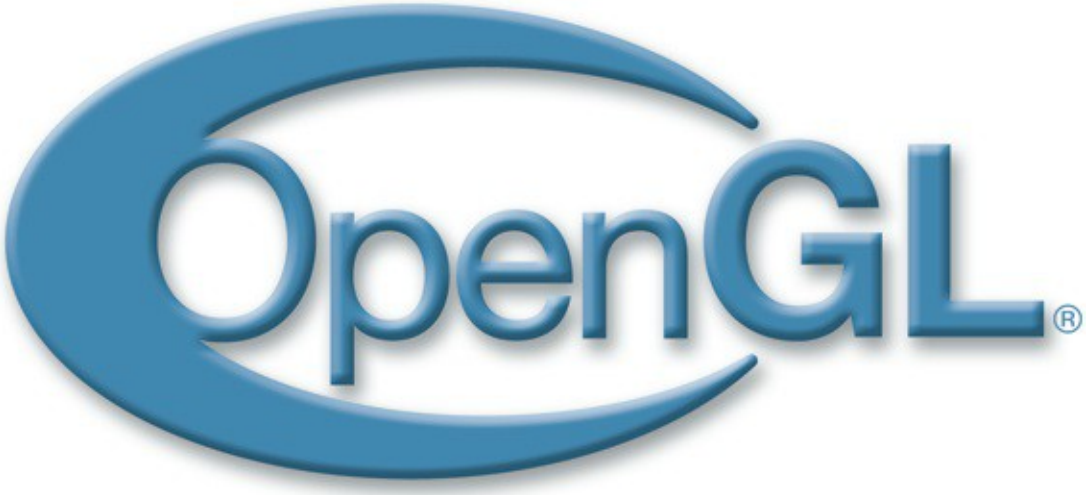
GRAPHICS AND CODE *DESIGN*



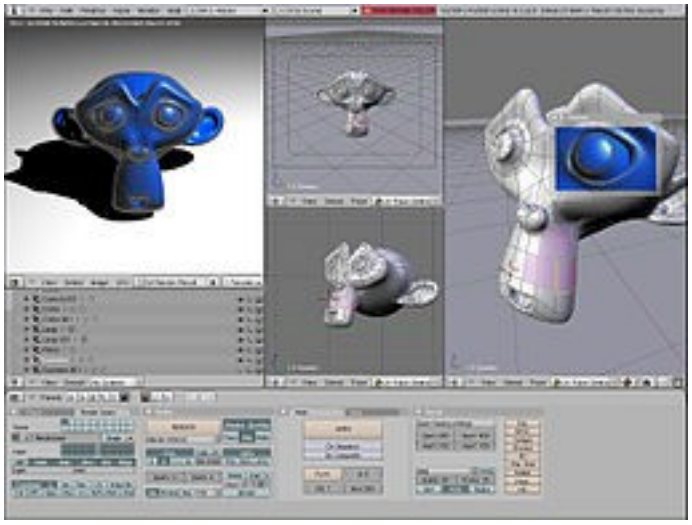
Author: Gountis Agelos
A.E.M: 2231

Περιεχόμενα σελ:

Τι είναι Γραφικά (Graphics);

Γραφικά (Graphics) είναι η παραγωγή οπτικών δηλώσεων πάνω σε κάποια επιφάνεια, όπως τοίχος, καμβάς, αγγείο, οθόνη υπολογιστή, χαρτί, πέτρα κτλ. . Περιλαμβάνει οτιδήποτε αφορά τη δημιουργία συμβόλων, διαγραμμάτων, logo, γράφων, σχεδίων (γραμμικών και μη γραμμικών), συμβόλων, γεωμετρικών σχεδίων και ούτω καθεξής.

Γραφικά Ηλεκτρονικών Υπολογιστών (Computer Graphics).



Τα γραφικά ηλεκτρονικών υπολογιστών είναι γραφικά που δημιουργούνται χρησιμοποιώντας ηλεκτρονικούς υπολογιστές οι οποίοι αναπαριστούν τα δεδομένα εικόνας, με τη βοήθεια ειδικού υλικού ή λογισμικού γραφικών.

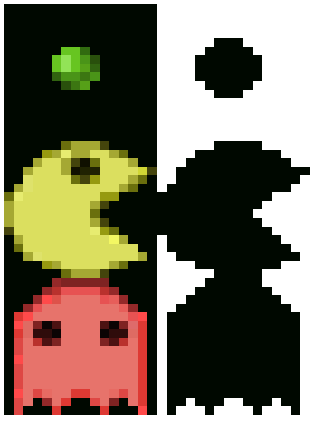
Η κατανόηση των υπολογιστών και η αλληλεπίδραση με αυτούς καθώς και η ερμηνεία των δεδομένων έχει γίνει ευκολότερη λόγω των γραφικών ηλεκτρονικών υπολογιστών. Η ανάπτυξη των γραφικών ηλεκτρονικών υπολογιστών είχε σημαντικό αντίκτυπο σε πολλούς τύπους πολυμέσων και έχουν φέρει επανάσταση στο animation, στις ταινίες και στη βιομηχανία βιντεοπαιχνιδιών.

Ο όρος γραφικά ηλεκτρονικών υπολογιστών έχει χρησιμοποιηθεί με μία ευρεία έννοια για να περιγράψει σχεδόν οτιδήποτε αφορά τους υπολογιστές και δεν είναι κείμενο ή ήχος. Τυπικά, ο όρος αυτός αναφέρεται σε πολλά διαφορετικά πράγματα όπως:

- Στην αντιπροσώπευση και τον χειρισμό δεδομένων εικόνας από έναν ηλεκτρονικό υπολογιστή.
- Στις διάφορες τεχνολογίες που χρησιμοποιούνται για τη δημιουργία και την επεξεργασία εικόνων.
- Στο υπο-τμήμα της πληροφορικής το οποίο ασχολείται με μεθόδους που χρησιμοποιούνται για την ψηφιακή σύνθεση και επεξεργασία οπτικού περιεχομένου.

Τα είδη των γραφικών.

- **Δισδιάστατα (2D Computer Graphics)**



Τα δισδιάστατα γραφικά είναι η "βασισμένη στους ηλεκτρονικούς υπολογιστές" δημιουργία των ψηφιακών εικόνων. Χρησιμοποιούνται κυρίως σε εφαρμογές που αναπτύχθηκαν αρχικά πάνω σε παραδοσιακές τεχνολογίες εκτύπωσης και σχεδίου, όπως η τυπογραφία.
Σ' αυτές τις εφαρμογές, η δισδιάστατη εικόνα δεν είναι απλά μία αντιπροσώπευση ενός αντικειμένου στον πραγματικό κόσμο, αλλά είναι ένα ανεξάρτητο τεχνούργημα με επιπρόσθετη σημασιολογική αξία. Γι' αυτό το λόγο προτιμώνται τα δισδιάστατα μοντέλα επειδή παρέχουν πιο άμεσο έλεγχο της εικόνας απ' ότι τα τρισδιάστατα γραφικά, των οποίων η προσέγγιση πλησιάζει πιο πολύ την φωτογραφία απ' ότι την τυπογραφία.

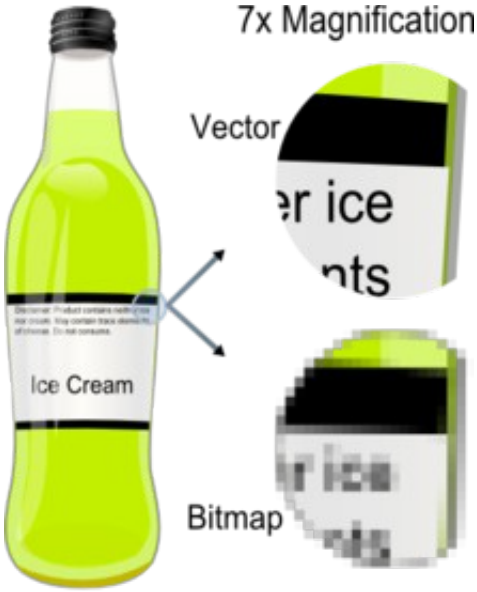
- **Τέχνη με Pixels (Pixel Art)**

Ένα μεγάλο τμήμα της ψηφιακής τέχνης είναι η "Τέχνη με Pixels". Δημιουργείται με τη χρήση προγραμμάτων γραφικών raster, όπου οι εικόνες επεξεργάζονται σε

επίπεδο pixel. Τα γραφικά στα περισσότερα (ή σε σχετικά περιορισμένα) βιντεοπαιχνίδια , ειδικά αυτά που τρέχουν σε παλιές κινητές συσκευές χρησιμοποιούν Pixel Art.

- **<u>Διανυσματικά Γραφικά (Vector Graphics)</u>**

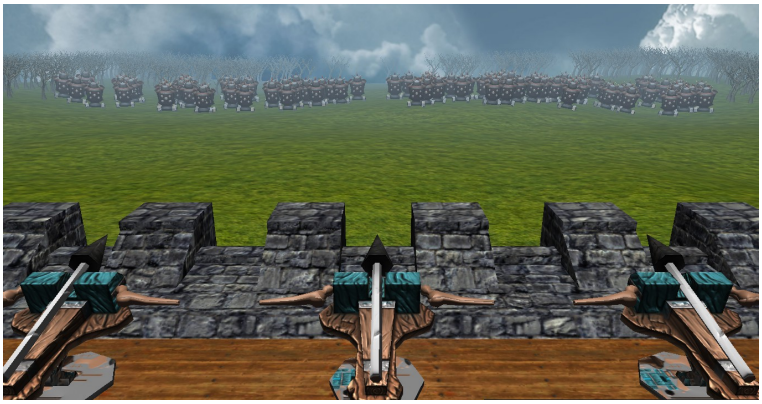
Τα formats των διανυσματικών γραφικών είναι συμπληρωματικά των γραφικών raster. Τα γραφικά raster είναι η αντιπροσώπευση των εικόνων ως πίνακες από pixels και τυπικά χρησιμοποιούνται για την απεικόνιση φωτογραφικών εικόνων. Τα διανυσματικά γραφικά συνίστανται στην κωδικοποίηση πληροφοριών που αφορούν σχήματα και χρώματα που αποτελούν την εικόνα, γεγονός που επιτρέπει μεγαλύτερη ευελιξία στην απόδοσή (rendering) της.



Ανάλογα με την περίπτωση προτιμάται η χρήση είτε διανυσματικών γραφικών είτε γραφικών raster. Σε ορισμένες περιπτώσεις χρησιμοποιούνται και τα δύο αυτά είδη γραφικών. Η κατανόηση των πλεονεκτημάτων και των περιορισμών της κάθε τεχνολογίας, και της σχέσης μεταξύ τους είναι απαραίτητη για την αποτελεσματική χρήση των εργαλείων επεξεργασίας τους.

- **<u>Τρισδιάστατα Γραφικά (3D Graphics)</u>**

Τα τρισδιάστατα γραφικά αν συγκριθούν με τα δισδιάστατα γραφικά, είναι γραφικά τα οποία χρησιμοποιούν τρισδιάστατη αντιπροσώπευση



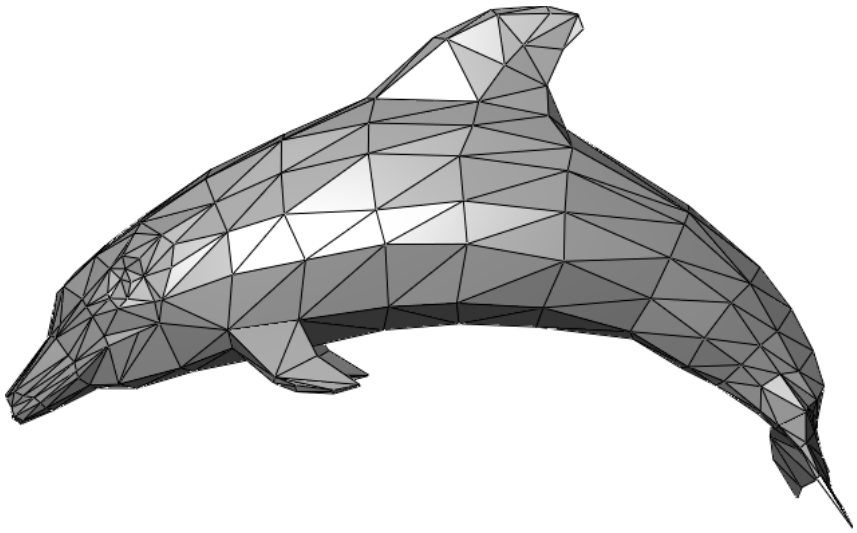
γεωμετρικών δεδομένων. Για να επιτευχθεί καλύτερη απόδοση τα δεδομένα αυτά αποθηκεύονται στον υπολογιστή.

Παρ' όλες τις διαφορές μεταξύ τρισδιάστατων και δισδιάστατων γραφικών, τα τρισδιάστατα γραφικά βασίζονται σε παρόμοιους αλγόριθμους με αυτούς των δισδιάστατων γραφικών. Στα λογισμικά γραφικών η διάκριση μεταξύ δισδιάστατων και τρισδιάστατων γραφικών δεν είναι ξεκάθαρη σε ορισμένες περιπτώσεις. Δισδιάστατες εφαρμογές μπορεί να χρησιμοποιήσουν τρισδιάστατες τεχνικές για να πετύχουν εφέ όπως ο φωτισμός , ενώ τα τρισδιάστατα γραφικά μπορεί κυρίως να χρησιμοποιήσουν δισδιάστατες τεχνικές απεικόνισης.

Τα τρισδιάστατα γραφικά είναι παρόμοια με τα τρισδιάστατα μοντέλα. Τα μοντέλα περιέχονται μέσα σε αρχεία γραφικών δεδομένων. Το τρισδιάστατο μοντέλο είναι η αντιπροσώπευση οποιουδήποτε τρισδιάστατου αντικειμένου. Μέχρι τη στιγμή της απεικόνισής του ένα μοντέλο δεν θεωρείται γραφικό. Λόγω της τρισδιάστατης εκτύπωσης τα τρισδιάστατα μοντέλα δεν περιορίζονται μόνο στον εικονικό χώρο. Τέλος, μπορούν να χρησιμοποιηθούν και σε μη γραφικές εξομοιώσεις και σε διάφορους υπολογισμούς.

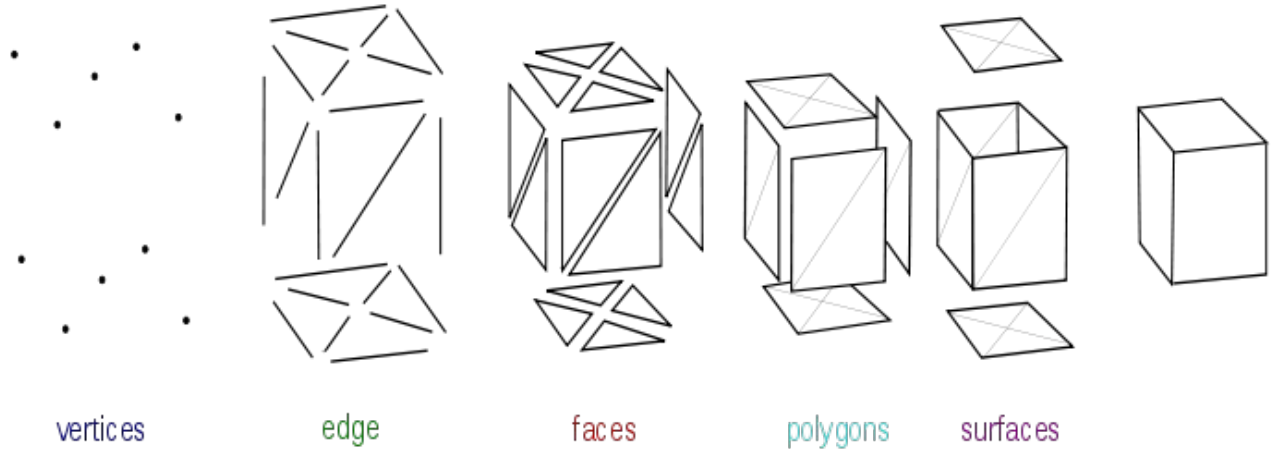
Πλέγματα Πολυγώνων (Polygon Meshes)

Το πλέγμα πολυγώνων είναι μία συλλογή από κορυφές (vertices), πλευρές (edges) και όψεις (faces), η οποία ορίζει το σχήμα ενός πολύεδρου αντικειμένου στα γραφικά ηλεκτρονικών υπολογιστών και στον μοντελισμό στερεών αντικειμένων.



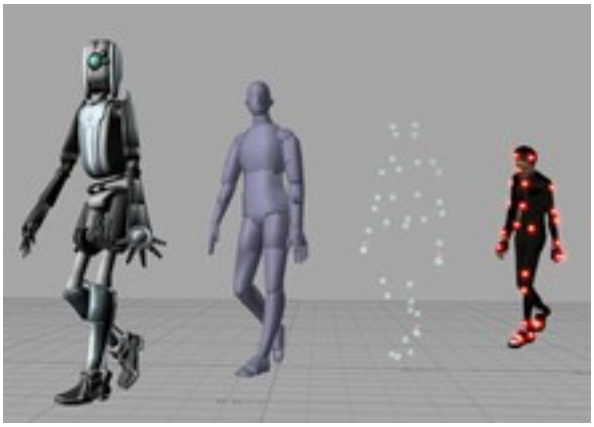
Οι όψεις συνήθως αποτελούνται από

τρίγωνα, τετράπλευρα ή άλλα απλά κυρτά πολύγωνα, μιας και απλοποιεί το rendering.



Computer Animation

Το computer animation είναι η τέχνη της δημιουργίας κινούμενων εικόνων χρησιμοποιώντας ηλεκτρονικούς υπολογιστές. Η χρήση τρισδιάστατων γραφικών για την δημιουργία animation γίνεται όλο και μεγαλύτερη, παρ' όλα αυτά χρησιμοποιούνται ακόμα δισδιάστατα γραφικά για στιλιστικούς λόγους και για για λόγους καλύτερης απόδοσης σε εφαρμογές rendering πραγματικού χρόνου.



Μερικές φορές τα animation χρησιμοποιούνται στους ηλεκτρονικούς υπολογιστές, μερικές φορές όμως χρησιμοποιούνται και σε άλλα μέσα, όπως

οι ταινίες. Το animation είναι επίσης γνωστό ως CGI (Computer-Generated Imagery or Computer-Generated imaging), ειδικά όταν χρησιμοποιείται σε ταινίες.

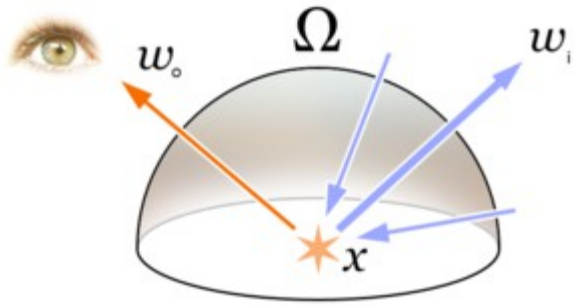
Οι εικονικές οντότητες που χρησιμοποιούνται στο animation μπορεί να περιέχουν ή να ελέγχονται από διάφορα γνωρίσματα (attributes), όπως τιμές μετασχηματισμών (location, orientation, scale) αποθηκευμένα μέσα στον πίνακα μετασχηματισμού του αντικειμένου. Το animation είναι η αλλαγή ενός γνωρίσματος του αντικειμένου στον χρόνο. Υπάρχουν πολλοί τρόποι για να επιτευχθεί αυτό. Ο ποιό στοιχειώδης είναι βασισμένος στη δημιουργία και την επεξεργασία των keyframes, από τα οποία το καθ' ένα αποθηκεύει μία τιμή σε μία χρονική στιγμή ανά γνώρισμα που θα μεταβληθεί. Τα λογισμικά επεξεργασίας και δημιουργίας γραφικών (ή και μία δικιά σας game engine) δημιουργούν ενδιάμεσες τιμές των γνωρισμάτων ανάλογα με τον χρόνο χρησιμοποιώντας διάφορες τεχνικές μαθηματικής παρεμβολής (interpolation). Για να δημιουργηθεί η ψευδαίσθηση της κίνησης, προβάλλονται στην οθόνη διαφορετικά στιγμιότυπα της εικόνας με μεγάλη ταχύτητα. Αυτή η τεχνική είναι ακριβώς η ίδια με την ψευδαίσθηση της κίνησης που δημιουργεί η τηλεόραση ή τα κινούμενα σχέδια.

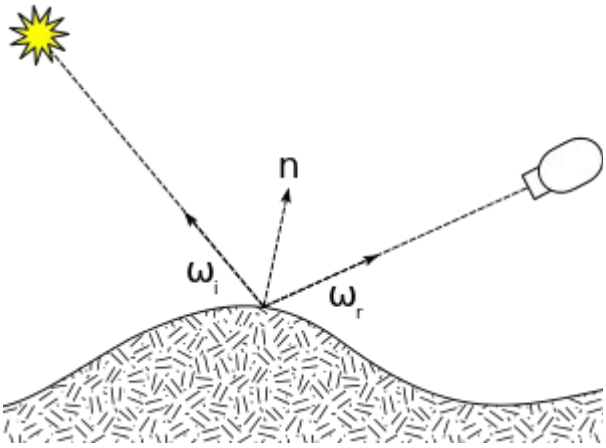
Rendering

Το rendering είναι η διαδικασία απεικόνισης ενός τρισδιάστατου ή δισδιάστατου μοντέλου χρησιμοποιώντας προγράμματα υπολογιστών. Ένα αρχείο σκηνής περιέχει αντικείμενα που ορίζονται σε μία αυστηρά ορισμένη γλώσσα ή δομή. Περιέχει δηλαδή πληροφορίες για τη γεωμετρία, την οπτική γωνία, τα textures, το φωτισμό, τη σκίαση κ.ο.κ της σκηνής. Τα δεδομένα αυτά έπειτα στέλνονται σε ένα πρόγραμμα rendering για να υποστούν επεξεργασία και να εξαχθεί το αποτέλεσμα σε μία ψηφιακή εικόνα η σε μία εικόνα raster (bitmap). Το πρόγραμμα rendering είναι συνήθως χτισμένο πάνω σε κάποιο λογισμικό γραφικών. Παρ' ότι το γεγονός ότι οι τεχνικές λεπτομέρειες των μεθόδων rendering διαφέρουν μεταξύ τους, η γενική

πρόκληση που τίθεται είναι η παραγωγή μιας δισδιάστατης εικόνας από μία τρισδιάστατη αντιπροσώπευση μίας σκηνής αποθηκευμένης σε ένα αρχείο. Η διαδικασία αντιμετώπισης αυτής της πρόκλησης αναφέρεται συνήθως με το όνομα "αγωγός γραφικών" (Graphics Pipeline) και επιτελείται με τη βοήθεια μίας συσκευής rendering, όπως μία GPU (Κάρτα γραφικών).

Η GPU είναι μία συσκευή που έχει την ικανότητα να βοηθήσει την CPU (Κεντρική Μονάδα Επεξεργασίας) στην εκτέλεση υπολογισμών. Αν θέλουμε μία σκηνή να φανεί σχετικά ρεαλιστική χρησιμοποιώντας εικονικό φωτισμό, το λογισμικό rendering πρέπει να επιλύσει την εξίσωση του rendering.



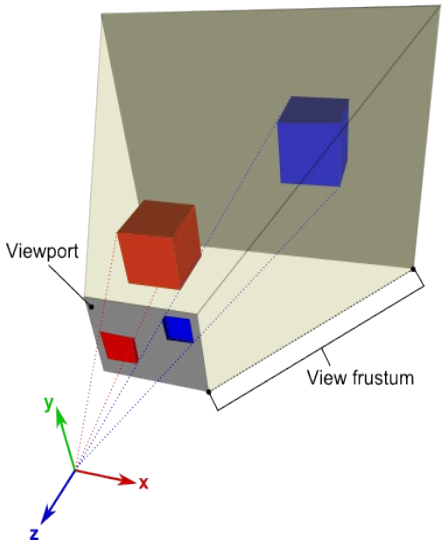


Η εξίσωση του rendering περιγράφει τη συνολική ποσότητα φωτός που εκπέμπεται από ένα σημείο x προς μία συγκεκριμένη οπτική γωνία χρησιμοποιώντας μία συνάρτηση για το εισερχόμενο φως και μια συνάρτηση κατανομής αμφίδρομης ανακλαστικότητας (Bidirectional Reflectance Distribution Function).

Η εξίσωση του rendering δεν υπολογίζει όλα τα φυσικά φαινόμενα του φωτισμού, αλλά είναι ένα γενικό μοντέλο φωτισμού για computer-generated εικόνες. Ο όρος "rendering" χρησιμοποιείται επίσης για να περιγράψει την διαδικασία του υπολογισμού των εφέ σε αρχεία επεξεργασίας βίντεο, για να παραχθεί το τελικό αποτέλεσμα.

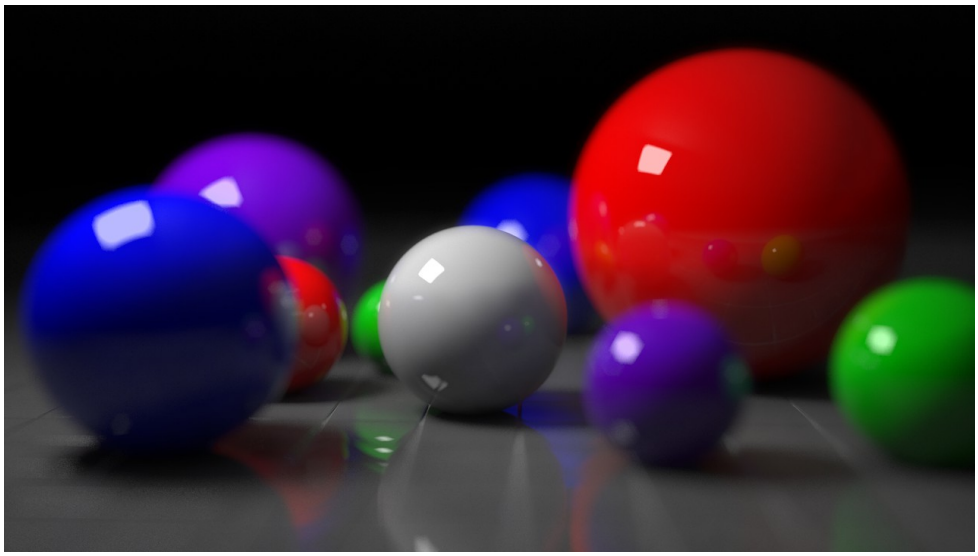
Τρισδιάστατη προβολή (3D Projection)

Η τρισδιάστατη προβολή είναι μία μέθοδος χαρτογράφησης τρισδιάστατων σημείων σε ένα δισδιάστατο επίπεδο. Οι περισσότερες πρόσφατες μέθοδοι που χρησιμοποιούνται για προβολή γραφικών δεδομένων είναι βασισμένες στη χρήση δισδιάστατων μέσων απεικόνισης. Γι' αυτό το λόγο η χρήση αυτής της μεθόδου είναι ιδιαίτερα δημοφιλής, ειδικά στα γραφικά ηλεκτρονικών υπολογιστών.

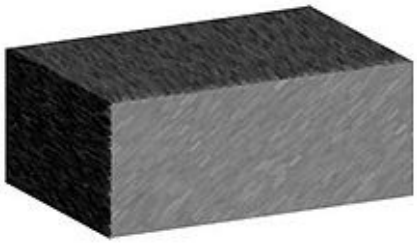


Ray Tracing

Το ray tracing είναι μία τεχνική δημιουργίας εικόνων ακολουθώντας τη διαδρομή του φωτός μέσα από τα pixels στο επίπεδο μίας εικόνας. Η τεχνική αυτή είναι ικανή να παράγει έναν πολύ υψηλό βαθμό φωτορεαλισμού αλλά με μεγάλο υπολογιστικό κόστος.

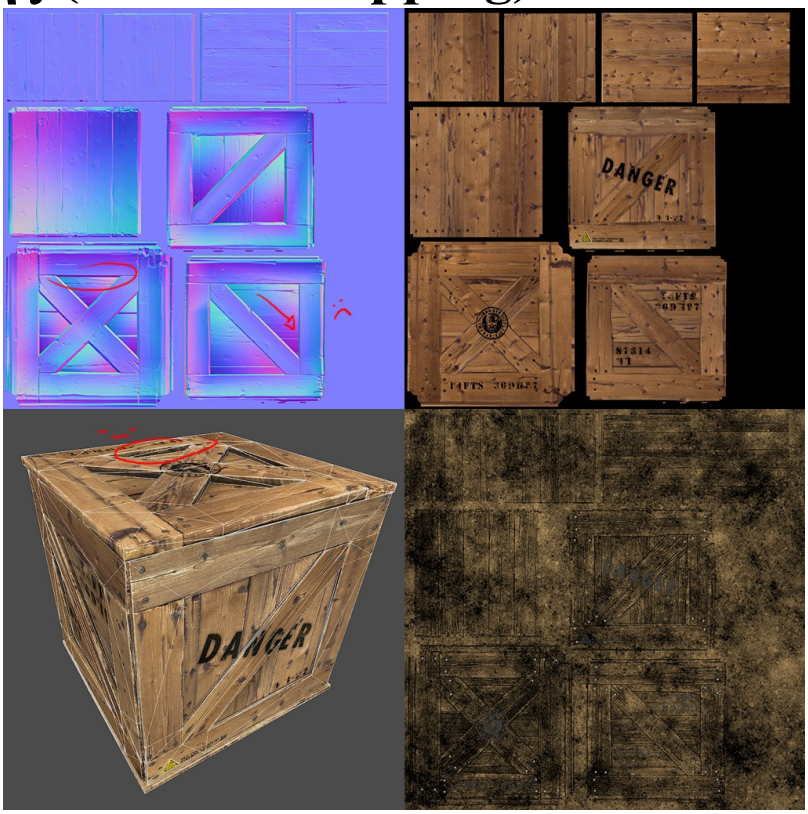


Σκίαση (Shading)



Ο όρος σκίαση αναφέρεται στην απεικόνιση του βάθους των τρισδιάστατων μοντέλων ή εικόνων μεταβάλλοντας τα επίπεδα της σκοτεινότητας. Είναι μία διαδικασία που χρησιμοποιείται στη ζωγραφική για την απεικόνιση των επιπέδων της σκοτεινότητας. Στα γραφικά ηλεκτρονικών υπολογιστών χρησιμοποιούνται ειδικοί αλγόριθμοι με τους οποίους επιτυγχάνεται αυτό το αποτέλεσμα. Κάποιοι από αυτούς τους αλγορίθμους είναι ο αλγόριθμος Flat Shading (Επίπεδης Σκίασης), ο αλγόριθμος του Gouraud, ο αλγόριθμος του Phong κ.α.

Χαρτογράφηση Υφής (Texture Mapping)



Η χαρτογράφηση υφής είναι μία μέθοδος για πρόσδοση ρεαλισμού, υφής η χρώματος στις επιφάνειες ενός τρισδιάστατου μοντέλου. Η εφαρμογή της τεχνικής αυτής στα γραφικά ηλεκτρονικών υπολογιστών εφευρέθηκε και εφαρμόστηκε από τον Dr. Edwin Catmull το 1974. Ένας χάρτης υφής εφαρμόζεται στην επιφάνεια ενός σχήματος

ή πολυγώνου. Η εφαρμογή πολλών τέτοιων υφών στις επιφάνειες ονομάζεται Mutlitexturing.

Anti-aliasing (AA)

Η οπτικοποίηση οντοτήτων που είναι ανεξάρτητες από την ανάλυση (όπως τα τρισδιάστατα μοντέλα) πάνω σε μία pixel-based συσκευή όπως μία LCD οθόνη αναπόφευκτα δημιουργεί aliasing artifacts, κυρίως στις άκρες της γεωμετρίας και στα άκρα των texture maps. Οι μέθοδοι που εφαρμόζουν anti-aliasing επιλύουν αυτά τα προβλήματα, έχοντας ως αποτέλεσμα εικόνες πιο ευχάριστες στην όψη για το θεατή, αλλά μπορεί να είναι πιο "ακριβές" υπολογιστικά.

OpenGL (Open Graphics Library)

Το OpenGL είναι μία διαγλωσσική, multi-platform διεπαφή προγραμματισμού εφαρμογών (Application Programming Interaface – API) που χρησιμοποιείται για οπτικοποίηση δισδιάστατων και τρισδιάστατων διανυσματικών γραφικών. Το API τυπικά χρησιμοποιείται για αλληλεπίδραση με μία GPU, ώστε να επιτευχθεί hardware-accelerated οπτικοποίηση.

Για τον προγραμματιστή το OpenGL είναι ένα σύνολο εντολών που επιτρέπουν τον προσδιορισμό των προγραμμάτων shader (ή για συντομία shaders), των δεδομένων που χρησιμοποιούνται από τους shaders και τις ρυθμιστικές πτυχές του OpenGL εκτός του πεδίου εφαρμογής των shaders. Συνήθως τα δεδομένα αντιπροσωπεύουν γεωμετρία σε δύο ή τρεις διαστάσεις και εικόνες υφής (textures), ενώ οι shaders ελέγχουν τη γεωμετρική μεταποίηση, το rasterization της γεωμετρίας, τον φωτισμό και τη σκίαση των fragments (pixels) που δημιουργούνται από το rasterization, με αποτέλεσμα να γίνεται απεικόνιση της γεωμετρίας στον framebuffer.

Ένα τυπικό πρόγραμμα που χρησιμοποιεί OpenGL αρχίζει με τις κλήσεις για να ανοίξει ένα παράθυρο στον frambuffer, στον οποίο το πρόγραμμα θα ζωγραφίσει. Στη συνέχεια, γίνονται κλήσεις για να δημιουργηθεί το πλαίσιο του OpenGL , κλήσεις για να προσδιοριστούν οι shaders, η γεωμετρία και τα textures. Τις κλήσεις αυτές ακολουθούν εντολές οι οποίες ζωγραφίζουν τη γεωμετρία μεταφέροντας συγκεκριμένα κομμάτια της στους shaders. Οι εντολές που χρησιμοποιούνται για να ζωγραφίζουν γεωμετρία προσδιορίζουν απλά γεωμετρικά αντικείμενα όπως σημεία, ευθύγραμμα τμήματα και πολύγωνα, τα οποία μπορούν να επεξεργαστούν περαιτέρω οι shaders. Υπάρχουν επίσης εντολές οι οποίες μπορούν να ελέγχουν άμεσα τον framebuffer διαβάζοντας και γράφοντας pixels.

Το OpenGL ως μηχανή καταστάσεων (state machine).

Το OpenGL είναι μία μηχανή καταστάσεων. Μπορεί κανείς να θέσει στη μηχανή αυτή διάφορες καταστάσεις οι οποίες παραμένουν ενεργές μέχρι να αλλαχθούν από τον προγραμματιστή. Ένα παράδειγμα μίας μεταβλητής (variable) καταστάσεως στο OpenGL είναι το τρέχον χρώμα. Μπορούμε να θέσουμε το τρέχον χρώμα σε κόκκινο, άσπρο ή οποιοδήποτε άλλο χρώμα, και από κει και μετά κάθε αντικείμενο ζωγραφίζεται με το χρώμα που θέσαμε στο OpenGL μέχρι να θέσουμε ένα διαφορετικό τρέχον χρώμα. Το τρέχον χρώμα είναι μία μόνο από τις μεταβλητές καταστάσεων που περιέχει το OpenGL. Άλλες ελέγχουν τους τρέχοντες μετασχηματισμούς view και projection, τα μοτίβα των γραμμών και των πολυγώνων, τη θέση και τα χαρακτηριστικά του φωτός, τις ιδιότητες των υλικών των αντικειμένων που ζωγραφίζονται κ.α. Πολλές μεταβλητές καταστάσεων αναφέρονται σε καταστάσεις οι οποίες μπορούν να ενεργοποιηθούν ή να απενεργοποιηθούν με τις εντολές glEnable() ή glDisable().

Κάθε μεταβλητή καταστάσεως έχει μία προκαθορισμένη τιμή, και σε οποιαδήποτε χρονική στιγμή μπορούμε να ζητήσουμε από το σύστημα να μας δώσει την τρέχουσα τιμή της. Τυπικά χρησιμοποιούμε μία από τις έξι εντολές που ακολουθούν για το κάνουμε αυτό: glGetBooleanv(), glGetDoublev(), glGetFloatv(), glGetIntegerv, glGetPointerv ή gllsEnabled(). Το ποια από τις παραπάνω εντολές θα χρησιμοποιήσουμε εξαρτάται από τον τύπο των δεδομένων που θέλουμε να μας επιστρέψει το σύστημα. Επιπρόσθετα μπορούμε να σώσουμε μία συλλογή από μεταβλητές καταστάσεως σε μία στοίβα γνωρισμάτων (attribute stack), με τις εντολές glPushAttrib() ή glPushClientAttrib(), να τις τροποποιήσουμε προσωρινά, και αργότερα να επαναφέρουμε τις τιμές τους χρησιμοποιώντας τις αντίστοιχες εντολές glPopAttrib() ή glPopClientAttrib().

Πίνακες Μετασχηματισμών (Transformation Matrices)

Στα γραφικά των ηλεκτρονικών υπολογιστών η χρήση πινάκων μετασχηματισμού είναι ιδιαίτερα ευρεία. Αυτό συμβαίνει κυρίως γιατί είναι εύκολη η υλοποίησή τους προγραμματιστικά και γιατί η υπολογιστική ισχύς που χρειάζεται δεν είναι ιδιαίτερα μεγάλη. Επίσης είναι πιο εύκολα κατανοητή σαν μέθοδος, σε σύγκριση με άλλες μεθόδους.

Οι πίνακες μετασχηματισμού έχουν πολλές χρήσεις, αλλά εμείς θα αναφερθούμε στους μετασχηματισμούς μετακίνησης (translation), στους μετασχηματισμούς περιστροφής (rotation) και στους μετασχηματισμούς κλίμακας (scaling).

Πριν αναφερθούμε όμως ειδικότερα στους πίνακες μετασχηματισμού είναι απαραίτητο να αναφερθούμε στις κορυφές (vertices).

Οι κορυφές δεν είναι τίποτα άλλο από τρισδιάστατα διανύσματα. Τα διανύσματα τριών διαστάσεων όπως είναι γνωστό αποτελούνται από μία τριπλέτα αριθμών (x, y, z). Στα γραφικά όμως εισάγουμε και έναν ακόμα αριθμό στο διάνυσμα αυτό, το w, οπότε το διάνυσμά μας πλέον παίρνει τη μορφή (x, y, z, w).

Αυτό θα γίνει πιο ξεκάθαρο παρακάτω, αλλά για τώρα να θυμάστε αυτό:

- Αν το w = 1 τότε το διάνυσμα (x, y, z, 1) είναι ένα σημείο στο χώρο.
- Αν το w = 0 τότε το διάνυσμα (x, y, z, 0) είναι μία κατεύθυνση.

Τι μας προσφέρει λοιπόν αυτό το w;

Για μία περιστροφή το w δεν προσφέρει τίποτα. Είτε το διάνυσμα είναι σημείο του χώρου, είτε δηλώνει κατεύθυνση, παίρνουμε το ίδιο αποτέλεσμα. Παρ' όλα αυτά , για μία μετακίνηση (όπου μετακινούμε ένα σημείο σε μία συγκεκριμένη κατεύθυνση), τα πράγματα είναι διαφορετικά. Τι θα σήμαινε “Μετακίνησε μία κατεύθυνση”; Όχι και πολλά!!!

Στα τρισδιάστατα γραφικά χρησιμοποιούμε πίνακες 4x4. Μας επιτρέπουν να μετασχηματίσουμε τα (x, y, z, w) διανύσματά μας. Αυτό επιτυγχάνεται πολλαπλασιάζοντας το διάνυσμα με τον πίνακα:

Πίνακας * Διάνυσμα (με αυτή τη σειρά!!!) = Μετασχηματισμένο Διάνυσμα

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} ax + by + cz + dw \\ ex + fy + gz + hw \\ ix + jy + kz + lw \\ mx + ny + oz + pw \end{bmatrix}$$

Πίνακες μετασχηματισμού μετακίνησης (Translation Matrices)

Αυτοί είναι οι πιο απλοί στην κατανόησή τους πίνακες μετασχηματισμού. Ένας πίνακας μετακίνησης έχει την εξής μορφή:

$$\begin{bmatrix} 1 & 0 & 0 & X \\ 0 & 1 & 0 & Y \\ 0 & 0 & 1 & Z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Όπου X, Y, Z, οι τιμές που θέλουμε να προσθέσουμε στην υπάρχουσα θέση. Οπότε αν θέλαμε να μετακινήσουμε το διάνυσμα (10, 10, 10, 1), 10 μονάδες στον άξονα X θα κάναμε το εξής:

$$\begin{bmatrix} 1 & 0 & 0 & 10 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 10 \\ 10 \\ 10 \\ 1 \end{bmatrix} = \begin{bmatrix} 1*10 + 0*10 + 0*10 + 10*1 \\ 0*10 + 1*10 + 0*10 + 0*1 \\ 0*10 + 0*10 + 1*10 + 0*1 \\ 0*10 + 0*10 + 0*10 + 1*1 \end{bmatrix} = \begin{bmatrix} 10 + 0 + 0 + 10 \\ 0 + 10 + 0 + 0 \\ 0 + 0 + 10 + 0 \\ 0 + 0 + 0 + 1 \end{bmatrix} = \begin{bmatrix} 20 \\ 10 \\ 10 \\ 1 \end{bmatrix}$$

Ο μοναδιαίος πίνακας μετασχηματισμού (Identity matrix)

Ο μοναδιαίος πίνακας μετασχηματισμού είναι μία ειδική περίπτωση! Δεν κάνει απολύτως τίποτα! Αλλά είναι ιδιαίτερα σημαντικός, όσο

σημαντικό είναι να ξέρει κανείς ότι αν πολλαπλασιάσουμε το A με το 1 θα πάρουμε A.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} 1*x+0*y+0*z+0*w \\ 0*x+1*y+0*z+0*w \\ 0*x+0*y+1*z+0*w \\ 0*x+0*y+0*z+1*w \end{bmatrix} = \begin{bmatrix} x+0+0+0 \\ 0+y+0+0 \\ 0+0+z+0 \\ 0+0+0+w \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Η πραγματική χρησιμότητα του μοναδιαίου πίνακα μετασχηματισμού θα φανεί στο παράδειγμα του προγράμματος που ακολουθεί παρακάτω.

Πίνακες μετασχηματισμού κλίμακας

Οι πίνακες μετασχηματισμού κλίμακας χρησιμοποιούνται για να αυξήσουμε την κλίμακα ενός διανύσματος (μέγεθος ή μέτρο).

$$\begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Οπότε αν θελήσουμε να αυξήσουμε την κλίμακα ενός διανύσματος (θέσης ή διεύθυνσης) κατά 2.0 προς όλες τις κατευθύνσεις, θα κάναμε το εξής:

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} 2*x+0*y+0*z+0*w \\ 0*x+2*y+0*z+0*w \\ 0*x+0*y+2*z+0*w \\ 0*x+0*y+0*z+1*w \end{bmatrix} = \begin{bmatrix} 2*x+0+0+0 \\ 0+2*y+0+0 \\ 0+0+2*z+0 \\ 0+0+0+1*w \end{bmatrix} = \begin{bmatrix} 2*x \\ 2*y \\ 2*z \\ w \end{bmatrix}$$

Όπως παρατηρείτε το w δεν έχει αλλάξει. Τώρα μπορεί να σας δημιουργηθεί η απορία: Ποια είναι η σημασία της “αύξησης της κλίμακας μίας κατεύθυνσης” ;

Η απάντηση θα ήταν: “Συνήθως καμία”, και γι' αυτό συνήθως δεν κάνουμε τέτοιο μετασχηματισμό. Υπάρχουν όμως μερικές σπάνιες

περιπτώσεις που μπορεί να φανεί χρήσιμος.

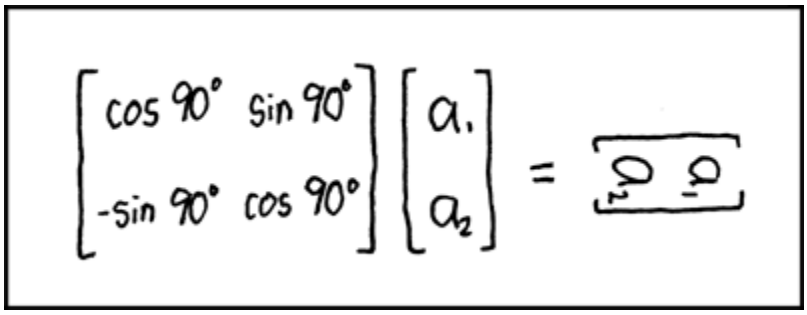
- Παρατηρείστε ότι ο μοναδιαίος πίνακας είναι μία ειδική περίπτωση των πινάκων μετασχηματισμού κλίμακας, με τα X, Y, Z = (1, 1, 1). Είναι επίσης μία ειδική περίπτωση των πινάκων μετασχηματισμού μετακίνησης όπου X, Y, Z = (0, 0, 0).

Πίνακες μετασχηματισμού περιστροφής.

Οι πίνακες περιστροφής, όπως υποδηλώνει και η ονομασία τους, χρησιμοποιούνται για την περιστροφή ενός σετ σημείων που ανήκουν σε ένα σύστημα συντεταγμένων. Ενώ το κάθε σημείο αποκτά τις καινούριες του συντεταγμένες, οι σχετικές τους αποστάσεις δεν αλλάζουν.

Όλες οι περιστροφές ορίζονται χρησιμοποιώντας τις τριγωνομετρικές συναρτήσεις του ημιτόνου (sine) και του συνημιτόνου (cosine).

Για ένα δισδιάστατο σύστημα συντεταγμένων ο πίνακας περιστροφής είναι κάπως έτσι, και ουσιαστικά επιτελεί τη λειτουργία που χιουμοριστικά απεικονίζεται στην παρακάτω εικόνα:



Ένωση πινάκων μετασχηματισμού (Matrix Concatenation)

Τώρα που πλέον γνωρίζουμε πως να περιστρέφουμε, να μετακινούμε και να αλλάζουμε την κλίμακα των διανυσμάτων μας, θα ήταν πολύ καλό να μπορούσαμε να συνδυάσουμε αυτούς τους μετασχηματισμούς. Αυτό επιτυγχάνεται πολλαπλασιάζοντας τους πίνακες μετασχηματισμού μεταξύ τους:

```
TransformedVector = TranslationMatrix * RotationMatrix * ScaleMatrix * OriginalVector;
```

ΠΡΟΣΟΧΗ: Η παραπάνω γραμμή πραγματοποιεί πρώτα την αλλαγή κλίμακας, έπειτα την περιστροφή και τελευταία τη μετακίνηση διότι έτσι δουλεύει ο πολλαπλασιασμός πινάκων.

Κάνοντας τις πράξεις με άλλη σειρά δεν θα είχε το ίδιο αποτέλεσμα. Δοκιμάστε το και μόνοι σας:

- Κάντε ένα βήμα μπροστά και μετά στρίψτε αριστερά.
- Στρίψτε αριστερά και μετά κάντε ένα βήμα μπροστά.

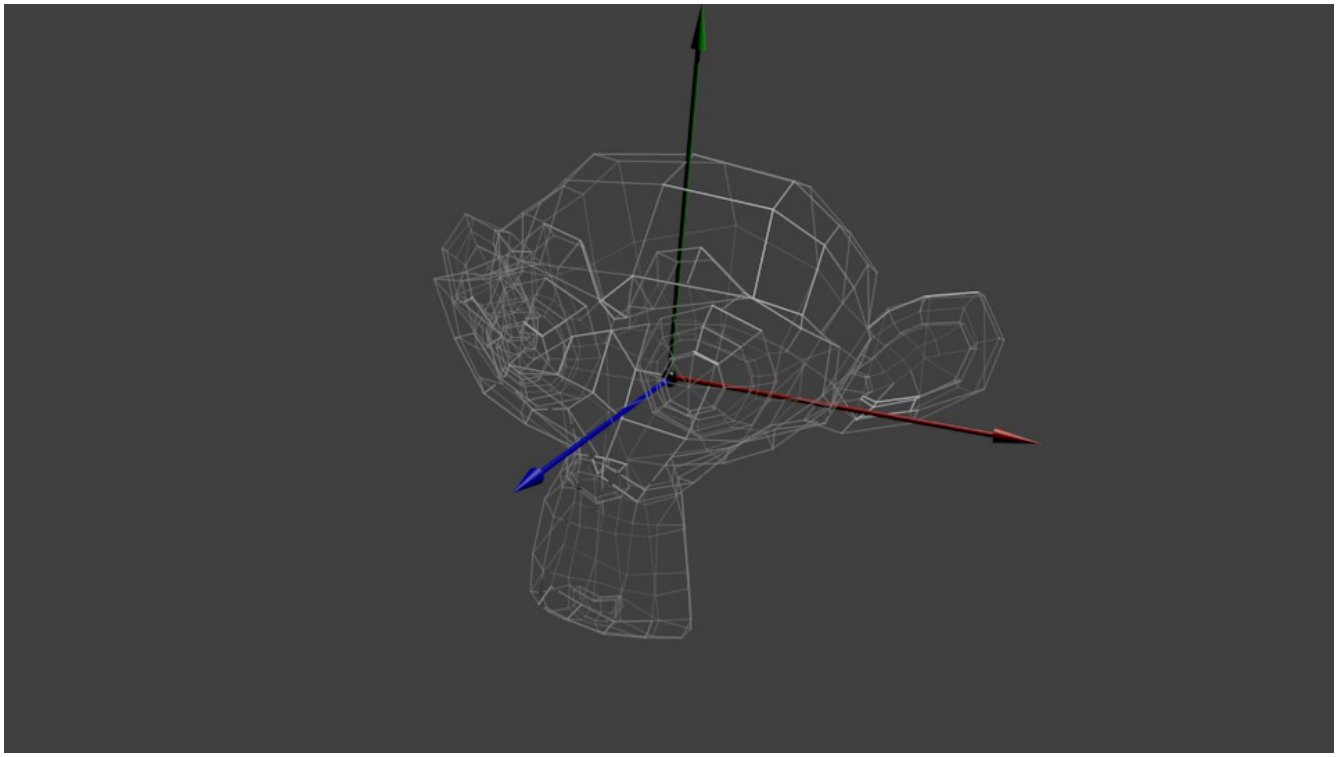
Η πραγματικότητα είναι ότι για ένα χαρακτήρα ή άλλα αντικείμενα σε ένα video game θα χρειαζόμασταν τη δεύτερη επιλογή, δηλαδή:

- Αλλαγή κλίμακας πρώτα και αν χρειάζεται. Στη συνέχεια περιστροφή για να αλλάξουμε την κατεύθυνση. Τέλος μετακίνηση.

Μετασχηματισμοί στο Fixed Function Pipeline του OpenGL.

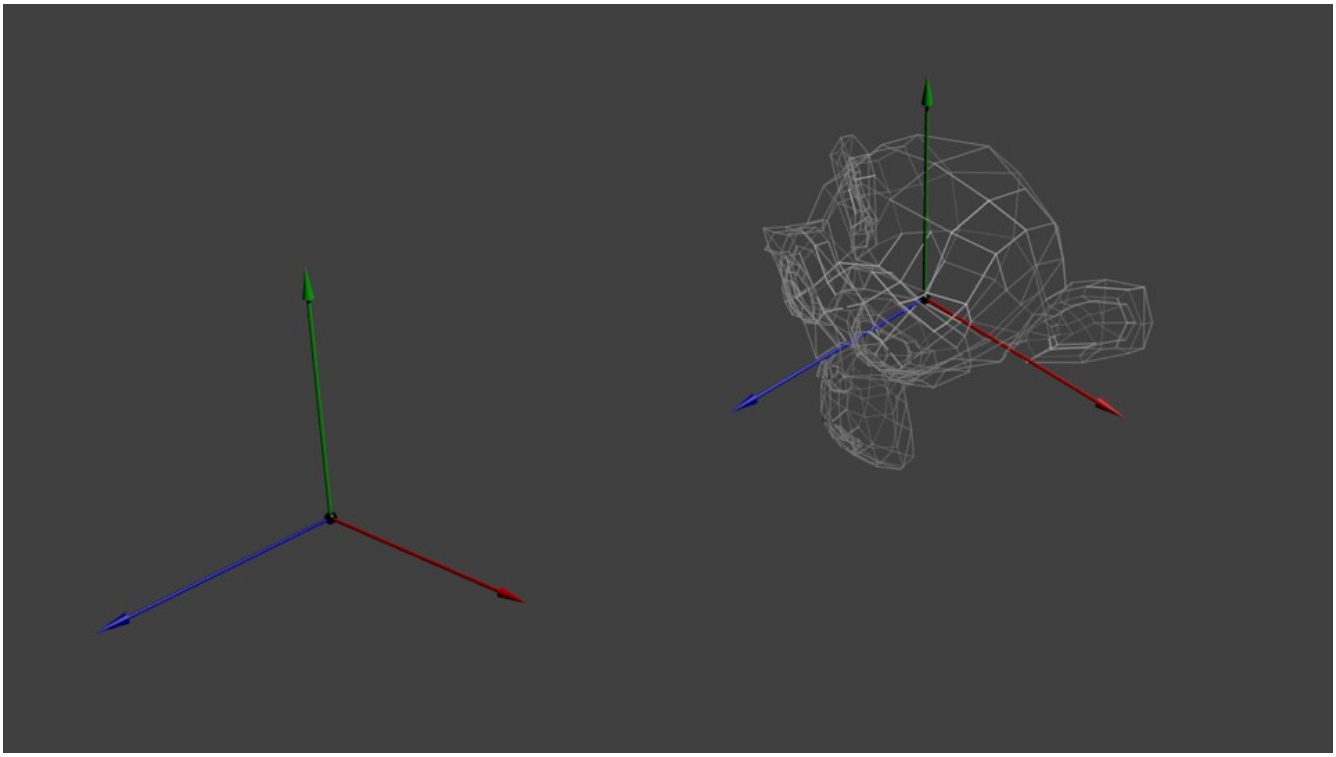
Το OpenGL αν δεν οριστούν shaders χρησιμοποιεί μία προκαθορισμένη διαδικασία επεξεργασίας των vertices και των μετασχηματισμών τους. Στη συνέχεια θα αναφερθούμε στους μετασχηματισμούς που πραγματοποιούνται σε ένα τρισδιάστατο μοντέλο για να μπορέσει να απεικονιστεί στην οθόνη χρησιμοποιώντας το fixed function pipeline του OpenGL.

Τα τρισδιάστατα μοντέλα αρχικά ορίζονται σε ένα τοπικό σύστημα συντεταγμένων (local coordinate system).



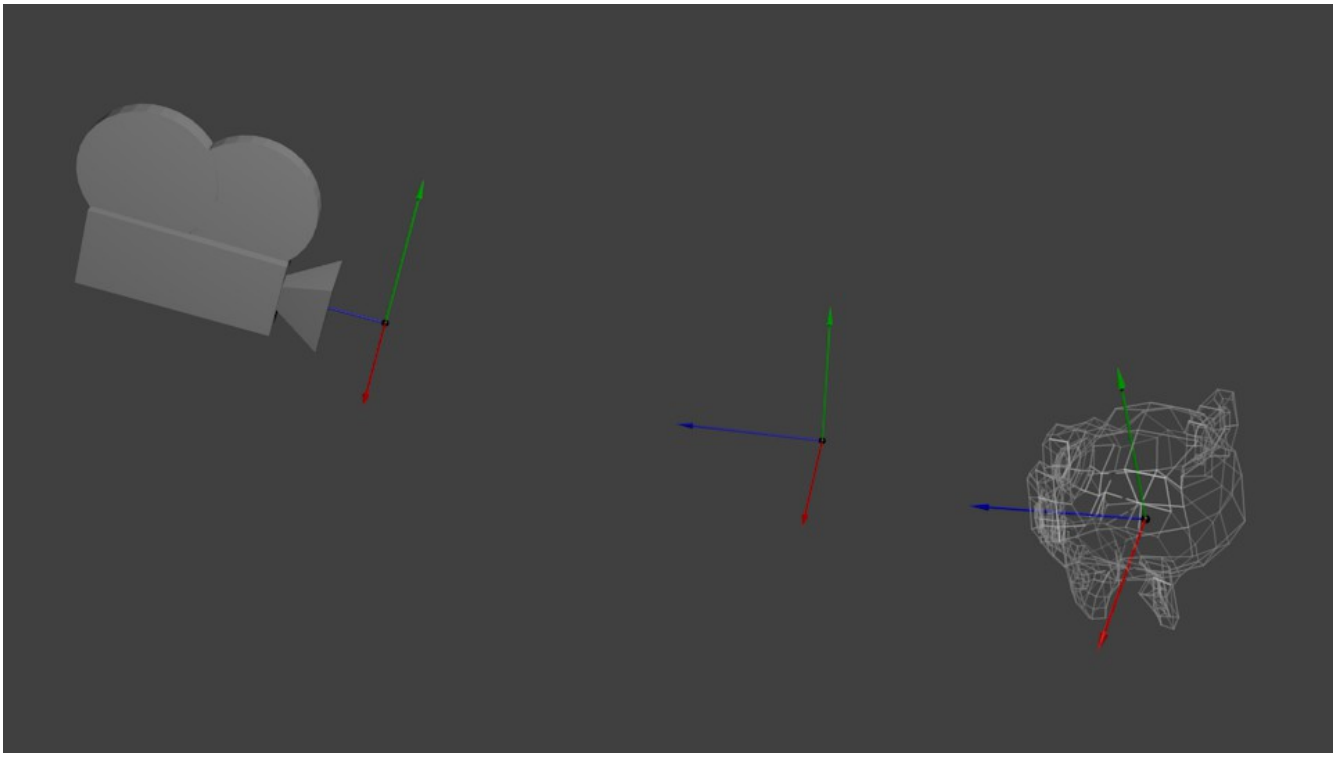
Ο ρόλος του τοπικού συστήματος συντεταγμένων είναι να μας διευκολύνει στον ορισμό των πολυγώνων.

Για να τοποθετηθούν τα αντικείμενα στη σκηνή, όπου το κάθε ένα πρέπει να έχει τη δική του θέση, τη δική του γωνία περιστροφής και τη δική του κλίμακα μέσα στη σκηνή, χρησιμοποιούμε τον world ή modeling μετασχηματισμό, και μεταφέρονται στο world ή global σύστημα συντεταγμένων.



Το κάθε τρισδιάστατο αντικείμενο λοιπόν έχει το δικό του world ή modeling μετασχηματισμό, ο οποίος το τοποθετεί μέσα στη σκηνή.

Έπειτα για να μπορέσει να πραγματοποιηθεί εύκολα η προβολή των τρισδιάστατων αντικειμένων στο επίπεδο της οθόνης, μεταφέρουμε τα αντικείμενα στο view σύστημα συντεταγμένων, στο οποίο αρχικά η "camera" βρίσκεται στην αρχή των αξόνων, δηλαδή το σημείο XYZ: (0, 0, 0), και έχει κατεύθυνση (στο OpenGL) τον άξονα -Z.



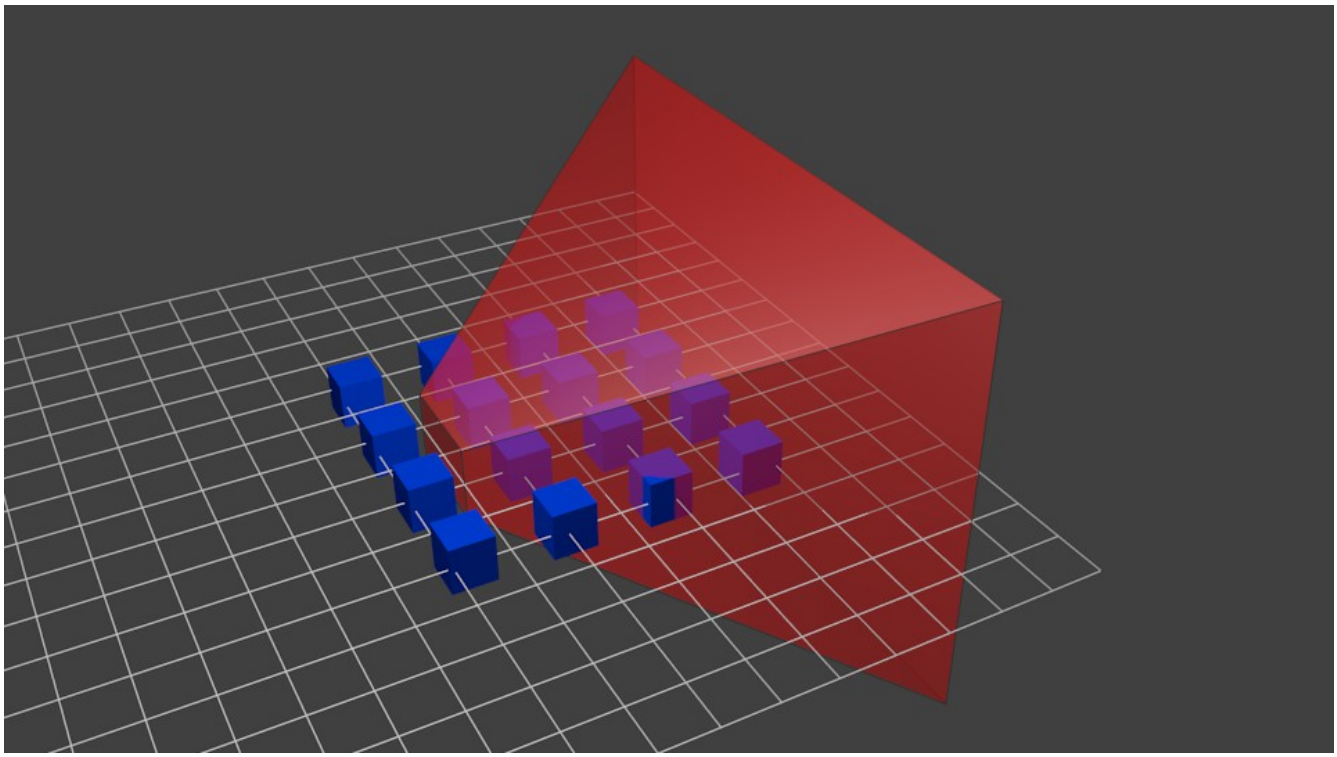
Για όλη τη σκηνή υπάρχει ένας μόνο μετασχηματισμός view που έχει να κάνει με το που βρίσκεται αρχικά ο παρατηρητής.

Στο OpenGL ο model μετασχηματισμός και ο view μετασχηματισμός θεωρείται ένας συνολικός μετασχηματισμός με το όνομα modelview και δημιουργείται ένας συνολικός πίνακας μετασχηματισμού που προκύπτει από τον πολλαπλασιασμό του model με τον view πίνακα μετασχηματισμού.

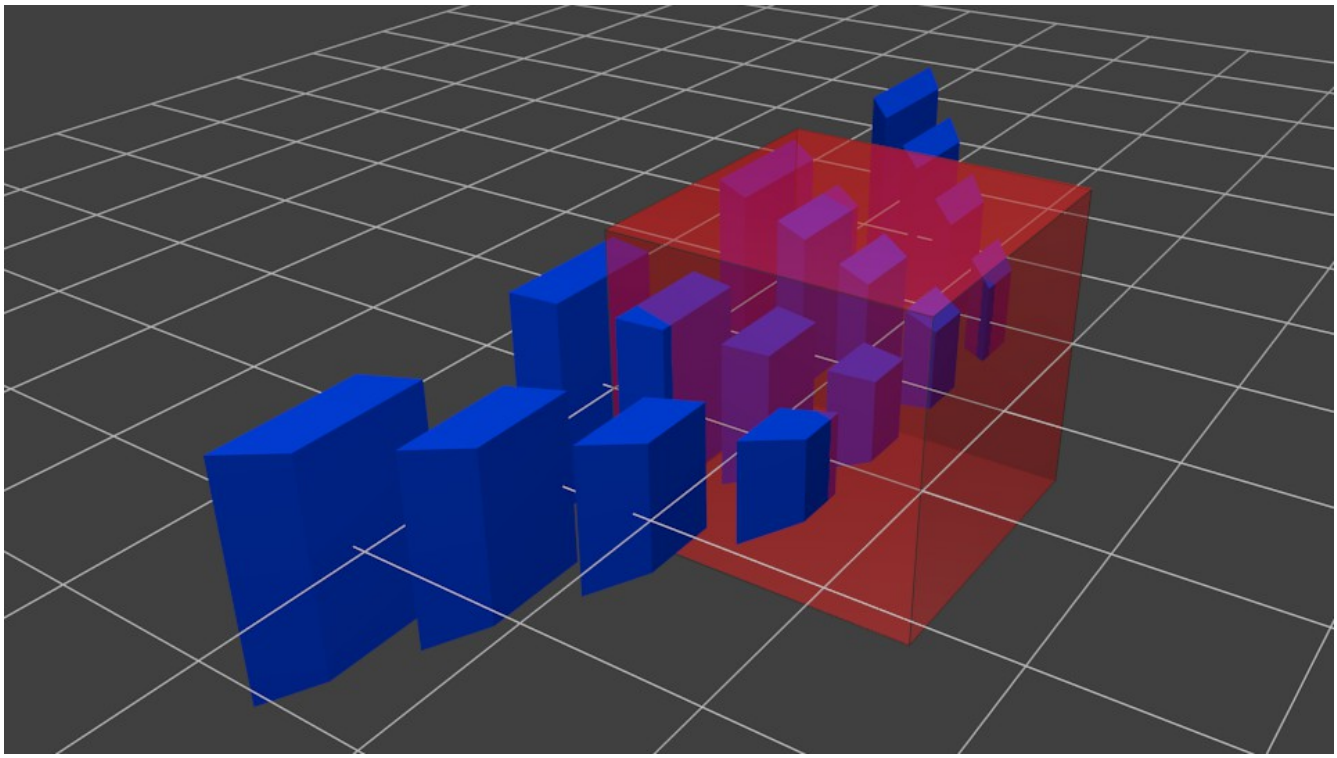
Στη συνέχεια μέσω του μετασχηματισμού projection, μεταφέρουμε τη σκηνή μας σε ένα άλλο σύστημα συντεταγμένων που ονομάζεται homogenous clipspace και οι συντεταγμένες έχουν πλέον τιμές από το -1 έως το 1.

Για να γίνει πιο κατανοητή η διαδικασία του μετασχηματισμού projection ακολουθεί το εξής παράδειγμα:

Έχουμε τα μπλε αυτά αντικείμενα ορισμένα σε view space και το κόκκινο αντιπροσωπεύει το εύρος θέασης του παρατηρητή.

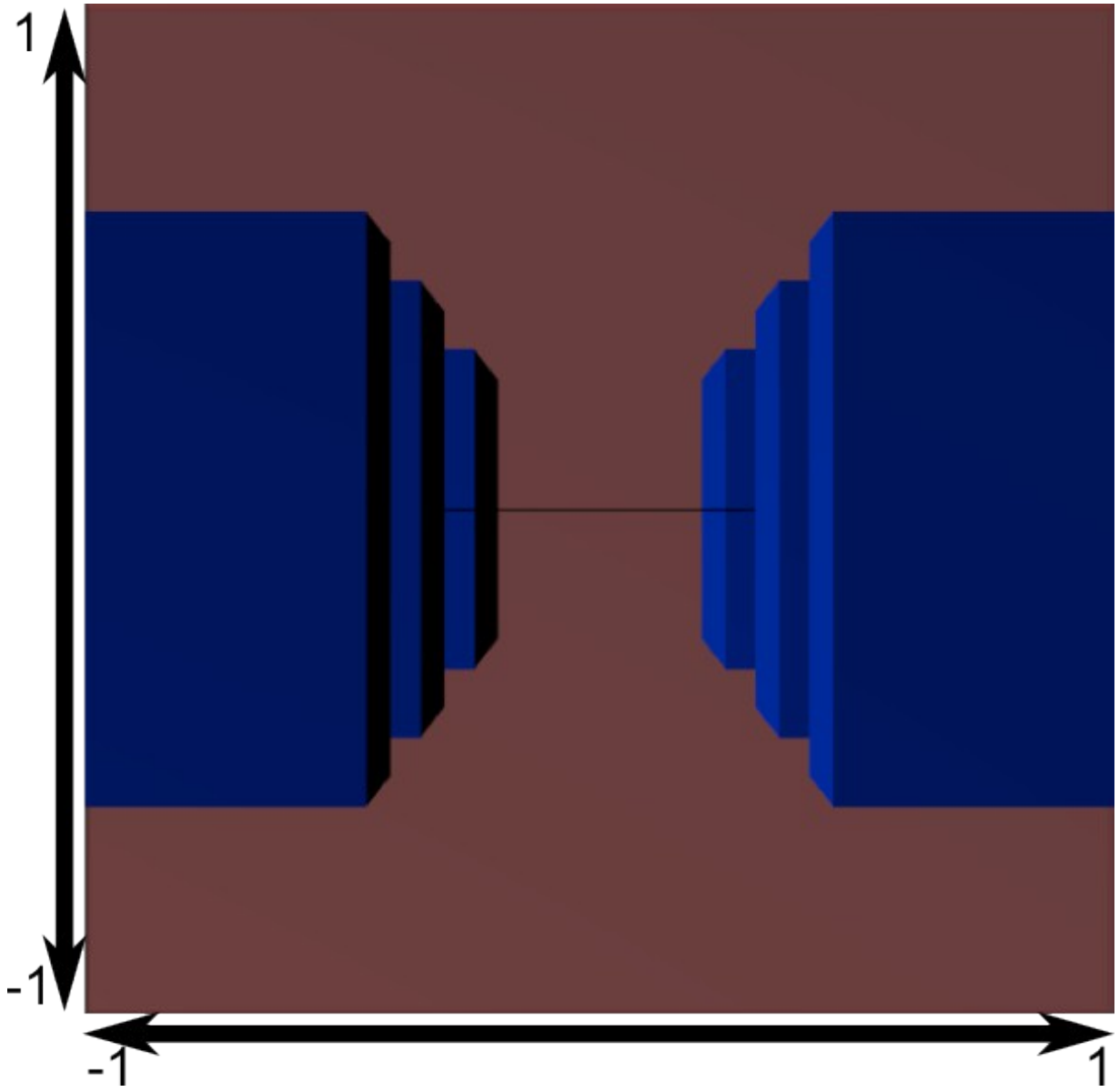


Πολλαπλασιάζοντας τα πάντα με τον πίνακα μετασχηματισμού projection παίρνουμε το εξής αποτέλεσμα:

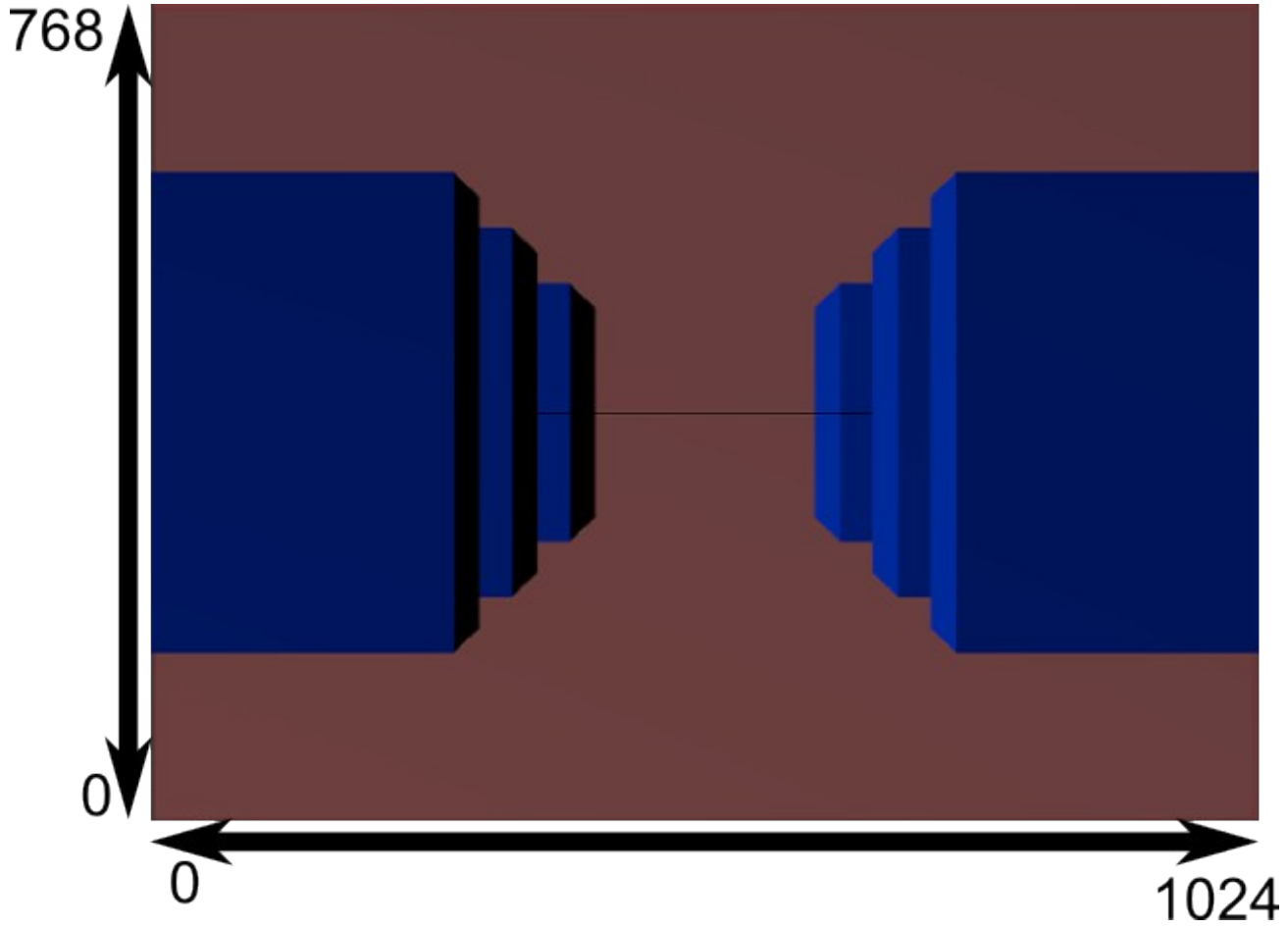


Σ' αυτή την εικόνα, ο κώνος θέασης είναι πλέον ένας τέλειος κύβος (με συντεταγμένες από -1 έως 1 σε όλους τους άξονες), και όλα τα μπλε αντικείμενα έχουν παραμορφωθεί με τον ίδιο τρόπο. Έτσι τα αντικείμενα που είναι κοντά στην camera (δηλαδή στην όψη του κύβου που δεν μπορούμε να δούμε) είναι μεγάλα, τα υπόλοιπα είναι μικρότερα. Μοιάζει με την πραγματικότητα!!!

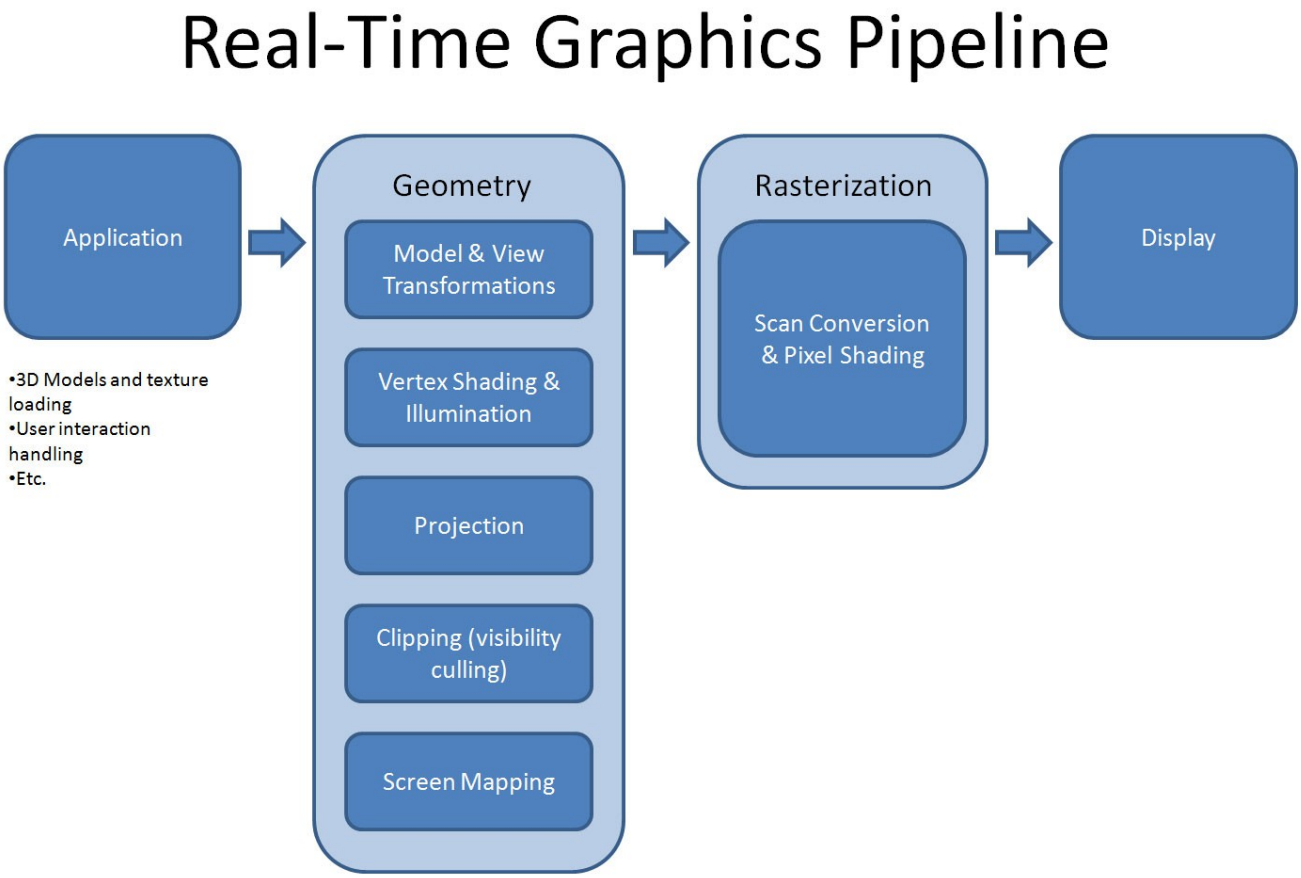
Ας δούμε λοιπόν πως φαίνεται η σκηνή “πίσω” από τον κώνο θέασης.



Στη συνέχεια ακολουθεί ένας τελευταίος μετασχηματισμός που μεταφέρει τη σκηνή από το homogenous clipspace σε screen space και λέγεται μετασχηματισμός viewport.



Συνοψίζοντας, το OpenGL ακολουθεί την εξής διαδικασία για να απεικονίσει μια τρισδιάστατη σκηνή στην οθόνη.



Σχεδιάζοντας μία Game Engine.

Εφόσον αναφέραμε ορισμένες βασικές έννοιες γενικότερα για τα γραφικά και ειδικότερα για το OpenGL, ήρθε η ώρα να σκεφτούμε έναν τρόπο να χρησιμοποιήσουμε αυτά τα εργαλεία για να δημιουργήσουμε το δικό μας video game!

Αρχικά το πρώτο πράγμα που πρέπει να σκεφτούμε πριν ξεκινήσουμε τη σχεδίαση της game engine είναι το τι ακριβώς θέλουμε να πετύχουμε με αυτήν. Έχοντας εξ' αρχής ένα ξεκάθαρο σχέδιο για το τι θέλουμε να είναι ικανή να πραγματοποιήσει η engine μας είναι ιδιαίτερα σημαντικό, πρώτον γιατί μας βοηθάει στο να σχεδιάσουμε τις ανάλογες κλάσεις και δεύτερον γλιτώνουμε ένα μεγάλο μέρος της αλλαγής κώδικα που αναπόφευκτα θα χρειαστεί να κάνουμε, είτε διότι άλλαξε κάτι στην αρχική ιδέα που είχαμε για το πως θα λειτουργεί το game μας, είτε διότι κάναμε κάποιο σχεδιαστικό λάθος το οποίο μας εμποδίζει στην υλοποίηση καινούριων features. Το game development είναι μία δυναμική διαδικασία. Τίποτα δεν είναι χαραγμένο σε πέτρα. Είναι σχεδόν αδύνατον κάνοντας develop ένα video game, να κατασταλάξει το development team σε ένα συγκεκριμένο game design και να μην αλλάξει απολύτως τίποτα μέχρι το τέλος της ανάπτυξής του.

DISCLAIMER: Ότι θα αναφερθεί παρακάτω είναι αντικατόπτριση της έως τώρα εμπειρίας μου με το game engine design και αποτέλεσμα πολλών ορών refactoring κώδικα λόγω των σχεδιαστικών λαθών που έκανα προσπαθώντας να πετύχω ένα καλό αποτέλεσμα.

Ας ξεκινήσουμε λοιπόν από τις ποιο βασικές κλάσεις απ' όλες. Δεν είναι καθόλου δύσκολο να μας έρθει στο μυαλό η δημιουργία αυτών των κλάσεων. Κατασκευάζουμε μία game engine η οποία θέλουμε να επεξεργάζεται δισδιάστατα ή τρισδιάστατα διανυσματικά γραφικά και προφανώς θέλουμε να μπορούμε να τα μετασχηματίζουμε αναλόγως στην οθόνη. Οι κλάσεις αυτές λοιπόν είναι:

- Η κλάση **Vector**.
- Η κλάση **Matrix**.
- Η κλάση **Quaternion**.

Η κλάση Vector θα ορίζει τα διανύσματά μας και θα πραγματοποιεί operations μεταξύ τους, η κλάση Matrix θα ορίζει και θα χειρίζεται τους πίνακες μετασχηματισμού, ενώ η κλάση Quaternion θα ορίζει και θα χειρίζεται operations μεταξύ Quaternions. Αυτές οι κλάσεις μπορούν να χρησιμοποιούνται ανεξάρτητα μέσα στην game engine οπουδήποτε χρειάζεται. Επίσης θα ήταν καλό σχεδιαστικά να ορίσουμε και μία κλάση **Math** η οποία θα υλοποιεί διάφορες αριθμητικές μεθόδους όπως interpolations, splines, noise functions κ.α.

Στη συνέχεια θα ορίσουμε μία κλάση η οποία θα είναι η ραχοκοκαλιά της game engine μας. Η κλάση αυτή είναι:

- Η κλάση **TransformNode**.

Στη engine μας θεωρούμε οτιδήποτε θα υποστεί μετασχηματισμούς ότι είναι TransformNode αντικείμενο, χρησιμοποιώντας κληρονομικότητα. Αυτή η κλάση θα χειρίζεται όλους τους μετασχηματισμούς του κάθε node σύμφωνα με το χρόνο.

Οι κλάσεις που κληρονομούν την TransformNode είναι:
- **Light** (Αποθηκεύει δεδομένα για τα φώτα)
- **Camera** (Αποθηκεύει πληροφορίες για την camera)
- **ParticleSystem** (Υπεύθυνη για να ζωγραφίζει εφέ όπως π.χ. φωτιά)
- **UIComponent** (Αντιπροσωπεύει ένα αντικείμενο του UI)
- **StaticObject** (Αντιπροσωπεύει τα στατικά αντικείμενα)
- **Entity** (Αντιπροσωπεύει τις οντότητες όπως π.χ εχθροί)

Αν θέλουμε να έχουμε animations στα γραφικά μας, και ειδικά στη περίπτωση που το κάθε μοντέλο στη σκηνή μας θέλουμε να έχει περισσότερα

από ένα, καλή πρακτική είναι να σχεδιάσουμε μία κλάση που να αποθηκεύει τις πληροφορίες για τα keyframes και να μην είναι υπεύθυνη γι' αυτό η κλάση TransformNode. Γι' αυτό το λόγο ορίζουμε την κλάση **Animation**. Instances αυτής της κλάσης περιέχονται μέσα στην TransformNode η οποία και τα χειρίζεται.

Τώρα που έχουμε χειριστεί όλους τους μετασχηματισμούς στη game engine μας, χρειαζόμαστε μία κλάση η οποία θα αποθηκεύει μέσα της την γεωμετρία (3d model) που θα μετασχηματίσουμε. Η κλάση αυτή είναι η κλάση **Mesh**. Είναι υπεύθυνη να δηλώσει στο OpenGL το πως θέλουμε να ζωγραφιστεί η γεωμετρία, και τέλος να τη ζωγραφίσει με τις ανάλογες κλήσεις στο OpenGL.

Πλέον μπορούμε ουσιαστικά να φορτώσουμε και να ζωγραφίσουμε στην οθόνη τρισδιάστατα μοντέλα, αλλά θέλουμε να τα έχουμε μαζί μέσα σε μία σκηνή. Όποτε δημιουργούμε την κλάση **Scene** η οποία κρατάει μέσα της αυτή την πληροφορία.

Τα περισσότερα video games που σέβονται τον εαυτό τους έχουν περισσότερες από μία "πίστες" γι' αυτό το λόγο χρειαζόμαστε μία κλάση **Stage** η οποία θα αντιπροσωπεύει μία πίστα του game μας και περιέχει μέσα τις διάφορες σκηνές.

Επίσης ανάλογα με τις ανάγκες του game μας μπορεί να χρειαστούμε και μία κλάση **Player** η οποία θα αποθηκεύει στοιχεία για τον παίχτη του παιχνιδιού (Health Points, Experience Points, Unlocks, Achievements κ.α.).

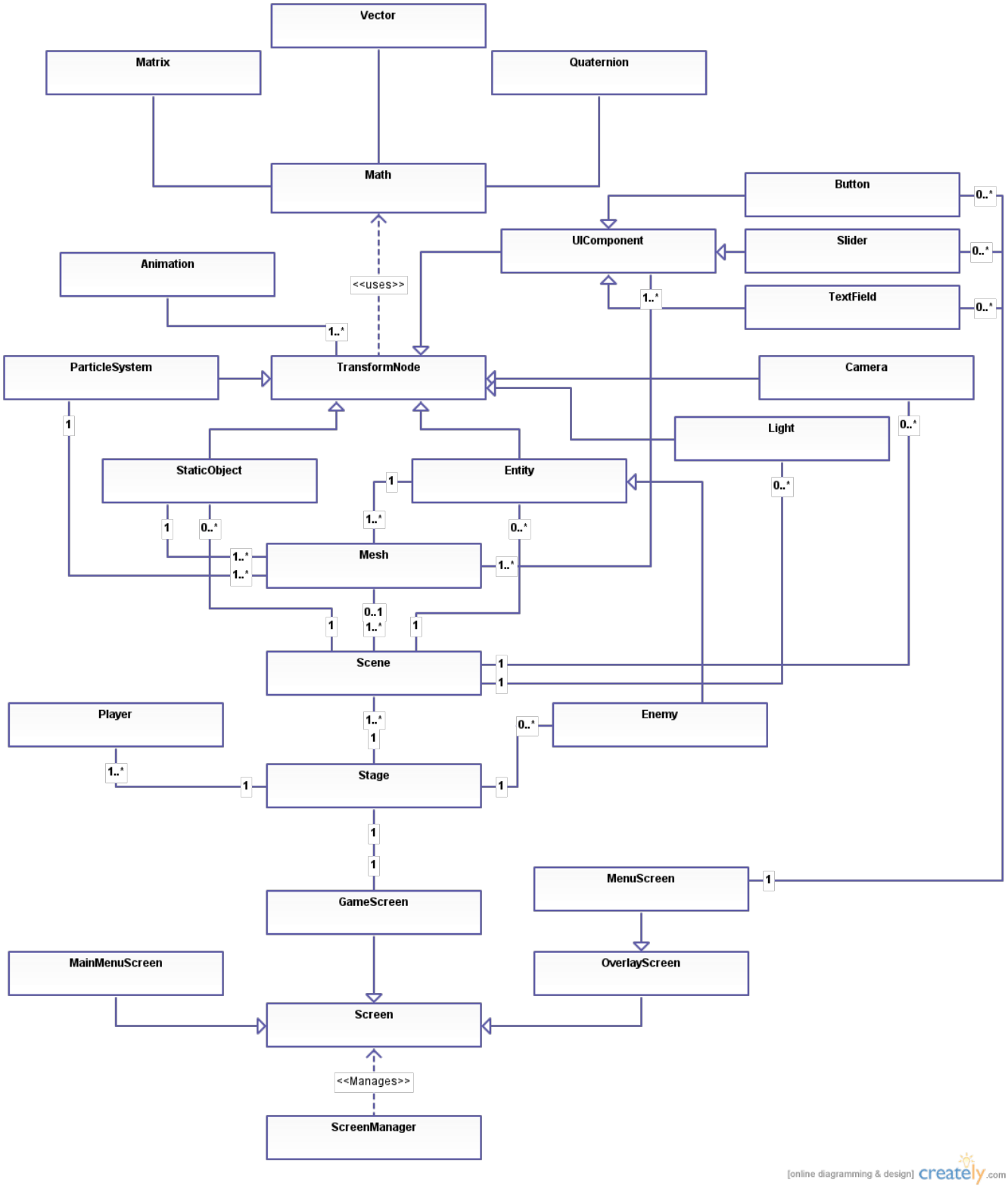
Όπως έχετε παρατηρήσει, στα περισσότερα games ποτέ σχεδόν δεν ξεκινάει το παιχνίδι αμέσως. Υπάρχουν διάφορα screens που ελέγχουν τη ροή του (Menu Screen, Pause Screen, Splash Screen). Για να πετύχουμε το ίδιο και στη δικιά μας game engine δημιουργούμε την κλάση **Screen** η οποία κρατάει πληροφορίες όπως τον χρόνο από την στιγμή ενεργοποίησης της. Η κλάση Screen είναι η Base class όλων των υπόλοιπων Screen κλάσεων που την κληρονομούν και υπερφορτώνουν τις συναρτήσεις της (πολυμορφισμός).

Τι κάνουμε όμως στην περίπτωση που χρειάζεται να ζωγραφίσουμε ένα UI πάνω στο **GameScreen** μας; (κλάση η οποία ζωγραφίζει και ελέγχει το

input για την ενεργή πίστα). Αν δημιουργήσουμε μία νέα κλάση η οποία απλά θα κληρονομεί την Screen και θα ζωγραφίζει το UI τότε θα σταματήσει να ζωγραφίζεται η GameScreen. Για να μπορέσουμε λοιπόν να ζωγραφίσουμε UI όπως για παράδειγμα ένα Pause Menu χωρίς να χαθεί η εικόνα του game μας, δημιουργούμε μία κλάση **OverlayScreen** η οποία εκτός από τον εαυτό της ζωγραφίζει και το Screen που της έχουμε θέσει σαν πατέρα (parent). Την κλάση αυτή την κληρονομούν όλες οι κλάσεις που χρειάζεται να ζωγραφίζουν κάτι πάνω από ένα οποιοδήποτε άλλο Screen. Τέλος, η base class Screen περιέχει μέσα της virtual συναρτήσεις, τις οποίες της υπερφορτώνουν οι derived classes, για να χειρίζεται τα inputs από το ποντίκι και το πληκτρολόγιο. Με αυτό το design κάθε Screen derived class μπορεί να έχει τα δικά τις controls.

Αυτός λοιπόν είναι ο σκελετός μίας σχετικά απλής game engine. Προφανώς κάθε game engine μπορεί να έχει διαφορές στο design, η επιλογή του οποίου είναι στα χέρια του κάθε προγραμματιστή γραφικών!

Ακολουθεί διάγραμμα UML με το game engine design που αναφέρθηκε παραπάνω:


Vector
Matrix
Quaternion
Math
Button
UIComponent
Slider
TextField
Animation
<<uses>>
ParticleSystem
TransformNode
Camera
Light
StaticObject
Entity
Mesh
Scene
Player
Enemy
Stage
GameScreen
MenuScreen
MainMenuScreen
OverlayScreen
Screen
<<Manages>>
ScreenManager
[online diagramming & design] creately.com

Στη συνέχεια ακολουθεί πρόγραμμα σε γλώσσα C που ζωγραφίζει έναν κύβο χρησιμοποιώντας το fixed function pipeline του OpenGL χρησιμοποιώντας τεχνική immediate mode:

- **Έκδοση για Linux**

```
#include <GL/glut.h>
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>

void draw_cube();
void display();
void idle();
void reshape(int x, int y);
void mouse_move(int x, int y);
void key_down(unsigned char key, int x, int y);

float rot_x, rot_y;

int main(int argc, char **argv)
{
	/*Initialize GLUT*/
	glutInit(&argc, argv);

	/*Initialize the Display Mode*/
	glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_DOUBLE);

	/*Initialize the Window Size*/
	glutInitWindowSize(600, 400);

	/*Create the Display Window*/
	glutCreateWindow("Cube Example");

	//function callbacks------------------
	/*Display function callback*/
	glutDisplayFunc(display);

	/*Idle function callback*/
	glutIdleFunc(idle);
```

```
	/*Motion function callback*/
	glutMotionFunc(mouse_move);

	/*Keyboard function callback*/
	glutKeyboardFunc(key_down);

	/*Calls the reshape funciton*/
	glutReshapeFunc(reshape);
	//-----------------------------------


	/*Enables Depth Testing*/
	glEnable(GL_DEPTH_TEST);

	/*Enables Back Face Culling*/
	glEnable(GL_CULL_FACE);

	glutMainLoop();
}

void draw_cube()
{
	/*State that we want to use quadrilaterals*/
	glBegin(GL_QUADS);

	/*State the color we want to use
	*note that if we don't state the use of a different one,
	*all the faces of the cube will have the same color*/
	glColor3f(1.0, 0.0, 0.0); //red

	/*State the position of each vertex*/
	glVertex3f(-1.0, -1.0, -1.0);
	glVertex3f(-1.0, 1.0, -1.0);
	glVertex3f(1.0, 1.0, -1.0);
	glVertex3f(1.0, -1.0, -1.0);

	glColor3f(0.0, 1.0, 0.0); //green
	glVertex3f(1.0, -1.0, 1.0);
	glVertex3f(1.0, 1.0, 1.0);
	glVertex3f(-1.0, 1.0, 1.0);
```

```
        glVertex3f(-1.0, -1.0, 1.0);

        glColor3f(0.0, 0.0, 1.0); //blue
        glVertex3f(-1.0, -1.0, 1.0);
        glVertex3f(-1.0, 1.0, 1.0);
        glVertex3f(-1.0, 1.0, -1.0);
        glVertex3f(-1.0, -1.0, -1.0);

        glColor3f(1.0, 1.0 , 0.0); //yellow
        glVertex3f(1.0, 1.0, -1.0);
        glVertex3f(1.0, 1.0, 1.0);
        glVertex3f(1.0, -1.0, 1.0);
        glVertex3f(1.0, -1.0, -1.0);

        glColor3f(1.0, 0.0, 1.0); //purple
        glVertex3f(-1.0, 1.0, 1.0);
        glVertex3f(1.0, 1.0, 1.0);
        glVertex3f(1.0, 1.0, -1.0);
        glVertex3f(-1.0, 1.0, -1.0);

        glColor3f(0.5, 1.0, 0.3); //something...
        glVertex3f(-1.0, -1.0, -1.0);
        glVertex3f(1.0, -1.0, -1.0);
        glVertex3f(1.0, -1.0, 1.0);
        glVertex3f(-1.0, -1.0, 1.0);

        /*State that the drawing has ended*/
        glEnd();
}

void display()
{
        /*Clears the Framebuffer*/
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        /*Specifies the clear color*/
        glClearColor(0.1, 0.1, 0.1, 1.0);

        /*Set the matrix mode to MODELVIEW*/
        glMatrixMode(GL_MODELVIEW);
```

```
	/*Set the Modelview Matrix to identity*/
	glLoadIdentity();

	/*Translate the view -5.0 units on the Z axis.*/
	glTranslatef(0.0, 0.0, -5.0);

	glRotatef(30.0, 1.0, 0.0, 0.0);

	glPushMatrix();

	/*Use the cursor position as an angle value to rotate the view
	 * on the x/y axis*/
	glRotatef(rot_y, 1.0, 0.0, 0.0);
	glRotatef(rot_x, 0.0, 1.0, 0.0);

	/*Draw out cube with the applied transformations*/
	draw_cube();

	glPopMatrix();

	/*Swap the buffers since we use Double buffering*/
	glutSwapBuffers();

	/*Check for Opengl Errors*/
	assert(glGetError() == GL_NO_ERROR);
}

void idle()
{
	/*Refresh the sceen and redisplay*/
	glutPostRedisplay();
}

void reshape(int x, int y)
{
	/*setup the OpenGL Viewport*/
	glViewport(0, 0, x, y);


	/*Calulate the aspect ratio*/
	float aspect = (float)x / (float)y;
```

```
        /*Set the matrix mode to Projection*/
        glMatrixMode(GL_PROJECTION);

        /*Set the projection matrix to identity*/
        glLoadIdentity();

        /*Calculate the projection matrix*/
        gluPerspective(45.0, aspect, 1.0, 100);
}

void mouse_move(int x, int y)
{
        rot_x = x;
        rot_y = y;
}

void key_down(unsigned char key, int x, int y)
{
        switch (key)
        {
                case 27:
                        exit(0);
                        break;
        }
}
```

Πηγές:

1. What is Computer Graphics? Cornell University Program of Computer Graphics.
2. University of Leeds ISS (2002). “What are Computer Graphics?”
3. Blythe, David. Advanced Graphics Programming Techniques Using OpenGL. Siggraph 1999.
4. Colin Smith, On Vertex-Vertex Meshes and Their Use in Geometric and Biological Modeling.
5. Bruce Baumgart, Winged-Edge Polyhedron Representation for Computer Vision. National Computer Conference, May 1975.
6. Tobler & Maierhofer, A Mesh Data Structure for Rendering and Subdivision. 2006.
7. http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/
8. The OpenGL Programming Guide a.k.a the Red Book.