w = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3$

$v = P - P_0$

$P = P_0 + v = P_0 + \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3$

Homogeneous Representation

- ***Vectors & Points***

$v = \beta_1 v_1 + \beta_2 v_2 + \beta_3 v_3$

$P = P_0 + v = P_0 + \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3$

$$\begin{aligned} v &= \beta_1 v_1 + \beta_2 v_2 + \beta_3 v_3 \\ &= [\beta_1 \quad \beta_2 \quad \beta_3 \quad 0] \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{bmatrix} \end{aligned}$$

$$P = [\alpha_1 \quad \alpha_2 \quad \alpha_3 \quad 1] \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{bmatrix}$$

Does the Homogeneous Representation Support Operations?

Operations :

- $\mathbf{v} + \mathbf{w} = [v_1,v_2,v_3,0]^T + [w_1,w_2,w_3,0]^T$
  $= [v_1 + w_1,\ v_2 + w_2,\ v_3 + w_3,\ 0]^T$ Vector
- $a\mathbf{v} = a[v_1,v_2,v_3,0]^T = [av_1,\ av_2,\ av_3,\ 0]^T$ Vector
- $a\mathbf{v} + b\mathbf{w} = a[v_1,v_2,v_3,0]^T + b[w_1,w_2,w_3,0]^T$
  $= [av_1 + bw_1,\ av_2 + bw_2,\ av_3 + bw_3,\ 0]^T$ Vector
- $P + \mathbf{v} = [p_1,\ p_2,\ p_3,\ 1]^T + [v_1,\ v_2,\ v_3,\ 0]^T$
  $= [p_1 + v_1,\ \ p_2 + v_2,\ \ p_3 + v_3,\ 1]^T$ Point
- $P - Q = [p_1,p_2,p_3,1]^T - [q_1,q_2,q_3,1]^T$
  $= [p_1 - q_1,\ p_2 - q_2,\ p_3 - q_3,\ 0]^T$ Vector

Linear Combination of Points

Points P, Q scalars a, b:

$$aP + bQ = a\,[p_1, p_2, p_3, 1]^T + b[q_1, q_2, q_3, 1]^T$$
$$= [ap_1 + bq_1,\; ap_2 + bq_2,\; ap_3 + bq_3,\; a + b]^T$$

What is this?

Linear Combination of Points

Points *P*, *Q* scalars *a*, *b*:

$$aP + bQ = a\,[p_1, p_2, p_3, 1]^T + b[q_1, q_2, q_3, 1]^T$$
$$= [ap_1+bq_1,\ ap_2+bq_2,\ ap_3+bq_3,\ a+b]^T$$

What is it?

- If $(a + b) = 0$ then vector!
- If $(a + b) = 1$ then point!
- Otherwise, ??

Affine Combinations of Points

Definition:

n points P_i: i = 1,…,n

n scalars a_i: i = 1,…,n

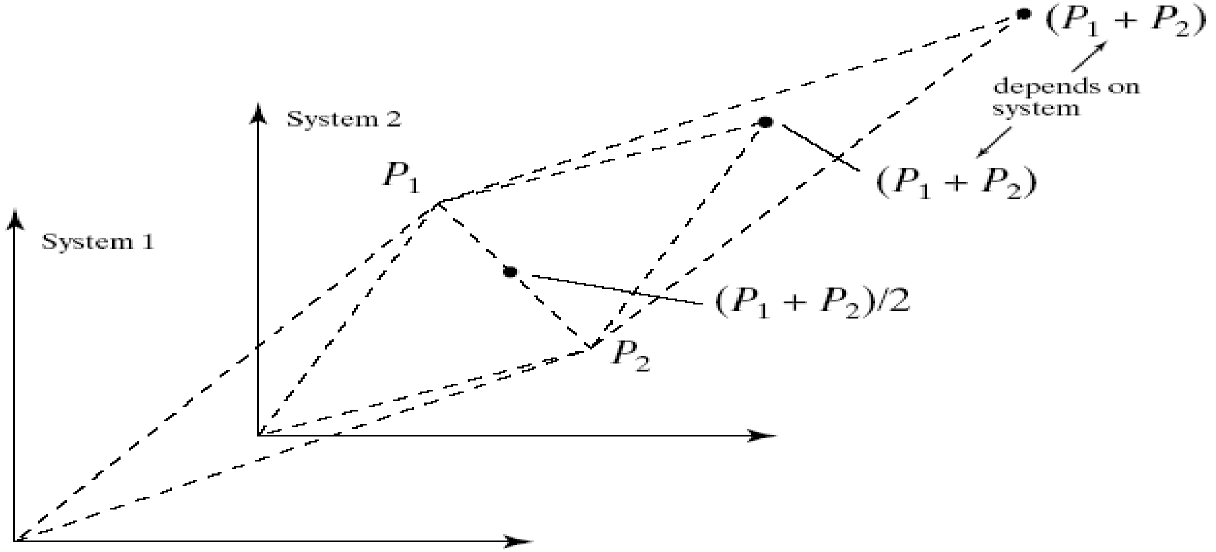
$$a_1P_1 + \ldots + a_nP_n \quad \text{iff} \quad a_1 + \ldots + a_n = 1$$

Example (n = 2): $0.5P_1 + 0.5P_2$

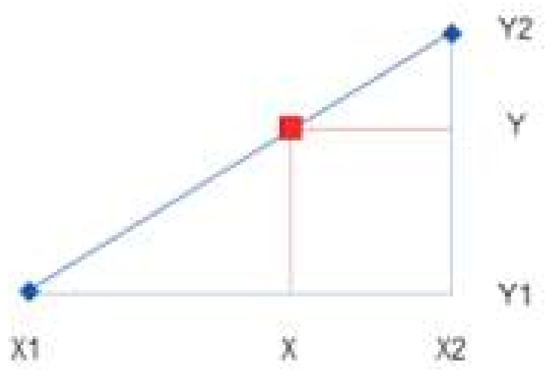
Example (n = 2): $(1\text{-}s)P_1 + sP_2$

Example (n = 3): $(1\text{-}s\text{-}t)P_1 + sP_2 + tP_3$

Geometric Interpretation



Exercise:



- List some points along a line from one point to another - This process is called convex interpolation

Linear interpolation (2 points)

- The formula to do that is quite short:

$$p_{interpolated} = (1-a) * p_1 + a * p_2$$

  - It’s only an interpolation (and called “convex”) if $0<=a<=1$
  - Otherwise it's an extrapolation
- You’ll be seeing that equation a lot

Linear interpolation (2 points)

- The formula to do that is quite short:

$$p_{interpolated} = (1-a) * p_1 + a * p_2$$

- Let (a) vary from 0 to 1 in steps - this is a parametric equation.
- Or we could imagine a parameter time (t) rather than (a) -- at each time t between 0 sec and 1 sec we reach a different point on the line segment. Now it's animated.

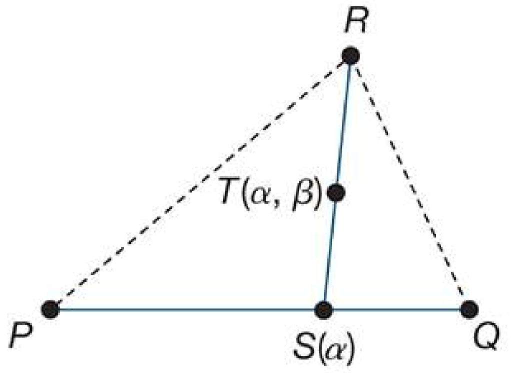
Linear interpolation (3 points = plane)

- Interpolation between 3 points

$$S(a) = (1-a) * P + a * Q$$

$$T(a,b) = (1-b) * S + b * R$$

$$T(a,b) = (1-b) * [(1-a) * P + a * Q] + b * R$$



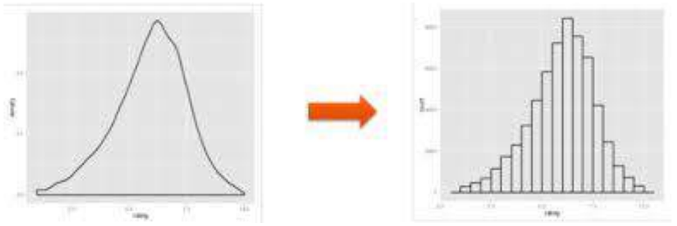
Making Shapes in Code

Computer graphics in practice

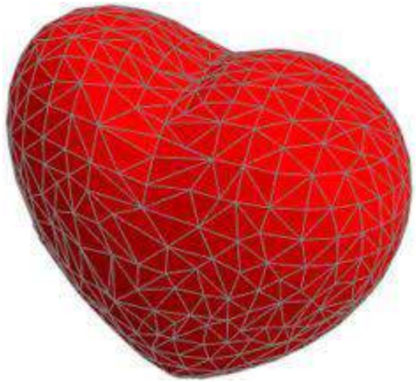
Summary

- Modeling
- Discretizing shapes (Vertices)
- Geometry
  - Data structures
  - Indexing

Discretization



- We don't know how to tell a computer to draw most shapes because of their complicated non-linear formulas.
- Instead, we linearize those shapes: Break them up into a finite number of line segments between N discrete points
- Piecewise planar shapes:



Polygon

Collection of points connected with lines

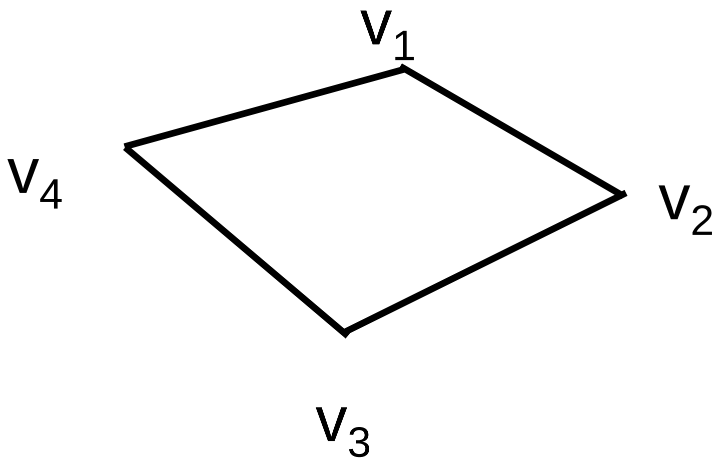
- Vertices: v_1,v_2,v_3,v_4
- Edges:

$e_1 = v_1v_2$

$e_2 = v_2v_3$

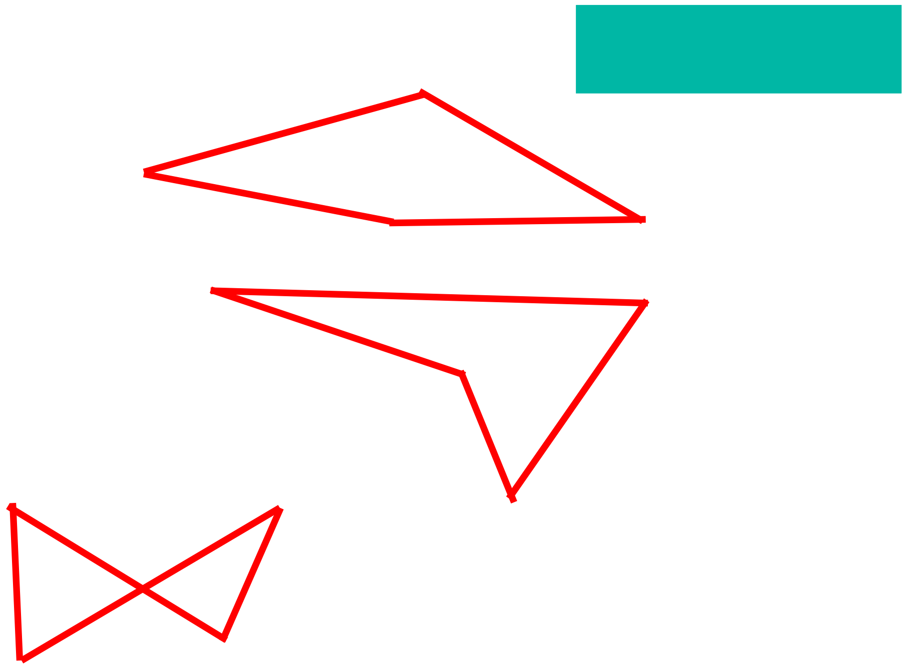
$e_3 = v_3v_4$

$e_4 = v_4v_1$



Polygons

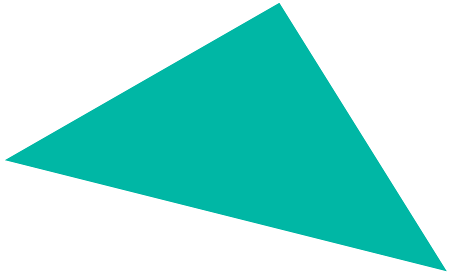
- **Closed** / open
- **Wireframe** / filled
- **Planar** / non-planar
- **Convex** / concave
- **Simple** / non-simple



Triangles

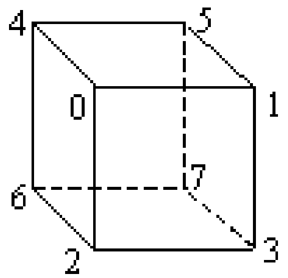
The most common primitive

- Simple
- Convex
- Planar



Polygonal Models / Data Structures

Indexed face set



faces

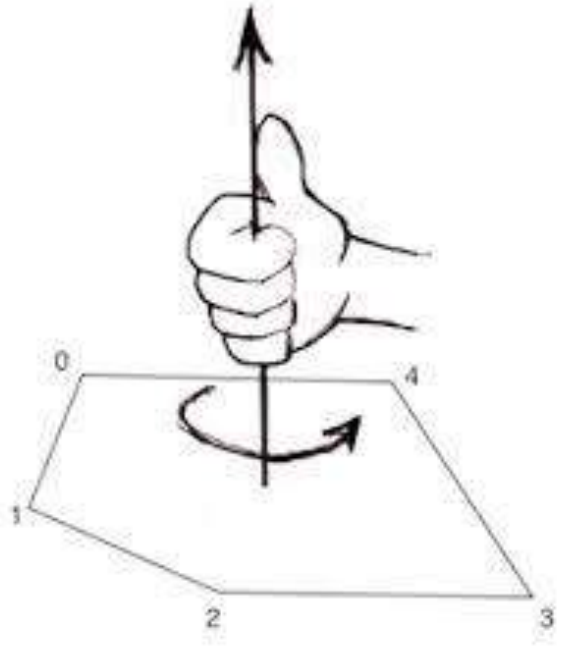
#	vertex list
0	0,2,3,1
1	1,3,7,5
2	5,7,6,4
3	4,6,2,0
4	4,0,1,5
5	2,6,7,3

vertex list

#	x,y,z
0	0,1,1
1	1,1,1
2	0,0,1
3	1,0,1
4	0,1,0
5	1,1,0
6	0,0,0
7	1,0,0

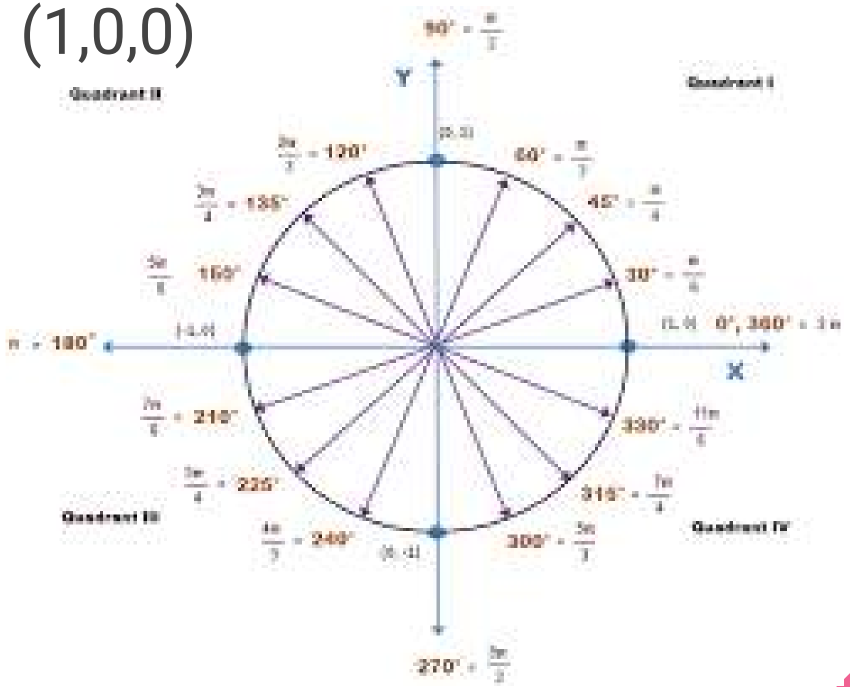
Polygonal Models / Data Structures

Face Normal



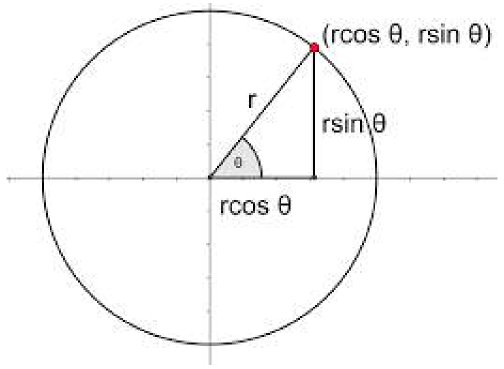
Let's list N points around a circle.

- First point: (1,0,0)



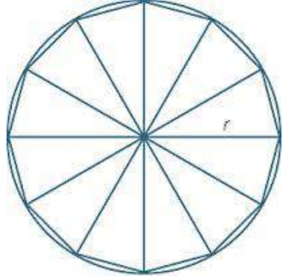
Let's list N points around a circle.

x = r*cos(Θ), y = r*sin(Θ) where theta is as shown below.



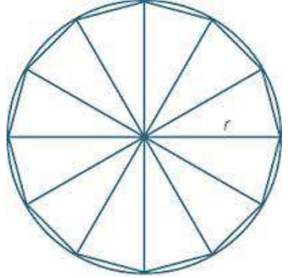
Using Θ as a variable input parameter, take N tiny steps from 0...2*PI.

Triangles



- We want to draw the whole 2D area, not just some points
    - Simplest 2d shape (remove any points and it will make it 1d) - this makes triangles the "2D simplex"

Triangles



- List the points in triangle order - two approaches:
    - Sort list into triples of points
        - (0,0), (1,0),(0.479, 0.878), (0,0), (0.479, 0.878), (0.841,0.540)...
        - Repeats are evident here
    - Or, make a separate list of sorted triples of indices
        - Indices are shorter to write, so more triangles can fit in a CPU cache:
        - 0,1,2,0,2,3,0,3,4,0,4,5,0,5,6,0,6,7...