Γραφικά Υπολογιστών

Ιόνιο Πανεπιστήμιο
Τμήμα Πληροφορικής

Οι διαφάνειες βασίζονται σε videos διαλέξεων του:
Cem Yuksel
Assoc. Prof,
School of Computing, University of Utah)

Στέργιος Παλαμάς, Επίκουρος Καθηγητής

Μάθημα 15:

Textures

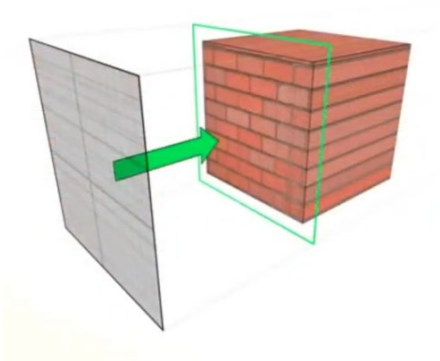
Textures

Ένας πολύ «στενός» ορισμός για τα Textures είναι ότι **είναι εικόνες που εφαρμόζουμε πάνω σε αντικείμενα.**

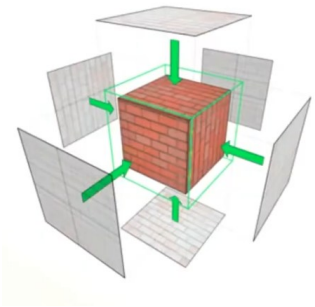
Textures

Το ερώτημα που γεννιέται είναι λοιπόν «Με ποιόν τρόπο εφαρμόζω την εικόνα πάνω στο αντικείμενο;»

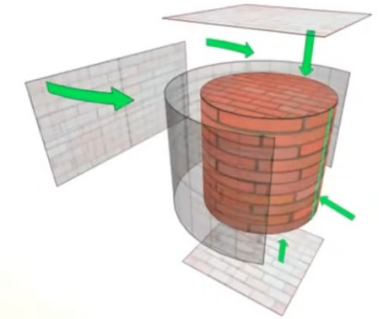
Planar Mapping

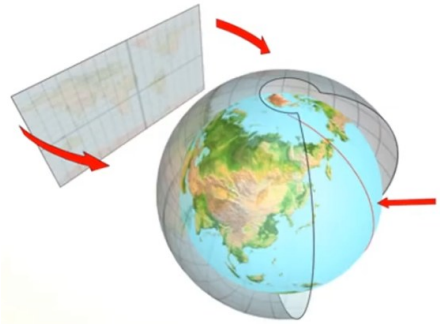


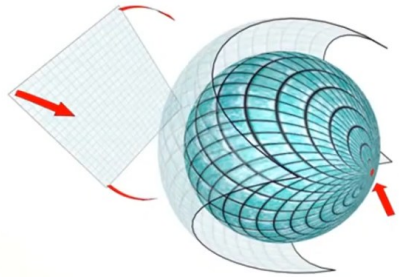
Cubical Mapping



Cylindrical Mapping



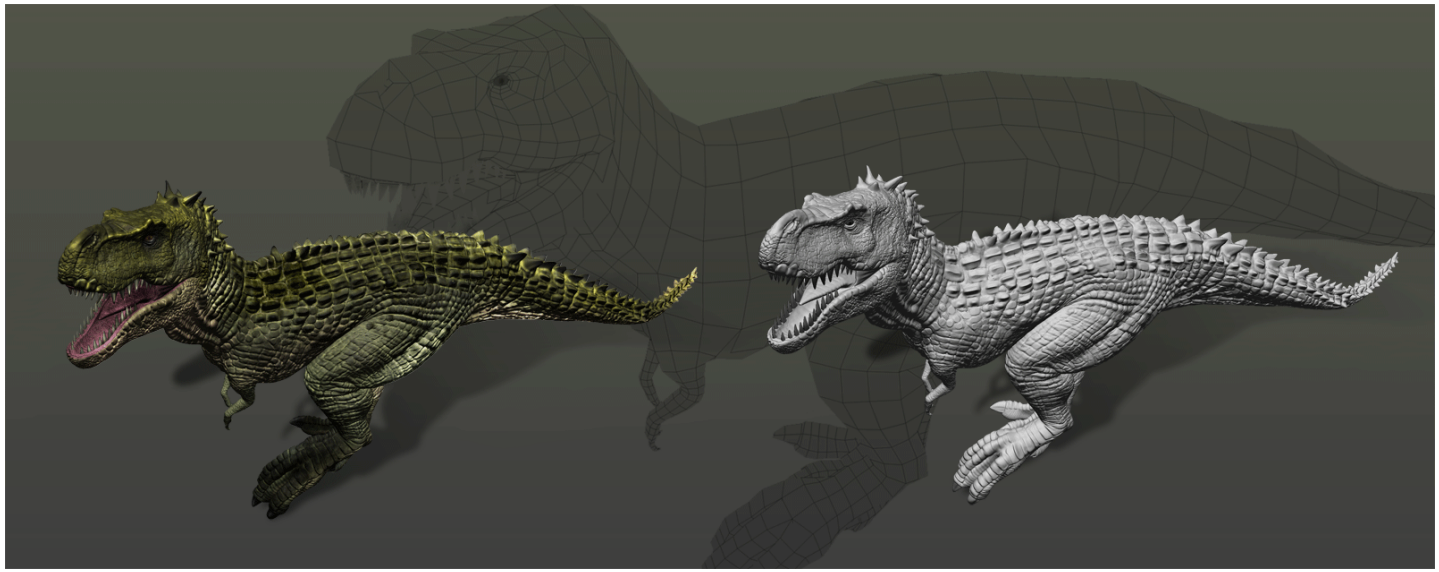




Spherical Mapping

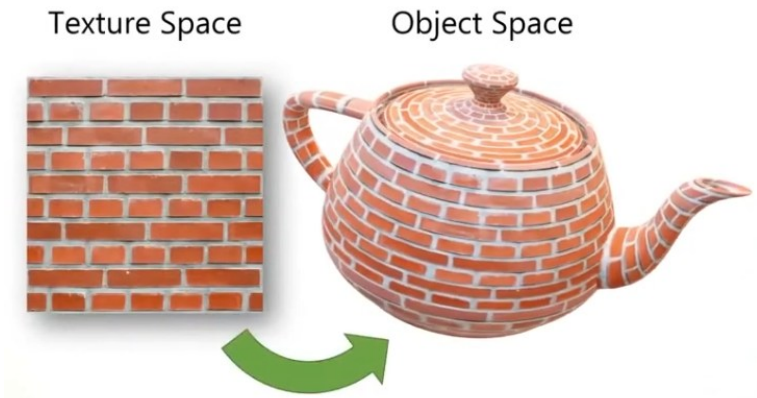
Textures

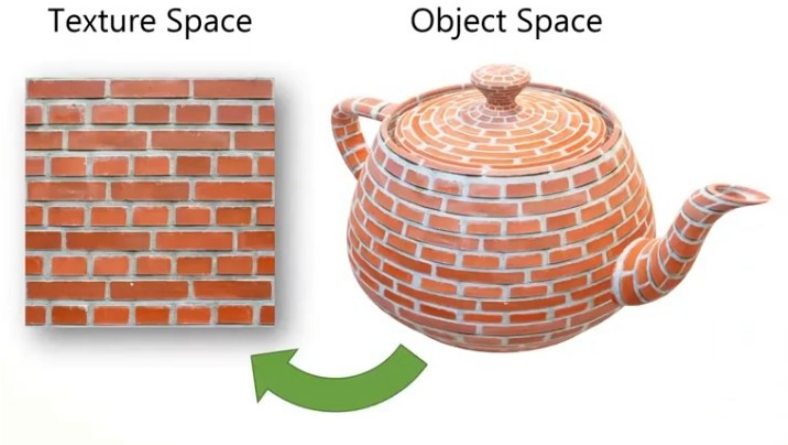
Τι γίνεται όμως με πολύπλοκα μοντέλα που δεν είναι απλά γεωμετρικά στερεά;

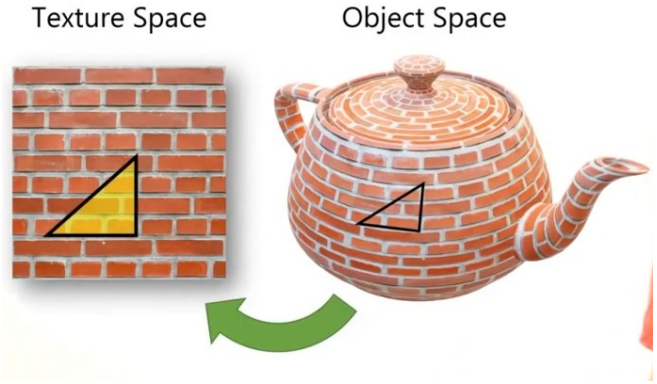


Σε τέτοιες περιπτώσεις ακολουθούμε μια διαδικασία που λέγεται **«texture mapping»**.

Το ζητούμενο στο texture mapping είναι να βρούμε μια συνάρτηση, που να πάρει το texture και να το αντιστοιχήσει στο αντικείμενο. Να κάνει δηλαδή την μετατροπή από το texture space στο Object Space. Στην πράξη όμως αυτό που κάνουμε είναι το ακριβώς αντίθετο! **Ζητάμε έναν τρόπο να αντιστοιχήσουμε το Object Space στο Texture Space!.**



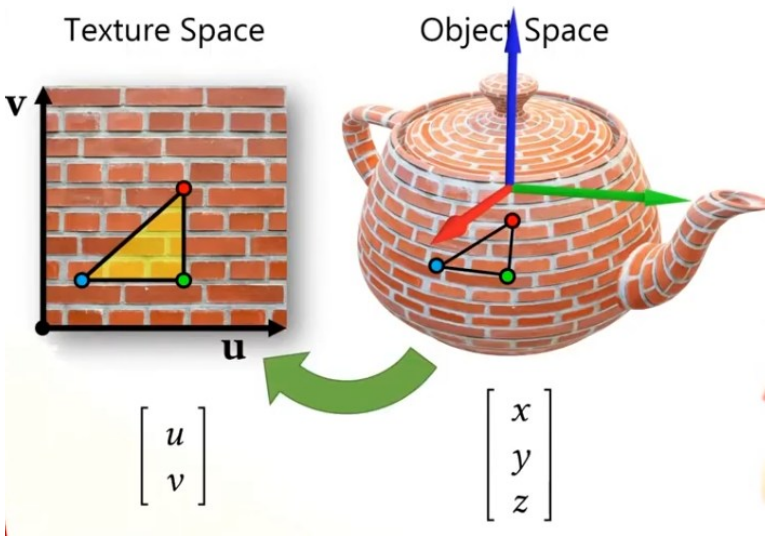




Καθώς όπως είδαμε τα αντικείμενά μας καταλήγουν να γίνονται triangular meshes, **ουσιαστικά αναζητούμε έναν τρόπο να αντιστοιχήσουμε κάθε τρίγωνο που απαρτίζει το αντικείμενο σε μια περιοχή του texture.**

Textures

Ή διαδικασία ΔΕΝ είναι ένας μαθηματικός τρόπος αντιστοίχισης, δεν περιγράφεται πχ από μία συνάρτηση.



Θα πρέπει για κάθε ένα από τα 3 vertices του τριγώνου να βρω το αντίστοιχο σημείο στο texture.

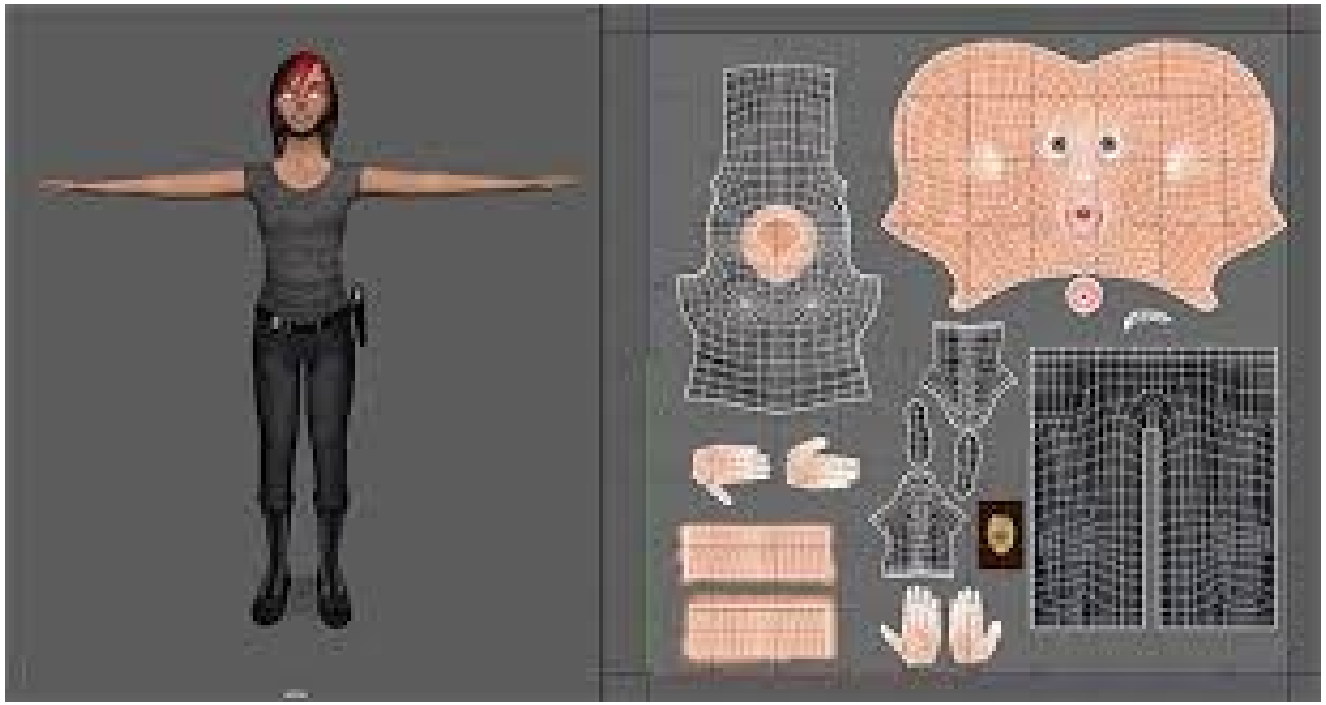
Συνεπώς έχω μια αντιστοίχιση από το Object Space (x,y,z) σε ένα 2D texture space (u,v).

Γι' αυτό και η όλη διαδικασία συχνά αναφέρεται και με τον όρο «UV mapping».

Είναι σημαντικό να έχουμε κατά νου ότι αυτή η διαδικασία δε γίνεται μέσω κάποιας μαθηματικής συνάρτησης. Ουσιαστικά αντιστοιχώ κάθε τρίγωνο σε μια περιοχή του texture.

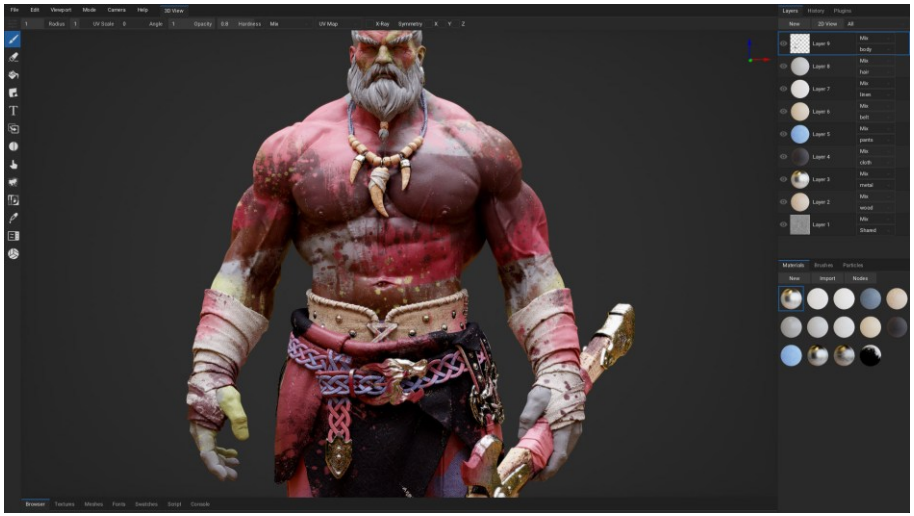
Αυτό που γίνεται λοιπόν είναι ότι κατά τη διάρκεια του σχεδιασμού του 3D αντικειμένου, με μηχανισμούς που παρέχουν τα κατάλληλα λογισμικά 3D modeling, γίνεται από το σχεδιαστή και το UV Mapping. ΤΟ λογισμικό βοηθάει εξαιρετικά τη διαδικασία, αλλά απαιτείται και χειρωνακτική εργασία από τον σχεδιαστή. Το UV Mapping λοιπόν μας έρχεται έτοιμο μαζί με το μοντέλο.

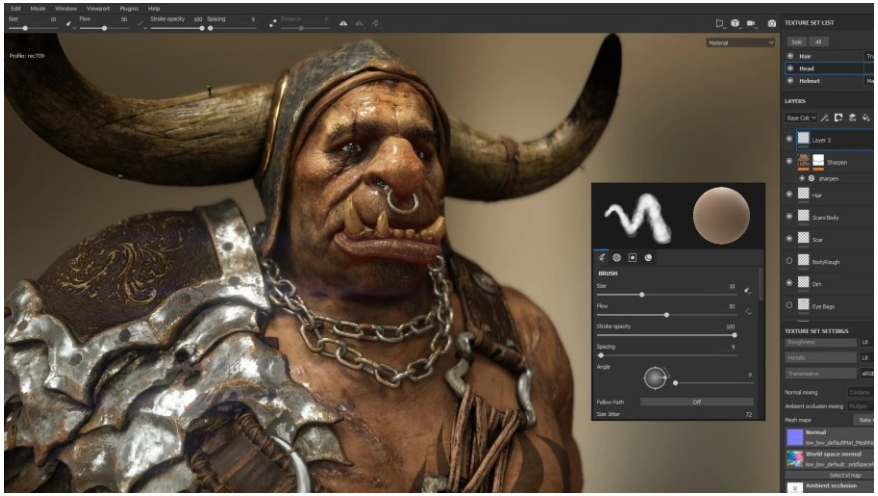
Textures



Συνήθως τα προγράμματα 3d modeling κάνουν το λεγόμενο «UV unwraping”. Ξεδιπλώνουν το 3D μοντέλο σε ένα δισδιάστατο χώρο διευκολύνοντας έτσι να αντιστοιχήσουμε στα πολύγωνα περιοχές μιας εικόνας ή να τα «βάψουμε» με εργαλεία που διαθέτει το λογισμικό.

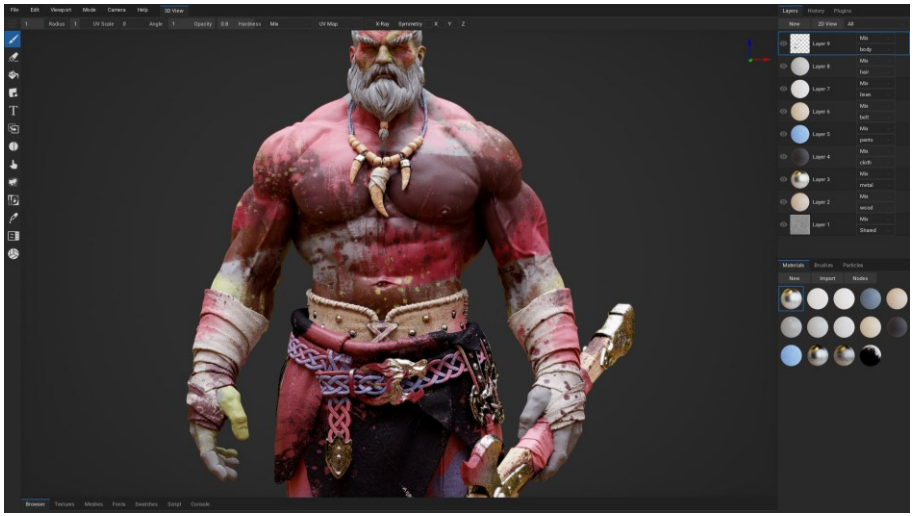
Textures

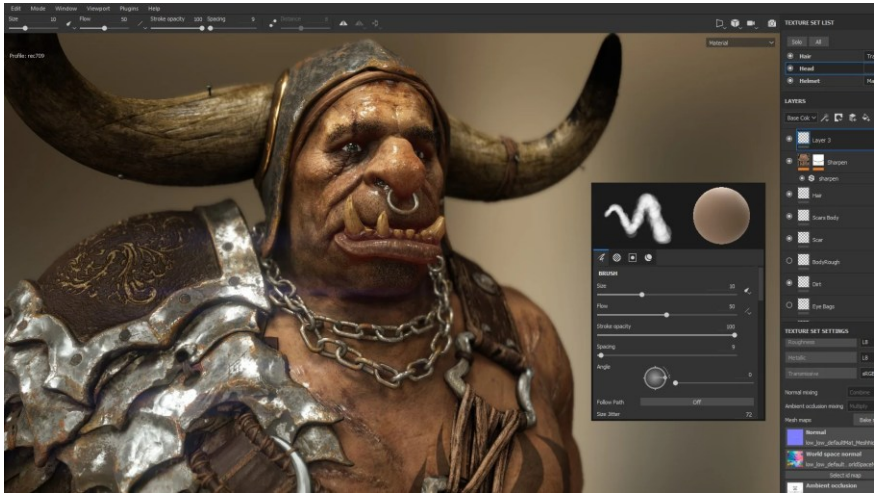




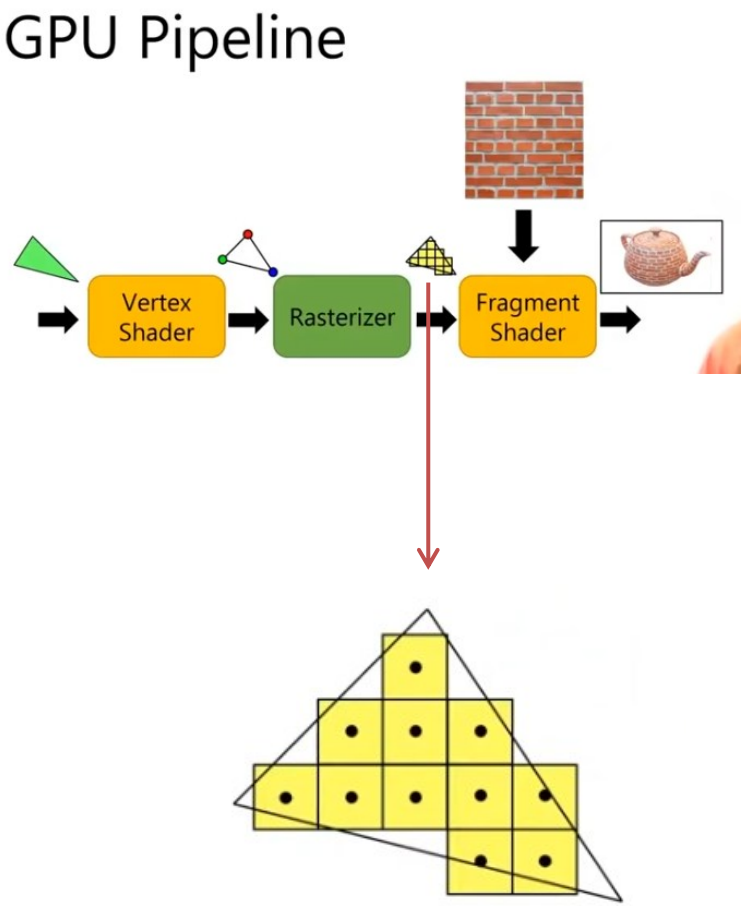
Υπάρχουν και λογισμικά που μας επιτρέπουν να «βάφουμε» κατευθείαν πάνω στο 3D model κατασκευάζοντας παράλληλα και το ίδιο το texture αλλά και το UV Mapping.

Textures





Υπάρχουν και λογισμικά που μας επιτρέπουν να «βάφουμε» κατευθείαν πάνω στο 3D model κατασκευάζοντας παράλληλα και το ίδιο το texture αλλά και το UV Mapping.



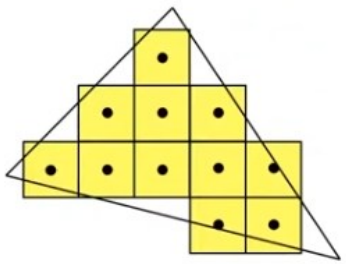
Στο GPU Pipeline, τα textures πηγαίνουν κατευθείαν στο Fragment Shader που χρωματίζει τα fragments που έχει φτιάξει ο Rasterizer.

Οι συντεταγμένες UV για κάθε vertex συνοδεύουν εξαρχής τα vertices σαν properties (μαζί με τ τις συντεταγμένες θέσεις τους και άλλα Properties που αφορούν τα vertices). Περνάνε από στάδιο σε στάδιο και φτάνουν στο Fragment shader για να τις χρησιμοποιήσει.

Ο Rasterizer για κάθε τρίγωνο του μοντέλου , κατασκευάζει τα fragments που το «γεμίζουν». Η GPU Λοιπόν θα πρέπει για το κέντρο κάθε fragment να υπολογίσει τις συντεταγμένες του texture (UV) που αντιστοιχούν !

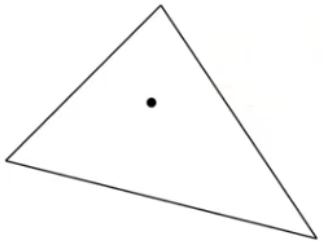
(Υπενθύμιση : είναι γνωστές οι συντεταγμένες UV μόνο για τα 3 vertices του τριγώνου – όχι για κάθε fragment στο εσωτερικό του!)

Textures



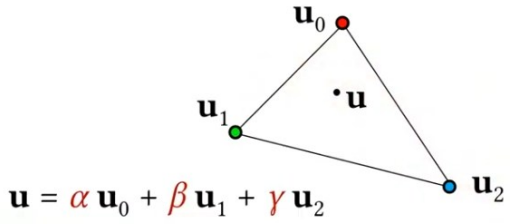
Ο Rasterizer για κάθε τρίγωνο του μοντέλου , κατασκευάζει τα fragments που το «γεμίζουν». Η GPU Λοιπόν θα πρέπει για το κέντρο κάθε fragment να υπολογίσει τις συντεταγμένες του texture (UV) που αντιστοιχούν !

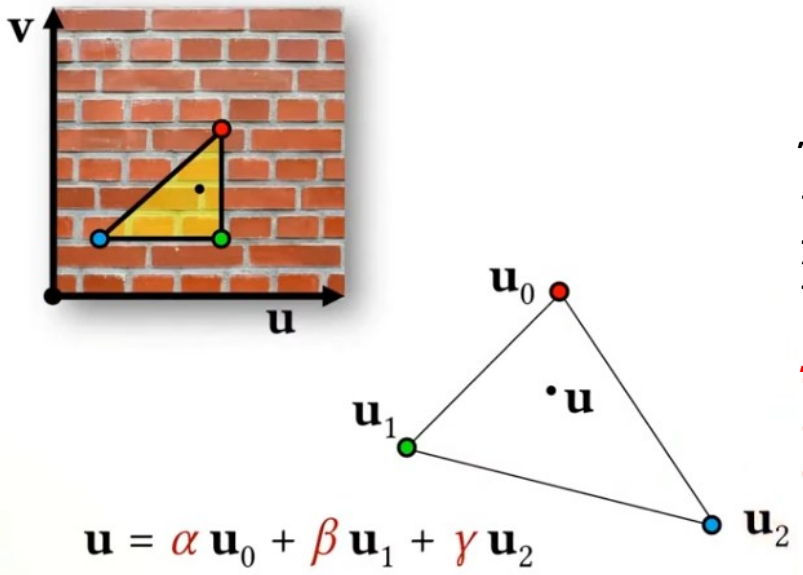
(Υπενθύμιση : είναι γνωστές οι συντεταγμένες UV μόνο για τα 3 vertices του τριγώνου – όχι για κάθε fragment στο εσωτερικό του!)



Αν εξετάσουμε λοιπόν ένα από τα παραπάνω κέντρα των fragments, τότε ακολουθώντας ακριβώς την ίδια λογική με τις βαρυκεντρικές συντεταγμένες που είδαμε στα triangular meshes, γνωρίζοντας τις UV συντεταγμένες των 3 vertices του τριγώνου (u_0, u_1 και u_2), **μπορούμε να υπολογίσουμε τις UV συντεταγμένες του u με γραμμικό interpolation των τριών vertices!**

Αυτό το κάνει αυτόματα η GPU ή το λογισμικό (πχ OpenGL).



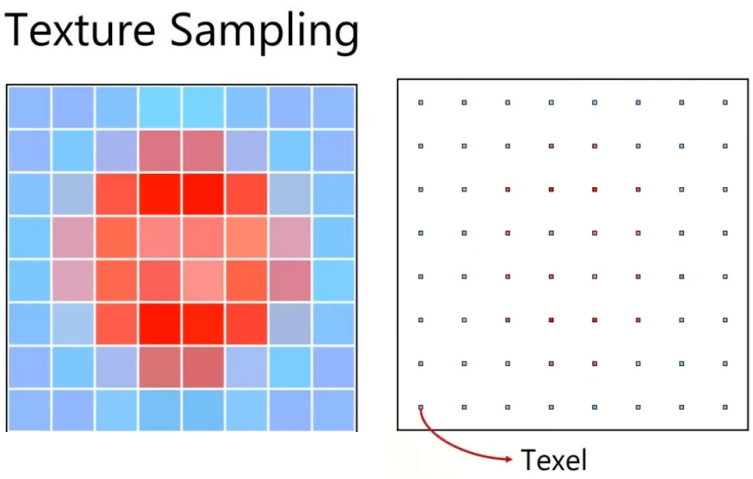


Έτσι λοιπόν μπορούμε να υπολογίσουμε τη θέση του u στο texture και να πάρουμε από εκεί το χρώμα με το οποίο θα χρωματιστεί όλο το συγκεκριμένο fragment και τελειώσαμε!

Όχι ακριβώς, στην πραγματικότητα η λήψη του χρώματος από το texture δεν είναι τόσο απλή, και εδώ μπαίνουμε στη διαδικασία του texture sampling!

Texture Sampling (Filtering)

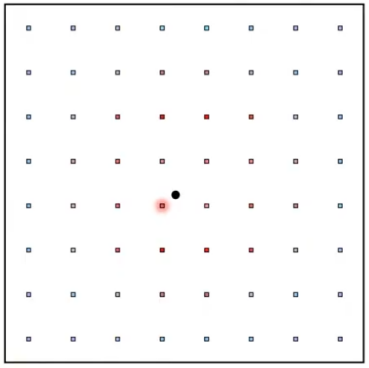
Texture sampling λέγεται η διαδικασία με την οποία λαμβάνουμε από ένα texture μια τιμή χρώματος για κάποιο σημείο του.



Κάθε pixel του texture θα πρέπει να το αντιλαμβανόμαστε όχι σαν χρωματισμένο «κουτάκι» αλλά σαν μια τιμή χρώματος στο κέντρο του, που ονομάζουμε Texel.

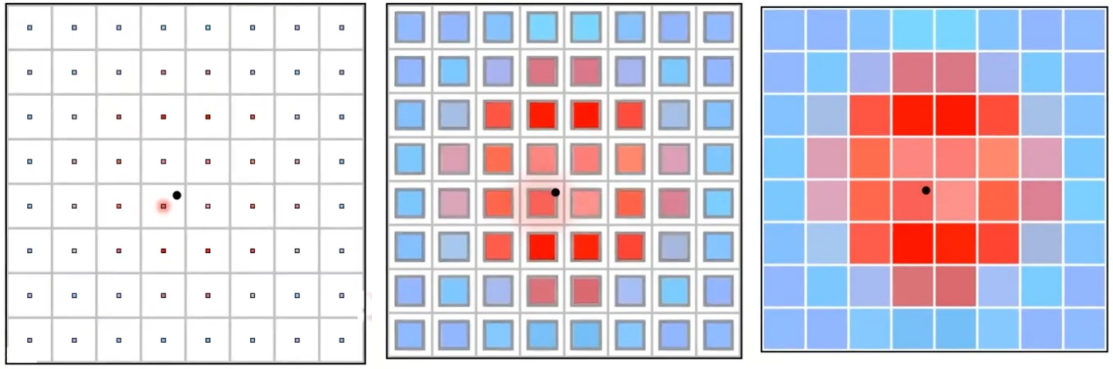
Texture Sampling (Filtering)

Nearest



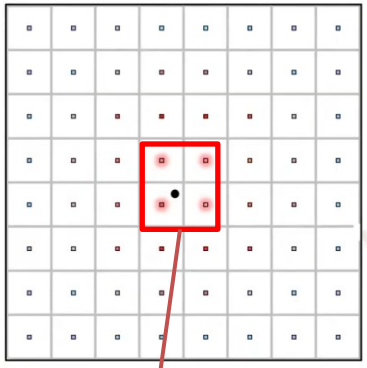
Ο απλούστερος λοιπόν τρόπος για να πάρουμε μια τιμή χρώματος σε ένα σημείο του texture είναι να πάρουμε το χρώμα του πιο κοντινού στο σημείο texel.

Nearest

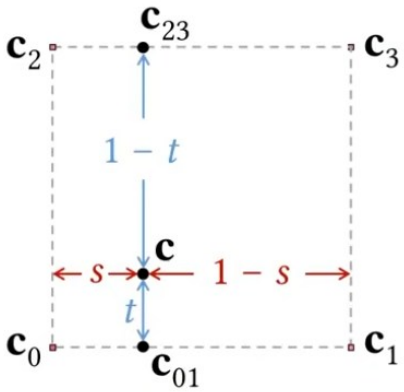


Με αυτό τον τρόπο, είναι σα να έχουμε χωρίσει το texture σε ένα grid και μέσα σε κάθε «κελί» του grid έχουμε ένα χρώμα. Είναι το απλούστερο που μπορούμε να κάνουμε αλλά ΔΕΝ έχει το καλύτερο αποτέλεσμα.

Bilinear Filtering



Ένας εναλλακτικός τρόπος είναι **να λάβουμε υπόψη τα 4 γειτονικά στο σημείο texels** και να κάνουμε μια γραμμική παρεμβολή (linear interpolation) στον οριζόντιο και τον κατακόρυφο άξονα – **μια BILINEAR λοιπόν παρεμβολή (Interpolation)**



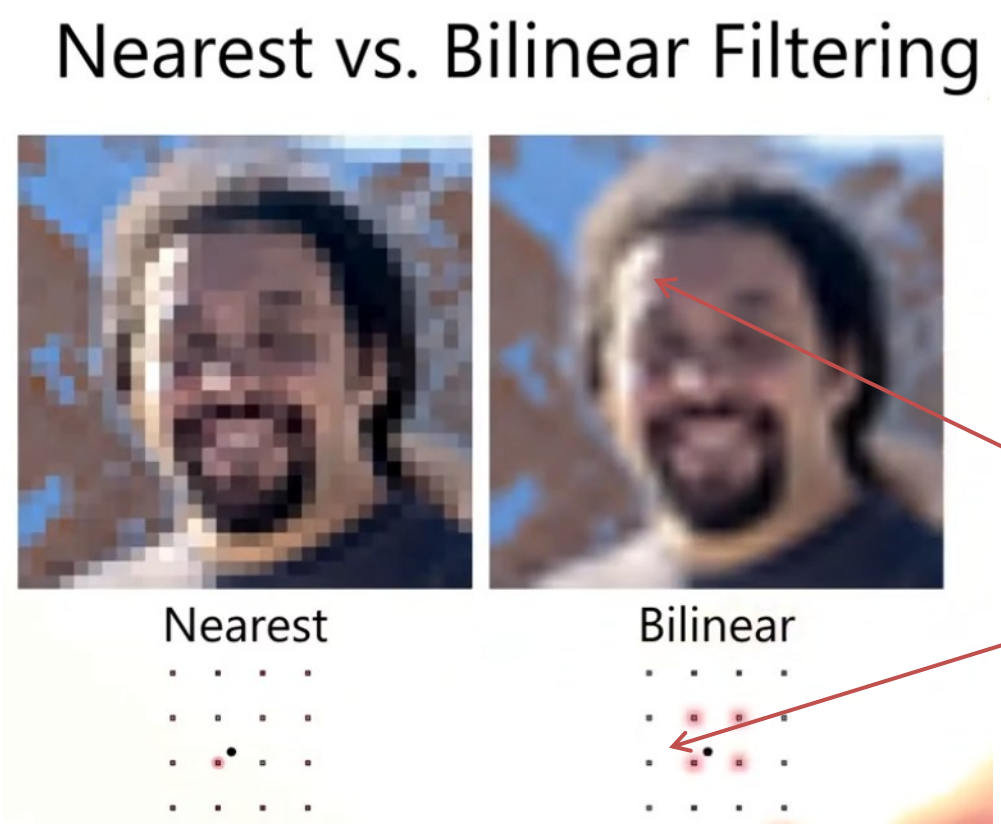
$$\mathbf{c}_{01} = (1-s)\,\mathbf{c}_0 + s\,\mathbf{c}_1$$
$$\mathbf{c}_{23} = (1-s)\,\mathbf{c}_2 + s\,\mathbf{c}_3$$
$$\mathbf{c} = (1-t)\,\mathbf{c}_{01} + t\,\mathbf{c}_{23}$$

Υπολογίζουμε λοιπόν την τιμή του χρώματος c σαν bilenear interpolation των χρωμάτων των τεσσάρων γειτονικών textles c_0, c_1, c_2 και c_3.

Οριζόντια (1ο interpolation):
Υπολογίζουμε δύο interpolated τιμές, τις :
c_{01} ως interpolation των c_0 και c_1
c_{23} ως interpolation των c_2 και c_3

Κατακόρυφα (2ο interpolation):
Υπολογίζουμε την τελική τιμή c σαν interpolation των c_{01} και c_{23} που υπολογίσαμε πριν.

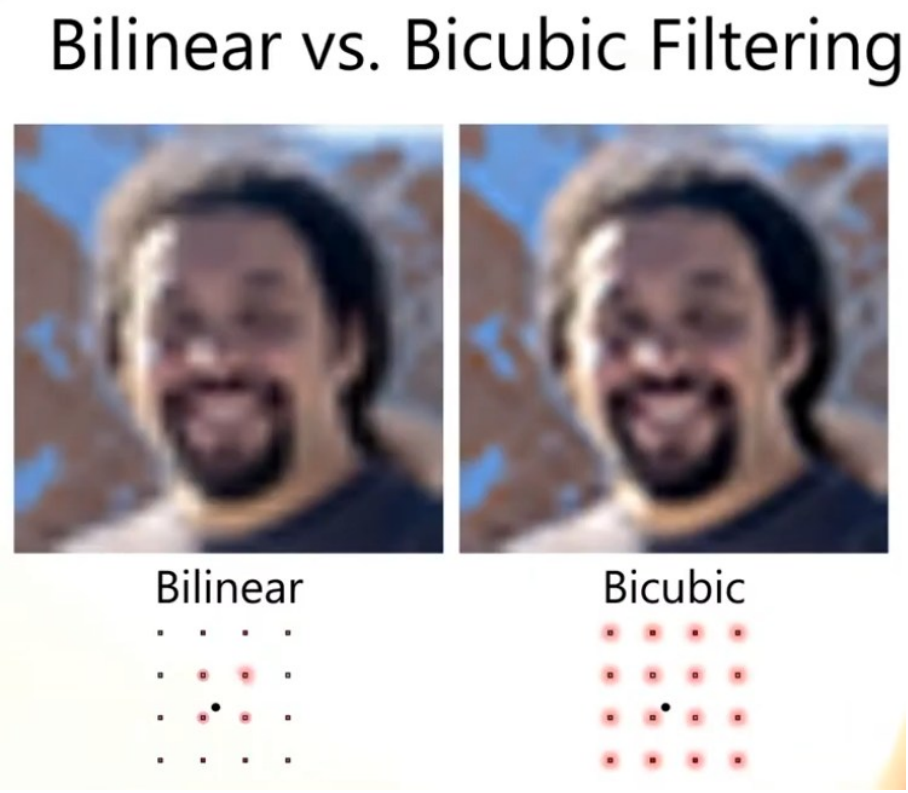
$$\mathbf{c} = (1-t)(1-s)\,\mathbf{c}_0 + (1-t)s\,\mathbf{c}_1 + t\,(1-s)\,\mathbf{c}_2 + ts\,\mathbf{c}_3$$



Το όφελος από το Bilinear sampling (filtering) είναι μια πιο ομαλή μετάβαση από texel σε texel του texture σε σχέση με το σαφώς διακριτό του nearest.

Ακόμα και στο bilinear όμως μπορούμε να διακρίνουμε σε κάποιες περιοχές τη μετάβαση.

Αυτό συμβαίνει γιατί αν το σημείο που κάνουμε δειγματοληψία πάει πχ αριστερότερα θα υπολογιστεί με μια διαφορετική τετράδα texels και θα προκύψει μια κάπως απότομη μετάβαση.



Για ακόμα ομαλότερη μετάβαση μπορούμε να κάνουμε Bicubic Filtering (sampling) λαμβάνοντας υπόψη ένα πλέγμα 4X4 των γειτονικών texels. Με αυτό τον τρόπο έχουμε μια ομαλότερη μετάβαση χρώματος καθώς το σημείο δειγματοληψίας κινείται μέσα στην περιοχή του texture.

Bicubic Filtering vs. …

Μεγαλύτερος αριθμός από interpolated texels δε θα φέρει σημαντικό όφελος.

Αν θέλουμε καλύτερο τελικό αποτέλεσμα η λύση είναι μία: **ΜΕΓΑΛΥΤΕΡΗΣ ανάλυσης texture!**

Ιδανική απεικόνιση έχω όταν το μέγεθος του texel είναι ίδιο ή μικρότερο από το μέγεθος του Pixel στην οθόνη.

Bilinear Filtering

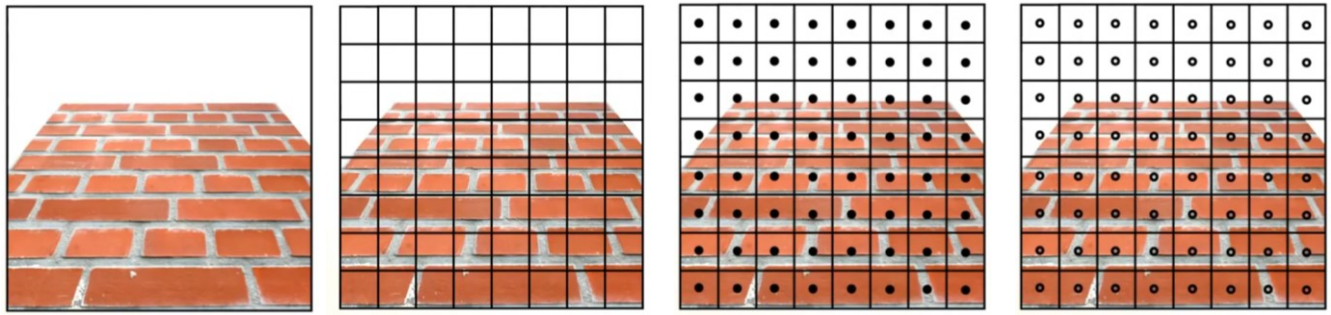
Bilinear Filtering

Texture Sampling (Filtering)

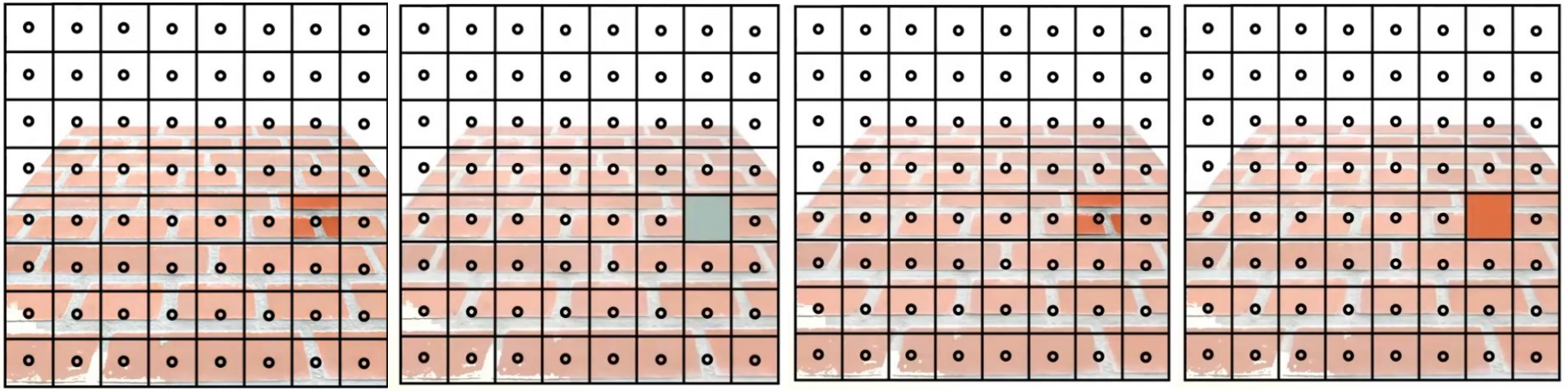
Ένα υψηλής ανάλυσης texture όμως χάνει τη σκοπιμότητά του όσο απομακρύνομαι από το αντικείμενο και μικραίνει το μέγεθός του στην οθόνη.

Δε χρειαζόμαστε τόσο υψηλής ανάλυσης texture σε τέτοια απόσταση!

Texture Sampling (Filtering)



Όταν εφαρμόζω ένα texture σε μια επιφάνεια και κάνω rendering από μια συγκεκριμένη κάμερα, κάθε pixel στην οθόνη θα χρωματιστεί με δειγματοληψία του texture στο κέντρο κάθε Pixel.

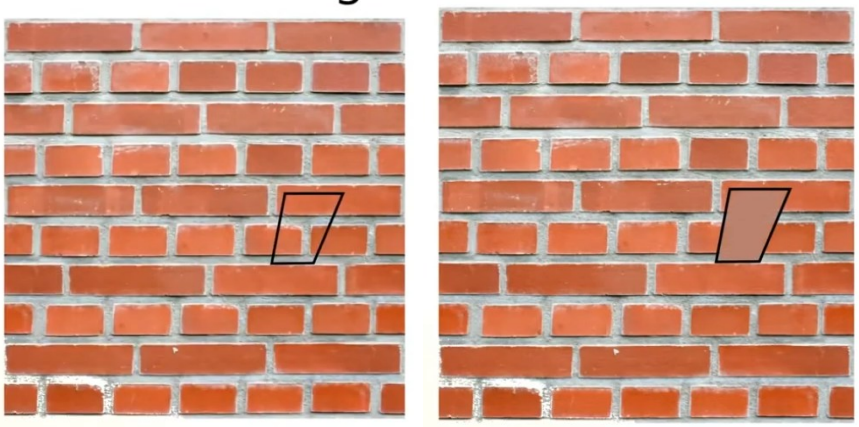


Κατά το rendering, για ένα συγκεκριμένο Pixel, θα γίνει η δειγματοληψία στο texture (με όποιον αλγόριθμο επιλέξουμε) και θα επιλεγεί το χρώμα που αντιστοιχεί και θα βαφεί όλο το Pixel.

Με μια πολύ μικρή μετατόπιση όμως της κάμερας, ή κίνηση του αντικειμένου, η δειγματοληψία μπορεί να δώσει ένα εντελώς διαφορετικό χρώμα.

Texture Sampling (Filtering)

Desired Filtering



Όταν θέλω να κάνω δειγματοληψία σε μια περιοχή ενός texture για να βρω με τι χρώμα θα βάψω ένα pixel, θέλω να προκύψει ένα αντιπροσωπευτικό χρώμα από τη συγκεκριμένη περιοχή του texture.

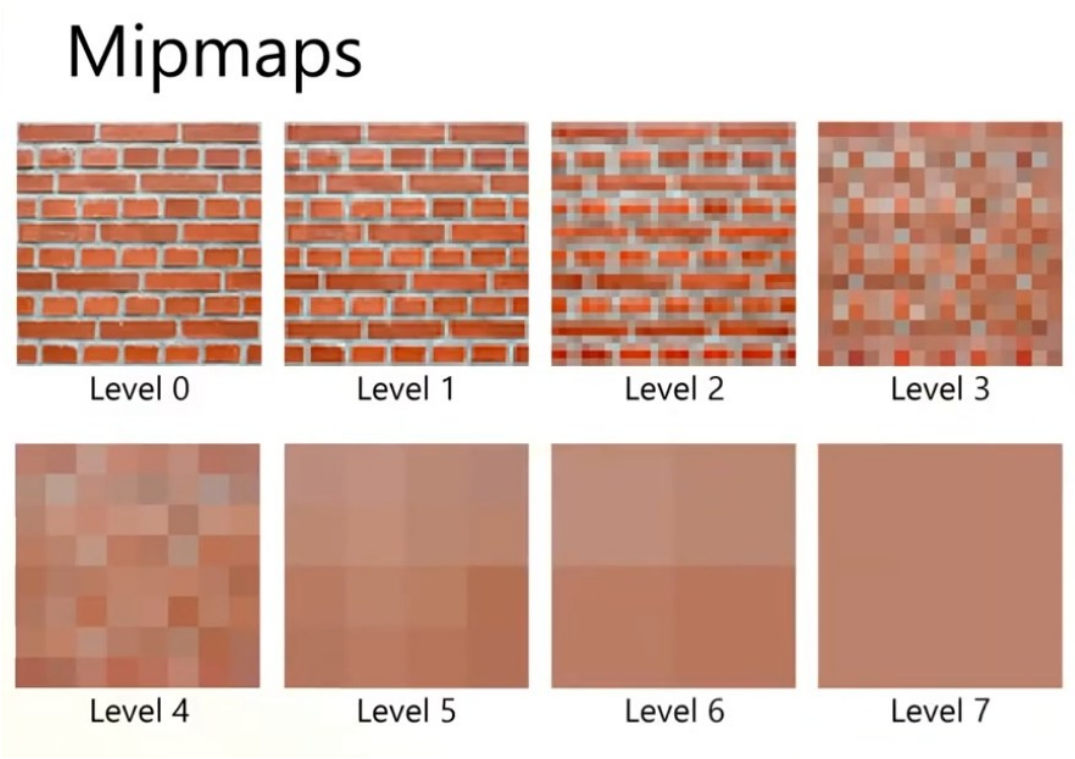
Αν όμως το texture είναι πολύ υψηλής ανάλυσης στην περιοχή αυτή που θα χρωματίσει ένα pixel μπορεί να αντιστοιχεί νέας πολύ μεγάλος αριθμός από texels. Αν μάλιστα το αντικείμενο είναι μακριά θα μπορούσε και ολόκληρο το texture να αντιστοιχεί στο χρωματισμό ενός pixel!

Αυτό εισάγει **έναν πολύ υψηλό υπολογιστικό φόρτο** χωρίς να υπάρχει κανένα όφελος στην ποιότητα.

Η λύση είναι το pre-filtering

Mipmaps

Ο πιο συνηθισμένος τρόπος για να κάνω το pre-filtering είναι τα MIPMAPS

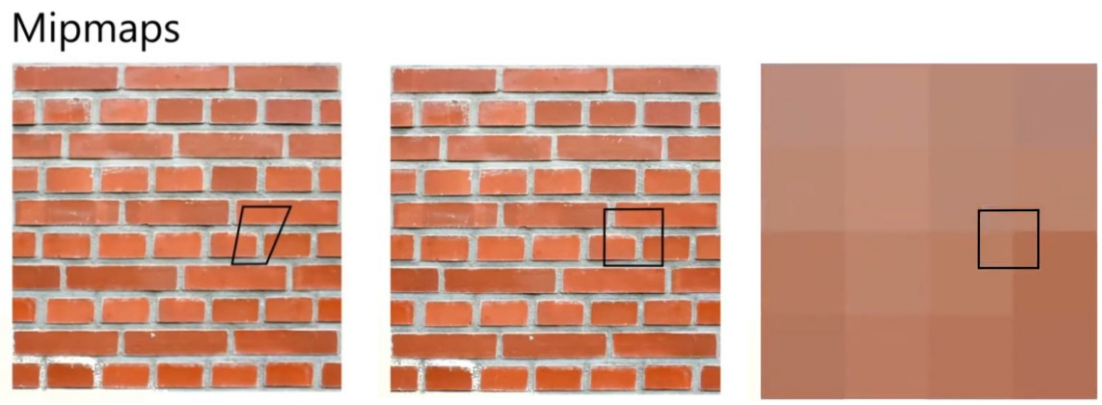


Αυτό που κάνω είναι από το αρχικό texture υψηλής ανάλυσης (level 0) να παράξω μια σειρά από αντίγραφά του , όλο και χαμηλότερης ανάλυσης (levels 1-7).

Κάθε επίπεδο έχει τη μισή ανάλυση από το προηγούμενό του , τόσο οριζόντια όσο και κατακόρυφα.

Αυτός είναι και ο λόγος που οι μηχανές γραφικών προτιμούσαν (ή επέβαλαν) τα textures να έχουν διαστάσεις που είναι δύναμη του 2 , ώστε να είναι εύκολη η παραγωγή των mipmaps. Κάθε 4 Pixels θα αντιστοιχούν σε 1 pixels του επόμενου level mipmap.

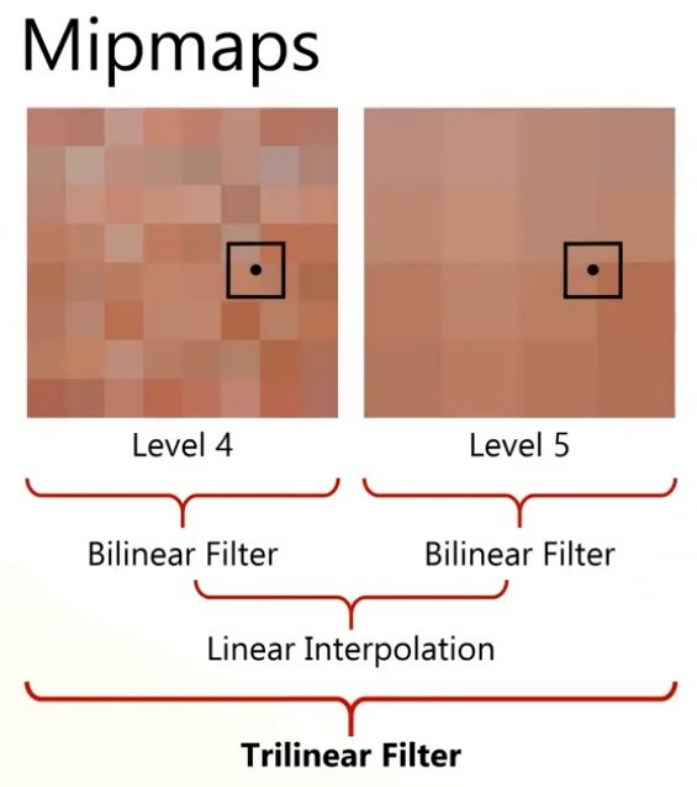
Mipmaps



Πώς χρησιμοποιώ τα mipmaps;

Όταν για να χρωματίσω ένα Pixel πρέπει να κάνω δειγματοληψία σε μια περιοχή του texture (εικόνα 1), χρησιμοποιώ μια τετράγωνη προσέγγιση της περιοχής (εικόνα 2).

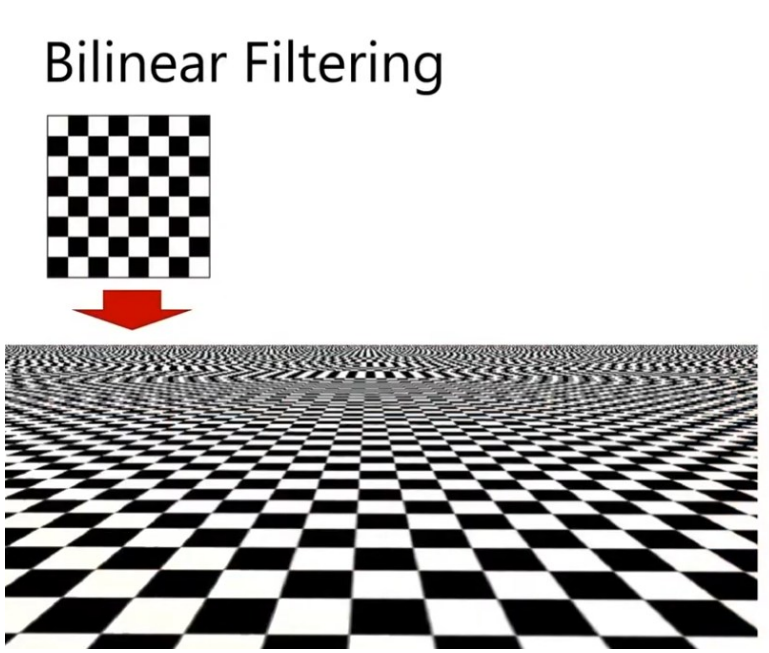
Αντί μετά να κάνω δειγματοληψία στο υψηλής ανάλυσης texture, επιλέγω το mipmap του οποίου το texel έχει παραπλήσιο μέγεθος με την περιοχή δειγματοληψίας (εικόνα 3). Έτσι η δειγματοληψία θα υπολογιστεί από πολύ λιγότερες τιμές.



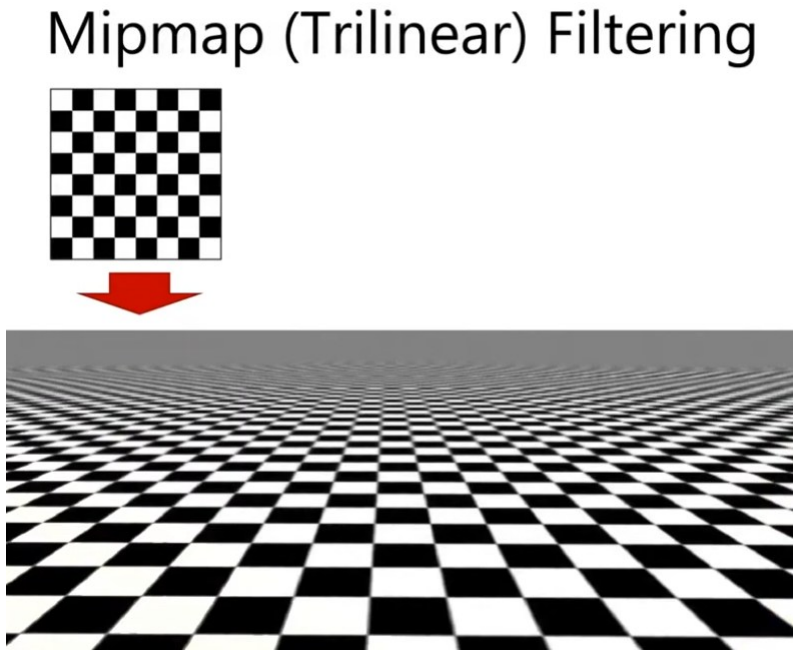
Ένα επιπλέον πρόβλημα είναι ότι τα διαθέσιμα mipmap levels μπορεί να είναι μεγαλύτερα (level 5) ή μικρότερα (level 4) από την περιοχή που θα ήθελα .

Το καλύτερο που μπορώ να κάνω είναι να κάνω bilinear interpolation στα δύο levels και στη συνέχεια ένα linear Interpolation στο χρώμα που θα δώσουν αυτά τα δύο. Αυτό το ονομάζω Trilinear Filtering.

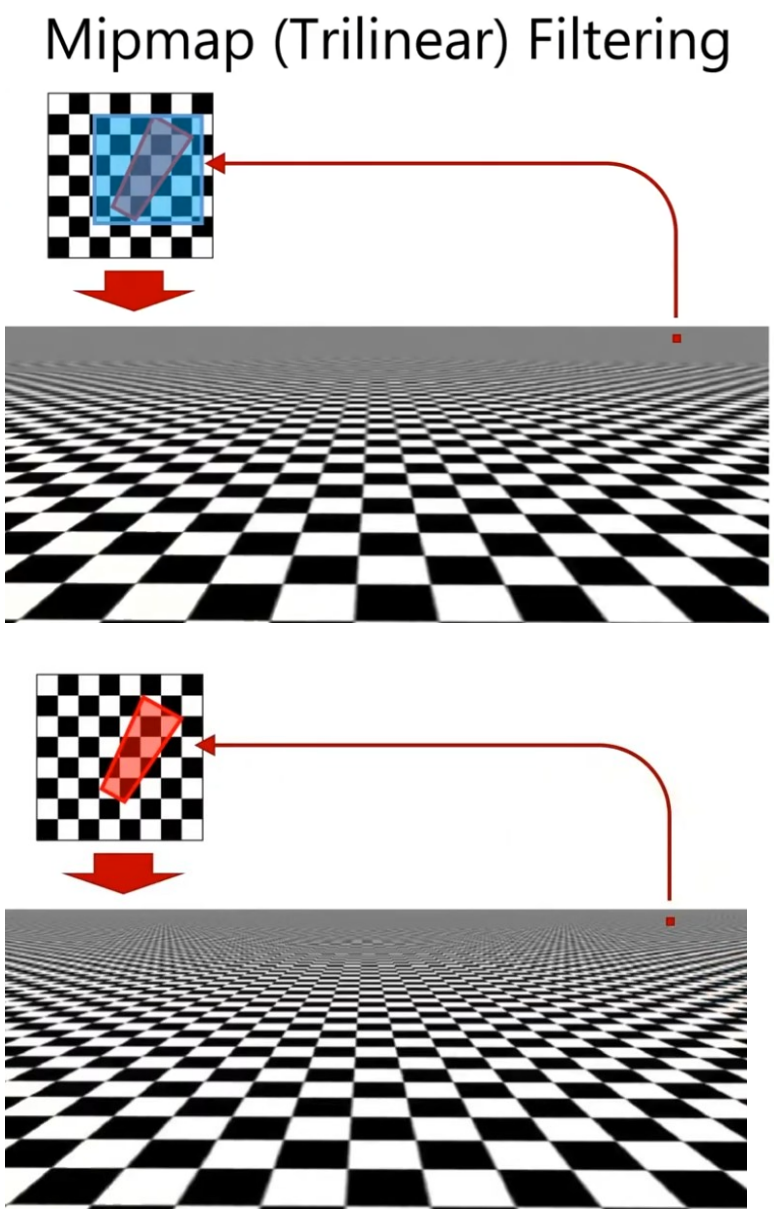
Mipmaps



Στο bilinear filtering τα πράγματα δείχνουν σωστά κοντά, αλλά όσο απομακρυνόμαστε και το texture αντιστοιχεί σε όλο και λιγότερα pixels της οθόνης, εμφανίζονται τεχνουργήματα.



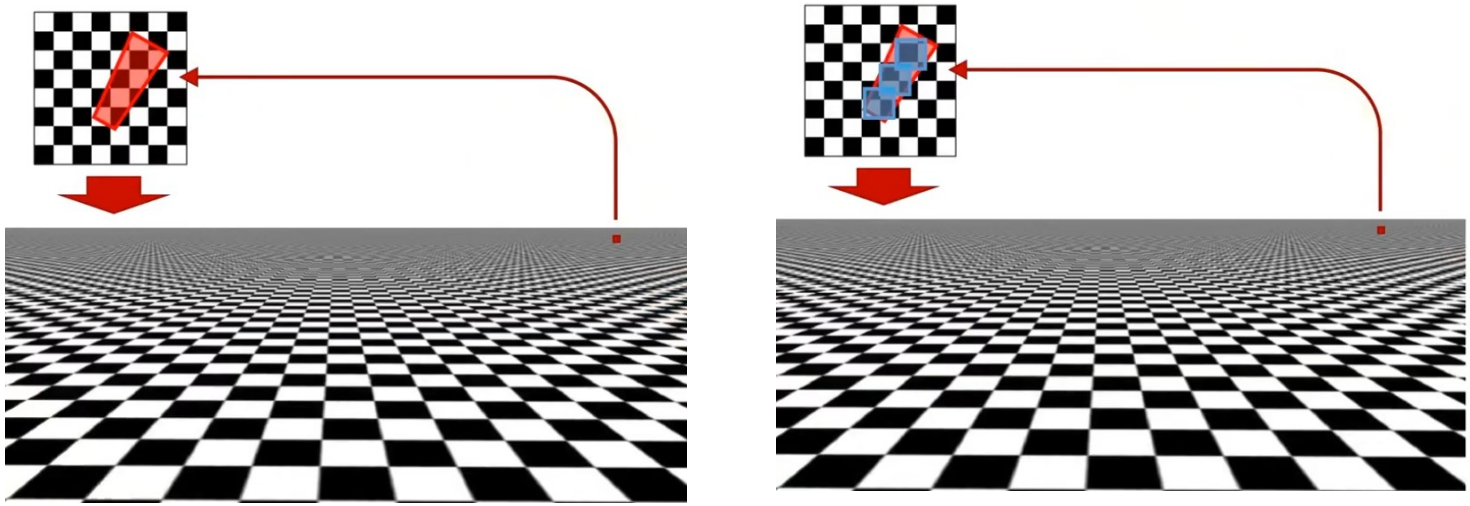
Με τα mipmaps (trilinear filtering) το αποτέλεσμα είναι πολύ καλύτερο καθώς όσο απομακρυνόμαστε το texture απλά θολώνει αντί να εισάγονται παραμορφώσεις.



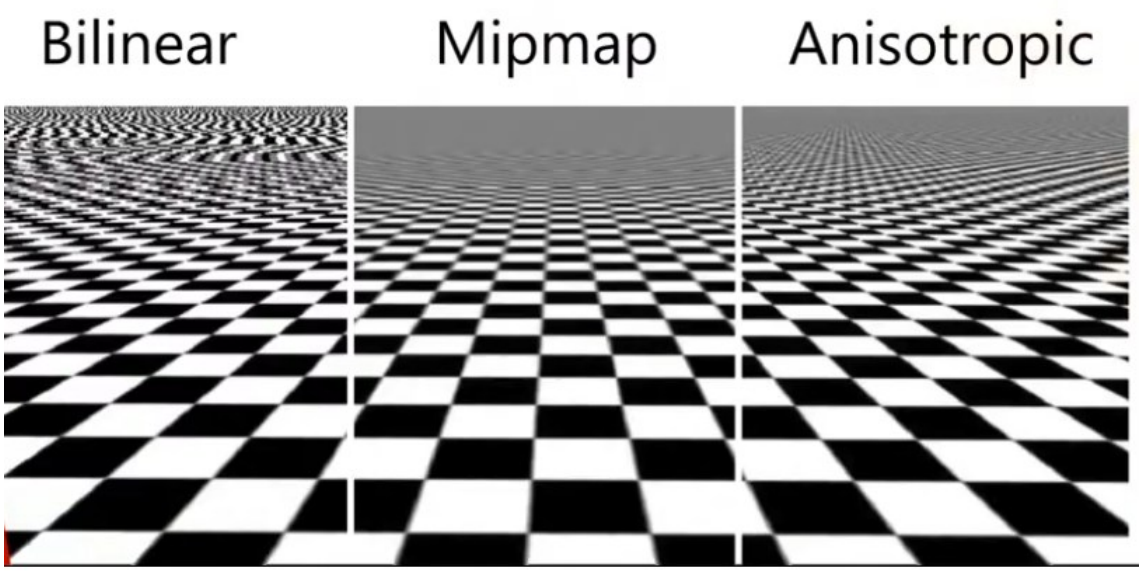
Ο λόγος που όσο απομακρυνόμαστε το texture θολώνει, είναι ότι στα απομακρυσμένα pixels, για το χρώμα τους έχουμε κάνει δειγματοληψία σε μια περιοχή του texture που προσεγγίζεται με ένα τετράγωνο και χρησιμοποιήσαμε το κατάλληλο level mipmap.

Αν κάπως καταφέρναμε να διατηρήσουμε όμως το σχήμα της περιοχής που δειγματοληπτούμε αντί να το προσεγγίζουμε με ένα τετράγωνο το αποτέλεσμα θα ήταν πιο ακριβές. **Αυτό το ονομάζουμε Anisotropic Filtering.**

Anisotropic Filtering with MipMaps



Αντί να προσεγγίζουμε την περιοχή δειγματοληψίας με ένα τετράγωνο που την περικλείει, κάνουμε πολλαπλά trilinear filterings σε μικρότερα επιμέρους τετράγωνα που προσεγγίζουν καλύτερα την περιοχή. Το Anisotropic Filtering ωστόσο είναι ακριβό υπολογιστικά.

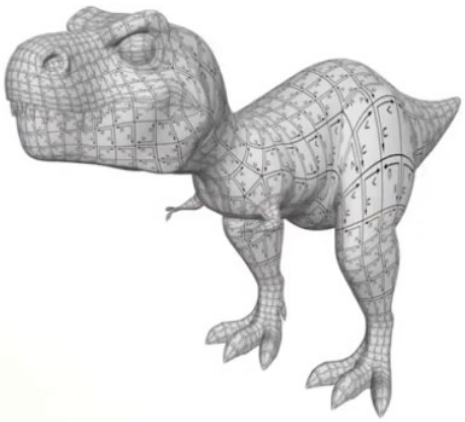


Οι σύγχρονες GPUs υποστηρίζουν σε επίπεδο hardware όλες αυτές τις διαδικασίες και τις εκτελούν εξαιρετικά γρήγορα.

Textures

- Raster Images (1D, 2D, 3D, …)
- Procedural Textures
- Alternative Structures

Ptex (Per-Face Textures)



Mesh Colors

Mesh Colors