ΗΥ416
ΓΡΑΦΙΚΑ ΥΠΟΛΟΓΙΣΤΩΝ

Αναγκαίες Μαθηματικές Έννοιες

Π. ΤΣΟΜΠΑΝΟΠΟΥΛΟΥ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Άλγεβρα

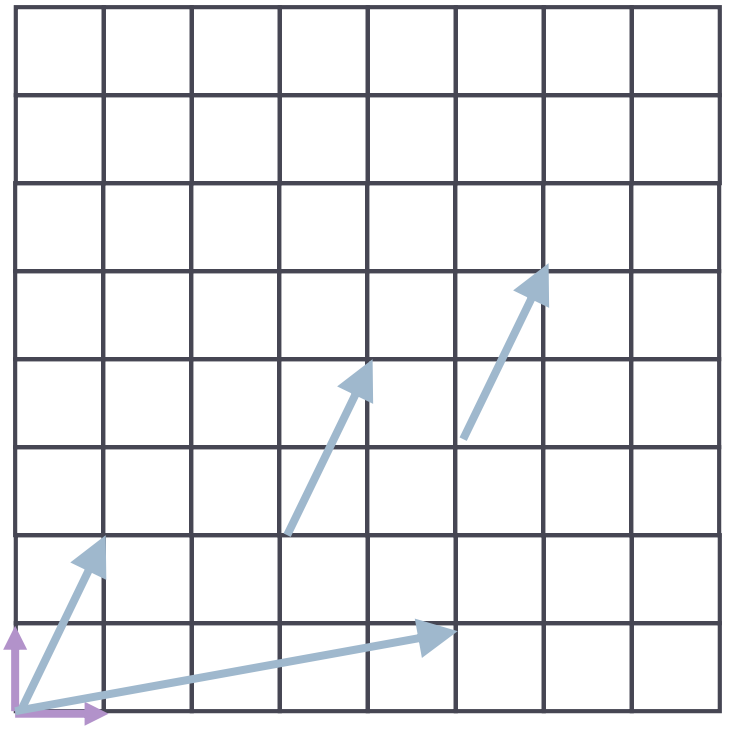
Συμβολισμοί Βαθμωτών, Διανυσμάτων, Πινάκων

- Βαθμωτός(scalar) a
  - (πεζά, πλάγια)
- Διάνυσμα(vector) $\mathbf{a} = \begin{bmatrix} a_1 & a_2 & \dots & a_n \end{bmatrix}$ ή $\vec{\mathrm{a}} = \begin{bmatrix} a_1 & a_2 & \dots & a_n \end{bmatrix}$
  - (πεζά, έντονα)
- Πίνακας(matrix)
  - (κεφαλαία, έντονα)

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Διανύσματα

- βέλος: μήκος και διεύθυνση
  - Προσανατολισμένο ευθύγραμμο τμήμα στον nD χώρο.
  - Προσοχή! Δεν έχει συγκεκριμένη αρχή και τέλος.



- offset / μετατόπιση
  - Με δεδομένο ένα σημείο αναφοράς (αρχή του συστήματος αξόνων) μπορούμε να υπολογίσουμε μετατοπίσεις διανυσμάτων.

Διανύσματα γραμμές - στήλες

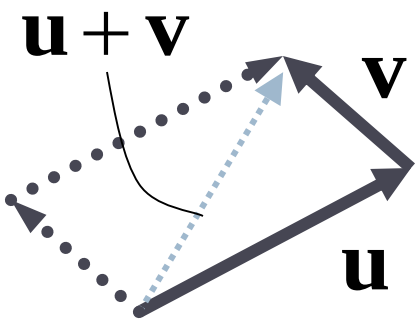
- Διάνυσμα γραμμή $\mathbf{a}_{row} = \begin{bmatrix} a_1 & a_2 & \dots & a_n \end{bmatrix}$
- Διάνυσμα στήλη $\mathbf{a}_{col} = \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_n \end{bmatrix}$
- Σχετίζονται με αναστροφή

$$\mathbf{a}_{col}^{T} = \mathbf{a}_{row}$$

Πρόσθεση Διανυσμάτων

- διάνυσμα + διάνυσμα = διάνυσμα
- Ισχύει ο κανόνας του παραλληλογράμμου

γεωμετρική



αλγεβρική

$$\mathbf{u}+\mathbf{v}=\begin{bmatrix} u_1+v_1 \\ u_2+v_2 \\ u_3+v_3 \end{bmatrix}$$

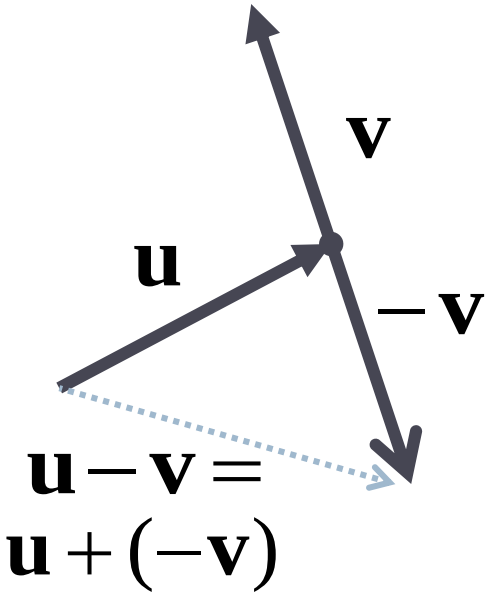
παραδείγματα:

$$(3,2)+(6,4)=(9,6)$$

$$(2,5,1)+(3,1,-1)=(5,6,0)$$

Αφαίρεση Διανυσμάτων

- διάνυσμα - διάνυσμα = διάνυσμα



$$\mathbf{u}-\mathbf{v}=\begin{bmatrix} u_1-v_1 \\ u_2-v_2 \\ u_3-v_3 \end{bmatrix}$$

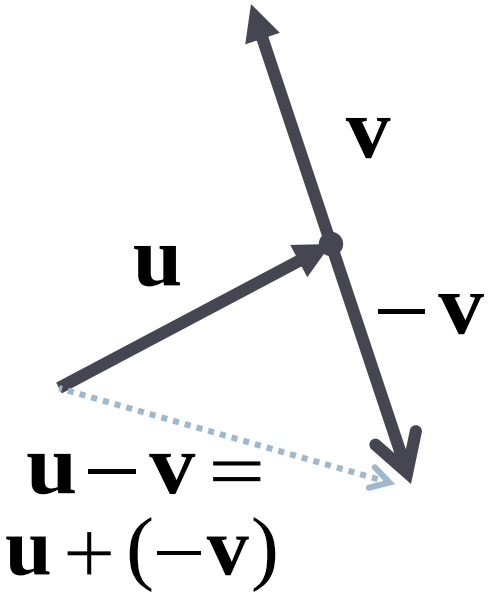
$$(3,2)-(6,4)=(-3,-2)$$

$$(2,5,1)-(3,1,-1)=(-1,4,0)$$

Αφαίρεση Διανυσμάτων

- διάνυσμα - διάνυσμα = διάνυσμα

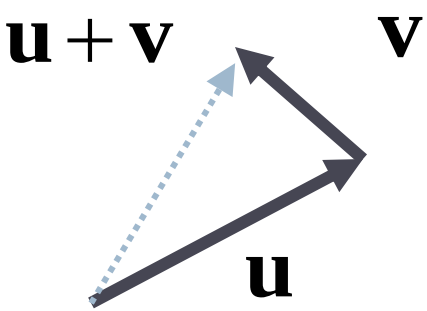
$$\mathbf{u}-\mathbf{v}=\begin{bmatrix} u_1-v_1 \\ u_2-v_2 \\ u_3-v_3 \end{bmatrix}$$

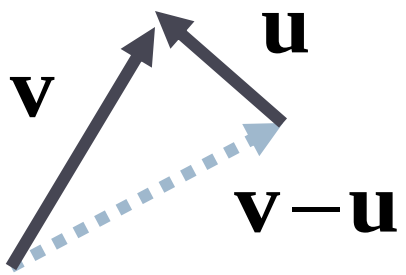


$$(3,2)-(6,4)=(-3,-2)$$

$$(2,5,1)-(3,1,-1)=(-1,4,0)$$

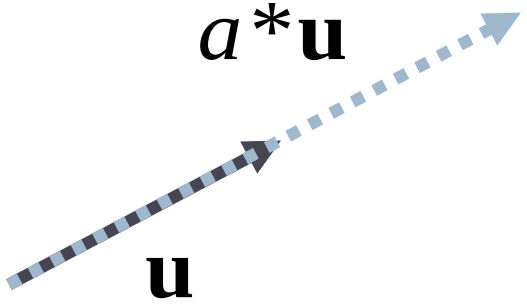
αντίστροφο επιχείρημα





Πολλαπλασιασμός βαθμωτού με διάνυσμα

- βαθμωτός * διάνυσμα = διάνυσμα
  - Το νέο διάνυσμα έχει κλιμακωθεί…



$$a*\mathbf{u} = (a*u_1, a*u_2, a*u_3)$$

$$2*(3,2) = (6,4)$$

$$.5*(2,5,1) = (1,2.5,.5)$$

Πολλαπλασιασμός διανυσμάτων

- διάνυσμα * διάνυσμα = βαθμωτός
- εσωτερικό γινόμενο διανυσμάτων
- ή και dot product $\mathbf{u} \bullet \mathbf{v}$

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \bullet \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = (u_1 * v_1) + (u_2 * v_2) + (u_3 * v_3)$$

Vector-Vector Multiplication

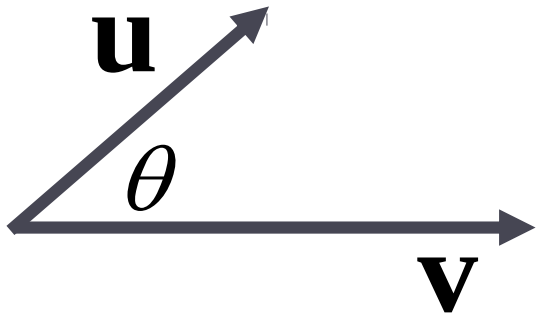
- multiply: vector * vector = scalar
- dot product, aka inner product

$$\mathbf{u} \bullet \mathbf{v}$$

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \bullet \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = (u_1 * v_1) + (u_1 * v_2) + (u_3 * v_3)$$

$$\mathbf{u} \bullet \mathbf{v} = \|\mathbf{u}\| \|\mathbf{v}\| \cos\theta$$

- geometric interpretation
  - lengths, angles
  - can find angle between two vectors

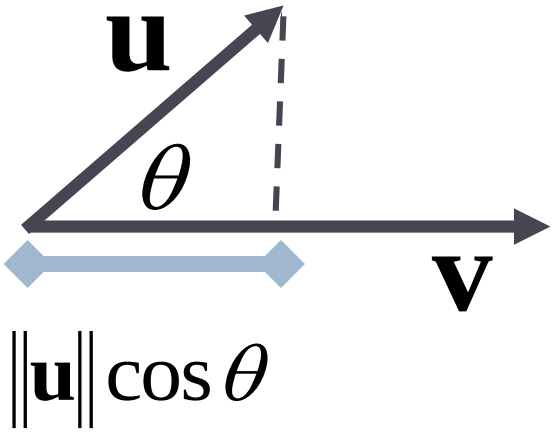


Εσωτερικό Γινόμενο (Dot Product)

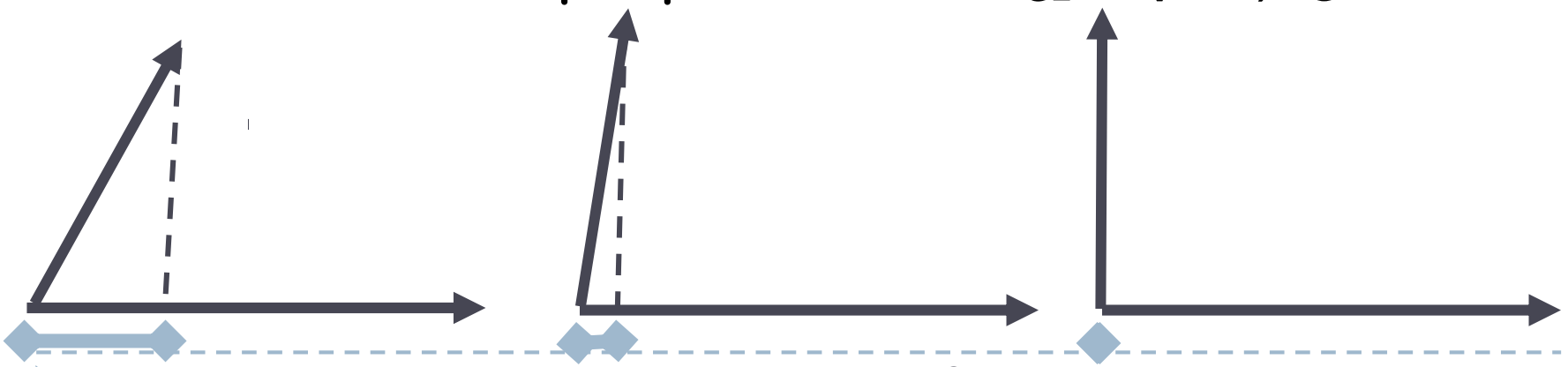
- can find length of projection of u onto v

$$\mathbf{u} \bullet \mathbf{v} = \|\mathbf{u}\|\|\mathbf{v}\| \cos\theta$$

$$\|\mathbf{u}\| \cos\theta = \frac{\mathbf{u} \bullet \mathbf{v}}{\|\mathbf{v}\|}$$



- as lines become perpendicular, $\mathbf{u} \bullet \mathbf{v} \to 0$



Dot Product Example

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \bullet \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = (u_1 * v_1) + (u_1 * v_2) + (u_3 * v_3)$$

$$\begin{bmatrix} 6 \\ 1 \\ 2 \end{bmatrix} \bullet \begin{bmatrix} 1 \\ 7 \\ 3 \end{bmatrix} = (6*1) + (1*7) + (2*3) = 6 + 7 + 6 = 19$$

Vector-Vector Multiplication, The Sequel

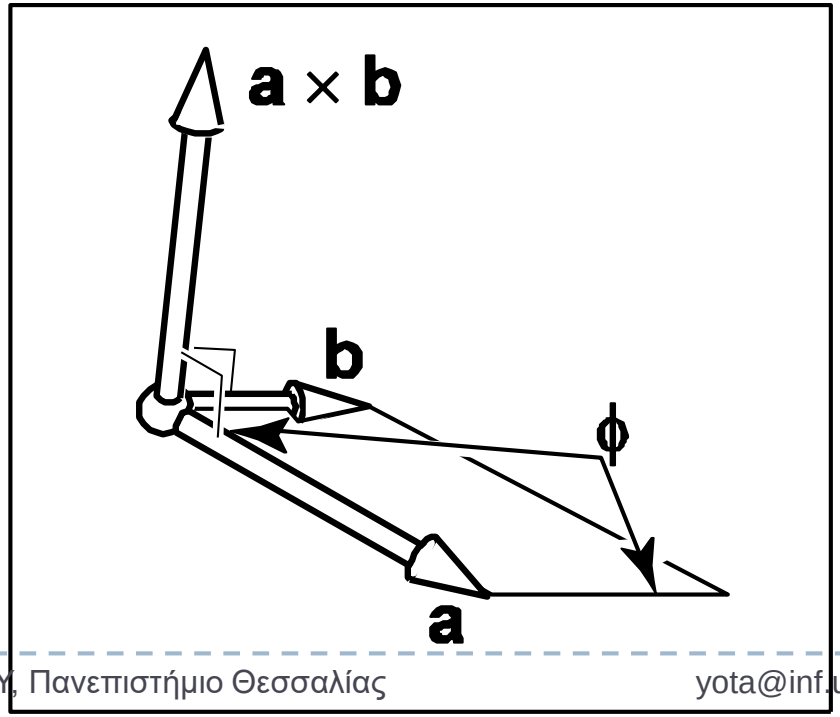
- multiply: vector * vector = vector
- cross product
  - algebraic

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \times \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} u_2 v_3 - u_3 v_2 \\ u_3 v_1 - u_1 v_3 \\ u_1 v_2 - u_2 v_1 \end{bmatrix}$$

  - geometric

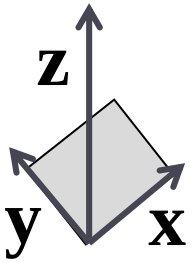
$$\|\mathbf{a} \times \mathbf{b}\| = \|\mathbf{u}\| \|\mathbf{v}\| \sin\theta$$

    - $\|\mathbf{a} \times \mathbf{b}\|$ parallelogram area
    - $\mathbf{a} \times \mathbf{b}$ perpendicular to parallelogram



RHS vs LHS Coordinate Systems

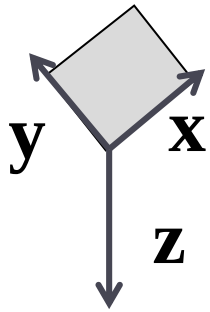
- right-handed coordinate system — convention



right hand rule:
index finger x, second finger y;
right thumb points up

$$\mathbf{z} = \mathbf{x} \times \mathbf{y}$$

- left-handed coordinate system



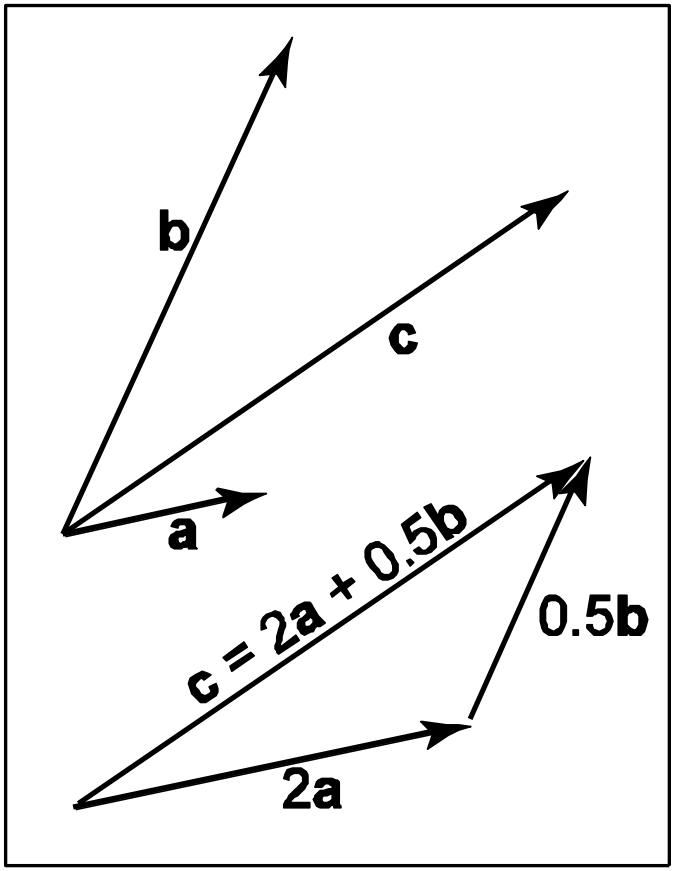
left hand rule:
index finger x, second finger y;
left thumb points down

$$\mathbf{z} = \mathbf{x} \times \mathbf{y}$$

Basis Vectors

- take any two vectors that are linearly independent (nonzero and nonparallel)
  - can use linear combination of these to define any other vector:

$$\mathbf{c} = w_1\mathbf{a} + w_2\mathbf{b}$$



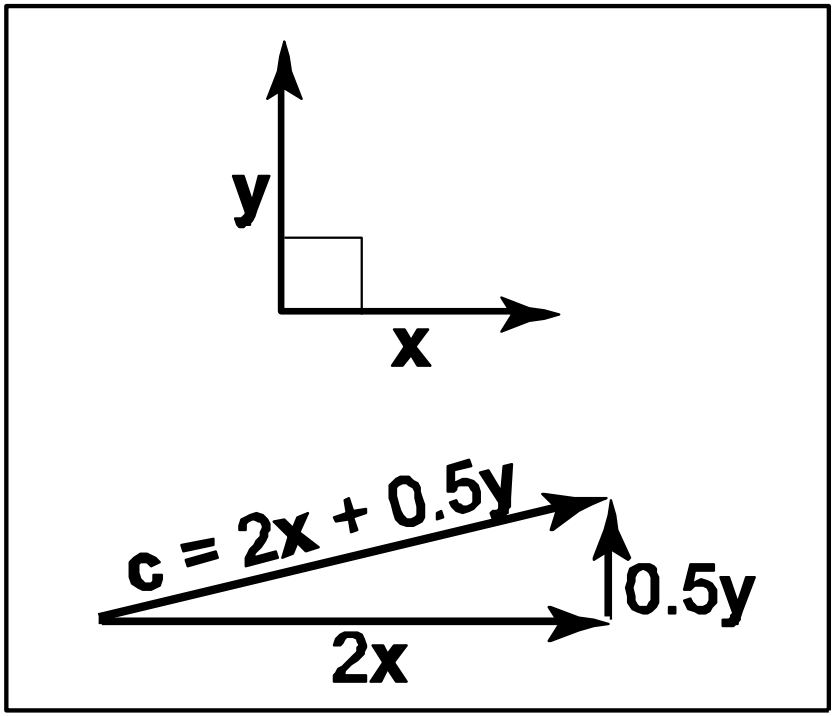
Orthonormal Basis Vectors

- if basis vectors are orthonormal (orthogonal (mutually perpendicular) and unit length)
  - we have Cartesian coordinate system
  - familiar Pythagorean definition of distance

orthonormal algebraic properties

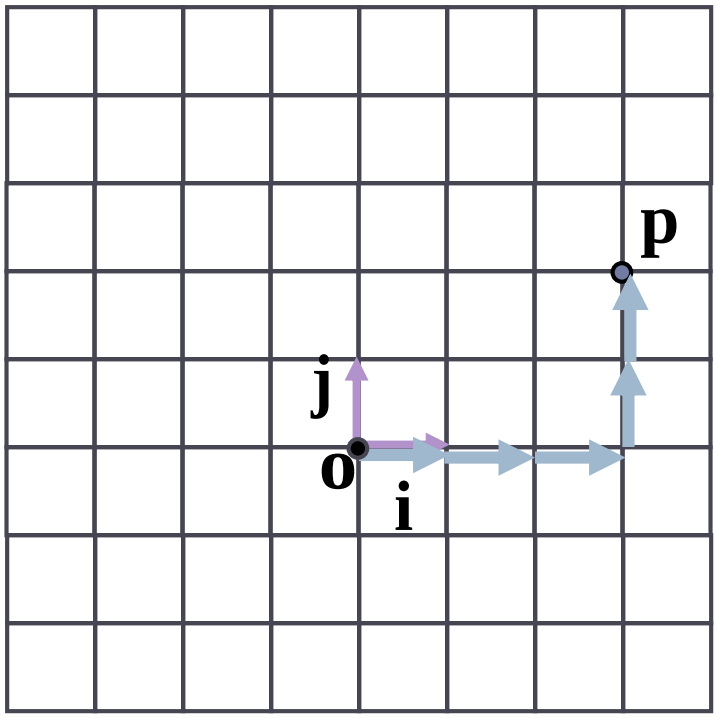
$$\|\mathbf{x}\| = \|\mathbf{y}\| = 1,$$

$$\mathbf{x} \bullet \mathbf{y} = 0$$



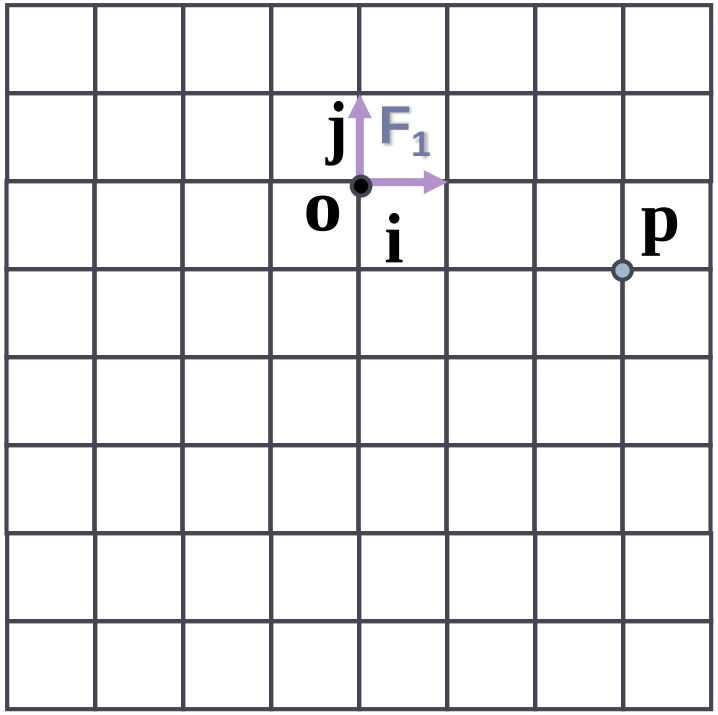
Basis Vectors and Origins

- coordinate system: just basis vectors
  - can only specify offset: vectors
- coordinate frame: basis vectors and origin
  - can specify location as well as offset: points



$$\mathbf{p} = \mathbf{o} + x\mathbf{i} + y\mathbf{j}$$

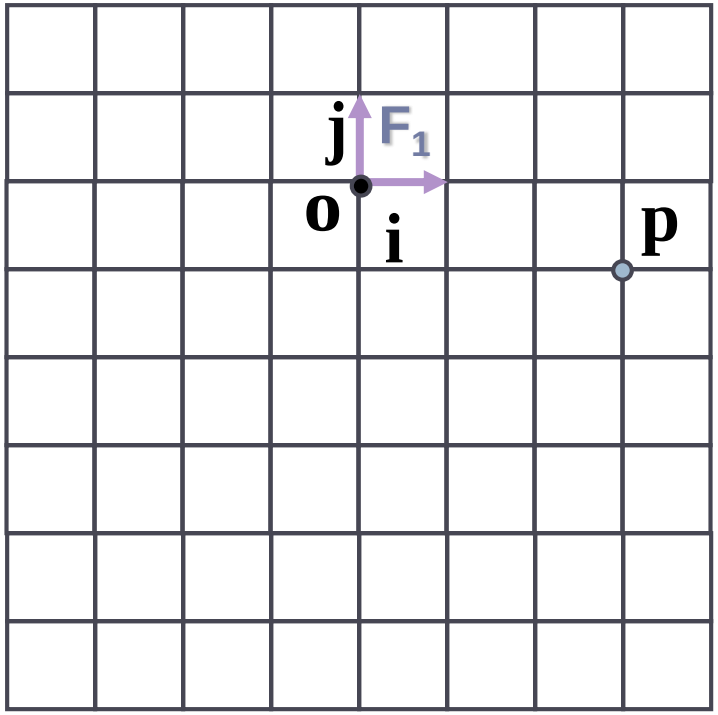
Working with Frames



$$\mathbf{p} = \mathbf{o} + x\mathbf{i} + y\mathbf{j}$$

F_1

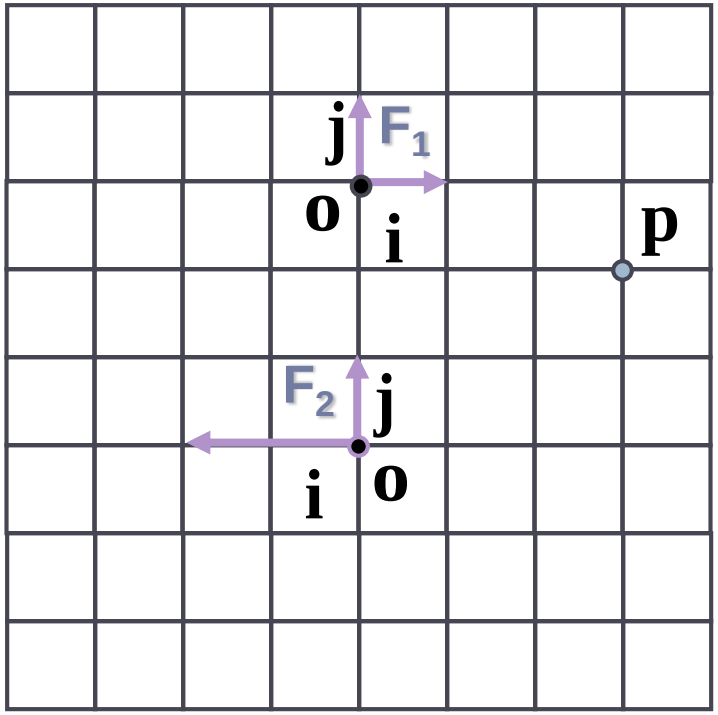
Working with Frames



$$\mathbf{p} = \mathbf{o} + x\mathbf{i} + y\mathbf{j}$$

F_1 p = (3,-1)

Working with Frames

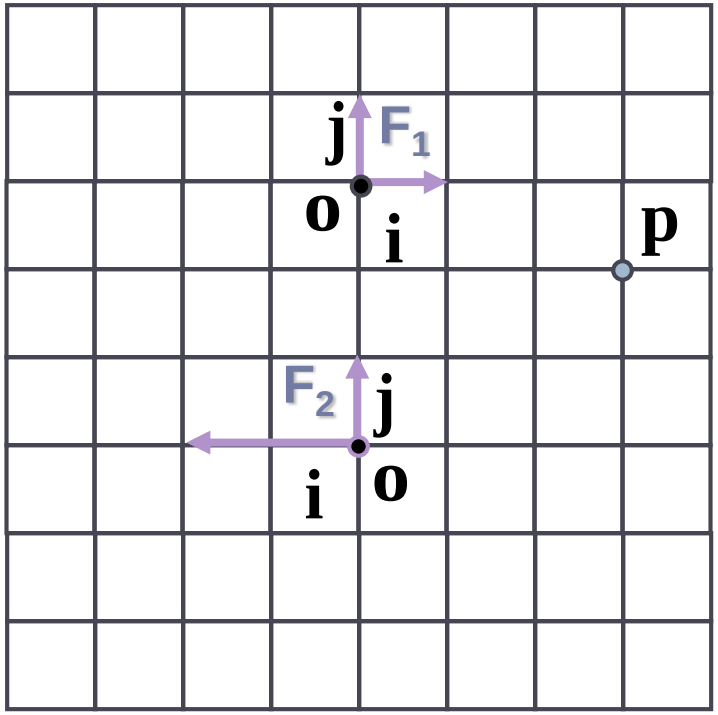


$$\mathbf{p} = \mathbf{o} + x\mathbf{i} + y\mathbf{j}$$

F_1 p = (3,-1)

F_2

Working with Frames

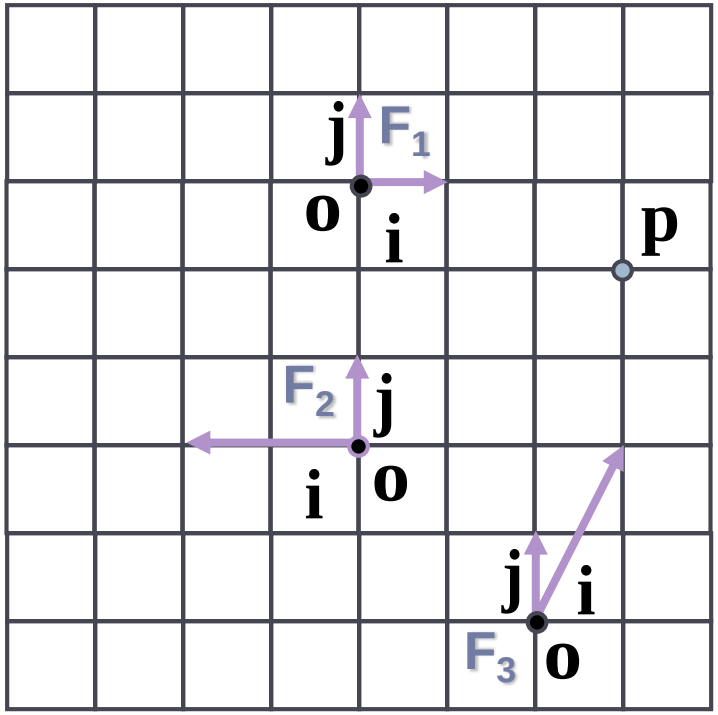


$$\mathbf{p} = \mathbf{o} + x\mathbf{i} + y\mathbf{j}$$

F_1 p = (3,-1)

F_2 p = (-1.5,2)

Working with Frames



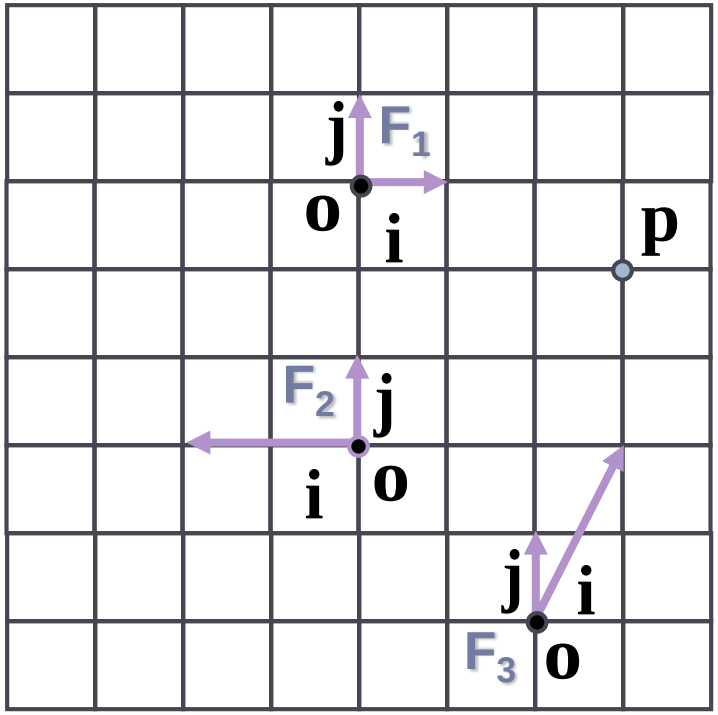
$$\mathbf{p} = \mathbf{o} + x\mathbf{i} + y\mathbf{j}$$

F_1 p = (3,-1)

F_2 p = (-1.5,2)

F_3

Working with Frames



$$\mathbf{p} = \mathbf{o} + x\mathbf{i} + y\mathbf{j}$$

F_1 p = (3,-1)

F_2 p = (-1.5,2)

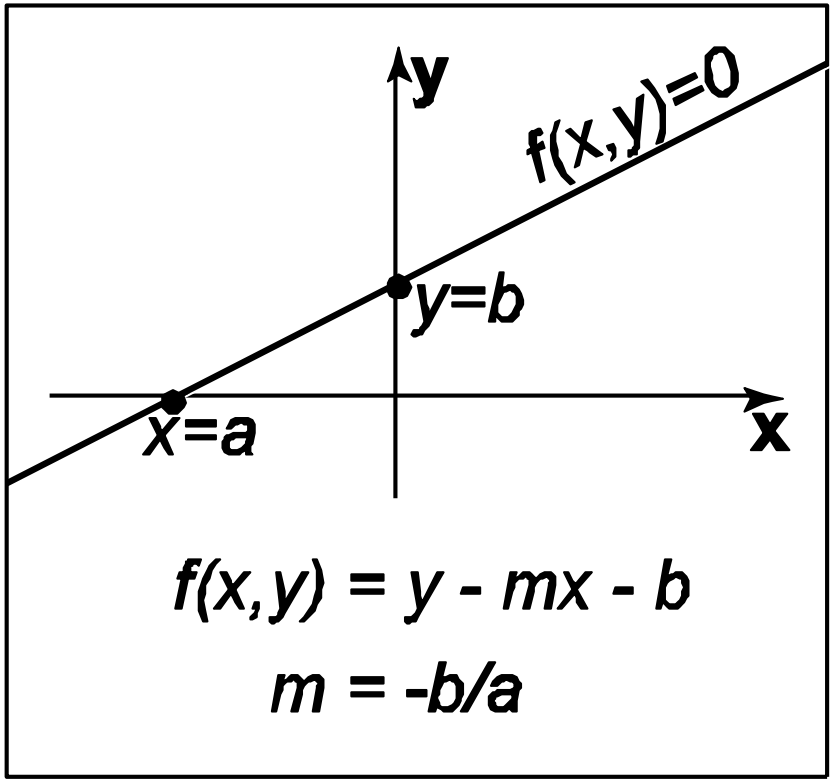
F_3 p = (1,2)

Named Coordinate Frames

- origin and basis vectors $\mathbf{p} = \mathbf{o} + a\mathbf{x} + b\mathbf{y} + c\mathbf{z}$
- pick canonical frame of reference
  - then don't have to store origin, basis vectors
  - just $\mathbf{p} = (a, b, c)$
  - convention: Cartesian orthonormal one on previous slide
- handy to specify others as needed
  - airplane nose, looking over your shoulder, ...
  - really common ones given names in CG
    - object, world, camera, screen, ...

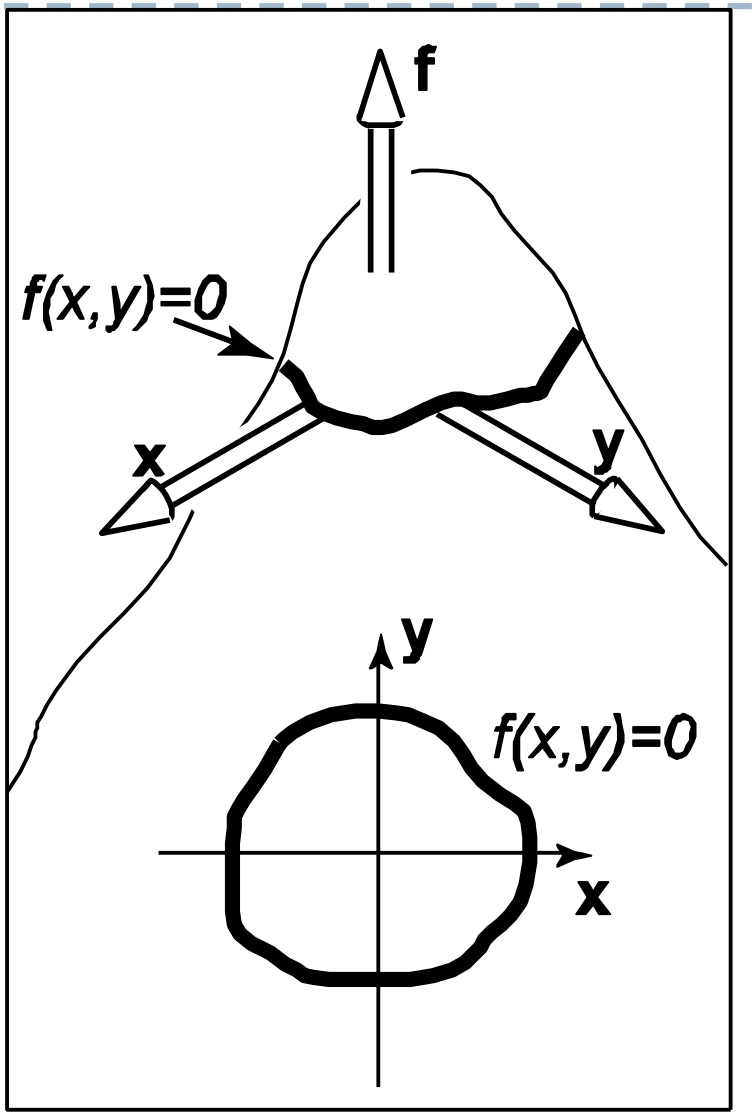
Lines

- slope-intercept form
  - $y = mx + b$
- implicit form
  - $y - mx - b = 0$
  - $Ax + By + C = 0$
  - $f(x,y) = 0$
- Vector form
  - $(P-P_0)\ \mathbf{n} = 0$ where P, P_0 are points on the line and $\mathbf{n}$ is the normal on the line



Implicit Functions

- find where function is 0
  - plug in (x,y), check if
    - 0: on line
    - < 0: inside
    - > 0: outside
- analogy: terrain
  - sea level: f=0
  - altitude: function value
  - topo map: equal-value contours (level sets)



Implicit Circles

- $f(x,y) = (x - x_c)^2 + (y - y_c)^2 - r^2$
  - circle is points (x,y) where f(x,y) = 0
- $p = (x,y), c = (x_c, y_c) : (\mathbf{p} - \mathbf{c}) \bullet (\mathbf{p} - \mathbf{c}) - r^2 = 0$
  - points **p** on circle have property that vector from **c** to **p** dotted with itself has value r^2
- $\|\mathbf{p} - \mathbf{c}\|^2 - r^2 = 0$
  - points points **p** on the circle have property that squared distance from **c** to **p** is r^2
- $\|\mathbf{p} - \mathbf{c}\| - r = 0$
  - points **p** on circle are those a distance *r* from center point **c**

Parametric Curves

- parameter: index that changes continuously
  - (x,y): point on curve
  - t: parameter

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} g(t) \\ h(t) \end{bmatrix}$$

- vector form
  - $\mathbf{p} = f(t)$

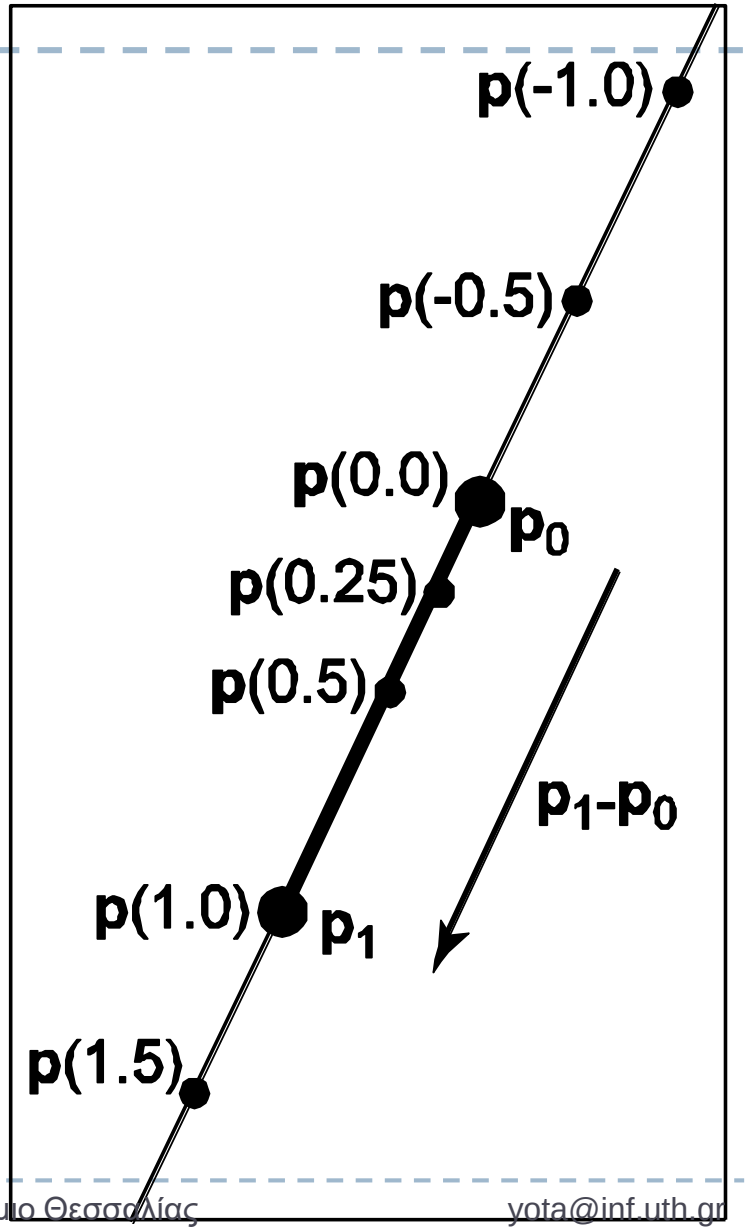
2D Parametric Lines

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_0 + t(x_1 - x_0) \\ y_0 + t(y_1 - y_0) \end{bmatrix}$$

$$\mathbf{p}(t) = \mathbf{p}_0 + t(\mathbf{p}_1 - \mathbf{p}_0)$$

$$\mathbf{p}(t) = \mathbf{o} + t(\mathbf{d})$$

- start at point $\mathbf{p}_0$, go towards $\mathbf{p}_1$, according to parameter t
  - $\mathbf{p}(0) = \mathbf{p}_0$, $\mathbf{p}(1) = \mathbf{p}_1$



Linear Interpolation

- parametric line is example of general concept
  - $\mathbf{p}(t) = \mathbf{p}_0 + t(\mathbf{p}_1 - \mathbf{p}_0)$
  - interpolation
    - **p** goes through **a** at $t = 0$
    - **p** goes through **b** at $t = 1$
  - linear
    - weights t, $(1-t)$ are linear polynomials in t

Ασκήσεις

- Ποια είναι η παραμετρική εξίσωση της ευθείας που διέρχεται από το μέσον του P_1P_0 ευθυγράμμου τμήματος και είναι κάθετος στο P_1P_0 ;
- Για ποια τιμή του t παίρνουμε το μέσον του P_1P_0
- Ποια είναι η παραμετρική παράσταση του κύκλου και έλλειψης;
- Πως ορίζονται τα εφαπτόμενα και κάθετα διανύσματα καμπυλών;

Matrix-Matrix Addition

- add: matrix + matrix = matrix

$$\begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} + \begin{bmatrix} n_{11} & n_{12} \\ n_{21} & n_{22} \end{bmatrix} = \begin{bmatrix} n_{11}+m_{11} & n_{12}+m_{12} \\ n_{21}+m_{21} & n_{22}+m_{22} \end{bmatrix}$$

- example

$$\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} + \begin{bmatrix} -2 & 5 \\ 7 & 1 \end{bmatrix} = \begin{bmatrix} 1+(-2) & 3+5 \\ 2+7 & 4+1 \end{bmatrix} = \begin{bmatrix} -1 & 8 \\ 9 & 5 \end{bmatrix}$$

Scalar-Matrix Multiplication

- multiply: scalar * matrix = matrix

$$a\begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} = \begin{bmatrix} a*m_{11} & a*m_{12} \\ a*m_{21} & a*m_{22} \end{bmatrix}$$

- example

$$3\begin{bmatrix} 2 & 4 \\ 1 & 5 \end{bmatrix} = \begin{bmatrix} 3*2 & 3*4 \\ 3*1 & 3*5 \end{bmatrix} = \begin{bmatrix} 6 & 12 \\ 3 & 15 \end{bmatrix}$$

Matrix-Matrix Multiplication

- row by column

$$\begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \begin{bmatrix} n_{11} & n_{12} \\ n_{21} & n_{22} \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix}$$

$$p_{11} = m_{11}n_{11} + m_{12}n_{21}$$

Matrix-Matrix Multiplication

- row by column

$$\begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \begin{bmatrix} n_{11} & n_{12} \\ n_{21} & n_{22} \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix}$$

$$p_{11} = m_{11}n_{11} + m_{12}n_{21}$$

$$p_{21} = m_{21}n_{11} + m_{22}n_{21}$$

Matrix-Matrix Multiplication

- row by column

$$\begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \begin{bmatrix} n_{11} & n_{12} \\ n_{21} & n_{22} \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix}$$

$$p_{11} = m_{11}n_{11} + m_{12}n_{21}$$

$$p_{21} = m_{21}n_{11} + m_{22}n_{21}$$

$$p_{12} = m_{11}n_{12} + m_{12}n_{22}$$

Matrix-Matrix Multiplication

- row by column

$$\begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \begin{bmatrix} n_{11} & n_{12} \\ n_{21} & n_{22} \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix}$$

$$p_{11} = m_{11}n_{11} + m_{12}n_{21}$$
$$p_{21} = m_{21}n_{11} + m_{22}n_{21}$$
$$p_{12} = m_{11}n_{12} + m_{12}n_{22}$$
$$p_{22} = m_{21}n_{12} + m_{22}n_{22}$$

Matrix-Matrix Multiplication

- row by column

$$\begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \begin{bmatrix} n_{11} & n_{12} \\ n_{21} & n_{22} \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix}$$

$$p_{11} = m_{11}n_{11} + m_{12}n_{21}$$
$$p_{21} = m_{21}n_{11} + m_{22}n_{21}$$
$$p_{12} = m_{11}n_{12} + m_{12}n_{22}$$
$$p_{22} = m_{21}n_{12} + m_{22}n_{22}$$

- noncommutative: **AB** != **BA**

Matrix Multiplication

- can only multiply if
  number of left cols = number of right rows
  - legal
  - undefined

$$\begin{bmatrix} a & b & c \\ e & f & g \end{bmatrix} \begin{bmatrix} h & i \\ j & k \\ l & m \end{bmatrix}$$

$$\begin{bmatrix} a & b & c \\ e & f & g \\ o & p & q \end{bmatrix} \begin{bmatrix} h & i \\ j & k \end{bmatrix}$$

Matrix-Vector Multiplication

- points as column vectors: postmultiply

$$\begin{bmatrix} x' \\ y' \\ z' \\ h' \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ h \end{bmatrix} \qquad \mathbf{p}' = \mathbf{M}\mathbf{p}$$

- points as row vectors: premultiply

$$\begin{bmatrix} x' & y' & z' & h' \end{bmatrix} = \begin{bmatrix} x & y & z & h \end{bmatrix} \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix}^T \qquad \mathbf{p}'^T = \mathbf{p}^T\mathbf{M}^T$$

Πίνακες

- ανάστροφος $\begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix}^T = \begin{bmatrix} m_{11} & m_{21} & m_{31} & m_{41} \\ m_{12} & m_{22} & m_{32} & m_{42} \\ m_{13} & m_{23} & m_{33} & m_{43} \\ m_{14} & m_{24} & m_{34} & m_{44} \end{bmatrix}$
- ταυτοτικός $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
- αντίστροφος $\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$
  - δεν είναι όλοι οι πίνακες αντιστρέψιμοι

Πίνακες και Γραμμικά Συστήματα

- Γραμμικό σύστημα n εξισώσεων με n αγνώστους

$$3x + 7y + 2z = 4$$
$$2x - 4y - 3z = -1$$
$$5x + 2y + z = 1$$

- σε μορφή πινάκων **Ax=b**

$$\begin{bmatrix} 3 & 7 & 2 \\ 2 & -4 & -3 \\ 5 & 2 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 4 \\ -1 \\ -1 \end{bmatrix}$$

Linear Vector Spaces

- Mathematical system for manipulating vectors
- Operations
  - Scalar-vector multiplication $u=\alpha v$
  - Vector-vector addition: $w=u+v$
- Expressions such as

  $v=u+2w-3r$

Make sense in a vector space

Affine Spaces

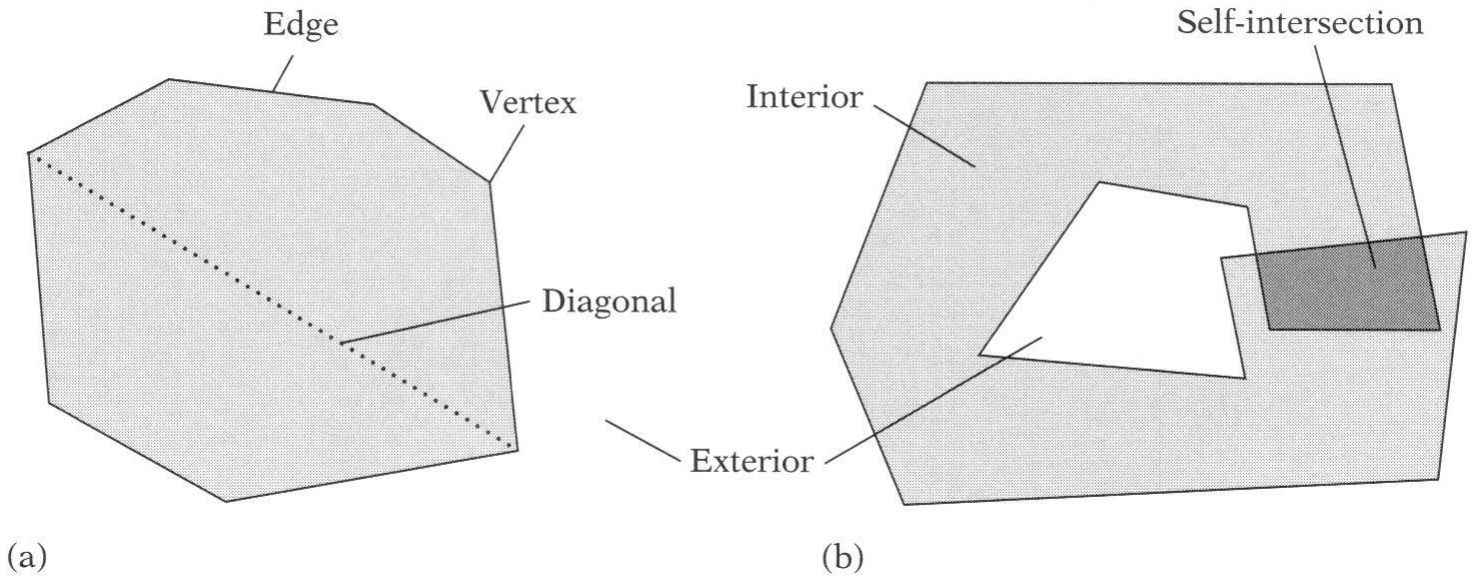
- Point + a vector space
- Operations
  - Vector-vector addition
  - Scalar-vector multiplication
  - Point-vector addition
  - Scalar-scalar operations
- For any point define
  - $1 \bullet P = P$
  - $0 \bullet P = \mathbf{0}$ (zero vector)

Γεωμετρία

Πολύγωνα - Polygons

- A polygon is a closed figure with n sides, defined by an ordered set of three or more points in the plane.
  - Each point is connected to the next with a line segment.
  - For a set of n points, the resulting polygon is also called an n-sided polygon or just n-gon.
- The polygon sides or edges are the line segments that make up the polygon boundary.
- The points themselves are called the polygon vertices.
  - Two vertices are adjacent if they are joined by an edge.

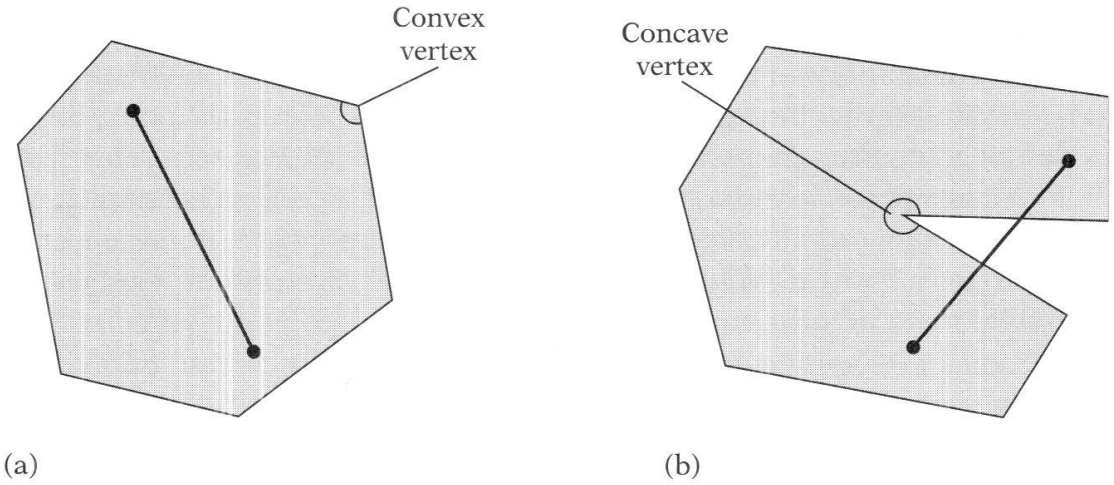
Polygons



- A polygon is simple if no two nonconsecutive edges have a point in common.
- A simple polygon partitions the plane into two disjoint parts
  - The interior
  - The exterior

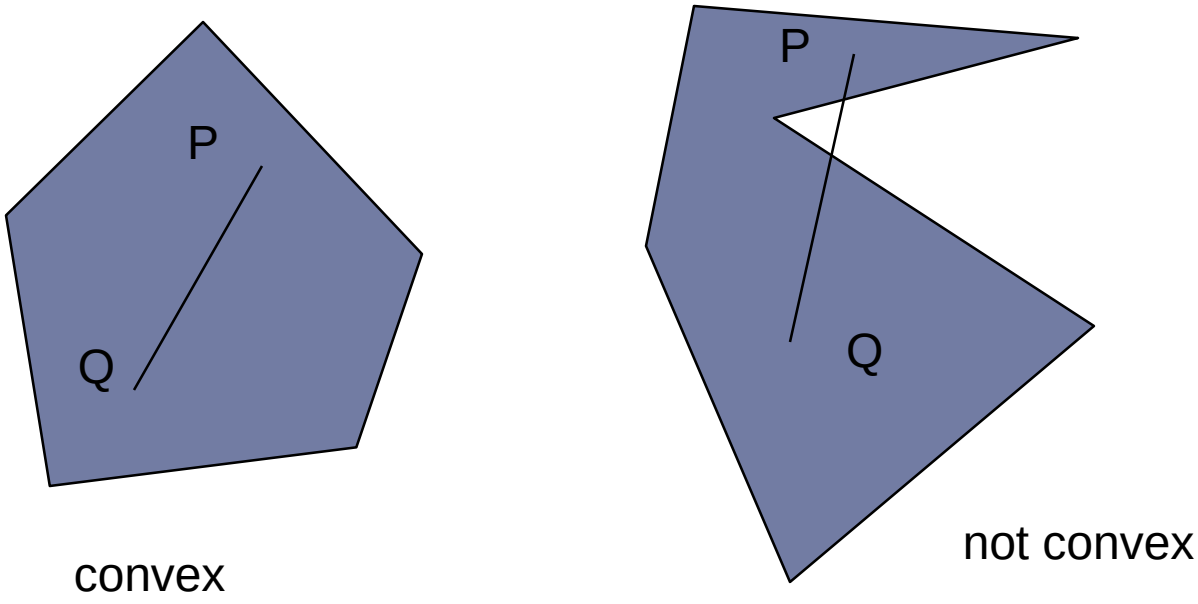
Polygons

- A polygon diagonal is a line segment that joins two polygon vertices and lies fully inside the polygon.
- A vertex is a convex vertex if the interior angle is less than or equal to 180 degrees.
  - If the angle is larger than 180 degrees, it is instead called a concave (or reflex) vertex.



Convexity

- An object is *convex* iff for any two points in the object all points on the line segment between these points are also in the object



Polygons

- A polygon P is a convex polygon if all line segments between any two points of P lie fully inside P.
- A polygon that is not convex is called a concave polygon.
- A polygon with one or more concave vertices is necessarily concave.
- But a polygon with only convex vertices is not always convex.
- The triangle is the only n-sided polygon always guaranteed to be convex.

Polygons

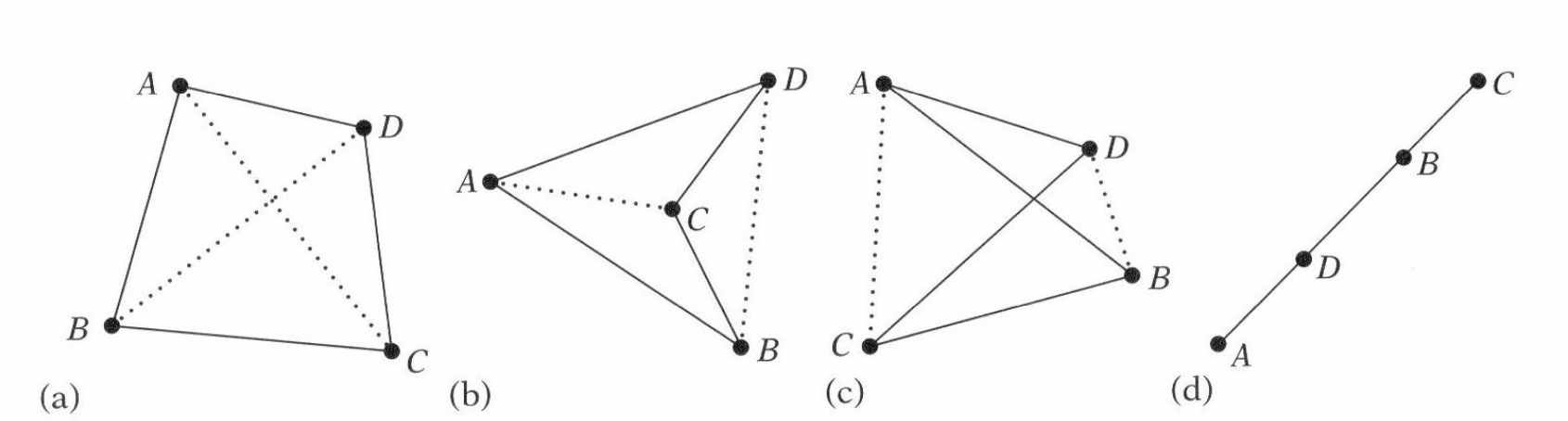
- In addition to the explicit vertex representation, convex polygons can also be described as the intersection of a finite number of halfspaces.
  - This representation is convenient for point containment tests.
- Two or more polygons can be joined at their edges to form a polygon mesh.
  - The degree of a vertex corresponds to the number of edges connected to the vertex.
  - A mesh will be considered closed if all polygons have been joined such that each edge is part of exactly two polygons.

Testing Polygon Convexity

- Most intersection tests and other operations performed on polygons in a collision detection system are faster when applied to convex rather than concave polygons.
  - Simplifying assumptions can be made in the former case.
  - Triangles are nice in this respect. → Always convex.
  - It may be more efficient to perform an intersection against a single convex n-gon rather than against multiple triangles covering the same area.
- Frequently, quadrilaterals are the only primitives supported in addition to triangles.

Testing Polygon Convexity

- Assume all vertices of the quad ABCD lie in the same plane.
- The quad is convex if and only if its two diagonals lie fully in the interior of the quad.



Testing Polygon-Quad Convexity

- This test is equivalent to testing if the two line segments AC and BD corresponding to the diagonals, intersect each other.
  - If they do, the quad is convex.
  - If they do not, the quad is concave or self-intersecting.
  - If the segments are parallel and overlapping, the quad is degenerate (into a line).
- To avoid considering a quad with three collinear vertices convex, the segments should only be considered intersecting if they overlap on their interior.

Testing Polygon Convexity

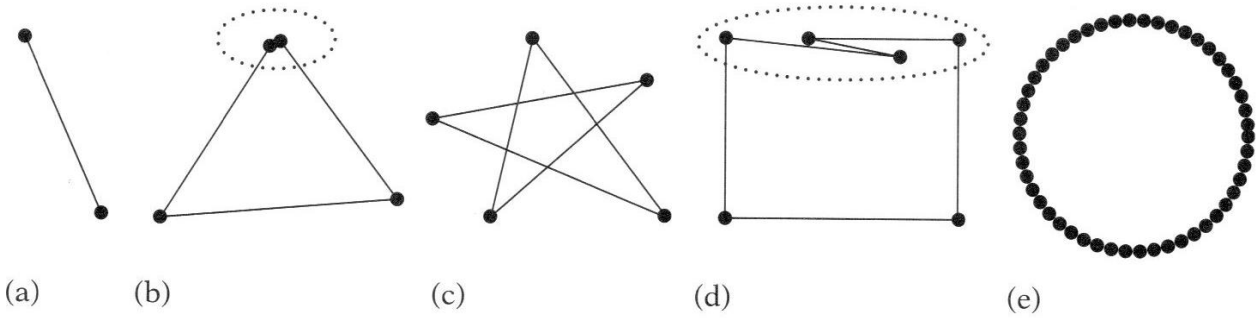
- It can be shown that the intersection of the segments is equivalent to the points A and C lying on opposite sides of the line through BD, as well as to the points B and D lying on opposite sides of the line through AC.
- This test is equivalent to the triangle BDA having opposite winding (περιέλιξη - προσανατολισμός) to BDC, as well as ACD having opposite winding to ACB.
  - The opposite winding can be detected by computing the normals of the triangles and examining the sign of the dot product between the normals of the triangles to be compared.

Testing Polygon Convexity

- If the dot product is negative, the normals point in opposing directions, and the triangles therefore wind in opposite order.
  - $(BD \times BA) \cdot (BD \times BC) < 0$
  - $(AC \times AD) \cdot (AC \times AB) < 0$
- For general n-gons, not just quads, a straightforward solution is to, for each polygon edge, test to see if all other vertices lie (strictly) on the same side of that edge.
  - If the test is true for all edges, the polygon is convex.
  - Otherwise it is concave.

Testing Polygon Convexity

- A separate check for coincident vertices is required to make the test robust.
- Although easy to implement, this test is expensive for large polygons, with an $O(n^2)$ complexity in the number of vertices.
- It is easy to come up with tests that are faster.
  - Many of them correctly classify only a subset of convex polygons and incorrectly classify some nonconvex polygons.

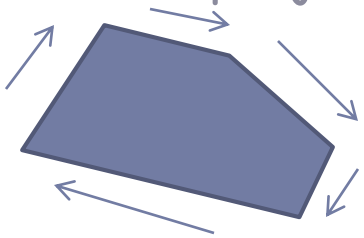


Testing Polygon Convexity

- For example..
  - A strictly convex polygon has interior angles that are all less than 180 degrees.
  - However, although this test is a necessary criterion for convexity it is not a sufficient one.
    - Testing the interior angles alone would thus incorrectly conclude that a pentagram (πεντάλφα) is a convex polygon.
    - This test only works if the polygon is known, a priori, not to be self-intersecting.

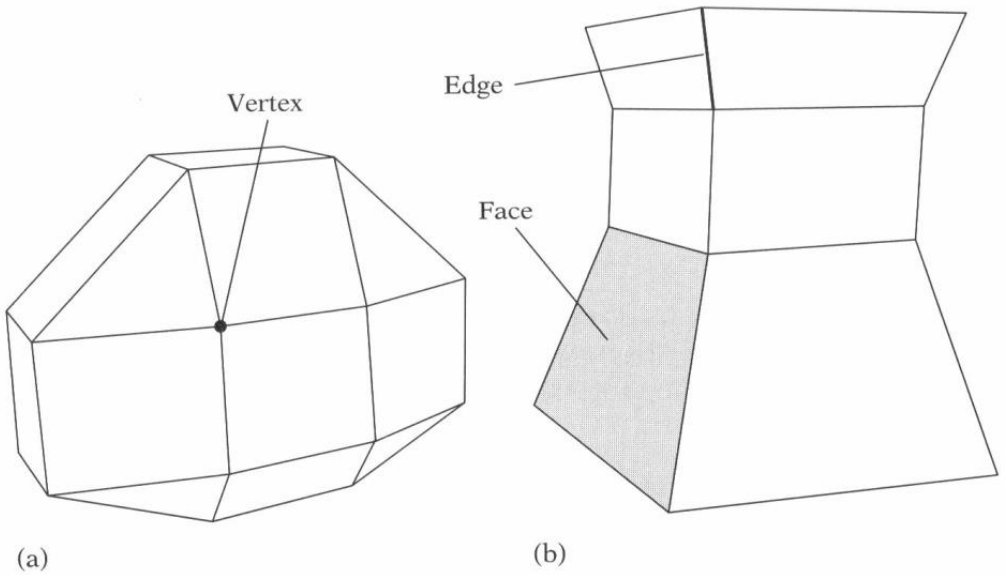
Testing Polygon Convexity

- Another basis for a convexity test is that there are only two changes in direction along any given axis when moving from vertex to vertex of a convex polygon, accounting for wraparound from the first to the last vertex.
- To detect the zigzag case, a test for change of direction would have to be performed for two different directions, such as both the x axis and the y axis.
  - This fails when all vertices of an originally convex polygon have been projected onto a single line.



Polyhedra

- A polyhedron is the 3D counterpart of a polygon.
  - It is a bounded and connected region of space in the shape of a multifaceted (πολύπλευρος) solid.
  - The polyhedron boundary consists of a number of polygonal faces (όψη) connected so that each polygon edge is part of exactly two faces.



Polyhedra

- The polyhedron boundary divides space into two disjoint regions: the interior and the exterior.
- A polyhedron is convex if the point set determined by its interior and boundary is convex.
  - A bounded convex polyhedron is also referred to as a polytope (πολύτοπο).
  - Polytopes can also be described as the intersection of a finite number of halfspaces.
- A d-simplex is the convex hull of d+1 affinely (συσχετισμένα) independent points in d-dimensional space.
  - A simplex is a d-simplex for some given d.

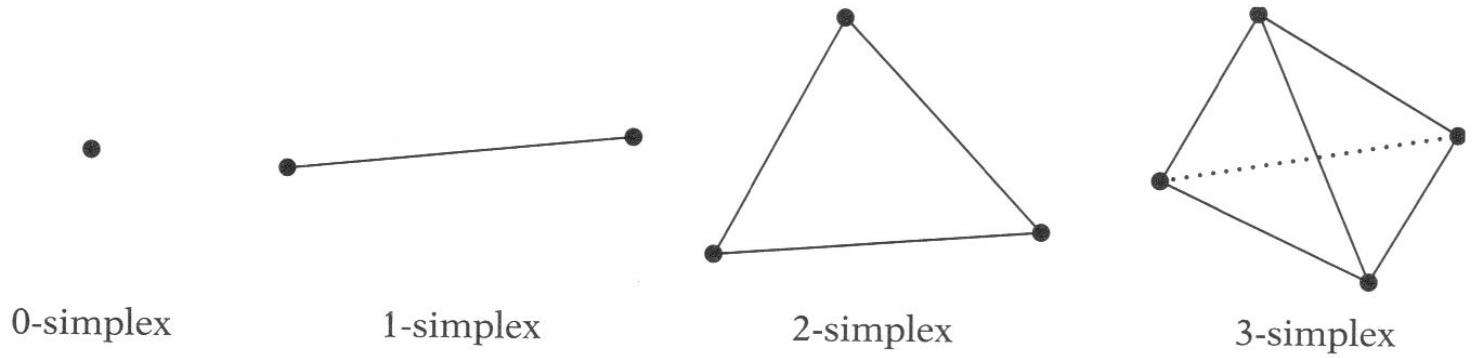
Polyhedra

0-simplex: a point,

1-simplex: a line segment,

2-simplex: a triangle,

3-simplex: a tetrahedron.



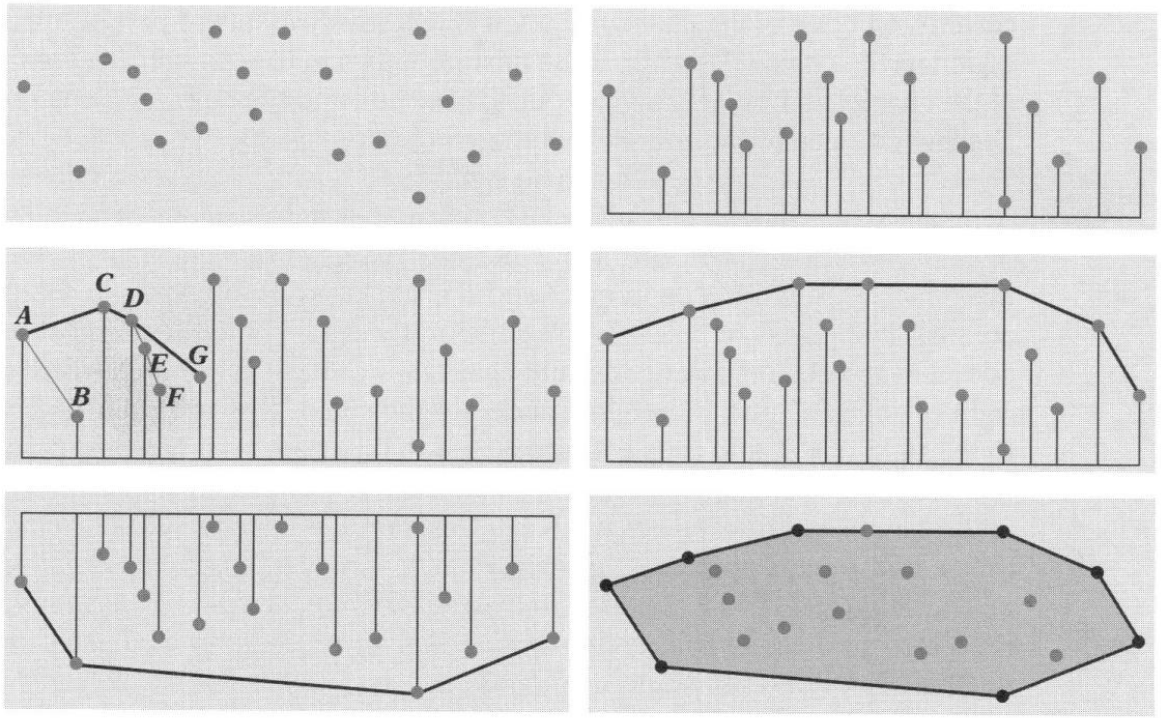
Testing Polyhedral Convexity

- A polyhedron P is convex if and only if for all faces of P all vertices of P lie (strictly) on the same side of that face.
  - A separate test for coincident vertices and collinear edges of the polyhedron faces is required to make the test robust, usually with some tolerance added for determining coincidency and collinearity.
  - The complexity of this test is $O(n^2)$
- A faster O(n) approach is to compute for each face F of P the centroid C of F, and for all neighboring faces G of F test if C lies behind the supporting plane of G.
  - If some C fails to lie behind the supporting plane of one or more neighboring faces, P is concave.

Computing Convex Hulls

Andrew's algorithm

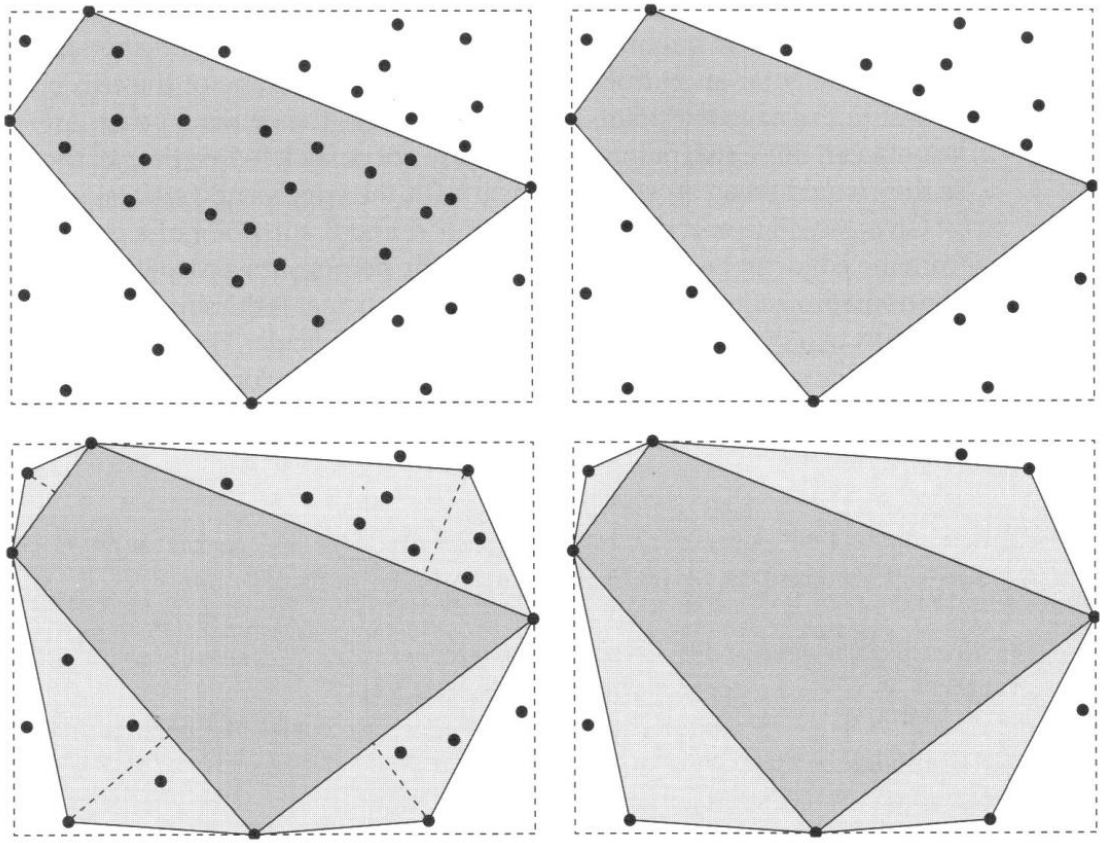
One of the most robust and easy to implement 2D convex hull algorithms



Computing Convex Hulls

The Quickhull algorithm

- A method that works in both 2D and 3D



Computing Convex Hulls

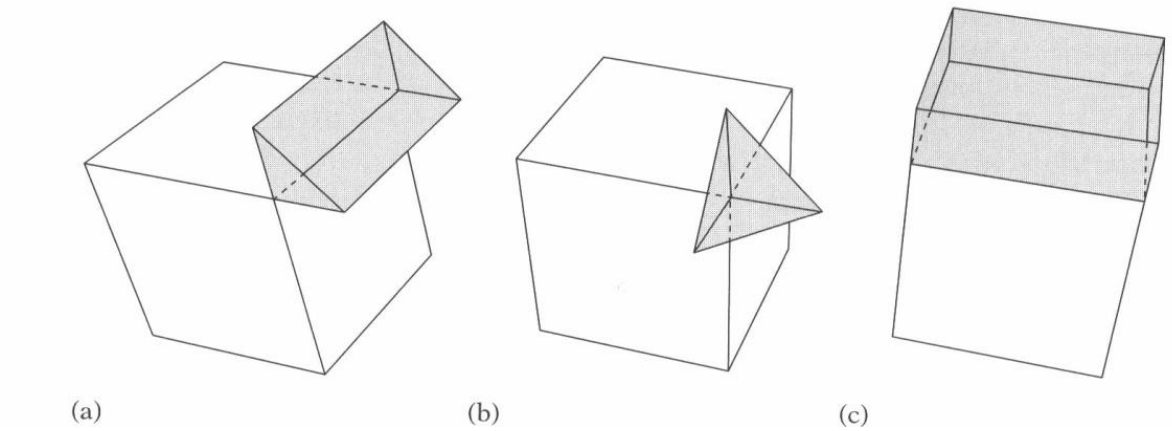
The Quickhull algorithm

- The initial hull approximation may not always be a quadrilateral.
  - To avoid problems, an implementation must be written to handle an initial hull approximation of a variable number of edges.
- There might not be a single unique point on the edge of the initial bounding box, or a single unique point farthest away from an edge.
  - One of them that lie closest to the edge endpoints must be chosen as the extreme point.

Voronoi Region

- Given a set S of points in the plane, the Voronoi region of a point P in S is defined as the set of points in the plane closer to P than to any other points in S.
  - Quite useful for collision detection applications

- Given a polyhedron P, let a feature of P be one of its vertices, edges, or faces.
- The Voronoi region of a feature F of P is then the set of points in space closer to F than to any other feature of P.



Voronoi Region

- The boundary planes of a Voronoi region are referred to as Voronoi planes.
- Given a convex polyhedron P, all points in space exterior to P can be classified as lying in a Voronoi feature region of a vertex, edge, or face of P, with the boundary between two neighboring Voronoi feature regions considered to belong to only one of the regions.
  - The Voronoi regions create a partitioning of space exterior to a polyhedron, they can be used to determine the closest point on a convex polyhedral object to some point Q in space.
  - This determination is done by walking from region to region until Q is found to be inside the region.

Voronoi Region

- The closest point on the object to Q is then the projection of Q onto the feature with which the given region is associated.

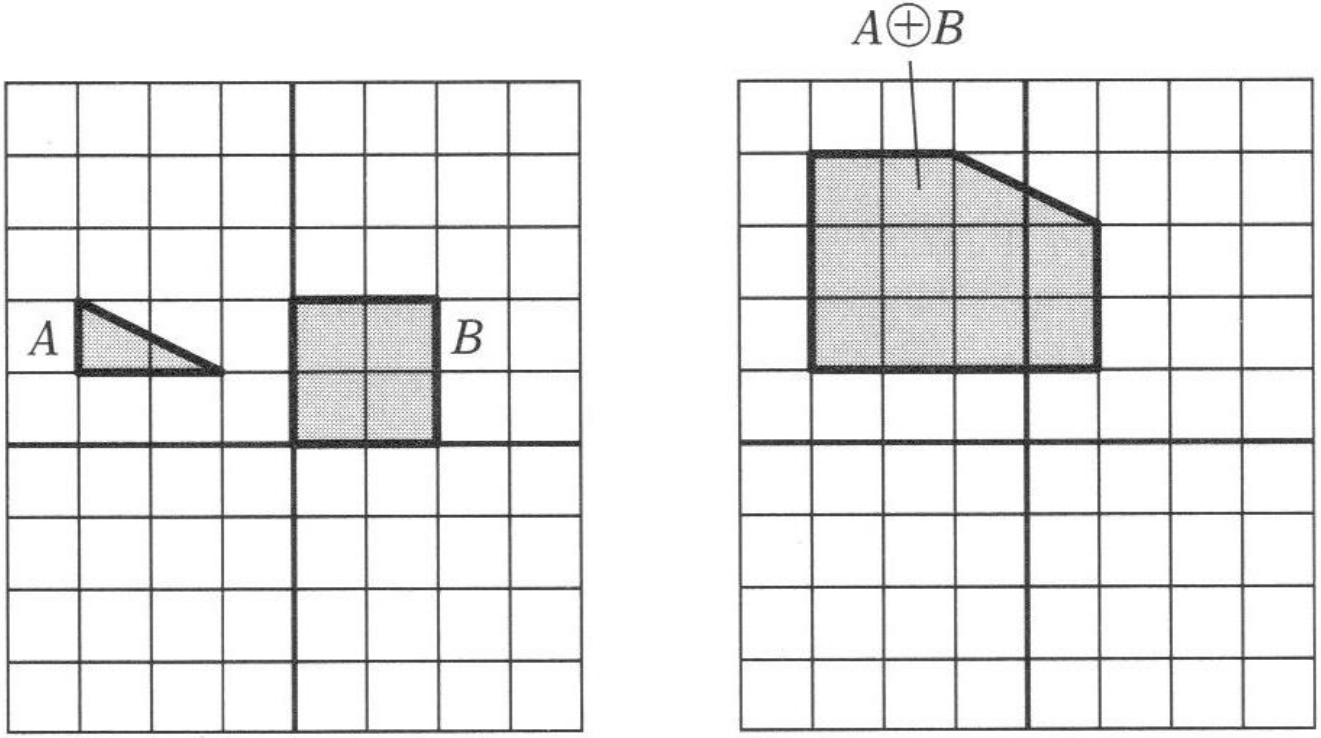
Minkowski Sum and Difference

- Let A and B be two point sets, and let a and b be the position vectors corresponding to pairs of points in A and B.
  - Minkowski sum is defined by

$$A \oplus B = \{\mathbf{a} + \mathbf{b} : \mathbf{a} \in A, \mathbf{b} \in B\}$$

  - Visually, the Minkowski sum can be seen as the region swept by A translated to every point in B.

Minkowski Sum and Difference



Minkowski Sum and Difference

- Let A and B be two point sets, and let a and b be the position vectors corresponding to pairs of points in A and B.
  - Minkowski difference is defined by

$$A - B = \{\mathbf{a} - \mathbf{b} : \mathbf{a} \in A, \mathbf{b} \in B\}$$

  - Geometrically, the Minkowski difference is obtained by adding A to the reflection of B about the origin.

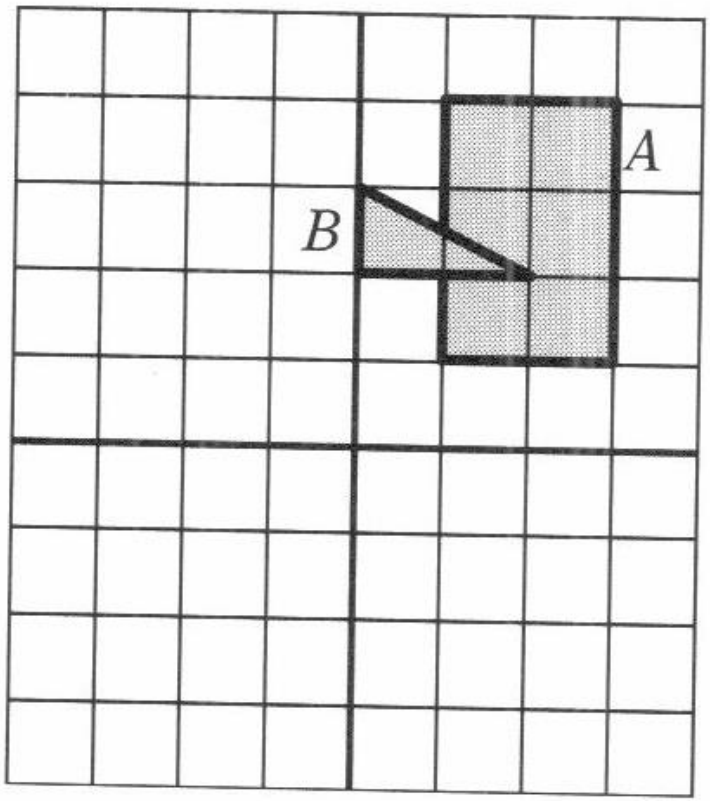
Minkowski Sum and Difference

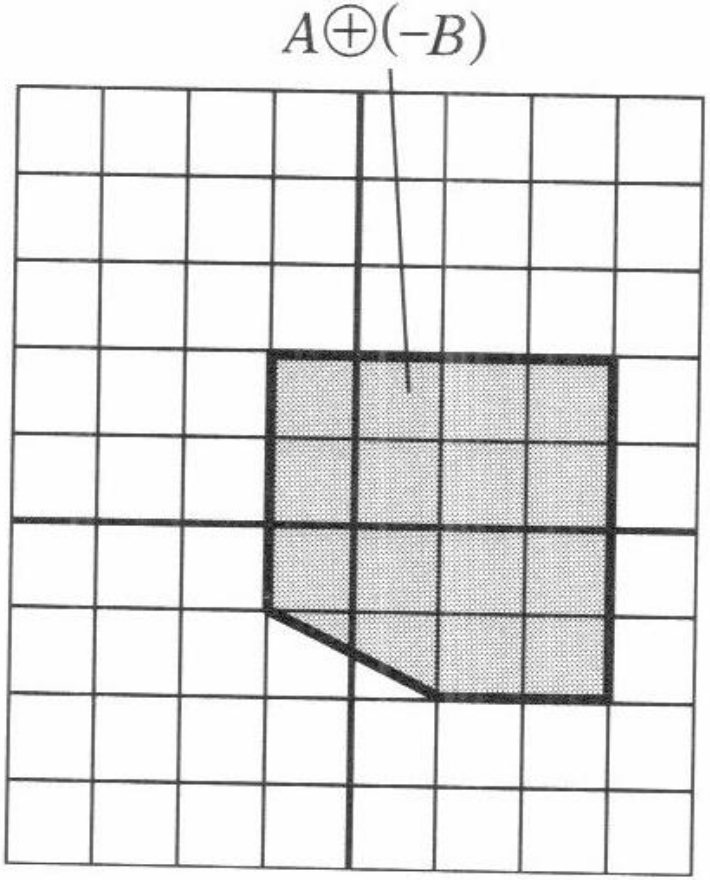
- For two convex polygons, P and Q, the Minkowski sum R has the properties that R is a convex polygon and the vertices of R are sums of the vertices of P and Q.
  - The Minkowski sum of two convex polyhedra is a convex polyhedra, with corresponding properties.
- Minkowski sums both directly and indirectly apply to collision detection.
  - Obstacles can be "grown" by the object at the same time the object is "shrunk" allowing the collision testing of the moving object to be treated as a moving point against the grown obstacles.

Minkowski Sum and Difference

- The Minkowski difference is important from a collision detection perspective because two point sets A and B collide if and only if their Minkowski difference C contains the origin.
  - It is possible to establish an even stronger result: computing the minimum distance between A and B is equivalent to computing the minimum distance between C and the origin.
- The Minkowski difference of two convex sets is also a convex set and thus its point of minimum norm is unique.

Minkowski Sum and Difference





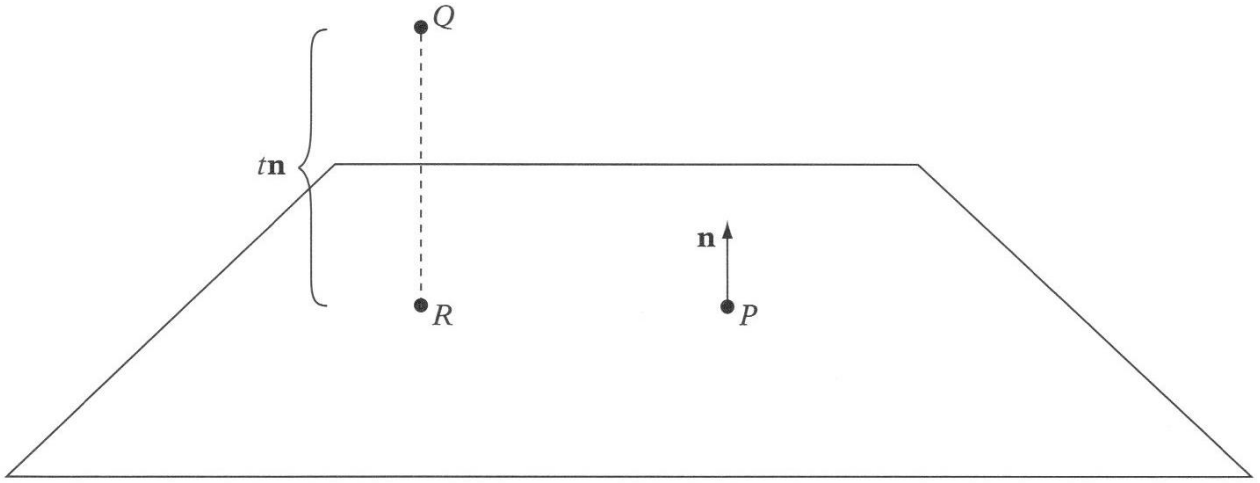
Closest-point Computations

- Closest-point queries are some of the most powerful of collision queries.
- Given the closest points between two objects, the distance between the objects is obtained.
  - If the combined maximum movement of two objects is less than the distance between them, a collision can be ruled out.
  - In a hierarchical representation, closest-point computations allow parts of the hierarchy that will never come close enough to collide to be pruned from further consideration.
- Obtaining the closest points between two objects can be seen as a minimization problem.

Closest Point on Plane to Point

- Given a plane Π, defined by a point P and a normal n, all points X on the plane satisfy the equation $n \cdot (X-P) = 0$.
- Q is an arbitrary point in space.
- The closest point R on the plane to Q is the orthogonal projection of Q onto the plane, obtained by moving Q perpendicularly toward the plane.
  - $R = Q - tn$, for some value of t.

Closest Point on Plane to Point



$$\mathbf{n} \cdot ((Q - t\mathbf{n}) - P) = 0 \Leftrightarrow$$ *(inserting R for X in plane equation)*

$$\mathbf{n} \cdot Q - t(\mathbf{n} \cdot \mathbf{n}) - \mathbf{n} \cdot P = 0 \Leftrightarrow$$ *(expanding dot product)*

$$\mathbf{n} \cdot (Q - P) = t(\mathbf{n} \cdot \mathbf{n}) \Leftrightarrow$$ *(gathering similar terms and moving t expression to RHS)*

$$t = \mathbf{n} \cdot (Q - P)/(\mathbf{n} \cdot \mathbf{n})$$ *(dividing both sides by $\mathbf{n} \cdot \mathbf{n}$)*

Substituting this expression for t in $R = Q - t\mathbf{n}$ gives the projection point R as

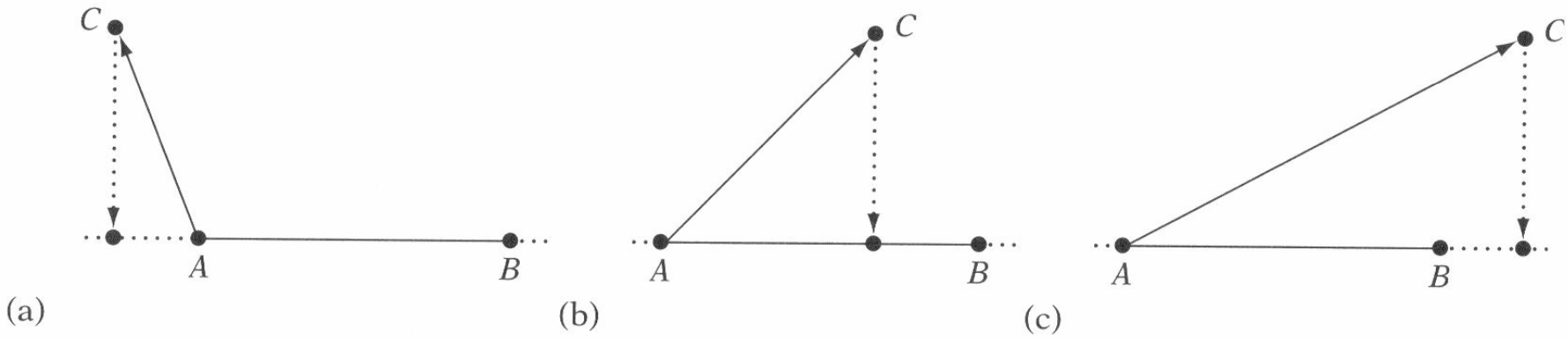
$$R = Q - (\mathbf{n} \cdot (Q - P)/(\mathbf{n} \cdot \mathbf{n}))\mathbf{n}.$$

Closest Point on Plane to Point

- When n is of unit length, t simplifies to $t = n \cdot (Q-P)$, giving R as simply $R = Q-(n \cdot (Q-P))n$.
- It is easy to see that for an arbitrary point Q, $t = n \cdot (Q-P)$ corresponds to the signed distance of Q from the plane in units of the length of n.

Closest Point on Line Segment to Point

- Let AB be a line segment specified by the endpoints A and B and let C be an arbitrary point. The problem is to determine the point D on AB closest to C.



Closest Point on Line Segment to Point

- Projecting C onto the extended line through AB provides the solution.
  - If the projection point P lies within the segment, P itself is the correct answer.
  - If P lies outside the segment, it is instead the segment endpoint closest to C that is the closest point.
- Any point on the segment is given by $P(t) = A + t(B-A)$, $0 \le t \le 1$.
- Using the projective properties of the dot product, the t corresponding to the projection of C onto the line is given by $t = (C-A) \cdot n / ||B-A||$, where $n=(B-A)/||B-A||$ is a unit vector in the direction of AB.
  - t should be clamped to the interval $0 \le t \le 1$.
  - D can then be obtained by substituting t into the parametric equation.

Closest Point on Line Segment to Point

- Distance of Point to Segment
  - The squared distance between a point C and a segment AB can be directly computed without explicitly computing the point D on AB closest to C.
  - There are three cases to consider
    - $AC \cdot AB \leq 0$: distance is $AC \cdot AC$
    - $AC \cdot AB \geq AB \cdot AB$: distance is $BC \cdot BC$
    - $0 < AC \cdot AB < AB \cdot AB$: distance is $CD \cdot CD$

Closest Point on AABB to Point

- Let B be an axis-aligned bounding box and P an arbitrary point in space.
- The point Q on B closest to P is obtained by clamping P to the bounds of B on a componentwise basis.
  - If P is inside B, the clamped point is P itself, which is also the point in B closest to P.
  - If P is in a face Voronoi region of B, the clamping operation will bring P to that face of B.
    - The clamping corresponds to an orthogonal projection of P onto B and must therefore result in the closest point on B.

Closest Point on AABB to Point

- The point Q on B closest to P is obtained by clamping P to the bounds of B on a componentwise basis.
  - When P is in a vertex Voronoi region of B, clamping P gives the vertex as a result, which again is the closest point on B.
  - When P is in an edge Voronoi region, clamping P corresponds to an orthogonal projection onto the edge, which also must be the closest point on B to P.
- This procedure works in both two and three dimensions

Closest Point on AABB to Point

- Distance of Point to AABB
  - When the point Q on an AABB B closest to a given point P is computed only to determine the distance between P and Q, the distance can be calculated without explicitly obtaining Q.

```
    for (int i = 0; i < 3; i++) {
        // For each axis count any excess distance outside box extents
        float v = p[i];
        if (v < b.min[i]) sqDist += (b.min[i] - v) * (b.min[i] - v);
        if (v > b.max[i]) sqDist += (v - b.max[i]) * (v - b.max[i]);
    }
    return sqDist;
}
```

Affine Sums

- Consider the "sum"

$P = \alpha_1 P_1 + \alpha_2 P_2 + \ldots\ldots + \alpha_n P_n$

Can show by induction that this sum makes sense iff

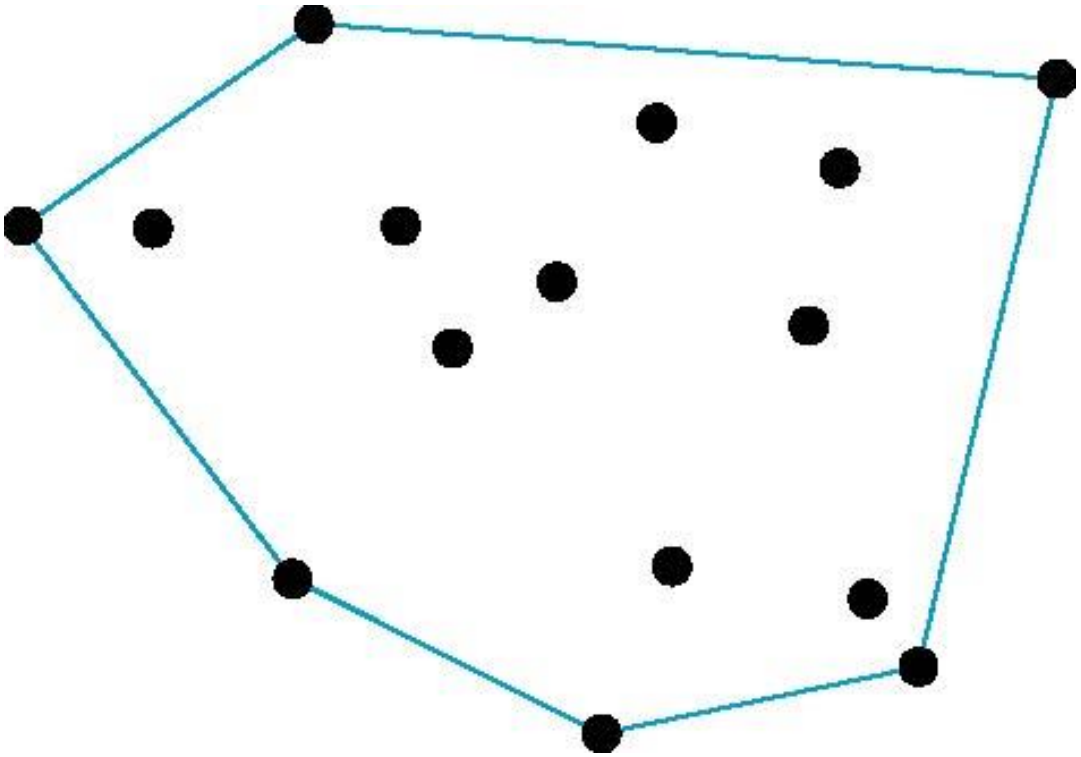
$\alpha_1 + \alpha_2 + \ldots\ldots \alpha_n = 1$

in which case we have the *affine sum* of the points $P_1, P_2, \ldots\ldots P_n$

- If, in addition, $\alpha_i >= 0$, we have the *convex hull* of $P_1, P_2, \ldots\ldots P_n$

Convex Hull

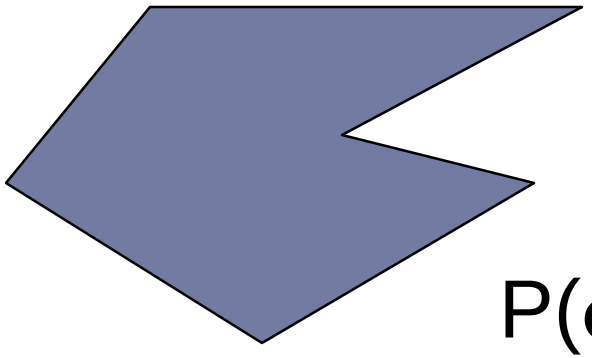
- Smallest convex object containing $P_1, P_2, \ldots . P_n$
- Formed by "shrink wrapping" points



Curves and Surfaces

- Curves are one parameter entities of the form $P(\alpha)$ where the function is nonlinear
- Surfaces are formed from two-parameter functions $P(\alpha, \beta)$
  - Linear functions give planes and polygons

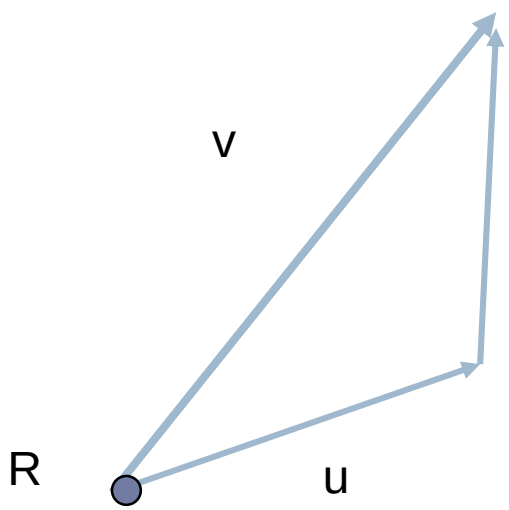
$P(\alpha)$



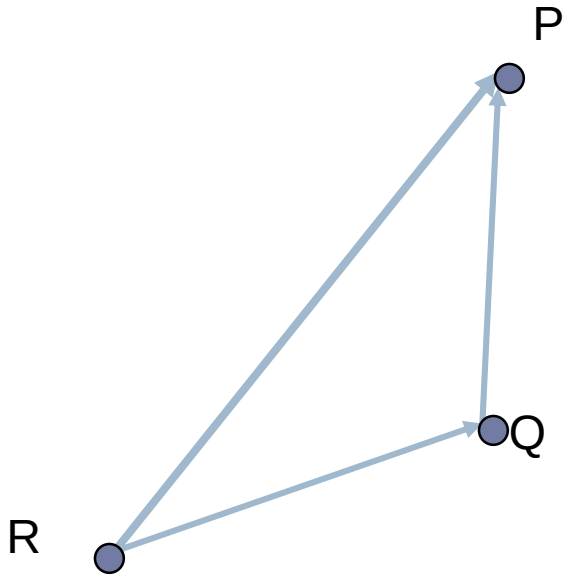
$P(\alpha, \beta)$

Planes

- A plane can be defined by a point and two vectors or by three points

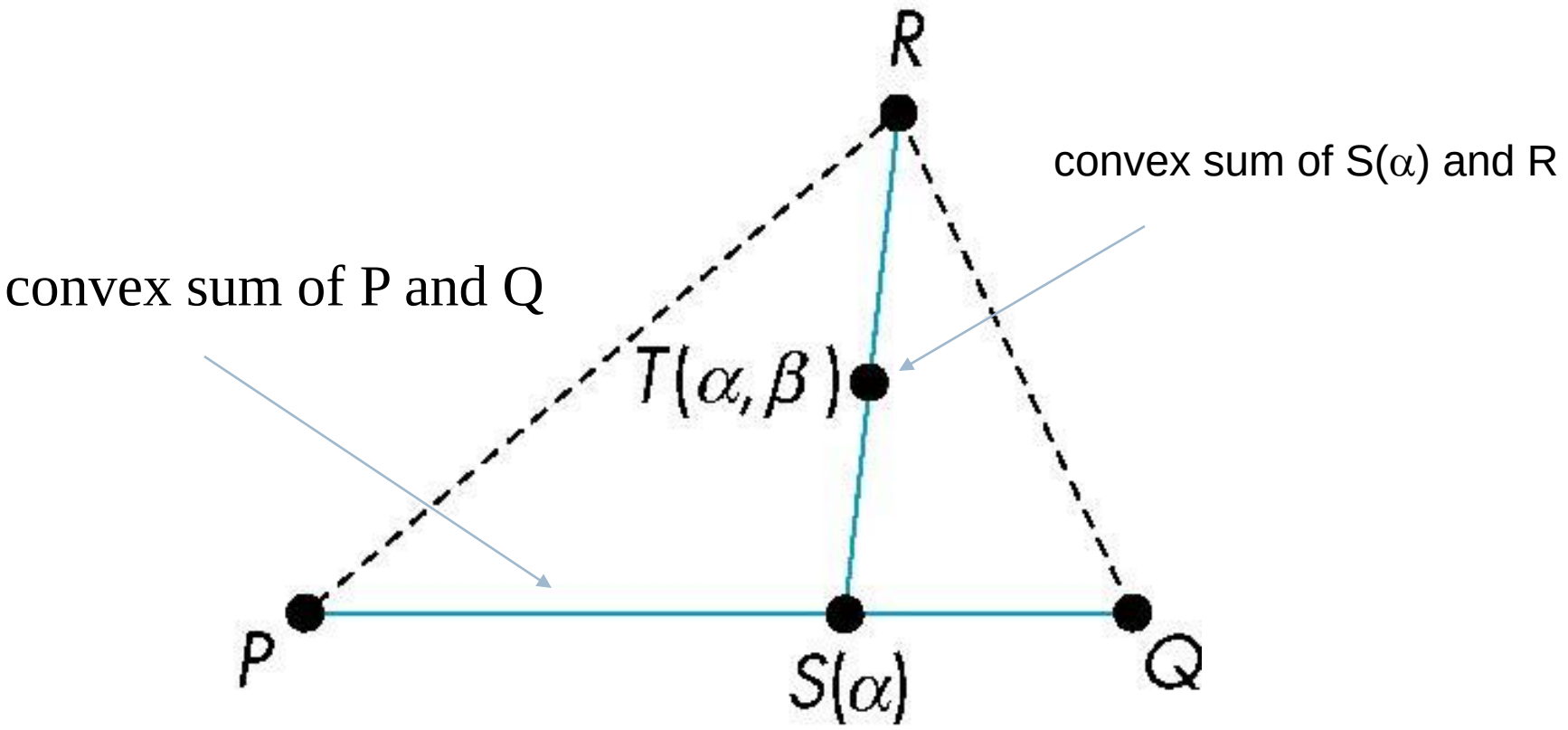


$P(\alpha,\beta)=R+\alpha u+\beta v$



$P(\alpha,\beta)=R+\alpha(Q-R)+\beta(P-Q)$

Triangles

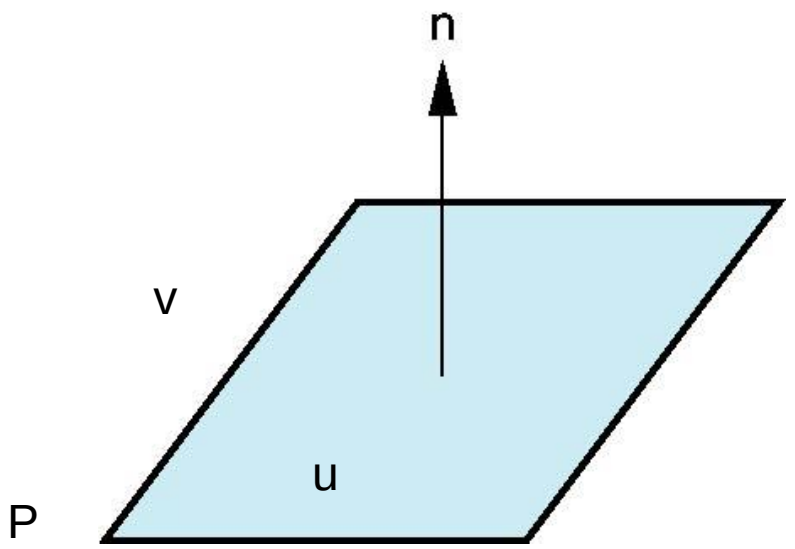


for 0<=α,β<=1, we get all points in triangle

Normals

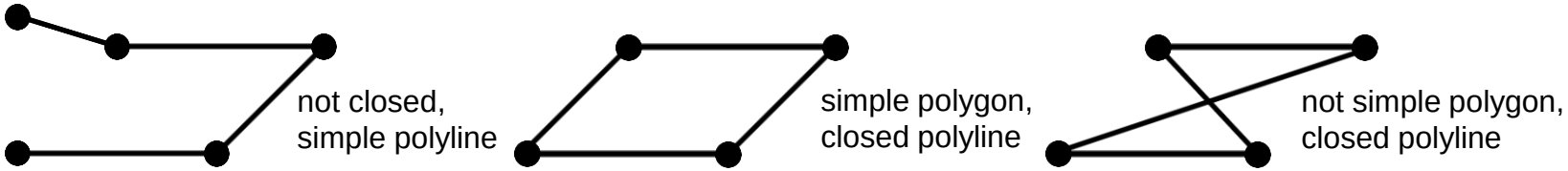
- Every plane has a vector n normal (perpendicular, orthogonal) to it
- From point-two vector form $P(\alpha,\beta)=R+\alpha u+\beta v$, we know we can use the cross product to find $n = u \times v$ and the equivalent form

$$(P(\alpha)-P) \cdot n=0$$



2D Object Definition (1/3)

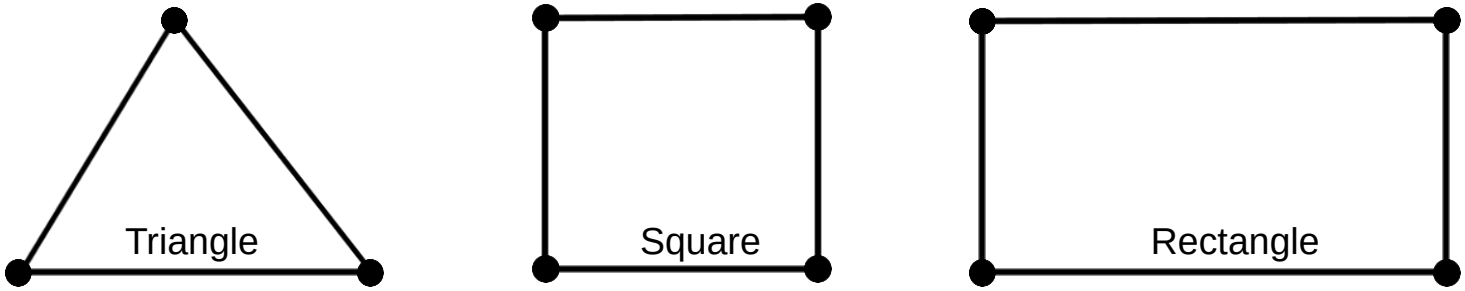
- Lines and polylines:
  - Polylines: lines drawn between ordered points
  - A closed polyline is a polygon, a simple polygon has no self-intersections



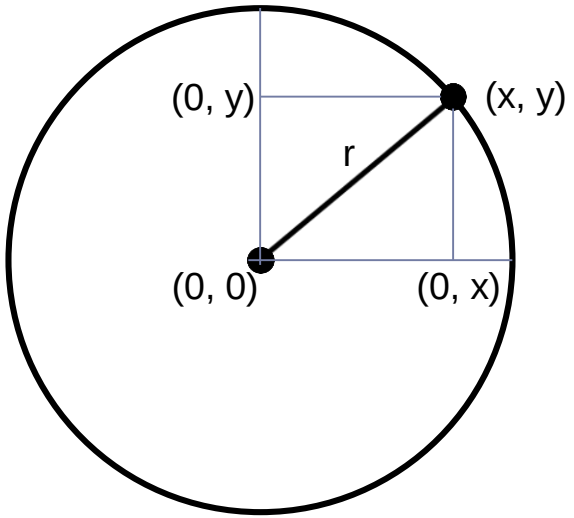
- Convex and concave polygons:
  - Convex: For every pair of points inside the polygon, the line between them is entirely inside the polygon.
  - Concave: For some pair of points inside the polygon, the line between them is not entirely inside the polygon. Not Convex.

2D Object Definition (2/3)

- Special Polygons:

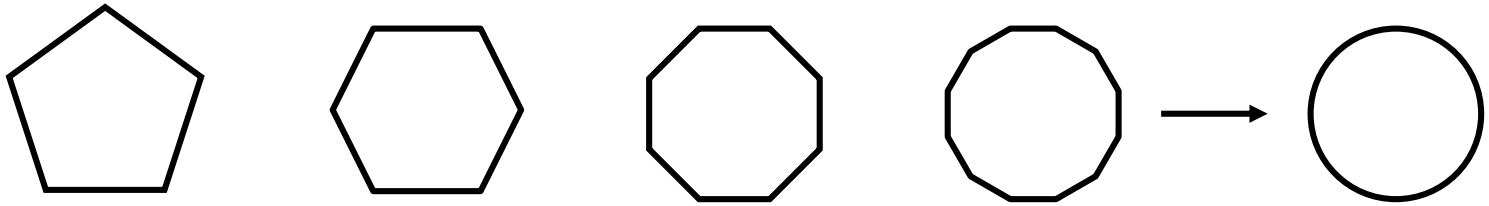


- Circles:
  - Set of all points equidistant from one point called the center
  - The distance from the center is the radius *r*
  - The equation for a circle centered at (0, 0) is $r^2 = x^2 + y^2$

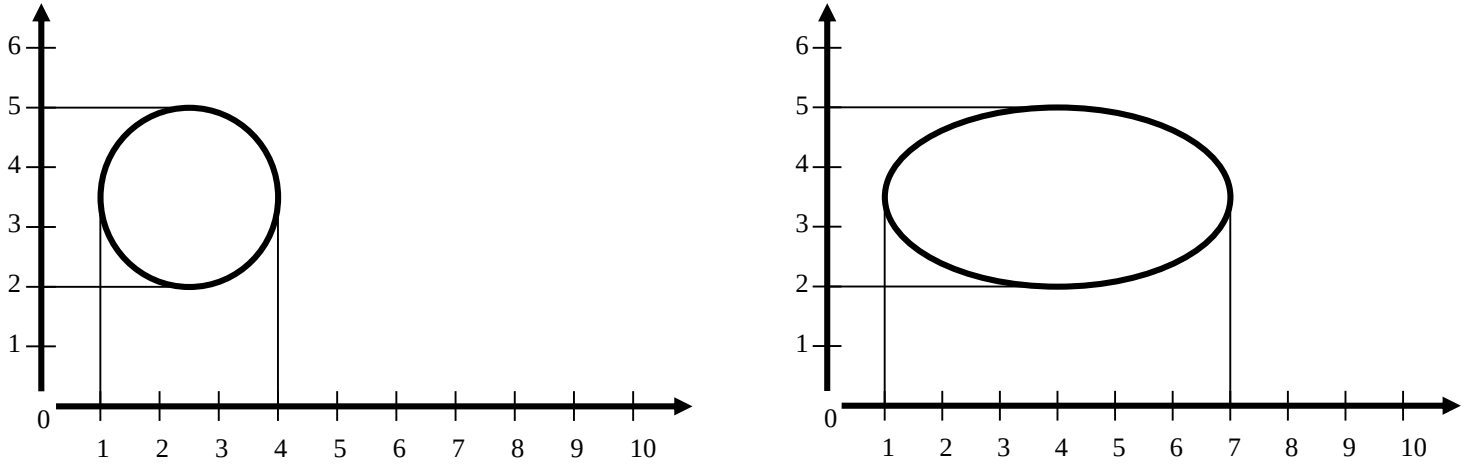


2D Object Definition (3/3)

- A circle can be approximated by a polygon with many sides.



- Axis aligned ellipse: a circle scaled in the x and/or y direction



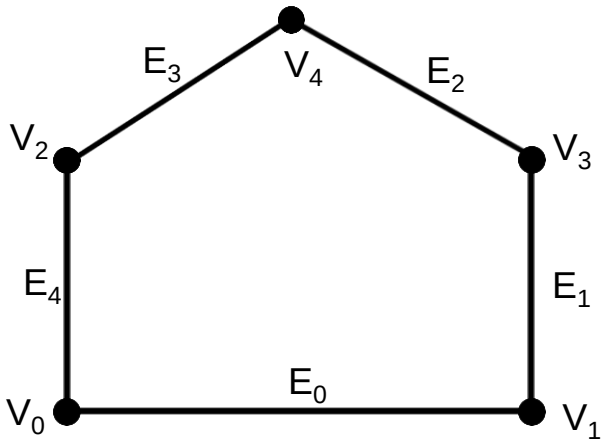
Scaled by a factor of 2 in the x direction and not scaled in the y direction. Width changes from 3 to 6.

Representing Shape

- Vertex and edge tables:
  - General purpose, minimal overhead, reasonably efficient
  - Each vertex listed once
  - Each edge is an ordered pair of indices to the vertex list (like triangles in WPF)

Vertices	
0	(0, 0)
1	(2, 0)
2	(0, 1)
3	(2, 1)
4	(1, 1.5)

Edges	
0	(0, 1)
1	(1, 3)
2	(3, 4)
3	(4, 2)
4	(2, 0)



  - Sufficient to draw shape and perform simple operations (transforms, point inside/outside)
  - Edges listed in counterclockwise order by convention

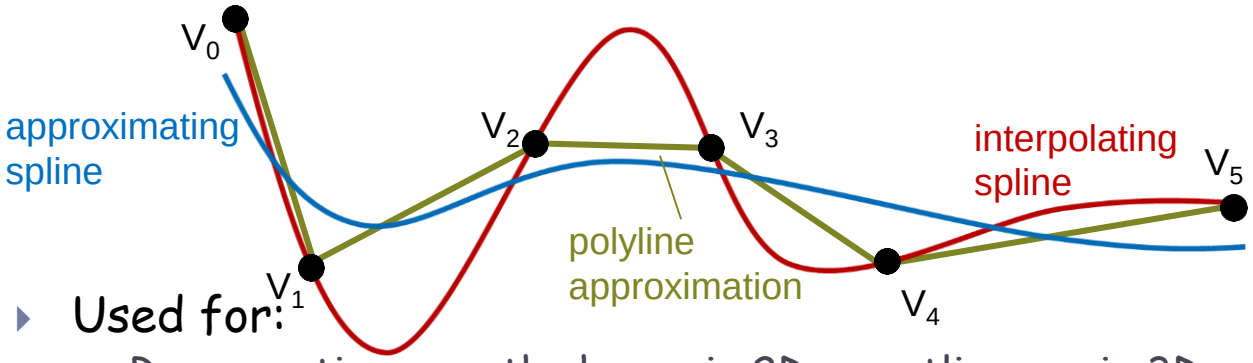
Splines

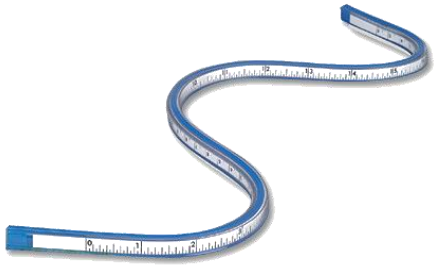
Splines (1/5) - Representing General Curves

- We can represent any polyline with vertices and edges. What about curves?
  - Don't want to store curves as raster graphics (aliasing, not scalable, memory intensive). We need a more efficient mathematical representation
  - Store control points in a list, find some way of smoothly interpolating between them
- Piecewise Linear Approximation
  - Not smooth, looks awful without many control points
- Trigonometric functions
  - Difficult to manipulate and control, computationally expensive to compute
- Higher order polynomials
  - Relatively cheap to compute, only slightly more difficult to operate on than polylines

Splines (2/5) – Spline Types and Uses

- Polynomial interpolation is typically used. Splines are second or third order parametric curves governed by control points or control vectors
- Used early on in automobile and aircraft industry to achieve smoothness – even small differences can make a big difference in efficiency and look





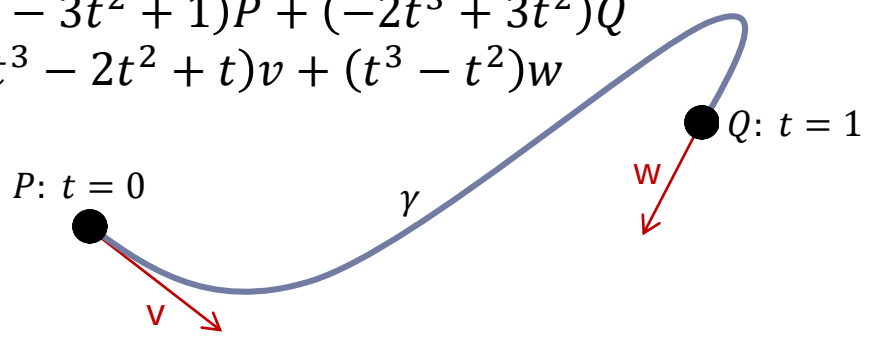
Splines still exist outside of computers. They're now called flexible curves.

- Used for:
  - Representing smooth shapes in 2D as outlines or in 3D using "patches" parameterized with two variables: s and t (see slide 12)
  - Animation paths for "tweening" between keyframes
  - Approximating "expensive" functions (polynomials are cheaper than log, sin, cos …)

Splines (3/5) – Hermite Curves

- Polylines are linear (1st order polynomial) interpolations between points
  - Given points P and Q, line between the two is given by the parametric equation:
    $$x(t) = (1 - \mathrm{t})\mathrm{P} + \mathrm{tQ}, \qquad 0 \le \mathrm{t} \le 1$$
  - $(1 - \mathrm{t})$ and t are called weighting functions of P and Q
- Splines are higher order polynomial interpolations between points
  - Like linear interpolation but with higher order weighting functions allowing better approximations/smoother curves
- One representation - Hermite curves (interpolating spline):
  - Determined by two control points P and Q, an initial tangent vector v and a final tangent vector w.
  - Satisfies:
    - $\gamma(0) = P$
    - $\gamma(1) = Q$
    - $\gamma'(0) = v$
    - $\gamma'(1) = w$

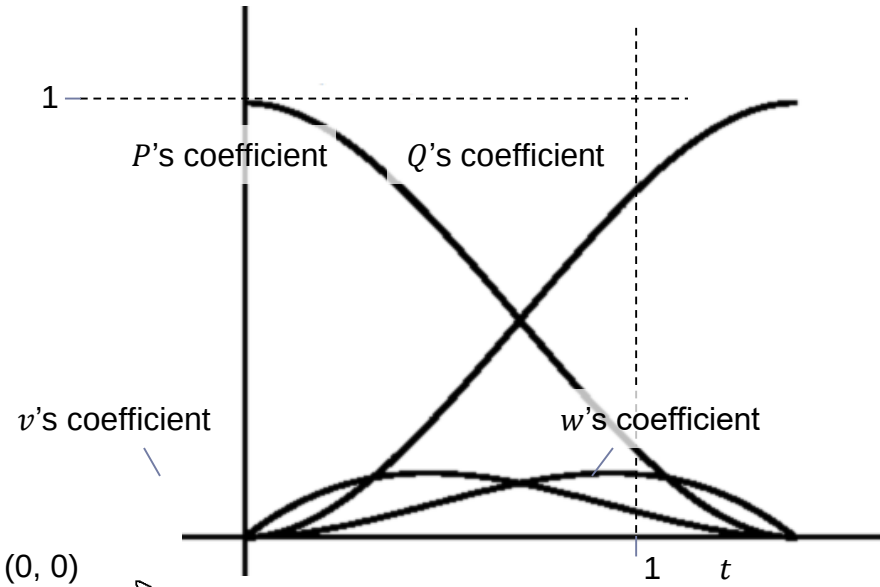
$$\gamma(t) = (2t^3 - 3t^2 + 1)P + (-2t^3 + 3t^2)Q + (t^3 - 2t^2 + t)v + (t^3 - t^2)w$$

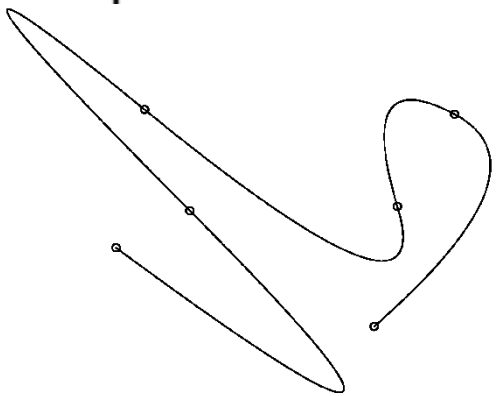


Splines (4/5) – Hermite Weighting Explained

- Polynomial splines have more complex weighting functions than lines
  - Coefficients for P and Q are now 3rd degree polynomials
- At $t = 0$:
  - Coefficient of P is 1, all others 0
  - Derivative of coefficient of v is 1, derivative of all others is 0
- At $t = 1$:
  - Coefficient of Q is 1, all others 0
  - Derivative of coefficient of w is 1, derivative of all others is 0
- Can be chained together to make more complex curves

Polynomial weighting functions in Hermite curve equation





Splines (5/5) – Bezier Curves

- Bezier representation is similar to Hermite
  - 4 points instead of 2 points and 2 vectors ($P_1 \dots P_4$)
  - Initial position P_1, tangent vector is $P_2 - P_1$
  - Final position P_4 tangent vector is $P_4 - P_3$
  - This representation allows a spline to be stored as a list of vertices with some global parameters that describe the smoothness and continuity

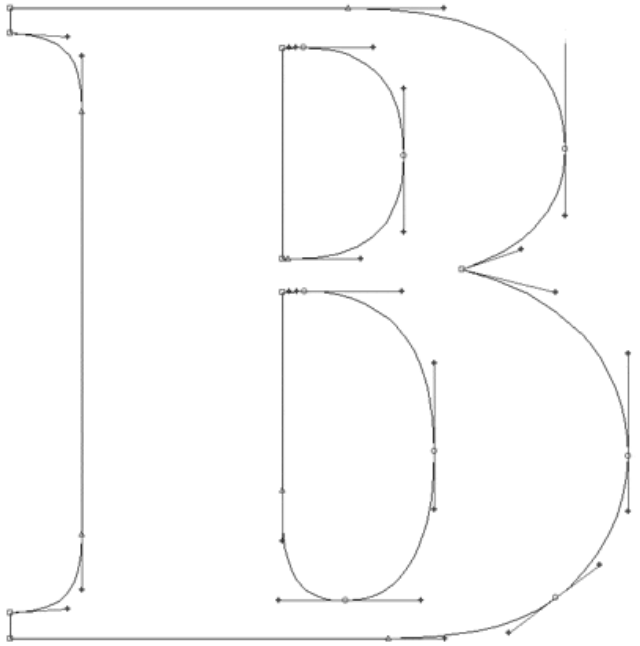


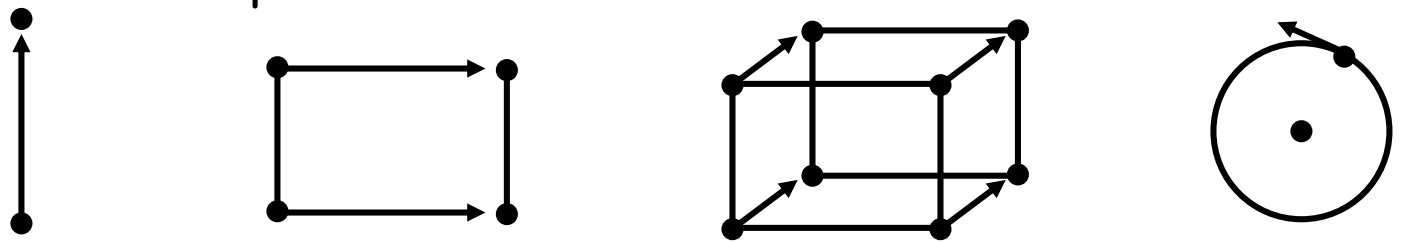
Image credit:
http://miphol.com/muse/2008/04/25/Bezier-courbes-anim.gif

- Bezier splines are widely used (Adobe, Microsoft) for font definition

www.cs.brown.edu/exploratories/freeSoftware/repository/edu/brown/cs/exploratories/applets/bezierSplines/bezier_splines_guide.html

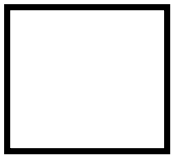
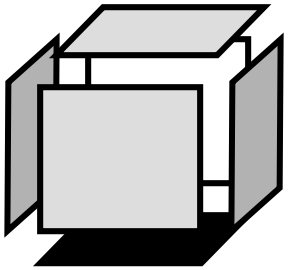
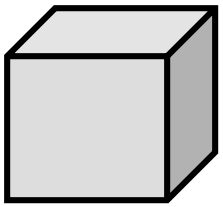
"Vertices in Motion" - Object Definition

- A line is drawn by tracing a point as it moves (one dimension added)
- A rectangle is drawn by tracing the vertices of a line as it moves perpendicularly to itself (2nd dimension added)
- A rectangular prism is drawn by tracing the vertices of a rectangle as it moves perpendicularly to itself (3rd dimension)
- A circle is drawn by tracing a point swinging at a fixed distance around a center point.

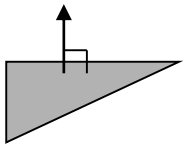
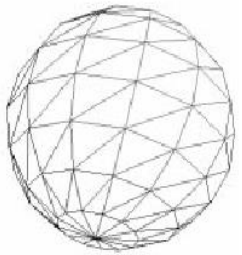
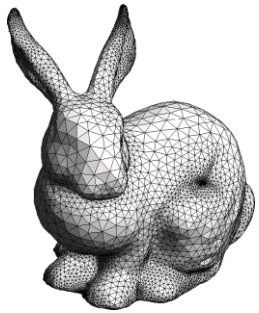


Building 3D Primitives

- Made out of 2D and 1D primitives

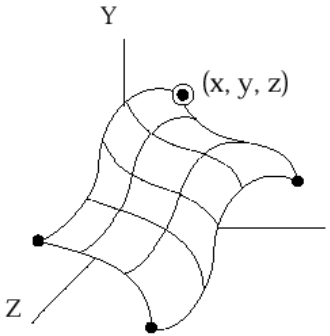





- Triangles are commonly used
- Many triangles used for a single object is a triangular mesh.

- Splines used to describe boundaries of "patches" – these can be "sewn together" to represent curved surfaces

$$x(s,t) = (1-s)^3 * (1-t)^3 * P_{1,1}$$
$$+(1-s)^3 * 3t(1-t)^2 * P_{1,2} + \ldots$$



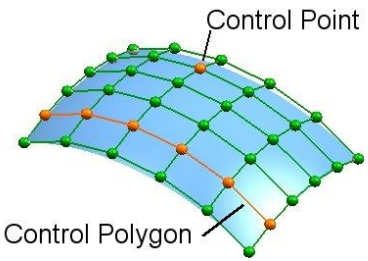


Image credit (Stanford Bunny): http://cmp.felk.cvut.cz/~dr/papers/Habil/img100.gif

Triangle Meshes

- Most common representation of shape in three dimensions
- All vertices of triangle are guaranteed to lie in one plane (not true for quadrilaterals or other polygons)
- Uniformity makes it easy to perform mesh operations such as subdivision, simplification, transformation etc.
- Many different ways to represent trianaular meshes

- See chapters 8 and 28 in book,
  en.wikipdia.org/wiki/polygon_mesh
  - Mesh transformation and deformati
  - Procedural generation techniques (upcoming labs on simulating terrain)

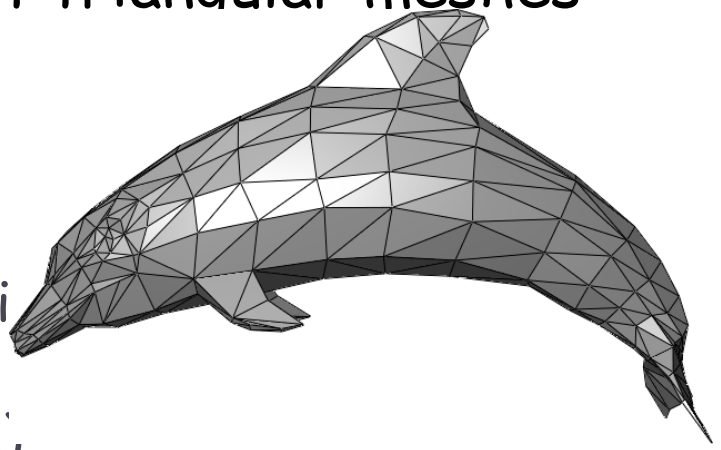


Image credit:
http://upload.wikimedia.org/wikipedia/en/f/fb/Dolphin_triangle_mesh.png

Triangular Mesh Representation

- Vertex and face tables, analogous to 2D vertex and edge tables
- Each vertex listed once, triangles listed as ordered triplets of indices into the vertex table
  - Edges inferred from triangles
  - It's often useful to store associated faces with vertices (i.e. computing normals: vertex normal average of surrounding face normals)
- Vertices listed in counter clockwise order in face table.
  - No longer just because of convention. CCW order differentiates front and back of face

Vertex List

v0	0, 0, 0	f0 f1 f12 f15 f7
v1	1, 0, 0	f2 f3 f13 f12 f1
v2	1, 1, 0	f4 f5 f14 f13 f3
v3	0, 1, 0	f6 f7 f15 f14 f5
v4	0, 0, 1	f6 f7 f0 f8 f11
v5	1, 0, 1	f0 f1 f2 f9 f8
v6	1, 1, 1	f2 f3 f4 f10 f9
v7	0, 1, 1	f4 f5 f6 f11 f10
v8	.5, .5, 0	f8 f9 f10 f11
v9	.5, .5, 1	f12 f13 f14 f15

Face List

f0	v0 v4 v5
f1	v0 v5 v1
f2	v1 v5 v6
f3	v1 v6 v2
f4	v2 v6 v7
f5	v2 v7 v3
f6	v3 v7 v4
f7	v3 v4 v0
f8	v8 v5 v4
f9	v8 v6 v5
f10	v8 v7 v6
f11	v8 v4 v7
f12	v9 v5 v4
f13	v9 v6 v5
f14	v9 v7 v6
f15	v9 v4 v7

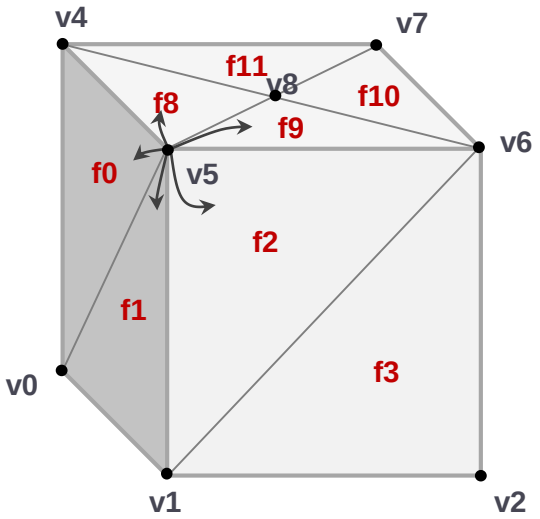


Diagram licensed under Creative Commons Attribution license. Created by Ben Herila based on http://upload.wikimedia.org/wikipedia/en/thumb/2/2d/Mesh_fv.jpg/500px-Mesh_fv.jpg

Ερωτήσεις

- Ιστοσελίδα μαθήματος (ενεργοποιημένη) :

  http://support.inf.uth.gr/courses/CE416/
  http://eclass.uth.gr/eclass/MHX101/

- E-mail λίστα του μαθήματος:

  ce416@inf-server.inf.uth.gr

  …και μέσω eclass…

- Π. Τσομπανοπούλου, Ε3-12, yota@uth.gr