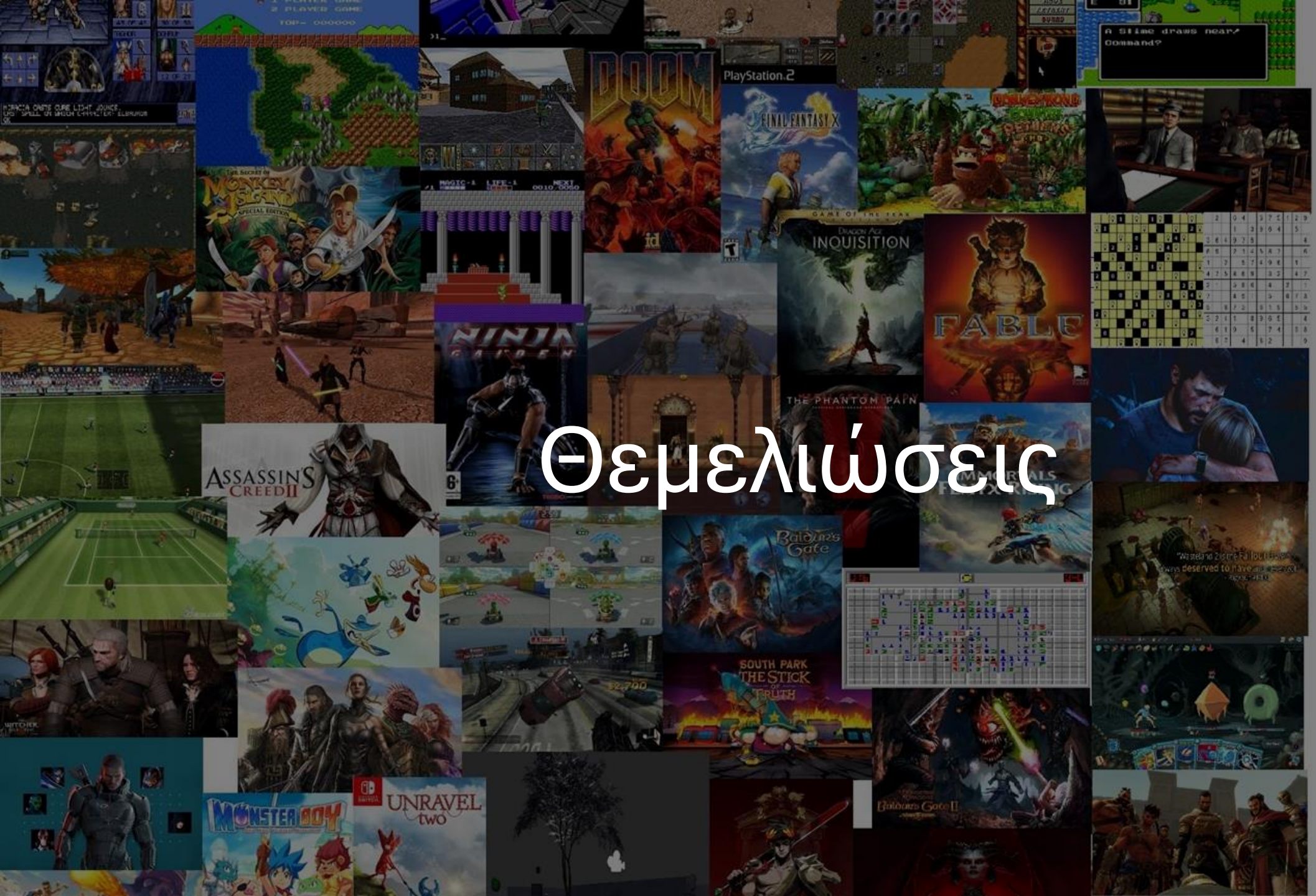


Θεμελιώσεις

Math is everything when it comes to video games. From having the ability to calculating the trajectory of an Angry Bird flying through the sky, to ensuring that a character can jump and come back down to the ground. Without the help of mathematics, video games simply wouldn't work.

Matthew S. Stenquist, Video Game Developer

Μαθηματικά

Για βιντεοπαιχνίδια...

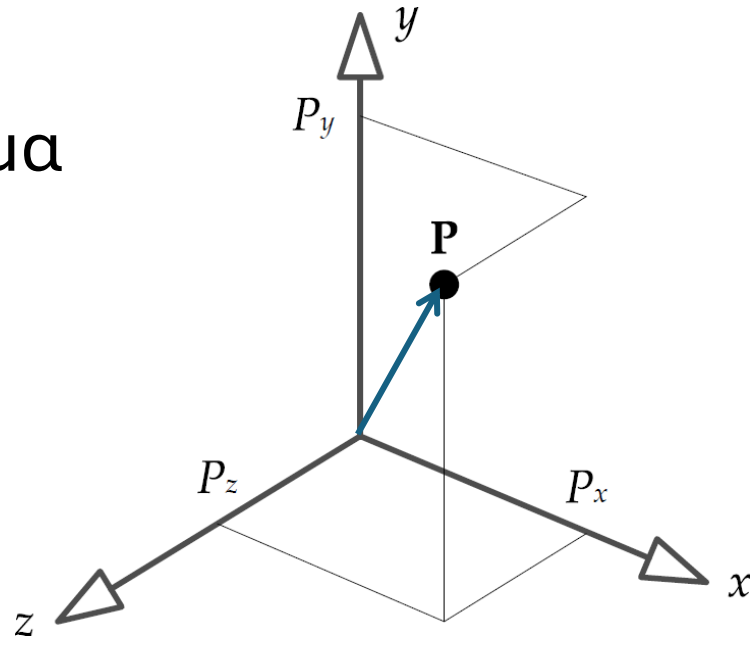
Μαθηματικά Για Βιντεοπαιχνίδια

- Σημεία και διανύσματα
- Συστήματα συντεταγμένων
- Μητρώα
- Quaternions
- Σύγκριση αναπαραστάσεων μετασχηματισμών περιστροφής
- Κάποια χρήσιμα μαθηματικά αντικείμενα
- Αναπαράσταση Καμπυλών

2d – Σημεία και *Τέρατα* (Εννοώ Διανύσματα ☺)

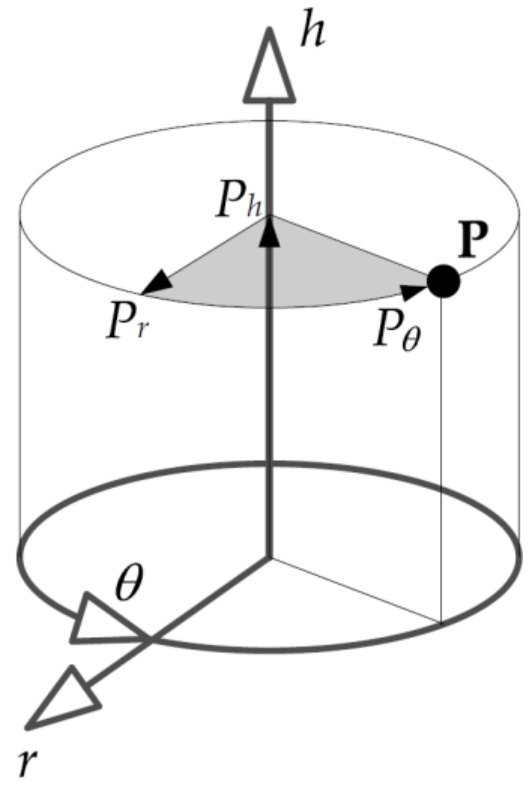
- Οι περισσότερες λειτουργίες σε 3d έχουν νόημα και σε 2d
- Χρησιμοποιείτε πάντα το 2d ως βάση, θα σας βοηθήσει

- Ένα σημείο είναι μια θέση στον n-διάστατο χώρο
- Συνήθως αναπαρίσταται στον καρτεσιανό σύστημα συντεταγμένων
  - Δύο ή τρεις αμοιβαίοι κάθετοι άξονες
  - Ένα σημείο $\mathbf{P}$ είναι μία τριάδα (P_x, P_y, P_z)
  - Το διάνυσμα $\vec{p}$ είναι το $[p_x\ p_y\ p_z]^T$

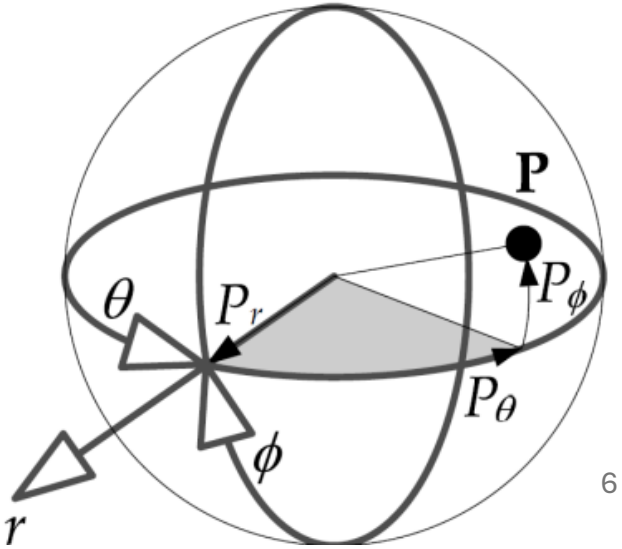


Άλλα Συστήματα Συντεταγμένων

- Κυλινδρικά
  - Χρήση ενός άξονα ύψους (h), ενός ακτινωτού άξονα (radial – r) και μίας γωνίας εκτροπής (yaw – θ)
  - Τα σημεία αναπαρίστανται ως (P_h, P_r, P_θ)



- Σφαιρικά
  - Γωνία βήματος (pitch – φ), γωνία εκτροπής (yaw – θ) και ακτινωτού άξονα (radial – r)
  - Τα σημεία αναπαρίστανται ως $(P_r, P_\varphi, P_\theta)$

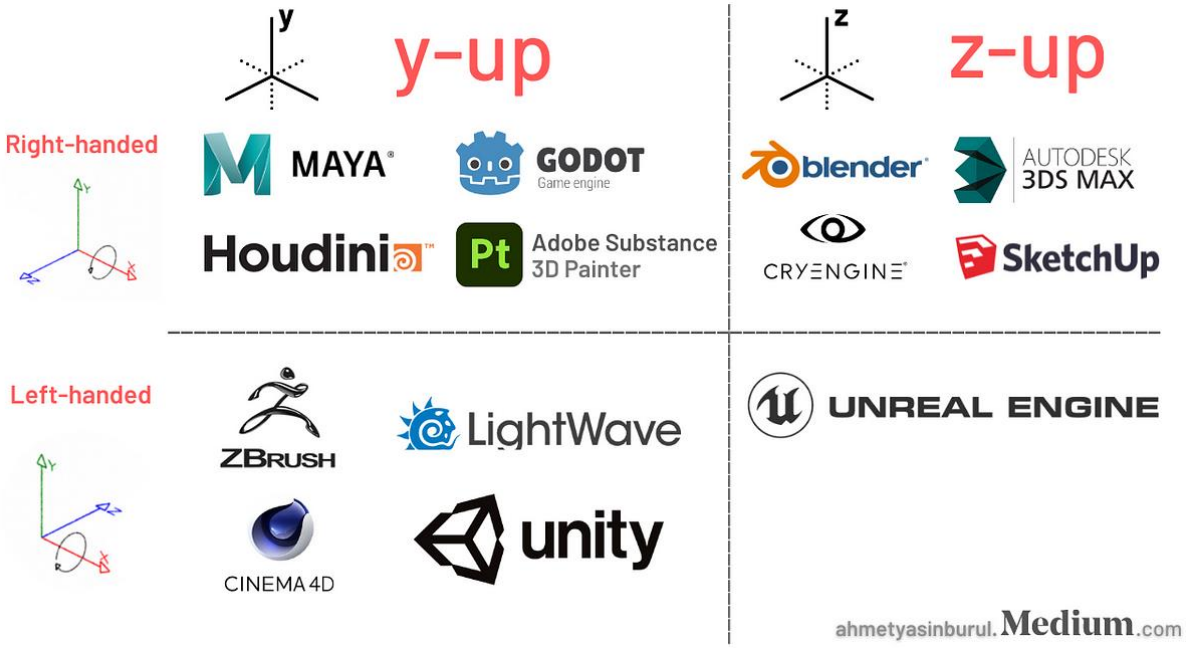


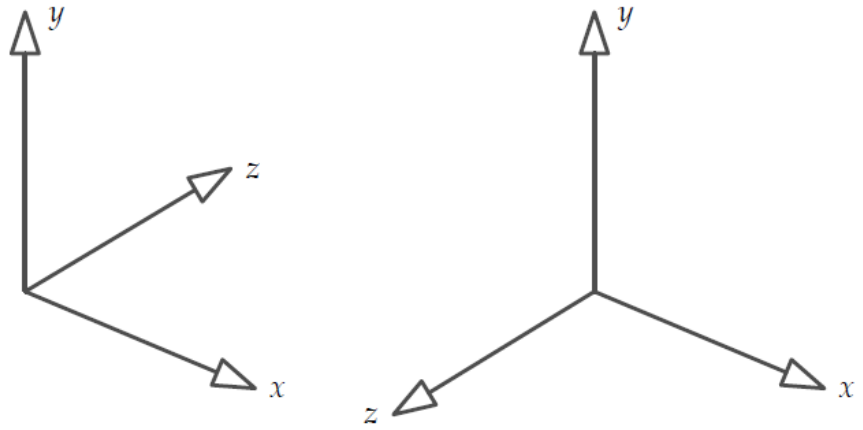
Επιλέγοντας Σύστημα

- Το καρτεσιανό σύστημα χρησιμοποιείται περισσότερο
- Μερικές φορές είναι πιο εύκολη η χρήση άλλου συστήματος
  - Τα αντικείμενα που στροβιλίζονται γύρω από έναν χαρακτήρα είναι ευκολότερα σε κυλινδρικό σχήμα
  - Οι εκρήξεις μπορεί να είναι ευκολότερες σε σφαιρικό

Αριστερόστροφο–Δεξιόστροφο Καρτεσιανό Σύστημα

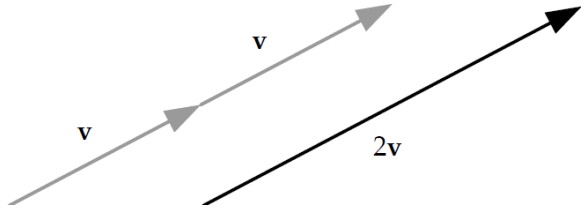
- Η μετατροπή είναι εύκολη, αντιστρέψτε την κατεύθυνση οποιουδήποτε άξονα.
- Οι προγραμματιστές γραφικών συνήθως επιλέγουν αριστερόστροφα συστήματα με το y να δείχνει προς τα πάνω, το x προς τα δεξιά και το z να δείχνει στην οθόνη
- Βοηθά με την τεχνική z-buffering





Αριστερόστροφο Δεξιόστροφο

Υπενθύμιση: Διανύσματα



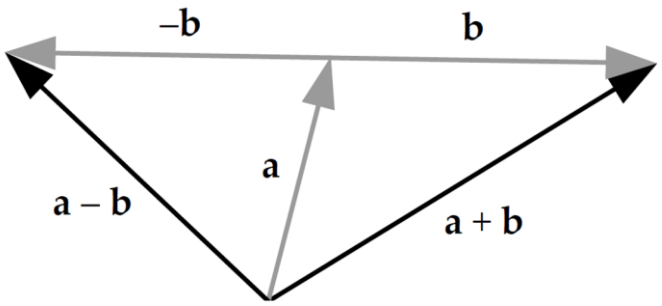
- Κλιμάκωση χωρίς αλλαγή κατεύθυνσης $s\mathbf{a} = (sa_x, sa_y, sa_z)$

- Ανομοιόμορφη κλιμάκωση $\mathbf{s} \otimes \mathbf{a} = (s_x a_x, s_y a_y, s_z a_z)$

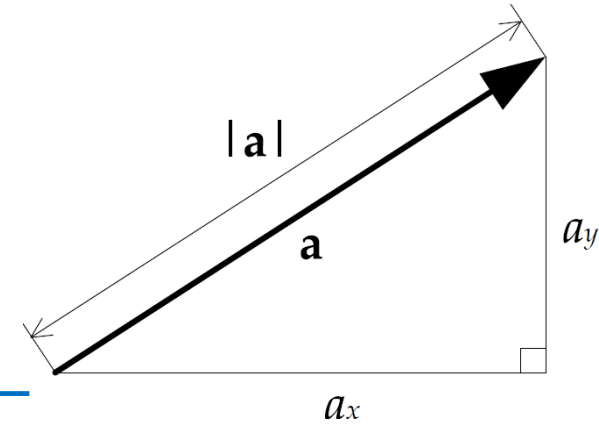
$$a\mathbf{S} = \begin{bmatrix} a_x & a_y & a_y \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} = \begin{bmatrix} a_x s_x & a_y s_y & a_z s_z \end{bmatrix}$$

- Πρόσθεση και Αφαίρεση

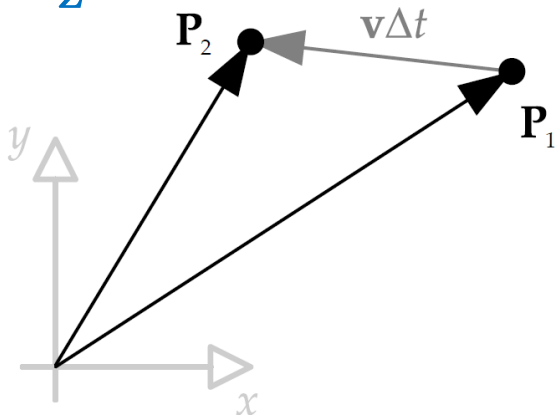
$$a + b = \left[(a_x + b_x), (a_y + b_y), (a_z + b_z) \right]$$
$$a - b = \left[(a_x - b_x), (a_y - b_y), (a_z - b_z) \right]$$



Άλλες Πράξεις σε Διανύσματα

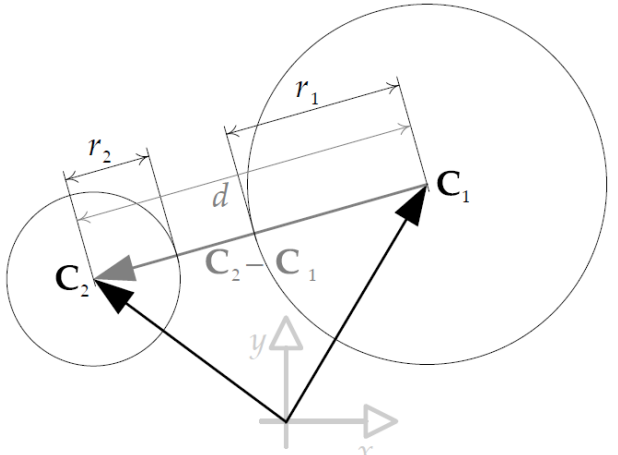


- Υπολογισμός μέτρου (απόσταση): $|a| = \sqrt{a_x^2 + a_y^2 + a_z^2}$

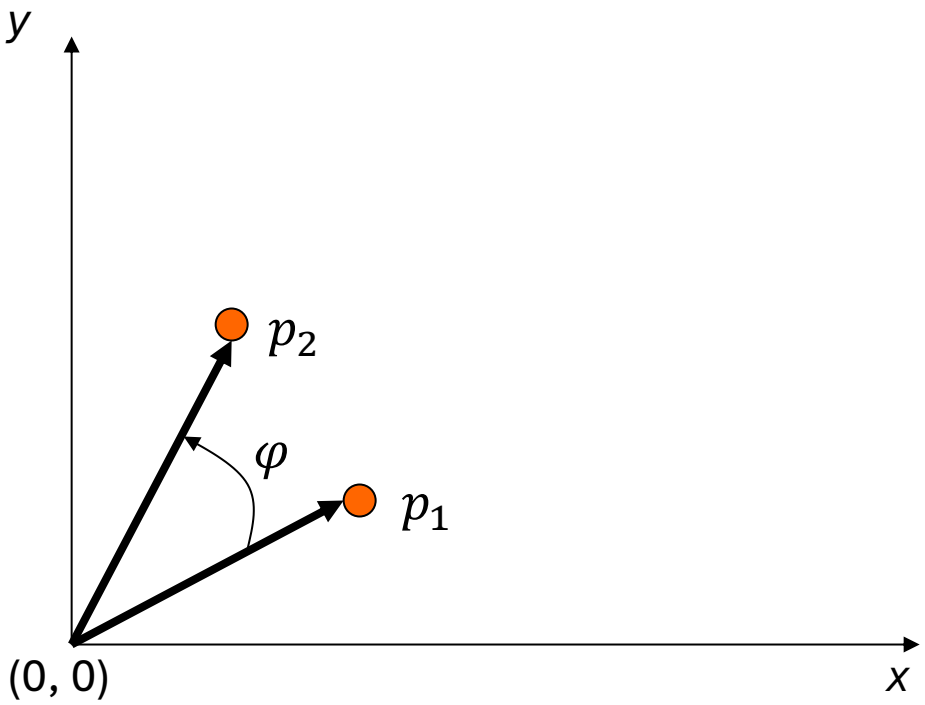


- Μετακίνηση αντικειμένων: $P_2 = P_1 + v\Delta t$

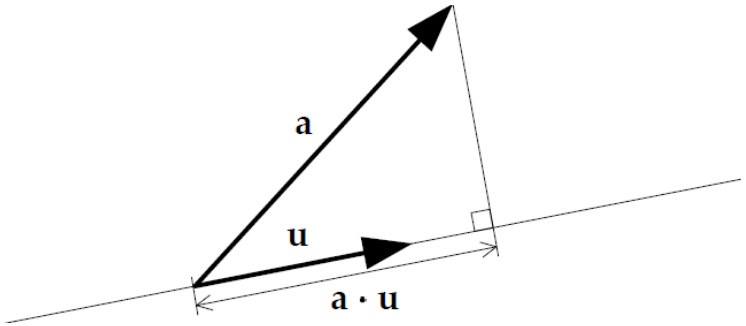
- Σύγκρουση αντικειμένων:
  - Αν $d < r_1 + r_2$ τότε συγκρούονται (;;;)
  - Πιο γρήγορο να συγκρίνετε $d^2 < (r_1 + r_2)^2$

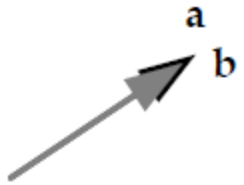


Εσωτερικό Γινόμενο

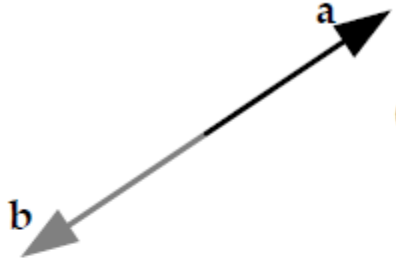


$$p_1 \cdot p_2 = x_1x_2 + y_1y_2 = p_2 \cdot p_1 = |p_1||p_2|cos\varphi$$

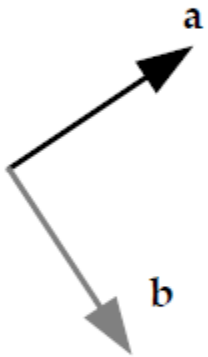




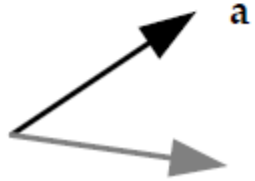
$(\mathbf{a} \cdot \mathbf{b}) = ab$



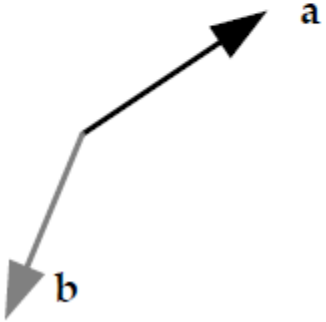
$(\mathbf{a} \cdot \mathbf{b}) = -ab$



$(\mathbf{a} \cdot \mathbf{b}) = 0$



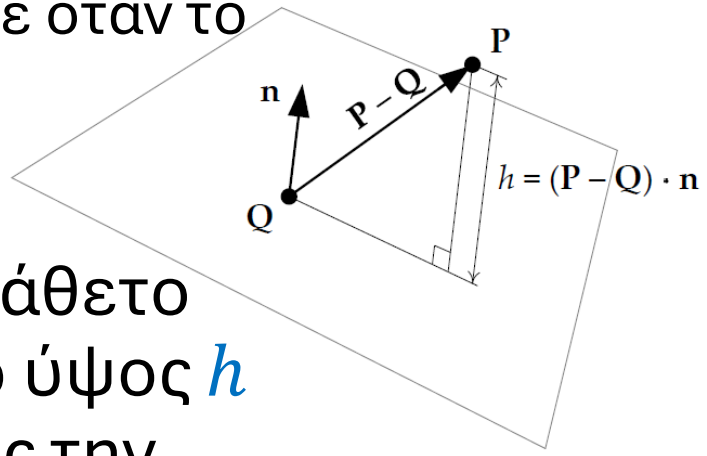
$(\mathbf{a} \cdot \mathbf{b}) > 0$



$(\mathbf{a} \cdot \mathbf{b}) < 0$

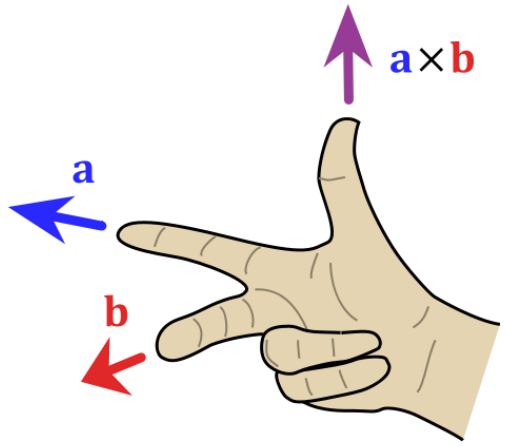
Μερικές Απλές Εφαρμογές

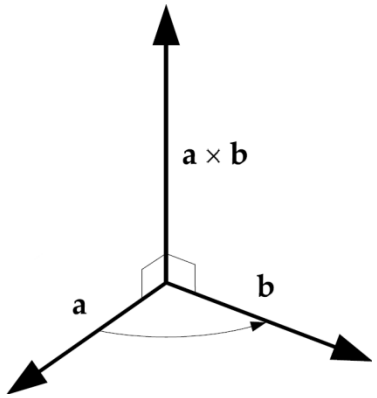
- Ορατότητα: μπορεί ο παίκτης P να δει τον εχθρό E;
  - $v = E - P$ δίνει τη σχετική θέση του E σε σχέση με τον P
  - αν το f είναι το διάνυσμα που ορίζει τι αντικρύζει ο P τότε όταν το $v \cdot f$ είναι αρνητικό ο E βρίσκεται πίσω από τον P



- Αν ορίσουμε ένα επίπεδο από το σημείο Q και ένα κάθετο μοναδιαίο διάνυσμα n τότε μπορούμε να βρούμε το ύψος h ενός σημείου P πάνω από το επίπεδο υπολογίζοντας την προβολή

Εξωτερικό Γινόμενο





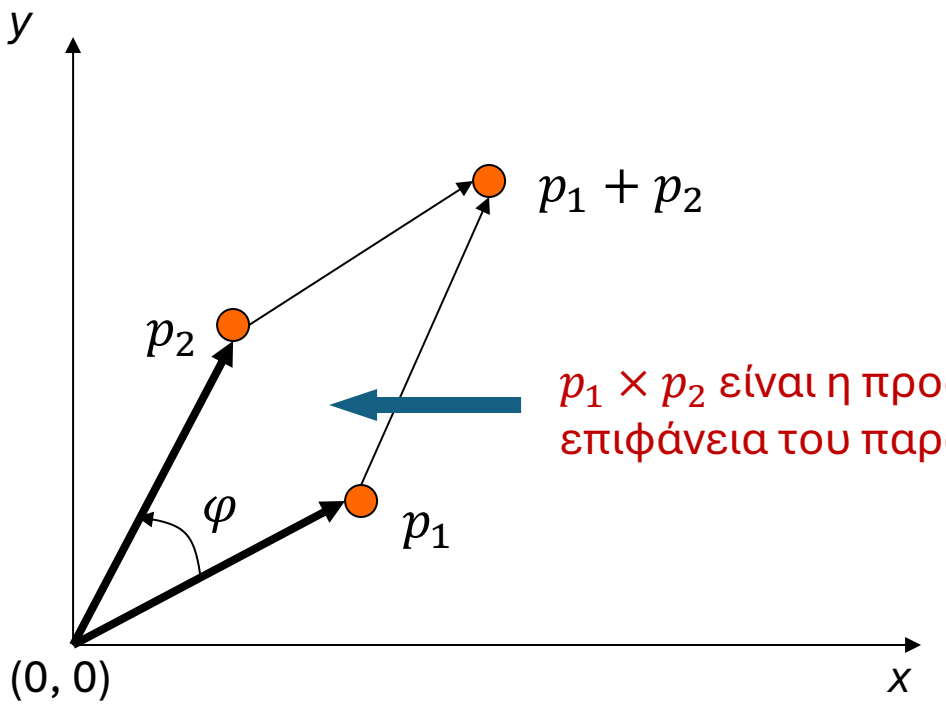
Δίνει ένα άλλο διάνυσμα που είναι κάθετο στα διανύσματα που πολλαπλασιάζονται:

$$a \times b = \begin{vmatrix} \hat{\imath} & \hat{\jmath} & \hat{k} \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix}$$

Στις δύο διαστάσεις, $a_z = b_z = 0$. Άρα: $a \times b = \begin{vmatrix} \hat{\imath} & \hat{\jmath} & \hat{k} \\ a_x & a_y & 0 \\ b_x & b_y & 0 \end{vmatrix} = \hat{k} \begin{vmatrix} a_x & a_y \\ b_x & b_y \end{vmatrix}$

Εξωτερικό Γινόμενο

$$p_1 \times p_2 = \begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} = -\begin{vmatrix} y_1 & y_2 \\ x_1 & x_2 \end{vmatrix} = -p_2 \times p_1$$



$$p_1 \times p_2 = x_1 y_2 - x_2 y_1 = -p_2 \times p_1 = |p_1||p_2| sin\varphi$$

Τα p_1 , p_2 είναι συνευθειακά αν και μόνο αν $p_1 \times p_2 = 0$

Εφαρμογές του Εξωτερικού Γινομένου

- Εύρεση ενός διανύσματος που είναι κάθετο σε δύο άλλα διανύσματα

- Εύρεση του κάθετου μοναδιαίου διανύσματος σε ένα επίπεδο

$$n = normalize\big((P_2 - P_1) \times (P_3 - P_1)\big)$$

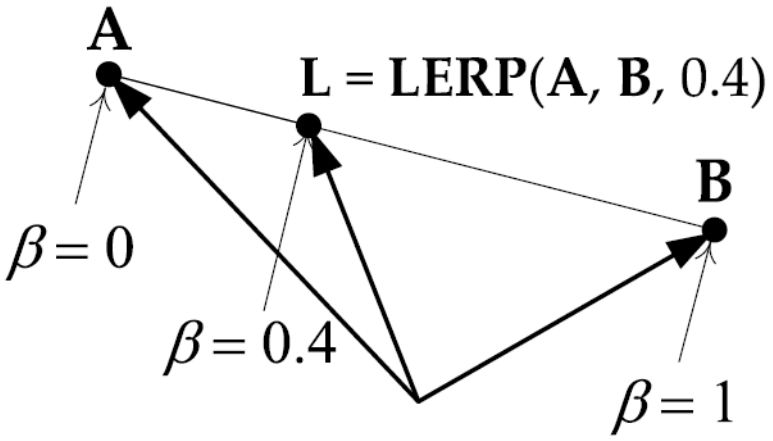
- Υπολογισμός ροπής
  - Δοθέντων δύναμης F και διανύσματος r από το κέντρο μάζας, η ροπή είναι

$$N = r \times F$$

LERP (Linear intERPolation)

Απλή γραμμική παρεμβολή μεταξύ δύο σημείων. $\beta \in [0,1]$

$$\mathbf{L} = \mathrm{LERP}(\mathbf{A}, \mathbf{B}, \beta) = (1-\beta)\mathbf{A} + \beta\mathbf{B}$$
$$= \left[(\mathbf{1}-\beta)\mathbf{A}_{\boldsymbol{x}} + \beta\mathbf{B}_{x}, (\mathbf{1}-\beta)\mathbf{A}_{\boldsymbol{y}} + \beta\mathbf{B}_{y}, (\mathbf{1}-\beta)\mathbf{A}_{\boldsymbol{z}} + \beta\mathbf{B}_{z}\right]$$



Στα παιχνίδια, συχνά χρειάζεται να βρούμε ένα διάνυσμα που βρίσκεται στη μέση μεταξύ δύο διανυσμάτων. Για παράδειγμα, εάν θέλουμε να κινούμε ομαλά ένα αντικείμενο από το σημείο **A** στο σημείο **B** κατά τη διάρκεια δύο δευτερολέπτων με 30 fps, θα πρέπει να υπολογίσετε 60 ενδιάμεσες θέσεις μεταξύ **A** και **B**.

MATRIX TRANSFORMATIONS

Μητρώα και Μετασχηματισμοί

Μετασχηματισμοί

- Ένα μητρώο 2×2 μπορεί να χρησιμοποιηθεί για να αναπαραστήσει περιστροφές σε 2 διαστάσεις

$$[r_x' \quad r_y'] = [r_x \quad r_y] \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix}$$

- Το ίδιο ισχύει και για τις 3 διαστάσεις

$$[r_x' \quad r_y' \quad r_z'] = [r_x \quad r_y \quad r_z] \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Ομογενείς Συντεταγμένες

- Μπορεί να χρησιμοποιηθεί ένα 3×3 μητρώο για αναπαράσταση του μετασχηματισμού μεταφοράς;

ΟΧΙ: $\mathbf{r} + \mathbf{t} = [r_x + t_x \quad r_y + t_y \quad r_z + t_z]$

- Μπορούμε όμως αν χρησιμοποιήσουμε μητρώο 4×4

$$\mathbf{r} + \mathbf{t} = [r_x \quad r_y \quad r_z \quad 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_x & t_x & 1 \end{bmatrix} = [r_x + t_x \quad r_y + t_y \quad r_z + t_z \quad 1]$$

Μητρώα Ατομικών Μετασχηματισμών

Ένας 4 x 4 μητρώο μετασχηματισμού διαχωρίζεται σε 4 υπομητρώα:

$$M = \begin{bmatrix} \mathbf{U}_{3\times3} & \mathbf{0}_{3\times1} \\ \mathbf{t}_{1\times3} & 1 \end{bmatrix}$$

Όταν πολλαπλασιάζουμε ένα σημείο με ένα μητρώο μετασχηματισμού τότε

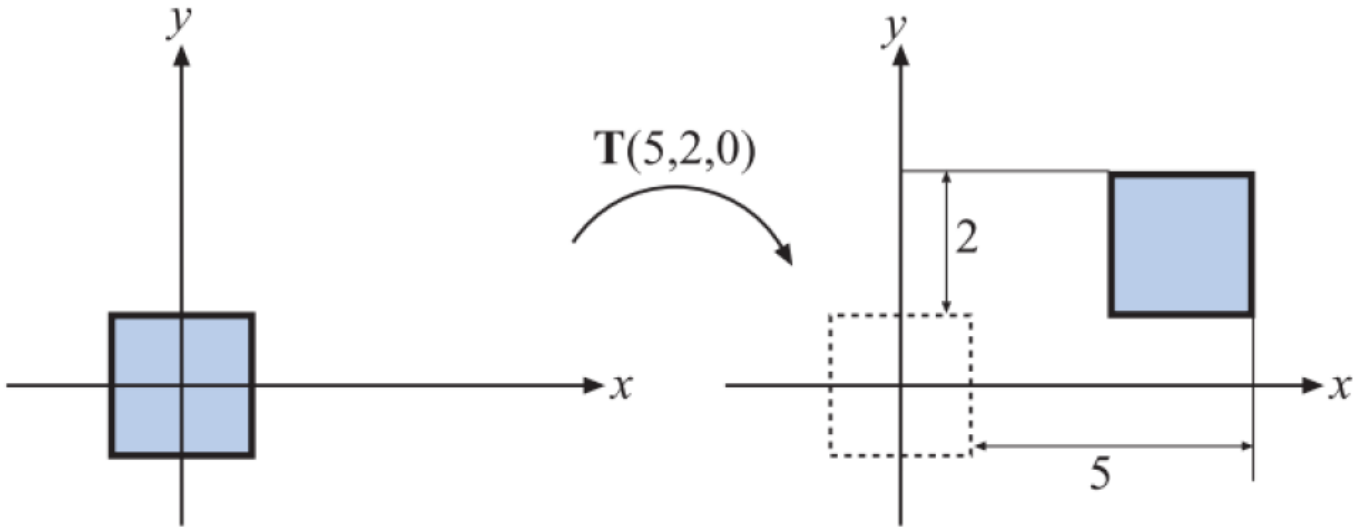
$$[\mathbf{r}'_{1\times3} \quad 1] = [\mathbf{r}_{1\times3} \quad 1] \begin{bmatrix} \mathbf{U}_{3\times3} & \mathbf{0}_{3\times1} \\ \mathbf{t}_{1\times3} & 1 \end{bmatrix} = [\mathbf{rU} + \mathbf{t} \quad 1]$$

- $\mathbf{U}$: αναπαριστά περιστροφή ή/και κλιμάκωση
- $\mathbf{t}$: αναπαριστά μεταφορά
- $\mathbf{0} = [\mathbf{0}\ \mathbf{0}\ \mathbf{0}]^{\mathbf{T}}$
- Ένα βαθμωτό μέγεθος 1

Στοιχειώδης Μεταφορά

$$\mathbf{r}+\mathbf{t}=[r_x \quad r_y \quad r_z \quad 1]\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ t_x & t_x & t_x & 1\end{bmatrix}=[r \quad 1]\begin{bmatrix}I & 0\\ t & 1\end{bmatrix}=[r+t \quad 1]$$

- Το αντίστροφο είναι απλά το μητρώο με αρνητικό **t**



Στοιχειώδης Περιστροφή

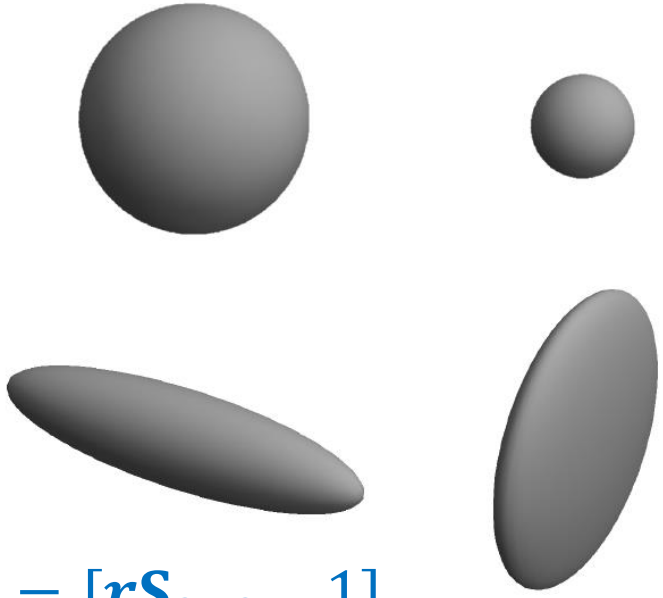
$$\text{rotate}_x(\mathbf{r}, \varphi) = [r_x \quad r_y \quad r_z \quad 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & \sin\varphi & 0 \\ 0 & -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{rotate}_y(\mathbf{r}, \theta) = [r_x \quad r_y \quad r_z \quad 1] \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{rotate}_y(\mathbf{r}, \gamma) = [r_x \quad r_y \quad r_z \quad 1] \begin{bmatrix} \cos\gamma & \sin\gamma & 0 & 0 \\ -\sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Το 1 εντός του μητρώου περιστροφής εμφανίζεται πάντα στον άξονα περιστροφής, το ημίτονο και το συνημίτονο εμφανίζονται στους άλλους άξονες
- Οι θετικές περιστροφές πηγαίνουν από
  - x προς y – γύρω από τον z
  - y προς z – γύρω από τον x
  - z προς x – γύρω από τον y
- Το αντίστροφο μιας στοιχειώδους περιστροφής αναπαρίσταται από το ανάστροφο μητρώο

Κλιμάκωση



$$\boldsymbol{rS} = [r_x \quad r_y \quad r_z \quad 1]\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = [r \quad 1]\begin{bmatrix} \mathbf{S}_{3\times3} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} = [\boldsymbol{r}\mathbf{S}_{3\times3} \quad 1]$$

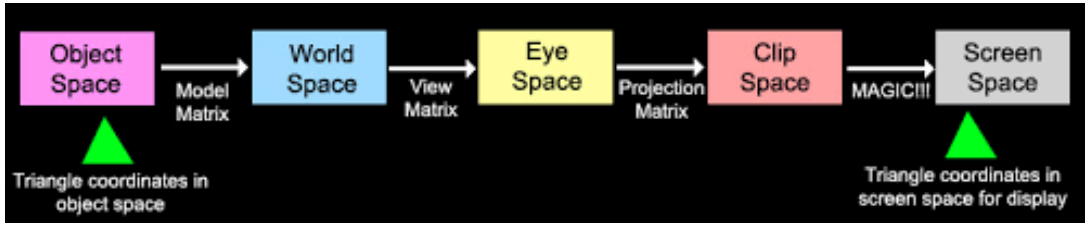
- Για την αντιστροφή της κλιμάκωσης απλά χρησιμοποιούμε τα αντίστροφα των συντελεστών κλιμάκωσης
- Αν όλες οι συντελεστές κλιμάκωσης είναι ίδιοι τότε έχουμε ομοιόμορφη κλιμάκωση

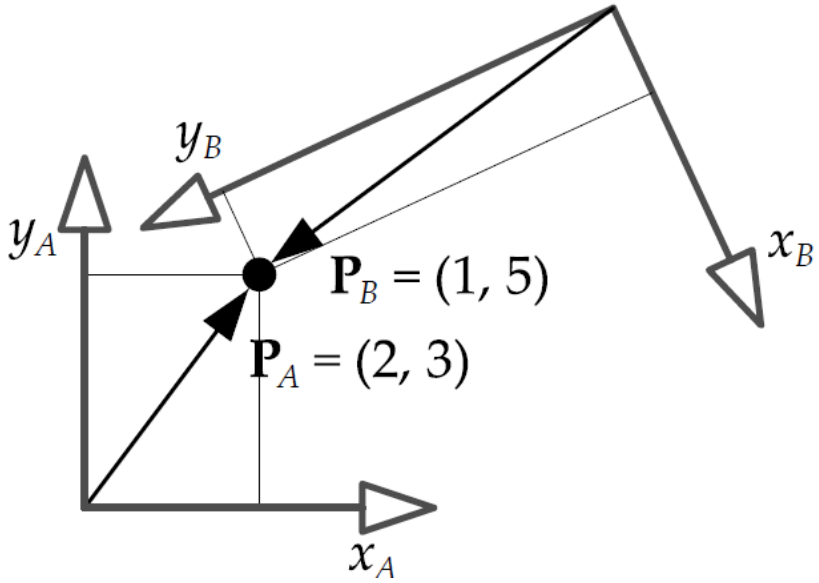
Βασική Ιδέα

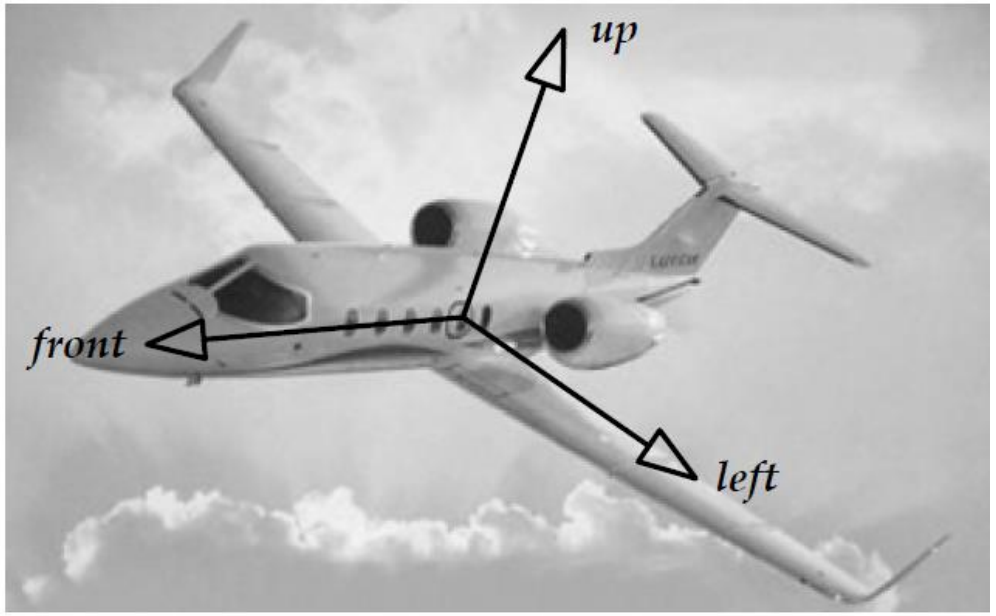
- Η γεωμετρία ενός πλέγματος (mesh) ορίζεται από το σύνολο κορυφών
- Η εφαρμογή μετασχηματισμού σε ένα πλέγμα σημαίνει μετασχηματισμό κάθε θέσης κορυφής
- Απλοί μετασχηματισμοί μπορούν να επιτευχθούν πολλαπλασιάζοντας έναν πίνακα επί τη θέση της κορυφής
  - Περιστροφή
  - Κλίμακα
  - Μεταφορά

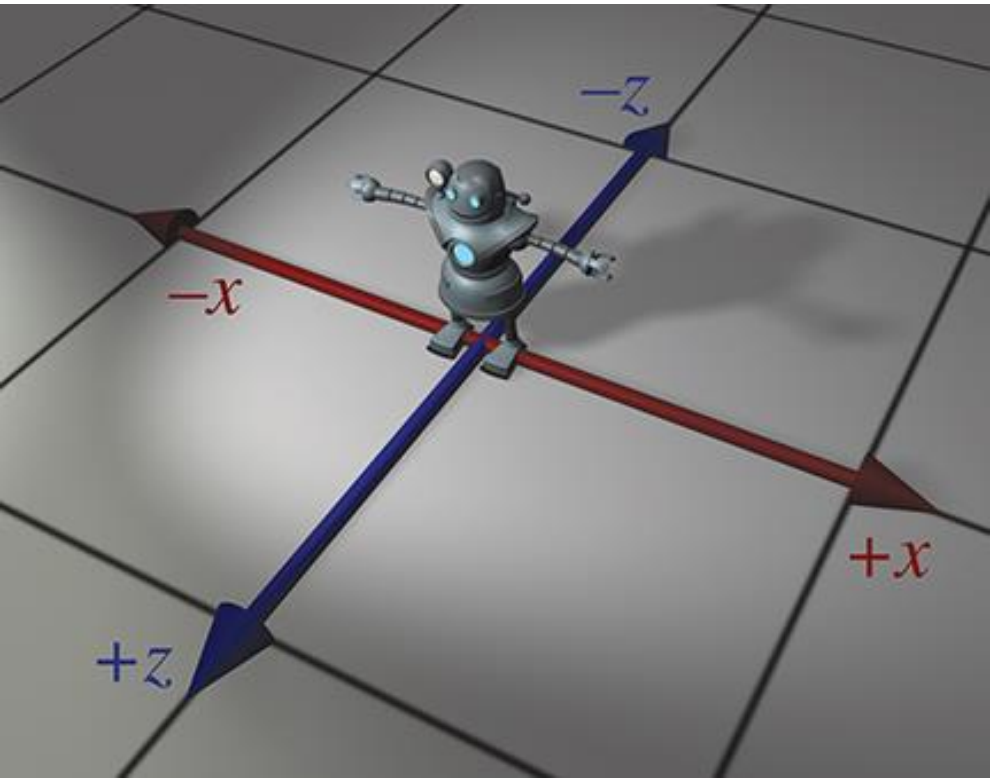
Πίσω στους Χώρους Συντεταγμένων

- Μπορούμε να σκεφτούμε ένα σημείο ως διάνυσμα σε σχέση με ένα δεδομένο σύνολο αξόνων
- Οι άξονες είναι απλώς για ένα πλαίσιο αναφοράς και αναφέρονται ως χώρος συντεταγμένων







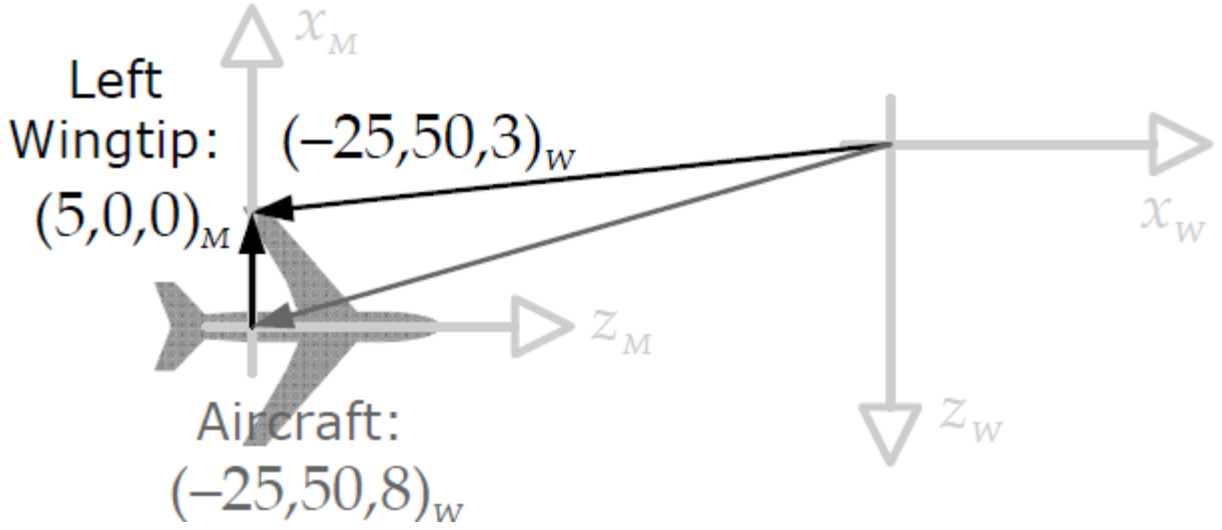


Χώρος Μοντέλου (Model Space)

- Όταν δημιουργείται ένα μοντέλο, οι κορυφές είναι σχετικές με ένα σύστημα συντεταγμένων που ονομάζεται χώρος μοντέλου
- Η αρχή του χώρου μοντέλου είναι συνήθως *στο κέντρο* του αντικειμένου
- Οι άξονες του χώρου του μοντέλου συνήθως ονομάζονται με κατευθύνσεις (μπροστά, αριστερά και επάνω)

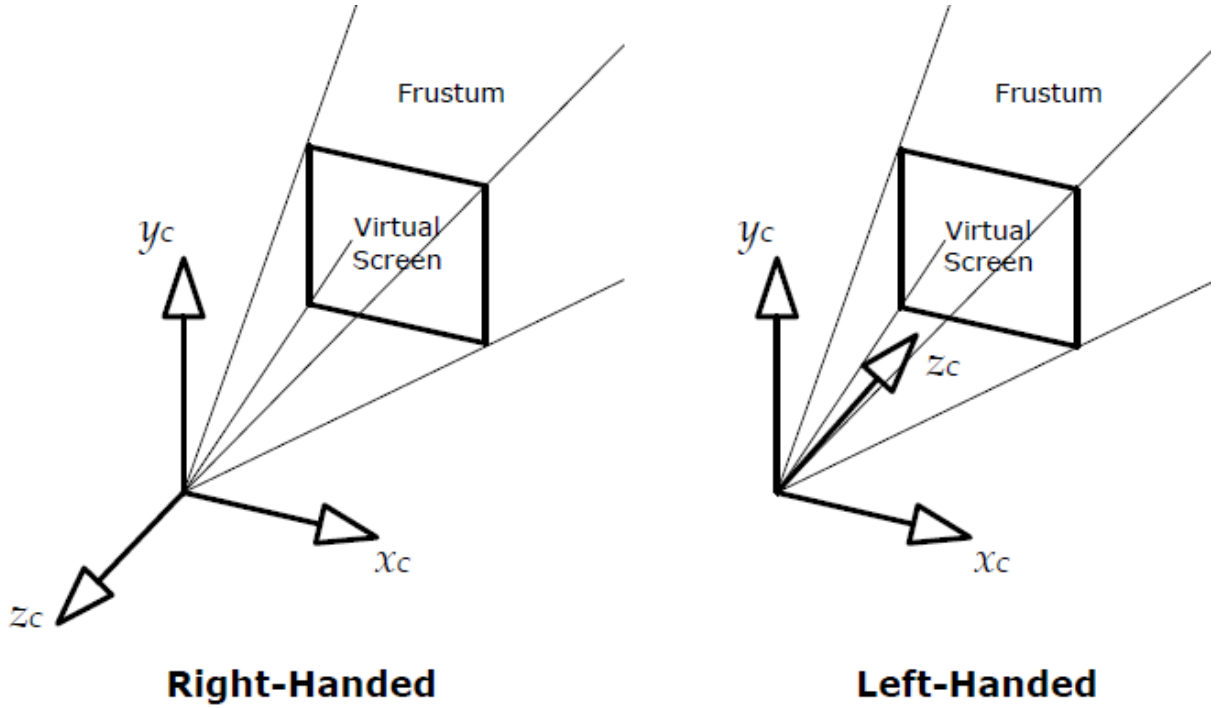
Χώρος Εικονικού Κόσμου (World Space)

- Ένα σταθερό σύστημα συντεταγμένων όπου ορίζονται τα αντικείμενα, οι προσανατολισμοί και οι κλίμακες
- Το κέντρο συνήθως τοποθετείται στο κέντρο της περιοχής παιχνιδιού
- Ο προσανατολισμός είναι αυθαίρετος αλλά συνήθως το y ή το z είναι επάνω



Χώρος Προβολής (View/Camera Space)

Ένας χώρος συντεταγμένων με αρχή τη θέση της κάμερας



Ιεραρχία Χώρων Συντεταγμένων

- Όλοι οι χώροι συντεταγμένων είναι σχετικοί
  - Πρέπει να τους καθορίσετε σε σχέση με άλλο χώρο
- Αυτό συνεπάγεται μια ιεραρχία χώρων
  - Ο χώρος του κόσμου βρίσκεται στη ρίζα
- Η κάμερα έχει μια θέση στο χώρο του κόσμου και άρα ο χώρος της της έχει το χώρο του κόσμου ως γονέα

Μητρώο Αλλαγής Βάσης

- Το μητρώο που μετασχηματίζει έναν χώρο σε αυτόν του γονιού του:

$$\mathbf{M}_{C\to P}$$

- Με αυτό το μητρώο κάθε θέση στο χώρο του παιδιού μπορεί να μετασχηματισθεί στη θέση στο χώρο του γονιού

$$\mathbf{P}_P = \mathbf{P}_C\mathbf{M}_{C\to P}$$

$$\mathbf{M}_{C\to P} = \begin{bmatrix} \mathbf{i}_C & 0 \\ \mathbf{j}_C & 0 \\ \mathbf{k}_C & 0 \\ \mathbf{t}_C & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{i}_{Cx} & \mathbf{i}_{Cy} & \mathbf{i}_{Cz} & 0 \\ \mathbf{j}_{Cx} & \mathbf{j}_{Cy} & \mathbf{j}_{Cz} & 0 \\ \mathbf{k}_{Cx} & \mathbf{k}_{Cy} & \mathbf{k}_{Cz} & 0 \\ \mathbf{t}_{Cx} & \mathbf{t}_{Cy} & \mathbf{t}_{Cz} & 1 \end{bmatrix}$$

$\mathbf{i}_C$ είναι το μοναδιαίο διάνυσμα στον x άξονα του χώρου του παιδιού εκφρασμένο σε συντεταγμένες του χώρου του γονιού

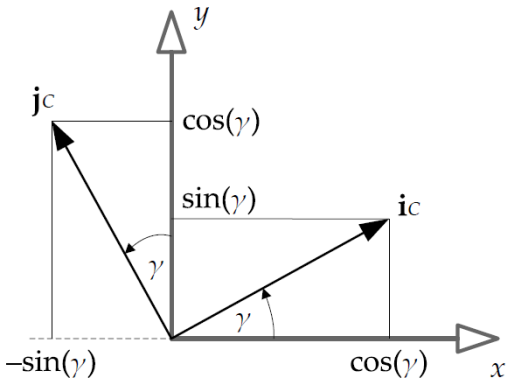
$\mathbf{j}_C$ είναι το μοναδιαίο διάνυσμα στον y άξονα του χώρου του παιδιού εκφρασμένο σε συντεταγμένες του χώρου του γονιού

$\mathbf{k}_C$ είναι το μοναδιαίο διάνυσμα στον z άξονα του χώρου του παιδιού εκφρασμένο σε συντεταγμένες του χώρου του γονιού

$\mathbf{t}_C$ είναι η μεταφορά του χώρου του παιδιού σε σχέση με το χώρο του γονιού

Παράδειγμα

- Έστω ένας χώρος (παιδί) που προκύπτει από περιστροφή κατά γ γύρω από τον άξονα z

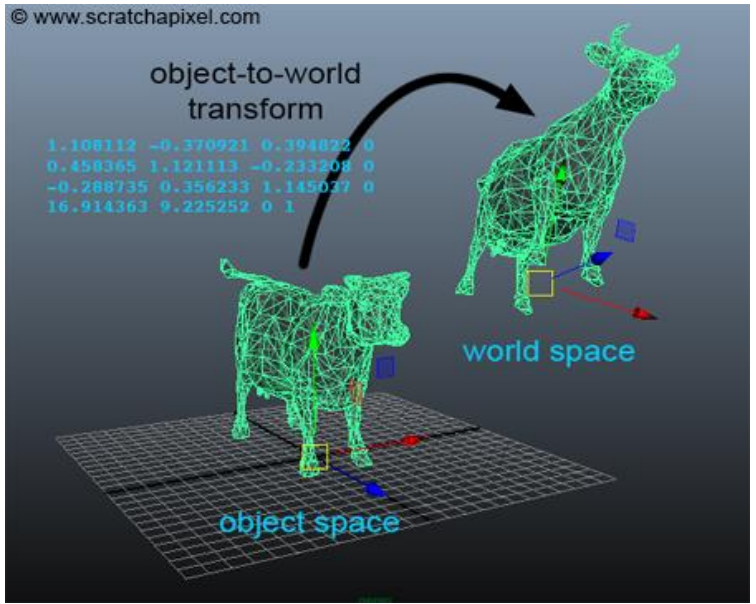


$$\mathbf{M_{C\to P}} = \begin{bmatrix} \cos\gamma & \sin\gamma & 0 & 0 \\ -\sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = rotate_z(r,\gamma)$$

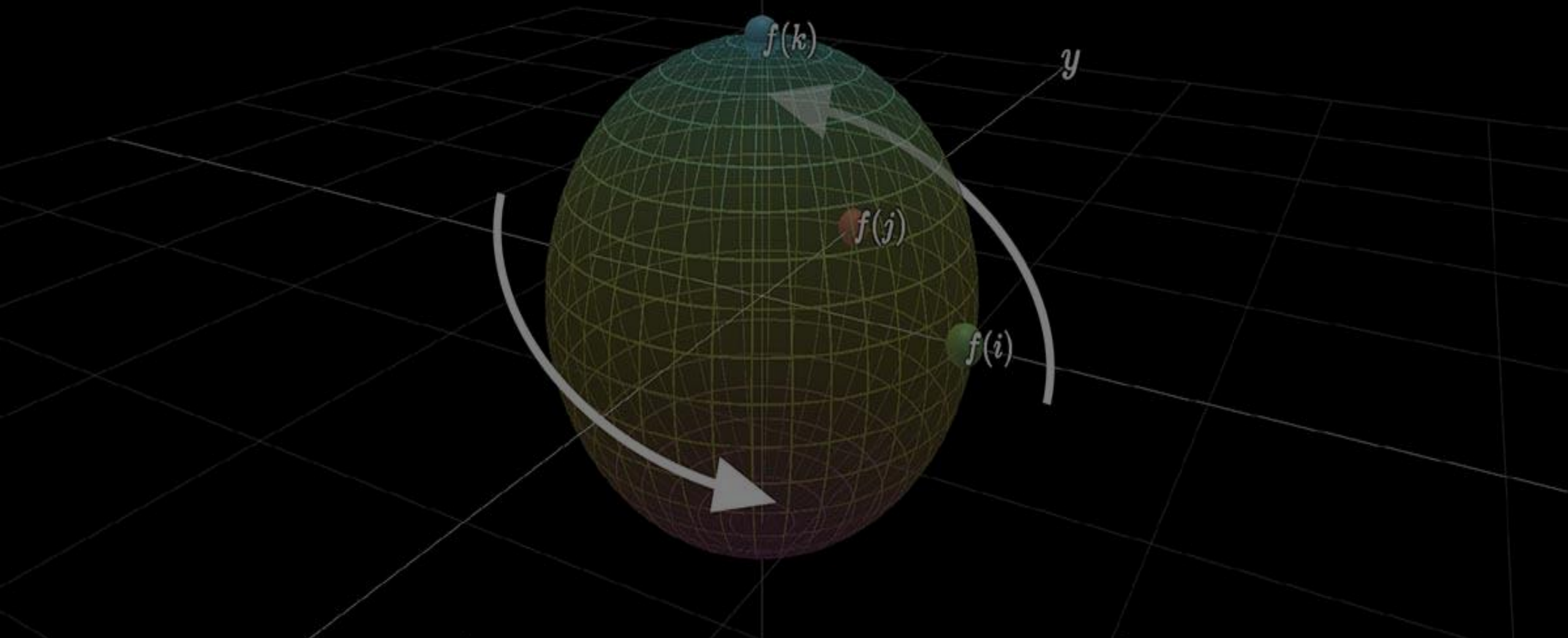
- Ισχύει: $i_{\mathbf{C}} = [\cos\gamma \quad \sin\gamma \quad 0]$

και $j_{\mathbf{C}} = [-\sin\gamma \quad \cos\gamma \quad 0]$

και $k_{\mathbf{C}} = [0 \quad 0 \quad 1]$



Quaternions and 3d rotation



Quaternions

Τόσο η Unity όσο και η Unreal χρησιμοποιούν Euler γωνίες αλλά εσωτερικά τις μετατρέπουν σε Quaternion για επεξεργασία.

- Στις 3 διαστάσεις χρησιμοποιούμε ένα 3×3 μητρώο για να αναπαραστήσουμε περιστροφές

- Προβλήματα:
  - Απαιτούνται 9 αριθμοί κινητής υποδιαστολής για κάθε μητρώο
  - Η περιστροφή ενός διανύσματος απαιτεί 9 πολλαπλασιασμούς και 6 προσθέσεις
  - Δύσκολο να κάνεις παρεμβολή

- Χρησιμοποιούμε τα quaternion $q = [q_x \quad q_y \quad q_z \quad q_w]$ μοναδιαίου μεγέθους $q_x^2 + q_y^2 + q_z^2 + q_w^2 = 1$. Με αυτόν τον τρόπο αναπαριστούμε πιο αποδοτικά 3d περιστροφές.

Quaternion

- Αποτελείται από δύο στοιχεία
  - Ένας μοναδιαίος άξονας περιστροφής $\hat{\boldsymbol{e}}$ που κλιμακώνεται με το ημίτονο της μισής γωνίας περιστροφής. Η φορά περιστροφής ακολουθεί τον κανόνα του δεξιού χεριού
  - Το συνημίτονο της μισής γωνίας

$$q = [\boldsymbol{q}_V \; q_s] = \left[\boldsymbol{\alpha}\sin\frac{\theta}{2} \; \cos\frac{\theta}{2}\right]$$

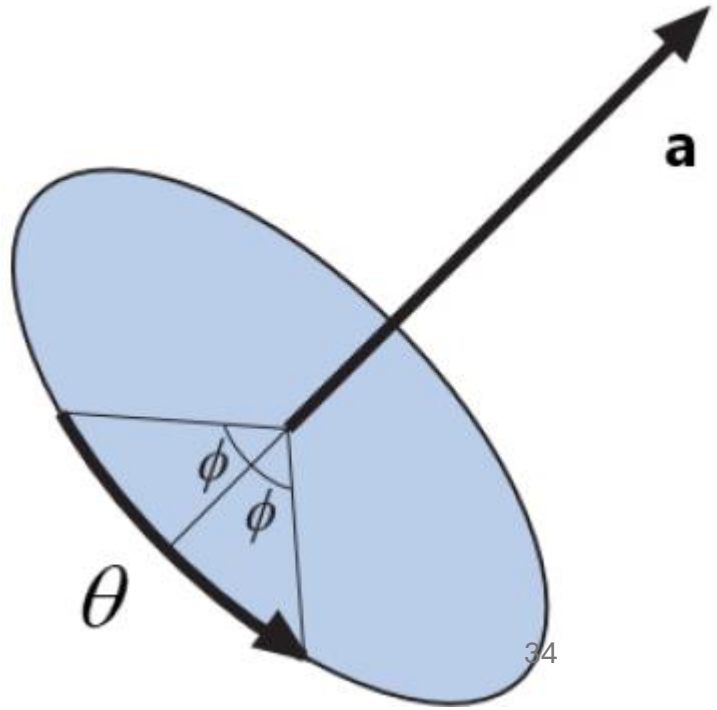
$q = [q_x \quad q_y \quad q_z \quad q_w]$, όπου:

$$q_x = q_{\mathrm{V}_x} = e_x \sin\frac{\theta}{2}$$

$$q_y = q_{\mathrm{V}_y} = e_y \sin\frac{\theta}{2}$$

$$q_z = q_{\mathrm{V}_z} = e_z \sin\frac{\theta}{2}$$

$$q_w = q_s = \cos\frac{\theta}{2}$$



Πράξεις

- Η πρόσθεση δύο quaternions δεν αντιστοιχεί στο άθροισμα δύο γωνιών
- Ο πολλαπλασιασμός είναι η πιο σημαντική πράξη – αντιπροσωπεύει μια περιστροφή ακολουθούμενη από μια άλλη

$$\mathrm{pq} = [(p_S \boldsymbol{q}_V + q_S \boldsymbol{p}_V + \mathbf{p}_V \times \boldsymbol{q}_V) \quad (p_S q_S - \mathbf{p}_V \cdot \boldsymbol{q}_V)]$$

Συζυγείς και Αντίστροφοι

- Το αντίστροφο q^{-1} ενός quaternion q, έχει την εξής ιδιότητα:

$$\mathrm{q}\mathrm{q}^{-1} = 0\mathbf{i} + 0\mathbf{j} + 0\mathbf{k} + 1$$

- Συζυγής:

$$\mathrm{q}^{*} = [-\mathbf{q}_v \quad q_s]$$

- Ο αντίστροφος είναι:

$$\mathrm{q}^{-1} = \frac{\mathrm{q}^{*}}{|\mathrm{q}|^2}$$

Είναι Απλό ☺

- Θυμηθείτε ότι τα quaternions έχουν μοναδιαίο μήκος

$$q^{-1} = \mathrm{q}^* = [-\mathbf{q}_V \quad q_S]$$

- Άλλες ιδιότητες

$$(\mathrm{pq})^* = \mathrm{q}^*\mathrm{p}^*$$

$$(pq)^{-1} = \mathrm{q}^{-1}\mathrm{p}^{-1}$$

Περιστροφή με Quaternions

Για να περιστρέψουμε ένα διάνυσμα $\mathbf{v}$ κατά ένα quaternion q

1. Αρχικά γράφουμε το διάνυσμα σε μορφή quaternion

$$\mathrm{v} = [\mathbf{v} \quad 0] = [v_x \quad v_y \quad v_z \quad 0]$$

2. Έπειτα πολλαπλασιάζουμε:

$$\mathrm{v}' = rotate(\mathrm{q}, \mathbf{v}) = \mathrm{q}\mathrm{v}\mathrm{q}^{-1}$$

ή

$$\mathrm{v}' = rotate(\mathrm{q}, \mathbf{v}) = \mathrm{q}\mathrm{v}\mathrm{q}^{*}$$

3. Έπειτα παίρνουμε το διάνυσμα από το $\mathrm{v}' = [\mathbf{v}' \quad 0]$

Συνένωση

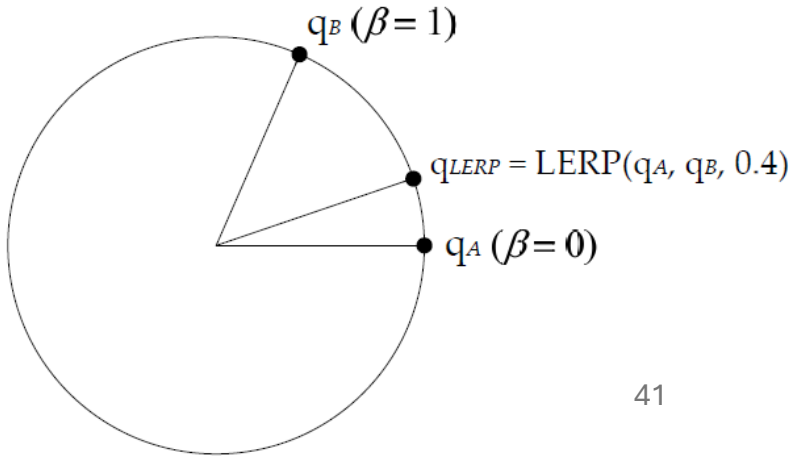
- Πολλαπλές περιστροφές με quaternion

$$\begin{aligned} \mathrm{q}_{net} &= \mathrm{q}_3\mathrm{q}_2\mathrm{q}_1 \\ \mathrm{v}' &= \mathrm{q}_3\mathrm{q}_2\mathrm{q}_1\mathrm{v}\mathrm{q}_1^{-1}\mathrm{q}_2^{-1}\mathrm{q}_3^{-1} \\ &= \mathrm{q}_{net}v\mathrm{q}_{net}^{-1} \end{aligned}$$

Περιστροφική Παρεμβολή (LERP)

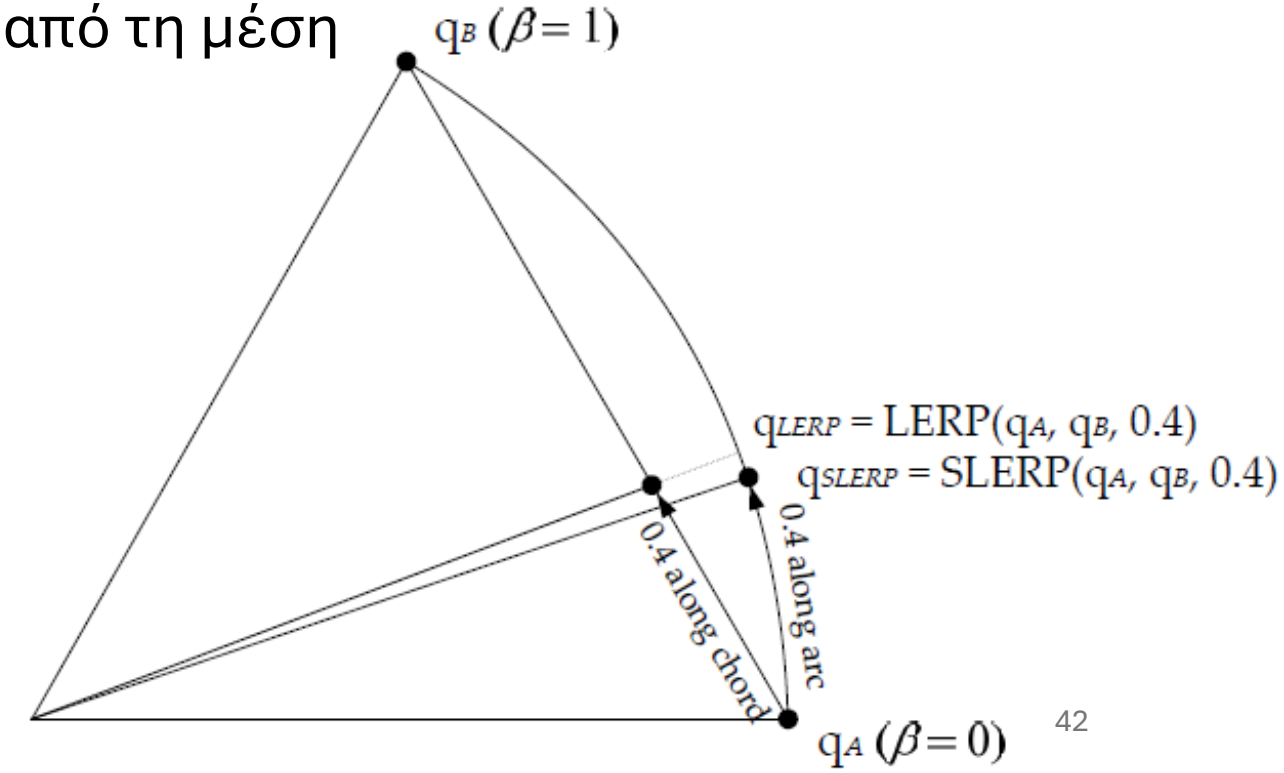
- Δοθέντων δύο quaternions q_A και q_B μπορούμε να βρούμε μία ενδιάμεση περιστροφή κατά ποσοστό β μεταξύ τους ως:

$$q_{LERP} = LERP(q_A, q_B, \beta) = \frac{(1-\beta)q_A + \beta q_B}{|(1-\beta)q_A + \beta q_B|} = normalize\left(\begin{bmatrix} (1-\beta)q_{A_x} + \beta q_{B_x} \\ (1-\beta)q_{A_y} + \beta q_{B_y} \\ (1-\beta)q_{A_z} + \beta q_{B_z} \\ (1-\beta)q_{A_w} + \beta q_{B_w} \end{bmatrix}^T\right)$$



Προβλήματα με LERP

- Αφού το LERP παρεμβάλει σε μία χορδή αντί της πραγματικής επιφάνειας της 4d υπερ-σφαίρας, η γωνιακή ταχύτητα δεν είναι σταθερή
  - Η αρχή και το τέλος είναι πιο αργά από τη μέση



Σφαιρικό LERP

- Η σφαιρική γραμμική παρεμβολή (SLERP) διορθώνει το πρόβλημα του LERP
- Το SLERP είναι όπως το LERP, αλλά το βάρος μεταξύ των δύο quaternion αλλάζει ανάλογα με τη μεταξύ τους γωνία

$$SLERP(\mathrm{p}, \mathrm{q}, \beta) = w_p \mathrm{p} + w_q \mathrm{q}$$

$$w_p = \frac{\sin(1-\beta)\theta}{\sin\theta}$$

$$w_q = \frac{\sin\beta\theta}{\sin\theta}$$

θ;

- Για να βρούμε τη γωνία θ κάνουμε:

$$\cos(\theta) = \mathrm{p} \cdot \mathrm{q} = p_x q_x + p_y q_y + p_z q_z + p_w q_w$$

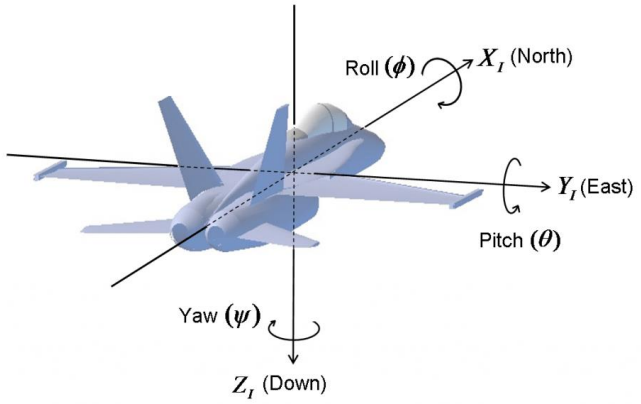
$$\theta = cos^{-1}(\mathrm{p} \cdot \mathrm{q})$$

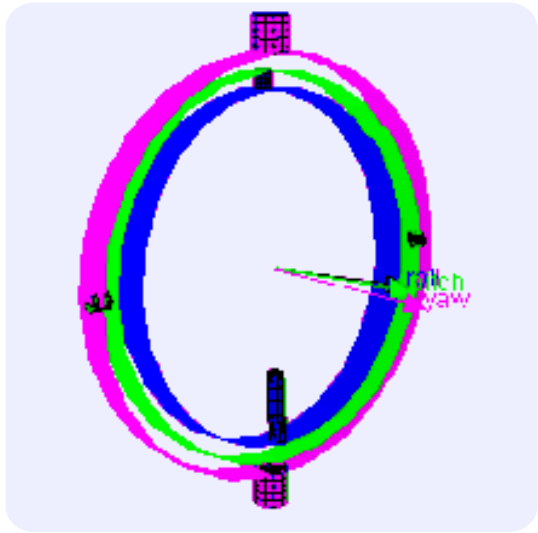
Να χρησιμοποιήσουμε το SLERP;

- Εξαρτάται
- Εάν μπορείτε να το κάνετε να είναι αρκετά γρήγορο, τότε OK
- Διαφορετικά το LERP είναι εντάξει και πιθανότατα ο παίκτης δεν μπορεί να δει διαφορά

Σύγκριση Αναπαραστάσεων Περιστροφής

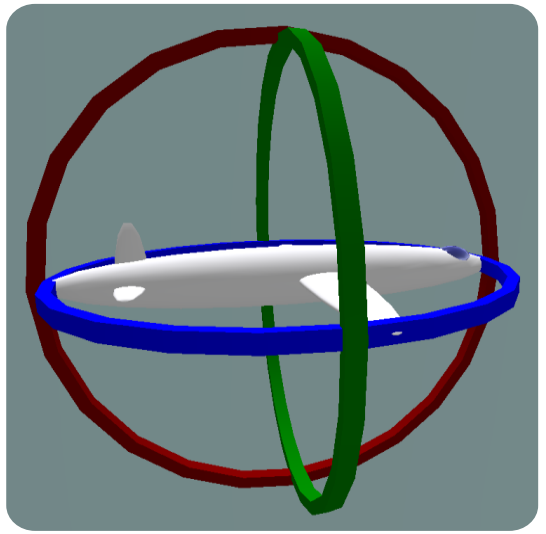
Γωνίες Euler

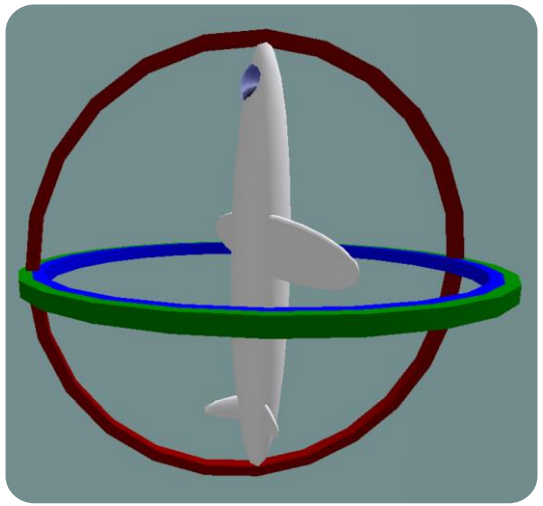




Yaw, Pitch, Roll $\mathrm{p} = [\theta_{\Upsilon}, \theta_{\mathrm{P}}, \theta_R]$

- Απλή και μικρού μεγέθους
- Εύκολη στην οπτικοποίηση
- Εύκολη παρεμβολή εκτός και αν πρόκειται για αυθαίρετο άξονα περιστροφής
- Επιρρεπές στο gimbal lock
- Η σειρά των περιστροφών παίζει ρόλο





$$\mathbf{PYR} \neq \mathbf{YPR} \neq \mathbf{RYP}$$

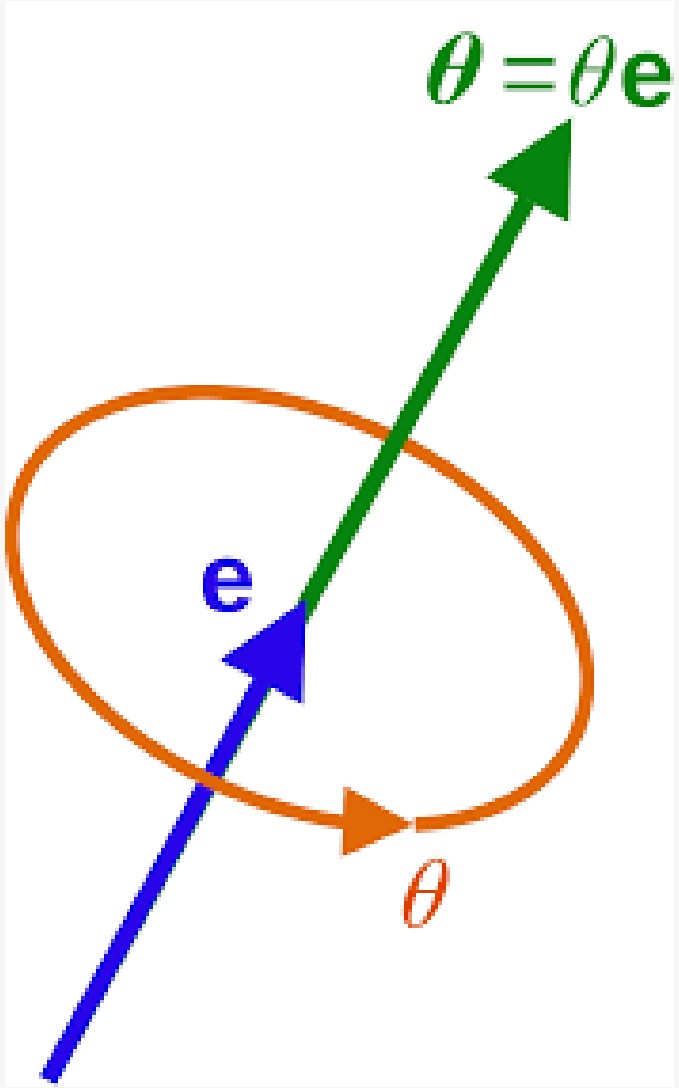
3 × 3 Μητρώα

- Υποστηρίζονται από CPUs και GPUs
- Δεν παρουσιάζουν το gimbal lock
- Απαιτούν αρκετό υπολογισμό και μνήμη
- Δεν είναι εύκολο να τους διαβάσετε
- Δεν είναι εύκολη η παρεμβολή

```
0 references
public static Matrix4x4 operator* (Matrix4x4 a, Matrix4x4 b) {
    return new Matrix4x4(
        a.M00 * b.M00 + a.M10 * b.M01 + a.M20 * b.M02 + a.M30 * b.M03,
        a.M00 * b.M10 + a.M10 * b.M11 + a.M20 * b.M12 + a.M30 * b.M13,
        a.M00 * b.M20 + a.M10 * b.M21 + a.M20 * b.M22 + a.M30 * b.M23,
        a.M00 * b.M30 + a.M10 * b.M31 + a.M20 * b.M32 + a.M30 * b.M33,
        a.M01 * b.M00 + a.M11 * b.M01 + a.M21 * b.M02 + a.M31 * b.M03,
        a.M01 * b.M10 + a.M11 * b.M11 + a.M21 * b.M12 + a.M31 * b.M13,
        a.M01 * b.M20 + a.M11 * b.M21 + a.M21 * b.M22 + a.M31 * b.M23,
        a.M01 * b.M30 + a.M11 * b.M31 + a.M21 * b.M32 + a.M31 * b.M33,
        a.M02 * b.M00 + a.M12 * b.M01 + a.M22 * b.M02 + a.M32 * b.M03,
        a.M02 * b.M10 + a.M12 * b.M11 + a.M22 * b.M12 + a.M32 * b.M13,
        a.M02 * b.M20 + a.M12 * b.M21 + a.M22 * b.M22 + a.M32 * b.M23,
        a.M02 * b.M30 + a.M12 * b.M31 + a.M22 * b.M32 + a.M32 * b.M33,
        a.M03 * b.M00 + a.M13 * b.M01 + a.M23 * b.M02 + a.M33 * b.M03,
        a.M03 * b.M10 + a.M13 * b.M11 + a.M23 * b.M12 + a.M33 * b.M13,
        a.M03 * b.M20 + a.M13 * b.M21 + a.M23 * b.M22 + a.M32 * b.M23,
        a.M03 * b.M30 + a.M13 * b.M31 + a.M23 * b.M32 + a.M33 * b.M33
        );
}
```

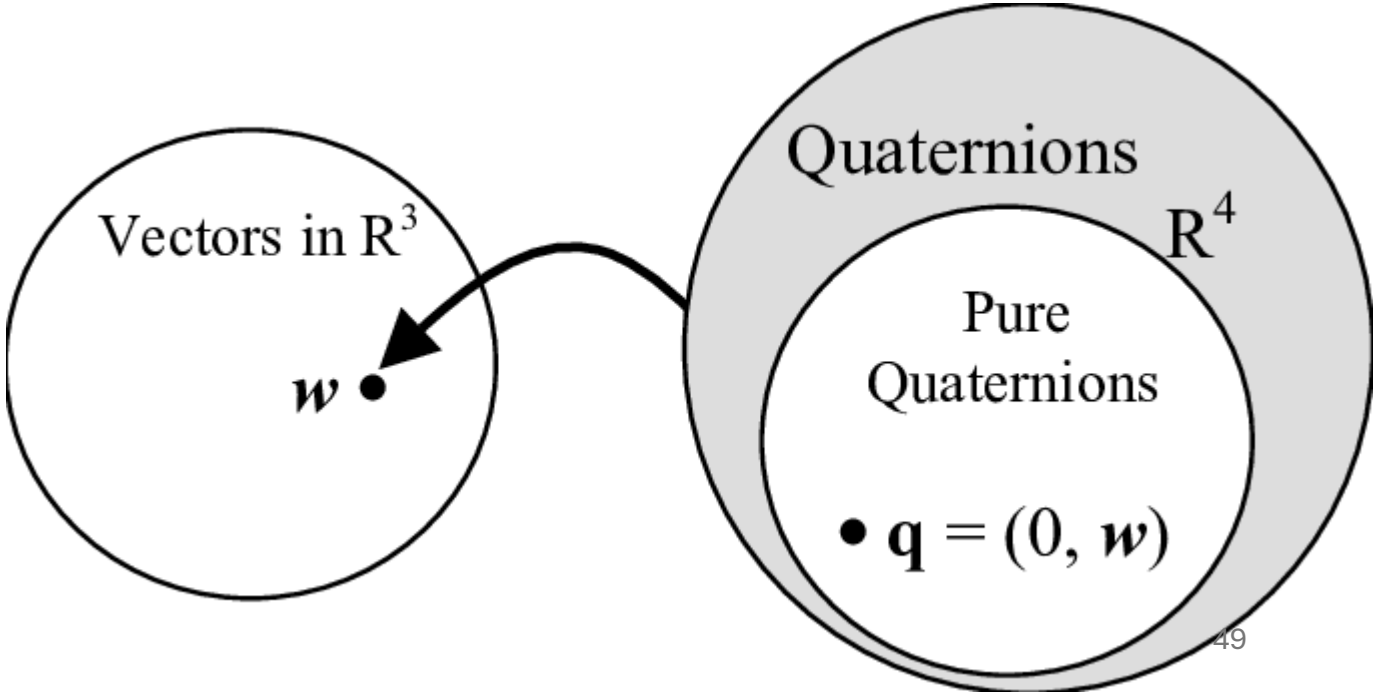
Άξονας/Γωνία

- Αναπαράσταση αντίστοιχη με τα quaternions $[\mathbf{a} \quad \theta]$
- Εύκολη στην κατανόηση
- Συμπαγής (4 αριθμοί κινητής υποδιαστολής)
- Δύσκολη παρεμβολή
- Δεν μπορούν εύκολα να εφαρμοσθούν σε σημεία και διανύσματα (πρέπει να μετατραπεί σε άλλη αναπαράσταση πρώτα)



Quaternions

- Συμπαγή (όπως αναπαράσταση άξονα/γωνίας)
- Εύκολα εφαρμόζεται σε σημεία και διανύσματα
- Εύκολη παρεμβολή
- Δύσκολο να διαβαστεί

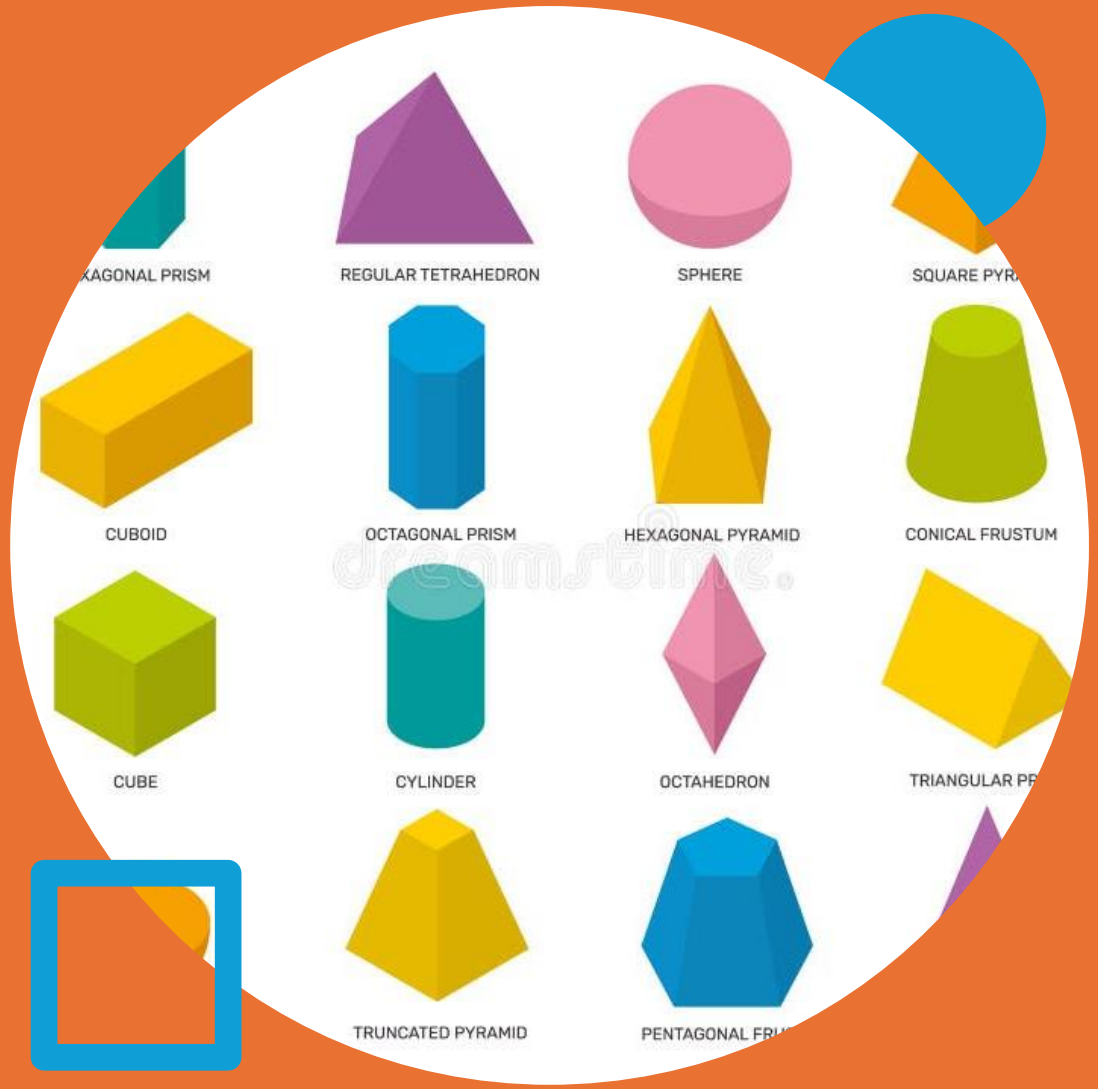


SQT (SRT) Μετασχηματισμοί

- Τα Quaternion μπορούν να αναπαραστήσουν περιστροφές
- Μπορούμε να τα συνδυάσουμε με μεταφορά και κλιμάκωση
- Αρκετά καλό σε πολλές περιπτώσεις (χρησιμοποιείται κυρίως σε animation)
  - Εύκολο στην κατανόηση
  - Εύκολη παρεμβολή (LERP σε κλιμάκωση και μεταφορά, LERP ή SLERP στην περιστροφή)
  - Συμπαγής αναπαράσταση (10 αριθμοί κινητής υποδιαστολής ή 8 ανάλογα με το αν έχουμε ομοιόμορφη ή ανομοιόμορφη κλιμάκωση – με ένα μητρώο 4×3 χρειαζόμαστε 12 συνολικά)

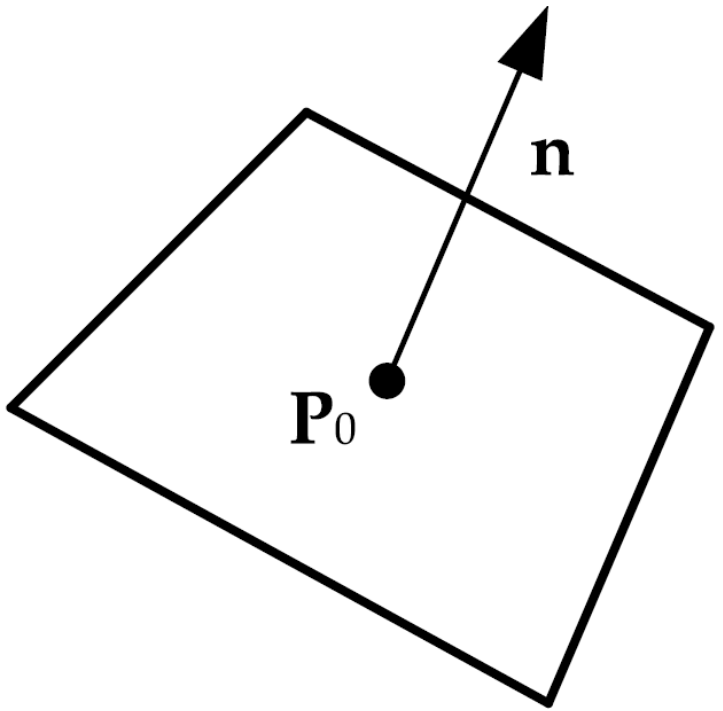
$$SQT = [\mathbf{s} \quad \mathrm{q} \quad \mathbf{t}]$$

Μερικά Χρήσιμα Γεωμετρικά Αντικείμενα



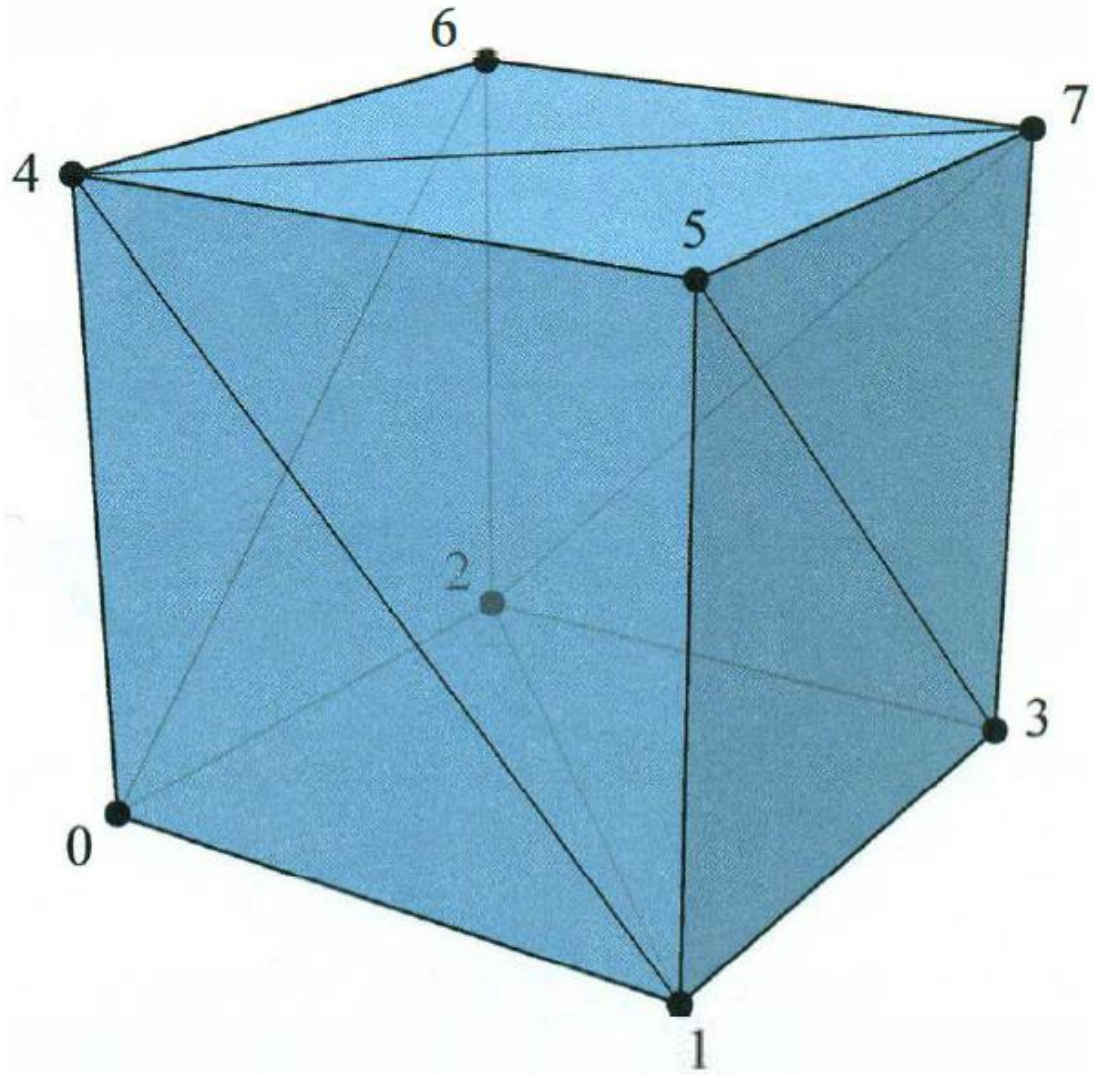
Σημείο – Κάθετος

- Ένα επίπεδο μπορεί να αναπαρασταθεί από
  - Ένα σημείο $\mathbf{P}_0$
  - Ένα μοναδιαίο διάνυσμα **n** που είναι κάθετο στο επίπεδο



Πλέγματα Τριγώνων

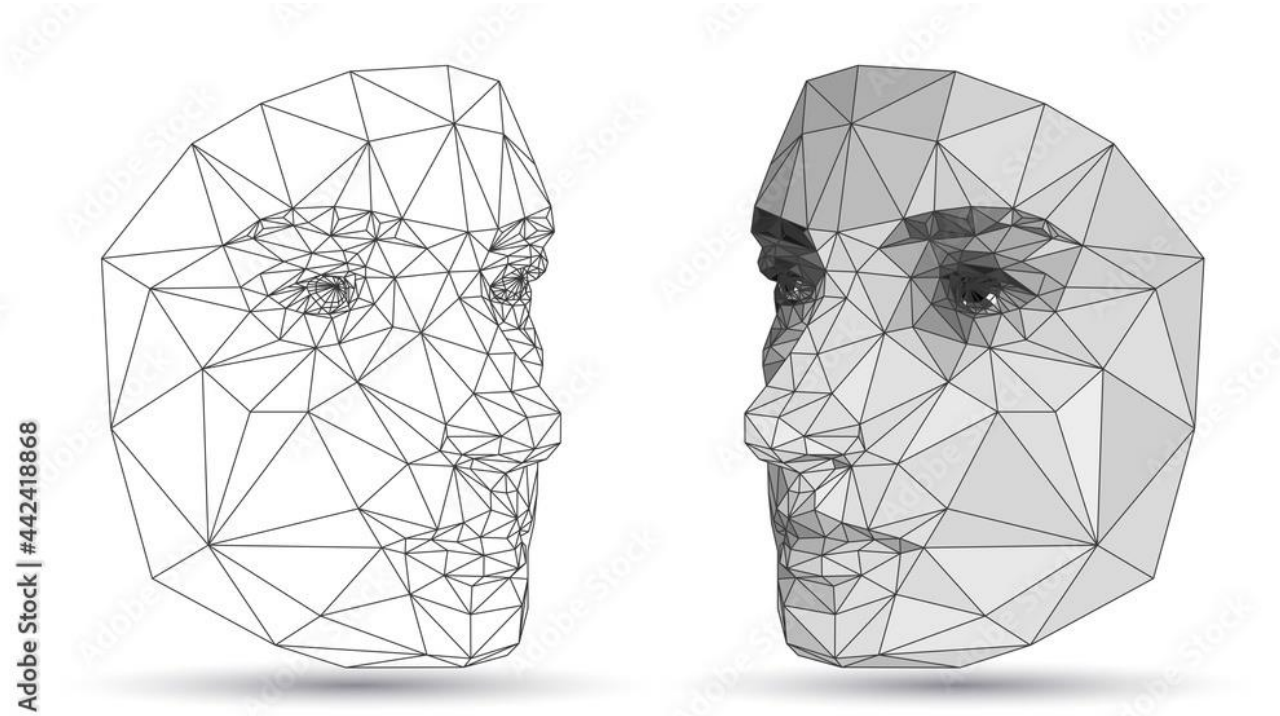
- Ένας κύβος με 8 κορυφές και 12 τρίγωνα
- Τριγωνοποίηση: διαδικασία διαχωρισμού πολυγώνου σε τρίγωνα
- Κατεύθυνση διαπέρασης: όταν βλέπουμε την εξωτερική πλευρά του τριγώνου με τη σειρά που αναφέρουμε τις κορυφές του.
  - π.χ., φορά αντίθετη του ρολογιού: (5,4,1) ή (1,3,5)



Εφαρμογή

- Τα σύνθετα τρισδιάστατα μοντέλα κατασκευάζονται από τρίγωνα

- Κατά τον προσδιορισμό της φωτεινότητας ενός τριγώνου, πρέπει να γνωρίζουμε τη γωνία μεταξύ του φέροντος επιπέδου και της δέσμης φωτός.

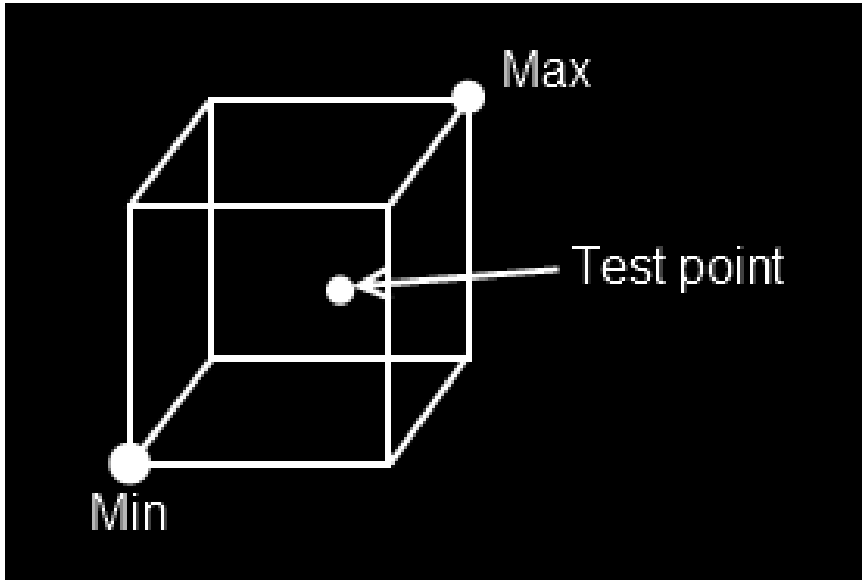


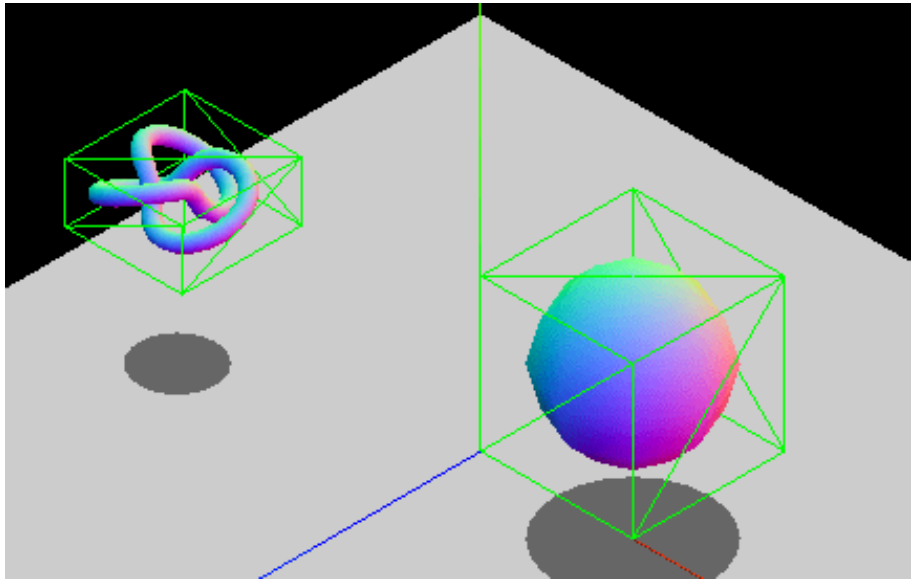
AABB

- Axis Aligned Bounding Box (Ορθοκανονικό Πλαίσιο Οριοθέτησης)
- Πολύ χρήσιμο σε ανίχνευση σύγκρουσης
- Αναπαρίσταται με 6 τιμές

$$[x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max}]$$

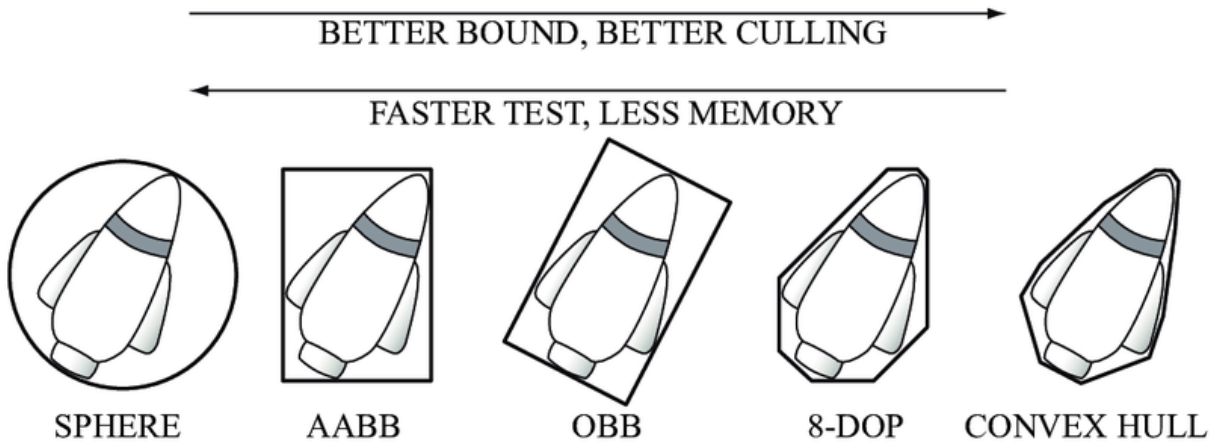
- Ή δύο σημεία $\mathbf{P}_{min}$ και $\mathbf{P}_{max}$
- Εύκολος ο έλεγχος αν ένα σημείο είναι μέσα στα όρια

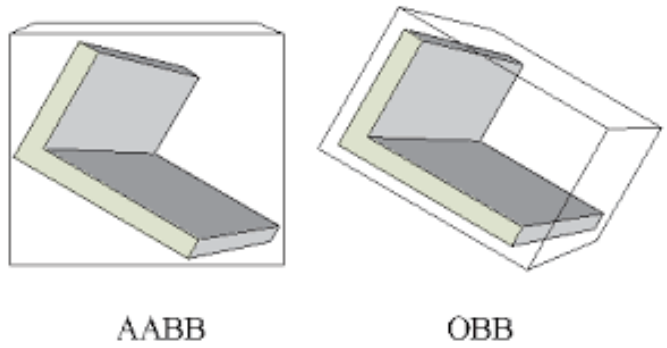




oBB

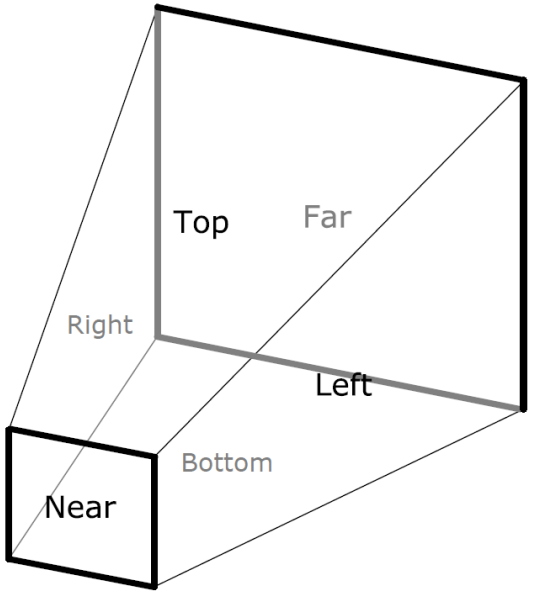
- Oriented Bounding Box (προσανατολισμένο πλαίσιο οριοθέτησης)
- Διαφέρει από το AABB μιας και προσανατολίζεται με τον τοπικό χώρο των αντικειμένων και όχι με τον χώρο του κόσμου
- Ο έλεγχος τομής προϋποθέτει την μετατροπή του σημείου στο OBB σύστημα συντεταγμένων

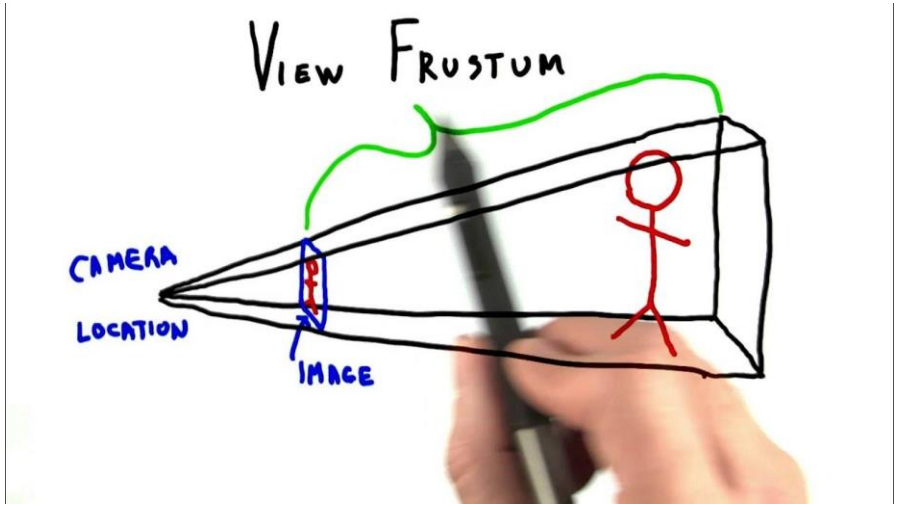




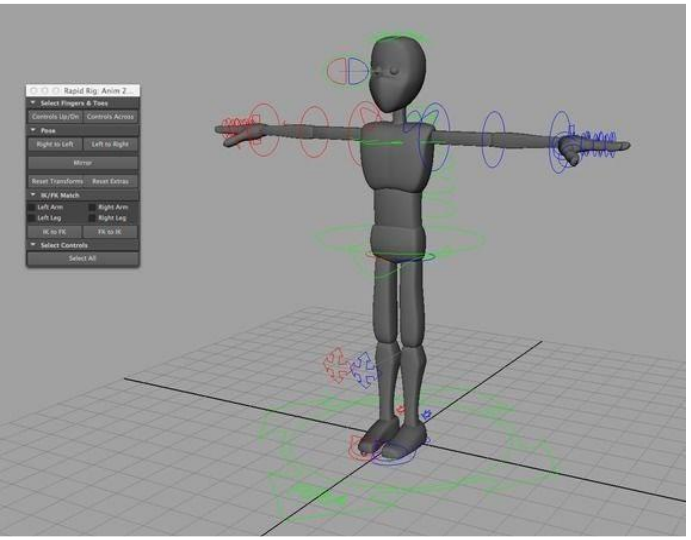
Frusta

- Ένα σύνολο 6 επιπέδων που δημιουργούν μία κομμένη πυραμίδα
- Συνήθως αναπαρίσταται από έναν πίνακα 6 επιπέδων με μορφή σημείο-κάθετος.





Rigging

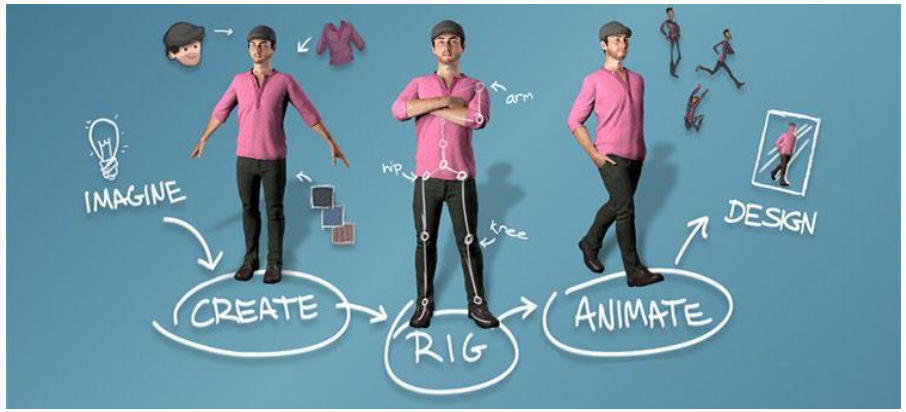


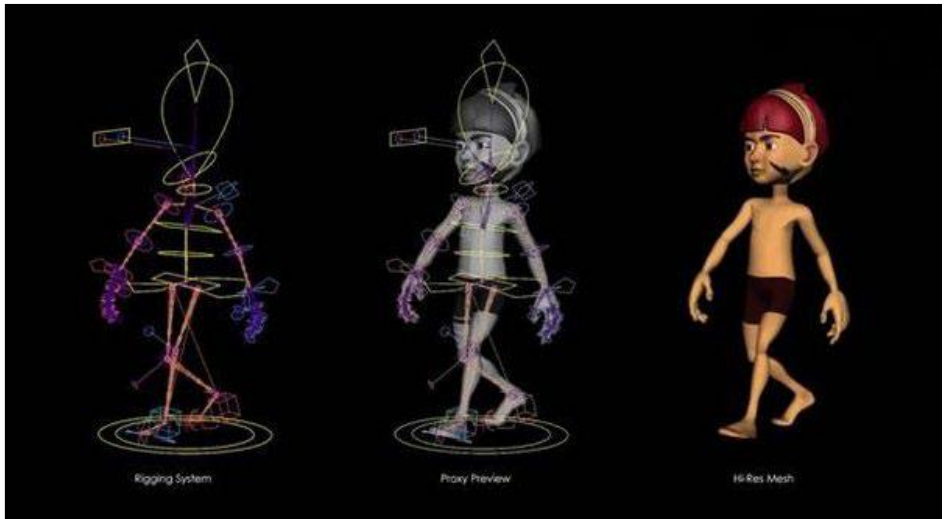
Το 3d rigging είναι η διαδικασία δημιουργίας ενός σκελετού για ένα τρισδιάστατο μοντέλο ώστε να μπορεί να κινηθεί.

Κάθε άρθρωση έχει έναν μετασχηματισμό που σχετίζεται με αυτήν

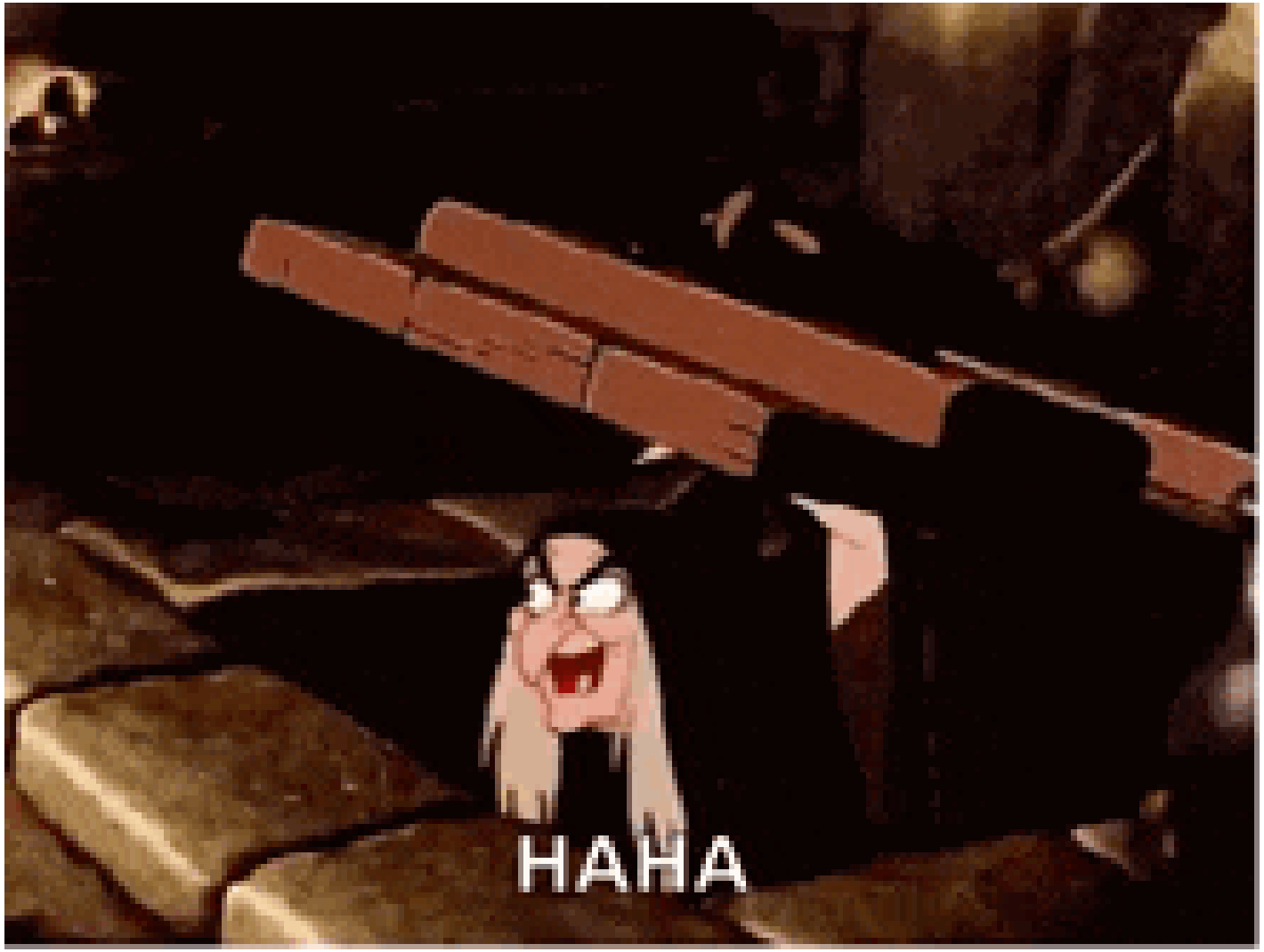
Ο σκελετός δημιουργεί ένα δέντρο μετασχηματισμών

π.χ., τα δάχτυλα μετασχηματίζονται από ένα γινόμενο μητρώων που ξεκινά από το στήθος, μετά στον ώμο, μετά στο μπράτσο, μετά στον αγκώνα, μετά στον καρπό, κ.λπ.





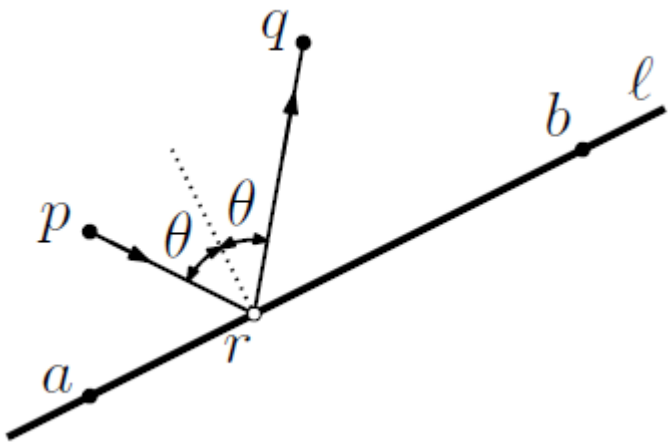
Ασκήσεις

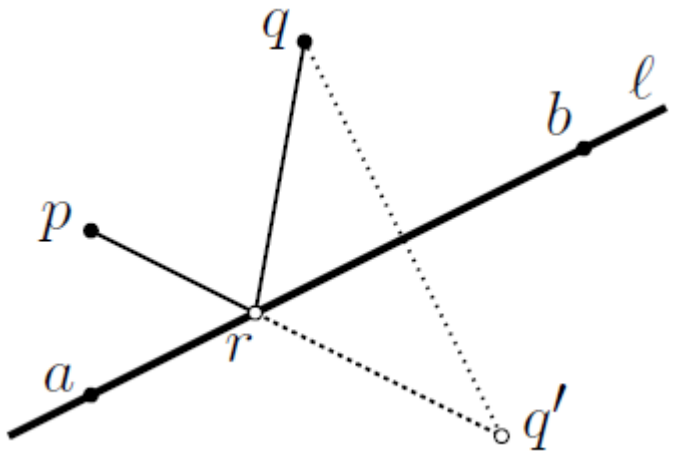


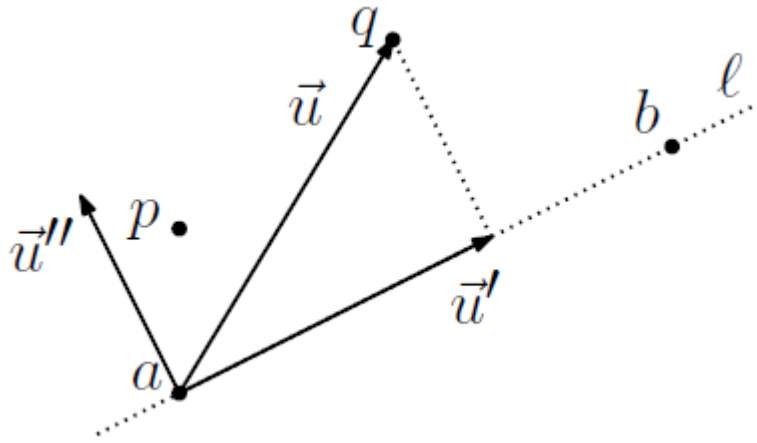
2-d Platformer

```
Vector3 Reflect (Vector3 a, Vector3 b, Vector3 p, Vector3 q) {
  Vector3 v = b - a; // vector from a to b
  Vector3 u = q - a; // vector from a to q
  Vector3 u1 = (Vector3.Dot(u,v)/Vector3.Dot(v,v)) * v;  // proj u on v
  Vector3 u2 = u - u1; // orthogonal complement
  Vector3 qq = q - 2 * u2; // reflection of q across line ab
                           // compute intersection scalar
  float t = ((a.x - p.x) * (qq.y - p.y) - (a.y - p.y) * (qq.x - p.x)) /
            ((b.y - a.y) * (qq.x - p.x) - (b.x - a.x) * (qq.y - p.y));
  if (t < 0 || t > 1) return NoHit;
  else return (1 - t) * a + t * b;
}
```

Εργάζεστε σε ένα δύο διαστάσεων παιχνίδι. Ο ήρωας του παιχνιδιού σας έχει ένα όπλο που εκτοξεύει μια ακτίνα λέιζερ. Αυτή η δέσμη μπορεί να αναπηδήσει σε ειδικά αντικείμενα (καθρέφτες) στο περιβάλλον σας. Για newbies (στο χαμηλότερο επίπεδο δυσκολίας), παρέχετε μια λειτουργία που θα γίνεται αυτόματα η στόχευση του όπλου για να χτυπήσει ένα δεδομένο στόχο αναπηδώντας σε έναν από αυτούς τους καθρέφτες. Πιο τυπικά, έστω a και b δύο σημεία στο επίπεδο. Έστω ℓ η ευθεία που διέρχεται από αυτά τα σημεία. Έστω p η θέση του λέιζερ όπλου και έστω q ο στόχος. Και τα δύο σημεία βρίσκονται στην ίδια πλευρά της ℓ. Θέλουμε να υπολογίσουμε το σημείο r στην ℓ έτσι ώστε μια βολή από το p προς το r θα αναπηδήσει και θα χτυπήσει το q. Αφού υποθέτουμε αντανάκλαση, η γωνία πρόσπτωσης είναι ίση με τη γωνία ανάκλασης.



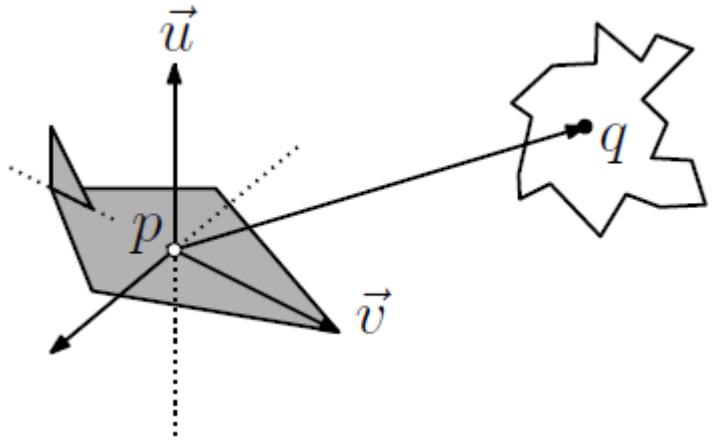


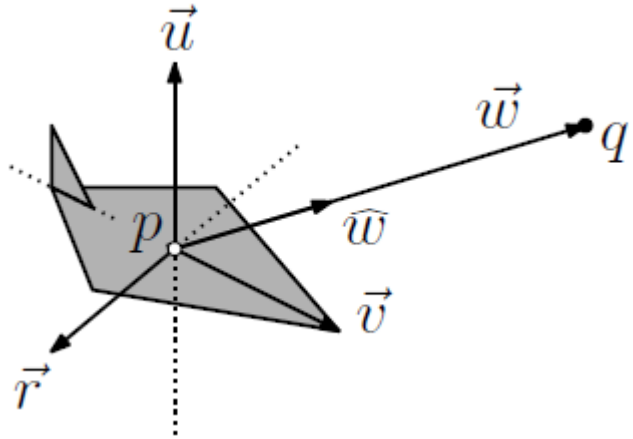


Μανούβρες Αποφυγής

```
void Evade (Vector3 p, Vector3 v,  Vector3 u, Vector3 q) {
  Vector3 w = q - p;  // vector from pilot to obstacle
  float l = w.magnitude; // distance to obstacle
  Vector3 ww = w.normalized;  // directional vector to obstacle
  if (Vector3.Dot (ww, u) >= 0) // obstacle is above?
    PitchDown ();
  else
    PitchUp ();
  Vector3 r = Vector3.Cross(v, u); // vector to pilot's right
  if (Vector3.Dot (ww, r) >= 0) // obstacle is to the right
    YawToLeft ();
  else
    YawToRight ();
}
```

Στο παιχνίδι που φτιάχνετε, προσομοιώνετε την τεχνητή νοημοσύνη για μερικά εξωγήινα διαστημόπλοια. Κάθε διαστημόπλοιο σχετίζεται με την τρέχουσα θέση του, ένα σημείο p και δύο διανύσματα μοναδιαίου μήκους. Το πρώτο διάνυσμα $\vec{v}$ υποδεικνύει την κατεύθυνση στην οποία κινείται το διαστημόπλοιο. Το δεύτερο διάνυσμα $\vec{u}$ είναι ορθογώνιο στο $\vec{v}$ και υποδεικνύει την κατεύθυνση «πάνω» σε σχέση με τον πιλότο του διαστημόπλοιου. Υπάρχουν διάφορα εμπόδια που πρέπει να αποφευχθούν (αστεροειδείς) και το σύστημα AI πρέπει να δίνει εντολές αλλαγής πορείας για να αποφευχθούν τα εμπόδια. Με δεδομένο ένα εμπόδιο σε κάποιο σημείο q, θέλουμε να καθορίσουμε αν πρέπει να στρίψει (yaw) προς τα αριστερά ή προς τα δεξιά και αν πρέπει να στρίψει (pitch) πάνω ή κάτω για να αποφευχθεί η σύγκρουση. (Δεν μας ενδιαφέρουν οι γωνίες αλλά μόνο η κατεύθυνση.)

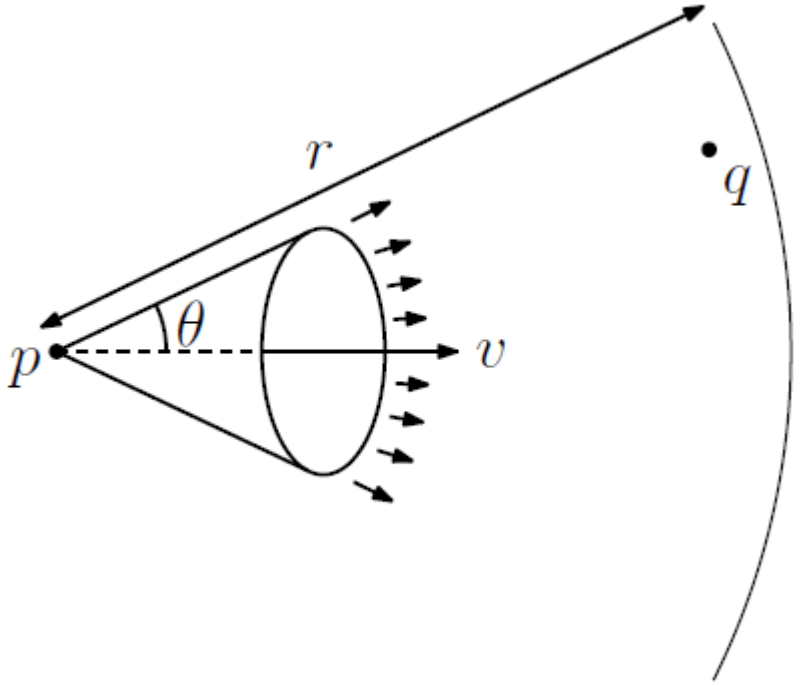




Προσομοίωση Καραμπίνας

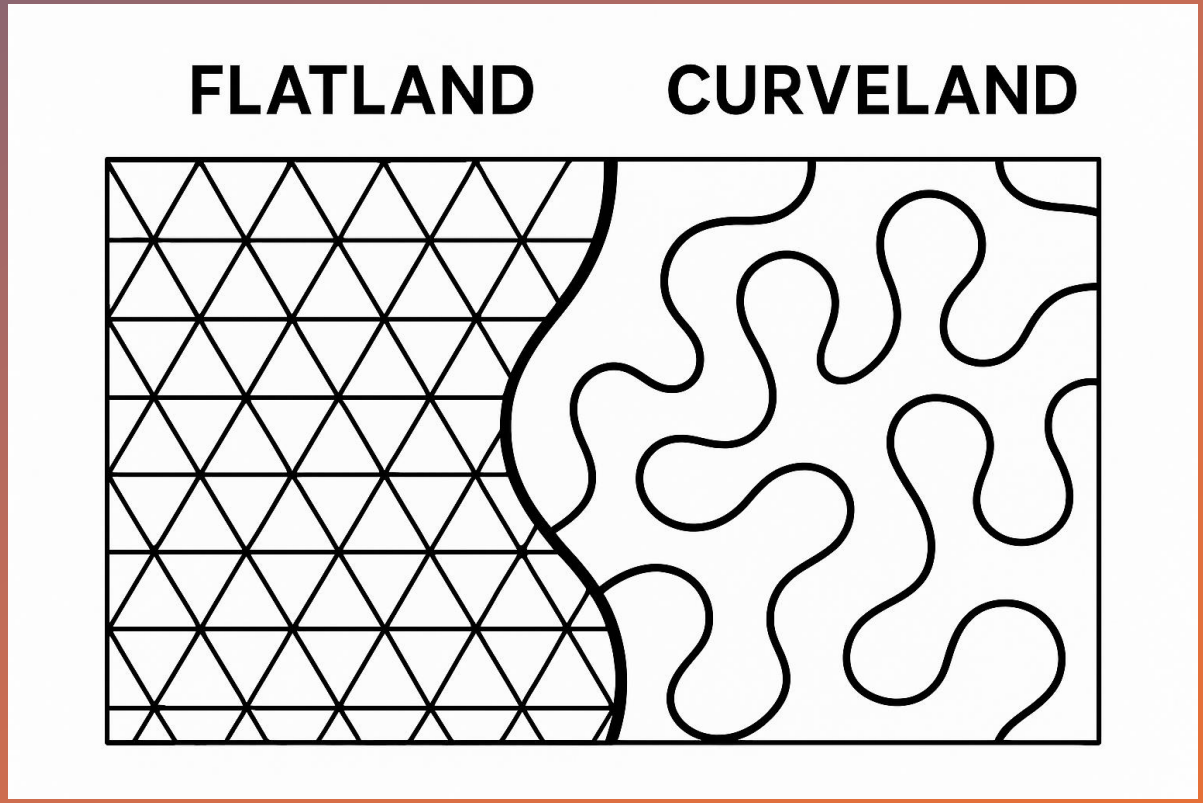
```
bool HitMe (Vector3 p, Vector3 v, float theta, float r, Vector3 q) {
  Vector3 u = q - p;  // vector from muzzle end (p) to target (q)
  float l = u.magnitude; // distance to target
  Vector3 uu = u.normalized;  // directional vector to target
  float c1 = Vector3.Dot (uu, v); // cosine of angle between u and v
  float c2 = Mathf.Cos (theta * Mathf.PI / 180); // cosine of hit cone
  return (c1 >= c2) && (l <= r); // target within cone and distance
}
```

Σας ζητήθηκε να αναπτύξετε ένα νέο όπλο που συμπεριφέρεται σαν κυνηγετικό όπλο. Έχει μεγάλη γωνία κρούσης (2θ), αλλά είναι αποτελεσματικό μόνο σε σχετικά μικρές αποστάσεις. Ας υποθέσουμε ότι το άκρο του ρύγχους του όπλου βρίσκεται σε ένα σημείο p (σε τρισδιάστατο χώρο) και σημαδεύει προς μία κατεύθυνση που δίνεται από ένα διάνυσμα $\vec{v}$ (το οποίο μπορείτε να υποθέσετε ότι έχει μοναδιαίο μήκος). Το όπλο εκτοξεύει σφαιρίδια σε γωνία θ γύρω από το $\vec{v}$, αλλά οι σφαίρες είναι αποτελεσματικές μέχρι απόσταση r από το άκρο του ρύγχους του όπλου. (Ας υποθέσουμε ότι το θ δίνεται σε μοίρες και είναι αυστηρά μικρότερο από 90^0.) Δεδομένου ενός σημείου q, γράψτε μια σύντομη διαδικασία για να προσδιορίσετε εάν το σημείο q θα χτυπηθεί όταν το όπλο εκπυρσοκροτήσει.

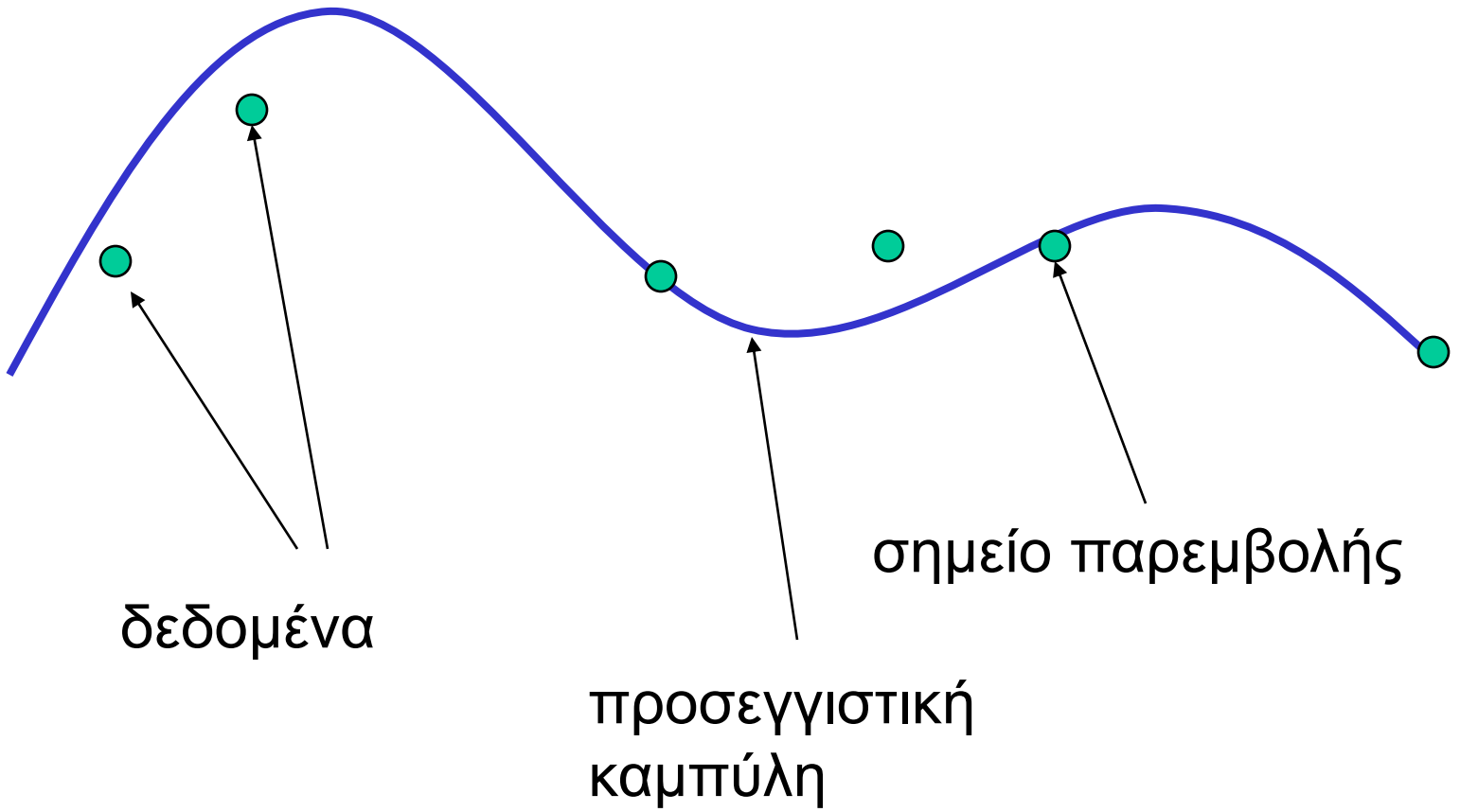


Μία Μικρή Παράκαμψη

Πηγαίνοντας από την Επιπεδοχώρα στην Καμπυλοχώρα



Μοντελοποίηση με Καμπύλες



Πότε μία Αναπαράσταση είναι καλή;

Υπάρχουν πολλοί τρόποι αναπαράστασης καμπυλών και επιφανειών.

Θέλουμε αναπαραστάσεις που να είναι

- Ευσταθείς
- Ομαλές
- Εύκολες να αποτιμηθούν
- Παρεμβολή ή προσέγγιση στα δεδομένα;
- Παράγωγους

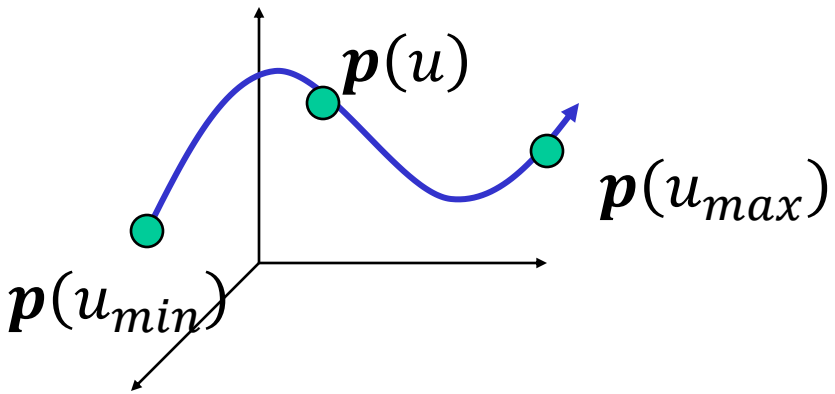
Παραμετρικές Καμπύλες

- Ξεχωριστές εξισώσεις για κάθε μεταβλητή

$$x = x(u)$$
$$y = y(u)$$
$$z = z(u)$$

$$\boldsymbol{p}(u) = [x(u), y(u), z(u)]^T$$

- Για $u_{max} \geq u \geq u_{min}$ ορίζεται η καμπύλη στις 2 ή 3 διαστάσεις

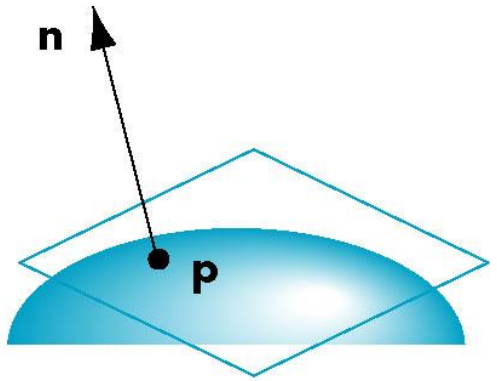


Κάθετα Διανύσματα

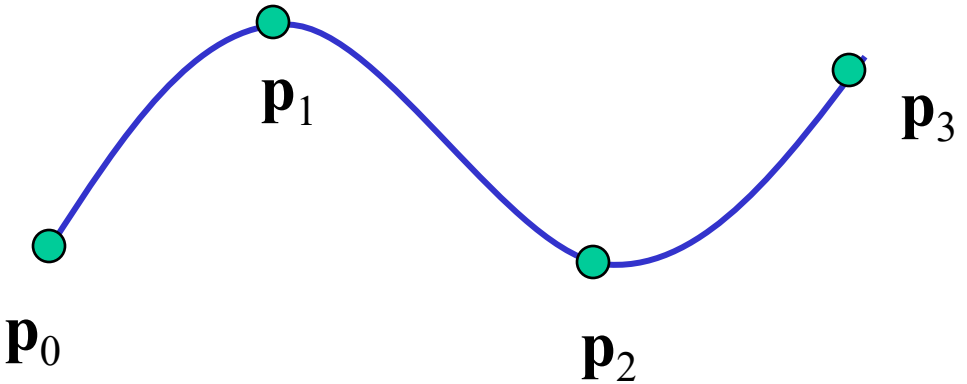
Μπορούμε να παραγωγίσουμε ως προς u και v για να βρούμε το κάθετο διάνυσμα σε κάθε σημείο $\boldsymbol{p}$

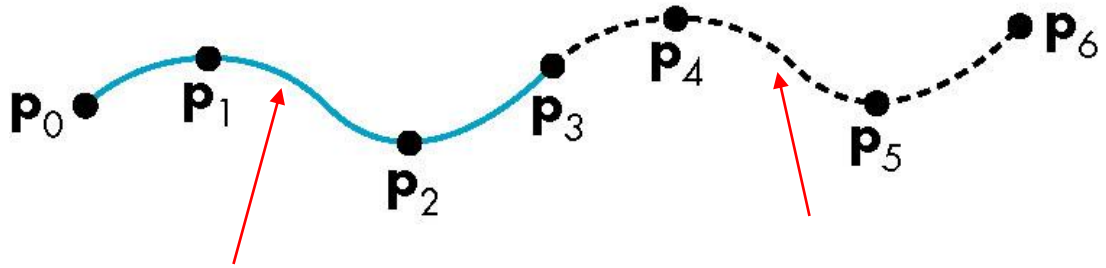
$$\frac{\partial \mathbf{p}(u,v)}{\partial u} = \begin{bmatrix} \partial \mathrm{x}(u,v)/\partial u \\ \partial \mathrm{y}(u,v)/\partial u \\ \partial \mathrm{z}(u,v)/\partial u \end{bmatrix} \qquad \frac{\partial \mathbf{p}(u,v)}{\partial v} = \begin{bmatrix} \partial \mathrm{x}(u,v)/\partial v \\ \partial \mathrm{y}(u,v)/\partial v \\ \partial \mathrm{z}(u,v)/\partial v \end{bmatrix}$$

$$\mathbf{n} = \frac{\partial \mathbf{p}(u,v)}{\partial u} \times \frac{\partial \mathbf{p}(u,v)}{\partial v}$$



Καμπύλη Παρεμβολής (κυβικά πολυώνυμα)



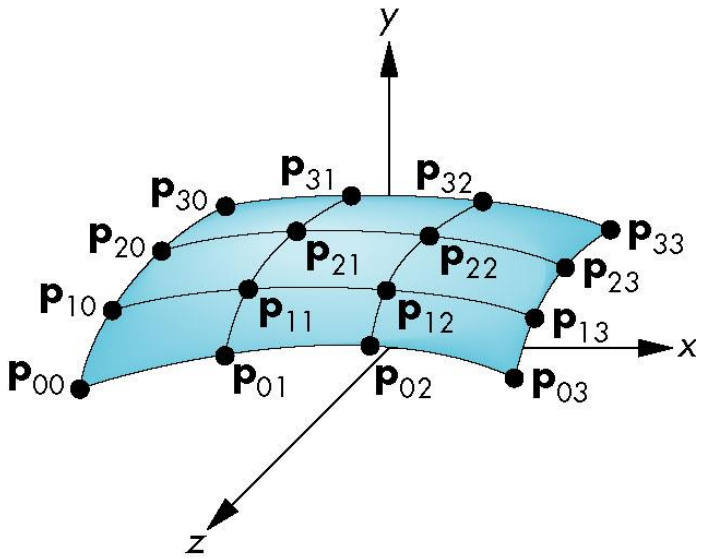


$\mathbf{p} = [p_0\ p_1\ p_2\ p_3]^T$

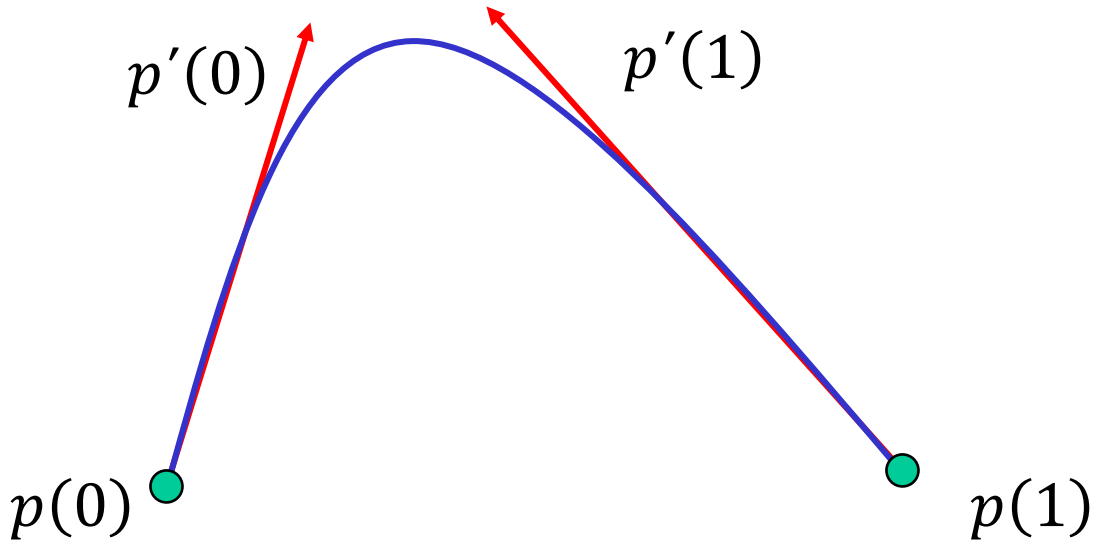
$\mathbf{p} = [p_3\ p_4\ p_5\ p_6]^T$

Παρεμβολή σε Τμήμα Επιφάνειας

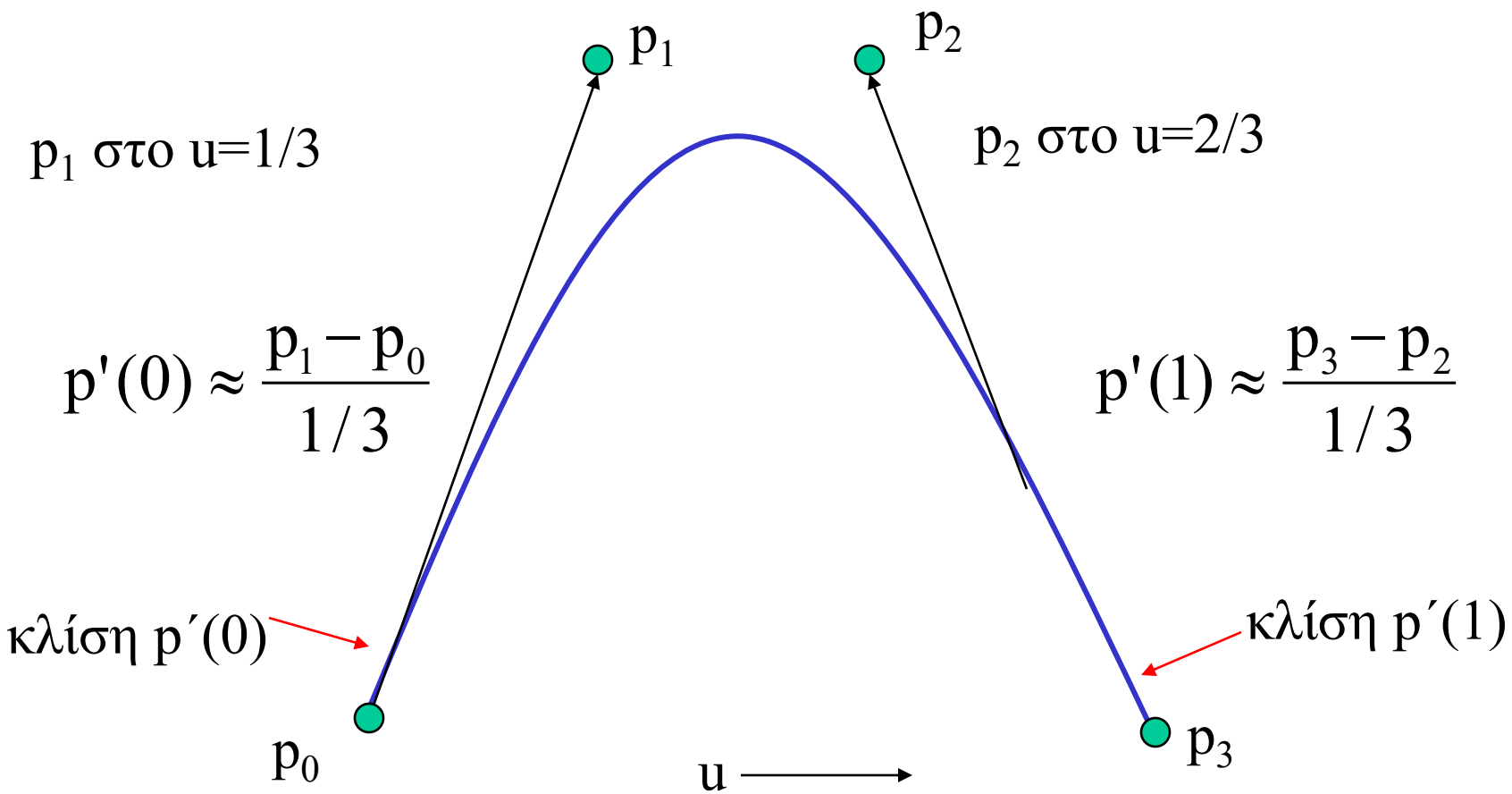
$$p(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} c_{ij}u^i v^j$$



Ερμιτιανή Μορφή

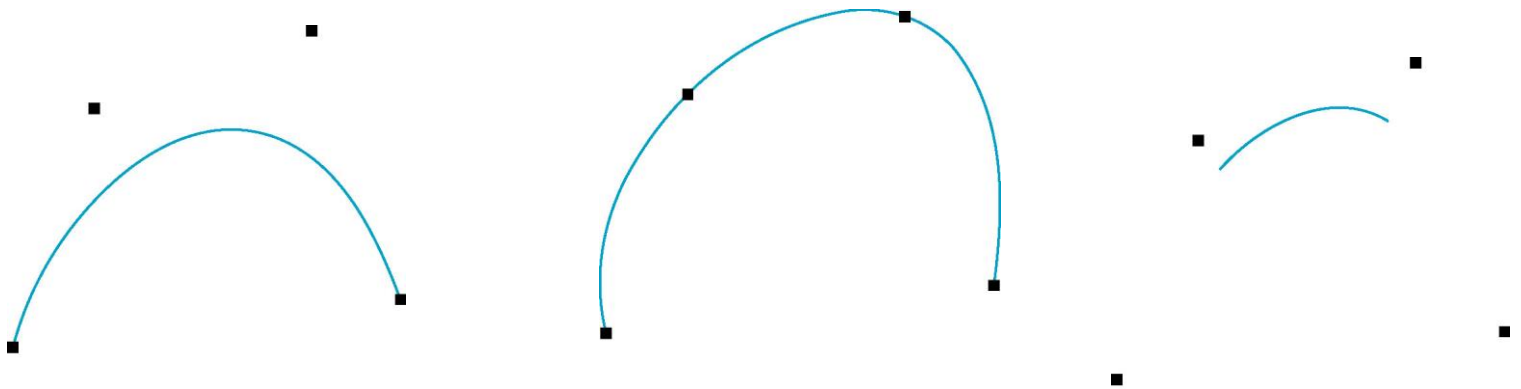


Bezier: Προσέγγιση Παραγώγων



B-Splines

- Basis Splines: χρήση του $\boldsymbol{p} = [p_{i-2}\ p_{i-1}\ p_i\ p_{i+1}]^T$ για τον ορισμό της καμπύλης μεταξύ p_{i-1} και p_i.



Bezier

Παρεμβολή

B-Spline