Πανεπιστήμιο Θεσσαλίας Βόλος 2013

Σχεδιασμός και Υλοποίηση εφαρμογής σε Android πλατφόρμα για την εκμάθηση της γλώσσας OpenGL

Design and Implementation of an OpenGL tutorial on Android platform

Διπλωματική Εργασία

του

Πολυχρόνη Χαριτίδη

Βόλος,Σεπτέμβριος 2013

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Σχεδιασμός και Υλοποίηση εφαρμογής σε Android πλατφόρμα για την εκμάθηση της γλώσσας OpenGL

Design and implementation of an OpenGL tutorial on Android platform

Διπλωματική Εργασία

του

Πολυχρόνη Χαριτίδη

Επιβλέποντες:

Παναγιώτα Τσομπανοπούλου	**Παναγιώτης Μποζάνης**
Επίκουρος Καθηγήτρια, Πανεπιστήμιο Θεσσαλίας	Αναπληρωτής Καθηγητής, Πανεπιστήμιο Θεσσαλίας

Εγκρίθηκε από την διμελή επιτροπή την ημερομηνία εξέτασης

(Υπογραφή)	*(Υπογραφή)*
.....	.....
ΚΥΡΙΟΣ ΕΠΙΒΛΕΠΩΝ	ΔΕΥΤΕΡΕΥΩΝ ΕΠΙΒΛΕΠΩΝ
Επίκουρος Καθηγήτρια Π.Θ.	Αναπληρωτής Καθηγητής Π.Θ.

Βόλος, Σεπτέμβριος 2013

(Υπογραφή)

.....

Πολυχρόνης Χαριτίδης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών,

Πανεπιστημίου Θεσσαλίας

Βόλος, Σεπτέμβριος 2013

Στην οικογένεια μου και στους φίλους μου

Ευχαριστίες

Με την περάτωση της παρούσας εργασίας, θα ήθελα να ευχαριστήσω θερμά τους επιβλέποντες της Διπλωματικής εργασίας κ. Παναγιώτα Τσομπανοπούλου και κ. Παναγιώτη Μποζάνη για την εμπιστοσύνη που επέδειξαν στο πρόσωπό μου, για την άριστη συνεργασία, τις ουσιώδεις υποδείξεις και παρεμβάσεις, που διευκόλυναν την εκπόνηση της Διπλωματικής εργασίας.

Επίσης θα ήθελα να εκφράσω την εκτίμησή μου για όλα τα παιδιά, με τα οποία συνεργάστηκα όλα αυτά τα χρόνια των προπτυχιακών σπουδών μου.

Τέλος, οφείλω ένα μεγάλο ευχαριστώ στην οικογένειά μου και στους φίλους μου και ιδιαίτερα στην Έφη και στο Λεωνίδα για την αμέριστη υποστήριξη και την ανεκτίμητη βοήθεια που μου παρείχαν τόσο κατά την διάρκεια των σπουδών μου, όσο και κατά την εκπόνηση της Διπλωματικής εργασίας.

Πολυχρόνης Χαριτίδης

Βόλος, 2013

Περίληψη

Σκοπός της παρούσας Διπλωματικής εργασίας ήταν η σχεδίαση και υλοποίηση μιας εφαρμογής στο Android λειτουργικό η οποία στόχο έχει να βοηθήσει το χρήστη να κατανοήσει τις βασικές αρχές της OpenGL μέσα από μια σειρά διαδραστικών παραδειγμάτων πάνω σε σημαντικούς τομείς της γλώσσας αυτής.

Συγκεκριμένα, ο χρήστης έχει τη δυνατότητα να ζωφραφίσει βασικά σχήματα, να τα μετακινεί , να τα περιστρέφει, να τα μεγαλώνει, να αλλάζει τα χρώματά τους χωρίς καν να γνωρίζει πώς θα μπορούσε να κάνει τέτοιες ενέργειες προγραμματιστικά. Πέρα από την επίδειξη των δυνατοτήτων αυτών η εφαρμογή εμφανίζει στο χρήστη τον κώδικα με τον οποίο γίνονται οι ενέργειες αυτές και έτσι ο χρήστης μπορεί να αφομοιώσει πολύ εύκολα βασικές εντολές της γλώσσας.

Οι τομείς της γλώσσας που καλύπτονται στην εφαρμογή την καθιστούν ικανή να βοηθήσει τους φοιτητές στο μάθημα των γραφικών του τμήματός μας.

Abstract

The purpose of this thesis was the design and implementation of an application to the Android operating system which aims to help the user understand the fundamentals of OpenGL language through a series of interactive examples on key areas of the language.

Specifically, the user has the ability to draw primitive shapes, to move them, to rotate them, to scale them, to change their colors without even knowing how he could develop such actions. Beyond demonstrating the potential of OpenGL, the application displays the code to the user and thus the user can learn the basic commands of the language easily.

The areas of the language covered in the application enable it to help students who attend graphics lessons at our department.

Περιεχόμενα

1.Εισαγωγή

Το χειμερινό εξάμηνο του 2012-2013 παρακολούθησα το μάθημα των Γραφικών της κυρίας Τσομπανοπούλου. Παρά το γεγονός ότι ενθουσιάστηκα με το αντικείμενο του μαθήματος δεν μπόρεσα να δώσω την προσοχή που ήθελα λόγο του αυξημένου φόρτου εργασίας την περίοδο εκείνη. Για το λόγο αυτό, έμαθα αρκετά πράγματα για τα γραφικά, όμως δεν εμβάθυνα όσο θα επιθυμούσα. Έτσι, τα γραφικά έμειναν σαν ένα «απωθημένο» μέσα μου.

Όταν έφτασε ο καιρός για την επιλογή του θέματος της διπλωματικής μου είχα πολλές εναλλακτικές στο μυαλό μου, ωστόσο αυτό που κυριαρχούσε ήταν κάτι σχετικό με τα γραφικά και την OpenGL. Άρχισα να ψάχνω στο internet οτιδήποτε είχε σχέση με OpenGL και εντελώς τυχαία κατέληξα σε ένα βίντεο επίδειξης μιας εφαρμογής σε tablet, η οποία μπορούσε να διδάξει στο χρήστη OpenGL. Τότε συνειδητοποίησα τι ήθελα να κάνω. Η αγάπη μου αφενός για τα γραφικά κι αφετέρου για τον προγραμματισμό, ειδικότερα σε κινητά, με έκαναν να μην το σκεφτώ δεύτερη φορά.

Έτσι, ξεκίνησα να αναπτύσσω μια εφαρμογή με σκοπό να βοηθήσω τους συμφοιτητές μου να κατανοήσουν εύκολα και γρήγορα την OpenGL από το κινητό τους. Κατά τη διάρκεια της υλοποίησης αντιμετώπισα αρκετά προβλήματα και εκπλήξεις, όπως το γεγονός ότι στο android δεν υπάρχει η OpenGL, όπως την μάθαμε στο μάθημα των γραφικών, αλλά η OpenGL ES η οποία έχει αρκετές διαφορές με την προηγούμενη αλλά και άλλες δυνατότητες. Για παράδειγμα, ένα απλό τρίγωνο στην OpenGL παράγεται σε 5 γραμμές κώδικα ενώ στην OpenGL ES σε 50. Η πολυπλοκότητα αυτή στη δημιουργία σχημάτων με οδήγησε σε διάφορες λύσεις που θα περιγραφούν αναλυτικότερα.

Στο ταξίδι αυτό απέκτησα νέες προγραμματιστικές ικανότητες, επικοινώνησα με πολλούς ανθρώπους που είχαν το ίδιο πάθος με τα γραφικά και τον προγραμματισμό, αγάπησα περισσότερο το αντικείμενο και κυρίως έμαθα πάρα πολλά ενδιαφέροντα πράγματα που ελπίζω να σας μεταφέρω μέσω της Διπλωματικής εργασίας αυτής αλλά και της εφαρμογής.

Οι παρακάτω σελίδες θα προσπαθήσουν να ρίξουν λίγο φως στα διαφορετικά στάδια ανάπτυξης της εφαρμογής.

2.Ιστορικά Στοιχεία για τα γραφικά

2.1 Η αρχή των γραφικών στους υπολογιστές

Το 1961 ο Ivan Sutherland ,ένας φοιτητής του MIT, δημιούργησε για το διδακτορικό του το Sketchpad, ένα σύστημα ζωγραφικής με ένα στυλό και έναν υπολογιστή Lincoln TX-2. Το user interface του Sketchpad ήταν επαναστατικό καθώς αποτέλεσε πρότυπο για τη δημιουργία των μετέπειτα user interfaces.

Ένας συμφοιτητής του Sutherland, ο Steve Russell, την ίδια περίοδο κάνει μία από τις μεγαλύτερες ανακαλύψεις μέχρι σήμερα , το παιχνίδι στον υπολογιστή(videogame). Ο Russell δημιούργησε το 1962 το θρυλικό παιχνίδι Spacewar το οποίο έτρεχε σε PDP-1 όπως φαίνεται και στην εικόνα.

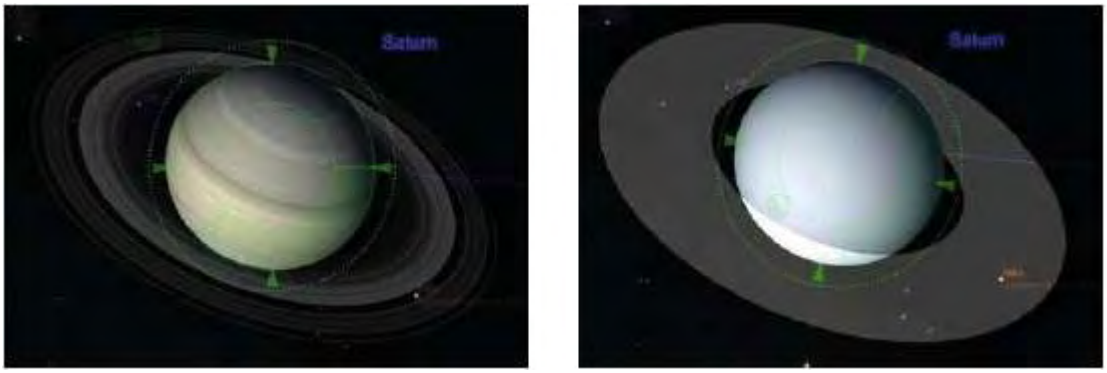
Εικόνα 1. Το Spacewar στον PDP-1

Το 1965 η IBM δημιούργησε αυτό που θεωρείται το πρώτο ευρέως χρησιμοποιούμενο εμπορικό τερματικό γραφικών, το 2250 (Εικόνα 2). Μαζί με το φτηνό υπολογιστή IBM-1130 ή τον IBM S/340 το τερματικό χρησιμοποιήθηκε ευρέως από την επιστημονική κοινότητα.

Ίσως ένα από τα πρώτα γνωστά παραδείγματα γραφικών υπολογιστή στην τηλεόραση ήταν η χρήση του 2250 για την αναπαράσταση διάφορων φάσεων αποστολής της NASA στο διάστημα. Το όλο εγχείρημα κόστισε $100,000.

Εικόνα 2. IBM-2250 terminal

Το 1968 ο Ivan Sutherland έχοντας πάει στο πανεπιστήμιο της Utah συνέχισε να ασχολείται και να διδάσκει γραφικά. Από τα εργαστήρια του πανεπιστημίου της Utah λοιπόν ο Sutherland άρχισε να έχει κάλους μαθητές. Ο Ed Catmull, για παράδειγμα, πιστεύοντας ότι οι υπολογιστές είναι εργαλείο για δημιουργία ταινιών παρήγαγε το πρώτο animation το οποίο απεικόνιζε ένα χέρι να ανοιγοκλείνει. Στη συνέχεια, εισήγαγε την ιδέα του texture mapping (Εικόνα 3), δηλαδή την εισαγωγή εικόνων στα σχήματα έτσι ώστε να φαίνονται πιο ρεαλιστικά.



Εικόνα 3. Ο Κρόνος με και χωρίς texture mapping

Αργότερα ο Catmull θα εργαστεί στην Pixar και τελικά θα γίνει πρόεδρος στην Disney Animation Studios.

Άλλοι διακεκριμένοι επιστήμονες από το πανεπιστήμιο της Utah που είχαν την επιρροή του Sutherland ήταν:

Ο John Warnock που ανέπτυξε το Portable Document Format (PDF) και ήταν συνιδρυτής της Adobe.

Ο Jim Clark που ίδρυσε την Silicon Graphics, η οποία εφοδίαζε το Hollywood με γραφικά, και δημιούργησε το 3D framework της OpenGL.

Ο Jim Blinn που δημιούργησε επαναστατικά γραφικά για την NASA (Εικόνα 4), καθώς και τον ανταγωνιστή της OpenGL, το Direct3D.

Εικόνα 4. Τα γραφικά που δημιούργησε ο Blinn για μια αποστολή της NASA

2.2 Η ραγδαία εξέλιξη των γραφικών

Το 1980 η επιστήμη των γραφικών άρχισε να γνωρίζει μεγάλη άνθηση αφενός λόγω των απαιτήσεων που είχαν οι ταινίες στο Hollywood και αφετέρου η όλο ένα αυξανόμενη υπολογιστική ισχύς των νέων υπολογιστών σε συνδυασμό με τη χαμηλή τιμή τους. Έτσι στη δεκαετία του 80΄ θα αρχίσουν να εφαρμόζονται πολλές νέες τεχνικές και καινοτομίες. Κάποιες απ' αυτές ήταν:

- Ο Frank Crow ανέπτυξε μια πρακτική μέθοδο *anti-aliasing*. Aliasing είναι το φαινόμενο κατά το οποίο παράγονται πριονωτές ακμές στα σχήματα, κάτι που οφείλεται στη χαμηλή ανάλυση της οθόνης. Η μέθοδος του Crow είχε τη δυνατότηνα να κάνει τις ακμές λείες και έτσι το αποτέλεσμα ήταν πιο φυσικό και πραγματικό.
- Ο Loren Carpenter ανέπτυξε μια τεχνική με την οποία μπορούσε να παράγει αλγοριθμικά λεπτομερή τοπία με τη χρήση fractals.
- Ο Turner Whitted ανέπτυξε την τεχνική *ray tracing,* η οποία μπορούσε να παράξει ρεαλιστικές σκηνές αντικειμένων με αντανάκλαση, όπως ποτήρια.(Εικόνα 5)
- Τέλος, η δημιουργία του *Star Trek II: The Wrath of Khan,* η οποία δημιουργήθηκε πλήρως από υπολογιστές έδειξε τις δυνατότητες που είχαν τα γραφικά την περίοδο εκείνη.

Εικόνα 5. Ρεαλιστικό τοπίο με χρήση νέων τεχνικών

Στη δεκαετία του 90' τα γραφικά άρχισαν να γίνονται αναπόσπαστο κομμάτι των ταινιών με παραδείγματα όπως *Terminator 2: Judgment Day, Toy Story, Jurassic Park*, *Titanic.* Στο τέλος της δεκαετίας αυτής είναι δύσκολο να βρει κανείς ταινία χωρίς τη βοήθεια γραφικών υπολογιστή.

2.3 Διάφορα λογισμικά γραφικών

Για την επίτευξη όλων των παραπάνω χρειάστηκε να δημιουργηθούν λογισμικά κατάλληλα για την παραγωγή γραφικών. Τα λογισμικά έχουν πολλές ομοιότητες αλλά και διαφορές μεταξύ τους ανάλογα την ειδίκευση και το λόγο που έχουν δημιουργηθεί.

2.3.1 OpenGL

Η Open Graphics Library (OpenGL) δημιουργήθηκε από τις καινοτόμες προσπάθειες της *Silicon Graphics (SGI),* το δημιουργό υπολογιστών για γραφικά με υψηλές απαιτήσεις. Το framework γραφικών της εταιρίας, *IRIS-GL*, την εποχή εκείνη είχε γίνει standard στη βιομηχανία των γραφικών. Έτσι, για να εδραιώσει την κυριαρχία της η SGI δημιούργησε ένα open-source framework βασισμένο στο *IRIS-GL* το *OpenGL, το 1992.*

Η ανάπτυξη των smart phones οδήγησε στη δημιουργία της OpenGL for Embedded Systems (OpenGL ES). Η OpenGL ES είναι πιο απλοποιημένη σε σχέση με την OpenGl για να τρέχει σε CPU χαμηλής ισχύος. Αυτό είχε σαν αποτέλεσμα να χρησιμοποιηθεί ευρέως σε πολλές πλατφόρμες όπως το Android, το iOS, το HP's WebOS, το Nintendo 3DS και στο BlackBerry.

Υπάρχουν τρεις εκδόσεις της OpenGL ES, η 1.x , η 2.x και η 3.x. Οι περισσότερες συσκευές τις υποστηρίζουν όλες. Η έκδοση 1.x βασίζεται περισσότερο στο πρωτότυπο OpenGL specification και είναι αυτή που χρησιμοποιείται στην παρούσα εργασία.

2.3.2 Direct3D

Το Direct3D (D3D) είναι η απάντηση της Microsoft στην OpenGL και απευθύνεται σε προγραμματιστές παιχνιδιών. Το 1995, η Microsoft αγόρασε τη μικρή εταιρία RenderMorphics που ειδικευόταν στη δημιουργία 3D framework, με το όνομα RealityLab, για παραγωγή παιχνιδιών. Το RealityLab μετατράπηκε στο Direct3D και πρωτοεμφανίστηκε στην αγορά το 1996. Αν και ήταν βασισμένο σε Windows-based συστήματα χρησιμοποιήθηκε από το ευρύ κοινό σε όλες τις πλατφόρμες της Microsoft οπως τα Windows, το Windows 7 Mobile και το Xbox.

2.3.3 QuickDraw 3D

Ένα παράδειγμα βιβλιοθήκης υψηλού επιπέδου γενικού σκοπού είναι η QuickDraw 3D. Κάτω απ' την QD3D υπάρχει ένα λεπτό στρώμα, το RAVE. Οι χρήστες μπορούν να χρησιμοποιήσουν τη στάνταρ έκδοση του RAVE για κανονικό rendering. Οι πιο φιλόδοξοι μπορούν να γράψουν το δίκο τους για να προβάλουν τα τοπία τους με πιο καλλιτεχνικό τρόπο.

Όταν ο Steve Jobs αποφάσισε ότι θα χρησιμοποιήσει την OpenGL για το Mac άρχισε η δύση του QuickDraw 3D.

2.3.4 OGRE

Ένα άλλο λογισμικό γραφικών είναι το OGRE(Object-oriented Rendering Engine). Δημιουργήθηκε το 2005 και μπορεί να χρησιμοποιήσει και την OpenGL και το Direct3D ως χαμηλά επίπεδα κάτι που προσφέρει στο χρήστη ένα σταθερό και δωρεάν εργαλείο. Το μέγεθος της κοινότητας του OGRE είναι εντυπωσιακό.

2.3.5 OpenSceneGraph

Το OpenSceneGraph δημιουργήθηκε για iOS συσκευές. Κάνει περίπου ό,τι και το QuickDraw 3D. Έχει πολλές δυνατότητες, όπως την εισαγωγή βίντεο σε 3D εφαρμογές. Η γνώση της OpenGL συνιστάται διότι πολλές από τις συναρτήσεις της OpenSceneGraph είναι παρόμοιες με αυτές της OpenGL.

2.3.6 Unity3D

Αντίθετα από το OGRE, το QD3D, ή το OpenSceneGraph, το Unity3D είναι ένα εμπορικό cross-platform game engine που τρέχει και στο Android και στο iOS. Είναι σχεδιασμένο εξ' ολοκλήρου για παιχνίδια και έχει δυνατότητες που εξειδικεύονται σ' αυτά.

3. Βασικά σχήματα

3.1 Βασικά σχήματα στην OpenGL

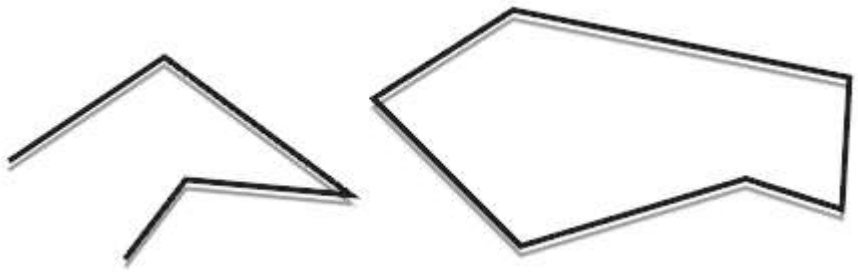
Τα σχήματα στην OpenGL μοιάζουν πολύ με αυτά που όλοι ξέρουμε. Ωστόσο, υπάρχουν κάποιες διαφορές. Μία διαφορά είναι το υπολογιστικό σφάλμα που προκαλείται από τους υπολογιστές. Μία άλλη σημαντική διαφορά έγκειται στο γεγονός ότι για την αναπαράσταση ενός σημείου στον υπολογιστή χρειαζόμαστε ένα pixel το οποίο είναι πολύ μεγαλύτερο από τη μαθηματική έννοια του σημείου. Έτσι, υπάρχει περίπτωση πολλά σημεία στην OpenGL να ζωγραφίζονται πάνω στο ίδιο pixel.

3.1.1 Σημεία

Ένα σημείο αναπαρίσταται από ένα σύνολο δεκαδικών αριθμών το οποίο ονομάζεται κορυφή (vertex). Όλοι οι εσωτερικοί υπολογισμοί γίνονται θεωρώντας ότι οι κορυφές είναι τριών διαστάσεων. Οι κορυφές που ορίζονται από το χρήστη ως δισδιάστατες παίρνουν αυτόματα από την OpenGL τη z-συντεταγμένη ίση με το μηδέν.

3.1.2 Γραμμές

Στην OpenGL, ο όρος ‘’γραμμή’’ αναφέρεται σε ένα τμήμα γραμμής και όχι στη μαθηματική έκδοση που εκτείνεται μέχρι το άπειρο και απ’τις δυο πλευρές. Υπάρχουν εύκολοι τρόποι να οριστεί μια ενωμένη ακολουθία από τμήματα γραμμών ή ακόμα και μια κυκλική ακολουθία από τμήματα (Εικόνα 6). Σε κάθε περίπτωση, ωστόσο, οι γραμμές που αποτελούν τις ενωμένες ακολουθίες ορίζονται από τις κορυφές στην αρχή και το τέλος τους.

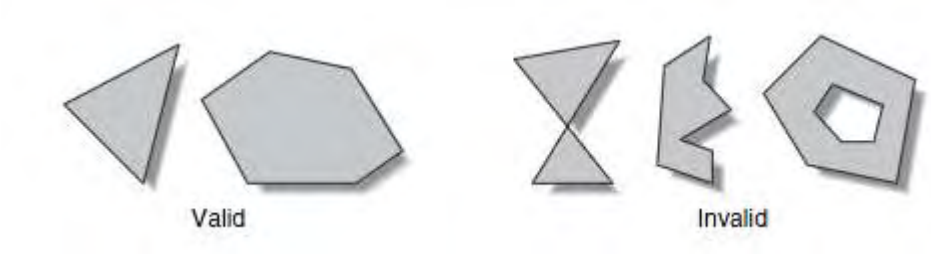


Εικόνα 6. Δύο ενωμένες ακολουθίες από τμήματα γραμμών

3.1.3 Πολύγωνα

Τα πολύγωνα είναι περιοχές που έχουν δημιουργηθεί από κλειστούς βρόγχους τμημάτων γραμμών, όπου τα τμήματα γραμμών ορίζονται από τις κορυφές στην αρχή και το τέλος κάθε γραμμής.

Γενικά, τα πολύγωνα μπορούν να γίνουν πολύπλοκα. Έτσι, η OpenGL επιβάλλει κάποιους περιορισμούς στο τι αποτελεί ένα βασικό πολύγωνο. Πρώτον, οι ακμές ενός OpenGL πολυγώνου δεν μπορούν να τέμνονται. Δεύτερον, τα OpenGL πολύγωνα πρέπει να είναι κυρτά (Εικόνα 7).



Εικόνα 7.Έγκυρα και άκυρα πολύγωνα

Ο λόγος για τον οποίο η OpenGL επιβάλλει τους παραπάνω περιορισμούς είναι γιατί είναι ευκολότερο και γρηγορότερο να ζωγραφιστούν τα πολύγωνα αυτά. Έτσι τα σύνθετα πολύγωνα (κοίλα, με τρύπες) μπορούν να περιγραφούν απο ένα σύνολο απλών πολυγώνων.

3.1.4 Καθορισμός κορυφών

Στην OpenGL κάθε γεωμετρικό σχήμα περιγράφεται από ένα σύνολο κορυφών. Χρησιμοποιείται η εντολή *glVertex*()* για να οριστεί μια κορυφή.

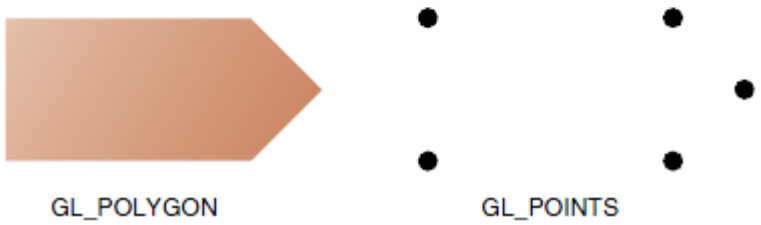
Η παραπάνω εντολή ορίζει μία κορυφή για χρήση της στην περιγραφή ενός γεωμετρικού αντικειμένου. Μπορεί να πάρει μέχρι τέσσερις συντεταγμένες (*x, y, z, w*) για μια συγκεκριμένη κορυφή ή το λιγότερο δύο επιλέγοντας την κατάλληλη έκδοση της εντολής. Αν επιλεχτεί έκδοση στην οποία η z ή η w δεν ορίζονται τότε η z θεωρείται 0 και η w θεωρείται 1. Οι κλήσεις της *glVertex*()* είναι έγκυρες μόνο μέσα σε ένα ζευγάρι εντολών *glBegin()* και *glEnd().*

Στην glVertex*() ο αστερίσκος υποδηλώνει την κατάληξη του κώδικα που απαιτείται για αυτήν τη συνάρτηση. Οι θέσεις των συντεταγμένων στην OpenGL μπορούν να δοθούν σε δύο, τρεις ή τέσσερις διαστάσεις. Έτσι, στην εντολή glVertex*() χρησιμοποιούμε τους αριθμούς 2, 3, 4 για να προσδιορίσουμε την διαστατικότητα της θέσης των συντεταγμένων. Πρέπει, επίσης, να δηλώσουμε ποιος τύπος δεδομένων θα χρησιμοποιηθεί για τον ορισμό των αριθμητικών τιμών των συντεταγμένων. Αυτό επιτυγχάνεται με μια δεύτερη κατάληξη στην εντολή. Οι καταλήξεις για τον ορισμό ενός τύπου δεδομένων είναι οι i (integer), s (short), f (float), d (double). Τέλος, αν χρησιμοποιηθεί πίνακας συντεταγμένων αντί για απευθείας αριθμητική τιμή προσαρτήσουμε μια τρίτη κατάληξη, την v (vector).

3.1.5 Ζωγραφική βασικών σχημάτων με την OpenGL

Για τη δημιουργία βασικών σχημάτων αφού καθοριστεί το σύνολο των κορυφών που απαιτείται με τη χρήση των εντολών *glVertex*()* αυτές θα πρέπει να μπουν ανάμεσα στην κλήση των εντολών *glBegin()* και *glEnd().* Το όρισμα που χρησιμοποιείται στην *glBegin()* καθορίζει τι τύπος γεωμετρικού σχήματος θα κατασκευαστεί από τις κορυφές (Εικόνα 8).

```
glBegin(GL_POLYGON);
   glVertex2f(0.0, 0.0);
   glVertex2f(0.0, 3.0);
   glVertex2f(4.0, 3.0);
   glVertex2f(6.0, 1.5);
   glVertex2f(4.0, 0.0);
glEnd();
```

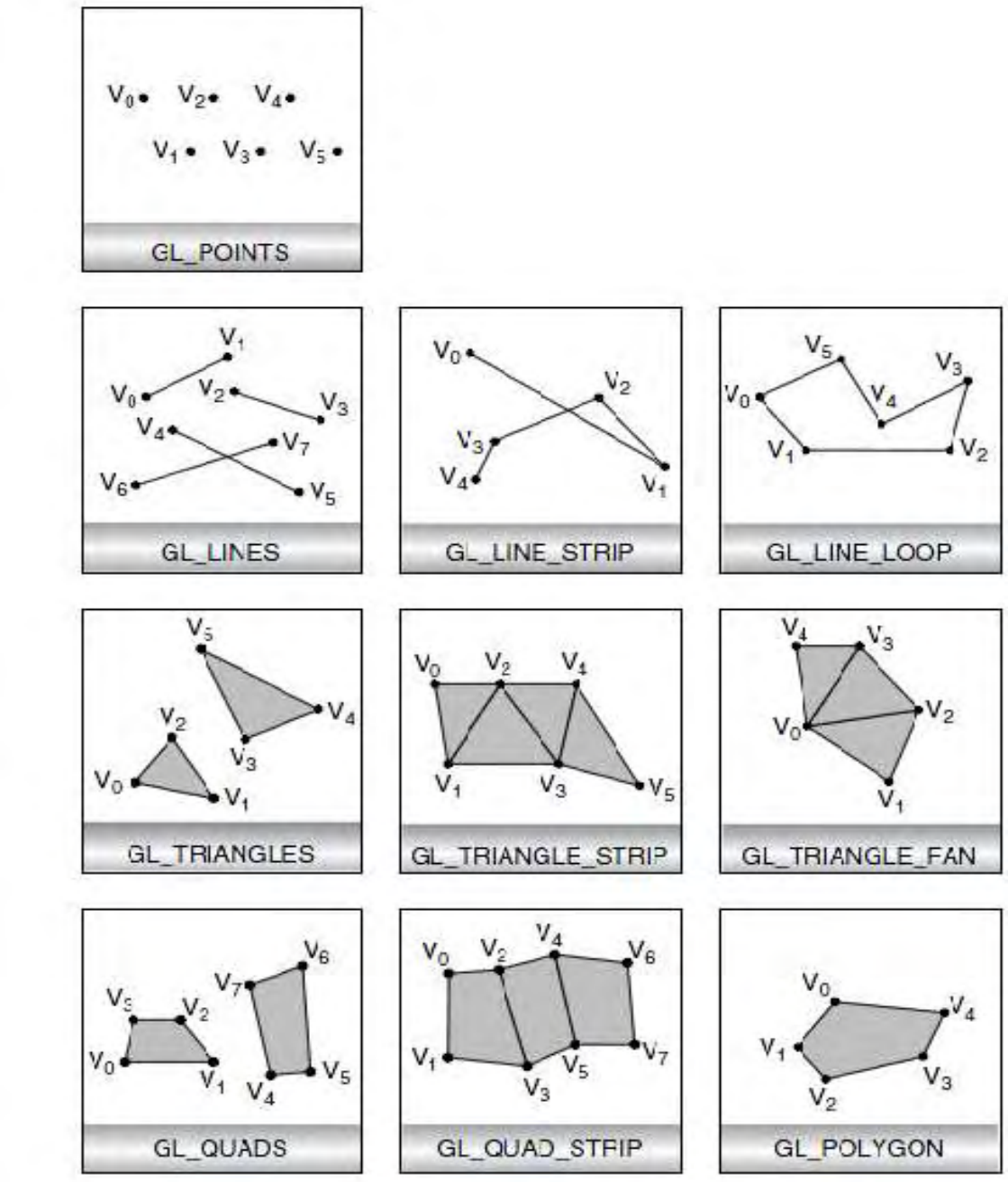


Εικόνα 8. Ζωγραφική Πολυγώνου η ενός Συνόλου Σημείων

void **glBegin**(GLenum *mode*);
Μαρκάρει την αρχή από μια λίστα δεδομένων κορυφής το οποίο περιγράφει ένα γεωμετρικό σχήμα. Ο τύπος του σχήματος φαίνεται από το mode το οποίο μπορεί να πάρει κάποια από τις τιμές του παρακάτω πίνακα.

void **glEnd**(void);
Μαρκάρει το τέλος της λίστας δεδομένων κορυφής.

Στην Εικόνα 9 φαίνονται όλα τα βασικά σχήματα και πώς αυτά περιγράφονται από τις κορυφές τους.



Εικόνα 9. Βασικά γεωμετρικά σχήματα

Θεωρώντας ότι n κορυφές (v_0,v_1,v_2…v_{n-1}) έχουν προσδιοριστεί ανάμεσα στις εντολές *glBegin(GLenum mode);* και *glEnd();* Τότε ανάλογα με την τιμή του mode έχουμε:

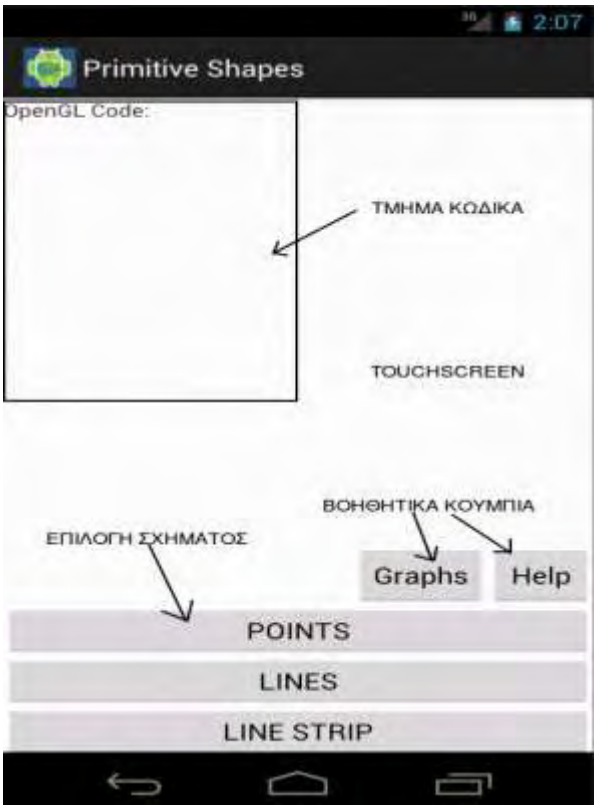
Τιμή	**Αποτέλεσμα**
GL_POINTS	Ζωγραφίζεται ένα σημείο για κάθε ένα από τις n κορυφές.
GL_LINES	Ζωγραφίζεται μια ακολουθία από μη ενωμένα τμήματα γραμμών. Τα τμήματα ζωγραφίζονται μεταξύ των κορυφών v_0 και v_1,v_2 και v_3 κ.ο.κ. Αν ο n είναι περιττός, τότε η κορυφή v_{n-1} αγνοείται.
GL_LINE_STRIP	Ζωγραφίζεται ένα τμήμα γραμμής από το v_0 στο v1, μετά από το v_1 στο v_2 κ.ο.κ. και τέλος ζωγραφίζεται τμήμα γραμμής από το v_{n-2} στο v_{n-1}. Το σύνολο των γραμμών που ζωγραφίζονται είναι n-1.
GL_LINE_LOOP	Το ίδιο με παραπάνω, με ένα τμήμα γραμμής να ζωγραφίζεται από το v_{n-1} στο v_0 ολοκληρώνοντας ένα βρόγχο.
GL_TRIANGLES	Ζωγραφίζεται μια ακολουθία από τρίγωνα χρησιμοποιώντας τις κορυφές v_0, v_1, v_2, μετά τις v_3, v_4, v_5 κ.ο.κ. Αν το n δεν είναι πολλαπλάσιο του 3, οι κορυφές που περισσεύουν αγνοούνται.
GL_TRIANGLE_STRIP	Ζωγραφίζεται μια ακολουθία από τρίγωνα χρησιμοποιώντας τις κορυφές v_0, v_1, v_2, μετά τις v_2, v_1, v_3, μετά τις v_2, v_3, v_4 κ.ο.κ.
GL_TRIANGLE_FAN	Ίδιο με το παραπάνω εκτός του ότι χρησιμοποιούν τις κορυφές v_0, v_1, v_2, μετά τις v_0, v_2, v_3, μετά τις v_0, v_3, v_4 κ.ο.κ.
GL_QUADS	Ζωγραφίζεται μία ακολουθία από τετράπλευρα χρησιμοποιώντας τις κορυφές v_0, v_1, v_2,v_3 μετά τις v_4, v_5, v_6, v_7 κ.ο.κ. Αν το n δεν είναι πολλαπλάσιο του 4, οι κορυφές που περισσεύουν αγνοούνται.
GL_QUAD_STRIP	Ζωγραφίζεται μία ακολουθία από τετράπλευρα χρησιμοποιώντας τις κορυφές v_0, v_1, v_3,v_2 μετά τις v_2, v_3, v_5, v_4 κ.ο.κ. Αν το n δεν είναι πολλαπλάσιο του 4, οι κορυφές που περισσεύουν αγνοούνται.
GL_POLYGON	Ζωγραφίζεται ένα πολύγωνο χρησιμοποιώντας τις κορυφές από v_0, ... , v_{n-1}. Το n πρέπει να είναι τουλάχιστον 3 αλλιώς δε ζωγραφίζεται τίποτα. Το πολύγωνο πρέπει να είναι κυρτό και οι ακμές του να μην τέμνονται.

3.2 Βασικά σχήματα στην εφαρμογή

Το πρώτο μέρος την εφαρμογής αφορά τη ζωγραφική των βασικών σχημάτων που υποστηρίζει η OpenGL.

3.2.1 Περιγραφή του interface

Η ιδέα είναι να μπορεί ο χρήστης να επιλέγει το γεωμετρικό σχήμα που επιθυμεί από μια λίστα στην οποία εμφανίζονται όλα τα πιθανά σχήματα και στη συνέχεια με το άγγιγμά του στην οθόνη να ορίζει και μια κορυφή. Έτσι, ανάλογα το σχήμα που θα διαλέξει και τον αριθμό τον κορυφών που έχει δημιουργήσει στην οθόνη, εμφανίζεται και το αντίστοιχο σχήμα. Επίσης, καθώς η εφαρμογή στόχο έχει να διδάξει το χρήστη OpenGL, υπάρχει ένα πλαίσιο στο οποίο μπορεί ο χρήστης να δει τον κώδικα με τον οποίο έγιναν όλες οι ενέργειες που μόλις είδε στην οθόνη του. Με άλλα λόγια ο χρήστης μπορεί να δει πώς μπορεί να ζωγραφίσει τα σχήματα σε περιβάλλον OpenGL. Τέλος, για επιπλέον βοήθεια στο χρήστη υπάρχει ένα κουμπί που εξηγεί αναλυτικά τις εντολές που εμφανίζονται, ενώ ένα άλλο εμφανίζει όλα τα βασικά σχήματα που μπορεί να ζωγραφίσει ο χρήστης (Εικόνα 10).



Εικόνα 10. Το interface για τα primitive shapes

3.2.2 Υλοποίηση

3.2.2.1 Το πρόβλημα

Πέρα από τον προγραμματισμό στο Android σε Java για δημιουργία του μέρους αυτού αντιμετώπισα τα πρώτα προβλήματα, καθώς αρχικά προσπάθησα να χρησιμοποιήσω την OpenGL ES για τη ζωγραφική των σχημάτων. Όπως αποδείχτηκε αργότερα κάτι τέτοιο ήταν πολύ δύσκολο να γίνει. Οι βασικοί λόγοι ήταν δύο.

Ο πρώτος ήταν ότι ο σχεδιασμός σχημάτων στην OpenGL ES είναι πολύ πιο περίπλοκος από το σχεδιασμό σε συμβατική OpenGL, απαιτεί πολλές γραμμές κώδικα σε αντίθεση με τις ελάχιστες της συμβατικής και είναι αρκετά δυσνόητος. Υπάρχουν έτοιμα σχήματα στο διαδίκτυο, ωστόσο υπήρχε κι άλλο πρόβλημα που αποτελεί το δεύτερο λόγο.

Ο δεύτερος λόγος είναι ότι η εργασία απαιτεί δημιουργία δυναμικών σχημάτων με το άγγιγμα του χρήστη. Όμως, τα σχήματα αυτά δεν ήταν εύκολα παραμετροποιήσιμα με αποτέλεσμα να μην είναι δυνατή η δημιουργία δυναμικών σχημάτων.

3.2.2.2 Η λύση

Λαμβάνοντας τα παραπάνω υπόψη κατάλαβα ότι έπρεπε να στραφώ αλλού. Τη λύση έδωσε η βιβλιοθήκη graphics του android, η οποία μπορεί να ζωγραφίσει γραμμές και άλλα σχήματα. Από ‘κει και πέρα η ιδέα είναι απλή.

Δημιούργησα μια κλάση PrimitiveShapesActivity.java, η οποία είναι το activity που ελέγχει τα κουμπιά και αρχικοποιεί το layout shapes.xml που είναι το Interface της εφαρμογής (Βλ. παράρτημα). Από το layout όρισα την περιοχή στην οποία ο χρήστης θα μπορεί να αγγίζει την οθόνη και να ζωγραφίζει μέσω της κλάσης PrimitiveShapes.java.

Στην εικόνα 11 φαίνεται ο απλοποιημένος κώδικας της κλάσης PrimitiveSpapes.java. Όπως φαίνεται χρησιμοποιείται η βιβλιοθήκη graphics και συγκεκριμένα οι κλάσεις Canvas και Paint. Ενώ η κλάση PrimitiveSpapes.java εκτείνει την κλάση View. Η κλάση Paint βοηθάει στον καθορισμό του χρώματος γραφής, το πάχος αλλά και το στυλ με το οποίο θα ζωγραφιστούν τα σχήματα.

```
package openglTutorial;

import android.view.View;
import android.view.View.OnTouchListener;
import android.graphics.Canvas;
import android.graphics.Paint;

public class PrimitiveShapes extends View implements OnTouchListener {

      List<Vertex> vertices = new ArrayList<Vertex>();
    public DrawView(Context context) {
        super(context);
        //Καθορισμός χρωμάτων και στοιχείων γραφής
    }

    public void onDraw(Canvas canvas) {

      choice = checkShapeChoice();
      if (choice == POINTS){
            for every vertex in vetices{
                  canvas.drawPoint(vertices);
            }
      printCode();
      }
      else if (choice == TRIANGLES){
            for every 3 vertex in vertices
                  canvas.drawLine(vertives[0],vertives[1]);
                  canvas.drawLine(vertives[1],vertives[2]);
                  canvas.drawLine(vertives[2],vertives[3]);
            }
      }
      printCode();
      /* παρομοίως και τα άλλα σχήματα */
    }

    public boolean onTouch(View view, MotionEvent event) {

             Vertex vextex = new Vertex();
             vertex.x = event.getX();
             vertex.y = event.getY();
             vertices.add(vertex);
    }

}
```

Εικόνα 11. Ο απλοποιημένος κώδικας της κλάσης PrimitiveShapes.java

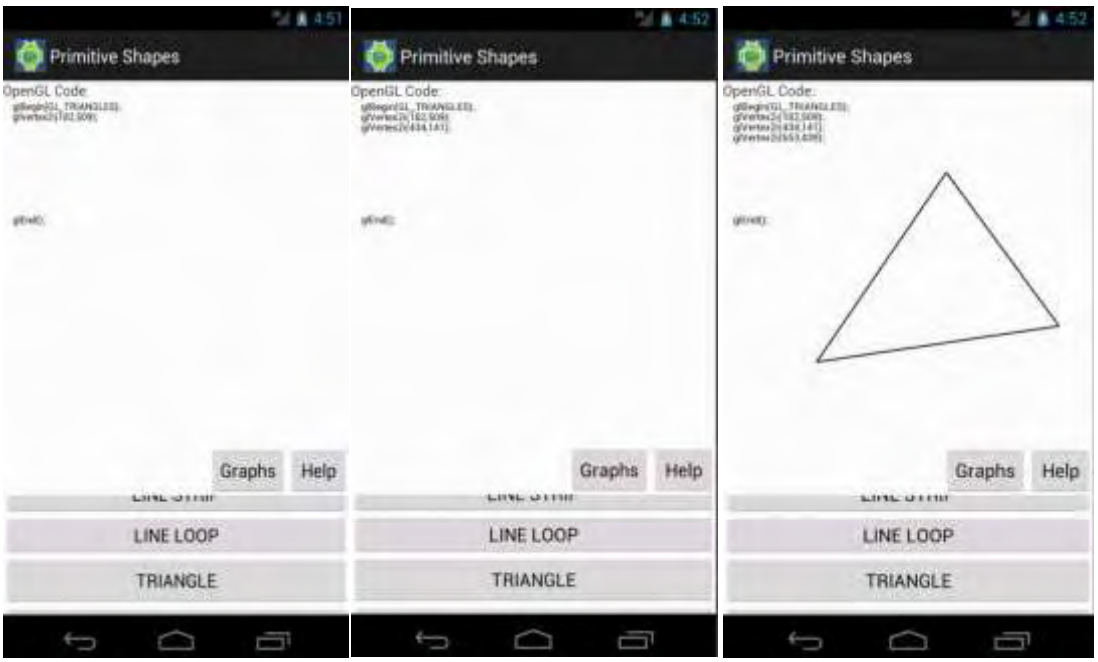
Στη συνέχεια παρατηρούμε τη μέθοδο onDraw (Canvas canvas), στην οποία γίνεται η ζωγραφική. Η μέθοδος αυτή καλείται συνέχεια και εμφανίζει στην οθόνη οτιδήποτε έχει ζωγραφιστεί με τη βοήθεια του αντικειμένου canvas.

Έτσι, όταν κάθε φορά ο χρήστης αγγίζει την οθόνη δημιουργεί μια κορυφή η οποία αποθηκεύεται σε μια λίστα. Η λίστα αυτή ανατρέχεται κάθε στιγμή μέσα στην onDraw() μέθοδο και ανάλογα με το πόσα στοιχεία έχει σε συνδυασμό με την επιλογή σχήματος που δόθηκε από το χρήστη ζωγραφίζεται το ανάλογο σχήμα.

Τα περισσότερα σχήματα δημιουργούνται με τμήματα γραμμών μεταξύ των κορυφών τους. Έτσι, για παράδειγμα, ένα τρίγωνο με κορυφές v_0, v_1, v_2 αποτελείται από τρεις γραμμές μεταξύ των κορυφών v_0-v_1,v_1-v_2,v_2-v_3 κ.ο.κ.

Στην περίπτωση που έχει επιλεχθεί ένα σχήμα το οποίο απαιτεί παραπάνω από μία κορυφές για να εμφανιστεί τότε στην οθόνη δεν εμφανίζεται τίποτα και ο χρήστης θα πρέπει να δώσει περισσότερες κορυφές.

Για παράδειγμα, αν ο χρήστης επιλέξει να ζωγραφίσει τρίγωνα επιλέγοντας το κουμπί Triangles θα πρέπει να αγγίξει 3 φορές την οθόνη για να δει το πρώτο τρίγωνο (Εικόνα 12).



Εικόνα 12. Σταδιακή δημιουργία ενός τριγώνου. Αρχικά δεν εμφανίζεται τίποτα μέχρι να συμπληρωθούν τρεις κορυφές

Όπως βλέπουμε στην εικόνα 12, ο κώδικας ανανεώνεται συνεχώς, ώστε να μπορεί ο χρήστης να παρακολουθεί βήμα βήμα την εξέλιξη του.

Κατά τον ίδιο τρόπο ζωγραφίζονται και τα υπόλοιπα γεωμετρικά σχήματα.

3.2.2.3 Παρατηρήσεις

Ο αριθμός των κορυφών που μπορεί να προσθέσει ο χρήστης είναι καθορισμένος για κάθε σχήμα. Έτσι, μετά από ένα αριθμό κορυφών ο πίνακας που κρατάει τις κορυφές αδειάζει, η οθόνη καθαρίζει και ο χρήστης μπορεί να εισάγει κορυφές από την αρχή.

Οι εντολές αφορούν τον δισδιάστατο χώρο, συνεπώς τα σχήματα ζωγραφίζονται στο επίπεδο όπου το z θεωρείται ίσο με το μηδέν.

4.Μετασχηματισμοί

4.1 Μετασχηματισμοί στην OpenGL

4.1.1 Δισδιάστατη Μετατόπιση (2-D Translation)

Μπορούμε να εκτελέσουμε μια μετατόπιση ενός σημείου προσθέτοντας σχετικές αποστάσεις στις συντεταγμένες ώστε να παράγουμε μια νέα θέση συντεταγμένων. Στην πράξη μετακινούμε την αρχική θέση του σημείου κατά μήκος του ίχνους ενός ευθύγραμμου τμήματος στη νέα του τοποθεσία. Ομοίως, μια μετατόπιση εφαρμόζεται σε ένα αντικείμενο που ορίζεται από πολλαπλές θέσεις συντεταγμένων με την ίδια μετατόπιση κατά μήκος παράλληλων ιχνών. Κατόπιν, ολόκληρο το αντικείμενο προβάλλεται στην νέα του τοποθεσία.

Για να μετατοπίσουμε μια δισδιάστατη θέση προσθέτουμε αποστάσεις μετατόπισης.

$x' = x + t_x$ και $y' = y + t_y$

Η μετατόπιση είναι ένας μετασχηματισμός στερεού σώματος που μετακινεί αντικείμενα χωρίς παραμόρφωση. Δηλαδή κάθε σημείο πάνω στο αντικείμενο μετατοπίζεται με την ίδια ποσότητα.

4.1.2 Δισδιάστατη Περιστροφή (2-D Rotation)

Ένας μετασχηματισμός περιστροφής ενός αντικειμένου παράγεται ορίζοντας έναν άξονα περιστροφής και μια γωνία περιστροφής. Μετά, όλα τα στοιχεία του αντικειμένου μετασχηματίζονται σε νέες θέσεις περιστρέφοντας τα σημεία κατά τη γωνία που ορίστηκε γύρω από τον άξονα περιστροφής.

Μια δισδιάστατη περιστροφή ενός αντικειμένου λαμβάνεται επανατοποθετώντας το αντικείμενο κατά μήκος του κυκλικού ίχνους στο επίπεδο xy. Σ' αυτή την περίπτωση περιστρέφουμε το αντικείμενο γύρω από έναν άξονα περιστροφής που είναι κάθετος στο επίπεδο xy. Οι παράμετροι της δισδιάστατης περιστροφής είναι η γωνία περιστροφής a και μια θέση (x_r, y_r) που ονομάζεται σημείο περιστροφής, γύρω από το οποίο το αντικείμενο θα περιστραφεί. Το σημείο περιστροφής είναι το σημείο τομής του άξονα περιστροφής με το επίπεδο xy.

Οι εξισώσεις που εκφράζουν τη δισδιάστατη περιστροφή είναι:

$x' = x\cos(a) - y\sin(a)$ και $y' = x\sin(a) + y\cos(a)$

Οι παραπάνω εξισώσεις μπορούν να εκφραστούν και με τη μορφή πίνακα:

$$R_a = \begin{bmatrix} \cos(a) & -\sin(a) \\ \sin(a) & \cos(a) \end{bmatrix}$$

4.1.3 Δισδιάστατη Κλιμάκωση (2-D Scaling)

Ο μετασχηματισμός κλιμάκωσης εφαρμόζεται για να αλλάξει το σχήμα ενός αντικειμένου. Μια απλή πράξη κλιμάκωσης δύο διαστάσεων εκτελείται πολλαπλασιάζοντας τις θέσεις (x,y) ενός αντικειμένου με παράγοντες κλιμάκωσης s_x ,s_y για να πάρουμε τις μετασχηματισμένες συντεταγμένες. Έτσι :

$x' = xs_x$ και $y' = ys_y$

ή σε μορφή πίνακα:

$$S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Οι παράγοντες κλιμάκωσης μπορούν να πάρουν οποιεσδήποτε θετικές τιμές. Τιμές μικρότερες του 1 μειώνουν το μέγεθος του αντικειμένου ενώ τιμές μεγαλύτερες του 1 παράγουν μεγεθύνεις. Το σχήμα παραμένει αμετάβλητο αν οι παράγοντες κλιμάκωσης πάρουν την τιμή 1. Αν οι συντελεστές έχουν την ίδια τιμή παράγεται ομοιόμορφη κλιμάκωση με αποτέλεσμα το σχήμα να διατηρεί τις αναλογίες του. Αν όμως οι τιμές είναι διαφορετικές, παράγεται διαφορική κλιμάκωση, η οποία έχει αρκετές εφαρμογές.

4.1.4 Τρισδιάστατη Μετατόπιση(3-D Translation)

Μια θέση P = (x,y,z) στον τρισδιάστατο χώρο μετατοπίζεται σε μια τοποθεσία

P' = (x',y',z') προσθέτοντας τις αποστάσεις μετατόπισης t_x, t_y , t_z . Έτσι:

$x' = x + x_t$, $y' = y + y_t$ και $z' = z + z_t$

Οι τρισδιάστατες πράξεις αυτές μπορούν να εκφραστούν σε μορφή πίνακα:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Ένα αντικείμενο μετατοπίζεται σε τρεις διαστάσεις μετασχηματίζοντας καθεμία από τις συντεταγμένες θέσεων που το ορίζουν, και μετά κατασκευάζοντας ξανά το αντικείμενο στη νέα τοποθεσία. Μετατοπίζεται κάθε κορυφή κάθε επιφάνειας ενός αντικειμένου που αναπαρίσταται ως ένα σύνολο από πολυγωνικές επιφάνειες και προβάλλονται ξανά οι πολυγωνικές έδρες στις μετατοπισμένες θέσεις.

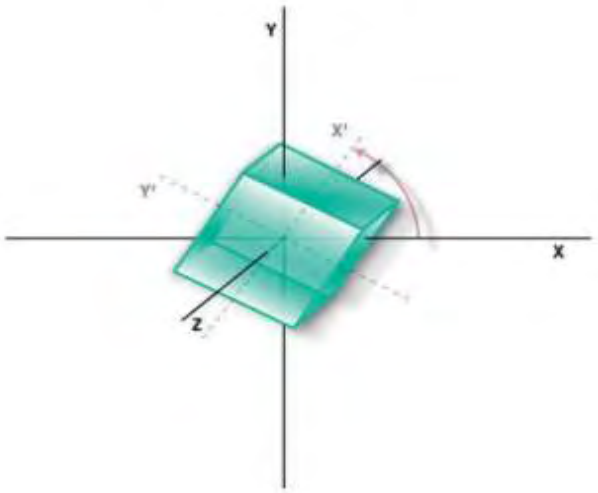
4.1.5 Τρισδιάστατη Περιστροφή (3-D Rotation)

Οι δισδιάστατες εξισώσεις περιστροφής γύρω από τον άξονα z (Εικόνα 13) επεκτείνονται εύκολα στις τρεις διαστάσεις:

$x' = \cos(a) - y\sin(a)$, $y' = x\sin(a) + y\cos(a)$ και $z' = z$

και σε μορφή πίνακα είναι:

$$R(z,a)=\begin{bmatrix}\cos(a) & -\sin(a) & 0 & 0\\ \sin(a) & \cos(a) & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$



Εικόνα 13.Περιστροφή γύρω απο τον άξονα z

Οι εξισώσεις περιστροφής γύρω από τον άξονα x είναι:

y’ = ycos(a) – zsin(a), z’ = ysin(a) + zcos(a) , x’ = x

και ο αντίστοιχος πίνακας:

$$R(x,a)=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & \cos(a) & -\sin(a) & 0\\ 0 & \sin(a) & \cos(a) & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

Οι εξισώσεις περιστροφής γύρω από τον άξονα y είναι:

z’ = z cos(a) – xsin(a), x’ = zsin(a) + xcos(a) , y’ = y

και ο αντίστοιχος πίνακας:

$$R(y,a)=\begin{bmatrix}\cos(a) & 0 & \sin(a) & 0\\ 0 & 1 & 0 & 0\\ -\sin(a) & 0 & \cos(a) & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

4.1.6 Τρισδιάστατη Κλιμάκωση(3-D Scaling)

Η έκφραση του μετασχηματισμού τρισδιάστατης κλιμάκωσης μιας θέσης P=(x,y,z) σε σχέση με την αρχή των συντεταγμένων είναι μια απλή επέκταση της δισδιάστατης κλιμάκωσης. Έτσι, ο πίνακας μετασχηματισμού κλιμάκωσης γίνεται:

$$S=\begin{bmatrix}S_x & 0 & 0 & 0\\ 0 & S_y & 0 & 0\\ 0 & 0 & S_z & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

4.1.7 Οι εντολές μετασχηματισμού στην OpenGL

Ένας πίνακας μετατόπισης 4 επί 4 κατασκευάζεται με την ακόλουθη ρουτίνα:

glTranslate*(tx, ty, tz);

Οι παράμετροι μετατόπισης tx, ty , tz μπορούν να πάρουν οποιεσδήποτε πραγματικές τιμές και η μία και μόνο κατάληξή μου επικολλάται σ' αυτή τη συνάρτηση είναι f (float) ή d (double). Για δισδιάστατες εφαρμογές θέτουμε την tz = 0.0. Ακόμη μια δισδιάστατη θέση αναπαρίσταται με έναν πίνακα-στήλη τεσσάρων στοιχείων, με το στοιχείο z ίσο με το 0. Ο πίνακας μετατόπισης που παράγεται από αυτή την συνάρτηση χρησιμοποιείται για να μετασχηματιστούν οι θέσεις αντικειμένων που ορίστηκαν μετά την κλήση αυτής της συνάρτησης. Για παράδειγμα, μετατοπίζουμε τις θέσεις συντεταγμένων που ορίστηκαν μετά από αυτή τη συνάρτηση 25 μονάδες στη φορά του x και – 10 μονάδες στη φορά του y με την εντολή:

glTranslatef(25.0, -10,0, 0.0);

Ομοίως, ένας πίνακας περιστροφής 4 επί 4 παράγεται με την εντολή:

glRotate*(theta, vx , vy, vz);

όπου το διάνυσμα v = (vx,vy,vz) μπορεί να δεχτεί οποιεσδήποτε τιμές κινητής υποδιαστολής στα στοιχεία του. Αυτό το διάνυσμα ορίζει τον προσανατολισμό ενός άξονα περιστροφής που περνάει από την αρχή των συντεταγμένων. Αν το v δεν ορίζεται ως μοναδιαίο διάνυσμα, κανονικοποιείται αυτόματα πριν υπολογιστούν τα στοιχεία του πίνακα περιστροφής. Η κατάληξη μπορεί να είναι είτε f είτε d και η παράμετρος theta παίρνει την γωνία περιστροφής σε μοίρες, την οποία η ρουτίνα μετατρέπει σε ακτίνα για τους τριγωνομετρικούς υπολογισμούς. Αυτή η συνάρτηση παράγει έναν πίνακα περιστροφής χρησιμοποιώντας τους υπολογισμούς του τετραδικού συστήματος που εφαρμόζεται σε θέσεις που ορίζονται μετά την κλήση αυτής της συνάρτησης. Για παράδειγμα, η εντολή

glRotatef(90.0,0.0,0.0,1.0);

κατασκευάζει τον πίνακα για μια περιστροφή 90 μοιρών γύρω από τον άξονα z.

Με την ακόλουθη ρουτίνα παίρνουμε ένα πίνακα κλιμάκωσης 4 επί 4 σε σχέση με την αρχή των συντεταγμένων.

glScale*(sx,,sy, sz);

Η κατάληξη είναι πάλι είτε f είτε d και οι παράμετροι κλιμακώσεις μπορούν να πάρουν οποιεσδήποτε πραγματικές τιμές. Συνεπώς, αυτή η συνάρτηση θα παράγει και αντανακλάσεις όταν δοθούν αρνητικές τιμές στις παραμέτρους κλιμάκωσης. Για παράδειγμα, η ακόλουθη εντολή παράγει έναν πίνακα που κλιμακώνει κατά δύο φορές στη φορά του x, κατά τρεις φορές στη φορά του y, και αντανακλά σε σχέση με τον άξονα x.

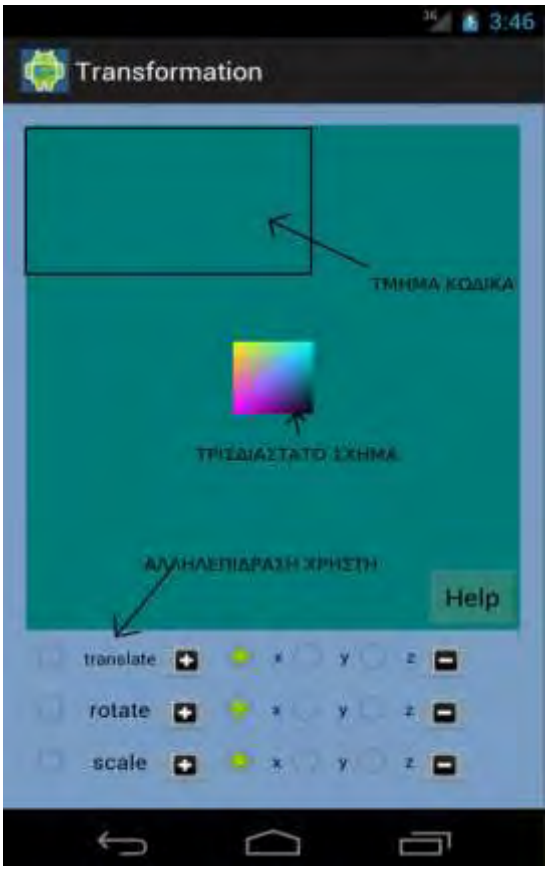
glScalef(2.0, -3.0 , 1,0);

4.2 Μετασχηματισμοί στην εφαρμογή

4.2.1 Περιγραφή του interface

Η ιδέα είναι να μπορεί ο χρήστης να εφαρμόζει τις παραπάνω εντολές σε κάποιο σχήμα έτσι ώστε να βλέπει μόνος του τι κάνει η κάθε εντολή μόνη της αλλά και όλες μαζί σε συνδυασμό. Επίσης, για να μπορούν να φαίνονται και τα αποτελέσματα στον τρισδιάστατο χώρο είναι ανάγκη για δημιουργία τρισδιάστατου σχήματος. Έτσι, το interface αποτελείται από το χώρο όπου φαίνονται οι εντολές, το τρισδιάστατο πλάνο με την ύπαρξη ενός πολύχρωμου κύβου, κουμπί βοήθειας για

επεξήγηση των εντολών και φυσικά ο χώρος στον οποίον ο χρήστης μπορεί να αλληλεπιδράσει με την εφαρμογή, δηλαδή τα κουμπιά με τα οποία μπορεί να εισάγει εντολές κι αν αλλάξει τις τιμές των ορισμάτων τους (Εικόνα 14).



Εικόνα 14. Το interface των μετασχηματισμών.

4.2.2 Υλοποίηση

Για την υλοποίηση αυτού του μέρους χρειάστηκε η επιστράτευση της OpenGL ES. Αρχικά, δημιούργησα την κλάση TransformationsActivity.java, η οποία δημιουργεί το activity για τη συγκεκριμένη ενέργεια (βλ. παράρτημα). Η κλάση αυτή είναι υπεύθυνη για το χειρισμό των κουμπιών, καθώς και την αρχικοποίηση του layout που αποτελεί το interface στο αρχείο transformations.xml. Στο αρχείο αυτό, για τη δημιουργία του τρισδιάστατου χώρου, χρησιμοποιήθηκε ως όρισμα η κλάση TransformationsGLSurfaceView.java, η οποία περιέχει τον κώδικα της OpenGL ES.

Στην εικόνα 15 φαίνεται ο απλοποιημένος κώδικας της κλάσης TransformationsGLSurfaceView.java.

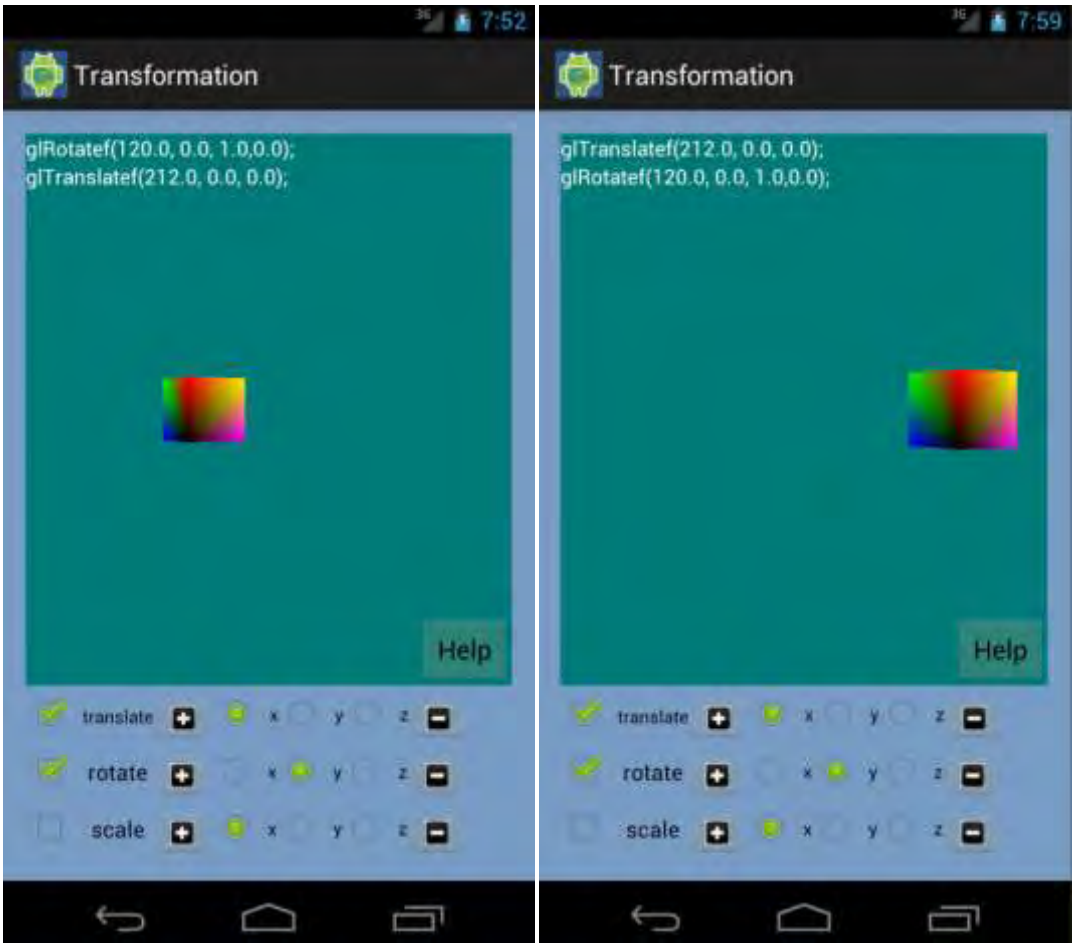
```
package openglTutorial;

public class TransformationsGLSurfaceView extends GLSurfaceView implements
Renderer {
	Cube c;
	public TransformationsGLSurfaceView(Context context) {
		super(context);
		this.context = context;
		c = new Cube();
	}
	public void onSurfaceCreated(GL10 gl, EGLConfig config) {
		// διάφορες αρχικοποιήσεις
	}
	public void onDrawFrame(GL10 gl) {

		choice = getUserData();
		if (choice == TRANSLATE){
			gl.glTranslatef(curX, curY, curZ);
		}
		if (choice == ROTATE){
			gl.glRotatef(angle,rotX,rotY,rotZ);
		}
		if (choice == SCALE){
			gl.glScalef(scalX,scalY,scalZ);
		}
		printCode();
		cube.draw(); // εμφάνιση του σχήματος
	}

	public void onSurfaceChanged(GL10 gl, int width, int height) {
		gl.glViewport(0, 0, width, height); // Reset The Current Viewport
		GLU.gluPerspective(gl, fov_degrees, aspect, camZ / 10, camZ * 10);
		GLU.gluLookAt(gl, 0, 0, camZ, 0, 0, 0, 0, 1, 0); // move camera back
	}


}
```

Εικόνα 15. Ο απλοποιημένος κώδικας της TransformationsGLSurfaceVeiw.java

Παρατηρούμε ότι για τη δημιουργία OpenGL ES προγραμμάτων απαιτούνται 3 βασικές μέθοδοι. Η onSurfaceCreated() περιέχει κάποιες αρχικοποιήσεις που δεν παίζουν κάποιο ρόλο σ' αυτό το στάδιο. Η onSurfaceChanged() περιλαμβάνει κάποιες ρυθμίσεις για την κάμερα και τον τρισδιάστατο χώρο που θα δούμε αναλυτικότερα στο επόμενο κεφάλαιο. Η σημαντικότερη από όλες είναι η onDrawFrame(GL10 gl) στην οποία γίνονται οι μετασχηματισμοί και εμφανίζεται ο κύβος στο χώρο. Παρατηρούμε ότι οι εντολές μετασχηματισμού στην OpenGL ES είναι ακριβώς ίδιες με αυτές της OpenGL. Ο κύβος ζωγραφίζεται με τη μέθοδο draw(), ενώ η κλάση Cube.java περιέχει τη δημιουργία του κύβου σε OpenGL ES περιβάλλον (βλ. Παράρτημα, κλάση My3DTexture.java) που διαφέρει από τον απλοϊκό τρόπο δημιουργίας σχημάτων στην OpenGL.

Όπως βλέπουμε κάθε φορά που ο χρήστης επιλέγει μια εντολή, αυτή εφαρμόζεται στον κύβο με τις παραμέτρους που αυτός δίνει. Όταν ο χρήστης αναιρεί την εντολή αυτή δεν εφαρμόζεται πια και γυρνάει τον κύβο στην αρχική του κατάσταση. Επιπρόσθετα, μετά από κάθε βήμα εμφανίζονται (ή εξαφανίζονται) οι εντολές με τα ορίσματα που έχουν πάρει από το χρήστη έτσι ώστε αυτός να μπορεί να τις δει.

Σημαντικό ρόλο παίζει η σειρά με την οποία δίνονται οι εντολές καθώς μια πιθανή μετατόπιση ακολουθούμενη από μια περιστροφή έχει διαφορετικά αποτελέσματα από την αντίθετη διαδικασία (Εικόνα 16).



Εικόνα 16.Η περιστροφή και μετατόπιση έχει διαφορετικά αποτελέσματα από την αντίθετη διαδικασία (μετατόπιση και μετά περιστροφή), όπως φαίνεται παραπάνω.

Επειδή ο χώρος στην οθόνη του κινητού είναι περιορισμένος η εντολή rotate είναι υλοποιημένη, ώστε να περιστρέφει τον κύβο γύρω από κάποιον από τους βασικούς άξονες περιστροφής. Συνεπώς, αν ο χρήστης επιλέξει τον άξονα y και στη συνέχεια περιστρέψει κατά 60 μοίρες τον κύβο, παράγεται η εντολή *glRotatef(60.0,0.0,1.0,0.0);*, αν αμέσως μετά επιλεχθεί ο άξονας x τότε ο κύβος μετατοπίζεται αυτόματα 60 μοίρες γύρω από τον άξονα, καθώς η εντολή γίνεται: glRotatef(60.0,1.0,0.0,0.0);

Επιπρόσθετα, δεν υπάρχει η δυνατότητα να χρησιμοποιηθεί η ίδια εντολή περισσότερες από μία φορές. Ο λόγος που ισχύει η προηγούμενη πρόταση βασίζεται στην απλότητα που ήθελα να διατηρηθεί σε όλη την εφαρμογή, καθώς σκοπός της εφαρμογής είναι η επίδειξη των εντολών της OpenGL και όχι οι πολύπλοκοι μετασχηματισμοί.

Έτσι, ο χρήστης μπορεί να χρησιμοποιήσει τις βασικές εντολές μετασχηματισμού και να βγάλει μόνος του τα δικά του συμπεράσματα για το τι μπορεί να κάνει η κάθε μία (Εικόνα 17).

Εικόνα 17. Χρήση και των 3 εντολών μετασχηματισμού.

5. Θέαση

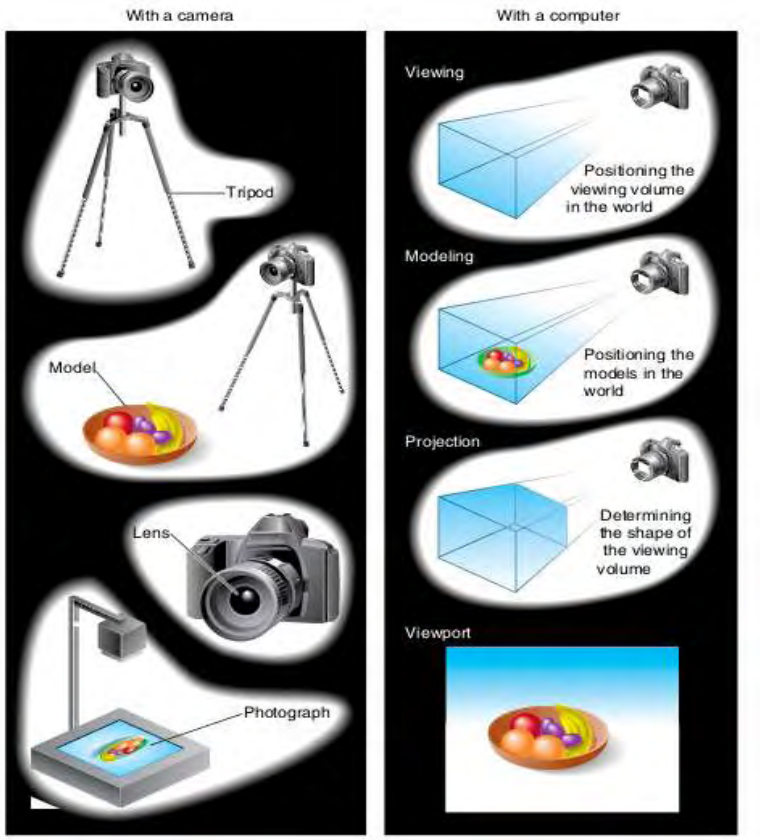
Επειδή η δισδιάστατη θέαση αποτελεί υποπερίπτωση της τρισδιάστατης, η εργασία μου αναφέρεται στην τελευταία. Τον ίδιο κανόνα ακολουθεί και η εφαρμογή, η οποία χρησιμοποιεί ρουτίνες τρισδιάστατης θέασης για την εμφάνιση των αντικειμένων και του πλάνου.

5.1 Τρισδιάστατη θέαση στην OpenGL

Σε αντίθεση με την εικόνα μιας κάμερας, μπορούμε να επιλέξουμε διαφορετικές μεθόδους προβολής μιας σκηνής πάνω στο επίπεδο θέασης. Μια μέθοδος για να βάλουμε την περιγραφή ενός στέρεου σώματος πάνω σε ένα επίπεδο θέασης είναι να προβάλλουμε σημεία της επιφάνειας του αντικειμένου κατά μήκος παράλληλων γραμμών. Αυτή η τεχνική, που ονομάζεται *παράλληλη προβολή*(*parallel projection*), χρησιμοποιείται σε σχέδια μηχανολογίας και αρχιτεκτονικής για να αναπαραστήσει ένα αντικείμενο με ένα σύνολο από όψεις που δείχνουν τις ακριβείς διαστάσεις του αντικειμένου.

Μια άλλη μέθοδος παραγωγής μιας όψης τρισδιάστατης σκηνής είναι να προβάλλουμε σημεία σε ένα επίπεδο θέασης κατά μήκος ιχνών που συγκλίνουν. Αυτή η διαδικασία που ονομάζεται *προοπτική προβολή (perspective projection*), έχει ως αποτέλεσμα τα αντικείμενα που βρίσκονται μακρύτερα από τη θέση θέασης να προβάλλονται μικρότερα από τα αντικείμενα ίδιου μεγέθους που βρίσκονται πλησιέστερα στη θέση θέασης. Μια σκηνή που παράγεται χρησιμοποιώντας μια προοπτική προβολή φαίνεται πιο ρεαλιστική, αφού αυτός είναι και ο τρόπος που τα μάτια μας και οι φακοί μιας κάμερας σχηματίζουν εικόνες.

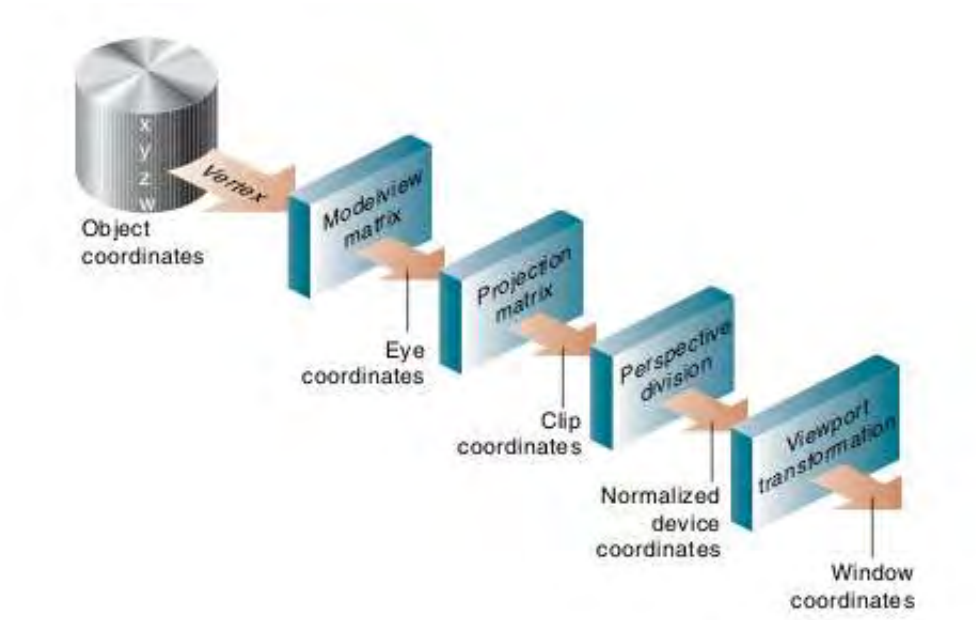
Οι διαδικασίες παραγωγής μιας απεικόνισης γραφικών μιας τρισδιάστατης σκηνής είναι κάπως ανάλογες με τις διαδικασίες που σχετίζονται με τη λήψη μιας φωτογραφίας. Πρώτα απ’ όλα, χρειάζεται να επιλέξουμε μια θέση θέασης που αντιστοιχεί με το πού θα θέλαμε να τοποθετήσουμε μια κάμερα. Μετά, πρέπει να αποφασίσουμε για τον προσανατολισμό της κάμερας. Τέλος, όταν κλείνουμε τον φωτοφράχτη, η σκηνή κόβεται στο μέγεθος ενός επιλεγμένου παραθύρου αποκοπής, το οποίο αντιστοιχεί στο διάφραγμα ή τον τύπο του φακού μιας κάμερας και το φως από τις ορατές επιφάνειες προβάλλεται πάνω στο φιλμ της κάμερας (Εικόνα 18).



Εικόνα 18. Αναλογία με φωτογραφική μηχανή.

Στον υπολογιστή ο τρόπος με τον οποίο μετασχηματίζεται μια τρισδιάστατη σκηνή σε συντεταγμένες συσκευής είναι ο ακόλουθος: Αφού έχει μοντελοποιηθεί μια σκηνή σε παγκόσμιες συντεταγμένες, επιλέγεται ένα σύστημα συντεταγμένων θέασης και η περιγραφή της σκηνής μετατρέπεται σε συντεταγμένες θέασης. Το σύστημα συντεταγμένων θέασης ορίζει τις παραμέτρους θέασης, συμπεριλαμβανομένων της θέσης και του προσανατολισμού του επιπέδου προβολής, το οποίο μπορούμε να δούμε ως το επίπεδο του φιλμ της κάμερας. Ένα δισδιάστατο παράθυρο αποκοπής, που αντιστοιχεί σε έναν επιλεγμένο φακό κάμερας, ορίζεται πάνω στο επίπεδο προβολής και δημιουργείται μια τρισδιάστατη περιοχή αποκοπής. Αυτή η περιοχή αποκοπής ονομάζεται όγκος θέασης, και το σχήμα και μέγεθός του εξαρτάται από τις διαστάσεις του παραθύρου αποκοπής, τον τύπο της προβολής που επιλέγουμε, και τις επιλεγμένες θέσεις περιορισμού κατά μήκος της διεύθυνσης θέασης. Οι πράξεις προβολής εκτελούνται για να μετατρέψουμε την περιγραφή συντεταγμένων θέασης της σκηνής σε θέσεις συντεταγμένων πάνω στο επίπεδο προβολής. Τα αντικείμενα χαρτογραφούνται σε κανονικές συντεταγμένες και όλα τα μέρη της σκηνής εκτός του όγκου θέασης ψαλιδίζονται. Οι πράξεις αποκοπής μπορούν να εφαρμοστούν αφού έχουν ολοκληρωθεί όλοι οι μετασχηματισμοί συντεταγμένων που είναι ανεξάρτητοι συσκευής. Με αυτόν τον τρόπο μπορούν να συνενωθούν οι μετασχηματισμοί

συντεταγμένων για μέγιστη απόδοση. Το τελικό βήμα είναι να χαρτογραφήσουμε τις συντεταγμένες του παραθύρου άποψης σε συντεταγμένες συσκευής μέσα σε ένα επιλεγμένο παράθυρο παρουσίασης (Εικόνα 19).



Εικόνα 19. Στάδια τρισδιάστατων μετασχηματισμών.

5.2 Συναρτήσεις θέασης στην OpenGL και OpenGL ES

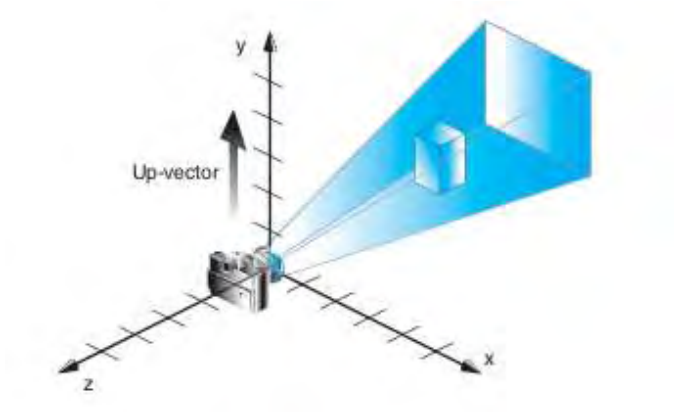
5.2.1 Η συνάρτηση μετασχηματισμού θέασης

Οι παράμετροι θέασης ορίζονται με την ακόλουθη συνάρτηση της GLU, η οποία βρίσκεται στη βιβλιοθήκη OpenGL Utility, διότι καλεί τις ρουτίνες μετατόπισης και περιστροφής στη βασική βιβλιοθήκη της OpenGL.

gluLookAt(x0 , y0 ,z0 , xref, yref, zref , Vx , Vy, Vz);

Οι τιμές όλων των παραμέτρων σε αυτήν τη συνάρτηση πρέπει να είναι τιμές κινητής υποδιαστολής διπλής ακρίβειας. Αυτή η συνάρτηση ορίζει την αρχή του πλαισίου αναφοράς θέασης ως τη θέση παγκόσμιων συντεταγμένων P_0 =(x0,y0,z0), τη θέση αναφοράς ως P_{ref} = (xref, yref,zref) και το ανοδικό διάνυσμα θέασης ως V = (Vx, Vy, Vz). Ο θετικός άξονας z_{view} για το πλαίσιο θέασης είναι στη φορά N = P_0 – P_{ref}. Το διάνυσμα P_0 εκφράζει τη θέση της κάμερας, το P_{ref} το κέντρο της θέασης και το V τον προσανατολισμό της κάμερας.

Αν δεν καλέσουμε τη συνάρτηση gluLookAt, οι τρέχουσες παράμετροι θέασης είναι P_0 =(0,0,0) , P_{ref} =(0,0,-1) ,V =(0,1,0) (Εικόνα 20).



Εικόνα 20. Προκαθορισμένες παράμετροι θέασης στην gluLookAt().

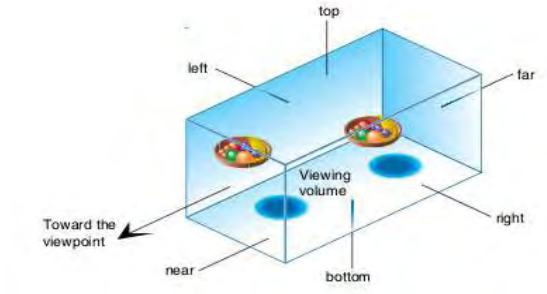
5.2.2 Συνάρτηση ορθογώνια προβολής

Οι παράμετροι ορθογώνιας προβολής επιλέγονται με τη συνάρτηση:

glOrtho(left, right ,bottom, top, near , far);

Όλες οι τιμές των παραμέτρων σε αυτή τη συνάρτηση πρέπει να είναι αριθμοί κινητής υποδιαστολής διπλής ακρίβειας. Χρησιμοποιούμε την παραπάνω εντολή για να επιλέξουμε τις συντεταγμένες του παραθύρου αποκοπής και τις αποστάσεις των κοντινών και μακρινών επιπέδων από την αρχή θέασης. Το κοντινό επίπεδο αποκοπής είναι, επίσης, το επίπεδο θέασης και συνεπώς, το παράθυρο αποκοπής είναι πάντοτε πάνω στο κοντινό επίπεδο του όγκου θέασης.

Η συνάρτηση glOrtho παράγει μια παράλληλη προβολή που είναι κάθετη στο επίπεδο θέασης (το κοντινό επίπεδο αποκοπής). Έτσι, αυτή η συνάρτηση δημιουργεί έναν πεπερασμένο όγκο θέασης ορθογώνιας προβολής για τα επίπεδα και το παράθυρο αποκοπής που ορίζονται (Εικόνα 21).



Εικόνα 21. Ο όγκος θέασης της glOrtho().

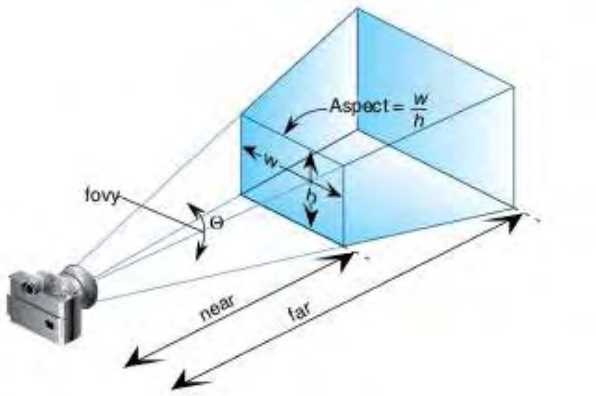
Στην OpenGL, τα κοντινά και μακρινά επίπεδα αποκοπής δεν είναι προαιρετικά, πρέπει πάντοτε να ορίζονται για οποιονδήποτε μετασχηματισμό προβολής. Οι παράμετροι near και far υποδηλώνουν αποστάσεις στην αρνητική διεύθυνση z από την αρχή των συντεταγμένων θέασης. Μία αρνητική τιμή για οποιαδήποτε παράμετρο δείχνει μια απόσταση πίσω από την αρχή θέασης, κατά μήκος του θετικού άξονα z. Μπορούμε να θέσουμε οποιεσδήποτε τιμές σε αυτές τις παραμέτρους εφόσον near < far.

5.2.3 Συνάρτηση συμμετρικής προοπτικής προβολής

Ένας συμμετρικός όγκος θέασης (κόλουρος κώνος) προοπτικής προβολής δημιουργείται με την ακόλουθη συνάρτηση της GLU.

gluPerspective(theta , aspect, near, far);

Με καθεμιά από τις τέσσερις παραμέτρους να παίρνει αριθμό κινητής υποδιαστολής διπλής ακρίβειας. Οι πρώτες δύο παράμετροι ορίζουν το μέγεθος και τη θέση του παραθύρου αποκοπής πάνω στο κοντινό επίπεδο, οι άλλες δύο παράμετροι ορίζουν τις αποστάσεις από το σημείο θέασης στο κοντινό και μακρινό επίπεδο αποκοπής. Η παράμετρος theta αναπαριστά τη γωνία του οπτικού πεδίου. Η παράμετρος aspect παίρνει μια τιμή για το λόγο διαστάσεων (width / height) του παραθύρου αποκοπής (Εικόνα 22).



Εικόνα 22. Όγκος προοπτικής προβολής από την gluPerspective().

Τα κοντινά και μακρινά επίπεδα μιας προοπτικής προβολής στην OpenGL πρέπει πάντοτε να βρίσκονται κάπου πάνω στον αρνητικό άξονα z, ενώ κανένα δεν μπορεί να βρίσκεται πίσω από τη θέση θέασης. Αυτός ο περιορισμός δεν ισχύει σε μια ορθογώνια προβολή, αλλά αποκλείει την ανάστροφη προοπτική προβολή ενός αντικειμένου όταν το επίπεδο θέασης βρίσκεται πίσω από το σημείο θέασης. Συνεπώς, και τα δύο near και far πρέπει να πάρουν θετικές τιμές. Αν δεν ορίσουμε

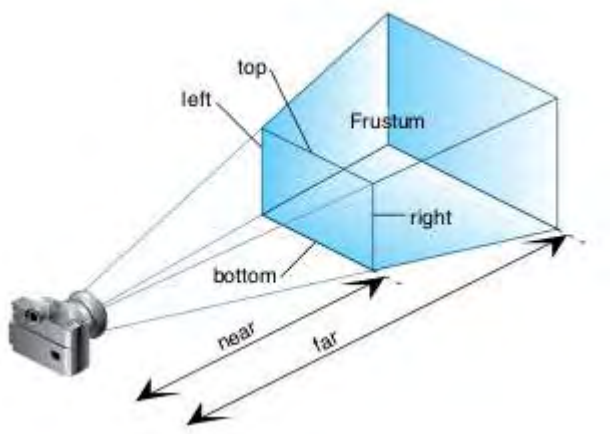
μια συνάρτηση προβολής η σκηνή μας προβάλλεται χρησιμοποιώντας την τρέχουσα ορθογώνια προβολή.

5.2.4 Γενική συνάρτηση προοπτικής προβολής

Μπορούμε να χρησιμοποιήσουμε την ακόλουθη συνάρτηση για να ορίσουμε μια προοπτική προβολή που έχει είτε ένα συμμετρικό είτε έναν πλάγιο κόλουρο κώνο ορατότητας.

glFrustum(left, right, bottom, top , near, far) ;

Αυτή η συνάρτηση έχει τις ίδιες παραμέτρους με τη συνάρτηση ορθογώνιας παράλληλης προβολής, αλλά τώρα οι αποστάσεις των κοντινών και μακρινών επίπεδων αποκοπής πρέπει να είναι θετικές. Οι πρώτες τέσσερις παράμετροι θέτουν τις συντεταγμένες του παραθύρου αποκοπής πάνω στο κοντινό επίπεδο, οι τελευταίες δυο παράμετροι ορίζουν τις αποστάσεις από την αρχή των συντεταγμένων προς τα κοντινά και μακρινά επίπεδα αποκοπής κατά μήκος του αρνητικού άξονα z (Εικόνα 23). Αν δεν καλέσουμε ρητά την εντολή προβολής, η OpenGL εφαρμόζει την τρέχουσα ορθογώνια προβολή στη σκηνή. Ο όγκος θέασης σ' αυτήν την περίπτωση είναι ο συμμετρικός κύβος.



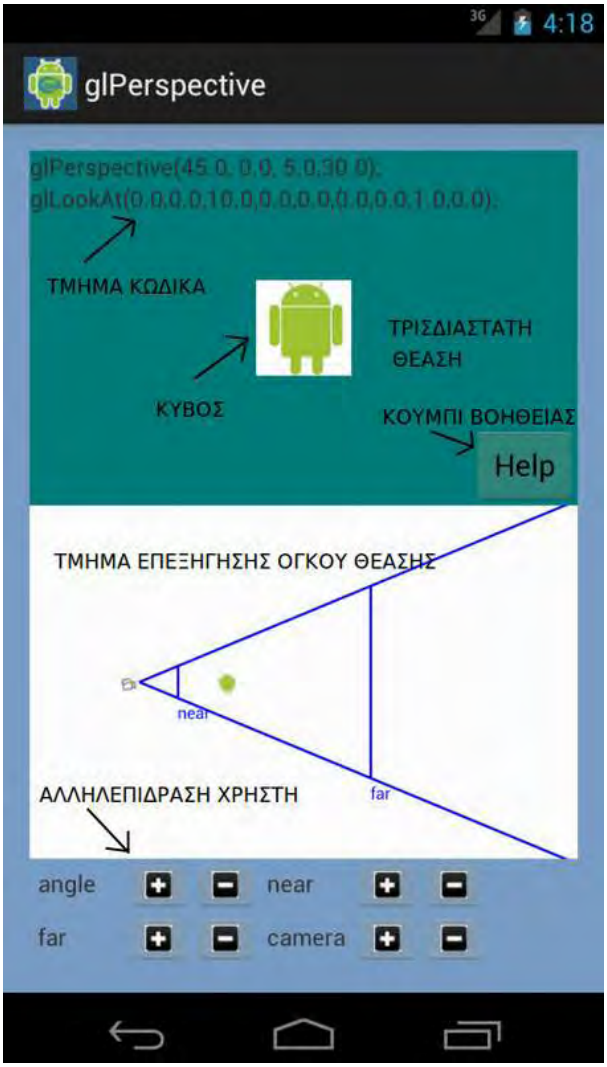
Εικόνα 23. Όγκος προοπτικής προβολής από την glFrustum().

5.3 Θέαση στην εφαρμογή

Η εφαρμογή χρησιμοποιεί τις τρισδιάστατες συναρτήσεις θέασης της OpenGL ES που είδαμε παραπάνω. Για την επίδειξη και των τριών συναρτήσεών θέασης δημιουργήθηκαν τρία διαφορετικά κουμπιά στο μενού με το καθένα να αντιπροσωπεύει την αντίστοιχη συνάρτηση.

5.3.1 Περιγραφή του interface

Όπως και στα προηγούμενα μέρη το interface αποτελείται από το χώρο θέασης της τρισδιάστατης προβολής, το τμήμα του κώδικα σε OpenGL , το κουμπί βοήθειας για πληροφορίες σχετικές με την εντολή και το μέρος όπου ο χρήστης μπορεί να αλληλεπιδράσει με την εφαρμογή πατώντας κουμπιά και δίνοντας τα αντίστοιχα ορίσματα στις συναρτήσεις. Η διαφορά σε αυτό το μέρος είναι ότι υπάρχει και ένα τμήμα που προσπαθεί να εξηγήσει στο χρήστη τον όγκο θέασης της εκάστοτε συνάρτησης (Εικόνα 24).



Εικόνα 24. Το interface για την εντολή glPerspective.

5.3.2 Υλοποίηση

5.3.2.1 glPerspective()

Η κλάση PerpectiveActivity.java είναι το activity για το μέρος αυτό και περιλαμβάνει το χειρισμό των κουμπιών αλλά και την αρχικοποίηση του layout perspective.xml που αποτελεί το παραπάνω interface (βλ. παράρτημα). Το perspective.xml με τη σειρά του ορίζει το χώρο τρισδιάστατης προβολής μέσω της κλάσης ViewingGLSurfaceView.java που περιλαμβάνει τον κώδικα OpenGL ES αλλά και το χώρο επεξήγησης του όγκου προοπτικής προβολής μέσω της κλάσης ExplainPerspective.java, η οποία χρησιμοποίει τη βιβλιοθήκη graphics του android.

Στην εικόνα 22 φαίνεται ο απλοποιημένος κώδικας της ViewingGLSurfaceView.java.

```
package openglTutorial;

public class ViewingGLSurfaceView extends GLSurfaceView implements Renderer {
      TexturedCube cube;
      public ViewingGLSurfaceView (Context context) { ext) {
            super(context);
            this.context = context;
            cube = new TexturedCube();
      }
      public void onSurfaceCreated(GL10 gl, EGLConfig config) {
            // διάφορες αρχικοποιήσεις
      }
      public void onDrawFrame(GL10 gl) {


            if(Frustrum){
                  gl.glFrustumf(left, right, bottom,  top, near, far);


            }else if (Ortho){
                  gl.glOrthof(left, right, bottom,  top, near, far);


            }
            else if (Perspective){
                  GLU.gluPerspective(gl, theta, aspect, near,  far);

            }
            printCode();
            cube.draw(); // εμφάνιση του σχήματος
      }

      public void onSurfaceChanged(GL10 gl, int width, int height) {
            gl.glViewport(0, 0, width, height);
      }


}
```

Εικόνα 25. Ο απλοποιημένος κώδικας της ViewingGLSurfaceView.java.

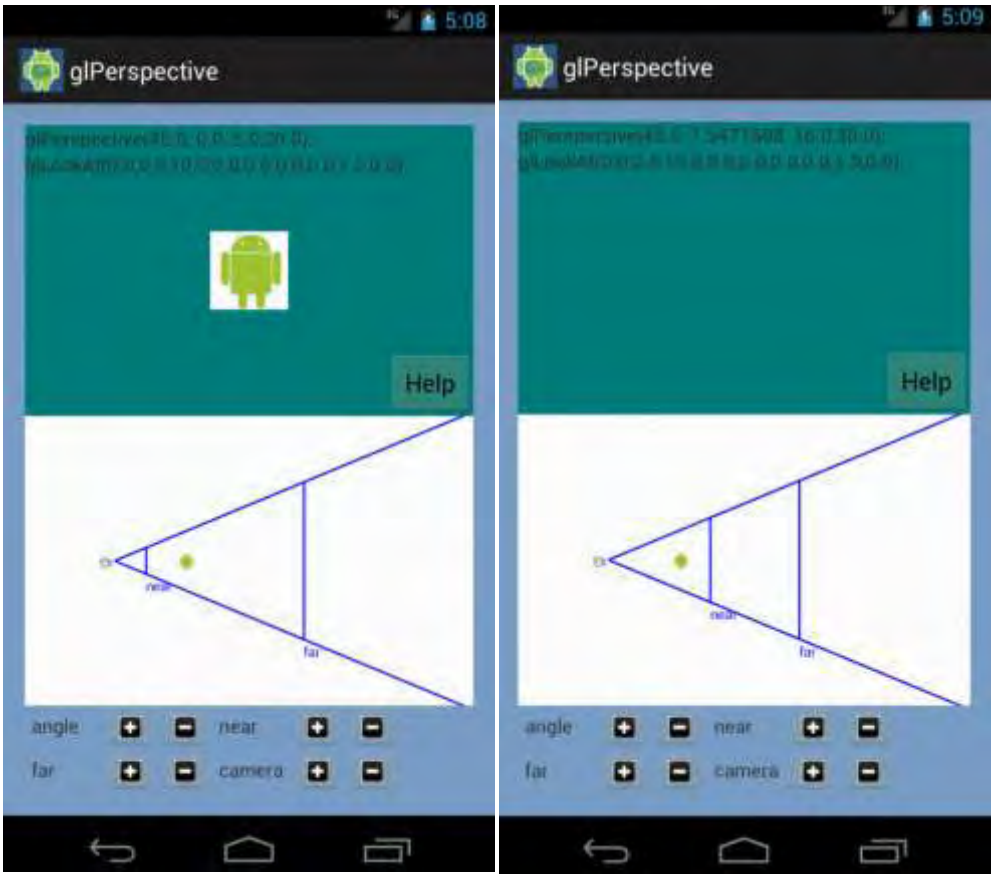
Όπως παρατηρούμε η κλάση αυτή δημιουργεί τον τρισδιάστατο χώρο και για τις τρεις εντολές μέσα στην onDrawFrame() και όχι μέσα στην onSurfaceChanged() που

είδαμε σε προηγούμενα μέρη της εφαρμογής. Αυτό γίνεται γιατί η on DrawFrame() καλείται συνέχεια και μπορεί να αλλάζει δυναμικά όταν ο χρήστης επιλέγει άλλη εντολή. Από τα παραπάνω βλέπουμε ότι η κλάση για την εμφάνιση του χώρου και του κύβου είναι μία σε αντίθεση με τις κλάσεις για το activity, το layout και την επεξήγηση του χώρου , οι οποίες είναι τρεις για κάθε ένα απ' αυτά.

Το μόνο που γίνεται στην onSurfaceChanged() είναι να θέσουμε το λόγο των διαστάσεων του παραθύρου άποψης ίσο με το λόγο των διαστάσεων του παραθύρου αποκοπής μέσω της εντολής glViewport(0,0,width,height);.

Έτσι, αν ο χρήστης πατήσει το κουμπί της perspective προβολής, θα δει το πλάνο σύμφωνα με την εντολή αυτή.

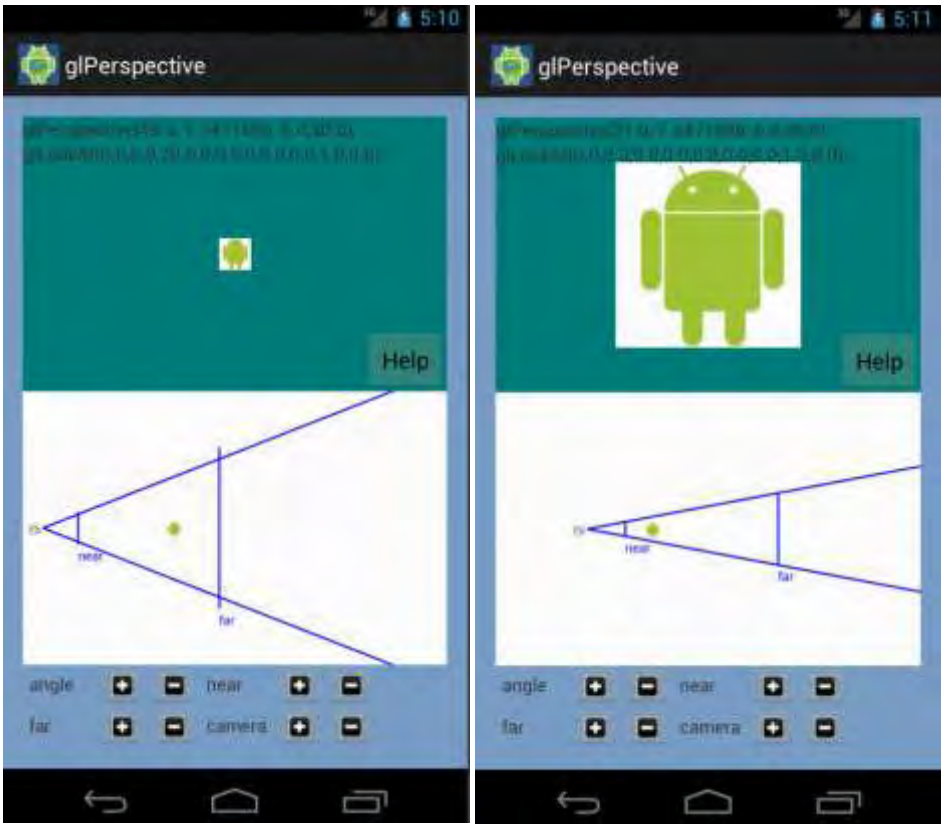
Η επεξήγηση του όγκου θέασης που γίνεται στην κλάση ExplainPerspective.java και χρησιμοποιεί τη βιβλιοθήκη graphics είναι απλά μια ζωγραφική (όπως στη ζωγραφική βασικών σχημάτων στην ενότητα 3) του όγκου θέασης και προσπαθεί να επεξηγήσει τα near και far πλάνα αποκοπής. Αν, για παράδειγμα, ο κύβος μας είναι εκτός των near και far τότε αποκόπτεται (Εικόνα 26).



Εικόνα 26. Το σχήμα είναι εντός των κοντινού και μακρινού πλάνου αποκοπής και φαίνεται (αριστερά). Το σχήμα είναι εκτός των πλάνων αποκοπής και αποκόπτεται (δεξιά).

Η επεξήγηση τους όγκου θέασης αποτελείται από ένα εικονίδιο κάμερας που αντιπροσωπεύει το σημείο θέασης, ένα εικονίδιο κύβου που χρησιμοποιείται για να δείξει τη θέση του κύβου στο τρισδιάστατο επίπεδο, καθώς και οι δύο γραμμές που ανάλογα τη γωνία, δείχνουν στο χρήστη τον κώνο που αποτελεί τον όγκο θέασης, αλλά και άλλες δύο γραμμές που συμβολίζουν τα near και far πλάνα αποκοπής.

Όπως βλέπουμε από την Εικόνα 26 ο χρήστης έχει τη δυνατότητα να μετακινήσει την κάμερα κατά μήκος του άξονα y, μέσω της εντολής glLookAt() που είδαμε προηγουμένως, και συνεπώς, ο κύβος να φαίνεται μικρότερος ή μεγαλύτερος. Επίσης, ο χρήστης μπορεί να πειράξει τη γωνία της προοπτικής θέασης και να δει τα αντίστοιχα αποτελέσματα (Εικόνα 27).



Εικόνα 27. Απομάκρυνση της κάμερας και μεγέθυνση της γωνίας θέασης έχει ως αποτέλεσμα ο κύβος να φαίνεται μικρός (αριστερά). Η κάμερα κινείται πιο κοντά στον κύβο και η γωνία θέασης μικραίνει με αποτέλεσμα ο κύβος να φαίνεται μεγαλύτερος (δεξιά).

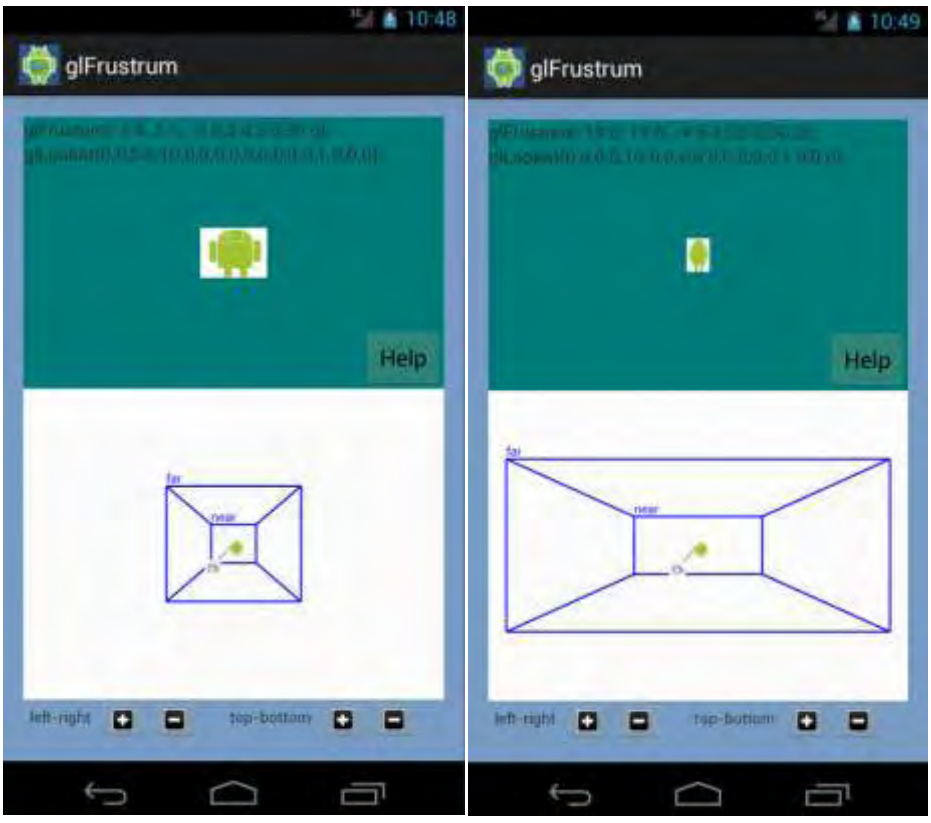
5.3.2.2 glFrustum()

Όπως είδαμε και πριν, η εντολή glFrustum() δημιουργεί προοπτικές προβολές και είναι γενικότερη από την glPerspective(). Η διαδικασία που ακολουθείται είναι παρόμοια με προηγουμένως. Η κλάση FrustrumActivity.java είναι το activity για το

μέρος αυτό και περιλαμβάνει το χειρισμό των κουμπιών, αλλά και την αρχικοποίηση του layout frustum.xml που αποτελεί το interface (βλ. παράρτημα). Το frustum.xml με τη σειρά του ορίζει το χώρο τρισδιάστατης προβολής μέσω της κλάσης ViewingGLSurfaceView.java που περιλαμβάνει τον κώδικα OpenGL ES, αλλά και το χώρο επεξήγησης του όγκου προοπτικής προβολής μέσω της κλάσης ExplainFrustum.java, η οποία χρησιμοποιεί τη βιβλιοθήκη graphics του android.

Όπως είδαμε, παραπάνω χρησιμοποιείται η ίδια κλάση για την τρισδιάστατη θέαση αλλά αυτή τη φορά επιλέγεται η εντολή glFrustum() με τις παραμέτρους που δίνει ο χρήστης. Αυτό έχει ως αποτέλεσμα τη δημιουργία ενός κόλουρου κώνου θέασης.

Αυτό που διαφέρει είναι ο τρόπος που γίνεται η επεξήγηση για τον όγκο της προοπτικής προβολής που δημιουργεί η glFrustum() (Εικόνα 23). Ενώ στην προηγούμενη κλάση επεξήγησης ο χρήστης έβλεπε τον όγκο θέασης από πλάγια τώρα τον βλέπει από μπροστά δίνοντας έμφαση στις παραμέτρους left, right, bottom και top, έτσι ώστε να φαίνονται εύκολα τα αποτελέσματα από την αλλαγή των παραμέτρων αυτών (Εικόνα 28).



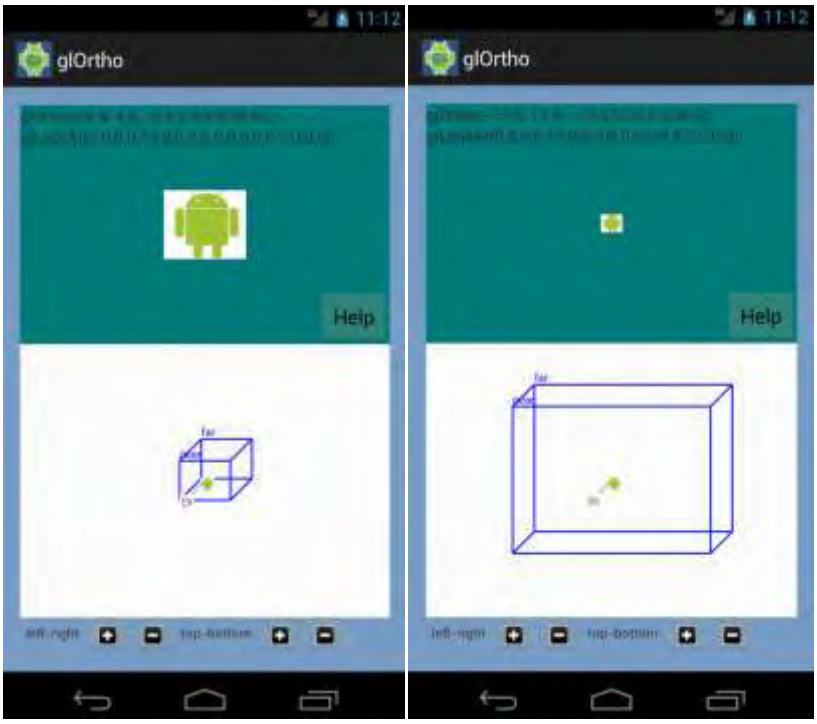
Εικόνα 28. Κανονική εμφάνιση του κύβου (αριστερά πάνω). Η απόσταση left-right αυξάνεται αρκετά και αυτό έχει ως αποτέλεσμα ο κύβος να φαίνεται μικρότερος και συμπιεσμένος (δεξιά πάνω). Κάτω παρατηρούμε την επεξήγηση του όγκου θέασης. Τα πλάνα αποκοπής near και far δημιουργούν έναν κώνο κανονικών διαστάσεων (αριστερά κάτω). Μετά την αύξηση των αποστάσεων left-right και top–bottom, ο κώνος παραμορφώνεται και τα πλάνα αποκοπής μεγαλώνουν (δεξιά κάτω).

5.3.2.3 glOrtho()

Όπως είδαμε και πριν, η εντολή glOrtho() δημιουργεί παράλληλες τρισδιάστατες προβολές. Η διαδικασία που ακολουθείται για τη δημιουργία του μέρους αυτού είναι παρόμοια με προηγουμένως. Η κλάση OrthoActivity.java είναι το activity για το μέρος αυτό και περιλαμβάνει το χειρισμό των κουμπιών αλλά και την αρχικοποίηση του layout ortho.xml που αποτελεί το interface (βλ. παράρτημα). Το ortho.xml με τη σειρά του ορίζει το χώρο τρισδιάστατης προβολής μέσω της κλάσης ViewingGLSurfaceView.java που περιλαμβάνει τον κώδικα OpenGL ES, αλλά και το χώρο επεξήγησης του όγκου παράλληλης προβολής μέσω της κλάσης ExplainOrtho.java, η οποία χρησιμοποίει τη βιβλιοθήκη graphics του android.

Όπως είδαμε παραπάνω, χρησιμοποιείται η ίδια κλάση για την τρισδιάστατη θέαση αλλά αυτή τη φορά επιλέγεται η εντολή glOrtho() με τις παραμέτρους που δίνει ο χρήστης. Αυτό έχει ως αποτέλεσμα τη δημιουργία συμμετρικού ορθογώνιου όγκου θέασης.

Αυτό που διαφέρει είναι ο τρόπος που γίνεται η επεξήγηση για τον όγκο της προοπτικής προβολής που δημιουργεί η glOrtho() (Εικόνα 23). Ενώ στην προηγούμενη κλάση επεξήγησης ExplainFrustrum.java ο χρήστης έβλεπε τον κωνικό όγκο θέασης της glFrustrum τώρα βλέπει τον ορθογώνιο της glOrtho() δίνοντας έμφαση στις παραμέτρους left, right, bottom και top, έτσι ώστε να φαίνονται εύκολα τα αποτελέσματα από την αλλαγή των παραμέτρων αυτών (Εικόνα 29).



Εικόνα 29. Ορθογώνιος όγκος θέασης παράλληλης προβολής (αριστερά). Αν ο χρήστης αυξήσει την απόσταση μεταξύ left-right και top-bottom, ο κύβος φαίνεται μικρότερος (δεξιά).

6. Μίξη χρωμάτων

Σε πολλές εφαρμογές είναι βολικό να μπορούμε να συνδυάζουμε τα χρώματα από αντικείμενα που αλληλεπικαλύπτονται ή να αναμιγνύουμε ένα αντικείμενο με το φόντο. Μερικά παραδείγματα είναι η προσομοίωση του εφέ του πινέλου, ο σχηματισμός μιας σύνθετης εικόνας από δύο ή παραπάνω εικόνες, η μοντελοποίηση εφέ διαφάνειας και η εξομάλυνση των ορίων των αντικειμένων σε μια σκηνή. Τα περισσότερα πακέτα γραφικών παρέχουν μεθόδους παραγωγής διαφόρων εφέ μίξης χρωμάτων και αυτές οι διαδικασίες ονομάζονται συναρτήσεις μίξης χρωμάτων (color-blending functions) ή συναρτήσεις σύνθεσης εικόνων (image-compositing functions).

6.1 Μίξη χρωμάτων στην OpenGL και OpenGL ES

Στην OpenGL, τα χρώματα δύο αντικειμένων μπορούν να αναμιχτούν πρώτα φορτώνοντας ένα αντικείμενο στη μνήμη πλαισίων και μετά συνδυάζοντας το χρώμα του δεύτερου αντικειμένου με το χρώμα στη μνήμη πλαισίου. Το τρέχων χρώμα στη μνήμη πλαισίων αναφέρεται ως το χρώμα προορισμού(destination color) της OpenGL και το χρώμα του δεύτερου αντικειμένου είναι το πηγαίο χρώμα (source color) της OpenGL. Οι μέθοδοι μίξης μπορούν να εκτελεστούν μόνο σε κατάσταση RGB ή RGBA. Για να εφαρμόσουμε τη μίξη χρωμάτων σε μια εφαρμογή, πρώτα χρειάζεται να ενεργοποιήσουμε αυτή τη λειτουργία της OpenGL χρησιμοποιώντας την εντολή:

glEnable(GL_BLEND);

Απενεργοποιούμε τις ρουτίνες μίξης χρωμάτων της OpenGL με:

glDisable(GL_BLEND);

Εάν η μίξη χρωμάτων δεν είναι ενεργοποιημένη, τότε το χρώμα ενός αντικειμένου απλά αντικαθιστά τα περιεχόμενα της μνήμης πλαισίου στη θέση του αντικειμένου.

Τα χρώματα μπορούν να αναμιχθούν με ένα πλήθος διαφορετικών τρόπων, ανάλογα με τα εφέ που θέλουμε να επιτύχουμε και παράγουμε διαφορετικά χρωματικά εφέ ορίζοντας δυο σύνολα από παράγοντες μίξης (blending factors). Ένα σύνολο από παράγοντες μίξης είναι για το τρέχων αντικείμενο στη μνήμη πλαισίων, και το άλλο σύνολο από παράγοντες μίξης είναι για το εισερχόμενο αντικείμενο. Το νέο αναμιγμένο χρώμα που μετά φορτώνεται στη μνήμη πλαισίων υπολογίζεται ως

$(S_rR_s + D_rR_d, S_gG_s + D_gG_d, S_bBs + D_bB_d, S_aA_s + D_aA_d)$.

Όπου τα συστατικά του πηγαίου χρώματος RGBA είναι (R_s, G_s ,B_s ,A_s), τα συστατικά του χρώματος προορισμού είναι (R_d, G_d ,B_d ,A_d), οι πηγαίοι παράγοντες μίξης είναι (S_r, S_g ,S_b, S_a) και οι παράγοντες μίξης προορισμού είναι (D_r, D_g ,D_b, D_a). Οι τιμές που υπολογίζονται για τα στοιχεία του συνδυασμένου χρώματος περιορίζονται στην εμβέλεια από μηδέν μέχρι ένα. Δηλαδή, οποιοδήποτε άθροισμα μεγαλύτερο του ένα παίρνει την τιμή ένα, ενώ οποιοδήποτε άθροισμα μικρότερο του μηδενός παίρνει την τιμή μηδέν. Επιλέγουμε τις τιμές των παραγόντων μίξης με τη συνάρτηση:

glBlendFunc(sFactor, dFactor);

Οι παράμετροι sFactor και dFactor, οι πηγαίοι παράμετροι και οι παράμετροι προορισμού, παίρνουν τιμές μιας συμβολικής σταθεράς, ορίζοντας ένα προκαθορισμένο σύνολο από τέσσερις συντελεστές μίξης. Στην παρακάτω εικόνα φαίνονται οι τιμές που μπορούν να πάρουν οι παράμετροι.

Constant	RGB Blend Factor	Alpha Blend Factor
GL_ZERO	(0, 0, 0)	0
GL_ONE	(1, 1, 1)	1
GL_SRC_COLOR	(R_s, G_s, B_s)	A_s
GL_ONE_MINUS_SRC_COLOR	$(1, 1, 1)-(R_s, G_s, B_s)$	$1 - A_s$
GL_DST_COLOR	(R_d, G_d, B_d)	A_d
GL_ONE_MINUS_DST_COLOR	$(1, 1, 1)-(R_d, G_d, B_d)$	$1 - A_d$
GL_SRC_ALPHA	(A_s, A_s, A_s)	A_s
GL_ONE_MINUS_SRC_ALPHA	$(1, 1, 1)-(A_s, A_s, A_s)$	$1 - A_s$
GL_DST_ALPHA	(A_d, A_d, A_d)	A_d
GL_ONE_MINUS_DST_ALPHA	$(1, 1, 1)-(A_d, A_d, A_d)$	$1 - A_d$
GL_CONSTANT_COLOR	(R_c, G_c, B_c)	A_c
GL_ONE_MINUS_CONSTANT_COLOR	$(1, 1, 1)-(R_c, G_c, B_c)$	$1 - A_c$
GL_CONSTANT_ALPHA	(A_c, A_c, A_c)	A_c
GL_ONE_MINUS_CONSTANT_ALPHA	$(1, 1, 1)-(A_c, A_c, A_c)$	$1 - A_c$
GL_SRC_ALPHA_SATURATE	(f, f, f); $f = \min(A_s, 1-A_d)$	1

Εικόνα 30. Τιμές που παίρνουν οι πηγαίοι παράμετροι και οι παράμετροι προορισμού.

Η προκαθορισμένη τιμή του sFactor είναι η GL_ONE και του dFactor είναι η GL_ZERO. Συνεπώς, οι τρέχουσες τιμές των παραγόντων μίξης έχουν ως αποτέλεσμα οι εισερχόμενες χρωματικές τιμές να αντικαθιστούν τις τρέχουσες τιμές στη μνήμη πλαισίου.

6.2 Μίξη χρωμάτων στην εφαρμογή

6.2.1 Περιγραφή του interface

Όπως και στα προηγούμενα μέρη, το interface αποτελείται από το χώρο θέασης της προβολής στο οποίο εμφανίζονται δύο τετράγωνα με διαφορετικό χρώμα, το τμήμα του κώδικα σε OpenGL, το κουμπί βοήθειας για πληροφορίες σχετικές με την εντολή και το μέρος όπου ο χρήστης μπορεί να αλληλεπιδράσει με την εφαρμογή πατώντας κουμπιά και δίνοντας τα αντίστοιχα ορίσματα στις συναρτήσεις.

6.2.2 Υλοποίηση

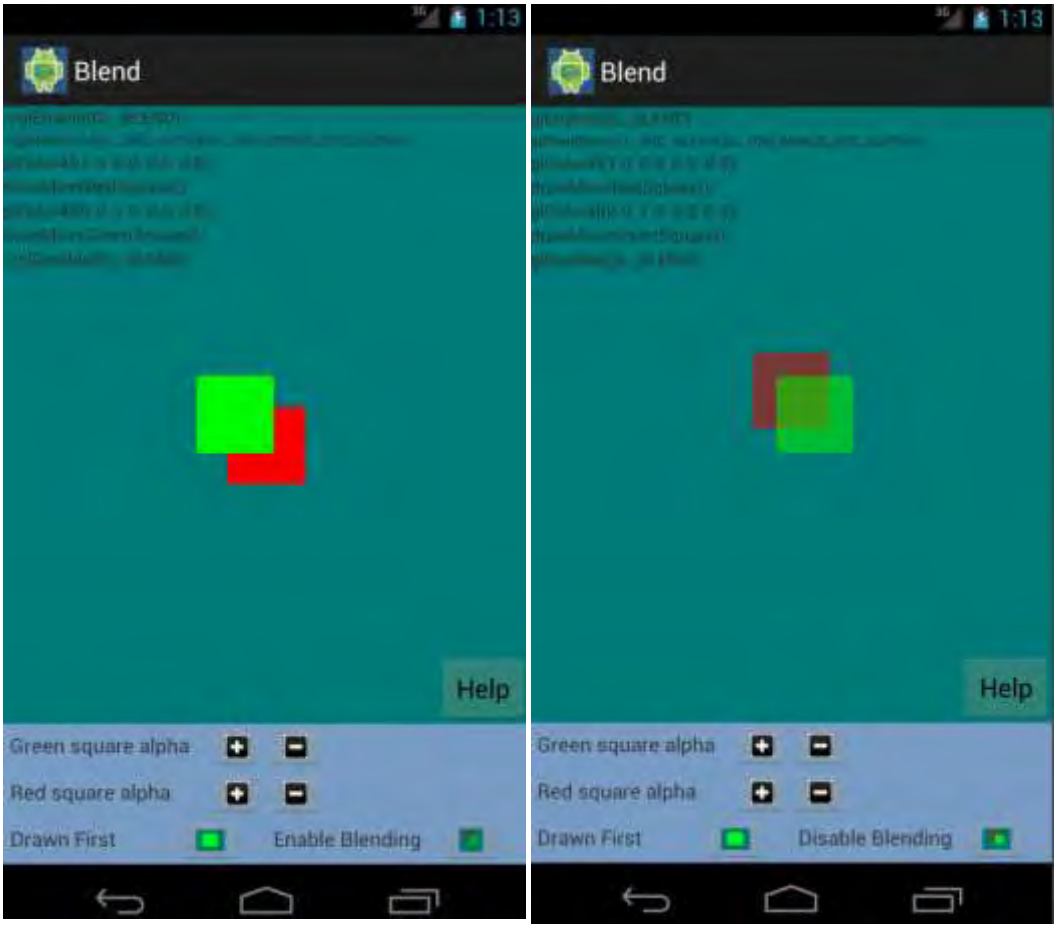
Η κλάση BlendingActivity.java είναι το activity για το μέρος αυτό και περιλαμβάνει το χειρισμό των κουμπιών αλλά και την αρχικοποίηση του layout blending.xml που αποτελεί το παραπάνω interface (βλ. παράρτημα). Το blending.xml με τη σειρά του ορίζει το χώρο προβολής μέσω της κλάσης BlendingGLSurfaceView.java που περιλαμβάνει τον κώδικα OpenGL ES.

Στην εικόνα 31 φαίνεται ο απλοποιημένος κώδικας της BlendingGLSurfaceView.java.

```
package openglTutorial;

public class BlendingGLSurfaceView extends GLSurfaceView implements Renderer {

	public BlendingGLSurfaceView (Context context) {
		super(context);
		this.context = context;

	}
	public void onSurfaceCreated(GL10 gl, EGLConfig config) {
		// διάφορες αρχικοποιήσεις
		gl.glBlendFunc(GL10.GL_SRC_ALPHA, GL10.GL_ONE_MINUS_SRC_ALPHA);
		//Set The Blending Function For Translucency
	}
	public void onDrawFrame(GL10 gl) {


	if(blend) {
		gl.glEnable(GL10.GL_BLEND);			//Turn Blending On
	} else {
		gl.glDisable(GL10.GL_BLEND);		//Turn Blending On
	}
	DrawAndMoveGreenSquare();
	DrawAndMoveRedSquare();
	printCode();

	}

	public void onSurfaceChanged(GL10 gl, int width, int height) {
		//εντολές για την δημιουργία του πλάνου
	}


}
```

Εικόνα 31. Ο απλοποιημένος κώδικας της BlendingGLSurfaceView.java.

Όπως μπορούμε να δούμε από τον κώδικα στη μέθοδο onSurfaceCreated() οι παράμετροι που χρησιμοποιούνται για το sFactor είναι το GL_SCR_ALPHA και για το dFactor το GL_ONE_MINUS_SRC_ALPHA για καλύτερα αποτελέσματα μίξης (βλ. ενότητα 6.1). Στην onDrawFrame() βλέπουμε ότι ο χρήστης μπορεί να ενεργοποιήσει και να απενεργοποιήσει το blending με τις εντολές που είδαμε στο 6.1. Αυτόματα ο κώδικας που είναι υπεύθυνος για το blending μπαίνει σε σχόλια (Εικόνα 31).

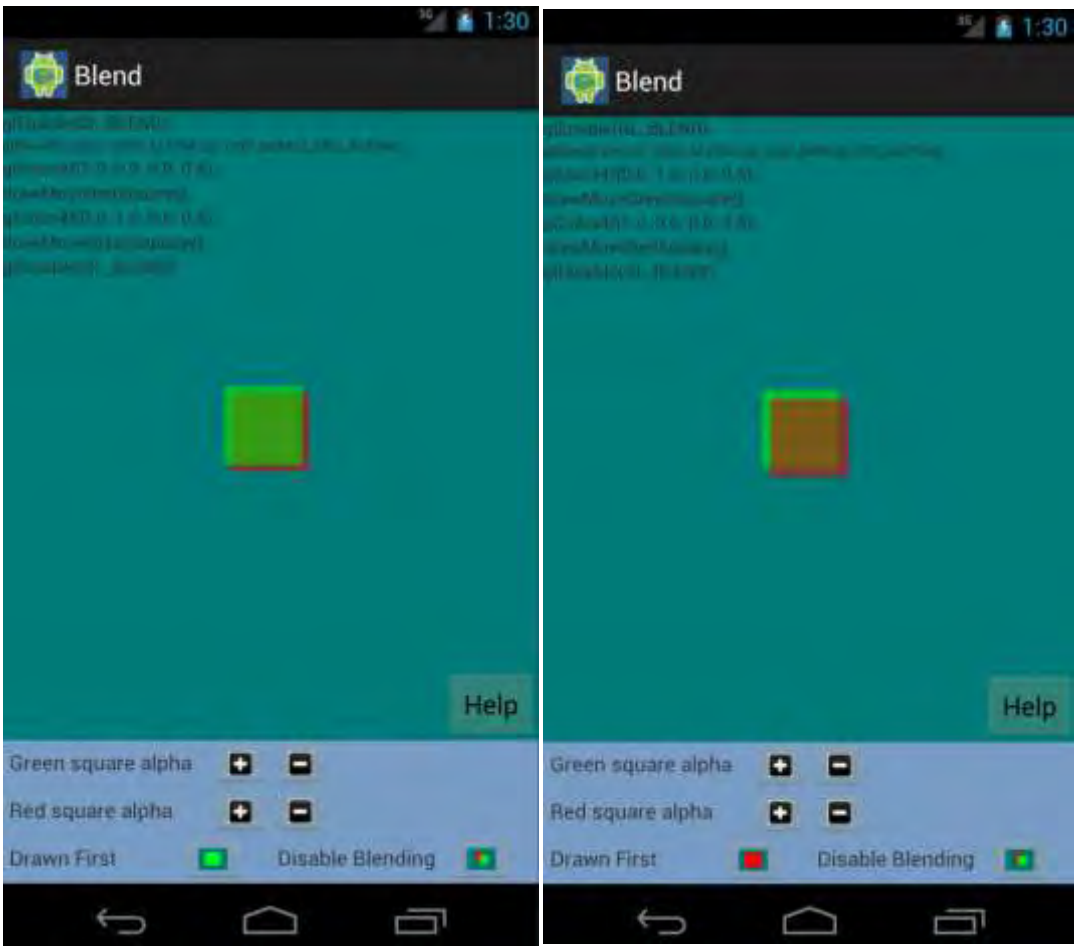


Εικόνα 32. Η κίνηση των τετραγώνων με απενεργοποιημένη τη μίξη χρωμάτων (αριστερά). Ενεργοποίηση της μίξης χρωμάτων (δεξιά).

Οι συναρτήσεις DrawAndMoveRedSquare() και DrawAndMoveGreenSquare() δεν κάνουν τίποτα περισσότερο από το να ζωγραφίζουν τα αντίστοιχα τετράγωνα στην οθόνη, όπως είδαμε σε προηγούμενα μέρη, αλλά και να δημιουργούν μια κίνηση, ώστε να αναμιγνύονται τα χρώματά τους με εντολές μετασχηματισμών μετατόπισης που είδαμε σε προηγούμενο κεφάλαιο.

Στη μέθοδο onSurfaceChanged() υπάρχουν εντολές για τη δημιουργία ενός απλού τρισδιάστατου χώρου παράλληλης προβολής με τις εντολές που είδαμε στο κεφάλαιο της θέασης.

Ο χρήστης επίσης μπορεί να πειράξει τις παραμέτρους alpha των τετραγώνων, έτσι ώστε να το κάνει διαφανή (alpha = 0) ή να μην επηρεάζονται καθόλου από το blending (alpha = 1). Επίσης, μπορεί να καθορίσει ποιο τετράγωνο ζωγραφίζεται πρώτο. Κάτι τέτοιο παίζει μεγάλο ρόλο καθώς το αναμιγμένο χρώμα που φαίνεται αλλάζει ανάλογα με το ποιο τετράγωνο είναι ζωγραφισμένο πρώτο (Εικόνα 33).



Εικόνα 33. Το κόκκινο τετράγωνο ζωγραφισμένο πρώτο (αριστερά). Το πράσινο τετράγωνο ζωγραφισμένο πρώτο (δεξιά). Βλέπουμε ότι η σειρά σχεδίασης παίζει ρόλο στο χρώμα που θα εμφανιστεί στο σημείο αλληλεπίδρασης.

7. Φωτισμός και Ομίχλη

Ένα μοντέλο φωτισμού υπολογίζει τα εφέ φωτισμού μιας επιφάνειας χρησιμοποιώντας τις διάφορες οπτικές ιδιότητες που έχουν δοθεί σ' αυτή την επιφάνεια. Αυτές οι ιδιότητες περιλαμβάνουν το βαθμό διαφάνειας, τους συντελεστές χρωματικής αντανάκλασης και διάφορες παραμέτρους επιφανειακών υφών.

Όταν το φως πέφτει πάνω σε μια αδιαφανή επιφάνεια, μέρος του αντανακλάται και μέρος του απορροφάται. Η ποσότητα του προσπίπτοντος φωτός που αντανακλάται από την επιφάνεια εξαρτάται από τον τύπο του υλικού. Τα γυαλιστερά αντικείμενα αντανακλούν περισσότερο από το προσπίπτων φως, ενώ οι θαμπές επιφάνειες απορροφούν περισσότερο από το προσπίπτων φως. Επίσης, μέρος του προσπίπτοντος φωτός πάνω σε μια διάφανη επιφάνεια μεταδίδεται μέσα στο υλικό.

Οι επιφάνειες που είναι τραχιές ή με πολλά μικρά εξογκώματα τείνουν να διασκορπίζουν το ανακλώμενο φως σε όλες τις κατευθύνσεις. Αυτό το διασκορπισμένο φως ονομάζεται *διάχυτη αντανάκλαση (diffuse reflection).* Μια πολύ τραχιά θαμπή επιφάνεια, παράγει κυρίως διάχυτες αντανακλάσεις, έτσι ώστε η επιφάνεια να εμφανίζεται εξίσου φωτεινή από οποιαδήποτε γωνία θέασης. Αυτό που ονομάζουμε χρώμα ενός αντικειμένου είναι το χρώμα της διάχυτης αντανάκλασης όταν το αντικείμενο φωτίζεται σε λευκό φως, το οποίο απαρτίζεται από ένα συνδυασμό όλων των χρωμάτων. Για παράδειγμα, ένα γαλάζιο αντικείμενο αντανακλά το γαλάζιο στοιχείο του λευκού φωτός και απορροφά όλα τα άλλα χρωματικά στοιχεία. Αν δούμε το γαλάζιο αντικείμενο κάτω από κόκκινο φως, αυτό εμφανίζεται ως μαύρο αφού απορροφάται όλο το προσπίπτων φως.

Πέραν από τη διασκόρπιση του διάχυτου φωτός, μέρος του φωτός που αντανακλάται συγκεντρώνεται σε λαμπερό ή φωτεινό σημείο, που ονομάζεται *κατοπτρική αντανάκλαση(specular reflection).* Αυτό το εφέ της λάμψης είναι πιο έντονο σε γυαλιστερές επιφάνειες απ' ότι σε θαμπές. Ακόμη, μπορούμε να δούμε με την κατοπτρική αντανάκλαση όταν κοιτάμε προς μια φωτισμένη γυαλιστερή επιφάνεια, όπως ένα γυαλισμένο μέταλλο, ένα μήλο ή το μέτωπο ενός ανθρώπου μόνο όταν κοιτάμε την επιφάνεια από μια συγκεκριμένη κατεύθυνση.

Ένας ακόμα παράγοντας που πρέπει να ληφθεί υπόψη σε ένα μοντέλο φωτισμού, είναι ο *φωτισμός περιβάλλοντος (ambient light)* σε μια σκηνή. Μια επιφάνεια που δεν εκτίθεται απευθείας σε μια πηγή φωτός μπορεί να εξακολουθεί να είναι ορατή εξαιτίας του φωτός που αντανακλάται από κοντινά αντικείμενα που φωτίζονται. Έτσι, ο φωτισμός περιβάλλοντος μιας σκηνής είναι το εφέ φωτισμού που παράγεται από το φως που αντανακλάται από τις διάφορες επιφάνειες σε μια σκηνή. Το

συνολικό αντανακλώμενο φως από την επιφάνεια είναι το άθροισμα των συνεισφορών από πηγές φωτός και από το φως που αντανακλάται από άλλα φωτισμένα αντικείμενα.

Ένας άλλος παράγοντας που περιλαμβάνεται μερικές φορές σε ένα μοντέλο φωτισμού είναι η επίδραση της ατμόσφαιρας πάνω στο χρώμα ενός αντικειμένου. Μια ομιχλώδης ατμόσφαιρα κάνει τα χρώματα να ξεθωριάζουν και τα αντικείμενα να φαίνονται σκοτεινότερα. Έτσι, θα μπορούσαμε να ορίσουμε μια συνάρτηση που να τροποποιεί τα χρώματα των επιφανειών ανάλογα με την ποσότητα της σκόνης, του καπνού ή της ομίχλης που θέλουμε να προσομοιώσουμε στην ατμόσφαιρα. Το εφέ της ομιχλώδους ατμόσφαιρας συχνά προσομοιώνεται σε μια εκθετική συνάρτηση εξασθένισης όπως:

$f_{atmo}(d) = e^{-pd}$ ***(1)*** ή

$f_{atmo}(d) = e^{-(pd)^2}$ ***(2)***

Η τιμή που παίρνει το d είναι η απόσταση του αντικειμένου από τη θέση θέασης. Ακόμη, μπορούμε να χρησιμοποιήσουμε την παράμετρο p σε οποιαδήποτε από αυτές τις δύο εκθετικές συναρτήσεις για να ορίσουμε μια θετική τιμή πυκνότητας για την ατμόσφαιρα. Οι υψηλότερες τιμές του p παράγουν μια πιο πυκνή ατμόσφαιρα και έχουν ως αποτέλεσμα τα χρώματα των επιφανειών να είναι πιο σκοτεινά. Αφότου έχει υπολογιστεί το χρώμα της επιφάνειας ενός αντικειμένου, πολλαπλασιάζουμε αυτό το χρώμα με μια από τις συναρτήσεις ατμόσφαιρας για να ελαττώσουμε την έντασή του κατά ποσότητα, που εξαρτάται από την τιμή που θέτουμε για την πυκνότητα της ατμόσφαιρας.

Θα μπορούσαμε αντί για εκθετική συνάρτηση να χρησιμοποιήσουμε τη γραμμική συνάρτηση ενδείξεων βάθους για να απλοποιήσουμε τους υπολογισμούς ατμοσφαιρικής εξασθένισης.

$f_{depth}(d) = d_{max} - d / d_{max} - d_{min}$ ***(3)***

Αυτό μειώνει την ένταση των επιφανειακών χρωμάτων των μακρινών αντικειμένων, όμως τότε δεν έχουμε καθόλου πρόβλεψη για την μεταβολή της πυκνότητας της ατμόσφαιρας.

7.1 Συναρτήσεις φωτισμού και ομίχλης της OpenGL και OpenGL ES

Με την ακόλουθη εντολή ορίζουμε την τιμή μιας ιδιότητας μιας πηγής φωτός:

glLight(lightName ,lightProperty, propertyValue);*

Ανάλογα με τον τύπο δεδομένων της τιμής της ιδιότητας προστίθεται η κατάληξη i ή f στο όνομα της συνάρτησης. Για διανυσματικά δεδομένα προστίθεται, επίσης, και η κατάληξη v και τότε η παράμετρος propertyValue είναι ένας δείκτης σε έναν πίνακα. Αναφερόμαστε σε κάθε πηγή φωτός με ένα αναγνωριστικό, ενώ η παράμετρος lightName παίρνει ένα από τα συμβολικά αναγνωριστικά της OpenGL GL_LIGHT0, GL_LIGHT1 … GL_LIGHT7. Αν και μερικές υλοποιήσεις της OpenGL επιτρέπουν περισσότερες από 8 πηγές φωτισμού. Ομοίως, η παράμετρος lightProperty πρέπει να πάρει μια από τις δέκα συμβολικές σταθερές της OpenGL (Εικόνα 34).

Parameter Name	Default Values	Meaning
GL_AMBIENT	(0.0, 0.0, 0.0, 1.0)	ambient intensity of light
GL_DIFFUSE	(1.0, 1.0, 1.0, 1.0) or (0.0, 0.0, 0.0, 1.0)	diffuse intensity of light (default for light 0 is white; for other lights, black)
GL_SPECULAR	(1.0, 1.0, 1.0, 1.0) or (0.0, 0.0, 0.0, 1.0)	specular intensity of light (default for light 0 is white; for other lights, black)
GL_POSITION	(0.0, 0.0, 1.0, 0.0)	(*x*, *y*, *z*, *w*) position of light
GL_SPOT_DIRECTION	(0.0, 0.0, −1.0)	(*x*, *y*, *z*) direction of spotlight
GL_SPOT_EXPONENT	0.0	spotlight exponent
GL_SPOT_CUTOFF	180.0	spotlight cutoff angle
GL_CONSTANT_ATTENUATION	1.0	constant attenuation factor
GL_LINEAR_ATTENUATION	0.0	linear attenuation factor
GL_QUADRATIC_ATTENUATION	0.0	quadratic attenuation factor

Εικόνα 34. Οι συμβολικές σταθερές που παίρνει η παράμετρος lightProperty και η επεξήγησή τους.

Μετά τον ορισμό όλων των ιδιοτήτων ενεργοποιούμε την κάθε πηγή φωτός με την εντολή:

glEnable(lightName);

Αλλά πρέπει να ενεργοποιήσουμε και τις ρουτίνες φωτισμού με την εντολή:

glEnable(GL_LIGHTING);

Από τα παραπάνω βλέπουμε ότι αν θέλουμε να ορίσουμε μια πηγή φωτός στη θέση P = (2,2,0) τότε οι εντολές που θα χρειαστούμε είναι:

GLFloat lightPos [] ={2.0,2.0,0.0,1.0};

glLightfv (GL_LIGHT0 , GL_POSITION, lightPos);

glEnable(GL_LIGHT0);

Στη συνέχεια, για να ορίσουμε το χρώμα φωτισμού περιβάλλοντος μαύρο και τα χρώματα διάχυτου και κατοπτρικού φωτισμού λευκά χρησιμοποιούμε τις εντολές:

GLFloat white [] ={1.0,1.0,1.0,1.0};

GLFloat black[] ={0.0,0.0,0.0,1.0};

glLightfv (GL_LIGHT0 , GL_AMBIENT, black);

glLightfv (GL_LIGHT0 , GL_DIFFUSE, white);

glLightfv (GL_LIGHT0 , GL_SPECULAR, white);

Σε συνολικό επίπεδο μπορούν να οριστούν αρκετές παράμετροι φωτισμού της OpenGL. Αυτές οι τιμές χρησιμοποιούνται για τον έλεγχο του τρόπου με τον οποίο εκτελούνται μερικοί υπολογισμοί φωτισμού, ενώ η τιμή της συνολικής παραμέτρου τίθεται με την ακόλουθη συνάρτηση:

glLightModel* (paramName , paramValue);

Προσθέτουμε μια κατάληξη i ή f ανάλογα με τον τύπο δεδομένων της τιμής της παραμέτρου. Ακόμη, για διανυσματικά δεδομένα προσθέτουμε επίσης και την κατάληξη v. Η παράμετρος paramName παίρνει μια συμβολική σταθερά της OpenGL που ορίζει τη συνολική ιδιότητα που θα ορίσουμε και η παράμετρος paramValue παίρνει μια μόνο τιμή ή ένα σύνολο τιμών. Με τη χρήση της συνάρτησης glLightModel μπορούμε να ορίσουμε πώς θα υπολογιστούν τα εφέ των κατοπτρικών λάμψεων, όπως επίσης και να επιλέξουμε να εφαρμόσουμε το μοντέλο φωτισμού στις πίσω έδρες των πολυγωνικών επιφανειών. Οι τιμές που παίρνουν οι παράμετροι φαίνονται στην εικόνα 35.

Parameter Name	Default Value	Meaning
GL_LIGHT_MODEL_AMBIENT	(0.2, 0.2, 0.2, 1.0)	ambient RGBA intensity of the entire scene
GL_LIGHT_MODEL_LOCAL_VIEWER	0.0 or GL_FALSE	how specular reflection angles are computed
GL_LIGHT_MODEL_TWO_SIDE	0.0 or GL_FALSE	specifies one-sided or two-sided lighting
GL_LIGHT_MODEL_COLOR_CONTROL	GL_SINGLE_COLOR	whether specular color is calculated separately from ambient and diffuse

Εικόνα 35. Οι τιμές που παίρνουν οι παράμετροι της glLightModel().

Έτσι, αν θέλουμε να εφαρμόσουμε τους υπολογισμούς φωτισμού και στις εμπρός και στις πίσω έδρες ενός αντικειμένου χρησιμοποιούμε την εντολή:

glLightModelf(GL_LIGHT_MODEL_TWO_SIDE, 0.0);

Οι συντελεστές αντανάκλασης και οι άλλες οπτικές ιδιότητες των επιφανειών δίνονται χρησιμοποιώντας τη συνάρτηση:

glMaterial(surFace, surfProperty, propertyValue);*

Στη συνάρτηση προστίθεται η κατάληξη i ή f ανάλογα με τον τύπο δεδομένων της ιδιότητας, όπως επίσης και η κατάληξη v όταν δίνουμε ιδιότητες σε μορφή διανύσματος. Η παράμετρος surface παίρνει μια από τις συμβολικές σταθερές Gl_FRONT, GL_BACK ή GL_FRONT_AND_BACK, ενώ η παράμετρος surfProperty είναι μια συμβολική σταθερά που καθορίζει μια παράμετρο της επιφάνειας και propertyValue παίρνει την αντίστοιχη τιμή (Εικόνα 36).

Parameter Name	Default Value	Meaning
GL_AMBIENT	(0.2, 0.2, 0.2, 1.0)	ambient color of material
GL_DIFFUSE	(0.8, 0.8, 0.8, 1.0)	diffuse color of material
GL_AMBIENT_AND_DIFFUSE		ambient and diffuse color of material
GL_SPECULAR	(0.0, 0.0, 0.0, 1.0)	specular color of material
GL_SHININESS	0.0	specular exponent
GL_EMISSION	(0.0, 0.0, 0.0, 1.0)	emissive color of material
GL_COLOR_INDEXES	(0, 1, 1)	ambient, diffuse, and specular color indices

Εικόνα 36. Οι παράμετροι της glMaterial().

Έτσι, αν θέλαμε να κάνουμε ένα αντικείμενο να έχει γαλάζιο διάχυτο φωτισμό και λευκή κατοπτρική αντανάκλαση. Χρησιμοποιούμε τις εντολές:

GLFloat cyan [] = {0.0,1.0,1.0,1.0};

GLFloat white[] = {1.0,1.0,1.0,1.0};

glMaterialfv(GL_FROND_AND_BACK,GL_DIFFUSE,cyan);

glMaterialfv(GL_FROND_AND_BACK,GL_SPECULAR,white);

Επίσης, για να θέσουμε το χρώμα εκπομπής μιας επιφάνειας (π.χ. σε λευκό) χρησιμοποιούμε την εντολή:

glMaterialfv(GL_FROND_AND_BACK,GL_EMISSION,white);

Όπως αναφέρθηκε και προηγουμένως, μπορούμε να χρησιμοποιήσουμε μια συνάρτηση εξασθένισης της έντασης της ατμόσφαιρας για να προσομοιώσουμε τη σκηνή μέσα από μια ομιχλώδη ατμόσφαιρα. Οι διάφορες παράμετροι της ατμόσφαιρας μπορούν να οριστούν χρησιμοποιώντας τη συνάρτηση:

glFog(atmoParameter, paramValue);*

Μια κατάληξη i ή f υποδηλώνει τον τύπο των τιμών των δεδομένων, ενώ η κατάληξη v χρησιμοποιείται με διανυσματικά δεδομένα.

Για να ορίσουμε το χρώμα της ατμόσφαιρας δίνουμε στην παράμετρο atmoParameter τη συμβολική σταθερά GL_FOG_COLOR της OpenGL. Για παράδειγμα, υποδηλώνουμε ότι η ατμόσφαιρα έχει γκρίζο-γαλάζιο χρώμα με:

GLFloat atmoColor[] = {0.8, 0.8, 1.0, 1.0};

glFogfv(GL_FOG_COLOR, atmoColor);

Μετά, μπορούμε να επιλέξουμε τη συνάρτηση εξασθένισης της ατμόσφαιρας που θα χρησιμοποιηθεί για το συνδυασμό του χρώματος του αντικειμένου με το χρώμα της ατμόσφαιρας. Αυτό επιτυγχάνεται με τη συμβολική σταθερά GL_FOG_MODE:

glFogi(GL_FOG_MODE, atmoFunc);

Εάν η παράμετρος atmoFunc πάρει την τιμή GL_EXP, τότε χρησιμοποιείται για τη συνάρτηση εξασθένισης της ατμόσφαιρας η εξίσωση (1) που είδαμε παραπάνω. Με την τιμή GL_EXP2 επιλέγουμε την εξίσωση (2). Για οποιεσδήποτε από τις εκθετικές συναρτήσεις, μπορούμε να επιλέξουμε μια τιμή πυκνότητας της ατμόσφαιρας με:

glFog (GL_FOG_DENSITY, atmoDensity);

Μια τρίτη επιλογή εξασθένισης της ατμόσφαιρας είναι η γραμμική συνάρτηση (3). Σε αυτήν την περίπτωση η παράμετρος atmoFunc παίρνει την τιμή GL_LINEAR.

Για να ορίσουμε τα dmax και dmin από τη συνάρτηση (3) χρησιμοποιούμε τις εντολές glFogf(GL_FOG_END,maxValue) και glFog(GL_FOG_START,minValue).

7.2 Φωτισμός και ομίχλη στην εφαρμογή

Στην εφαρμογή το κεφάλαιο αυτό χωρίζεται σε δύο μέρη. Το φωτισμό (Lighting) και την ομίχλη (Fog). Το interface του φωτισμού και της ομίχλης μοιάζει με όλα τα άλλα τμήματα της εφαρμογής καθώς παρέχουν τμήμα εμφάνισης κώδικα, τρισδιάστατο χώρο επίδειξη των αντίστοιχων δυνατοτήτων, κουμπί βοήθειας, και κουμπιά αλληλεπίδρασης με το χρήστη.

7.2.1 Υλοποίηση Φωτισμού

Η κλάση LightingActivity.java είναι το activity για το μέρος αυτό και περιλαμβάνει το χειρισμό των κουμπιών αλλά και την αρχικοποίηση του layout lighting.xml που αποτελεί το interface (βλ. παράρτημα). Το lighting.xml με τη σειρά του ορίζει το χώρο προβολής μέσω της κλάσης LightingGLSurfaceView.java που περιλαμβάνει τον κώδικα OpenGL ES.

Στην εικόνα 37 φαίνεται ο απλοποιημένος κώδικας της LightingGLSurfaceView.java.

Όπως βλέπουμε στη μέθοδο onSurfaceCreated() έχουμε τις αρχικοποιήσεις της πηγής φωτός GL_LIGHT0, το οποίο παίρνει τιμές από το χρήστη. Οι αρχικές τιμές είναι λευκό για το διάχυτο φωτισμό και την κατοπτρική αντανάκλαση και μαύρο για το φωτισμό περιβάλλοντος. Παρακάτω βλέπουμε να αρχικοποιούνται τα χρώματα του υλικού της σφαίρας. Τα χρώματα είναι γαλάζιο για το διάχυτο φωτισμό και το φωτισμό περιβάλλοντος και λευκό για την κατοπτρική αντανάκλαση.

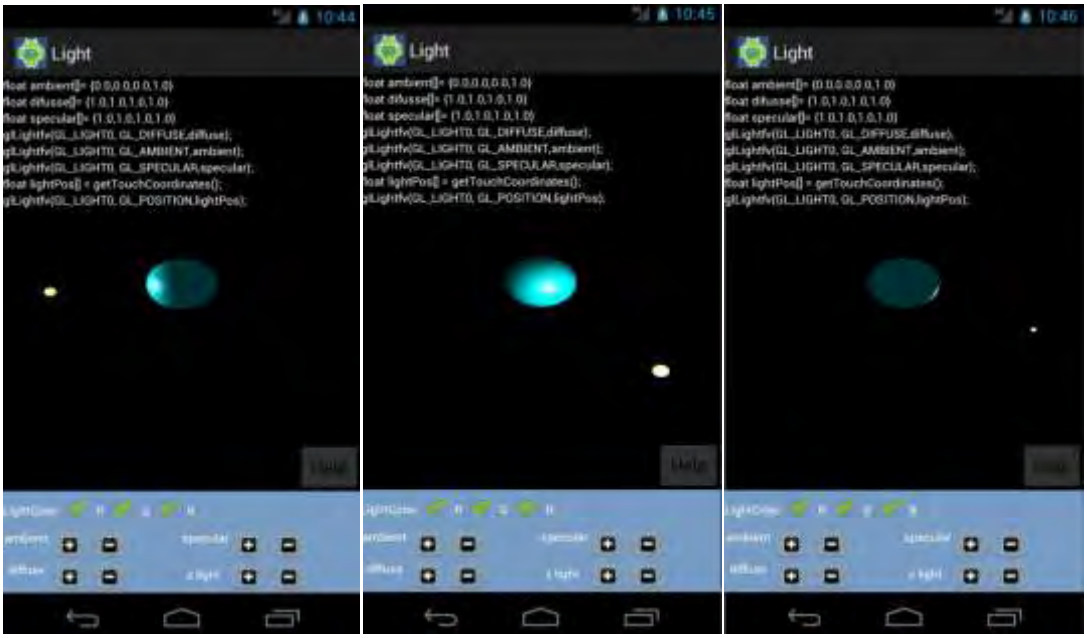
```
public void onSurfaceCreated(GL10 gl, EGLConfig config) {

gl.glLightfv(GL_LIGHT0, GL10.GL_POSITION, makeFloatBuffer(pos));
gl.glLightfv(GL_LIGHT0, GL10.GL_DIFFUSE, makeFloatBuffer(diffuse));
gl.glLightfv(GL_LIGHT0, GL10.GL_SPECULAR, makeFloatBuffer(specular));
gl.glLightfv(GL_LIGHT0, GL10.GL_AMBIENT, makeFloatBuffer(ambient));
gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_DIFFUSE, makeFloatBuffer(cyan));
gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_AMBIENT, makeFloatBuffer(cyan));
gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_SPECULAR, makeFloatBuffer(white));
gl.glEnable(GL10.GL_LIGHTING );
gl.glLightModelf(GL10.GL_LIGHT_MODEL_TWO_SIDE, 0.0f);
gl.glEnable(GL_LIGHT0);
}

public void onDrawFrame(GL10 gl) {
      gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
      gl.glClearColor(0.0f,0.0f, 0.0f, 1.0f);

      gl.glMatrixMode(GL10.GL_MODELVIEW);
      gl.glLoadIdentity();

      gl.glLightfv(GL_LIGHT0, GL10.GL_POSITION, makeFloatBuffer(pos)););
      gl.glLightfv(GL_LIGHT0, GL10.GL_AMBIENT, makeFloatBuffer(ambient)););
      gl.glLightfv(GL_LIGHT0, GL10.GL_DIFFUSE, makeFloatBuffer(diffuse)););
      gl.glLightfv(GL_LIGHT0, GL10.GL_SPECULAR, makeFloatBuffer(specular)););


      gl.glTranslatef(0.0f,0.0f,0.0f);
      sphere.draw(gl);

gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_EMISSION, makeFloatBuffer(paleYellow));


      gl.glTranslatef(pos[0],pos[1],pos[2]);
      light.draw(gl);


gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_EMISSION, makeFloatBuffer(black));

}


public void onSurfaceChanged(GL10 gl, int width, int height) {
      //εντολές για την δημιουργια του πλάνου
}
```

Εικόνα 37. Απλοποιημένος κώδικας της LightingGLSurfaceView.Java

Στη συνέχεια, έχουμε τις εντολές που ενεργοποιούν τα τρέχοντα φώτα και τις εντολές φωτισμού αλλά και την επιλογή του μοντέλου φωτισμού με την εντολή glLightModel() που εξηγήθηκε παραπάνω.

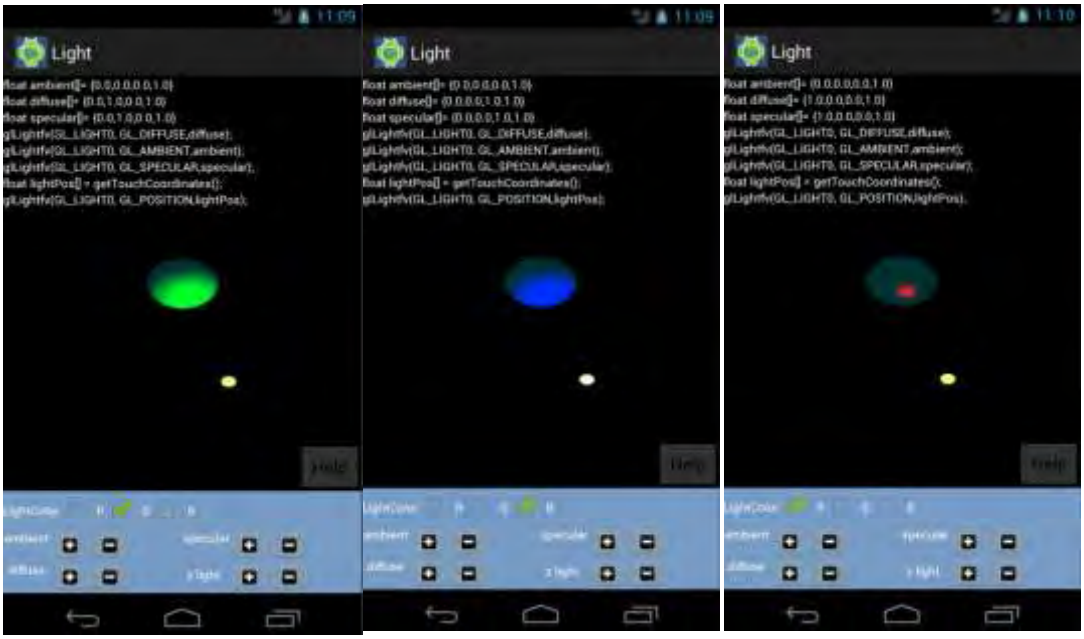
Μετέπειτα, στη μέθοδο onDrawFrame() έχουμε ξανά τις εντολές του φωτός επειδή αυτές αλλάζουν καθώς ο χρήστης πειράζει τις τιμές των χρωμάτων, συνεπώς καλούνται ξανά εδώ αφού η onDrawFrame() ζωγραφίζει τη σκηνή και άρα καλείται επαναληπτικά. Ακολουθεί η εντολή δημιουργίας της γαλάζιας σφαίρας καθώς και μιας μικρότερης η οποία είναι κίτρινη με τη χρήση της εντολής glMaterial(), με την παράμετρο GL_EMISSION που είδαμε παραπάνω. Η κίτρινη σφαίρα προσομοιώνει το φως το οποίο μπορεί ο χρήστης να μετακινεί με το δάκτυλό του αλλά και με το κουμπί z light κατά μήκος του άξονα z (Εικόνα 38).

Τέλος, η μέθοδος onSurfaceCreated καλείται για τη δημιουργία του πλάνου που εδώ δημιουργείται με την εντολή glFrustum() που είδαμε στο κεφάλαιο 4.



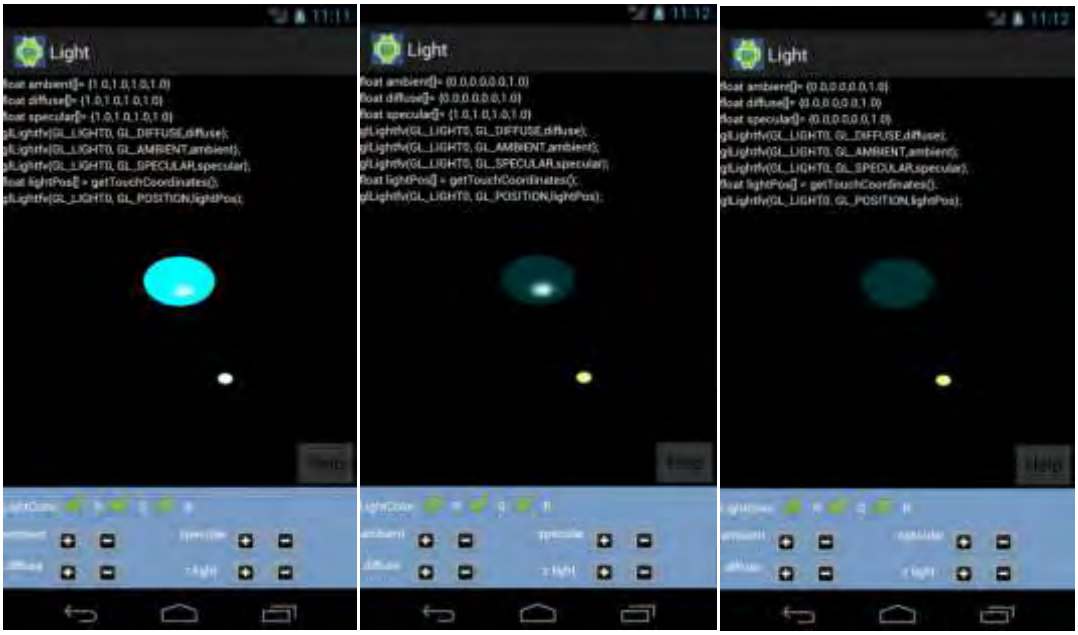
Εικόνα 38. Ο χρήστης μπορεί ακουμπώντας την οθόνη να δώσει τις συντεταγμένες της πηγής φωτός και παράλληλα να μετακινήσει την κίτρινη σφαίρα. Η σφαίρα και η πηγή φωτός μπορούν να κινηθούν και κατά μήκος του άξονα z με τη χρήση του κουμπιού z light.

Παρατηρούμε ότι ο χρήστης μπορεί να αλλάξει και το χρώμα του φωτισμού. Ενδιαφέρον παρουσιάζει η περίπτωση όπου το χρώμα γίνεται κόκκινο (Εικόνα 39).



Εικόνα 39. Το χρώμα γίνεται πράσινο (αριστερά). Το χρώμα γίνεται μπλε (μέση). Ενώ το χρώμα είναι κόκκινο βλέπουμε μόνο την κατοπτρική αντανάκλαση κόκκινη (δεξιά). Αυτό συμβαίνει γιατί το χρώμα της κατοπτρικής αντανάκλασης του υλικού της γαλάζιας σφαίρας είναι το λευκό και συνεπώς αποτυπώνεται ότι το χρώμα φωτός θα πέσει πάνω του. Άρα, επειδή το υλικό της σφαίρας έχει φως διάχυσης γαλάζιο (0,1,1) και το φως διάχυσης της πηγής φωτός είναι κόκκινο (1,0,0), δηλαδή αντίστροφα χρώματα, το αποτέλεσμα είναι το μαύρο χρώμα.

Τέλος, ο χρήστης μπορεί να μεταβάλλει τα διανύσματα ambient, diffuse και specular με τα ακόλουθα αποτελέσματα (Εικόνα 40).



Εικόνα 40. Το διάνυσμα ambient με μέγιστες τιμές (αριστερά). Ενεργοποιημένο μόνο το διάνυσμα specular (μέση). Όλα τα διανύσματα με μηδενικές τιμές (δεξιά).

7.2.2 Υλοποίηση ομίχλης

Η κλάση FogActivity.java είναι το activity για το μέρος αυτό και περιλαμβάνει το χειρισμό των κουμπιών αλλά και την αρχικοποίηση του layout fog.xml που αποτελεί το interface (βλ. παράρτημα). Το fog.xml με τη σειρά του ορίζει το χώρο προβολής μέσω της κλάσης FogGLSurfaceView.java που περιλαμβάνει τον κώδικα OpenGL ES.

Στην Εικόνα 40 φαίνεται ο απλοποιημένος κώδικας της FogGLSurfaceView.java.

Όπως βλέπουμε στη μέθοδο onSurfaceCreated() έχουμε την αρχικοποίηση του τύπου ομίχλης μέσω της εντολής glFogf(GL10.GL_FOG_MODE, fogMode[fogFilter]); Με το fogMode να είναι πίνακας που περιέχει τις συμβολικές σταθερές GL_EXP,GL_EXP2 και GL_LINEAR. Στη συνέχεια, ορίζουμε το χρώμα της ομίχλης αλλά και την πυκνότητα για τις εκθετικές συναρτήσεις, καθώς και τις παραμέτρους αρχής και τέλους για τη γραμμική συνάρτηση.

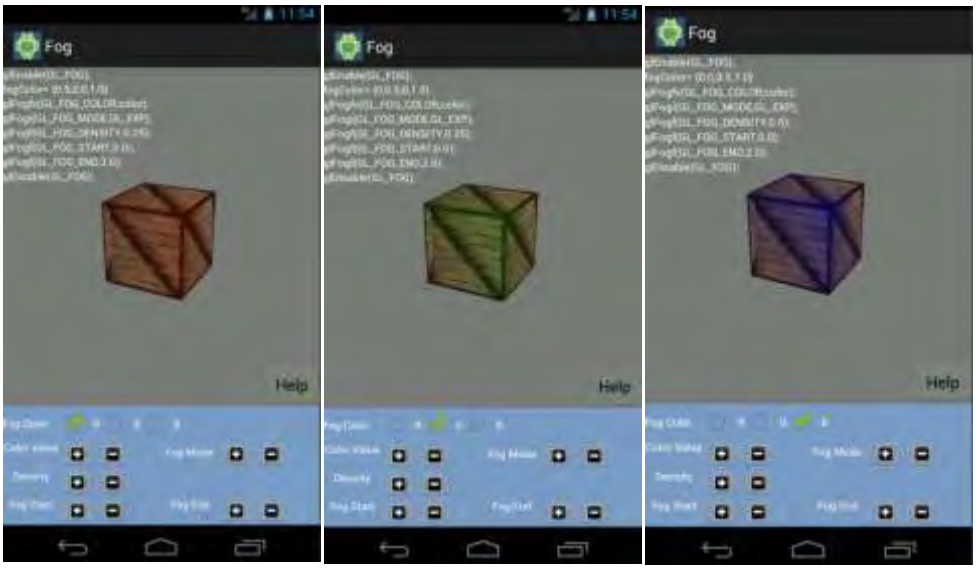
Στη μέθοδο onDrawFrame() ξαναγράφουμε τις παραπάνω εντολές, καθώς αυτές μπορεί να κληθούν με διαφορετικές παραμέτρους από το χρήστη και στη συνέχεια ζωγραφίζουμε τον κύβο, ο οποίος μπορεί να περιστρέφεται με το άγγιγμα του χρήστη όπως φαίνεται από τον κώδικα.

Τέλος, η μέθοδος onSurfaceCreated καλείται για τη δημιουργία του πλάνου που εδώ δημιουργείται με την εντολή glFrustum() που είδαμε στο κεφάλαιο 4.

```
public void onSurfaceCreated(GL10 gl, EGLConfig config) {

        gl.glFogf(GL10.GL_FOG_MODE, fogMode[fogFilter]);
        gl.glFogfv(GL10.GL_FOG_COLOR, makeFloatBuffer(fogColor));
        gl.glFogf(GL10.GL_FOG_START, start);
        gl.glFogf(GL10.GL_FOG_END, end);
        gl.glEnable(GL10.GL_FOG);
    }

    public void onDrawFrame(GL10 gl) {
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
        gl.glClearColor(0.0f,0.0f, 0.0f, 1.0f);

        gl.glMatrixMode(GL10.GL_MODELVIEW);
        gl.glLoadIdentity();

        gl.glFogf(GL10.GL_FOG_MODE, fogMode[fogFilter]);
        gl.glFogfv(GL10.GL_FOG_COLOR, makeFloatBuffer(fogColor));
        gl.glFogf(GL10.GL_FOG_DENSITY, density);
        gl.glFogf(GL10.GL_FOG_START, start);
        gl.glFogf(GL10.GL_FOG_END, end);

        //Rotate around the axis based on the rotation matrix (rotation, x, y, z)
        gl.glRotatef(xrot, 1.0f, 0.0f, 0.0f);    //X
        gl.glRotatef(yrot, 0.0f, 1.0f, 0.0f);    //Y

        cube.draw(gl, filter);                          //Draw the Cube

        //Change rotation factors
        xrot += xspeed;
        yrot += yspeed;

    }


    public void onSurfaceChanged(GL10 gl, int width, int height) {
        //εντολές για την δημιουργια του πλάνου
    }
```

Εικόνα 41. Ο απλοποιημένος κώδικας της FogGLSurfaceView.java.

Το πώς επηρεάζει ο χρήστης το χρώμα της ομίχλης φαίνεται στην εικόνα 42.

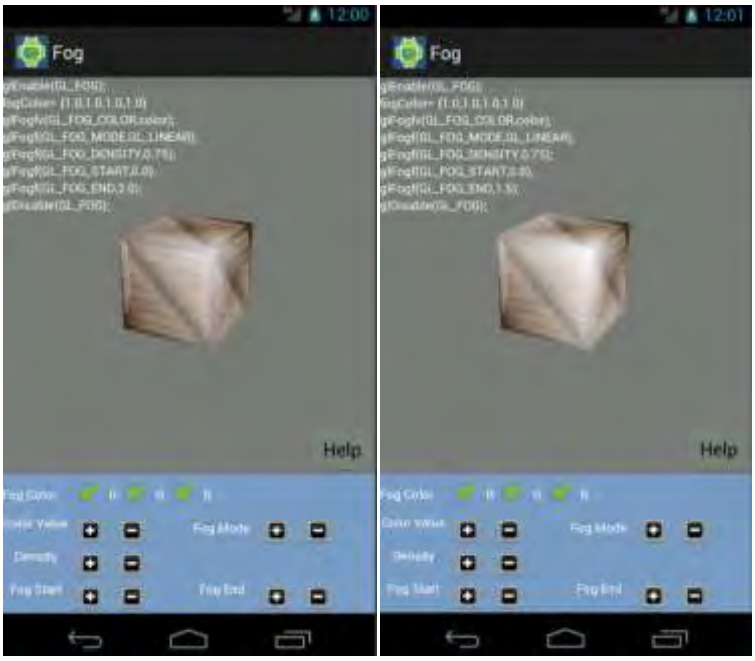


Εικόνα 42. Κόκκινη ομίχλη (αριστερά).Πράσινη ομίχλη με ίδια πυκνότητα (μέση). Μπλε ομίχλη με μεγαλύτερη πυκνότητα (δεξιά).

Στην εικόνα 42 βλέπουμε ότι ο επιλεγμένος τύπος ομίχλης είναι ο GL_EXP και μπορεί να συνδυαστεί με την πυκνότητα για το επιθυμητό αποτέλεσμα. Στην εικόνα 43 φαίνεται η διαφορά του GL_EXP και GL_EXP2 με την ίδια πυκνότητα.

Εικόνα 43.Η συνάρτηση ομίχλης GL_EXP και GL_EXP2 με την ίδια πυκνότητα.

Τέλος, η γραμμική συνάρτηση και το πώς επηρεάζεται από τους παράγοντες start και end φαίνεται στην εικόνα 44.



Εικόνα 44. Η γραμμική συνάρτηση GL_LINEAR. Η συνάρτηση με αλλαγμένο τον παράγοντα end (δεξιά).

8. Τελικές σκέψεις-Συμπεράσματα

Ο αριθμός των θεμάτων που καλύφθηκαν στην εφαρμογή και στην παρούσα εργασία είναι πολύ μικρός σε σχέση με αυτά που υπάρχουν στην OpenGL. Έπρεπε να γίνει μια προσεκτική επιλογή της ύλης που θα εξεταστεί. Έτσι, έγινε μια προσπάθεια να εξεταστούν τα βασικά στοιχεία της γλώσσας και συγκριτικά με το χρόνο που έγινε η ανάπτυξη θεωρώ ότι η προσπάθεια αυτή ήταν ικανοποιητική. Ωστόσο, πέρα από τα πλαίσια της διπλωματικής αυτής εργασίας στόχος είναι να συνεχιστεί η ανάπτυξη της εφαρμογής και να επεκταθεί σε βαθύτερα κομμάτια της γλώσσας.

Θα με χαροποιούσε ιδιαίτερα το γεγονός να ακούσω σχόλια και προτάσεις από συμφοιτητές μου, αλλά και να συνεργαστώ με οποιονδήποτε θέλει να ασχοληθεί είτε με την επέκταση της εφαρμογής είτε με τη δημιουργία μιας καινούριας. Όσοι λοιπόν βρήκατε ενδιαφέρον διαβάζοντας τις παραπάνω σελίδες ή παίζοντας με την εφαρμογή στο κινητό σας μη διστάσετε να επικοινωνήσετε μαζί μου.

Έχοντας, πλέον, περισσότερη εμπειρία στον προγραμματισμό android συσκευών, αλλά και στον κόσμο των γραφικών μπορώ να αναφέρω ως συμπέρασμα ότι είναι ένα πολύ όμορφο και ενδιαφέρον αντικείμενο να ασχοληθεί κανείς.

Βιβλιογραφία

1. Γραφικά Υπολογιστών με OpenGL ,Hearn Baker 3η έκδοση εκδόσεις Τζιόλα
2. OpenGL Programming Guide, Dave Shreiner 7η έκδοση
3. OpenGL SuperBible, Addison, Wesley 5η έκδοση
4. Pro OpenGL ES for Android, Smithwick, Verma
5. http://www.glprogramming.com/red/
6. http://www.glprogramming.com/blue/
7. http://www.opengl.org/archives/resources/faq/technical/
8. Σχετικά με την ανάπτυξη κώδικα: http://developer.android.com
9. Χρήσιμα OpenGL ES παραδείγματα στο android: http://insanitydesign.com/wp/projects/nehe-android-ports/
10. Texture mapping examples: http:/obviam.net
11. Android application development tutorials: http://www.mybringback.com/series/android-basics

Παράρτημα

Επίδειξη κώδικα

Για να μπορέσει ο αναγνώστης να δει πώς γίνονται προγραμματιστικά πολλά από τα μέρη που περιγράφηκαν στην εργασία, παρουσιάζεται ο κώδικας για το μέρος της θέασης και συγκεκριμένα για την εντολή glPerpsective(). Επιλέγεται το συγκεκριμένο μέρος γιατί συνδυάζει και κλάσεις γραμμένες με τη βιβλιοθήκη graphics, αλλά και κλάσεις γραμμένες με τη βοήθεια της OpenGL ES.

Ο κώδικας της PerspectiveActivity.java

```
package com.xroapp.openglTutorial;


import android.app.Activity;
import android.app.Dialog;
import android.content.Context;
import android.os.Bundle;
import android.os.Handler;
import android.util.Log;
import android.view.MotionEvent;
import android.view.View;
import android.view.View.OnTouchListener;
import android.widget.Button;
import android.widget.ImageButton;
import android.widget.TextView;
import android.widget.Toast;


public class PerspectiveActivity extends Activity {
        Handler mHandler;
        ViewingGLSurfaceView view;
        ExplainPerspective exp;
        TextView t,t2;
```

```
        Context context;
        Toast toast;
        int duration = Toast.LENGTH_SHORT;
        CharSequence text = "Near and Far parametres should take positive values";
    @Override
    public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      context = getApplicationContext()
            // arxikopoihsh toy perspective.xml
      setContentView(R.layout.perpective);
      view = (ViewingGLSurfaceView)findViewById(R.id.surView);
            // eyresh koympiwn;
      ImageButton ib = (ImageButton) findViewById(R.id.inc);
      ImageButton ib2 = (ImageButton) findViewById(R.id.inc2);
      ImageButton ib3 = (ImageButton) findViewById(R.id.inc3);
      ImageButton ib4 = (ImageButton) findViewById(R.id.inc4);
      ImageButton db = (ImageButton) findViewById(R.id.dec);
      ImageButton db2 = (ImageButton) findViewById(R.id.dec2);
      ImageButton db3 = (ImageButton) findViewById(R.id.dec3);
      ImageButton db4 = (ImageButton) findViewById(R.id.dec4);
            // arxikopoihs twn textfields
      t = (TextView) findViewById(R.id.show);
      t.setText("glPerspective("+sur.pRenderer.perAngle+", "+sur.pRenderer.aspect+",
"+sur.pRenderer.znear+","+sur.pRenderer.zfar+");");
      t2 = (TextView) findViewById(R.id.show2);
      t2.setText("glLookAt(0.0,0.0,"+sur.pRenderer.eyez+",0.0,0.0,0.0,0.0,1.0,0.0);");
      exp = (ExplainPerpective) findViewById(R.id.explainView);
      Button help = (Button)findViewById(R.id.help);
            // listeners
      ib.setOnTouchListener(new OnTouchListener() {
```

```
@Override
public boolean onTouch(View v, MotionEvent event) {

  switch(event.getAction()) {
   case MotionEvent.ACTION_DOWN:
      if (mHandler != null) return true;
      mHandler = new Handler();
      mHandler.postDelayed(mAction, 0);
      break;
   case MotionEvent.ACTION_UP:
      if (mHandler == null) return true;
      mHandler.removeCallbacks(mAction);
      mHandler = null;
      break;
   }

   return true;
  }

Runnable mAction = new Runnable() {
        @Override public void run() {
            float radians,degrees;
            if (sur.pRenderer.perAngle< 180){
                                sur.pRenderer.flag = true;
                                sur.pRenderer.perAngle++;
                                degrees = sur.pRenderer.perAngle;
                                radians = degrees *((float)Math.PI/180);
                                exp.rads = radians/2;
```

```
                                    t.setText("glPerspective("+sur.pRenderer.perAngle+",
"+sur.pRenderer.aspect+", "+sur.pRenderer.znear+","+sur.pRenderer.zfar+");");
                                    exp.invalidate();
                        }
                                    mHandler.postDelayed(this, 70);
                    }
                };
        });

            // oi upoloipoi listeners paraleipontai gia syntomia
}
```

Ο κωδικας του perspective.xml

```
<LinearLayout xmlns:tools="http://schemas.android.com/tools"
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/LL"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:background="#7D9EC0"
   android:gravity="bottom"
   android:orientation="vertical"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context=".Draw" >

   <FrameLayout
      android:layout_width="match_parent"
```

```
 android:layout_height="212dp" >

 < com.xroapp.openglTutorial .ViewingGLSurfaceView
    android:id="@+id/surView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:focusable="true"
    android:focusableInTouchMode="true" />
<RelativeLayout
android:layout_width="match_parent"
android:layout_height="match_parent">
 <LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
        android:orientation="vertical" >

    <TextView
       android:id="@+id/show"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="glPerspective" />

    <TextView
       android:id="@+id/show2"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="glLookAt" />

 </LinearLayout>
```

```
    <Button
    android:id="@+id/help"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_alignParentRight="true"
    android:text="Help" />
 </RelativeLayout>
</FrameLayout>

< com.xroapp.openglTutorial.ExplainPerpective
   android:id="@+id/explainView"
   android:layout_width="match_parent"
   android:layout_height="113dp"
   android:layout_weight="0.08"
   android:background="#FFFFFFFF" />

<LinearLayout
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:orientation="vertical" >

   <LinearLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:orientation="horizontal" >

      <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
```

```
    android:minWidth="57dp"
    android:padding="5dp"
    android:text="angle" />

<ImageButton
    android:id="@+id/inc"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:contentDescription="@string/desc3"
    android:src="@drawable/pluss"
    android:textColor="#0000FF" />

<ImageButton
    android:id="@+id/dec"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:contentDescription="@string/desc3"
    android:src="@drawable/minuss"
    android:textColor="#0000FF" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:minWidth="57dp"
    android:padding="5dp"
    android:text="near" />

<ImageButton
    android:id="@+id/inc2"
    android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:contentDescription="@string/desc3"
        android:src="@drawable/pluss"
        android:textColor="#0000FF" />

    <ImageButton
        android:id="@+id/dec2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:contentDescription="@string/desc3"
        android:src="@drawable/minuss"
        android:textColor="#0000FF" />
</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:minWidth="57dp"
        android:padding="5dp"
        android:text="far" />

    <ImageButton
        android:id="@+id/inc3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
    android:contentDescription="@string/desc3"
    android:src="@drawable/pluss"
    android:textColor="#0000FF" />

<ImageButton
    android:id="@+id/dec3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:contentDescription="@string/desc3"
    android:src="@drawable/minuss"
    android:textColor="#0000FF" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:minWidth="57dp"
    android:padding="5dp"
    android:text="camera" />

<ImageButton
    android:id="@+id/inc4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:contentDescription="@string/desc3"
    android:src="@drawable/pluss"
    android:textColor="#0000FF" />

<ImageButton
    android:id="@+id/dec4"
    android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
            android:contentDescription="@string/desc3"
            android:src="@drawable/minuss"
            android:textColor="#0000FF" />
      </LinearLayout>
   </LinearLayout>

</LinearLayout>
```

Ο κώδικας της ViewingGLSurfaceView.java

```
package com.xroapp.openglTutorial;

import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;
import android.content.Context;
import android.opengl.GLSurfaceView;
import android.util.Log;
import android.opengl.GLU;
import java.lang.Math;

class ViewingGLSurfaceView extends GLSurfaceView implements Renderer
{
   public static float fov_degrees =  45f;
   public static float fov_radians =  fov_degrees / 180 * (float) Math.PI;
   public float aspect;
   public static float camZ;
   boolean mTranslucentBackground;
   float mAngle;
   public float mAngleX;
   public float mAngleY;
```

```
    boolean draw = false;
    boolean flag = false;
    boolean useFrustrum = false,useOrtho = false;
    private GLText glText;
    private Context context;
    int getHeight,getWidth,dist;
    int commands=0;
    My3DTexture t3d;
    float perAngle = 45f;
    float mTransY ;
    float znear=5,zfar=30,eyez=10;
    float fH=3,fW=3;

public ViewingGLSurfaceView(boolean useTranslucentBackground,Context context)
{
        super(context);
        this.setRenderer(this);
        mTranslucentBackground = useTranslucentBackground;
        t3d = new My3DTexture();
}
public void onDrawFrame(GL10 gl)
{

        gl.glClear( GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT );
        gl.glMatrixMode(GL10.GL_PROJECTION); // Select The Projection Matrix
        gl.glLoadIdentity(); // Reset The Projection Matrix

        if(useFrustrum){
                gl.glFrustumf(-fW, fW, -fH, fH, znear, zfar);
                GLU.gluLookAt(gl, 0, 0, eyez, 0, 0, 0, 0, 1, 0);
```

```
        }else if (useOrtho){
                gl.glOrthof(-fW, fW, -fH, fH, znear, zfar);
            GLU.gluLookAt(gl, 0, 0, eyez, 0, 0, 0, 0, 1, 0);
        }  else{
                GLU.gluPerspective(gl, perAngle, aspect, znear,  zfar);
                GLU.gluLookAt(gl, 0, 0, eyez, 0, 0, 0, 0, 1, 0);
        }

        gl.glMatrixMode( GL10.GL_MODELVIEW );          // Activate Model View
        gl.glLoadIdentity();                        // Load Identity Matrix

        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);

        gl.glPushMatrix();
        gl.glRotatef(-90.0f,0.0f,0.0f,1.0f);
        t3d.draw(gl);
        gl.glPopMatrix();

        gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);

}
public void onSurfaceChanged(GL10 gl, int width, int height)
{
        getHeight= height;
        getWidth = width;
        dist = getHeight/10;
        aspect =  (float) width / (float) height;
         camZ = height / 2 / (float) Math.tan(fov_radians / 2);
        gl.glViewport(0, 0, width, height);
```

```
 }
public void onSurfaceCreated(GL10 gl, EGLConfig config) //15
{
      gl.glClearColor( 0.0f, 0.5f, 0.5f, 1.0f );
      glText = new GLText( gl, context.getAssets() );
     glText.load( "font.ttf", 14, 2, 2 );
     gl.glEnable(GL10.GL_CULL_FACE);
     t3d.loadGLTexture(gl, this.context);
     gl.glEnable(GL10.GL_TEXTURE_2D);
     gl.glShadeModel(GL10.GL_SMOOTH);          //Enable Smooth Shading
     gl.glClearDepthf(1.0f);                   //Depth Buffer Setup
     gl.glEnable(GL10.GL_DEPTH_TEST);          //Enables Depth Testing
     gl.glDepthFunc(GL10.GL_LEQUAL);           //The Type Of Depth Testing To Do
    gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10.GL_NICEST);
 }
}
```

Ο κώδικας της ExplainPerspective.java

```
package com.xroapp.openglTutorial;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Paint.Style;
import android.graphics.drawable.Drawable;
import android.util.AttributeSet;
import android.view.View;
```

```
public class ExplainPercpective extends View  {
   boolean working = true;
   Drawable mCustomImage;
   int i;
   Paint paint = new Paint();
   Bitmap b,b2;
   float znear = 50;
   float zfar=300;
   float rads = (float) (Math.PI/8);
   float eyez;
   float normalize;

   public ExplainPerspective(Context context) {
      super(context);
      paint.setColor(Color.BLUE);
      paint.setAntiAlias(true);
      paint.setTextSize(20.0f);
      paint.setStyle(Style.FILL);
      paint.setStrokeWidth(3.0f);
      b=BitmapFactory.decodeResource(getResources(), R.drawable.droid4);
      b2=BitmapFactory.decodeResource(getResources(), R.drawable.camera);

   }
   public ExplainPerspective(Context context, AttributeSet attrs) {
       super(context,attrs);
       paint.setColor(Color.BLUE);
       paint.setAntiAlias(true);
       paint.setTextSize(20.0f);
       paint.setStyle(Style.FILL);
       paint.setStrokeWidth(3.0f);
```

```
        b=BitmapFactory.decodeResource(getResources(), R.drawable.droid4);
        b2=BitmapFactory.decodeResource(getResources(), R.drawable.camera);
    }

    public ExplainPerspective(Context context, AttributeSet attrs, int defStyle) {
        super(context,attrs,defStyle);
        paint.setColor(Color.BLUE);
        paint.setAntiAlias(true);
        paint.setTextSize(20.0f);
        paint.setStyle(Style.FILL);
        paint.setStrokeWidth(3.0f);
        b=BitmapFactory.decodeResource(getResources(), R.drawable.droid4);
        b2=BitmapFactory.decodeResource(getResources(), R.drawable.camera);
    }

    @Override
    public void onDraw(Canvas canvas) {

            if (working){
                    eyez = getWidth()/5f;
                    working = false;
                    normalize =(float)Math.cos(rads);
            }

            canvas.drawLine(eyez, getHeight()/2f,
getWidth()/5f+getWidth()*(float)Math.cos(rads),getHeight()/2f+
getWidth()*(float)Math.sin(rads), paint);

            canvas.drawLine(eyez, getHeight()/2f, getWidth()/5f+getWidth()*(float)Math.cos(-
rads),getHeight()/2f+  getWidth()*(float)Math.sin(-rads), paint);

            canvas.drawText("near", eyez+(znear)*(float)Math.cos(rads),getHeight()/2f+
znear*(float)Math.sin(rads)+25, paint);
```

```
        canvas.drawText("far", eyez+(zfar)*(float)Math.cos(rads),getHeight()/2f+
(zfar)*(float)Math.sin(rads)+25, paint);

        canvas.drawLine(eyez+(znear)*(float)Math.cos(rads),getHeight()/2f+
znear*(float)Math.sin(rads), eyez+znear*(float)Math.cos(-rads),getHeight()/2f+
znear*(float)Math.sin(-rads), paint);

        canvas.drawLine(eyez+(zfar)*(float)Math.cos(rads),getHeight()/2f+
zfar*(float)Math.sin(rads), eyez+zfar*(float)Math.cos(-rads),getHeight()/2f+
zfar*(float)Math.sin(-rads), paint);



        canvas.drawBitmap(b, getWidth()/5f+(100)*(float)Math.cos(rads),getHeight()/2-10,
paint);

        canvas.drawBitmap(b2, eyez-25,getHeight()/2-10, paint);

   }


}
```

Η κλάση My3DTexture.java

```
package com.xroapp.openglTutorial;


import java.io.IOException;

import java.io.InputStream;

import java.nio.ByteBuffer;

import java.nio.ByteOrder;

import java.nio.FloatBuffer;

import javax.microedition.khronos.opengles.GL10;

import android.content.Context;

import android.graphics.Bitmap;

import android.graphics.BitmapFactory;

import android.opengl.GLUtils;

public class My3DTexture {
```

```
        /** The buffer holding the vertices */
        private FloatBuffer vertexBuffer;
        /** The buffer holding the texture coordinates */
        private FloatBuffer textureBuffer;
        /** The buffer holding the indices */
        private ByteBuffer indexBuffer;

        /** Our texture pointer */
        private int[] textures = new int[1];
        private float vertices[] = {          //Vertices according to faces
        -1.0f, -1.0f, 1.0f, /*Vertex 0*/      1.0f, -1.0f, 1.0f,  /*v1*/
        -1.0f, 1.0f, 1.0f,  /*v2 */           1.0f, 1.0f, 1.0f,   /*v3*/
        1.0f, -1.0f, 1.0f,                    1.0f, -1.0f, -1.0f,
        1.0f, 1.0f, 1.0f,                     1.0f, 1.0f, -1.0f,
        1.0f, -1.0f, -1.0f,                   -1.0f, -1.0f, -1.0f,
        1.0f, 1.0f, -1.0f,                    -1.0f, 1.0f, -1.0f,
       -1.0f, -1.0f, -1.0f,                   -1.0f, -1.0f, 1.0f,
      -1.0f, 1.0f, -1.0f,                     -1.0f, 1.0f, 1.0f,
      -1.0f, -1.0f, -1.0f,                    1.0f, -1.0f, -1.0f,
      -1.0f, -1.0f, 1.0f,                     1.0f, -1.0f, 1.0f,
      -1.0f, 1.0f, 1.0f,                      1.0f, 1.0f, 1.0f,
      -1.0f, 1.0f, -1.0f,                     1.0f, 1.0f, -1.0f,
};
    /** The initial texture coordinates (u, v) */
   private float texture[] = {
                                            //Mapping coordinates for the vertices
0.0f, 0.0f,                                 0.0f, 1.0f,
1.0f, 0.0f,                                 1.0f, 1.0f,
0.0f, 0.0f,                                 0.0f, 1.0f,
1.0f, 0.0f,                                 1.0f, 1.0f,
```

```
0.0f, 0.0f,                         0.0f, 1.0f,
1.0f, 0.0f,                         1.0f, 1.0f,
0.0f, 0.0f,                         0.0f, 1.0f,
.0f, 0.0f,                          1.0f, 1.0f,
0.0f, 0.0f,                         0.0f, 1.0f,
1.0f, 0.0f,                         1.0f, 1.0f,
0.0f, 0.0f,                         0.0f, 1.0f,
1.0f, 0.0f,                         1.0f, 1.0f,
};
   /** The initial indices definition */
   private byte indices[] = {
                    //Faces definition
                    0,1,3, 0,3,2,           //Face front
                    4,5,7, 4,7,6,           //Face right
                    8,9,11, 8,11,10,        //...
                    12,13,15, 12,15,14,
                    16,17,19, 16,19,18,
                    20,21,23, 20,23,22,
 };

          /**
           * The Cube constructor.
           */
          public My3DTexture() {
                    //
                    ByteBuffer byteBuf = ByteBuffer.allocateDirect(vertices.length * 4);
                    byteBuf.order(ByteOrder.nativeOrder());
                    vertexBuffer = byteBuf.asFloatBuffer();
                    vertexBuffer.put(vertices);
                    vertexBuffer.position(0);
```

```
        //
        byteBuf = ByteBuffer.allocateDirect(texture.length * 4);
        byteBuf.order(ByteOrder.nativeOrder());
        textureBuffer = byteBuf.asFloatBuffer();
        textureBuffer.put(texture);
        textureBuffer.position(0);

        //
        indexBuffer = ByteBuffer.allocateDirect(indices.length);
        indexBuffer.put(indices);
        indexBuffer.position(0);
}
public void draw(GL10 gl) {
        //Bind our only previously generated texture in this case
        gl.glBindTexture(GL10.GL_TEXTURE_2D, textures[0]);

        //Point to our buffers
        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);

        //Set the face rotation
        gl.glFrontFace(GL10.GL_CCW);

        //Enable the vertex and texture state
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer);
        gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, textureBuffer);

        //Draw the vertices as triangles, based on the Index Buffer information
```

```
                gl.glDrawElements(GL10.GL_TRIANGLES, indices.length,
GL10.GL_UNSIGNED_BYTE, indexBuffer);


                //Disable the client state before leaving
                gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
                gl.glDisableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
        }


        public void loadGLTexture(GL10 gl, Context context) {
                //Get the texture from the Android resource directory
                InputStream is =
context.getResources().openRawResource(R.drawable.droidd);
                Bitmap bitmap = null;
                try {
                        //BitmapFactory is an Android graphics utility for images
                        bitmap = BitmapFactory.decodeStream(is);


                } finally {
                        //Always clear and close
                        try {
                                is.close();
                                is = null;
                        } catch (IOException e) {
                        }
                }


                //Generate one texture pointer...
                gl.glGenTextures(1, textures, 0);
                //...and bind it to our array
                gl.glBindTexture(GL10.GL_TEXTURE_2D, textures[0]);
```

```
                //Create Nearest Filtered Texture

                gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MIN_FILTER,
GL10.GL_NEAREST);

                gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MAG_FILTER,
GL10.GL_LINEAR);



                //Different possible texture parameters, e.g. GL10.GL_CLAMP_TO_EDGE

                gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_S,
GL10.GL_REPEAT);

                gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_T,
GL10.GL_REPEAT);



                //Use the Android GLUtils to specify a two-dimensional texture image from
our bitmap

                GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmap, 0);



                //Clean up

                bitmap.recycle();

        }

}
```