ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΟΙΚΟΝΟΜΙΚΩΝ ΚΑΙ ΚΟΙΝΩΝΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«ΑΝΑΠΤΥΞΗ ΤΡΙΣΔΙΑΣΤΑΤΩΝ ΓΡΑΦΙΚΩΝ ΜΕ ΤΗ ΧΡΗΣΗ ΤΗΣ ΒΙΒΛΙΟΘΗΚΗΣ OPENGL ΣΤΗ ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ JAVA»

ΚΑΡΑΜΠΕΛΑ ΑΝΑΣΤΑΣΙΑ

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ: ΜΑΡΓΑΡΙΤΗΣ ΚΩΝΣΤΑΝΤΙΝΟΣ

ΘΕΣΣΑΛΟΝΙΚΗ 2011

Πρόλογος

Η παρούσα πτυχιακή εργασία πραγματοποιήθηκε στο Πανεπιστήμιο Μακεδονίας, Οικονομικών και Κοινωνικών Επιστημών, στο τμήμα Εφαρμοσμένης Πληροφορικής. Στόχος αυτής της πτυχιακής είναι η παρουσίαση των βασικότερων χαρακτηριστικών της γλώσσας προγραμματισμού Java OpenGL η οποία χρησιμοποιείται στην ανάπτυξη τρισδιάστατων γραφικών διεπαφών χρήστη. Επίσης, βασικό κομμάτι της εργασίας αποτελεί η ανάπτυξη της δικής μου εφαρμογής με τη χρήση του συγκεκριμένου πακέτου.

Ευχαριστώ θερμά τον επιβλέποντα καθηγητή κύριο Κωνσταντίνο Γ. Μαργαρίτη για την υποστήριξη και την πολύτιμη βοήθεια που μου προσέφερε προκειμένου να ολοκληρώσω με επιτυχία την παρούσα εργασία. Θα ήθελα να εκφράσω την ευγνωμοσύνη μου για την εμπιστοσύνη και την κατανόηση που έδειξε σε όλη την διάρκεια συγγραφής της εργασίας καθώς και για το πολύτιμο υποστηρικτικό υλικό που μου προσέφερε.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένεια μου και τους φίλους μου για την αμέριστη κατανόηση και συμπαράσταση που μου προσέφεραν όλους αυτούς του μήνες.

Περιεχόμενα

Εισαγωγή

Η OpenGL αποτελεί μια ευρέως διαδεδομένη γλώσσα ανάπτυξης τρισδιάστατων εφαρμογών η οποία χρησιμοποιείται σήμερα από πολλούς προγραμματιστές και υποστηρίζεται από τις περισσότερες κάρτες γραφικών. Είναι η γλώσσα που χρησιμοποιείται σε τρισδιάστατα παιχνίδια, όπου απαιτείται υψηλή ανάλυση και λεπτομέρειες για την απόδοση του σκηνικού, καθώς και όπου αλλού απαιτούνται γραφικά υψηλής ακρίβειας. Το χαρακτηριστικό αυτό την καθιστά κατάλληλη και σε ιατρικές εφαρμογές για προσομοίωση ανθρώπινων οργάνων, καθώς και σε αναπαράσταση αρχιτεκτονικών σχεδίων.

Ένα από τα βασικότερα χαρακτηριστικά της που με ώθησαν στο να ασχοληθώ με την παρούσα εργασία είναι το γεγονός ότι η βιβλιοθήκες της OpenGL είναι ανοικτού κώδικα και μπορεί ο καθένας να έχει πρόσβαση σε αυτές. Επίσης, κατά τη γνώμη μου ένα ακόμη χρήσιμο χαρακτηριστικό της είναι ότι οι OpenGL εφαρμογές μπορούν να εκτελεστούν σε οποιοδήποτε λειτουργικό σύστημα αρκεί να υπάρχει εγκατεστημένη η κατάλληλη βιβλιοθήκη.

Στην παρούσα εργασία ασχολήθηκα με την συγγραφή προγραμμάτων OpenGL με τη χρήση της γλώσσας προγραμματισμού Java. Σκοπός της πτυχιακής είναι η παρουσίαση της γλώσσας Java OpenGL με τρόπο κατανοητό και σύντομο έτσι ώστε ο χρήστης μετά από την μελέτη του κειμένου να είναι σε θέση να αναπτύξει το δικό του πρόγραμμα χρησιμοποιώντας τα εργαλεία και τις τεχνικές που παραθέτω. Η εργασία είναι δομημένη με τρόπο που μυεί τον αναγνώστη σταδιακά στις λειτουργίες και τα εργαλεία της γλώσσας ώστε αυτός να μπορέσει να κατανοήσει πλήρως τον κώδικα πριν την περιγραφή των πιο σύνθετων εφαρμογών που υπάρχουν στις δύο τελευταίες ενότητες της εργασίας.

Η συγκεκριμένη πτυχιακή είναι χωρισμένη σε τέσσερις μεγάλες ενότητες οι οποίες περιλαμβάνουν κάποια επιμέρους κεφάλαια.

Στην 1^{η} ενότητα παραθέτω κάποια στοιχεία που αφορούν τη γλώσσα OpenGL γενικότερα. Υπάρχει μια ιστορική ανασκόπηση της εξέλιξης της γλώσσας από την στιγμή που δημιουργήθηκε μέχρι σήμερα, καθώς και χρήσιμες πληροφορίες που αφορούν τη λειτουργία της και τα πλεονεκτήματα – μειονεκτήματα της σε σχέση με αντίστοιχες γλώσσες.

Στη 2^{η} ενότητα αναλύω τα κύρια χαρακτηριστικά και της λειτουργίες της Java OpenGL. Η ενότητα αυτή περιέχει το βασικό περιεχόμενο της πτυχιακής χωρισμένο σε κεφάλαια με βάση τη λειτουργία της JOGL που αναλύεται κάθε φορά. Ξεκινάει με την περιγραφή των βασικών σημείων ενός JOGL προγράμματος και προχωράει στον σχεδιασμό τρισδιάστατων αντικειμένων για να καταλήξει σε πιο εξειδικευμένα χαρακτηριστικά, όπως η υφή και διάφορα εφέ που αφορούν τον φωτισμό του σκηνικού.

Στην 3^{η} ενότητα παραθέτω μια υποδειγματική εφαρμογή η οποία συνδυάζει τις περισσότερες από τις λειτουργίες που αναφέρθηκαν στη θεωρία και αποτελεί το πρώτο εξελιγμένο πρόγραμμα. Πρόκειται για μια εφαρμογή που βρήκα έτοιμη στο διαδίκτυο και στην οποία πραγματοποίησα κάποιες τροποποιήσεις προκειμένου να πετύχω ένα πιο εντυπωσιακό αποτέλεσμα.

Στην 4^{η} ενότητα αναπτύσσω μια εξολοκλήρου δική μου εφαρμογή η οποία προσομοιάζει ένα τρισδιάστατο δωμάτιο στο οποίο υπάρχει η δυνατότητα περιήγησης του χρήστη. Το

πρόγραμμα αυτό περιλαμβάνει τις περισσότερες δυνατότητες της JOGL που ανέλυσα στην θεωρία.

Στην 5η και τελευταία ενότητα αναφέρω γενικά συμπεράσματα στα οποία κατέληξα μετά την υλοποίηση της συγκεκριμένης εργασίας και στην μελλοντική έρευνα που μπορεί να γίνει στο πεδίο αυτό.

I. OpenGL

1. Ιστορικά στοιχεία

Κάπου στις αρχές του 1990 η Silicon Graphics δημιούργησε ένα standard πακέτο βιβλιοθηκών API για την απεικόνιση 3D γραφικών σε διαφόρους σταθμούς εργασίας, το IrisGL. Το πακέτο αυτό, αν και ξεχώρισε για την ευκολία στη χρήση του, δυστυχώς ήταν άρρηκτα δεμένο με το X Window System, σε αντίθεση με το ανταγωνιστικό πακέτο PHIGS που ήταν ανοιχτού τύπου. Ως αποτέλεσμα οι ανταγωνιστές τις Silicon Graphics (Sun Microsystems, Hewlett-Packard, IBM κ.α.) άρχισαν να παράγουν οι ίδιοι 3D λογισμικό ως επέκταση του PHIGS.

Ο ανταγωνισμός αυτός οδήγησε την Silicon Graphics στην δημιουργία της OpenGL που αναπτύχθηκε ως πρότυπο ανεξάρτητο από το λειτουργικό σύστημα στο ποίο εκτελείται. Το 1992 η Silicon Graphics ηγήθηκε τη δημιουργία της OpenGL Architecture Review Board (ARB), μιας επιτροπής αποτελούμενης από διάφορες εταιρείες και οργανισμούς που ασχολούνται με τον χώρο και κατευθύνουν πλέον το πρότυπο OpenGL.

Το 1995 η Microsoft κυκλοφόρησε το Direct3D το οποίο θα αποτελέσει τον κυριότερο ανταγωνιστή του OpenGL. Το Δεκέμβριο του 1997 η Microsoft και η Silicon Graphics ξεκίνησαν το project Fahrenheit που αποτέλεσε μια κοινή προσπάθεια ενοποίησης των interfaces του PenGL και Direct3D. Το 1998 η Hewlett – Packard προστέθηκε στο project αυτό το ποίο δυστυχώς εγκαταλείφθηκε το 1999 λόγο οικονομικών και στρατηγικών αιτιών.

Στον πίνακα 1 αναφέρονται με την σειρά εμφάνισής τους οι εκδόσεις της OpenGL καθώς και τα επιπρόσθετα χαρακτηριστικά κάθε έκδοσης. Μια κάρτα γραφικών που υποστηρίζει μια συγκεκριμένη έκδοση περιλαμβάνει τα χαρακτηριστικά τόσο τις πιο πρόσφατης όσο και των προηγούμενων εκδόσεων της OpenGL. [5]

Έτος	Έκδοση	Χαρακτηριστικά
1992	OpenGL 1.0	Δημιουργήθηκε από τους Mark Segal και Kurt Akeley
1997	OpenGL 1.1	Εστιάζει στη υποστήριξη διαφόρων textures από λογισμικό GPU
1998	OpenGL 1.2	Υποστηρίζει textures που προσδίδουν όγκο, packed pixels, normal rescaling, clamped/edge texture sampling και επεξεργασία εικόνας
1998	OpenGL 1.2.1	Προσθέτει στην έκδοση 1.2 τα πολλαπλά textures ή texture units (επιτρέπουν την απόδοση διαφορετικής υφής σε κάθε pixel κατά τη διαδικασία εμφάνισης του αντικειμένου στη οθόνη)
2001	OpenGL 1.3	Υποστηρίζει cubemap texture, multi-texturing, multi-sampling και τον συνδυασμό texture units με πράξεις όπως add, compine, dot3 και border clamp
2002	OpenGL 1.4	Προσθέτει υλικό για την υποστήριξη shadowing, συντεταγμένων ομίχλης, αυτόματη δημιουργία bipmap και επιπλέον textures
2003	OpenGL 1.5	Υποστηρίζει vertex buffer objects (VBOs), ερωτήματα απόφραξης (occlusion queries) και επιπλέον συναρτήσεις shadowing
2004	OpenGL 2.0	Υποστηρίζει μια GPU – based γλώσσα assembly, την ARB, η οποία θα γίνει πρότυπο για vertex και fragment shaders. Οι κάρτες

		γραφικών που την υποστηρίζουν είναι οι πρώτες που επιτρέπουν τον προγραμματισμό shaders από τον χρήστη
2006	OpenGL 2.1	Εισήγαγε την υποστήριξη pixel buffer objects (PBOs), sRGB textures (gamma – corrected textures), μη – τετραγωνικών πινάκων και μια αναθεώρηση της Shading Language GLSL 1.20
2008	OpenGL 3.0	Προσθέτει υποστήριξη για frame buffer objects, υλικό instancing, vertex array objects (VAOs) και sRGB framebuffers. Εισήγαγε έναν μηχανισμό απόσβεσης ώστε να απλουστεύσει το API μελλοντικών εκδόσεων
Μάρτιος 2009	OpenGL 3.1	Εισάγει μια σειρά χαρακτηριστικών που κάνουν το API ευκολότερο στην χρήση και εισάγει την OpenGL Shading Language έκδοση 1.40 (GLSL)
Αύγουστος 2009	OpenGL 3.2	Περιλαμβάνει την έκδοση 1.50 της OpenGL Shading Language (GLSL)
Μάρτιος 2010	OpenGL 3.3	Κυκλοφόρησε ταυτόχρονα με την OpenGL 4.0 και συμπληρώθηκε από μια σειρά νέων ARB extensions. Ενσωματώνει όσο το δυνατόν περισσότερη λειτουργικότητα της OpenGL 4.0 και μπορεί να εκτελείται παράλληλα σε προηγούμενης γενιάς λειτουργικό GPU.
Μάρτιος 2010	OpenGL 4.0	Εισάγει αρκετά νέα χαρακτιριστηκα στον τομέα της σκίασης (OpenGL Shading Language)
Αύγουστος 2010	OpenGL 4.1	Η νέα αυτή έκδοση περιέχει επιπρόσθετα χαρακτηριστικά στις προδιαγραφές, πολλά από τα οποία βοηθούν την OpenGL να ευθυγραμμιστεί με τις προδιαγραφές του Direct3D 11

Πίνακας 1: Χρονική εξέλιξη της OpenGL

2. Γενικές Πληροφορίες για την OpenGL

Όπως αναφέρθηκε και παραπάνω, η OpenGL αποτελεί μια διεπαφή που δίνει στον χρήστη τη δυνατότητα απεικόνισης τρισδιάστατων σκηνών. Η διεπαφή αυτή αποτελείται από περισσότερες από 700 διαφορετικές εντολές (670 εντολές του OpenGL Version 3.0 και περίπου 50 του OpenGL Utility Library) που καθορίζουν τα αντικείμενα και τις συναρτήσεις που επιθυμεί να χρησιμοποιήσει ο χρήστης για την διαμόρφωση της 3D σκηνής του.

Η OpenGL προκειμένου να αποδεσμευτεί από το λογισμικό ενός υπολογιστή και να λειτουργήσει ανεξάρτητα από την πλατφόρμα στην οποία υλοποιείται δεν περιέχει εντολές για την διαχείριση παραθύρων και δεν υποστηρίζει είσοδο από τον χρήστη (user input). Αντίθετα ο χρήστης θα πρέπει να χρησιμοποιεί τις αντίστοιχες εντολές που υποστηρίζει το σύστημα στο οποίο τρέχει το πρόγραμμά του.

Επίσης, δεν υποστηρίζει εντολές υψηλού επιπέδου για την περιγραφή 3D αντικειμένων. Οι εντολές αυτές θα επέτρεπαν την περιγραφή πολύπλοκων αντικειμένων, όπως οχήματα, μέλη ενός ανθρωπίνου σώματος κ.λ.π. Ωστόσο, στην OpenGL ένα 3D αντικείμενο διαμορφώνεται σταδιακά ως ένα σύνολο από βασικά γεωμετρικά αρχέτυπα, όπως τα σημεία, τις γραμμές και τα πολύγωνα. [1]

Το πακέτο OpenGL είναι μια βιβλιοθήκη βασισμένη στην γλώσσα προγραμματισμού C. Η αρχική της υλοποίηση ήταν βασισμένη σε μια μηχανή καταστάσεων (state machine), ωστόσο νεότερες εκδόσεις της την μετέτρεψαν σε ένα σύστημα κατευθυνόμενο από αντικείμενα.

Αν και η βασική της υλοποίηση είναι σε C, μπορεί να υλοποιηθεί και σε άλλες γλώσσες. Η OpenGL δεν εξαρτάται από καμία γλώσσα συγκεκριμένα και μπορεί να κληθεί από οποιαδήποτε γλώσσα προγραμματισμού με την κατάλληλη σύνδεση. Τέτοιοι σύνδεσμοι υπάρχουν για τις παρακάτω γλώσσες: Lisp, Fortran, BASIC, VisualBasic, PureBasic,Ada, Pascal, Delphi, Python, Lua, P erl, Haskell, Java, Ruby και C#.

Πολλές βιβλιοθήκες έχουν αναπτυχθεί ένα επίπεδο παραπάνω ή παράλληλα με την OpenGL ώστε να παρέχουν χαρακτηριστικά που δεν είναι διαθέσιμα στην βασική έκδοση. Βιβλιοθήκες όπως η OpenGL Utility Library (GLU) συναντούνται σε κάθε υλοποίηση της OpenGL, καθώς και άλλες όπως η OpenGL Utility Library Toolkit (GLUT) και η Simple DirectMedia Layer (SDL) που υποστηρίζουν λειτουργίες παραθύρων και ποντικιού. Πιο απλές λειτουργίες γραφικής διεπαφής υπάρχουν τις βιβλιοθήκες OpenGL User Interface Library (GLUI) και Fast, Light Toolkit (FLTK). Άλλες βιβλιοθήκες προσφέρουν στους χρήστες OpenGL τη δυνατότητα να διαχειριστούν τις διάφορες επεκτάσεις και εκδόσεις της γλώσσας. Μερικά παραδείγματα είναι η GLEW (the OpenGL Extension Wrangler Library) και η GLEE (the OpenGL Easy Extension Library).

Επίσης διατηρούνται και κάποιες βιβλιοθήκες υψηλότερου επιπέδου κατευθυνόμενες από τα αντικείμενα μιας 3D σκηνής όπως η PLIB, η OpenSG, η OpenSceneGraph και η OpenGL Performer οι οποίες υποστηρίζουν την προσομοίωση real – time εφαρμογών. Άλλες βιβλιοθήκες υποστηρίζουν την ανάπτυξη παράλληλων OpenGL προγραμμάτων που χρησιμοποιούνται στην Εικονική Πραγματικότητα. [5]

3. Χρήσεις της OpenGL

Υπάρχουν υλοποιήσεις της OpenGL σε πολλά λειτουργικά συστήματα όπως τα Microsoft Windows και συστήματα βασισμένα σε UNIX, όπως Mac OS X, GNU/Linux και OpenSolaris. Η OpenGL πλέον χρησιμοποιείται οπουδήποτε απαιτείται η δημιουργία ισχυρών 3D μοντέλων, όπως παιχνίδια, ιατρικές εφαρμογές και αρχιτεκτονικά σχέδια.

Η OpenGL έχει επιλεγεί ως η βασικότερη βιβλιοθήκη γραφικών στα: iPhones, Android (mobile operating system) και Symbian OS (Nokia’s mobile operating system) με τη μορφή της OpenGL ES (OpenGL for Embedded Systems). Η OpenGL ES έχει αναπτυχθεί για χρήση σε συσκευές όπως κινητά τηλέφωνα, PDAs ή palmtop computers και σε κονσόλες video games. Η OpenGL ES διοικείται από μια μη – κερδοσκοπικού χαρακτήρα κοινοπραξία, την Kronos Group, Inc.

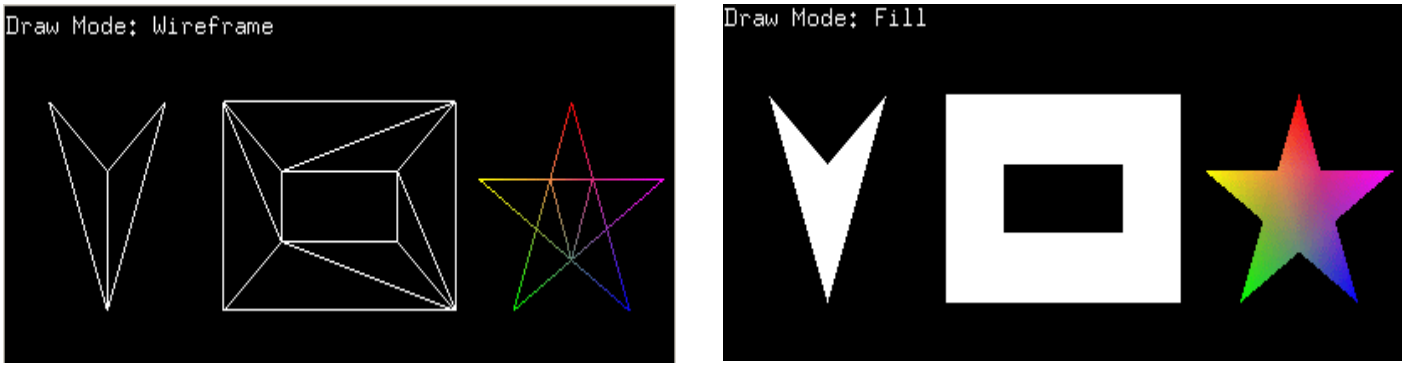
Επικρατεί ένας μύθος ότι παραλλαγές της OpenGL χρησιμοποιούνται στο Nitendo και σε πλατφόρμες της Sony, ωστόσο κάτι τέτοιο δεν ισχύει καθώς τα συστήματα αυτά έχουν αναπτύξει τις δικές τους βιβλιοθήκες. Επίσης, το Sony Playstation 3 διαθέτει μια υλοποίηση της OpenGL που δε χρησιμοποιείται λόγο προβλημάτων απόδοσης. [6]

4. Τρόπος Αναπαράστασης Αντικειμένων στην OpenGL (Polygon Rasterization)

Για να κατανοήσουμε την αναπαράσταση ενός 3D σκηνικού φανταζόμαστε τα διάφορα αντικείμενα που αυτή περιλαμβάνει ως σύνολο από πολύγωνα που αντιστοιχούν στην επιφάνεια του κάθε αντικειμένου. Τα πολύγωνα αυτά είναι που τελικά θα σχεδιαστούν στην οθόνη του υπολογιστή μας.

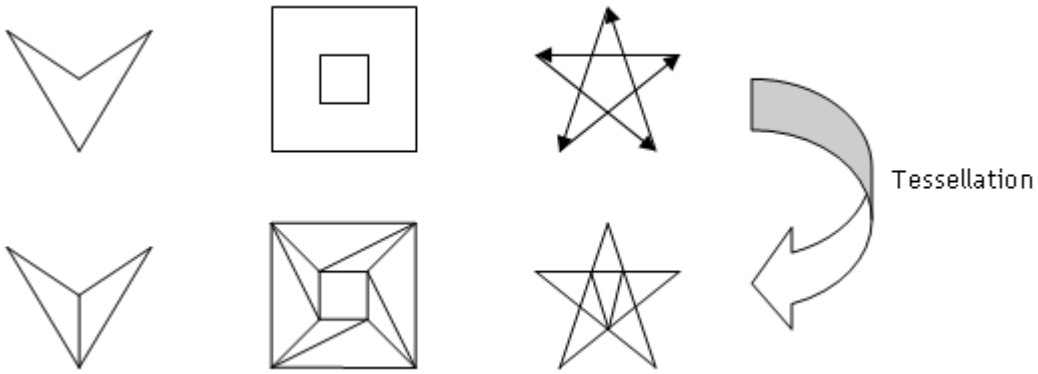
Η OpenGL για την απεικόνιση 3D γραφικών χρησιμοποιεί τη μέθοδο αναπαράστασης γνωστή ως polygon rasterization. Με τη μέθοδο αυτή μεταφέρουμε τα 3D πολύγωνα των αντικειμένων σε ένα 2D επίπεδο (το επίπεδο της εικόνας μας) και γεμίζουμε τα pixels του πολυγώνου που αντιστοιχεί στο κάθε αντικείμενο με το κατάλληλο χρώμα. Η τεχνική αυτή είναι η γρηγορότερη που υπάρχει για την αναπαράσταση 3D σκηνικών και χρησιμοποιείται ευρέως σε realtime παιχνίδια και σε εξειδικευμένες κάρτες γραφικών, όπως αυτές που περιλαμβάνονται στους περισσότερους υπολογιστές σήμερα.

Η OpenGL προκειμένου να μπορεί να υποστηρίζει γρήγορο λογισμικό για την εκτέλεση του polygon rasterization έχει θεσπίσει ορισμένους κανόνες σχετικά με την εγκυρότητα των διαφόρων τύπων πολυγώνων, αυτοί είναι: α) οι ακμές των πολυγώνων δεν τέμνονται, β) οι ακμές των πολυγώνων δεν είναι κυρτές, δηλαδή δύο σημεία μιας ακμής μπορούν συνδέονται μόνο με μια ευθεία γραμμή (το πλήθος των ακμών ενός πολυγώνου είναι απεριόριστο) και γ) πολύγωνα που περιέχουν “τρύπες” στο εσωτερικό τους δεν μπορούν να αναπαρασταθούν σωστά, σε ορισμένα συστήματα μπορεί να “γεμίσει” όλο το πολύγωνο με χρώμα ή ένα μέρος του χωρίς να αντιστοιχεί όμως στο επιθυμητό αποτέλεσμα. [2]



Εικόνα 1: Ο τρόπος κατασκευής αντικειμένων από βασικά πολύγωνα

Η OpenGL χρησιμοποιεί μια τεχνική η οποία διαχωρίζει πολύπλοκα πολύγωνα τα οποία δεν πληρούν τις παραπάνω προϋποθέσεις στα βασικά πολύγωνα που χρησιμοποιεί η OpenGL για 3D αναπαράσταση. Η τεχνική αυτή ονομάζεται Tessellation (ψηφιδοποίηση ή ψηφίδωση). Στις παρακάτω εικόνες παρουσιάζονται τα στάδια που περνάει ένα “πολύπλοκο” πολύγωνο μέχρι να καταφέρει να εμφανιστεί στην οθόνη του υπολογιστή μας. [7]



Εικόνα 2: Μετατροπή των μη «έγκυρων» πολυγώνων σε «έγκυρα»

Τα πολύγωνα όπως φαίνονται με γέμισμα (δεξιά) και χωρίς γέμισμα (αριστερά) των εσωτερικών pixels με χρώμα.

5. Σύγκριση OpenGL με Αντίστοιχα Εργαλεία Ανάπτυξης Γραφικών (Java 3D & Direct 3D)

Η χρήση της OpenGL για την ανάπτυξη 3D σκηνικών προσφέρει κάποια πλεονεκτήματα σε σχέση με τη χρήση τόσο της Java 3D, όσο και του DirectX για τον ίδιο σκοπό.

Αρχικά, ένας από τους κυριότερους λόγους που οδηγούν στην υιοθέτηση της OpenGL από ολοένα και περισσότερους προγραμματιστές είναι ότι αποτελεί ένα λογισμικό ανοιχτού κώδικα το οποίο είναι ανεξάρτητο από την πλατφόρμα στην οποία τρέχει (multi – platform). Αντίθετα το Direct3D υλοποιείται μόνο σε λειτουργικά συστήματα τις Microsoft περιορίζοντας την φορητότητα των προγραμμάτων.

Καθώς νέες συσκευές, όπως τα κινητά τηλέφωνα, smartfhones, i-phones κ.α. επικρατούν στην αγορά η OpenGL κερδίζει έδαφος καθώς μπορεί και υποστηρίζει την ανάπτυξη παιχνιδιών και άλλων γραφικών σε κάθε είδους συσκευή. [6]

Το DirectX και η Java 3D είναι ολοκληρωμένες βιβλιοθήκες η οποία περιλαμβάνουν και στοιχεία χαμηλότερου επιπέδου, πέρα από τα γραφικά. Αντίθετα η OpenGL είναι μια γλώσσα υψηλότερου επιπέδου αποκλειστικά για την ανάπτυξη γραφικών.

Τα προγράμματα σε OpenGL «τρέχουν» πιο γρήγορα σε σχέση με τα αντίστοιχα σε Java 3D, για αυτό το λόγο η OpenGL, συγκριτικά με την Java3D, προτιμάται για την ανάπτυξη παιχνιδιών.

Αντίθετα, σε σύγκριση με το Direct3D, η OpenGL χρησιμοποιείται στην ανάπτυξη γραφικών για επαγγελματικούς σκοπούς κυρίως ενώ το Direct3D είναι αυτό που επικρατεί στον τομέα των παιχνιδιών.

Η OpenGL δίνει τη δυνατότητα στον προγραμματιστή να έχει άμεση πρόσβαση στον τρόπο που θα γίνει η απόδοση της 3D σκηνής του (γλώσσα υψηλού επιπέδου), μέσα από εντολές «απαίτησης» και όχι με εντολές τύπου «ερώτησης» όπως στη Java 3D.

Επίσης, η OpenGL είναι ένα πακέτο βιβλιοθηκών που έχει βελτιστοποιηθεί με κάθε δυνατό τρόπο τόσο σε επίπεδο hardware και software, όσο και σε επίπεδο ανεξάρτητων πλατφόρμων (από φθηνότερα PC’s και κονσόλες παιχνιδιών μέχρι τους πιο εξελιγμένους υπερ-υπολογιστές γραφικών). [8]

II. Java OpenGL

1. Γενικά Στοιχεία της JOGL

Το πακέτο Java OpenGL (JOGL) είναι ένα σύνολο βιβλιοθηκών API που επιτρέπει την OpenGL να χρησιμοποιείται στην γλώσσα προγραμματισμού Java. Αρχικά αναπτύχθηκε από τους Kenneth Bradley Russell και Christopher John Kline και στη συνέχεια εξελίχθηκε από το Game Technology Group της Sun Microsystems. Αποτελεί υλοποίηση του SR-231 (Java Bindings for OpenGL) και είναι ένα ανεξάρτητο πακέτο λογισμικού ανοικτού κώδικα στο ποίο ο καθένας μπορεί να έχει εύκολα πρόσβαση.

Η JOGL επιτρέπει την πρόσβαση στα περισσότερα χαρακτηριστικά που είναι διαθέσιμα και στην γλώσσα προγραμματισμού C με την αξιοσημείωτη διαφορά ότι μπορεί να πραγματοποιεί κλήσεις συστήματος που προκαλούνται από παράθυρα (window-system related calls) με τη χρήση του GLUT (OpenGL Utility Toolkit) της Java. [5]

Η JOGL έχει σχεδιαστεί για να υποστηρίζει μόνο τις πιο πρόσφατες εκδόσεις της Java. Πιο συγκεκριμένα η JOGL είναι συμβατή με την έκδοση Java 2 Platform Standard Edition 1.4 καθώς και όλες της νεότερες εκδόσεις της. Επίσης, υποστηρίζει μόνο truecolor rendering (15 bits ανά pixel και περισσότερα) και όχι color-indexed modes. Έχει σχεδιαστεί με τη χρήση των νέων βιβλιοθηκών της Java, New I/O (NIO) και χρησιμοποιεί, εσωτερικά, χαμηλού επιπέδου εντολές εισόδου - εξόδου σε πολλές εφαρμογές. [9]

Όταν ένα JOGL πρόγραμμα εκτελείται, η βασική βιβλιοθήκη του OpenGL C προσπελάζεται με κλήσεις στο Java Native Interface (JNI),δηλαδή η OpenGL χρησιμοποιεί το JNI για να έχει πρόσβαση στις βιβλιοθήκες γραφικών της OpenGL. Εξαιτίας του μεγάλου όγκου της OpenGL καθώς και των συνεχόμενων βελτιώσεων και προσθηκών που πραγματοποιούνται στον αρχικό κώδικα είναι δύσκολο για τους προγραμματιστές να πραγματοποιούν όλες τις αλλαγές αυτές κατευθείαν στην java. Για αυτό το λόγο αποφάσισαν τη δημιουργία ενός εργαλείου μετατροπής που ονομάζεται Gluegen. Το εργαλείο αυτό αναλύει τα OpenGL header files της C και δημιουργεί τον κατάλληλο JOGL κώδικα καθώς και το JNI που τα συνδέει μεταξύ τους. Αυτό σημαίνει ότι οποιαδήποτε αλλαγή στην OpenGL μπορεί να μεταφερθεί αυτόματα στην JOGL. [3]

Το binding της Jogl είναι γραμμένο σχεδόν εξολοκλήρου σε Java. Υπάρχουν περίπου συνολικά 150 γραμμές κώδικα σε C μέσα στο σύνολο του πηγαίου κώδικα της Jogl, ενώ ο υπόλοιπος κώδικας παράγεται αυτόματα στην διαδικασία του build με τη βοήθεια του εργαλείου GlueGen, όπως αναφέρθηκε παραπάνω. [9]

Η απόφαση για την αντιστοιχία του κώδικα JOGL με αυτόν της C έχει τα πλεονεκτήματα και τα μειονεκτήματά της. Το επίπεδο αφαίρεσης της JOGL είναι πλησιέστερο στο μοντέλο της C, δηλαδή σχετικά χαμηλότερο από αυτό της Java και άλλων βιβλιοθηκών γραφικών όπως της Java 3D. Το γεγονός αυτό έχει ως πλεονέκτημα ότι οι συναρτήσεις που χρησιμοποιούνται στο JOGL είναι αρκετά κοντά σε αυτές της C και οι προγραμματιστές μπορούν εύκολα να περάσουν από τη μια γλώσσα στην άλλη. Από την άλλη πλευρά υπάρχει το μειονέκτημα ότι οι βιβλιοθήκες της JOGL δεν είναι αρκετά αντικειμενοστραφείς. [5]

2. OpenGL API

2.1. Εγκατάσταση του OpenGL API

Για κάποιον που ξεκινάει την ενασχόληση του με την JOGL και επιθυμεί να την εγκαταστήσει στον υπολογιστή του, οι πιο πρόσφατες εκδόσεις της είναι διαθέσιμες στη σελίδα java.net. Το λογισμικό που χρησιμοποίησα για την ανάπτυξη των προγραμμάτων μου είναι το Java Binding for OpenGL API, version 1.1.10 το οποίο κατέβασα από την παραπάνω σελίδα.

Το πακέτο της JOGL έχει την μορφή ενός zip αρχείου το οποίο περιλαμβάνει τις βασικές κλάσεις ώστε να μπορείτε να καλέσετε την OpenGL μέσω της Java καθώς και τις ανάλογες JNI native libraries. Η JOGL επίσης εξαρτάται από κάποιες run-time support κλάσεις καθώς και από τον native code του GlueGen, τα οποία επίσης περιέχονται στο zip αρχείο.

Ακολουθούν οδηγίες για την εγκατάσταση της OpenGL στον υπολογιστή σας.

1. Για να κατεβάσετε το απαραίτητο αυτό πακέτο για την ανάπτυξη των JOGL προγραμμάτων σας επισπευτείτε το σύνδεσμο: https://jogl.dev.java.net/servlets/ProjectDocumentList?folderID=9260&expandFolder=9260&folderID=0

2. Στη συνέχεια με κλικ στο κατάλληλο σύνδεσμο, ανάλογα με το λειτουργικό σύστημα στο οποίο θα τρέξει η JOGL (π.χ. jogl-1.1.1-windows-i586.zip), κατεβάζουμε το αρχείο στον υπολογιστή μας και κάνουμε extract.

3. Κάνουμε αντιγραφή – επικόλληση των αρχείων jogl.jar και gluegen-rt.jar στον τρέχον κατάλογο "C:/Program Files/Java/jdk…/jre/lib/ext/ "

4. κάνουμε αντιγραφή – επικόλληση των αρχείων jogl.dll, gluegen-rt.dll, jogl_awt.dll και jogl_cg.dll στο path "C:/Program Files/Java/jdk…/jre/bin/"

Στο σημείο αυτό έχουμε εγκαταστήσει την JOGL στον υπολογιστή μας και είμαστε έτοιμοι να αναπτύξουμε τον κώδικά μας. Για όσους χρησιμοποιούν περιβάλλοντα ανάπτυξης κώδικα, όπως το NetBeans και το Eclipse, θα πρέπει να συμβουλευτούν το IDE documentation ώστε να εισάγουν τα jar αρχεία και τις βιβλιοθήκες με τον σωστό τρόπο. [9]

Το περιβάλλον που χρησιμοποίησα για την ανάπτυξη και δοκιμή των προγραμμάτων μου είναι το NetBeans 6.8. Τα απαραίτητα plug-ins για ανάπτυξη JOGL κώδικα είναι διαθέσιμα στη επίσημη σελίδα του NetBeans με την ονομασία NetBeans OpenGL Pack. Θα πρέπει να τονίσω ότι το πακέτο αυτό είναι συμβατό μόνο με εκδόσεις του NetBeans από 6.8 και κάτω. Η νέα έκδοση σου πακέτου για το NetBeans 6.9.1 δεν είναι ακόμα διαθέσιμη.

Αφού κατεβάσουμε το zip αρχείο κάνουμε extract σε έναν φάκελο ο οποίος τώρα περιέχει όλα τα αρχεία .nbm τα οποία κάνουμε install στο NetBeans με τον εξής τρόπο:
1. Στην γραμμή εργαλειών του NetBeans πηγαίνουμε στο Tools → Plugins.
2. Επιλέγουμε την καρτέλα Downloaded και πατάμε Add Plugins…
3. Πηγαίνουμε στον φάκελο όπου έχουμε κάνει extract το αρχείο, επιλέγουμε όλα τα αρχεία με την κατάληξη .nbm και κάνουμε Open.

4. Τώρα τα αρχεία έχουν προστεθεί στη λίστα με τα κατεβασμένα plugins και επιλέγουμε Install.
5. Στο σημείο αυτό η διαδικασία έχει ολοκληρωθεί και κλείνουμε τα παράθυρα που έχουμε ανοίξει.

2.2. Βασικές Κλάσεις

Ακολουθεί μια σύντομη παρουσίαση του API της OpenGL καθώς και μια περιγραφή των σημαντικότερων σημείων του κώδικα για την ανάπτυξη ενός OpenGL προγράμματος.

Οι βασικότερες κλάσεις που αποτελούν την διασύνδεση μας με το 3D σκηνικό της OpenGL είναι :

- GLEventListener:
  Όταν ένα πρόγραμμα γραμμένο σε JOGL είναι έτοιμο να εκτελεστεί ελέγχει πρώτα αν υπάρχουν GLEventListeners. Σε περίπτωση που υπάρχουν καλεί μια από τις τέσσερις μεθόδους της κλάσης: display(), displayChanged(), init() και reshape(). Από αυτές, οι μέθοδοι που χρησιμοποιούνται συχνότερα είναι οι dysplay() και init().

- GLDrawable, GLCanvas, GLJPanel:
  Η GLDrawable είναι στην ουσία μια διασύνδεση στην οποία η OpenGL «ζωγραφίζει» το 3D σκηνικό της. Οι GLCanvas και GLJPanel είναι κλάσεις που υλοποιούν την GLDrawable. Οι GLEventListeners που αναφέρθηκαν παραπάνω διαχειρίζονται τις κλάσεις αυτές σαν GLDrawables χωρίς να γνωρίζουν ποια ακριβώς κλάση έχει υλοποιηθεί. Οι βασικότερες μέθοδοι που χρησιμοποιούνται για αυτό το σκοπό είναι οι: addEventListener(), getGL() και getGLU. Η ιδιότητα αυτή ονομάζεται πολυμορφισμός και κυριαρχεί στα προγράμματα της OpenGL.

- GLCapabilities:
  Πρόκειται για τη διασύνδεση η οποία περιλαμβάνει όλλες τις βασικές δυνατότητες και τα χαρακτηριστικά της γλώσσας που είναι διαθέσημα στον χρήστη.

Οι παραπάνω βασικές κλάσεις βρίσκονται στο folder javax.media.opengl, οπότε πριν από την έναρξη κάθε OpenGL προγράμματος θα πρέπει να εισάγουμε την βιβλιοθήκη: import javax.media.opengl.*. [3]

3. Ανάπτυξη του Πρώτου JOGL Προγράμματος

3.1. Το πρόγραμμα SimpleJoglApp

Στο σημείο αυτό, καθώς και στις ενότητες που ακολουθούν, αναπτύσσω σταδιακά εφαρμογές οι οποίες θα βοηθήσουν τον χρήστη να κατανοήσει την δομή ενός JOGL προγράμματος αλλά και να μπορεί να αναπτύξει ο ίδιος τα δικά του προγράμματα. Τα περισσότερα από τα προγράμματα προέρχονται από το βιβλίο του Gene Davis με τίτλο Learning Java Bindings for OpenGL και παρατίθενται με ελάχιστες τροποποιήσεις και δικά μου σχόλια.

Επέλεξα για την ανάπτυξη των προγραμμάτων μου να ακολουθήσω την τακτική στην οποία υπάρχουν δύο κλάσεις. Το πρώτο βήμα για την ανάπτυξη μιας εφαρμογής είναι η δημιουργία ενός project το όνομα του οποίου συμπίπτει με το όνομα της κύριας κλάσης. Στη συνέχεια, μέσα στο project αναπτύσσουμε τις δύο κλάσεις: μια κύρια κλάση που περιλαμβάνει την main και καλεί την δευτερεύουσα κλάση, η οποία έχει συνήθως τη λέξη View ως συνθετικό του ονόματός της και είναι υπεύθυνη για το σχεδιασμό του 3D ή 2D σκηνικού μας.

Για να ξεκινήσουμε την γνωριμία μας με την JOGL παραθέτω ένα πρότυπο πρόγραμμα το οποίο περιλαμβάνει τον βασικό κώδικα για τη δημιουργία μιας 2D εφαρμογής. Θα κρατήσουμε αυτόν τον κώδικα ως βάση για την ανάπτυξη των προγραμμάτων μας.

Πρόκειται για μια εφαρμογή (applet) που έχει τίτλο SimpleJoglApp. Ακολουθεί η βασική κλάση η οποία περιλαμβάνει τη main. Θα σχολιάσω τμηματικά τον κώδικα ώστε να γίνει κατανοητή η λειτουργία του κάθε κομματιού.

Ακολουθεί η βασική κλάση η οποία περιλαμβάνει τη main. Στις πρώτες σειρές εισάγουμε τις απαραίτητες βιβλιοθήκες για να τρέξει το πρόγραμμα μας:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.media.opengl.*;
```

Η κλάση αυτή σχεδιάζει ένα JFrame (παράθυρο) μέσα στο οποίο θα δημιουργηθεί η εφαρμογή μας. Η πρώτη μέθοδος που καλείται είναι η main η οποία δημιουργεί το applet μας και φροντίζει ότι σχεδιάζουμε να εμφανίζεται στην οθόνη.

```
/**
 * This is a basic JOGL app. Feel free to
 * reuse this code or modify it.
 */
public class SimpleJoglApp extends JFrame {

  public static void main(String[] args) {
    final SimpleJoglApp app = new SimpleJoglApp();

    // show what we've done
    SwingUtilities.invokeLater (
      new Runnable() {
        public void run() {
          app.setVisible(true);
        }
      }
    );
  }

  public SimpleJoglApp() {
    //set the JFrame title
    super("Simple JOGL Application");

    //kill the process when the JFrame is closed
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //only three JOGL lines of code ... and here they are
    GLCapabilities glcaps = new GLCapabilities();
    GLCanvas glcanvas = new GLCanvas();
     glcanvas.addGLEventListener(new SimpleGLEventListener());

    //add the GLCanvas just like we would any Component
```

```
    getContentPane().add(glcanvas, BorderLayout.CENTER);
    setSize(500, 300);

    //center the JFrame on the screen
    centerWindow(this);
  }
```

Στη μέθοδο SimpleJoglApp δημιουργούμε έναν GLCanvas στον οποίο κάνουμε add έναν GLEventListener. Με αυτό τον τρόπο όταν ο GLEventListener λαμβάνει μια κλήση από τον GLDrawable (ο οποίος υλοποιείται στην επόμενη κλάση SimpleGLEventListener) εμφανίζει ότι αυτός προστάζει να σχεδιαστεί. Επίσης, ακολουθούν οι απαραίτητες ρυθμίσεις για την εμφάνιση του παραθύρου στο κέντρο της οθόνης του υπολογιστή με τη βοήθεια της μεθόδου centerWindow.

```
  public void centerWindow(Component frame) {
      Dimension
      screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize  = frame.getSize();

if (frameSize.width  > screenSize.width )
      frameSize.width  = screenSize.width;
if (frameSize.height > screenSize.height)
      frameSize.height = screenSize.height;

    frame.setLocation (
      (screenSize.width  - frameSize.width ) >> 1,
      (screenSize.height - frameSize.height) >> 1
    );
  }
}
```

Οι τρις σειρές στη μέθοδο SimpleJoglApp αποτελούν την βάση του Jogl προγράμματος. Οι εντολές:

- GLCapabilities glcaps = new GLCapabilities():

καθορίζει ποια γραφικά χαρακτηριστικά της OpenGL είναι διαθέσιμα στις βιβλιοθήκες της JOGL και στο JVM. Στην επόμενη σειρά:

- GLCanvas glcanvas = new GLCanvas():

δημιουργούμε τον καμβά μέσα στον οποίο θα σχεδιαστεί το 2D σκηνικό μας. Όπως αναφέραμε και παραπάνω η κλάση αυτή υλοποιεί την GLDrawable, την βασική διασύνδεση χρήστη. Τέλος με την εντολή:

- glcanvas.addGLEventListener(new SimpleGLEventListener():

προσθέτουμε έναν GLEventListener στο GLCanvas. Η υλοποίηση του GLEventListener, που στο παράδειγμά μας είναι ο SimpleGLEventListener αναλαμβάνει να ζωγραφίσει στο πρόγραμμα ότι επιθυμεί ο χρήστης με τη βοήθεια των μεθόδων που αναφέρθηκαν παραπάνω (display(), displayChanged(), init(), reshape()).

Στο σημείο αυτό έχει ολοκληρωθεί ο κώδικας της πρώτης κλάσης και ακολουθεί η επόμενη κλάση SimpleGLEventListener στην οποία υπάρχει ο περισσότερος κώδικας σε JOGL:

```
import java.awt.*;
import java.awt.event.*;
import javax.media.opengl.*;
import javax.media.opengl.glu.GLU;
/**
```

```
* For our purposes only two of the
* GLEventListeners matter. Those would
* be init() and display().
*/
public class SimpleGLEventListener implements GLEventListener
{
/**
* Take care of initialization here.
*/
```

Η πρώτη μέθοδος που υλοποιούμε είναι η init στην οποία γίνεται η αρχικοποίηση του προγράμματός μας. Οι εντολές που θα γράψουμε στη μέθοδο αυτή θα εκτελεστούν μόνο μία φορά, στην έναρξη του προγράμματος μας.

```
 public void init(GLAutoDrawable gld) {
      }
```

Η επόμενη μέθοδος - display είναι υπεύθυνη για τον σχεδιασμό των αντικειμένων. Η μέθοδος αυτή αντίθετα από την init εκτελείται στην αρχή για να σχεδιάσει το σκηνικό μας, αλλά και από τον Animator κάθε φορά που μια αλλαγή συμβαίνει στο σκηνικό.

```
/**
* Take care of drawing here.
*/
public void display(GLAutoDrawable drawable) {
      }
```

Τόσο η init(), όσο και η display() δεν περιέχουν κώδικα άρα το μόνο που κάνει το πρόγραμμά μας είναι να εμφανίζει ένα μαύρο παράθυρο το οποίο αποτελεί τον καμβά στον οποίο μπορούμε να ζωγραφίσουμε το 3D σκηνικό μας.

```
/**
* Called when the GLDrawable (GLCanvas
* or GLJPanel) has changed in size. We
* won't need this, but you may eventually
* need it -- just not yet.
*/

public void reshape(GLAutoDrawable drawable, int x, int y, int width, int height) {
    }

/**
* If the display depth is changed while the
* program is running this method is called.
* Nowadays this doesn't happen much, unless
* a programmer has his program do it.
*/
    public void displayChanged(GLAutoDrawable drawable, boolean modeChanged,
boolean deviceChanged) {
    }
}
```

Τα παραπάνω δύο προγράμματα αποτελούν τον σκελετό πάνω στον οποίο ο προγραμματιστής, όπως θα δούμε αργότερα, μπορεί να χτίσει το 3D σκηνικό του. Ότι ακολουθήσει θα έχει ως βάση τα προγράμματα αυτά και για αυτό το λόγο δεν θα αναφερθεί ξανά ολόκληρος ο κώδικας, παρά μόνο τα κομμάτια που αποτελούν επιπρόσθετη πληροφορία.

3.2. Βασικότερες Συναρτήσεις

Στην παρούσα ενότητα γίνεται μια αναφορά στις βασικότερες συναρτήσεις της JOGL καθώς και το πού τις συναντάμε μέσα σε ένα πρόγραμμα. Τα κομμάτια κώδικα που ακολουθούν αποτελούν τα εργαλεία που έχει ο χρήστης για την ανάπτυξη της εφαρμογής του. Είναι εντολές οι οποίες τις περισσότερες φορές γράφονται στην init() και την display(), τις βασικότερες μεθόδους ενός JOGL προγράμματος.

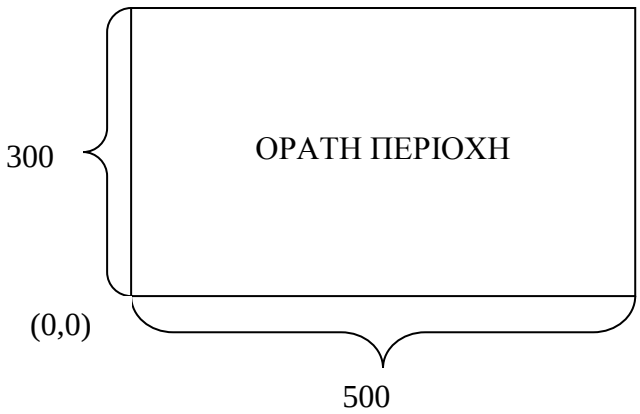
Ακολουθεί ένα παράδειγμα του κώδικα που πάντα περιλαμβάνεται στην init(). Ορισμένες από τις εντολές και τις συναρτήσεις που μας βοηθούν να αρχικοποιήσουμε το πρόγραμμα μας αναλύονται παρακάτω:

- GLU glu = new GLU();
  GL gl = gld.getGL();

  Οι δύο πρώτες γραμμές κώδικα δημιουργούν ένα αντίγραφο του τρέχοντος GL αντικειμένου καθώς και του GLU αντικειμένου πάνω στο οποίο θα δημιουργηθεί η 3D σκηνή μας. Οι δύο αυτές εντολές συναντούνται πρώτες τόσο στην init() όσο και στην display().

- glViewPort(int x, int y, int width, int weight):
  Η συνάρτηση αυτή καθορίζει την περιοχή της οθόνης που μπορεί να δει ο χρήστης. Τα x και y καθορίζουν (σε pixel) την κάτω αριστερή γωνία του ορθογωνίου παραλληλόγραμμου που αποτελεί την ορατή περιοχή, ενώ τα width και weight καθορίζουν το μήκος και το ύψος του.

  π.χ. με την εντολή glViewport(0, 0, 500, 300); η ορατή περιοχή έχει την παρακάτω μορφή:



Εικόνα 3: Η διαστάσεις και η αρχή των αξόνων της ορατής περιοχής

- gluOrtho2D(double left , double right, double bottom, double top):
  Η μέθοδος αυτή θέτει το σύστημα συντεταγμένων που επιθυμεί να χρησιμοποιήσει ο χρήστης και είναι αυτό που εμφανίζεται στο GLCanvas. Τις περισσότερες φορές για λόγους ευκολίας οι τιμές των συντεταγμένων που θέτουμε είναι ίδιες με αυτές της glViwePort.

  Στο παράδειγμά μας είναι: gluOrtho2D(0.0, 500.0, 0.0, 300.0);

- glLineWidth(float width):

  Στο σύστημα συντεταγμένων που χρησιμοποιούμε ζωγραφίζουμε συνεχώς γραμμές ώστε να απεικονίσουμε τα αντικείμενά μας. Η συνάρτηση αυτή καθορίζει το πάχος της γραμμής στην ανάλυση της οθόνης μας και όχι, όπως αρκετοί νομίζουν, το πάχος της γραμμής στο σύστημα συντεταγμένων. Για αυτό το λόγο βλέπουμε ότι όσο και να μεγαλώνουμε το παράθυρο της εφαρμογής μας τα αντικείμενα παραμένουν στο ίδιο μέγεθος.

- glClearColor(float red, float green, float blue, float alpha):

  Η JOGL αφήνει τον χρήστη να επιλέξει το χρώμα με το οποίο ο καμβάς είναι καθαρός, δηλαδή το χρώμα που έχει ο GLCanvas πριν από τη σχεδίαση ή κάθε φορά που επιλέγουμε να τον καθαρίσουμε. Το χρώμα αυτό παράγεται από τη μείξη των 3 χρωμάτων και οι τιμές τους κυμαίνονται ανάμεσα στο 0.0 και 1.0 (διαιρώντας την τιμή κάθε χρώματος με το 255), καθώς και από τον παράγοντα «διαφάνεια» που δηλώνεται από το χρώμα alpha.

  π.χ. RGB(124, 11, 183) ⇨ glClearColor(0.486f, 0.043f, 0.718f, 1.0f)

  Από τη στιγμή που ο χρήστης θα επιλέξει ένα χρώμα μπορεί οποιαδήποτε στιγμή στο πρόγραμμά του να καθαρίσει το GLCanvas με την εντολή: glClear(GL.GL_COLOR_BUFFER_BIT)

- glColor3f(float red, float green, float blue):

  Είναι η συνάρτηση με την οποία ο χρήστης επιλέγει το χρώμα με το οποίο θα χρωματίσει ένα αντικείμενο στο 3D σκηνικό του. Η επιλογή του χρώματος γίνεται όπως πριν με τις τιμές των 3 χρωμάτων να κυμαίνονται ανάμεσα στο 0.0 και στο 1.0.

Οι τρις πρώτες συναρτήσεις, οι glViewPort, gluOrtho2D και glLineWidth, είναι προορατικές και δεν είναι απαραίτητη η τροποποίησή τους. Μπορούμε κάλλιστα να τις αφήσουμε στις προεπιλεγμένες τιμές χωρίς να υπάρξει κάποια επίπτωση στο πρόγραμμά μας. Οι υπόλοιπες δύο βασικές εντολές της init() που δεν αναφέραμε παραπάνω περιγράφονται στην επόμενη ενότητα των 3D γραφικών.

4. Τρισδιάστατα Σημεία

Για τον σχεδιασμό των αντικειμένων τόσο η OpenGL, όσο και η JOGL χρησιμοποιούν πίνακες. Κάθε εικόνα που δημιουργεί η JOGL είναι στην ουσία ένα σύνολο από πίνακες που αλληλεπιδρούν μεταξύ τους με έναν τρόπο καθορισμένο από την OpenGL. Παραδείγματος χάριν ο παρακάτω πίνακας αναπαριστά τα εξής τέσσερα σημεία σε ένα τρισδιάστατο σύστημα συντεταγμένων:

- (12, 5, -10)
- (3, 14, 0)
- (34, 13, 4)
- (-5, -1, -13)

$$\begin{bmatrix} 12 & 5 & -10 \\ 3 & 14 & 0 \\ 34 & 13 & 4 \\ -5 & -1 & -13 \end{bmatrix}$$

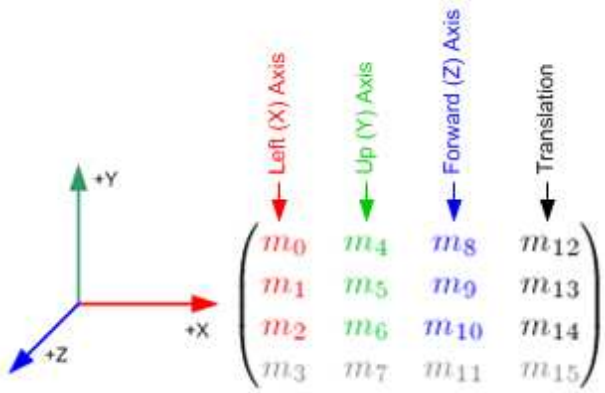
Αξίζει να αναφέρουμε μια στοιχειώδη διαφορά στο τρόπο που η Java και η C αποθηκεύουν πολυδιάστατους πίνακες. Η Java από τεχνικής άποψης δεν υποστηρίζει πολυδιάστατους πίνακες, αλλά τους αντιμετωπίζει σαν πίνακες αποτελούμενους από επιμέρους πίνακες. Αντίθετα στην C ένας δισδιάστατος πίνακας αποθηκεύεται σαν μια ενιαία οντότητα στης οποίας τα στοιχεία μπορούμε να ανατρέξουμε με τη χρήση ενός και μόνο δείκτη.

Ωστόσο, για την απλοποίηση της JOGL, ο χρήστης δεν χρειάζεται να χειρίζεται ο ίδιος πίνακες καθώς υπάρχουν μέθοδοι που κάνουν την δουλεία αυτή. Η μέθοδος:

- glMatrixMode(mode):
  είναι αυτή που θέτει τον τρόπο χειρισμού των πινάκων σε ένα πρόγραμμα JOGL. Οι δύο βασικοί τρόποι χειρισμού των πινάκων στην JOGL είναι οι GL_PROJECTION και GL_MODELVIEW. [3]

  Σκεφτόμαστε το GL_PROJECTION σαν τη λειτουργία μιας κάμερας που κρατάει ο χρήστης. Δηλαδή, πρόκειται για την προβολή του 3D αντικειμένου που αντιλαμβάνεται το μάτι του χρήστη στην οθόνη του υπολογιστή, η οποία διαθέτει 2 διαστάσεις. Επίσης, καθορίζει ποια αντικείμενα ή τμήματα αντικειμένου δεν θα εμφανιστούν στην οθόνη διότι δεν είναι ορατά από τον χρήστη.

  Η λειτουργία GL_MODELVIEW βοηθάει κυρίως στον μετασχηματισμό των αντικειμένων, που συμβαίνει όταν μετακινούμαστε στο 3D σκηνικό. Συνδυάζει την μήτρα προβολής του αντικειμένου με τη μήτρα μοντελοποίησης σε μια ενιαία μήτρα. Όταν χρειάζεται να μετακινήσουμε την κάμερα, δηλαδή το σημείο από το οποίο βλέπει ο χρήστης, εφαρμόσουμε στην μήτρα αυτή τον αντίστροφο μετασχηματισμό. Ο μετασχηματισμός αντικειμένων αναλύεται σε επόμενη ενότητα.



Εικόνα 4: Παράδειγμα μιας μήτρας τύπου GL_MODELVIEW. Η τελευταία στήλη είναι η στήλη του μετασχηματισμού [10]

Υπάρχουν κάποιες μέθοδοι οι οποίες μπορούν να κληθούν μόνο σε λειτουργία GL_PROJECTION, αυτές είναι οι:

- gluPerspective()
- glFrustum() //commonly misspelled glFrustrum
- glOrtho()
- gluOrtho2D
- glLoadIdentity

Οπότε πριν χρησιμοποιήσουμε οποιαδήποτε από τις παραπάνω μεθόδους θα πρέπει να αλλάξουμε τη λειτουργία σε GL_PROJECTION, καθώς ο GL_MODELVIEW είναι ο προεπιλεγμένος τρόπος χειρισμού πινάκων.

Όταν καλούμε τη μέθοδο glMatrixMode() η επόμενη ακριβώς κλήση είναι η:

- glLoadIdentity():
  η οποία αρχικοποιεί τον πίνακα που μόλις φορτώσαμε στην ταυτότητά του. Η ταυτότητα είναι διαφορετική για κάθε πίνακα και συνήθως έχει την παρακάτω μορφή: [3]

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

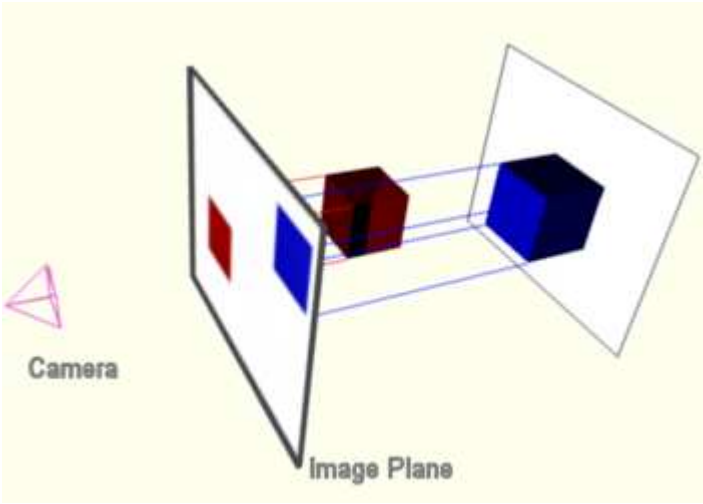
5. Προβολή - Projection

5.1. Θεωρία

Αφού θέσουμε τη λειτουργία σε GL_PROJECTION και καλέσουμε την glMatrixMode() η αμέσως επόμενη κίνηση είναι να καθορίσουμε το είδος της «προβολής». Με αυτόν τον τρόπο θέτουμε στον κόσμο μας κάποια όρια, έτσι ώστε να γνωρίζει η JOGL τι θα πρέπει να σχεδιάσει, καθώς και τον τρόπο με τον οποίο θα φαίνονται τα αντικείμενά μας. Έχουμε 2 είδη προβολών:

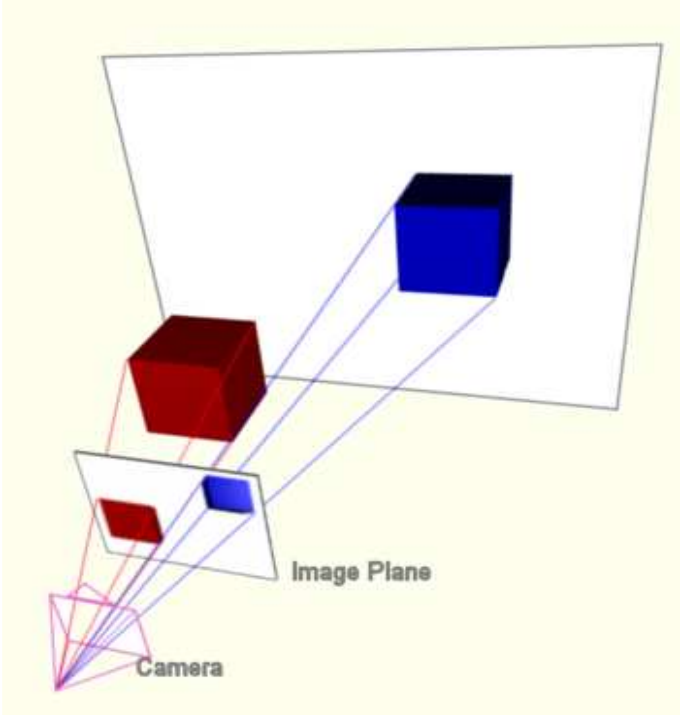
- Orthographic (or parallel):
  Στην ορθογραφική προβολή τα αντικείμενα δεν έχουν διαφορά ανάλογη της απόστασης από την κάμερα. Για αυτό το λόγο τα αντικείμενα φαίνονται επίπεδα και δεν μπορεί κάποιος να καταλάβει από το μέγεθος τους την απόστασή τους από την κάμερα. Παρόλο που το αποτέλεσμα της είναι λιγότερο ρεαλιστικό βρίσκει εφαρμογή σε CAD συστήματα και 2D παιχνίδια. Οι συναρτήσεις που μας βοηθούν στην εφαρμογή της Orthographic προβολής είναι οι:

  - glOrtho(double left, double right, double bottom, double top, double nearVal, double farVal): Οι τιμές left-right καθορίζουν τις συντεταγμένες στον άξονα x, οι τιμές bottom-top και nearVal-farVal καθορίζουν τις τιμές στους άξονες y και z
  - gluOrtho2D(double left, double right, double bottom, double top): που χρησιμοποιείται στα 2D γραφικά με τον τρόπο που αναφέρθηκε παραπάνω



Εικόνα 5: Ορθογραφική προβολή αντικειμένου [11]

- Perspective:
  Στην προοπτική προβολή ο τρόπος που φαίνονται τα αντικείμενα αλλάζει ανάλογα με την απόσταση τους από την κάμερα. Σημαντικό χαρακτηριστικό της προοπτικής προβολής είναι η κεντρική σμίκρυνση, δηλαδή η μείωση του μεγέθους των αντικειμένων κατά έναν παράγοντα που είναι ανάλογος της απόστασης του αντικειμένου από την κάμερα.

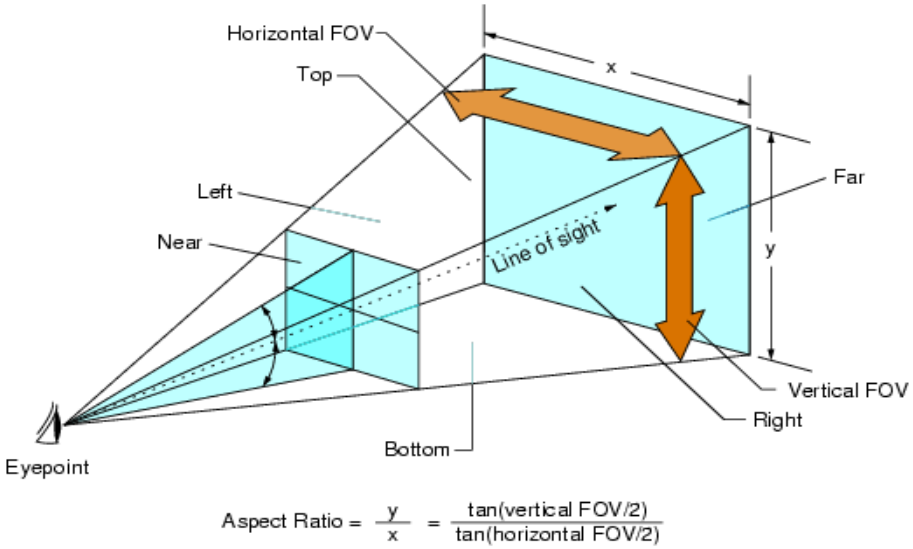


Εικόνα 6: Προοπτική προβολή αντικειμένου [11]

Ο τρόπος με τον οποίο ο χρήστης μπορεί να καθορίσει τα όρια του οπτικού του πεδίου, δηλαδή ποιο μέρος του 3D σκηνικού μπορεί να δει κάθε φορά, είναι η δημιουργία μιας οριοθετημένης περιοχής που ονομάζεται «frustum». Όπως φαίνεται και στην εικόνα 7 μπορούμε να φανταστούμε την περιοχή αυτή σαν μια πυραμίδα με την κορυφή της προς την κάμερα. Ανάλογα με τη θέση του αντικειμένου σε σχέση με αυτή την πυραμίδα πραγματοποιείται και η ανάλογη παραμόρφωση. Η περιοχή από την κάμερα και μέχρι κοντινό όριο «Near» δεν είναι ορατή από τον χρήστη. Ορατά είναι μόνο αντικείμενα που

βρίσκονται ανάμεσα στα όρια «Near», «Far», καθώς και στο πλαίσιο που δημιουργείται από τις μεταβλητές «Top», «Bottom», «Left», «Right». [4] Η OpenGL μας παρέχει συναρτήσεις να δημιουργήσουμε ένα «frustum»:

- void gluFrustum (double left, double right, double bottom, double top, double nearl, double far)



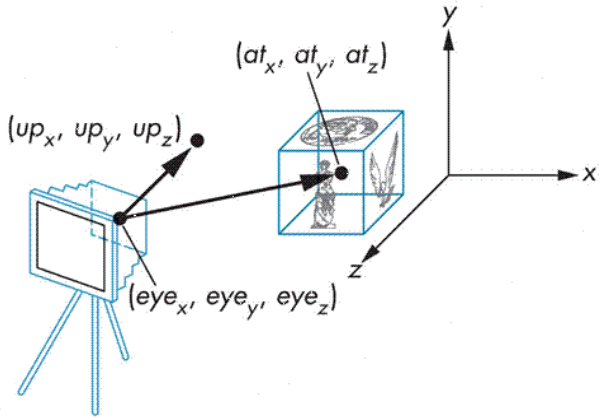
Εικόνα 7: Η ορατή περιοχή του χρήστη

Επίσης μας παρέχει και μια συνάρτηση για την δημιουργία μιας προβολής η οποία υπολογίζει αυτόματα το «frustum» για εμάς. Η συνάρτηση αυτή καλείται μόνο σε λειτουργία GL_PROJECTION.

- gluPerspective(double fovy, double aspect, double near, double far): Όπως γίνεται ορατό και στην παραπάνω εικόνα η παράμετρος FOVy καθορίζει το πεδίο ορατότητας του χρήστη και είναι η γωνία προβολής στον άξονα y, για μια καλή προβολή στην οθόνη ενός υπολογιστή οι 60 βαθμοί αρκούν. Η παράμετρος aspect στην ουσία είναι η προβολή στον άξονα x που ισούται με x(width)/y(height), δηλαδή το πλάτος της ορατής περιοχής δια το ύψος. Τέλος, οτιδήποτε βρίσκετε πιο κοντά από την παράμετρο «Near» και πιο μακριά από την παράμετρο «Far» δεν είναι ορατό από τον χρήστη. Οι παράμετροι αυτοί πρέπει να είναι θετικοί αριθμού και οι παράμετρος «Far» πρέπει να είναι μεγαλύτερος αριθμός από το «Near».

Το «frustum» που δημιουργούμε με τη χρήση της glPerspective() τοποθετείται ανάλογα με τη θέση της κάμερας. Η θέση της κάμερας καθορίζεται από την παρακάτω μέθοδο η οποία καλείται σε λειτουργία GL_MODELVIEW:

- gluLookAt(double eyeX, double eyeY, double eyeZ, double atX, double atY, double atZ, double upX, double upY, double upZ): Όπως φαίνεται και στην παρακάτω εικόνα, για να καθορίσουμε τη θέση της κάμερας χρησιμοποιούμε 3 σημεία στον τρισδιάστατο κόσμο. Το πρώτο είναι το «eye» το οποίο αποτελεί την θέση της κάμερας, το σημείο «at» είναι αυτό στο οποίο κοιτάει ο χρήστης, δηλαδή το κέντρο της σκηνής του, και το σημείο «up» καθορίζει την κατεύθυνση του διανύσματος που «δείχνει» προς τα επάνω.



Εικόνα 8: Τα διανύσματα που καθορίζουν τη θέση της κάμερας

Είναι σημαντικό να τονίσουμε ότι μόνο η μέθοδος gluLookAt() παίρνει σαν παραμέτρους συντεταγμένες. Στην μέθοδο gluPerspective() η παράμετροι «Near» και «Far» αντιπροσωπεύουν αποστάσεις από το σημείο «eye» στην κατεύθυνση που δείχνει το σημείο «at» και για αυτό οι τιμές που παίρνουν είναι πραγματικοί αριθμοί και όχι συντεταγμένες σημείου.

5.2. Το πρόγραμμα Planes

Ακολουθεί ένα απλό πρόγραμμα στο οποίο παρουσιάζονται οι εντολές οι οποίες συνήθως περιλαμβάνονται στις δύο κύριες μεθόδους της JOGL και βοηθούν στη ρύθμιση βασικών παραμέτρων του προγράμματος. Το όνομα της εφαρμογής είναι Planes (επίπεδα) και σχεδιάζει τρία παραλληλόγραμμα που βρίσκονται σε διαφορετική απόσταση από την κάμερα το καθένα.

Ο κώδικας που περιέχεται στην κύρια κλάση, με την ονομασία Planes, είναι ίδιος με τον κώδικα του προγράμματος SimpleJoglApplet. Οι μοναδικές διαφορές είναι το όνομα της κλάσης, που πλέον είναι Planes, καθώς και το όνομα του EventListener που κάνουμε add στο GLCanvas, το οποίο πλέον συμπίπτει με το όνομα της επόμενης κλάσεις, και είναι PlanesView. Έτσι λοιπόν η γραμμή του αρχικού προγράμματος:

```
glcanvas.addGLEventListener(new SimpleGLEventListener());
```

Αντικαθίσταται από την παρακάτω:

```
glcanvas.addGLEventListener(new PlanesView());
```

Ακολουθεί η δευτερεύουσα κλάση με την ονομασία PlanesView στην οποία περιλαμβάνονται οι μέθοδοι init() και display(), όπου υπάρχει και περισσότερος κώδικας σε JOGL:

```
import java.awt.*;
import javax.media.opengl.*;
import javax.media.opengl.glu.GLU;

/**
* For our purposes only two of the
* GLEventListeners matter. Those would
* be init() and display().
*/
```

```
public class PlanesView implements GLEventListener
{
```

Οι δύο απαραίτητες εντολές για την δημιουργία των GL και GLUT αντικειμένων πάνω στα οποία θα σχεδιάσουμε το σκηνικό μας είναι αυτές που αναγράφονται πρώτες στην init() αλλά και στην display(), όπως θα δούμε αργότερα. Με την επόμενη μέθοδο καθορίζουμε το default χρώμα το οποίο θα έχει το σκηνικό μας. Τέλος, στην init(), όπου ο τρόπος προβολής είναι ModelView by default, καθορίζουμε τη θέση της κάμερας με τη μέθοδο gluLookAt().

```
/**
* Take care of initialization here.
*/

 public void init(GLAutoDrawable gld) {
//The mode is GL.GL_MODELVIEW by default
//We will also use the default ViewPort
    GLU glu = new GLU();
    GL gl = gld.getGL();

    gl.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    //Define points for eye, at and up.
    //This is your camera. It ALWAYS goes
    //in the GL_MODELVIEW matrix.
    glu.gluLookAt(
    20, 18, 0,
    20, 18, 30,
    0, 1, 0
    );

    }
```

Στη μέθοδο display(), αλλάζουμε τη λειτουργία MatrixMode σε PROJECTION ώστε να καθορίσουμε το είδος της προβολής που επιθυμούμε (parallel ή perspective). Προηγουμένως θα πρέπει να αρχικοποιήσουμε τον πίνακα προβολής (projection matrix), στον οποίο αποθηκεύονται τα όρια της ορατής περιοχής, με τις τιμές του μοναδιαίου πίνακα. Η εντολή που το πετυχαίνει αυτό είναι η gl.glLoadIdentity().

```
/**
* Take care of drawing here.
*/
public void display(GLAutoDrawable gld) {

    float red = 1.0f;
    float green = 0.0f;
    float blue = 0.5f;

    GL gl = gld.getGL();
    GLU glu = new GLU();

    //We're changing the mode to GL.GL_PROJECTION
    //the only JOGL methods that should be called
    //while using the GL_PROJECTION matrix are:
    // gluPerspective
    // glFrustum
    // glOrtho
    // gluOrtho2D
    // glLoadIdentity
    // glLoadMatrix
    gl.glMatrixMode(GL.GL_PROJECTION);
    gl.glLoadIdentity();
```

Στη συνέχεια, αφού υπολογίσουμε την τιμή του aspect σύμφωνα με τις διαστάσεις του σκηνικού μας (width/height) υπολογίζουμε το aspect με τις κατάλληλες πράξεις. Αφού

καλέσουμε τη μέθοδο gluPerspective για να καθορίζουμε την ορατή περιοχή μας, αλλάζουμε το MatrixMode σε ModelView ώστε να εμφανιστούν σωστά τα τετράγωνα που θέλουμε να σχεδιάσουμε. Στην εικόνα που ακολουθεί μετά τον κώδικα γίνεται εμφανές ότι παρότι το μέγεθος των αντικειμένων είναι ίδιο αυτά φαίνονται διαφορετικού μεγέθους ανάλογα με το βάθος στο οποίο βρίσκεται το καθένα.

```
// Aspect is width/height
double w = ((Component) gld).getWidth();
double h = ((Component) gld).getHeight();
double aspect = w/h;
//Notice we're not using gluOrtho2D.
//When using gluPerspective near and far need
//to be positive.
//The arguments are:
//fovy, aspect, near, far
glu.gluPerspective(60.0, aspect, 25.0, 55.0);

gl.glMatrixMode(GL.GL_VIEWPORT);
//we don't want to initialize the GL_MODELVIEW
//using gl.glLoadIdentity() this time. It has
//settings from the init() that we wish to keep.
```

Με τις επόμενες δύο εντολές αρχικοποιούμε το χρώμα του σκηνικού μας και στη συνέχεια ορίζουμε το χρώμα που θα έχουν τα αντικείμενα που θα σχεδιάσουμε αμέσως μετά.

```
gl.glClear(GL.GL_COLOR_BUFFER_BIT);
gl.glColor3f(red, green, blue);
```

Από εδώ και κάτω υπάρχει ο κώδικας που σχεδιάζει τα παραλληλόγραμμα σε τρία διαφορετικά επίπεδα. Περισσότερες λεπτομέρειες υπάρχουν την επόμενη ενότητα «*Σχεδιασμός Αντικειμένων – Οι κλάσεις GL, GLU & GLUT*».

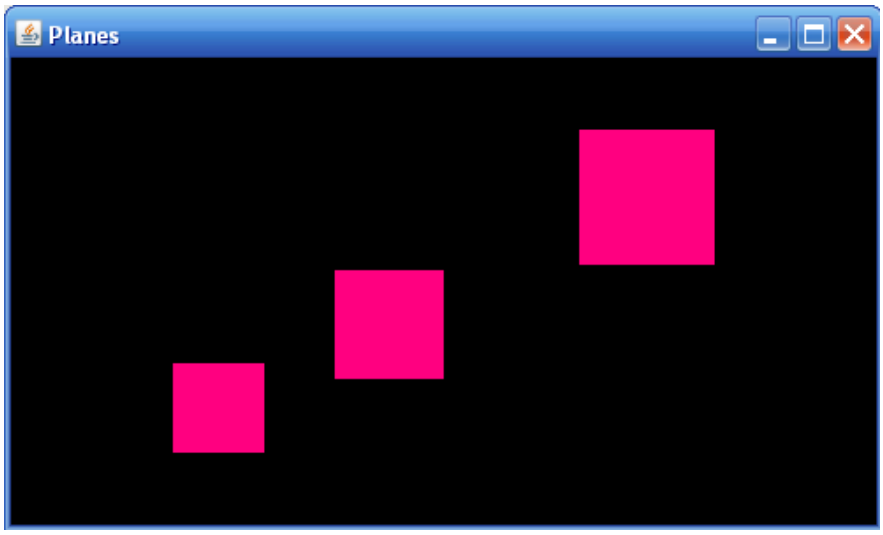
```
gl.glBegin(GL.GL_QUADS);
//notice that the three squares are
//exactly the same size. They appear
//different on screen because of
//perspective
//1st Plan
gl.glVertex3i(0, 30, 30);
gl.glVertex3i(10, 30, 30);
gl.glVertex3i(10, 20, 30);
gl.glVertex3i(0, 20, 30);
//2nd Plane
gl.glVertex3i(20, 20, 37);
gl.glVertex3i(30, 20, 37);
gl.glVertex3i(30, 10, 37);
gl.glVertex3i(20, 10, 37);
//3rd Plane
gl.glVertex3i(40, 10, 45);
gl.glVertex3i(50, 10, 45);
gl.glVertex3i(50, 0, 45);
gl.glVertex3i(40, 0, 45);

gl.glEnd();
}
```

Οι επόμενες μέθοδοι προς το παρόν παραμένουν κενές καθώς δεν παρουσιάζουν κάποια χρησιμότητα στο πρόγραμμα που μελετάμε.

```
/**
* Called when the GLDrawable (GLCanvas
* or GLJPanel) has changed in size. We
* won't need this, but you may eventually
```

```
* need it -- just not yet.
*/
public void reshape(GLAutoDrawable drawable, int x, int y, int width, int height) {
    }
/**
* If the display depth is changed while the
* program is running this method is called.
* Nowadays this doesn't happen much, unless
* a programmer has his program do it.
*/
 public void displayChanged(GLAutoDrawable drawable, boolean modeChanged, boolean
deviceChanged) {
    }
}
```

Το αποτέλεσμα του προγράμματος φαίνεται στην εικόνα 9.



Εικόνα 9: Αντικείμενα ίδιου μεγέθους αλλά διαφορετικού βάθους με τη χρήση της gluPerspective()

6. Σχεδιασμός Αντικειμένων – Οι κλάσεις GL, GLU & GLUT

6.1. Γενικά Στοιχεία

Όπως ήδη αναφέρθηκε η γλώσσα C αποτελεί την βάση της Java OpenGL, η οποία χρησιμοποιεί το JNI για να επικοινωνήσει με τις βασικές βιβλιοθήκες C OpenGL. Οι βιβλιοθήκες αυτές είναι οργανωμένες σε header files. Τα κυριότερα header files που είναι απαραίτητα για τον σχεδιασμό των αντικειμένων είναι τα “gl.h”, “glu.h” και “glut.h”.

Το “gl.h” είναι το κυρίως header file που χρησιμοποιείται σε όλα τα προγράμματα OpenGL, επίσης το ”glu.h” έχει την ίδια χρησιμότητα με τη μόνη διαφορά ότι δεν είναι απαραίτητο σε ιδιαίτερα απλοποιημένα προγράμματα. Τα δύο αυτά header files παρέχουν όλες τις χρήσιμες συναρτήσεις.

Η C δεν υποστηρίζει τη δημιουργία παραθύρων, δηλαδή δεν περιέχει Windowing Toolkit graphic libraries, ενώ η Java διαθέτει τα πακέτα γραφικών AWT και Swing. Όπως είναι φυσικό η OpenGL θα πρέπει να σχεδιαστεί σε παράθυρα, κάτι που δεν υποστηρίζει η C.

Εξαιτίας αυτού του γεγονότος δημιουργήθηκε η βιβλιοθήκη GLUT. Πρόκειται για ένα απλό windowing toolkit που επιτρέπει τη δημιουργία παραθύρων καθώς και την ύπαρξη γεγονότων καθοδηγούμενων από το ποντίκι και το πληκτρολόγιο. Το πακέτο GLUT δεν χρειάζεται σε προγραμματιστές JOGL οι οποίοι διαθέτουν ήδη τα AWT και Swing. Η JOGL επίσης διαθέτει κάποιες συναρτήσεις GLUT που δημιουργούν προκατασκευασμένα σχήματα. [3]

Όπως έγινε φανερό και στα παραδείγματα κώδικα που έχουν αναφερθεί μέχρι τώρα οι μέθοδοι glBegin() και glEnd() είναι αυτές που περικλείουν όλες τις εντολές σχεδιασμού. Οι εντολές που μας βοηθάνε να σχεδιάσουμε είναι του τύπου glVertex(). Η εντολή glBegin() καλείται πρώτη και παίρνει ως παράμετρο σταθερές τιμές της κλάσης GL, ανάλογα με το σχήμα που επιθυμούμε να σχεδιάσουμε. Η τιμής της παραμέτρου αυτής δείχνει τον τρόπο σύνδεσης των κορυφών που καθορίζονται από την εντολή glVertex(). Ακολουθεί απόσπασμα κώδικα που σχεδιάζει ένα τετράγωνο στον τρισδιάστατο χώρο:

```
gl.glBegin(GL.GL_QUADS);
gl.glVertex3i(0, 30, 30);
gl.glVertex3i(10, 30, 30);
gl.glVertex3i(10, 20, 30);
gl.glVertex3i(0, 20, 30);
gl.glEnd();
```

Οι κυριότερες εκδοχές της μεθόδου glVertex είναι οι παρακάτω:

- glVertex2d(double x, double y)
- glVertex2di(int x, int y)

Σχεδιασμός 2D αντικειμένων

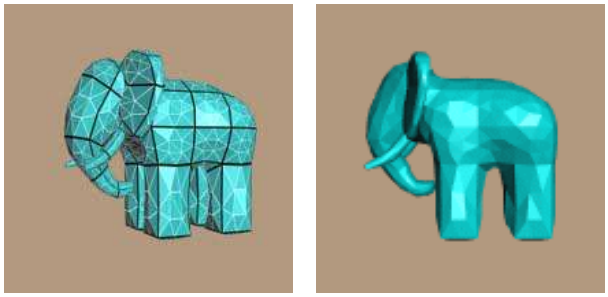
- glVertex3d(double x, double y, double z)
- glVertex3di(int x, int y, int z)

Σχεδιασμός 3D αντικειμένων

Μερικά από τα σημαντικότερα σχήματα που παίρνει σαν παράμετρο η glBegin() είναι:

- GL_POINTS
- GL_LINES
- GL_LINE_STRIP
- GL_LINE_LOOP
- GL_TRIANGLES
- GL_TRIANGLES_STRIP
- GL_TRIANGLE_FAN
- GL_QUADS
- GL_QUADS_STRIP
- GL_POLYGON

Όπως αναφέρθηκε και στην εισαγωγή, τα πολύπλοκα σχήματα στην OpenGL σχεδιάζονται σαν ομάδες από επιμέρους πρωταρχικά πολύγωνα. Με τη βοήθεια μιας διαδικασίας που ονομάζεται «Tessellation» τα πολύπλοκα πολύγωνα διαχωρίζονται στα βασικά πολύγωνα της OpenGL πριν σχεδιαστούν. Στη συνέχεια η ένωσή τους είναι αυτή που δημιουργεί το τελικό επιθυμητό σχήμα.



Εικόνα 10: Ο σχεδιασμός ενός πολύπλοκου σχήματος από βασικά πολύγωνα της OpenGL

Οι κλάσεις GLU και GLUT παρέχουν κάποια βασικά έτοιμα σχήματα τα οποία μπορεί ο χρήστης να χρησιμοποιήσει στα προγράμματά του τόσο αυτούσια, όσο και για τη σύνθεση πιο πολύπλοκων αντικειμένων.

6.2. Η κλάση GLU

Η κλάση GLU περιέχει μεθόδους για το σχεδιασμό κυλίνδρων, σφαιρών και δίσκων. Όλα αυτά γίνονται με τη χρήση «quadrics». Ακολουθεί ένα απόσπασμα κώδικα που χρησιμοποιεί «quadrics» για το σχεδιασμό μιας σφαίρας:

```
GLUquadric quad = glu.gluNewQuadric();
glu.gluQuadricDrawStyle(quad, GLU.GLU_LINE);
glu.gluSphere(quad, 1.0, 15, 15);
glu.gluDeleteQuadric(quad);
```

Στην πρώτη γραμμή δημιουργούμε ένα GLUquadric αντικείμενο. Το αντικείμενο αυτό βρίσκεται στο πακέτο javax.media.opengl.glu.GLUquadric. Στη συνέχεια χρησιμοποιούμε τη μέθοδο gluQuadricDrawSlyte(GLUquadric quad , GLenum draw), όπου η παράμετρος draw καθορίζει το επιθυμητό στιλ σχεδίασης. Τα διαθέσιμα στιλ σχεδίασης είναι τα: GLU_FILL, GLU_LINE, GLU_SILHOUETTE και GLU_POINT.

Το επόμενο βήμα είναι να σχεδιάσουμε το σχήμα που επιθυμούμε με τη χρήση μιας από τις παρακάτω μεθόδους:

- gluSphere(GLUquadric quad, double radius, int slices, int stacks): Η πρώτη παράμετρος είναι το «quadric» που δημιουργήσαμε προηγουμένως. Η παράμετρος radius καθορίζει την ακτίνα της σφαίρας. Οι παράμετροι slices και stacks μπορούν να παρομοιαστούν με τις γραμμές του γεωγραφικού μήκους και πλάτους αντίστοιχα μιας υδρογείου σφαίρας. Όσο μεγαλύτεροι είναι αυτοί οι αριθμοί τόσο πιο ρεαλιστική θα φαίνεται η σφαίρα.

- gluCylinder(GLUquadric quad, double base, double top, double height, int slices, int stacks): Η μέθοδος αυτή λειτουργεί ακριβώς όπως και η παραπάνω με τη μόνη διαφορά ότι τώρα έχουμε τρις παραμέτρους: την base, η οποία καθορίζει την ακτίνα της βάσης, την top που καθορίζει την ακτίνα του επάνω κύκλου, και την height που καθορίζει το ύψος του κυλίνδρου.

- gluDisk(GLUquadric quad, double inner, double outer, int slices, int loops): Και πάλι η πρώτη παράμετρος που συναντούμε είναι η quad. Οι παράμετροι inner και outer αναπαριστούν το εξωτερικό άκρο και την τρύπα που υπάρχει στο εσωτερικό του δίσκου αντίστοιχα. Η παράμετρος slices είναι τα κομμάτια στα οποία χωρίζεται ο δίσκος, ενώ η παράμετρος rings είναι τα δαχτυλίδια που υπάρχουν στην περιφέρειά του.

Το τελευταίο βήμα είναι να καλέσουμε τη μέθοδο gluDeleteQuadric() η οποία καταστρέφει το «quadric» που χρησιμοποιήσαμε ώστε να παράγουμε το σχήμα. Από τη στιγμή του το «quadric» έχει διαγραφεί δεν μπορούμε πλέον να το χρησιμοποιήσουμε στη σχεδίαση.

6.3. Η κλάση GLUT

Δυστυχώς, η κλάση GLUT προσφέρει στην Java μόνο ένα μέρος των συναρτήσεων που προσφέρει στη C. Οι συναρτήσεις αυτές περιέχουν μεθόδους για την δημιουργία σχημάτων και κειμένου. Πιο αναλυτικά, υποστηρίζει τη δημιουργία κύβων, κώνων, δωδεκάεδρων, σφαιρών και άλλων σχημάτων, καθώς και κειμένου. Τα δύο βήματα για το σχεδιασμό ενός GLUT σχήματος είναι η δημιουργία ενός GLUT αντικειμένου, με τον τρόπο που δημιουργούνται αντικείμενα στη Java, και η χρήση του αντικειμένου αυτού για την κλήση της επιθυμητής GLUT μεθόδου. Το παρακάτω απόσπασμα κώδικα χρησιμοποιείται για τη δημιουργία ενός εικοσαέδρου:

```
GLUT glut = new GLUT();
glut.glutWireIcosahedron();
```

Τα αντικείμενα που δημιουργούμε με αυτόν τον τρόπο εμφανίζονται ακριβώς στο κέντρο της ορατής μας περιοχής. Αργότερα θα αναλύσουμε πώς μπορούμε να τοποθετήσουμε ένα αντικείμενο στην θέση στην οποία επιθυμούμε καθώς και το πώς μπορούμε να μετακινηθούμε μέσα στο 3D σκηνικό μας.

6.4. Το πρόγραμμα GLUObjects

Ακολουθεί η εφαρμογή GLUObjects η οποία δείχνει τον τρόπο σχεδιασμού ενός αντικειμένου την κλάσης GLU στην JOGL. Το κύριο πρόγραμμα GLUObjects που περιλαμβάνει τη main δεν αναγράφεται για λόγους συντομίας καθώς δεν διαφέρει από τις αντίστοιχες κλάσεις που είδαμε παραπάνω. Το πρόγραμμα GLUObjectsView περιγράφεται παρακάτω βήμα – βήμα.

```
import java.awt.*;
import javax.media.opengl.*;
import javax.media.opengl.glu.GLU;
import javax.media.opengl.glu.GLUquadric;

/**
* For our purposes only two of the
* GLEventListeners matter. Those would
* be init() and display().
*/
public class GLUObjectsView implements GLEventListener
{
```

Το παράδειγμα αυτό μας δείχνει ότι η χρήση της συνάρτησης gluPerspective() μπορεί να γίνει στην init() και της gluLookAt() στην display() (αντίθετα με ότι συνέβαινε στα δύο παραπάνω παραδείγματα). Η τακτική αυτή θα γίνει ιδιαίτερα χρήσιμη αργότερα όταν οι τιμές των παραμέτρων της gluLookAt() θα αλλάζουν ώστε να ο χρήστης να κινείται στον χώρο. Προς το παρόν αρκεί να γνωρίζουμε ότι οι δύο αυτές συναρτήσεις μπορούν να κληθούν μέσα από δύο διαφορετικές μεθόδους.

```
/**
* Take care of initialization here.
*/
 public void init(GLAutoDrawable gld) {

//The mode is GL.GL_MODELVIEW by default
//We will also use the default ViewPort
    GLU glu = new GLU();
    GL gl = gld.getGL();

    //We're changing the mode to GL.GL_PROJECTION
    //This is where we set up the camera
    gl.glMatrixMode(GL.GL_PROJECTION);
    gl.glLoadIdentity();

    // Aspect is width/height
    double w = ((Component) gld).getWidth();
    double h = ((Component) gld).getHeight();
    double aspect = w/h;
    //Notice we're not using gluOrtho2D.
    //When using gluPerspective near and far need
    //to be positive.
    //The arguments are:
    //fovy, aspect, near, far
    glu.gluPerspective(60.0, aspect, 2.0, 13.0);

    gl.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

    }
```

Έτσι λοιπόν αλλάζουμε το MatrixMode σε PROJECTION ώστε να καλέσουμε την gluPerspective. Ακολουθεί η μέθοδος display που σχεδιάζει έναν δίσκο στο κέντρο της σκηνής.

```
/**
* Take care of drawing here.
*/
public void display(GLAutoDrawable gld) {

    float red = 0.0f;
    float green = 0.0f;
    float blue = 0.9f;

    GL gl = gld.getGL();
    GLU glu = new GLU();

    //We're changing the mode to GL.GL_PROJECTION
    //the only JOGL methods that should be called
    //while using the GL_PROJECTION matrix are:
    // gluPerspective
    // glFrustum
    // glOrtho
    // gluOrtho2D
    // glLoadIdentity
    // glLoadMatrix
    gl.glMatrixMode(GL.GL_MODELVIEW);
    gl.glLoadIdentity();
```

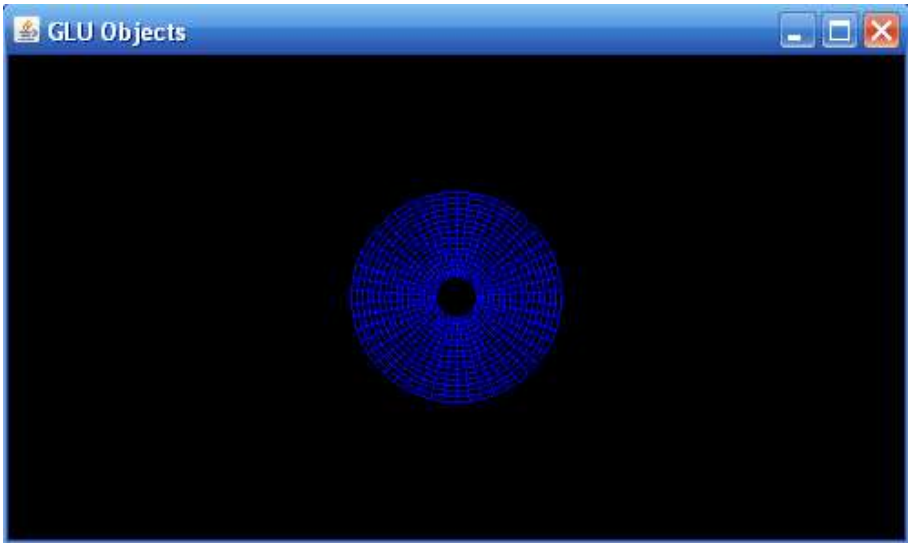
Αλλάζουμε και πάλι το MatrixMode σε MODELVIEW ώστε να καλέσουμε την gluLookAt(). Ακολουθεί ο «καθαρισμός» του σκηνικού και ο ορισμός των χρωμάτων.

```
    //Define points for eye, at and up.
    //This is your camera. It ALWAYS goes
    //in the GL_MODELVIEW matrix.
    glu.gluLookAt(
    0, 0, 4,
    0, 0, 0,
    0, 1, 0
    );

    gl.glClear(GL.GL_COLOR_BUFFER_BIT);

    gl.glColor3f(red, green, blue);
```

Όπως προαναφέραμε και στη θεωρία για να σχεδιάσουμε ένα αντικείμενο της κλάσης GLU πρώτα κατασκευάζουμε ένα Quadric και επιλέγουμε τον τρόπο σχεδιασμού του με τη μέθοδο gluQuadricDrawStyle(). Τέλος, σχεδιάζουμε το αντικείμενό μας, που στην προκειμένη είναι ένας δίσκος, και διαγράφουμε το Quadric που έχουμε κατασκευάσει αφού πλέον δεν το χρειαζόμαστε.

```
    GLUquadric quad = glu.gluNewQuadric();
    glu.gluQuadricDrawStyle(quad, GLU.GLU_LINE);
    glu.gluDisk(quad, 1.0, 0.2, 50, 15);
    glu.gluDeleteQuadric(quad);

    }

/**
* Called when the GLDrawable (GLCanvas
* or GLJPanel) has changed in size. We
* won't need this, but you may eventually
* need it -- just not yet.
*/
public void reshape(GLAutoDrawable drawable, int x, int y, int width, int height) {
    }

/**
* If the display depth is changed while the
* program is running this method is called.
* Nowadays this doesn't happen much, unless
* a programmer has his program do it.
*/
    public void displayChanged(GLAutoDrawable drawable, boolean modeChanged,
    boolean deviceChanged) {
    }
}
```

Το αντικείμενο που σχεδιάζεται φαίνεται στην εικόνα 11. Ο πλήρης κώδικας που προγράμματος που περιέχει και τις δύο κλάσεις (GLUObjects και GLUObjectsView) υπάρχει στα αρχεία demos.



Εικόνα 11: Ένα έτοιμο αντικείμενο της κλάσης GLU με τη μορφή GLU_LINE

7. Μετασχηματισμοί - Transformations

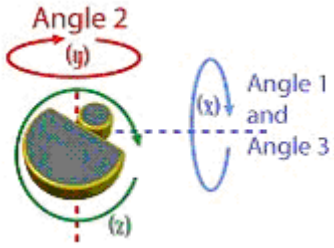
7.1. Θεωρία

Μπορούμε να μετακινηθούμε μέσα σε οποιοδήποτε 3D σκηνικό με πολλούς τρόπους. Καθώς κινούμαστε μετασχηματισμοί συμβαίνουν στα διάφορα αντικείμενα του σκηνικού μας. Όπως ήδη αναφέρθηκε, οι μετασχηματισμοί συμβαίνουν σε λειτουργία GL_MODELVIEW. Τα «έτοιμα» αντικείμενα της JOGL, όπως αυτά των κλάσεων GLU και GLUT, τοποθετούνται στο σημείο αναφοράς που είναι το κέντρο της ορατής περιοχής, με αποτέλεσμα οποιοδήποτε αντικείμενο να εμφανίζεται στην ίδια θέση και με το ίδιο μέγεθος. Αυτό μπορεί να αλλάξει με τη χρήση των μετασχηματισμών.

Υπάρχουν 3 είδη μετασχηματισμών στην JOGL που πραγματοποιούνται με τις παρακάτω μεθόδους:

- glScaled(double x, double y, double z):
  Ο μετασχηματισμός κλίμακας χρησιμοποιείται για την αλλαγή της κλίμακας σύμφωνα με την οποία σχεδιάζεται το αντικείμενο. Οι τιμές των x, y και z που παίρνει σαν παραμέτρους πολλαπλασιάζονται με την προϋπάρχουσα κλίμακα ώστε να δημιουργηθούν παραμορφώσεις στο αντικείμενο.

- glRotated(double angle, double x, double y, double z):
  Ο μετασχηματισμός περιστροφής χρησιμοποιείται για την περιστροφή του αντικειμένου ως προς τους άξονες x-y-z. Είναι σαν να φανταζόμαστε ότι η κάμερα γυρίζει γύρο-γύρο από το αντικείμενο ώστε να το δούμε από όλες της πλευρές του. Για την περιστροφή ως προς τους τρις άξονες χρησιμοποιούμε για τον κάθε άξονα μια ξεχωριστή κλήση της μεθόδου glRotated με διαφορετικές τιμές του angle αλλά και των x,y,z. Επί παραδείγματι, οι κλήσεις gl.glRotated(angle1, 1, 0, 0) και gl.glRotated(angle2, 0, 1, 0) χρησιμοπουούνται για περιστροφή ως προς τον άξονα x και y αντίστοιχα. Το παράδειγμα αυτό γίνεται πιο κατανοητό με τη βοήθεια της είκόνας 12. [4]



Εικόνα 12: Περιστροφή ως προς τους 3 άξονες

- glTranslated(double x, double y, double z):
  Ο μετασχηματισμός μεταφοράς χρησιμοποιείται για την μετακίνηση των «έτοιμων» αντικειμένων σε διαφορετική θέση από το σημείο αναφοράς. Το σημείο εμφάνισης του αντικειμένου διαφοροποιείται από το σημείο αναφοράς ανάλογα με τις τιμές των παραμέτρων x, y και z. Επίσης, επιτρέπει στον χρήστη να περιηγηθεί στον χώρο καθώς προσομοιάζει μια κάμερα που μπορεί να μετακινηθεί μόνο στους άξονες x-y-z χωρίς να μπορεί να «στρίψει».

7.2. Το Πρόγραμμα FirstPersonView

Ακολουθεί ο κώδικας ενός προγράμματος που σχεδιάζει έναν κύβο στον τρισδιάστατο κόσμο με τη χρήση των έτοιμων σχημάτων της κλάσης GLUT. Επίσης, μας δίνεται η δυνατότητα με το πάτημα ορισμένων πλήκτρων να μετακινηθούμε μέσα στον τρισδιάστατο κόσμο με τη χρήση των μεθόδων μετασχηματισμού που ήδη αναφέρθηκαν. Τέλος, εφαρμόζεται αλλαγή κλίμακας με τη χρήση της μεθόδου glScaled που αλλάζει την κλίμακα του άξονα y παραμορφώνοντας τον κύβο. Ακολουθεί η κύρια κλάση, με την ονομασία FirstPersonMovement, που οποία περιλαμβάνει τη main() και καλεί την επόμενη κλάση (FirstPersonView) η οποία περιέχει τον JOGL κώδικα.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.media.opengl.*;
/**
* This is a basic JOGL app. Feel free to
* reuse this code or modify it.
*/
public class FirstPersonMovement extends JFrame {

    static GLCanvas glcanvas = null;

    public static void main(String[] args) {
        final FirstPersonMovement app = new FirstPersonMovement();
        // show what we've done
        SwingUtilities.invokeLater (
        new Runnable() {
        public void run() {
        app.setVisible(true);
        }
        }
        );
    }

    public FirstPersonMovement() {
        //set the JFrame title
        super("First Person Movement");
```

Στις επόμενες σειρές του προγράμματος βλέπουμε ότι δημιουργείται ένα αντικείμενο της κλάσης FirsPersonView η οποία υλοποιεί τόσο τον GLEventListener, όσο και τον KeyListener. Στη συνέχεια προστίθεται ο GLEventListener στο glcanvas, όπως και στα προηγούμενα παραδείγματα. Αυτή τη φορά όμως προσθέτουμε την κλάση FirstPersonView για δεύτερη φορά αφού αυτή υλοποιεί και τον KeyListener. Ο KeyEventListener αποτελεί μια διεπαφή χρήστη η οποία βοηθάει στην παρακολούθηση γεγονότων πληκτρολογίου.

```
        //create our KeyDisplay which serves two purposes
        // 1) it is our GLEventListener, and
        // 2) it is our KeyListener
        FirstPersonView fpv = new FirstPersonView();

        //kill the process when the JFrame is closed
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //only three JOGL lines of code ... and here they are
        GLCapabilities glcaps = new GLCapabilities();
        glcanvas = new GLCanvas();
        glcanvas.addGLEventListener(fpv);
        glcanvas.addKeyListener(fpv);
```

Στο πρόγραμμα αυτό θα πρέπει να περάσουμε το αντικείμενο glcanvas στην κλάση FirstPersonView με τη χρήση της μεθόδου setGLCanvas() και αυτό γιατί το σκηνικό μας δεν θα παραμένει αμετάβλητο, αλλά θα καλούμε την repaint() κάθε φορά που μετακινούμαστε στον χώρο. Η παρακάτω γραμμή κώδικα μας εξασφαλίζει ακριβώς ότι κάθε φορά που ο χρήστης πιέζει ένα πλήκτρο για να μετακινηθεί στον χώρο το σκηνικό θα αλλάζει ανάλογα.

```
        //we'll want this for our repaint requests
        fpv.setGLCanvas(glcanvas);

        //add the GLCanvas just like we would any Component
        getContentPane().add(glcanvas, BorderLayout.CENTER);
        setSize(500, 300);
        //center the JFrame on the screen
        centerWindow(this);
    }

    public void centerWindow(Component frame) {
        Dimension screenSize =
        Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = frame.getSize();
        if (frameSize.width > screenSize.width )
        frameSize.width = screenSize.width;
        if (frameSize.height > screenSize.height)
        frameSize.height = screenSize.height;
        frame.setLocation (
        (screenSize.width - frameSize.width ) >> 1,
        (screenSize.height - frameSize.height) >> 1
        );
    }
}
```

Το παρακάτω πρόγραμμα με την ονομασία FirstPersonView μας δείχνει τον τρόπο με τον οποίο ο χρήστη μπορεί να μετακινηθεί στον χώρο του 3D σκηνικού πατώντας κάποια πλήκτρα από το πληκτρολόγιο.

```
import com.sun.opengl.util.GLUT;
import java.awt.*;
import java.awt.event.*;
import javax.media.opengl.*;
import javax.media.opengl.glu.GLU;

/**
```

```
* For our purposes only two of the
* GLEventListeners matter. Those would
* be init() and display().
*/
public class FirstPersonView implements GLEventListener, KeyListener
{
    double xPosition = 0;
    double zPosition = 0;
    double yPosition = 0;
    int anglex = 0;
    int angley = 0;

    float red = 0.0f;
    float green = 0.0f;
    float blue = 1.0f;
    GLCanvas glc;
```

Εδώ γίνεται η αρχικοποίηση των μεταβλητών στης οποίες αποθηκεύεται η θέση του χρήστη (x,y,z) καθώς και της γωνίας περιστροφής ως προς την οποία περιστρέφετε ο χρήστης. Άλλες μεταβλητές αποτελούν οι default τιμές για το χρώμα του σκηνικού καθώς επίσης και το αντικείμενο GLCanvas. Ακολουθεί η μέθοδος setGLCanvas() η χρησιμότητα της οποίας εξηγήθηκε στο προηγούμενο πρόγραμμα.

```
     void setGLCanvas(GLCanvas glcanvas) {
        this.glc = glcanvas;
    }

/**
    * Take care of initialization here.
    */

     public void init(GLAutoDrawable gld) {

        //The mode is GL.GL_MODELVIEW by default
        //We will also use the default ViewPort
        GLU glu = new GLU();
        GL gl = gld.getGL();

        gl.glMatrixMode(GL.GL_PROJECTION);
        gl.glLoadIdentity();

        // Aspect is width/height
        double w = ((Component) gld).getWidth();
        double h = ((Component) gld).getHeight();
        double aspect = w/h;
        //When using gluPerspective near and far need
        //to be positive.
        //The arguments are:
        //fovy, aspect, near, far
        glu.gluPerspective(60.0, aspect, 2.0, 20.0);
        }

    /**
    * Take care of drawing here.
    */
    public void display(GLAutoDrawable drawable) {

        GL gl = drawable.getGL();
        GLU glu = new GLU();
        GLUT glut = new GLUT();

        gl.glMatrixMode(GL.GL_MODELVIEW);
        gl.glLoadIdentity();

        gl.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
```

```
//Define points for eye, at and up.
//This is your camera. It ALWAYS goes
//in the GL_MODELVIEW matrix.
glu.gluLookAt(
xPosition, yPosition, zPosition,
xPosition, yPosition, (zPosition+20),
0, 1, 0
);
gl.glClear(GL.GL_COLOR_BUFFER_BIT);
```

Σε αυτή την περίπτωση η μέθοδος gluLookAt() περιέχεται στην display() διότι η θέση της κάμερας (χρήστης) δεν είναι σταθερή αλλά αλλάζει καθώς περιηγούμαστε στον χώρο. Αυτό πετυχαίνεται με τη χρήση των μεταβλητών xPosition, yPosition και zPosition των οποίων οι τιμές αλλάζουν ώστε να αλλάζει και η θέση της κάμερας.

```
gl.glRotated(anglex, 1, 0, 0);
gl.glRotated(angley, 0, 1, 0);
```

Η δύο εντολές gl.glRotated() επιτυγχάνουν την περιστροφή των αντικειμένων κάθε φορά που μια από τις τιμές anglex και angley αλλάζουν. Η πρώτη γραμμή περιστρέφει τα αντικείμενα ως προς τον άξονα x ενώ η δεύτερη ως προς τον άξονα y. Το αποτέλεσμα που επιτυγχάνεται μοιάζει σαν ο χρήστης να περιστρέφει το κεφάλι του πάνω – κάτω και δεξιά – αριστερά, αντίστοιχα.

```
red = 0.0f;
green = 0.0f;
blue = 0.9f;
gl.glColor3f(red, green, blue);

//transforming the place the next shape
//will be drawn.
gl.glTranslated(2, 0, 2);
//We use wire here because default
//lighting is not good enough to
//use when rendering the solid version
glut.glutWireIcosahedron();
```

Οι παραπάνω δυο γραμμές καθορίζουν τη θέση που θα εμφανιστεί το αντικείμενο καθώς και το σχήμα και είδος του αντικειμένου. Η μέθοδος glTranslate() ορίζει το σημείο που θα εμφανιστεί το επόμενο αντικείμενο προσθέτοντας τις τιμές x,y,z στις τιμές του σημείου αναφοράς. Για την εμφάνιση του πρώτου αντικειμένου οι τιμές αναφοράς συμπίπτουν με της τιμές εισόδου τις μεθόδου gluLookAt(). Αφού ορίσουμε τη θέση του αντικειμένου επιλέγουμε το αντικείμενο της κλάσης GLUT ή GLU που θέλουμε να εμφανίσουμε. Με τον ίδιος τρόπο σχεδιάζουμε και τα επόμενα εικοσάεδρα.

```
//more shapes to navigate through
gl.glTranslated(-4, 0, 0);
glut.glutWireIcosahedron();

red = 0.0f;
green = 0.9f;
blue = 0.1f;
gl.glColor3f(red, green, blue);
gl.glTranslated(4, 0, 4);
glut.glutWireIcosahedron();
gl.glTranslated(-4, 0, 0);
glut.glutWireIcosahedron();

red = 0.9f;
green = 0.0f;
blue = 0.1f;
gl.glColor3f(red, green, blue);
```

```
        gl.glTranslated(4, 0, 4);
        glut.glutWireIcosahedron();
        gl.glTranslated(-4, 0, 0);
        glut.glutWireIcosahedron();

        red = 0.9f;
        green = 0.0f;
        blue = 0.9f;
        gl.glColor3f(red, green, blue);
        gl.glTranslated(4, 0, 4);
        glut.glutWireIcosahedron();
        gl.glTranslated(-4, 0, 0);
        glut.glutWireIcosahedron();

        red = 0.5f;
        green = 0.5f;
        blue = 0.5f;
        gl.glColor3f(red, green, blue);
        gl.glTranslated(4, 0, 4);
        glut.glutWireIcosahedron();
        gl.glTranslated(-4, 0, 0);
        glut.glutWireIcosahedron();
        }

    /**
    * Called when the GLDrawable (GLCanvas
    * or GLJPanel) has changed in size. We
    * won't need this, but you may eventually
    * need it -- just not yet.
    */
    public void reshape(GLAutoDrawable drawable, int x, int y, int width, int
    height) {
        }

    /**
    * If the display depth is changed while the
    * program is running this method is called.
    * Nowadays this doesn't happen much, unless
    * a programmer has his program do it.
    */
    public void displayChanged(GLAutoDrawable drawable, boolean modeChanged,
    boolean deviceChanged) {
        }
```

Οι τρις μέθοδοι που πρέπει να υλοποιούνται όταν δημιουργούμε έναν KeyListener είναι οι keyTyped(), keyPressed() και keyReleased(). Η μέθοδος που χρησιμοποιούμε πιο συχνά είναι η keyTyped(). Μόνο στη μέθοδο αυτή μπορούμε να κάνουμε χρήση της συνάρτησης getKeyChar() η οποία επιστρέφει ένα αναγνωριστικό για το πλήκτρο που πατήθηκε. Με αυτόν τον τρόπο, ανάλογα με το πλήκτρο εκτελείται και η ανάλογη ενέργεια.

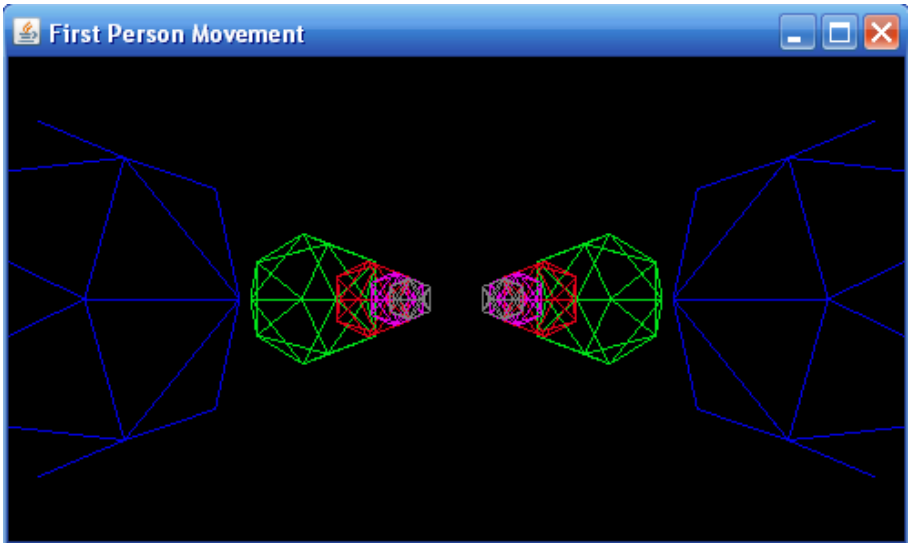
```
    public void keyTyped(KeyEvent e) {
        if (e.getKeyChar() == KeyEvent.VK_1)
        zPosition += 1;
        else if (e.getKeyChar() == KeyEvent.VK_2)
        zPosition -= 1;
        else if (e.getKeyChar() == KeyEvent.VK_3)
        xPosition += 1;
        else if (e.getKeyChar() == KeyEvent.VK_4)
        xPosition -= 1;
        else if (e.getKeyChar() == KeyEvent.VK_5)
        yPosition += 1;
        else if (e.getKeyChar() == KeyEvent.VK_6)
        yPosition -= 1;
        else if (e.getKeyChar() == KeyEvent.VK_7)
        anglex += 1;
        else if (e.getKeyChar() == KeyEvent.VK_8)
```

```
            anglex -= 1;
            else if (e.getKeyChar() == KeyEvent.VK_9)
            angley += 1;
            else if (e.getKeyChar() == KeyEvent.VK_0)
            angley -= 1;
            glc.repaint();

        }
```

Η μέθοδος keyTyped() περιέχει μια σειρά από συνθήκες if που ελέγχουν την είσοδο του πληκτρολογίου. Ανάλογα με το πλήκτρο που πατάει ο χρήστης οι τιμές των xPosition, yPosition και zPosition αυξάνονται ή μειώνονται επηρεάζοντας τις παραμέτρους τις gluLookAt. Έτσι αλλάζει η θέση της κάμερας και ο χρήστης μπορεί να μετακινηθεί μέσα στον χώρο. Οι τελευταίες τέσσερις συνθήκες αλλάζουν τις τιμές των anglex και angley ώστε ο χρήστης με τη βοήθεια της glRotate να μπορεί να περιστρέψει το οπτικό του πεδίο στον χώρο.

```
    public void keyPressed(KeyEvent ke) {

    }

    public void keyReleased(KeyEvent ke) {

    }

}
```

Το αποτέλεσμα του προγράμματος αυτού με τα διάφορα πολύχρωμα εικοσάεδρα διάσπαρτα στον χώρο φαίνεται στην εικόνα 13.



Εικόνα 13: Πολύχρωμα πολύγωνα της OpenGL

8. Φωτισμός και Χρώμα

8.1. Γενικά Στοιχεία

Μέχρι τώρα για τον σχηματισμό των έτοιμων αντικειμένων της κλάσης glut χρησιμοποιούσαμε μεθόδους που σχεδιάζανε μόνο τον σκελετό των αντικειμένων ή καλύτερα το περίγραμμά τους. Οι μέθοδοι αυτοί περιλάμβαναν στο όνομά τους την λέξη Wire και σχηματίζανε αντικείμενα που οπτικά έμοιαζαν με αυτό του παραπάνω προγράμματος. Παραδείγματα τέτοιων μεθόδων είναι οι glutWireIcosahedron(), glutWireCube(), glutWireCone() κ.α.

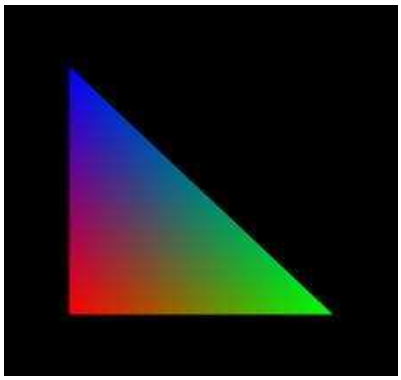
Το πρόβλημα αυτό της απεικόνισης πηγάζει από το γεγονός ότι οι πλευρές όλων των αντικειμένων είναι του ίδιου χρώματος, το οποίο καθορίζεται όπως ήδη αναφέρθηκε από τη μέθοδο glColor3f(float red, float green, float blue). Στον πραγματικό κόσμο όμως δεν συμβαίνει αυτό καθώς κάθε αντικείμενο έχει διαφορετικό χρώμα σε κάθε πλευρά του. Αν παρατηρήσουμε ένα αντικείμενο του πραγματικού κόσμου θα δούμε ότι παρότι θεωρητικά φαίνεται να έχει ένα χρώμα, οι πλευρές του διαφέρουν ως προς τη σκιά, ακόμα μπορεί να παρατηρήσουμε την αντανάκλαση άλλων αντικειμένων στην επιφάνειά του.

Εκτός από τον καθορισμό του χρώματος ενός αντικειμένου μπορούμε να επιλέξουμε και ένα μοντέλο σκίασης (shading model). Μια γραμμή ή ένα βασικό πολύγωνο της OpenGL μπορεί να ζωγραφιστεί με μόνο ένα χρώμα (flat shading) ή και με περισσότερα χρώματα (smooth shading ή *Gouraud* shading). Η παρακάτω συνάρτηση μας βοηθάει να επιλέξουμε το επιθυμητό shading model:

- glShadeModel(GL.mode):
  η δυνατές τιμές του mode είναι GL_FLAT και GL_SMOOTH

Με τη χρήση του flat shading η κάθε κορυφή ενός βασικού πολυγώνου της GOGL έχει το ίδιο ακριβώς χρώμα και έτσι σχηματίζεται ένα πολύγωνο το οποίο φαίνεται να έχει ένα μόνο χρώμα.

Αντίθετα, με το smooth shading, το χρώμα της κάθε κορυφής αντιμετωπίζεται με ξεχωριστό τρόπο. Σε μία γραμμή τα διαφορετικά χρώματα της κάθε κορυφής παρεμβάλλονται αναμεταξύ τους κατά μήκος του ευθυγράμμου τμήματος. Επίσης, σε ένα βασικό πολύγωνο, τα χρώματα του εσωτερικού του πολυγώνου παρεμβάλλονται στα χρώματα των κορυφών του. Ένα τέτοιο πολύγωνο φαίνεται στην εικόνα 14. [1]



Εικόνα 14: Τρίγωνο με τη χρήση smooth shading

8.2. Φωτισμός Σκηνικού

Για να απεικονίσουμε με ρεαλιστικό τρόπο τα αντικείμενα μας ώστε να φαίνεται το βάθος τους, όπως τα στερεά στον φυσικό κόσμο, χρειαζόμαστε μια πηγή φωτός. Πρώτα ενεργοποιούμε το στοιχείο του φωτισμό στην OpenGL με την παρακάτω εντολή:

- glEnable(GL.GL_LIGHTING)

Μόλις ενεργοποιήσουμε τον φωτισμό αυτόματα η εντολή glColor3f σταματάει να έχει χρησιμότητα. Τα αντικείμενά μας πλέον έχουν χρώμα γκρι και περιμένουν να λάβουν το χρώμα τους με την χρήση των materials που αναλύονται παρακάτω.

Είναι σημαντικό να γνωρίζουμε ότι η OpenGL διαθέτει έναν αριθμό από απενεργοποιημένα φώτα που μπορεί να χρησιμοποιήσει ο χρήστης. Ο μέγιστος αριθμός καθορίζεται από την τιμή GL.GL_MAX_LIGHTS και είναι ανάλογος με την έκδοση της JOGL που διαθέτουμε (συνήθως είναι 8). Τα φώτα αυτά έχουν τα ονόματα: GL_LIGHT0, GL_LIGHT1, GL_LIGHT2, GL_LIGHT3 κλπ και ενεργοποιούνται με την εντολή:

- glEnable(GL.GL_LIGHTx) όπου x μπορεί να είναι 1,2,3 κλπ [3]

Υπάρχουν τέσσερα είδη φωτισμού στην OpenGL. Αυτά είναι:

1) Ambient: Πρόκειται για το διάσπαρτο φως που υπάρχει σε ένα περιβάλλον του οποίου η κατεύθυνση είναι αδύνατον να προδριοριστεί. Το αποτέλεσμα αυτό προκύπτει όταν το φως που πέφτει σε μια επιφάνεια αντανακλάται προς όλες τις κατευθύνσεις. Το φως αυτό συνήθως έχει ίδιο χρώμα με την πηγή φωτός, ωστόσο σε ορισμένες περιπτώσεις όταν ένα χρώμα κυριαρχεί σε έναν χώρο το διάσπαρτο φως παίρνει την χροιά του χρώματος των αντικειμένων πάνω στα οποία αντανακλάται. Για ενεργοποιήσουμε τον ambient φωτισμό χρησιμοποιούμε έναν πίνακα τεσσάρων στοιχείων. Τα τρία πρώτα αντιπροσωπεύουν τα χρώματα RGB και παίρνουν τιμές από 0 μέχρι 1. Η τελευταία παράμετρος είναι η Alpha, που όπως ήδη αναφέρθηκε καθορίζει τη διαφάνεια. Στη συνέχεια με τη χρήση της μεθόδου glLightfv δημιουργούμε τον διάσπαρτο φωτισμό μας με βάση τον πίνακα ambient.Ο αντίστοιχος κώδικα είναι:
    - float[] ambient = {float red, float green, float blue,float alpha};
      gl.glLightfv(GL.GL_LIGHT0,GL.GL_AMBIENT,ambient,0);

2) Diffuse: Το είδος αυτό φωτός προέρχεται από μια συγκεκριμένη κατεύθυνση και πέφτει ακριβώς πάνω σε μια επιφάνεια. Κατά συνέπεια το φώς αυτό είναι πιο δυνατό από το ambient και αντανακλάται προς όλες τις κατευθύνσεις. Το αντικείμενο φαίνεται να είναι φωτισμένο με την ίδια ένταση από όποιο σημείο και αν το κοιτάξουμε. Γενικά οποιαδήποτε πηγή φωτός σε συγκεκριμένη θέση εκπέμπει diffuse light. Η ενεργοποίηση αυτού του είδους φωτισμού γίνεται όπως και πριν, με τη χρήση ενός πίνακα diffuse:
    - float[] diffuse = {float red, float green, float blue,float alpha};
      gl.glLightfv(GL.GL_LIGHT0, GL.GL_DIFFUSE, diffuse,0);

3) Specular: Το φως αυτό προέρχεται και πάλι από μια συγκεκριμένη κατεύθυνση αλλά αυτή τη φορά αντανακλάται από την επιφάνεια με έναν επιθυμητό τρόπο. Μοιάζει με τη συμπεριφορά μιας ακτίνας λέιζερ που αντανακλάται πλήρως σε ένα καθρέφτη ή αλλιώς την συμπεριφορά που έχει ένα κάτοπτρο όταν το διαπερνά μια ακτίνα φωτός. Οι γυαλιστερές επιφάνειες ή το πλαστικό έχουν τον παράγοντα αυτό σε μεγάλο βαθμό,

δηλαδή οι τιμές των RGB στον πίνακα specular έχουν υψηλές τιμές, ενώ επιφάνειες όπως αυτές μιας κιμωλίας ή ενός χαλιού δεν διαθέτουν τέτοιου είδους φωτισμό. Οι εντολές για την ενεργοποίησή του είναι:

- float[] specular = {float red, float green, float blue,float alpha};
  gl.glLightfv(GL.GL_LIGHT0, GL.GL_SPECULAR, specular,0);

4) Emission: Εκτός από τα παραπάνω είδη φωτός ένα αντικείμενο μπορεί να διαθέτει εσωτερικό φως, δηλαδή φώς που να εκπέμπει το ίδιο. Στην OpenGL, το στοιχείο αυτό προσθέτει ένταση στο χρώμα μιας επιφάνειας αλλά δεν προσθέτει επιπλέον φωτισμό στον χώρο και δεν επηρεάζεται από άλλες πηγές φωτός. Οι αντίστοιχες εντολές είναι:
   - float[] emission = {float red, float green, float blue,float alpha};
     gl.glLightfv(GL.GL_LIGHT0,GL.GL_EMISSION,emission,0);

Σπάνια χρησιμοποιούμε και τα τρία είδη φωτισμού. Τις περισσότερες φορές τα ambient και diffuse light είναι αρκετά ώστε να φωτίσουνε το σκηνικό μας. Περισσότερα είδη φωτισμού δημιουργούν ένα πολύ έντονα φωτισμένο σκηνικό. Επίσης θα πρέπει να αναφέρω ότι τη θέση του GL.GL_LIGHT0 στα παραπάνω παραδείγματα κώδικα μπορεί να πάρει οποιοδήποτε από τα διαθέσιμα φώτα της JOGL (GL_LIGHT0, GL_LIGHT1 κλπ). Αφού ορίσουμε τα είδη φωτισμού που υποστηρίζει το κάθε φώς θα πρέπει να το ενεργοποιήσουμε όταν το χρειαστούμε και να το απενεργοποιήσουμε όταν δε μας χρειάζεται πλέον. Η απενεργοποίηση του φωτός γίνεται ως εξής:

- glDisable(GL.GL_LIGHTx) όπου x μπορεί να είναι 1,2,3 κλπ

8.3. Θέση της πηγής φωτός

Μετά από μια σύντομη περιγραφή των ειδών φωτισμού που υποστηρίζει η OpenGL θα πρέπει να αναφέρουμε τον τρόπο με τον οποίο τοποθετούμε την πηγή φωτός σε μια συγκεκριμένη θέση. Για τον προσδιορισμό της θέσης χρησιμοποιούμε έναν πίνακα position τεσσάρων στοιχείων. Τα τρία πρώτα προσδιορίζουν τις συντεταγμένες x, y, z. Η τελευταία παράμετρος παίρνει και αυτή τιμές από 0 μέχρι 1.

Αν η τιμή της τελευταίας παραμέτρου είναι 0 τότε η πηγή φωτός είναι directional, δηλαδή δίνει την εντύπωση ότι είναι τοποθετημένη μακριά από τη σκηνή. Η απόσταση του αντικειμένου θεωρείται άπειρη και αυτό έχει ως αποτέλεσμα η ακτίνες φωτός να πέφτουν κάθετα πάνω στα αντικείμενά μας. Ένα παράδειγμα directional light είναι ο ήλιος. Αντίθετα για τιμή της παραμέτρου ίση με 1 έχουμε positional light, στο οποίο οι συντεταγμένες της θέσης της φωτεινής πηγής επηρεάζουν την κατεύθυνση που θα έχουν οι ακτίνες φωτός και κατά επέκταση το αποτέλεσμα που θα έχει ο φωτισμός πάνω στα αντικείμενα της σκηνής. Ένα παράδειγμα positional light είναι το φως που προκαλεί μια λάμπα γραφείου.

Η θέση που επιθυμούμε να τοποθετήσουμε την φωτεινή πηγή μας ενεργοποιείται επίσης με τη χρήση της συνάρτησης glLightfv και ο αντίστοιχος κώδικας είναι:

- float[] position = {float x, float y, float z, float w};
  gl.glLightfv(GL.GL_LIGHT0, GL.GL_POSITION, position,0);

Είναι προφανές ότι και σε αυτή την περίπτωση η πρώτη παράμετρος της συνάρτησης glLightfl είναι ένα από τα έτοιμα φώτα που μας παρέχει η OpenGL. [1]

8.4. **Depth Test**

Σε αυτή την ενότητα είναι σημαντικό να αναφέρουμε ότι η JOGL έχει αναπτύξει έναν μηχανισμό με τον οποίον ελέγχει αν ένα μέρος ενός αντικειμένου ή ένα αντικείμενο ολόκληρο θα πρέπει να εμφανιστεί στην οθόνη ή όχι. Παραδείγματος χάριν, αν ένα πολύγωνο που βρίσκεται στην ορατή περιοχή μας επικαλύπτεται από ένα άλλο πολύγωνο, το πρώτο θα πρέπει να εμφανιστεί ή όχι; Ο GLCanvas εμφανίζει οτιδήποτε υπάρχει στο 3D σκηνικό μας με τη σειρά με την οποία το εντοπίζει, αυτό γίνεται χάριν ευκολίας.

Σε ορισμένα προηγμένα συστήματα κανένα μέρος αντικείμενου ή αντικείμενο δεν σχεδιάζεται από το σύστημα χωρίς να ελεγχθεί αν είναι ορατό ή όχι. Ωστόσο στην JOGL κάτι τέτοιο δεν είναι εφικτό και έτσι θα πρέπει να προειδοποιήσουμε το πρόγραμμα για την ύπαρξη τέτοιων επικαλυπτόμενων πολυγώνων. Για να γίνει αυτό καλούμε τη συνάρτηση gl.glEnable(GL.GL_DEPTH_TEST). Επίσης, καθαρίζουμε τον depth buffer bit καλώντας τη εντολή glClear, γνωστή από τον καθαρισμό του φόντου της 3D σκηνής μας.

Αν δεν σιγουρευτούμε ότι η JOGL πραγματοποιεί τον έλεγχο βάθους μπορεί να καταλήξουμε με αντικείμενα που εμφανίζονται με λανθασμένο τρόπο.

Τέλος, αξίζει να δώσουμε την πληροφορίας πως στη συνάρτηση gluPerspective() η οποία τοποθετεί την «εικονική» κάμερά μας στο σκηνικό η τιμή της παραμέτρου near θα πρέπει θα είναι πάντα μεγαλύτερη του 0.1. Για μια τιμή πολύ κοντά στο 0 το έλεγχος του βάθους που αναφέραμε παραπάνω δεν θα λειτουργεί και τα αποτελέσματα δεν θα είναι τα επιθυμητά.

8.5. **Materials**

Το χρώμα με το οποίο εμφανίζεται ένα αντικείμενο εξαρτάται από τον φωτισμό αλλά και από το υλικά που σχετίζονται με το αντικείμενο, γνωστά ως materials. Το material color σχετίζεται με έναν πίνακα τεσσάρων στοιχείων. Τα τρία πρώτα στοιχεία καθορίζουν τις τιμές των χρωμάτων RGB και το τελευταίο την τιμή της παραμέτρου Alpha που ήδη αναφέραμε. Η εντολές που ενεργοποιούν το χρώμα είναι:

- float[] material = {float red, float green, float blue, float alpha};
  gl.glMaterialfv(GL.face , GL.pname ,material ,0);

Η παράμετρος face μπορεί να πάρει τις τιμές GL_FRONT, GL_BACK, ή GL_FRONT_AND_BACK και καθορίζει σε ποιες πλευρές του αντικειμένου εφαρμόζεται το συγκεκριμένο υλικό. Η παράμετρος pname παίρνει της τιμές GL_SPECULAR, GL_EMISSION και GL_AMBIENT, GL_DIFFUSE ή GL_AMBIENT_AND_DIFFUSE και καθορίζει το είδος φωτισμού του material color που θα εφαρμοστεί στις πλευρές του αντικειμένου. Τέλος η παράμετρος material είναι ένας πίνακας που περιλαμβάνει της τιμές του μοντέλου RGBA όπως ορίστηκαν προηγουμένως.

Αφού καθορίσουμε το material color με την χρήση των παραπάνω εντολών είμαστε έτοιμοι να σχεδιάσουμε το αντικείμενό μας. [1]

Εκτός από τον φωτισμό των αντικειμένων, η OpenGL διαθέτει 3 ακόμα τεχνικές που βοηθούν στην ρεαλιστική απεικόνιση των αντικειμένων. Οι τεχνικές αυτές είναι:

- Blending- Ανάμειξη: Η τεχνική αυτή καθορίζει μια συνάρτηση η οποία ελέγχει τον τρόπο με τον οποίο η προϋπάρχουσα τιμή χρώματος σε ένα pixel της οθόνης θα συνδυαστεί με τη νέα τιμή χρώματος που προκαλεί η τοποθέτηση ενός αντικειμένου στο σημείο αυτό. Το τελικό αποτέλεσμα είναι ότι τμήματα του σκηνικού μας φαίνονται ημιδιαφανή αφήνοντας τα προηγούμενα χρώματα που προϋπήρχαν να είναι ακόμα ορατά.

  Η παράμετρος που βοηθά στην υλοποίηση αυτής της τεχνικής είναι η τιμή Alpha του χρωματικού μοντέλου RGBA που χρησιμοποιούμε στην OpenGL. Την τιμή αυτή τη συναντήσαμε μέχρι τώρα στις συναρτήσεις glColor() και glClearColor() που καθορίζουν το χρώμα του αντικειμένου και το χρώμα «καθαρισμού» του σκηνικού. Επίσης, αποτελεί παράμετρος του πίνακα φωτισμού αλλά και του πίνακα material.

  Παραδείγματος χάριν όταν βλέπουμε τον κόσμο μέσα από ένα πράσινο τζάμι τότε τα χρώματα που αντιλαμβανόμαστε περιλαμβάνουν 20% το χρώμα πράσινο και 80% το πραγματικό χρώμα του αντικειμένου. Η τεχνική αυτή μας βοηθάει να προσομοιάσουμε τέτοιες περιπτώσεις.

  Αρχικά ενεργοποιούμε την τεχνική με τη χρήση της εντολής:
  - glEnable(GL.GL_BLEND);

  Για απενεργοποίηση της τεχνικής blending γράφουμε αντίστοιχα glDisable().

  Για να παρουσιάσουμε τις επόμενες συναρτήσεις θα πρέπει να κατανοήσουμε τον τρόπο που συνδυάζονται τα χρώματα για την επίτευξη της τεχνικής αυτής. Έστω ότι οι τιμές των παραμέτρων χρώματος του pixel που ανήκει στο αντικείμενο που θέλουμε να εισάγουμε στο σκηνικό μας (Sourse) είναι (Sr, Sg, Sb, Sa) και οι τιμές χρώματος που προϋπήρχαν στο συγκεκριμένο pixel (Destination) είναι (Dr, Dg, Db, Db). Τότε οι τελικές blended τιμές RGBA τις οποίες θα έχει τελικά το αντικείμενο υπολογίζονται ως εξής:

  (RsSr + RdDr, GsSg + GdDg, BsSb + BdDb, AsSa + AdDa)

  Οι (Rs, Gs, Bs, As) και (Rd, Gd, Bd, Ad) ονομάζονται blending factors και καθορίσουν τον τρόπο ανάμειξης των χρωμάτων των sourse και destination pixels αντίστοιχα.

  Η επόμενες συναρτήσεις είναι αυτές που καθορίζουν τις τιμές που μας βοηθούν στον υπολογισμό των παραπάνω blending factors:
  - glBlendFunc(GL.srcfactor, GL.destfactor):
    όπου οι παράγοντες επηρεάζουν και τις 4 τιμές των RGBA

  - glBlendFuncSeparate(GL.srcRGB, GL.destRGB, GL.srcAlpha, GL.destAlpha):
    όπου έχουμε ξεχωριστούς παράγοντες για τις τιμές RGB και την τιμή του Alpha

Σταθερά	RGB Blend Factor	Alpha Blend Factor
GL_ZERO	(0,0,0)	0
GL_ONE	(1,1,1)	1
GL_SRC_COLOR	(Rs,Gs,Bs)	As
GL_ONE_MINUS_SRC_COLOR	(1,1,1) - (Rs,Gs,Bs)	1 – As
GL_DST_COLOR	(Rd,Gd,Bd)	Ad
GL_DST_MINUS_SRC_COLOR	(1,1,1) - (Rd,Gd,Bd)	1 – Ad
GL_SRC_ALPHA	(As,As,As)	As
GL_ONE_MINUS_SRC_ALPHA	(1,1,1) - (As,As,As)	1 – As
GL_DST_ALPHA	(Ad,Ad,Ad)	Ad
GL_ONE_MINUS_DST_ALPHA	(1,1,1) - (Ad,Ad,Ad)	1 – Ad
GL_CONSTANT_COLOR	(Rc,Gc,Bc)	Ac
GL_ONE_MINUS_CONSTANT_COLOR	(1,1,1) - (Rc,Gc,Bc)	1 – Ac
GL_CONSTANT_ALPHA	(Ac,Ac,Ac)	Ac
GL_ONE_MINUS_CONSTANT_ALPHA	(1,1,1) - (Ac,Ac,Ac)	1 – Ac
GL_SRC_ALPHA_SATURATE	(f, f, f); f = min(As,1 - Ad)	1

Πίνακας 2: Οι δυνατές τιμές των παραμέτρων srcfactor και destfactor π.χ. glBlendFunc(GL.GL_ONE, GL. GL_DST_COLOR)

Όταν χρησιμοποιούμε τη σταθερά τύπου GL*CONSTANT* θα πρέπει να ορίσουμε τις τιμές των Rc, Gc, Bc, Ac με τη βοήθεια της συνάρτησης glBlendColor(float red, float green, float blue, float alpha).

Τέλος, χρειαζόμαστε τη συνάρτηση που καθορίζει την πράξη που θα εκτελεστεί προκειμένου να υπολογιστούν οι τελικές blended τιμές RGBA. Όπως και πριν υπάρχουν δύο διαθέσιμες συναρτήσεις:

- glBlendEquation(GL.mode):
  όπου mode η αντίστοιχη πράξη

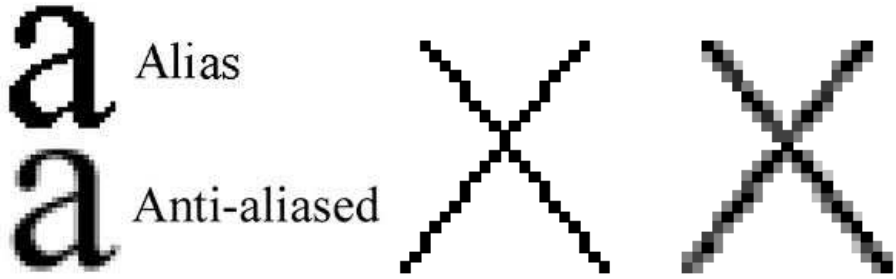
- glBlendEquationSeparate(GL.modeRGB, GL.modeAlpha):
  όπου έχουμε διαφορετικές πράξεις για τις τιμές RGB και Alpha

Blending Mode Parameter	Μαθηματική πράξη
GL_FUNC_ADD	CsS + CdD
GL_FUNC_SUBTRACT	CsS - CdD
GL_FUNC_REVERSE_SUBTRACT	CdD - CsS
GL_MIN	min(CsS, CdD)
GL_MAX	max(CsS, CdD)
GL_LOGIC_OP	CS *op* CD

Πίνακας 3: Οι δυνατές πράξεις για τον υπολογισμό των blended τιμών

- Antialising – Εξομάλυνση: Η τεχνική αυτή έχει μια ελάχιστη επίδραση στα περιγράμματα των αντικειμένων αλλάζοντας τα χρώματα έτσι ώστε οι άκρες των σημείων, των γραμμών και των πολυγώνων που εμφανίζονται ως σύνορα ανάμεσα στα αντικείμενα αυτά να φαίνονται πιο ομαλές.

Όλοι μας έχουμε προσέξει πως μερικές γραμμές, ιδιαίτερα αυτές που έχουν ελάχιστη απόκλιση από τον οριζόντιο ή κάθετο άξονα, φαίνονται να σχηματίζουν κάποιες αιχμηρές ακμές, όπως αυτές που διακρίνονται στην παρακάτω εικόνα. Αυτό το φαινόμενο ονομάζεται aliasing και η τεχνική για την αντιμετώπισή του antialising.



Εικόνα 15: Alising και antialising

Στις παραπάνω εικόνες βλέπουμε ότι μετά την εφαρμογή της τεχνικής η γραμμές καλύπτουν ένα επιπλέον πλαίσιο από γειτονικά pixels έτσι ώστε να υπάρχει ένα πιο ομαλό «σβήσιμο» της γραμμής. Ο τρόπος υπολογισμού των pixels αυτών ποικίλουν ανάλογα με την υλοποίηση της OpenGL.

Ένα τρόπος για να θέσουμε σε εφαρμογή την τεχνική του antialising είναι με την γνωστή μας συνάρτηση:

- glEnable(GL.GL_POINT_SMOOTH)
  ή glEnable(GL.GL_LINE_SMOOTH)

ανάλογα με το αν θέλουμε να εφαρμόσουμε την τεχνική σε σημεία ή σε γραμμές.

Επίσης μπορούμε να χρησιμοποιήσουμε τη συνάρτηση glHind() με τον τρόπο που περιγράφεται στη συνέχεια.

Η συνάρτηση αυτή μας βοηθάει στον έλεγχο της ισορροπίας ανάμεσα στην ποιότητα εμφάνισης μιας εικόνας και στην ταχύτητα εμφάνισης της και συντάσσεται με τον εξής τρόπο:

- glHint(GL.target, GL.hint): η παράμετρος target καθορίζει τις συμπεριφορά της OpenGL την οποία επιθυμούμε να ελέγξουμε και οι δυνατές τιμές που μπορεί να πάρει εμφανίζονται στον επόμενο πίνακα. Η παράμετρος hint παίρνει τις τιμή GL_FASTEST, για επιλογή τις πιο αποτελεσματικής – γρήγορης λύσης. Την τιμή GL_NICEST για την επιλογή της λύσης με τα καλύτερης ποιότητας αποτελέσματα και τέλος την τιμή GL_DONT_CARE όταν δεν μας ενδιαφέρει καμία επιλογή από τις δύο.

Εκτός από τις παραπάνω παραμέτρους υπάρχει και ένα πλήθως άλλων παραμέτρων των οποίων μπορούμε να καθορίσουμε την ποιότητά και τον τρόπο υπολογισμού τους ,όπως φαίνεται στον πίνακα 4.

Παράμετρος	Χαρακτηριστικά
GL_POINT_SMOOTH_HINT, GL_LINE_SMOOTH_HINT, GL_POLYGON_SMOOTH_HINT	Καθορίζει την ποιότητα των σημείων, των γραμμών ή των βασικών πολυγώνων στη διαδικασία του antialising
GL_FOG_HINT	Καθορίζει αν οι υπολογισμοί για το αποτέλεσμα της ομίχλης θα γίνουν ανά pixel (GL_NICEST) ή ανά κορυφή (GL_FASTEST)
GL_PERSPECTIVE_CORRECTION_HINT	Καθορίζει την ποιότητα παρεμβολής που χρώματος και των συντεταγμένων της υφής
GL_GENERATE_MIPMAP_HINT	Καθορίζει την ποιότητα και την απόδοση του αυτόματου mipmap level reneration
GL_TEXTURE_COMPRESSION_HINT	Καθορίζει την ποιότητα και την απόδοση της συμπίεσης των texture images
GL_FRAGMENT_SHADER_DERIVATIVE_HINT	Καθορίζει την ακρίβεια του fragment prossesing για έτοιμες GLSL shader συναρτήσεις dFdx, dFdy,και fwidth

Πίνακας 4: Οι παράμετροι του προγράμματος που μπορούμε να καθορίσουμε

- Fog – Ομίχλη: Η τεχνική της ομίχλης προσπαθεί να δημιουργήσει την ψευδαίσθηση του βάθους υπολογίζοντας τις τιμές των χρωμάτων βάση της απόστασης των αντικειμένων από το σημείο της κάμερας. Με αυτόν τον τρόπο αντικείμενα που βρίσκονται σε μεγάλη απόσταση από την κάμερα και ανήκουν στο φόντο μια σκηνής μοιάζουν να ξεθωριάζουν καθώς απομακρύνονται από αυτήν.

  Με τη χρήση του στοιχείου αυτού μπορούμε να προσομοιάσουμε διάφορα φαινόμενα που έχουν να κάνουν με την ατμόσφαιρα, όπως είναι η ομίχλη, ο καπνός, η ρύπανση και άλλα. Είναι ιδιαίτερα χρήσιμο εργαλείο που χρησιμοποιείται συχνά στους προσομοιωτές πτήσης.

  Το στοιχείο της ομίχλης ενεργοποιείται με την εντολή:
  - glEnable(GL.GL_FOG)

  Μπορούμε να αλλάξουμε το χρώμα της ομίχλης, από το προεπιλεγμένο που είναι το μαύρο, με τον εξής τρόπο:
  - float fogColor[] = {float red, float green, float blue, float alpha}
  - gl.glFogfv(GL.GL_FOG_COLOR, fogColor, 0)

  Επίσης με την παρακάτω συνάρτηση ελέγχουμε την ένταση της ομίχλης:
  - gl.glFogf(GL.GL_FOG_DENSITY, float density): όπου η τιμή της παραμέτρου density κυμαίνεται από το 0 μέχρι το 1, ανάλογα με την ένταση που επιθυμούμε

  Οι δύο συναρτήσεις που καθορίζουν τα σημεία έναρξης και λήξης του βάθους της ομίχλης καθορίζονται αντίστοιχα από τις παρακάτω συναρτήσεις:
  - gl.glFogf(GL.GL_FOG_START, float start)
  - gl.glFogf(GL.GL_FOG_END, float end)

Η τελική εικόνα της ομίχλης προκύπτει από την ανάμειξη του χρώματος τη ομίχλης με το χρώμα ενός αντικειμένου με τη χρήση ενός παράγοντα που ονομάζεται fog blending factor και συμβολίζεται με το γράμμα f. Ο παράγοντας αυτός υπολογίζεται με μια από τις παρακάτω συναρτήσεις:

$$f = e^{-(density \cdot z)} \quad \text{(GL\_EXP)}$$
$$f = e^{-(density \cdot z)^2} \quad \text{(GL\_EXP2)}$$
$$f = \frac{end - z}{end - start} \quad \text{(GL\_LINEAR)}$$

όπου z η απόσταση του αντικειμένου από την κάμερα
Η εντολή που καθορίζει την συνάρτηση που θα χρησιμοποιήσουμε είναι η:

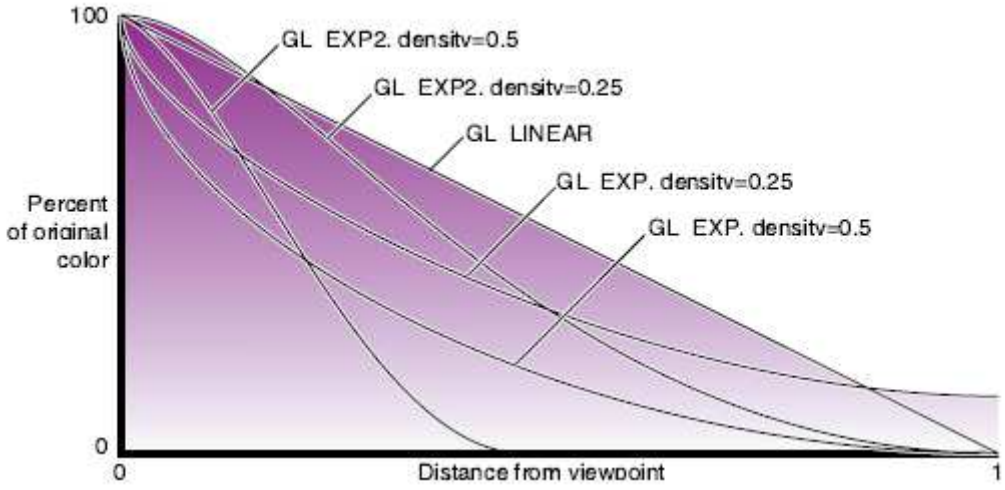
- gl.glFogi(GL.GL_FOG_MODE, GL.param): όπου η παράμετρος param παίρνει μια από τις τιμές GL_EXP (είναι η προεπιλεγμένη), GL_EXP2 ή GL_LINEAR

Μια άλλη μορφή της παρακάτω συνάρτησης είναι αυτή όπου σαν δεύτερο όρισμα έχουμε έναν πίνακα από παραμέτρους ο οποίος αποτελείται από ένα υποσύνολο των παραπάνω δυνατών τιμών. Ένα παράδειγμα τέτοιου πίνακα είναι ο παρακάτω:

- int fogMode[] = {GL.GL_EXP, GL.GL_EXP2, GL.GL_LINEAR}

Σε αυτή την περίπτωση το δεύτερο όρισμα ένα στοιχείο του παραπάνω πίνακα ανάλογα με την τιμή του fogfilter, όπου fogfilter δείκτης σε στοιχείο του πίνακα:

- gl.glFogi(GL.GL_FOG_MODE, fogMode[fogfilter]):
  στο συγκεκριμένο παράδειγμα το fogfilter παίρνει τιμές από 0 μέχρι 2



Εικόνα 16: Οι συναρτήσεις ομίχλης – έντασης για διάφορες τιμές των παραμέτρων

8.6. Το Πρόγραμμα LightingApp

Ακολουθεί ένα απλό πρόγραμμα για την κατανόηση του τρόπου χρωματισμού και φωτισμού των αντικειμένων της JOGL. Η εφαρμογή σχεδιάζει τρία αντικείμενα διαφορετικού χρώματος. προσθέτει στο σκηνικό μια πηγή φωτός, ομίχλη και ενεργοποιεί τη λειτουργία blendig. Στην ενότητα *Ανάπτυξη Υποδειγματικών Εφαρμογών* υπάρχει ένα πιο εξελιγμένο πρόγραμμα το οποίο παρουσιάζει τον τρόπο διαχωρισμού του συνολικού φωτισμού της σκηνής από τα επιμέρους είδη φωτισμού των αντικειμένων.

Το κύριο πρόγραμμα με την ονομασία LightingApp δεν αναφέρεται για λόγους συντομίας και προχωράμε στην παρουσίαση του προγράμματος LightingView στο οποίο σχεδιάζουμε το σκηνικό μας:

```
import com.sun.opengl.util.GLUT;
import java.awt.*;
import javax.media.opengl.*;
import javax.media.opengl.glu.GLU;

/**
* For our purposes only two of the
* GLEventListeners matter. Those would
* be init() and display().
*/
public class LightingView implements GLEventListener
{
    int anglex = 0;
    int angley = 0;

    GLCanvas glc;

     void setGLCanvas(GLCanvas glcanvas) {
        this.glc = glcanvas;
    }

/**
* Take care of initialization here.
*/

 public void init(GLAutoDrawable gld) {
    }
```

Στο παράδειγμα αυτό βλέπουμε ότι όλες οι αρχικοποιήσεις γίνονται στην μέθοδο display(). Ωστόσο, τόσο η τοποθέτηση της κάμερας όσο και οι απαραίτητες εντολές για την δημιουργία της φωτεινής πηγής θα μπορούσαν να γίνουν και στην init().

```
/**
* Take care of drawing here.
*/
public void display(GLAutoDrawable drawable) {

    GL gl = drawable.getGL();
    GLU glu = new GLU();
    GLUT glut = new GLUT();

    gl.glMatrixMode(GL.GL_PROJECTION);
    gl.glLoadIdentity();

    // Aspect is width/height
    double w = ((Component) drawable).getWidth();
    double h = ((Component) drawable).getHeight();
    double aspect = w/h;
    //When using gluPerspective near and far need
```

```
//to be positive.
//The arguments are:
//fovy, aspect, near, far
glu.gluPerspective(60.0, aspect, 2.0, 10.0);

gl.glMatrixMode(GL.GL_MODELVIEW);
gl.glLoadIdentity();

gl.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

//bleach out the scene with too much light.
//Define points for eye, at and up.
//This is your camera. It ALWAYS goes
//in the GL_MODELVIEW matrix.
glu.gluLookAt(
0, 0, 0,
0, 0, 20,
0, 1, 0
);
gl.glClear(GL.GL_COLOR_BUFFER_BIT);
```

Αφού κάνουμε τις απαραίτητες ρυθμίσεις, όπως σε κάθε πρόγραμμα που είδαμε μέχρι τώρα, ξεκινάμε να δημιουργούμε την φωτεινή πηγή. Πρώτα ορίζουμε τη θέση του φωτός με τη χρήση του πίνακα position.

```
//Time to set up the light. We will only
//use one light. If we used multiple lights,
//we would want to dim them so as to not
float position[] = {0f, 15f, -30f, 0f};
gl.glLightfv(GL.GL_LIGHT0, GL.GL_POSITION, position,0);
```

Στη συνέχεια δίνουμε στην προσθέτουμε στην φωτεινή πηγή τα είδη φωτισμού που επιθυμούμε. Επιλέξαμε τα ambient light και diffuse light των οποίων οι τιμές χρώματος ορίζονται με του αντίστοιχου πίνακες.

```
float diffuse[] = {.7f,.7f,.7f,0f};
gl.glLightfv(GL.GL_LIGHT0, GL.GL_DIFFUSE, diffuse,0);

float ambient[] = {.2f,.2f,.2f,0f};
gl.glLightfv(GL.GL_LIGHT0,GL.GL_AMBIENT,ambient,0);
```

Στο σημείο αυτό τελειώσαμε με τις αρχικοποιήσεις και προχωρούμε στην ενεργοποίηση του γενικού φωτισμού, αρχικά, και στη συνέχεια της πηγής φωτός που έχουμε δημιουργήσει.

```
gl.glEnable(GL.GL_LIGHTING);
gl.glEnable(GL.GL_LIGHT0);
```

Προχωράμε με την ενεργοποίηση του Depth Test ώστε να μην αποκρύπτονται πολύγωνα που κανονικά δεν θα έπρεπε να εμφανίζονται και στον καθαρισμό των Color και Depth Buffers.

```
//GL_DEPTH_TEST prevents us from seeing polygons
//that should be obscured. Remember to clear the
//GL.GL_DEPTH_BUFFER_BIT
gl.glEnable(GL.GL_DEPTH_TEST);

//Notice the depth buffer bit is also being cleared
gl.glClear(GL.GL_COLOR_BUFFER_BIT|GL.GL_DEPTH_BUFFER_BIT);
```

Οι επόμενες δύο εντολές ενεργοποιούν τη λειτουργία blending επιλέγοντας ως παράγοντα ανάμειξης και για τις τρις τιμές RGB την τιμή του Aplha, η οποία ισούται με τη μονάδα. επιλέγουν το επιθυμητό shade model. Αμέσως μετά επιλέγουμε shade model smooth ώστε οι πλευρές των πολυγώνων να μπορούν να έχουν παραπάνω από ένα χρώματα.

```
    gl.glBlendFunc(GL.GL_SRC_ALPHA, GL.GL_ONE);
    gl.glEnable(GL.GL_BLEND);

    gl.glShadeModel(GL.GL_SMOOTH);
```

Το τμήμα κώδικα που ακολουθεί ενεργοποιεί τη λειτουργία της ομίχλης με τις κατάλληλες εντολές. Αρχικά επιλέγει το χρώμα που θα έχει, στη συνέχεια επιλέγει το fog mode αλλά και την έντασή της. Τέλος, με τη χρήση της μεθόδου glHint() επιλέγουμε ότι ο τρόπος υπολογισμού της ομίχλης δεν είναι κάτι που μας απασχολεί.

```
    gl.glEnable(GL.GL_FOG);
    float fogColor[] = {0.5f, 0.5f, 0.5f, 1.0f};
    int fogMode = GL.GL_EXP;
    gl.glFogi(GL.GL_FOG_MODE, fogMode);  // Fog Mode
    gl.glFogfv(GL.GL_FOG_COLOR, fogColor, 0);  // Set Fog Color
    gl.glFogf(GL.GL_FOG_DENSITY, 0.1f);    // How Dense Will The Fog Be
    gl.glFogf(GL.GL_FOG_START, 1.0f);    // Fog Start Depth
    gl.glFogf(GL.GL_FOG_END, 5.0f);      // Fog End Depth
    gl.glHint(GL.GL_FOG_HINT, GL.GL_DONT_CARE);  // Fog Hint Value
```

Με την παρακάτω γραμμή κώδικα επιλέγουμε η διαδικασία εξομάλυνσης της εμφάνισης των πολυγώνων να εξασφαλίζει την καλύτερη οπτικά ποιότητα.

```
    gl.glHint(GL.GL_POLYGON_SMOOTH_HINT,GL.GL_NICEST);
```

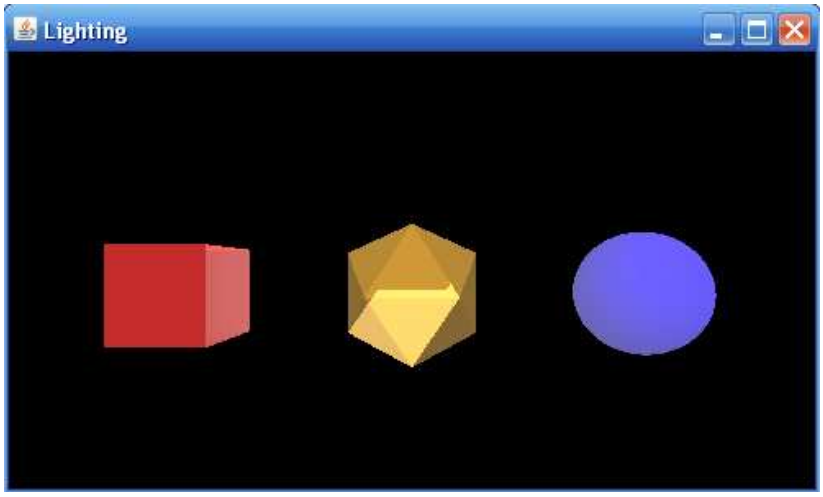
Το σετ με τις τέσσερις εντολές που ακολουθεί είναι υπεύθυνο για τη σωστή εμφάνιση των αντικειμένων μας. Αρχικά επιλέγουμε με τη μέθοδο glTranslated() το σημείο εμφάνισης του αντικειμένου. Καταχωρούμε στον πίνακα material1 το χρώμα του αντικειμένου και με τη επόμενη εντολή φωτίζουμε το αντικείμενό μας ώστε να φανεί το χρώμα του. Η πλευρές που θα φωτιστούν είναι η εμπρός και η πίσω, τα είδη φωτισμού είναι τα ambient και diffuse, και φυσικά το χρώμα του θα είναι αυτό που αποθηκεύσαμε στον πίνακα material1. Τέλος, με την κατάλληλη εντολή ζωγραφίζουμε ένα εικοσάεδρο.

```
    //Placing icosahedron
    gl.glTranslated(0, -.3, 5);
    float material1[] = {0.9f, 0.6f, 0.06f, 1.0f};
gl.glMaterialfv(GL.GL_FRONT_AND_BACK,GL.GL_AMBIENT_AND_DIFFUSE,material1,0);
    glut.glutSolidIcosahedron();
```

Με τον ίδιο τρόπο στο ίδιο σκηνικό ζωγραφίζουμε έναν κύβο και μια σφαίρα.

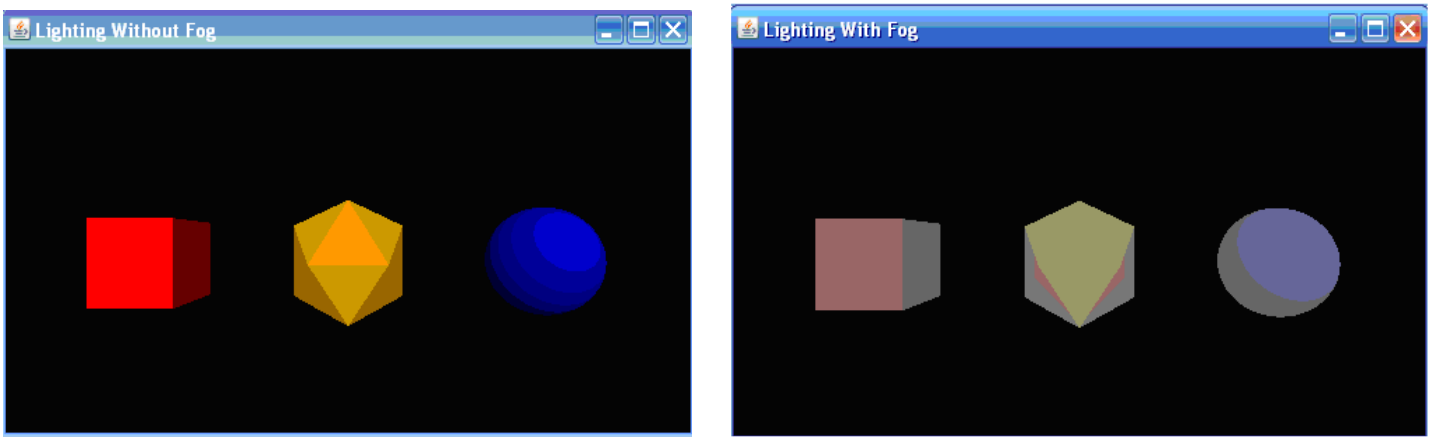
```
    //Placing cube
    gl.glTranslated(3, 0, 0);
    float material2[] = {0.9f,0f,0f,1.0f};
gl.glMaterialfv(GL.GL_FRONT_AND_BACK,GL.GL_AMBIENT_AND_DIFFUSE,material2,0);
    glut.glutSolidCube(1.2f);

    //Placing sphere
    //The Cone parameters are:
    // radius of base
    // height of cone
    // slices or subdivisions around cone
    // stacks or subdivisions from base to peak
    gl.glTranslated(-6, 0, 0);
    float material3[] = {0.05f,0f,0.7f,1.0f};
gl.glMaterialfv(GL.GL_FRONT_AND_BACK,GL.GL_AMBIENT_AND_DIFFUSE,material3,0);
    glut.glutSolidSphere(0.8, 100, 360);

    }

/**
* Called when the GLDrawable (GLCanvas
* or GLJPanel) has changed in size. We
* won't need this, but you may eventually
```

```
* need it -- just not yet.
*/
public void reshape(GLAutoDrawable drawable, int x, int y, int width, int height) {
    }

public void displayChanged(GLAutoDrawable glad, boolean bln, boolean bln1) {
    }
}
```



Εικόνα 17: Το αποτέλεσμα του φωτισμού και των εφέ που χρησιμοποιήσαμε στην εφαρμογή Lighting

Με τη διαγραφή των εντολών που ενεργοποιούν την ανάμειξη των χρωμάτων των pixels (blending) το αποτέλεσμα που προκύπτει είναι οπτικά πιο κατανοητό καθώς κάνει εμφανή την παρουσία τις ομίχλης, όπως φαίνεται και στην επόμενη εικόνα:



Εικόνα 18: Το στοιχείο της ομίχλης και ο τρόπος που επιδρά στα ίδια αντικείμενα φωτισμένα με τον ίδιο τρόπο

9. Display Lists

9.1. Θεωρία

Μια display list είναι μια ομάδα από εντολές OpenGl οι οποίες έχουν αποθηκευτεί για μελλοντική χρήση. Κάθε φορά που ενεργοποιείται μια display list τότε εκτελούνται οι εντολές που περιέχει με τη σειρά με την οποία έχουν γραφτεί. Μια display list μπορεί να δημιουργηθεί μέσα σε οποιοδήποτε πρόγραμμα και ο τρόπος παρουσιάζεται αμέσως μετά.

Οι display lists μπορούν επίσης να χρησιμοποιηθούν για την αποθήκευση ενός γεωμετρικού σχήματος το οποίο επιθυμούμε να επαναλάβουμε παραπάνω από μια φορά στο πρόγραμμά μας. Παραδείγματος χάριν, όταν θέλουμε να σχεδιάζουμε ένα ποδήλατο μπορούμε να αποθηκεύσουμε το γεωμετρικό σχήμα μιας ρόδας ποδηλάτου σε μια display list, έτσι θα ενεργοποιήσουμε τη λίστα αυτή 2 φορές ώστε να σχεδιάσουμε 2 ίδιες ρόδες.

Εξίσου χρήσιμη είναι η δυνατότητα να σχεδιάσουμε το ίδιο σχήμα σε διαφορετική θέση ή ακόμα και με διαφορετικό μέγεθος. Έτσι μπορούμε να μετακινήσουμε αντικείμενα μέσα στον χώρο καλώντας για κάθε μετακίνηση την display list που περιέχει το αντικείμενό μας αλλάζοντας κάθε φορά τις παραμέτρους της συνάρτησης gluLookAt(). Επίσης, μπορούμε να εμφανίσουμε ίδια αντικείμενα σε διαφορετικές θέσεις με αλλαγή της συνάρτησης glTranslated() κάθε φορά που ενεργοποιούμε τη λίστα.

Σχετικά με το προγραμματιστικό κομμάτι, κάθε λίστα έχει έναν ακέραιο αριθμό ως αναγνωριστικό. Όταν δημιουργούμε μια λίστα θα πρέπει να είμαστε προσεχτικοί ώστε να διαλέξουμε ένα αριθμό ο οποίος να μην αντιστοιχεί σε ήδη υπάρχουσα λίστα. Διαφορετικά, θα ξαναγράψουμε επάνω στην ίδια λίστα. Για να αποφύγουμε το πρόβλημα αυτό χρησιμοποιούμε τη παρακάτω συνάρτηση για να παράγουμε έναν ή περισσότερους δείκτες για μια συγκεκριμένη λίστα:

- glGenList(int range):
  Παίρνει ως όρισμα το πλήθος των δεικτών που χρειαζόμαστε να δείχνουν σε μία λίστα (display list indices). Όταν δημιουργούμε μια λίστα η τιμή του range θα πρέπει να είναι ίση ή μεγαλύτερη της μονάδας. Η συνάρτηση, επιστρέφει ένα ακέραιο ο οποίος σηματοδοτεί την αρχή του block των ελευθέρων display list δεικτών που ζητήσαμε. Οι δείκτες αυτοί δεσμεύονται και δεν επιστρέφονται σε επόμενη κλήση της συνάρτησης αν δεν διαγραφούν. Τέλος, η τιμή 0 επιστρέφεται όταν ο επιθυμητός αριθμός δεικτών δεν είναι διαθέσιμος ή όταν η τιμή του range είναι 0.

Την επιστρεφόμενη τιμή της συνάρτησης την αποθηκεύουμε σε έναν ακέραιο ο οποίος αποτελεί το αναγνωριστικό της display list. Στη συνέχεια δημιουργούμε την display list με τον εξής τρόπο:

- gl.NewList(int list,GL.mode):
  όπου list είναι το μοναδικό αναγνωριστικό της λίστας, το οποίο δημιουργήσαμε με την παραπάνω συνάρτηση. Η παράμετρος mode παίρνει την τιμή GL_COMPILE, όταν θέλουμε οι εντολές της display list να εκτελούνται κάλε φορά που κάνουμε compile χωρίς να αποθηκευτούν , ενώ με την τιμή GL_COMPILE_AND_EXECUTE όταν θέλουμε να εκτελεστούν αμέσως και να τοποθετηθούν στην display list για μελλοντική χρήση.

Αμέσως μετά καλούμε τη μέθοδο στην οποία έχουμε γράψει της εντολές που θέλουμε να αποθηκεύσουμε στην display list. Αφού αποθηκεύσουμε το επιθυμητό περιεχόμενο στην display list καλούμε την παρακάτω συνάρτηση για να τερματίσουμε τη λίστα:

- glEndList()

Μια λίστα συνήθως δημιουργείται μέσα στην init, εκεί όπου αρχικοποιούμε το πρόγραμμα, μέσα στην display σε οποιοδήποτε σημείο επιθυμούμε μπορούμε να ενεργοποιήσουμε τη λίστα με τη συνάρτηση:

- glCallList(int list):
  όπου list το αναγνωριστικό της λίστας [1]

9.2. Το πρόγραμμα TheTorusList

Στη ενότητα αυτή αναλύω ένα πρόγραμμα που σχεδιάζει στο σκηνικό μας δύο σπείρες, διαφορετικού χρώματος και θέσης, με τη βοήθεια μιας display list. Η εφαρμογή αυτή περιλαμβάνεται στα demos της σελίδας Nehe. Ακολουθεί το πρόγραμμα TheTorusView μαζί με τις απαραίτητες επεξηγήσεις για κάθε κομμάτι κώδικα ενώ το κύριο πρόγραμμα TheTorusList δεν αναφέρεται καθόλου.

```
import com.sun.opengl.util.GLUT;
import java.awt.*;
import java.awt.event.*;
import javax.media.opengl.*;
import javax.media.opengl.glu.GLU;
/**
* For our purposes only two of the
* GLEventListeners matter. Those would
* be init() and display().
*/
public class TheTorusView implements GLEventListener, KeyListener
{
```

Πριν να ξεκινήσουμε την εγγραφή των μεθόδων μας δηλώνουμε τις απαραίτητες μεταβλητές. Στην ακέραιη μεταβλητή theTorus θα αποθηκευτεί το αναγνωριστικό τις λίστας που θα δημιουργήσουμε στη συνέχεια.

```
    double xPosition = 0;
    double zPosition = 0;
    double yPosition = 0;

    int theTorus;
    float anglex = 0;
    float angley = 0;

    float red = 0.0f;
    float green = 0.0f;
    float blue = 1.0f;


    GLCanvas glc;

     void setGLCanvas(GLCanvas glcanvas) {
        this.glc = glcanvas;
    }
```

Ακολουθεί η μέθοδος init() στην οποία εκτός από την τοποθέτηση της κάμερας και τον καθορισμό της ορατής περιοχής δημιουργούμε και τη λίστα.

```
/**
* Take care of initialization here.
*/
 public void init(GLAutoDrawable gld) {

    //The mode is GL.GL_MODELVIEW by default
    //We will also use the default ViewPort
    GLU glu = new GLU();
    GL gl = gld.getGL();

    gl.glMatrixMode(GL.GL_PROJECTION);
    gl.glLoadIdentity();

    // Aspect is width/height
    double w = ((Component) gld).getWidth();
    double h = ((Component) gld).getHeight();
    double aspect = w/h;
    //When using gluPerspective near and far need
    //to be positive.
    //The arguments are:
    //fovy, aspect, near, far
    glu.gluPerspective(60.0, aspect, 2.0, 20.0);
```

Οι επόμενες τέσσερις εντολές δημιουργούν τη λίστα με το αναγνωριστικό theTorus. Πρώτα παράγουμε το αναγνωριστικό, στη συνέχεια με την επιλογή GL_COLPILE επιλέγουμε ο κώδικας της λίστας να εκτελείται κάθε φορά που κάνουμε compile. Ο κώδικας που ακολουθεί την εντολή αυτή είναι και ο κώδικας της λίστας. Στην περίπτωσή μας επιλέξαμε να εισάγουμε των κώδικα αυτό σε μια μέθοδο και να την καλέσουμε στο κατάλληλο σημείο. Τέλος, διαγράφουμε την λίστα και απελευθερώνουμε τους δείκτες που είχαme δεσμευτεί προηγουμένως.

```
    theTorus = gl.glGenLists(1);
    gl.glNewList(theTorus, GL.GL_COMPILE);
    drawTorus(gl, 10, 25);
    gl.glEndList();
    }

/**
* Take care of drawing here.
*/
public void display(GLAutoDrawable drawable) {

    GL gl = drawable.getGL();
    GLU glu = new GLU();
    GLUT glut = new GLUT();

    gl.glMatrixMode(GL.GL_MODELVIEW);
    gl.glLoadIdentity();

    gl.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

    //Define points for eye, at and up.
    //This is your camera. It ALWAYS goes
    //in the GL_MODELVIEW matrix.
    glu.gluLookAt(
    xPosition, yPosition, zPosition,
    xPosition, yPosition, (zPosition+20),
    0, 1, 0
    );
    gl.glClear(GL.GL_COLOR_BUFFER_BIT);
```

Ακολουθεί ο κώδικας με τον οποίο επιλέγουμε μοντέλο σκίασης και δημιουργούμε μια πηγή φωτός με τια κατάλληλες εντολές. Τέλος, πραγματοποιούμαι το depth test ώστε να βελτιώσουμε την εμφάνιση του σκηνικού μας.

```
    gl.glShadeModel(GL.GL_SMOOTH);
    gl.glEnable(GL.GL_LINE_SMOOTH);

    float position[] = {0f, 15f, -30f, 0f};
    gl.glLightfv(GL.GL_LIGHT0, GL.GL_POSITION, position,0);

    float diffuse[] = {.7f,.7f,.7f,0f};
    gl.glLightfv(GL.GL_LIGHT0, GL.GL_DIFFUSE, diffuse,0);

    float specular[] = {.2f,.2f,.2f,0f};
    gl.glLightfv(GL.GL_LIGHT0, GL.GL_SPECULAR, specular,0);

    float ambient[] = {.2f,.2f,.2f,0f};
    gl.glLightfv(GL.GL_LIGHT0,GL.GL_AMBIENT,ambient,3);

    gl.glEnable(GL.GL_LIGHTING);
    gl.glEnable(GL.GL_LIGHT0);

    //GL_DEPTH_TEST prevents us from seeing polygons
    //that should be obscured. Remember to clear the
    //GL.GL_DEPTH_BUFFER_BIT
    gl.glEnable(GL.GL_DEPTH_TEST);

    //Notice the depth buffer bit is also being cleared
    gl.glClear(GL.GL_COLOR_BUFFER_BIT|GL.GL_DEPTH_BUFFER_BIT);
```

Στο σημείο αυτό ήρθε η ώρα να τοποθετήσουμε τα αντικείμενά μας με τον πλεόν γνωστό τρόπο. Καθορίζουμε το χρώμα τους, προσθέτουμε τον επιθυμητό φωτισμό και επιλέγουμε με τη μέθοδο glTranslated() το σημείο εμφάνισης του αντικειμένου. Στη συνέχεια με της δύο μεθόδους glRotated() επιτρέπουμε στον χρήστη να περιστραφεί ώστε να δει το αντικείμενο από άλλη οπτική γωνία.

```
    float material1[] = {red, green, blue, 1.0f};
gl.glMaterialfv(GL.GL_FRONT_AND_BACK,GL.GL_AMBIENT_AND_DIFFUSE,material1,0);

    //transforming the place the next shape
    //will be drawn.
    gl.glTranslated(0, 0, 4);
    gl.glRotated(anglex, 1, 0, 0);
    gl.glRotated(angley, 0, 1, 0);
```

Στην επόμενη σειρά καλούμε τη λίστα με τη βοήθεια της μεθόδου glCallList() περνώντας ως όρισμα το αναγνωριστικό με την ονομασία theTorus.

```
    //We use wire here because default
    //lighting is not good enough to
    //use when rendering the solid version
    gl.glCallList(theTorus);
```

Με την ίδια διαδικασία καλούμε για δεύτερη φορά τη λίστα και τοποθετούμε, σε διαφορετική θέση αυτή τη φορά, ένα ίδιο αντικείμενο.

```
    float material2[] = {1f, 0.2f, 0.2f, 1.0f};
    gl.glMaterialfv(GL.GL_FRONT_AND_BACK,GL.GL_AMBIENT_AND_DIFFUSE,material2,0);
    gl.glTranslated(0,0,2);
    gl.glCallList(theTorus);

    }
```

Στη συνέχεια, δημιουργούμε τη μέθοδο drawTorus() που αποτελεί τον κώδικα της λίστας. Στη μέθοδο αυτή με τη χρήση μαθηματικών συναρτήσεων δημιουργούμε το σχήμα μιας σπείρας. Για να το πετύχουμε αυτό επιλέγουμε είδος σχήματος το GL_QUAD_STRIP και τοποθετούμε τα σημεία με τη μέθοδο glVertex3d() στις συντεταγμένες x,y,z που παράγονται από τις μαθηματικές συναρτήσεις.

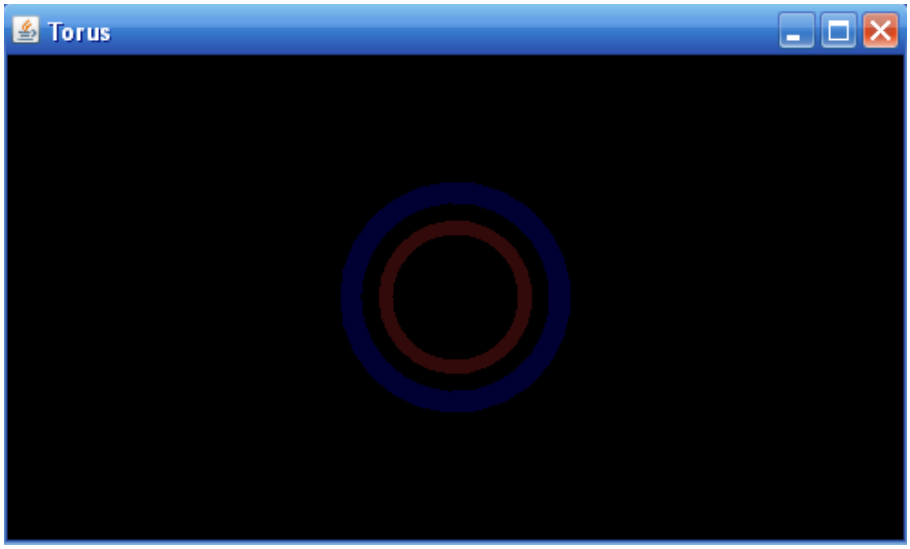
```
public void drawTorus(GL gl, int numc,int numt){
     double s, t, x, y, z, twopi = 2 * Math.PI;
    for (int i = 0; i < numc; i++)
    {
      gl.glBegin(GL.GL_QUAD_STRIP);
      for (int j = 0; j <= numt; j++)
      {
        for (int k = 1; k >= 0; k--)
        {
          s = (i + k) % numc + 0.5;
          t = j % numt;
          x = (1 + 0.1 * Math.cos(s * twopi / numc))
              * Math.cos(t * twopi / numt);
          y = (1 + 0.1 * Math.cos(s * twopi / numc))
              * Math.sin(t * twopi / numt);
          z = 0.1 * Math.sin(s * twopi / numc);
          gl.glVertex3d(x, y, z);
        }// k
      }// j
      gl.glEnd();
    }// i
 }

/**
* Called when the GLDrawable (GLCanvas
* or GLJPanel) has changed in size. We
* won't need this, but you may eventually
* need it -- just not yet.
*/
public void reshape(GLAutoDrawable drawable, int x, int y, int width, int height) {
    }

/**
* If the display depth is changed while the
* program is running this method is called.
* Nowadays this doesn't happen much, unless
* a programmer has his program do it.
*/
    public void displayChanged(GLAutoDrawable drawable, boolean modeChanged,
boolean deviceChanged) {
    }
```

Η μέθοδος keyTyped() επιτρέπει στον χρήστη να κινηθεί μέσα στο 3D σκηνικό με τη χρήση του πληκτρολογίου. Ο κώδικα είναι ο ίδιος με αυτόν που ήδη αναλύσαμε στο πρόγραμμα FirstPersonMovement.

```
    public void keyTyped(KeyEvent e) {
       if (e.getKeyChar() == KeyEvent.VK_1)
               zPosition += 1;
               else if (e.getKeyChar() == KeyEvent.VK_2)
               zPosition -= 1;
               else if (e.getKeyChar() == KeyEvent.VK_3)
               xPosition += 1;
               else if (e.getKeyChar() == KeyEvent.VK_4)
               xPosition -= 1;
               else if (e.getKeyChar() == KeyEvent.VK_5)
               yPosition += 1;
               else if (e.getKeyChar() == KeyEvent.VK_6)
               yPosition -= 1;
```

```
                else if (e.getKeyChar() == KeyEvent.VK_7)
                anglex += 1;
                else if (e.getKeyChar() == KeyEvent.VK_8)
                anglex -= 1;
                else if (e.getKeyChar() == KeyEvent.VK_9)
                angley += 1;
                else if (e.getKeyChar() == KeyEvent.VK_0)
                angley -= 1;
                glc.repaint();

    }

    public void keyPressed(KeyEvent ke) {

    }

    public void keyReleased(KeyEvent ke) {

    }

}
```

Το σκηνικό που δημιουργήσαμε φαίνεται στην εικόνα 19. Ο πλήρης κώδικας που προγράμματος που περιέχει και τις δύο κλάσεις (TheTorusList και TheTorusView) υπάρχει στα αρχεία demos.



Εικόνα 19: Το αντικείμενο της σπείρας σε διαφορετική θέση και με διαφορετικό χρώμα

10. Textures

10.1. Θεωρία

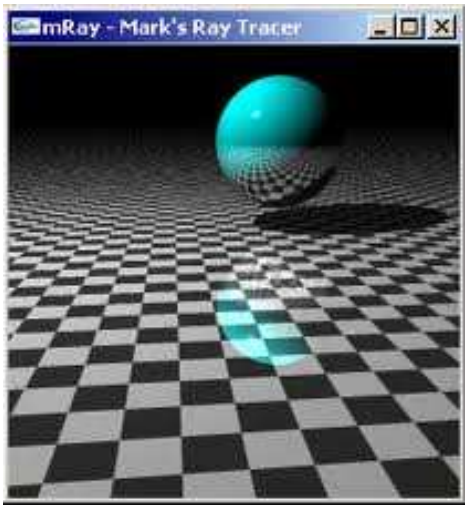
Τα αντικείμενα που σχεδιάσαμε μέχρι τώρα διέθεταν ένα μόνο χρώμα ή μια μείξη χρωμάτων, τεχνική που επιτυγχάνεται με τη χρήση του smooth shading που αναφέρθηκε παραπάνω. Όμως, για την ρεαλιστική αναπαράσταση ορισμένων αντικειμένων, όπως αυτά που διαθέτουν μια χαρακτηριστική υφή, αυτό δεν είναι αρκετό.

Παραδείγματος χάριν, για το σχεδιασμό ενός τοίχου με τούβλα θα μπορούσαμε να χρησιμοποιήσουμε μια σειρά από ορθογώνια ίδιου χρώματος που θα απεικόνιζαν τα τούβλα. Ωστόσο, η τεχνική αυτή θα είχε σαν αποτέλεσμα την δημιουργία ενός επίπεδου μονόχρωμου τοίχου που θα απείχε πολύ από την πραγματικότητα.

Για να αντιμετωπίσουμε τέτοιου είδους καταστάσεις χρησιμοποιούμε ένα στοιχείο της OpenGL που ονομάζεται texture. Το στοιχείο αυτό επιτρέπει την προσκόλληση μιας εικόνας στην επιφάνεια ενός βασικού πολυγώνου της OpenGL. Έτσι, μπορούμε να χρησιμοποιήσουμε την εικόνα ενός πραγματικού τοίχου από τούβλα και να την εμφανίσουμε σε ένα ορθογώνιο που θα αποτελεί το πλαίσιο του τοίχου μας. Η εικόνα που απεικονίζει τον τοίχο έχει τις ίδιες ιδιότητες με ένα αντικείμενο της OpenGL. Καθώς απομακρυνόμαστε από το αντικείμενο, ο τοίχος και τα τούβλα φαίνονται πιο μικρά, όπως ακριβώς συμβαίνει και την πραγματικότητα.

Η τεχνική αυτή μπορεί να χρησιμοποιηθεί και σε άλλες περιπτώσεις, όπως είναι η απεικόνιση του εδάφους με μια εικόνα που θα δείχνει βλάστηση ή κάτι αντίστοιχο. Επίσης, υπάρχουν έτοιμα wallpapers και texture patterns που κάνουν τα αντικείμενα μας να φαίνονται ξύλινα, μάλλινα, μαρμάρινα κλπ. Υπάρχουν μονοδιάστατα, δισδιάστατα, καθώς και τρισδιάστατα textures τα οποία μπορούν να προσκολληθούν πάνω σε βασικά πολύγωνα, σε έτοιμα αντικείμενα της OpenGL καθώς και σε ομάδες πολυγώνων κ.α. Η χρήση των textures μπορεί να δώσει πολλές δυνατότητες σε έναν προγραμματιστή, ωστόσο η επιπλέον διερεύνηση του προσπερνάει τα πλαίσια αυτού του βιβλίου.

Κάτι που αξίζει να αναφέρουμε είναι ότι σε μεταλλικά αντικείμενα ή καθρέπτες μπορούμε να απεικονίσουμε την αντανάκλαση του γύρο περιβάλλοντος με την προσκόλληση σε αυτά μιας εικόνας που να το απεικονίζει. Για να φαίνεται η αντανάκλαση των αντικειμένων πιο ρεαλιστική, όπως στην παρακάτω εικόνα, χρησιμοποιούμε τεχνικές που μας επιτρέπουν να αναμείξουμε ένα texture με το υπάρχον χρώμα ενός αντικειμένου.



Εικόνα 20: Η αντανάκλαση αντικειμένων με τη χρήση των textures

Για να χρησιμοποιήσουμε μια εικόνα ως texture θα πρέπει αρχικά να δημιουργήσουμε ένα αντικείμενο τύπου Texture στο οποίο θα εκχωρήσουμε την εικόνα αφού τη φορτώσουμε στο πρόγραμμα μας.

- private Texture aTexture;

Στη συνέχεια, σε οποιοδήποτε σημείο του προγράμματός μας, μπορούμε να εκχωρήσουμε την εικόνα ενός αρχείου στην μεταβλητή aTexture με τη χρήση μια συνάρτησης που επιστρέφει ένα αντικείμενο τύπου Texture, παραδείγματος χάριν:

- aTexture = load("anImg.jpg");

Είναι προτιμότερο η διαδικασία αυτή να γίνεται με τη χρήση συνάρτησης η οποία θα μπορεί να χρησιμοποιηθεί πολλές φορές και για διαφορετικές εικόνες, αλλάζοντας μόνο το όνομα του αρχείου το οποίο περιέχει την εικόνα. Μια τέτοια συνάρτηση είναι και η παρακάτω η οποία φορτώνει μια εικόνα τύπου .jpg:

```
Texture load(String filename){
   Texture texture = null;

   try
  {
    //Create an OpenGL texture from the specified file. Do not            //create
    mipmaps.

   texture = TextureIO.newTexture (new File (filename), false);

    //Use the NEAREST magnification function when the pixel being
    //textured maps to an area less than or equal to one texture
    //element (texel).

   texture.setTexParameteri(GL.GL_TEXTURE_MAG_FILTER,GL.GL_NEAREST);

    //Use the NEAREST minification function when the pixel being
    //textured maps to an area greater than one texel.

   texture.setTexParameteri(GL.GL_TEXTURE_MIN_FILTER, GL.GL_NEAREST);
  }
  catch (Exception e)
  {
   System.out.println ("error loading texture from "+filename);
  }

  return texture;
}
```

Το texture που φορτώσαμε με τον παραπάνω τρόπο μπορούμε να το χρησιμοποιήσουμε για να δημιουργήσουμε την επιθυμητή υφή σε οποιοδήποτε αντικείμενο της OpenGL, είτε το έχουμε σχεδιάσει εμείς, είτε είναι έτοιμο.

Ανάλογα με το αν πρόκειται για έτοιμο αντικείμενο της JOGL ή για αντικείμενο που σχεδιάζουμε εμείς η διαδικασία που ακολουθούμε είναι διαφορετική. Στην δεύτερη περίπτωση θα πρέπει να αποθηκεύσουμε τις συντεταγμένες της εικόνας που θα χρησιμοποιήσουμε σαν υφή και να τις αντιστοιχήσουμε στις συντεταγμένες του αντικειμένου που θα σχεδιάσουμε.

10.2. Το πρόγραμμα TextureApp

Το πρόγραμμα που έχω βάλει σαν παράδειγμα μας δείχνει τον τρόπο με τον οποίο μπορούμε να εισάγουμε ένα texture σε αντικείμενο που σχεδιάζουμε εμείς. Ο κώδικας του προγράμματος TextureView στον οποίο γίνεται ο σχεδιασμός του αντικειμένου είναι ο παρακάτω:

```
import com.sun.opengl.util.GLUT;
import com.sun.opengl.util.texture.Texture;
import com.sun.opengl.util.texture.TextureCoords;
import com.sun.opengl.util.texture.TextureIO;
import java.awt.*;
import javax.media.opengl.*;
import javax.media.opengl.glu.GLU;
import java.io.*;

/**
* For our purposes only two of the
* GLEventListeners matter. Those would
* be init() and display().
*/
public class TextureView implements GLEventListener
{

    GLCanvas glc;
    private Texture wallTexture;
```

Στη μεταβλητή τύπου Texture αποθηκεύουμε την υφή με την οποία θα ντύσουμε το αντικείμενό μας. Για κάθε διαφορετική υφή που θέλουμε να εισάγουμε στο πρόγραμμά μας θα πρέπει να δηλώνουμε και μια μεταβλητή. Στη μεταβλητή αυτή αποθηκεύουμε την υφή για να τη χρησιμοποιήσουμε οπουδήποτε σε οποιοδήποτε σημείο μέσα στο πρόπγραμμα.

```
     void setGLCanvas(GLCanvas glcanvas) {
        this.glc = glcanvas;
    }
```

Αυτό που κάνουμε στην init() είναι να αποθηκεύσουμε στην μεταβλητή wallTexture την υφή που επιστρέφει η συνάρτηση load. Η συνάρτηση αυτή δέχεται σαν όρισμα μια εικόνα και την επιστρέφει με τη μορφή υφής. Η load περιγράφεται παρακάτω.

```
/**
* Take care of initialization here.
*/
 public void init(GLAutoDrawable gld) {

     wallTexture = load("Wand.jpg");
    }

/**
* Take care of drawing here.
*/
public void display(GLAutoDrawable drawable) {


            GL gl = drawable.getGL();
            GLU glu = new GLU();
            GLUT glut = new GLUT();

            gl.glMatrixMode(GL.GL_PROJECTION);
            gl.glLoadIdentity();
            // Aspect is width/height
            double w = ((Component) drawable).getWidth();
            double h = ((Component) drawable).getHeight();
            double aspect = w / h;
            //When using gluPerspective near and far need
            //to be positive.
            //The arguments are:
            //fovy, aspect, near, far
            glu.gluPerspective(60.0, aspect, 2.0, 10.0);

            gl.glMatrixMode(GL.GL_MODELVIEW);
```

```
            gl.glLoadIdentity();
            gl.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
            //bleach out the scene with too much light.
            //Define points for eye, at and up.
            //This is your camera. It ALWAYS goes
            //in the GL_MODELVIEW matrix.
            glu.gluLookAt(0, 0, 0, 0, 0, 20, 0, 1, 0);

            gl.glClear(GL.GL_COLOR_BUFFER_BIT);
```

Στο σημείο αυτό έχουμε τελειώσει με τις αρχικοποιήσεις της θέσης της κάμερας και του οπτικού πεδίου και συνεχίζουμε με την τοποθέτηση του φωτισμού στο σκηνικό μας.

```
            //Time to set up the light. We will only
            //use one light. If we used multiple lights,
            //we would want to dim them so as to not
            // bleach out the scene with too much light.
            float[] position = {0f, 15f, -30f, 0f};
            gl.glLightfv(GL.GL_LIGHT0, GL.GL_POSITION, position, 0);
            float[] diffuse = {.7f, .7f, .7f, 0f};
            gl.glLightfv(GL.GL_LIGHT0, GL.GL_DIFFUSE, diffuse, 0);
            float[] ambient = {.2f, .2f, .2f, 0f};
            gl.glLightfv(GL.GL_LIGHT0, GL.GL_AMBIENT, ambient, 0);
            gl.glEnable(GL.GL_LIGHTING);
            gl.glEnable(GL.GL_LIGHT0);

            //GL_DEPTH_TEST prevents us from seeing polygons
            //that should be obscured. Remember to clear the
            //GL.GL_DEPTH_BUFFER_BIT
            gl.glEnable(GL.GL_DEPTH_TEST);
            //Notice the depth buffer bit is also being cleared
            gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
```

Μετά από την τοποθέτηση του φωτισμού, ο οποίος είναι απαραίτητος σε κάθε πρόγραμμα ώστε να φαίνονται τα αντικείμενα μας, σχεδιάζουμε τα αντικείμενά με την διαδικασία που ακολουθήσαμε και στο πρόγραμμα LinghtingView. Τα τρία σχήματα είναι ένα εικοσάεδρο, ένας κύβος και μια σφαίρα.

```
            //Placing icosahedron
            gl.glPushMatrix();
            gl.glTranslated(0, 0, 5);
            float[] material1 = {0.9f, 0.6f, 0.06f, 1.0f};
gl.glMaterialfv(GL.GL_FRONT_AND_BACK,GL.GL_AMBIENT_AND_DIFFUSE,material1,0);
            glut.glutSolidIcosahedron();
            gl.glPopMatrix();

            //Placing cube
            gl.glPushMatrix();
            gl.glTranslated(3, 0, 5);
            float[] material2 = {0.9f, 0f, 0f, 1.0f};
gl.glMaterialfv(GL.GL_FRONT_AND_BACK,GL.GL_AMBIENT_AND_DIFFUSE,material2,0);
            glut.glutSolidCube(1.2f);
            gl.glPopMatrix();

            //Placing sphere
            //The Cone parameters are:
            // radius of base
            // height of cone
            // slices or subdivisions around cone
            // stacks or subdivisions from base to peak
            gl.glPushMatrix();
            gl.glTranslated(-3, 0, 5);
            float[] material3 = {0.05f, 0f, 0.7f, 1.0f};
gl.glMaterialfv(GL.GL_FRONT_AND_BACK,GL.GL_AMBIENT_AND_DIFFUSE,material3,0);
            glut.glutSolidSphere(0.8, 100, 360);
            gl.glPopMatrix();
```

Στις παραπάνω γραμμές κώδικα χρησιμοποιήσαμε δύο καινούριες εντολές που δεν έχουμε δει μέχρι τώρα, τις glPushMatrix() και glPopMatrix(). Οι εντολή pushMatrix χρησιμοποιείται για να αποθηκεύσουμε την θέση του προηγούμενου αντικειμένου αποθηκεύοντας ουσιαστικά τις προηγούμενες τιμές του πίνακα translate. Η εντολή popMatrix από την άλλη επαναφέρει τις προηγούμενες, αποθηκευμένες, τιμές του πίνακα translate.

Έτσι λοιπόν η χρήση των glPushMatrix() και glPopMatrix() πριν και μετά τον σχεδιασμό καθενός από τα προηγούμενα αντικείμενα επαναφέρει την τιμή του πίνακα translate στη θέση που ορίστηκε στη συνάρτηση gluLookAt(), δηλαδή την (0,0,0).

```
drawWall(gl,7,0,3,0);
drawWall(gl,7,0,-3,0);
drawWall(gl,-7,0,-3,0);
drawWall(gl,-7,0,3,0);
```

Η συνάρτηση drawWall σχεδιάζει ένα ορθογώνιο στο οποίο εισάγουμε την υφή wallTexture ώστε να μοιάζει με τοίχο από τούβλα. Καλούμε τη συνάρτηση αυτή τέσσερις φορές και περνάμε ως παραμέτρους τις συντεταγμένες x και y των τεσσάρων σημείων που ορίζουν το ορθογώνιο στο οποίο θα «ζωγραφίζουμε» τον τοίχο. Ως αποτέλεσμα, θα σχηματίσουν τέσσερα μικρά ορθογώνια καθένα από τα οποία θα αναπαριστά ένα κομμάτι από τον τοίχο που θέλουμε να δημιουργήσουμε. Σκοπός της τεχνικής με τα τέσσερα ορθογώνια είναι η πιο ρεαλιστική αναπαράσταση του τοίχου ο οποίος θα αποτελείται από πιο μικρά τούβλα αντί για μεγαλύτερα που θα είχαμε αν δημιουργούσαμε μόνο ένα ορθογώνιο.

```
    }

void drawWall(GL gl,int x1, int x2, int y1,int y2){
```

Στη συνάρτηση drawWall το πρώτο που κάνουμε είναι να προσδιορίσουμε το χρώμα του ορθογωνίου που θα σχεδιάσουμε και να το φωτίσουμε με τη χρήση της μεθόδου glMaterialfv().

```
        float[] texMatMix = {1.0f, 1.0f, 1.0f, 1.0f};
gl.glMaterialfv(GL.GL_FRONT_AND_BACK,GL.GL_AMBIENT_AND_DIFFUSE,texMatMix,0);
```

Όταν σχεδιάζουμε τα δικά μας αντικείμενα είναι σημαντικό να προσαρμόζουμε τις συντεταγμένες του texture στις συντεταγμένες του αντικειμένου στο οποίο θέλουμε να εισάγουμε την υφή ώστε να το «ντύσουμε» με τον σωστό τρόπο. Ο κώδικας που υπολογίζει και αποθηκεύει της συντεταγμένες του texture είναι ο εξής:

```
TextureCoords tc = wallTexture.getImageTexCoords ();
float tx1 = tc.left ();
float ty1 = tc.top ();
float tx2 = tc.right ();
float ty2 = tc.bottom ();
// Enable two-dimensional texturing.´
```

Αφού αποθηκεύσουμε της συντεταγμένες του texture ξεκινάμε το σχεδιασμό του αντικειμένου, που στην περίπτωσή μας είναι ένα απλό ορθογώνιο. Πριν σχεδιάσουμε το αντικείμενο ενεργοποιούμε την κατάλληλη υφή, στην περίπτωση μας βρίσκεται αποθηκευμένη στη μεταβλητή wallTexture, και την απενεργοποιούμαι όταν δεν τη χρειαζόμαστε πια. Η εντολή wallTexture.bind() κάνει τα δεδομένα του αντικειμένου wallTexture διαθέσιμα ώστε να μπορούν να χρησιμοποιηθούν σε ένα textured model.

```
            wallTexture.enable ();

            // Bind this texture to the current rendering context.

            wallTexture.bind ();


            gl.glBegin(GL.GL_QUADS);
            gl.glVertex3i(x2, y1, 8);
            gl.glTexCoord2f (ty1, tx2);
            gl.glVertex3i(x2, y2, 8);
            gl.glTexCoord2f (ty2, tx2);
            gl.glVertex3i(x1, y2, 8);
            gl.glTexCoord2f (ty2, tx1);
            gl.glVertex3i(x1, y1, 8);
            gl.glTexCoord2f(ty1,tx1);
            gl.glEnd();
```

Η συνάρτηση glTexCoord2f (float x, float y) μας βοηθάει να αντιστοιχίσουμε μια κορυφή ενός texture με την αντίστοιχη κορυφή του αντικειμένου για το οποίο προορίζεται, ώστε το πρόγραμμα να γνωρίζει πως ακριβώς θα τοποθετήσει το texture μέσα στο αντικείμενο. Η κορυφή του αντικειμένου στην οποία θα αντιστοιχιστεί η κάθε κορυφή του texture είναι αυτή που σχεδιάστηκε ακριβώς πριν από αυτή με την εντολή gl.glVertex3i(x, y, z). Έτσι, σύμφωνα με τον παραπάνω κώδικα η κορυφή x2-y1-8 του ορθογωνίου θα αντιστοιχιστεί στην κορυφή ty1-tx2 του texture.

```
            wallTexture.disable();
}

/**
* Called when the GLDrawable (GLCanvas
* or GLJPanel) has changed in size. We
* won't need this, but you may eventually
* need it -- just not yet.
*/
public void reshape(GLAutoDrawable drawable, int x, int y, int width, int height) {
    }

public void displayChanged(GLAutoDrawable glad, boolean bln, boolean bln1) {
    }
```

Ακολουθεί η συνάρτηση load η οποία δέχεται ως όρισμα το όνομα ενός αρχείου εικόνας και επιστρέφει ένα αντικείμενο τύπου Texture με την εικόνα αυτή. Τη συνάρτηση αυτή μπορούμε να τη χρησιμοποιήσουμε σε οποιοδήποτε πρόγραμμα θέλουμε να χρησιμοποιήσουμε textures.

```
Texture load(String filename){
```

Πρώτα αρχικοποιούμε την επιστρεφόμενη μεταβλητή.

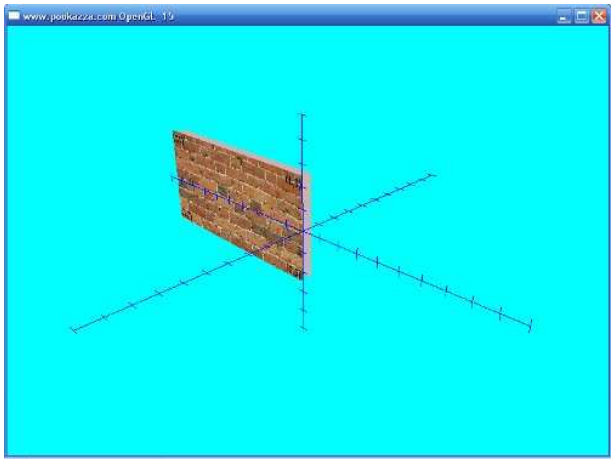
```
    Texture texture = null;
```

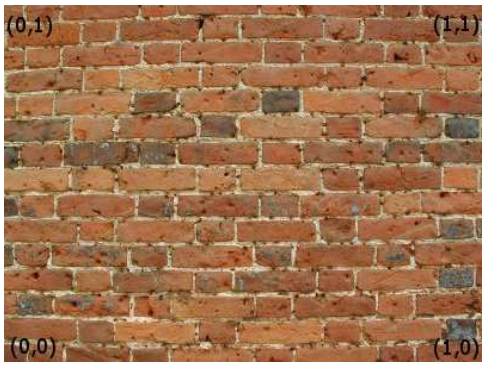
Ο κώδικας που είναι υπεύθυνος για το διάβασμα του αρχείου εικόνας περικλείεται σε αγκύλες try-catch ώστε να εμφανιστεί μήνυμα σφάλματος σε περίπτωση που το αρχείο δεν είναι διαθέσιμο.

```
      try
      {
          // Create an OpenGL texture from the specified file. Do not create
          // mipmaps.

          texture = TextureIO.newTexture (new File (filename), false);
```

```
            // Use the NEAREST magnification function when the pixel being
            // textured maps to an area less than or equal to one texture
            // element (texel).

            texture.setTexParameteri(GL.GL_TEXTURE_MAG_FILTER, GL.GL_NEAREST);

            // Use the NEAREST minification function when the pixel being
            // textured maps to an area greater than one texel.

            texture.setTexParameteri(GL.GL_TEXTURE_MIN_FILTER, GL.GL_NEAREST);
        }
        catch (Exception e)
        {
            System.out.println ("error loading texture from "+filename);
        }

        return texture;
    }
}
```

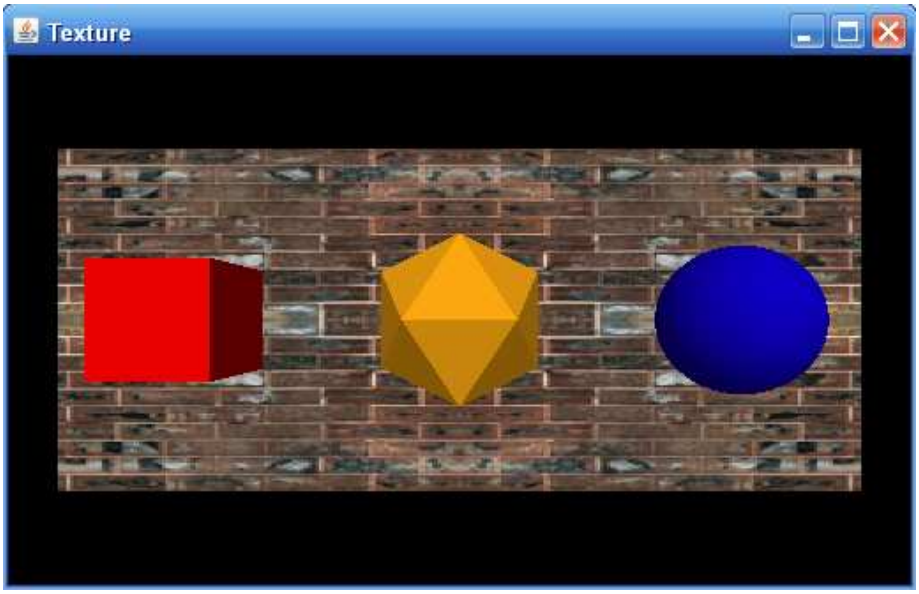
Τέλος, για ευκολότερη κατανόηση του τρόπου εισαγωγής ενός texture σε ένα αντικείμενο, παραθέτω τις παρακάτω δύο εικόνες. Οι εικόνες αυτές δείχνουν τις τιμές των μεταβλητών tx1, ty1, tx2 και ty2, που αναπαριστούν τις κορυφές ενός texture, όπως αντιστοιχίζονται στις κορυφές ενός ορθογωνίου με φορά σύμφωνη με την κατεύθυνση περιστροφής των δεικτών του ρολογιού.





Εικόνες 21 & 22: Η τοποθέτηση του ορθογωνίου στο σύστημα των συντεταγμένων και η αντιστοίχηση των κορυφών του texture (δεύτερη εικόνα) με τις αντίστοιχες κορυφές του ορθογωνίου [13]

Στο σημείο αυτό η περιγραφή του κώδικα έχει ολοκληρωθεί και ο χρήστης θα πρέπει να είναι πλέον ικανός να εισάγει και ο ίδιος υφή στα αντικείμενα που σχεδιάζει. Το πρόγραμμα SolarSystemView που περιγράφεται στην επόμενη ενότητα μας δείχνει πως μπορούμε να εισάγουμε textures σε έτοιμα αντικείμενα της JOGL.



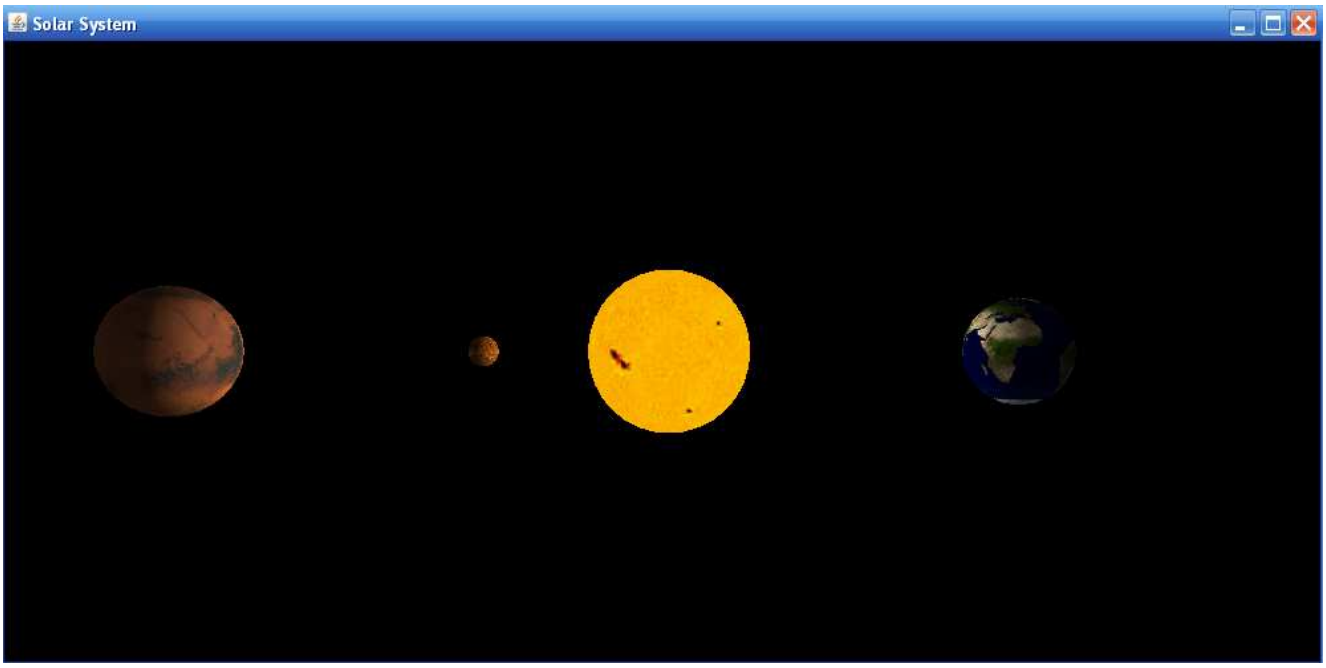
Εικόνα 23: Το αποτέλεσμα του προγράμματος TextureApp

III. Ανάπτυξη υποδειγματικής εφαρμογής

Στην ενότητα αυτή παραθέτω μια εφαρμογή την οποία βρήκα έτοιμη στη σελίδα robot.unipv.it, η οποία έχει αναπτυχθεί από το «Laboratorio di robotica». Στο tutorial που υπάρχει στη σελίδα αυτή με την ονομασία «Computer Graphics - Jogl Tutorials and Examples» υπάρχει θεωρία και έτοιμο υλικό με προγράμματα για κάθε ενότητα της JOGL. [14]

Το πρόγραμμα που αποτέλεσε τη βάση για την εφαρμογή που ακολουθεί περιέχεται στο αρχείο tutorial2.zip και ονομάζεται Practice07GLUPerspective. Το αρχικό πρόγραμμα σχεδίαζε τέσσερις σφαίρες διαφορετικού μεγέθους και χρώματος οι οποίες περιστρέφονταν. Επίσης, δύο από αυτές ήταν φωτιζόμενες και μια από αυτές κινούνταν με τη χρήση εισόδου από το πληκτρολόγιο.

Μετά από τροποποιήσεις και προσθήκες το πρόγραμμα πλέον σχεδιάζει ένα ηλιακό σύστημα στο ποίο οι αρχικές σφαίρες έχουν χρησιμοποιηθεί για την αναπαράσταση των πλανητών. Επίσης, το μόνο φωτιζόμενο αντικείμενο είναι μια σφαίρα που υπάρχει στο κέντρο του σκηνικού και αναπαριστά τον ήλιο. Οι υπόλοιπες τρεις σφαίρες περιστρέφονται σε τροχιά γύρο από αυτόν, αλλά και γύρο από τον εαυτό τους. Η τελική εικόνα του ηλιακού συστήματος που δημιουργήθηκε είναι η παρακάτω:



Εικόνα 24: Το αποτέλεσμα του προγράμματος SolarSystem με την απεικόνιση ενός ηλιακού συστήματος

Ακολουθεί ο κώδικας της εφαρμογής και μια σύντομη περιγραφή των επιπλέον πραγμάτων που περιλαμβάνει το πρόγραμμα και τα οποία δεν έχουν αναφερθεί μέχρι τώρα στη θεωρία. Τα εργαλεία της OpenGL που με οδήγησαν σε αυτό το αποτέλεσμα είναι η χρήση textures για την αναπαράσταση της επιφάνειας των πλανητών, η χρήση φωτισμού (lighting) για την απεικόνιση του αυτό-φωτιζόμενου ήλιου και η χρήση Animation σε συνδυασμό με τις εντολές glRotate οι οποίες εξασφαλίζουν την συνεχόμενη περιστρεφόμενη κίνηση των σωμάτων.

Το πρόγραμμα με την ονομασία SolarSystem το οποίο περιλαμβάνει τη main είναι αυτό που καλεί την επόμενη κλάση SolarSystemView για να σχεδιάσει το ηλιακό σύστημα. Η κλάση SolarSystemView παρουσιάζεται παρακάτω ενώ η κύρια κλάση SolarSystem δεν παρουσιάζεται ολόκληρη για ευνόητους λόγους. Το μοναδικό στοιχείο που θα πρέπει να προσθέσουμε στην κύρια αυτή κλάση είναι η δημιουργία ενός αντικειμένου τύπου Animator ώστε η εικόνα που θα σχεδιάσουμε να είναι κινούμενη και όχι στατική.

Για το σκοπό αυτό κάποιο σημείο της μεθόδου SolarSystem(), που ανήκει στην κλάση SolarSystem, προσθέτουμε τις εξής δύο γραμμές κώδικα:

```
Animator animator = new Animator(glcanvas);
animator.start();
```

Έτσι λοιπόν θέτουμε σε λειτουργία ένα αντικείμενο τύπου animator το οποίο για όσο χρονικό διάστημα το πρόγραμμα είναι ανοιχτό η μέθοδος display() της κλάσης SolarSystemView θα εκτελείται επανειλημμένα. Με αυτό τον τρόπο οι πλανήτες μας θα βρίσκονται σε συνεχή κίνηση σύμφωνα με τον τρόπο θα ορίσουμε εμείς στης μέθοδο display().

Ο πλήρης κώδικας της κλάσης SolarSystemView είναι ο παρακάτω:

```
import com.sun.opengl.util.texture.Texture;
import com.sun.opengl.util.texture.TextureIO;
import java.io.File;
import javax.media.opengl.*;
import javax.media.opengl.glu.GLU;
import javax.media.opengl.glu.GLUquadric;


public class SolarSystemView implements GLEventListener
{
```

Στο σημείο αυτό δηλώνουμε και αρχικοποιούμε τις μεταβλητές του προγράμματός μας. Η μεταβλητές t και k μας βοηθούν στην περιστροφή των αντικειμένων καθώς οι τιμές τους αλλάζουν κάθε φορά που εκτελείται η μέθοδος display(). Στις μεταβλητές w (width) και h (height) αποθηκεύονται οι διαστάσεις του παραθύρου μέσα στο οποίο θα εμφανιστεί η εφαρμογή μας. Στη συνέχεια υπάρχουν οι μεταβλητές στις οποίες θα αποθηκευτούν με τη μορφή textures οι εικόνες που θα εφαρμόσουμε στα αντικείμενά μας. Τέλος, με τη χρήση της μεταβλητής quadric θα σχεδιάσουμε τα αντικείμενά μας με τον γνωστό τρόπο.

```
       private static float t=0,k=0;
       private float w=1,h=1;
        private Texture earthTexture;
        private Texture marsTexture;
        private Texture venusTexture;
        private Texture sunTexture;
        private GLUquadric  quadric;


        GLCanvas glc;


     void setGLCanvas(GLCanvas glcanvas) {
        this.glc = glcanvas;
    }

/**
* Take care of initialization here.
```

```
*/

 public void init(GLAutoDrawable gld) {

       GL gl=gld.getGL();
       gl.glEnable(GL.GL_LIGHTING);

       float ambient[]= {0.2f,0.2f,0.2f,1};
       gl.glLightModelfv(GL.GL_LIGHT_MODEL_AMBIENT , ambient,0);
```

Η συνάρτηση glLightModelfv, ανάλογα με την τιμή τις πρώτης παραμέτρου, καθορίζει κάποιες ιδιότητες του μοντέλου φωτεινότητας. Στην περίπτωση μας, η ιδιότητα αυτή είναι η ένταση του διάσπαρτου φωτός που υπάρχει σε ολόκληρο το σκηνικό μας (GL_LIGHT_MODEL_AMBIENT). Η δεύτερη παράμετρος της συνάρτησης είναι ο πίνακας ambient ο οποίος περιέχει τις τιμές RGBA που καθορίζουν την ένταση του ambient φωτός. ***Addison Wesley2009 26/2/2011***

```
       gl.glEnable(GL.GL_LIGHT0);
       float position1[]= {-0.5f,0,00,1};
       gl.glLightfv(GL.GL_LIGHT0, GL.GL_POSITION, position1, 0);
       float intensity1[]= {1,1,1,1};
       gl.glLightfv(GL.GL_LIGHT0, GL.GL_DIFFUSE, intensity1, 0);
       float specIntensity2[]= {1,1,1,1};
       gl.glLightfv(GL.GL_LIGHT0, GL.GL_SPECULAR, specIntensity2, 0);
```

Στο σημείο αυτό έχουμε ήδη δημιουργήσει μια πηγή φωτός και προχωράμε σε αρχικοποιήσεις που αφορούν το υλικό (material) των αντικειμένων μας. Χρησιμοποιούμε την ιδιότητα ColorMaterial ώστε να δηλώσουμε εξ αρχής, με τη χρήση της μεθόδου glColorMaterial() της πλευρές των αντικειμένων που θα φωτίζονται καθώς και το είδος του φωτός που θα τα επηρεάζει. ***Addison Wesley200926/2/2011***

```
       gl.glEnable(GL.GL_COLOR_MATERIAL);
       gl.glColorMaterial(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT_AND_DIFFUSE);
       float specColor[]= {1,1,1,1};
       gl.glMaterialfv(GL.GL_FRONT_AND_BACK,GL.GL_SPECULAR, specColor,0);
       gl.glMaterialf(GL.GL_FRONT_AND_BACK,GL.GL_SHININESS, 80);
```

Η ενεργοποίηση του GL_COLOR_MATERIAL και η μέθοδος glColorMaterial(), σε αυτό το σημείο του κώδικα, απαλλάσσουν τον προγραμματιστή από τη χρήση την μεθόδου glMaterial() κάθε φορά που σχεδιάζει ένα αντικείμενο. Εστί λοιπόν το χρώμα κάθε αντικειμένου που δημιουργούμε αργότερα την display() καθορίζεται απλά και μόνο από τη συνάρτηση glColor3f(). Τέλος, καλούμε τη συνάρτηση load() τέσσερις φορές, μια για κάθε πλανήτη, ώστε να αποθηκεύσουμε στις αντίστοιχες μεταβλητές τις εικόνες των πλανητών με τη μορφή texture.

```
       earthTexture = load("earth.jpg");
       marsTexture = load("mars.jpg");
       venusTexture = load("venus.png");
       sunTexture = load("sun.jpg");
    }

/**
* Take care of drawing here.
*/
public void display(GLAutoDrawable gld) {

       GL gl=gld.getGL();
       gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);

       gl.glEnable(GL.GL_DEPTH_TEST);
       gl.glCullFace(GL.GL_FRONT);
       gl.glEnable(GL.GL_CULL_FACE);
```

```
gl.glFrontFace(GL.GL_CW);
```

Είναι η πρώτη φορά που συναντάμε τη μέθοδο glCullFace(), η οποία για να έχει ισχύ θα πρέπει οπωσδήποτε να ενεργοποιήσουμε το χαρακτηριστικό GL_CULL_FACE. Η μέθοδος αυτή καθορίζει ποία από τα πολύγωνα της OpenGL θα πρέπει να διαγραφούν πριν να μετατραπούν σε συντεταγμένες τις οθόνης και εμφανιστούν σε αυτή. Οι δυνατές τιμές είναι GL_FRONT, GL_BACK και GL_FRONT_AND_BACK. Η σταθερά GL_FRONT επιτρέπει να εμφανίζονται στην οθόνη μόνο οι συντεταγμένες των πολυγώνων που βρίσκονται «μπροστά» και είναι ορατά από τον χρήστη. Τέλος, η μέθοδος glFrontFace(), της οποίας η default τιμή είναι η GL_CW, καθορίζει τον τρόπο υπολογισμού των συντεταγμένων των πολυγώνων στην οθόνη. Η τιμή GL_CW υπολογίζει με αριστερόστροφο προσανατολισμό τις συντεταγμένες των πολυγώνων στην οθόνη. [1]

```
gl.glMatrixMode(GL.GL_PROJECTION);
gl.glLoadIdentity();
gl.glScalef(600/w, 600/h, 1);
```

Η τελευταία εντολή με τη χρήση της glScalef φροντίζει ώστε τα αντικείμενα μας να μην φαίνονται παραμορφωμένα με την αλλαγή του παραθύρου των windows αλλά να προσαρμόζονται σε αυτό.

```
//Projective transform
GLU glu=new GLU();
glu.gluPerspective(50.0f, 1, 1.0, 20.0);
gl.glTranslatef(0, 0, -10);
gl.glRotatef(0.2f*t,0,1,0);
 t++;
```

Οι δύο τελευταίες εντολές τοποθετούν τα αντικείμενά μας στην θέση -10 στον άξονα z και εξασφαλίζουν την συνεχή περιστροφή τους ως προς τον άξονα y. Πιο συγκεκριμένα η αύξηση της τιμής του t σε κάθε κλήση της display() περιστρέφει τους πλανήτες γύρο από τον ήλιο και τον τελευταίο γύρο από τον εαυτό του.

```
gl.glMatrixMode(GL.GL_MODELVIEW);
gl.glLoadIdentity();

quadric=glu.gluNewQuadric();
 glu.gluQuadricTexture(quadric, true);
```

Οι παραπάνω εντολές δημιουργούν ένα quadric το οποίο χρησιμοποιούμε για να καλέσουμε τα αυτόματα σχήματα της κλάσης GLU και να σχεδιάσουμε τις σφαίρες μας. Η δεύτερη εντολή καθιστά το quadric έτοιμο για να δεχθεί ένα texture. Αμέσως μετά καλούμε τη συνάρτηση glLoadIdentity() ώστε να αρχικοποιήσουμε τον πίνακα προβολής και να αποθηκεύσουμε σε αυτόν την θέση του επόμενου αντικειμένου που θέλουμε να σχεδιάσουμε. Στη συνέχεια, θα παρατηρήσουμε ότι καλούμε τη συνάρτηση αυτή πριν από τον σχεδιασμό κάθε αντικειμένου ώστε αυτό να εμφανιστεί σε νέα θέση και έτσι να εξασφαλίζεται η ορθή λειτουργία ενός animation.

```
 gl.glLoadIdentity();
gl.glTranslatef(0.5f,0,6);
 gl.glRotated(90, 1, 0, 0);
 gl.glRotatef(90*k, 0, 0, 1);
```

Η πρώτη εντολή glTranslatef τοποθετεί το αντικείμενο (πλανήτη) στην κατάλληλη αρχική θέση ώστε να κινείται σε μια σταθερή τροχιά. Η επόμενη εντολή glRotated(90, 1, 0, 0) περιστρέφει το αντικείμενο 90° στον άξονα του x ώστε μετά την εφαρμογή του texture ο πλανήτης να φαίνεται ότι έχει την σωστή πολικότητα. Η τελευταία εντολή glRotated(90*k,

0, 0, 1) είναι υπεύθυνη για την συνεχή περιστροφή των πλανητών γύρο από τον εαυτό τους καθώς η τιμή του k αυξάνεται, όπως θα δούμε παρακάτω, κάθε φορά που εκτελείται η display.

```
gl.glColor3f(1,1,1);
marsTexture.enable();
marsTexture.bind();
glu.gluSphere(quadric, 0.8f, 16, 16); //mars
marsTexture.disable();
```

Με της παραπάνω εντολές δημιουργούμε πρώτα τον πλανήτη Άρη. Αρχικά επιλέγουμε για χρώμα του αντικειμένου το λευκό έτσι ώστε η εικόνα (texture) που θα εφαρμοστεί στην επιφάνειά του να εμφανιστεί χωρίς την επιρροή κάποιου χρώματος. Στη συνέχεια, ενεργοποιούμε το κατάλληλο texture, που στην περίπτωση αυτή είναι αποθηκευμένο στη μεταβλητή marsTexture, και με την εντολή bind κάνουμε τα δεδομένα της εικόνας διαθέσιμα για χρήση. Τέλος, σχεδιάζουμε την σφαίρα με τη χρήση της εντολής gluSpherekαι απενεργοποιούμε το texture το οποίο δεν χρειαζόμαστε πλέον.

Με τον ίδιο τρόπο σχεδιάζουμε και τους υπόλοιπους πλανήτες μέχρι να φτάσουμε στο σχεδιασμό της σφαίρας που αναπαριστά τον ήλιο.

```
gl.glLoadIdentity();
gl.glTranslatef(-0.8f,0,2.5f);
gl.glRotated(90, 1, 0, 0);
gl.glRotatef(90*k, 0, 0, 1);

gl.glColor3f(1,1,1);
venusTexture.enable();
venusTexture.bind();
glu.gluSphere(quadric, 0.2f, 10, 10);//venus
venusTexture.disable();

gl.glLoadIdentity();
gl.glTranslatef(-0.2f,0,-4);
gl.glRotated(90, 1, 0, 0);
gl.glRotatef(90*k, 0, 0, 1);

gl.glColor3f(1,1,1);
earthTexture.enable();
earthTexture.bind();
glu.gluSphere(quadric, 0.6f, 10, 10);//earth
earthTexture.disable();
```

Στο σημείο αυτό θα πρέπει να απενεργοποιήσουμε τον φωτισμό καθώς ο ήλιος είναι το σώμα το οποίο αποτελεί την φωτεινή πηγή μας και δεν θα πρέπει να ενεργεί κανένα είδους φωτισμός επάνω του όταν το σχεδιάζουμε. Στη συνέχεια, σχεδιάζουμε τον ήλιο με τον ίδιο ακρυβώς τρόπο που σχεδιάσαμε τους υπόλοιπους πλανήτες.

```
gl.glDisable(GL.GL_LIGHTING);

gl.glLoadIdentity();
gl.glTranslatef(-0.5f,0,0);
gl.glRotated(90, 1, 0, 0);

gl.glColor3f(1,1,0);
sunTexture.enable();
sunTexture.bind();
glu.gluSphere(quadric, 1f, 16, 16);//sun
sunTexture.disable();
```

Με τις δύο τελευταίες εντολές αυξάνουμε τη μεταβλητή k, που βοηθάει στην περιστροφή των πλανητών γύρο από τον εαυτό τους, ώστε η τιμή της να έχει αλλάξει την επόμενη φορά που θα κληθεί η display. Επίσης, ενεργοποιούμε τον φωτισμό ώστε τα σώματα που θα σχεδιαστούν, και πάλι σε επόμενη κλήση της display, να επηρεάζονται φαίνεται ότι φωτίζονται από τον ήλιο.

```
        k=k+0.01f;
        gl.glEnable(GL.GL_LIGHTING);
    }
```

Είναι η πρώτη φορά που χρησιμοποιούμε τη μέθοδο reshape σε πρόγραμμα και ήρθε η ώρα να κατανοήσουμε τη χρησιμότητά της. Το αντικείμενο glCanvas αντιστοιχεί στο παράθυρο μέσα στο οποίο εμφανίζεται η εφαρμογή μας. Έτσι όταν αλλάζει η διάσταση του παραθύρου αυτού καλείται η μέθοδος reshape. Οι δύο μεταβλητές width και height που αποτελούν ορίσματα της μεθόδου αντιστοιχούν στις διαστάσεις του παραθύρου, έτσι λοιπόν σε κάθε κλήση της μεθόδου οι τιμές αυτές αποθηκεύονται στις μεταβλητές w και h. Οι μεταβλητές αυτές όπως έχουμε ήδη δει θα χρησιμοποιηθούν στην display για να εμφανίσουν τα αντικείμενά μας με τις σωστές διαστάσεις, μέσω της εντολής gl.glScalef(600/w, 600/h, 1).

```
/**
* Called when the GLDrawable (GLCanvas
* or GLJPanel) has changed in size.
*/
public void reshape(GLAutoDrawable drawable, int x, int y, int width, int height) {
    GL gl=drawable.getGL();
    w=width;
    h=height;
    }

public void displayChanged(GLAutoDrawable glad, boolean bln, boolean bln1) {
    }
```

Η τελευταία μέθοδος υλοποιούμε στο πρόγραμμα SolarSyatemView είναι η ίδια που χρησιμοποιήσαμε και στα υπόλοιπα προγράμματά μας για να ανοίξουμε μια εικόνα και να φορτώσουμε το περιεχόμενό της σε ένα texture. Η περιγραφή της μεθόδου παραλείπεται για λόγους συντομίας.

```
Texture load(String filename){
    Texture texture = null;

      try
      {
          // Create an OpenGL texture from the specified file. Do not create
          // mipmaps.

          texture = TextureIO.newTexture (new File (filename), false);

          // Use the NEAREST magnification function when the pixel being
          // textured maps to an area less than or equal to one texture
          // element (texel).

          texture.setTexParameteri(GL.GL_TEXTURE_MAG_FILTER, GL.GL_NEAREST);

          // Use the NEAREST minification function when the pixel being
          // textured maps to an area greater than one texel.

          texture.setTexParameteri(GL.GL_TEXTURE_MIN_FILTER, GL.GL_NEAREST);
      }
      catch (Exception e)
      {
          System.out.println ("error loading texture from "+filename);
      }
```

```
        return texture;
    }
}
```

IV. Ανάπτυξη της δικής μου εφαρμογής

1. Εισαγωγή

Στην τελευταία ενότητα της παρούσας εργασίας αναπτύσσω μια δική μου εφαρμογή. Πρόκειται για ένα πρόγραμμα το οποίο σχεδιάζει ένα 3D δωμάτιο που περιλαμβάνει κάποια βασικά αντικείμενα. Ο βασικός μου στόχος είναι η δυνατότητα περιήγησης του χρήστη σε ένα εικονικό κόσμο-δωμάτιο με τη χρήση απλά του πληκτρολογίου του.

Στο σημείο αυτό περιγράφω τα βασικά μέρη του προγραμματός μου τα οποία συνθέτουν το 3D σκηνικό. Όπως ανέφερα και παραπάνω ο κώδικας σχεδιάζει ένα δωμάτιο με κάποια βασικά έπιπλα. Πιο αναλυτικά, οι τοίχοι του δωματίου έχουν επενδυθεί με ταπετσαρία την οποία εφάρμοσα με χρήση texture. Επίσης, με τον ίδιο τρόπο σχεδίασα ένα ξύλινο δάπεδο και την οροφή του δωματίου με μια μονόχρωμη ταπετσαρία. Στην οροφή υπάρχει ένα φωτιστικό που αποτελείται από μια σφαιρική λάμπα σε ορθογώνια βάση. Η λάμπα αυτή αποτελεί και την πηγή φωτός που φωτίζει όλο το υπόλοιπο δωμάτιο. Τέλος, στους τοίχους του δωματίου τοποθέτησα ορισμένους πίνακες ζωγραφικής με τον εξής τρόπο: δημιούργησα κάποια ορθογώνια πλαίσια, τα οποία τοποθέτησα πάνω στους τοίχους του δωματίου και εφάρμοσα σε αυτά textures που απεικονίζουν πίνακες ζωγραφικής.

Όσον αφορά τα έπιπλα του δωματίου αυτά περιορίζονται σε μια τραπεζαρία με τέσσερις καρέκλες που είναι τοποθετημένα στο κέντρο του δωματίου. Το τραπέζι καθώς και οι καρέκλες διαθέτουν κάποια ξύλινα μέρη τα οποία προσομοιάζουν ορθογώνια κομμάτια ξύλου. Στα κομμάτια αυτά έχει εφαρμοστεί texture με όψη ξύλου. Το επάνω μέρος του τραπεζίου είναι ξύλινο και τα πόδια πάνω στα οποία στηρίζεται έχουν σχήμα κυλινδρικό και πράσινη μεταλλική απόχρωση.

Με τον ίδιο τρόπο είναι κατασκευασμένες και οι καρέκλες των οποίων η πλάτη και το κάθισμα αποτελείται από τα ίδια ξύλινα ορθογώνια που χρησιμοποιήσαμε και στο τραπέζι, μόνο που στην περίπτωση αυτή έχουν διαφορετικές διαστάσεις. Επίσης, και στις καρέκλες, χρησιμοποίησα κυλινδρικά πόδια που συγκρατούν τα υπόλοιπα ξύλινα μέρη τους. Τέλος, στο κέντρο του τραπεζιού υπάρχει μια τσαγέρα σε ανοιχτό μπλε μεταλλικό χρώμα. Με τη παρουσία του αντικειμένου της τσαγέρας στο σκηνικό μου ήθελα να τονίσω το γεγονός ότι αποτελεί ένα από τα βασικότερα έτοιμα αντικείμενα της OpenGL που συναντάμε συχνά σε προγράμματα.

Το project που δημιούργησα για την υλοποίηση του προγράμματός μου αποτελείται από δύο αρχεία – κλάσεις. Η πρώτη κλάση ονομάζεται Room και περιλαμβάνει την κύρια μέθοδο main μέσα στην οποία καλώ το άλλο πρόγραμμα μου, το RoomView. Στο δεύτερο πρόγραμμα υπάρχει ουσιαστικά όλος ο κώδικας της JOGL και είναι αυτό το οποίο σχεδιάζει ότι υπάρχει στο 3D σκηνικό. Ακλουθεί ο κώδικας του προγράμματος RoomView μαζί με τις απαραίτητες επεξηγήσεις για τη λειτουργία του κάθε κομματίου.

2. Ανάλυση του Κώδικα

Το πρώτο πράγμα που συναντάμε σε κάθε πρόγραμμα είναι η εισαγωγή των απαραίτητων κλάσεων.

```
import com.sun.opengl.util.GLUT;
import com.sun.opengl.util.texture.Texture;
import com.sun.opengl.util.texture.TextureCoords;
import com.sun.opengl.util.texture.TextureIO;
import java.awt.*;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import javax.media.opengl.*;
import javax.media.opengl.glu.GLU;
import java.io.*;

/**
* For our purposes only two of the
* GLEventListeners matter. Those would
* be init() and display().
*/
public class RoomView implements GLEventListener, KeyListener
{
```

Οι μεταβλητές που δηλώνω πρώτα είναι αυτές στις οποίες αποθηκεύεται η θέση της κάμερας, δηλαδή το σημείο του χώρου στο οποίο βρίσκεται ο χρήστης, και καθορίζει το οπτικό του πεδίο. Στις μεταβλητές angley και anglex αποθηκεύεται η γωνία περιστροφής της κάμερας και στις δύο τελευταίες ακέραιες μεταβλητές το αναγνωριστικό κάθε μιας από τις δύο λίστες που χρησιμοποιώ στο πρόγραμμα.

```
    double xPosition = 0;
    double zPosition = 4;
    double yPosition = 4;
    int angley = 0;
    int anglex = 0;
    int chair;
    int woodenCube;
```

Στη συνέχεια, δηλώνω της μεταβλητές που θα φιλοξενήσουν τα Textures που θα χρησιμοποιήσω στα αντικείμενά μου καθώς και τους πίνακες όπου αποθηκεύω τα απαραίτητα χρώματα.

```
    GLCanvas glc;
    private Texture wallTexture;
    private Texture floorTexture;
    private Texture ceilingTexture;
    private Texture woodTexture;
    private Texture paint1Texture;
    private Texture paint2Texture;
    private Texture paint3Texture;
    private Texture paint4Texture;
    private Texture paint5Texture;

    float[] materialGreen = {0.2f, 1.0f, 0.7f, 1.0f};
    float[] materialWhite = {1.0f, 1.0f, 1.0f, 1.0f};
    float[] materialBlack = {0.3f,0.3f,0.0f,0.0f};

     void setGLCanvas(GLCanvas glcanvas) {
        this.glc = glcanvas;
    }
/**
* Take care of initialization here.
*/
```

```
public void init(GLAutoDrawable gld) {
```

Το πρώτο πράγμα που κάνω στην init είναι να καλέσω την κατάλληλη μέθοδο load η οποία φορτώνει κάποιες εικόνες και τις αποθηκεύει με τη μορφή texture στις αντίστοιχες μεταβλητές. Οι εικόνες αυτές απεικονίζουν την υφή που θα έχουν κάποια από τα βασικά μέρη του δωματίου καθώς και οι πέντε πίνακες που θα υπάρχουν στους τοίχους.

```
wallTexture = load("wall.jpg");
floorTexture = load("floor.jpg");
ceilingTexture = load("ceiling.jpg");
woodTexture = load("wood.jpg");
paint1Texture = load("images1.jpg");
paint2Texture = load("images2.jpg");
paint3Texture = load("images3.jpg");
paint4Texture = load("images4.jpg");
paint5Texture = load("images5.jpg");

GLUT glut = new GLUT();
GL gl = gld.getGL();
GLU glu = new GLU();
```

Στο σημείο αυτό καλώ τις δύο μεθόδους που δημιουργούν τις λίστες που χρησιμοποιώ στο πρόγραμμά μου. Η πρώτη σχεδιάζει ένα ξύλινο ορθογώνιο, που χρησιμοποιείται στην κατασκευή της τραπεζαρίας, και η δεύτερη σχεδιάζει μια καρέκλα απαλλάσσοντάς μας από τον κόπο να σχεδιάζουμε τέσσερις ίδιες καρέκλες επαναλαμβάνοντας τον ίδιο κώδικα τέσσερις φορές.

```
WoodList(gl,glut);
buildChairList(gl,glut);
```

Στη συνέχεια, ενεργοποιώ το στοιχείο του φωτισμού και ενεργοποιώ δύο φώτα. Το πρώτο (GL_LIGHT0) είναι το φώς που εκπέμπει η λάμπα του δωματίου και όπως θα δούμε αργότερα τοποθετείται σε μια συγκεκριμένη θέση. Το δεύτερο (GL_LIGHT1) αντιπροσωπεύει τον γενικό φωτισμό που υπάρχει διάχυτος στο δωμάτιο.

```
gl.glEnable(GL.GL_LIGHTING);
gl.glEnable(GL.GL_LIGHT0);
gl.glEnable(GL.GL_LIGHT1);

// prepare spotlight
float spot_ambient[] =  {0.2f,0.2f,0.2f,1.0f };
float spot_diffuse[] =  {0.8f,0.8f,0.8f,1.0f };
float spot_specular[] =  {0.8f,0.8f,0.8f,1.0f };

gl.glLightfv(GL.GL_LIGHT0, GL.GL_AMBIENT,  spot_ambient,0);
gl.glLightfv(GL.GL_LIGHT0, GL.GL_DIFFUSE,  spot_diffuse,0);
gl.glLightfv(GL.GL_LIGHT0, GL.GL_SPECULAR, spot_specular,0);

// prepare a general fixed light
float fix_ambient[] =  {0.1f,0.1f,0.1f,1.0f };
float fix_diffuse[] =  {0.3f,0.3f,0.3f,1.0f };
float fix_specular[] =  {0.1f,0.1f,0.1f,1.0f };

gl.glLightfv(GL.GL_LIGHT1, GL.GL_AMBIENT,  fix_ambient,0);
gl.glLightfv(GL.GL_LIGHT1, GL.GL_DIFFUSE,  fix_diffuse,0);
gl.glLightfv(GL.GL_LIGHT1, GL.GL_SPECULAR, fix_specular,0);
gl.glEnable(GL.GL_LIGHT2);
```

Το τελευταίο πράγμα που κάνω στην init είναι να επιλέξω μοντέλο σκίασης και να ορίσω το πεδίο ορατότητας του χρήστη με τη μέθοδο gluPerspective.

```
      // smooth the drawing
      gl.glShadeModel(GL.GL_SMOOTH);

      gl.glMatrixMode(GL.GL_PROJECTION);
      gl.glLoadIdentity();

      // Aspect is width/height
      double w = ((Component) gld).getWidth();
      double h = ((Component) gld).getHeight();
      double aspect = w / h;

      //When using gluPerspective near and far need
      //to be positive.
      //The arguments are:
      //fovy, aspect, near, far
      glu.gluPerspective(60.0, aspect, 1.0, 200.0);
      glc.repaint();

    }
```

Ακολουθεί η μέθοδος display στην οποία, όπως και σε κάθε πρόγραμμα JOGL, γράφω τον κώδικα που σχεδιάζει τα αντικείμενα του 3D σκηνικού μου. Η τεχνική που ακολούθησα για μια πιο προσιτή παρουσίαση του κώδικα είναι η εξής: το κάθε αντικείμενο του σκηνικού μου σχεδιάζεται από μια ξεχωριστή μέθοδο την οποία καλώ στην display. Έτσι λοιπόν ο κώδικας είναι μοιρασμένος σε μεθόδους με την κατάλληλη ονομασία οι οποίες καθώς τις καλώ σχεδιάζουν τμηματικά το σκηνικό μου.

```
/**
* Take care of drawing here.
*/
public void display(GLAutoDrawable drawable) {

            GL gl = drawable.getGL();
            GLU glu = new GLU();
            GLUT glut = new GLUT();
            gl.glMatrixMode(GL.GL_MODELVIEW);
            gl.glLoadIdentity();
            gl.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

            //Define points for eye, at and up.
            //This is your camera. It ALWAYS goes
            //in the GL_MODELVIEW matrix.
            glu.gluLookAt(xPosition,yPosition,zPosition,xPosition,yPosition,
(zPosition+20), 0, 1, 0);
            gl.glClear(GL.GL_COLOR_BUFFER_BIT);
```

Μετά τον καθορισμό της θέσης της κάμερας και την πραγματοποίηση των απαραίτητων ρυθμίσεων τοποθετώ την πηγή φωτός ώστε να συμπίπτει με τη θέση της λάμπας. Πρώτα καθορίζω τη θέση της πηγής και στη συνέχεια την κατεύθυνση και την γωνία αποκοπής του φωτός, δηλαδή την γωνία μόνο μέσα στην οποία διαχέεται το φώς. Η τιμή της μεταβλητής y του πίνακα spot_direction ισούται με -1 ώστε το φώς να κατευθύνεται προς την αρνητική πλευρά του άξονα y, δηλαδή να έχει κατεύθυνση προς τα κάτω.

```
            // set spotlight position
            float spot_position[] = {0f, 8f, 9f, 0f};
            gl.glLightfv(GL.GL_LIGHT0, GL.GL_POSITION,  spot_position,0);

            //set spotlight direction
            float spot_direction[] = {0.0f,-1.0f,0.0f};
           gl.glLightfv(GL.GL_LIGHT0,GL.GL_SPOT_DIRECTION,spot_direction,0);

            //set spotlight cut-off
            float spot_angle=90.0f;
```

```
gl.glLightf(GL.GL_LIGHT0, GL.GL_SPOT_CUTOFF,(float)spot_angle);

//GL_DEPTH_TEST prevents us from seeing polygons
//that should be obscured. Remember to clear the
//GL.GL_DEPTH_BUFFER_BIT
gl.glEnable(GL.GL_DEPTH_TEST);
//Notice the depth buffer bit is also being cleared
gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
```

Μετά την ενεργοποίηση του depth test καλώ τις δύο συναρτήσεις glRotated οι οποίες περιστρέφουν το σκηνικό ως προς του άξονες y και x κάθε φορά που οι μεταβλητές angley και anglex μεταβάλλονται.

```
//rotate the scene
gl.glRotated(angley, 0, 1, 0);
gl.glRotated(anglex, 1, 0, 0);
```

Το μοναδικό αντικείμενο του σκηνικού το οποίο σχεδιάζω μέσα στην display είναι η τσαγέρα. Αυτό γίνεται λόγο της συντομίας του κώδικα ο οποίος περιορίζεται στον ορισμό της υφής της τσαγέρας και η τοποθέτησή της επάνω στο τραπέζι. Όπως συμπεραίνουμε από τον παραπάνω κώδικα η τσαγέρα έχει χρώμα γαλάζιο και η επιφάνεια της γυαλίζει λόγο της χρήσης του στοιχείου GL_SHININESS.

```
//set the color and draw a teapot
float materialpot[] = {0.2f, 0.4f, 0.6f, 0.3f};
float shine=0.25f;
gl.glMaterialfv(GL.GL_FRONT,GL.GL_AMBIENT,materialpot,0);
gl.glMaterialfv(GL.GL_FRONT,GL.GL_DIFFUSE,materialpot,0);
gl.glMaterialf(GL.GL_FRONT,GL.GL_SHININESS,shine*128.0f);

gl.glPushMatrix();
gl.glTranslated(0,2.5,9);
glut.glutSolidTeapot(0.5);
gl.glPopMatrix();
```

Οι επόμενες δύο εντολές καλούν τη συνάρτηση drawWall η οποία σχεδιάζει δυο διαδοχικούς τοίχους του δωματίου κάθε φορά που καλείται. Για αυτό το λόγο καλώ την drawWall δύο φορές προκειμένου να σχεδιαστούν και οι τέσσερις τοίχοι. Οι τιμές που περνάω ως ορίσματα βοηθούν στον υπολογισμό των τεσσάρων σημείων που ορίζουν έναν τοίχο. Θα δείξω αργότερα, στην περιγραφή της συνάρτησης drawWall, ότι ανάλογα με την τιμή της τελευταίας μεταβλητής (true/false) οι τιμές αυτών των σημείων υπολογίζονται με διαφορετικό τρόπο.

```
//draw each wall in two parts
drawWall(gl,-10,9,-1,2,true);
drawWall(gl,10,9,-1,2,false);
```

Οι συναρτήσεις που καλώ στη συνέχεια είναι εύκολο να κατανοήσει κάποιος τη λειτουργία τους. Πρώτα, για κάθε πίνακα καλώ και μια διαφορετική συνάρτηση. Στη συνέχεια, καλώ δύο φορές τη συνάρτηση drawFloorCeiling, την πρώτη φορά, με παράμετρο την τιμή true, σχεδιάζει το πάτωμα του δωματίου και τη δεύτερη φορά την οροφή. Η τιμή της δεύτερης παραμέτρου είναι αυτή που καθορίζει το y η τιμή του οποίου παραμένει αμετάβλητης όταν σχεδιάζουμε το πάτωμα ή την οροφή.

```
//draw the four paintings
drawPainting1(gl);
drawPainting2(gl);
drawPainting3(gl);
drawPainting4(gl);
drawPainting5(gl);
```

```
            //draw the floor
            drawFloorCeiling(gl,-1,true);

            //draw the ceiling
            drawFloorCeiling(gl,9,false);
```

Στο σημείο αυτό ενεργοποιώ και ετοιμάζω για χρήση την υφή woodTexture προκειμένου να σχεδιάσω την ξύλινη τραπεζαρία. Τον σχεδιασμό αναλαμβάνουν οι συναρτήσεις drawTable και drawChairs. Τέλος, σχεδιάζω το φωτιστικό στην οροφή του δωματίου.

```
            //draw the table and the four chairs,
            //first enable the tableTexture and
            //disable it when drawing finished
            woodTexture.enable();
            woodTexture.bind();
            drawTable(gl,glut);
            drawChairs(gl);
            woodTexture.disable();

            //draw the light
            drawLight(glut,gl);

}

/**
* Draw two of the four walls of the room,
* draw each wall as a separate part and
* place one part after the other so they
* look like one single wall
**/
void drawWall(GL gl,int x,int y1,int y2,int z,boolean flag){

      // set wall material
      gl.glMaterialfv(GL.GL_FRONT,GL.GL_AMBIENT,materialWhite,0);
      gl.glMaterialfv(GL.GL_FRONT,GL.GL_DIFFUSE,materialWhite,0);
      gl.glMaterialfv(GL.GL_FRONT,GL.GL_SPECULAR,materialWhite,0);
```

Τον καθορισμό της υφής του τοίχου ακολουθεί ο κώδικας ο οποίος αποθηκεύει τις συντεταγμένες της υφής που πρόκειται να εφαρμοστεί στο αντικείμενο. Αρχικά, η συνάρτηση getImageTexCoords «εξάγει» από την υφή τις συντεταγμένες της και τις αποθηκεύει στην μεταβλητή tc. Στη συνέχεια, η κάθε πλευρά της υφής αντιστοιχίζεται και σε μια μεταβλητή. Τέλος, ενεργοποιούμε την υφή και την προετοιμάζουμε για χρήση.

```
      //get the image coordinates and
      //save them to new variables
      TextureCoords tc = wallTexture.getImageTexCoords ();
      float tx1 = tc.left ();
      float ty1 = tc.top ();
      float tx2 = tc.right ();
      float ty2 = tc.bottom ();

      // Enable  the two-dimensional wallTexture
      wallTexture.enable ();

      // Bind this texture to the current rendering context
      wallTexture.bind ();
```

Η χρήση του pushMatrix προδιαθέτει το σχεδιασμό αντικειμένου. Το αντικείμενο που σχεδιάζω είναι ένα ορθογώνιο πλαίσιο που προσομοιάζει τον τοίχο του δωματίου πάνω στον οποίο θα εφαρμοστεί η ταπετσαρία. Χρησιμοποιώ μια συνθήκη for ώστε ο τοίχος μου να σχεδιαστεί σε δύο διαδοχικά κομμάτια και να φαίνεται πιο ρεαλιστική η ταπετσαρία. Οι τιμές των παραμέτρων εισόδου τις συνάρτησης glVertex3i προσδιορίζουν τις τέσσερις γωνίες κάθε τοίχου στις οποίες, με τη συνάρτηση glTexCoord2f, αντιστοιχίζονται οι

κατάλληλες συντεταγμένες τις υφής. Η τεχνική αυτή εξασφαλίζει την ορθή εφαρμογή της υφής στο αντικείμενο. Μετά την πρώτη επανάληψη το μόνο που αλλάζει στον σχεδιασμό του δεύτερου διαδοχικού τοίχου είναι η τιμή της παραμέτρου z. Θα πρέπει να αναφέρω ότι ο πρώτος τοίχος που σχεδιάζεται με κάθε κλήση της drawWall είναι ένας από τους πλαϊνούς τοίχους που αρχικά εμφανίζονται δεξιά και αριστερά του χρήστη.

```
        gl.glPushMatrix();

        //Draw the first wall as a total of two parts
        gl.glBegin(GL.GL_QUADS);
        for (int i=0;i<2;i++){
           gl.glVertex3i(x,y1, z);
           gl.glTexCoord2f (ty1, tx2);
           gl.glVertex3i(x,y2, z);
           gl.glTexCoord2f (ty2, tx2);
           gl.glVertex3i(x, y2, z+8);
           gl.glTexCoord2f (ty2, tx1);
           gl.glVertex3i(x, y1, z+8);
           gl.glTexCoord2f(ty1,tx1);

           z+=7;
        }
```

Η τιμή της flag καθορίζει αν ο τοίχος που θα σχεδιαστεί δεύτερος είναι αυτός του στην αρχή βρίσκεται ακριβώς απέναντι του ή αν είναι ο τοίχος που βρίσκεται πίσω από τον χρήστη και δεν είναι ορατός. Στην πρώτη περίπτωση, η τιμή του flag είναι true και επομένως οι τιμές των z και x δεν αλλάζουν. Στην δεύτερη περίπτωση, το flag είναι false και το z μειώνεται, ώστε να φτάσει στο σημείο που πρέπει να σχεδιαστεί ο τοίχος, και η τιμή του x γίνεται αρνητική. Με τον ίδιο τρόπο που σχεδίασα τον πρώτο τοίχο σαν μια ένωση από δύο επιμέρους κομμάτια κατασκευάζω και τον δεύτερο. Σε αυτή την περίπτωση η μεταβλητή που αλλάζει είναι η x διότι πρόκειται για τους τοίχους που οπτικά είναι παράλληλοι στον άξονα των x. Αφού γίνει και αυτό απενεργοποιώ την υφή η οποία πλέον δε μου χρειάζεται.

```
        if (flag == false){
            z=z-15;
            x=-x;
        }

        //Draw the second wall as a total of two parts
        for (int i=0;i<2;i++){
           gl.glVertex3i(x,y1, z+1);
           gl.glTexCoord2f (ty1, tx2);
           gl.glVertex3i(x,y2, z+1);
           gl.glTexCoord2f (ty2, tx2);
           gl.glVertex3i(x+10, y2, z+1);
           gl.glTexCoord2f (ty2, tx1);
           gl.glVertex3i(x+10, y1, z+1);
           gl.glTexCoord2f(ty1,tx1);

           x+=10;
        }
        gl.glEnd();

        gl.glPopMatrix();

        wallTexture.disable();
    }
```

Οι επόμενες πέντε συναρτήσεις σχεδιάζουν πέντε πίνακες ζωγραφικής στους τοίχους του δωματίου. Η τεχνική που ακολούθησα για κάθε πίνακα είναι οι ίδια: σχεδιάζω ένα

ορθογώνιο παραλληλόγραμμο επάνω σε έναν τοίχο και στη συνέχεια εφαρμόζω σε αυτό την κατάλληλη υφή στην οποία είναι αποθηκευμένη η εικόνα του πίνακα.

```
/**
* Draw the first painting
**/
void drawPainting1(GL gl){
```

Κάθε φορά που πρόκειται να εφαρμόσω υφή σε ένα αντικείμενο θέτω πρώτα το υλικό του αντικειμένου. Στη συνέχεια εξάγω τις συντεταγμένες της υφής και ενεργοποιώ την υφή μου. Τέλος, σε κάθε γωνία του ορθογωνίου που σχεδιάζω αντιστοιχίζω και μια συντεταγμένη της υφής ώστε να εμφανίζεται με τον σωστό τρόπο και απενεργοποιώ την υφή.

```
    // set the material of the first painting
    gl.glMaterialfv(GL.GL_FRONT,GL.GL_AMBIENT,materialWhite,0);
    gl.glMaterialfv(GL.GL_FRONT,GL.GL_DIFFUSE,materialWhite,0);
    gl.glMaterialfv(GL.GL_FRONT,GL.GL_SPECULAR,materialWhite,0);

    //get the image coordinates and
    //save them to new variables
    TextureCoords tc = paint1Texture.getImageTexCoords ();
    float tx1 = tc.left ();
    float ty1 = tc.top ();
    float tx2 = tc.right ();
    float ty2 = tc.bottom ();

    paint1Texture.enable ();

    paint1Texture.bind ();

    gl.glPushMatrix();

    // draw the context and apply the first painting
    gl.glBegin(GL.GL_QUADS);
    gl.glVertex3f(-6,7,16.99f);
    gl.glTexCoord2f (ty1, tx2);
    gl.glVertex3f(-6,4, 16.99f);
    gl.glTexCoord2f (ty2, tx2);
    gl.glVertex3f(-3, 4, 16.99f);
    gl.glTexCoord2f (ty2, tx1);
    gl.glVertex3f(-3, 7, 16.99f);
    gl.glTexCoord2f(ty1,tx1);
    gl.glEnd();

    gl.glPopMatrix();

    paint1Texture.disable();
}
```

Τα μοναδικά στοιχεία που αλλάζουν σε κάθε συνάρτηση που σχεδιάζει έναν διαφορετικό πίνακα είναι η υφή και οι συντεταγμένες του τοίχου πάνω στον οποίο θα σχεδιαστεί ο πίνακας. Στην init κάλεσα την συνάρτηση load πέντε φορές για τους πέντε διαφορετικούς πίνακες του δωματίου, έτσι λοιπόν σε κάθε μια από τις παρούσες συναρήσεις χρησιμοποιώ και ένα από τα διαφορετικά αυτά textures. Επίσης, οι τιμές που εισάγω στις συναρτήσεις **`glVertex3f`** όταν σχεδιάζω το ορθογώνιο που θα φιλοξενήσει τον κάθε πίνακα διαφέρουν ανάλλογα με τον τοίχο στον οποίο θέλω να εμφανίσω τον τοποθετήσω.

```
/**
* Draw the second painting
**/
void drawPainting2(GL gl){

     // the process is the same as in the drawPainting1
```

```
      gl.glMaterialfv(GL.GL_FRONT,GL.GL_AMBIENT,materialWhite,0);
      gl.glMaterialfv(GL.GL_FRONT,GL.GL_DIFFUSE,materialWhite,0);
      gl.glMaterialfv(GL.GL_FRONT,GL.GL_SPECULAR,materialWhite,0);

      TextureCoords tc = paint2Texture.getImageTexCoords ();
      float tx1 = tc.left ();
      float ty1 = tc.top ();
      float tx2 = tc.right ();
      float ty2 = tc.bottom ();

      paint2Texture.enable ();

      paint2Texture.bind ();

     gl.glPushMatrix();

     gl.glBegin(GL.GL_QUADS);
     gl.glVertex3f(-9.99f,6,12);
     gl.glTexCoord2f (ty1, tx2);
     gl.glVertex3f(-9.99f,3, 12);
     gl.glTexCoord2f (ty2, tx2);
     gl.glVertex3f(-9.99f, 3, 7);
     gl.glTexCoord2f (ty2, tx1);
     gl.glVertex3f(-9.99f, 6, 7);
     gl.glTexCoord2f(ty1,tx1);
     gl.glEnd();

     gl.glPopMatrix();

     paint2Texture.disable();
}

/**
* Draw the third painting
**/
void drawPainting3(GL gl){

     // the process is the same as in the drawPainting1
      gl.glMaterialfv(GL.GL_FRONT,GL.GL_AMBIENT,materialWhite,0);
      gl.glMaterialfv(GL.GL_FRONT,GL.GL_DIFFUSE,materialWhite,0);
      gl.glMaterialfv(GL.GL_FRONT,GL.GL_SPECULAR,materialWhite,0);

      TextureCoords tc = paint3Texture.getImageTexCoords ();
      float tx1 = tc.left ();
      float ty1 = tc.top ();
      float tx2 = tc.right ();
      float ty2 = tc.bottom ();

      paint3Texture.enable ();

      paint3Texture.bind ();

      gl.glPushMatrix();
      gl.glBegin(GL.GL_QUADS);
      gl.glVertex3f(-2,6,2.01f);
      gl.glTexCoord2f (ty1, tx2);
      gl.glVertex3f(-2,3, 2.01f);
      gl.glTexCoord2f (ty2, tx2);
      gl.glVertex3f(2, 3, 2.01f);
      gl.glTexCoord2f (ty2, tx1);
      gl.glVertex3f(2, 6, 2.01f);
      gl.glTexCoord2f(ty1,tx1);
      gl.glEnd();
      gl.glPopMatrix();

      paint3Texture.disable();
}
```

```
/**
* Draw the forth painting
**/
void drawPainting4(GL gl){

      // the process is the same as in the drawPainting1
      gl.glMaterialfv(GL.GL_FRONT,GL.GL_AMBIENT,materialWhite,0);
      gl.glMaterialfv(GL.GL_FRONT,GL.GL_DIFFUSE,materialWhite,0);
      gl.glMaterialfv(GL.GL_FRONT,GL.GL_SPECULAR,materialWhite,0);

      TextureCoords tc = paint4Texture.getImageTexCoords ();
      float tx1 = tc.left ();
      float ty1 = tc.top ();
      float tx2 = tc.right ();
      float ty2 = tc.bottom ();

      paint4Texture.enable ();

      paint4Texture.bind ();

      gl.glPushMatrix();

      gl.glBegin(GL.GL_QUADS);
      gl.glVertex3f(6,7,16.99f);
      gl.glTexCoord2f (ty1, tx2);
      gl.glVertex3f(6,4, 16.99f);
      gl.glTexCoord2f (ty2, tx2);
      gl.glVertex3f(2, 4, 16.99f);
      gl.glTexCoord2f (ty2, tx1);
      gl.glVertex3f(2, 7, 16.99f);
      gl.glTexCoord2f(ty1,tx1);
      gl.glEnd();

      gl.glPopMatrix();

      paint4Texture.disable();
}

/**
* Draw the fifth painting
**/
void drawPainting5(GL gl){

     // the process is the same as in the drawPainting1
      gl.glMaterialfv(GL.GL_FRONT,GL.GL_AMBIENT,materialWhite,0);
      gl.glMaterialfv(GL.GL_FRONT,GL.GL_DIFFUSE,materialWhite,0);
      gl.glMaterialfv(GL.GL_FRONT,GL.GL_SPECULAR,materialWhite,0);

      TextureCoords tc = paint5Texture.getImageTexCoords ();
      float tx1 = tc.left ();
      float ty1 = tc.top ();
      float tx2 = tc.right ();
      float ty2 = tc.bottom ();

      paint5Texture.enable ();

      paint5Texture.bind ();

      gl.glPushMatrix();

      gl.glBegin(GL.GL_QUADS);
      gl.glVertex3f(9.99f,6,12);
      gl.glTexCoord2f (ty1, tx2);
      gl.glVertex3f(9.99f,3, 12);
      gl.glTexCoord2f (ty2, tx2);
      gl.glVertex3f(9.99f, 3, 7);
      gl.glTexCoord2f (ty2, tx1);
      gl.glVertex3f(9.99f, 6, 7);
```

```
      gl.glTexCoord2f(ty1,tx1);
      gl.glEnd();

      gl.glPopMatrix();

      paint5Texture.disable();
}
```

Η συνάρτηση που ακολουθεί σχεδιάζει το πάτωμα και την οροφή του δωματίου. Τα δύο αυτά αντικείμενα του χώρου είναι ουσιαστικά δύο ορθογώνια παραλληλόγραμμα καθένα από τα οποία βρίσκεται σε διαφορετικό ύψος. Το πάτωμα είναι ένα μεγάλο ορθογώνιο που τοποθετείται χαμηλά στον τρισδιάστατο χώρο, ενώ η οροφή είναι ένα ορθογώνιο που βρίσκεται ψηλά στον χώρο και η συντεταγμένη του y έχει την μεγαλύτερη τιμή. Η συνάρτηση drawFloorCeiling ανάλογα με την τιμή του flag σχεδιάζει το πάτωμα ή την οροφή με τον τρόπο που αναλύω παρακάτω.

```
/**
* Draw the floor and ceiling of the room,
* floor generated when flag is true and
* ceiling generated wnen flag is false
**/
void drawFloorCeiling(GL gl,int k,boolean flag){

      // set the material of the floor/ceiling
      float shine=0.25f;
      gl.glMaterialfv(GL.GL_FRONT,GL.GL_AMBIENT,materialWhite,0);
      gl.glMaterialfv(GL.GL_FRONT,GL.GL_DIFFUSE,materialWhite,0);
      gl.glMaterialfv(GL.GL_FRONT,GL.GL_SPECULAR,materialWhite,0);
      gl.glMaterialf(GL.GL_FRONT,GL.GL_SHININESS,shine*128.0f);

      //get the image coordinates and
      //save them to new variables
      TextureCoords tc = floorTexture.getImageTexCoords ();
      float tx1 = tc.left ();
      float ty1 = tc.top ();
      float tx2 = tc.right ();
      float ty2 = tc.bottom ();
```

Μετά από τη γνωστή πλέον διαδικασία που ακολουθούμε πριν από το σχεδιασμό κάθε αντικειμένου με υφή υπάρχει η συνθήκη η οποία ορίζει αν πρόκειται να σχεδιαστεί η οροφή ή το πάτωμα του δωματίου. Όταν η τιμή του flag είναι true τότε ενεργοποιώ την υφή του πατώματος (floorTexture) και άρα το πάτωμα πρόκειται να σχεδιαστεί, σε διαφορετική περίπτωση ενεργοποιώ την υφή της οροφής και γίνεται αντιληπτό ότι η συνάρτηση θα σχεδιάσει την οροφή.

```
      // choses whether to draw floor or ceiling
      if (flag == true){
         // enable texture for the floor
         floorTexture.enable ();

         // Bind this texture to the current rendering context.
         floorTexture.bind ();
      } else {
         //enable texture for thn ceiling
         ceilingTexture.enable ();

         // Bind this texture to the current rendering context.
         ceilingTexture.bind ();
      }

      gl.glPushMatrix();
```

Οι επαναλήψεις που ακολουθούν εξασφαλίζουν, κυρίως, την ρεαλιστική απεικόνιση του πατώματος του δωματίου. Οι δύο συνθήκες for (η μια εμφωλευμένη) έχουν σαν αποτέλεσμα το πάτωμα και η οροφή να σχεδιάζονται σαν ένα σύνολο από 300 τετράγωνα κομμάτια. Στην περίπτωση του πατώματος η τεχνική αυτή δημιουργεί το εφέ ενός ξύλινου παρκέ φτιαγμένου από μικρά κομμάτια ξύλου τοποθετημένα κάθετα το ένα με το άλλο. Έτσι το πάτωμα του δωματίου φαίνεται πιο εντυπωσιακό και το texture πλέον εφαρμόζεται σε κάθε κομμάτι του πατώματος ξεχωριστά. Στην περίπτωση της οροφής η τεχνική αυτή δεν έχει ιδιαίτερη χρησιμότητα καθώς το αντίστοιχο texture είναι μονόχρωμο με αποτέλεσμα να μην ξεχωρίσουν τα κομμάτια από τα οποία αποτελείται.

```
      // draw 300 floor tails and
      // apply the floorTexture in each tile
      // or
      // draw the ceiling in 300 separate pieces and
      // apply the ceilingTexture in each piece
      for (int j=2;j<17;j++){
         for(int i=-10;i<10;i++){
            gl.glBegin(GL.GL_QUADS);
            gl.glVertex3i(i, k, j+1);
            gl.glTexCoord2f (ty1, tx2);
            gl.glVertex3i(i,k, j);
            gl.glTexCoord2f (ty2, tx2);
            gl.glVertex3i(i+1,k , j);
            gl.glTexCoord2f (ty2, tx1);
            gl.glVertex3i(i+1, k, j+1);
            gl.glTexCoord2f(ty1,tx1);
            gl.glEnd();
         }
      }

      gl.glPopMatrix();
```

Αφού ολοκληρωθεί η διαδικασία σχεδιασμού του πατώματος / οροφής απενεργοποιώ το αντίστοιχο texture και πάλι ελέγχοντας την τιμή του flag.

```
      // disable the proper texture
      if(flag == true)
            floorTexture.disable();
      else
          ceilingTexture.disable();
}
```

Η συνάρτηση drawTable σχεδιάζει ένα ξύλινο τραπέζι στο κέντρο του δωματίου. Για να δημιουργήσω το τραπέζι καλώ την λίστα WoodList, η οποία σχεδιάζει έναν ξύλινο κύβο, και με τη βοήθεια της εντολής glScaled παραμορφώνω τον κύβο ώστε να μοιάζει με το επάνω μέρος του τραπεζιού. Τέλος, τα πόδια του τραπεζιού αποτελούν τέσσερις κύλινδροι σε χρώμα πρασινωπό. Τα βήματα αυτά περιγράφονται αναλυτικότερα παρακάτω.

```
/**
* Draw the table
* consisting of a wooden table-top and
* four legs
**/
void drawTable(GL gl, GLUT glut){
```

Πρώτα σχεδιάζω το επάνω μέρος του τραπεζιού ως εξής: με την εντολή glPushMatrix() μηδενίζω τον πίνακα glTranslated και αμέσως μετά τοποθετώ το αντικείμενο στην θέση που επιθυμώ. Με την εντολή glScaled(3, 0.1, 1.8) παραμορφώνω τον κύβο που πρόκειται να σχεδιάσω ώστε αυτός να αυξήσει τις διαστάσεις του στους άξονες των x και z και να μειωθεί

κατά πολύ στη διάσταση του y, έτσι ώστε πλέον να μοιάζει με ένα ορθογώνιο κομμάτι ξύλου. Τέλος, καλώ τη λίστα woodenCube η οποία σχεδιάζει τον ξύλινο κύβο.

```
    // draw the table-top
    // as a scaled wooden cube
    // using the wood list
    gl.glPushMatrix();
    gl.glTranslated(0, 2, 9);
    gl.glScaled(3, 0.1, 1.8);
    gl.glCallList(woodenCube);
    gl.glPopMatrix();
```

Μετά το επάνω μέρος του τραπεζιού ήρθε η ώρα να σχεδιάσω τα πόδια του. Πρώτα ορίζω το υλικό τους ώστε να είναι πράσινο, με μια ελαφριά γυαλάδα, και προχωρώ στον σχεδιασμό τους. Κάθε πόδι τοποθετείται στις τέσσερις γωνίες του τραπεζίου με τη συνάρτηση glTranslated και σχεδιάζεται με τη χρήση ενός έτοιμου αντικειμένου της κλάσης GLU, τον κύλινδρο (glutSolidCylinder). Προτού χρησιμοποιήσω το αντικείμενο αυτό το περιστρέφω κατά 90^{o} έτσι ώστε τα πόδια - κύλινδροι να είναι κάθετα στο πάτωμα.

```
    // set the material of the four table-legs
    float shine=0.25f;
    gl.glMaterialfv(GL.GL_FRONT,GL.GL_AMBIENT,materialGreen,0);
    gl.glMaterialfv(GL.GL_FRONT,GL.GL_DIFFUSE,materialGreen,0);
    gl.glMaterialfv(GL.GL_FRONT,GL.GL_SPECULAR,materialGreen,0);
    gl.glMaterialf(GL.GL_FRONT,GL.GL_SHININESS,shine*128.0f);

    // draw a cylinder as the first table-leg
    gl.glPushMatrix();
    gl.glTranslated(2,2,8);
    gl.glRotated(90, 1, 0, 0);
    glut.glutSolidCylinder(0.2, 3, 8, 8);

    // draw the second table-leg
    gl.glTranslated(-4,0,0);
     glut.glutSolidCylinder(0.2, 3, 8, 8);
    gl.glPopMatrix();

    // the third table-leg
    gl.glPushMatrix();
    gl.glTranslated(-2,2,10);
    gl.glRotated(90, 1, 0, 0);
     glut.glutSolidCylinder(0.2, 3, 8, 8);

    // the last table leg
    gl.glTranslated(4,0,0);
    glut.glutSolidCylinder(0.2, 3, 8, 8);
    gl.glPopMatrix();
    gl.glPopMatrix();

}
```

Η επόμενη συνάρτηση σχεδιάζει τέσσερις καρέκλες γύρο από το τραπέζι και είναι ιδιαίτερα απλοϊκή στη μορφή της. Για κάθε καρέκλα πρώτα καθορίζω τη θέση στην οποία θα τοποθετηθεί και στη συνέχεια καλώ τη λίστα chair η οποία αναλαμβάνει εξολοκλήρου το σχεδιασμό της καρέκλας. Τη διαδικασία αυτή την πραγματοποιώ τέσσερις φορές, μια για κάθε καρέκλα.

```
/**
* Draw the four chairs of the table
**/
void drawChairs(GL gl){

    // call the chair list four times,
```

```
    // draw the same chair in four different places

    gl.glPushMatrix();//right
    gl.glTranslated(-4, 1, 9);
    gl.glCallList(chair);
    gl.glPopMatrix();
```

Ενώ στην πρώτη καρέκλα δεν χρειάστηκε να χρησιμοποιήσω τη συνάρτηση glRotated, αφού η φορά της είναι σύμφωνη με τη φορά με την οποία σχεδιάστηκε εξαρχής από τη λίστα, δεν συμβαίνει το ίδιο και με τις επόμενες καρέκλες. Πριν να σχεδιάσω καθεμία από τις άλλες τρις καρέκλες θα πρέπει να τις περιστρέψω, με τη συνάρτηση glRotated, ώστε αυτές να είναι στραμμένες προς το τραπέζι και νη μην έχουν την αρχική φορά με την οποία σχεδιάζονται στη λίστα.

```
    gl.glPushMatrix();//left
    gl.glTranslated(4, 1, 9);
    // rotate the chair in order to look towards the table
    gl.glRotated(180, 0, 1, 0);
    gl.glCallList(chair);
    gl.glPopMatrix();

    gl.glPushMatrix();//back
    gl.glTranslated(0, 1, 13);
    // rotate the chair in order to look towards the table
    gl.glRotated(90, 0, 1, 0);
    gl.glCallList(chair);
    gl.glPopMatrix();

    gl.glPushMatrix();//front
    gl.glTranslated(0, 1, 5);
    // rotate the chair in order to look towards the table
    gl.glRotated(270, 0, 1, 0);
    gl.glCallList(chair);
    gl.glPopMatrix();
}
```

Η επόμενη συνάρτηση σχεδιάζει το φωτιστικό που υπάρχει την οροφή του δωματίου. Πρόκειται για μια σφαιρική λάμπα η οποία έχει σαν βάση ένα ορθογώνιο κουτί. Γίνεται εύκολα κατανοητό ότι η λάμπα σχεδιάζεται σαν μια σφαίρα και η βάση της λάμπας σαν ένας κύβος στον οποίο έχω εφαρμόσει παραμόρφωση ώστε να μοιάζει με ορθογώνιο.

```
/**
* Draw the light
**/
void drawLight(GLUT glut,GL gl){
```

Προτού σχεδιάσω τη λάμπα θα πρέπει να απενεργοποιήσω τον φωτισμό διότι η λάμπα αποτελεί την πηγή φωτός και άρα δεν επηρεάζεται από τον φωτισμό που υπάρχει στο δωμάτιο. Στη συνέχεια, καθορίζω τη θέση της λάμπας, η οποία συμπίπτει με τη θέση του φωτός GL_LIGHT0 όπως είδαμε παραπάνω, και ορίζω το χρώμα τις να είναι λευκό γυαλιστερό. Αφού σχεδιάσω τη λάμπα ενεργοποιώ ξανά τον φωτισμό ώστε να συνεχίσω με τη βάση της. Τα έτοιμα αντικείμενα της κλάσης GLUT που χρησιμοποίησα είναι τα glutSolidSphere για το σχεδιασμό της σφαίρας και glutSolidCube για το σχεδιασμό της βάσης.

```
    // draw the bulb as a white shpere,
    // first disable lighting because
    // the bulb emitis and not receive light
    gl.glDisable(GL.GL_LIGHTING);
    gl.glPushMatrix();
    gl.glTranslated(0, 8.65, 9);
    float shine=0.25f;
```

```
    gl.glColor3f(1, 1, 1);
    gl.glMaterialf(GL.GL_FRONT,GL.GL_SHININESS,shine*128.0f);
    glut.glutSolidSphere(0.25f, 100, 300);
    // enable lighting again to draw the rest of the light
    gl.glEnable(GL.GL_LIGHTING);

    // draw a black cube as the base of the bulb
    gl.glTranslated(0, 0.35, 0);
    gl.glScaled(1, 0.5, 1);
    gl.glMaterialfv(GL.GL_FRONT,GL.GL_AMBIENT,materialBlack,0);
    gl.glMaterialfv(GL.GL_FRONT,GL.GL_DIFFUSE,materialBlack,0);
    gl.glMaterialfv(GL.GL_FRONT,GL.GL_SPECULAR,materialBlack,0);
    glut.glutSolidCube(0.5f);

    gl.glPopMatrix();

}
```

Ακολουθούν οι συναρτήσεις στις οποίες δημιουργώ τις δύο λίστες που χρησιμοποιώ στο πρόγραμμά μου. Η πρώτη συνάρτηση είναι η WoodList, η οποία δημιουργεί μια λίστα που σχεδιάζει έναν ξύλινο κύβο. Χρησιμοποιώ τους κύβους αυτούς στην κατασκευή των ξύλινων μερών της τραπεζαρίας.

```
/**
* Draw a wooden cube that is
* used to make the table-top and a part of each chair
**/
void WoodList(GL gl,GLUT glut){
```

Πριν αναπτύξω τον κώδικα που σχεδιάζει τον ξύλινο κύβο δημιουργώ τη λίστα και το αναγνωριστικό της με την ονομασία woodenCube. Στη συνέχεια, ορίζω το υλικό του κύβου και εξάγω τις συντεταγμένες της υφής. Τονίζω ότι στη συγκεκριμένη συνάρτηση δεν ενεργοποιώ την υφή που θα εφαρμόσω καθώς αυτή έχει ενεργοποιηθεί ήδη στην μέθοδο display πριν από την κλήση των συναρτήσεων drawTable και drawChairs. Οι συναρτήσεις αυτές είναι και οι μόνες που χρησιμοποιουν την υφή woodTexture.

```
    // generate the list named woodenCube
    woodenCube = gl.glGenLists(1);
    gl.glNewList(woodenCube,GL.GL_COMPILE);

    // set rthe material of the cube
    gl.glMaterialfv(GL.GL_FRONT,GL.GL_AMBIENT,materialWhite,0);
    gl.glMaterialfv(GL.GL_FRONT,GL.GL_DIFFUSE,materialWhite,0);
    gl.glMaterialfv(GL.GL_FRONT,GL.GL_SPECULAR,materialWhite,0);

    //get the image coordinates and
    //save them to new variables
    TextureCoords tc = woodTexture.getImageTexCoords ();
    float tx1 = tc.left ();
    float ty1 = tc.top ();
    float tx2 = tc.right ();
    float ty2 = tc.bottom ();
```

Οι παρακάτω εντολές σχεδιάζουν έναν κύβο και εφαρμόζουν σε αυτόν μια υφή. Το τμήμα αυτό του κώδικα υπήρχε έτοιμο στη σελίδα NeHe Productions, στις έτοιμες εφαρμογές της ενότητας OpenGL Tutorials. Πρόκειται για έναν τυποποιημένο τρόπο εφαρμογής texture σε κύβο και μπορεί να χρησιμοποιηθεί σε πολλά προγράμματα της γλώσσας JOGL. Ο κώδικας δημιουργεί τέσσερα τετράγωνα τα οποία αποτελούν τις τεσσερις πλευρές του κύβου. [] Σε κάθε πλευρά του εφαρμόζει την υφή αντιστοιχίζοντας την κάθε γωνία της τετράγωνης πλευράς στις αντίστοιχες συντεταγμένες της υφής.

```
    gl.glBegin(GL.GL_QUADS); // Start Drawing Quads
```

```
     // Top Face
gl.glVertex3f(-1.0f,  1.0f, -1.0f); // Top Left Of The Texture and Quad
        gl.glTexCoord2f(ty1, tx2);
gl.glVertex3f(-1.0f,  1.0f,  1.0f); // Bottom Left Of The Texture and Quad
       gl.glTexCoord2f(ty2, tx2);
gl.glVertex3f( 1.0f,  1.0f,  1.0f); // Bottom Right Of The Texture and Quad
        gl.glTexCoord2f(ty2, tx1);
gl.glVertex3f( 1.0f,  1.0f, -1.0f); // Top Right Of The Texture and Quad
        gl.glTexCoord2f(ty1, tx1);
     // Bottom Face
gl.glVertex3f(-1.0f, -1.0f, -1.0f); // Top Right Of The Texture and Quad
        gl.glTexCoord2f(ty1, tx1);
gl.glVertex3f( 1.0f, -1.0f, -1.0f); // Top Left Of The Texture and Quad
        gl.glTexCoord2f(ty1, tx2);
gl.glVertex3f( 1.0f, -1.0f,  1.0f); // Bottom Left Of The Texture and Quad
        gl.glTexCoord2f(ty2, tx2);
gl.glVertex3f(-1.0f, -1.0f,  1.0f); // Bottom Right Of The Texture and Quad
        gl.glTexCoord2f(ty2, tx1);
    // Front Face
gl.glVertex3f(-1.0f, -1.0f,  1.0f); // Bottom Left Of The Texture and Quad
        gl.glTexCoord2f(ty2, tx2);
gl.glVertex3f( 1.0f, -1.0f,  1.0f); // Bottom Right Of The Texture and Quad
        gl.glTexCoord2f(ty2, tx1);
gl.glVertex3f( 1.0f,  1.0f,  1.0f); // Top Right Of The Texture and Quad
        gl.glTexCoord2f(ty1, tx1);
gl.glVertex3f(-1.0f,  1.0f,  1.0f); // Top Left Of The Texture and Quad
        gl.glTexCoord2f(ty1, tx2);
    // Back Face
gl.glVertex3f(-1.0f, -1.0f, -1.0f); // Bottom Right Of The Texture and Quad
        gl.glTexCoord2f(ty2, tx1);
gl.glVertex3f(-1.0f,  1.0f, -1.0f); // Top Right Of The Texture and Quad
        gl.glTexCoord2f(ty1, tx1);
gl.glVertex3f( 1.0f,  1.0f, -1.0f); // Top Left Of The Texture and Quad
        gl.glTexCoord2f(ty1, tx2);
gl.glVertex3f( 1.0f, -1.0f, -1.0f); // Bottom Left Of The Texture and Quad
        gl.glTexCoord2f(ty2, tx2);
    // Right face
gl.glVertex3f( 1.0f, -1.0f, -1.0f); // Bottom Right Of The Texture and Quad
        gl.glTexCoord2f(ty2, tx1);
gl.glVertex3f( 1.0f,  1.0f, -1.0f); // Top Right Of The Texture and Quad
        gl.glTexCoord2f(ty1, tx1);
gl.glVertex3f( 1.0f,  1.0f,  1.0f); // Top Left Of The Texture and Quad
        gl.glTexCoord2f(ty1, tx2);
gl.glVertex3f( 1.0f, -1.0f,  1.0f); // Bottom Left Of The Texture and Quad
        gl.glTexCoord2f(ty2, tx2);
    // Left Face
gl.glVertex3f(-1.0f, -1.0f, -1.0f); // Bottom Left Of The Texture and Quad
        gl.glTexCoord2f(ty2, tx2);
gl.glVertex3f(-1.0f, -1.0f,  1.0f); // Bottom Right Of The Texture and Quad
        gl.glTexCoord2f(ty2, tx1);
gl.glVertex3f(-1.0f,  1.0f,  1.0f); // Top Right Of The Texture and Quad
        gl.glTexCoord2f(ty1, tx1);
gl.glVertex3f(-1.0f,  1.0f, -1.0f); // Top Left Of The Texture and Quad
        gl.glTexCoord2f(ty1, tx2);
    gl.glEnd(); // Done Drawing Quads
```

Μετά από το τέλος του κώδικα που σχεδιάζει τον ξύλινο κύβο τερματίζω και τη λίστα μου.

```
    // end code included in the list
    gl.glEndList();
}
```

Η δεύτερη συνάρτηση, η οποία περιλαμβάνει τη δημιουργία λίστας, είναι η buildChairList. Όπως δηλώνει και το όνομα της, η λίστα αυτή είναι υπεύθυνη για το σχεδιασμό μιας καρέκλας με όλα τα επιμέρους κομμάτια που αυτή περιλαμβάνει. Τα ξύλινα μέρη της

καρέκλας είναι η θέση και το πίσω μέρος, δηλαδή η πλάτη. Για να σχεδιάσω τα μέρη αυτά καλώ τη λίστα woodenCube, που περιέγραψα παραπάνω, και με τη βοήθεια της συνάρτησης glScaled προσαρμόζω τις διαστάσεις του ξύλινου κύβου στο σχήμα που θέλω να πετύχω.

```
/**
* Draw a chair consisting of
* two wooden parts and four legs
**/
void buildChairList(GL gl,GLUT glut){

    // generate the list named chair
    chair = gl.glGenLists(1);
    gl.glNewList(chair,GL.GL_COMPILE);

    // draw the wooden seat of the chair
    gl.glScaled(0.8, 0.1, 0.8);
    gl.glCallList(woodenCube);

    // draw the wooden back of the chair
    gl.glTranslated(-0.65, 15,0);
    gl.glScaled(0.1, 8, 1);
    gl.glCallList(woodenCube);
```

Αφού λοιπόν σχεδιάσω τα δύο αυτά κάθετα ορθογώνια κομμάτια ξύλου τοποθετώ τα πόδια της καρέκλας. Για το σχεδιασμό των ποδιών χρησιμοποιώ το έτοιμο αντικείμενο της κλάσης GLUT, glutSolidCylinder, τέσσερις φορές αφού το περιστρέψω κατά 90º ώστε ο κύλινδρος να είναι κάθετος στο πάτωμα. Η υφή που έχω επιλέξει έχει την ίδια απόχρωση με τα πόδια του τραπεζιού. Τέλος, θα πρέπει να αναφέρω ότι οι δύο πρώτοι κύλινδροι έχουν μεγαλύτερο ύψος ώστε να στηρίζουν την πλάτη της καρέκλας.

```
    // set the material of the four legs
    float shine=0.25f;
    gl.glMaterialfv(GL.GL_FRONT,GL.GL_AMBIENT,materialGreen,0);
    gl.glMaterialfv(GL.GL_FRONT,GL.GL_DIFFUSE,materialGreen,0);
    gl.glMaterialfv(GL.GL_FRONT,GL.GL_SPECULAR,materialGreen,0);
    gl.glMaterialf(GL.GL_FRONT,GL.GL_SHININESS,shine*128.0f);

    // draw each leg of the chair as a cylinder
    gl.glTranslated(-2,0,0.6);
    gl.glScaled(7, 1, 0.5);
    gl.glRotated(90, 1, 0, 0);
    glut.glutSolidCylinder(0.2, 4, 8, 8);

    gl.glTranslated(0,-2.5,0);
    glut.glutSolidCylinder(0.2, 4, 8, 8);

    gl.glTranslated(2,0,2);
    glut.glutSolidCylinder(0.2, 2, 8, 8);

    gl.glTranslated(0,2.5,0);
    glut.glutSolidCylinder(0.2, 2, 8, 8);

    // end code included in the list
    gl.glEndList();
}
```

Οι δύο επόμενες συναρτήσεις δεν χρησιμεύουν στο πρόγραμμα μου και για αυτό το λόγο έχω αφήσει κενό το περιεχόμενό τους.

```
/**
* Called when the GLDrawable (GLCanvas
* or GLJPanel) has changed in size.
* We don't need this
 **/
```

```
public void reshape(GLAutoDrawable drawable, int x, int y, int width, int height)
{}

/**
* If the display depth is changed while the
* program is running this method is called.
* We don't need this
**/
public void displayChanged(GLAutoDrawable glad, boolean bln, boolean bln1) {}
```

Η συνάρτηση load είναι αυτή που καλώ πρώτη στη μέθοδο init. Πρόκειται για ακόμη μια τυποποιημένη συνάρτηση της JOGL την οποία χρησιμοποίησα και στις προηγούμενες εφαρμογές που ανέπτυξα, κάθε φορά που χρησιμοποιούσα textures. Η συνάρτηση αυτή καλείται τόσες φορές όσα είναι και τα διαφορετικά textures που χρησιμοποιώ στο πρόγραμμα. Η μοναδική παράμετρος που δέχεται ως είσοδο η συνάρτηση είναι το όνομα του αρχείου που περιέχει την εικόνα. Αφού φορτώσει την εικόνα του αρχείου σε ένα αντικείμενο τύπου texture και πραγματοποιήσει τις απαραίτητες ρυθμίσεις επιστρέφει το texture ώστε αυτό να αποθηκευθεί σε μια μεταβλητή για μελλοντική χρήση, όπως ακριβώς πραγματοποίησα στην init.

```
/**
 * Load an image from a file and retunn it as a texture
 **/
Texture load(String filename){
    Texture texture = null;

      try
      {
          // Create an OpenGL texture from the specified file. Do not create
          // mipmaps
          texture = TextureIO.newTexture (new File (filename), false);

          // Use the NEAREST magnification function when the pixel being
          // textured maps to an area less than or equal to one texture
          // element (texel)
          texture.setTexParameteri(GL.GL_TEXTURE_MAG_FILTER, GL.GL_NEAREST);

          // Use the NEAREST minification function when the pixel being
          // textured maps to an area greater than one texel
          texture.setTexParameteri(GL.GL_TEXTURE_MIN_FILTER, GL.GL_NEAREST);
      }
      catch (Exception e)
      {
          System.out.println ("error loading texture from "+filename);
      }

      return texture;
}
```

Η τελευταία μέθοδος του προγράμματος Room ανιχνεύει οποιοδήποτε πάτημα του πληκτρολογίου. Ο χρήστης έχει τη δυνατότητα να μετακινηθεί μέσα στο δωμάτιο με διάφορους τρόπους ανάλογα με το πλήκτρο που θα πατήσει. Πιο αναλυτικά όταν πατάει το πλήκτρο με τον αριθμό 1 μετακινείται ελάχιστα προς τα αριστερά και όταν πατάει το πλήκτρο 2 μετακινείται προς τα δεξιά. Με το πλήκτρο 3 μετακινείται προς τα πίσω και με το πλήκτρο 4 προς τα εμπρός. Τα τέσσερα επόμενα πλήκτρα περιστρέφουν την οπτική γωνία του χρήστη με τον εξής τρόπο: όταν πατάει το πλήκτρο με τον αριθμό 5 ο χρήστης περιστρέφεται προς τα αριστερά και όταν πατάει το 6 περιστρέφεται προς τα δεξιά. Τέλος, με το πλήκτρο 7 στρέφεται προς τα επάνω ενώ με το πάτημα του 8 προς τα κάτω.

```
/**
 * Change the viewers aspect and rotate the scene
 * when the user types a number
```

```
 **/
public void keyTyped(KeyEvent e) {
      // the first four keys move the viewer into the x and z axis
      if (e.getKeyChar() == KeyEvent.VK_1)
          xPosition += 0.1;
      else if (e.getKeyChar() == KeyEvent.VK_2)
          xPosition -=0.1;
      else if (e.getKeyChar() == KeyEvent.VK_3)
          zPosition -=0.1;
      else if(e.getKeyChar() == KeyEvent.VK_4)
          zPosition +=0.1;
      // the last four keys rotate the scene around the y and x axis
      else if(e.getKeyChar() == KeyEvent.VK_5)
          angley -=1;
      else if(e.getKeyChar() == KeyEvent.VK_6)
          angley +=1;
      else if(e.getKeyChar() == KeyEvent.VK_7)
          anglex +=1;
      else if(e.getKeyChar() == KeyEvent.VK_8)
          anglex -=1;
```

Όσον αφορά το προγραμματιστικό κομμάτι, για την ανίχνευση των πλήκτρων χρησιμοποίησα τη συνάρτηση getKeyChar() η οποία εντοπίζει την τιμή του πλήκτρου που έχει πατηθεί. Στη συνέχεια ελέγχω με τις κατάλληλες συνθήκες με ποιο από τα πλήκτρα (αριθμοί 1-8) ταυτίζεται και όταν βρεθεί το πλήκτρο αυτό πραγματοποιώ τις αντίστοιχες αλλαγές.

- Στις δύο πρώτες συνθήκες μειώνω και αυξάνω την τιμή του xPosition έτσι ώστε η θέση του χρήστη να αλλάζει στον άξονα των x και αυτός να κινείται αριστερά και δεξιά.
- Στις επόμενες δύο συνθήκες το ίδιο κάνω με τις τιμές του zPosition έτσι ώστε ο χρήστης να μπορεί να μετακινηθεί στον άξονα των z. Όταν το z μειώνεται ο χρήστης μετακινείται προς τα πίσω, ενώ όταν το z αυξάνεται μετακινείται προς τα μπροστά.
- Στις συνθήκες που αφορούν τα πλήκτρα 5 και 6 μεταβάλλεται η τιμή της μεταβλητής angley. Η μεταβλητή αυτή αποτελεί παράμετρο της συνάρτησης glRotated και έχει σαν αποτέλεσμα την περιστροφή του σκηνικού ως προς τον άξονα του y. Έτσι, όταν το angley μειώνεται το σκηνικό, και κατά επέκταση ο χρήστης, περιστρέφεται προς τα αριστερά, ενώ όταν η τιμή του angley αυξάνεται περιστρέφεται προς τα δεξιά.
- Τέλος, με τον ίδιο τρόπο ο χρήστης μπορεί να περιστραφεί και ως προς τον άξονα του x μεταβάλλοντας την τιμή της μεταβλητής anglex. Όταν η μεταβλητή αυτή αυξάνεται ο χρήστης περιστρέφεται προς τα επάνω, ενώ όταν η μεταβλητή anglex μειώνεται ο χρήστης περιστρέφεται προς τα κάτω.

Όταν πραγματοποιηθούν οι αλλαγές που επιφέρει το πάτημα ενός πλήκτρου σε μια από τις παραπάνω μεταβλητές καλώ την repaint. Αυτό έχει σαν αποτέλεσμα το σκηνικό μας να σχεδιαστεί από την αρχή με τον χρήστη να βρίσκεται σε διαφορετική θέση, ανάλογα με τη νέα τιμή της μεταβλητής που άλλαξε.

```
      // repaint the GLCnvas so that the changes affect the scene
      glc.repaint();
    }

// We don't need the last two methods of KeyListener
public void keyPressed(KeyEvent e) {}

public void keyReleased(KeyEvent e) {}
```

```
}
```

Το αποτέλεσμα του προγράμματος μου φαίνεται στην εικόνα 25. Στο στιγμιότυπο που απεικονίζεται, ο χρήστης έχει ήδη μετακινηθεί μέσα στον χώρο έτσι ώστε να μπορεί να βλέπει την τραπεζαρία διαγώνια, όπως φαίνεται στην εικόνα.

Η ποιότητα του αποτελέσματος που πέτυχα με την χρήστη της Java OpenGL για την ανάπτυξη του προγράμματός μου είναι αρκετά ικανοποιητική, σε σημείο που να φαίνονται όλες οι λεπτομέρειες του σκηνικού. Το χαρακτηριστικό αυτό καθιστά την JOGL κατάλληλη για την ανάπτυξη εφαρμογών υψηλής ποιότητας και ευκρίνειας που μας επιτρέπουν να προσομοιάσουμε σκηνικά που οπτικά δεν θα απέχουν πολύ από την πραγματικότητα.

Εικόνα 25: Μια άποψη του δωματίου που σχεδιάζει το πρόγραμμα Room

3. Αδυναμίες και Περιορισμοί

Στην ενότητα αυτή αναφέρω ορισμένες αδυναμίες του προγράμματος μου που σύμφωνα με την εμπειρία μου περιορίζουν το οπτικό αποτέλεσμα της εφαρμογής. Οι αδυναμίες αυτές αφορούν τόσο την συμπεριφορά ορισμένων στοιχείων της γλώσσας Java OpenGL όσο και αδυναμίες του κώδικα. Ακολουθεί σύντομη αναφορά στους περιορισμούς αυτούς.

Αρχικά, με την έναρξη της εκτέλεσης του προγράμματος Room η πρώτη εικόνα που εμφανίζεται στον χρήστη είναι εμφανώς πιο φωτισμένη σε σχέση με αυτή που αντικρίζει στη συνέχεια. Το πρόβλημα διορθώνεται όταν το σκηνικό μας σχεδιάζεται για δεύτερη φορά και ο φωτισμός πλέον παίρνει την σωστή τιμή που έχω καθορίσει στο πρόγραμμα. Για να αποφευχθεί ο έντονος αυτός φωτισμός τοποθέτησα στο τέλος της μεθόδου init την εντολή glc.repaint(). Με τον τρόπο αυτό αφού έχει ενεργοποιηθεί ο φωτισμός και έχουν τεθεί η

τιμές του σχεδιάζεται ξανά το σκηνικό ώστε αυτή τη φορά να έχει το σωστό φωτισμό. Η αδυναμία αυτή γίνεται αντιληπτή από τον χρήστη μόνο με την έναρξή του προγράμματος και μια ένα δευτερόλεπτο πριν εκτελεστεί η εντολή repaint.

Ο επόμενος περιορισμός αφορά και αυτός τον φωτισμό του χώρου. Είναι εμφανές ότι ενώ τα υπόλοιπα αντικείμενα του δωματίου φωτίζονται κανονικά σύμφωνα με την πηγή φωτός, που είναι η λάμπα, το επάνω μέρος του τραπεζιού φαίνεται πιο σκούρο και λιγότερο φωτισμένο σε σχέση με τα υπόλοιπα αντικείμενα. Ενώ η τσαγέρα που είναι τοποθετημένη απάνω στο τραπέζι αντανακλά τον σωστό τρόπο το φώς της λάμπας, η ίδια η επιφάνεια του τραπεζιού φαίνεται να φωτίζεται ελάχιστα σε σχέση με τον περιβάλλοντα χώρο.

Τον τελευταίο περιορισμό αποτελεί το γεγονός ότι ο χρήστης είναι ελεύθερος να περιηγηθεί και έξω από τα όρια του δωματίου όπου υπάρχει μόνο κενός χώρος. Αυτό έχει ως αποτέλεσμα ο χρήστης συχνά να αντικρίζει ένα κενό μαύρο διάστημα, όταν διαπερνά τα όρια των τοίχων του δωματίου, χωρίς να υπάρχει τρόπος να παραμείνει μόνο μέσα σε αυτά. Όλες οι προσπάθειες μου για περιορισμό των κινήσεων του χρήστη μέσα στα όρια του δωματίου έχουν επιφέρει μη επιθυμητά αποτελέσματα και διατάρασσαν την ομαλή περιήγησή του στο χώρο.

V. Συμπεράσματα

Στην ενότητα αυτή παραθέτω τα σημαντικότερα στοιχεία από όσα αναφέρθηκαν στην παρούσα εργασία καθώς και τα συμπεράσματα στα οποία κατέληξα μετά από την ανάλυση των παραπάνω πληροφοριών. Στη συνέχεια προτείνω κάποιες ιδέες για πιθανή μελλοντική έρευνα η οποίες θα συμπλήρωναν αρμονικά και θα πήγαιναν ένα βήμα παραπέρα τη δική μου προσπάθεια.

Η OpenGL είναι ουσιαστικά μια γλώσσα υψηλού επιπέδου η οποία δίνει στον χρήστη τη δυνατότητα να ελέγξει και να επηρεάσει άμεσα τον τρόπο απόδοσης της 3D σκηνής του. Είναι ένα ισχυρό εργαλείο στα χέρια του προγραμματιστή και για αυτό το λόγο χρησιμοποιείται για την ανάπτυξη μοντέλων υψηλής ποιότητας και απόδοσης τόσο σε υπολογιστές όσο και σε κινητές συσκευές. Η OpenGL υποστηρίζει με αποδοτικό τρόπο την ανάπτυξη realtime παιχνιδιών και άλλου είδους γραφικών σε κάθε είδους συσκευή λόγο τις μεταφερσημότητας των προγραμμάτων, τα οποία μπορούν να εκτελεστούν σε όλα τα λειτουργικά συστήματα.

Η ραγδαία ανάπτυξη των portable συσκευών όπως τα i-phones, smartphones, androids κλπ είχε ως αποτέλεσμα την χρήση της OpenGL από ολοένα και περισσότερους προγραμματιστές καθώς αποτελεί τη βασικότερη γλώσσα υποστήριξης των συσκευών αυτών. Γενικότερα μια μορφή της OpenGL, η OpenGL for Embedded Systems, είναι η γλώσσα που υποστηρίζεται από τα λειτουργικά συστήματα των περισσοτέρων mobile phones σήμερα.

Όσον αφορά το κομμάτι της γλώσσας Java OpenGL, τα σημαντικότερα στοιχεία της θεωρίας στα οποία αξίζει να δώσει βάση ο αναγνώστης είναι τα εξής: α) η JOGL υποστηρίζεται μόνο από νεότερες εκδόσεις της Java και β) ενώ η βασική γλώσσα ανάπτυξης της OpenGL είναι η C, κάθε αλλαγή στις βιβλιοθήκες της γλώσσας μπορεί να μεταφερθεί αυτόματα από τη C στις βιβλιοθήκες της JOGL. Το πολύτιμο αυτό εργαλείο που κάνει τόσο εύκολη τη μετάβαση του κώδικα από τη γλώσσα C στην Java ονομάζεται Gluegen.

Έκτος από το θεωρητικό κομμάτι, ένας από τους σημαντικότερους σκοπούς της εργασίας ήταν η ανάπτυξη του δικού μου προγράμματος εικονικής περιήγησης του χρήση σε ένα τρισδιάστατο δωμάτιο. Στην τελευταία ενότητα της εργασίας αναπτύσσω το πρόγραμμα Room στο οποίο παρουσιάζω έναν τρόπο ανάπτυξης τρισδιάστατων εσωτερικών χώρων οι οποίοι πιθανόν να περιλαμβάνουν έπιπλα και άλλες λεπτομέρειες που μπορεί να έχει ένα πραγματικό δωμάτιο.

Κατά τη γνώμη μου, το πρόγραμμα αυτό αποτελεί την κατάλληλη αφετηρία για την παραγωγή εφαρμογών μεγαλύτερου μεγέθους οι οποίες θα προσομοιάζουν τρισδιάστατους χώρους με αποδοτικό τρόπο. Μελλοντική έρευνα που μπορεί να γίνει στο πεδίο αυτό αφορά την ψηφιακή αναπαράσταση σημαντικών χώρων, όπως μουσεία, γκαλερί, βιβλιοθήκες, μνημεία και άλλου είδους εσωτερικούς χώρους, αλλά και εξωτερικούς υπό κάποιες προϋποθέσεις, οι οποίοι αξίζουν να επισκεφτεί κάποιος. Με αυτό τον τρόπο ο χρήστης θα μπορεί να περιηγείται με τη χρήση του ποντικιού ή του πληκτρολογίου του σε έναν εικονικό χώρο ο οποίος προσομοιάζει τον χώρο που επιθυμεί να επισκεφτεί.

Σε μια εποχή όπου αναπτύσσονται όλο και περισσότερες εφαρμογές τύπου εικονικών χαρτών , όπως το Google Maps και Street View, μια αρκετά χρήσιμη και εντυπωσιακή εφαρμογή θα ήταν η δυνατότητα εντοπισμού των σημαντικότερων κτιρίων στον χάρτη και η

είσοδος και περιήγηση του χρήστη σε αυτούς. Έτσι, παραδείγματος χάριν, σε μια περιήγηση του χρήστη στους δρόμους του Παρισιού, αν το επιθυμεί ο ίδιος, θα μπορεί να επισκεφτεί το μουσείο του Λούβρου και να περιηγηθεί με εικονικό τρόπο στις αίθουσες θαυμάζοντας τα εκθέματά του. Ήδη, ένα επιπλέον χαρακτηριστικό του προγράμματος Google Earth φαίνεται να αποτελεί τον προάγγελο μιας πιθανής εφαρμογής εικονικής περιήγησης στον χώρο. Η εταιρεία Google έχει σχεδιάσει με τρισδιάστατο τρόπο ορισμένα από τα σημαντικότερα μνημεία και τοποθεσίες στην ξηρά, αλλά και υποθαλάσσια όπως τα ναυάγια του Τρίγωνου των Βερμούδων, ώστε ο χρήστης να μπορεί να τα δει εικονικά και να περιηγηθεί σε αυτά όπου είναι δυνατό.

Επίσης, η εικονική αναπαράσταση χώρων μπορεί να έχει εκπαιδευτικούς σκοπούς, καθώς παρέχεται στους μαθητές η δυνατότητα να περιηγηθούν σε μουσεία και άλλου είδους χώρους εκπαιδευτικού χαρακτήρα εύκολα με τη χρήση υπολογιστή. Τέλος, το εργαλείο αυτό μπορεί να φανεί χρήσιμο ακόμα και σε ιδιώτες ή καταστηματάρχες οι οποίοι για οποιοδήποτε λόγο, όπως η προσέλκυση πελατών και η διαφήμιση, επιθυμούν να παρουσιάσουν τον χώρο τους στο κοινό. Σε γενικές γραμμές, συμπεραίνουμε ότι η συστηματική και επιστημονική έρευνα στον τομέα της εικονικής περιήγησης σε χώρους μπορεί να αποδώσει σημαντικά αποτελέσματα και να παρέχει χρήσιμα εργαλεία σε ένα ευρύ φάσμα δραστηριοτήτων.

Βιβλιογραφία

[1] **Dave Shreiner, The Khronos OpenGL ARB Working Group, 2009. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Versions 3.0 and 3.1*, 7η έκδοση. United States: Addison-Wesley Publishing Company.**

[2] **Mason Woo , Jackie Neider, Tom Davis, OpenGL Architecture Review Board, 1996. *Opengl Programming Guide: The Official Guide to Learning Opengl*, Version 1.1. United States: Addison-Wesley Publishing Company.**

[3] **Gene Davis, 2004, *Learning Java Bindings for OpenGL (JOGL)*. Woods Cross City, Utah**

[4] **Θ. Θεοχάρης, Α. Μπεμ, 1999, *Γραφικά Αρχές & Αλγόριθμοι.* Αθήνα: Εκδόσεις Συμμετρία**

Web Sites

[5] **Wikipedia, the free encyclopedia, *OpenGL*. Διαθέσιμο στον δικτυακό τόπο: http://en.wikipedia.org/wiki/OpenGL (τελευταία πρόσβαση στις 10/09/2010)**

[6] **Wikipedia, the free encyclopedia, *Comparison of OpenGL and Direct3D*. Διαθέσιμο στον δικτυακό τόπο: http://en.wikipedia.org/wiki/Comparison_of_OpenGL_and_Direct3D (τελευταία πρόσβαση στις 05/11/2010)**

[7] **Song Ho Ahn, 2006, *OpenGL Tesellation*. Διαθέσιμο στον δικτυακό τόπο: http://www.songho.ca/opengl/gl_tessellation.html (τελευταία πρόσβαση στις 10/09/2010)**

[8] **By Bill Day, JavaWorld.com, 1999, *3D graphics programming in Java, Part 3: OpenGL*. Διαθέσιμο στον δικτυακό τόπο: http://www.javaworld.com/javaworld/jw-05-1999/jw-05-media.html?page=1 (τελευταία πρόσβαση στις 05/11/2010)**

[9] **Java.net - The Source for Java Technology Collaboration, *Jogl - User's Guide*. Διαθέσιμο στον δικτυακό τόπο: http://download.java.net/media/jogl/builds/archive/jsr-231-1.1.0/jogl-1.1.0-windows-i586/Userguide.html (τελευταία πρόσβαση στις 05/02/2012)**

[10] **Song Ho Ahn, 2008, *OpenGL Transformation*. Διαθέσιμο στον δικτυακό τόπο: http://www.songho.ca/opengl/gl_transform.html#example2 (τελευταία πρόσβαση στις 15/11/2010)**

[11] **Rab 3D, General 3D Tutorials, *Creating a Page Template.* Διαθέσιμο στον δικτυακό τόπο: http://www.rab3d.com/rab3d/tutorial/608/tutorial_608-10.html (τελευταία πρόσβαση στις 15/11/2010)**

[12] **OpenGL - The Industry's Foundation for High Performance Graphics, *glRotate – OpenGL documendtation.* Διαθέσιμο στον δικτυακό τόπο: http://www.opengl.org/sdk/docs/man/xhtml/glRotate.xml (τελευταία πρόσβαση στις 22/11/2010)**

[13] **www.pookazza.com, *OpenGL: Texture Mapping.* Διαθέσιμο στον δικτυακό τόπο: http://www.pookazza.com/pookazza/index.php?option=com_content&view=article&id=19:-texture-mapping&catid=1:opengl&Itemid=2&lang=th (τελευταία πρόσβαση στις 20/12/2010)**

[14] **Laboratorio di Robotica, *Computer Graphics - Jogl Tutorials and Examples.* Διαθέσιμο στον δικτυακό τόπο: http://robot.unipv.it/index.php/didattica/grafica-3d/54#Tutorial1. (τελευταία πρόσβαση στις 15/01/2011)**

[15] **NeHe (Neon Helium) Productions, Texture Mapping – Lesson: 06. Διαθέσιμο στον δικτυακό τόπο: http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=06 (τελευταία πρόσβαση στις 2/01/2010)**

[16] **IncrediFriends. Διαθέσιμο στον δικτυακό τόπο: http://incredifriends.com/if/viewtopic.php?f=22&t=851 (τελευταία πρόσβαση στις 13/03/2011)**

Λεξικό Όρων

- A -

- **API (Application Programming Interface) -** διεπαφή των προγραμματιστικών διαδικασιών που ένα λειτουργικό σύστημα, βιβλιοθήκη ή εφαρμογή παρέχει προκειμένου να επιτρέπει να γίνονται προς αυτό αιτήσεις από άλλα προγράμματα ή / και ανταλλαγή δεδομένων
- **Applet -** μικρή εφαρμογή που εκτελεί μια συγκεκριμένη εργασία
- **Ambient Light -** διάσπαρτο φως, φωτισμός του γύρο περιβάλλοντος

- B -

- **Blending -** ανάμειξη

- C -

- D -

- **Diffuse Light -** διάχυτο φως, πέφτει με μια συγκεκριμένη κατεύθυνση και αντανακλάται προς όλες τις κατευθύνσεις
- **Directional Light -** φως με συγκεκριμένη κατεύθυνση
- **Depth Test -** έλεγχος βάθους για τον έλεγχος της ορατότητας ή μη τμημάτων αντικειμένων
- **Display List -** μια ομάδα από εντολές που έχουν αποθηκευτεί για μελλοντική εκτέλεση

- E -

- **Emission Light -** εσωτερικό φως που εκπέμπει ένα αντικείμενο

- F -

- **Flat Shading -** χρωματισμός κάθε επιφάνειας του αντικειμένου/ πολυγώνου με ένα χρώμα
- **Fog -** ομίχλη

- G -

- **GLCanvas -** αντικείμενο με προεπιλεγμένο σύνολο δυνατοτήτων της OpenGL όπου σχεδιάζουμε του τρισδιάστατο σκηνικό
- **GL (OpenGL Library) -** βασική βιβλιοθήκη γραφικών της OpenGL
- **GLU (OpenGL Utility Library) -** βοηθητική βιβλιοθήκη γραφικών της OpenGL
- **GLUT (OpenGL Utility Toolkit) -** βοηθητικό πακέτο εργαλείων για την ανάπτυξη OpenGL προγραμμάτων

- H -

- **Header Files** - αρχεία πηγαίου κώδικα που χρησιμοποιούνται σε όλα τα OpenGL προγράμματα

- I -

- J -

- **JNI (Java Native Interface) –** προγραμματιστικό πλαίσιο που επιτρέπει κώδικα σε Java να εκτελείται στην εικονική μηχανή της Java (JVM - Java Virtual Machine)
- **JFrame** - κλάση που δημιουργεί ένα πλαίσιο/ παράθυρο στο οποίο αναπτύσσουμε μια εφαρμογή

- K -

- L -

- **Lighting** - φωτισμός

- M -

- **Main** - μέθοδος στην οποία το πρόγραμμα αρχίζει να εκτελείται
- **Material** - υλικό αντικειμένου
- **Model View** - λειτουργία στην οποία αντιλαμβανόμαστε το μετασχηματισμό των αντικειμένων στο τρισδιάστατο σκηνικό

- N -

- O -

- **Orthographic Projection** - ορθογραφική προβολή

- P -

- **Polygon Rasterization** μέθοδος αναπαράστασης αντικειμένου σαν να αποτελείται από επιμέρους πολύγωνα
- **Projection** - προβολή αντικειμένου
- **Perspective Projection** - προοπτική προβολή

- Q –

- **Quadrics** - βοηθητικό αντικείμενο που χρησιμοποιούμε για την παραγωγή αντικειμένων της κλάσης GLU

- R -

- **Rotate** - περιστροφή
- **RGB(A) (Red - Green – Blue - (Alpha)) –** πρότυπο χρωματισμού όπου καθορίζουμε τις τιμές τριών βασικών χρωμάτων και του παράγοντα Άλφα

- S -

- **Shading** - φωτοσκίαση
- **Smooth Shading** - χρωματισμός κάθε επιφάνειας του αντικειμένου/ πολυγώνου με περισσότερο του ενός χρώματα
- **Specular Light** - πέφτει με μια συγκεκριμένη κατεύθυνση και αντανακλάται προς μια συγκεκριμένη κατεύθυνση

- T -

- **Tessellation** - ψηφιδοποίηση ή ψηφίδωση
- **Translate** - μεταφορά ή μετακίνηση
- **Texture** - υφή αντικειμένου

- U –

- V –

- W –

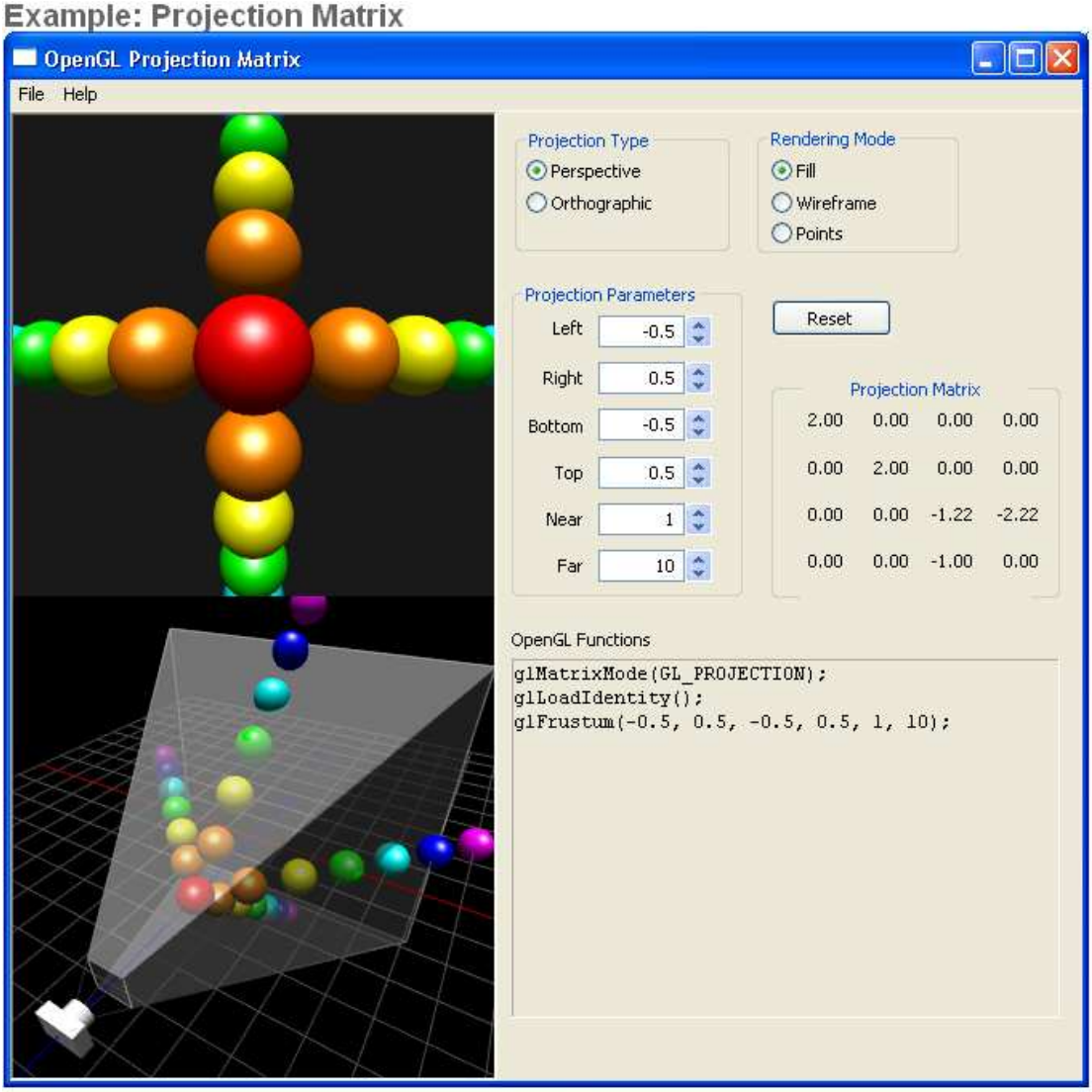
- X –

- Y –

- Z -

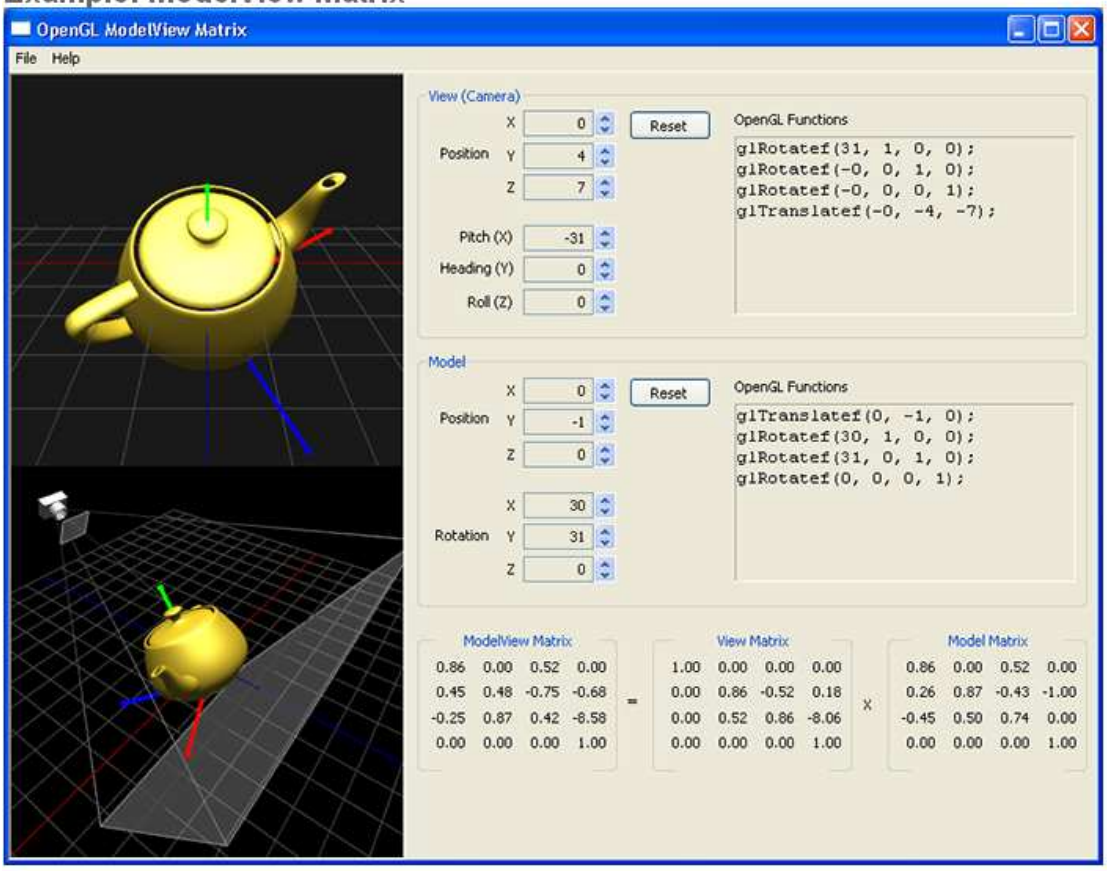
Παράρτημα

Example: Projection Matrix

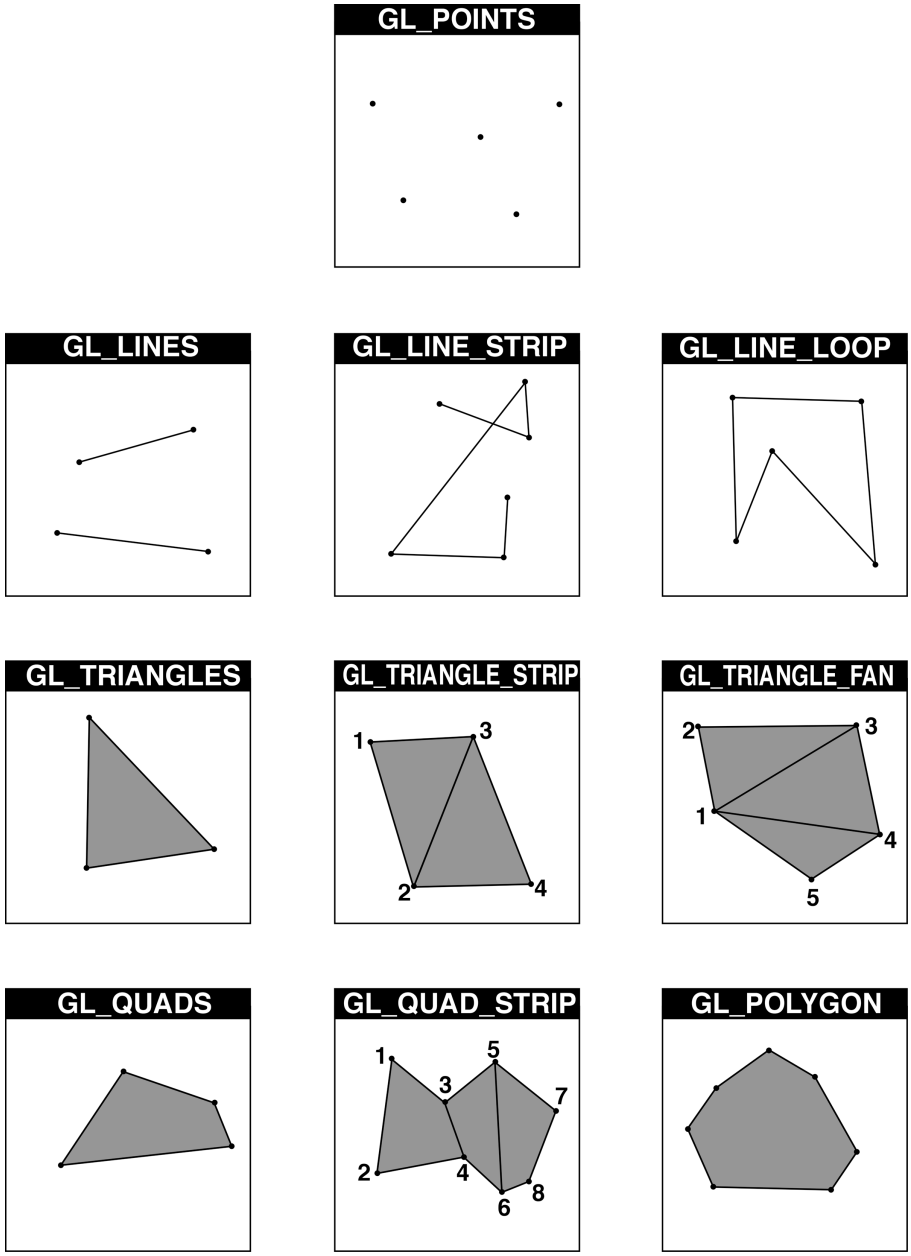


Στην παραπάνω εικόνα φαίνεται ένα στιγμιότυπο μιας εφαρμογής η οποία υπολογίζει τον πίνακα Projection Matrix, ο οποίος χρησιμοποιείται για τον προσδιορισμό της θέσης των αντικειμένων σε λειτουργία GL_PROJECTION. Η εφαρμογή μας δείχνει τον τρόπο υπολογισμού του Projection μετασχηματισμού με βάση τις παραμέτρους των συναρτήσεων glFrustum() ή glOrtho(). [10]

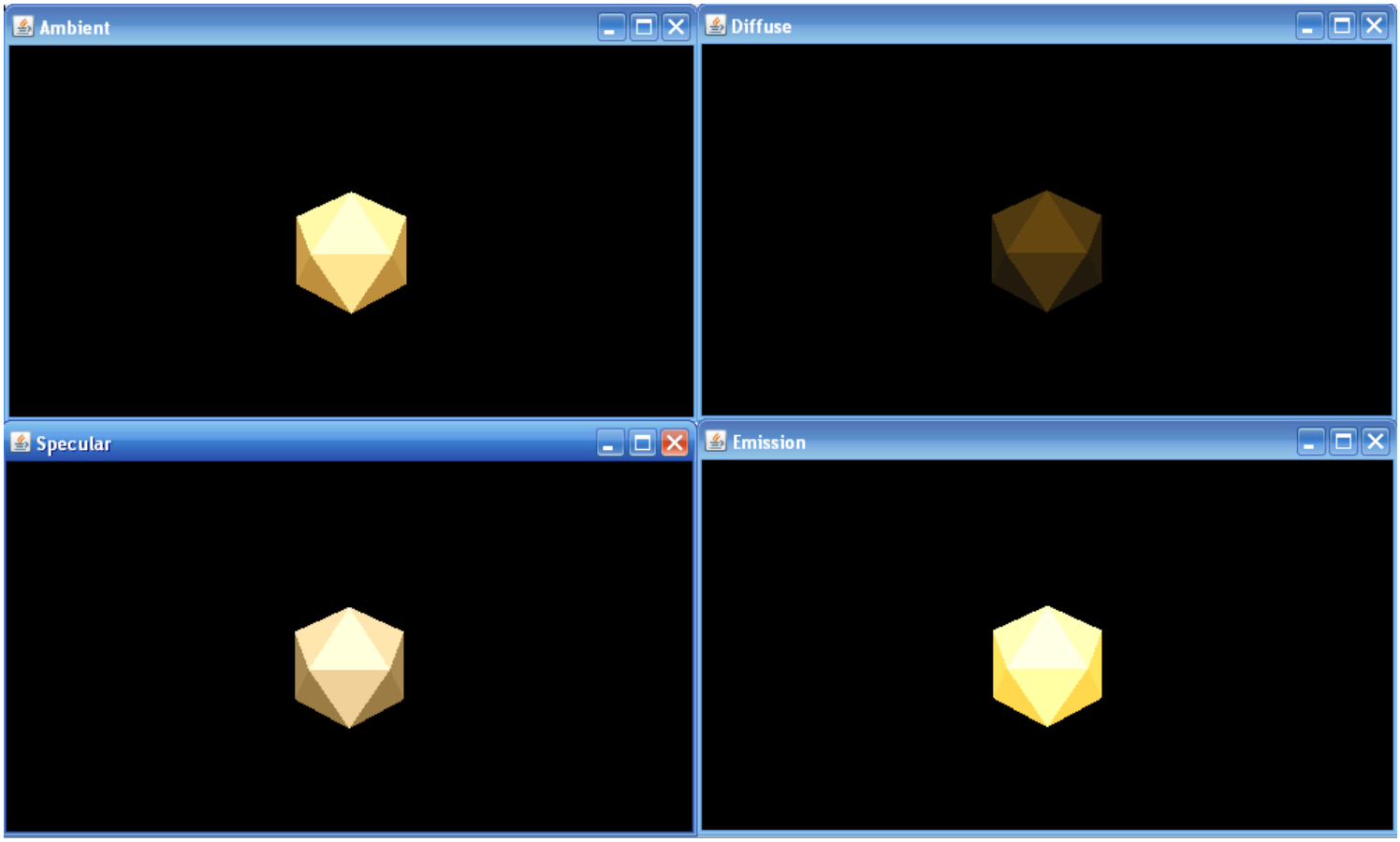
Example: ModelView Matrix



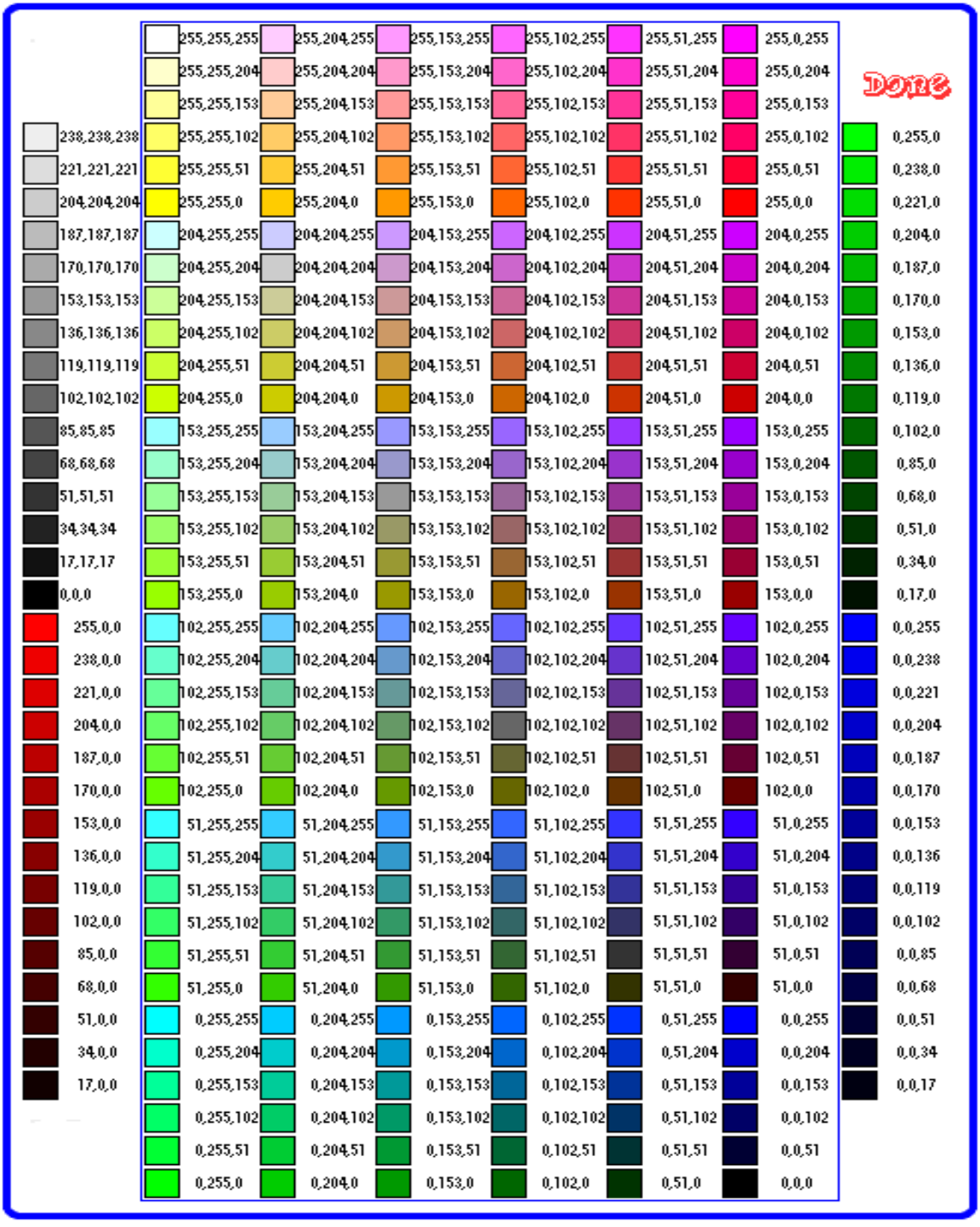
Στην εικόνα αυτή φαίνεται ένα στιγμιότυπο εφαρμογής για τον υπολογισμό του πίνακα ModelView Matrix από τους πίνακες View Matrix και Model Matrix. Οι συναρτήσεις της OpenGL οι οποίες μπορούν να χειριστούν τον μετασχηματισμό της μήτρας ModelView είναι οι glTranslatef() και glRotatef(), όπως φαίνεται και στην εικόνα. Έτσι λοιπόν για να μετακινήσουμε ένα αντικείμενο σε λειτουργία ModelVIew καλούμε πρώτα τη συνάρτηση Translatef() και στη συνέχεια τη συνάρτηση glRotatef(), όσες φορές επιθυμούμε. [10]



Όπως γνωρίζουμε, για το σχηματισμό αντικειμένων από τον χρήστη καλούμε πρώτα τη μέθοδο glBegin(). Το αντικείμενο που δέχεται σαν είσοδο για το σχηματισμό του αντικειμένου είναι μια σταθερά της κλάσης GL. Ο παραπάνω πίνακας δείχνει τα είδη σχεδιασμού που υπάρχουν και τις αντίστοιχες σταθερές τιμές που τα ενεργοποιούν. Τα είδη αυτά σχεδιασμού ουσιαστικά αναφέρονται στον τρόπο ένωσης των κορυφών των αντικειμένων που προσδιορίζονται με τη χρήση της συνάρτησης glVertex() και τα οποία βρίσκονται ανάμεσα στις εντολές glBegin() και glEnd(). [3]



Οι παραπάνω εικόνες παρουσιάζουν την επίδραση που έχουν πάνω στο ίδιο αντικείμενο τα τέσσερα βασικά είδη φωτισμού. Οι τιμές χρώματος που χρησιμοποίησα για την ενεργοποίηση των τεσσάρων αυτών ειδών φωτισμού είναι ίδιες. Επίσης, ο φωτισμός έχει την ίδια ένταση σε όλες τις περιπτώσεις.[1]



Πίνακας των τιμών των παραμέτρων RGB (red-green-blue) για τα βασικά χρώματα. Για να χρησιμοποιήσουμε τις τιμές αυτές ως είσοδο στην συνάρτηση glColor3f() της OpenGL διαιρούμε καθέναν από τους τρις αριθμούς με το 255 και στην συνέχεια εισάγουμε το αποτέλεσμα στη συνάρτηση. [16]