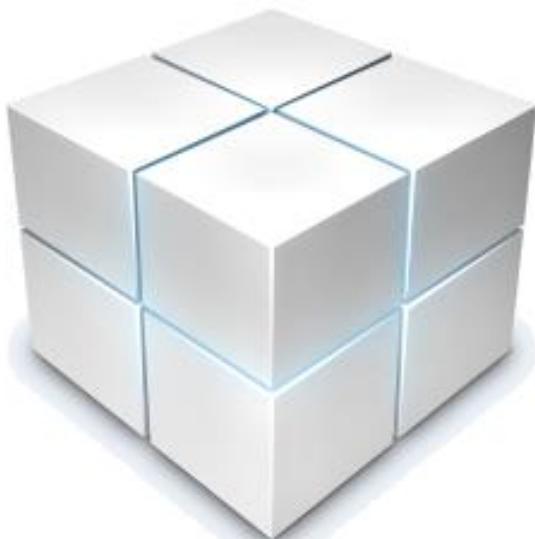




**ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ
ΑΝΑΤΟΛΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ ΚΑΙ ΘΡΑΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ**

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

**«ΜΕΛΕΤΗ, ΣΧΕΔΙΑΣΗ ΚΑΙ ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΗΣ
ΛΟΓΙΣΜΙΚΟΥ ΓΙΑ ΤΗΝ ΕΚΤΕΛΕΣΗ ΕΡΓΑΣΤΗΡΙΑΚΩΝ
ΑΣΚΗΣΕΩΝ ΣΤΟ ΜΑΘΗΜΑ ΓΡΑΦΙΚΑ ΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ
ΤΟΝ ΠΕΙΡΑΜΑΤΙΣΜΟ ΓΕΝΙΚΟΤΕΡΑ ΜΕ ΤΗΝ OPENGL»**



**Μενζιλτσίδου Στυλιανή (ΑΕΜ:1674)
Κορμούλη Ευανθία (ΑΕΜ:1718)**

Επιβλέπων Καθηγητής: Παχίδης Θεόδωρος

ΚΑΒΑΛΑ 2014

ΠΕΡΙΕΧΟΜΕΝΑ

Ευχαριστίες

Περίληψη πτυχιακής

Κεφάλαιο 1. Εισαγωγή στα γραφικά υπολογιστών

1.1 Εισαγωγή	1.1
1.1.1 Τι είναι τα γραφικά υπολογιστών	1.1
1.1.2. Ιστορική αναδρομή	1.1
1.1.3 Χρήση των Γραφικών σήμερα	1.2
1.2 Κατηγορίες γραφικών υπολογιστών	1.2
1.2.1 Δισδιάστατα (2D) γραφικά υπολογιστών	1.3
1.2.1.1 Διανυσματικά γραφικά (Vector Graphics)	1.3
1.2.1.2 Γραφικά χαρτογράφησης (Bitmap Graphics)	1.3
1.2.2 Τρισδιάστατα (3D) γραφικά υπολογιστών	1.4
1.2.3 Στατικά Γραφικά Υπολογιστών	1.4
1.2.4 Γραφικά Υπολογιστών Πραγματικού Χρόνου	1.4
1.3 Έννοιες και χαρακτηριστικά των γραφικών υπολογιστών	1.4
1.3.1 Εικονοστοιχείο (Pixel)	1.4
1.3.2 Απόδοση (Rendering)	1.5
1.3.2.1 3D Προβολή (3D projection)	1.5
1.3.2.2 Ανίχνευση ακτινών (Ray tracing)	1.5
1.3.2.3 Σκίαση (Shading)	1.5
1.3.2.4 Χαρτογράφηση υφής (Texture mapping)	1.6

1.4 Συσκευές Εξόδου	1.6
1.4.1 Συσκευές εμφάνισης γραφικών	1.6
1.4.1.1 Οθόνη	1.6
1.4.1.2 Συσκευές εκτύπωσης	1.7
1.4.1.3 Κάρτα Γραφικών	1.7
1.4.1.4 Κεντρική Μονάδα επεξεργασίας (CPU)	1.7
1.5 Οι δημοφιλέστερες γλώσσες δημιουργίας γραφικών σήμερα	1.7
1.5.1 Java	1.7
1.5.2 .NET	1.8
1.5.3 C#	1.8
1.5.4 C++	1.8
1.6 Παρόμοιες Εκπαιδευτικές Εφαρμογές Γραφικών	1.8
1.6.1 SeeGL	1.8
1.6.2 Processing	1.9
1.6.2.1 Τι είναι η Processing;	1.9
1.6.2.2 Η Processing και η Java	1.9
1.6.2.3 Χαρακτηριστικά της Processing	1.9
1.6.2.4 Η εφαρμογή της Processing	1.10
1.6.2.4.1 Το περιβάλλον ανάπτυξης	1.10
1.6.2.4.2 Βασικές λειτουργίες	1.10
1.6.2.4.3 Ένα τυπικό πρόγραμμα στη Processing	1.10
1.6.2.4.4 Παράδειγμα με την χρήση της Processing, δημιουργώντας τετράγωνα, κύκλους, τρίγωνα.	1.11
1.6.3 Μηχανή Erembus (Erembus Engine)	1.12
1.6.3.1 Εργαλεία που χρησιμοποιηθήκαν για τη δημιουργία της μηχανής	1.12
1.6.3.1.1 Γλώσσα Προγραμματισμού	1.12
1.6.3.1.2 Περιεχόμενο εφαρμογής	1.12

1.6.3.1.3 Απαιτήσεις συστήματος	1.13
1.6.3.1.4 Εικόνες από τη μηχανή	1.13
1.6.4 Virtual Supermarket	1.14
1.6.4.1 Λειτουργίες (κουμπιά) του εικονικού σουπερμάρκετ	1.15
1.6.4.2 Ανάπτυξη εφαρμογής	1.16
1.6.5 SEDEA	1.16
1.6.6.Stellarium	1.18
1.6.6.1 Χαρακτηριστικά του προγράμματος	1.19
1.6.6.2 Ανάπτυξη εφαρμογής και οι απαιτήσεις του συστήματος	1.19
1.7 Συγκρίσεις εφαρμογών	1.20
1.7.1 SeeGL vs Learning OpenGl	1.20
1.7.2 Processing vs Learning OpenGl	1.20
1.7.3 Erembus Engine vs Learning OpenGl	1.21
1.7.4 Virtual Supermarket Vs Learning OpenGl	1.21
1.7.5 Sedea vs Learning OpenGl	1.22
1.7.6 Stellarium vs Learning OpenGl	1.22
1.8 Υλικό διεξαγωγής για τη ανάπτυξη γραφικών	1.22
1.8.1 Το υλικό που χρησιμοποιήθηκε στην παρούσα πτυχιακή εργασία	1.23
1.9 Βιβλιοθήκες δημιουργίας γραφικών	1.25
1.9.1 Εισαγωγή	1.25
1.9.1.1 Τι είναι μια βιβλιοθήκη γραφικών	1.25
1.9.1.2 Τι είναι τα Graphics API	1.26
1.9.2 Διάφορες βιβλιοθήκες δημιουργίας γραφικών	1.26
1.9.2.1 OpenGl	1.26
1.9.2.2 Direct X	1.26
1.9.2.2.1 Χαρακτηριστικά της DirectX	1.28

1.9.2.2 Direct3D	1.29
1. 9.2.3 OpenGl vs Direct X.	1.30
1.9.3 Glide	1.30
1.9.4 QuickDraw 3D	1.31
1.9.5 Mantle	1.31
1.9.6 WebGL	1.31
1.9.7 Java 3D	1.31
1.9.8 QSDK	1.32
1.9.9 Simple DirectMedia Layer (SDL)	1.32
1.9.10 Βιβλιοθήκη SFML(Simple and Fast Multimedia Library)	1.32
1.9.11 Βιβλιοθήκη BPLcc_2D_Ansi_C.	1.33
1.9.12 Βιβλιοθήκη Allegro	1.33
1.9.13 Cairo	1.34
1.9.14 Mesa	1.35
1.9.15 Skia Graphics Engine	1.35
1.9.16 Weaver Framework	1.36
1.9.17 G2 2D Graphics Library	1.36
1.9.18 GD Βιβλιοθήκη	1.36
1.9.19 LibAfterImage	1.36
1.9.20 Libart	1.36
1.9.21 GraphiX	1.37
1.9.22 Dislin	1.37
1.9.23 GX2	1.37
1.10 ΣΥΜΠΕΡΑΣΜΑΤΑ	1.38

Κεφάλαιο 2. Η βιβλιοθήκη σχεδίασης γραφικών opengl και άλλες παρόμοιες βιβλιοθήκες

2.1 Εισαγωγή στην Opengl	2.1
2.2 Χαρακτηριστικά της OpenGL	2.1
2.3 Κατηγορίες Βιβλιοθηκών της OpenGL	2.3
2.3.1 Βιβλιοθήκες παρόμοιες με την glut	2.5
2.3.1.1 FREEGLUT	2.5
2.3.1.2 CPW	2.5
2.3.1.3 Fast Light Toolkit	2.6
2.3.1.4 GLFW	2.6
2.3.1.5 Εργαλειοθήκη GLOW	2.6
2.3.1.6 GLT OpenGL C++ Εργαλειοθήκη και GlutMaster	2.7
2.3.1.7 GLUI User Interface Library	2.8
2.3.1.8 NGL	2.10
2.3.1.9 SDL	2.10
2.4 Άλλες βιβλιοθήκες της Opengl	2.10
2.4.1 Βιβλιοθήκες Χαμηλότερου Επιπέδου της OpenGL	2.10
2.4.1 .1 OpenGL Mathematics (GLM)	2.10
2.4.1.2 Libktx	2.10
2.4.1.3 OpenSceneGraph	2.10
2.4.1.4 GLX	2.10
2.4.1.5 DRI	2.11
2.4.2 Βιβλιοθήκες Ανωτέρου Επιπέδου της OpenGL	2.11
2.4.2.1 Open Inventor by VSG	2.11
2.4.2.2 Coin	2.11
2.4.2.3 Gizmo 3D	2.12

2.4.2.4 OSG	2.12
2.4.2.5 OpenRM	2.12
2.5 Μέθοδος εκτέλεσης του κώδικα του χρήστη και διαδικασία εμφάνισης του γραφήματος στην οθόνη	2.12

Κεφάλαιο 3-Ανάλυση - περιγραφή του λογισμικού που έχει αναπτυχθεί.

3.1 Εισαγωγή	3.1
3.2 Επιλογή γλώσσας προγραμματισμού και πλατφόρμας για την ανάπτυξη της εφαρμογής	3.1
3.3 Προβλήματα και λύσεις	3.1
Πρόβλημα 3.3.1	3.1
Λύση προβλήματος 3.3.1	3.2
Πρόβλημα 3.3.2	3.2
Λύση προβλήματος 3.3.2	3.3
Πρόβλημα 3.3.3	3.5
Λύση προβλήματος 3.3.3	3.5
Πρόβλημα 3.3.4	3.6
Λύση προβλήματος 3.3.4	3.6
Πρόβλημα 3.3.5	3.6
Λύση προβλήματος 3.3.5	3.7
Πρόβλημα 3.3.6	3.8
Λύση προβλήματος 3.3.6	3.8
Πρόβλημα 3.3.7	3.8
Λύση προβλήματος 3.3.7	3.8
Πρόβλημα 3.3.8	3.8
Λύση προβλήματος 3.3.8	3.9
Πρόβλημα 3.3.9	3.10

Λύση προβλήματος 3.3.9	3.10
Πρόβλημα 3.3.10	3.11
Λύση προβλήματος 3.3.10	3.11
3.4 Απαιτήσεις και Προδιαγραφές της εφαρμογής	3.12
Απαίτηση 3.4.1	3.12
Απαίτηση 3.4.2	3.12
Απαίτηση 3.4.3	3.12
Απαίτηση 3.4.4	3.13
Απαίτηση 3.4.5	3.13
Απαίτηση 3.4.6	3.13
Απαίτηση 3.4.7	3.14
Απαίτηση 3.4.8	3.14
Απαίτηση 3.4.9	3.14
Απαίτηση 3.4.10	3.15
Απαίτηση 3.4.11	3.15
Απαίτηση 3.4.12	3.15
Απαίτηση 3.4.13	3.16
Απαίτηση 3.4.14	3.16
3.4.1 Προδιαγραφές της εφαρμογής	3.17
Προδιαγραφή 3.4.1.1	3.17
Προδιαγραφή 3.4.2.1	3.18
Προδιαγραφή 3.4.2.2	3.19
Προδιαγραφή 3.4.3.1	3.20
Προδιαγραφή 3.4.3.2	3.21
Προδιαγραφή 3.4.4.1	3.22
Προδιαγραφή 3.4.5.1	3.23

Προδιαγραφή 3.4.6.1	3.24
Προδιαγραφή 3.4.7.1	3.25
Προδιαγραφή 3.4.8.1	3.26
Προδιαγραφή 3.4.9.1	3.27
Προδιαγραφή 3.4.10.1	3.28
Προδιαγραφή 3.4.11.1	3.29
Προδιαγραφή 3.4.12.1	3.30
Προδιαγραφή 3.4.13.1	3.31
Προδιαγραφή 3.4.14.1	3.32
3.5 Διαγράμμα Δομής	3.33
3.5.1 Διαγράμματα Ροής Ελέγχου	3.34
3.5.2 Εικόνες Εφαρμογής	3.37
3.6 Εγχειρίδιο της OpenGL	3.38
3.6.1 Βασικές αρχές σχεδίασης	3.38
3.6.2 Τύποι δεδομένων	3.38
3.6.3 Σχηματισμός εντολών	3.38
3.6.4 Ιδιότητες	3.40
3.6.5 Ενταμιευτές και χρώματα	3.40
3.6.6 Βασικά Σχήματα	3.41
3.6.6.1 Καθορισμός Κορυφών	3.41
3.6.6.2 Σχεδίαση γραμμών	3.41
3.6.6.3 Διακεκομμένες γραμμές	3.43
3.6.6.4 Τρίγωνα	3.46
3.6.6.5 Ορθογώνια	3.48
3.6.6.6 Σχεδίαση κύκλων	3.49
3.6.6.7 Όψεις πολυγώνων	3.50

3.6.6.8 Καταστολή όψεων	3.51
3.6.6.9 Λίστες απεικόνισης (display lists)	3.52
3.6.6.10 Μητρώα σημείων - Μητρώα χρωμάτων	3.54
3.6.7 Μετασχηματισμοί συντεταγμένων	3.56
3.6.7.1 Μητρώα μετασχηματισμού	3.56
3.6.7.2 Μετασχηματισμός οπτικής γωνίας	3.61
3.6.7.3 Παράλληλη προβολή	3.62
3.6.7.3.1 Πλάγια παράλληλη προβολή	3.64
3.6.8 Απόδοση τρισδιάστατων σκηνών – Κινούμενα γραφικά	3.66
3.6.8.1 Ενταμιευτής βάθους	3.66
3.6.8.2 Τρισδιάστατες επιφάνειες	3.68
3.6.8.2.1 Κύβος	3.68
3.6.8.2.2 Σφαίρα	3.69
3.6.8.2.3 Κώνος	3.71
3.6.8.2.4 Κύλινδρος	3.72
3.6.8.2.5 Κυκλικός δίσκος – Δακτύλιος	3.73
3.6.8.2.6 Κυκλικός τομέας – Τομέας δακτυλίου	3.76
3.6.8.2.7 Τόρος	3.78
3.6.8.3 Μίξη χρωμάτων	3.79
3.6.8.4 Διπλή ενταμίευση	3.82
3.6.9 Φωτορεαλισμός	3.83
3.6.9.1 Πηγές φωτισμού	3.83
3.6.9.2 Κατηγορίες πηγών φωτισμού	3.85
3.6.9.3 Μοντελοποίηση πηγών φωτισμού στην OpenGL	3.86
3.6.9.4 Καθολικές παράμετροι φωτισμού	3.86
3.6.9.5 Μοντελοποίηση ανακλωσών επιφανειών στην OpenGL	3.86

3.6.9.6 Συντελεστές ανάκλασης περιβάλλοντος φωτισμού	3.87
3.6.9.7 Συντελεστές ανάκλασης διάχυτου φωτισμού	3.88
3.6.9.8 Συντελεστές κατοπτρικής ανάκλασης	3.89
3.6.9.9 Ατμοσφαιρικά εφέ (ομίχλη)	3.91
3.6.10 Απόδοση υφής στην OpenGL	3.94
3.6.10.1 Απόδοση υφής σε καμπύλες (1D)	3.94
3.6.10.2 Απόδοση υφής σε επιφάνειες (2D)	3.95
3.6.10.3 Ρυθμίσεις απόδοσης υφής	3.96
3.6.10.4 Συμίκρυνση υφής - Μεγέθυνση υφής	3.96
3.6.10.5 Αποκοπή υφής - Επανάληψη υφής	3.97
3.6.10.6 Αυτόματη απόδοση υφής σε τετραγωνικές επιφάνειες	3.101
3.6.10.7 Διαχείριση πολλαπλών μητρώων υφής	3.101
3.6.10.8 Αλληλεπίδραση υφής και μοντέλου σκίασης	3.104

Κεφάλαιο 4. Παραδείγματα εφαρμογής

4.1 Παρουσίαση των βασικών παραδειγμάτων της εφαρμογής και των αντίστοιχων απεικονίσεων τους	4.1
4.1.1 SimplePoint (Δημιουργία απλού σημείου)	4.1
4.1.2 SimpleLine (Δημιουργία γραμμής)	4.4
4.1.3 LineStrip (Ενωση διαδοχικών σημείων μεταξύ τους, εκτός του αρχικού και τελικού σημείου)	4.7
4.1.4 LineLoop (Ενωση διαδοχικών σημείων μεταξύ τους όπως επίσης ένωση του αρχικού και τελικού σημείου τους)	4.10
4.1.5 SimpleTriangle (Δημιουργία τριγώνου)	4.13
4.1.6 OpenGLFont ()	4.16
4.1.7 ColorChange(Παράδειγμα αλλαγής χρώματος)	4.19
4.1.8 Cube (Δημιουργία κύβου με χρώματα)	4.26
4.1.9 Cylinders (Δημιουργία κυλίνδρων)	4.29

4.1.10 Cone (Δημιουργία κινούμενου κώνου)	4.34
4.1.11 Octahedron (Δημιουργία Οκτάπλευρου)	4.37
4.1.12 Rotating Cube (Δημιουργία Περιστρεφόμενου Κύβου)	4.39
4.1.13 SolidSphereLight (Δημιουργία Συμπαγούς Φωτιζόμενης Σφαίρας)	4.42
4.1.14 AddSpotLight (Προσθήκη Φωτισμού Σε Κύβο)	4.45
4.2 Ενδεικτικά Παραδείγματα	4.51
4.2.1 Λειτουργία κώδικα χωρίς λάθη	4.48
4.2.2 Λειτουργία κώδικα με λάθη	4.50

Κεφάλαιο 5. Συμπεράσματα και μελλοντικές προτάσεις για βελτίωση

5.1 Συμπεράσματα	5.1
5.2 Μελλοντικές προτάσεις για βελτίωση	5.1
Παράρτημα Α – Εγχειρίδιο Εγκατάστασης Εφαρμογής	Π1.1
Παράρτημα Β-Εγχειρίδιο Χρήσης	Π2.1
Παράρτημα Γ-Πηγαίος κώδικας	

Κλάση FileManager	Π3.1
Κλάση OutputRedirect	Π3.2
Κλάση Program	Π3.4
Κλάση ZipStorer	Π3.4
Κώδικας newExampleForm για την φόρμα της εισαγωγής νέου παραδείγματος	Π3.15
Κώδικας της κύριας φόρμας της εφαρμογής	Π3.15
Κώδικας φόρμας επεξεργασίας παραδείγματος	Π3.27
Κώδικας φόρμας αποθήκευσης επεξεργασμένου παραδείγματος	Π3.29

ΒΙΒΛΙΟΓΡΑΦΙΑ

ΕΥΧΑΡΙΣΤΙΕΣ

Σε αυτό το σημείο θα θέλαμε να ευχαριστήσουμε πολύ τον επιβλέποντα καθηγητή για την πολύτιμη βοήθεια και υπομονή του. Μας καθοδήγησε σωστά στην ολοκλήρωση τόσο της εφαρμογής, αλλά και του βιβλίου της πτυχιακής εργασίας. Επίσης τις οικογένειες μας για την υπομονή και την στήριξη που μας πρόσφεραν κατά την διάρκεια όχι μόνο αυτής της πτυχιακής αλλά και καθόλη την διάρκεια των σπουδών μας. Ευχαριστούμε ιδιαιτέρως τις οικογένειες μας που προσπαθούν να μας προσφέρουν ένα καλύτερο μέλλον.

Ευχόμαστε η εφαρμογή να βοηθήσει όσο περισσότερο γίνεται τους φοιτητές στην καλύτερη κατανόηση του μαθήματος «Γραφικά Υπολογιστών». Ευχόμαστε οι φοιτητές να ζήσουν την φοιτητική τους ζωή γιατί είναι από τις καλύτερες περιόδους αλλά επίσης να εισπράξουν όσες περισσότερες γνώσεις μπορούν για το αντικειμενό τους και να το αγαπήσουν, για να βγουν μετά στην αγορά εργασίας με καλές προοπτικές.

Αυτή η πτυχιακή είναι αφιερωμένη σε μια καλή μας φίλη που δυστυχώς δεν πρόλαβε να πάρει το δικό της πτυχίο.

ΠΕΡΙΛΗΨΗ

Στο μάθημα Γραφικά Υπολογιστών του τμήματος Μηχανικών Πληροφορικής, δεν υπάρχει η δυνατότητα παρακολούθησης εργαστηρίου. Το μάθημα πραγματοποιείται σε θεωρητικό επίπεδο και έτσι ο φοιτητής δεν μπορεί να υλοποιήσει σε πρακτικό επίπεδο αυτά που διδάσκεται στη θεωρία. Για το λόγο αυτό δημιουργήθηκε η ανάγκη για την ανάπτυξη ενός εικονικού εργαστηρίου ώστε να διευκολύνει το φοιτητή στην καλύτερη κατανόηση του θεωρητικού μαθήματος «Γραφικά Υπολογιστών».

Σκοπός της πτυχιακής εργασίας είναι η δημιουργία μιας εύχρηστης εφαρμογής, η οποία δίνει τη δυνατότητα στο φοιτητή να παρακολουθεί μια λίστα με ομαδοποιημένα παραδείγματα, να αναπτύσσει κώδικα και να παρακολουθεί το αποτέλεσμα της εκτέλεσης του σε ένα νέο παράθυρο απεικόνισης, καθώς επίσης να ενσωματώνει δικά του παραδείγματα στην εφαρμογή.

Στα ομαδοποιημένα παραδείγματα της εφαρμογής περιέχουν περιπτώσεις απλών σχημάτων (απλή γραμμή, τρίγωνο, κ.ο.κ), τρισδιάστατα (3D) γραφικά (κύβος, κώνοι κ.ο.κ), παραδείγματα κίνησης και φωτοσκίασης (rendering). Τα συγκεκριμένα παραδείγματα ο χρήστης δεν μπορεί να τα τροποποιήσει. Ωστόσο έχει τη δυνατότητα α) να αναπτύξει δικό του κώδικα στην εφαρμογή, να τον «τρέχει» και να τον αποθηκεύει, β) να επεξεργάζεται τα έτοιμα παραδείγματα της εφαρμογής σε μια νέα φόρμα χρησιμοποιώντας ορισμένα κομμάτια του κώδικα τους και να τα ενσωματώσει στο δικό του γ) να παρατηρεί τυχόν λάθη που μπορεί να εντοπιστούν στον κώδικα του και να τα διορθώνει δ) να βλέπει το αποτέλεσμα του κώδικα που ανέπτυξε σε ένα παράθυρο απεικόνισης και γενικά να πειραματίζεται με την OpenGL. Τα αρχεία που δημιουργεί ο χρήστης αποθηκεύονται σε ένα ξεχωριστό φάκελο που υπάρχει στον σκληρό δίσκο του συστήματος και στο μενού των παραδειγμάτων. Επιπλέον, στην εφαρμογή υπάρχει «Βοήθεια» με εγχειρίδια χρήσης για το χρήστη και λειτουργίες επεξεργασίας κειμένου για μεγαλύτερη διευκόλυνσή του.

Κατά συνέπεια, η συγκεκριμένη εφαρμογή αποτελεί ένα αρχικό στάδιο εκμάθησης και κατανόησης του μαθήματος «Γραφικά Υπολογιστών».

ΚΕΦΑΛΑΙΟ 1

ΕΙΣΑΓΩΓΗ ΣΤΑ ΓΡΑΦΙΚΑ ΥΠΟΛΟΓΙΣΤΩΝ

1.1 Εισαγωγή

1.1.1 Τι είναι τα γραφικά υπολογιστών

Τα γραφικά υπολογιστών (computer graphics), είναι ένας επιστημονικός κλάδος της πληροφορικής που ασχολείται με τη θεωρία και την τεχνολογία αλγορίθμικής σύνθεσης εικόνων σε ηλεκτρονικό υπολογιστή. Με λίγα λόγια, είναι η δημιουργία, ο χειρισμός και η αποθήκευση γεωμετρικών αντικειμένων (μοντελοποίηση) και των εικόνων τους. Η απεικόνιση αυτών των εικόνων καθώς και η επεξεργασία τους γίνονται σε οθόνες ή σε συσκευές εκτύπωσης.

Με πιο απλά λόγια τα γραφικά είναι οπτικές παρουσιάσεις σε μια επιφάνεια, όπως μια οθόνη υπολογιστή. Παραδείγματα είναι οι φωτογραφίες, σχέδια, γραφικά σχέδια, χάρτες, μηχανολογικά σχέδια, ή άλλες εικόνες. Τα γραφικά συχνά συνδυάζουν κείμενο και εικονογράφηση [1].

1.1.2. Ιστορική αναδρομή

Τα γραφικά υπολογιστών είναι ένας κλάδος της επιστήμης υπολογιστών που τις τελευταίες αλλά και ειδικά τα τελευταία χρόνια αναπτύσσεται με γοργούς ρυθμούς. Για πρώτη φορά τα γραφικά υπολογιστών χρησιμοποιήθηκαν για στρατιωτικούς και ερευνητικούς σκοπούς. Ως έννοια χρησιμοποιήθηκε για πρώτη φορά στις αρχές της δεκαετίας του 1950. Στις αρχές της δεκαετίας του '70, αφενός αντιμετωπίζονταν δυσκολίες λόγω περιορισμένων δυνατοτήτων του υλικού (hardware) και αφετέρου οι υπολογιστές που μπορούσαν να χειριστούν γραφικά ήταν απαγορευτικά ακριβοί. Στην πραγματικότητα, μόνο πανεπιστήμια, μεγάλες εταιρίες και κυβερνητικοί παράγοντες είχαν τη δυνατότητα να έχουν στην κατοχή τους ισχυρά υπολογιστικά συστήματα. Η εξέλιξη τους ακολουθεί την εξέλιξη των ηλεκτρονικών υπολογιστών.

Η αρχή έγινε με το Whirlwind Computer το 1945 στο MIT υπό την εποπτεία του Jay Forrester ως μέρος του προγράμματος ASCA (Navy's Airplane Stability and Control Analyzer). Το σύστημα αυτό θα προσέφερε ένα προγραμματιζόμενο περιβάλλον εξομοιωτή πτήσης και η πρώτη παρουσίαση έγινε το 1951. Το σύστημα αυτό ήταν το πρώτο που είχε τη δυνατότητα απεικόνισης κειμένου και γραφικών σε «τάλαντοσκόπιο» σε πραγματικό χρόνο.

Τις δεκαετίες του 1950 και 1960, όταν άρχισαν να εμφανίζονται οι πρώτες τεχνολογίες ψηφιακών γραφικών, τα υπολογιστικά συστήματα και οι τεχνικές δημιουργίας εικόνων είχαν ακόμα αρκετά περιορισμένες δυνατότητες.

Τα επόμενα χρόνια δημιουργήθηκαν και άλλες εφευρέσεις. Κάποιες από αυτές που αξίζει να αναφερθούν είναι οι εξής:

- Το 1962 ο Jack Bresenham δημιουργεί έναν αλγόριθμο που απεικονίζει ευθείες γραμμές σε οθόνη με pixels, τον οποίο μετά επεκτείνει και για κύκλους.
- Στα τέλη της δεκαετίας του '60, ο Larry Roberts, δείχνει τη χρησιμότητα των ομοιογενών συντεταγμένων, των 4x4 πινάκων και τον αλγόριθμο αναγνώρισης κρυμμένων γραμμών.
- Το 1973 ο Bui Tuong Phong αναπτύσσει το μοντέλο φωτισμού Phong. Τεχνική που χρησιμοποιείται ακόμη και σήμερα και θεωρείται μια από τις πιο ρεαλιστικές τεχνικές φωτισμού.
- Το 1980, οι τεχνικές μοντελοποίησης και σκίασης σημειώνουν πρόοδο. Δημιουργούνται καινοτόμες τεχνικές προσέγγισης φωτορεαλισμού και συναρτήσεις υφής (texture) οι οποίες αποδίδουν με πιστότητα τη μορφολογία των επιφανειών.
- Στα τέλη της δεκαετίας του '80, παρουσιάζεται μια νέα δομή δεδομένων, τα BSP trees (Binary Space Partitioning). Παρότι αρχικά δεν αναγνωρίστηκε η χρησιμότητά της, αργότερα αποδείχτηκε μια από τις πιο σημαντικές δομές δεδομένων στα γραφικά υπολογιστών.
- Τα 3D γραφικά έγιναν πιο δημοφιλή στη δεκαετία του 1990 εξαιτίας της χρήσης τους στα παιχνίδια, στα πολυμέσα και στα κινούμενα σχέδια. Το 1995 δημιουργήθηκε το Toy Story, η πρώτη μεγάλου μήκους ταινία υπολογιστή που δημιουργείται από κινούμενα σχεδία. Το 1996, δημιουργήθηκε το Quake, ένα από τα πρώτα 3D παιχνίδια.
- Τη δεκαετία του 2000 τα γραφικά υπολογιστών που χρησιμοποιούνται στις ταινίες και τα βιντεοπαιχνίδια άρχισαν σταδιακά να γίνονται πιο «ρεαλιστικά», όπως τα παιχνίδια της Final Fantasy και η ταινία The Polar Express.

1.1.3 Χρήση των Γραφικών Σήμερα

Στη σημερινή εποχή τα γραφικά υπολογιστών έχουν μπει για τα καλά στη ζωή μας. Χρησιμοποιούνται για πάρα πολλούς σκοπούς και έχουν πληθώρα εφαρμογών.

Μερικά παραδείγματα είναι τα εξής:

- Τέχνη και ψυχαγωγία
- Κινηματογράφος, τηλεόραση, βιβλία, περιοδικά, ηλεκτρονικά παιχνίδια
- Επεξεργασία εικόνας
- Τροποποίηση εικόνας, απαλοιφή θορύβου
- Εικονική εξομοίωση
- Εξομοιωτές πτήσης, εικονική πραγματικότητα
- Σχεδιασμός
- Αρχιτεκτονικός σχεδιασμός, σχεδιασμός ηλεκτρονικών κυκλωμάτων
- Επιστημονική ανάλυση
- Ιατρική, βιολογία [2]

1.2 Κατηγορίες γραφικών υπολογιστών

Τα γραφικά υπολογιστών μπορούν να διακριθούν σε κατηγορίες, ανάλογα με κάποιο κριτήριο:

Με βάση το πλήθος των διαστάσεων οι οποίες συμμετέχουν στην απεικόνισή τους:

- Δισδιάστατα (2D) γραφικά υπολογιστών
 - Τρισδιάστατα (3D) γραφικά υπολογιστών
- Με βάση τη χρονική στιγμή κατά την οποία λαμβάνει χώρα η απόδοσή τους (rendering):
- Στατικά γραφικά υπολογιστών
 - Γραφικά υπολογιστών πραγματικού χρόνου [1]

1.2.1 Δισδιάστατα (2D) γραφικά υπολογιστών

Τα δισδιάστατα γραφικά υπολογιστών αφορούν την προσπάθεια απεικόνισης γραφικών δύο διαστάσεων στην οθόνη μιας ψηφιακής συσκευής (π.χ. ενός υπολογιστή). Συνήθως τέτοια γραφικά χρησιμοποιούνται για τη δημιουργία γραφικών διασυνδέσεων του χρήστη, αλλά και για εικονογραφήσεις βιβλίων, περιοδικών και λοιπών εντύπων. Μετά τη σύνθεσή τους, αποθηκεύονται σε ψηφιακά αρχεία εικόνας και η επιπλέον επεξεργασία τους παύει να αποτελεί αντικείμενο του πεδίου των γραφικών, αλλά της ψηφιακής επεξεργασίας εικόνας.

Τα δισδιάστατα γραφικά μπορούν να διαμεριστούν στις εξής δύο μεγάλες κατηγορίες:

1.2.1.1 Διανυσματικά γραφικά (Vector Graphics)

Χρησιμοποιούνται για τη δημιουργία εικόνων όπως λογότυποι, σήματα κατατεθέντα κ.τ.λ. Αποθηκεύονται ως μαθηματικοί τύποι αναλυτικής γεωμετρίας οι οποίοι περιγράφουν γεωμετρικά σχήματα, επομένως δεν ισχύει σε αυτά η έννοια της ανάλυσης εικόνας και μπορούν να μεγεθυνθούν / σμικρυνθούν χωρίς απώλεια ποιότητας.

1.2.1.2 Γραφικά χαρτογράφησης (Bitmap Graphics)

Όλα τα γραφικά που δημιουργούνται από ψηφιοποίηση υπαρκτών αντικειμένων (φωτογραφίες, εικόνες από σαρωτές κ.τ.λ.) ανήκουν σε αυτή την κατηγορία. Τα γραφικά αυτά λειτουργούν όπως ακριβώς ένα ψηφιδωτό: όσο μικρότερες και περισσότερες ψηφίδες χρησιμοποιούνται (μεγαλύτερη ανάλυση), τόσο πιο ευκρινές και ακριβές είναι το τελικό αποτέλεσμα. Η ανάλυση μετράται σε κουκκίδες (ψηφίδες) ανά ίντσα (dots per inch, dpi). Για μια οθόνη, η ανάλυση των 72 ή 96 dpi είναι επαρκέστατη, αν όμως η εικόνα προορίζεται για επαγγελματική εκτύπωση, το ελάχιστο απαιτούμενο είναι τα 300 dpi. Η απώλεια ποιότητας κατά τη μεγέθυνση μίας εικόνας είναι αντιστρόφως ανάλογη της ανάλυσής της. Επειδή οι οθόνες όπου τα γραφικά προβάλλονται χαρακτηρίζονται από τη δική τους ανάλυση, μία εικόνα με ανάλυση πολύ μικρότερη από της οθόνης πρέπει να προβληθεί σε μικρές διαστάσεις για να μη φαίνεται θολή και χωρίς ευκρίνεια.

Ένα ακόμη χαρακτηριστικό των γραφικών είναι το βάθος χρώματος, δηλαδή το πλήθος των δυαδικών ψηφίων (bits) που χρησιμοποιούνται για την περιγραφή του χρώματος κάθε ψηφίδας (ή κάθε περιοχής στα διανυσματικά γραφικά). Το σημερινό πρότυπο χρησιμοποιείται για τις οθόνες με βάθος χρώματος 24 bits (η οθόνη και η εκτύπωση χρησιμοποιούν διαφορετικά χρωματικά πρότυπα). Υπάρχουν και γραφικά με μεγαλύτερο βάθος χρώματος, που προορίζονται για ειδικές χρήσεις, καθώς το ανθρώπινο μάτι δεν μπορεί να διακρίνει περισσότερες από 16,7 εκατομμύρια χρωματικές διαβαθμίσεις.

Για τις εφαρμογές του Διαδικτύου χρησιμοποιούνται συνήθως γραφικά ψηφίδων, γιατί τα διανυσματικά γραφικά δεν υποστηρίζονται από παλαιότερες εκδόσεις φυλλομετρητών που χρησιμοποιούνται ακόμα από ένα αρκετά μεγάλο ποσοστό των χρηστών του Διαδικτύου.

Ο τύπος των γραφικών αναγνωρίζεται συνήθως από την επέκταση του ονόματος του αρχείου, στο οποίο είναι αποθηκευμένα (δηλ. το τμήμα εκείνο του ονόματος που βρίσκεται δεξιά από την τελεία που χωρίζει στα δύο το όνομα ενός αρχείου). Οι πλέον συνήθεις τύποι είναι:

- Διανυσματικά γραφικά:.svg,.cdr,.ai
- Γραφικά χαρτογράφησης: .tif, .bmp, jpg, .gif, .png (οι τρεις τελευταίοι τύποι είναι οι κατάλληλοι για το Διαδίκτυο).

1.2.2 Τρισδιάστατα (3D) γραφικά υπολογιστών

Τα τρισδιάστατα γραφικά υπολογιστών αποτελούν προσπάθειες απεικόνισης γραφικών τριών διαστάσεων στην οθόνη – απεικόνισης δυο διαστάσεων – μιας ψηφιακής συσκευής (π.χ. ενός υπολογιστή). Το γεγονός ότι η απεικόνιση χρησιμοποιεί τρεις διαστάσεις τα καθιστά ιδιαίτερα ρεαλιστικά. Τέτοιου είδους γραφικά χρησιμοποιούνται συνήθως από προγράμματα όπως παιχνίδια υπολογιστών, εικονικούς κόσμους. Τα τρισδιάστατα γραφικά βρίσκουν επίσης εφαρμογή στον κινηματογράφο, για τη δημιουργία σκηνών εικονικών κόσμων αλλά και ειδικών εφέ, που είναι αδύνατον να γυριστούν ως πραγματικές σκηνές.

1.2.3 Στατικά Γραφικά Υπολογιστών

Τα στατικά γραφικά υπολογιστών αποτελούν αντικείμενα γραφικών τα οποία δεν αποδίδονται τη στιγμή που εκτελούνται αλλά έχουν αποδοθεί μία φορά κατά τη δημιουργία τους. Παράδειγμα τέτοιων γραφικών είναι τα μικρά βίντεο, τα οποία εμφανίζονται σε διάφορα παιχνίδια, και τα οποία έχουν «γυριστεί» μια φορά και κάθε φορά που θα τα παρακολουθήσουμε παραμένουν ίδια. Για τη δημιουργία τους χρησιμοποιείται κάποιο πρόγραμμα δημιουργίας γραφικών και κίνησης (animation) όπως το 3dStudio Max, το Maya, το Lightwave, το Blender, το cinema4d κτλ.

1.2.4 Γραφικά Υπολογιστών Πραγματικού Χρόνου

Τα γραφικά υπολογιστών πραγματικού χρόνου αποτελούν αντικείμενα γραφικών τα οποία αποδίδονται τη στιγμή που εκτελούνται. Για παράδειγμα τα γραφικά που εμφανίζονται στην οθόνη ενός υπολογιστή, ο οποίος εκτελεί ένα παιχνίδι, ανήκουν συνήθως σε αυτήν την κατηγορία. Για τη δημιουργία τους απαιτείται κάποια μηχανή απόδοσης γραφικών (graphics rendering engine) πραγματικού χρόνου, όπως για παράδειγμα το Ogre3d, το Irrlicht, το Crystal Space κ.τ.λ. [1].

1.3 Έννοιες και χαρακτηριστικά των γραφικών υπολογιστών

1.3.1 Εικονοστοιχείο (Pixel)

Σε ένα μεγεθυσμένο τμήμα εικόνας, το κάθε εικονοστοιχείο (pixel) αποδίδεται σαν τετράγωνο.

Στην ψηφιακή απεικόνιση, ένα pixel αποτελεί ένα μοναδικό σημείο σε μια ψηφιακή εικόνα. Τα pixel βασίζονται στην τακτική πλέγματος 2D και συχνά απεικονίζονται με κουκκίδες ή τετράγωνα.

Κάθε pixel αποτελεί ένα δείγμα μιας αρχικής εικόνας και το σύνολο αυτών των δειγμάτων αναπαριστούν μια ακριβέστερη απεικόνιση του πρωτότυπου. Η ένταση κάθε pixel ποικίλει: όσον αφορά τα χρώματα, κάθε pixel συνήθως διαθέτει τρία χρώματα, το κόκκινο, το μπλε και το πράσινο.

1.3.2 Απόδοση (Rendering)

Στην απόδοση υπάρχει η δημιουργία ενός 3D μοντέλου με τη βοήθεια των προγραμμάτων ηλεκτρονικών υπολογιστών. Ένα αρχείο σκηνής περιέχει αντικείμενα σε μια καθορισμένη γλώσσα ή σε μια δομή δεδομένων. Μπορεί να περιέχει δεδομένα όπως τη γεωμετρία, την όψη, την υφή, το φωτισμό και τη σκίαση. Στη συνέχεια τα δεδομένα που περιέχονται στο αρχείο σκηνής μεταφέρονται σε ένα πρόγραμμα για την επεξεργασία των δεδομένων αυτών και τέλος την αναπαραγωγή και απεικόνιση της ψηφιακής εικόνας. Το πρόγραμμα απόδοσης βρίσκεται συνήθως ενσωματωμένο στο λογισμικό γραφικών του υπολογιστή.

Επιπλέον η απόδοση έχει τα εξής χαρακτηριστικά:

1.3.2.1 3D Προβολή (3D projection)

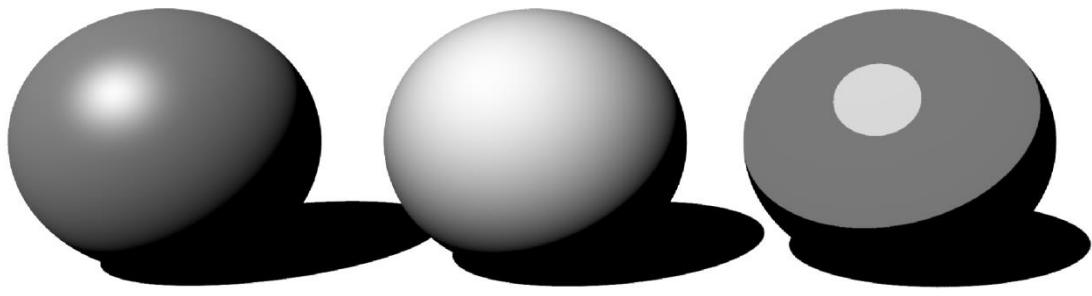
Είναι μια μέθοδος χαρτογράφησης τρισδιάστατων σημείων σε ένα δισδιάστατο επίπεδο. Η χρήση αυτού του τύπου προβολής είναι ευρέως διαδεδομένη, ιδίως σε γραφικά ηλεκτρονικών υπολογιστών.

1.3.2.2 Ανίχνευση ακτινών (Ray tracing)

Είναι μια τεχνική για τη δημιουργία μιας εικόνας με τον εντοπισμό της διαδρομής του φωτός μέσα από τα pixels σε ένα επίπεδο εικόνας. Η τεχνική είναι ικανή να παράγει μια υψηλή ποιότητα φωτορεαλισμού. Συνήθως είναι υψηλότερη από εκείνη των τυπικών μεθόδων απόδοσης, αλλά έχει μεγάλο κόστος.

1.3.2.3 Σκίαση (Shading)

Η Σκίαση αναπαριστά σε βάθος την απεικόνιση με διαφορετικά επίπεδα του σκότους για τα 3D γραφικά. Είναι μια διαδικασία που χρησιμοποιείται στη σχεδίαση για την απεικόνιση των επιπέδων του σκότους με την εφαρμογή πολυμέσων. Υπάρχουν διάφορες τεχνικές σκίασης συμπεριλαμβανομένων τη διαγώνια σκίαση όπου οι κάθετες γραμμές διαφοροποιούνται σε ένα μοτίβο πλέγματος, ώστε να σκιαστεί μια περιοχή. Όσο πιο κοντά είναι οι γραμμές μεταξύ τους, τόσο πιο σκιασμένη εμφανίζεται η περιοχή. Παρόμοια, όσο μεγαλύτερη είναι η απόσταση μεταξύ των γραμμών, τόσο φωτεινότερη εμφανίζεται η περιοχή.

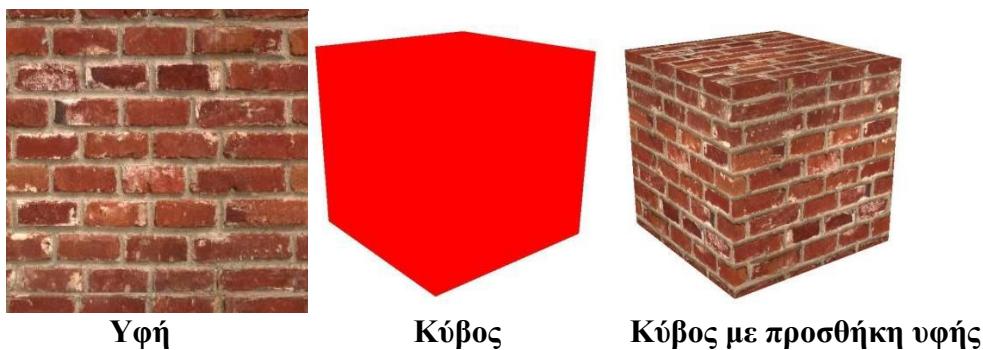


Σχήμα 1.1 Παράδειγμα Σκίασης

1.3.2.4 Χαρτογράφηση υφής (Texture mapping)

Αποτελεί μια μέθοδο που προσθέτει χρώμα και υφή στην επιφάνεια 3D γραφικών και γενικά σε γραφικά υπολογιστών. Γενικά επιτρέπει την προσαρμογή εικόνων σε ένα ή περισσότερα πολύγωνα, ώστε να γίνουν πιο «ρεαλιστικά».

Το texture mapping φέρνει όλο το ρεαλισμό σε διάφορες εφαρμογές γραφικών (παιχνίδια, εξομοιωτές, εικονική πραγματικότητα κ.α.). Οι υφές ουσιαστικά είναι ορθογώνιοι πίνακες από δεδομένα. Το κάθε κομμάτι δεδομένων ονομάζεται texel. Παρόλο που οι υφές είναι τετράγωνοι πίνακες μπορούν να προσαρμοστούν και σε μη τετραγωνικές επιφάνειες (κύκλοι, κύλινδροι) αλλά σε οποιοδήποτε γεωμετρικό αντικείμενο. Επίσης διαθέτουν συντεταγμένες. Η ελάχιστη τιμή που μπορεί να πάρουν είναι 0 και η μέγιστη 1.



Σχήμα 1.2 Παράδειγμα texture mapping

1.4 Συσκευές Εξόδου

1.4.1 Συσκευές εμφάνισης γραφικών

1.4.1.1 Οθόνη

Η οθόνη του υπολογιστή είναι μια συσκευή με τη δυνατότητα παρουσίασης εικόνων. Αποτελεί απαραίτητο μέσο για τη δημιουργία γραφικών και την απεικόνισή τους. Το μέγεθος της οθόνης, μετριέται σε ίντσες και εκφράζει την απόσταση μεταξύ της κάτω

και της απέναντι επάνω γωνίας του πλαισίου απεικόνισης. Όσο μεγαλύτερη είναι αυτή η τιμή τόσο μεγαλύτερη είναι και η εικόνα που παρουσιάζεται στην οθόνη.

1.4.1.2 Συσκευές εκτύπωσης

- Inkjet εκτυπωτής
- Laser εκτυπωτής
- 3D εκτυπωτές
- Σαρωτής [3]

1.4.1.3 Κάρτα Γραφικών

Η κάρτα γραφικών είναι το τμήμα του υπολογιστή, που λαμβάνει δεδομένα από την Κεντρική Μονάδα Επεξεργασίας (CPU) για να τα μετατρέψει σε εικόνα, η οποία θα προβληθεί στην οθόνη και αποτελεί το πιο σημαντικό κομμάτι των γραφικών.

Ο επεξεργαστής γραφικών της κάρτας, που ονομάζεται μονάδα επεξεργασίας γραφικών (Graphics Processing Unit, GPU), είναι παρόμοιος με τον επεξεργαστή ενός υπολογιστή. Διαθέτει επίσης μνήμη RAM.

Επιπλέον θα πρέπει να δοθεί ιδιαίτερη σημασία στο είδος του bandwidth (εύρος ζώνης) που προσφέρει. Το bandwidth μιας κάρτας γραφικών προσδιορίζει την ποσότητα των δεδομένων που μπορεί να στείλει στον υπολογιστή κατά τη διάρκεια ενός δευτερολέπτου. Με απλά λόγια, το bandwidth είναι η ταχύτητα της μνήμης VRAM (VideoRAM) της κάρτας γραφικών και μετριέται σε GB/s (gigabytes ανά δευτερόλεπτο). Μια κάρτα γραφικών με μεγαλύτερο bandwidth μπορεί να «σχεδιάζει» γραφικά γρηγορότερα, καθώς και να «σχεδιάζει» μεγαλύτερης ποιότητας εικόνες. Οι τιμές των καρτών γραφικών ποικίλουν ανάλογα με την απόδοση και την ταχύτητα τους, διότι ο καθένας επιλέγει κάρτες γραφικών ανάλογα με τις ανάγκες που θέλει να καλύψει (π.χ. τα ηλεκτρονικά παιχνίδια).

1.4.1.4 Κεντρική Μονάδα επεξεργασίας (CPU)

Η **Κεντρική Μονάδα Επεξεργασίας - KME** (αγγλικά: Central Processing Unit - CPU) είναι το κεντρικό εξάρτημα ενός ηλεκτρονικού υπολογιστή, και συχνά αναφέρεται απλά ως επεξεργαστής. Η KME ελέγχει τη λειτουργία του υπολογιστή και εκτελεί τις λειτουργίες επεξεργασίας δεδομένων. Στην ουσία ο επεξεργαστής όσον αφορά τα γραφικά, «αποφασίζει» τι θα κάνει με το κάθε pixel στην οθόνη.

Οι μεγαλύτεροι κατασκευαστές μικροεπεξεργαστών Intel και AMD ενσωμάτωσαν τις εντολές MMX στους επεξεργαστές τους και αργότερα τις SSE SSE2 SSE3 SSE4 SSE5 3DNow! και 3DNow!. Αυτά είναι σύνολο εντολών SIMD (Single Instruction, Multiple Data) τα οποία σχεδιάστηκαν για να κάνουν πράξεις με αριθμούς κινητής υποδιαστολής, διανύσματα και γενικά μαθηματικές πράξεις που χρησιμοποιούνται σε παιχνίδια και multimedia εφαρμογές [4].

1.5 Οι δημοφιλέστερες γλώσσες δημιουργίας γραφικών σήμερα

1.5.1 Java

Η Java αναπτύχθηκε από την Sun Microsystems το 1996. Υποστηρίζεται σε πολλές πλατφόρμες όπως τα Windows, UNIX, Linux, κλπ. Ο πηγαίος κώδικας

μεταγλωττίζεται σε κώδικα που τρέχει σε μια εικονική μηχανή της Java (JVM). Η σύνταξη της Java προέρχεται από την γλώσσα προγραμματισμού C.

1.5.2 .NET

Αποτελεί ένα συγκεκριμένο πλαίσιο λογισμικού της Microsoft και δημιουργήθηκε στα τέλη της δεκαετίας του '90. Έχει το πλεονέκτημα της δυνατότητας επιλογής μεταξύ των γλωσσών C++, C#, Visual Basic και άλλων γλωσσών, ή ακόμη και να «αναμειχθούν» αυτές οι γλώσσες. Το μειονέκτημα είναι η περιορισμένη επιλογή πλατφορμών (τρέχει μόνο σε περιβάλλον Windows).

Η Microsoft έχει αναπτύξει ένα ολοκληρωμένο περιβάλλον για .NET λογισμικό που είναι το Microsoft Visual Studio.

1.5.3 C#

Η C# είναι μία ολοκληρωμένη αντικειμενοστραφής γλώσσα προγραμματισμού που δημιουργήθηκε από τη Microsoft τον Ιούλιο του 2000. Είναι σχεδιασμένη για τη δημιουργία λογισμικού σε .Net Framework.

Τα πάντα στη C# είναι αντικείμενα και παρέχει άμεση πρόσβαση σε τεράστιες βιβλιοθήκες κλάσεων του .Net Framework και ασφάλεια των τύπων της.

1.5.4 C++

Η C++ είναι μια γενικού σκοπού γλώσσα προγραμματισμού και αναπτύχθηκε από τον Μπιάρνε Στρούτρουπ το 1979. Θεωρείται μέσου επιπέδου γλώσσα, καθώς περιλαμβάνει έναν συνδυασμό χαρακτηριστικών από γλώσσες υψηλού και χαμηλού επιπέδου. Είναι μια μεταγλωττιζόμενη γλώσσα πολλαπλών παραδειγμάτων, με τύπους. Υποστηρίζει δομημένο, αντικειμενοστραφή και γενικό προγραμματισμό ως βελτίωση της ήδη υπάρχουσας γλώσσας προγραμματισμού C, και αρχικά ονομάστηκε «C with Classes», δηλαδή C με Κλάσεις. Μετονομάστηκε σε C++ το 1983. Σήμερα τείνει να είναι η γλώσσα της επιλογής για τη δημιουργία γραφικών και παιχνιδιών.

1.6 Παρόμοιες Εκπαιδευτικές Εφαρμογές Γραφικών

1.6.1 SeeGL

Σε αυτή την εφαρμογή παρουσιάζεται η ανάπτυξη του λογισμικού εργαλείου SeeGL. Πρόκειται για ένα εκπαιδευτικό εργαλείο, που προορίζεται κυρίως για να χρησιμοποιηθεί από χρήστες που θέλουν να γνωρίσουν τη βιβλιοθήκη γραφικών OpenGL. Ο χρήστης μπορεί να συντάξει ένα πρόγραμμα OpenGL, παρατηρώντας την κατάσταση όλων των στοιχείων που είναι βασισμένα στην OpenGL, καθώς και τις αλλαγές που προκύπτουν από την εκτέλεση κάθε εντολής και να δει την τελική εικόνα. Σε περίπτωση απροσδόκητου αποτελέσματος, με τη βοήθεια ενός οπτικού έλεγχου, ακολουθώντας το αρχείο εργασιών ο χρήστης θα μπορεί να ανακαλύψει αν το πρόβλημα προκλήθηκε από λάθους παραμέτρους ή από κάποια λάθος εντολή [5].

1.6.2 Processing

1.6.2.1 Τι είναι η Processing;

Η Processing είναι μια γλώσσα προγραμματισμού ανοικτού κώδικα και παράλληλα ένα **προγραμματιστικό περιβάλλον** για ανθρώπους που θέλουν να προγραμματίσουν εικόνες, animation και ήχο. Όλα ξεκίνησαν το 2001 όταν δύο απόφοιτοι του πανεπιστημίου MIT, Benjamin Fry και Casey Reas ξεκίνησαν την ανάπτυξη της γλώσσας Processing πάνω σε Java. Παρόλο που η γλώσσα αναπτύχθηκε στη Java, το συντακτικό της είναι απλουστευμένο και το προγραμματιστικό της μοντέλο βασίζεται στα γραφικά. Αρχικά αναπτύχθηκε σαν ένα σχεδιαστικό πρόγραμμα και για να διδάξει βασικές αρχές προγραμματισμού μέσα σε ένα οπτικό πλαίσιο, αλλά στη συνέχεια εξελίχθηκε σε ένα εργαλείο δημιουργίας ολοκληρωμένων επαγγελματικών εργασιών. Αυτή την στιγμή υπάρχουν δεκάδες χιλιάδες σπουδαστές, καλλιτέχνες, σχεδιαστές, ερευνητές και «χομπίστες» που χρησιμοποιούν την Processing για διδασκαλία, προτυποποίηση και παραγωγή.

1.6.2.2 Η Processing και η Java

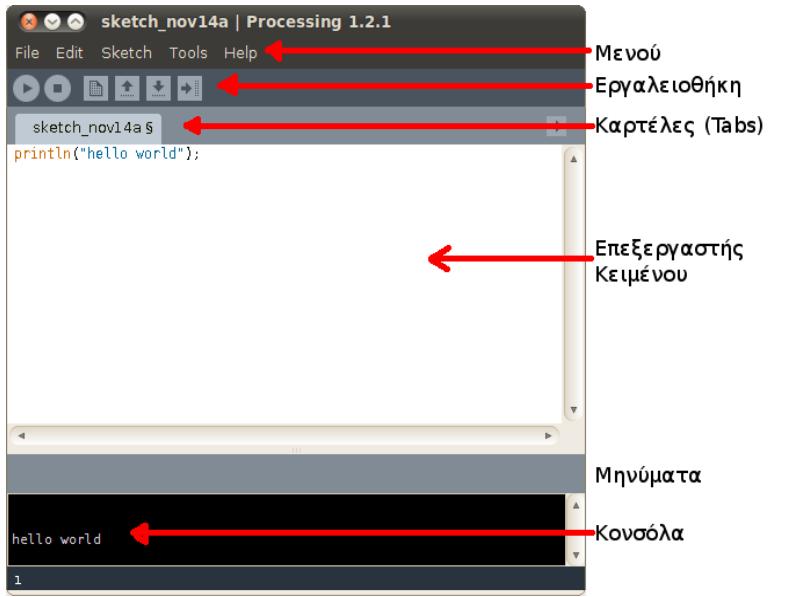
Το περιβάλλον της Processing είναι γραμμένο σε Java. Προγράμματα που έχουν γραφτεί στην Processing είναι επίσης μεταφρασμένα σε Java και έπειτα τρέχουν σαν Java προγράμματα. Προγράμματα που είναι γραμμένα σε Java και Processing, συνήθως τρέχουν γρηγορότερα από προγράμματα που βασίζονται σε scripting γλώσσες όπως ActionScript και Lingo, κάτι το οποίο είναι πολύ σημαντικό για εφαρμογές με γραφικά. Μεγάλες διακρίσεις μεταξύ του Processing και της Java είναι οι βιβλιοθήκες γραφικών της Processing και το απλοποιημένο στυλ προγραμματισμού που δεν προϋποθέτει οι χρήστες να καταλαβαίνουν πιο προηγμένες έννοιες όπως κλάσεις, αντικείμενα, ή animation (ενώ εξακολουθούν να είναι διαθέσιμες για προχωρημένους χρήστες). Τέτοιες τεχνικές λεπτομέρειες πρέπει να είναι ειδικά προγραμματισμένες σε Java, αλλά ολοκληρωμένες στην Processing, κάνοντας τα προγράμματα μικρότερα και ευκολότερα να διαβαστούν.

1.6.2.3 Χαρακτηριστικά της Processing

- Είναι ελεύθερο/ανοικτό λογισμικό με άδεια χρήσης GPL/LGPL.
- Είναι πολυπλατφορμική, μπορεί να τρέξει σε λειτουργικά συστήματα GNU/Linux, Mac OS X και Windows.
- Δημιουργεί διαδραστικά προγράμματα χρησιμοποιώντας δισδιάστατα (2D) ή τρισδιάστατα (3D) γραφικά.
- Ενσωματώνει την OpenGL για επιτάχυνση 3D.
- Δημιουργεί stand-alone desktop εφαρμογές και web-based εφαρμογές (applets).
- Υπάρχουν αρκετές βιβλιοθήκες επέκτασης της γλώσσας, για εφαρμογές ήχου, βίντεο, τεχνητής ορασης, κ.α
- Η Processing βασίστηκε στις δυνατότητες γραφικών της γλώσσας προγραμματισμού Java, απλοποιώντας τη χρήση και δημιουργώντας νέα χαρακτηριστικά.

1.6.2.4 Η εφαρμογή της Processing

1.6.2.4.1 Το περιβάλλον ανάπτυξης



Σχήμα 1.3: Το περιβάλλον ανάπτυξης

1.6.2.4.2 Βασικές λειτουργίες

	Έλεγχος του κώδικα για λάθη/Εκτέλεση του προγράμματος.
	Τερματισμός εκτέλεσης του προγράμματος.
	Δημιουργία νέου έργου.
	Εμφάνιση μενού με τα αποθηκευμένα έργα. Πατώντας σε ένα από αυτά ανοίγει το έργο για επεξεργασία.
	Αποθήκευση του έργου.
	Δημιουργία αυτόνομης εφαρμογής (Export).

Σχήμα 1.4 Βασικές λειτουργίες

1.6.2.4.3 Ένα τυπικό πρόγραμμα στη Processing έχει την παρακάτω δομή:

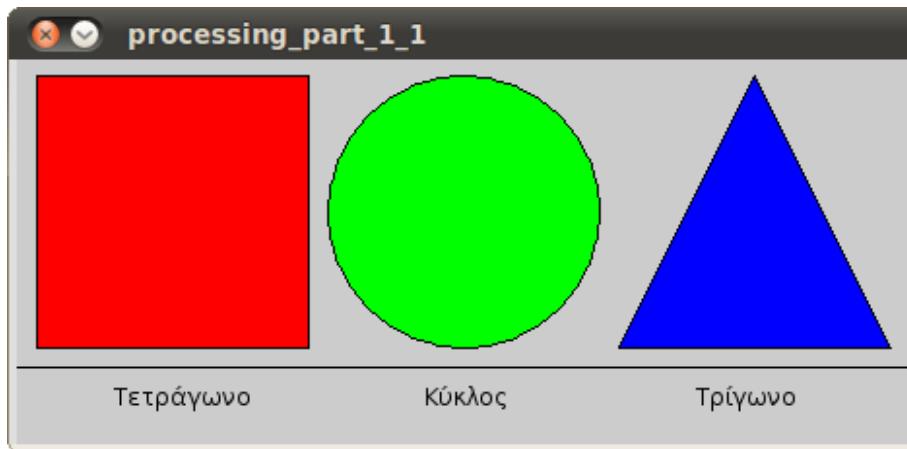
```
// δήλωση μεταβλητών
void setup() {
// αρχικοποίησεις
}

void draw() {
// κώδικας
}
```

Όπως παρατηρείται υπάρχουν δυο βασικές συναρτήσεις σε ένα τυπικό πρόγραμμα της Processing. Η συνάρτηση `setup()` εκτελείται στην αρχή του προγράμματος και για μία μόνο φορά. Χρησιμοποιείται για τις αρχικοποιήσεις ιδιοτήτων και βιβλιοθηκών, όπως για παράδειγμα: το μέγεθος της οθόνης, το χρώμα φόντου, το φόρτωμα εικόνων κ.α. Οι μεταβλητές που δηλώνονται στη `setup()` δεν είναι προσβάσιμες από άλλες συναντήσεις, συμπεριλαμβανομένης και της `draw()`. Η συνάρτηση `draw()` εκτελείται αμέσως μετά τη `setup()` και ο κώδικας που γράφεται μέσα στη συνάρτηση αυτή επαναλαμβάνεται συνεχώς μέχρι να τερματιστεί η εφαρμογή ή μέχρι να κληθεί η συνάρτηση `noLoop()` [6], [7].

1.6.2.4.4 Παράδειγμα με την χρήση της Processing, δημιουργώντας τετράγωνα, κύκλους, τρίγωνα

```
void setup() {
    size(460, 200, JAVA2D); // size(x, y, renderer) – Ορίζει το μέγεθος του παραθύρου και τον renderer που θα χρησιμοποιηθεί για τον σχεδιασμό των αντικειμένων. Στο παράδειγμα χρησιμοποιείται η JAVA2D για redering γραφικών δυο διαστάσεων (x, y)
    colorMode(RGB, 255); // Ορίζει τον τρόπο με τον οποίο η Processing μεταφράζει τα χρώματα. Στό παράδειγμα ορίζει τα χρώματα βάση του μοντέλου RGB (Red, Green, Blue) και τιμές χρώματος από 0 ως 255.
    rectMode(CORNERS); // Προσδιορίζει τον τρόπο με τον οποίο θα σχεδιαστεί ένα τετράγωνο. Υπάρχουν οι μέθοδοι CORNER, CENTER, CENTERS και RADIUS. Στο παράδειγμα χρησιμοποιείται η μέθοδος CORNERS που χρησιμοποιεί δύο σημεία για τον σχεδιασμό του τετραγώνου, το σημείο πάνω αριστερά (x1,y1) και το σημείο κάτω δεξιά (x2,y2).
    ellipseMode(CORNERS); // Προσδιορίζει τον τρόπο με τον οποίο θα σχεδιαστεί μία έλλειψη. Υπάρχουν οι μέθοδοι CORNER, CENTER, CENTERS και RADIUS. Στο παράδειγμα χρησιμοποιείται η μέθοδος CORNERS που χρησιμοποιεί δύο σημεία για τον σχεδιασμό της έλλειψης, το σημείο πάνω αριστερά (x1, y1) και το σημείο κάτω δεξιά (x2, y2).
    noLoop(); // Τερματίζει την επαναληπτική εκτέλεση των εντολών που βρίσκονται εντός της συνάρτησης draw().
}
void draw() {
    fill(255,0,0); // χρησιμοποιεί τιμές του μοντέλου RGB για το γέμισμα αντικειμένων με χρώμα.
    rect(10,10,150,150); // Σχεδιάζει στην οθόνη ένα τετράγωνο χρησιμοποιώντας τη μέθοδο CORNER
    fill(0,255,0);
    ellipse(160,10,300,150); // Σχεδιάζει στην οθόνη μία έλλειψη χρησιμοποιώντας τη μέθοδο CORNERS
    fill(0,0,255);
    triangle(310, 150, 450, 150, 380, 10); // Σχεδιάζει στην οθόνη ένα τρίγωνο χρησιμοποιώντας τρία σημεία (x1, y1), (x2, y2) και (x3, y3), το κάθε σημείο αντιστοιχεί και σε μία ακμή του τριγώνου.
    line(0, 160, 460, 160); // Σχεδιάζει στην οθόνη μία γραμμή χρησιμοποιώντας δύο σημεία (x1, y1), (x2, y2).
    fill(0,0,0);
    text("Τετράγωνο", 50, 180); // Τυπώνει απλό κείμενο στην οθόνη, στο σημείο (x,y).
    text("Κύκλος", 210, 180);
    text("Τρίγωνο", 350, 180);
    save("example_1_1.jpg");// Αποθηκεύει την τελική εικόνα στο σκληρό δίσκο. Δέχεται σαν όρισμα το όνομα και τον τύπο του αρχείου. Η Processing αποθηκεύει ψηφιακές εικόνες σε μορφή TIFF, TARGA, JPEG, και PNG. }
```



Σχήμα 1.5 Γραφική απεικόνιση παραδείγματος

1.6.3 Μηχανή Erembus (Erembus Engine)

Η Erembus Engine είναι μια μηχανή γραφικών η οποία δημιουργήθηκε στα πλαίσια πτυχιακής εργασίας του Μπεκιάρη Γεώργιου το 2008 στο Τμήμα Ηλεκτρονικών Υπολογιστικών Συστημάτων του ΤΕΙ ΠΕΙΡΑΙΑ.

Η μηχανή αυτή χρησιμοποιεί το Direct3D. Στα αρχικά σχέδια του φοιτητή ήταν να υποστηρίζει και OpenGL αλλά δεν ήταν εφικτό αυτό για το λόγο ότι έπρεπε να χρησιμοποιηθεί η καινούργια για τότε έκδοση της OpenGL (έκδοση 3.0) και όχι η προηγούμενη έκδοση διότι η προηγούμενη έκδοση δεν μπορεί να συγκριθεί με το Direct3D 10 λόγω της εντελώς διαφορετικής τους φιλοσοφίας και σχεδίασης. Έτσι η Erembus Engine υποστηρίζει μόνο Direct3D 10 αλλά είναι δομημένη έτσι ώστε να μπορεί να υποστηρίζει και άλλα 3D Graphics API με πολύ μικρές αλλαγές στον κώδικα.

1.6.3.1 Εργαλεία που χρησιμοποιήθηκαν για τη δημιουργία της μηχανής

1.6.3.1.1 Γλώσσα Προγραμματισμού

Η γλώσσα προγραμματισμού που χρησιμοποιήθηκε είναι η C/C++. Με τα πλεονεκτήματα του αντικειμενοστραφούς προγραμματισμού που έχει η C++ κρατήθηκε μια απλή οργανωμένη δομή στον κώδικα. Ο compiler που χρησιμοποιήθηκε είναι ο compiler που συνοδεύει το δημοφιλές IDE της Microsoft, την Microsoft Visual C++.NET 2005. Η συγγραφή του κώδικα έγινε με τέτοιο τρόπο έτσι ώστε με ελάχιστες αλλαγές να μπορεί να γίνει compile και με τον GCC(g++) compiler. Προτιμήθηκε ο Visual C++ compiler επειδή έχει έναν πολύ ισχυρό και ευέλικτο Debugger αλλά και έναν Memory Leak Detector.

1.6.3.1.2 Περιεχόμενο εφαρμογής

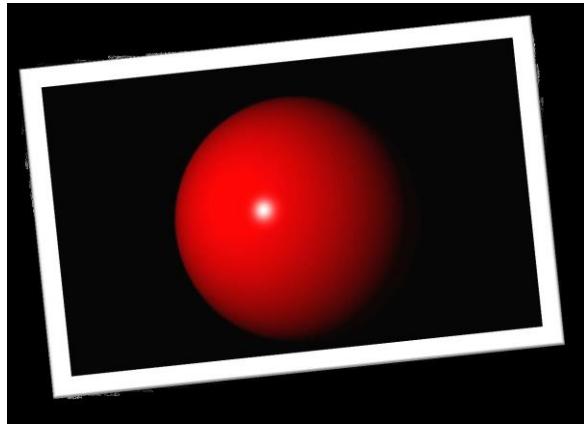
Τα μοντέλα που σχεδιάστηκαν για να φορτώνονται στη μηχανή γραφικών περιορίζονται στα βασικά γεωμετρικά σχήματα, χωρίς αυτό να σημαίνει ότι η μηχανή γραφικών δεν μπορεί να φορτώσει και πιο πολύπλοκα μοντέλα. Ο τύπος αρχείων που χρησιμοποιεί η μηχανή είναι «obj» λόγω της απλότητας του.

Η επέκταση του αρχείου OBJ είναι γνωστή ως Wavefront 3D Object αρχείου που αναπτύχθηκε από Wavefront Technologies. Είναι μια μορφή αρχείου που χρησιμοποιείται για ένα τρισδιάστατο αντικείμενο που περιέχει 3D συντεταγμένες (γραμμή πολύγωνο και τα σημεία), χάρτες υφής, καθώς και άλλες πληροφορίες αντικειμένου. Περιέχει ένα πρότυπο 3D μορφή εικόνας που μπορούν να εξαχθούν και να ανοίξει πολλά προγράμματα επεξεργασίας εικόνας 3D. Αρχεία αντικείμενο μπορεί να είναι σε μορφή ASCII (.Obj) ή δυαδική μορφή (.Mod), ωστόσο, δεν περιέχει ορισμούς των χρωμάτων για τα πρόσωπα. Λόγω του σχήματός τους, είναι αναγνώσιμο από τον άνθρωπο. Υποστηρίζει τόσο πολυγωνικά αντικείμενα και ελεύθερης μορφής αντικείμενα. Πολυγωνική γεωμετρία χρησιμοποιεί σημεία, γραμμές, και αντιμετωπίζει τον καθορισμό αντικειμένων, ενώ ελεύθερης μορφής γεωμετρία χρησιμοποιεί καμπύλες επιφάνειες. Τα αρχεία σε μορφή OBJ μπορεί να ανοίξει με την Autodesk Maya 2013, μπλέντερ, και MeshLab στα Microsoft Windows, Mac OS και Linux πλατφόρμες [58].

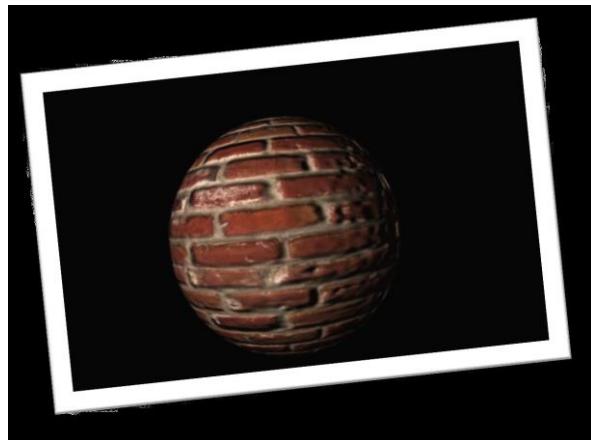
1.6.3.1.3 Απαιτήσεις συστήματος

Η μηχανή αυτή απαιτεί Windows Vista ή Windows Server 2008 ή νεότερο λειτουργικό σύστημα για να τρέξει. Αυτό οφείλεται ότι το Direct3D 10 δεν τρέχει στα παλαιότερα λειτουργικά συστήματα της Microsoft. Επίσης απαιτεί κάρτα γραφικών με υποστήριξη DirectX 10 [2].

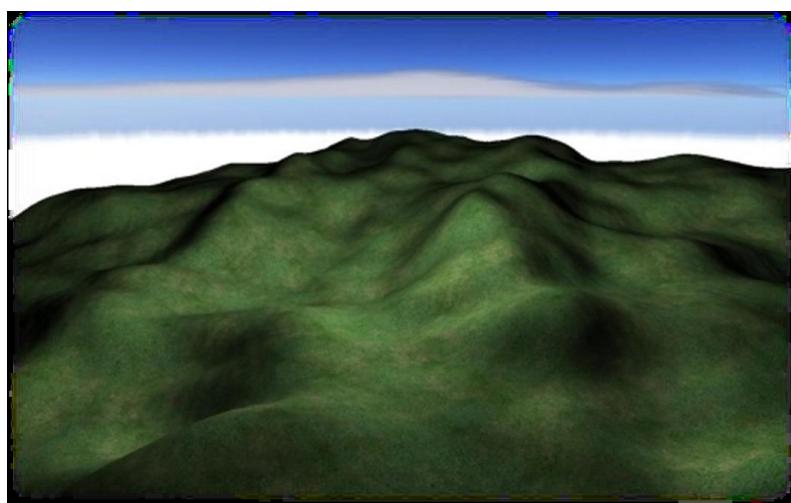
1.6.3.1.4 Εικόνες από τη μηχανή



Σχήμα 1.6 Απεικόνιση σφαίρας με φωτοσκίαση



Σχήμα 1.7 Απεικόνιση σφαίρας με υφή



Σχήμα 1.8 Τρισδιάστατη απεικόνιση ανάγλυφης επιφάνειας

1.6.4 Virtual Supermarket

Αυτή η εφαρμογή αποτελείται από ένα 3D σουύπερ μάρκετ με μια μεγάλη ποικιλία προϊόντων, οι εικονικοί ηθοποιοί που εκπροσωπούν διαφορετικούς εργαζομένους και διαφορετικές διαθέσιμες λειτουργίες. Η βασική αποστολή σε αυτό το εργαλείο είναι να κάνετε τα ψώνια. Ο χρήστης πρέπει να μετακινηθεί μέσα στο σουύπερ μάρκετ ψάχνοντας για τα προϊόντα που αναφέρονται σε μια λίστα για ψώνια. Η λίστα με τα ψώνια επιλέγεται στην αρχή και περιέχει τα αντικείμενα με τα οποία θα δουλέψει σε κάθε συνεδρία. Ο δάσκαλος χρειάζεται να οργανώσει κάθε συνεδρία να αποφασίσει ποια λίστα πρόκειται να χρησιμοποιήσουν ανάλογα με ποια αντικείμενα θέλει να εργαστούν. Πλοήγηση μέσα στο περιβάλλον γίνεται με τη χρήση ενός joystick ή τα βελάκια του πληκτρολογίου. Η χρήση ενός joystick διευκολύνει την αλληλεπίδραση για όλους τους χρήστες, διότι είναι πιο διαισθητική και εύκολη στη χρήση από το πληκτρολόγιο. Η τοποθεσία μέσα στο σουύπερ μάρκετ αντιπροσωπεύεται από ένα εικονικό τρόλεϊ που κινείται με το χρήστη. Επίσης, το εικονικό καροτσάκι έχει τη λειτουργικότητα ενός πραγματικού τρόλεϊ σε ένα σουύπερ μάρκετ, δηλαδή, λειτουργεί

ως μια αποθήκη για τα προϊόντα που είναι απαραίτητο να αγοράσετε για να ολοκληρώσετε τη λίστα.

Για την επιλογή των αντικειμένων ή διαδραστικών στοιχείων στην εικονική ρύθμιση, είναι δυνατό να χρησιμοποιήσετε το ποντίκι ή μια οθόνη αφής. Και οι δύο διεπαφές είναι δυνατόν, ανάλογα με τη διαθεσιμότητά τους στο σχολείο ή εκεί όπου χρησιμοποιείται το λογισμικό, αλλά η οθόνη αφής που είναι πιο εύκολη στη χρήση για την πραγματοποίηση επιλογών στο εσωτερικό εικονικό κόσμου, γιατί το άτομο χρησιμοποιεί μόνο το δάχτυλό του / της για να αγγίξει το στοιχείο που πρέπει να επιλεγεί. Εκτός από τα παραδοσιακά καθήκοντα μέσα σε ένα σούπερ μάρκετ, προσφέρονται διάφορες λειτουργίες για να εργαστούν σε συγκεκριμένες δεξιότητες άτομα με μαθησιακές δυσκολίες, αξιοποιώντας τις δυνατότητες των εφαρμογών εικονικής πραγματικότητας για την εισαγωγή εκπαιδευτικών περιεχομένων. Αυτές οι λειτουργίες αντιπροσωπεύονται από εικονογράμματα σε μια γραμμή εργαλείων και είναι δυνατόν να αλλάξουν μεταξύ τους μόνο επιλέγοντας το κατάλληλο κουμπί που αντιπροσωπεύει την λειτουργία που θέλει να ενεργοποιήσει.



Σχήμα 1.9 Επιλογή ενός προϊόντος στον εικονικό καρότσι με τη λίστα με τα ψώνια στο Virtual σούπερ μάρκετ.



Σχήμα 1.10 Κουμπιά με όλες τις λειτουργίες.

1.6.4.1 Λειτουργίες (κουμπιά) του εικονικού σουπερμάρκετ

Λειτουργική χρήση και το παιχνίδι: Μαθαίνει να παίζει με τα αντικείμενα και τις μινιατούρες. Υπάρχουν προ-ηχογραφημένα βίντεο ενσωματωμένα στο εικονικό κόσμο. Αυτές οι λειτουργίες μπορούν να χρησιμοποιηθούν για να διδάξει σημαντικές έννοιες για την καθημερινή ζωή, που σχετίζονται με τις εργασίες για γνωστικές δεξιότητες και κατανόηση του περιβάλλοντος.

Φανταστικό παιχνίδι: Με τη λειτουργία αυτή είναι δυνατόν να διδάξει για το φανταστικό παιχνίδι με μινιατούρες από αντικείμενα. Και πάλι, το περιεχόμενο παρουσιάζεται από βίντεο ολοκληρωμένα στο σούπερ μάρκετ και μπορεί να συνδέεται με την εργασία πάνω στις γνωστικές δεξιότητες.

Φανταστικές μεταμορφώσεις και μαγεία: Εδώ θέλει να δείξει τα πλεονεκτήματα από τη χρήση της Εικονικής Πραγματικότητας για την εμφάνιση πληροφοριών που είναι δύσκολο να εξηγηθούν και να φανούν στον πραγματικό κόσμο. Όπως ο χρήστης κάνει μια φανταστική μεταμόρφωση σε ένα αντικείμενο που έχει διαλέξει σε ένα τελείως διαφορετικό αντικείμενο και με άλλες λειτουργίες. Ο μετασχηματισμός αυτός γίνεται με ένα τρισδιάστατο animation ενσωματωμένο στο εικονικό κόσμο. Αυτό επιχειρεί να προωθήσει την ανάπτυξη ορισμένων ικανοτήτων φαντασίας, θεωρώντας αυτές ως μέρος των γνωστικών δεξιοτήτων.

Imaginary use: Υπάρχει βίντεο με πληροφορίες για τη χρήση κάθε αντικείμενου, προσπαθώντας να παρέχει ένα άλλο εργαλείο για να δουλέψουν πάνω στις γνωστικές δεξιότητες.

Επίσης ο χρήστης μπορεί να ακούσει τους εικονικούς υπαλλήλους να μιλάνε και να δώσει την παραγγελία του, περιμένοντας να την πάρει εξασκώντας έτσι πάλι τις κοινωνικές του δεξιότητες. Στο τέλος ολοκληρώνοντας την λίστα πηγαίνει κανονικά στο ταμείο και γίνεται κανονικά η όλη συναλλαγή.

1.6.4.2 Ανάπτυξη εφαρμογής

Τα χαρακτηριστικά της εφαρμογής που παρουσιάστηκε πριν είναι ότι χρησιμοποιήθηκε η βιβλιοθήκη γραφικών OpenGL ενσωματώνοντας και την αντικειμενοστραφή γλώσσα C++. Λόγω της δυναμικής αυτού του είδους των βιβλιοθηκών για τη δημιουργία προσαρμοσμένων εργαλείων και τις δυνατότητες για την επίτευξη καλύτερης ποιότητας και αποτελεσμάτων από οποιαδήποτε άλλη μεθοδολογία. Αυτή η βιβλιοθήκη μας παρέχει την καλύτερη διαχείριση σκηνής και τον έλεγχο γραφικών που απαιτούνται για να δομηθεί και να αλληλεπιδρά η εφαρμογή σε πραγματικό χρόνο. Η προσαρμοστικότητα του σε όλα τα στοιχεία που εμπλέκονται στο σύστημα επιτρέπει την επαναχρησιμοποίηση για άλλα νέα περιβάλλοντα και την ευελιξία να ενσωματώσουν όλες τις επιθυμητές λειτουργίες. Ο σχεδιασμός και η δημιουργία μοντέλων, όπως και τα animations που ενσωματώθηκαν αναπτυχθήκαν σε γνωστά σχεδιαστικά εργαλεία 3D Studio Max και Photoshop. Όλα αυτά τα συστατικά, έχουν ενσωματωθεί σε ένα καλά καθορισμένο και δομημένο σύστημα.

Τα αποτελέσματα σε τέτοιες εφαρμογές γραφικών υπολογιστών χρησιμοποιούνται για ανθρώπους με ειδικές ανάγκες, που προσπαθούν να τους βοηθήσουν στη ανάπτυξη της γνώσης τους και την ποιότητα ζωής τους [8].

1.6.5 SEDEA

Πρόκειται για μια εφαρμογή πληροφορικής για τη διαδικασία της παρέμβασης, για την ανάπτυξη της ακοής και την ομιλία των παιδιών. Αποτελεί ένα καινοτόμο διδακτικό εργαλείο που διοργανώνει, διευκολύνει και εμπλουτίζει το έργο της ακουστικής αποκατάστασης και της ομιλίας στα ισπανόφωνα παιδιά και ενήλικες.

Ένα πρόγραμμα με διαδοχικές δραστηριότητες, που πρέπει να χρησιμοποιείται ανάλογα με την ακουστική ικανότητα που έχει το παιδί. Επιτρέπει την εργασία σε ένα οργανωμένο και προοδευτικό τρόπο από την ανίχνευση καθημερινών ήχων μέχρι πολύπλοκων καταστάσεων ομιλίας όπως ο διάλογος και συζήτηση.



Σχήμα 1.11: Αρχική εικόνα προγράμματος

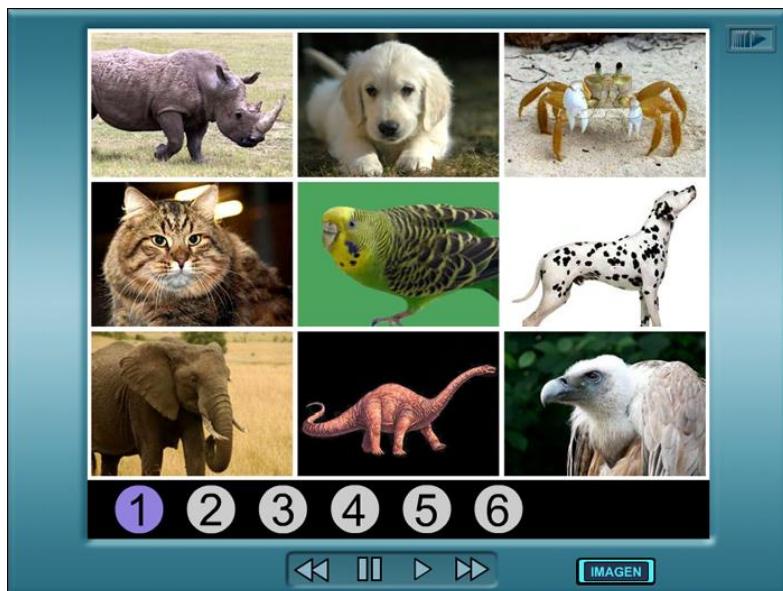


Σχήμα 1.12: Λειτουργία από το πρόγραμμα

Το πρόγραμμα χωρίζεται στα εξής στάδια:

- **ANIXNEYΣΗ:** Το παιδί θα μάθει να δίνει προσοχή, να απαντά με την παρουσία του ήχου και να περιμένει όταν δεν υπάρχει κανένας ήχος.
- **ΔΙΑΚΡΙΣΕΙΣ:** Το παιδί θα ανακαλύψει σταδιακά τις διαφορές μεταξύ των ήχων και θα είναι σε θέση να αντιλαμβάνεται τις ομοιότητες μεταξύ τους.
- **ΤΑΥΤΟΤΗΤΑ:** Το παιδί θα είναι σε θέση να επιλέξει έναν ήχο, λέξη ή φράση σε ένα περιορισμένο εύρος και να το αναπαράγει και να το εντοπίσει.
- **ΑΝΑΓΝΩΡΙΣΗ:** Το παιδί θα επαναλάβει τις λέξεις και φράσεις που παρουσιάζονται σε κλειστά και ανοιχτά περιβάλλοντα, κατανοώντας τη σημασία τους.

- **ΚΑΤΑΝΟΗΣΗ:** Το παιδί θα είναι σε θέση να ακολουθήσει πολύ διαφορετικές επικοινωνιακές καταστάσεις, διάλογους, συζητήσεις, νέα, κείμενα, οδηγίες, καθώς και την επίλυση πολλών διαδραστικών δραστηριοτήτων που απαιτούν να ακούει.



Σχήμα 1.13: Λειτουργία από το πρόγραμμα

Το πρόγραμμα απευθύνεται κυρίως:

- Σε άτομα με βαρηκοϊα, σε ειδικά κοχλιακά εμφυτεύματα που μπορεί να υλοποιήσει το πρόγραμμα με ένα ρυθμό που ικανοποιεί τις δικές τους ανάγκες.
- Σε λογοθεραπευτές και εκπαιδευτικούς που φροντίζουν παιδιά με τα παραπάνω χαρακτηριστικά και επιθυμούν πληροφορίες και θέλουν να έχουν μια καινοτόμο υλική βάση από την οποία να αναπτύξουν το έργο τους και την εκπαιδευτική αποκατάσταση.
- Σε γονείς που επιθυμούν να συνεργαστούν στη διαδικασία της ακουστικής εκπαίδευσης των παιδιών τους.
- Τα παιδιά με προσοχή, τις διακρίσεις και ακουστικές δυσκολίες μνήμης, αντά έχουν άμεσο αντίκτυπο στην προφορική γλώσσα [9].

1.6.6.Stellarium

Το Stellarium είναι ένα δωρεάν πρόγραμμα ανοιχτού λογισμικού. Εμφανίζει έναν ρεαλιστικό ουρανό σε 3D όπως ακριβώς θα τον βλέπατε με γυμνά μάτια, κιάλια ή τηλεσκόπιο. Χρησιμοποιείται με προβολείς πλανηταρίων. Είναι ένα εκπαιδευτικό πρόγραμμα που μαθαίνει στο χρήστη για τον γαλαξία, περιέχει ένα κατάλογο με πάνω από 600,000 άστρα, απεικόνιση των αστερισμών και των σχεδίων τους, εικόνες νεφελωμάτων, ρεαλιστικό γαλαξία, πολύ ρεαλιστική ατμόσφαιρα, ανατολή και δύση του Ήλιου, τους πλανήτες και τους δορυφόρους τους.



Σχήμα 1.14: Απεικόνιση από το πρόγραμμα

1.6.6.1 Χαρακτηριστικά του προγράμματος

Το μενού χειρισμού διαθέτει πολύ δυνατή εστίαση (zoom), ελεγχος του χρόνου, πολύγλωσσο μενού χειρισμού, ευρυγώνια προβολή για θόλους πλανηταρίων, σφαιρική κατοπτρική προβολή για τον δικό σας θόλο, γραφικό μενού χειρισμού και εκτεταμένος έλεγχος μέσω πληκτρολογίου, έλεγχος τηλεσκοπίου απεικόνιση, ισημερινό και αζιμουθιακό πλέγμα, ισημερινό και αζιμουθιακό πλέγμα, σπινθήρισμα άστρων, εξομοίωση εκλείψεων, εξομοίωση υπερκαινοφανών, μεταβαλλόμενα τοπία με σφαιρική πανοραμική προβολή.

Η προσαρμοστικότητα του προγράμματος

- Σύστημα προσθέτων που προσθέτει τεχνικούς δορυφόρους, εξομοίωση προσοφθαλμιών, ρυθμίσεις τηλεσκοπίων και άλλα
- Ικανότητα να προστίθενται νέα αντικείμενα του ηλιακού συστήματος από πηγές στο διαδίκτυο
- Πρόσθεση αντικειμένων του χρήστη όπως άστρα, τοπία, εικόνες αστερισμών [10].

1.6.6.2 Ανάπτυξη εφαρμογής και οι απαιτήσεις του συστήματος

Η ανάπτυξη της εφαρμογής έγινε με τις γλώσσες προγραμματισμού C++ και C. Οι απεικονίσεις του προγράμματος έγιναν με τη χρήση της βιβλιοθήκης OpenGL.

Απαιτήσεις του συστήματος

Οι ελάχιστες απαιτήσεις που απαιτούνται για την χρήση του προγράμματος είναι

- Linux/Unix; Windows 2000/XP/Vista/7/8; 64-bit Mac OS X 10.6.8 ή μεγαλύτερο
- 3D κάρτα γραφικών με υποστήριξη OpenGL 1.2
- 256 MB RAM
- 120 MB στο δίσκο

Οι προτεινόμενες απαιτήσεις που απαιτούνται για την χρήση του προγράμματος είναι

- Linux/Unix; Windows XP/Vista/7/8; 64-bit OS X 10.7.0 ή μεγαλύτερο
- 3D κάρτα γραφικών με υποστήριξη OpenGL 2.1 ή μεγαλύτερο
- 1 GB RAM ή περισσότερο
1.5 GB στο δίσκο

1.7 Συγκρίσεις εφαρμογών

1.7.1 SeeGL vs Learning OpenGL

Η εφαρμογή SeeGL παρέχει δυνατότητες παρόμοιες με την εφαρμογή που υλοποιείται στην πτυχιακή εργασία. Αποτελεί και αυτό ένα εκπαιδευτικό πρόγραμμα για την εκμάθηση της OpenGL.

Τόσο στην εφαρμογή της πτυχιακής εργασίας, όσο και στην εφαρμογή της SeeGL ο χρήστης θα μπορεί να γράψει κώδικα σε OpenGL, να τον εκτελεί και το αποτέλεσμα της εκτέλεσης αυτής να απεικονίζεται σε νέο παράθυρο. Επίσης οι δυο εφαρμογές έχουν την ίδια λειτουργία όσον αφορά τα λάθη που μπορεί να προκύψουν στη σύνταξη του κώδικα. Βασική όμως διαφορά είναι στον τρόπο εντοπισμού αυτών των λαθών. Στην SeeGL υπάρχει ένα αρχείο εργασιών όπου καταγράφονται τα βήματα εκτέλεσης του κώδικα και έτσι ο χρήστης θα μπορεί να αντιληφθεί εάν το πρόβλημα προκλήθηκε από λάθος παραμέτρους ή από λάθος εντολή. Στην εφαρμογή της πτυχιακής εργασίας, σε ένα ειδικό πλαίσιο κειμένου εμφανίζονται τα λάθη, καθώς και ο αριθμός γραμμής κάθε λάθους.

Συνοπτικά, η συγκεκριμένη εφαρμογή αποτελεί ένα πολύ καλό εκπαιδευτικό εγχειρίδιο για τη δημιουργία γραφικών με την OpenGL. Στη σημερινή εποχή υπάρχουν ελάχιστες εκπαιδευτικές εφαρμογές για τη σχεδίαση και ανάπτυξη γραφικών με την OpenGL.

1.7.2 Processing vs Learning OpenGL

Το Processing είναι και γλώσσα προγραμματισμού και προγραμματιστικό περιβάλλον. Δημιουργεί και αυτό γραφικά μέσω προγραμματισμού όπως η παρούσα πτυχιακή, με την διαφορά ότι αυτή η εφαρμογή χρησιμοποιεί τη γλώσσα προγραμματισμού Processing ενώ στην πτυχιακή χρησιμοποιείται η OpenGL. Η πλατφόρμα Processing έχει ενσωματωμένη την OpenGL για επιτάχυνση 3D.

Η πλατφόρμα Processing περιέχει και αυτή **μενού** απλώς είναι πιο εκτενές και με περισσότερες λειτουργίες από της πτυχιακής. Η **εφαρμογή της πτυχιακής** περιέχει 4 λειτουργίες στο μενού, αρχείο,επεξεργασία,παραδείγματα και βοήθεια. Το αρχείο περιέχει το υπομενού αποθήκευση, εκτέλεση και έξοδο. Η λειτουργία των παραδειγμάτων περιέχει έτοιμα παραδείγματα της OpenGL και ένα υποφάκελο custom που αποθηκεύονται τα νέα παραδείγματα. Και η βοήθεια περιέχει εγχειρίδιο χρήσης της εφαρμογής, της OpenGL και των παραδειγμάτων. Και η Επεξεργασία που περιέχει τις λειτουργίες αποκοπή, αναίρεση, αντιγραφή,επικόλληση. Η **Processing** περιέχει 5 λειτουργίες στο μενού της, file, edit, sketch, tools και help. Το μενού file το οποίο έχει τις λειτουργίες διαχείρισης των αρχείων όπως αποθήκευση, άνοιγμα, νέα καρτέλα, παραδείγματα, κλείσιμο, εκτύπωση και άλλα. Το μενού edit το οποίο περιέχει τις εντολές λειτουργιών όπως undo, copy paste, select all και άλλα. Το μενού sketch περιέχει τις λειτουργίες εκτέλεσης και παύσης εκτέλεσης του κώδικα και άλλες λειτουργίες. Το μενού tools το οποίο έχει τις λειτουργίες δημιουργία

φόντου, επιλογή χρώματος, δημιουργία βίντεο από συνεχόμενα γραφήματα. Το μενού help περιέχει βοήθεια για την εφαρμογή.

Επίσης η πλατφόρμα Processing περιέχει και **εργαλειοθήκη** όπως η εφαρμογή της πτυχιακής. Η Processing περιέχει 6 κουμπιά, οι λειτουργίες τους είναι εκτέλεση του προγράμματος, τερματισμός εκτέλεσης προγράμματος, δημιουργία νέου έργου, επιλογή και εμφάνιση παραδειγμάτων, αποθήκευση έργου και δημιουργία αυτόνομης εφαρμογής (export). Η εφαρμογή της πτυχιακής περιέχει 7 κουμπιά, αποθήκευση, αποκοπή, αντιγραφή, επικόλληση, εκτέλεση, εισαγωγή παραδειγμάτος και έξοδος.

Επιπλέον και δυο εφαρμογές περιέχουν μια φόρμα για να γράφει ο χρήστης τον κώδικα του. Μια φόρμα όπου εμφανίζονται τα λάθη του κώδικα μετά την εκτέλεση του και ένα νέο παράθυρο όπου εμφανίζεται ή απεικόνιση του γραφήματος.

Η εφαρμογή Processing είναι πολυπλατφορική μπορεί να τρέξει σε πολλά διαφορετικά λειτουργικά συστήματα όπως Linux, Mac, Windows ενώ η πλατφόρμα της πτυχιακής λειτουργεί μόνο σε περιβάλλον Windows.

1.7.3 Erembus Engine vs Learning OpenGL

Η γλώσσα προγραμματισμού που χρησιμοποιήθηκε για την δημιουργία της Erembus Engine είναι η C/C++ ενώ η ανάπτυξη της εφαρμογής της πτυχιακής έγινε με την γλώσσα προγραμματισμού C#. Η μηχανή αυτή χρησιμοποιεί για τη δημιουργία των γραφικών το Direct3D ενώ η πλατφόρμα της πτυχιακής χρησιμοποιεί την OpenGL. Στα αρχικά σχέδια της Erembus Engine ήταν να υποστηρίζει και OpenGL αλλά δεν ήταν εφικτό τελικά.

Τα μοντέλα που υπάρχουν στην μηχανή Erembus είναι τα βασικά γεωμετρικά σχήματα αλλά μπορούν προστεθούν και πιο πολύπλοκα μοντέλα. Στην εφαρμογή εκμάθησης OpenGL τα παραδείγματα είναι από τα πιο απλά στα πιο πολύπλοκα και από όλο το φάσμα των ενοτήτων της OpenGL.

Η μηχανή Erembus και η εφαρμογή εκμάθησης της πτυχιακής λειτουργούν και οι δυο μόνο σε λειτουργικό σύστημα Windows.

Στη μηχανή Erembus ο compiler που χρησιμοποιήθηκε είναι ο compiler της Microsoft, ο Microsoft Visual C++.NET 2005. Με ελάχιστες αλλαγές μπορεί να γίνει compile και με τον GCC(g++) compiler. Στην εφαρμογή εκμάθησης OpenGL οι compiler που χρησιμοποιούνται είναι C/C++ compiler και ο msbuilt compiler.

1.7.4 Virtual Supermarket Vs Learning OpenGL

Αυτή η εφαρμογή είναι μια εκπαιδευτική εφαρμογή που αποτελείται από ένα 3D σούπερ μάρκετ και έχει σκοπό να βοηθήσει άτομα με μαθησιακές δυσκολίες. Ο σκοπός είναι να αγοραστούν όλα τα προϊόντα από μια λίστα, να αλληλεπιδράσουν με τους υπαλλήλους και να δουν οι χρήστες πως μπορούν να κινηθούν σε ένα τέτοιο χώρο, μέσα από όλες τις εργασίες που κάνει ο χειριστής βελτιώνει κάποιες ικανότητες και δεξιότητες. Η εφαρμογή της πτυχιακής έχει ως σκοπό της να βοηθήσει τον χρήστη να μάθει να χειρίζεται και να δημιουργεί παραδείγματα σε OpenGL. Επομένως και οι δυο αποτελούν αποκλειστικά εκπαιδευτικές εφαρμογές που διδάσκουν διαφορετικό υλικό και απευθύνονται σε διαφορετικό κοινό, αλλά κοινό τους σημείο είναι ότι περιέχουν γραφικά υπολογιστών.

Τα χαρακτηριστικά της εφαρμογής του σουπερμάρκετ είναι ότι χρησιμοποιήθηκε η βιβλιοθήκη γραφικών OpenGL ενσωματώνοντας και την αντικειμενοστραφή γλώσσα C++. Το ίδιο κοινό έχει και η εφαρμογή εκμάθησης της πτυχιακής, έχει

χρησιμοποιήσει τη βιβλιοθήκη γραφικών OpenGL και τη γλώσσα προγραμματισμού C++ για τη δημιουργία των έτοιμων παραδειγμάτων και τη συγγραφή του κώδικα από το χρήστη και τη γλώσσα προγραμματισμού C# για την ανάπτυξη της εφαρμογής. Ακόμη στην εφαρμογή του εικονικού συνπεριμάρκετ ο σχεδιασμός και η δημιουργία μοντέλων, όπως και τα animations που ενσωματώθηκαν αναπτύχθηκαν σε γνωστά σχεδιαστικά εργαλεία 3D Studio Max και Photoshop.

1.7.5 Sedea vs Learning OpenGL

Η εφαρμογή Sedea είναι μια εκπαιδευτική εφαρμογή όπως της παρούσας πτυχιακής εργασίας με διαφορετικό εκπαιδευτικό υλικό. Βοηθάει παιδιά με πρόβλημα στην ακοή και την ομιλία να βελτιώσουν τις ικανότητες τους. Η εφαρμογή της πτυχιακής έχει ως σκοπό της να βοηθήσει το χρήστη να εξουκειωθεί με τη χρήση ανάπτυξης κώδικα σε OpenGL. Και οι δύο εφαρμογές χρησιμοποιούν γραφικά, η μια είναι αφιερωμένη στην εκμάθηση γραφικών, η άλλη χρησιμοποιεί τα γραφικά για την δημιουργία της εφαρμογής, όπως για παράδειγμα η κατασκευή και η εμφάνιση των εικόνων των μουσικών οργάνων, των ζώων και άλλα.

1.7.6 Stellarium vs Learning OpenGL

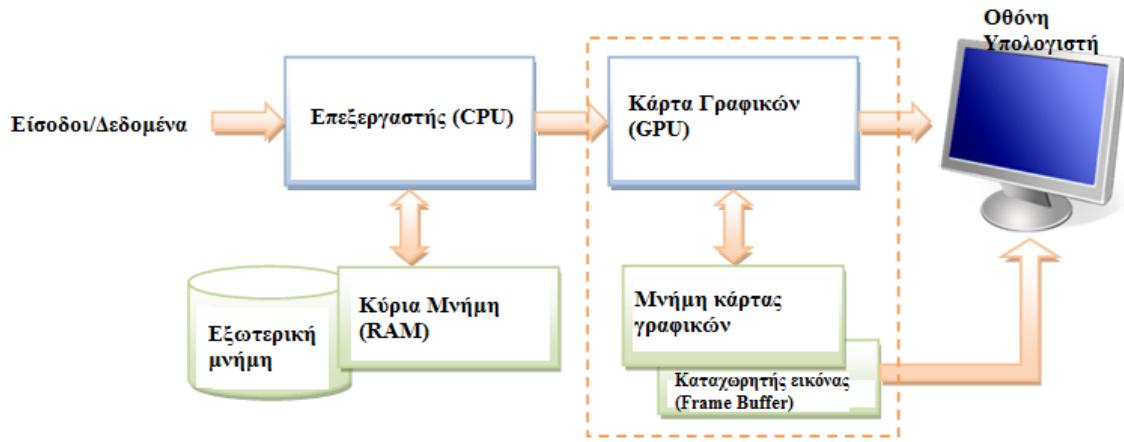
Η εφαρμογή Stellarium εμφανίζει σε τρισδιάστατη μορφή τον ουρανό. Είναι ένα πρόγραμμα που σκοπό του έχει να μάθει στο χρήστη για τον γαλαξία. Όπως και η εφαρμογή της πτυχιακής έχει σκοπό να μάθει στο χρήστη να γράφει κώδικα σε OpenGL και να βλέπει την δημιουργία γραφικών μέσω του προγράμματος.

Κάτι παρόμοιο που έχουν οι δύο εφαρμογές στην ανάπτυξη της εφαρμογής είναι ότι και οι δύο χρησιμοποίησαν τη γλώσσα προγραμματισμού C++, η εφαρμογή Stellarium χρησιμοποίησε και την γλώσσα C. Ακόμη και στις δύο εφαρμογές οι απεικονίσεις των προγραμμάτων έγιναν με τη χρήση της βιβλιοθήκης OpenGL, στη μια για την απεικόνιση του γαλαξία στην άλλη για την απεικόνιση των γραφημάτων από την εκτέλεση του κώδικα. Επιπλέον η εφαρμογή Stellarium απαιτεί για να λειτουργήσει 3D κάρτα γραφικών με υποστήριξη OpenGL 1.2.

Το πρόγραμμα Stellarium είναι πολυπλατφορμικό δηλαδή μπορεί να χρησιμοποιηθεί σε πολλά λειτουργικά συστήματα όπως Unix, Linux, Mac, Windows ενώ η εφαρμογή της πτυχιακής μπορεί να δουλέψει μόνο σε λειτουργικό Windows.

1.8 Υλικό διεξαγωγής για την ανάπτυξη των γραφικών

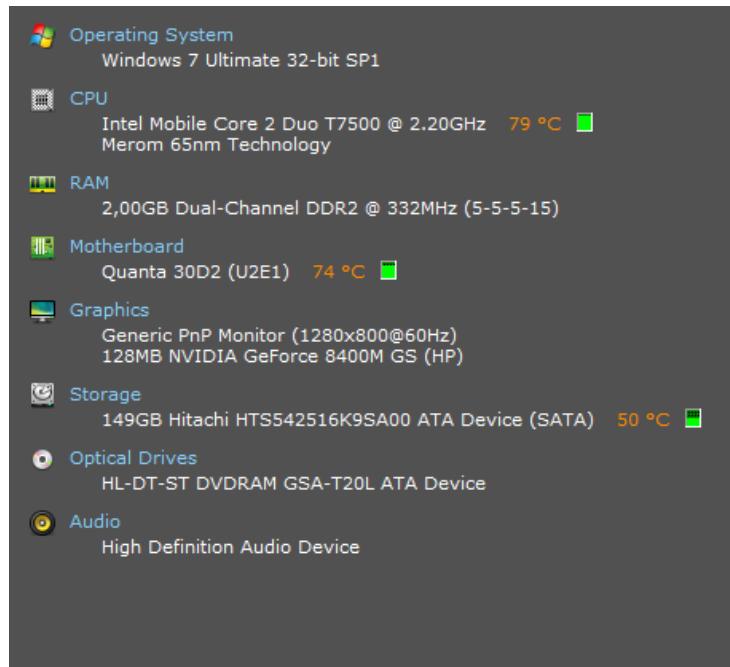
Το υλικό για την υλοποίηση γραφικών είναι το υλικό του υπολογιστή που παράγει γραφικά υπολογιστών και επιτρέπει την εμφάνιση τους σε μια οθόνη, χρησιμοποιώντας μια κάρτα γραφικών σε συνδυασμό με ένα πρόγραμμα οδήγησης συσκευής [59].



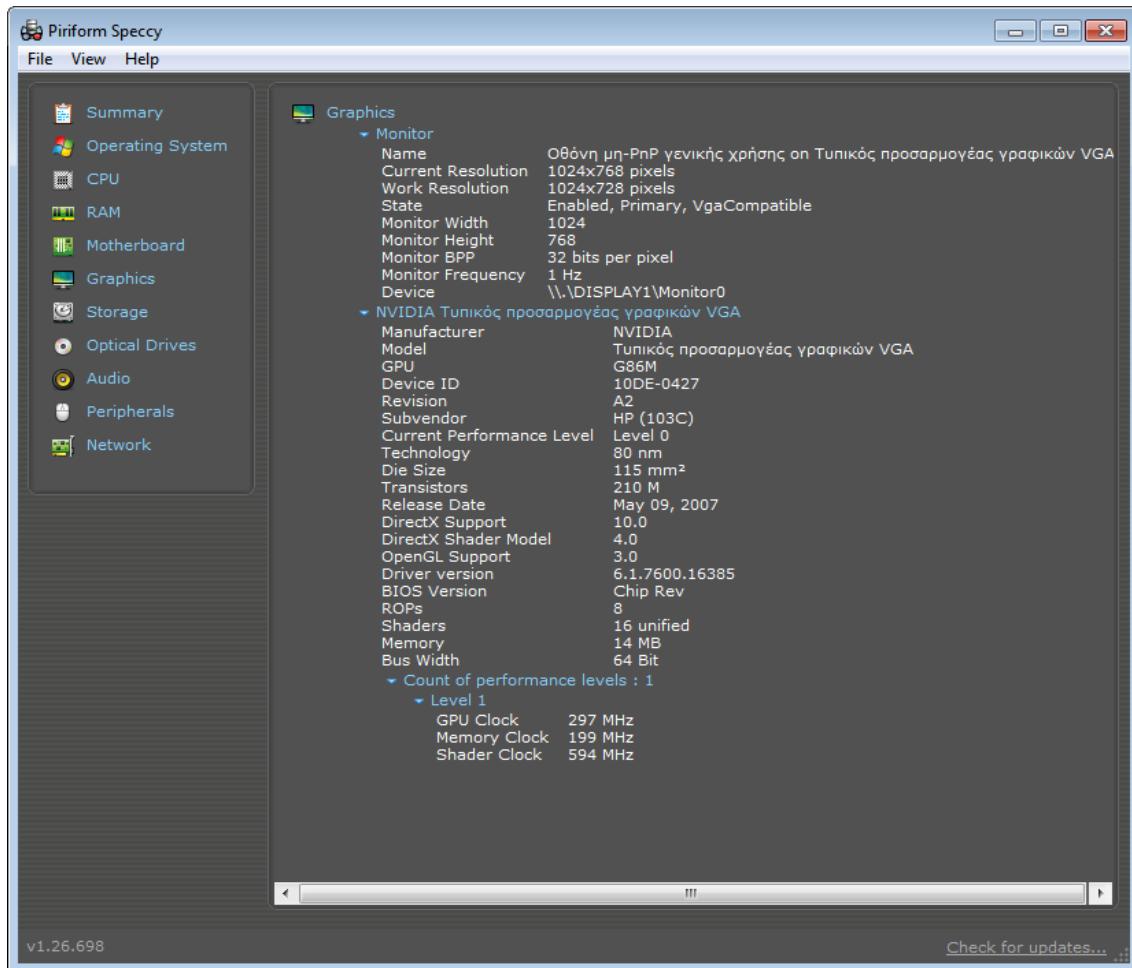
Σχήμα 1.15 Απαραίτητο υλικό για την ανάπτυξη γραφικών

1.8.1 Το υλικό που χρησιμοποιήθηκε στην παρούσα πτυχιακή εργασία

Οι υπολογιστές που χρησιμοποιήθηκαν για την ολοκλήρωση όλης της πτυχιακής είναι 2. Ο πρώτος έχει τα εξής χαρακτηριστικά:

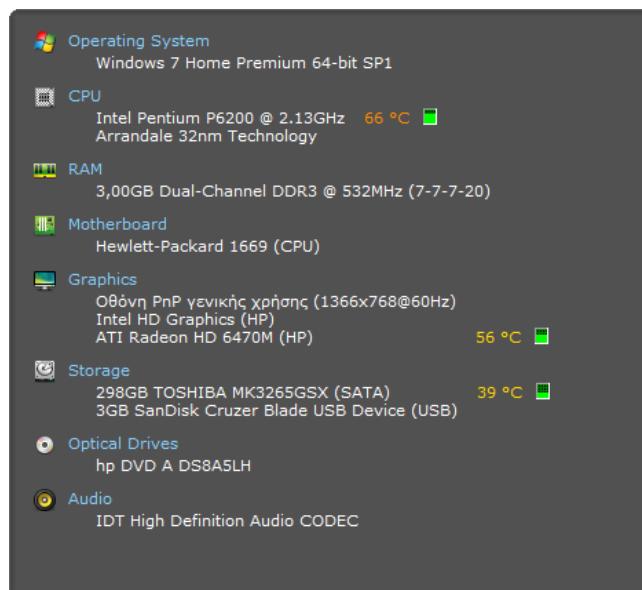


Σχήμα 1.16: Γενικά χαρακτηριστικά του υπολογιστή 1



Σχήμα 1.17: Χαρακτηριστικά κάρτας γραφικών υπολογιστή 1

Ο δεύτερος έχει τα εξής χαρακτηριστικά:



Σχήμα 1.19: Γενικά χαρακτηριστικά του υπολογιστή 2

```

Graphics
  ▾ Monitor
    Name          Οθόνη PnP γενικής χρήσης στο Radeon HD 6470M
    Current Resolution 1366x768 pixels
    Work Resolution 1366x728 pixels
    State         Enabled, Primary
    Monitor Width 1366
    Monitor Height 768
    Monitor BPP   32 bits per pixel
    Monitor Frequency 60 Hz
    Device        \\.\DISPLAY3\Monitor0
  ▾ Intel HD Graphics
    Manufacturer   Intel
    Model          HD Graphics
    Device ID     8086-0046
    Revision       3
    Subvendor     HP (103C)
    Current Performance Level Level 0
    Voltage        1,100 V
    Driver version 8.801.3.1000
      ▾ Count of performance levels : 1
        Level 1
  ▾ ATI Radeon (TM) HD 6470M
    Manufacturer   ATI
    Model          Radeon HD 6470M
    GPU            Seymour
    Device ID     1002-6760
    Subvendor     HP (103C)
    Current Performance Level Level 0
    Voltage        1,100 V
    Die Size       67 mm2
    Release Date  2010
    DirectX Support 11.0
    DirectX Shader Model 5.0
    OpenGL Support 4.2
    GPU Clock     700,0 MHz
    Temperature   54 °C █
    Core Voltage  1.100 V
    Bios Core Clock 123,36
    Bios Mem Clock 123,36
    Driver version 8.801.3.1000
    BIOS Version  BR039054.003
    ROPs          32
    Pixel Fillrate 22,4 GPixel/s
      ▾ Count of performance levels : 3
        ▾ Level 1
          GPU Clock  100 MHz
          Memory Clock 150 MHz
        ▾ Level 2
          GPU Clock  400 MHz
          Memory Clock 900 MHz
        ▾ Level 3
          GPU Clock  700 MHz
          Memory Clock 900 MHz

```

Σχήμα 1.20: Χαρακτηριστικά κάρτας γραφικών υπολογιστή 2

1.9 Βιβλιοθήκες δημιουργίας γραφικών

1.9.1 Εισαγωγή

1.9.1.1 Τι είναι μια βιβλιοθήκη γραφικών

Βιβλιοθήκη γραφικών είναι μια βιβλιοθήκη προγράμματος η οποία έχει σχεδιαστεί για να βοηθήσει στην απόδοση γραφικών του υπολογιστή στην οθόνη. Συνήθως περιλαμβάνει την παροχή βελτιστοποιημένων εκδόσεων και λειτουργιών που χειρίζονται κοινές εργασίες απόδοσης. Κάτι τέτοιο μπορεί να δημιουργηθεί στο λογισμικό και να «τρέχει» στον επεξεργαστή (CPU), αποτελώντας συνηθισμένο φαινόμενο στα ενσωματωμένα συστήματα. Στους κοινούς υπολογιστές γίνεται με επιτάχυνση του υλικού από την GPU. Με τη χρησιμοποίηση αυτών των λειτουργιών, ένα πρόγραμμα μπορεί να συνθέσει και να εμφανίσει μια εικόνα σε μια οθόνη. Αυτό απαλλάσσει τον προγραμματιστή από τη δημιουργία και τη βελτιστοποίηση αυτών

των λειτουργιών, και του επιτρέπει να επικεντρωθεί στην δημιουργία του προγράμματος γραφικών [12].

1.9.1.2 Τι είναι τα Graphics API

Τα Graphics API (Application Programming Interface – Διεπαφή Προγραμματισμού Εφαρμογών) είναι μια συλλογή από έτοιμα υποπρογράμματα που χρησιμοποιούνται για την ανάπτυξη λογισμικού από τους προγραμματιστές. Αναπτύσσει εφαρμογές για τη δημιουργία 2D και 3D γραφικών. Το πλεονέκτημά τους είναι ότι έχουν την υποστήριξη από τους κατασκευαστές των καρτών γραφικών. Αυτό έχει ως αποτέλεσμα να έχουν υποστήριξη σε επίπεδο hardware και τα παιχνίδια και οι εφαρμογές που τα χρησιμοποιούν να έχουν καλές επιδόσεις και ποιότητα. Ουσιαστικά, πρόκειται για διεπαφές που αναλαμβάνουν την επικοινωνία μεταξύ υλικού και λογισμικού. Παρακάτω εξετάζονται οι διεπαφές αυτές [2].

1.9.2 Διάφορες βιβλιοθήκες δημιουργίας γραφικών

1.9.2.1 OpenGL

Η OpenGL (Open Graphics Library) είναι ένα cross platform API που δημιουργήθηκε από τη Silicon Graphics το 1992. Αρχικά χρησιμοποιήθηκε για εφαρμογές CAD εικονικής πραγματικότητας και εξομοιωτές πτήσης. Υποστηρίζονταν από επαγγελματικές κάρτες γραφικών με υψηλό κόστος. Το 1997 η OpenGL άρχισε να χρησιμοποιείται και από τα παιχνίδια με αποτέλεσμα την κυκλοφορία φθηνών καρτών γραφικών με την υποστήριξη της OpenGL. Περιέχει ένα σύνολο εντολών, καθώς και την ακριβή συμπεριφορά που πρέπει να έχουν. Η OpenGL θα αναπτυχθεί αναλυτικότερα στο επόμενο κεφάλαιο [2].

1.9.2.2 DirectX

Η βιβλιοθήκη DirectX αναπτύχθηκε από την Microsoft το 1995. Αποτελεί μια συλλογή προγραμμάτων που βασίζονται στα Windows για εφαρμογές πολυμέσων, όπως τρισδιάστατα γραφικά, διαμόρφωση στοιχείων ελέγχου και ρύθμιση δικτύου. Αποτελεί μια τεχνολογία που βοηθά στη δημιουργία των ειδικών οπτικών και ηχητικών εφέ που συνοδεύουν τα παιχνίδια και χρησιμοποιείται σαν βάση για την εκτέλεση παιχνιδιών σε υπολογιστές με Windows και στην κονσόλα παιχνιδιών Xbox 360 της Microsoft. Η βιβλιοθήκη DirectX είναι το πρότυπο που προσαρμόζει το σύνολο του υλικού στις απαιτήσεις ενός παιχνιδιού και διευκολύνει την ανάπτυξή του. Βοηθά τον υπολογιστή να βελτιώσει τις επιδόσεις απαιτητικών προγραμμάτων πολυμέσων (παιχνίδια, ταινίες). Διαθέτει έντεκα εκδόσεις έως σήμερα, ενώ βρίσκεται υπό κατασκευή και η δωδέκατη έκδοση [13].

Πίνακας 1.1 Εκδόσεις του DirectX

Εκδοση DirectX	Λειτουργικό Σύστημα που υποστηρίζει	Ημερομηνία Κυκλοφορίας
DirectX 1.0		30 Σεπτεμβρίου 1995
DirectX 2.0	Χρησιμοποιούνταν για ένα σύνολο	1996

	εφαρμογών (web browsers, email)	
DirectX 2.0a	Windows 95 OSR2 και NT 4.0	5 Ιουνίου 1996
DirectX 3.0		15 Σεπτεμβρίου 1996
	Μεταγενέστερο πακέτο της DirectX 3.0 περιλαμβάνει την έκδοση Direct3D 4.04.00.0069	1996
DirectX 3.0a	Windows NT 4.0 SP3 Τελευταία υποστηριζόμενη έκδοση DirectX για τα Windows NT 4.0	Δεκέμβριος 1996
DirectX 3.0b	Μια ενημερωμένη έκδοση της 3.0a που παρουσίασε προβλήματα με την Ιαπωνική έκδοση των Windows 95	Δεκέμβριος 1996
DirectX 4.0		
DirectX 5.0	Διαθέσιμο για Windows 2000	4 Αυγούστου 1997
DirectX 5.2	H DirectX 5.2 εκδόθηκε για τα Windows 95	5 Μαΐου 1998
	Αποκλειστικά για Windows 98	25 Μαΐου 1998
DirectX 6.0	Windows CE	7 Αυγούστου 1998
DirectX 6.1		3 Φεβρουαρίου 1999
DirectX 6.1a	Αποκλειστικά για Windows 98 SE	5 Μαΐου 1999
DirectX 7.0		22 Σεπτεμβρίου 1999
	Windows 2000	17 Φεβρουαρίου 2000
DirectX 7.0a		8 Μαρτίου 2000
		2000
DirectX 7.1		14 Σεπτεμβρίου 2000
DirectX 8.0		12 Νοεμβρίου 2000
DirectX 8.0a	Τελευταία υποστηριζόμενη έκδοση για Windows 95	5 Φεβρουαρίου 2001
DirectX 8.1	Windows XP, Windows XP SP1, Windows Server 2003 και Xbox	25 Οκτωβρίου 2001
	Windows 98 και Windows 2000	8 Νοεμβρίου 2001
DirectX 8.1a	Περιλαμβάνει μια αναβάθμιση του Direct3D (D3d8.dll)	2002
DirectX 8.1b	Περιλαμβάνει μια αναβάθμιση για το DirectShow για τα Windows 2000 (Quartz.dll)	
DirectX 8.2	Ιδιο με το DirectX 8.1b αλλά περιλαμβάνει και το DirectPlay 8.2	2002
DirectX 9.0		19 Δεκεμβρίου 2002
DirectX 9.0a		26 Μαρτίου 2003
DirectX 9.0b		13 Αυγούστου 2003
DirectX 9.0c	Windows XP SP2	
		4 Αυγούστου 2004
	Windows XP SP2, SP3, Windows Server 2003 SP1 και Windows Server 2003 R2	6 Αυγούστου 2004 / 21 Απριλίου 2008

DirectX Διμηνιαίες Αναβαθμίσεις	Η έκδοση που κυκλοφόρησε στις 9 Φεβρουαρίου 2005 και αποτελεί την πρώτη έκδοση για τα 64bit Windows. Η έκδοση για τα Windows 98 κυκλοφόρησε στις 13 Δεκεμβρίου 2006 και για τα Windows 2000 στις 5 Φεβρουαρίου 2010. Τον Απρίλιο του 2006 κυκλοφορεί η πρώτη έκδοση που υποστηρίζει τα Windows Vista και τον Αύγουστο του 2009 κυκλοφορεί η πρώτη έκδοση που υποστηρίζει τα Windows 7	Κυκλοφορεί ανά δυο μήνες για το διάστημα: από Οκτώβριο 2004 έως Αύγουστο 2007. Τελευταία έκδοση κυκλοφορίας: Ιούνιος 2010
DirectX 10	Αποκλειστικά για Windows Vista	30 Νοεμβρίου 2006
DirectX 10.1	Windows Vista SP1, Windows Server 2008 περιλαμβάνει την έκδοση Direct3D 10.1	4 Φεβρουαρίου 2008
	Για εκδόσεις Windows Vista: SP2, Windows Server 2008 SP2 περιλαμβάνει την έκδοση Direct3D 10.1	28 Απριλίου 2009
DirectX 11	Windows 7, Windows Server 2008 R2	22 Οκτωβρίου 2009
	Windows Vista SP2 and Windows Server 2008 SP2, διαμέσου της πλατφόρμας αναβάθμισης για Windows Vista και Windows Server 2008	27 Οκτωβρίου 2009
	Windows 7 SP1, Windows Server 2008 R2 SP1	16 Φεβρουαρίου 2011
DirectX 11.1	Windows 7 SP1, Windows 8, Windows RT, Windows Server 2012	1 Αυγούστου 2012
DirectX 11.2	Windows 8.1, Windows RT, Windows Server 2012 R2	18 Οκτωβρίου 2013
DirectX 12	Xbox One, Windows 8, Windows 8.1, Windows Phone 8.1, Windows RT, Windows Server 2012 R2	2015

1.9.2.2.1 Χαρακτηριστικά της DirectX

Η βιβλιοθήκη DirectX περιλαμβάνει τις εξής χαρακτηριστικά – υποκατηγορίες:

Direct3D (D3D): Για τη δημιουργία 3D γραφικών. Επιτρέπει την εμφάνιση των κινούμενων γραφικών τριών διαστάσεων στην οθόνη του υπολογιστή. Η Direct3D σχεδιάστηκε ώστε να παρέχει μια ισχυρή σύνδεση ανάμεσα στην κάρτα γραφικών του υπολογιστή και στα προγράμματα λογισμικού που μπορούν να αποδίδουν αντικείμενα τριών διαστάσεων (3D). Όσο ταχύτερα επεξεργάζεται ο υπολογιστής τα κινούμενα 3D γραφικά, τόσο πιο ρεαλιστικά θα φαίνονται τα αντικείμενα 3D.

DXGI: Για την απαρίθμηση διάφορων προσαρμογέων (adapters) και τη διαχείριση των εκδόσεων της Direct3D 10 και πάνω.

Direct2D: Για τη δημιουργία 2D γραφικών.

DirectWrite: Για τη δημιουργία γραμματοσειρών

DirectSound3D (DS3D): Για την αναπαραγωγή 3D ήχων

DirectX Media: Περιλαμβάνει το DirectAnimation για 2D και 3D γραφικά, το DirectShow για την αναπαραγωγή πολυμέσων, το DirectX Transform για τη

διαδικτυακή αλληλεπίδραση και το Direct3D Retained Mode για υψηλότερης ποιότητας 3D γραφικών. Το DirectShow περιλαμβάνει ορισμένες πρόσθετες λειτουργίες της DirectX για την επεξεργασία του ηχητικού σήματος καθώς και το DirectX Video Acceleration για ταχύτερη αναπαραγωγή βίντεο.

DirectX Diagnostics (DxDiag): Πρόκειται για ένα εργαλείο για τη διάγνωση και τη δημιουργία αναφορών της DirectX (ήχος, βίντεο κ.α).

DirectX Media Objects: Χρησιμοποιείται για αντικείμενα ροής (streaming objects), όπως οι κωδικοποιητές και οι αποκωδικοποιητές.

DirectSetup: Για την εγκατάσταση του DirectX και την εύρεση της έκδοσης του DirectX που εγκαθιστάται.

DirectXMath: Παρέχει ένα βέλτιστο περιβάλλον για την αριθμητική και τη γραμμική άλγεβρα σε φορείς κινητής υποδιαστολής (2D, 3D και 4D).

XAudio2 - XACT3: Βιβλιοθήκες ήχου που αποτελούν τη βάση για την επεξεργασία του ηχητικού σήματος για τα γραφικά.

Xinput: Γραφικό περιβάλλον. Άλληλεπιδρά με την κονσόλα του Xbox 360 σε τηλεχειριστήρια παιχνιδιών, όταν συνδεθεί σε έναν υπολογιστή με λογισμικό των Windows.

Η Microsoft έχει καταργήσει τις παρακάτω κατηγορίες, αλλά μερικές από αυτές εξακολουθούν και χρησιμοποιούνται σε κάποιες εφαρμογές και πολυμέσα:

DirectDraw: Βοηθά στη δημιουργία οπτικών εφέ δύο διαστάσεων (2D). Η κάρτα γραφικών του υπολογιστή καθώς και πολλά προγράμματα λογισμικού χρησιμοποιούν το DirectDraw για τη μεταξύ τους επικοινωνία πριν την αποστολή της ολοκληρωμένης οπτικής εικόνας στην οθόνη. Τα παιχνίδια υπολογιστή, τα πακέτα γραφικών 2D και οι δυνατότητες συστήματος των Windows χρησιμοποιούν όλα το DirectDraw. Αντικαταστάθηκε από την Direct2D.

DirectInput: Για τη διασύνδεση με συσκευές εισόδου (πληκτρολόγια, ποντίκια, χειριστήρια ή άλλους ελεγκτές παιχνιδιών). Αντικαταστάθηκε από το γραφικό περιβάλλον Xinput.

DirectPlay: Για την επικοινωνία μέσω ενός τοπικού δικτύου ή ευρείας περιοχής. Μετά την 8^η έκδοση της αντικαταστάθηκε από την δωρεάν υπηρεσία παιχνιδιών Games for Windows Live της κονσόλας Xbox Live.

DirectSound: Ενισχύει την απόδοση των ηχητικών εφέ και επιτρέπει τη δημιουργία εφέ στη μίξη και στην αναπαραγωγή του ήχου. Παρέχει μια σύνδεση μεταξύ των προγραμμάτων λογισμικού και του υλικού ενός υπολογιστή. Παρέχει στα προγράμματα λογισμικού πολυμέσων (παιχνίδια και ταινίες) επιτάχυνση υλικού, δυνατότητες μίξης και πρόσβαση στην κάρτα ήχου. Αντικαταστάθηκε από βιβλιοθήκες ήχου, συμπεριλαμβανομένων του XAudio2 (ένα χαμηλό επίπεδο ήχου) και του XACT3 (ένα υψηλότερο επίπεδο ήχου).

DirectMusic: Για την αναπαραγωγή άλμπουμ μουσικής (soundtracks) Αντικαταστάθηκε από τις βιβλιοθήκες ήχου, XAudio2 και XACT3. [14]

1.9.2.2 Direct3D

Η Direct3D αποτελεί μέρος της DirectX της εταιρίας Microsoft και την πρώτη φορά παρουσιάστηκε στην έκδοση 3 της DirectX και βοηθά στην απόδοση τρισδιάστατων γραφικών. Η Microsoft ανέπτυξε την Direct3D για να ανταγωνιστεί με την OpenGL. Η Direct3D λειτουργεί ως βασικό περιβάλλον γραφικών τόσο για Xbox και Xbox 360. Ένα πλεονέκτημα της είναι η ικανότητα να εμφανίζει γραφικά σε προβολή πλήρους οθόνης και όχι μέσα σε ένα ενσωματωμένο παράθυρο. Επιπλέον προσφέρει

υπηρεσίες, όπως η χαρτογράφηση βίντεο, σωστή απεικόνιση υφής και ατμοσφαιρικές επιδράσεις.

Η γραφική έξοδος δεν απαιτεί βιβλιοθήκες τρίτων με σκοπό τη διεκπεραίωση των καθηκόντων. Η Direct3D ωστόσο, περιορίζει την πλήρη προσβασιμότητα όταν χρησιμοποιείται με λειτουργικά συστήματα εκτός των Windows. Ακόμα και όταν χρησιμοποιείται σε συνδυασμό με λογισμικό ανοιχτού κώδικα, οι χρήστες δεν μπορούν να έχουν πλήρη πρόσβαση στην λειτουργικότητα της.

Η Direct3D διαθέτει και αυτή 11 εκδόσεις έως σήμερα και ενδέχεται να κυκλοφορήσει και η έκδοση Direct3D 12 το 2015 [15].

1. 9.2.3 OpenGL vs Direct X.

Η DirectX και η OpenGL είναι δύο βιβλιοθήκες που έχουν μεγάλο ανταγωνισμό μεταξύ τους, αν και είναι σχεδόν ισοδύναμες, καθώς βασίζονται στην αρχιτεκτονική της διασωλήνωσης γραφικών. Η πιο σημαντική διαφορά εντοπίζεται στη φορητότητα τους (portability).

Η OpenGL ενσωματώνεται σε ένα μεγάλο εύρος από πλατφόρμες και λειτουργικά συστήματα, ενώ η DirectX περιορίζεται αποκλειστικά στο λειτουργικό σύστημα Windows και στην παιχνιδομηχανή Xbox.

Η DirectX ελέγχεται αποκλειστικά από τη Microsoft, ενώ η OpenGL αποτελεί μια ανοιχτή συνεργασία πολλών εταιριών [16].

Πίνακας 1.2 Διαφορές μεταξύ της OpenGL και της DirectX

Θέμα	OpenGL	DirectX
Πλατφόρμα	Windows, Linux, MacOs, Playstation 3, Gamecube, Google Android κ.α.	Windows, Xbox, Xbox 360
Γλώσσες προγραμματισμού	Καλή υποστήριξη για όλες σχεδόν τις γλώσσες	Καλή υποστήριξη για τις γλώσσες C/C++, Visual Basic και όλες τις .NET γλώσσες
Βοηθητικές βιβλιοθήκες	GLU Library. Βιβλιοθήκη με ελάχιστες συναρτήσεις	D3DX Library. Βιβλιοθήκη με πολλές συναρτήσεις
Συγγραφή κώδικα	Δεν απαιτείται η χρήση δεικτών. Κώδικας μικρότερος σε μέγεθος από την DirectX για το ίδιο αποτέλεσμα	Εκτεταμένη η χρήση δεικτών. Κώδικας μεγαλύτερος από ότι στην OpenGL για το ίδιο αποτέλεσμα

1.9.3 Glide

Το Glide ήταν ένα Graphic API αποκλειστικά για τη δημιουργία 3D γραφικών. Αναπτύχθηκε από την 3Dfx για τις κάρτες γραφικών της Voodoo και χρησιμοποιήθηκε μόνο για παιχνίδια. Βασίζεται στις αρχές της OpenGL με τη διαφορά ότι είχε κρατήσει μόνο όσες εντολές χρειάζονταν στα παιχνίδια. Αυτό είχε ως αποτέλεσμα το Glide να αποτελεί ένα μικρό API που μπορούσε να ολοκληρωθεί εξ ολοκλήρου στο hardware (κάρτες γραφικών Voodoo).

Με αυτόν τον τρόπο δημιουργήθηκαν κάποιοι περιορισμοί όπως για παράδειγμα την υποστήριξη 16bit χρώματος. Αργότερα όμως η nVidia με τη χρήση του chip Riva TNT και χρησιμοποιώντας την OpenGL και την Direct3D ξεπέρασε τις κάρτες γραφικών Voodoo με το Glide.

1.9.4 QuickDraw 3D

To QuickDraw 3D ή QD3D είναι ένα API 3D γραφικών που αναπτύχθηκε από την Apple Inc το 1995, αρχικά για τους υπολογιστές τους Macintosh.

To QD3D είναι υψηλού επιπέδου API με ένα πλούσιο σύνολο εντολών και αρχέτυπων για τη δημιουργία 3D γραφικών. Αποτελεί γενικά ένα πιο ολοκληρωμένο και πιο εύκολο API. Χρησιμοποιείται μια σειρά ιδεών της εταιρίας Apple, όπως π.χ. για το πώς η τεχνολογία 3D και το υλικό πρέπει να συνεργαστούν. Αρχικά είχαν χαμηλή απόδοση, λόγω της έλλειψης επιτάχυνσης του υλικού.

Η Apple σταμάτησε να δουλεύει με το QD3D, όταν την ανέλαβε ο Steve Jobs το 1998 ανακοινώνοντας ότι η μελλοντική υποστήριξη 3D θα βασίζεται στην OpenGL [17].

1.9.5 Mantle

Η Mantle αποτελεί μια προδιαγραφή γραφικών API που αναπτύχθηκε από την AMD ως εναλλακτική λύση για την Direct3D και την OpenGL. Προς το παρόν εφαρμόζεται αποκλειστικά για τις κάρτες γραφικών της AMD Graphics. Στόχοι της Mantle είναι 1) να επιτρέψει στα παιχνίδια και στις εφαρμογές να χρησιμοποιούν το CPU και GPU πιο αποτελεσματικά, 2) να εξαλειφθούν τα σημεία συμφόρησης της CPU, 3) επιτρέπει την αποτελεσματική κλιμάκωση σε πολλαπλούς πυρήνες, και 4) επιτρέπει τον έλεγχο των γραφικών με την κατάργηση ορισμένων στοιχείων του υλικού [18].

1.9.6 WebGL

Η WebGL (Web Graphics Library) αποτελεί μια Javascript API και βασίζεται στην OpenGL ES 2.0. Χρησιμοποιείται για την απόδοση 3D και 2D γραφικών σε οποιοδήποτε πρόγραμμα περιήγησης στο διαδίκτυο, χωρίς τη χρήση προσθέτων (plugins). Ενσωματώνεται πλήρως σε όλα τα πρότυπα web του προγράμματος περιήγησης και επιτρέπει την επιτάχυνση της GPU για την επεξεργασία της εικόνας και εμφανίζει το αποτέλεσμά στην ιστοσελίδα. Τα στοιχεία της WebGL μπορούν να «αναμειχθούν» με στοιχεία της HTML και έτσι γίνεται η σύνθεση με άλλα μέρη της σελίδας (π.χ. σύνθεση φόντου σελίδας). Τα προγράμματα της WebGL αποτελούνται από έναν κώδικα γραμμένο σε JavaScript και εκτελείται στη μονάδα επεξεργασίας γραφικών του υπολογιστή (GPU). Σχεδιάστηκε από την μη κερδοσκοπική ομάδα Khronos [19].

1.9.7 Java 3D

Η Java 3D αποτελεί ‘επέκταση’ της Java και αναπτύχθηκε από την Sun Microsystems. Αποτελεί ένα API που βασίζεται στη δημιουργία 3D εφαρμογών για την πλατφόρμα της Java.

Η επέκταση αυτή προσφέρει μια συλλογή από υψηλού επιπέδου δομές για τη δημιουργία, την απόδοση και το χειρισμό ενός 3D γράφου σκηνής ο οποίος αποτελείται από γεωμετρικά σχήματα, υλικά, φώτα, ήχους κ.α. Ο γράφος σκηνής χρησιμοποιείται για να οργανώσει και να διαχειριστεί μια 3D εφαρμογή. Ο όρος «γράφος» σκηνής χρησιμοποιείται έναντι του όρου «δέντρο» σκηνής.

Υπάρχουν δύο παραλλαγές της Java 3D: η μία στηρίζεται στην OpenGL και η άλλη στην Direct X. [20]

1.9.8 QSDK

Η QSDK είναι ένα API σκηνής γραφήματος. Ο ήχος και η κίνηση υποστηρίζονται πλήρως. Είναι διαθέσιμο σε Macintosh, PlayStation 2 και Xbox πλατφόρμες [21].

1.9.9 Simple DirectMedia Layer (SDL)

Η Simple DirectMedia Layer (SDL) είναι μια βιβλιοθήκη ανεξαρτήτου πλατφόρμας που έχει σχεδιαστεί για να παρέχει πρόσβαση χαμηλού επιπέδου για ήχο, συσκευές εισόδου και υλικό γραφικών μέσω OpenGL και Direct3D (δηλαδή όχι DirectX). Η βιβλιοθήκη SDL είναι γραμμένη σε C και είναι δωρεάν λογισμικό ανοικτού κώδικα. Οι προγραμματιστές λογισμικού το χρησιμοποιούν για να δημιουργήσουν ηλεκτρονικά παιχνίδια και άλλες εφαρμογές πολυμέσων που μπορούν να τρέξουν σε πολλά λειτουργικά συστήματα όπως Android, iOS, Linux, Mac OS X, Windows και άλλες πλατφόρμες. Διαχειρίζεται βίντεο, γεγονότα, ψηφιακό ήχο, CD-ROM, νήματα, από κοινού φόρτωση αντικειμένων, δικτύωση και χρονόμετρο [22].

1.9.10 Βιβλιοθήκη SFML(Simple and Fast Multimedia Library)

Η SFML είναι μια βιβλιοθήκη γραφικών για προγραμματισμό. Χρησιμοποιείται για τη δημιουργία παιχνιδιών και περιλαμβάνει αρκετές δυνατότητες, όπως χρήση ήχου, χρήση δικτύου και πολλά ακόμη.

Απλή και γρήγορη Βιβλιοθήκη Πολυμέσων (SFML) είναι ένα φορητό API για τον προγραμματισμό πολυμέσων. Είναι γραμμένο σε C++ με συνδέσεις (bindings) διαθέσιμες για C, D, Python, Ruby, Ocaml,.Net και Go. Μπορεί να θεωρηθεί ως μια αντικειμενοστραφής εναλλακτική λύση για την SDL.

Η Βιβλιοθήκη SFML παρέχει επιτάχυνση 2D γραφικών που χρησιμοποιούν OpenGL, υποστηρίζει το OpenGL παραθύρων και παρέχει διαφορετικές ενότητες για ευκολία στα πολυμέσα και προγραμματισμό παιχνιδιών. Η SFML ιστοσελίδα προσφέρει πλήρες πακέτο SDK σε ενιαίο πακέτο και εγχειρίδιο χρήστης για να διευκολύνει τους προγραμματιστές.

Η SFML είναι βιβλιοθήκη πολυμέσων (multimedia)

Η SFML παρέχει μια απλή διεπαφή με τις διάφορες συνιστώσες του υπολογιστή, για να διευκολύνει την ανάπτυξη των παιχνιδιών και των πολυμεσικών εφαρμογών. Αποτελείται από πέντε ενότητες: το σύστημα, το παράθυρο, τα γραφικά, τον ήχο και το δίκτυο.

Οι ενότητες που διατίθενται σήμερα είναι οι εξής:

Η μονάδα System διαχειρίζεται το ρολόι και τα νήματα (threads).

Η μονάδα Παράθυρο (Window) διαχειρίζεται τα παράθυρα και την αλληλεπίδραση των χρηστών.

Η μονάδα γραφικών καθιστά εύκολο να εμφανίζει απλά σχήματα και εικόνες.

Η μονάδα ήχου παρέχει μια διεπαφή για να χειρίστεί τους ήχους και τη μουσική.
Η μονάδα δικτύου χειρίζεται sockets με ένα φορητό τρόπο.

Η βιβλιοθήκη SFML είναι πολυπλατφορμική (multi-platform)

Με την SFML, η εφαρμογή μπορεί να μεταγλωτιστεί και να τρέξει στα πιο κοινά λειτουργικά συστήματα: Windows, Linux, Mac OS X και σύντομα Android και iOS.

Η βιβλιοθήκη SFML είναι πολυγλωσσική

Η βιβλιοθήκη SFML έχει επίσημες συνδέσεις για τη C και .Net γλώσσες. Χάρη στην ενεργή κοινότητα, είναι επίσης διαθέσιμη σε πολλές άλλες γλώσσες, όπως η Java, Ruby, Python, Go, κ.ο.κ. [23], [24],

1.9.11 Βιβλιοθήκη BPLcc_2D_Ansi_C.

Αυτή αποτελεί έναν καθαρό ανοιχτό κώδικα ο οποίος αποκρύπτεται από το χρήστη όταν δημιουργεί τη γραφική εφαρμογή του, παρέχοντας του ένα σωρό εργαλεία για να φορτώνει κάποιους στάνταρ τύπους εικόνων, να εμφανίζει αντικείμενα στην οθόνη, να ελέγχει πληκτρολόγιο και ποντίκι, να υποστηρίζει φόντα-backgrounds και να υποστηρίζεται από έξτρα κώδικα για χρήση εξωτερικών βιβλιοθηκών (φόρτωση ήχων - mp3 π.χ.) κ.τ.λ.

Η βιβλιοθήκη αυτή φτιάχτηκε σε C και λόγω της σωστής δομής, φτιάχτηκε μία σουίτα C++, την οποία οι εξοικειωμένοι χρήστες με την C++ μπορούν να την αξιοποιήσουν για πιο σύντομο και ευανάγνωστο κώδικα.

Δεν προσφέρεται για επαγγελματική ανάπτυξη εφαρμογών καθώς δεν κάνει καμιά αξιοποίηση περιφερειακών (π.χ. κάρτα γραφικών) με αποτέλεσμα να παρουσιάζει καθυστερήσεις σε μεγάλο όγκο γραφικών απεικόνισης (π.χ. περιστροφή εικόνας κ.τ.λ.). Το νόημα της βιβλιοθήκης βρίσκεται στο να αποτελεί ένα πολύ εύχρηστο εργαλείο γραφικού προγραμματισμού στη C/C++, όπου ο χρήστης δεν δίνει έμφαση στο πως θα απεικονίσει στο σύστημα τα γραφικά όσο στο πως θα οργανώσει και υλοποιήσει την λογική της εφαρμογής του.

Η βιβλιοθήκη BPLcc_2D_Ansi_C μπορεί να χρησιμοποιηθεί άνετα για διδακτικούς σκοπούς διότι:

Δίνεται η ευκαιρία στους χρήστες να δοκιμάσουν και να υλοποιήσουν απλά και σύνθετα γραφικά και αν θέλουν να μελετήσουν και τους κώδικες.

Η βιβλιοθήκη είναι ανοικτό κώδικα, υποστηρίζεται από αρκετή γραπτή πληροφορία και παραδείγματα, στα οποία μπορούν να συνεισφέρουν και οι ίδιοι οι φοιτητές ή οι ενδιαφερόμενοι, φτιάχνοντας δικά τους tutorial και ασκήσεις ή και αναπτύσσοντας την βιβλιοθήκη με έξτρα συναρτήσεις / add-ons.

Ο κώδικας της βιβλιοθήκης μπορεί να μελετηθεί, ώστε οι ενδιαφερόμενοι να πάρουν μια γενικότερη ιδέα για το πως δουλεύει το win32 sdk.

Για τους αρχάριους φοιτητές και μη, ο προγραμματισμός γίνεται πιο κατανοητός και ελκυστικός όταν καλούνται να φτιάξουν αλληλεπιδραστικά γραφικά [25].

1.9.12 Βιβλιοθήκη Allegro

Η Allegro είναι μια βιβλιοθήκη για την ανάπτυξη παιχνιδιών. Η λειτουργία της βιβλιοθήκης περιλαμβάνει υποστήριξη για βασικά 2D γραφικά, επεξεργασία εικόνας, κειμένου, με έξοδο ήχου, MIDI μουσική, εισόδους και χρονόμετρα, συμβολοσειρές (strings) Unicode, πρόσβαση στο σύστημα αρχείων, το χειρισμό αρχείων, αρχείων δεδομένων, και (περιορισμένη, μόνο λογισμικού) 3D γραφικά. Η βιβλιοθήκη είναι

γραμμένη στη γλώσσα προγραμματισμού C και έχει σχεδιαστεί για να χρησιμοποιείται με C ή C++.

Από την έκδοση 4.0, τα προγράμματα που χρησιμοποιούν τη βιβλιοθήκη λειτουργούν σε DOS, Microsoft, Windows, BeOS, Mac OS X, καθώς και διάφορα συστήματα Unix με (ή χωρίς) X Window System. Η έκδοση 5.0 υποστηρίζει τα Microsoft Windows, Mac OS X, Unix-like συστήματα, το Android, και iOS.

Η βιβλιοθήκη Allegro παρέχει τις ακόλουθες γραφικές λειτουργίες:

1) Διανυσματική σχεδίαση:

Pixels, γραμμές, ορθογώνια, τρίγωνα, κύκλοι, ελλείψεις, τόξα, σφήνες Bézier

Γέμισμα σχήματος, με ή χωρίς περίγραμμα

Πολύγωνα: επίπεδα, Gouraud, με υφή (3D) και ημιδιαφανή

2) Παλέτες χρωμάτων:

Τροποποίηση παλέτας χρωμάτων (ανάγνωση, γραφή, μετατροπή)

Μετατροπή του μοντέλου χρώματος RGB <-> HSV

3) Κείμενο:

Υποστήριξη για διαφορετικές κωδικοποιήσεις και μετατροπές, η προεπιλογή είναι UTF-8

Φόντο bitmap (κάλυψη(masking), χρωματισμός, ευθυγράμμιση)

4) Διάφορα:

Δυνατότητα σχεδίασης απευθείας πάνω στην οθόνη ή σε οποιουδήποτε μεγέθους μνήμη bitmaps

Κύλιση υλικού (σημαίνει ότι ο χρήστης μπορεί να χρησιμοποιήσει μητρώα κύλισης που προβλέπονται από το υλικό για να μετακινήσει τα περιεχόμενα της οθόνης) και τριπλό buffering (εφόσον είναι διαθέσιμο), λειτουργία X split screen (Η λειτουργία αυτή παρέχει στους χρήστες την δυνατότητα διαχωρισμού τις οθόνης στα 2), Λειτουργίες animation για τύπους FLI / FLC

Addons(πρόσθετα)

Η κοινότητα των χρηστών Allegro συνέβαλαν σε αρκετές επεκτάσεις της βιβλιοθήκης για το χειρισμό λειτουργιών όπως κύλιση tile maps και την εισαγωγή και την εξαγωγή των διαφόρων μορφών αρχείων (π.χ. PNG, GIF, JPEG εικόνες, βίντεο MPEG, OGG, MP3, IT, S3M, XM μουσική, φόντα TTF, και άλλα). Υπάρχουν επίσης συνδέσεις για πολλές γλώσσες προγραμματισμού που είναι διαθέσιμη, όπως Python, Perl, Σχέδιο, C#, D και άλλες.

Η έκδοση της Allegro 4.x και κάτω μπορεί να χρησιμοποιηθεί σε συνδυασμό με OpenGL χρησιμοποιώντας τη βιβλιοθήκη AllegroGL η οποία επεκτείνει τη λειτουργικότητα της Allegro στην OpenGL και ως εκ τούτου και του υλικού. Η έκδοση Allegro 5 υποστηρίζει την OpenGL [26].

1.9.13 Cairo

Η Cairo είναι μια βιβλιοθήκη που χρησιμοποιείται για να παρέχει γραφικά που βασίζονται σε διανύσματα, ανεξάρτητο API για προγραμματιστές λογισμικού. Είναι

σχεδιασμένη για να παρέχει βασικά δισδιάστατα σχέδια. Η βιβλιοθήκη αυτή είναι σχεδιασμένη να χρησιμοποιεί την επιτάχυνση υλικού, όταν είναι διαθέσιμη. Η βιβλιοθήκη Cairo είναι ένα δωρεάν ανοιχτού κώδικα λογισμικό και είναι γραμμένη σε C. Μια βιβλιοθήκη γραμμένη σε μια γλώσσα προγραμματισμού μπορεί να χρησιμοποιηθεί σε άλλη γλώσσα, εφόσον είναι γραμμένες οι συνδέσεις (bindings) (μια σύνδεση από μια γλώσσα προγραμματισμού σε μια βιβλιοθήκη ή υπηρεσία ενός λειτουργικού συστήματος είναι μια διεπαφή προγραμματισμού εφαρμογών (API) που παρέχει κώδικα «κόλλα» για να χρησιμοποιηθεί η βιβλιοθήκη ή υπηρεσία σε μια συγκεκριμένη γλώσσα προγραμματισμού.) Η βιβλιοθήκη Cairo έχει μια σειρά από συνδέσεις για διάφορες γλώσσες, συμπεριλαμβανομένης της C++, PHP, Factor, Haskell, Lua, Perl, Python, Ruby, Scheme, Smalltalk και πολλά άλλα.

Το μοντέλο σχεδίασης

Το μοντέλο σχεδίασης της βιβλιοθήκης Cairo είναι κάπως ανορθόδοξο και βασίζεται σε ένα μοντέλο τριών στρωμάτων.

Κάθε διαδικασία σχεδίασης πραγματοποιείται σε τρία στάδια:

Πρώτα δημιουργείται μια μάσκα, που περιλαμβάνει ένα ή περισσότερα βασικά σχήματα ή φόρμες δηλαδή κύκλους, τετράγωνα, καμπύλες Bézier, κλπ.

Στη συνέχεια πρέπει να οριστεί η πηγή που μπορεί να είναι ένα χρώμα, ένα χρώμα διαβάθμισης, ένα bitmap ή κάποια διανυσματικά γραφικά και από τα βαμμένα μέρη της πηγής δημιουργείται ένα δείγμα χρώματος με τη βοήθεια της παραπάνω καθορισμένης μάσκας.

Τέλος, το αποτέλεσμα μεταφέρεται στον προορισμό ή στην επιφάνεια, η οποία παρέχεται από το back-end για την έξοδο.

Αυτό αποτελεί μια ριζικά διαφορετική προσέγγιση από διανυσματικά γραφικά SVG [27].

1.9.14 Mesa

Η Mesa είναι μια συλλογή βιβλιοθηκών ελεύθερου ανοιχτού κώδικα που εφαρμόζουν OpenGL και πολλά άλλα APIs που σχετίζονται με επιτάχυνση υλικού 3D απόδοσης και 3D γραφικών υπολογιστών. Η βιβλιοθήκη Mesa χρησιμοποιείται στα Linux, BSD και άλλα λειτουργικά συστήματα. Επιπλέον στα APIs, η βιβλιοθήκη Mesa φιλοξενεί τις διαθέσιμες ελεύθερες υλοποιήσεις των οδηγών εγκατάστασης (drivers) γραφικών προγραμμάτων. Η ανάπτυξη της Mesa ξεκίνησε τον Αύγουστο του 1993 από τον Brian Paul, ο οποίος εξακολούθει να διατηρεί το πακέτο και περιέχει πλέον πολλές συνεισφορές από διάφορους άλλους ανθρώπους και εταιρείες σε όλο τον κόσμο, λόγω της ευρείας υιοθέτησης της [28].

1.9.15 Skia Graphics Engine

Η μηχανή γραφικών Skia (Skia Graphics Engine) είναι μια συμπαγής ανοικτού κώδικα βιβλιοθήκη γραφικών γραμμένη σε C++. Αρχικά αναπτύχθηκε από την Skia Inc, η οποία στη συνέχεια εξαγοράστηκε από την Google το 2005, η οποία στη συνέχεια κυκλοφόρησε το λογισμικό ως λογισμικό ανοικτού κώδικα. Τώρα είναι γνωστή ως Skia, χρησιμοποιείται σε Mozilla Firefox, Google Chrome, Chrome OS, Chromium OS, Sublime Text 3, Android και Firefox OS. Η βιβλιοθήκη Skia

βρίσκεται επίσης στο playbook BlackBerry, αν και η έκταση της χρήσης της είναι ασαφής [29].

1.9.16 Weaver Framework

Το πλαίσιο Weaver είναι μια βιβλιοθήκη για 2D γραφικά, γραμμένη σε C για Linux. Έχει υποστήριξη για αρχεία ήχου (χρησιμοποιώντας τη μορφή Ogg Vorbis) και αρχεία γραφικών (χρησιμοποιώντας τη μορφή PNG). Έχει επίσης μια γλώσσα σεναρίων (scripting language) που δημιουργεί βασικό κώδικα για το λογισμικό [30].

1.9.17 G2 2D Graphics Library

Αυτή η ανοιχτού κώδικα βιβλιοθήκη 2D γραφικών είναι γραμμένη σε C και έχει μια διεπαφή σε C, Fortran και Perl. Παρέχει λειτουργίες που μπορούν να χρησιμοποιηθούν για να δημιουργηθούν γραφικά για Postscript, X11, PNG, και Win32. Η βιβλιοθήκη έχει δοκιμαστεί σε Linux, AIX, Digital Unix, SunOS, IRIX, VMS και Windows NT/2000 [30].

1.9.18 GD Βιβλιοθήκη

Η GD είναι μια ανοικτή βιβλιοθήκη κώδικα για τη δυναμική δημιουργία των εικόνων από τους προγραμματιστές. Η βιβλιοθήκη GD είναι γραμμένη σε C, και τα «περιτυλίγματα» (Βιβλιοθήκες Wrapper ή περιτυλίγματα βιβλιοθήκης αποτελούνται από ένα λεπτό στρώμα του κώδικα, το οποίο μεταφράζει υπάρχουσα διεπαφή μιας βιβλιοθήκης σε ένα συμβατό περιβάλλον) είναι διαθέσιμα για Perl, PHP, Tcl, Pascal, Haskell, REXX και άλλες γλώσσες.

Η βιβλιοθήκη GD δημιουργεί εικόνες PNG, JPEG και GIF, μεταξύ άλλων μορφών. Διαθέτει λειτουργίες για να σχεδιαστούν γραμμές και τόξα, να γραφτεί κείμενο, να χρωματίσει ο χρήστης τις εικόνες, να χρησιμοποιήσει τις επιλογές αποκοπή και επικόλληση εικόνων και άλλα. Η βιβλιοθήκη συνήθως χρησιμοποιείται για να δημιουργήσει γραφήματα, γραφικά, εικονίδια, και σχεδόν οτιδήποτε άλλο. Αν και δεν περιορίζεται στη χρήση στο διαδίκτυο, οι πιο κοινές εφαρμογές του GD περιλαμβάνουν την ανάπτυξη ιστοσελίδας [31].

1.9.19 LibAfterImage

Η libAfterImage είναι μια βιβλιοθήκη επεξεργασίας εικόνας για τα Windows που υποστηρίζει τη φόρτωση, την αποθήκευση, την ανάμιξη, την απόδοση και τον χειρισμό των εικόνων. Μορφές εικόνας όπως XCF, XPM, PPM / PNM, BMP, ICO, JPEG, PNG, GIF, TIFF μπορούν να εισαχθούν στη βιβλιοθήκη, η οποία επιτρέπει επίσης να χρησιμοποιηθούν η γραμματοσειρά TrueType. Μπορεί να χρησιμοποιηθεί η βιβλιοθήκη χωρίς Windows και χειρίζεται κλιμάκωση, χρωματισμό, ημι-διαφανείς αποδόσεις 3D κειμένου, κ.λ.π. [30].

1.9.20 Libart

Η Libart είναι μια βιβλιοθήκη για υψηλής απόδοσης 2D γραφικά. Αυτή τη στιγμή χρησιμοποιείται ως μηχανή απόδοσης για το Gnome Canvas. Είναι, επίσης, η μηχανή

απόδοσης για την Gill και την Gnome Illustration app. Ο πηγαίος κώδικας του libart είναι σε Gnome CVS.

Η βιβλιοθήκη Libart είναι ελεύθερο λογισμικό (όλα τα στοιχεία είναι είτε GPL ή LGPL). Είναι επίσης διαθέσιμη για εμπορική χρήση. Υποστηρίζει ένα πολύ ισχυρό μοντέλο απεικόνισης, το ίδιο όπως το SVG και η Java 2D API. Περιλαμβάνει όλες τις λειτουργίες απεικόνισης PostScript και προσθέτει αδιαφάνεια.

Η βιβλιοθήκη Libart είναι επίσης εξαιρετικά συντονισμένη για σταδιακή απόδοση. Περιέχει δομές δεδομένων και αλγόριθμους κατάλληλους για τον ταχύ και ακριβή υπολογισμό της «περιοχής ενδιαφέροντος» (επιλεγμένο υποσύνολο των δειγμάτων σε ένα σύνολο δεδομένων που προσδιορίζονται για ένα συγκεκριμένο σκοπό. Η εικόνα μπορεί να θεωρηθεί ότι περιέχει υπο-εικόνες, καθώς και μια γραμμή παραγωγής απόδοσης (rendering pipeline) σε δύο φάσεις, βελτιστοποιημένη για διαδραστική απεικόνιση [32].

1.9.21 GraphiX

Η GraphiX είναι μια βιβλιοθήκη γραφικών για FreePascal (είναι μεταγλωττιστής) τρέχει σε DOS, Windows και Linux. Έχει διάφορες δυνατότητες, περιλαμβανομένης της στήριξης των διαφόρων αναλύσεων γραφικών από 320x200 έως 1600x1200 (και παραπάνω) χρησιμοποιώντας απευθείας το μοντέλο RGB (για 15, 16, 24 και 32 bit), λειτουργίες γραφικών 8 bit, βιβλιοθήκη του ποντικιού (διάφορα χρώματα και μεγέθη, απεριόριστοι δείκτες ποντικιού), χειρισμό βιβλιοθήκης εικόνας (φόρτωση εικόνων BMP, GIF, ICO, JPG, PCX, PBM / PGM / PPM, PNG, TGA, κλπ), βιβλιοθήκη για φόντα (FNT φόντα bitmap, φ CHR-BGI, VGA-BIOS γραμματοσειράς 16x8), βιβλιοθήκη για εφέ γραφικών (ανάμειξη alpha, λειτουργίες μάσκας, περιστροφή, κλιμάκωση), βιβλιοθήκη για εξόδο τρίγωνου (για 3D, με σκίαση, με υφή κλπ), καθώς και βίντεο και βιβλιοθήκη κινούμενων γραφικών (AVI, FLI / FLC, GIF, MOV Quicktime) [33].

1.9.22 Dislin

Είναι μια βιβλιοθήκη για την απεικόνιση δεδομένων, όπως καμπύλες, γραφήματα με μάρες, διαγράμματα πίτας, 3D γραφικά με χρώμα, 3D γραφικά, στοιχειώδη εικόνες, επιφάνειες, περιγράμματα και χάρτες. Οι εκδόσεις είναι διαθέσιμες για C, μεταγλωττιστές Fortran 77 και Fortran 90 στο Linux, FreeBSD, OpenVMS, MS-DOS, τα Windows 95/98/NT, και σε Unix. Σε ορισμένα λειτουργικά συστήματα υποστηρίζονται επίσης οι γλώσσες Java, Perl, και Python. (Δεν είναι όλες οι εκδόσεις της βιβλιοθήκης δωρεάν) [30].

1.9.23 GX2

GX2 είναι μια βιβλιοθήκη γραφικών για Borland Pascal 7 και τον μεταγλωττιστή Free Pascal. Χρησιμοποιεί μια αντικειμενοστραφή διεπαφή και υποστηρίζει τη φόρτωση και την αποθήκευση των αρχείων σε μορφή BMP, GIF, PNG, JPG και PCX. Περιέχει 65 εικόνες παρασκηνίων (φόντο-background) και υποστηρίζει φόντο τύπου True Type και τη μετατροπή των φόντων Borland XP. Οι περισσότερες από τις εργασίες στην διαχείριση εικόνων (π.χ. μετατροπή, άνοιγμα) γίνονται από τη βιβλιοθήκη αυτόματα. Άλλα χαρακτηριστικά που υποστηρίζονται περιλαμβάνουν γραμμές και κύκλους, μίξη alpha (διαφάνεια), διαφάνεια (προσθήκη / αφαίρεση

αυτής), περιστροφή, αμφιταλάντευση χρώματος (αλγόριθμος για σωστή-ακριβή επιλογή χρωμάτων), ξεθώριασμα, έγχρωμη εκτύπωση σε Epson και HP συμβατούς εκτυπωτές, και άλλα. Η βιβλιοθήκη χρησιμοποιεί 32 bit DPMI κώδικα, για να εκμεταλευτεί πλήρως το MMX [30].

1.10 Συμπεράσματα

Σε αυτήν την πτυχιακή εργασία για τη συγγραφή κώδικα γραφικών που θα αναπτύσσει ο χρήστης, χρησιμοποιείται η OpenGL για τον λόγο ότι είναι συμβατή με πολλές γλώσσες προγραμματισμού (C, C++ κ.ο.κ.), καθώς επίσης είναι συμβατή με πολλά λειτουργικά συστήματα (windows,Linux κ.ο.κ.). Επιπλέον έχει την ικανότητα να δημιουργεί υψηλής ποιότητας και απόδοσης γραφικά.

ΚΕΦΑΛΑΙΟ 2

Η ΒΙΒΛΙΟΘΗΚΗ ΣΧΕΔΙΑΣΗΣ ΓΡΑΦΙΚΩΝ OPENGL ΚΑΙ ΆΛΛΕΣ ΠΑΡΟΜΟΙΕΣ ΒΙΒΛΙΟΘΗΚΕΣ

2.1 Εισαγωγή στην OpenGL

Η OpenGL (open graphics library) αναπτύχθηκε από την Silicone Graphics Inc. το 1992. Στις αρχές χρησιμοποιήθηκε για εφαρμογές CAD εικονικής πραγματικότητας και εξομοιωτές πτήσης και υποστηριζόταν από επαγγελματικές κάρτες γραφικών οι οποίες είχαν πολύ υψηλό κόστος. Το 1997 και μετά, άρχισε να χρησιμοποιείται και για την ανάπτυξη παιγνιδιών με αποτέλεσμα την κυκλοφορία φτηνών καρτών γραφικών με υποστήριξη OpenGL.

Ο όρος OpenGL δεν αναφέρεται σε μια συγκεκριμένη βιβλιοθήκη αλλά σε ένα πρότυπο υλοποίησης βιβλιοθηκών σχεδίασης γραφικών. Εμπεριέχει το σύνολο των συναρτήσεων που πρέπει να υλοποιεί μία βιβλιοθήκη γραφικών προκειμένου να είναι συμβατή με αυτό.

Δεν υπάρχει ιδιαίτερος περιορισμός ως προς τη γλώσσα προγραμματισμού, στην οποία θα υλοποιηθεί το πρότυπο της OpenGL. Το γεγονός ότι δεν αναφέρεται σε μια συγκεκριμένη βιβλιοθήκη, αλλά σε ένα πρότυπο που ορίζει τη λειτουργικότητα μιας βιβλιοθήκης σχεδίασης, συνεπάγεται ότι ο κώδικας που θα αναπτυχθεί είναι ανεξάρτητος πλατφόρμας και μπορεί να εκτελεστεί σε οποιοδήποτε περιβάλλον προγραμματισμού, χωρίς να τροποποιηθεί ριζικά η δομή του.

Οι βιβλιοθήκες των περισσότερων μεταγλωττιστών (assemblers) περιέχουν ή μπορεί να ενσωματωθεί σε αυτούς μια υλοποίηση της OpenGL [34].

2.2 Χαρακτηριστικά της OpenGL

Αποτελεί βιομηχανικό πρότυπο Η OpenGL αποτελεί ένα πραγματικά ανοικτό, ουδέτερο και πολυπλατφορμικό πρότυπο γραφικής παράστασης.

Συνδέεται με διάφορες γλώσσες προγραμματισμού (C, C++, Java, Python, Pascal, κ.ά.). Μια βιβλιοθήκη που υλοποιεί το πρότυπο της OpenGL μπορεί να συνταχθεί σε οποιαδήποτε γλώσσα προγραμματισμού, με τη δυνατότητα προσθήκης και εξωτερικών βιβλιοθηκών ανάλογα με τη γλώσσα επιλογής (η OpenGL αποτελεί πρότυπο ανεξαρτήτου πλατφόρμας).

Αξιοπιστία και φορητότητα Όλες οι εφαρμογές OpenGL δημιουργούν γραφικά για οποιαδήποτε πλατφόρμα, ανεξάρτητα από τι είδους λειτουργικό σύστημα υπάρχει.

Εύκολη στη χρήση Οι ρουτίνες της είναι αποδοτικές με αποτέλεσμα τα προγράμματα OpenGL να έχουν λιγότερες γραμμές κώδικα από άλλα προγράμματα που γράφονται χρησιμοποιώντας άλλες βιβλιοθήκες. Επιπρόσθετα, με τις βιβλιοθήκες της OpenGL, ο χρήστης δεν είναι απαραίτητο να γνωρίζει συγκεκριμένα χαρακτηριστικά του υλικού που σχετίζονται με τη δημιουργία των γραφικών.

- 5) Οι περισσότεροι μεταγλωττιστές εμπεριέχουν ή μπορεί να ενσωματωθεί σε αυτούς μία βιβλιοθήκη της OpenGL.
- 6) Σχεδιαστικά χαρακτηριστικά

6.1) Βασικά γαρακτηριστικά:

Βασική σχεδίαση: Είναι η σχεδίαση βασικών σχημάτων που μπορεί εύκολα να δημιουργήσει ο χρήστης. Η βασική σχεδίαση περιέχει την σχεδίαση σημείων, γραμμών τρίγωνων, πολύγωνων, τετράγωνων και κύκλων και με το συνδυασμό των προηγουμένων και με άλλες λειτουργίες σχηματίζονται και οι υπόλοιπες εικόνες.

Μετασχηματισμοί: οι στοιχειώδεις μετασχηματισμοί είναι οι μετασχηματισμοί μεταπότισης, κλιμάκωσης, κλίσης και περιστροφής.

Χρώμα: Τα χρωματικά μοντέλα είναι δυο: RGB και RGBA, για το χρωματισμό φόντου και των σχημάτων.

Φωτισμός: Μεγάλο μέρος της φωτορεαλιστικής ψευδαίσθησης στα γραφικά παίζει ο φωτισμός. Όταν αναφέρεται ο προγραμματιστής στο φωτισμό, εννοεί αλγορίθμους που επιτρέπουν να υπολογιστεί τι χρώμα να δοθεί σε κάθε επιφάνεια (η ακόμα και σε κάθε pixel) των αντικειμένων που σχεδιάζονται, ώστε να φαίνονται σαν να υπάρχει μια, η περισσότερες, φωτεινές πηγές τριγύρω που τα φωτίζουν. Για να επιτεχθεί αυτό το αποτέλεσμα, χρειάζεται να εξεταστούν δύο θέματα: αλγόριθμοι σκίασης (shading), και μοντέλα φωτισμού (illumination models ή reflectance models).

Λίστες απεικόνισης: Συχνά, είναι βολικό η περιγραφή ενός σύνθετου γεωμετρικού σήματος να περικλείεται σε μια αυτόνομη ενότητα κώδικα, η οποία θα εκτελείται κάθε φορά που χρειάζεται να σχεδιαστεί αυτό το σχήμα. Στην OpenGL τη δυνατότητα αυτή δίνουν οι λίστες απεικόνισης (display lists). Οι λίστες απεικόνισης διευκολύνουν την επαναχρησιμοποίηση του κώδικα και απαλλάσσουν τον προγραμματιστή από περιττές επαναλαμβανόμενες δηλώσεις του ίδιου σύνθετου σχήματος.

6.2) Προηγμένα γαρακτηριστικά:

Απεικόνιση υφής: Είναι πολύ δύσκολο να προσομοιωθεί η μορφολογία φυσικών επιφανειών όπως η επιφάνεια ξύλου, μαρμάρου, εδάφους, γρασιδιού κ.λ.π αυτό επιτυγχάνεται με την απόδοση υφής.

Μητρώα σημείων: Για να διευκολύνει τη σύνθεση πολύπλοκων σκηνών, η OpenGL παρέχει την τεχνική των μητρώων σημείων (vertex arrays). Η χρησιμότητα των μητρώων σημείων αναδεικνύεται σε εφαρμογές μεγάλης κλίμακας, στις οποίες υπάρχει ένα πολύ μεγάλος όγκος δεδομένων (δειγμάτων με τη μορφή σημείων) και πρέπει να σχηματιστεί μια γραφική απεικόνιση αυτών των δεδομένων με τον ελάχιστο δυνατό αριθμό εντολών.

Ανάμειξη χρωμάτων: καθορίζει μια ανάμειξη λειτουργίας που συνδυάζει τιμές χρώματος από μια πηγή και έναν προορισμό. Το τελικό αποτέλεσμα είναι ότι τμήματα της σκηνής φαίνονται ημιδιαφανή.

Χειρισμό Frame buffer: είναι μια επέκταση της OpenGL για να κάνει ευέλικτη την απόδοση εκτός οθόνης (στο παρασκήνιο), συμπεριλαμβανομένης της απόδοσης σε υφή. Η αποδομένη εικόνα «αιχμαλωτίζεται» και μετέπειτα υπόκειται σε σκιάσεις ή άλλους χειρισμούς. Αυτό, επιτρέπει να πραγματοποιούνται σήμερα διάφορα εφέ στα γραφικών. (όπως θόλωση εικόνας κ.ο.κ.)

7) Τεχνικά γαρακτηριστικά Η υποστήριξη του OpenGL ES που πρακτικά σημαίνει την υποστήριξη μετάβασης εφαρμογών από σταθερούς υπολογιστές σε φορητές

συσκευές και αντίστροφα. Αναμένεται πως αυτό θα αλλάξει τον τρόπο που θα σχεδιάζονται οι μελλοντικές εφαρμογές.

8) Χρήση επεκτάσεων Παρέχονται πρόσθετες λειτουργικότητες με τη μορφή των επεκτάσεων. Οι επεκτάσεις μπορεί να εισαγάγουν νέες λειτουργίες και νέες σταθερές, και μπορούν να χαλαρώσουν ή να καταργήσουν τους περιορισμούς στις υπάρχουσες λειτουργίες της OpenGL [35], [36], [37], [38].

2.3 Κατηγορίες Βιβλιοθηκών της OpenGL

Βασική βιβλιοθήκη (*OpenGL core library* ή GL):

Η βασική βιβλιοθήκη της OpenGL έχει σχεδιαστεί ως μια βελτιωμένη διεπαφή ανεξάρτητη από το hardware, περιέχει τις κύριες εντολές σχεδίασης όπως η σχεδίαση βασικών γεωμετρικών σχημάτων, ο ορισμός χρωμάτων κλπ. Όλες οι εντολές της βιβλιοθήκης αυτής διακρίνονται από το πρόθεμα *gl*. Πολλές από τις συναρτήσεις της δέχονται προκαθορισμένα ορίσματα (συμβολικές σταθερές) τα οποία έχουν οριστεί στη βιβλιοθήκη και αντιστοιχούν σε διάφορες παραμέτρους ή καταστάσεις λειτουργίας. Οι σταθερές αυτές ξεκινούν με το πρόθεμα *gl_*.

OpenGL Utility Library (GLU):

Η βιβλιοθήκη GLU είναι χτισμένη πάνω στην κορυφή της OpenGL και παρέχει σημαντικές λειτουργίες και περισσότερα μοντέλα κτισμάτος-δημιουργίας (όπως quadric επιφάνειες). Περιλαμβάνει συναρτήσεις που εκτελούν σύνθετους αλγορίθμους όπως π.χ. τον καθορισμό μητρώων προβολής και το σχηματισμό σύνθετων καμπυλών και επιφανειών. Κάθε υλοποίηση της OpenGL εμπεριέχει τη βιβλιοθήκη GLU. Όλες οι εντολές της βιβλιοθήκης GLU ξεκινούν με το πρόθεμα *glu_* (π.χ., gluLookAt, gluPerspective).

Μερικά από τα χαρακτηριστικά της GLU περιλαμβάνουν:

Κλιμάκωση των 2D εικόνες και δημιουργία mipmap πυραμίδων (mipmap: Κατά την εφαρμογή της υφής στο αντικείμενο η OpenGL επιλέγει μια ανάλυση της υφής που ταιριάζει καλύτερα στο μέγεθος του αντικειμένου στην υφή και την εφαρμόζει)

Μετασχηματισμό από συντεταγμένες αντικειμένου σε συντεταγμένες συσκευής και αντίστροφα

Υποστήριξη για επιφάνειες NURBS

Υποστήριξη για το tessellation (ένα tessellation δημιουργείται όταν ένα σχήμα επαναλαμβάνεται ξανά και ξανά καλύπτοντας ένα σχέδιο χωρίς κενά ή επικαλύψεις.) των πολυγωνικών βασικών κοίλων ή δεσμών τόξων

Ειδικοί πίνακες μετασχηματισμού για τη δημιουργία προοπτικής και παράλληλων προβολών, τοποθέτηση κάμερας.

Απόδοση βασικού δίσκου, κυλίνδρου και σφαίρας.

Ερμηνεύοντας τις τιμές σφάλματος της OpenGL ως κείμενο ASCII

Λειτουργίες της GLU:

Αρχικοποίηση

Χειρισμός εικόνων για χρήση σε υφή

Μετασχηματισμός σημείων

Τessellating πολύγωνων
Χρήση λειτουργιών επανάκλησης
Χρησιμοποιώντας ψηφιδωτά(Tessellation) αντικείμενα
Καθορισμός επανακλήσεων
Καθορισμός του πολύγωνου ως ψηφιδωτό(Tessellated)
Απόδοση απλών επιφάνειων
Χρησιμοποιώντας NURBS καμπύλες και επιφάνειες
Χειρισμός σφαλμάτων

OpenGL Utility Toolkit (GLUT):

Η OpenGL είναι ένας γρήγορος και ευέλικτος τρόπος για την επικοινωνία με το hardware των γραφικών του υπολογιστή χωρίς να ενδιαφέρουν το χρήστη οι λεπτομέρειες υλοποίησης του. Από την άλλη, δεν προσφέρει καθόλου λειτουργίες GUI (Graphical User Interface), δηλαδή δεν έχει τη δυνατότητα να ανοίξει και να κλείσει παράθυρα στο λειτουργικό σύστημα, να ζωγραφίσει σε αυτά, ούτε να καταλάβει το πάτημα ενός πλήκτρου ή την κίνηση του ποντικιού, ούτε μπορεί να διαβάσει ένα αρχείο από το δίσκο.

Αυτό έγινε επί σκοπού, μιας και η OpenGL σχεδιάστηκε να «τρέχει» σε πολλά λειτουργικά συστήματα τα οποία έχουν το δικό τους τρόπο επικοινωνίας με την οθόνη, το ποντίκι, το δίσκο, το πληκτρολόγιο.

Για να γίνει αυτό θα πρέπει να χρησιμοποιηθούν απευθείας οι εντολές του λειτουργικού μας συστήματος (Win32 εντολές στην περίπτωση των Windows). Εναλλακτικά μπορούν να χρησιμοποιηθεί μια από τις έτοιμες βιβλιοθήκες εντολών που υπάρχουν και που θα κάνουν αυτές τις λειτουργίες για τον χρήστη εύκολα.

Μια από τις πιο διαδεδομένες βιβλιοθήκες για αυτό το σκοπό είναι το GLUT (OpenGL Utility Toolkit) που είναι και αυτή σχεδιασμένη (η εργαλειοθήκη) να τρέχει σε πολλά λειτουργικά συστήματα.

Η GLUT προσφέρει ένα σύνολο εντολών που αναλαμβάνουν να ανοίξουν και να κλείσουν εύκολα παράθυρα, να καταγράψουν το πάτημα ενός πλήκτρου ή την κίνηση του ποντικιού). Η βιβλιοθήκη αυτή περιλαμβάνει ακόμη εντολές απεικόνισης παραθύρων στην οθόνη, δημιουργία μενού, κλπ. Όλες οι εντολές της ξεκινούν με το πρόθεμα glut_.

Δεν είναι κατάλληλη να δημιουργηθεί μια κανονική εφαρμογή με αυτό (π.χ. ένα παιχνίδι) όμως είναι πολύ κατάλληλο για εκπαιδευτικές και πρότυπες εφαρμογές.

Η GLUT υποστηρίζει:

Πολλαπλά παράθυρα για OpenGL απόδοση
Οδηγός επανάκλησης διαχείρισης γεγονότων
Εξελιγμένες συσκευές εισόδου
Μια «αδρανή» ρουτίνα και χρονόμετρα
Μια απλή εγκατάσταση αναπτυσσόμενου μενού
Ρουτίνες Utility για να δημιουργήσει διάφορα συμπαγή και περιγραμμικά αντικείμενα πλαισίων.
Διάφορες λειτουργίες διαχείρισης παραθύρων

2.3.1 Βιβλιοθήκες παρόμοιες με την glut

Ενώ η GLUT θέτει το βασικό πρότυπο για μια cross-platform διαχείριση παραθύρων και εργαλειοθηκών GUI και API, υπάρχουν πολλές άλλες εργαλειοθήκες και βιβλιοθήκες κτισμένες με βάση την OpenGL σαν ανανεώσεις ή μοντέρνες αντικαταστάσεις της βιβλιοθήκης GLUT.

Τα πλαίσια εφαρμογής παρέχουν ένα στρώμα στήριξης για τους προγραμματιστές για να ελέγχουν παραθυρικά συστήματα πλατφόρμων και διαχείρισης γεγονότων. Widget Βιβλιοθήκες (βιβλιοθήκες με στοιχεία ελέγχου-controls) ή UI Πλαίσια κάνουν πιο εύκολη την εμφάνιση κοινών παραθυρικών έλεγχων, όπως εισαγωγή κειμένου και κουμπιών [46].

2.3.1.1 FREEGLUT

Η freeglut είναι μια υλοποίηση ανοικτού κώδικα που επεκτείνει την λειτουργικότητα της αρχικής GLUT. Η βιβλιοθήκη FREEGLUT σημαίνει Free Graphics Library Utility Toolkit και παρέχει κάποιες απλές λειτουργίες παραθύρων, το χειρισμό των εισόδων και τη δημιουργία μενού. Συνιστάται για βασικά προγράμματα OpenGL, αλλά όχι για προηγμένα. Η FREEGLUT λειτουργεί χρησιμοποιώντας συναρτήσεις επανάκλησης. Περνώντας τις συναρτήσεις ως παραμέτρους σε ένα χειρισμό γεγονότος στην FREEGLUT ενημερώνεται η FREEGLUT ότι χρειάζεται η συνάρτηση να κληθεί όταν το συμβάν γίνει.

2.3.1.2 CPW

CPW είναι μια βιβλιοθήκη πλαίσιου εφαρμογής για εφαρμογές OpenGL και παιχνίδια. Είναι σχεδιασμένη με ταχύτητα, απλότητα και έχοντας στο μυαλό τη φορητότητα. Η πηγή της είναι αληθινά δωρεάν λογισμικό, δεν υπάρχουν απαγορεύσεις στη χρήση της. Η διεπαφή της CPW διαμορφώθηκε μετά την GLUT API αλλά δεν είναι συμβατή με αυτό. Ο πυρήνας της βιβλιοθήκης περιλαμβάνει υποστήριξη για τη δημιουργία παραθύρων και γεγονότων, εισόδου χρήστη, δημιουργία μενού, απόδοση σε πλήρη οθόνη, χρονοδιάγραμμα απόδοσης, ομαλοποιημένες απεικονίσεις γραμματοσειρών TrueType, βασική φόρτωση της εικόνας και αποθήκευση, χρονόμετρα, πρωτόγονη ζωγραφική και πολλά άλλα.

CPW υποστηρίζει πρόσθετη λειτουργικότητα μέσω CPW Add-ons(πρόσθετα). Τα τρέχοντα πρόσθετα περιλαμβάνουν υποστήριξη για ήχο μέσω της βιβλιοθήκης ήχου OpenAL και την αυτόματη εκκίνηση της OpenGL 1.2, OpenGL 1.3, και άλλες δημοφιλείς επεκτάσεις API, συμπεριλαμβανομένου του NVidia pixel.

Σχεδιαστικοί στόχοι της βιβλιοθήκης CPW

Σταθερότητα Η GLUT έχει πάντα ένα θέμα με τη σταθερότητα. Κατά κάποιο τρόπο, αυτό βοήθησε να δημιουργήσει την αίσθηση ότι η OpenGL δεν είναι σταθερή σε πλατφόρμες όπως τα Windows. Οι προγραμματιστές της CPW προσπάθησαν να δημιουργήσουν μια σταθερή παραθυροποίηση της OpenGL και μια βιβλιοθήκη γεγονότων στη CPW μέσα από την απλότητα, την οργάνωση και ένα καθαρό στυλ κωδικοποίησης.

Μοντέρνο CPW, υποστηρίζει μοντέρνες διεπαφές που θέλουν να χρησιμοποιούν οι προγραμματιστές παιχνιδιών. Για παράδειγμα, ο προσαρμογέας πλατφόρμας Win32 υποστηρίζει τόσο την Direct Input (για την απόδοση του παιχνιδιού) όσο και την WinBase Input (για σταθερά γεγονότα που οδηγούν σε πολύ-παραθυρικές εφαρμογές.)

Ενσωματωμένες Εφαρμογές CPW έχει σχεδιαστεί για ενσωματωμένη χρήση και συνεπώς, αυτός ήταν ο πρωταρχικός στόχος του σχεδιασμού του έργου. CPW κάνει χρήση «προσαρμογέα υποδοχής» για αλληλεπιδράσεις με το τοπικό λειτουργικό σύστημα ή την εφαρμογή υποδοχής. Δεν υπάρχουν κλήσεις για έξοδο και η βιβλιοθήκη ελευθερώνει πόρους που διαθέτει. Η CPW είναι βασισμένη στο πλαίσιο και χωρίς καθόλου στατικά στην πηγή. Είναι ασφαλής για multi-threaded (πολλών νημάτων-διεργασιών) προγραμματισμό κατά την ίδια διαδικασία.

Ποιότητα Γραμματοσειράς Η CPW χρησιμοποιεί τη FreeType 2.0 βιβλιοθήκη γραμματοσειράς για την εξαιρετική ομαλοποιημένη απόδοση της γραμματοσειράς.

Διασκεδαστική Ανάπτυξη Ένα από τα πιο απολαυστικά έργα προγραμματισμού είναι ο σχεδιασμός ενός παιχνιδιού. Δυστυχώς, για να προγραμματιστεί ένα παιχνίδι σήμερα περιλαμβάνει μεγάλη πολυπλοκότητα όπως απλά ένα παράθυρο πλήρους οθόνης για να εμφανιστεί ή να ληφθεί ένα μικρό αρχείο ήχου για να παίξει. Αν αυτό παίρνει 3 μήνες μελέτης σύνθετων εγγράφων αρι μόνο για να ληφθεί ένα πολύγωνο στην οθόνη, ο προγραμματιστής του παιχνιδιού κατά πάσα πιθανότητα πρόκειται να αφήσει την ιδέα πριν καν πάει τόσο μακριά. Η CPW είναι μια προσπάθεια ώστε να ολοκληρωθούν τα πρώτα βήματα του προγραμματισμού παιχνιδιών. Επίσης, η CPW πάει ένα βήμα παραπέρα υποστηρίζοντας τις σύγχρονες διασυνδέσεις που απαιτούνται στην ανάπτυξη των παιχνιδιών σήμερα.

Αυστηρή ANSI c Η βιβλιοθήκη CPW είναι καθαρή ANSI c. Η CPW δεν χρησιμοποιεί C++, αλλά είναι κατάλληλη για χρήση σε εφαρμογές C++.

Χρήσεις της CPW

Γρήγορη πρωτοτυποποιήση μιας ιδέας παιχνιδιού

Γράφει πιο ολοκληρωμένα παιχνίδια

Γράφει σε OpenGL παραθυρικές εφαρμογές [43], [44].

2.3.1.3 Fast Light Toolkit

FLTK (προφέρεται "fulltick") είναι ένα εργαλείο cross-platform C++ GUI για UNIX / Linux (X11), Microsoft Windows, και το Mac OS X. FLTK προσφέρει μοντέρνα λειτουργικότητα GUI, υποστηρίζει 3D γραφικά μέσω της OpenGL και του ενσωματωμένου GLUT εξομοιωτή.

FLTK έχει σχεδιαστεί για να είναι αρκετά μικρή και δουλεύει μια χαρά ως κοινή βιβλιοθήκη. FLTK περιλαμβάνει επίσης ένα εξαιρετικό δημιουργό (builder) GUI που ονομάζεται FLUID που μπορεί να χρησιμοποιηθεί για τη δημιουργία εφαρμογών μέσα σε λίγα λεπτά [45].

2.3.1.4 GLFW

Η GLFW είναι ανοικτού κώδικα, πολυ-πλατφορμική βιβλιοθήκη για δημιουργία παράθυρων με περιεχόμενα OpenGL και τη διαχείριση των εισόδων και γεγονότων. Είναι εύκολο να ενταχθεί σε ήδη υπάρχουσες εφαρμογές.

Η GLFW είναι γραμμένη σε C και έχει υποστήριξη για Windows, OS X και πολλά συστήματα όπως τα Unix που χρησιμοποιούν το σύστημα X Window, όπως το Linux και το FreeBSD [46].

2.3.1.5 Εργαλειοθήκη GLOW

Η εργαλειοθήκη GLOW είναι ένα cross-platform αντικειμενοστραφές πλαίσιο για τη δημιουργία διαδραστικών εφαρμογών που χρησιμοποιούν OpenGL ή παρόμοια APIs

όπως το Mesa. Είναι, στην καρδιά του, ένα C++ περίβλημα για GLUT, παρέχοντας ένα πλήρες αντικειμενοστραφές API για τη δημιουργία παραθύρων, μενού και άλλων στοιχείων GUI και για τη διαχείριση γεγονότος. Το GLOW διαθέτει επίσης μια επεκτάσιμη cross-platform βιβλιοθήκη widget για τη δημιουργία ισχυρών διεπαφών χρήστη, μια σειρά από χρήσιμα utilities, όπως μια εφαρμογή του arcball 3D. Παρέχει την πλήρη ενσωμάτωση της OpenGL για την απεικόνιση του API. Το GLOW έχει σχεδιαστεί με την μέθοδο "Write once, run anywhere" (WORA) δηλαδή αναπτύσσεται το λογισμικό οπουδήποτε, σε οποιαδήποτε συσκευή και πρέπει να τρέχει σε οποιαδήποτε συσκευή χωρίς προβλήματα ενώ θα πρέπει ο πηγαίος κώδικας να είναι συμβατός με οποιαδήποτε πλατφόρμα που υποστηρίζει την OpenGL και την εργαλειοθήκη GLUT [47].

2.3.1.6 GLT OpenGL C++ Εργαλειοθήκη και GlutMaster

Το GLT είναι μια C++ βιβλιοθήκη κλάσεων για τον προγραμματισμό διαδραστικών 3D γραφικών με OpenGL. Μπορεί να χρησιμοποιηθεί ως μια αντικειμενοστραφής διεπαφή για OpenGL, ή ως μια βιβλιοθήκη με λειτουργικότητα για μετασχηματισμούς, σχήματα ή γραμματοσειρές κ.τ.λ.

Το GlutMaster είναι μια C++ πλατφόρμα για τη βιβλιοθήκη GLUT, παρέχει ένα φορητό παράθυρο, πληκτρολόγιο, ποντίκι και μενού για προγράμματα OpenGL. Το GlutMaster μπορεί επίσης να χρησιμοποιηθεί σε συνδυασμό με το Open Inventor. (Σημείωση: η χρήση GlutMaster με OpenInventor έχει περιορισμούς όσον αφορά στοιχεία διεπαφής χρήστη). Αυτή η έκδοση του GlutMaster είναι μια πλήρης επανεγγραφή, με την πρόσθετη υποστήριξη για τα μενού.

Το GLT είναι φορητή πλατφόρμα. Εξαρτάται μόνο από την OpenGL. Έχει συνταχθεί και δοκιμαστεί σε Windows NT με το Visual C++ 6.0 ενώ υποστηρίζονται επίσης οι Cygwin gcc, Linux και SGI IRIX.

Το GLT είναι ένα έργο σε εξέλιξη και σε καμία περίπτωση δεν καλύπτει το σύνολο των προδιαγραφών της OpenGL. Ωστόσο, περιλαμβάνει ήδη πολλές χρήσιμες κλάσεις και είναι σχεδιασμένο έτσι ώστε να μπορούν να προστεθούν εύκολα επιπλέον κλάσεις. Μπορεί επίσης να χρησιμοποιηθεί ως ένα χρήσιμο σημείο αναφοράς ή για την επίλυση κοινών προβλημάτων. Είναι διαθέσιμο υπό τους όρους της άδειας χρήσης ανοιχτού κώδικα LGPL. Αυτό επιτρέπει την εμπορική και μη εμπορική χρήση. Προορίζεται για μια ευρεία ποικιλία εφαρμογών, συμπεριλαμβανομένων και (αλλά όχι ειδικά) παιχνιδιών.

Οι GLT κλάσεις είναι βολικές, περιεκτικές, χρήσιμες και πλήρεις. Τα χαρακτηριστικά της γλώσσας C++, όπως η υπερφόρτωση τελεστών και οι αυτόματοι δημιουργοί και οι καταστροφείς βελτιώνουν τη συνοχή, την αξιοπιστία και την πυκνότητα του κώδικα. Το GLT κάνει χρήση αυτών, και συμπληρώνει τη πρότυπη C++ βιβλιοθήκη συμπεριλαμβανομένων των ροών και των επαναληπτών.

Τα προγράμματα GLT είναι φορητά. Όλες οι λειτουργίες GLT είναι φορητές και σε άλλες πλατφόρμες. Τα GLT προγράμματα τείνουν να μην εξαρτώνται από ειδικά στρώματα της πλατφόρμας για τη διαχείριση των παραθύρων, την υποστήριξη γραμματοσειρών και άλλων.

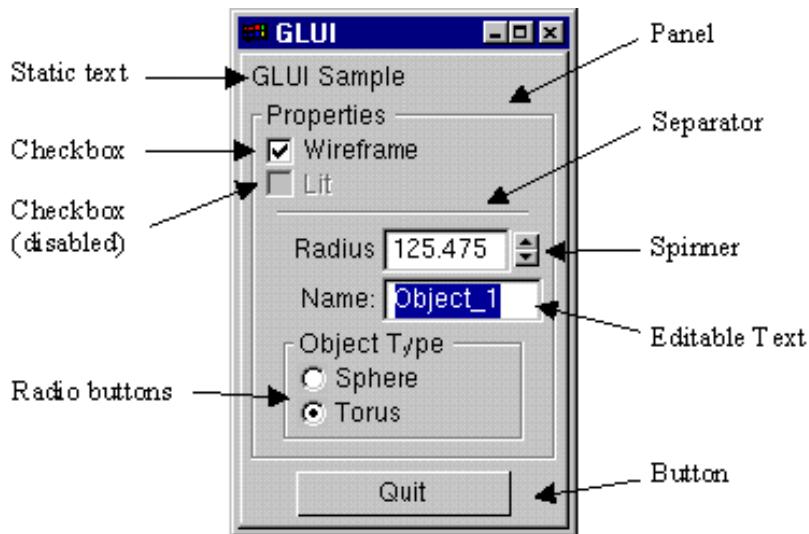
Ενώ το GLT είναι ένα περίβλημα για OpenGL, η βιβλιοθήκη επιτρέπει πλήρη πρόσβαση στα APIs, για πραγματική ευελιξία.

Τα GLT προγράμματα είναι αυτόνομα. Οι βιβλιοθήκες είναι στατικά συνδεδεμένες οπουδήποτε είναι δυνατόν, οι πηγές όπως οι υφές είναι μεταγλωτισμένες στο εκτελέσιμο αρχείο. Τα οφέλη περιλαμβάνουν την ευκολία της εγκατάστασης και την ευρωστία [48].

2.3.1.7 GLUI User Interface Library

Η GLUI είναι μια βιβλιοθήκη διεπαφής χρήστη που χρησιμοποιεί εντολές της C++ και είναι βασισμένη στη GLUT. Παρέχει στοιχεία ελέγχου όπως κουμπιά, πλαίσια ελέγχου, κουμπιά επιλογής σε εφαρμογές OpenGL. Είναι ανεξάρτητο παραθυρικό σύστημα, χρησιμοποιώντας GLUT ή FreeGLUT.

Παρακάτω φαίνεται ένα παράδειγμα GLUI παραθύρου που δείχνει τους διαφορετικούς έλεγχους. Το παράθυρο αυτό αποδίδεται εξ ολοκλήρου με την OpenGL. Συνεπώς, φαίνεται το ίδιο σε PC, Mac, SGI, καθώς και άλλα συστήματα Unix, χρησιμοποιεί είτε την εφαρμογή της SGI της OpenGL, της Microsoft ή της Mesa.



Σχήμα 2.1 Παράδειγμα GLUI παραθύρου

Χαρακτηριστικά της GLUI User Interface Library περιλαμβάνουν:

Πλήρης ενσωμάτωση με GLUT εργαλειοθήκη

Απλή δημιουργία ενός νέου παραθύρου διεπαφής χρήστη με μια μόνο γραμμή κώδικα

Υποστήριξη για πολλαπλά παράθυρα διεπαφή χρήστη

Πρότυπο ελέγχου διεπαφής χρήστη, όπως:

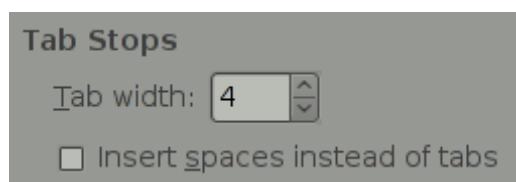
κουμπιά

Πλαίσια ελέγχου για boolean μεταβλητές

Κουμπιά επιλογής για αλληλοαποκλειόμενες επιλογές

Επεξεργάσιμα πλαίσια κειμένου για την εισαγωγή κειμένου, ακεραίων μεταβλητών και κινητής υποδιαστολής τιμές.

Spinners (αυξομειώνει νούμερα) για διαδραστικό χειρισμό ακεραίων και κινητής υποδιαστολής τιμές



Σχήμα 2.2 Spinners

Στατικά πεδία κειμένου

Πάνελ για την ομαδοποίηση του συνόλου των ελέγχων (controls)

Διαχωριστικές γραμμές για να βοηθήσει να οργανωθούν οπτικά οι ομάδες των ελέγχων.

Οι έλεγχοι μπορούν να δημιουργήσουν ανακλήσεις όταν αλλάζουν οι τιμές τους
Οι μεταβλητές μπορούν να συνδεθούν με τους ελέγχους και να ενημερώνονται αυτόματα, όταν η αξία των ελέγχων αλλάζει («live μεταβλητές»)

Οι έλεγχοι μπορούν να συγχρονιστούν αυτόματα ώστε να αντικατοπτρίζει τις αλλαγές στον τομέα των ζώντων μεταβλητών.

Οι έλεγχοι μπορούν να προκαλέσουν στην GLUT γεγονότα επανέκθεσης όταν αλλάζουν οι τιμές τους.

Η διάταξη και το μέγεθος των ελέγχων είναι αυτόματη.

Οι έλεγχοι μπορούν να ομαδοποιηθούν σε στήλες.

Ο χρήστης μπορεί να κάνει ελέγχους χρησιμοποιώντας το πλήκτρο Tab

Τι νέο υπάρχει στην έκδοση 2

Η GLUI έκδοση 2 περιλαμβάνει τις παρακάτω νέες δυνατότητες και στοιχεία ελέγχου:

Το GLUI ελέγχει μέσα στο κύριο παράθυρο γραφικών.

Λειτουργίες για καθαρή καταστροφή παραθύρων GLUI και υποπαραθύρων.

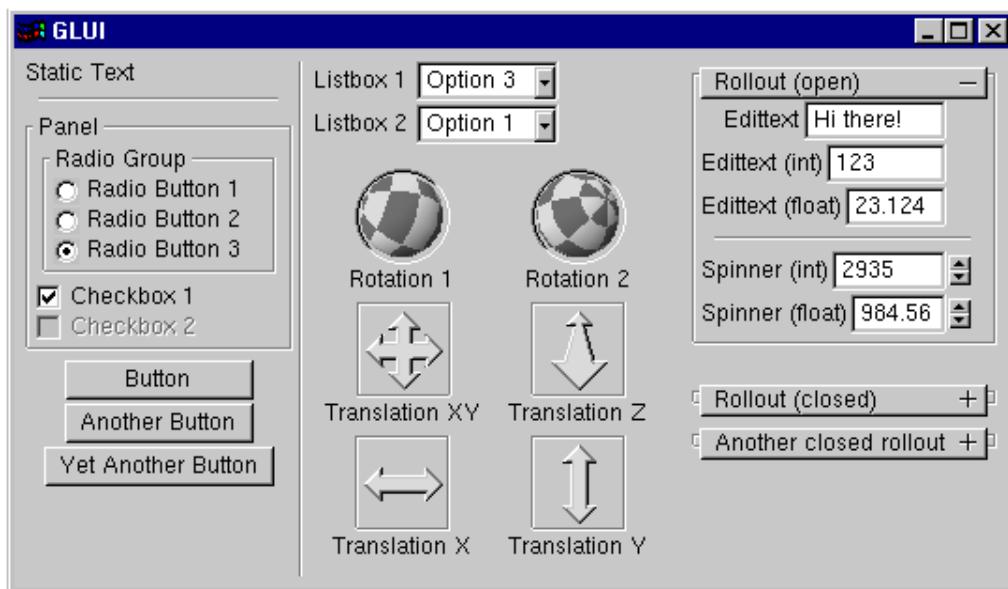
Λειτουργίες για εμφάνιση και κρύψιμο παραθύρων GLUI και υποπαραθύρων.

Μια sync_live_all () για τον αυτόματο συγχρονισμό όλων των ζωντανών μεταβλητών σε όλα τα παράθυρα GLUI ταυτόχρονα.

Rollouts - πτυσσόμενο πάνελ για τη μείωση της ακαταστασίας στην οθόνη.

Listboxes - επιτρέπει στο χρήστη να επιλέξει ένα στοιχείο από μια λίστα strings.

Περιστροφή και μετάφραση ελέγχων - για εύκολη λήψη 3D εισόδου αλληλεπίδρασης από το χρήστη [49].



Σχήμα 2.3 Ορισμένα χαρακτηριστικά της GLUI

2.3.1.8 NGL

NGL είναι ένα πλαίσιο εφαρμογής παρόμοιο με πολλούς τρόπους με το GLUT, αλλά με πολλές βελτιώσεις για να κάνουν ένα εύκολο και όμως ισχυρό εργαλείο για cross-platform (ανεξάρτητου πλατφόρμας) εφαρμογής σε C++ προγραμματισμό [43].

2.3.1.9 SDL

Η Simple DirectMedia Layer είναι μια βιβλιοθήκη ανεξαρτήτου πλατφόρμας, έχει σχεδιαστεί για να παρέχει πρόσβαση σε χαμηλό επίπεδο ήχου, πληκτρολόγιου, ποντικού, joystick και το υλικό των γραφικών μέσω OpenGL και Direct3D. Χρησιμοποιείται από το λογισμικό αναπαραγωγής βίντεο, εξομοιωτές και δημοφιλή παιχνίδια.

Η SDL υποστηρίζει επίσημα τα Windows, Mac OS X, Linux, iOS και Android. Η υποστήριξη για άλλες πλατφόρμες μπορεί να βρεθεί στον πηγαίο κώδικα. Είναι γραμμένη σε C και λειτουργεί με C++ ενώ υπάρχουν συνδέσεις που διατίθενται για διάφορες άλλες γλώσσες, συμπεριλαμβανομένης της C# και Python.

Η SDL 2.0 διανέμεται υπό την άδεια zlib. Αυτή η άδεια επιτρέπει να χρησιμοποιείται η SDL ελεύθερα σε οποιοδήποτε λογισμικό [57].

2.4 Άλλες βιβλιοθήκες της OpenGL

2.4.1 Βιβλιοθήκες Χαμηλότερου Επιπέδου της OpenGL

2.4.1.1 OpenGL Mathematics (GLM)

Η OpenGL Mathematics (GLM) είναι μια C++ βιβλιοθήκη για τα μαθηματικά για 3D λογισμικό που βασίζεται στις προδιαγραφές του OpenGL Shading Language (GLSL) [50].

2.4.1.2 Libktx

Η Libktx, μέρος του σετ εργαλείων KTX, είναι μια βιβλιοθήκη λειτουργιών για την εγγραφή αρχείων μορφής KTX και εμφανίσεων των GL υφών από αυτά [50].

2.4.1.3 OpenSceneGraph

Η OpenSceneGraph είναι μια υψηλού επιπέδου 3D εργαλειοθήκη γραφικών με δυνατότητες της OpenGL, παρέχοντας παράλληλα πολλές δικές της δυνατότητες. Η OpenSceneGraph μπορεί να υπερηφανεύεται για μια μεγάλη κοινότητα χρηστών και έχει χρησιμοποιηθεί για την οπτική προσομοίωση, παιχνίδια, εικονική πραγματικότητα επιστημονικής απεικόνισης και τη μοντελοποίηση [50].

2.4.1.4 GLX

Το GLX χρησιμοποιείται για εφαρμογές OpenGL στα Unix, αλλά και για τη διαχείριση της αλληλεπίδρασης με το σύστημα X Window (σύστημα παραθύρων για οθόνες bitmap σε περιβάλλον UNIX).

Υποστηρίζει υλικό για επιτάχυνση απόδοσης, ανάγνωση για την προεπεξεργασία των δεδομένων σε ένα offscreen παράθυρο και άμεση είσοδο βίντεο και FBConfig, ένα πιο

ισχυρό και ευέλικτο περιβάλλον για την επιλογή διαμορφώσεων ενός ρυθμιστικού πλαισίου. Δίνει τη δυνατότητα σε προγράμματα που χρησιμοποιούν την OpenGL να «προβάλλονται» μέσα από παράθυρα (μέσω του συστήματος X Window) [50].

2.4.1.5 DRI

Το DRI δηλαδή Direct Rendering Infrastructure επιτρέπει την επιτάχυνση του υλικού για 3D γραφικά στο Linux. Πιο συγκεκριμένα, πρόκειται για μια αρχιτεκτονική λογισμικού για το συντονισμό του πυρήνα στα Linux, για το σύστημα παραθύρων X, για 3D γραφικών στο υλικό και μιας μηχανής απόδοσης βασισμένη στην OpenGL.

Το Direct Rendering Infrastructure, επίσης γνωστό ως DRI είναι ένα πλαίσιο που επιτρέπει την άμεση πρόσβαση στο γραφικά του υλικού στο X Window System με ασφαλή και αποτελεσματικό τρόπο. Περιλαμβάνει αλλαγές στον X server, σε διάφορες client βιβλιοθήκες, και στον πυρήνα (kernel). Η πιο σημαντική χρήση του DRI είναι η δημιουργία γρήγορων εφαρμογών OpenGL παρέχοντας επιτάχυνση στο υλικό στην εφαρμογή Mesa. Σε αρκετά 3D γραφικά έχουν επιταχυνθεί οι οδηγοί και έχουν γραφτεί με τις προδιαγραφές DRI, συμπεριλαμβανομένων οδηγών για chipsets που παράγονται από 3DFX, AMD (πρώην ATI), Intel και η Matrox [51], [52].

2.4.2 Βιβλιοθήκες Ανωτέρου Επιπέδου της OpenGL

2.4.2.1 Open Inventor by VSG

Η Open Inventor είναι μια αντικειμενοστραφής, ανεξαρτήτου πλατφόρμας (cross-platform) 3D εργαλειοθήκη γραφικών για την ανάπτυξη της βιομηχανικής ισχύος και διαδραστικών εφαρμογών χρησιμοποιώντας C++, .NET ή Java. Εύκολη στη χρήση της, είναι επεκτάσιμη αρχιτεκτονική και το μεγάλο σύνολο των προηγμένων συστατικών παρέχει στους προγραμματιστές μια πλατφόρμα υψηλού επιπέδου για την ταχεία προτυποποίηση και την ανάπτυξη των εφαρμογών 3D γραφικών.

Η Open Inventor από την VSG είναι η εμπορική, η τρέχουσα εξέλιξη της Open Inventor και παρέχει μια ενημερωμένη, ιδιαίτερα βελτιστοποιημένη, πλήρως εξοπλισμένη εφαρμογή δημοφιλούς αντικειμενοστραφούς γράφου σκηνής API για C++, .NET και Java. Οι αιτήσεις που τροφοδοτούνται από την Open Inventor και από την VSG, επίσης επωφελούνται από ισχυρές επεκτάσεις όπως VolumeViz LDM για πολύ μεγάλο όγκο δεδομένων, MeshViz XLM για την υποστήριξη του πλέγματος υψηλής απόδοσης ή ScaleViz για multi-GPUs και immersive VR [51], [53].

2.4.2.2 Coin

Η Coin είναι μια 3D βιβλιοθήκη γραφικών, C++ API που βασίζεται στην Open Inventor 2.1 API και την OpenGL. Λειτουργεί σε Win32, Linux, IRIX και Mac OS X. Η Coin είναι ο πυρήνας της Coin3D. Η Coin3D είναι ένα υψηλού επιπέδου, retained – mode εργαλειοθήκη για την αποτελεσματική ανάπτυξη 3D γραφικών.

Coin βασίζεται στο API της βιβλιοθήκης Open Inventor, αλλά αναπτύχθηκε ανεξάρτητα από το μηδέν πριν η SGI Open Inventor γίνει ανοικτό κώδικα. Δεν μοιράζεται κανένα κώδικα με SGI Open Inventor εκτός από τυχαίες συμπτώσεις καθοδηγείται από την Open Inventor API design. Η Coin επιτυγχάνει το στόχο της Open Inventor 2.1 για συμβατότητα, από το φθινόπωρο του 2000 και από τότε έχει επεκταθεί με μια τεράστια σειρά από πρόσθετα χαρακτηριστικά, που κυμαίνονται από την υποστήριξη 3D ήχου σε GLSL υποστήριξη shader, πρόσθετες μορφές αρχείων,

όπως VRML97 και ένα μεγάλο τον αριθμό των εσωτερικών αλλαγών για να συμβαδίσουν με τις νεότερες, πιο αισιόδοξες τεχνικές απόδοσης (rendering) OpenGL που δεν ήταν διαθέσιμες παλαιότερα [54].

2.4.2.3 Gizmo 3D

Το Gizmo3D είναι ένας υψηλής απόδοσης γράφος σκηνής που βασίζεται σε OpenGL 3D και είναι αποτέλεσμα της οπτικοποίησης της εργαλειοθήκης C++ για Linux, WIN32 και IRIX, που χρησιμοποιούνται στο παιχνίδι ή σε VisSim ανάπτυξη. Είναι παρόμοιο με άλλες μηχανές σκηνικών γραφής όπως Cosmo3D/Performer/ Fahrenheit/ Inventor/VisKit/Vtree, αλλά είναι μια πολλαπλή πλατφόρμα με πολύ υψηλή απόδοση. Προσθέτει, επίσης, κάποια πιο προηγμένα χαρακτηριστικά της τέχνης για αποτελεσματικές παρουσιάσεις και αλληλεπίδρασεις γραφημάτων [51].

2.4.2.4 OSG

Το OSG είναι ένα ανοικτού κώδικα υψηλής απόδοσης 3D εργαλείο γραφικών, που χρησιμοποιείται από προγραμματιστές εφαρμογών σε τομείς όπως η οπτική προσομοίωση παιχνιδιών, η εικονική πραγματικότητα, η επιστημονική απεικόνιση και μοντελοποίηση. Γραμμένο εξ ολοκλήρου σε C++ και OpenGL τρέχει σε όλες τις πλατφόρμες των Windows, OSX, Linux, IRIX, Solaris και των λειτουργικών συστημάτων FreeBSD [55].

2.4.2.5 OpenRM

Το OpenRM είναι ένα περιβάλλον ανάπτυξης ανοικτού κώδικα που χρησιμοποιείται για την κατασκευή φορητών 2D/3D/stereo γραφικών υψηλών επιδόσεων, για εικονική πραγματικότητα, για εφαρμογές οπτικοποίησης και παιχνίδια για Unix/X11 και Win32 πλατφόρμες. Πρόκειται για ένα γράφο σκηνής API που χρησιμοποιεί OpenGL ως πλατφόρμα γραφικών και επιτάχυνση του υλικού για γραφικά. Ένας γράφος σκηνής είναι ένας χρήσιμος τρόπος ώστε να οργανωθούν τα δεδομένα για την απόδοση με ένα τρόπο που είναι ιδιαίτερα αποτελεσματικός για τις μηχανές απεικόνισης γραφικών (στις περισσότερες περιπτώσεις) [51].

2.5 Μέθοδος εκτέλεσης του κώδικα του χρήστη και διαδικασία εμφάνισης του γραφήματος στην οθόνη

Ο χρήστης για να κάνει αρχικά compile τον κώδικα του, τρέχει πρώτα το πρόγραμμα με τον C++ compiler (που στην εφαρμογή έχει ονομαστεί cl compiler), ο οποίος αποτελεί ένα utility του Microsoft Visual Studio.

Όταν τρέχει το πρόγραμμα:

1. Στη περίπτωση που ο κώδικας του χρήστη περιέχει λάθη, ο μεταγλωττιστής cl εμφανίζει τα λάθη, καθώς και τον αριθμό γραμμής που βρίσκεται το καθένα από αυτά στο ειδικό πλαίσιο της εφαρμογής (Building Status)
2. Στη περίπτωση που ο κώδικας του χρήστη δεν περιέχει συντακτικά λάθη, ο μεταγλωττιστής cl διαβάζει το αρχείο τύπου.cpp και δημιουργεί ένα αρχείο τύπου .obj. Στην εφαρμογή, παρότι δημιουργείται το αρχείο .cpp με τον κώδικα του χρήστη, η εφαρμογή δεν μπορεί να το ‘διαβάσει’ και να το τρέξει, εμφανίζοντας το επιθυμητό αποτέλεσμα στην οθόνη. Αυτό συμβαίνει διότι η εφαρμογή ‘βλέπει’ τον κώδικα που αναπτύσσει ο χρήστης, όπως και το κάθε βασικό παράδειγμα της εφαρμογής σαν

ολοκληρωμένο project (αρχεία τύπου.vcxproj) και όχι σαν απλό αρχείο αντικειμένου (.obj)

Για τον παραπάνω λόγο, χρησιμοποιείται ένα επιπλέον utility του Microsoft Visual Studio, ο MsBuild compiler. Πρόκειται για έναν μεταγλωττιστή που διαβάζει αρχεία τύπου.vcxproj. Εφόσον καλείται ο συγκεκριμένος μεταγλωττιστής και δεν υπάρχει κάποιο λογικό σφάλμα στον κώδικα, μέσω του linker ενώνονται οι απαραίτητες βιβλιοθήκες της εφαρμογής με το αρχείο vcxproj και έπειτα δημιουργείται το αρχείο τύπου .exe, το οποίο εμφανίζει το γράφημα.

Ο χρήστης έχει τη δυνατότητα από τη στιγμή που θα τρέξει το πρόγραμμα, μέχρι και την εμφάνιση του γραφήματος να βλέπει και τη διαδικασία που έγινε μέχρι να εμφανιστεί το γράφημα στο ειδικό πλαίσιο (Building Status).

ΚΕΦΑΛΑΙΟ 3

ΑΝΑΛΥΣΗ – ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΛΟΓΙΣΜΙΚΟΥ ΠΟΥ ΕΧΕΙ ΑΝΑΠΤΥΧΘΕΙ ΚΑΘΩΣ ΚΑΙ ΤΩΝ ΠΡΟΒΛΗΜΑΤΩΝ ΠΟΥ ΠΑΡΟΥΣΙΑΣΤΗΚΑΝ.

3.1 Εισαγωγή

Στο κεφάλαιο αυτό αναλύονται όλες οι αναγκαίες επιλογές που λήφθηκαν για την υλοποίηση της εφαρμογής. Σε πρώτη φάση γίνεται αναφορά στο πακέτο ανάπτυξης εφαρμογών που επιλέχθηκε και στους λόγους οι οποίοι οδήγησαν στην επιλογή αυτή. Έπειτα ακολουθεί μια λεπτομερής προσέγγιση της υλοποίησης του συστήματος, μέσα από την αναφορά στα δομικά στοιχεία, στο τρόπο υλοποίησης τους, καθώς και στο είδος των προβλημάτων που παρατηρήθηκαν κατά την κατασκευή του.

3.2 Επιλογή γλώσσας προγραμματισμού και πλατφόρμας για την ανάπτυξη της εφαρμογής

Η εφαρμογή υλοποιείται με το πρόγραμμα Microsoft Visual Studio 2010 και τη γλώσσα προγραμματισμού C# (Csharp). Ο κώδικας που θα εκτελείται για την OpenGL (γραφικά) θα τρέχει με τις βιβλιοθήκες της C++.

Η γλώσσα προγραμματισμού C# επιλέχθηκε για την υλοποίηση της εφαρμογής διότι: Έχει τη δυνατότητα για code compilation για ήδη υπάρχουσα κλάση (εξωτερικού κώδικα πάντα). Μπορεί να κάνει compile μέσα από ένα string ή ένα αρχείο.

Διαχειρίζεται εύκολα τα set up (εκτελέσιμα) αρχεία, καθώς επίσης και τα references (αναφορές - εξωτερικού κώδικα) αυτόματα. Τα references περιέχουν τις βασικές βιβλιοθήκες της C# και της OpenGL.

Διαχειρίζεται καλύτερα τις μεταβλητές τύπου string. Γενικά η C#, διαθέτει έτοιμες βιβλιοθήκες, είναι πιο εύχρηστη για να κάνει compile σε string αρχεία και παρέχει πιο ολοκληρωμένες λύσεις.

3.3 Προβλήματα και λύσεις

Είδος των προβλημάτων που παρατηρήθηκαν κατά την ανάπτυξη της εφαρμογής και οι λύσεις που δόθηκαν.

Πρόβλημα 3.3.1

Αρχικά στην εφαρμογή για την ανάπτυξη του κώδικα του φοιτητή δοκιμάστηκε η C#, αλλά για να εκτελεί κώδικα OpenGL θα έπρεπε ο χρήστης να ενσωματώσει εξωτερικές βιβλιοθήκες, οι οποίες έχουν υλοποιηθεί από ανεξάρτητους προγραμματιστές (παραδείγματος χάριν δεν ενσωματώθηκαν από την Microsoft). Επιπλέον, προστέθηκε στη C# η βιβλιοθήκη TaoGL, διότι δεν υπάρχει αυτόνομη OpenGL στην C#, αλλά υπήρχαν προβλήματα ως προς την εκτέλεση του κώδικα, οπότε αφαιρέθηκε.

Γενικότερα δεν αποτελεί σωστή πρακτική η χρήση βιβλιοθηκών από ανεξάρτητους προγραμματιστές σε εκπαιδευτικές εφαρμογές, διότι δεν είναι δεδομένο ότι λειτουργούν σωστά, οπότε ο χρήστης δεν θα μπορούσε να ξεχωρίσει αν τα λάθη που τυχόν υπήρχαν στην εκτέλεση του κώδικα του ήταν από τον ίδιο ή από τη μη

λειτουργικότητα κάποιας ενσωματωμένης εξωτερικής βιβλιοθήκης. Αν το σφάλμα συμβεί λόγω της βιβλιοθήκης, ο χρήστης θα είναι αδύνατο να μπορέσει να το εντοπίσει.

Λύση προβλήματος 3.3.1

Τελικά, η γλώσσα προγραμματισμού που θα εκτελεί κώδικα OpenGL ο χρήστης θα είναι η C++, επειδή ενσωματώνει τις απαραίτητες βιβλιοθήκες για την εκτέλεση κώδικα OpenGL και δεν χρειάζεται η προσθήκη εξωτερικών βιβλιοθηκών.

Πρόβλημα 3.3.2

Βασική ιδέα για το διαχωρισμό των αρχείων που θα ενώνονται με την εκτέλεση του κώδικα, ήταν η ύπαρξη τριών αρχείων:

Start.cpp: Περιέχει όλες τις βασικές βιβλιοθήκες του προγράμματος που αναπτύσσει ο χρήστης (#include...)

Main.cpp: Περιέχει τον κώδικα του χρήστη

End.cpp: Κλείνει την main () και θα είχε την εξής μορφή:

Start.cpp

```
# include ....  
  
//θα περιέχονται μόνο οι απαραίτητες βιβλιοθήκες
```

Main.cpp

```
void main () {  
  
    //κώδικας του φοιτητή  
  
    gl....  
  
    //μόνο εντολές της OpenGL
```

End.cpp

```
} //κλείσιμο της main
```

```

#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glut.h>

void main (int argc, char **argv)
{
    glClearColor(1.0,1.0,1.0,0.0) // βάζει χρώμα στο backround
    glColor3f(0.0f,0.0f,0.0f) // θέτει το χρώμα του αντικειμένου
    glPointSize(4.0) // θέτει ένα σημείο που θα έχει διαστάσεις 4x4 pixels
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 500.0, 0.0, 400.0); // περιοχή προβολής σε παγκόσμιες συντεταγμένες
    glClear(GL_COLOR_BUFFER_BIT); // «καθαρισμός» οθόνης
    glBegin(GL_POINTS);
        glVertex2i (100, 50);
        glVertex2i (100, 130);
        glVertex2i (150, 130);
    glEnd();
    glFlush(); //εμφανίζει το αποτέλεσμα στην οθόνη
    glutInit (&argc, argv); //αρχικοποίηση της εργαλειοθήκης
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB); // ρύθμιση της απεικόνισης του
    αντικειμένου στην οθόνη
    glutInitWindowSize (640, 480); //καθορίζει το μέγεθος του παραθύρου που θα εμφανίζεται το
    αντικείμενο
    glutInitWindowPosition (10, 10); // θέτει την αρχική θέση για το παράθυρο απεικόνισης
    glutCreateWindow ("Το πρώτο μου πρόγραμμα σε OpenGL!"); // δημιουργεί το παράθυρο και
    θέτει τον τίτλο του
    glutDisplayFunc (myDisplay);
    myInit(); // επιπλέον αρχικοποιήσεις όπου είναι απαραίτητο
    glutMainLoop();
}

```

Διαπιστώθηκε στη συνέχεια, ότι με την παραπάνω δομή που είχε ο κώδικας, ο χρήστης θα έγραφε ολόκληρο τον κώδικα στην main χωρίς να μπορεί ο ίδιος να υλοποιήσει δικές του συναρτήσεις. Με αυτόν τον τρόπο θα περιόριζε το χρήστη να υλοποιεί δικές του συναρτήσεις και να κάνει τον κώδικα του πιο δομημένο, οπότε η δομή έπρεπε να αλλάξει για να παρέχει στον χρήστη και αυτή τη δυνατότητα.

Λύση προβλήματος 3.3.2

Προκύπτουν δύο αρχεία τα οποία είναι:

Start.cpp: Περιέχει όλες τις βασικές βιβλιοθήκες του προγράμματος που αναπτύσσει ο χρήστης (#include...)

Main.cpp: Περιέχει τον κώδικα του χρήστη

Η τελική δομή που θα έχει ο κώδικας του χρήστη θα είναι της παρακάτω μορφής:

Start.cpp

```

#include .....

//θα περιέχονται μόνο οι απαραίτητες βιβλιοθήκες

```

Main.cpp

```
//ολόκληρος ο κώδικας του φοιτητή
```

```
void customFunction () {  
  
    //συναρτήσεις του χρήστη  
}  
//
```

Ενδεικτικό παράδειγμα:

```
#include <windows.h>  
#include <gl\gl.h>  
#include <gl\glu.h>  
#include <gl\glut.h>  
void myInit (void)  
{  
    glClearColor(1.0,1.0,1.0,0.0) // βάζει χρώμα στο background  
    glColor3f(0.0f,0.0f,0.0f) // θέτει το χρώμα του αντικειμένου  
    glPointSize(4.0) // θέτει ένα σημείο για να έχει διαστάσεις 4x4 pixels  
    glMatrixMode(GL_PROJECTION);  
    glLoadIdentity();  
    gluOrtho2D(0.0, 500.0, 0.0, 400.0); // περιοχή προβολής σε παγκόσμιες συντεταγμένες  
}  
void myDisplay(void)  
{  
    glClear(GL_COLOR_BUFFER_BIT); // «καθαρισμός» οθόνης  
    glBegin(GL_POINTS);  
    glVertex2i (100, 50);  
    glVertex2i (100, 130);  
    glVertex2i (150, 130);  
    glEnd();  
    glFlush(); //εμφανίζει το αποτέλεσμα στην οθόνη  
}  
void main (int argc, char **argv)  
{  
    glutInit (&argc, argv); //αρχικοποίηση της εργαλειοθήκης  
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB); // ρύθμιση της απεικόνισης του αντικειμένου  
    στην οθόνη  
    glutInitWindowSize (640, 480); //καθορίζει το μέγεθος του παραθύρου που θα εμφανίζεται το  
    αντικείμενο  
    glutInitWindowPosition (10, 10); // θέτει την αρχική θέση για το παράθυρο απεικόνισης  
    glutCreateWindow ("Το πρώτο μου πρόγραμμα σε OpenGL!"); // δημιουργεί το παράθυρο και θέτει τον  
    τίτλο του  
    glutDisplayFunc (myDisplay);
```

```
myInit(); // επιπλέον αρχικοποιήσεις όπου είναι απαραίτητο  
glutMainLoop();}
```

Πρόβλημα 3.3.3

Ο χρήστης ανοίγει την εφαρμογή και ξεκινά να αναπτύσσει τον κώδικα του. Όταν τον εκτελεί, το αποτέλεσμα εμφανίζεται σε νέο παράθυρο που απεικονίζει το γράφημα. Σε περίπτωση όμως που χρειαστεί να γίνουν κάποιες τροποποιήσεις (π.χ. προσθήκη παραδειγμάτων ή εντολών, δημιουργία συναρτήσεων, κ.λ.π.) στον κώδικα και ξαναεκτελεστεί, συνεχίζει να εμφανίζεται το αποτέλεσμα της αρχικής μορφής του κώδικα στο παράθυρο απεικόνισης, κάτι που δεν είναι λογικά ορθό και αποτελεί βασικό πρόβλημα διότι με αυτόν τον τρόπο ο χρήστης δεν θα μπορεί να συνεχίσει την εργασία του. Για παράδειγμα, έστω ότι υπάρχουν δυο εντολές στο πρόγραμμα. Η πρώτη εντολή εμφανίζει ένα τετράγωνο και η δεύτερη ένα τρίγωνο. Με την εκτέλεση του κώδικα, αντί να εμφανίζονται και τα δυο σχήματα, εμφανίζεται μόνο το τετράγωνο στο παράθυρο απεικόνισης.

Λύση προβλήματος 3.3.3

Το παραπάνω πρόβλημα προκύπτει διότι όταν η βιβλιοθήκη glut της OpenGL διαπιστώνει ότι το πλάνο για το παράθυρο απεικόνισης πρέπει να ξανασχεδιαστεί.

Για τον λόγο αυτό, προστέθηκε η εντολή της OpenGL ***glutDisplayFunc***. Η εντολή ***glutDisplayFunc(void func())*** εντάσσεται σε μια ειδική κατηγορία συναρτήσεων της βιβλιοθήκης GLUT, οι οποίες αποκαλούνται συναρτήσεις κλήσης (callback functions). Η συγκεκριμένη εντολή ***glutDisplayFunc*** παίρνει ως όρισμα τη συνάρτηση απεικόνισης που δημιουργεί ο χρήστης έστω ***display()***, η οποία θα εκτελείται όποτε διαπιστώνεται ότι υπάρχει απαίτηση σχεδίασμού/επανασχεδιασμού της σκηνής. Η σύνταξη της εντολής είναι ***void glutDisplayFunc(void display())***.

Ουσιαστικά, όλες οι ρουτίνες σχεδίασης συνήθως τοποθετούνται στο εσωτερικό της συνάρτησης ***display***. Η συνάρτηση κλήσης ***display*** πρέπει να έχει συγκεκριμένη δομή. Δεν επιστρέφει τιμή και δε δέχεται ορίσματα.

Η ***display*** θα εκτελείται στις περιπτώσεις μετακίνησης του παραθύρου σχεδίασης, της επαναφοράς του στο προσκήνιο.

Τυπικό παράδειγμα προγράμματος OpenGL για την χρήση της εντολής *glutDisplayFunc*

```
#include <glut.h>  
void display()  
{  
    glClearColor(1,1,1,1); // καθορίζει το χρώμα που χρησιμοποιείται κάθε φορά που εκτελείται εντολή καθαρισμού της οθόνης  
    glClear(GL_COLOR_BUFFER_BIT); // «καθαρισμός» οθόνης  
    glBegin(GL_LINES); // δηλώνει την έναρξη ορισμού ενός ή περισσοτέρων γεωμετρικών σχημάτων.  
    glColor3f(1,0,0); // ορίζουμε το τρέχον χρώμα σχεδιάσης  
    glVertex2i(20,20); // ορίζει σημεία στο δισδιάστατο χώρο.  
    glVertex2i(40,40); // ορίζει σημεία στο διδιάστατο χώρο  
    glEnd(); // ορίζει τη λήξη της επιλεγόμενης σχεδίασης  
    glFlush(); // εμφανίζει το αποτέλεσμα στην οθόνη  
}  
int main(int argc, char** argv)  
{  
    glutInit(&argc,argv); // αρχικοποίηση της εργαλειοθήκης
```

```

glutInitWindowPosition(50,50); // καθορίζει τη θέση στην οθόνη, στην οποία θα εμφανιστεί το παράθυρο
της εφαρμογής (συντεταγμένη της άνω αριστερής κορυφής).
glutInitWindowSize(640,480); // καθορίζει το πλάτος και ύψος του παραθύρου της εφαρμογής σε pixels
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB); // ρύθμιση της απεικόνισης του αντικειμένου στην
οθόνη
glutCreateWindow("A sample OpenGL application"); // εμφανίζει το παράθυρο της εφαρμογής στην
οθόνη και του αποδίδει έναν τίτλο
glMatrixMode(GL_PROJECTION);
gluOrtho2D(0,50,0,50); // περιοχή προβολής σε παγκόσμιες συντεταγμένες
glutDisplayFunc(display); // Η συνάρτηση αυτή εκτελείται κάθε φορά που η εφαρμογή διαπιστώσει ότι
απαιτείται επανασχεδιασμός της σκηνής
glutMainLoop(); // ενεργοποιεί τον κύκλο διαχείρισης γεγονότων
return 0;
}

```

Πρόβλημα 3.3.4

Για να είναι πιο εύχρηστη η ανάπτυξη κώδικα OpenGL από τον φοιτητή έγινε η προσπάθεια προσθήκης της λειτουργίας autocomplete. Η λειτουργία αυτή δίνει την δυνατότητα στο χρήστη όταν πληκτρολογεί τα πρώτα γράμματα μιας εντολής να εμφανίζεται μια λίστα με τις εντολές της OpenGL και να μπορεί να επιλέγει όποια εντολή επιθυμεί.

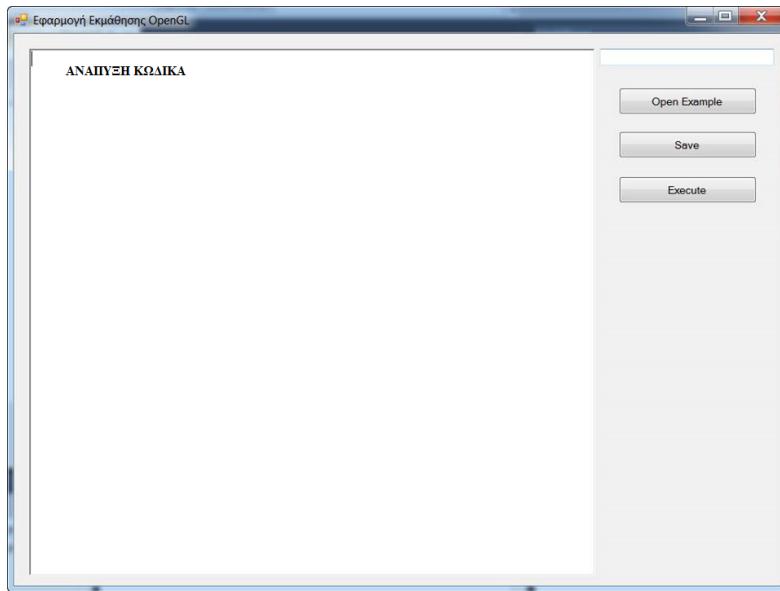
Κατά την πληκτρολόγηση των πρώτων γραμμάτων της εντολής, εμφανίζόταν η λίστα των υποψήφιων εντολών κανονικά. Όταν όμως επιλέγονταν κάποια από τις υπάρχουσες εντολές, η επιλεχθείσα εντολή δεν αντικαθιστούνταν από τα πληκτρολογημένα γράμματα αλλά συμπληρώνονταν δίπλα τους με αποτέλεσμα τη μη ορθότητα της εντολής.

Λύση προβλήματος 3.3.4

Δυστυχώς το πρόβλημα δεν μπόρεσε να λυθεί και η λειτουργία αφαιρέθηκε από την εφαρμογή.

Πρόβλημα 3.3.5

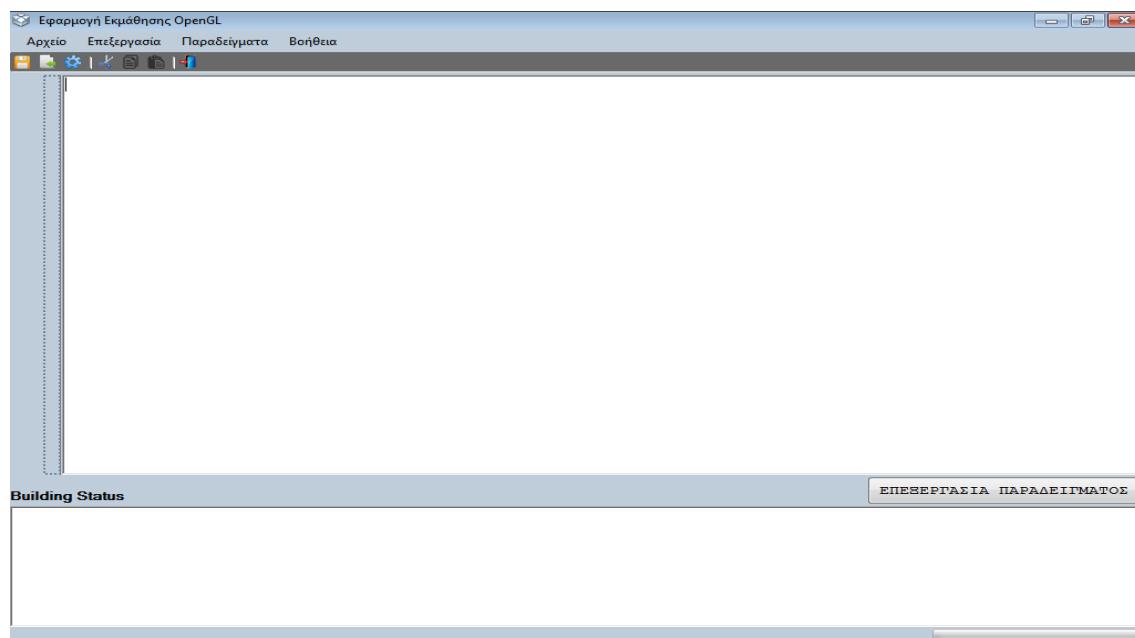
Αρχικά στην εφαρμογή δημιουργήθηκαν τα λειτουργικό κουμπιά «Open Example», «Save», «Import» και «Execute». Με το πάτημα Open Example, εμφανίζονταν ο φάκελος examples, που περιέχει τα βασικά παραδείγματα που υπάρχουν στην εφαρμογή και έτσι ο φοιτητής επιλέγει το παράδειγμα που χρειάζεται και το πρόσθετει στον κώδικα του. Με το πάτημα το κουμπιού Save θα αποθήκευταν τα δικά του παραδείγματα σε φάκελο της επιλογής του. Με το κουμπί Execute εκτελεί τον κώδικα και με το κουμπί Import θα πρόσθετε τα δικά του παραδείγματα στην εφαρμογή. Η εφαρμογή όμως φαινόταν αρκετά απλοϊκή και δεν βοηθούσε ιδιαίτερα στη λειτουργικότητα της, ως προς τον χρήστη. Η αρχική μορφή της εφαρμογής ήταν η εξής:



Σχήμα 3.1 Αρχική μορφή του interface της εφαρμογής

Λύση προβλήματος 3.3.5

Το interface της εφαρμογής υπέστη αρκετές αλλαγές. Τα τέσσερα λειτουργικά κουμπιά αντικαταστάθηκαν από ένα μενού και την γραμμή εργαλείων. Το κουμπί Open Example αντικαταστάθηκε από το μενού «Παραδείγματα», όπου υπάρχει κατηγοριοποίηση των βασικών παραδειγμάτων της εφαρμογής για πιο εύκολη αναζήτηση τους από τον χρήστη και ένας υποφάκελος για αποθήκευση τροποποιημένων παραδειγμάτων. Τα κουμπιά Save και Execute υπάρχουν και στο μενού «Αρχείο» ως λειτουργίες και υπάρχουν και στη γραμμή εργαλείων. Το κουμπί Import προστίθεται στη γραμμή με τα υπόλοιπα κουμπιά με την ονομασία «Εισαγωγή παραδείγματος». Επιπλέον προστέθηκαν τα λειτουργικά κουμπιά «Αποκοπή», «Αντιγραφή» και «Επικόλληση». Η τελική μορφή της εφαρμογής φαίνεται στην παρακάτω εικόνα:



Σχήμα 3.2: Τελική μορφή του interface της εφαρμογής

Πρόβλημα 3.3.6

Αρχική σκέψη ήταν ο φοιτητής να μην μπορεί να αποθηκεύσει τα ήδη υπάρχοντα παραδείγματα, ώστε να μπορεί να έχει πάντα πρόσβαση στα προαποθηκευμένα παραδείγματα της εφαρμογής. Όμως εφόσον ο φοιτητής θα μπορεί να έχει πρόσβαση στον συγκεκριμένο φάκελο μέσω του Windows Explorer, ουσιαστικά θα μπορεί να τα τροποποιήσει.

Λύση προβλήματος 3.3.6

Για τη λύση αυτού του προβλήματος αποφασίστηκε να δημιουργηθεί η επιπλέον λειτουργία «Επεξεργασία Παραδείγματος» ώστε ο χρήστης να έχει τη δυνατότητα να επεξεργαστεί σε μια νέα φόρμα τα παραδείγματα της εφαρμογής χωρίς να τροποποιηθούν τα βασικά παραδείγματα. Αυτό γίνεται διότι όταν ο χρήστης πάει να αποθηκεύσει το τροποποιημένο παράδειγμα, το παράδειγμα αποθηκεύεται σε έναν υποφάκελο που υπάρχει στα βασικά παραδείγματα (φάκελος Custom). Με αυτόν τον τρόπο θα υπάρχουν και τα βασικά παραδείγματα και δικά του τροποποιημένα παραδείγματα στην εφαρμογή.

Πρόβλημα 3.3.7

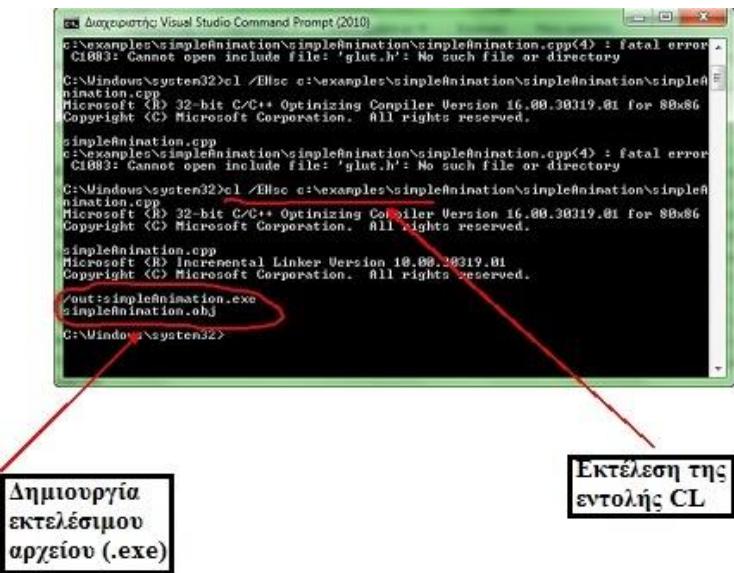
Αρχικά, για το compilation του κώδικα του χρήστη χρησιμοποιείται η κλάση CodeDomProvider, η οποία είναι υπεύθυνη κλάση για παραμετροποίηση του κώδικα του χρήστη. Τυπικά, χτίζει έναν μεταγλωττιστή και κάνει code compilation επιβεβαιώνοντας ότι ο κώδικας της εφαρμογής που θα ελεγχθεί είναι συντακτικά ορθός. Το θέμα όμως ήταν ότι η συγκεκριμένη κλάση δεν ‘δούλεψε’ στην παρούσα εφαρμογή, διότι κάνει compilation μόνο τα προγράμματα που είναι γραμμένα στην γλώσσα προγραμματισμού C#, ενώ εδώ ο κώδικας του χρήστη αναπτύσσεται σε C++.

Λύση προβλήματος 3.3.7

Το Microsoft Visual Studio περιλαμβάνει ένα utility, τον C/C++ compiler ο οποίος χρησιμοποιείται για να κάνει compilation σε κώδικα C++ (και να τρέχει τα cpp αρχεία (κώδικας του χρήστη). Ουσιαστικά ο συγκεκριμένος compiler δημιουργεί αρχεία τύπου.obj. Ο συγκεκριμένος compiler συντάσσεται ως εξής: cl \ EHsc path αρχείου.

Πρόβλημα 3.3.8

Παρατηρήθηκε στα παραδείγματα ότι όταν εκτελούνται εξωτερικά, δείχνει να δημιουργείται το αρχείο.exe π.χ. του παραδείγματος simple Animation όπως φαίνεται από την εκτέλεση του compiler του Visual Studio:



Δημιουργία
εκτελέσιμου
αρχείου (.exe)

Εκτέλεση της
εντολής CL

Σχήμα 3.3: Διαδικασία δημιουργίας εκτελέσιμου αρχείου εξωτερικά της εφαρμογής

Το θέμα όμως που δημιουργείται είναι ότι παρόλο που στον compiler φαίνεται ότι το εκτελέσιμο αρχείο έχει δημιουργηθεί, όταν ανοιχτεί ο φάκελος με το παράδειγμα που εκτελέστηκε, το αρχείο.exe δεν εμφανίζεται μέσα στο φάκελο. Αυτό συνεπάγεται ότι δεν μπορεί να εμφανιστεί και το γράφημα στο παράθυρο απεικόνισης.

Λύση προβλήματος 3.3.8

Στην ουσία, έπρεπε να βρεθεί το κατάλληλο utility, ώστε ο κώδικας του χρήστη να μετατρέπεται σε γράφημα, μέσα στην εφαρμογή. Να δημιουργείται το εκτελέσιμο αρχείο και με την εκτέλεση του να εμφανίζει το αποτέλεσμα. Για τον λόγο αυτό χρησιμοποιείται και ένας ακόμη compiler του Visual Studio, ο MSBuilt compiler.

Η εφαρμογή αντιλαμβάνεται τα παραδείγματα ως project (αρχεία.sln) και όχι σαν απλά αρχεία (.cpp). Το MSBuilt αποτελεί έναν ενσωματωμένο compiler του Visual Studio και χρησιμοποιείται γενικά για τη δόμηση ενός project.

Επειδή ο κώδικας του χρήστη και των βασικών παραδειγμάτων της εφαρμογής θα βασίζεται σε βιβλιοθήκες της γλώσσας προγραμματισμού C++, το project που θα δημιουργείται με την εκτέλεση του MSBuilt compiler θα έχει την κατάληξη.vcxproj.

```

C:\Windows\system32\cmd.exe
C:\Windows\Microsoft.NET\Framework64\v4.0.30319>msbuild c:\examples\simplePoint\simplePoint.vcxproj
Microsoft (R) Build Engine Έκδοση 4.0.30319.1
[Microsoft .NET Framework, Έκδοση 4.0.30319.1]
Πινευματικό δικαιώματα <C> Microsoft Corporation 2007. Με την επιφύλαξη και με δικαιώματος.

Έωρη δόμησης 10/5/2014 11:20:43 μμ.
Έργο "c:\examples\simplePoint\simplePoint\simplePoint.vcxproj" στου κόμβου οεπιλεγμένον προορισμός.
InitializeBuildStatus:
  Μηνιουργείται το "Debug\simplePoint.unsuccessfulbuild", επειδή καθορίστηκε "AlwaysCreate".
CLCompile:
  Ήδει από έξοδοι είναι ευημερωμένες.
Link:
  C:\Program Files (<x86>)\Microsoft Visual Studio 10.0\VC\bin\link.exe /E
  ORT:QUEUE /OUT:"Debug\simplePoint.exe" /NOLOGO /LIBPATH:C:\examples\GL
  .lib kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi
  _shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbcsp32.lib /
  T /ManifestFile:"Debug\simplePoint.exe.intermediate.manifest" /MANIFESTU
  EVEL='asInvoker' UIACCESS='false' /DEBUG /PDB:"c:\examples\simplePoint\sim
  plePoint.lib" /MACHINE:X86 Debug\simplePoint.obj
  simplePoint.vcxproj -> c:\examples\simplePoint\Debug\simp
  lePoint.exe
Manifest:
  C:\Program Files (<x86>)\Microsoft SDKs\Windows\v7.0A\bin\mt.exe /nologo
  /outputresource:"Debug\simplePoint.exe;#1" /manifest Debug\simplePo
  int\simplePoint.manifest
FinalizeBuildStatus:
  Μηνιρροφή του ρεξειού "Debug\simplePoint.unsuccessfulbuild".
  Επωρή με το "Debug\simplePoint.lastbuildstate".
Η δόμηση του έργου "c:\examples\simplePoint\simplePoint.vcxp
οκληρώθηκε <προειπλεγμένον προορισμό>".

Η δόμηση ολοκληρώθηκε με επιτυχία.
  Θ προειδοποιήστες
  Θ σφάλματα
Παρερχόμενος χρόνος 00:00:00.86
C:\Windows\Microsoft.NET\Framework64\v4.0.30319>

```

Σχήμα 3.4. Διαδικασία εκτέλεσης του MSBuilt compiler ('κτίσμα' project και δημιουργία εκτελέσιμου αρχείου).

Επιπλέον, παρατηρώντας την ανάλυση του MSBBuilt compiler φαίνεται και η χρήση του cl compiler, δίνεται η δυνατότητα να ανιχνευθούν τα λάθη που έγιναν κατά τη δημιουργία του cl compiler, όσον αφορά τις παραμέτρους που χρησιμοποιούνται από το Visual Studio για τον σωστό συντακτικό έλεγχο (compilation).

Πρόβλημα 3.3.9

Θα πρέπει να ληφθεί υπόψη η αυτόματη προσθήκη των βιβλιοθηκών της OpenGl τόσο στα παραδείγματα της εφαρμογής, όσο και στον κώδικα του χρήστη. Σε περίπτωση που κάποιος υπολογιστής δεν έχει εγκατεστημένο το Microsoft Visual Studio, δεν θα μπορούν να ενσωματωθούν οι απαραίτητες βιβλιοθήκες της OpenGl αυτόματα. Έτσι ο χρήστης όταν εκτελεί τον κώδικα του θα έχει ως αποτέλεσμα τη μη σωστή λειτουργικότητα του κώδικα του.

Λύση προβλήματος 3.3.9

Το θέμα αυτό λύθηκε με την προσθήκη του φακέλου GL στον φάκελο με τα βασικά παραδείγματα της εφαρμογής. Ο φάκελος αυτός περιέχει το σύνολο των βασικών βιβλιοθηκών της OpenGl (glut.h, glu.h, gl.h, glut32.lib κ.ο.κ).

Στην ουσία, το glut32.dll είναι μια βιβλιοθήκη την οποία χρειάζεται το εκτελέσιμο αρχείο για να βρει κάποιες λειτουργίες της OpenGl.

Πρόβλημα 3.3.10

Η εφαρμογή εγκαθιστάται κανονικά σε λειτουργικό σύστημα Windows Home Premium αλλά όταν επιχειρεί ο χρήστης να εγκαθιστήσει τα παραδείγματα εμφανίζεται error και δεν επιτρέπει την εγκατάσταση τους και συνεπώς την εκτέλεση των κώδικα των παραδειγμάτων.

Το πρόβλημα αυτό οφείλεται στη χρήση του.net framework στο λειτουργικό σύστημα Windows Home Premium το οποίο είναι απαραίτητο για την εκτέλεση των παραδειγμάτων. Ο MSBUILD compiler χρησιμοποιεί μια έκδοση του.net framework το οποίο στα home premium δεν λειτουργεί σωστά.

Λύση προβλήματος 3.3.10

Δυστυχώς μετά από αρκετές προσπάθειες να βρεθεί λύση κατέστει αδύνατο. Η εγκατάσταση των παραδειγμάτων αφαιρέθηκε από το πρόγραμμα. Δυστυχώς όμως το πρόβλημα συνεχίζει να υπάρχει και η εφαρμογή δεν λειτουργεί σε λειτουργικό σύστημα Windows Home Premium.

3.4 Απαιτήσεις και Προδιαγραφές της εφαρμογής

Απαίτηση 3.4.1

Τίτλος

Προσθήκη εκπαιδευτικού υλικού.

Περιγραφή απαίτησης

Δυνατότητα προσθήκης εκπαιδευτικού υλικού - παραδειγμάτων τόσο από τον καθηγητή όσο και από τον φοιτητή σε ξεχωριστό φάκελο από τα υπόλοιπα παραδείγματα και εμφάνισης αυτού του φακέλου στο μενού με τα υπόλοιπα παραδείγματα. Με αυτό τον τρόπο ο χρήστης έχει μεγαλύτερη διευκόλυνση να βρει τις εργασίες του και να τις επεξεργαστεί.

Αιτιολόγηση

Με την προσθήκη επιπλέον παραδειγμάτων και εκπαιδευτικού υλικού η εφαρμογή θα μπορεί να αναπτυχτεί και να εξελιχτεί περαιτέρω.

Απαίτηση 3.4.2

Τίτλος

Δημιουργία νέου παραδείγματος στην εφαρμογή.

Περιγραφή απαίτησης

Με το πάτημα του κουμπιού 'Εισαγωγή παραδείγματος' εμφανίζεται ένα νέο παράθυρο για να ονομάσει ο χρήστης τη νέα του εργασία και στη συνέχεια υπάρχει το κουμπί 'Δημιουργία' για να αποθηκευθεί το παράδειγμα στο φάκελο custom και να εμφανιστεί και στο μενού με τα παραδείγματα στο αντίστοιχο υπομενού.

Αιτιολόγηση

Βασικός σκοπός είναι να μην τροποποιηθούν ή χαθούν τα παραδείγματα που είναι ενσωματωμένα στην εφαρμογή από τις εργασίες των χρηστών.

Απαίτηση 3.4.3

Τίτλος

Υλοποίηση και ανάπτυξη της εφαρμογής

Περιγραφή απαίτησης

Η πλατφόρμα (interface) της εφαρμογής υλοποιείται σε γλώσσα προγραμματισμού C#. Ο κώδικας που θα αναπτύσσεται και θα εκτελείται για την OpenGL θα τρέχει με τις εντολές της γλώσσας προγραμματισμού C++ και η δημιουργία της εφαρμογής του εικονικού εργαστηρίου με το Microsoft Visual Studio 2010.

Αιτιολόγηση

Η γλώσσα προγραμματισμού C# έχει τη δυνατότητα για code compilation, διαχειρίζεται εύκολα τα εκτελέσιμα αρχεία και τις μεταβλητές τύπου string. Γενικά η C#, διαθέτει έτοιμες βιβλιοθήκες και είναι πιο εύχρηστη.

Η γλώσσα προγραμματισμού που θα εκτελεί κώδικα OpenGL ο χρήστης θα είναι η C++. Ενσωματώνει τις απαραίτητες βιβλιοθήκες για την εκτέλεση κώδικα σε OpenGL και δεν χρειάζεται η προσθήκη εξωτερικών βιβλιοθηκών.

Απαίτηση 3.4.4

Τίτλος

Δημιουργία του λειτουργικού κουμπιού «Αποθήκευση».

Περιγραφή απαίτησης

Το συγκεκριμένο κουμπί αποθηκεύει τα προγράμματα που υλοποιούν οι χρήστες στον φάκελο που επιλέγει ο καθένας.

Αιτιολόγηση

Η ύπαρξη και η λειτουργία αυτού του κουμπιού είναι σημαντική, διότι ο χρήστης αποθηκεύει τα προγράμματα του στον υπολογιστή σταδιακά ώστε να μην χάσει την εξέλιξη της εργασίας του και χρειαστεί να την αναπτύξει εκ νέου (π.χ από διακοπή ρεύματος). Επιπλέον, θα μπορεί να παρακολουθεί τα παραδείγματα του και να τα τροποποιεί ανάλογα με τις ανάγκες του.

Απαίτηση 3.4.5

Τίτλος

Δημιουργία του λειτουργικού κουμπιού «Εκτέλεση».

Περιγραφή απαίτησης

Το συγκεκριμένο κουμπί ‘τρέχει’ τον κώδικα που αναπτύσσει ο χρήστης στην εφαρμογή.

Αιτιολόγηση

Αποτελεί από τα πιο σημαντικά λειτουργικά κουμπιά της εφαρμογής, διότι κάνει compilation τον κώδικα που αναπτύσσει ο χρήστης και ελέγχει την ορθότητά του, δημιουργώντας το εκτελέσιμο αρχείο και εμφανίζοντας τελικά το γράφημα.

Απαίτηση 3.4.6

Τίτλος

Δημιουργία του λειτουργικού κουμπιού «Εισαγωγή Παραδείγματος».

Περιγραφή απαίτησης

Το συγκεκριμένο κουμπί επιτρέπει στο χρήστη να προσθέτει δικά του παραδείγματα στην εφαρμογή.

Αιτιολόγηση

Έκτος από τα βασικά παραδείγματα της εφαρμογής, ο χρήστης μπορεί να προσθέτει δικά του παραδείγματα εξίσου σημαντικά και λειτουργικά. Η τεχνολογία των γραφικών αναπτύσσεται ραγδαία και με αυτόν τον τρόπο δίνεται η δυνατότητα προσθήκης πιο εξελιγμένων προγραμμάτων στην εφαρμογή.

Απαίτηση 3.4.7

Τίτλος

Δημιουργία του λειτουργικού κουμπιού «Έξοδος».

Περιγραφή απαίτησης

Το συγκεκριμένο κουμπί επιτρέπει την έξοδο από την εφαρμογή.

Αιτιολόγηση

Το κουμπί αυτό διευκολύνει τον χρήστη και με ένα απλό πάτημα αυτού του κουμπιού κλείνει την εφαρμογή.

Απαίτηση 3.4.8

Τίτλος

Δημιουργία ειδικού πλαισίου κειμένου (Building Status).

Περιγραφή απαίτησης

Στην εφαρμογή, θα υπάρχει ένα ειδικό πλαίσιο κειμένου (Building Status) στο οποίο θα εμφανίζονται τα λάθη που θα εντοπίζει ο compiler. Επίσης εμφανίζεται και ο αριθμός της γραμμής όπου βρίσκεται το λάθος του κώδικα. Επιπλέον ο χρήστης θα μπορεί να βλέπει την διαδικασία δημιουργίας και εκτέλεσης του γραφημάτος του.

Αιτιολόγηση

Το ειδικό πλαίσιο κειμένου δημιουργείται για να μπορεί ο χρήστης με την εκτέλεση του προγράμματος του, να ενημερώνεται για τυχόν λάθη και σφάλματα που περιέχει ο κώδικας του. Έτσι θα μπορεί να τα εντοπίζει και να κάνει τις απαραίτητες διορθώσεις του.

Απαίτηση 3.4.9

Τίτλος

Εμφάνιση παραθύρου απεικόνισης του γραφήματος.

Περιγραφή απαίτησης

Όταν θα εκτελείται ο κώδικας που αναπτύσσει ο χρήστης στην εφαρμογή, το αποτέλεσμα της εκτέλεσης αυτής, εφόσον ο κώδικας δεν περιέχει λάθη, θα εμφανίζεται σε ένα νέο παράθυρο που θα απεικονίζει το γράφημα.

Αιτιολόγηση

Ο χρήστης θα είναι σε θέση να κατανοήσει την αντιστοιχία μεταξύ των εντολών που χρησιμοποιεί και πως εμφανίζονται αυτές οι εντολές σε μορφή γραφήματος.

Απαίτηση 3.4.10

Τίτλος

Δημιουργία της εφαρμογής ως εκτελέσιμο αρχείο.

Περιγραφή απαίτησης

Η εφαρμογή θα έχει τη μορφή εκτελέσιμου αρχείου (set up), ώστε να μπορεί να εγκατασταθεί σε οποιονδήποτε υπολογιστή.

Αιτιολόγηση

Η εφαρμογή με τη μορφή εκτελέσιμου αρχείου αποτελεί σημαντική λύση. Οποιοσδήποτε χρήστης επιθυμεί να χρησιμοποιήσει την εφαρμογή, δεν θα είναι απαραίτητο να έχει εγκατεστημένο το Microsoft Visual Studio στον υπολογιστή του. Θα έχει τη δυνατότητα με μια απλή εγκατάσταση της εφαρμογής να έχει πρόσβαση σε αυτήν.

Απαίτηση 3.4.11

Τίτλος

Δημιουργία μενού στην εφαρμογή.

Περιγραφή απαίτησης

Στην εφαρμογή θα υπάρχει ένα μενού (Αρχείο – Παραδείγματα-Επεξεργασία-Βοήθεια) για να έχει πρόσβαση ο χρήστης στις λειτουργίες που θέλει να πραγματοποιήσει (Αποθήκευση, Εκτέλεση κ.λ.π), και επίσης τα παραδείγματα της εφαρμογής θα εμφανίζονται ανά κατηγορία (2D, 3D, Animation κ.λ.π). Ακόμη θα υπάρχει το μενού Επεξεργασία για να μπορεί να διαλέγει ανάμεσα σε λειτουργίες όπως αποκοπή, αντιγραφή, επικόλληση και νέα σελίδα, επίσης θα μπορεί να απευθύνεται στο μενού Βοήθεια για οποιαδήποτε βοήθεια για τα παραδείγματα και την εφαρμογή.

Αιτιολόγηση

Το μενού προστέθηκε στη εφαρμογή για ευκολότερη χρήση της εφαρμογής και των λειτουργιών της από το χρήστη.

Απαίτηση 3.4.12

Τίτλος

Λειτουργία μορφοποιήσης κώδικα.

Περιγραφή απαίτησης

Καθώς ο χρήστης ‘φορτώνει’ ένα από τα προαποθηκευμένα παραδείγματα της εφαρμογής μια συνάρτηση στον κώδικα διαμορφώνει τον κώδικα του παραδείγματος ώστε να είναι πιο ευανάγνωστος και εμφανίζει κάποιες λέξεις-κλειδιά της γλώσσας προγραμματισμού C++ χρωματισμένες. Το ίδιο γίνεται και όταν γράφεται νέος κώδικας.

Αιτιολόγηση

Η λειτουργία αυτή προστέθηκε για να διευκολύνει τον χρήστη στην ανάγνωση και κατανόηση του κώδικα καθώς και στην εγγραφή δικού του κώδικα.

Απαίτηση 3.4.13

Τίτλος

Δημιουργία των λειτουργικών κουμπιών «Αντιγραφή», «Αποκοπή», «Επικόλληση» .

Περιγραφή απαίτησης

Τα συγκεκριμένα κουμπιά αντιγράφουν, αποκόπτουν και κάνουν επικόλληση κάποιο συγκεκριμένο κείμενο ή κομμάτι κώδικα που θα επιλέξει ο χρήστης από την εφαρμογή.

Αιτιολόγηση

Η ύπαρξη και η λειτουργία αυτών των κουμπιών είναι σημαντική, βοηθάει τον χρήστη να επεξεργαστεί πιο εύκολα και γρήγορα είτε τα έτοιμα παραδείγματα είτε ένα νέο δικό του.

Απαίτηση 3.4.14

Τίτλος

Δημιουργία του λειτουργικού κουμπιού «Νέα σελίδα».

Περιγραφή απαίτησης

Το συγκεκριμένο κουμπί δημιουργεί μια νέα σελίδα στην εφαρμογή.

Αιτιολόγηση

Η ύπαρξη του συγκεκριμένου κουμπιού οφείλεται στην ανάγκη του χρήστη να ανοίξει μια νέα σελίδα παράλληλα με κάποιο έτοιμο παραδειγμα ώστε να μπορεί να βοηθηθεί να δημιουργήσει ένα νέο δικό του παράδειγμα. Γενικά είναι μεγάλη διευκόλυνση να υπάρχει η δυνατότητα ο χρήστης να παρακολουθεί και να δημιουργεί παράλληλες εργασίες.

3.4.1 Προδιαγραφές της εφαρμογής

Προδιαγραφή 3.4.1.1

Υπηρεσία - Λειτουργία	Δυνατότητα προσθήκης εκπαιδευτικού υλικού – παραδειγμάτων.
Περιγραφή	Δυνατότητα προσθήκης εκπαιδευτικού υλικού και παραδειγμάτων τόσο από τον καθηγητή όσο και από τον φοιτητή.
Δεδομένα Εισόδου	Όνομα νέου παραδείγματος
Προέλευση	Κουμπί “Εισαγωγή Παραδείγματος”
Δεδομένα Εξόδου	Εμφάνιση νέου παράθυρου για ονομασία του νέου παραδείγματος
Προορισμός	Υποφάκελος custom στο φάκελο examples
Ενέργεια	Επιλογή ονομασίας νέου παραδείγματος και πάτημα κουμπιού «Δημιουργία» για αποθήκευση του παραδείγματος στο φάκελο custom και εμφάνιση του παραδείγματος στο μενού παραδείγματα και στο υπομενού custom.
Απαίτηση	Προσθήκη εκπαιδευτικού υλικού
Προϋπόθεση ή Προσυνθήκη	Να μην είναι από τα βασικά παραδείγματα της εφαρμογής
Αποτέλεσμα ή Μετασυνθήκη	Εμφάνιση παράθυρου για ονομασία αρχείου και αποθήκευσή του
Πλευρικά Φαινόμενα ή Παρενέργειες	Μη δυνατότητα εισαγωγής νέου παραδείγματος.

Προδιαγραφή 3.4.2.1

Υπηρεσία - Λειτουργία	Εμφάνιση νέου παραθύρου κατά την εισαγωγή παραδείγματος στην εφαρμογή για την ονομασία του νέου παραδείγματος.
Περιγραφή	Με το πάτημα του κουμπιού ‘Εισαγωγή παραδείγματος’ εμφανίζεται ένα νέο παράθυρο για να ονομάσει ο χρήστης τη νέα του εργασία και στη συνέχεια υπάρχει το κουμπί ‘Δημιουργία’ για να αποθηκευτεί το παράδειγμα στο φάκελο custom και να εμφανιστεί και στο μενού με τα παραδείγματα στο αντίστοιχο υπομενού.
Δεδομένα Εισόδου	Πάτημα κουμπιού ‘Εισαγωγή Παραδείγματος’
Προέλευση	Κώδικας του χρήστη
Δεδομένα Εξόδου	Εμφάνιση του νέου παραθύρου.
Προορισμός	Οθόνη
Ενέργεια	Εμφάνιση του νέου παραθύρου για δημιουργία νέου πραδείγματος
Απαίτηση	Δημιουργία νέου παραδείγματος στην εφαρμογή
Προϋπόθεση ή Προσυνθήκη	Πάτημα του κουμπιού
Αποτέλεσμα ή Μετασυνθήκη	Δημιουργία νέου παραδείγματος
Πλευρικά Φαινόμενα ή Παρενέργειες	Μη εμφάνιση του παραθύρου εισαγωγής παραδείγματος

Προδιαγραφή 3.4.2.2

Υπηρεσία - Λειτουργία	Απαγόρευση αποθήκευσης των προγραμμάτων του χρήστη στο φάκελο με τα βασικά παραδείγματα της εφαρμογής.
Περιγραφή	Οι φοιτητές θα αναπτύσσουν δικό τους κώδικα στην εφαρμογή ενώ παράλληλα θα μπορούν να ενσωματώσουν τα παραδείγματα της εφαρμογής στο κώδικα τους. Έπειτα με το κουμπί «Αποθήκευση», θα μπορούν να αποθηκεύσουν το αρχείο τους στο φάκελο όπου βρίσκονται τα βασικά παραδείγματα, με διαφορετική ονομασία.
Δεδομένα Εισόδου	Κουμπί «Αποθήκευση»
Προέλευση	Υπομενού «Προσθήκη Παραδείγματος»
Δεδομένα Εξόδου	Εμφάνιση παράθυρου για επιλογή τοποθεσίας αρχείου
Προορισμός	Φάκελος επιλογής του χρήστη
Ενέργεια	Εγγραφή κώδικα από το χρήστη και δυνατότητα προσθήκης τμημάτων κώδικα από τα βασικά παραδειγμάτα στο δικό τους κώδικα
Απαίτηση	Δημιουργία νέου παραδείγματος στην εφαρμογή
Προϋπόθεση ή Προσυνθήκη	Το όνομα του project να μην είναι το ίδιο με κάποιο όνομα των βασικών παραδειγμάτων
Αποτέλεσμα ή Μετασυνθήκη	Αποθήκευση του νέου project στον επιθυμητό φάκελο (εκτός του φακέλου των παραδειγμάτων)
Πλευρικά Φαινόμενα ή Παρενέργειες	Αποθήκευση με το ίδιο όνομα από τα βασικά παραδείγματα

Προδιαγραφή 3.4.3.1

Υπηρεσία - Λειτουργία	Επιλογή προγραμμάτων υλοποίησης και ανάπτυξης της εφαρμογής.
Περιγραφή	Η εφαρμογή θα υλοποιηθεί με το πρόγραμμα Microsoft Visual Studio 2010 και ο κώδικας της εφαρμογής θα αναπτυχθεί με την γλώσσα προγραμματισμού C#.
Δεδομένα Εισόδου	Κώδικας C#.
Προέλευση	Πληκτρολόγιο, ποντίκι
Δεδομένα Εξόδου	Εμφάνιση περιβάλλοντος Microsoft Visual Studio 2010 για τη δημιουργία της εφαρμογής
Προορισμός	Compiler, Οθόνη
Ενέργεια	Δημιουργία του interface της εφαρμογής
Απαίτηση	Υλοποίηση και ανάπτυξη της εφαρμογής
Προϋπόθεση ή Προσυνθήκη	Εγκατάσταση Microsoft Visual Studio 2010 για δημιουργία της εφαρμογής
Αποτέλεσμα ή Μετασυνθήκη	Δημιουργία της εφαρμογής
Πλευρικά Φαινόμενα ή Παρενέργειες	-

Προδιαγραφή 3.4.3.2

Υπηρεσία - Λειτουργία	Επιλογή προγραμμάτων υλοποίησης και ανάπτυξης του κώδικα του χρήστη.
Περιγραφή	Ο χρήστης θα αναπτύσσει τον κώδικα του αποκλειστικά με την OpenGL, και θα βασίζεται στις βιβλιοθήκες της γλώσσας προγραμματισμού C++
Δεδομένα Εισόδου	Κώδικας OpenGL, Κώδικας C++
Προέλευση	Πληκτρολόγιο, ποντίκι
Δεδομένα Εξόδου	Παράθυρο ανάπτυξης κώδικα στην εφαρμογή
Προορισμός	Πλαίσιο κειμένου, Οθόνη
Ενέργεια	Ανάπτυξη κώδικα από τον χρήστη
Απαίτηση	Υλοποίηση και ανάπτυξη της εφαρμογής
Προϋπόθεση ή Προσυνθήκη	Άνοιγμα της εφαρμογής για εγγραφή του κώδικα
Αποτέλεσμα ή Μετασυνθήκη	Εμφάνιση του αποτελέσματος της εκτέλεσης του κώδικα
Πλευρικά Φαινόμενα ή Παρενέργειες	Μη εμφάνιση του γραφήματος

Προδιαγραφή 3.4.4.1

Υπηρεσία - Λειτουργία	Αποθήκευση παραδείγματος από τον χρήστη
Περιγραφή	Το συγκεκριμένο κουμπί αποθηκεύει τα προγράμματα και τα project των χρηστών στον φάκελο των βασικών παραδειγμάτων της εφαρμογής στον υποφάκελο custom
Δεδομένα Εισόδου	Εισαγωγή ονόματος του παραδείγματος που πρόκειται να αποθηκευτεί
Προέλευση	Επιλογή κουμπιού αποθήκευσης
Δεδομένα Εξόδου	Εμφάνιση του ειδικού παραθύρου για επιλογή του φακέλου που θα αποθηκεύσει το παράδειγμά του
Προορισμός	Οθόνη, σκληρός δίσκος
Ενέργεια	Πάτημα κουμπιού και η αντίστοιχη λειτουργία του
Απαίτηση	Δημιουργία του λειτουργικού κουμπιού «Αποθήκευση».
Προϋπόθεση ή Προσυνθήκη	Λειτουργικότητα του κουμπιού
Αποτέλεσμα ή Μετασυνθήκη	Άνοιγμα του ειδικού παραθύρου για την επιλογή φακέλου αποθήκευσης του προγράμματος
Πλευρικά Φαινόμενα ή Παρενέργειες	Μη αποθήκευση της εξέλιξης του παραδείγματος

Προδιαγραφή 3.4.5.1

Υπηρεσία - Λειτουργία	Εκτέλεση είτε των βασικών παραδειγμάτων ή του παραδείγματος που αναπτύσσει ο χρήστης
Περιγραφή	Το συγκεκριμένο κουμπί θα εκτελεί τόσο τα βασικά παραδείγματα, όσο και τον κώδικα του χρήστη και θα δημιουργεί το εκτελέσιμο αρχείο (.exe).
Δεδομένα Εισόδου	Έλεγχος σφαλμάτων
Προέλευση	Επιλογή κουμπιού «Εκτέλεση»
Δεδομένα Εξόδου	Εκτέλεση των βασικών παραδειγμάτων και των προγραμμάτων που αναπτύσσουν οι χρήστες
Προορισμός	Παράθυρο απεικόνισης, Πλαίσιο λαθών
Ενέργεια	Πάτημα κουμπιού και η αντίστοιχη λειτουργία του
Απαίτηση	Δημιουργία του λειτουργικού κουμπιού «Εκτέλεση»
Προϋπόθεση ή Προσυνθήκη	Λειτουργικότητα του κουμπιού
Αποτέλεσμα ή Μετασυνθήκη	Εμφάνιση γραφήματος σε νέο παράθυρο απεικόνισης
Πλευρικά Φαινόμενα ή Παρενέργειες	Μη εμφάνιση γραφήματος

Προδιαγραφή 3.4.6.1

Υπηρεσία - Λειτουργία	Δυνατότητα στο χρήστη να προσθέτει δικά του παραδείγματα στην εφαρμογή.
Περιγραφή	Το συγκεκριμένο κουμπί θα επιτρέπει στο χρήστη να εισάγει δικά του παραδείγματα στην εφαρμογή, τα οποία θα αποθηκένονται σε έναν υποφάκελο και θα μπορεί να βλέπει το γράφημα.
Δεδομένα Εισόδου	Όνομα νέου παραδείγματος
Προέλευση	Επιλογή κουμπιού «Εισαγωγή Παραδείγματος»
Δεδομένα Εξόδου	Προσθήκη των παραδειγμάτων που αναπτύσσονται οι χρήστες
Προορισμός	Οθόνη, σκληρός δίσκος
Ενέργεια	Πάτημα κουμπιού και η αντίστοιχη λειτουργία του
Απαίτηση	Δημιουργία του λειτουργικού κουμπιού «Εισαγωγή Παραδείγματος»
Προϋπόθεση ή Προσυνθήκη	Λειτουργικότητα του κουμπιού
Αποτέλεσμα ή Μετασυνθήκη	Εμφάνιση των παραδειγμάτων των χρηστών στον υποφάκελο Custom
Πλευρικά Φαινόμενα ή Παρενέργειες	Μη εμφάνιση του παραθύρου εισαγωγής παραδείγματος και μη δυνατότητα δημιουργίας νέου παραδείγματος

Προδιαγραφή 3.4.7.1

Υπηρεσία - Λειτουργία	Επιτρέπει την έξοδο από την εφαρμογή
Περιγραφή	Το συγκεκριμένο κουμπί θα επιτρέπει στο χρήστη να κλείνει την εφαρμογή
Δεδομένα Εισόδου	-
Προέλευση	Επιλογή κουμπιού «Έξοδος»
Δεδομένα Εξόδου	Κλείσιμο της εφαρμογής
Προορισμός	Πλατφόρμα της εφαρμογής
Ενέργεια	Πάτημα κουμπιού και η αντίστοιχη λειτουργία του
Απαίτηση	Υπαρξη του λειτουργικού κουμπιού «Έξοδος»
Προϋπόθεση ή Προσυνθήκη	Λειτουργικότητα του κουμπιού
Αποτέλεσμα ή Μετασυνθήκη	Έξοδος από την εφαρμογή
Πλευρικά Φαινόμενα ή Παρενέργειες	-

Προδιαγραφή 3.4.8.1

Υπηρεσία - Λειτουργία	Εμφανίζει τη διαδικασία εκτέλεσης του κώδικα και τον αριθμό σφαλμάτων που υπάρχουν
Περιγραφή	Υπαρξη ειδικού πλαισίου κειμένου (Building Status) στην εφαρμογή τόσο για την εύρεση και την εμφάνιση των λαθών του κώδικα όσο και για την εμφάνιση της διαδικασίας της απεικόνισης του γραφήματος.
Δεδομένα Εισόδου	«Χτίσιμο παραδείγματος» και έλεγχος σφαλμάτων
Προέλευση	Επιλογή κουμπιού «Εκτέλεση»
Δεδομένα Εξόδου	Εμφάνιση σφαλμάτων του κώδικα
Προορισμός	Οθόνη
Ενέργεια	Επιλογή κουμπιού Εκτέλεση για τον συντακτικό και λογικό έλεγχο του κώδικα του χρήστη
Απαίτηση	Υπαρξη ειδικού πλαισίου κειμένου (Building Status).
Προϋπόθεση ή Προσυνθήκη	Επιλογή του κουμπιού Εκτέλεση
Αποτέλεσμα ή Μετασυνθήκη	Εμφάνιση των λαθών του κώδικα, καθώς επίσης και σε ποια γραμμή του κώδικα βρίσκονται τα λάθη αυτά
Πλευρικά Φαινόμενα ή Παρενέργειες	Μη εμφάνιση λαθών και μη δυνατότητα διόρθωσης του

Προδιαγραφή 3.4.9.1

Υπηρεσία - Λειτουργία	Εμφάνιση γραφήματος σε νέο παράθυρο
Περιγραφή	Εκτέλεση του κώδικα και εμφάνιση του αποτελέσματος σε ένα νέο παράθυρο απεικόνισης.
Δεδομένα Εισόδου	Εκτέλεση του κώδικα του χρήστη
Προέλευση	Κουμπί «Εκτέλεση»
Δεδομένα Εξόδου	Εμφάνιση του παράθυρου απεικόνισης
Προορισμός	Οθόνη, Σκληρός δίσκος
Ενέργεια	Επιλογή του κουμπιού «Εκτέλεση»
Απαίτηση	Εμφάνιση παραθύρου απεικόνισης του γραφήματος.
Προϋπόθεση ή Προσυνθήκη	Μη ύπαρξη λαθών στον κώδικα
Αποτέλεσμα ή Μετασυνθήκη	Εμφάνιση του παράθυρου απεικόνισης με το γράφημα
Πλευρικά Φαινόμενα ή Παρενέργειες	Σφάλματα στον κώδικα

Προδιαγραφή 3.4.10.1

Υπηρεσία - Λειτουργία	Δυνατότητα εγκατάστασης της εφαρμογής
Περιγραφή	Η εφαρμογή θα πρέπει να παραδοθεί ως αρχείο εγκατάστασης (set up), ώστε να μπορεί να εγκατασταθεί σε οποιονδήποτε υπολογιστή.
Δεδομένα Εισόδου	-
Προέλευση	Πλατφόρμα του Microsoft Visual Studio
Δεδομένα Εξόδου	Εμφάνιση της εγκατάστασης της εφαρμογής
Προορισμός	Φάκελος επιλογής μας, οθόνη
Ενέργεια	Αρχή διαδικασίας εγκατάστασης και επιλογή φακέλου αποθήκευσης και εγκατάστασης εφαρμογής
Απαίτηση	Δημιουργία της εφαρμογής ως εκτελέσιμο αρχείο
Προϋπόθεση ή Προσυνθήκη	Συμβατότητα με τα χαρακτηριστικά του υπολογιστή (λειτουργικό σύστημα, μνήμη, επεξεργαστής)
Αποτέλεσμα ή Μετασυνθήκη	Εγκατάσταση και άνοιγμα της εφαρμογής
Πλευρικά Φαινόμενα ή Παρενέργειες	Μη δυνατότητα χρήσης της εφαρμογής

Προδιαγραφή 3.4.11.1

Υπηρεσία - Λειτουργία	Το μενού χρησιμοποιείται για να κάνει την εφαρμογή εύχρηστη και κατανοητή για όλους τους χρήστες
Περιγραφή	Στην εφαρμογή θα υπάρχει ένα μενού (Αρχείο – Παραδείγματα) για να έχει πρόσβαση ο χρήστης στις λειτουργίες που θέλει να πραγματοποιήσει (Αποθήκευση, Εκτέλεση κ.λ.π), και επίσης τα παραδείγματα της εφαρμογής θα εμφανίζονται ανά κατηγορία (2D, 3D, Animation κ.λ.π).
Δεδομένα Εισόδου	Ποντίκι
Προέλευση	Εργαλειοθήκη του Microsoft Visual Studio
Δεδομένα Εξόδου	Εμφάνιση του μενού στην εφαρμογή
Προορισμός	Πάνω μέρος της εφαρμογής
Ενέργεια	Επιλογή λειτουργιών όπως αποθήκευση, εκτέλεση και επιλογή παραδειγμάτων
Απαίτηση	Δημιουργία μενού στην εφαρμογή
Προϋπόθεση ή Προσυνθήκη	Σωστή λειτουργία μενού
Αποτέλεσμα ή Μετασυνθήκη	Χρήση μενού
Πλευρικά Φαινόμενα ή Παρενέργειες	Μη δυνατότητα χρήσης μενού

Προδιαγραφή 3.4.12.1

Υπηρεσία - Λειτουργία	Κατά την εμφάνιση προαποθηκευμένου παραδείγματος ή εγγραφή νέου διαμορφώνεται ο κώδικας και χρωματίζονται κάποιες λέξεις κλειδιά.
Περιγραφή	Καθώς ο χρήστης ‘φορτώνει’ ένα από τα προαποθηκευμένα παραδείγματα της εφαρμογής μια συνάρτηση στον κώδικα διαμορφώνει τον κώδικα του παραδείγματος ώστε να είναι πιο ευανάγνωστος και εμφανίζει κάποιες λέξεις-κλειδιά της γλώσσας προγραμματισμού C++ χρωματισμένες. Το ίδιο γίνεται και όταν γράφεται νέος κώδικας.
Δεδομένα Εισόδου	Εμφάνιση προαποθηκευμένου παραδείγματος ή εγγραφή κώδικα
Προέλευση	Συνάρτηση κώδικα εφαρμογής
Δεδομένα Εξόδου	Διαμόρφωση κώδικα παραδειγμάτων και χρωματισμός λέξεις κλειδιών.
Προορισμός	Κώδικας παραδειγμάτων
Ενέργεια	Διαμόρφωσή κώδικα παραδειγμάτων
Απαίτηση	Λειτουργία μορφοποίησης κώδικα.
Προϋπόθεση ή Προσυνθήκη	Σωστή λειτουργία κώδικα
Αποτέλεσμα ή Μετασυνθήκη	Ευανάγνωστος κώδικας παραδειγμάτων
Πλευρικά Φαινόμενα ή Παρενέργειες	Μη μορφοποίηση του κώδικα

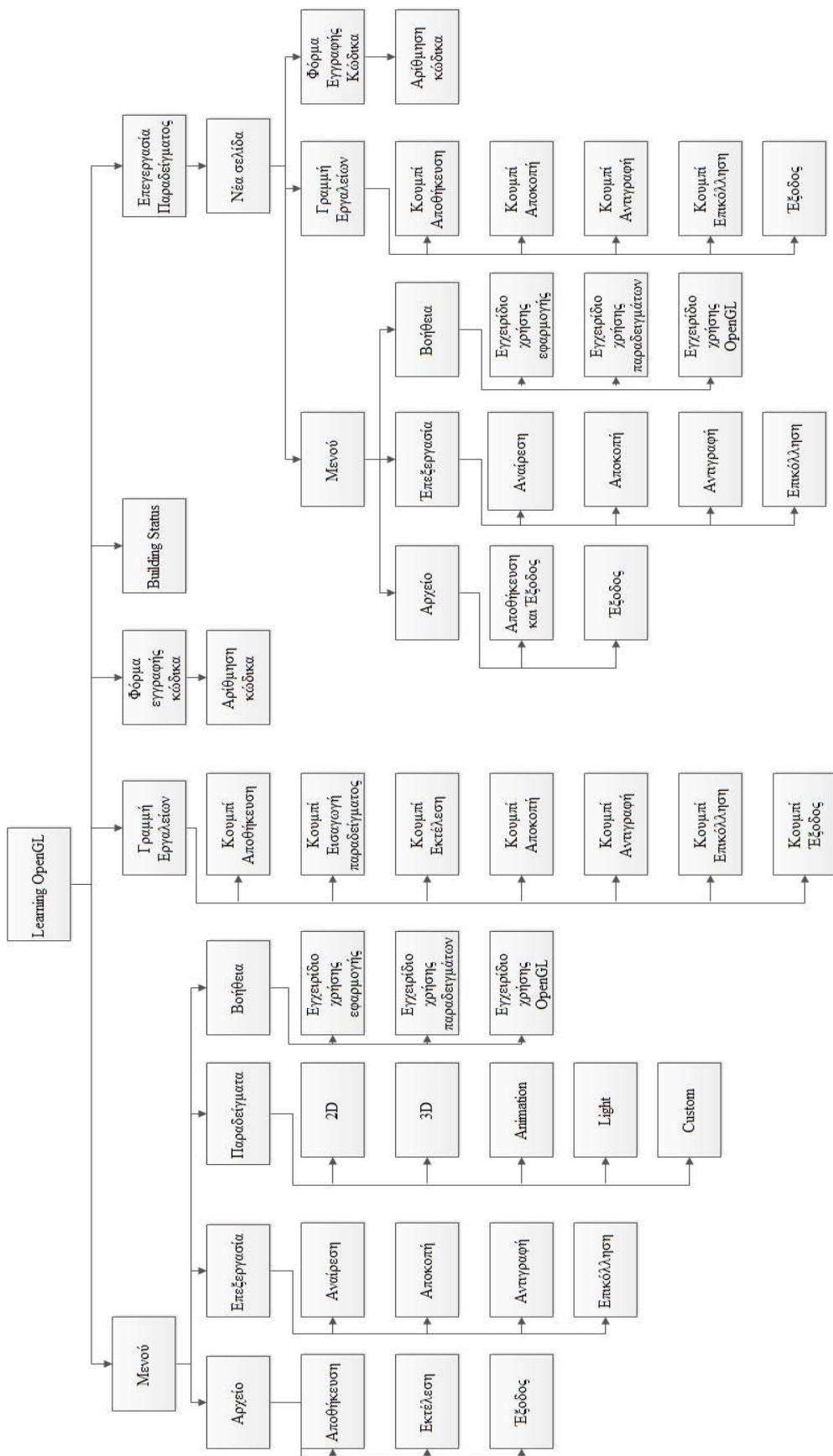
Προδιαγραφή 3.4.13.1

Υπηρεσία - Λειτουργία	Επεξεργασία κειμένου από τον χρήστη
Περιγραφή	Τα συγκεκριμένα κουμπιά αντιγράφουν, αποκόπτουν και επικολλούν κάποιο συγκεκριμένο κείμενο που θα επιλέξει ο χρήστης από την εφαρμογή, σε κάποιο άλλο σημείο που θα επιλέξει.
Δεδομένα Εισόδου	Το συγκεκριμένο κείμενο που έχει επιλεχθεί
Προέλευση	Επιλογή κουμπιού αποθήκευσης
Δεδομένα Εξόδου	Αντιγραφή, αποκοπή ή επικόλληση του αντίστοιχου επιλεγμένου κειμένου
Προορισμός	Οθόνη
Ενέργεια	Επιλογή κουμπιού και η αντίστοιχη λειτουργία του
Απαίτηση	Υπαρξή των λειτουργικών κουμπιών «Αντιγραφή», «Αποκοπή», «Επικόλληση».
Προϋπόθεση ή Προσυνθήκη	Λειτουργικότητα των κουμπιών
Αποτέλεσμα ή Μετασυνθήκη	Τα δεδομένα που επιλέχθηκαν έχουν αντιγραφεί, αποκοπεί ή επικολληθεί
Πλευρικά Φαινόμενα ή Παρενέργειες	-

Προδιαγραφή 3.4.14.1

Υπηρεσία - Λειτουργία	Δημιουργία νέας σελίδας
Περιγραφή	Το συγκεκριμένο κουμπί δημιουργεί μια νέα σελίδα στην εφαρμογή.
Δεδομένα Εισόδου	Εισαγωγή νέας σελίδας
Προέλευση	Επιλογή κουμπιού «Νέα σελίδα»
Δεδομένα	Νέα σελίδα
Εξόδου	
Προορισμός	Οθόνη
Ενέργεια	Επιλογή κουμπιού και η αντίστοιχη λειτουργία του
Απαίτηση	Υπαρξη του λειτουργικού κουμπιού «Νέα σελίδα».
Προϋπόθεση ή Προσυνθήκη	Λειτουργικότητα του κουμπιού
Αποτέλεσμα ή Μετασυνθήκη	Εμφάνιση νέας σελίδας
Πλευρικά Φαινόμενα ή Παρενέργειες	Μη εμφάνιση νέας σελίδας

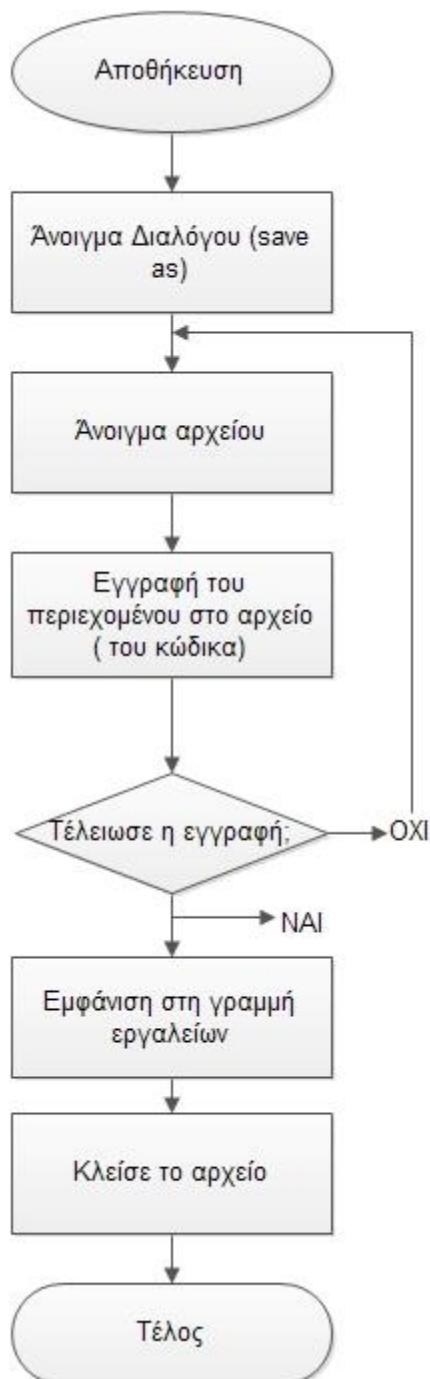
3.5 ΔΙΑΓΡΑΜΜΑ ΔΟΜΗΣ ΤΗΣ ΕΦΑΡΜΟΓΗΣ



Σχήμα 3.5 Διάγραμμα δομής

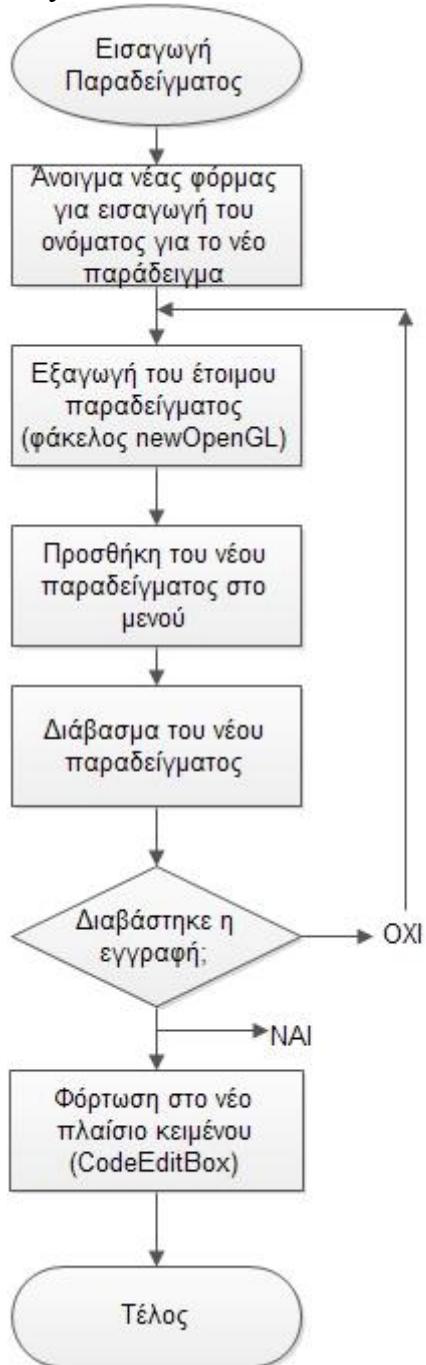
3.5.1 Διαγράμματα ροής

a) Αποθήκευση



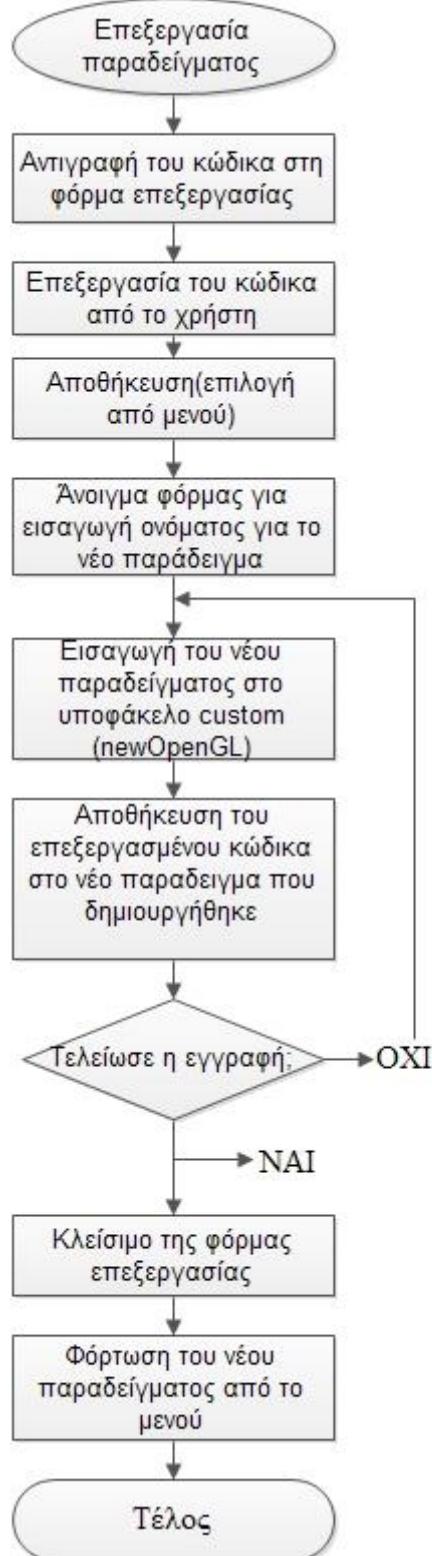
Σχήμα 3.6 Διάγραμμα ροής αποθήκευσης

β) Εισαγωγή παραδείγματος



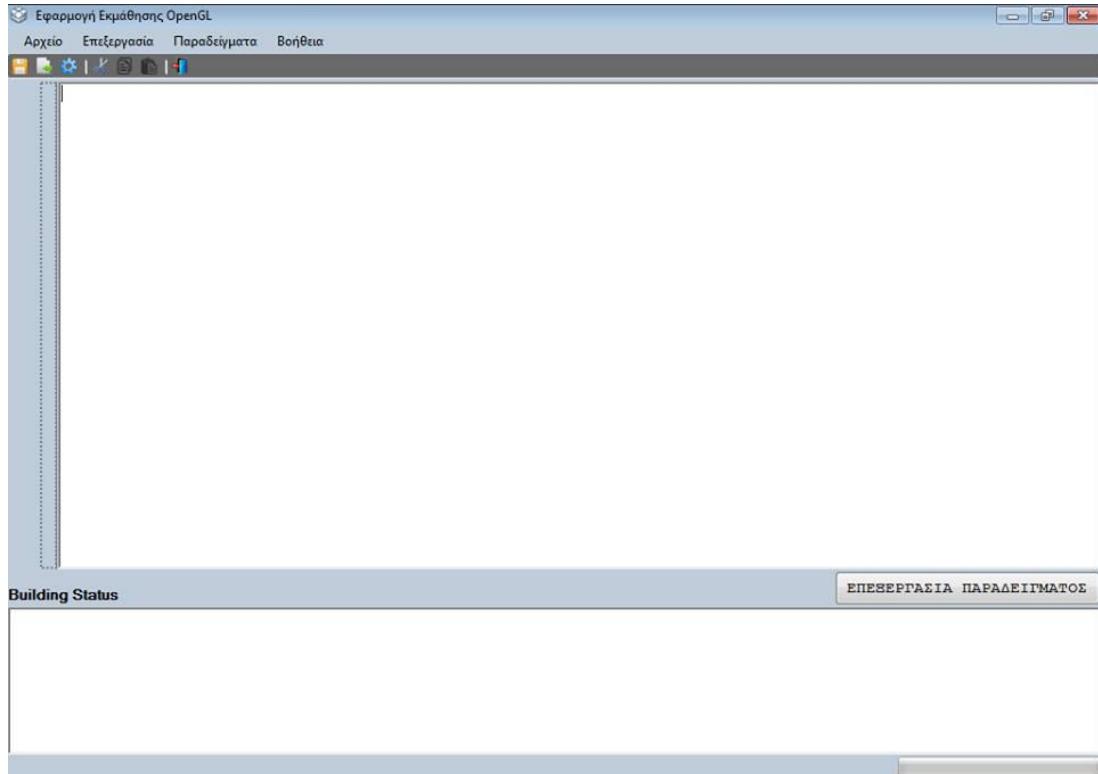
Σχήμα 3.7 Διάγραμμα ροής εισαγωγή παραδείγματος

γ) Επεξεργασία παραδείγματος

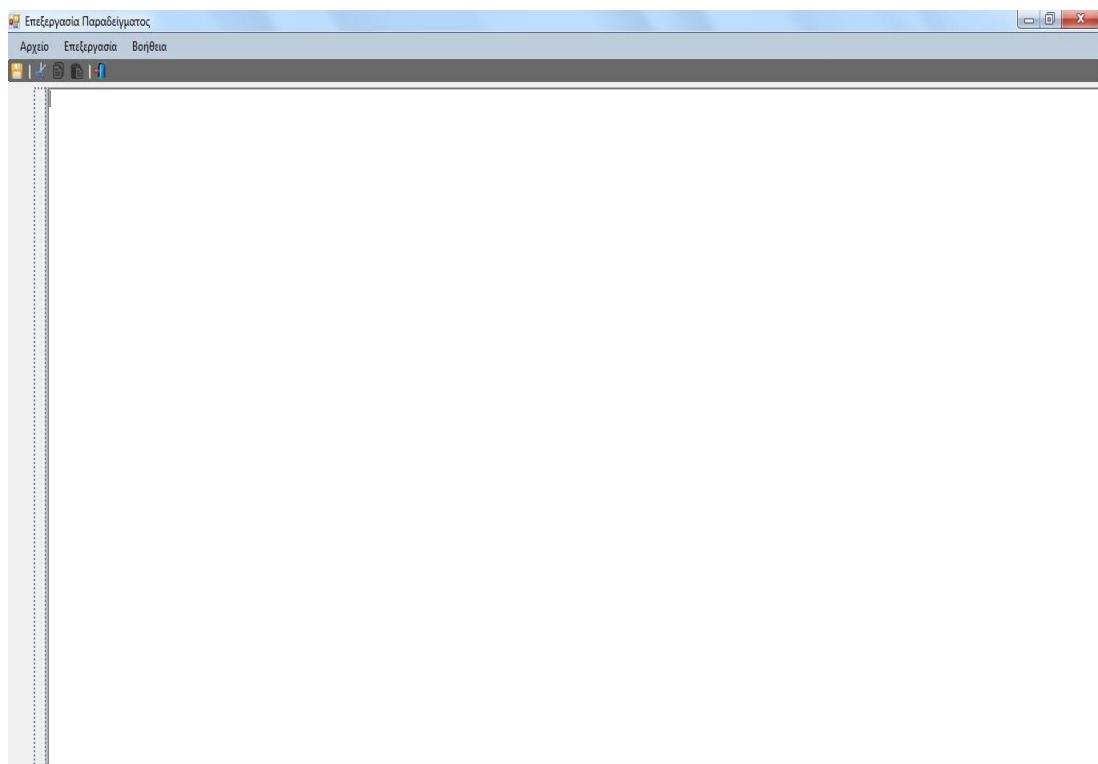


Σχήμα 3.8 Διάγραμμα ροής επεξεργασίας παραδείγματος

3.5.2 Εικόνες Εφαρμογής



Σχήμα 3.9 Γενική εικόνα της εφαρμογής



Σχήμα 3.10 Εικόνα νέας φόρμας επεξεργασίας παραδείγματος

3.6 Εγχειρίδιο της OpenGL

3.6.1 Βασικές αρχές σχεδίασης

3.6.2 Τύποι δεδομένων

Σε υλοποιήσεις της OpenGL σε C, οι τύποι δεδομένων που συναντά κανείς είναι όμοιοι με τους αντίστοιχους που ορίζονται στη γλώσσα C. Ο Πίνακας 1 παρουσιάζει τις αντιστοιχίες πρωτογενών τύπων δεδομένων που συναντά κανείς στην OpenGL με τους καθορισμένους στη γλώσσα C τύπους.

Πίνακας 3.1: Τύποι δεδομένων στην OpenGL

Τύπος της OpenGL	Τύπος δεδομένων	Αντίστοιχος τύπος δεδομένων στη C	Επίθημα
GLbyte	Ακέραιος 8 bits	signed char	b
GLshort	Ακέραιος 16 bits	Short	s
GLint / GLsizei	εκτεταμένος ακέραιος 32 bits	int / long	i
GLfloat / GLclampf	κινητής υποδιαστολής	Float	f
GLdouble / GLclampd	κινητής υποδιαστολής διπλής ακρίβειας (64 bits)	double	d
GLubyte / GLboolean	ακέραιος 8 bits χωρίς πρόσημο	unsigned char	ub
GLushort	ακέραιος 16 bits χωρίς πρόσημο	unsigned short	us
GLuint / GLenum / GLbitfield	εκτεταμένος ακέραιος 32 bits χωρίς πρόσημο	unsigned int / unsigned long	ui

3.6.3 Σχηματισμός εντολών

Στην OpenGL, οι εντολές ανάλογα με το περιεχόμενο τους αλλάζουν μορφή. Παρακάτω αναφέρονται οι τρόποι μετατροπής τους:
τον τύπο των ορισμάτων που δέχονται (π.χ. ακέραιοι ή πραγματικοί),
τις διαστάσεις του χώρου (π.χ. σχεδίαση σε δύο ή τρεις διαστάσεις)
τον αριθμό των συνιστωσών των χρωματικών τιμών (π.χ. τρεις στο μοντέλο RGB, τέσσερις στο μοντέλο RGBA με μίξη χρωμάτων)
τον τρόπο με τον οποίο επιλέγουμε να περάσουμε τις παραμέτρους στην εντολή (πέρασμα αριθμητικών τιμών (call by value) ή πέρασμα διανυσμάτων υπό τη μορφή μητρώων (call by reference)).

Η μορφή που έχουν συνήθως οι εντολές είναι: Όνομα εντολής + Διάσταση χώρου + Πρωτογενής τύπος δεδομένων ορισμάτων + Τρόπος κλήσεως ορισμάτων

Παράδειγμα:

Η εντολή για το παράδειγμα είναι η **glVertex**, η οποία δηλώνει τις συντεταγμένες των σημείων για την δημιουργία των γεωμετρικών σχημάτων.

glVertex2f(GLfloat x, GLfloat y);
εδώ η εντολή είναι η **glVertex**,
το 2 δηλώνει ότι η σχεδίαση θα γίνει σε δισδιάστατο χώρο,
το f δηλώνει ότι τα ορίσματα θα είναι σε μορφή float.

glVertex3i(GLint x, GLint y, GLint z);
εδώ η εντολή είναι η **glVertex**,
το 3 δηλώνει ότι η σχεδίαση θα γίνει σε τρισδιάστατο χώρο,
το i δηλώνει ότι τα ορίσματα θα είναι σε μορφή int.

glVertex3fv(const GLfloat *coordArray);
εδώ η εντολή είναι η **glVertex**,
το 3 δηλώνει ότι η σχεδίαση θα γίνει σε τρισδιάστατο χώρο,
το f δηλώνει ότι τα ορίσματα θα είναι σε μορφή float.
Το ν δηλώνει ότι οι παράμετροι περνιούνται με τη μορφή call by reference, δηλαδή εδώ
έχουμε δείκτη στο πίνακα coord

Ένα τυπικό πρόγραμμα με τις βασικές εντολές

```
#include <glut.h>
void display()
{
    glClearColor(1,1,1,1); // καθορίζει το χρώμα που χρησιμοποιείται κάθε φορά που εκτελείται εντολή
    καθαρισμού της οθόνης
    glClear(GL_COLOR_BUFFER_BIT); // «καθαρισμός» οθόνης
    glBegin(GL_LINES); // δηλώνει την έναρξη ορισμού ενός ή περισσοτέρων γεωμετρικών σχημάτων.
    glColor3f(1,0,0); // ορίζουμε το τρέχον χρώμα σχεδίασης
    glVertex2i(20,20); // ορίζει σημεία στο δισδιάστατο χώρο.
    glVertex2i(40,40); // ορίζει σημεία στο δισδιάστατο χώρο
    glEnd(); // ορίζει τη λήξη της επιλεγόμενης σχεδίασης
    glFlush(); // εμφανίζει το αποτέλεσμα στην οθόνη
}
int main(int argc, char** argv)
{
    glutInit(&argc,argv); // ενεργοποιεί την χρήση εντολών της βιβλιοθήκης glut.
    glutInitWindowPosition(50,50); // καθορίζει τη θέση στην οθόνη, στην οποία θα εμφανιστεί το παράθυρο
    της εφαρμογής (συντεταγμένη της άνω αριστερής κορυφής).
    glutInitWindowSize(640,480); // καθορίζει το πλάτος και ύψος του παραθύρου της εφαρμογής σε pixels
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB); // ρύθμιση της απεικόνισης του αντικειμένου στην
    οθόνη (μοντέλο ενταμιευτή, χρωματικό μοντέλο)
    glutCreateWindow("A sample OpenGL application"); // εμφανίζει το παράθυρο της εφαρμογής στην
    οθόνη και του αποδίδει έναν τίτλο
    glMatrixMode(GL_PROJECTION); // επιλογή μητρώου, επεξεργασία μητρώου προβολής.
    gluOrtho2D(0,50,0,50); // περιοχή προβολής σε παγκόσμιες συντεταγμένες, τέσσερις παράμετροι: left,
    right, bottom και top το αριστερό, δεξιό, κάτω και άνω όριο του παραθύρου αποκοπής. Τα όρια του
    παραθύρου αποκοπής ορίζονται στο σύστημα συντεταγμένων σκηνής.
    glutDisplayFunc(display); // Η συνάρτηση αυτή εκτελείται κάθε φορά που η εφαρμογή διαπιστώσει ότι
    απαιτείται επανασχεδιασμός της σκηνής
    glutMainLoop(); // ενεργοποιεί τον κύκλο διαχείρισης γεγονότων
    return 0;
}
```

3.6.4 ΙΔΙΟΤΗΤΕΣ

α) Ιδιότητες δύο καταστάσεων: Για να δηλώσουμε στην OpenGL την ενεργοποίηση ή την απενεργοποίηση ιδιοτήτων δύο καταστάσεων, αναθέτουμε σε αντίστοιχες προκαθορισμένες παραμέτρους της OpenGL τις τιμές GL_TRUE ή GL_FALSE αντίστοιχα. Η ενεργοποίηση ή απενεργοποίηση των λειτουργιών αυτών γίνεται με τις εντολές glEnable και glDisable.

Παράδειγμα: **glEnable(GL_LINE_STIPPLE)//** Ενεργοποίηση χρήσης διακεκομμένων γραμμών

Εάν πρέπει να ελεχθεί εάν μια ιδιότητα είναι ενεργοποιημένη ή απενεργοποιημένη είναι η εξής εντολή: **GLboolean glIsEnabled(GLenum capability)** ; Όπου στην παράμετρο μπαίνει η ιδιότητα που πρέπει να ελεχθεί. Και επιστρέφει GL_TRUE ή GL_FALSE, ανάλογα με το αν η ιδιότητα είναι ενεργοποιημένη ή όχι.

β) Σύνθετες ιδιότητες κατάστασης: Οι σύνθετες μεταβλητές κατάστασης παίρνουν πάνω από δύο μεταβλητές και έχει καθοριστεί ένα σύνολο εντολών επισκόπησης οι οποίες επιστρέφουν την τιμή ή τις τιμές που τις χαρακτηρίζουν.

Παράδειγμα

```
glGetFloatv(GL_CURRENT_COLOR, parameters); //
```

Όπου parameterName η εξεταζόμενη παράμετρος, στην περίπτωση του τρέχοντος χρώματος σχεδίασης στο χρωματικό μοντέλο RGB το μητρώο parameters θα περιέχει τις τιμές των τριών χρωματικών συνιστωσών και θα πρέπει να έχει διάσταση 3.

Το όρισμα parameters είναι ένας δείκτης του μητρώου στο οποίο αποθηκεύονται οι τιμές που προσδιορίζουν την εκάστοτε παράμετρο.

3.6.5 Ενταμιευτές και χρώματα

Πριν αρχίσει ο σχεδιασμός μιας νέας σκηνής, απαιτείται ο καθαρισμός του ενταμιευτή χρωματικών τιμών (color buffer) του υπολογιστή, δηλαδή της περιοχής μνήμης όπου αποθηκεύονται οι χρωματικές πληροφορίες για τη σχεδιαζόμενη σκηνή. Με τον όρο “καθαρισμό” ουσιαστικά εννοείται η αρχικοποίηση των τιμών του ενταμιευτή με κάποια προκαθορισμένη τιμή. Ο καθαρισμός γίνεται με το χρώμα φόντου που επιλέγει ο χρήστης.

Το χρώμα καθαρισμού της οθόνης είναι μια μεταβλητή κατάστασης που η τιμή της καθορίζεται με την εντολή:

glClearColor(GLfloat red, GLfloat green, GLfloat blue, GLfloat alpha) ;

Η τέταρτη παράμετρος είναι συντελεστής μίξης χρωμάτων.

Δεδομένου ότι η μηχανή της OpenGL περιέχει πολλούς ενταμιευτές (Πίνακας 3.2), πρεπει να καθορίστει το είδος του ενταμιευτή που επιθυμεί ο προγραμματιστής να καθαρίσει, περνώντας ως όρισμα την κατάλληλη σταθερά στην εντολή **glClear()**. Προκειμένου λ.χ. να καθαρίστει ο ενταμιευτής χρωματικών τιμών, δίνεται ως όρισμα τη σταθερά **GL_COLOR_BUFFER_BIT**.

Παράδειγμα 1

```
glClear(GL_COLOR_BUFFER_BIT);
```

Παράδειγμα 2

```
glClearColor(0.0, 0.0, 0.0, 0.0); // ορίζει ως χρώμα καθαρισμού το μαύρο
glClear(GL_COLOR_BUFFER_BIT); // Καθαρισμός ενταμιευτή χρώματος.
```

Πίνακας 3.2: Κατηγορίες ενταμιευτών

Ενταμιευτής	Παράμετρος
Color Buffer	GL_COLOR_BUFFER_BIT
Depth Buffer	GL_DEPTH_BUFFER_BIT
Accumulation Buffer	GL_ACCUM_BUFFER_BIT
Stencil Buffer	GL_STENCIL_BUFFER_BIT

Εντολή καθορισμού χρώματος:

Για να καθορίσουμε ένα χρώμα, χρησιμοποιούμε την εντολή **glColor3f()**
glColor3f(1,0,0) ;// δίνει το κόκκινο χρώμα.

Πίνακας 3.3 Βασικά χρώματα

Εντολή	Χρώμα
glColor3f(0,0,0);	Μαύρο
glColor3f(1,0,0);	Κόκκινο
glColor3f(0,1,0);	Πράσινο
glColor3f(0,0,1);	Μπλε
glColor3f(1,0,1);	Πορφυρό
glColor3f(0,1,1);	Κυανό
glColor3f(1,1,1);	Λευκό
glColor3f(1,1,0);	Κίτρινο

3.6.6 Βασικά Σχήματα

3.6.6.1 Καθορισμός Κορυφών

Στην OpenGL, όλα τα γεωμετρικά σχήματα περιγράφονται δηλώνοντας τις κορυφές τους. Για τον καθορισμό μιας κορυφής χρησιμοποιεί την εντολή **glVertex**.

Παράδειγμα

```
glVertex2s(2,3); // Δήλωση σημείου με συντεταγμένες (x,y)=(2,3)
```

Η εντολή **glVertex*()** πρέπει να εκτελείται μεταξύ των εντολών **glBegin()** και **glEnd()**, μέσα σε αυτές τις δύο εντολές δηλώνονται οι συντεταγμένες των σημείων των γεωμετρικών σχημάτων που θα σχεδιαστούν. Στη **glBegin()** σαν παράμετρος μπαίνει το γεωμετρικό σχήμα που θα υλοποιηθεί και με την εντολή **glEnd()** τελειώνει η υλοποίηση του σχήματος.

3.6.6.2 Σχεδίαση γραμμών

Υπάρχουν διαφορετικοί τρόποι σχεδίασης γραμμών. Υπάρχουν τρεις επιλογές:

GL_LINES: Τα σημεία που δίνονται σχηματίζουν γραμμές ανά ζεύγη. Δηλαδή αν δοθούν τα σημεία v1,v2,v3,v4 θα σχηματιστούν 2 γραμμές αντίστοιχα (v1,v2), (v3,v4).

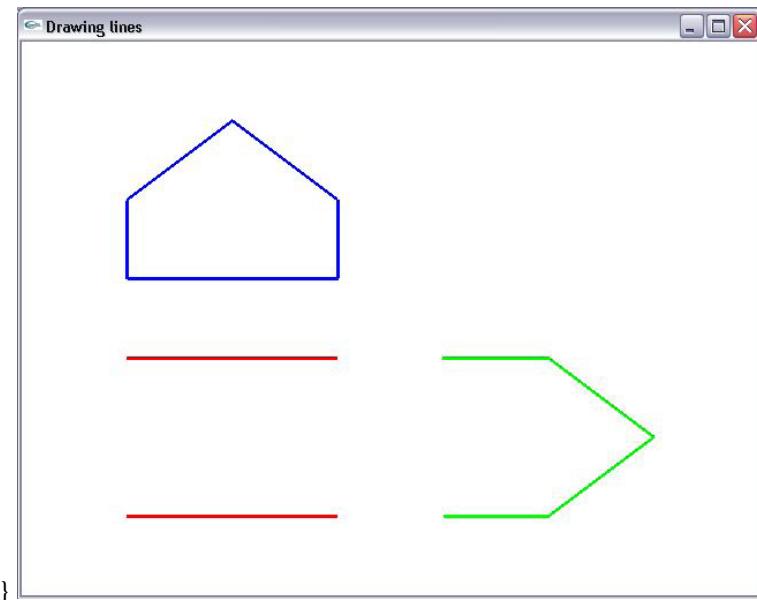
GL LINE STRIP: Ανάλογα τα σημεία που θα δοθούν σχεδιάζονται διαδοχικά ευθύγραμμα τμήματα. Δηλαδή τα σημεία (v1,v2),(v2,v3),(v3,v4),(v4,v5), η μια γραμμή διαδέχεται και ενώνεται με την άλλη.

GL LINE LOOP:

Ομοίως με την GL_LINE_STRIP με τη μόνη διαφορά ότι το πρώτο και τελευταίο σημείο συνενώνονται, σχεδιάζοντας έτσι μία κλειστή γραμμή (loop).

Παράδειγμα: Σχεδίαση γραμμών

```
#include <glut.h>
void display()
{
    glClearColor(1,1,1,0);
    glClear(GL_COLOR_BUFFER_BIT);
    glLineWidth(3); // πάχος γραμμών, για παράμετρο παίρνει έναν αριθμό που δηλώνει το πάχος της γραμμής.
    glColor3f(1,0,0);// καθορισμός κόκκινου χρώματος
    glBegin(GL_LINES);// δημιουργία γραμμών
    glVertex2i(0,0);// δήλωση συντενταγμένων σημείων
    glVertex2i(10,0);
    glVertex2i(0,10);
    glVertex2i(10,10);
    glEnd();
    glColor3f(0,1,0);// καθορισμός πράσινου χρώματος
    glBegin(GL_LINE_STRIP);// δημιουργία διαδοχικών ευθύγραμμων τμημάτων
    glVertex2i(15,10);// δήλωση συντενταγμένων σημείων
    glVertex2i(20,10);
    glVertex2i(25,5);
    glVertex2i(20,0);
    glVertex2i(15,0);
    glEnd();
    glColor3f(0,0,1);// καθορισμός μπλε χρώματος
    glBegin(GL_LINE_LOOP);// δημιουργία διαδοχικών ευθύγραμμων τμημάτων, ενώνεται το τελευταίο σημείο με το πρώτο και δημιουργώντας μια loop.
    glVertex2i(0,15); //δήλωση συντενταγμένων σημείων
    glVertex2i(0,20);
    glVertex2i(5,25);
    glVertex2i(10,20);
    glVertex2i(10,15);
    glEnd();
    glFlush();
}
int main(int argc, char** argv)
{
    glutInit(&argc,argv);// ενεργοποιει την χρήση εντολων της βιβλιοθηκης glut
    glutInitWindowPosition(50,50);// θεση παραθυρου
    glutInitWindowSize(640,480);// μεγεθος παραθυρου σε pixels
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGBA);// ρύθμιση της απεικόνισης του αντικειμένου στην οθόνη(μοντέλο ενταμιευτη, χρωματικο μοντέλο)
    glutCreateWindow("Drawing lines");// δημιουργια παραθυρου και αποδοση τίτλου
    glMatrixMode(GL_PROJECTION);// επιλογη επεξεργασιας μητρωου προβολης
    gluOrtho2D(-5,30,-5,30);// δήλωση παράλληλης προβολής
    glutDisplayFunc(display);// δήλωση συναρτησης που θα εκτελει κάθε φορα που σχεδιαζεται η σκηνη.
    glutMainLoop();//ενεργοποιει τον κύκλο διαχείρισης γεγονότων
    return 0;
```



Σχήμα 3.11 Σχεδίαση γραμμών

3.6.6.3 Διακεκομένες γραμμές

Σχηματίζονται με την εντολή `glLineStipple`. Παίρνει 2 παραμέτρους, η πρώτη κλιμακώνει το μοτίβο, δηλαδή αναπαράγει κάθε δυαδικό ψηφίο όσες φορές δηλώνει η παράμετρος. Η δεύτερη παράμετρος καθορίζει το μοτίβο των γραμμών. Μετατρέπεται η δεκαεξαδική μορφή σε δυαδική μορφή. Το 0 είναι τα κενά και οι άσσοι οι γραμμές.

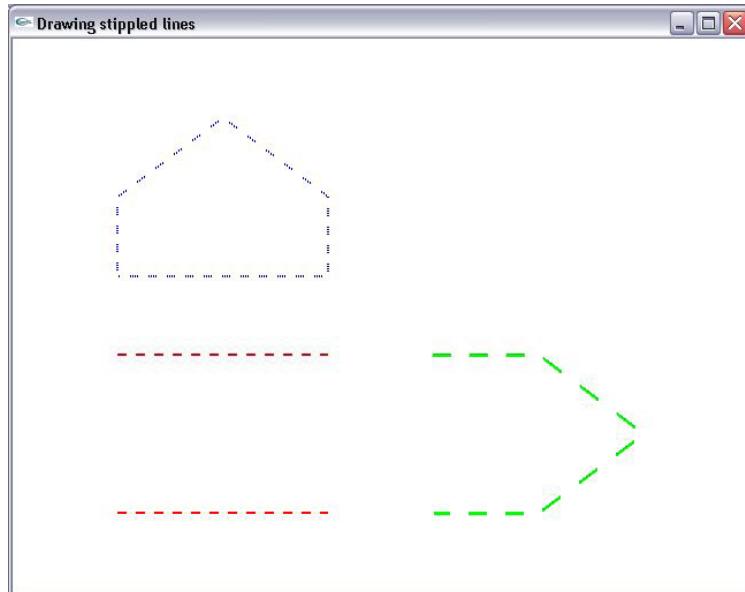
Παράδειγμα: Σχεδίαση διακεκομένων γραμμών

```
#include <glut.h>
void display()
{
    glClearColor(1,1,1,0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1,0,0);
    glLineWidth(2);
    glLineStipple(1,0x00FF);
    glBegin(GL_LINES);
    glVertex2i(0,0);
    glVertex2i(10,0);
    glVertex2i(0,10);
    glVertex2i(10,10);
    glEnd();
    glColor3f(0,1,0);
    glLineWidth(3);
    glLineStipple(2,0x00FF);
    glBegin(GL_LINE_STRIP);
    glVertex2i(15,10);
    glVertex2i(20,10);
    glVertex2i(25,5);
    glVertex2i(20,0);
    glVertex2i(15,0);
    glEnd();
    glColor3f(0,0,1);
    glLineWidth(2);
    glLineStipple(1,0xA0A);
    glBegin(GL_LINE_LOOP);
```

```

glVertex2i(0,15);
glVertex2i(0,20);
glVertex2i(5,25);
glVertex2i(10,20);
glVertex2i(10,15);
glEnd();
glFlush();
}
int main(int argc, char** argv)
{
glutInit(&argc,argv);
glutInitWindowPosition(50,50);
glutInitWindowSize(640,480);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutCreateWindow("Drawing stippled lines");
glMatrixMode(GL_PROJECTION);
gluOrtho2D(-5,30,-5,30);
glEnable(GL_LINE_STIPPLE);
glutDisplayFunc(display);
glutMainLoop();
return 0;
}

```



Σχήμα 3.12 Διακεκομμένες γραμμές

3.6.6.3 Πολύγωνα

Τα πολύγωνα είναι περιοχές που περικλείονται από απλούς κλειστούς βρόχους ευθύγραμμων τμημάτων, όπου τα ευθύγραμμα τμήματα καθορίζονται από τις κορυφές στα άκρα τους.

Υπάρχουν οι εξής διαφοροποιήσεις ως προς το σχηματισμό των πολυγώνων:

GL_POLYGON:

Σχεδιάζει ένα πολύγωνο χρησιμοποιώντας τα σημεία ως κορυφές. Οι κορυφές πρέπει να είναι τουλάχιστον ίσο με 3, ειδάλλως δεν ορίζεται πολυγωνική επιφάνεια. Επιπλέον, το καθορισμένο πολύγωνο πρέπει να είναι κυρτό και να μην έχει τεμνόμενες πλευρές. Αν το πολύγωνο δεν πληρεί αυτές τις προϋποθέσεις, το αποτέλεσμα της σχεδίασής του θα είναι απρόβλεπτο.

GL QUADS

Σχεδιάζει τετράπλευρα. Τα σημεία είναι ανά 4. Εάν τα σημεία δεν είναι πολλαπλάσιο του 4, τότε η μία ή οι δύο ή οι τρεις τελευταίες κορυφές παραλείπονται.

GL QUAD STRIP

Σχεδιάζεται αλληλουχία τετραπλεύρων με μία κοινή πλευρά. Το η πρέπει να είναι τουλάχιστον ίσο με 4 πριν σχεδιαστεί τετράπλευρο. Για η περιττό αριθμό, η τελευταία κορυφή παραλείπεται .

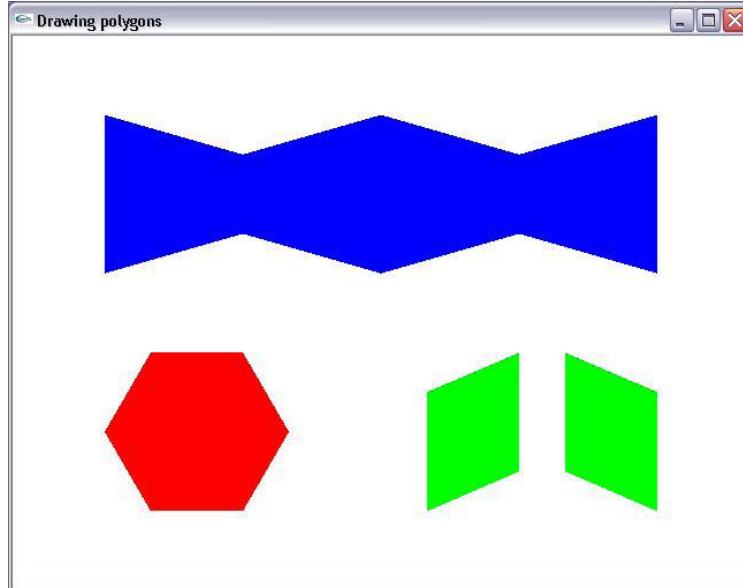
Παράδειγμα: Σχεδίαση πολυγώνων

```
#include <glut.h>
void display()
{
    glClearColor(1,1,1,0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1,0,0);
    glBegin(GL_POLYGON); // δημιουργία πολυγώνου
    glVertex2f(2.5,0);
    glVertex2f(7.5,0);
    glVertex2f(10.5);
    glVertex2f(7.5,10);
    glVertex2f(2.5,10);
    glVertex2f(0.5);
    glEnd();
    glColor3f(0,1,0);
    glBegin(GL_QUADS); // δημιουργία τετράπλευρων
    glVertex2f(17.5,0);
    glVertex2f(17.5,7.5);
    glVertex2f(22.5,10);
    glVertex2f(22.5,2.5);
    glVertex2f(25,2.5);
    glVertex2f(30,0);
    glVertex2f(30,7.5);
    glVertex2f(25,10);
    glEnd();
    glColor3f(0,0,1);
    glBegin(GL_QUAD_STRIP); //διαδοχικά τετράπλευρα με μια κοινή πλευρά
    glVertex2f(0,15);
    glVertex2f(0,25);
    glVertex2f(7.5,17.5);
    glVertex2f(7.5,22.5);
    glVertex2f(15,15);
    glVertex2f(15,25);
    glVertex2f(22.5,17.5);
    glVertex2f(22.5,22.5);
    glVertex2f(30,15);
    glVertex2f(30,25);
    glEnd();
    glFlush();
}
int main(int argc, char** argv)
{
    glutInit(&argc,argv);
    glutInitWindowPosition(50,50);
    glutInitWindowSize(640,480);
```

```

glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutCreateWindow("Drawing polygons");
glMatrixMode(GL_PROJECTION);
gluOrtho2D(-5,35,-5,30);
glutDisplayFunc(display);
glutMainLoop();
return 0;
}

```



Σχήμα 3.13 Πολύγωνα

3.6.6.4 Τρίγωνα

Διακρίνουμε τις εξής διαφοροποιήσεις στη σχεδίαση των τριγώνων:

GL_TRIANGLES:

Σχεδιάζει τρίγωνα. Σχεδιάζονται τρίγωνα με κορυφές (v1,v2,v3) (v4,v5,v6) και ούτω καθεξής. Αν το n δεν είναι ακέραιο πολλαπλάσιο του 3, η τελευταία ή οι δύο τελευταίες κορυφές παραλείπονται.

GL_TRIANGLE_STRIP:

Σχεδιάζει μια αλληλουχία τριγώνων. Διαδοχικά τρίγωνα έχουν μία κοινή πλευρά. Παραδείγματος χάριν (v1,v2,v3) (v3,v4,v5) (v5,v6,v7).

GL_TRIANGLE_FAN:

Σχεδιάζει τρίγωνα που έχουν το πρώτο σημείο κοινό και διαδοχικά τρίγωνα έχουν μία κοινή πλευρά. Επομένως τα τρίγωνα σχηματίζονται από τις κορυφές (v1,v2,v3) (v1,v3,v4) (v1,v4,v5).

Παράδειγμα: Σχεδίαση τριγώνων

```

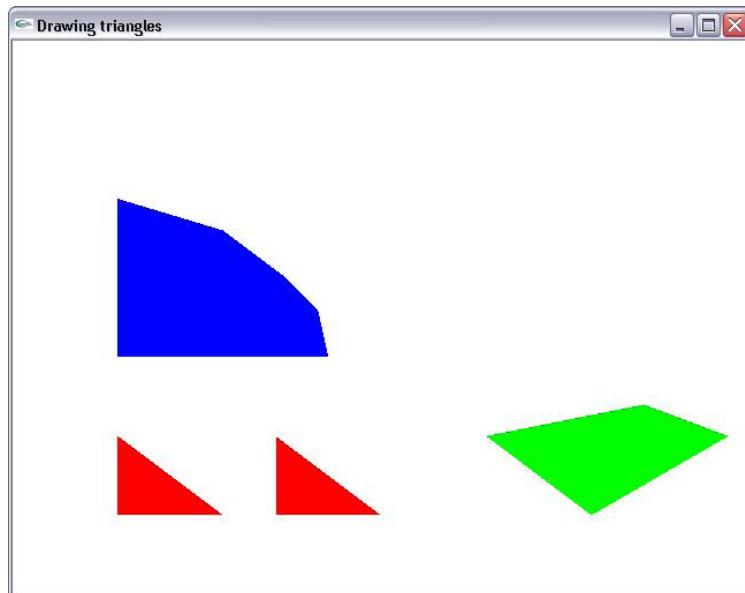
#include <glut.h>
void display()
{
    glClearColor(1,1,1,0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1,0,0);
    glBegin(GL_TRIANGLES);

```

```

glVertex2f(0,0);
glVertex2f(5,0);
glVertex2f(0,5);
glVertex2f(7.5,0);
glVertex2f(12.5,0);
glVertex2f(7.5,5);
glEnd();
glColor3f(0,1,0);
glBegin(GL_TRIANGLE_STRIP);
glVertex2f(17.5,5);
glVertex2f(22.5,0);
glVertex2f(22.5,5);
glVertex2f(25,5);
glEnd();
glColor3f(0,0,1);
glBegin(GL_TRIANGLE_FAN);
glVertex2f(0,10);
glVertex2f(10,10);
glVertex2f(9.5,13);
glVertex2f(8,15);
glVertex2f(5,18);
glVertex2f(0,20);
glEnd();
glFlush();
}
int main(int argc, char** argv)
{
glutInit(&argc,argv);
glutInitWindowPosition(50,50);
glutInitWindowSize(640,480);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutCreateWindow("Drawing triangles");
glMatrixMode(GL_PROJECTION);
gluOrtho2D(-5,30,-5,30);
glutDisplayFunc(display);
glutMainLoop();
return 0;
}

```



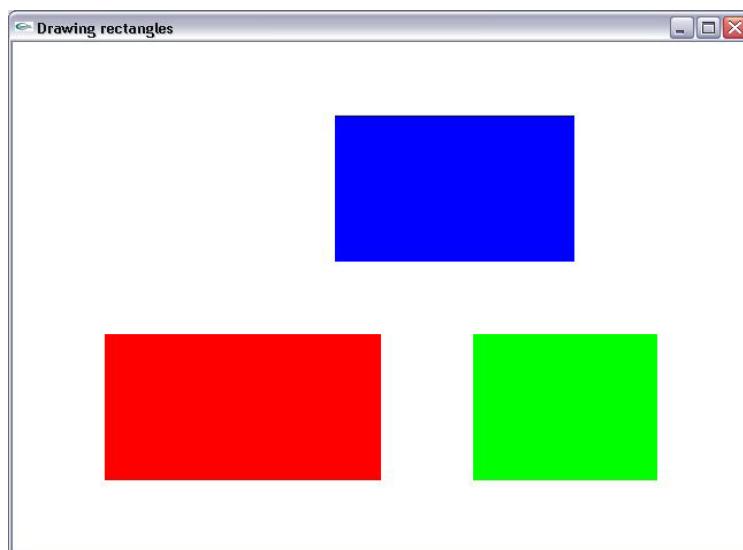
Σχήμα 3.14 Τρίγωνα

3.6.6.5 Ορθογώνια

Για το σχεδιασμό ενός στοιχειώδους ορθογωνίου χρησιμοποιείτε η εντολή `glRect*()`.
Παράδειγμα: `void glRect{ sifd}(TYPE x1, TYPE y1, TYPE x2, TYPE y2);`
`void glRect{ sifd}v(TYPE *v1, TYPE *v2);`

Παράδειγμα: Σχεδιάση ορθογωνίων

```
#include <glut.h>
void display()
{
    glClearColor(1,1,1,0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1,0,0);
    glRectf(0,0,15,10);
    glColor3f(0,1,0);
    glRectf(20,0,30,10);
    glColor3f(0,0,1);
    glRectf(12.5,15,25.5,25);
    glFlush();
}
int main(int argc, char** argv)
{
    glutInit(&argc,argv);
    glutInitWindowPosition(50,50);
    glutInitWindowSize(640,480);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutCreateWindow("Drawing rectangles");
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(-5,35,-5,30);
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```



Σχήμα 3.15 Ορθογώνια

3.6.6.6 Σχεδίαση κύκλων

Η σχεδίαση κυκλικών σχημάτων επιτυγχάνεται χρησιμοποιώντας την παραμετρική εξίσωση κύκλου σε πολικές συντεταγμένες. Στην περίπτωση αυτή, κάθε συντεταγμένη της περιφέρειας ενός κύκλου προκύπτει από τις εξισώσεις

$$x = x + r * \cos \theta \quad 0 \leq \theta \leq 2\pi$$

$$y = y + r * \sin \theta$$

όπου x, y οι συντεταγμένες του κέντρου του κύκλου και r η ακτίνα του.

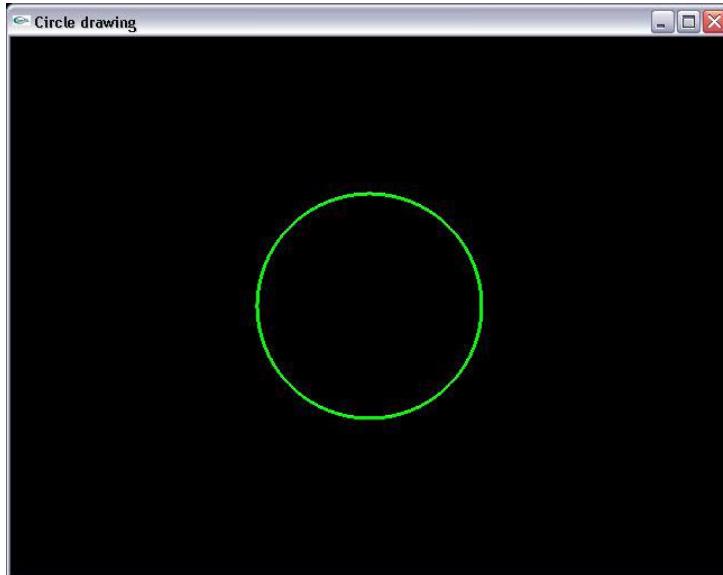
Παράδειγμα: Σχεδίαση κύκλου

```
#include <glut.h>
#include <math.h>
#define PI 3.14159
#define circlePoints 256
int i;
void display()
{
    GLfloat angleStep=2*PI/(float)circlePoints;
    GLuint pointsPerQuarter=circlePoints/4;
    GLfloat x[circlePoints];
    GLfloat y[circlePoints];
    GLfloat radius=10;
    glClearColor(0,0,0,0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0,1,0);
    glLineWidth(3);
    for(i=0;i<pointsPerQuarter/2;i++)
    {
        //Define points in first quadrant
        x[i]=radius*cos(i*angleStep);
        y[i]=radius*sin(i*angleStep);
        x[pointsPerQuarter-1-i]=y[i];
        y[pointsPerQuarter-1-i]=x[i];
        //Define points in second quadrant
        x[pointsPerQuarter+i]=-y[i];
        y[pointsPerQuarter+i]=x[i];
        x[2*pointsPerQuarter-1-i]=-x[i];
        y[2*pointsPerQuarter-1-i]=y[i];
        //Define points in third quadrant
        x[2*pointsPerQuarter+i]=-x[i];
        y[2*pointsPerQuarter+i]=-y[i];
        x[3*pointsPerQuarter-1-i]=-y[i];
        y[3*pointsPerQuarter-1-i]=-x[i];
        //Define points in fourth quadrant
        x[3*pointsPerQuarter+i]=y[i];
        y[3*pointsPerQuarter+i]=-x[i];
        x[4*pointsPerQuarter-1-i]=x[i];
        y[4*pointsPerQuarter-1-i]=-y[i];
    }
    glBegin(GL_LINE_LOOP);
    for (i=0;i<circlePoints;i++)
    {
        glVertex2f(x[i],y[i]);
    }
    glEnd();
    glFlush();
}
int main(int argc, char** argv)
```

```

{
glutInit(&argc,argv);
glutInitWindowPosition(50,50);
glutInitWindowSize(640,480);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGBA);
glutCreateWindow("Circle drawing");
glMatrixMode(GL_PROJECTION);
gluOrtho2D(-32,32,-24,24);
glutDisplayFunc(display);
glutMainLoop();
return 0;
}

```



Σχήμα 3.16 Σχεδίαση κύκλων

3.6.6.7 Όψεις πολυγώνων

Στην OpenGL κάθε πολύγωνο χαρακτηρίζεται από δύο όψεις: τη μπροστινή και την πίσω όψη. Η επιλογή του ποια όψη θα χαρακτηρίστει ως μπροστινή ή πίσω είναι αυθαίρετη. Συνήθως χαρακτηρίζεται η εξωτερική τους όψη ως “μπροστινή” και η εσωτερική ως “πίσω” όψη. Οι όψεις είναι σημαντικές, για παράδειγμα εάν θέλει ο χρήστης να βάλει διαφορετική υφή σε κάθε όψη.

Κάθε όψη ενός πολυγώνου (μπροστινή και πίσω) μπορεί να σχεδιαστεί με διαφορετικό τρόπο, π.χ. μπορεί να σχεδιαστούν οι μπροστινές όψεις των πολυγώνων συμπαγείς και οι πίσω όψεις ως περιγράμματα. Τροποποιούμε τις όψεις με την εντολή `glPolygonMode()`.

Ολόκληρη η εντολή είναι: `void glPolygonMode(GLenum face, GLenum mode);`

face: σε ποια όψη θα γίνει η τροποποίηση

mode: δηλώνει τον τρόπο απεικόνισης της όψης που θα γίνει η τροποποίηση

Η παράμετρος **face** δέχεται τρείς πιθανές τιμές:

`GL_FRONT`: Η τροποποίηση θα γίνει στις μπροστινές όψεις

`GL_BACK`: Η τροποποίηση αφορά τις πίσω όψεις

`GL_FRONT_AND_BACK`: Η τροποποίηση αφορά και τις δύο όψεις.

Για την παράμετρο **mode** υπαρχουν τρεις επιλογές:

`GL_FILL`: Η όψη σχεδιάζεται σημπαγής.

`GL_LINE`: Σχεδιάζεται μόνο το περίγραμμα της όψης.

`GL_POINT`: Σχεδιάζονται μόνο οι κορυφές της όψης.

Παραδείγματα: `glPolygonMode(GL_FRONT,GL_LINE) ; //τροποποιείται η μπροστινή όψη και δημιουργείται μόνο με περίγραμμα.`
`glPolygonMode(GL_BACK,GL_POINT); //τροποποιείται η πίσω όψη και δημιουργούνται μόνο οι κορυφές της όψης.`

3.6.6.8 Καταστολή όψεων

Σε έναν εξωτερικό παρατηρητή σε μια κλειστή πολυγωνική επιφάνεια η πίσω όψη πάντα θα κρύβεται από την μπροστινή. Παρομοίως σε ένα εσωτερικό παρατηρητή η μπροστινή όψη θα κρύβεται από την πίσω. Στην περίπτωση αυτή, υπάρχει η δυνατότητα να αυξηθεί η ταχύτητα σχεδιασμού, αναθέτοντας στην OpenGL να απορρίπτει τη σχεδίαση όψεων που επιθυμεί ο κάθε χρήστης. Εάν δεν δηλώσουμε ποια επιθυμούμε το σύστημα θα απορρίψει από μόνο του τη πίσω όψη.

Για την ενέργεια αυτή η εντολή είναι: **glCullFace()**

Πρώτα πρέπει να ενεργοποιησουμε αυτή την ιδιοτητα με την εντολή

void glEnable(GL_CULL_FACE);

Ο καθορισμός του τρόπου λειτουργίας της απόρριψης όψεων γίνεται με την εντολή **glCullFace();**

void glCullFace(GLenum mode);

Το όρισμα mode δέχεται τις τιμές **GL_FRONT**, **GL_BACK** ή **GL_FRONT_AND_BACK** για να δηλώσει front-facing, back-facing ή όλα τα πολύγωνα.

Παράδειγμα: Αλλαγή στον τρόπο σχεδίασης όψεων πολυγώνων

```
#include <glut.h>
void display()
{
    glClearColor(0,0,0,0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.5,0,0);
    //Front face of a polygon - Vertices defined in counter clockwise order
    glBegin(GL_POLYGON);
    glVertex2i(0,0);
    glVertex2i(10,0);
    glVertex2i(15,10);
    glVertex2i(10,20);
    glVertex2i(0,20);
    glEnd();
    //Back face of a polygon - Vertices defined in clockwise order
    glColor3f(0,0.5,0);
    glBegin(GL_POLYGON);
    glVertex2i(30,0);
    glVertex2i(25,10);
    glVertex2i(30,20);
    glVertex2i(40,20);
    glVertex2i(40,0);
    glEnd();
    glFlush();
}
int main(int argc, char** argv)
{
    glutInit(&argc,argv);
    glutInitWindowPosition(50,50);
    glutInitWindowSize(640,480);
```

```

glutInitDisplayMode(GLUT_SINGLE|GLUT_RGBA);
glutCreateWindow("Polygon front and back faces");
glMatrixMode(GL_PROJECTION);
gluOrtho2D(-10,50,-10,50);
//Polygon front faces are to be filled.
glPolygonMode(GL_FRONT,GL_FILL);
//Polygon back faces are to be drawn as lines
glPolygonMode(GL_BACK,GL_LINE);
glutDisplayFunc(display);
glutMainLoop();
return 0;
}

```

3.6.6.9 Λίστες απεικόνισης (display lists)

Οι λίστες απεικόνισης διευκολύνουν την επαναχρησιμοποίηση κώδικα και απαλλάσσουν τον προγραμματιστή από περιττές επαναλαμβανόμενες δηλώσεις του ίδιου σύνθετου σχήματος. Δηλαδή όταν γραφτεί ένα κομμάτι κώδικα ενός δύσκολου σχήματος για να μην γράφεται πάλι ολόκληρος ο κώδικας απλά καλείτε ο ίδιος κώδικας μέσω της λίστας απεικόνισης. Μια λίστα απεικόνισης μπαίνει μεταξύ δύο εντολών: των **glNewList** και **glEndList**. Όταν υπάρχουν πολλές λίστες απεικόνισης στο πρόγραμμα, η κάθε λίστα έχει ένα ακέραιο αριθμό σαν αναγνωριστικό. Για να γίνει αυτό πρέπει να δεσμευτεί το κατάλληλο εύρος τιμών. Αυτό γίνεται με την εντολή **glGenLists()**. Ολόκληρη η εντολή: **GLuint glGenLists(GLint range);**

Range: το πλήθος των αναγνωριστικών που θα χρησιμοποιηθεί.

Παράδειγμα: **listID=glGenLists(2)** ; Δεσμεύουμε για δύο αναγνωριστικά. Ο πρώτος έχει ακέραιη τιμή listID, ο δεύτερος listID+1 κτλ.

Η αναγνωριστική τιμή σε μια λίστα απεικόνισης ανατίθεται κατά την δήλωσής της στην εντολή **glNewList**:

void glNewList(GLuint listID, GLenum listMode) ;

listID: το αναγνωριστικό που θέλουμε να αποδώσουμε στη λίστα απεικόνισης.

listMode: έχει δύο πιθανές τιμές

GL_COMPILE: Δηλώνεται ο κώδικας σχεδιασμού του σύνθετου αντικειμένου που περιγράφεται στη λίστα απεικόνισης

GL_COMPILE_AND_EXECUTE: Δηλώνεται και εκτελείτε ταυτόχρονα ο κώδικας σχεδιασμού που περιέχεται στη λίστα απεικόνισης.

Ο κώδικας που περιέχεται σε μία λίστα απεικόνισης **εκτελείται** δίνοτας τον αναγνωριστικό της αριθμό ως όρισμα στην εντολή **glCallList()**:

void glCallList(GLuint listID) ;

Η **διαγραφή** μίας ή περισσοτέρων λιστών απεικόνισης (αποδέσμευση των αναγνωριστικών αριθμών τους) γίνεται με την εντολή **glDeleteLists**:

glDeleteLists(startId, nLists) ;

startId: ο αναγνωριστικός αριθμός της πρώτης λίστας απεικόνισης

nLists: το πλήθος των λιστών που θα διαγραφούν.

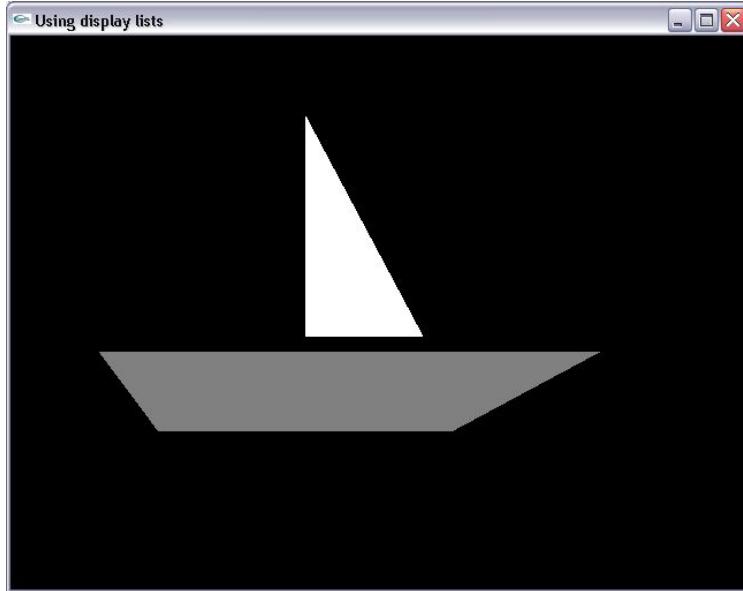
Παράδειγμα:

glDeleteLists(someListID,3) ;

Διαγράφονται οι λίστες απεικόνισης με αναγνωριστικούς αριθμούς από το πρώτο που λέει η πρώτη παράμετρος μέχρι πόσους λέει η δεύτερη παράμετρος.

Παράδειγμα: Ορισμός και εκτέλεση λίστας απεικόνισης

```
#include <glut.h>
GLuint listID;
void Boat(GLsizei displayListID)
{
    glNewList(displayListID,GL_COMPILE);
    glColor3f(0.5,0.5,0.5);
    glBegin(GL_POLYGON);
    glVertex2f(15,10);
    glVertex2f(20,15);
    glVertex2f(3,15);
    glVertex2f(5,10);
    glEnd();
    glColor3f(1,1,1);
    glBegin(GL_TRIANGLES);
    glVertex2f(14,16);
    glVertex2f(10,30);
    glVertex2f(10,16);
    glEnd();
    glEndList();
}
void display()
{
    glClearColor(0,0,0,0);
    glClear(GL_COLOR_BUFFER_BIT);
    glCallList(listID);
    glFlush();
}
int main(int argc, char** argv)
{
    glutInit(&argc,argv);
    glutInitWindowPosition(50,50);
    glutInitWindowSize(640,480);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGBA);
    glutCreateWindow("Using display lists");
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0,25,0,35);
    listID=glGenLists(1);
    Boat(listID);
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```



Σχήμα 3.17 Λίστες απεικόνισης (display lists)

3.6.6.10 Μητρώα σημείων - Μητρώα χρωμάτων

Η χρησιμότητα των μητρώων σημείων αναδεικνύεται σε εφαρμογές μεγάλης κλίμακας, στις οποίες υπάρχει ένας πολύ μεγάλος όγκος δεδομένων (δειγμάτων με τη μορφή σημείων) και θέλει ο χρήστης να σχηματίσει μια γραφική απεικόνιση αυτών των δεδομένων με τον ελάχιστο δυνατό αριθμό εντολών.

Αρχικά απαιτείται η **ενεργοποίηση** της χρήσης μητρώων σημείων-χρωμάτων με την εντολή **glEnableClientState**:

void glEnableClientState(GLenum array);

Array: η κατηγορία των δεδομένων που περιέχονται στο μητρώο. Η OpenGL μπορεί να υποστηρίξει τους εξής τύπους μητρώων:

GL_VERTEX_ARRAY: Ενεργοποιείται η χρήση μητρώων που περιέχουν συντεταγμένες σημείων. Χρησιμοποιείται όταν θέλει ο χρήστης να ορίσει τις συντεταγμένες πολλαπλών κορυφών.

GL_COLOR_ARRAY: Ενεργοποιείται η χρήση μητρώων που περιέχουν συντελεστές χρωματικών τιμών. Χρησιμοποιείται όταν θέλει ο χρήστης να δηλώσει χρωματικές τιμές σε πολλαπλές κορυφές.

GL_TEXTURE_ARRAY: Χρησιμεύει στην απόδοση υφής.

Η χρήση πολλαπλών σημείων ή/και χρωματικών τιμών εκτελείται με την εντολή glDrawElements:

Void glDrawElements(GLenum mode, GLsizei count, GLenum type, GLvoid *indices);

***indices:** δείκτης στο μητρώο που περιέχει τους δείκτες των σημείων ή/και χρωμάτων με τη σειρά χρήσης τους.

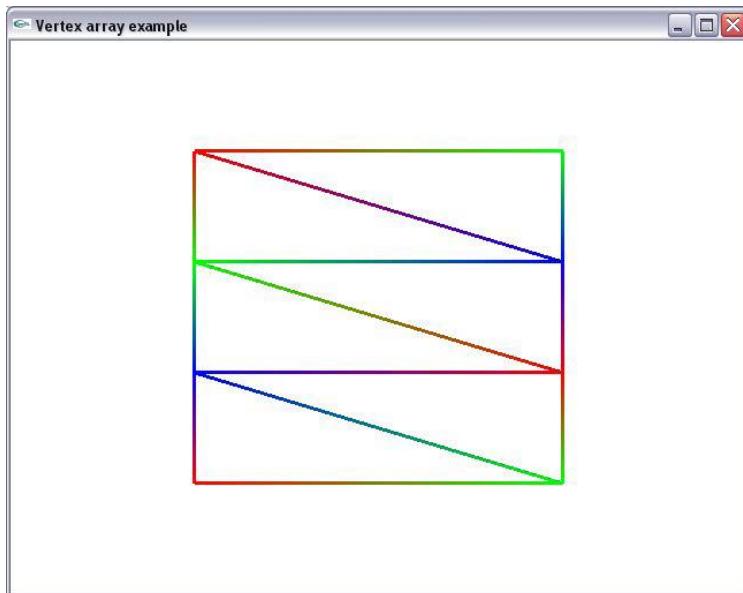
Mode: καθορίζει τι σχήματα ορίζουν τα σημεία και παίρνει τις ίδιες τιμές που χρησιμοποιούνται στη δομή glBegin/glEnd (GL_LINES, GL_TRIANGLES, GL_QUADS κ.λ.π.).

count: προσδιορίζει το πλήθος των σημείων ή/και χρωματικών τιμών που περιέχονται στα μητρώα σημείων ή/και χρωμάτων.

Type: καθορίζει τον πρωτογενή τύπο δεδομένων με τον οποίο δίνονται οι δείκτες στο μητρώο indices. Οι επιτρεπόμενες αριθμητικές σταθερές είναι GL_UNSIGNED_BYTE, GL_UNSIGNED_SHORT και GL_UNSIGNED_INT για πρωτογενείς τύπους δεδομένων Glubyte, GLshort και GLuint αντίστοιχα.

Παράδειγμα: Συεδίαση ταινίας τριγώνων

```
#include <glut.h>
GLuint vertexArray[]={0,0, 10,0, 0,5, 10,5, 0,10, 10,10, 0,15, 10,15};//τα σημεία της κάθε κορυφής x,y, 8 κορυφές.
GLfloat colorArray[]={1,0,0, 0,1,0, 0,0,1, 1,0,0, 0,1,0, 0,0,1, 1,0,0, 0,1,0};τα χρώματα που θα μπουν σε κάθε σημείο, 8 χρώματα στο χρωματικό μοντέλο RGB.
GLuint vertexIndex[]={0,1,2,3,4,5,6,7} ; //δίνει την σειρά χρήσης των σημείων και των χρωμάτων
void display()
{
glClearColor(1,1,1,0);
glClear(GL_COLOR_BUFFER_BIT);
glEnableClientState(GL_VERTEX_ARRAY) ; //ενεργοποίηση της χρήσης μητρώων σημείων
glEnableClientState(GL_COLOR_ARRAY) ; ενεργοποίηση της χρήσης μητρώων χρωμάτων
glVertexPointer(2,GL_INT,0,vertexArray) ; //Οταν ορίζει ο προγραμματιστής τις συντεταγμένες όλων των κορυφών σε ένα μητρώο έστω vertexArray, μετά δηλώνεται στην OpenGL ένας δείκτης για το μητρώο σημείων που είναι οι συντεταγμένες, παράμετροι: 1.οι διαστάσεις του χώρου, 2.τύπος δεδόμενων,3. καθορίζει την απόσταση μεταξύ των συντεταγμένων διαδοχικών σημείων στο μητρώο, 4. είναι δείκτης στο μητρώο με τις τιμές των συντεταγμένων
glColorPointer(3,GL_FLOAT,0,colorArray) ; //Εάν θέλουμε να ορίσουμε πολλαπλές χρωματικές τιμές, η πρώτη παράμετρος είναι για τις χρωματικά μοντέλα, 3 για το RGB, 4 για το RGBA.
glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
glDrawElements(GL_TRIANGLE_STRIP,8,GL_UNSIGNED_INT,vertexIndex) ;
glFlush();
}
int main(int argc, char** argv)
{
glutInit(&argc,argv);
glutInitWindowPosition(50,50);
glutInitWindowSize(640,480);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutCreateWindow("Vertex array example");
glMatrixMode(GL_PROJECTION);
gluOrtho2D(-5,15,-5,20);
glutDisplayFunc(display);
glutMainLoop();
return 0;
}
```



Σχήμα 3.18 Σχεδίαση ταινίας τριγώνων

Πίνακας 3.4 Έγκυρες εντολές εντός της δομής glBegin() και glEnd()

Εντολή	Σημασία
glVertex*()	Θέτει τις συντεταγμένες της κορυφής.
glColor*()	Θέτει το τρέχον χρώμα.
glIndex*()	Θέτει τον τρέχοντα χρωματικό πίνακα.
glNormal*()	Θέτει τις συντεταγμένες του κανονικού διανύσματος.
glEvalCoord*()	Δημιουργεί συντεταγμένες.
glCallList(), glCallLists()	Εκτελεί την/τις λίστα/λίστες απεικόνισης.
glTexCoord*()	Θέτει τις συντεταγμένες της υφής.
glEdgeFlag*()	Ελέγχει το σχεδιασμό των άκρων.
glMaterial*()	Θέτει τις ιδιότητες του υλικού.

3.6.7 Μετασχηματισμοί συντεταγμένων

Νωρίτερα αναλύθηκαν βασικές εντολές σχεδίασης, μέσω των οποίων ο προγραμματιστής μπορεί να να καθορίσει τις συντεταγμένες της σκηνής στις οποίες επιθυμεί να σχεδιάσει γεωμετρικά σχήματα. Ωστόσο υπάρχουν περιπτώσεις στις οποίες είναι επιθυμητή η επιβολή ενός μετασχηματισμού συντεταγμένων ή μιας αλυσίδας μετασχηματισμών συντεταγμένων, πριν τον προσδιορισμό της θέσης στην οποία θα σχεδιαστεί ένα σημείο. Δηλαδή μετασχηματισμός είναι η απεικόνιση ενός σημείου ή διανύσματος σε άλλο σημείο ή διάνυσμα από αυτό που ορίζεται αρχικά από την εντολή.

3.6.7.1 Μητρώα μετασχηματισμού

Η μηχανή καταστάσεων της OpenGL προβλέπει τη χρήση δύο διαφορετικών μητρώων, τα οποία, συνδυαζόμενα, παράγουν την τελική απεικόνιση της σκηνής στην οθόνη του

χρήστη. Τα μητρώα αυτά είναι το **μητρώο μετασχηματισμού μοντέλου** (modelview matrix) και το **μητρώο προβολής** (projection matrix).

Στο **μητρώο μετασχηματισμού μοντέλου** εμπεριέχονται οι μετασχηματισμοί που εφαρμόζονται προκειμένου η σκηνή να αποδοθεί από διαφορετικές οπτικές γωνίες. Στην περίπτωση αυτή, η περιγραφή της σκηνής ανάγεται σε σύστημα συντεταγμένων που η θέση του καθορίζεται από την οπτική γωνία του θεατή και οι μετασχηματισμοί που εμπλέκονται στη διαδικασία αυτή ενσωματώνονται στο μητρώο μετασχηματισμού μοντέλου.

Η διαδικασία της προβολής ουσιαστικά είναι μια διεργασία, κατά την οποία, οι συντεταγμένες της σκηνής αντιστοιχίζονται, βάσει ενός καθορισμένου κανόνα, σε συντεταγμένες πάνω στο επίπεδο προβολής. Ο μαθηματικός κανόνας στον οποίο βασίζεται η αντιστοιχίση αυτή, αποθηκεύεται **στο μητρώο προβολής**. Το **μητρώο προβολής** πολλαπλασιάζεται με τις συντεταγμένες των σημείων της σκηνής (τις ανηγμένες στο σύστημα συντεταγμένων του παρατηρητή) και παράγει (σε ενδιάμεσο στάδιο) τις συντεταγμένες στις οποίες προβάλλονται στο επίπεδο του παρατηρητή, το λεγόμενο επίπεδο προβολής.

Στην OpenGL, σε κάθε χρονική στιγμή, είναι δυνατή η πρόσβαση μόνο σε ένα από τα δύο μητρώα. Ο χρήστης καθορίζει ποιο από τα μητρώα επιθυμεί να τροποποιήσει την εκάστοτε χρονική στιγμή με την εντολή **glMatrixMode**:

glMatrixMode(matrixMode) ;

Η **matrixMode** παίρνει μια από τις δύο παραμέτρους

GL_MODELVIEW: μετάβαση στην κατάσταση επεξεργασίας του μητρώου μετασχηματισμού μοντέλου.

GL_PROJECTION: μετάβαση στην κατάσταση επεξεργασίας του μητρώου προβολής.

Στοιχειώδεις μετασχηματισμοί

Μετατόπιση

Στην OpenGL η μετατόπιση των συντεταγμένων της σκηνής στο χώρο εκτελείται με την εντολή **glTranslate**.

glTranslatef (GLfloat xtr,GLfloat ytr, GLfloat ztr) ;

glTranslated (GLdouble xtr, GLdouble ytr, GLdouble ztr) ;

Κλιμάκωση

Στην κλιμάκωση, οι συντεταγμένες πολλαπλασιάζονται με ένα σταθερό ανά διεύθυνση συντελεστή. Στην OpenGL, η κλιμάκωση εκτελείται με την εντολή **glScale**:

glScalef(GLfloat sx, GLfloat sy, GLfloat sz) ;

glScaled(GLdouble sx, GLdouble sy GLdouble sz) ;

όπου sx,sy,sz, οι συντελεστές κλιμάκωσης κατά τις διευθύνσεις x,y,z, αντίστοιχα.

Κλίση

Στους μετασχηματισμούς κλίσης, η τιμή μίας από τις συντεταγμένες x,y,z των σημείων μεταβάλλεται γραμμικά ως προς μία ή περισσότερες εκ των άλλων δύο συντεταγμένων.

Περιστροφή

Στην OpenGL μπορούμε να εκτελέσουμε μετασχηματισμούς περιστροφής ως προς οποιαδήποτε άξονα περιστροφής. Αυτό επιτυγχάνεται ορίζοντας τις συνιστώσες του διανύσματος που ορίζει τη διεύθυνση ενός άξονα περιστροφής. Ο άξονας περιστροφής διέρχεται από την αρχή των αξόνων.

Μετασχηματισμοί περιστροφής εκτελούνται με την εντολή **glRotate**:

glRotatef (GLfloat angle, GLfloat vx, GLfloat vy, GLfloat vz) ;

glRotated (GLdouble angle, GLdouble vx, GLdouble vy, GLdouble vz) ;

παράμετροι:

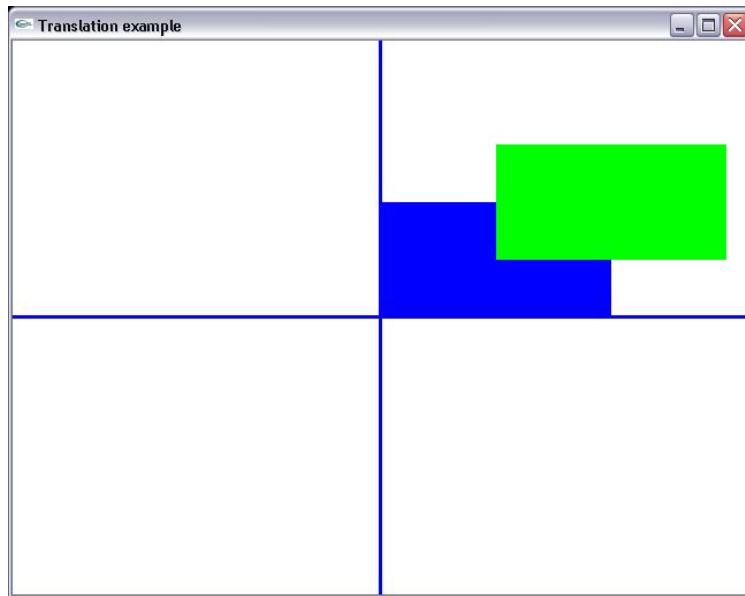
angle: η γωνία περιστροφής σε μοίρες

vx,vy,vz: οι συνιστώσες του διανύσματος που εκφράζει τη διεύθυνση του άξονα περιστροφής (Ο άξονας περιστροφής διέρχεται από την αρχή των αξόνων του συστήματος συντεταγμένων.)

Η φορά περιστροφής καθορίζεται από τη φορά του διανύσματος (vx,vy,vz,) σύμφωνα με τον κανόνα του δεξιού χεριού.

Παράδειγμα: Μετασχηματισμός μετατόπισης

```
#include <glut.h>
void display()
{
    glLoadIdentity(); // αρχικοποίηση των μητρώων προβολής και μετασχηματισμού μοντέλου στο μητρώο.
    glClearColor(1,1,1,0);
    glClear(GL_COLOR_BUFFER_BIT);
    glLineWidth(3);
    glColor3f(0,0,1);
    glBegin(GL_LINES);
    //Σχεδιάζοντας τον Oy άξονα
    glVertex2f(0,-24);
    glVertex2f(0,24);
    // Σχεδιάζοντας τον Ox άξονα
    glVertex2f(-32,0);
    glVertex2f(32,0);
    glEnd();
    //Σχεδιάζοντας ένα ορθογώνιο
    glRecti(0,0,20,10);
    glColor3f(0,1,0);
    //Όλα τα δηλωμένα σχήματα θα μετατοπιστούν σε σχέση με τις συντεταγμένες που δίνονται.
    glTranslatef(10,5,0);
    //Αυτό το ορθογώνιο θα σχεδιαστεί μετατοπισμένο
    glRecti(0,0,20,10);
    glFlush();
}
int main(int argc, char** argv)
{
    glutInit(&argc,argv);
    glutInitWindowPosition(50,50);
    glutInitWindowSize(640,480);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutCreateWindow("Translation example");
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(-32,32,-24,24);
    glMatrixMode(GL_MODELVIEW);
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```



Σχήμα 3.19 Μετασχηματισμός μετατόπισης

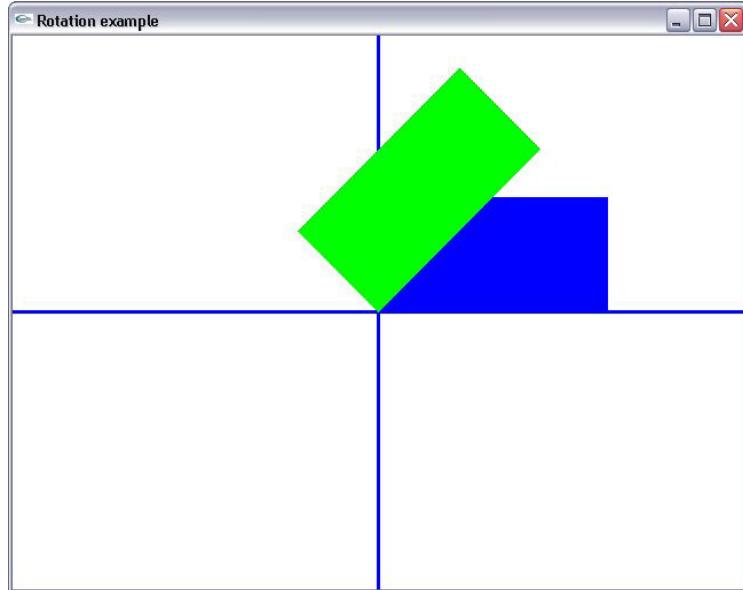
Παράδειγμα: Μετασχηματισμός περιστροφής

```
#include <glut.h>
void display()
{
    glLoadIdentity(); // αρχικοποίηση των μητρώων προβολής και μετασχηματισμού μοντέλου στο μητρώο.
    glClearColor(1,1,1,0);
    glClear(GL_COLOR_BUFFER_BIT);
    glLineWidth(3);
    glColor3f(0,0,1);
    glBegin(GL_LINES);
    // Σχεδιάζοντας τον Oy άξονα
    glVertex2f(0,-24);
    glVertex2f(0,24);
    // Σχεδιάζοντας τον Ox άξονα
    glVertex2f(-32,0);
    glVertex2f(32,0);
    glEnd();
    //Σχεδιαζόντας ένα μπλε ορθογωνιό
    glRecti(0,0,20,10);
    glColor3f(0,1,0);
    // Όλα τα σχήματα που δηλώνονται μετά από αυτή την εντολή θα περιστρέφονται δεξιόστροφα κατά 45
    μοίρες, σε σχέση με τις συντεταγμένες που εχουν δοθεί.
    glRotatef(45,0,0,1); //πραγματοποιείται μετασχηματισμός περιστροφής
    // Θα σχεδιαστεί ένα πράσινο ορθογώνιο που περιστρέφεται κατά 45 μοίρες.
    glRecti(0,0,20,10);
    glFlush();
}
int main(int argc, char** argv)
{
    glutInit(&argc,argv);
    glutInitWindowPosition(50,50);
    glutInitWindowSize(640,480);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutCreateWindow("Rotation example");
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(-32,32,-24,24);
```

```

glMatrixMode(GL_MODELVIEW);
glutDisplayFunc(display);
glutMainLoop();
return 0;
}

```



Σχήμα 3.20 Μετασχηματισμός περιστροφής

Παράδειγμα: Σύνθετος μετασχηματισμός (περιστροφή και μετατόπιση)

```

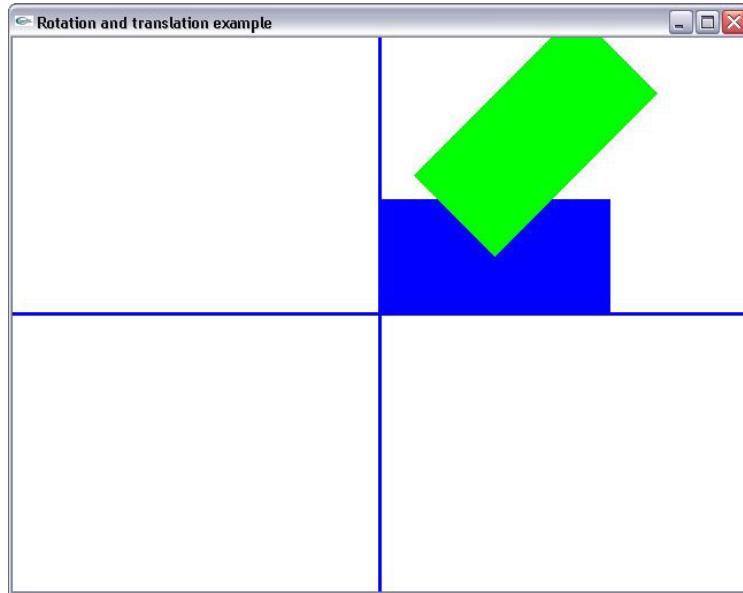
#include <glut.h>
void display()
{
    glLoadIdentity(); // αρχικοποίηση των μητρώων προβολής και μετασχηματισμού μοντέλου στο μητρώο.
    glClearColor(1,1,1,0);
    glClear(GL_COLOR_BUFFER_BIT);
    glLineWidth(3);
    glColor3f(0,0,1);
    glBegin(GL_LINES);
    //Σχεδιάζοντας τον Oy άξονα
    glVertex2f(0,-24);
    glVertex2f(0,24);
    // Σχεδιάζοντας τον Ox άξονα
    glVertex2f(-32,0);
    glVertex2f(32,0);
    glEnd();
    //Σχεδιάζοντας ένα μπλε ορθογώνιο.
    glRecti(0,0,20,10);
    glColor3f(0,1,0);
    // Οι μετασχηματισμοί δίνονται με την αντίστροφη σειρά (η πρώτη που εφαρμόζεται στις συντεταγμένες είναι το τελευταίο που έχει δηλωθεί)
    glTranslatef(10,5,0);
    glRotatef(45,0,0,1);
    // Ο σύνθετος μετασχηματισμός θα εφαρμοστεί σε αυτό το ορθογώνιο.
    glRecti(0,0,20,10);
    glFlush();
}
int main(int argc, char** argv)
{
    glutInit(&argc,argv);
}

```

```

glutInitWindowPosition(50,50);
glutInitWindowSize(640,480);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutCreateWindow("Rotation and translation example");
glMatrixMode(GL_PROJECTION);
gluOrtho2D(-32,32,-24,24);
glMatrixMode(GL_MODELVIEW);
glutDisplayFunc(display);
glutMainLoop();
return 0;
}

```



Σχήμα 3.21 Μετασχηματισμός περιστροφής και μετατόπισης

3.6.7.2 Μετασχηματισμός οπτικής γωνίας

Στην OpenGL, ο μετασχηματισμός οπτικής γωνίας εκτελείται με απλό τρόπο χρησιμοποιώντας την εντολή **gluLookAt**:

gluLookAt(GLdouble eyeX, GLdouble eyeY, GLdouble eyeZ, GLdouble centerX, GLdouble centerY, GLdouble centerZ, GLdouble upX, GLdouble upY, GLdouble upZ);
Οι συντεταγμένες *eyeX,eyeY,eyeZ*, καθορίζουν τη θέση του παρατηρητή ως προς το σύστημα συντεταγμένων σκηνής. Ο προσανατολισμός της κάμερας καθορίζεται από το διάνυσμα με αρχή το σημείο (*eyeX,eyeY,eyeZ*) και πέρας το σημείο (*centerX,centerY,centerZ*). Οι τιμές (*upX,upY,upZ*) καθορίζουν τον προσανατολισμό του διανύσματος άνω κατεύθυνσης.

Με τη χρήση της εντολής **gluLookAt**, ο προγραμματιστής μπορεί να καθορίσει με ευκολία τη θέση και τον προσανατολισμό παρατήρησης. Με αυτό τον τρόπο, η OpenGL απαλάσσει τον προγραμματιστή από τον ορισμό περίπλοκων μετασχηματισμών αλλαγής συστήματος συντεταγμένων.

Δεδομένου ότι η **gluLookAt** επενεργεί στο μητρώο μετασχηματισμού μοντέλου, πρέπει να μεταβούμε στην κατάσταση τροποποίησής του πριν την εκτέλεση της εντολής, δίνοντας στην **glMatrixMode** την παράμετρο **GL_MODELVIEW**:

glMatrixMode(GL_MODELVIEW) ;

Η **gluLookAt** ουσιαστικά πολαπλασιάζει το μητρώο μετασχηματισμού οπτικής γωνίας που ορίστηκε παραπάνω με το τρέχον μητρώο μετασχηματισμού μοντέλου από δεξιά. Επομένως είναι σημαντικό η εντολή **gluLookAt** και οι μετέπειτα εντολές μετασχηματισμού μοντέλου να εκτελεστούν με τη σωστή διαδοχή. Δεδομένου ότι ο

μετασχηματισμός αλλαγής οπτικής γωνίας εφαρμόζεται στα σημεία της σκηνής αφού η τελευταία συντεθεί, η εντολή **gluLookAt** θα πρέπει να δηλωθεί μετά την εντολή αρχικοποίησης του μητρώου μετασχηματισμού μοντέλου και πριν από οποιοδήποτε μετασχηματισμό μοντέλου.

3.6.7.3 Παράλληλη προβολή

Την απλούστερη μορφή προβολής αποτελεί η παράλληλη ή ορθογραφική προβολή. Στην προβολή αυτή, τα σημεία της σκηνής προβάλλονται στο επίπεδο προβολής ακολουθώντας δέσμες κάθετες προς το επίπεδο προβολής. Κάθε σημείο δηλαδή προβάλλεται στην τομή της δέσμης του και του επιπέδου προβολής.

Στην OpenGL, ο ορισμός της παράλληλης προβολής στις τρεις διαστάσεις συνδυάζεται με την δήλωση των επιπέδων αποκοπής χρησιμοποιώντας την εντολή **glOrtho**:

void glOrtho(GLdouble xwmin, GLdouble xwmax, GLdouble ywmin, GLdouble ywmax, GLdouble dnear, GLdouble dfar);

xwmin, xwmax, ywmin και ywmax: το αριστερό, το δεξιό, το κάτω και το άνω επίπεδο αποκοπής αντίστοιχα.

dnear και dfar: Οι συντεταγμένες του εγγύς και μακρινού επιπέδου αποκοπής δίνονται ως θετικοί αριθμοί.

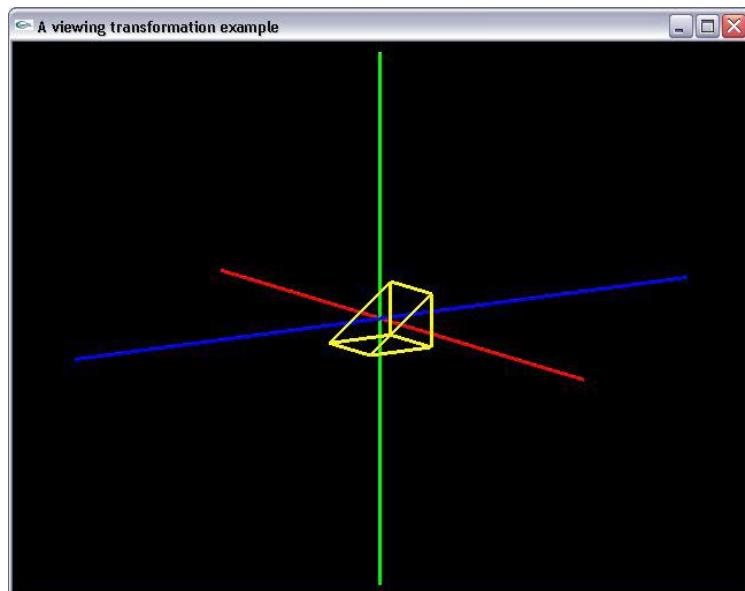
Παράδειγμα: Μετασχηματισμός οπτικής γωνίας και παράλληλη προβολή

```
#include <glut.h>
GLfloat eyeX,eyeY,eyeZ;
GLfloat toX,toY,toZ;
GLfloat viewUpX, viewUpY, viewUpZ;
void display()
{
    glClearColor(0,0,0,0);
    glClear(GL_COLOR_BUFFER_BIT);
    // Σχεδιάζοντας το x αξονα σε κοκκινο
    glColor3f(1,0,0);
    glBegin(GL_LINES);
    glVertex3f(-100,0,0);
    glVertex3f(100,0,0);
    glEnd();
    //Σχεδιάζοντας το x αξονα σε πρασινο
    glColor3f(0,1,0);
    glBegin(GL_LINES);
    glVertex3f(0,-100,0);
    glVertex3f(0,100,0);
    glEnd();
    // Σχεδιάζοντας το x αξονα σε μπλε
    glColor3f(0,0,1);
    glBegin(GL_LINES);
    glVertex3f(0,0,-100);
    glVertex3f(0,0,100);
    glEnd();
    //Σχεδιάζοντας ένα κίτρινο σχήμα
    glColor3f(1,1,0);
    glBegin(GL_LINE_LOOP);
    glVertex3f(-10,-10,10);
    glVertex3f(10,-10,10);
    glVertex3f(10,-10,-10);
    glVertex3f(-10,-10,-10);
    glEnd();
```

```

glBegin(GL_LINE_LOOP);
glVertex3f(-10,-10,10);
glVertex3f(10,-10,10);
glVertex3f(10,10,-10);
glVertex3f(-10,10,-10);
glEnd();
glBegin(GL_LINE_LOOP);
glVertex3f(-10,-10,-10);
glVertex3f(10,-10,-10);
glVertex3f(10,10,-10);
glVertex3f(-10,10,-10);
glEnd();
glFlush();
}
int main(int argc, char **argv)
{
glutInit(&argc, argv);
glutInitWindowPosition(50,50);
glutInitWindowSize(640,480);
glutCreateWindow("A viewing transformation example");
glMatrixMode(GL_PROJECTION);
glOrtho(-100,100,-100,100,-100,100);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glMatrixMode(GL_MODELVIEW);
eyeX=30;
eyeY=10;
eyeZ=20;
toX=0;
toY=0;
toZ=0;
viewUpX=0;
viewUpY=1;
viewUpZ=0;
gluLookAt(eyeX,eyeY,eyeZ,toX,toY,toZ,viewUpX,viewUpY,viewUpZ);
glutDisplayFunc(display);
glutMainLoop();
return 0;
}

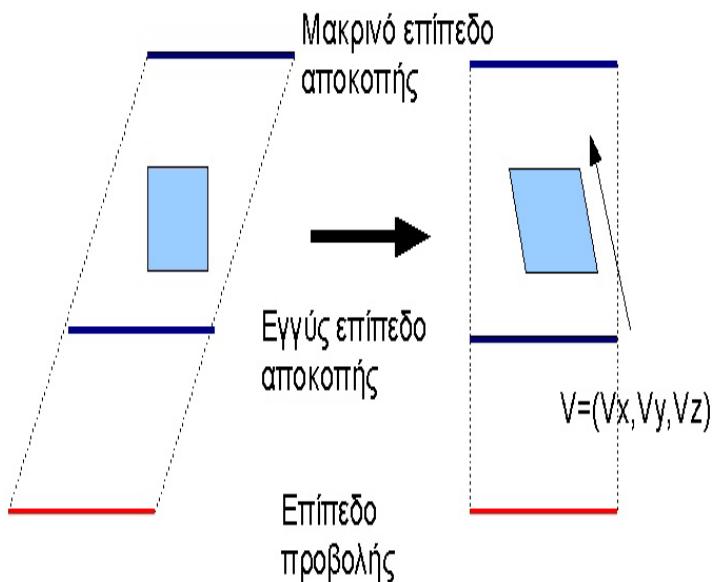
```



Σχήμα 3.22 Παράλληλη προβολή

3.6.7.3.1 Πλάγια παράλληλη προβολή

Η πλάγια παράλληλη προβολή μπορεί να εκφραστεί ισοδύναμα ως μια παράλληλη προβολή η οποία εφαρμόζεται αφού προηγηθεί στη σκηνή μετασχηματισμός κλίσης. Ουσιαστικά, στην πλάγια παράλληλη προβολή, απομονώνουμε το περιεχόμενο της σκηνής που περικλείεται εντός ενός πλαγίου παραλληλεπιπέδου (Σχήμα παρακάτω). Με το μετασχηματισμό κλίσης μεταφέρουμε αυτό το τμήμα της σκηνής εντός των ορίων ενός ορθογωνίου παραλληλεπιπέδου και κατόπιν εφαρμόζουμε τρισδιάστατη αποκοπή. Στην OpenGL, ο ορισμός ενός μητρώου πλάγιας παράλληλης προβολής είναι εφικτός με την άμεση ανάθεση τιμών στο μητρώο προβολής χρησιμοποιώντας τις εντολές `glLoadMatrix` και `glMultMatrix`.



Σχήμα 3.23 Ορισμός πλάγιας παράλληλης προβολής

Παράδειγμα: Πλάγια παράλληλη προβολή

```
#include <glut.h>
GLfloat *X;
GLfloat *Y;
GLfloat *Z;
//Shearing vector components
GLfloat vx,vy,vz;
GLdouble xwmin, xwmax, ywmin, ywmax, znear, zfar;
GLfloat *shearingMatrix;
void DrawCube(GLfloat *X, GLfloat *Y, GLfloat *Z)
{
    glBegin(GL_LINE_LOOP);
    glVertex3f(X[0],Y[0],Z[0]);
    glVertex3f(X[1],Y[1],Z[1]);
    glVertex3f(X[2],Y[2],Z[2]);
    glVertex3f(X[3],Y[3],Z[3]);
    glEnd();
    glBegin(GL_LINE_LOOP);
    glVertex3f(X[4],Y[4],Z[4]);
```

```

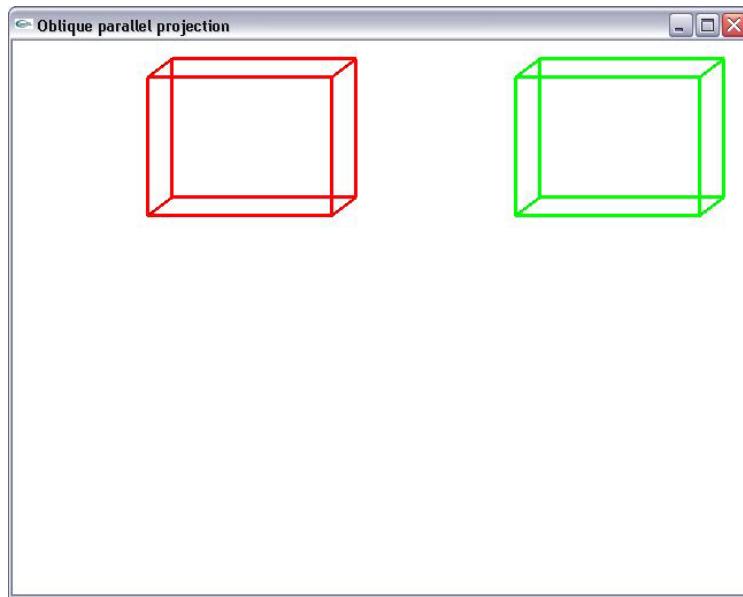
glVertex3f(X[5],Y[5],Z[5]);
glVertex3f(X[6],Y[6],Z[6]);
glVertex3f(X[7],Y[7],Z[7]);
glEnd();
glBegin(GL_LINES);
glVertex3f(X[0],Y[0],Z[0]);
glVertex3f(X[4],Y[4],Z[4]);
glVertex3f(X[1],Y[1],Z[1]);
glVertex3f(X[5],Y[5],Z[5]);
glVertex3f(X[2],Y[2],Z[2]);
glVertex3f(X[6],Y[6],Z[6]);
glVertex3f(X[3],Y[3],Z[3]);
glVertex3f(X[7],Y[7],Z[7]);
glEnd();
}
void display()
{
glClear(GL_COLOR_BUFFER_BIT);
X[0]=-5; Y[0]=25; Z[0]=-10;
X[1]=-5; Y[1]=25; Z[1]=-20;
X[2]=-20; Y[2]=25; Z[2]=-20;
X[3]=-20; Y[3]=25; Z[3]=-10;
X[4]=-5; Y[4]=10; Z[4]=-10;
X[5]=-5; Y[5]=10; Z[5]=-20;
X[6]=-20; Y[6]=10; Z[6]=-20;
X[7]=-20; Y[7]=10; Z[7]=-10;
glColor3f(1,0,0);
DrawCube(X,Y,Z);
X[0]=10; Y[0]=25; Z[0]=-10;
X[1]=10; Y[1]=25; Z[1]=-20;
X[2]=25; Y[2]=25; Z[2]=-20;
X[3]=25; Y[3]=25; Z[3]=-10;
X[4]=10; Y[4]=10; Z[4]=-10;
X[5]=10; Y[5]=10; Z[5]=-20;
X[6]=25; Y[6]=10; Z[6]=-20;
X[7]=25; Y[7]=10; Z[7]=-10;
glColor3f(0,1,0);
DrawCube(X,Y,Z);
glFlush();
}
int main(int argc,char **argv)
{
X=new GLfloat[8];
Y=new GLfloat[8];
Z=new GLfloat[8];
124
vx=0.2;
vy=0.2;
vz=1;
xwmin=-30;
xwmax=30;
ywmin=-30;
ywmax=30;
znear=-5;
zfar=-30;
glutInit(&argc,argv);
glutInitWindowPosition(50,50);
glutInitWindowSize(640,480);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutCreateWindow("Oblique parallel projection");

```

```

glClearColor(1,1,1,0);
glMatrixMode(GL_MODELVIEW);
gluLookAt(0,0,0,0,-1,0,1,0);
glMatrixMode(GL_PROJECTION);
glOrtho(xwmin,xwmax,ywmin,ywmax,-znear,-zfar);
shearingMatrix=new GLfloat[16];
shearingMatrix[0]=1;
shearingMatrix[1]=0;
shearingMatrix[2]=0;
shearingMatrix[3]=0;
shearingMatrix[4]=0;
shearingMatrix[5]=1;
shearingMatrix[6]=0;
shearingMatrix[7]=0;
shearingMatrix[8]=-vx/vz;
shearingMatrix[9]=-vy/vz;
shearingMatrix[10]=1;
shearingMatrix[11]=0;
shearingMatrix[12]=znear*vx/vz; shearingMatrix[13]=znear*vy/vz;
shearingMatrix[14]=0;
shearingMatrix[15]=1;
glMultMatrixf(shearingMatrix);
glutDisplayFunc(display);
glutMainLoop();
return 0;
}

```



Σχήμα 3.24 Πλάγια παράλληλη προβολή

3.6.8 Απόδοση τρισδιάστατων σκηνών – Κινούμενα γραφικά

3.6.8.1 Ενταμιευτής βάθους

Προκειμένου να αξιοποιηθεί ο έλεγχος τιμών βάθους των επιφανειών, θα πρέπει η δυνατότητα αυτή να ενεργοποιηθεί από τον προγραμματιστή, δίνοντας στην εντολή **glEnable** το όρισμα **GL_DEPTH_TEST**.
glEnable(GL_DEPTH_TEST) ;

Επίσης δηλώνεται στην **glutInitDisplayMode** τη χρήση ενταμιευτή βάθους δίνοντας το όρισμα **GL_DEPTH**.

glutInitDisplayMode(GL_DEPTH) ;

Επιπλέον, θα πρέπει, στη συνάρτηση **display**, πριν το σχεδιασμό ή επανασχεδιασμό ενός καρέ, να αρχικοποιείται ο ενταμιευτής τιμών βάθους με την εντολή **glClear**, όπως ακριβώς αρχικοποιείται και ο ενταμιευτής χρωματικών τιμών:

glClear(GL_COLOR_BUFFER_BIT) ;

glClear(GL_DEPTH_BUFFER_BIT) ;

ή με μία εντολή καθαρισμού

glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT) ;

Η αρχικοποίηση του θέτει ως προκαθορισμένη τιμή στα στοιχεία του τη μονάδα (η οποία είναι και η μέγιστη τιμή βάθους που αποδίδεται σε κανονικοποιημένες συντεταγμένες). Μπορούμε να μεταβάλουμε αυτή την αρχική τιμή με τη χρήση της εντολής **glClearDepth**:

void glClearDepth(GLdouble maxDepth) ;

maxDepth: ορίζει τη μέγιστη τιμή βάθους που θα χρησιμοποιείται κατά τον καθαρισμό του ενταμιευτή βάθους.

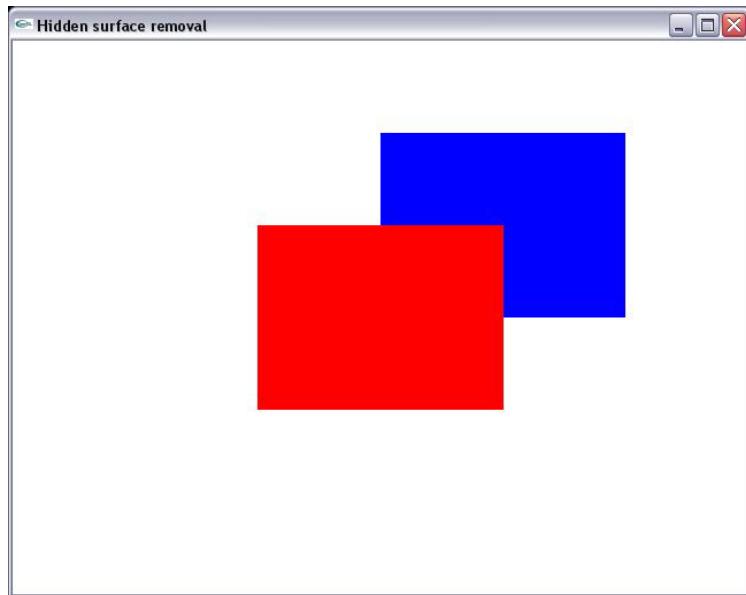
Παράδειγμα: Καταστολή κρυμμένων επιφανειών

```
#include <glut.h>
GLdouble eyeX;
GLdouble eyeY;
GLdouble eyeZ;
GLdouble toX;
GLdouble toY;
GLdouble toZ;
void display()
{
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    //Σχεδιασμός δύο πολύγωνων με επικαλυπτόμενες προβολές
    // Το πλησιέστερο πολύγωνο σχεδιάζεται σε κόκκινο
    glColor3f(1,0,0);
    glBegin(GL_POLYGON);
    glVertex3f(-10,-10,-10);
    glVertex3f(10,-10,-10);
    glVertex3f(10,10,-10);
    glVertex3f(-10,10,-10);
    glEnd();
    //Το πολύγωνο που είναι πιο μακριά σημειώνεται ως μπλε
    glColor3f(0,0,1);
    glBegin(GL_POLYGON);
    glVertex3f(0,0,-20);
    glVertex3f(20,0,-20);
    glVertex3f(20,20,-20);
    glVertex3f(0,20,-20);
    glEnd();
    glFlush();
}
int main (int argc,char **argv)
{
    eyeX=0;
    eyeY=0;
    eyeZ=0;
    toX=0;
    toY=0;
```

```

toZ=-1;
glutInit(&argc,argv);
glutInitWindowPosition(50,50);
glutInitWindowSize(640,480);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutCreateWindow("Hidden surface removal");
glClearColor(1,1,1,0);
glEnable(GL_DEPTH_TEST);
glMatrixMode(GL_MODELVIEW);
gluLookAt(eyeX,eyeY,eyeZ,toX,toY,toZ,0,1,0);
glMatrixMode(GL_PROJECTION);
glOrtho(-30,30,-30,30,5,50);
glutDisplayFunc(display);
glutMainLoop();
return 0;
}

```



Σχήμα 3.25 Καταστολή κρυμμένων επιφανειών

3.6.8.2 Τρισδιάστατες επιφάνειες

Οι εντολές της βιβλιοθήκης GLUT έχουν δύο παραλλαγές: η πρώτη εμφανίζει το περίγραμμα (**wireframe**) των πολυγώνων που προσεγγίζουν την επιφάνεια και ξεκινούν με το πρόθεμα **glutWire**. Η δεύτερη παραλλαγή των εντολών σχεδιάζει τα στοιχειώδη πολύγωνα της επιφάνειας συμπαγή. Οι εντολές αυτές ξεκινούν με το πρόθεμα **glutSolid**.

3.6.8.2.1 Κύβος

Ο κύβος σχεδιάζεται με τις εντολές **glutWireCube** και **glutSolidCube** της βιβλιοθήκης GLUT.

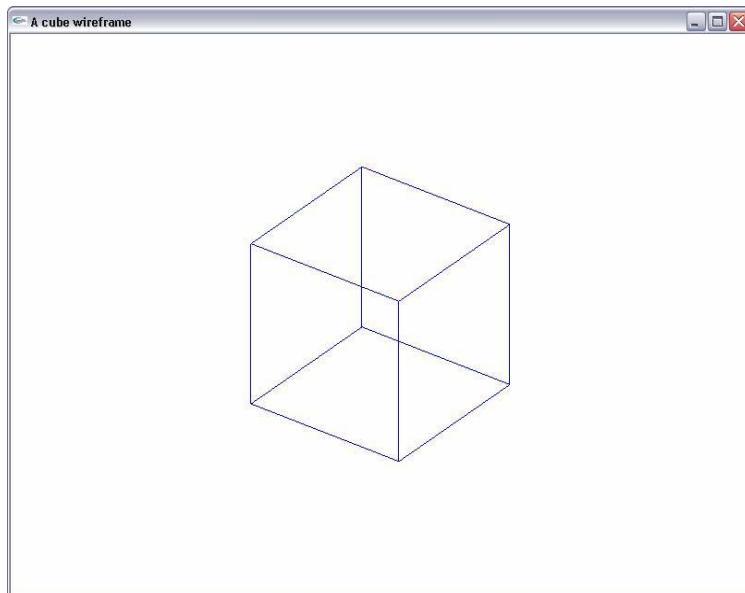
void glutWireCube (GLdouble edgeLength); για τη σχεδίαση του περιγράμματος κύβου

void glutSolidCube (GLdouble edgeLength); για τη σχεδίαση συμπαγούς κυβικού σχήματος,

Η παράμετρος **edgeLength** καθορίζει το μήκος των ακμών.

Παράδειγμα: Σχεδίαση κυβικού πλέγματος

```
#include <glut.h>
void display()
{
    glColor3f(0,0,1); //ορίζεται το τρέχον χρώμα σχεδιάσης.
    glClearColor(1,1,1,0); // καθορίζει το χρώμα που χρησιμοποιείται κάθε φορά που
    εκτελείται εντολή καθαρισμού της οθόνης.
    glClear(GL_COLOR_BUFFER_BIT); //καθαρισμός ενταμιευτη χρώματος
    glutWireCube(40); //σχεδιασμός κύβου με περίγραμμα μόνο, το 40 είναι το μήκος των
    κορυφών.
    glFlush();
}
int main(int argc, char **argv)
{
    glutInit(&argc, argv); // Ενεργοποιεί τη χρήση των εντολών της βιβλιοθήκης GLUT.
    glutInitWindowPosition(50,50); // Δηλώνει τη θέση όπου θα εμφανιστεί το παράθυρο
    την οθόνη
    glutInitWindowSize(800,600); // ορίζει το μέγεθος του παραθύρου σε pixels
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB); Καθορίζει ρυθμίσεις απεικόνισης
    (μμοντέλο ενταμίευσης, χρωματικό μμοντέλο κ.λπ.), εδώ χρησιμοποιούμε μόνο
    ενταμιευτή και το χρωματικό μοντέλο RGB
    glutCreateWindow("A cube wireframe"); // δημιουργεί το παράθυρο και του δίνει τίτλο
    glMatrixMode(GL_PROJECTION); // Επιλέγει ποιο μητρώο θα τροποποιήσουμε,
    προβολής ή μετασχηματισμού μοντέλου, εδώ μητρώο προβολής.
    glOrtho(-80,80,-60,60,0,100); // Δήλωση παράλληλης προβολής
    glMatrixMode(GL_MODELVIEW); // Επιλέγει ποιο μητρώο θα τροποποιήσουμε,
    προβολής ή μετασχηματισμού μοντέλου, εδώ μητρώο μετασχηματισμού μοντέλου
    gluLookAt(-30,-30,40,0,0,0,0,1,0); //ο μετασχηματισμός οπτικής γωνίας
    glutDisplayFunc(display); // Δηλώνει την συνάρτηση που θα εκτελείται κάθε φορά που
    απαιτείται σχεδιασμός της σκηνής
    glutMainLoop(); // Ενεργοποιεί τον κύκλο ακρόασης γεγονότων
    return 0;
}
```



Σχήμα 3.26 Σχεδίαση κυβικού πλέγματος

3.6.8.2.2 Σφαίρα

Μια σφαιρική επιφάνεια ακτίνας r σχηματίζεται από όλα τα σημεία της σκηνής που απέχουν απόσταση r από το κέντρο της σφαίρας. Αν το κέντρο της σφαίρας βρίσκεται στην αρχή των αξόνων οι εξισώσεις ορισμού της σφαίρας στο καρτεσιανό σύστημα είναι:

$$x^2 + y^2 + z^2 = r^2$$

Η εντολή για τη σχεδίαση σφαιρών:

void glutWireSphere(GLdouble radius, GLint slices, GLint stacks); για τη σχεδίαση σφαιρικού πλέγματος

void glutSolidSphere(GLdouble radius, GLint slices, GLint stacks); για τη σχεδιάση μιας συμπαγούς σφαιρικής επιφανείας.

Παράμετροι

radius: δηλώνει την ακτίνα της σφαίρας.

Slices: δηλώνει το πλήθος των κατακορύφων υποδιαιρέσεων (μεσημβρινοί)

Stacks: δηλώνει το πλήθος των οριζοντίων υποδιαιρέσεων (γεωγραφικά πλάτη)

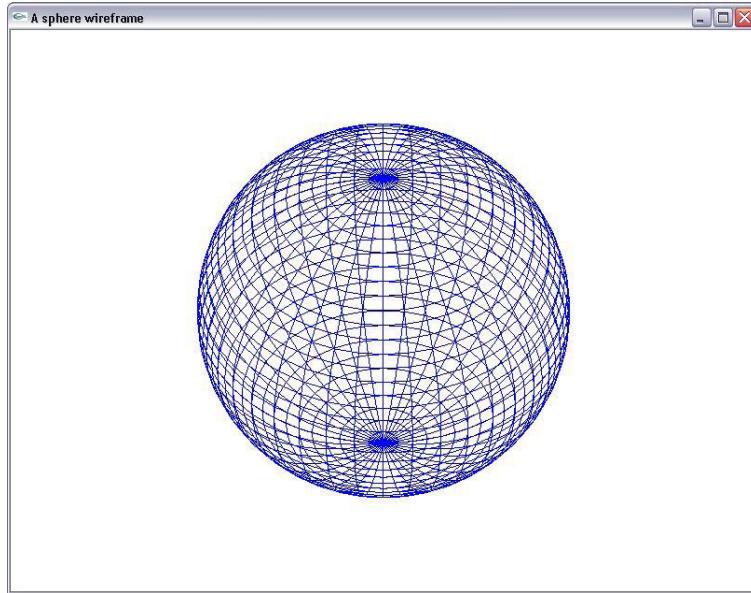
Παράδειγμα: Σχεδίαση περιγράμματος σφαίρας

```
#include <glut.h>
void display()
{
    glColor3f(0,0,1);
    glClearColor(1,1,1,0);
    glClear(GL_COLOR_BUFFER_BIT);
    glutWireSphere(40,40,40);
    glFlush();
}
int main(int argc, char **argv)
{
    glutInit(&argc,argv);
    glutInitWindowPosition(50,50);
    glutInitWindowSize(800,600);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
```

```

glutCreateWindow("A sphere wireframe");
glMatrixMode(GL_PROJECTION);
glOrtho(-80,80,-60,60,0,100);
glMatrixMode(GL_MODELVIEW);
gluLookAt(0,-40,40,0,0,0,1,0);
glutDisplayFunc(display);
glutMainLoop();
return 0;
}

```



Σχήμα 3.27 Σχεδίαση σφαιρικού πλέγματος

3.6.8.2.3 Κώνος

Η εντολή για τη σχεδίαση του κώνου είναι:

glutWireCone(GLdouble base, GLdouble height, GLint slices, GLint stacks); για τη σχεδίαση κωνικού περιγράμματος.

glutSolidCone (GLdouble base, GLdouble height, GLint slices, GLint stacks); για τη σχεδίαση μιας συμπαγούς κωνικής επιφανείας.

Παράμετροι

Base: ορίζει την ακτίνα της βάσης του κώνου

Height: ορίζει το ύψος του.

Slices: καθορίζει το πλήθος των οριζόντιων υποδιαιρέσεων που χρησιμοποιούνται για την προσέγγιση του κώνου (μετρούνται διατρέχοντας την περιφέρεια μιας διατομής του κώνου)

Stacks: καθορίζει το πλήθος των κατακόρυφων υποδιαιρέσεων που χρησιμοποιούνται για την προσέγγιση του κώνου (μετρούνται διατρέχοντας τον κώνο από τη βάση ως την κορυφή του).

Παράδειγμα: Σχεδιάση περιγράμματος κωνικής επιφανείας

```

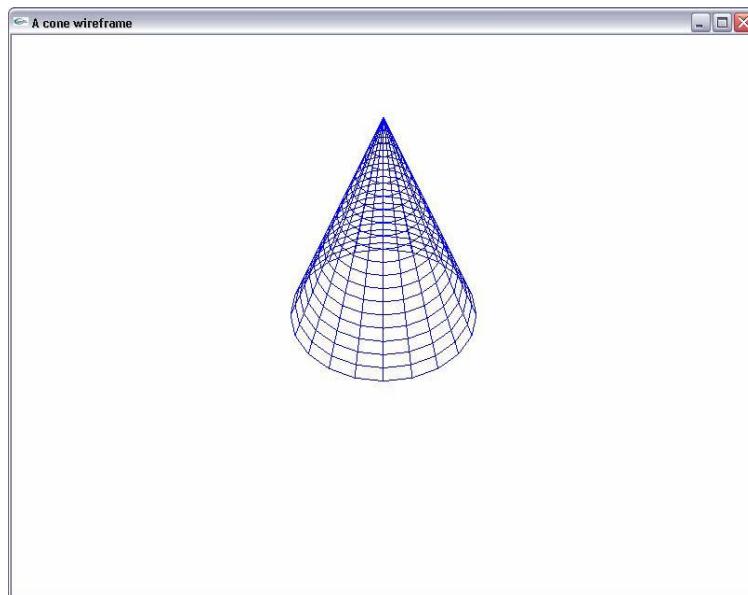
#include <glut.h>
void display()
{
    glColor3f(0,0,1);
    glClearColor(1,1,1,0);

```

```

glClear(GL_COLOR_BUFFER_BIT);
glutWireCone(20,60,20,20);
glFlush();
}
int main(int argc, char **argv)
{
glutInit(&argc,argv);
glutInitWindowPosition(50,50);
glutInitWindowSize(800,600);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutCreateWindow("A cone wireframe");
glMatrixMode(GL_PROJECTION);
glOrtho(-80,80,-60,60,0,100);
glMatrixMode(GL_MODELVIEW);
gluLookAt(0,-40,40,0,0,0,0,1,0);
glutDisplayFunc(display);
glutMainLoop();
return 0;
}

```



Σχήμα 3.28 Κωνικού πλέγματος

3.6.8.2.4 Κύλινδρος

Εντολή για την σχεδίαση κύκλου:

**gluCylinder(GLUquadric *qObj, GLdouble baseRadius, GLdouble topRadius,
GLdouble height, GLdouble slices, GLdouble stacks);**

qObj: δείκτης στο αντικείμενο της κυλινδρικής επιφάνειας.

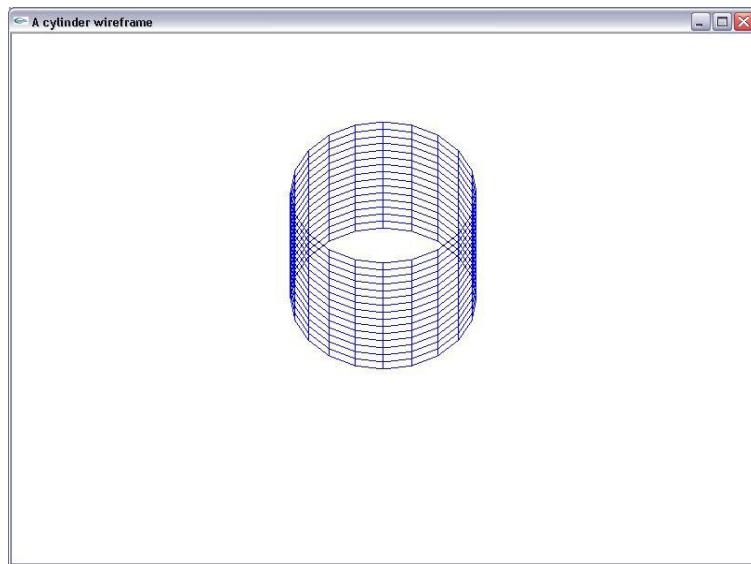
baseRadius και **topRadius**: ακτίνα για την κυκλική επιφάνεια της βάσης και της κορυφής του κυλίνδρου αντίστοιχα.

height: εκφράζει το ύψος του κυλίνδρου.

slices και **stacks**: ορίζουν το πλήθος των οριζόντιων και κάθετων υποδιαιρέσεων που προσεγγίζουν την επιφάνεια του κυλίνδρου.

Παράδειγμα: Σχεδίαση κυλινδρικού πλέγματος

```
#include <glut.h>
GLUquadric *cylinder;
void display()
{
    glColor3f(0,0,1);
    glClearColor(1,1,1,0);
    glClear(GL_COLOR_BUFFER_BIT);
    gluQuadricDrawStyle(cylinder,GLU_LINE); //cylinder το αντικείμενο που θα σχεδιαστεί, GLU_LINE μόνο το περίγραμμα
    gluCylinder(cylinder,20,20,40,20,15); //δεικτης στο αντικείμενο, ακτίνες της βάσης και της κορυφής, ύψος κυλίνδρου, το πλήθος των οριζοντίων και κάθετων υποδιαιρέσεων
    glFlush();
}
int main(int argc, char **argv)
{
    glutInit(&argc,argv);
    glutInitWindowPosition(50,50);
    glutInitWindowSize(800,600);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutCreateWindow("A cylinder wireframe");
    glMatrixMode(GL_PROJECTION);
    glOrtho(-80,80,-60,60,0,100);
    glMatrixMode(GL_MODELVIEW);
    gluLookAt(0,-30,40,0,0,0,0,1,0);
    cylinder=gluNewQuadric();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0; }
```



Σχήμα 3.29 Σχεδίαση Κυλινδρικού Πλέγματος

3.6.8.2.5 Κυκλικός δίσκος – Δακτύλιος

Ο κυκλικός δίσκος, σε αντίθεση με τον κύλινδρο, είναι επίπεδος. Η εντολή σχεδίασης είναι **gluDisk**. Με την ίδια εντολή, σχεδιάζεται και ο δακτύλιος, η επιφάνεια του οποίου εκτείνεται μεταξύ μιας εσωτερικής και μιας εξωτερικής ακτίνας.

void gluDisk (GLUquadric *quadObject, GLdouble innerRadius, GLdouble outerRadius, GLint slices, GLint loops);

quadObject: εκφράζει το αντικείμενο στο οποίο αντιστοιχίζουμε την επιφάνεια.

innerRadius: δηλώνει την εσωτερική ακτίνα από την οποία ξεκινάει ο σχηματισμός του δακτυλίου.

innerRadius=0: σχεδιάζουμε έναν κυκλικό δίσκο.

outerRadius: ορίζει την ακτίνα μέχρι την οποία εκτείνεται ο δίσκος ή ο δακτύλιος.

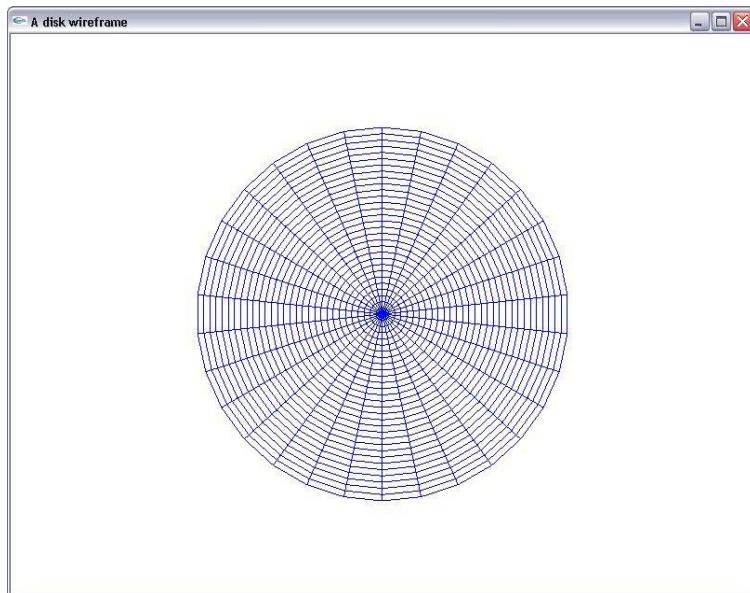
Slices: ορίζει το πλήθος των υποδιαιρέσεων (“φετών”) που μετράμε διαγράφοντας κυκλική πορεία σταθερής ακτίνας.

Loops: ορίζει το πλήθος των ομόκεντρων κύκλων που χρησιμοποιούνται για την προσέγγιση της επιφάνειας και που μετρούνται αναχωρώντας από το κέντρο του κύκλου και με κίνηση προς την την περιφέρεια.

Ο κυκλικός δίσκος σχεδιάζεται στο επίπεδο XY του συστήματος συντεταγμένων σκηνής και το κέντρο του τοποθετείται στην αρχή των αξόνων.

Παράδειγμα: Σχεδίαση πλέγματος κυκλικού δίσκου

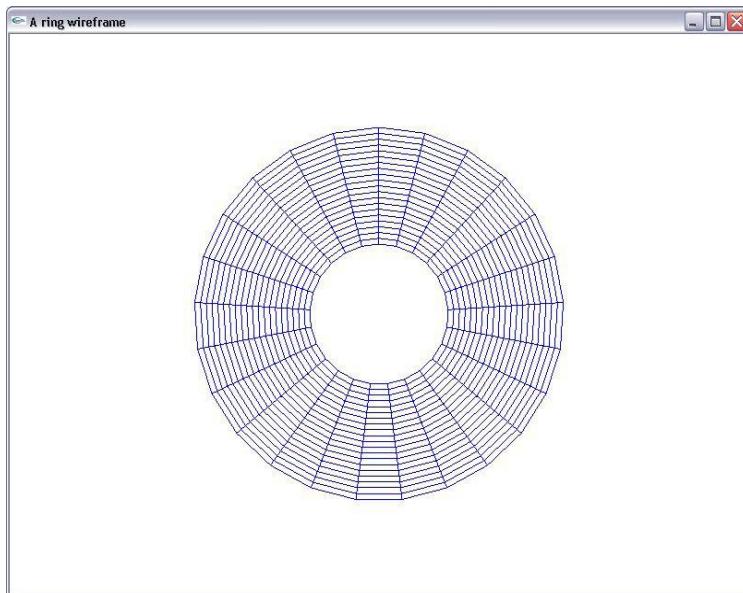
```
#include <glut.h>
GLUquadric *disk;
void display()
{
    glColor3f(0,0,1);
    glClearColor(1,1,1,0);
    glClear(GL_COLOR_BUFFER_BIT);
    gluQuadricDrawStyle(disk,GLU_LINE); // δυο παραμετροί, 1.το αντικείμενο σχεδιασμού, 2. Η μορφη
    σχεδιασμού, εδώ είναι μονο το περιγραμμα
    gluDisk(disk,0,40,30,30); // δημιουργια του δίσκου
    glFlush();
}
int main(int argc, char **argv)
{
    glutInit(&argc,argv);
    glutInitWindowPosition(50,50);
    glutInitWindowSize(800,600);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutCreateWindow("A disk wireframe");
    glMatrixMode(GL_PROJECTION);
    glOrtho(-80,80,-60,60,0,100);
    disk=gluNewQuadric();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```



Σχήμα 3.30 Σχεδίαση πλέγματος Κυκλικού δίσκου

Παράδειγμα: Σχεδίαση πλέγματος κυκλικού δακτυλίου

```
#include <glut.h>
GLUquadric *disk;
void display()
{
    glColor3f(0,0,1);
    glClearColor(1,1,1,0);
    glClear(GL_COLOR_BUFFER_BIT);
    gluQuadricDrawStyle(disk,GLU_LINE);
    gluDisk(disk,15,40,25,20);
    glFlush();
}
int main(int argc, char **argv)
{
    glutInit(&argc,argv);
    glutInitWindowPosition(50,50);
    glutInitWindowSize(800,600);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutCreateWindow("A ring wireframe");
    glMatrixMode(GL_PROJECTION);
    glOrtho(-80,80,-60,60,0,100);
    disk=gluNewQuadric();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```



Σχήμα 3.31 Σχεδίαση πλέγματος Κυκλικού δακτυλίου

3.6.8.2.6 Κυκλικός τομέας – Τομέας δακτυλίου

Αντί για ένα συμπαγή δίσκο ή δακτύλιο, έχουμε επίσης τη δυνατότητα σχεδίασης ενός μόνο γωνιακού τους τμήματος, δηλαδή ενός κυκλικού τομέα ή ενός τομέα δακτυλίου. Αυτό γίνεται με την εντολή **gluPartialDisk**:

gluPartialDisk (GLUquadric *qObj, GLdouble innerRadius, GLdouble outerRadius, GLdouble slices, GLdouble loops, GLdouble startAngle, GLdouble sweepAngle);

startAngle: καθορίζει τη γωνία από την οποία ξεκινάει ο σχεδιασμός του σχήματος.

sweepAngle: καθορίζει το γωνιακό εύρος του δακτυλίου. Η γωνιακή θέση αντιστοιχεί στην κατεύθυνση προς τα πάνω (προς το θετικό ημιάξονα y) και η τιμή της γωνιακής θέσης αυξάνεται κατά την αρνητική φορά. 0

Οι υπόλοιπες παράμετροι είναι ίδιες με του κυκλικού δακτυλίου.

Οι τομείς σχεδιάζονται επί του επιπέδου XY και με το κέντρο τους στην αρχή του συστήματος συντεταγμένων σκηνής.

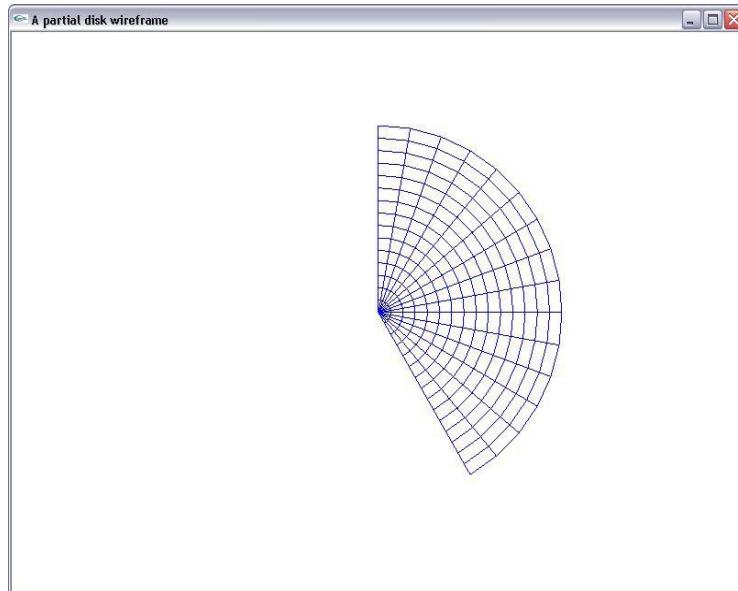
Παράδειγμα: Σχεδίαση πλέγματος κυκλικού τομέα

```
#include <glut.h>
GLUquadric *disk;
void display()
{
    glColor3f(0,0,1);
    glClearColor(1,1,1,0);
    glClear(GL_COLOR_BUFFER_BIT);
    gluQuadricDrawStyle(disk,GLU_LINE);
    gluPartialDisk(disk,0,40,15,15,0,150);
    glFlush();
}
int main(int argc, char **argv)
{
    glutInit(&argc,argv);
    glutInitWindowPosition(50,50);
    glutInitWindowSize(800,600);
```

```

glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutCreateWindow("A partial disk wireframe");
glMatrixMode(GL_PROJECTION);
glOrtho(-80,80,-60,60,0,100);
disk=gluNewQuadric();
glutDisplayFunc(display);
glutMainLoop();
return 0;
}

```



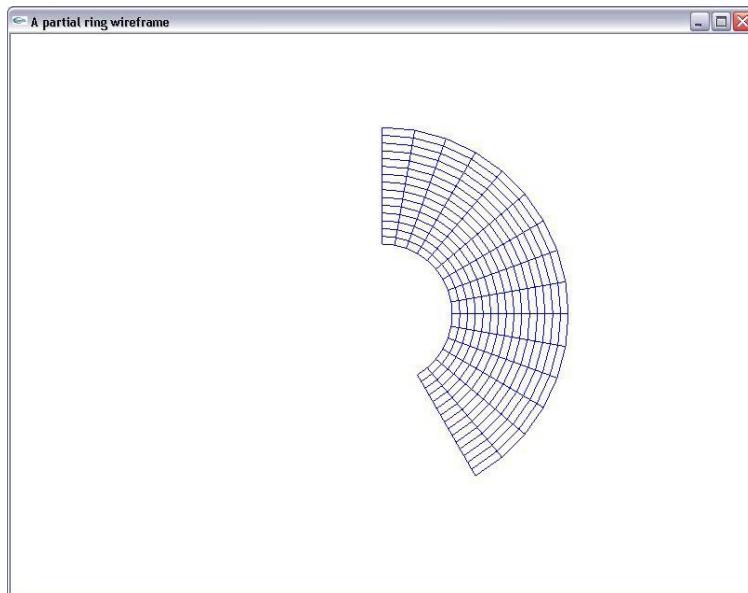
Σχήμα 3.32 Σχεδίαση πλέγματος κυκλικού τομέα

Παράδειγμα: Σχεδίαση πλέγματος τομέα δακτυλίου

```

#include <glut.h>
GLUquadric *disk;
void display()
{
    glColor3f(0,0,1);
    glClearColor(1,1,1,0);
    glClear(GL_COLOR_BUFFER_BIT);
    gluQuadricDrawStyle(disk,GLU_LINE);
    gluPartialDisk(disk,15,40,15,15,0,150);
    glFlush();
}
int main(int argc, char **argv)
{
    glutInit(&argc,argv);
    glutInitWindowPosition(50,50);
    glutInitWindowSize(800,600);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutCreateWindow("A partial ring wireframe");
    glMatrixMode(GL_PROJECTION);
    glOrtho(-80,80,-60,60,0,100);
    disk=gluNewQuadric();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}

```



Σχήμα 3.33 Σχεδίαση πλέγματος τομέα δακτυλίου

3.6.8.2.7 Τόρος

Ο τόρος είναι ένα τρισδιάστατο σχήμα παρόμοιο με το σχήμα κουλουριού. Προκύπτει εάν θεωρήσουμε έναν κύκλο που περιστρέφεται κατά 360 μοίρες στον τρισδιάστατο χώρο και ως προς άξονα περιστροφής που βρίσκεται στο ίδιο επίπεδο με τον κύκλο. Ανάλογα με την ακτίνα του κύκλου και την απόσταση του κέντρου του κύκλου από τον άξονα περιστροφής προκύπτουν παραλλαγές του τόρου.

Η εντολή δύο παραλλαγές:

void glutWireTorus(GLdouble innerRadius, GLdouble outerRadius, GLint sides, GLint rings);

σχεδιάζει το περίγραμμα των στοιχειωδών πολυγώνων που συνθέτουν την επιφάνεια του τόρου

void glutSolidTorus(GLdouble innerRadius, GLdouble outerRadius, GLint sides, GLint rings);

δημιουργεί τον τόρο με συμπαγείς πολυγωνικές επιφάνειες.

Οι παράμετροι

innerRadius: εκφράζει την ακτίνα της κυκλικής διατομής του τόρου.

outerRadius: είναι η απόσταση του κέντρου της διατομής του τόρου από τον άξονά του.

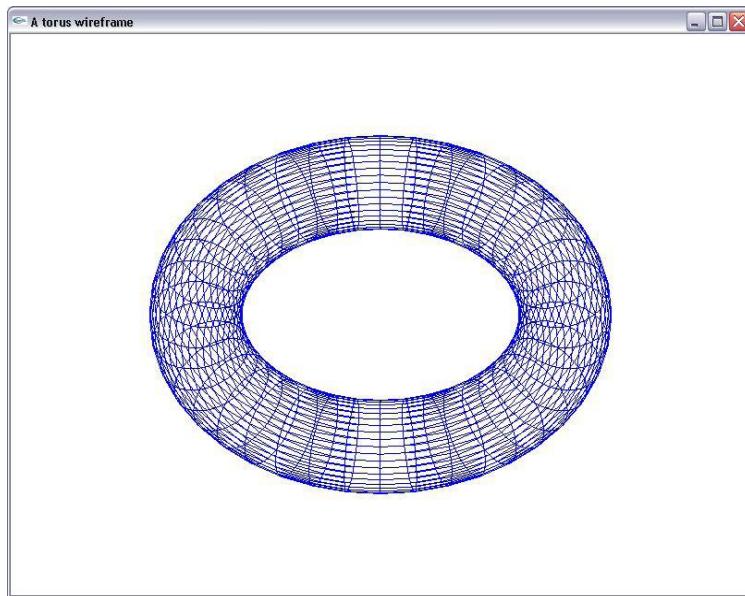
Sides: καθορίζει το πλήθος των υποδιαιρέσεων που προσεγγίζουν τα όρια της διατομής του τόρου (μετρούνται καθώς διατρέχουμε την περιφέρεια μιας κυκλικής διατομής του τόρου).

Rings: καθορίζει το πλήθος των κυκλικών διατομών που χρησιμοποιούμε για την προσέγγιση του τόρου (μετρούνται καθώς διατρέχουμε μια κλειστή διαδρομή κατά μήκος του τοροειδούς δακτυλίου).

Ο τόρος σχεδιάζεται θεωρώντας ως άξονά του τον άξονα Oz και θέτοντας το κέντρο του την αρχή του συστήματος συντεταγμένων σκηνής.

Παράδειγμα: Σχεδίαση τοροειδούς περιγράμματος

```
#include <glut.h>
void display()
{
    glColor3f(0,0,1);
    glClearColor(1,1,1,0);
    glClear(GL_COLOR_BUFFER_BIT);
    glutWireTorus(10,40,40,40);
    glFlush();
}
int main(int argc, char **argv)
{
    glutInit(&argc,argv);
    glutInitWindowPosition(50,50);
    glutInitWindowSize(800,600);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutCreateWindow("A torus wireframe");
    glMatrixMode(GL_PROJECTION);
    glOrtho(-80,80,-60,60,0,100);
    glMatrixMode(GL_MODELVIEW);
    gluLookAt(0,-40,40,0,0,0,0,1,0);
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```



Σχήμα 3.34 Σχεδίαση τοροειδούς περιγράμματος

3.6.8.3 Μίξη χρωμάτων

Κατά τη σχεδίαση επικαλυπτόμενων σχημάτων εάν οι συντεταγμένες των νέων σχημάτων επικαλύπτονται με τις συντεταγμένες προηγουμένων σχημάτων οι χρωματικές τιμές στις επικαλυπτόμενες θέσεις αντικαθίστανται με τις χρωματικές τιμές του νέου σχήματος. Ωστόσο στην περίπτωση επικαλυπτόμενων σχημάτων υπάρχει η δυνατότητα να αναμίχθουν οι χρωματικές τιμές τους και να παραχθεί ένας ενδιάμεσος

χρωματισμός στα κοινά σημεία τους. Με τον τρόπο αυτό μπορεί ο χρήστης όπως λ.χ. να προσομοιώσει διαφανείς ή ημιδιαφανείς επιφάνειες.

Για τη μίξη χρησιμοποιείται ευρέως το χρωματικό μοντέλο RGBA. Αυτό το μοντέλο ορίζεται για κάθε χρώμα τις τρεις συνιστώσες του και μία τέταρτη συνιστώσα τον συντελεστή alpha, ο οποίος χρησιμοποιείται ως συντελεστής μίξης.

Στη μίξη χρωμάτων ορίζονται **δύο στρώματα**: το **στρώμα προορισμού** (destination) και το **στρώμα πηγής** (source). Με τον όρο **στρώμα προορισμού** αναφέρεται στην υπάρχουσα χρωματική τιμή RGBA που είναι αποθηκευμένη στον ενταμιευτή χρωματικών τιμών. Η υπάρχουσα χρωματική τιμή σε κάθε σημείο του ενταμιευτή χρωμάτων είναι ίση, είτε με την τιμή του φόντου, είτε με τη χρωματική τιμή του τελευταίου αντικειμένου που σχεδιάστηκε. **Η χρωματική τιμή της πηγής** ταυτίζεται με το χρώμα του σχήματος που σκοπεύουμε να συνδυάσουμε με τα υπάρχοντα χρώματα του ενταμιευτή χρωμάτος.

Για τη μίξη χρωμάτων, αρχικά πρέπει να ενεργοποιηθεί η λειτουργία με την εντολή: **glEnable(GL_BLEND);**

Στη συνέχεια ορίζονται οι συντελεστές μίξης που αντιστοιχίζονται στο στρώμα πηγής και στο στρώμα προορισμού με την εντολή **glBlendFunc**:

void glBlendFunc(GLenum sFactor, GLenum dFactor);

sFactor: καθορίζει τους συντελεστές μίξης για το στρώμα πηγής.

dFactor: καθορίζει τους συντελεστές μίξης για το στρώμα προορισμού.

Καθορίζονται με τις εξής αριθμητικές σταθερές:

GL_ZERO: Θέτει τους συντελεστές μίξης (0,0,0,0) για το εκάστοτε στρώμα.

GL_ONE: Ορίζει τους συντελεστές μίξης (1,1,1,1) για το εκάστοτε στρώμα.

GL_SRC_ALPHA: Επιλέγουμε για συντελεστές μίξης τον εκάστοτε στρώματος, τη συνιστώσα alpha του χρώματος στο στρώμα πηγής. (As,As,As,As)

GL_DST_ALPHA: Επιλέγουμε ως συντελεστή μίξης για το εκάστοτε στρώμα τη συνιστώσα alpha του χρώματος στο στρώμα προορισμού. (Ad,Ad,Ad,Ad)

GL_ONE_MINUS_SRC_ALPHA: Επιλέγουμε ως συντελεστή μίξης για το εκάστοτε στρώμα το συμπλήρωμα της συνιστώσας As ως προς τη μονάδα (1-As,1-As,1-As,1-As).

GL_ONE_MINUS_DST_ALPHA: Επιλέγουμε ως συντελεστή μίξης για το εκάστοτε στρώμα το συμπλήρωμα της συνιστώσας Ad ως προς τη μονάδα. (1-Ad,1-Ad,1-Ad,1-Ad).

Η προκαθορισμένη τιμή για την παράμετρο **sFactor** είναι **GL_ONE** και για την παράμετρο **dFactor** **GL_ZERO**. Δηλαδή, στην αρχική κατάσταση το χρώμα πηγής αντικαθιστά το χρώμα προορισμού.

Με τη μίξη χρωμάτων υπάρχει η δυνατότητα να προσομοιώσουμε φαινόμενα διαφάνειας. Στην περίπτωση αυτή, η διαφάνεια μιας επιφάνειας καθορίζεται με τη χρήση της αλpha συνιστώσας της. Είναι σύνηθες να ορίζεται μια επιφάνεια ως πλήρως διαφανή για τιμή alpha ίση με 1 και πλήρως αδιαφανή για τιμή alpha ίση με 0.

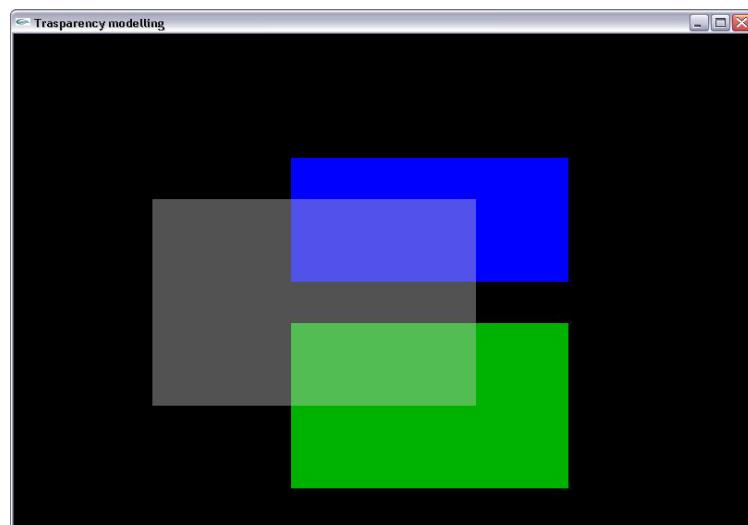
Παράδειγμα: Μοντελοποίηση διαφάνειας

```
#include <glut.h>
void init()
{
    glutInitWindowPosition(50,50);
    glutInitWindowSize(800,600);
    glutCreateWindow("Transparency modelling");
    //Χρησιμοποιώντας το χρωματικό μοντέλο RGBA
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGBA);
    glMatrixMode(GL_PROJECTION);
```

```

glOrtho(-40,40,-30,30,0,50);
glClearColor(0,0,0,0);
// Ενεργοποίηση ανάμειξη χρωμάτων
glEnable(GL_BLEND);
// ενεργοποιώντας την απομάκρυνση της κρυμμένης επιφάνειας
glEnable(GL_DEPTH_TEST);
// Μοντελοποίηση διαφάνειας: S=(1-As,1-As,1-As,1-As) D=(As,As,As,As)
glBlendFunc(GL_ONE_MINUS_SRC_ALPHA,GL_SRC_ALPHA);
}
void display()
{
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glBegin(GL_QUADS);
// Σχέδιαση ενός αδιαφανές μπλε ορθογώνιου
	glColor4f(0,0,1,0);
	glVertex3f(-10,0,-20);
	glVertex3f(20,0,-20);
	glVertex3f(20,15,-20);
	glVertex3f(-10,15,-20);
// Σχέδιαση ενός αδιαφανές πράσινου ορθογώνιου
	glColor4f(0,7,0,0);
	glVertex3f(-10,-5,-20);
	glVertex3f(20,-5,-20);
	glVertex3f(20,-25,-20);
	glVertex3f(-10,-25,-20);
// Σχέδιαζοντας ένα διαφανές γκρι ορθογώνιο (As=0.6)
	glColor4f(0.8,0.8,0.8,0.6);
	glVertex3f(-25,-15,-10);
	glVertex3f(10,-15,-10);
	glVertex3f(10,10,-10);
	glVertex3f(-25,10,-10);
glEnd();
glFlush();
}
int main (int argc, char **argv)
{
glutInit(&argc,argv);
init();
glutDisplayFunc(display);
glutMainLoop();
}

```



Σχήμα 3.35 Μοντελοποίηση διαφάνειας

3.6.8.4 Διπλή ενταμίευση

Για να αποφευχθεί το τρεμοπαίξιμο της σκηνής (flickering) και η υποβάθμιση της ποιότητας των κινούμενων γραφικών χρησιμοποιείται πάντα η τεχνική της διπλής ενταμίευσης.

Στη διπλή ενταμίευση, οι εφαρμογές αντί για έναν, χρησιμοποιούν **δύο ενταμιευτές** χρωματικών τιμών: τον **ενταμιευτή προσκηνίου** και τον **ενταμιευτή παρασκηνίου**. Ο ενταμιευτής προσκηνίου περιέχει το τρέχον καρέ που σαρώνεται από τον DAC (Digital-Analogue converter) της οθόνης και απεικονίζεται στο χρήστη. Ο ενταμιευτής παρασκηνίου χρησιμοποιείται από την OpenGL ως αποθηκευτικός χώρος στον οποίο σχεδιάζεται το επόμενο καρέ. Όταν ολοκληρωθεί η σχεδίαση του επόμενου καρέ, κατά τη φάση της κάθετης επαναφοράς της δέσμης σάρωσης της οθόνης, οι δύο ενταμιευτές εναλλάσσουν τους ρόλους τους.

Η διπλή ενταμίευση ενεργοποιείται δίνοντας στην εντολή **glutInitDisplayMode** το όρισμα **GLUT_DOUBLE**.

glutInitDisplayMode(GLUT_DOUBLE);

Επιπλέον, στο τέλος της συνάρτησης display, αφού έχει ολοκληρωθεί ο καθορισμός της σκηνής, χρησιμοποιείται η εντολή **glutSwapBuffers**, η οποία εναλάσσει τους ενταμιευτές προσκηνίου και παρασκηνίου:

void glutSwapBuffers();

Όταν στη συνάρτηση display χρησιμοποιείται η **glutSwapBuffers**, δεν είναι αναγκαία η εκτέλεση της **glFlush**, διότι η εκτέλεση της **glutSwapBuffers** προβαίνει στην προώθηση όλων των εντολών σχεδιασμού της σκηνής.

Παράδειγμα: Εφαρμογή διπλής ενταμίευσης

```
#include <glut.h>
GLuint x1=10;
GLuint y1=10;
GLuint x2=x1+50;
GLuint y2=y1+50;
GLuint angle;
void display()
{
    glClearColor(1,1,1,0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3f(1,0,0);
    glRecti(x1,y1,x2,y2);
    glutSwapBuffers();
}
void animate()
{
    glMatrixMode(GL_MODELVIEW);
    // Αυξάνοντας τη γωνία περιστροφής στην οθονη που φαινεται
    // Εάν η γωνία υπερβαίνει τις 360 μοίρες που το προσάρμοσαν στην περιοχή [0,360]
    angle= (angle+1)%360;
    glLoadIdentity();
    // Ορίστε μια περιστροφή γύρω από τον άξονα z.
    glRotatef(angle,0,0,1);
    glutPostRedisplay();
}
int main(int argc, char** argv)
{
    angle=0;
```

```

glutInit(&argc,argv);
glutInitWindowPosition(50,50);
glutInitWindowSize(640,480);
glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
glutCreateWindow("An animated square");
glMatrixMode(GL_PROJECTION);
gluOrtho2D(-160,160,-120,120);
glutIdleFunc(animate);
glutDisplayFunc(display);
glutMainLoop();
return 0;
}

```

3.6.9 Φωτορεαλισμός

Για την απόδοση σκηνών που προσομοιώνουν τον πραγματικό κόσμο, τα εργαλεία που έχουν παρουσιάστει έως τώρα δεν επαρκούν. Πρέπει να προσομοιώθει ο τρόπος με τον οποίο παράγονται οι εικόνες στον πραγματικό κόσμο χρησιμοποιώντας ένα φωτορεαλιστικό μοντέλο (ή μοντέλο σκίασης).

Στο φωτορεαλιστικό μοντέλο η απόδοση μίας σκηνής καθορίζεται:

- α) από τις ιδιότητες των πηγών φωτισμού
- β) από τις ανακλαστικές ιδιότητες των επιφανειών

3.6.9.1 Πηγές φωτισμού

Μια εικόνα σχηματίζεται από την πρόσπτωση ακτινών φωτός σε μια φωτευαίσθητη επιφάνεια (π.χ. ανθρώπινο μάτι ή αισθητήρας κάμερας). Για να σχηματιστεί μια εικόνα, απαιτείται η ύπαρξη φωτός, το οποίο προέρχεται από μία ή περισσότερες πηγές εκπομπής (πηγές φωτισμού). Χωρίς πηγές φωτισμού δε σχηματίζεται εικόνα. Στην OpenGL απαιτείται η ενεργοποίησή του μοντέλου σκίασης με την εντολή **glEnable**:

`glEnable(GL_LIGHTING);`

Για να οριστεί μια πηγή φωτισμού, πρέπει να ορίσουμε ένα σύνολο παραμέτρων που χαρακτηρίζουν τη συμπεριφορά της.

α) Χαρακτηριστικά που προσδιορίζονται με μια αριθμητική τιμή:

Ορίζονται με την εντολή **glLightf**

void glLightf (GLenum light, GLenum parameterName, GLfloat parameter);

light: συμβολική σταθερά που προσδιορίζει την πηγή φωτισμού στην οποία αποδίδουμε το χαρακτηριστικό. (GL_LIGHT0 -GL_LIGHT7).

parameterName: συμβολική σταθερά που καθορίζει μια τη ιδιότητα της πηγής φωτισμού (θα παρουσιαστούν αναλυτικά στη συνέχεια της ενότητας).

parameter: η αριθμητική τιμή που αποδίδουμε στην ιδιότητα **parameterName**

β) Χαρακτηριστικά που προσδιορίζονται από ένα σύνολο αριθμητικών τιμών:

Ορίζονται με την εντολή **glLightfv**:

void glLightfv (GLenum light, GLenum parameterName, const GLfloat * parameterArray);

parameterArray: δείκτης σε μητρώο που περιέχει τις τιμές που προσδιορίζουν το χαρακτηριστικό **parameterName**.

Οι υπόλοιποι παράμετροι είναι ίδιοι με την προηγούμενη εντολή

Στο τέλος, απαιτείται η ενεργοποίηση κάθε μίας πηγής φωτισμού με εντολές **glEnable**:
`glEnable (GL_LIGHTx); (x=0,1,...7)`

3.6.9.2 Κατηγορίες πηγών φωτισμού

Με κριτήριο τη θέση τους στη σκηνή:

- α) Σημειακές πηγές
- β) Πηγές σε άπειρη απόσταση

Με κριτήριο τη διεύθυνση εκπομπής φωτός:

- α) Ομοιόμορφες πηγές
- β) Κατευθυντικές πηγές (σποτ)

Σημειακές πηγές

Καταλαμβάνουν άπειρα μικρό χώρο και δηλώνονται ορίζοντας τη θέση τους στη σκηνή. Εάν επιπλέον η πηγή είναι ομοιόμορφη, οι ακτίνες της διαδίδονται προς όλες τις κατευθύνσεις και με την ίδια ένταση.

Δήλωση σημειακής πηγής: **glLightfv (lightName, GL_POSITION, positionVector);**

glLightName: το όνομα της πηγής (GL_LIGHT0 -GL_LIGHT7)

positionVector: δείκτης σε μητρώο με τέσσερα στοιχεία

(positionVector[0], positionVector[1], positionVector[2]): συντεταγμένες της πηγής φωτισμού (για σημειακές πηγές)

positionVector[3]: Μη μηδενική τιμή ορίζει σημειακή πηγή.

Παράδειγμα δήλωσης σημειακής πηγής

GLfloat light0Position [] = {5,5,0,1};

glLightfv (GL_LIGHT0, GL_POSITION, light0Position);

Ορίζουμε την πηγή GL_LIGHT0 ως σημειακή (το τέταρτο στοιχείο του μητρώου light0Position είναι μη μηδενικό)

Η πηγή φωτισμού έχει συντεταγμένες θέσης:

(x,y,z) = (5,5,0)

Η θέση της πηγής υφίσταται τους μετασχηματισμούς μοντέλου που έχουν δηλωθεί στο αντίστοιχο μητρώο.

Πηγές σε άπειρη απόσταση:

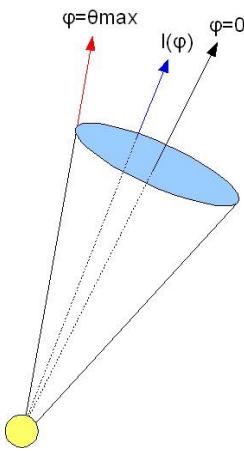
Σε ορισμένες περιπτώσεις θεωρείται ότι η πηγή φωτισμού βρίσκεται σε άπειρη απόσταση από τη σκηνή, ακολουθείται λ.χ. αυτή η παραδοχή όταν μοντελοποιείται ο φωτισμός ενός τοπίου από τον ήλιο. Στην περίπτωση πηγών σε άπειρη απόσταση, όλες οι εκπεμπόμενες ακτίνες ακολουθούν παράλληλες τροχιές σε αντίθεση με την περίπτωση σημειακών πηγών που οι ακτίνες διαδίδονται σφαιρικά.

Στην OpenGL, ορίζεται μια πηγή σε άπειρη απόσταση με την εντολή

glLightfv(GL_LIGHTX, GL_POSITION, positionVector);

Ορισμός κατευθυντικών πηγών

Μια σημειακή πηγή φωτισμού μπορεί να καθοριστεί ούτως ώστε να εκπέμπει σε ένα περιορισμένο γωνιακό εύρος, όπως ένα φωτιστικό σποτ. Σε αυτή την περίπτωση, το τμήμα της σκηνής που φωτίζεται από την κατευθυντική πηγή περικλείεται σε έναν κώνο. Το άνοιγμα του κώνου διάχυσης καθορίζεται από μια γωνία αποκοπής maxθ, όπως φαίνεται στο παρακάτω σχήμα. Σημεία της σκηνής που βρίσκονται εκτός του κώνου διάχυσης δε φωτίζονται καθόλου από τη κατευθυντική πηγή.



Σχήμα 3.36 Ορισμός κατευθυντικών πηγών

Στην OpenGL μια κατευθυντική πηγή υλοποιείται ορίζοντας τις εξής τρείς παραμέτρους: την κατεύθυνση μέγιστης εκπομπής, τη γωνία κάλυψης και τη γωνιακή εξασθένηση της εκπεμπόμενης έντασής της.

α) Κατεύθυνση μέγιστης εκπομπής:

glLightfv(GL_LIGHTX, GL_SPOT_DIRECTION, directionVector);

directionVector: το διάνυσμα που καθορίζει τη διεύθυνση εκπομπής

Αρχικά καθορισμένη κατεύθυνση: ο αρνητικός ημιάξονας Oz (0,0,-1)

β) Γωνία κάλυψης

glLightf(GL_LIGHTX, GL_SPOT_CUTOFF, cutoffAngle);

cutoffAngle: η γωνία αποκοπής σε μοίρες

Αρχικά ορισμένη γωνία αποκοπής: 180 μοίρες (πανκατευθυντική πηγή).

γ) Γωνιακή εξασθένηση

glLightf(GL_LIGHTX, GL_SPOT_EXPONENT, exponent);

exponent: η τιμή του εκθέτη κατευθυντικότητας ($0 \leq \text{exponent} \leq 128$)

Αρχικά ορισμένη τιμή: 0 (ομοιόμορφη εκπομπή)

Η ένταση των σημειακών πηγών εξασθενεί κατά την απομάκρυνση από τη πηγή (ακτινική εξασθένηση).

3.6.9.3 Μοντελοποίηση πηγών φωτισμού στην OpenGL

Θεωρούμε ότι το φως που εκπέμπεται από μία πηγή μπορεί να διαχωριστεί σε τρεις συνιστώσες

α) Συνιστώσα περιβάλλοντος φωτισμού

γ) Συνιστώσα που προκαλεί αποκλειστικά ανακλάσεις διάχυσης

γ) Συνιστώσα που προκαλεί αποκλειστικά κατοπτρικές ανακλάσεις

Αυτός ο διαχωρισμός δεν υφίσταται στην πραγματικότητα, ωστόσο μας δίνει ευελιξία ως προς τα φαινόμενα ανακλάσεων που επιθυμούμε να εμφανίζονται στη σκηνή.

Συνιστώσα περιβάλλοντος φωτισμού

Ο περιβάλλον φωτισμός (ambient light): εκτείνεται σε ολόκληρη τη σκηνή. Επιδρά ομοιόμορφα σε όλες τις επιφάνειες της σκηνής. Η ένταση και το χρώμα του περιβάλλοντος φωτισμού ορίζονται δηλώνοντας τις χρωματικές συνιστώσες του.

Δήλωση στην OpenGL:

glLightfv(lightName, GL_AMBIENT, emittedAmbient);

emittedAmbient: μητρώο που περιέχει τις συνιστώσες του περιβάλλοντος φωτισμού της πηγής **lightName**.

Πχ, με τις εντολές

```
GLfloat light1Ambient[]={0,0,1,0};
```

```
glLightfv(GL_LIGHT1,GL_AMBIENT,light1Ambient);
```

δηλώνουμε ότι η πηγή GL_LIGHT1 εκπέμπει συνιστώσα περιβάλλοντος φωτισμού μπλε χρώματος.

Συνιστώσα “διάχυτου φωτισμού”

Προκαλεί αποκλειστικά ανακλάσεις διάχυσης. Επιδρά μόνο στις επιφάνειες που έχουν οπτική επαφή με την πηγή φωτισμού. Στην OpenGL ορίζεται με την εντολή:

```
glLightfv(GL_LIGHTX, GL_DIFFUSE, emittedDiffuse);
```

emittedDiffuse: ο πίνακας με τα βάρη της διάχυτης συνιστώσας.

Π.χ. με τις εντολές

```
GLfloat light1Diffuse[]={0,1,0,0};
```

```
glLightfv(GL_LIGHT1,GL_DIFFUSE, light1Diffuse);
```

ορίζεται για την πηγή GL_LIGHT1 μία διάχυτη συνιστώσα πράσινου χρώματος.

Συνιστώσα “κατοπτρικού φωτισμού”:

Συνεισφέρει αποκλειστικά στην εμφάνιση κατοπτρικών ανακλάσεων. Ορίζεται με τη δήλωση των χρωματικών συνιστώσων της. Η κατοπτρική συνιστώσα μιας πηγής φωτισμού ορίζεται με την glLightfv ως εξής:

```
glLightfv(GL_LIGHTX, GL_SPECULAR, emittedSpecular);
```

όπου emittedSpecular μητρώο με τα βάρη της “κατοπτρικής συνιστώσας φωτισμού”.

Π.χ. Με τις εντολές

```
GLfloat light1Specular[]={1,0,0,0};
```

```
glLightfv(GL_LIGHT1, GL_SPECULAR, light1Specular);
```

ορίζεται για την πηγή GL_LIGHT1 μία κατοπτρική συνιστώσα κόκκινου χρώματος .

3.6.9.4 Καθολικές παράμετροι φωτισμού

Εκτός από τη ρύθμιση μεμονωμένων πηγών φωτισμού, η OpenGL υποστηρίζει και καθολικές παραμέτρους φωτισμού. Οι καθολικές παράμετροι δεν εντάσσονται σε κάποια από τις πηγές φωτισμού. Για να ρυθμιστούν οι καθολικές παράμετροι φωτισμού, χρησιμοποιείται η εντολή **glLightModel{if}{v}**:

```
void glLightModel*( parameterName, parameterValue );
```

όπου **parameterName** η καθολική παράμετρος που ορίζουμε και **parameterValue** η τιμή ή το μητρώο τιμών που προσδιορίζουν την παράμετρο **parameterName**.

Μια συχνά ρυθμιζόμενη καθολική παράμετρος είναι ο καθολικός περιβάλλον φωτισμός. Η οποία ορίζεται με την χρήση της εντολής **glLightModelfv** ως εξής:

```
GLfloat globalAmbient[]={r,g,b};
```

```
glLightModelfv( GL_LIGHT_MODEL_AMBIENT, globalAmbient );
```

όπου globalAmbient οι συνιστώσες του καθολικού περιβάλλοντος φωτισμού.

3.6.9.5 Μοντελοποίηση ανακλωσών επιφανειών στην OpenGL

Η εμφάνιση μιας σκηνής δεν εξαρτάται μόνο από τα χαρακτηριστικά των πηγών φωτισμού, αλλά και από τα ανακλαστικά χαρακτηριστικά των επιφανειών. Επομένως, η απόδοση μιας σκηνής καθορίζεται από το συνδυασμό των χαρακτηριστικών των πηγών και των επιφανειών.

Οι ιδιότητες των επιφανειών καθορίζονται με τρόπο παρόμοιο με αυτόν που ορίζονται οι ιδιότητες των πηγών. Ανάλογα με τον τύπο του χαρακτηριστικού, χρησιμοποιείται η εντολή

glMaterialf(GLenum face, GLenum property, GLfloat propertyValue); για ιδιότητες που προσδιορίζονται με μία αριθμητική τιμή ή

glMaterialfv(GLenum face, GLenum property, GLfloat *propertyValues); για ιδιότητες που περιγράφονται από ένα σύνολο αριθμητικών τιμών.

Face: καθορίζει την όψη στην οποία αποδίδεται το κάθε χαρακτηριστικό:

GL_FRONT: Το χαρακτηριστικό αποδίδεται στη μπροστινή όψη των επιφανειών.

GL_BACK: Το χαρακτηριστικό αποδίδεται στην πίσω όψη των επιφανειών.

GL_FRONT_AND_BACK: Το χαρακτηριστικό αποδίδεται και στις δύο όψεις των επιφανειών.

Το όρισμα **propertyValue** ή **propertyValues** είναι η αριθμητική τιμή ή ο δείκτης στο μητρώο των αριθμητικών τιμών που καθορίζουν το χαρακτηριστικό αντίστοιχα.

Σε κάθε χρονική στιγμή ορίζεται ένας μόνο ενεργός συνδυασμός χαρακτηριστικών επιφανείας. Δηλαδή, τα χαρακτηριστικά των επιφανειών λειτουργούν ως μεταβλητές κατάστασης. Προκειμένου λοιπόν να μεταβάλλουμε τις τρέχουσες ανακλαστικές ιδιότητες των επιφανειών επιφάνειας, πρέπει να μεταβάλλονται οι τρέχουσες ενεργές τιμές.

3.6.9.6 Συντελεστές ανάκλασης περιβάλλοντος φωτισμού

Προκαλούν την ανάκλαση της συνιστώσας περιβάλλοντος φωτισμού των πηγών. Ο συντελεστής ανάκλασης περιβάλλοντος φωτισμού ορίζεται με την εντολή

glMaterialfv (face, GL_AMBIENT, ambientCoefficients);

ambientCoefficients: το μητρώο με τις αντίστοιχες χρωματικές συνιστώσες.

Σκηνές που μοντελοποιούνται αποκλειστικά με την επίδραση περιβάλλοντος φωτισμού αποδίδουν ένα χρώμα σε κάθε επιφάνεια με ομοιόμορφο συντελεστή ανάκλασης. Αυτό έχει ως αποτέλεσμα να στερούνται φυσικότητας και αίσθησης βάθους, αφού ο περιβάλλον φωτισμός επιδρά με τον ίδιο τρόπο σε όλες τις επιφάνειες ανεξαρτήτως της θέσης και του προσανατολισμού τους. Γι' αυτό το λόγο, η μοντελοποίηση περιβάλλοντος φωτισμού χρησιμοποιείται σε συνδυασμό με πρόσθετες πηγές φωτισμού

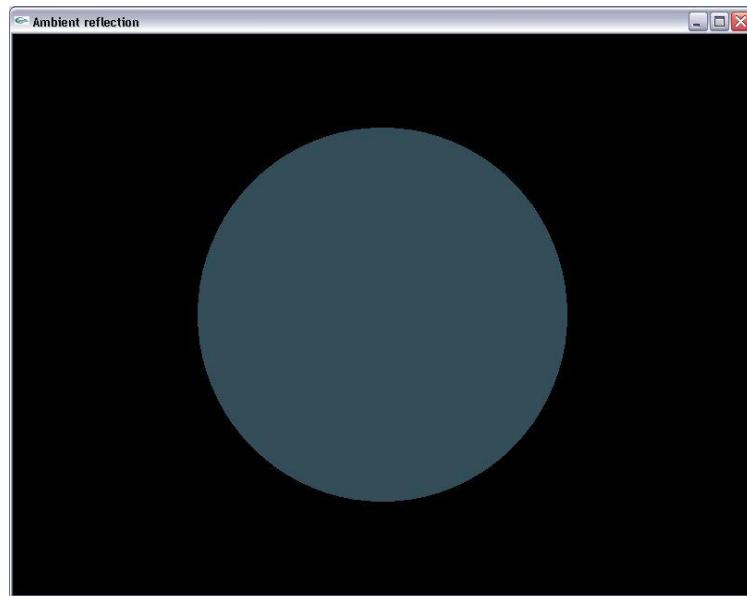
Παράδειγμα: Μοντελοποίηση περιβάλλοντος φωτισμού

```
#include <glut.h>
void init()
{
    glutInitWindowPosition(50,50);
    glutInitWindowSize(800,600);
    glutCreateWindow("Ambient reflection");
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glClearColor(0,0,0,0);
    glMatrixMode(GL_MODELVIEW);
    glTranslatef(0,0,-40);
    glMatrixMode(GL_PROJECTION);
    glOrtho(-80,80,-60,60,0,50);
    glEnable(GL_LIGHTING);
    // Καθορισμός παγκόσμιων ιδιοτήτων του περιβάλλοντος φωτισμού
    GLfloat globalAmbient[]={0.5,0.5,0.5};
    glLightModelfv(GL_LIGHT_MODEL_AMBIENT,globalAmbient);
    // Καθορισμός «αντανάκλαση του περιβάλλοντος» συντελεστών για επιφάνειες
    GLfloat ambientMat[]={0.4,0.6,0.7};
    glMaterialfv(GL_FRONT_AND_BACK,GL_AMBIENT,ambientMat);
}
void display()
{
```

```

glClear(GL_COLOR_BUFFER_BIT);
// Σχεδίαση μιας σφαίρας
glutSolidSphere(40,80,80);
glFlush();
}
int main(int argc, char** argv)
{
glutInit(&argc,argv);
//Initializations
init();
glutDisplayFunc(display);
glutMainLoop();
}

```



Σχήμα 3.37 Μοντελοποίηση περιβάλλοντος φωτισμού

3.6.9.7 Συντελεστές ανάκλασης διάχυτου φωτισμού

Ανακλούν τη συνιστώσα διάχυτου φωτισμού των πηγών και δηλώνονται με την εντολή:

glMaterialfv(glFace, GL_DIFFUSE, diffuseCoefficients);

diffuseCoefficients: μητρώο που περιέχει τους συντελεστές ανάκλασης διάχυτου φωτισμού.

Παράδειγμα: Μοντελοποίηση διάχυτης ανάκλασης

```

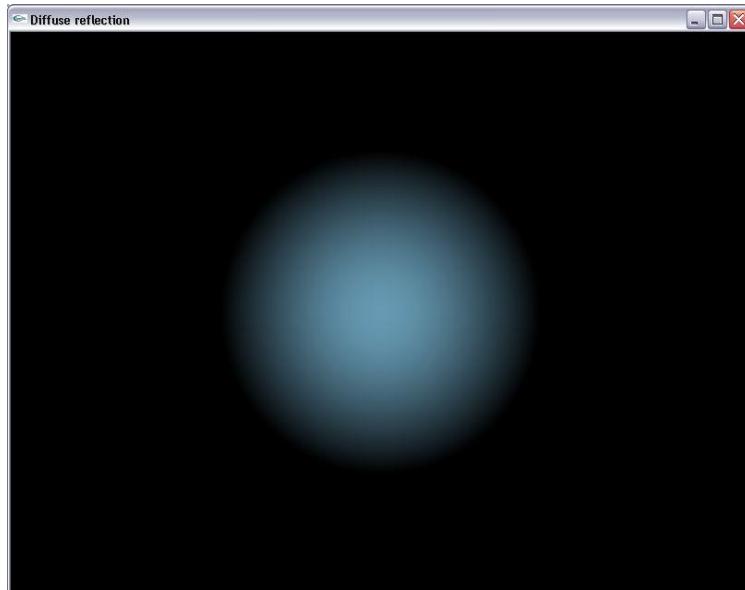
#include <glut.h>
void init()
{
glutInitWindowPosition(50,50);
glutInitWindowSize(800,600);
glutCreateWindow("Diffuse reflection");
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glClearColor(0,0,0,0);
glMatrixMode(GL_PROJECTION);
glOrtho(-80,80,-60,60,0,50);
glEnable(GL_LIGHTING);
//Disabling global ambient light component
GLfloat globalAmbient[]={0,0,0};

```

```

glLightModelfv(GL_LIGHT_MODEL_AMBIENT,globalAmbient);
// Καθορισμός της θέσης της σημειακής πηγής φωτός GL_LIGHT0 at // (x,y,z)=(0,0,40)
GLfloat light0Position[]={0,0,40,1};
glLightfv(GL_LIGHT0,GL_POSITION,light0Position);
// Ορισμός «διάχυτες» ιδιότητες φωτισμού για GL_LIGHT0
GLfloat light0Diffuse[]={1,1,1};
glLightfv(GL_LIGHT0,GL_DIFFUSE,light0Diffuse);
// Καθορισμός «διάχυσης» συντελεστών για επιφάνειες
GLfloat diffuseMat[]={0.4,0.6,0.7};
glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,diffuseMat);
 glEnable(GL_LIGHT0);
// Μεταποίζοντας όλων των επακόλουθων σχήματα από x=-40
glMatrixMode(GL_MODELVIEW);
glTranslatef(0,0,-40);
}
void display()
{
glClear(GL_COLOR_BUFFER_BIT);
// Σχεδιάζοντας μια σφαίρα
	glutSolidSphere(40,80,80);
glFlush();
}
int main(int argc, char** argv)
{
	glutInit(&argc,argv);
// Αρχικοποιήσεις
init();
	glutDisplayFunc(display);
	glutMainLoop();
}

```



Σχήμα 3.38 Μοντελοποίηση διάχυτης ανάκλασης

3.6.9.8 Συντελεστές κατοπτρικής ανάκλασης

Οι συντελεστές κατοπτρικής ανάκλασης ορίζουν το ποσοστό της κατοπτρικής συνιστώσας φωτισμού που ανακλά η επιφάνεια. Ορίζονται με την εντολή **glMaterialfv(face, GL_SPECULAR, specularCoefficients);**

Επιπλέον, μπορούμε να καθορίσουμε την τραχύτητα της επιφάνειας (εύρος κώνου διάχυσης), δίνοντας τον εκθέτη του γωνιακού παράγοντα εξασθένησης με την εντολή:
glMaterialf (face, GL_SHININESS, n);

n: ο εκθέτης γωνιακής εξασθένησης, ο οποίος κυμαίνεται μεταξύ των τιμών 0 και 128. Εάν δε δηλωθεί εκθέτης γωνιακής εξασθένησης από τον προγραμματιστή, η μηχανή της OpenGL θεωρεί ως προκαθορισμένη την τιμή 0, οπότε η ανακλώσα επιφάνεια ανακλά ομοιόμορφα την προσπίπτουσα κατοπτρική συνιστώσα προς όλες τις κατευθύνσεις.

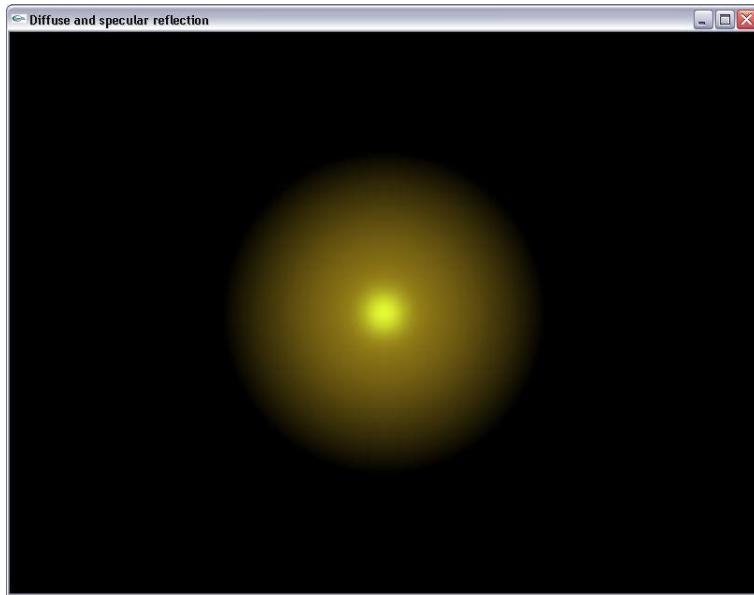
Παράδειγμα: Μοντελοποίηση διάχυτης και κατοπτρικής ανάκλασης

```
#include <glut.h>
void init()
{
    glutInitWindowPosition(50,50);
    glutInitWindowSize(800,600);
    glutCreateWindow("Diffuse reflection");
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glClearColor(0,0,0,0);
    glMatrixMode(GL_PROJECTION);
    glOrtho(-80,80,-60,60,0,50);
    glEnable(GL_LIGHTING);
    // Απενεργοποίηση της παγκόσμιας περιβάλλοντος φωτισμού
    GLfloat globalAmbient[]={0,0,0};
    glLightModelfv(GL_LIGHT_MODEL_AMBIENT,globalAmbient);
    // Καθορισμός της θέσης της σημειακής πηγής φωτός GL_LIGHT0 στο (x,y,z)=(0,0,40)
    GLfloat light0Position[]={0,0,40,1};
    glLightfv(GL_LIGHT0,GL_POSITION,light0Position);
    // Ορισμός των «διάχυτων» ιδιότητων φωτισμού για GL_LIGHT0
    GLfloat light0Diffuse[]={1,1,1};
    glLightfv(GL_LIGHT0,GL_DIFFUSE,light0Diffuse);
    // Ορισμός των «κατοπτρικών» ιδιότητων φωτισμού για GL_LIGHT0
    GLfloat light0Specular[]={.5,1,1};
    glLightfv(GL_LIGHT0,GL_SPECULAR,light0Specular);
    // Καθορισμός «διάχυσης» συντελεστών για επιφάνειες
    GLfloat diffuseMat[]={0.6,0.5,0.1};
    glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,diffuseMat);
    //Defining "specular reflection" coefficients for surfaces Ορισμός των «κατοπτρικών ανάκλασεων» συντελεστών για επιφάνειες
    GLfloat specularMat[]={.6,.5,.1};
    glMaterialfv(GL_FRONT_AND_BACK,GL_SPECULAR,specularMat);
    // Καθορισμός εκθέτη υλικού λάμψης
    glMaterialf(GL_FRONT_AND_BACK,GL_SHININESS,64);
    glEnable(GL_LIGHT0);
    // Μετατοπίζοντας όλων των επακόλουθων σχήματα από x=-40 glMatrixMode(GL_MODELVIEW);
    glTranslatef(0,0,-40);
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    //Σχεδιάζοντας την σφαίρα
    glutSolidSphere(40,80,80);
    glFlush();
}
int main(int argc, char** argv)
{
    glutInit(&argc,argv);
    // Αρχικοποιήσεις
    init();
```

```

glutDisplayFunc(display);
glutMainLoop();
}

```



Σχήμα 3.39 Μοντελοποίηση διάχυτης και κατοπτρικής ανάκλασης

Σημείωση:

Ένας κανόνας για τη ρεαλιστική απόδοση μιας ανακλώσας επιφάνειας ορίζει ότι, οι συντελεστές διάχυτης ανάκλασης και οι συντελεστές ανάκλασης περιβάλλοντος φωτισμού είναι οι ίδιοι. Προκειμένου λοιπόν να αποδώσουμε τις ιδιότητες αυτές με μία εντολή, μπορούμε να χρησιμοποιήσουμε την εντολή `glMaterialfv` με το όρισμα `GL_AMBIENT_AND_DIFFUSE` ως εξής:

```

GLfloat ambientAndDiffuse[]={r,g,b};
glMaterialfv(GL_FRONT_AND_BACK,GL_AMBIENT_AND_DIFFUSE,ambient
AndDiffuse)

```

3.6.9.9 Ατμοσφαιρικά εφέ (ομίχλη)

Φαινόμενο ομίχλης: Εξασθένηση της παρατηρούμενης έντασης της ανακλώσας επιφάνειας που αντιλαμβάνεται ένας παρατηρητής όσο απομακρύνεται από αυτήν.

Απόδοση ομίχλης: Θεωρούμε ότι μεταξύ της ανακλώσας επιφάνειας και του παρατηρητή παρεμβάλλεται ένα ημιδιαφανές εμπόδιο (ατμόσφαιρα). Η διαφάνεια του εμποδίου καθορίζεται από το πάχος που μεσολαβεί μεταξύ του παρατηρητή και της ανακλώσας επιφάνειας.

Μοντελοποίηση ομίχλης

Αρχικά ενεργοποιείται η δυνατότητα απόδοσης ομίχλης

```
glEnable(GL_FOG);
```

Ρυθμίζονται οι παραμέτροι ομίχλης με την εντολή `glFog`:

```
void glFog{if}{v}(TYPE parameterName, TYPE parameterValue);
```

Για να προσδιοριστεί το εφέ ομίχλης καθορίζονται τρεις παραμέτροι: το χρώμα, η πυκνότητα και το μοντέλο εξασθένησής της.

a) **Χρώμα ομίχλης**

```
glFogfv(GL_FOG_COLOR, fogColorComponents);
```

fogColorComponents: το μητρώο με τις χρωματικές συνιστώσες.

Αρχική τιμή {0,0,0}

β) Πυκνότητα ομίχλης

glFogf(GL_FOG_DENSITY, density);

Π.χ. με τις εντολές

GLfloat *fogColorArray={0.5,0.5,0.5};

glFogfv(GL_FOG_COLOR,fogColorArray);

αποδίδεται στην ομίχλη γκρίζο χρώμα.

Εάν δεν αποδοθεί χρώμα στην ομίχλη, η προκαθορισμένη του τιμή είναι {0,0,0,0} (μαύρο).

γ) Μοντέλο εξασθένησης

Η επιλογή του μοντέλου εξασθένησης γίνεται προσδιορίζοντας την παράμετρο

GL_FOG_MODE:

glFogi(GL_FOG_MODE, mode);

Η παράμετρος mode παίρνει τις εξής τιμές

GL_EXP: εκθετική εξασθένηση (προεπιλεγμένο)

GL_EXP2: τετραγωνική εκθετική εξασθένηση

GL_LINEAR: γραμμική εξασθένηση

Το αρχικά επιλεγμένο μοντέλο είναι το μοντέλο εκθετικής εξασθένησης (**GL_EXP**).

Πέραν της απόδοσης ομιχλωδών τοπίων, το φαινόμενο ομίχλης είναι χρήσιμο για τη σαφή απόδοση σκηνών που περιέχουν κλειστές επιφάνειες σε μορφή πλέγματος. Σε σκηνές στις οποίες αποδίδουμε το πλέγμα τρισδιάστατων κλειστών επιφανειών (σφαιρών, κυλίνδρων κλπ.)

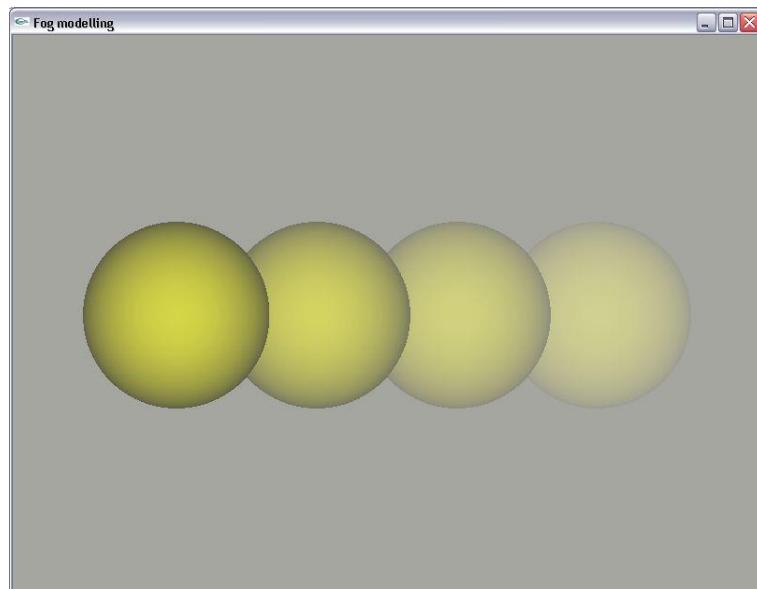
Παράδειγμα: Μοντελοποίηση ομίχλης

```
#include <glut.h>
void init()
{
    glutInitWindowPosition(50,50);
    glutInitWindowSize(800,600);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutCreateWindow("Fog modelling");
    glMatrixMode(GL_PROJECTION);
    glOrtho(-80,80,-60,60,0,100);
    glClearColor(0,0,0,1);
    // Ενεργοποίηση αφαίρεσης κρυμμένης επιφάνειας
    glEnable(GL_DEPTH_TEST);
    // Ενεργοποίηση μοντέλου σκίασης
    glEnable(GL_LIGHTING);
    // Καθορίζοντας μια μακρινή πηγή φωτός με τις ακτίνες φωτός που ταξιδεύουν κατά μήκος του άξονα z
    GLfloat light0Position[]={0,0,1,0};
    glLightfv(GL_LIGHT0,GL_POSITION,light0Position);
    // Καθορισμός υλικού περιβάλλοντος και διάχυτων ιδιότητων ανάκλασης
    GLfloat reflectionCoefficients[]={0.7,0.7,0.2};
    glMaterialfv(GL_FRONT,GL_AMBIENT_AND_DIFFUSE,reflectionCoefficients);
    glEnable(GL_LIGHT0);
    // Ενεργοποίηση ατμοσφαιρικών εφέ
    glEnable(GL_FOG);
    // Ρύθμιση χρώματος ομίχλης: (0.8, 0.8, 0.8)
    GLfloat fogColor[]={0.8,0.8,0.8};
    glFogfv(GL_FOG_COLOR,fogColor);
    // Ρύθμιση πυκνότητας ομίχλης σε 0.015. Ένα εκθετικό μοντέλο εξασθένισης υποτίθεται.
    glFogf(GL_FOG_DENSITY,0.015);
}
```

```

void display()
{
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    // Σχεδιάζοντας μια σφαίρα με κέντρο στη θέση (-30, -0, -25)
    glLoadIdentity();
    glTranslatef(-45,0,-25);
    glutSolidSphere(20,80,80);
    // Σχεδιάζοντας μια σφαίρα στο κέντρο στη θέση (0,0, -40)
    glLoadIdentity();
    glTranslatef(-15,0,-40);
    glutSolidSphere(20,80,80);
    // Σχεδιάζοντας μια σφαίρα με κέντρο στη θέση (30,0, -60)
    glLoadIdentity();
    glTranslatef(15,0,-60);
    glutSolidSphere(20,80,80);
    // Σχεδιάζοντας μια σφαίρα με κέντρο στη θέση (30,0,-80)
    glLoadIdentity();
    glTranslatef(45,0,-80);
    glutSolidSphere(20,80,80);
    // Σχεδιάζοντας ένα "σχέδιο φόντου" (z=-99)
    glLoadIdentity();
    glBegin(GL_QUADS);
    glVertex3f(-80,-60,-99);
    glVertex3f(80,-60,-99);
    glVertex3f(80,60,-99);
    glVertex3f(-80,60,-99);
    glEnd();
    glFlush();
}
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}

```



Σχήμα 3.40 Μοντελοποιήση ομίχλης

3.93

3.6.10 Απόδοση υφής στην OpenGL

Στο προηγούμενο κεφάλαιο αναφερθήκε στο φωτορεαλισμό ως μια τεχνική ρεαλιστικής απόδοσης σκηνών σε ότι αφορά τις ανακλαστικές τους ιδιότητες. Ωστόσο, σε αρκετές περιπτώσεις, οι ανακλαστικές ιδιότητες των επιφανειών που παρατηρούνται σε πραγματικές σκηνές έχουν τόσο σύνθετες ιδιότητες που η μοντελοποίησή τους αποκλειστικά με τη χρήση του μοντέλου σκίασης είναι μια πολύ επίπονη έως αδύνατη διαδικασία. Είναι πολύ δύσκολο να προσομοιώσει κανείς τη μορφολογία φυσικών επιφανειών όπως λχ την επιφάνεια ξύλου, μαρμάρου, εδάφους, γρασιδιού κ.λ.π.

Στις περιπτώσεις απόδοσης φυσικών σκηνών με περίπλοκη μορφολογία, καταφεύγουμε στη σχεδιάση ρεαλιστικών επιφανειών βασιζόμενοι σε προκαθορισμένα πρότυπα επιφανειών. Τα πρότυπα αυτά αποδίδουν με πιστότητα τη μορφολογία των επιφανειών και μπορούν να ληφθούν με τη μορφή ψηφιοποιημένων εικόνων. Στα γραφικά υπολογιστών ένα τέτοιο πρότυπο απόδοσης μορφολογικών ιδιοτήτων αποκαλείται υφή (texture) και η διαδικασία της απόδοσης των ιδιοτήτων του σε μια επιφάνεια ονομάζεται απόδοση υφής (texture mapping).

Στην OpenGL η απόδοση υφής σε γραμμές και επιφάνειες εκτελείται αναθέτοντας σε κάθε μία κορυφή συντεταγμένη(η/ες) υφής. Με τον τρόπο αυτό η μηχανή της OpenGL αποδίδει τα στοιχεία του μητρώου υφής κατά μήκος της γραμμής ή στην έκταση της επιφάνειας ούτως ώστε να ικανοποιούνται οι οριακές συνθήκες που επιβάλλει ο προγραμματιστής.

Η ανάθεση συντεταγμένων υφής γίνεται με τη χρήση της εντολής **glTexCoord{1,2}{i,s,f,d}**

Για ορίσματα κινητής υποδιαστολής (GLfloat) οι συναρτήσεις είναι:

void glTexCoord1f (GLfloat s); για μονοδιάστατες υφές

void glTexCoord2f (GLfloat s, GLfloat t); για δισδιάστατες υφές.

Με τα ορίσματα s, t της **glTexCoord** ορίζουμε τις τρέχουσες συντεταγμένες υφής, οι οποίες αποδίδονται σε όλα τα σημεία που θα δηλωθούν στη συνέχεια του κώδικα. Οι τρέχουσες συντεταγμένες υφής διατηρούν τις τιμές που τους ανατέθηκαν την τελευταία φορά, επομένως λειτουργούν ως μεταβλητές κατάστασης.

Προκειμένου να επεκτείνουμε μια υφή σε μια επιφάνεια, θα πρέπει να μεταβάλουμε τις τρέχουσες συντεταγμένες υφής με διαδοχικές εντολές **glTexCoord** πριν τη δήλωση κάθε κορυφής της επιφανείας.

Πχ με τις παρακατω εντολές ορίζουμε ένα ευθύγραμμο τμήμα. Στο άκρο με συντεταγμένες σκηνής (10,20) αναθέτουμε τη συντεταγμένη (μονοδιάστατης) υφής s=0 και στο άκρο με συντεταγμένες σκηνής (20,20) αποδίδουμε τη συντεταγμένη υφής s=0.7. Επομένως, η OpenGL θα αποδώσει στα ενδιάμεσα pixels του ευθυγράμμου τμήματος τις χρωματικές τιμές των texels που οι συντεταγμένες υφής τους κυμαίνονται από 0 έως 0.7.

```
glBegin(GL_LINES);
glTexCoord1f(0);
glVertex2f(10,20);
glTexCoord1f(0.7);
glVertex2f(20,20);
glEnd();
```

3.6.10.1 Απόδοση υφής σε καμπύλες (1D)

Για την απόδοση μονοδιάστατης υφής σε καμπύλες, αρχικά απαιτείται η ενεργοποίηση της λειτουργίας με την εντολή glEnable:

glEnable(GL_TEXTURE_1D);

Η φόρτωση ενός μητρώου στοιχείων υφής γίνεται με την εντολή **glTexImage1D**:

void glTexImage1D(GLenum target, GLint level, GLint internalFormat, GLsizei width, GLint border, GLenum format, const GLvoid *texelArray);

Ορίσματα

Target: παίρνει την τιμή GL_TEXTURE_1D για να προσδιορίστει το μονοδιάστατο μητρώο υφής.

Level: καθορίζει αν το μητρώο υφής χρησιμοποιείται ως μια βαθμίδα σε μια πυραμίδα μητρώων. Αυτή η παράμετρος είναι χρήσιμη μόνο όταν χρησιμοποιεί ο προγραμματιστής μητρώα υφής σε πολλαπλές αναλύσεις. Με τιμή 0 ορίζεται ότι η υφή ανήκει στην κορυφή της πυραμίδας και είναι η συνιστώμενη τιμή όταν δεν χρησιμοποιούνται πολλαπλές αναλύσεις.

Width: καθορίζει το πλήθος των texels που περιέχονται στο μητρώο υφής. Τα μητρώα υφής έχουν περιορισμούς σε ό,τι αφορά το πλήθος των στοιχείων τους. Συγκεκριμένα το πλήθος των texels θα πρέπει να είναι δύναμη του 2 (4, 8, 16, 32, 64, 128 κτλ).

Border: καθορίζεται αν η υφή θα περιβάλλεται από όριο πάχους ενός pixel. Με την τιμή ορίζουμε ότι δε χρησιμοποιείται όριο, ενώ με τιμή ορίζουμε ότι χρησιμοποιούμε όριο. Το όριο χρησιμοποιείται για το διαχωρισμό γραμμικών ή επιφανειακών τμημάτων στα οποία αποδίδονται διαφορετικές υφές. Το χρώμα του ορίου θα πρέπει να δηλωθεί με την προσθήκη δύο επιπλέον στοιχείων στο μητρώο υφής: ένα στην αρχή και ένα στοιχείου στο τέλος του μητρώου.

internalFormat: καθορίζει το πλήθος των συνιστωσών του χρωματικού μοντέλου στο οποίο περιγράφουμε τα texels του μητρώου υφής. Οι τιμές που χρησιμοποιούνται συνήθως είναι οι εξής:

GL_LUMINANCE: Τα texels ορίζουν αποχρώσεις του γκρίζου.

GL_RGB: Τα texels περιγράφονται στο μοντέλο RGB (με 3 συνιστώσες).

GL_RGBA: Τα texels προσδιορίζονται στο μοντέλο RGBA (με 4 συνιστώσες).

Format: καθορίζει τη διαδοχή με την οποία δίνονται οι συνιστώσες του χρωματικού μοντέλου. **Οι υποστηριζόμενες τιμές είναι:**

GL_RGB: Ορίζεται η διαδοχή συνιστωσών κόκκινο-πράσινο-μπλε (χρησιμοποιείται σε αρχεία εικόνων τύπου BMP (bitmaps))

GL_BGR_EXT: για διαδοχή συνιστωσών μπλε-πράσινο-κόκκινο (χρησιμοποιείται σε αρχεία εικόνων τύπου DIB (device independent bitmaps))

GL_RED,GL_GREEN, GL_BLUE, GL_ALPHA: οι τιμές του μητρώου ορίζουν αποκλειστικά συνιστώσες του γκρίζου, του κόκκινου, του πράσινου, του μπλε ή τιμών alpha αντίστοιχα (μονοχρωματικά μητρώα υφής)

GL_RGBA: ορίζουμε τη διαδοχή συνιστωσών κόκκινο-πράσινο-μπλε-alpha

GL_BGRA_EXT: Ορίζουμε τη διαδοχή συνιστωσών μπλε-πράσινο-κόκκινο-alpha

Type: καθορίζει τον πρωτογενή τύπο δεδομένων με τον οποίο δίνονται τα texels στο μητρώο υφής. Δέχεται τις τιμές GL_UNSIGNED_BYTE (η πιο συνήθης, ειδικά όταν οι υφές φορτώνονται από αρχεία εικόνων), GL_INT και GL_FLOAT.

texelArray: είναι δείκτης στο μητρώο που περιέχει τις χρωματικές τιμές των texels. Στην περίπτωση που το μητρώο είναι μονοδιάστατο, τα στοιχεία του μητρώου προσδιορίζουν τις τιμές των texels ανά n-άδες όπου n το πλήθος των χρωματικών συνιστωσών που περιγράφουν κάθε texel.

3.6.10.2 Απόδοση υφής σε επιφάνειες (2D)

Για απόδοση υφής σε επιφάνειες απαιτείται η ενεργοποίηση της απόδοσης δισδιάστατων υφών με την εντολή **glEnable**:

glEnable(GL_TEXTURE_2D);

Η καταχώρηση ενός διδιάστατου μητρώου υφής γίνεται με την εντολή **glTexImage2D**:

```
void glTexImage2D(GLenum target, GLint level, GLint internalFormat,GLsizei width,
GLsizei height, GLint border, GLenum format, GLenum type, const GLvoid
*texelArray );
```

η οποία δέχεται τα ίδια ορίσματα με αυτά που δέχεται και η εντολή **glTexImage1D**.

Παράδειγμα:

```
glBegin(GL_POLYGON);
glTexCoord2f(0,0); glVertex2f(0,0);
glTexCoord2f(1,0); glVertex2f(10,0);
glTexCoord2f(1,1); glVertex2f(10,10);
glTexCoord2f(0,1); glVertex2f(0,10);
glEnd();
```

Στο παραπάνω παράδειγμα ορίζεται μια ορθογώνια επιφάνεια και αποδίδουμε στην κορυφή $(x,y)=(0,0)$ συντεταγμένες υφής $(s,t)=(0,0)$, στην κορυφή $(x,y)=(10,0)$ συντεταγμένες υφής $(s,t)=(1,0)$, στην κορυφή $(x,y)=(10,10)$ συντεταγμένες υφής $(s,t)=(1,1)$ και στην κορυφή $(x,y)=(0,10)$ συντεταγμένες υφής $(s,t)=(0,1)$.

3.6.10.3 Ρυθμίσεις απόδοσης υφής

Οι αλγόριθμοι απόδοσης υφής επιδέχονται ρυθμίσεις οι οποίες δηλώνονται από τον προγραμματιστή με την εντολή **glTexParameter**

glTexParameter{if}{v}(GLenum target, GLenum parameterName, TYPE parameterValue);

target: άν η ρύθμιση επιβάλλεται σε μονοδιάστατες ή δισδιάστατες υφές.

GL_TEXTURE_1D: μονοδιάστατες υφές

GL_TEXTURE_2D: όταν ρυθμίζουμε παραμέτρους απόδοσης που αφορούν δισδιάστατες υφές.

parameterName: δίνουμε το όνομα της παραμέτρου που καθορίζεται.

parameterValue: δίνεται η τιμή ή το μητρώο τιμών που προσδιορίζει την παράμετρο **parameterName**.

Παραμέτροι που ρυθμίζονται με τη χρήση της **glTexParameter**:

Σμίκρυνση υφής - Μεγέθυνση υφής

Αποκοπή υφής - Επανάληψη υφής

3.6.10.4 Σμίκρυνση υφής - Μεγέθυνση υφής

Κατά την απόδοση υφής σε μια καμπύλη ή επιφάνεια υπάρχουν δύο περιπτώσεις: η υφή είτε να σμικρύνθει είτε να μεγεθυνθεί, ούτως ώστε να προσαρμοστεί στα όρια που καθορίζει ο προγραμματιστής. Οι δύο αυτές περιπτώσεις αντιμετωπίζονται ξεχωριστά από τη μηχανή της OpenGL. Αυτό σημαίνει ότι μπορούν να επιβληθούν ξεχωριστές ρυθμίσεις για κάθε μία περίπτωση, σε ό,τι αφορά την προσέγγιση της χρωματικής τιμής των pixels της καμπύλης. Η επιλογή αυτή γίνεται με την εντολή **glTexParameter**:

void glTexParameter(GL_TEXTURE_1D/GL_TEXTURE_2D,target parameterName, parameterValue);

parameterName: παίρνει την τιμή **GL_TEXTURE_MIN_FILTER** ή **GL_TEXTURE_MAG_FILTER**, ανάλογα με το αν θα καθορίστει η συμπεριφορά της απόδοσης υφής κατά τη σμίκρυνση ή τη μεγέθυνσή της αντίστοιχα.

parameterValue παίρνει τις εξής τιμές:

GL_NEAREST: Εάν οι συντεταγμένες υφής ενός pixel δε συμπίπτουν ακριβώς με τις συντεταγμένες υφής ενός texel, αποδίδουμε τις τιμές του πλησιέστερου γειτονικού texel.

GL_LINEAR: Εάν οι συντεταγμένες υφής ενός pixel δε συμπίπτουν ακριβώς με τις συντεταγμένες υφής ενός texel προσεγγίζουμε την αποδιδόμενη χρωματική τιμή από τις τιμές των πλησιέστερων texels, βάσει γραμμικής παρεμβολής.

Π.χ. με τις εντολές

```
glTexParameteri
```

```
(GL_TEXTURE_1D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
```

```
glTexParameteri (GL_TEXTURE_1D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
```

Δηλώνετα ότι, κατά την απόδοση μονοδιάστατης υφής, εφαρμόζεται την προσέγγιση πλησιέστερου γείτονα σε περιπτώσεις σμίκρυνσης υφής και την προσέγγιση γραμμικής παρεμβολής σε περιπτώσεις μεγέθυνσης της υφής.

3.6.10.5 Αποκοπή υφής - Επανάληψη υφής

Στην OpenGL, αντί για την τεχνική επανάληψης της υφής, μπορεί να εφαρμόστει η τεχνική της αποκοπής (clamping). Στην τεχνική αποκοπής, εάν μία συντεταγμένη υφής ενός σημείου βρίσκεται εκτός του διαστήματος [0,1], θεωρούμε το πλησιέστερο texel με συντεταγμένη υφής στο διάστημα [0,1]. Παραδείγματος χάριν εάν σε ένα σημείο αποδώσουμε συντεταγμένες υφής (1.5,0.7) με την τεχνική αποκοπής θα του αποδοθεί η χρωματική τιμή του texel που έχει συντεταγμένες υφής (1,0.7). Εάν σε ένα σημείο αποδώσουμε συντεταγμένες υφής (1.3,-0.8) θα του αποδώσουμε τη χρωματική τιμή του texel που έχει συντεταγμένες υφής (1,0).

Η επαναλαμβανόμενη ή αποκοπόμενη απόδοση στοιχείων υφής καθορίζεται ξεχωριστά για κάθε διεύθυνση s, t χρησιμοποιώντας την εντολή **glTexParameter**:

```
void glTexParameterf(GLenum target, GLenum parameterName, GLfloat parameterValue);
```

target: παίρνει την τιμή GL_TEXTURE_1D για μονοδιάστατες υφές ή GL_TEXTURE_2D για διδιάστατες υφές.

parameterName: καθορίζει τη διεύθυνση για την οποία ρυθμίζουμε τον τρόπο απόδοσης υφής. Παίρνει τις τιμές:

GL_TEXTURE_WRAP_S: προσδιορίζουμε τη ρύθμιση που θα ισχύσει κατά τη διάσταση των συντεταγμένων υφής (κατά μήκος των γραμμών) s

GL_TEXTURE_WRAP_T: προσδιορίζουμε τη ρύθμιση που θα ισχύσει κατά τη διάσταση των συντεταγμένων υφής (κατά μήκος των στηλών) t

parameterValue: καθορίζει με ποιο κριτήριο θα αποδίδεται υφή σε σημεία με συντεταγμένες υφής εκτός του διαστήματος (0,1). Οι διαθέσιμες επιλογές είναι:

GL_CLAMP: Στο σημείο ανατίθεται η πλησιέστερη συντεταγμένη υφής που ορίζεται στο διάστημα (0,1). Δηλαδή για σημεία στα οποία η συντεταγμένη υφής είναι μεγαλύτερη της μονάδας ανατίθεται η συντεταγμένη υφής 1 και για σημεία στα οποία η συντεταγμένη υφής είναι μικρότερη του μηδενός ανατίθεται η τιμή 0.

GL_REPEAT: Σε κάθε σημείο ανατίθεται η χρωματική τιμή του texel που οι συντεταγμένες υφής του είναι ίσες με το δεκαδικό μέρος των συντεταγμένων υφής του σημείου.

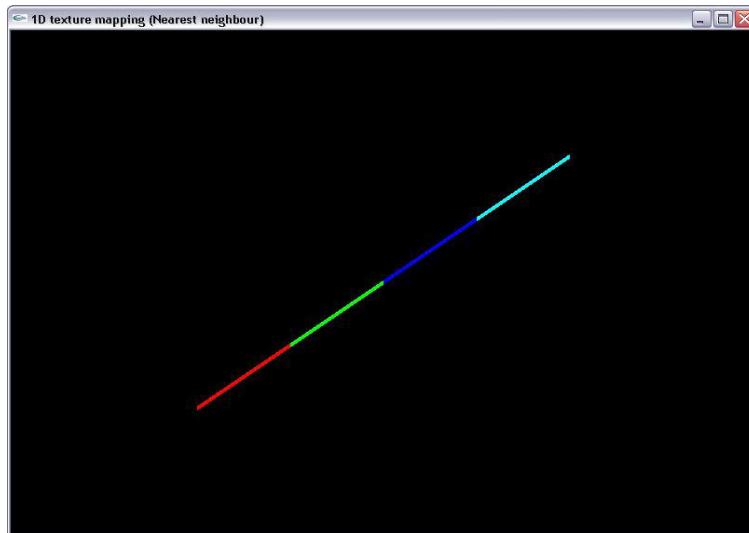
Παράδειγμα: Απόδοση υφής στη μία διάσταση

```
#include <glut.h>
void init()
{
```

```

glutInitWindowPosition(50,50);
glutInitWindowSize(800,600);
glutCreateWindow("1D texture mapping (Nearest neighbour)");
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGBA);
glMatrixMode(GL_PROJECTION);
gluOrtho2D(-80,80,-60,60);
glClearColor(0,0,0,0);
// Καθορισμός μιας μονοδιάστατης υφής με 4 texels (texture element) στο χρωματικό μοντέλο RGB
GLfloat texture[4][3]={
{1,0,0},
{0,1,0},
{0,0,1},
{0,1,1}
};
glEnable(GL_TEXTURE_1D);
// Ρύθμιση χαρτογράφησης υφής του nearest-neighbour (πλησιέστερος γείτονας)
glTexParameteri(GL_TEXTURE_1D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
glTexParameteri(GL_TEXTURE_1D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
//Καταχώρηση μονοδιάστατου πίνακα υφής
glTexImage1D(GL_TEXTURE_1D,0,GL_RGB,4,0,GL_RGB,GL_FLOAT,&texture[0][0]);
}
void display()
{
glLineWidth(5);
glClear(GL_COLOR_BUFFER_BIT);
glBegin(GL_LINE_STRIP);
glTexCoord1f(0);
glVertex2f(-40,-30);
glTexCoord1f(1);
glVertex2f(40,30);
glEnd();
glFlush();
}
int main(int argc, char **argv)
{
glutInit(&argc,argv);
init();
glutDisplayFunc(display);
glutMainLoop();
return 0;
}

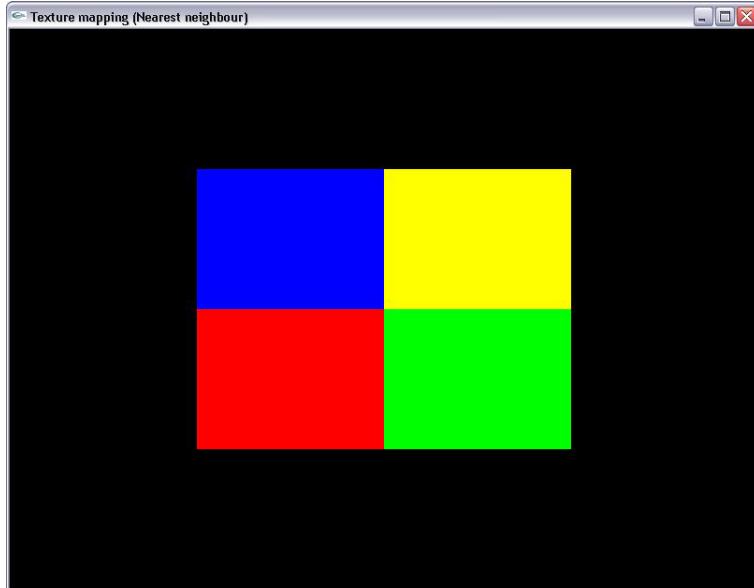
```



Σχήμα 3.41 Απόδοση υφής στη μια διάσταση

Παράδειγμα: Απόδοση υφής σε ορθογώνια επιφάνεια (πλησιέστερος γείτονας)

```
#include <glut.h>
void init()
{
    glutInitWindowPosition(50,50);
    glutInitWindowSize(800,600);
    glutCreateWindow("Texture mapping (Nearest neighbour)");
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(-80,80,-60,60);
    glEnable(GL_TEXTURE_2D);
    glClearColor(0,0,0,0);
    // Καθορίζοντας ένα πίνακα υφής 4x4
    GLfloat texture[2][2][3]={
        {{1,0,0}, {0,1,0}},
        {{0,0,1}, {1,1,0}}
    };
    // Επιλέγοντας την χαρτογράφηση υφής «πλησιέστερο γείτονα»
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
    // Καταχωρηση δισδιαστατου πινακα υφης
    glTexImage2D(GL_TEXTURE_2D,0,GL_RGB,2,2,0,GL_RGB,GL_FLOAT,&texture[0][0][0]);
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);
    glTexCoord2f(0,0);
    glVertex2f(-40,-30);
    glTexCoord2f(1,0);
    glVertex2f(40,-30);
    glTexCoord2f(1,1);
    glVertex2f(40,30);
    glTexCoord2f(0,1);
    glVertex2f(-40,30);
    glEnd();
    glFlush();
}
int main(int argc, char **argv)
{
    glutInit(&argc,argv);
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```



Σχήμα 3.42 Απόδοση υφής σε ορθογώνια επιφάνεια

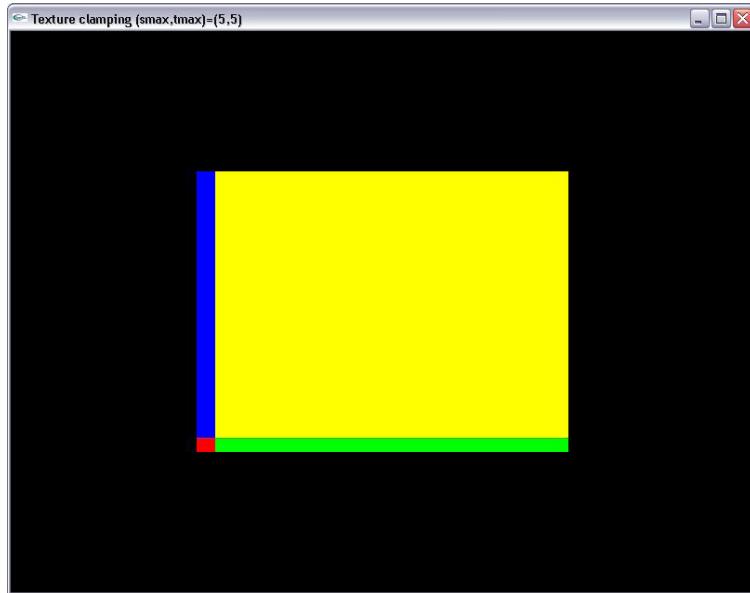
Παράδειγμα: Αποκοπή υφής στις δύο διαστάσεις

```
#include <glut.h>
void init()
{
    glutInitWindowPosition(50,50);
    glutInitWindowSize(800,600);
    glutCreateWindow("Texture wrapping (smax,tmax)=(5,5)");
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(-80,80,-60,60);
    glEnable(GL_TEXTURE_2D);
    glClearColor(0,0,0,0);
    // Ορίζοντας ένα 2x2 πίνακα υφής
    GLfloat texture[2][2][3]={
        {{1,0,0}, {0,1,0}},
        {{0,0,1}, {1,1,0}}
    };
    // Ενεργοποίηση πρότυπο σύσφιξης κατά μήκος s και t συντεταγμένες
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_CLAMP);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_CLAMP);
    // Επιλέγοντας την χαρτογράφηση υφής «πλησιέστερο γείτονα»
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
    // Καταχωρηση δισδιαστατου πινακα υφης
    glTexImage2D(GL_TEXTURE_2D,0,GL_RGB,2,2,0,GL_RGB,GL_FLOAT,&texture[0][0][0]);
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    // Οριακά συντεταγμένες υφής: (tmin, tmin) = (0,0), (tmax, tmax) = (5,5)
    glBegin(GL_POLYGON);
    glTexCoord2f(0,0);
    glVertex2f(-40,-30);
    glTexCoord2f(10,0);
    glVertex2f(40,-30);
    glTexCoord2f(10,10);
    glVertex2f(10,10);
}
```

```

glVertex2f(40,30);
glTexCoord2f(0,10);
glVertex2f(-40,30);
glEnd();
glFlush();
}
int main(int argc, char **argv)
{
glutInit(&argc,argv);
init();
glutDisplayFunc(display);
glutMainLoop();
return 0;
}

```



Σχήμα 3.43 Απόδοση υφής στη δυο διάστασεις

3.6.10.6 Αυτόματη απόδοση υφής σε τετραγωνικές επιφάνειες

Σε σύνθετες επιφάνειες η διαδικασία απόδοσης υφής περισσότερο πολύπλοκη διαδικασία. Ωστόσο η OpenGL επιτρέπει να αποδοθεί αυτόματα συντεταγμένες υφής σε τετραγωνικές επιφάνειες (που σχηματίζονται με εντολές της GLU) με σχετικά απλό τρόπο με τη χρήση της εντολής **gluQuadricTexture**:

void gluQuadricTexture (GLUquadric *quadObject, GLboolean textureCoords);
qObj: το αντικείμενο που αντιστοιχεί στην τετραγωνική επιφάνεια.
textureCoords: καθορίζει αν ενεργοποιείται ή απενεργοποιείται η απόδοση υφής στο εκάστοτε αντικείμενο με τις τιμές GL_TRUE ή GL_FALSE αντίστοιχα.
Η gluQuadricTexture αποδίδει υφή σε όλο το εύρος της τετραγωνικής επιφανείας.

3.6.10.7 Διαχείριση πολλαπλών μητρώων υφής

Αρκετές γραφικές εφαρμογές απαιτούν τη χρήση περισσοτέρων από ένα μητρώο υφής. Π.χ. όταν στη σκηνή υπάρχουν επιφάνειες με διαφορετικά χαρακτηριστικά υφής, απαιτείται η τήρηση πολλαπλών μητρώων υφής.

Η OpenGL επιτρέπει τη διαχείριση πολλαπλών μητρώων υφής αναθέτωντας σε κάθε μητρώο και από ένα αναγνωριστικό αριθμό. Η ανάθεση αναγνωριστικών αριθμών σε

κάθε μητρώο επιτρέπει την ανάκλησή του όποτε είναι αναγκαίο. Αυτή η προσέγγιση είναι πιο αποτελεσματική σε σχέση με την επαναλαμβανόμενη φόρτωση του κάθε μητρώου πριν τη χρήση του.

Αρχικά δημιουργείται ένα πλήθος αναγνωριστικών αριθμών υφής με την εντολή **glGenTextures**:

void glGenTextures(GLsizei n, GLuint *textureID);

textureID: το μητρώο που περιέχει τους αναγνωριστικούς αριθμούς

n: το πλήθος των αναγνωριστικών αριθμών.

Κατόπιν η ανάθεση αναγνωριστικού αριθμού σε ένα μητρώο υφής γίνεται με την εντολή **glBindTexture**.

void glBindTexture(GLenum target, GLuint textureID);

target: παίρνει την τιμή GL_TEXTURE_1D για μονοδιάστατη υφή ή GL_TEXTURE_2D για διδιάστατη υφή.

textureID: αντιστοιχεί στον αναγνωριστικό αριθμό που αναθέτουμε στην τρέχουσα υφή.

Παράδειγμα

```
GLuint textureID[2];
glGenTextures(2,textureID);
glBindTexture(GL_TEXTURE_2D,textureID[0]);
glTexImage2D(.....); //Ορισμός υφής No 1
glTexParameteri(GL_TEXTURE_2D, GL_MIN_FILTER,GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D,GL_MAG_FILTER,GL_LINEAR);
glBindTexture(GL_TEXTURE_2D,textureID[1]);
glTexImage2D(.....) //Ορισμός υφής No 2
glTexParameteri(GL_TEXTURE_2D,GL_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D,GL_MAG_FILTER, GL_NEAREST);
glBindTexture(GL_TEXTURE)
//Κώδικας σχεδιασμού στον οποίο αποδίδεται η υφή No1
glBindTexture(GL_TEXTURE_2D, textureID[1]);
//Κώδικας σχεδιασμού στον οποίο αποδίδεται η υφή No 2
```

Επεξήγηση

Αρχικά δημιουργείται ένας πίνακας δύο στοιχείων textureID και καταχωρείται για την αποθήκευση αναγνωριστικών υφής με την εντολή **glGenTextures**.

Κατόπιν με την πρώτη εντολή **glBindTexture** μεταβαίνει ο προγραμματιστής στην κατάσταση ορισμού/χρήσης της υφής με αναγνωριστικό αριθμό textureID[0]. Με την εντολή **glTexImage2D** καταχωρεί υπό αυτόν τον αναγνωριστικό αριθμό την υφή No 1. Επιπλέον ορίζεται ότι κατά την απόδοση της υφής No 1 εφαρμόζεται η προσέγγιση γραμμικής παρεμβολής.

Στη συνέχεια, με τη δεύτερη εντολή **glBindTexture** μεταβαίνουμε στην κατάσταση ορισμού/χρήσης της υφής με αναγνωριστικό αριθμό textureID[1]. Με τη δεύτερη εντολή **glTexImage2D** καταχωρούμε υπό αυτόν τον αναγνωριστικό αριθμό το μητρώο υφής No 2. Επίσης ορίζεται ότι για το μητρώο υφής No 2 ενεργοποιείται η προσέγγιση πλησιέστερου γείτονα. Κατόπιν, εκτελώντας την τρίτη εντολή **glBindTexture** και δίνοντας το όρισμα textureID[0], επαναφέρει ο χρήστης ως ενεργό υφή την υφή No 1. Επανεκτέλεση της **glBindTexture** με όρισμα τον αναγνωριστικό textureID[1] μας επαναφέρει στην κατάσταση χρήσης της υφής No 2.

Παράδειγμα: Διαχείριση δύο μητρώων υφής

```
// Λειτουργίες φόρτωσης εικόνας παραλείπονται...
GLUquadric *earth;
```

```

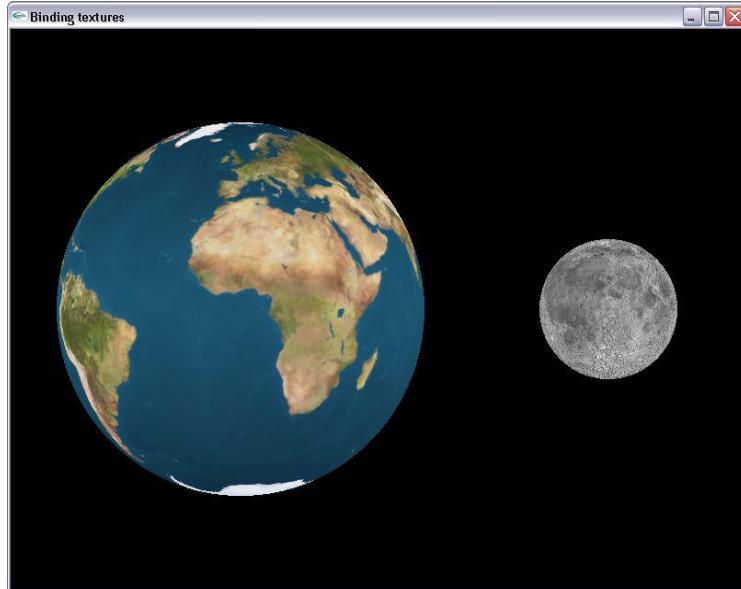
GLUquadric *moon;
GLuint *textureID;
void init()
{
    glutInitWindowPosition(50,50);
    glutInitWindowSize(800,600);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutCreateWindow("Binding textures");
    glMatrixMode(GL_PROJECTION);
    glOrtho(-80,80,-60,60,0,100);
    glClearColor(0,0,0,0);
    glEnable(GL_TEXTURE_2D);
    glEnable(GL_DEPTH_TEST);
    // Φόρτωση Γη και η Σελήνη υφές (Υποθέτοντας ότι η θέση του καταλόγου C:\Πλανήτες)
    Image *earthTexture=new Image();
    ImageLoad("C:\\Planets\\Earth.bmp",earthTexture);
    Image *moonTexture=new Image();
    ImageLoad("C:\\Planets\\Moon.bmp",moonTexture);
    // Παραγωγή δύο αναγνωριστικών υφής
    textureID=new GLuint[2];
    glGenTextures(2,textureID);
    //Initializing quadrics
    earth=gluNewQuadric();
    moon=gluNewQuadric();
    //Enabling automatic texture mapping on quadrics
    gluQuadricTexture(earth,GL_TRUE);
    gluQuadricTexture(moon,GL_TRUE);
    // Δεσμευτική υφή της Γης
    glBindTexture(GL_TEXTURE_2D,textureID[0]);
    glTexImage2D(GL_TEXTURE_2D,0,GL_RGB,earthTexture->sizeX,earthTexture-
    >sizeY,0,GL_RGB,GL_UNSIGNED_BYTE,earthTexture->data);
    // Καθορισμός ιδιοτήτων υφής για textureID [0]
    glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
    glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
    // Δεσμευτική υφή Σελήνης
    glBindTexture(GL_TEXTURE_2D,textureID[1]);
    glTexImage2D(GL_TEXTURE_2D,0,GL_RGB,moonTexture->sizeX,moonTexture-
    >sizeY,0,GL_RGB,GL_UNSIGNED_BYTE,moonTexture->data);
    // Καθορισμός ιδιοτήτων υφής για textureID [1]
    glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
    glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    //Changing globe positioning in the scene
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(-30,0,-50);
    glRotatef(-90,1,0,0);
    // Εφαρμόζοντας την υφή της Γης στην πρώτη σφαίρα
    glBindTexture(GL_TEXTURE_2D,textureID[0]);
    gluQuadricTexture(earth,GL_TRUE);
    gluSphere(earth,40,80,80);
    // Αλλαγή θέσης φεγγάριου στη σκηνή
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(50,0,-50);
    glRotatef(-90,1,0,0);
    // Εφαρμόζοντας την υφή της Σελήνης για τη δεύτερη σφαίρα
}

```

```

glBindTexture(GL_TEXTURE_2D,textureID[1]);
gluSphere(moon,15,80,80);
glFlush();
}
int main(int argc, char **argv)
{
glutInit(&argc,argv);
init();
glutDisplayFunc(display);
glutMainLoop();
return 0;
}

```



Σχήμα 3.44 Διαχείριση δυο μητρώων υφής

3.6.10.8 Αλληλεπίδραση υφής και μοντέλου σκίασης

Όταν ο προγραμματιστής αποδίδει σε μια επιφάνεια τα χαρακτηριστικα μιας υφής και ταυτόχρονα ενεργοποιεί το μοντέλο σκίασης, η τελική απεικόνιση της επιφάνειας καθορίζεται από από τη συνδυασμένη επίδραση των παραμέτρων υφής και των παραμέτρων ανακλαστικότητας. Στην περίπτωση αυτή, ο προγραμματιστής έχει την ευχέρεια να καθορίσει τον τρόπο με τον οποίο αλληλεπιδρούν οι μεταβλητές κατάστασης του χρώματος και της τρέχουσας υφής. Εάν κατά το σχεδιασμό μιας επιφάνειας αποδίδεται σε αυτή ένα ενεργό χρώμα (συντελεστές ανάκλασης) και ένα ενεργό μητρώο υφής η αλληλεπίδραση των μεταβλητών αυτών καθορίζεται με την εντολή **glTexEnv**:

```
void glTexEnv( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
applicationMethod);
```

applicationMethod: ο τρόπος με τον οποίο αλληλεπιδρά η υφή με τις παραμέτρους σκίασης της επιφάνειας. Ορίζονται οι εξής επιλογές:

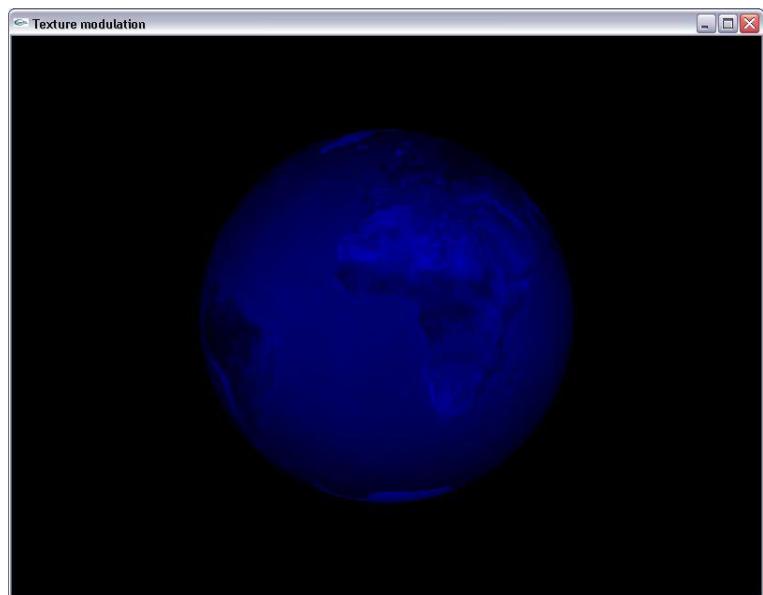
GL_REPLACE: Το χρώμα της υφής αντικαθιστά το χρώμα (συντελεστή ανάκλασης) της επιφανείας

GL_MODULATE: Το χρώμα που ορίζει η υφή πολλαπλασιάζεται με το τρέχον χρώμα της επιφανείας (διαμορφώνει το χρώμα της επιφανείας), συνιστώσα προς συνιστώσα. Αυτός είναι και ο προεπιλεγμένος τρόπος αλληλεπίδρασης υφής και μοντέλου σκίασης.

[56]

Παράδειγμα: Διαμόρφωση γρώματος επιφανείας από στοιχεία υφής

```
#include <stdio.h> // Header file for standard file i/o.  
#include <stdlib.h> // Header file for malloc/free.  
#include <glut.h>  
// Λειτουργίες φόρτωσης εικόνας παραλείπεται  
GLUquadric *sphere;  
void init()  
{  
    glutInitWindowPosition(50,50);  
    glutInitWindowSize(800,600);  
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);  
    glutCreateWindow("Texture modulation");  
    glMatrixMode(GL_PROJECTION);  
    glOrtho(-80,80,-60,60,0,100);  
    glClearColor(0,0,0,0);  
    // Ενεργοποίηση σκίασης  
    glEnable(GL_LIGHTING);  
    GLfloat mat[]={0,0,1};  
    glMaterialfv(GL_FRONT_AND_BACK,GL_AMBIENT_AND_DIFFUSE,mat);  
    // Ενεργοποίηση πηγή φωτός 0 (Εξ 'ορισμού μια μακρινή πηγή φωτός με ακτίνες που ταξιδεύουν κατά μήκος του αρνητικού άξονα z)  
    glEnable(GL_LIGHT0);  
    glEnable(GL_TEXTURE_2D);  
    glEnable(GL_DEPTH_TEST);  
    Image *textureImage=new Image();  
    ImageLoad("C:\\Planets\\Earth.bmp",textureImage);  
    sphere=gluNewQuadric();  
    glTexImage2D(GL_TEXTURE_2D,0,GL_RGB,textureImage->sizeX,textureImage->sizeY,0,GL_RGB,GL_UNSIGNED_BYTE,textureImage->data);  
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);  
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);  
    gluQuadricTexture(sphere,GL_TRUE);  
    glTexEnv(GL_TEXTURE_ENV,GL_TEXTURE_ENV_MODE,GL_MODULATE);  
}  
void display()  
{  
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);  
    // Άλλαγή θέσης υδρογείου στη σκηνή  
    glMatrixMode(GL_MODELVIEW);  
    glLoadIdentity();  
    glTranslatef(0,0,-50);  
    glRotatef(-90,1,0,0);  
    gluSphere(sphere,40,80,80);  
    glFlush();  
}  
int main(int argc, char **argv)  
{  
    glutInit(&argc,argv);  
    init();  
    glutDisplayFunc(display);  
    glutMainLoop();  
    return 0;  
}
```



Σχήμα 3.45 Διαμόρφωση χρώματος επιφανείας από στοιχεία υφής

ΚΕΦΑΛΑΙΟ 4

4.1 Παρουσίαση των βασικών παραδειγμάτων της εφαρμογής και των αντίστοιχων απεικονίσεων τους

4.1.1 SimplePoint (Δημιουργία απλού σημείου)

Κώδικας σε OpenGl

```
#include <stdlib.h>
#include <stdio.h>
#include <glut.h> //OpenGL Graphics Utility Library
//#include "SimpleDraw.h"
//These variables set the dimensions of the rectangular region we wish to view .
const double Xmin = 0.0, Xmax = 3.0;
const double Ymin = 0.0, Ymax = 3.0;
//glutKeyboardFunc is called below to set this function to handle
//all "normal" ascii key presses .
//Only space bar and escape key have an effect .
void myKeyboardFunc( unsigned char key, int x, int y )
{
    if( key == 27 ){
        exit(1); // "27" is theEscape key
    }
}
/*
*drawScene() handles the animation and the redrawing of the
*graphics window contents .
*/
void drawScene(void)
{
    // Clear the rendering window
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    // Set drawing color to white
    glColor3f( 1.0, 1.0, 1.0 );
    // Draw one point
    glBegin(GL_POINTS);
        glVertex2f( 1.0, 1.0 );
    glEnd();
    // Flush the pipeline . (Not usually necessary .)
    glFlush();
}/*end of drawScene*/
//Called when the window is resized
//w, h - width and height of the window in pixels .
void resizeWindow(int w, int h)
{
    double scale, center;
    double windowXmin, windowXmax, windowYmin, windowYmax;
    //Define the portion of the window used for OpenGL rendering .
    glViewport( 0, 0, w, h ); //View port uses whole window
    //Set up the projection view matrix : orthographic projection
    //Determine the min and max values for x and y that should appear in the window .
```

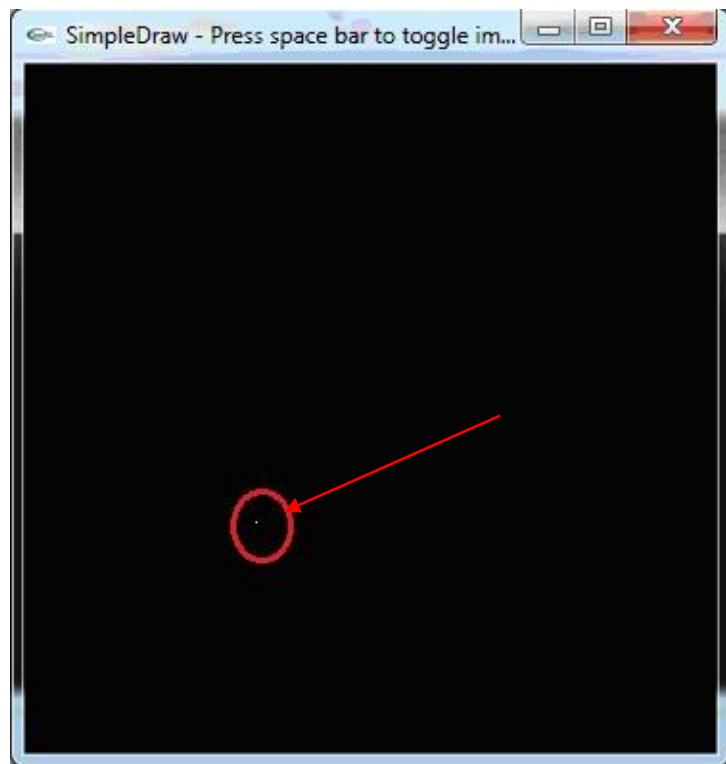
```

//The complication is that the aspect ratio of the window may not match the
//aspect ratio of the scene we want to view .
w = (w==0) ? 1 : w;
h = (h==0) ? 1 : h;
if ( (Xmax-Xmin)/w < (Ymax-Ymin)/h ) {
    scale = ((Ymax-Ymin)/h)/((Xmax-Xmin)/w);
    center = (Xmax+Xmin)/2;
    windowXmin = center - (center-Xmin)*scale;
    windowXmax = center + (Xmax-center)*scale;
    windowYmin = Ymin;
    windowYmax = Ymax;
}
else {
    scale = ((Xmax-Xmin)/w)/((Ymax-Ymin)/h);
    center = (Ymax+Ymin)/2;
    windowYmin = center - (center-Ymin)*scale;
    windowYmax = center + (Ymax-center)*scale;
    windowXmin = Xmin;
    windowXmax = Xmax;
}
//Now that we know the max & min values for x & y that should be visible in the window ,
//we set up the orthographic projection .
glMatrixMode( GL_PROJECTION );
glLoadIdentity();
glOrtho( windowXmin, windowXmax, windowYmin, windowYmax, -1, 1 );
}

//Main routine
//Set up OpenGL , define the callbacks and start the main loop
int main( int argc, char** argv )
{
    glutInit(&argc,argv);
    // The image is not animated so single buffering is OK .
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH );
    // Window position (from top corner) , and size (width and height)
    glutInitWindowPosition( 20, 60 );
    glutInitWindowSize( 360, 360 );
    glutCreateWindow( "SimpleDraw - Press space bar to toggle images" );
    // Initialize OpenGL as we like it ..
    glEnable ( GL_DEPTH_TEST );//initRendering();
    //Set up callback functions for key presses
    glutKeyboardFunc( myKeyboardFunc );      //Handles "normal" ascii symbols
    //glutSpecialFunc( mySpecialKeyFunc ); // Handles "special" keyboard keys
    // Set up the callback function for resizing windows
    // View port uses whole window
    glutReshapeFunc( resizeWindow );
    // call this whenever window needs redrawing
    glutDisplayFunc( drawScene );
    //fprintf(stdout, "Press space bar to toggle images ; escape button to quit.\n");
    // Start the main loop . glutMainLoop never returns .
    glutMainLoop();
    return(0); // This line is never reached .
}

```

Αποτέλεσμα απεικόνισης



Σχήμα 4.1 Απεικόνιση ενός σημείου

4.1.2 SimpleLine (Δημιουργία γραμμής)

Κώδικας σε OpenGl

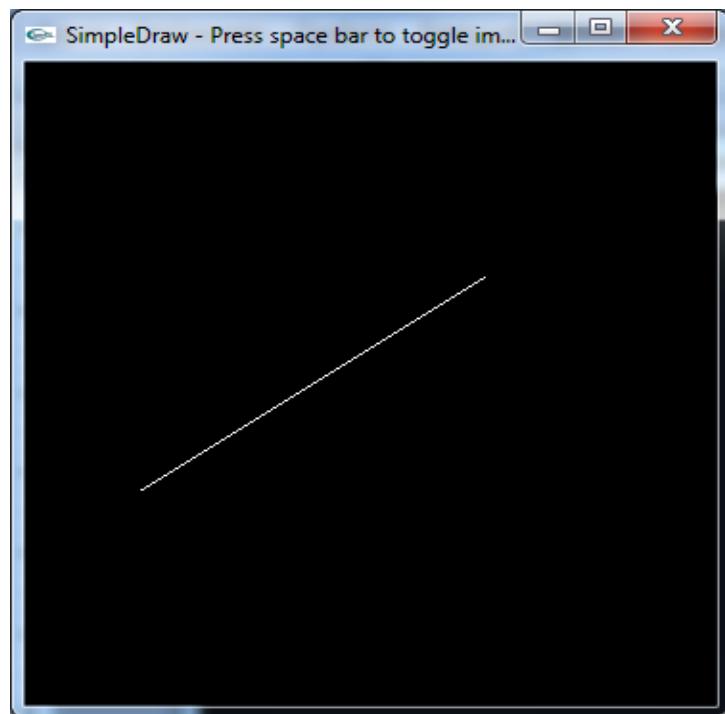
```
#include <stdlib.h>
#include <stdio.h>
#include <glut.h> // OpenGL Graphics Utility Library
//#include "SimpleDraw.h"
//These variables set the dimensions of the rectangular region we wish to view .
const double Xmin = 0.0, Xmax = 3.0;
const double Ymin = 0.0, Ymax = 3.0;
// glutKeyboardFunc is called below to set this function to handle
// all "normal" ascii key presses .
// Only space bar and escape key have an effect .
void myKeyboardFunc( unsigned char key, int x, int y )
{
    if( key == 27 ){
        exit(1); // "27" is the Escape key
    }
}
/*
 * drawScene() handles the animation and the redrawing of the
 * graphics window contents .
*/
void drawScene(void)
{
    // Clear the rendering window
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    // Set drawing color to white
    glColor3f( 1.0, 1.0, 1.0 );
    // Draw one point
    glBegin( GL_LINES );
        glVertex2f( 0.5, 1.0 );
        glVertex2f( 2.0, 2.0 );
    glEnd();
    // Flush the pipeline . (Not usually necessary .)
    glFlush();
}/*end of drawScene*/
// Called when the window is resized
// w, h - width and height of the window in pixels .
void resizeWindow(int w, int h)
{
    double scale, center;
    double windowXmin, windowXmax, windowYmin, windowYmax;
    // Define the portion of the window used for OpenGL rendering .
    glViewport( 0, 0, w, h );// View port uses whole window
    // Set up the projection view matrix : orthographic projection
    // Determine the min and max values for x and y that should appear in the window .
    // The complication is that the aspect ratio of the window may not match the
    // aspect ratio of the scene we want to view .
    w = (w==0) ? 1 : w;
    h = (h==0) ? 1 : h;
    if( (Xmax-Xmin)/w < (Ymax-Ymin)/h ) {
        scale = ((Ymax-Ymin)/h)/((Xmax-Xmin)/w);
        center = (Xmax+Xmin)/2;
        windowXmin = center - (center-Xmin)*scale;
        windowXmax = center + (Xmax-center)*scale;
        windowYmin = Ymin;
```

```

        windowYmax = Ymax;
    }
    else {
        scale = ((Xmax-Xmin)/w)/((Ymax-Ymin)/h);
        center = (Ymax+Ymin)/2;
        windowYmin = center - (center-Ymin)*scale;
        windowYmax = center + (Ymax-center)*scale;
        windowXmin = Xmin;
        windowXmax = Xmax;
    }
    //Now that we know the max & min values for x & y that should be visible in the window ,
    //we set up the orthographic projection .
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    glOrtho( windowXmin, windowXmax, windowYmin, windowYmax, -1, 1 );
}
// Main routine
// Set up OpenGL , define the callbacks and start the main loop
int main( int argc, char** argv )
{
    glutInit(&argc,argv);
    // The image is not animated so single buffering is OK .
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH );
    // Window position (from top corner), and size (width and height)
    glutInitWindowPosition( 20, 60 );
    glutInitWindowSize( 360, 360 );
    glutCreateWindow( "SimpleDraw - Press space bar to toggle images" );
    // Initialize OpenGL as we like it..
    glEnable ( GL_DEPTH_TEST ); //initRendering();
    // Set up callback functions for key presses
    glutKeyboardFunc( myKeyboardFunc ); // Handles "normal" ascii symbols
    //glutSpecialFunc( mySpecialKeyFunc ); // Handles "special" keyboard keys
    // Set up the callback function for resizing windows
    // View port uses whole window
    glutReshapeFunc( resizeWindow );
    // call this whenever window needs redrawing
    glutDisplayFunc( drawScene );
    // fprintf(stdout, "Press space bar to toggle images ; escape button to quit.\n");
    // Start the main loop . glutMainLoop never returns .
    glutMainLoop( );
    return(0); // This line is never reached .
}

```

Αποτέλεσμα απεικόνισης



Σχήμα 4.2: Απεικόνιση γραμμής

4.1.3 LineStrip (Ενωση διαδοχικών σημείων μεταξύ τους, εκτός του αρχικού και τελικού σημείου)

Κώδικας σε OpenGL

```
#include <stdlib.h>
#include <stdio.h>
#include <glut.h> // OpenGL Graphics Utility Library
//#include "SimpleDraw.h"
// These variables set the dimensions of the rectangular region we wish to view .
const double Xmin = 0.0, Xmax = 3.0;
const double Ymin = 0.0, Ymax = 3.0;
// glutKeyboardFunc is called below to set this function to handle
// all "normal" ascii key presses .
// Only space bar and escape key have an effect .
void myKeyboardFunc( unsigned char key, int x, int y )
{
    if( key == 27 ){
        exit(1); // "27" is theEscape key
    }
}
/*
 * drawScene() handles the animation and the redrawing of the
 * graphics window contents .
*/
void drawScene(void)
{
    // Clear the rendering window
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    // Set drawing color to white
    glColor3f( 1.0, 1.0, 1.0 );
    // Draw one point
    glBegin( GL_LINE_STRIP );
        glVertex2f( 0.5, 1.0 );
        glVertex2f( 2.0, 2.0 );
        glVertex2f( 1.8, 2.6 );
        glVertex2f( 0.7, 2.2 );
        glVertex2f( 1.6, 1.2 );
        glVertex2f( 1.0, 0.5 );
    glEnd();
    // Flush the pipeline . (Not usually necessary .)
    glFlush();
}/*end of drawScene*/
// Called when the window is resized
//           w, h - width and height of the window in pixels .
void resizeWindow(int w, int h)
{
    double scale, center;
    double windowXmin, windowXmax, windowYmin, windowYmax;
    // Define the portion of the window used for OpenGL rendering .
    glViewport( 0, 0, w, h ); // View port uses whole window
    // Set up the projection view matrix: orthographic projection
    // Determine the min and max values for x and y that should appear in the window.
    // The complication is that the aspect ratio of the window may not match the
    // aspect ratio of the scene we want to view .
    w = (w==0) ? 1 : w;
    h = (h==0) ? 1 : h;
    if( (Xmax-Xmin)/w < (Ymax-Ymin)/h ) {
        scale = ((Ymax-Ymin)/h)/((Xmax-Xmin)/w);
```

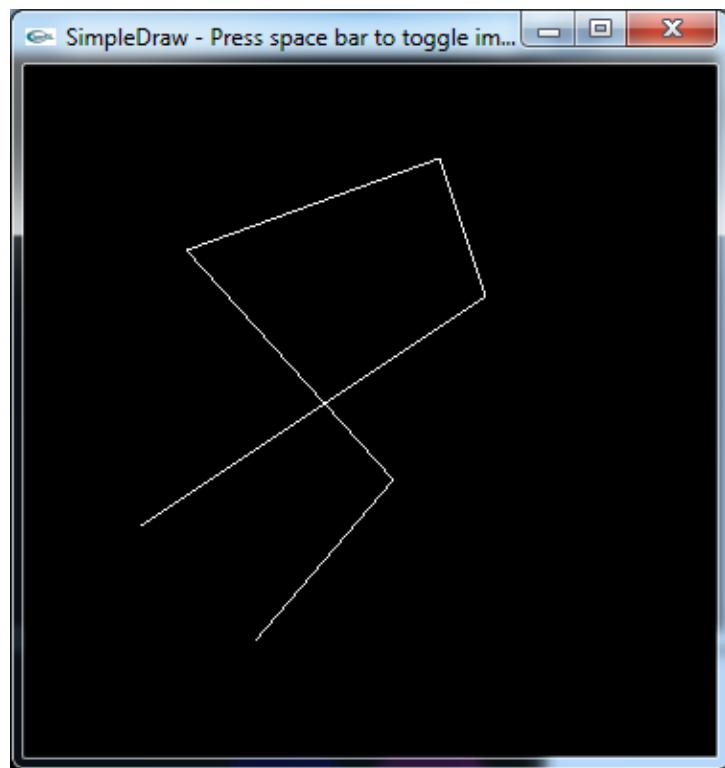
```

center = (Xmax+Xmin)/2;
windowXmin = center - (center-Xmin)*scale;
windowXmax = center + (Xmax-center)*scale;
windowYmin = Ymin;
windowYmax = Ymax;
}
else {
    scale = ((Xmax-Xmin)/w)/((Ymax-Ymin)/h);
    center = (Ymax+Ymin)/2;
    windowYmin = center - (center-Ymin)*scale;
    windowYmax = center + (Ymax-center)*scale;
    windowXmin = Xmin;
    windowXmax = Xmax;
}
//Now that we know the max & min values for x & y that should be visible in the window ,
//we set up the orthographic projection .
glMatrixMode( GL_PROJECTION );
glLoadIdentity();
glOrtho( windowXmin, windowXmax, windowYmin, windowYmax, -1, 1 );
}

// Main routine
// Set up OpenGL , define the callbacks and start the main loop
int main( int argc, char** argv )
{
    glutInit(&argc,argv);
    // The image is not animated so single buffering is OK.
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH );
    // Window position (from top corner), and size (width and height)
    glutInitWindowPosition( 20, 60 );
    glutInitWindowSize( 360, 360 );
    glutCreateWindow( "SimpleDraw - Press space bar to toggle images" );
    // Initialize OpenGL as we like it.
    glEnable ( GL_DEPTH_TEST );//initRendering();
    // Set up callback functions for key presses
    glutKeyboardFunc( myKeyboardFunc );    //Handles "normal" ascii symbols
    // glutSpecialFunc( mySpecialKeyFunc ); //Handles "special" keyboard keys
    // Set up the callback function for resizing windows
    // View port uses whole window
    glutReshapeFunc( resizeWindow );
    // call this whenever window needs redrawing
    glutDisplayFunc( drawScene );
    //fprintf(stdout, "Press space bar to toggle images; escape button to quit.\n");
    // Start the main loop . glutMainLoop never returns .
    glutMainLoop( );
    return(0);           // This line is never reached .
}

```

Αποτέλεσμα απεικόνισης



Σχήμα 4.3: Απεικόνιση της ένωσης πολλών διαδοχικών σημείων (μη ένωση αρχής και τέλους)

4.1.4 LineLoop (Ενωση διαδοχικών σημείων μεταξύ τους όπως επίσης ένωση του αρχικού και τελικού σημείου τους)

Κώδικας σε OpenGL

```
#include <stdlib.h>
#include <stdio.h>
#include <GL/glut.h> // OpenGL Graphics Utility Library
//#include "SimpleDraw.h"
// These variables set the dimensions of the rectangular region we wish to view .
const double Xmin = 0.0, Xmax = 3.0;
const double Ymin = 0.0, Ymax = 3.0;
// glutKeyboardFunc is called below to set this function to handle
// all "normal" ascii key presses .
// Only space bar and escape key have an effect .
void myKeyboardFunc( unsigned char key, int x, int y )
{
    if( key == 27 ){
        exit(1); // "27" is theEscape key
    }
}
/*
 * drawScene() handles the animation and the redrawing of the
 * graphics window contents .
*/
void drawScene(void)
{
    // Clear the rendering window
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    // Set drawing color to white
    glColor3f( 1.0, 1.0, 1.0 );
    // Draw one point
    glBegin( GL_LINE_LOOP );
        glVertex2f( 0.5, 1.0 );
        glVertex2f( 2.0, 2.0 );
        glVertex2f( 1.8, 2.6 );
        glVertex2f( 0.7, 2.2 );
        glVertex2f( 1.6, 1.2 );
        glVertex2f( 1.0, 0.5 );
    glEnd();
    // Flush the pipeline . (Not usually necessary .)
    glFlush();
} /*end of drawScene*/
// Called when the window is resized
// w, h - width and height of the window in pixels .
void resizeWindow(int w, int h)
{
    double scale, center;
    double windowXmin, windowXmax, windowYmin, windowYmax;
    // Define the portion of the window used for OpenGL rendering .
    glViewport( 0, 0, w, h ); // View port uses whole window
    // Set up the projection view matrix : orthographic projection
    // Determine the min and max values for x and y that should appear in the window .
    // The complication is that the aspect ratio of the window may not match the
    // aspect ratio of the scene we want to view .
    w = (w==0) ? 1 : w;
    h = (h==0) ? 1 : h;
    if( ( Xmax-Xmin)/w < (Ymax-Ymin)/h ) {
        scale = ((Ymax-Ymin)/h)/((Xmax-Xmin)/w);
```

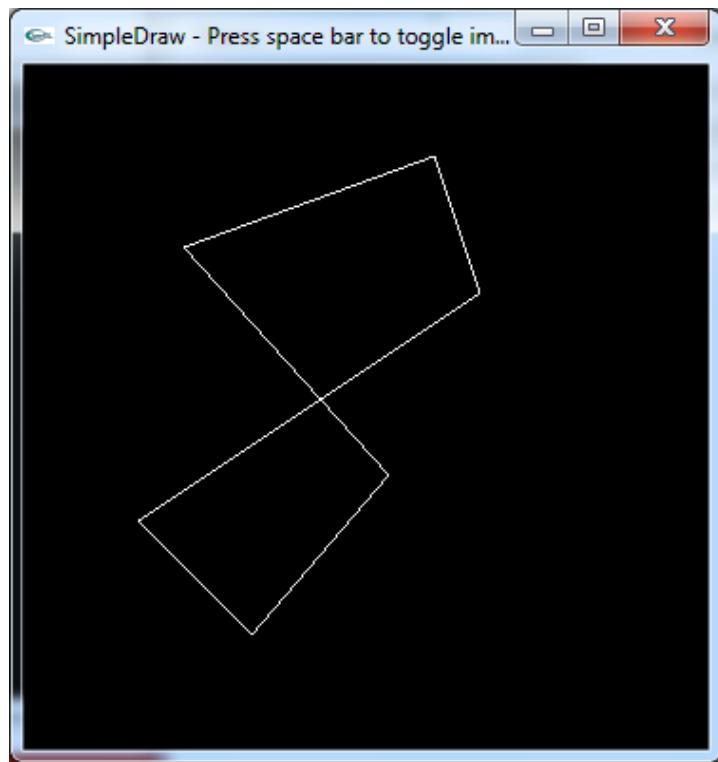
```

        center = (Xmax+Xmin)/2;
        windowXmin = center - (center-Xmin)*scale;
        windowXmax = center + (Xmax-center)*scale;
        windowYmin = Ymin;
        windowYmax = Ymax;
    }
    else {
        scale = ((Xmax-Xmin)/w)/((Ymax-Ymin)/h);
        center = (Ymax+Ymin)/2;
        windowYmin = center - (center-Ymin)*scale;
        windowYmax = center + (Ymax-center)*scale;
        windowXmin = Xmin;
        windowXmax = Xmax;
    }
    // Now that we know the max & min values for x & y that should be visible in the window ,
    //we set up the orthographic projection .
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    glOrtho( windowXmin, windowXmax, windowYmin, windowYmax, -1, 1 );
}

// Main routine
// Set up OpenGL, define the callbacks and start the main loop
int main( int argc, char** argv )
{
    glutInit(&argc,argv);
    // The image is not animated so single buffering is OK .
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH );
    // Window position (from top corner) , and size (width and height)
    glutInitWindowPosition( 20, 60 );
    glutInitWindowSize( 360, 360 );
    glutCreateWindow( "SimpleDraw - Press space bar to toggle images" );
    // Initialize OpenGL as we like it.
    glEnable ( GL_DEPTH_TEST );//initRendering() ;
    // Set up callback functions for key presses
    glutKeyboardFunc( myKeyboardFunc );    //Handles "normal" ascii symbols
    // glutSpecialFunc( mySpecialKeyFunc ); //Handles "special" keyboard keys
    // Set up the callback function for resizing windows
    // View port uses whole window
    glutReshapeFunc( resizeWindow );
    // call this whenever window needs redrawing
    glutDisplayFunc( drawScene );
    //fprintf( stdout , "Press space bar to toggle images ; escape button to quit.\n" );
    // Start the main loop . glutMainLoop never returns .
    glutMainLoop();
    return(0);           // This line is never reached .
}

```

Αποτέλεσμα απεικόνισης



Σχήμα 4.4: Απεικόνιση της ένωσης όλων των διαδοχικών σημείων μεταξύ τους

4.1.5 SimpleTriangle (Δημιουργία τριγώνου)

Κώδικας σε OpenGl

```
#include <stdlib.h>
#include <stdio.h>
#include <glut.h> // OpenGL Graphics Utility Library
// #include "SimpleDraw.h"
// These variables set the dimensions of the rectangular region we wish to view .
const double Xmin = 0.0, Xmax = 3.0;
const double Ymin = 0.0, Ymax = 3.0;
// glutKeyboardFunc is called below to set this function to handle
// all "normal" ascii key presses .
// Only space bar and escape key have an effect .
void myKeyboardFunc( unsigned char key, int x, int y )
{
    if( key == 27 ){
        exit(1); //"27" is theEscape key
    }
}
/*
 * drawScene() handles the animation and the redrawing of the
 *           graphics window contents .
*/
void drawScene(void)
{
    // Clear the rendering window
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    // Draw one point
    glBegin( GL_TRIANGLES );
        glColor3f( 1.0, 1.0, 1.0 );
        glVertex3f( 0.3, 1.0, 0.5 );
        glVertex3f( 2.7, 0.85, 0.0 );
        glVertex3f( 2.7, 1.15, 0.0 );
    glEnd();
    // Flush the pipeline . (Not usually necessary .)
    glFlush();
}/*end of drawScene*/
// Called when the window is resized
// w, h - width and height of the window in pixels .
void resizeWindow(int w, int h)
{
    double scale, center;
    double windowXmin, windowXmax, windowYmin, windowYmax;
    // Define the portion of the window used for OpenGL rendering .
    glViewport( 0, 0, w, h ); // View port uses whole window
    // Set up the projection view matrix : orthographic projection
    // Determine the min and max values for x and y that should appear in the window .
    // The complication is that the aspect ratio of the window may not match the
    // aspect ratio of the scene we want to view .
    w = (w==0) ? 1 : w;
    h = (h==0) ? 1 : h;
    if( (Xmax-Xmin)/w < (Ymax-Ymin)/h ) {
        scale = ((Ymax-Ymin)/h)/((Xmax-Xmin)/w);
        center = (Xmax+Xmin)/2;
        windowXmin = center - (center-Xmin)*scale;
        windowXmax = center + (Xmax-center)*scale;
        windowYmin = Ymin;
    }
}
```

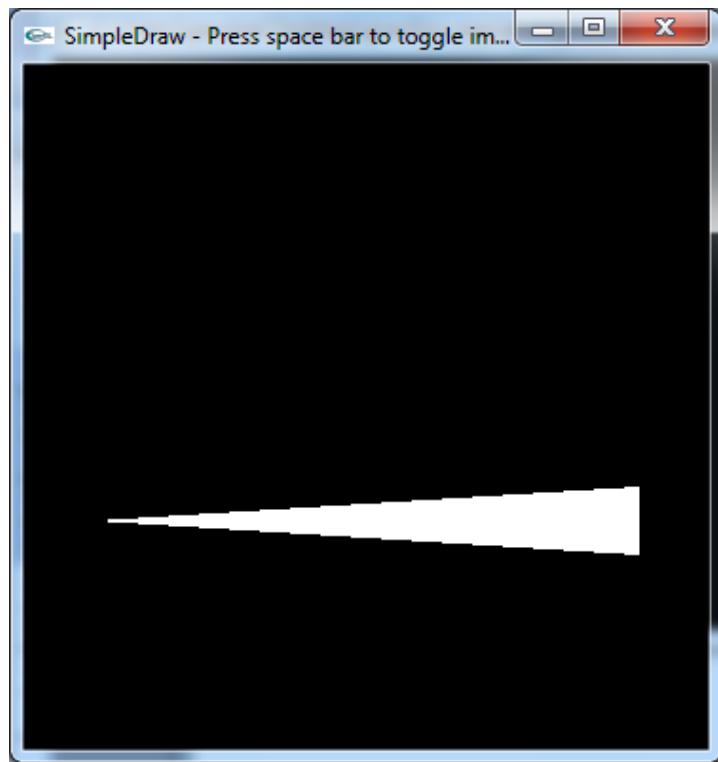
```

        windowYmax = Ymax;
    }
    else {
        scale = ((Xmax-Xmin)/w)/((Ymax-Ymin)/h);
        center = (Ymax+Ymin)/2;
        windowYmin = center - (center-Ymin)*scale;
        windowYmax = center + (Ymax-center)*scale;
        windowXmin = Xmin;
        windowXmax = Xmax;
    }
    // Now that we know the max & min values for x & y that should be visible in the window ,
    //we set up the orthographic projection .
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    glOrtho( windowXmin, windowXmax, windowYmin, windowYmax, -1, 1 );
}
// Main routine
// Set up OpenGL , define the callbacks and start the main loop
int main( int argc, char** argv )
{
    glutInit(&argc,argv);
    // The image is not animated so single buffering is OK .
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH );
    // Window position (from top corner), and size (width and height)
    glutInitWindowPosition( 20, 60 );
    glutInitWindowSize( 360, 360 );
    glutCreateWindow( "SimpleDraw - Press space bar to toggle images" );
    // Initialize OpenGL as we like it..
    glEnable ( GL_DEPTH_TEST );//initRendering();

    // Set up callback functions for key presses
    glutKeyboardFunc( myKeyboardFunc );      //Handles "normal" ascii symbols
    //glutSpecialFunc( mySpecialKeyFunc ); //Handles "special" keyboard keys
    // Set up the callback function for resizing windows
    // View port uses whole window
    glutReshapeFunc( resizeWindow );
    // call this whenever window needs redrawing
    glutDisplayFunc( drawScene );
    //fprintf(stdout , "Press space bar to toggle images ; escape button to quit.\n");
    // Start the main loop . glutMainLoop never returns .
    glutMainLoop( );
    return(0);           // This line is never reached .
}

```

Αποτέλεσμα απεικόνισης



Σχήμα 4.5: Απεικόνιση τριγώνου

4.1.6 OpenGLFont()

```
#include <string.h>
#include <glut.h>

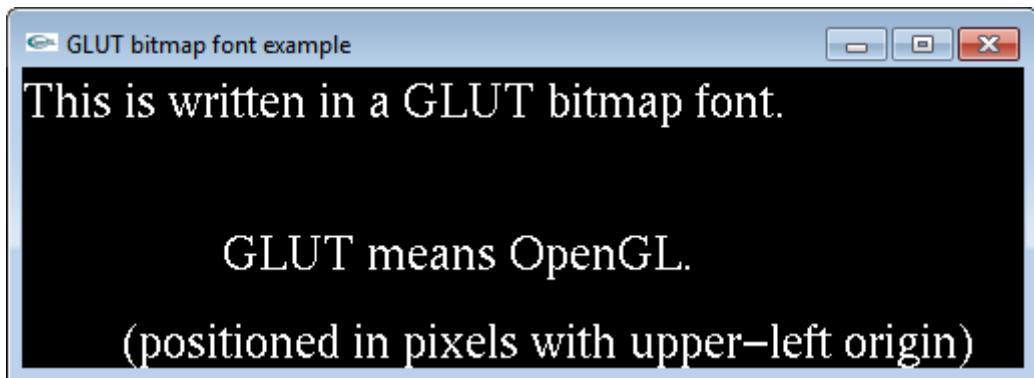
void *font = GLUT_BITMAP_TIMES_ROMAN_24;
void *fonts[] =
{
    GLUT_BITMAP_9_BY_15,
    GLUT_BITMAP_TIMES_ROMAN_10,
    GLUT_BITMAP_TIMES_ROMAN_24
};
char defaultMessage[] = "GLUT means OpenGL.";
char *message = defaultMessage;
void
selectFont(int newfont)
{
    font = fonts[newfont];
    glutPostRedisplay();
}
void
selectMessage(int msg)
{
    switch (msg) {
        case 1:
            message = "abcdefghijklmnopqrstuvwxyz";
            break;
        case 2:
            message = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
            break;
    }
}
void
selectColor(int color)
{
    switch (color) {
        case 1:
            glColor3f(0.0, 1.0, 0.0);
            break;
        case 2:
            glColor3f(1.0, 0.0, 0.0);
            break;
        case 3:
            glColor3f(1.0, 1.0, 1.0);
            break;
    }
    glutPostRedisplay();
}
void
tick(void)
{
    glutPostRedisplay();
}
void
output(int x, int y, char *string)
{
    int len, i;
    glRasterPos2f(x, y);
    len = (int) strlen(string);
```

```

for (i = 0; i < len; i++) {
    glutBitmapCharacter(font, string[i]);
}
}
void
display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    output(0, 24, "This is written in a GLUT bitmap font.");
    output(100, 100, message);
    output(50, 145, "(positioned in pixels with upper-left origin)");
    glutSwapBuffers();
}
void
reshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, w, h, 0);
    glMatrixMode(GL_MODELVIEW);
}
int
main(int argc, char **argv)
{
    int i, msg_submenu, color_submenu;
    glutInit(&argc, argv);
    for (i = 1; i < argc; i++) {
        if (!strcmp(argv[i], "-mono")) {
            font = GLUT_BITMAP_9_BY_15;
        }
    }
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(500, 150);
    glutCreateWindow("GLUT bitmap font example");
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutIdleFunc(tick);
    msg_submenu = glutCreateMenu(selectMessage);
    glutAddMenuEntry("abc", 1);
    glutAddMenuEntry("ABC", 2);
    color_submenu = glutCreateMenu(selectColor);
    glutAddMenuEntry("Green", 1);
    glutAddMenuEntry("Red", 2);
    glutAddMenuEntry("White", 3);
    glutCreateMenu(selectFont);
    glutAddMenuEntry("9 by 15", 0);
    glutAddMenuEntry("Times Roman 10", 1);
    glutAddMenuEntry("Times Roman 24", 2);
    glutAddSubMenu("Messages", msg_submenu);
    glutAddSubMenu("Color", color_submenu);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
    glutMainLoop();
    return 0; /* ANSI C requires main to return int . */
}

```

Αποτέλεσμα απεικόνισης



Σχήμα 4.6: Απεικόνιση φόντου με κείμενο

4.1.7 ColorChange(Παράδειγμα αλλαγής χρώματος)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>
#include <GL/glut.h>

#define MAXOJBS 10000
#define MAXSELECT 100
#define MAXFEED 300
#define SOLID 1
#define LINE 2
#define POINT 3

GLint windW = 300, windH = 300;
GLuint selectBuf[MAXSELECT];
GLfloat feedBuf[MAXFEED];
GLint vp[4];
float zRotation = 90.0;
float zoom = 1.0;
GLint objectCount;
GLint numObjects;
struct object {
    float v1[2];
    float v2[2];
    float v3[2];
    float color[3];
} objects[MAXOJBS];
GLenum linePoly = GL_FALSE;
static void
InitObjects(GLint num)
{
    GLint i;
    float x, y;
    if (num > MAXOJBS) {
        num = MAXOJBS;
    }
    if (num < 1) {
        num = 1;
    }
    objectCount = num;
    srand((unsigned int) time(NULL));
    for (i = 0; i < num; i++) {
        x = (rand() % 300) - 150;
        y = (rand() % 300) - 150;
        objects[i].v1[0] = x + (rand() % 50) - 25;
        objects[i].v2[0] = x + (rand() % 50) - 25;
        objects[i].v3[0] = x + (rand() % 50) - 25;
        objects[i].v1[1] = y + (rand() % 50) - 25;
        objects[i].v2[1] = y + (rand() % 50) - 25;
        objects[i].v3[1] = y + (rand() % 50) - 25;
        objects[i].color[0] = ((rand() % 100) + 50) / 150.0;
        objects[i].color[1] = ((rand() % 100) + 50) / 150.0;
        objects[i].color[2] = ((rand() % 100) + 50) / 150.0;
    }
}
static void
Init(void)
```

```

{
    numObjects = 10;
    InitObjects(numObjects);
}

static void
Reshape(int width, int height)
{
    windW = width;
    windH = height;
    glViewport(0, 0, windW, windH);
    glGetIntegerv(GL_VIEWPORT, vp);
}

static void
Render(GLenum mode)
{
    GLint i;
    for (i = 0; i < objectCount; i++) {
        if (mode == GL_SELECT) {
            glLoadName(i);
        }
        glColor3fv(objects[i].color);
        glBegin(GL_POLYGON);
        glVertex2fv(objects[i].v1);
        glVertex2fv(objects[i].v2);
        glVertex2fv(objects[i].v3);
        glEnd();
    }
}

static GLint
DoSelect(GLint x, GLint y)
{
    GLint hits;
    glSelectBuffer(MAXSELECT, selectBuf);
    glRenderMode(GL_SELECT);
    glInitNames();
    glPushName(~0);
    glPushMatrix();
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPickMatrix(x, windH - y, 4, 4, vp);
    gluOrtho2D(-175, 175, -175, 175);
    glMatrixMode(GL_MODELVIEW);
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glScalef(zoom, zoom, zoom);
    glRotatef(zRotation, 0, 0, 1);
    Render(GL_SELECT);
    glPopMatrix();
    hits = glRenderMode(GL_RENDER);
    if (hits <= 0) {
        return -1;
    }
    return selectBuf[(hits - 1) * 4 + 3];
}

static void
RecolorTri(GLint h)
{
    objects[h].color[0] = ((rand() % 100) + 50) / 150.0;
    objects[h].color[1] = ((rand() % 100) + 50) / 150.0;
    objects[h].color[2] = ((rand() % 100) + 50) / 150.0;
}

```

```

    }
static void
DeleteTri(GLint h)
{
    objects[h] = objects[objectCount - 1];
    objectCount--;
}
static void
GrowTri(GLint h)
{
    float v[2];
    float *oldV;
    GLint i;
    v[0] = objects[h].v1[0] + objects[h].v2[0] + objects[h].v3[0];
    v[1] = objects[h].v1[1] + objects[h].v2[1] + objects[h].v3[1];
    v[0] /= 3;
    v[1] /= 3;
    for (i = 0; i < 3; i++) {
        switch (i) {
            case 0:
                oldV = objects[h].v1;
                break;
            case 1:
                oldV = objects[h].v2;
                break;
            case 2:
                oldV = objects[h].v3;
                break;
        }
        oldV[0] = 1.5 * (oldV[0] - v[0]) + v[0];
        oldV[1] = 1.5 * (oldV[1] - v[1]) + v[1];
    }
}
static void
Mouse(int button, int state, int mouseX, int mouseY)
{
    GLint hit;
    if (state == GLUT_DOWN) {
        hit = DoSelect((GLint) mouseX, (GLint) mouseY);
        if (hit != -1) {
            if (button == GLUT_LEFT_BUTTON) {
                RecolorTri(hit);
            } else if (button == GLUT_MIDDLE_BUTTON) {
                GrowTri(hit);
            } else if (button == GLUT_RIGHT_BUTTON) {
                DeleteTri(hit);
            }
            glutPostRedisplay();
        }
    }
}
static void
Draw(void)
{
    glPushMatrix();
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-175, 175, -175, 175);
    glMatrixMode(GL_MODELVIEW);
    glClearColor(0.0, 0.0, 0.0, 0.0);
}

```

```

glClear(GL_COLOR_BUFFER_BIT);
glScalef(zoom, zoom, zoom);
glRotatef(zRotation, 0, 0, 1);
Render(GL_RENDER);
glPopMatrix();
glutSwapBuffers();
}
static void
DumpFeedbackVert(GLint * i, GLint n)
{
    GLint index;
    index = *i;
    if (index + 7 > n) {
        *i = n;
        printf(" ???\n");
        return;
    }
    printf(" (%g %g %g), color = (%4.2f %4.2f %4.2f)\n",
        feedBuf[index],
        feedBuf[index + 1],
        feedBuf[index + 2],
        feedBuf[index + 3],
        feedBuf[index + 4],
        feedBuf[index + 5]);
    index += 7;
    *i = index;
}
static void
DrawFeedback(GLint n)
{
    GLint i;
    GLint verts;
    printf("Feedback results (%d floats):\n", n);
    for (i = 0; i < n; i++) {
        switch ((GLint) feedBuf[i]) {
            case GL_POLYGON_TOKEN:
                printf("Polygon");
                i++;
                if (i < n) {
                    verts = (GLint) feedBuf[i];
                    i++;
                    printf(": %d vertices", verts);
                } else {
                    verts = 0;
                }
                printf("\n");
                while (verts) {
                    DumpFeedbackVert(&i, n);
                    verts--;
                }
                i--;
                break;
            case GL_LINE_TOKEN:
                printf("Line:\n");
                i++;
                DumpFeedbackVert(&i, n);
                DumpFeedbackVert(&i, n);
                i--;
                break;
            case GL_LINE_RESET_TOKEN:

```

```

printf("Line Reset:\n");
i++;
DumpFeedbackVert(&i, n);
DumpFeedbackVert(&i, n);
i--;
break;
default:
printf("%9.2f\n", feedBuf[i]);
break;
}
}
if (i == MAXFEED) {
printf("...\n");
}
printf("\n");
}
static void
DoFeedback(void)
{
GLint x;

glFeedbackBuffer(MAXFEED, GL_3D_COLOR, feedBuf);
(void) glRenderMode(GL_FEEDBACK);
glPushMatrix();
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(-175, 175, -175, 175);
glMatrixMode(GL_MODELVIEW);

glClearColor(0.0, 0.0, 0.0, 0.0);
glClear(GL_COLOR_BUFFER_BIT);
glScalef(zoom, zoom, zoom);
glRotatef(zRotation, 0, 0, 1);
Render(GL_FEEDBACK);
glPopMatrix();
x = glRenderMode(GL_RENDER);
if (x == -1) {
x = MAXFEED;
}
DrawFeedback((GLint) x);
}
/* ARGSUSED1 */
static void
Key(unsigned char key, int x, int y)
{
switch (key) {
case 'z':
zoom /= 0.75;
glutPostRedisplay();
break;
case 'Z':
zoom *= 0.75;
glutPostRedisplay();
break;
case 'f':
DoFeedback();
glutPostRedisplay();
break;
case 'T':
linePoly = !linePoly;
}
}

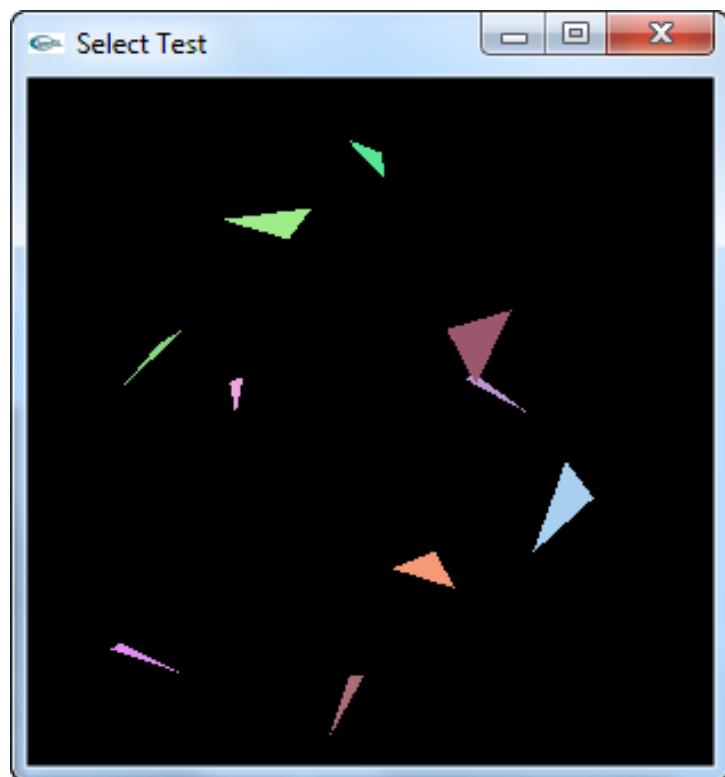
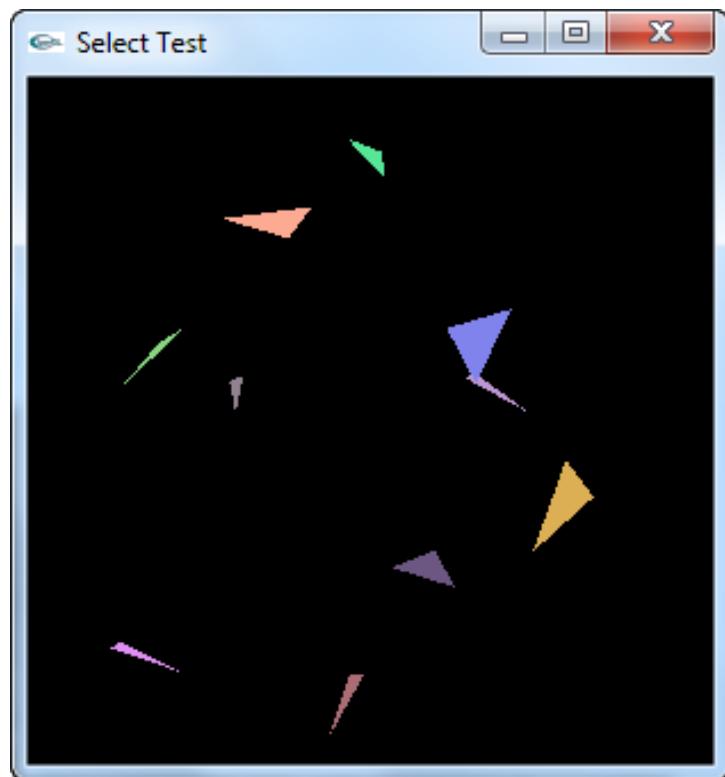
```

```

if (linePoly) {
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
} else {
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
}
glutPostRedisplay();
break;
case 27:
    exit(0);
}
/* ARGSUSED1 */
static void
SpecialKey(int key, int x, int y)
{
    switch (key) {
        case GLUT_KEY_LEFT:
            zRotation += 0.5;
            glutPostRedisplay();
            break;
        case GLUT_KEY_RIGHT:
            zRotation -= 0.5;
            glutPostRedisplay();
            break;
    }
}
int
main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);
    glutCreateWindow("Select Test");
    Init();
    glutReshapeFunc(Reshape);
    glutKeyboardFunc(Key);
    glutSpecialFunc(SpecialKey);
    glutMouseFunc(Mouse);
    glutDisplayFunc(Draw);
    glutMainLoop();
    return 0;          /* ANSI C requires main to return int. */
}

```

Αποτέλεσμα απεικόνισης



Σχήμα 4.7 και Σχήμα 4.8: ColorChange (Παράδειγμα αλλαγής χρώματος - όταν πατάει ο χρήστης πάνω στα τρίγωνα αλλάζουν χρώμα)

4.1.8 Cube (Δημιουργία κύβου με χρώματα)

Κώδικας σε OpenGl

```
#include <glut.h>

GLfloat n[6][3] = { /* Normals for the 6 faces of a cube . */
    {-1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {1.0, 0.0, 0.0},
    {0.0, -1.0, 0.0}, {0.0, 0.0, 1.0}, {0.0, 0.0, -1.0} };
GLint faces[6][4] = { /* Vertex indices for the 6 faces of a cube . */
    {0, 1, 2, 3}, {3, 2, 6, 7}, {7, 6, 5, 4},
    {4, 5, 1, 0}, {5, 6, 2, 1}, {7, 4, 0, 3} };
GLfloat v[8][3]; /* Will be filled in with X,Y,Z vertexes . */
void drawScene(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glBegin(GL_QUADS);
        glNormal3fv(&n[0][0]);
        glColor3f(1.0f, 1.0f, 0.0f);
        glVertex3fv(&v[faces[0][0]][0]);
        glVertex3fv(&v[faces[0][1]][0]);
        glVertex3fv(&v[faces[0][2]][0]);
        glVertex3fv(&v[faces[0][3]][0]);
    glEnd();
    glBegin(GL_QUADS);
        glNormal3fv(&n[1][0]);
        glColor3f(1.0f, 0.0f, 0.0f);
        glVertex3fv(&v[faces[1][0]][0]);
        glVertex3fv(&v[faces[1][1]][0]);
        glVertex3fv(&v[faces[1][2]][0]);
        glVertex3fv(&v[faces[1][3]][0]);
    glEnd();
    glBegin(GL_QUADS);
        glNormal3fv(&n[2][0]);
        glColor3f(0.0f, 1.0f, 0.0f);
        glVertex3fv(&v[faces[2][0]][0]);
        glVertex3fv(&v[faces[2][1]][0]);
        glVertex3fv(&v[faces[2][2]][0]);
        glVertex3fv(&v[faces[2][3]][0]);
    glEnd();
    glBegin(GL_QUADS);
        glNormal3fv(&n[3][0]);
        glColor3f(1.0f, 0.0f, 1.0f);
        glVertex3fv(&v[faces[3][0]][0]);
        glVertex3fv(&v[faces[3][1]][0]);
        glVertex3fv(&v[faces[3][2]][0]);
        glVertex3fv(&v[faces[3][3]][0]);
    glEnd();
    glBegin(GL_QUADS);
        glNormal3fv(&n[4][0]);
        glColor3f(1.0f, 1.0f, 1.0f);
        glVertex3fv(&v[faces[4][0]][0]);
        glVertex3fv(&v[faces[4][1]][0]);
        glVertex3fv(&v[faces[4][2]][0]);
        glVertex3fv(&v[faces[4][3]][0]);
    glEnd();
    glBegin(GL_QUADS);
        glNormal3fv(&n[5][0]);
```

```

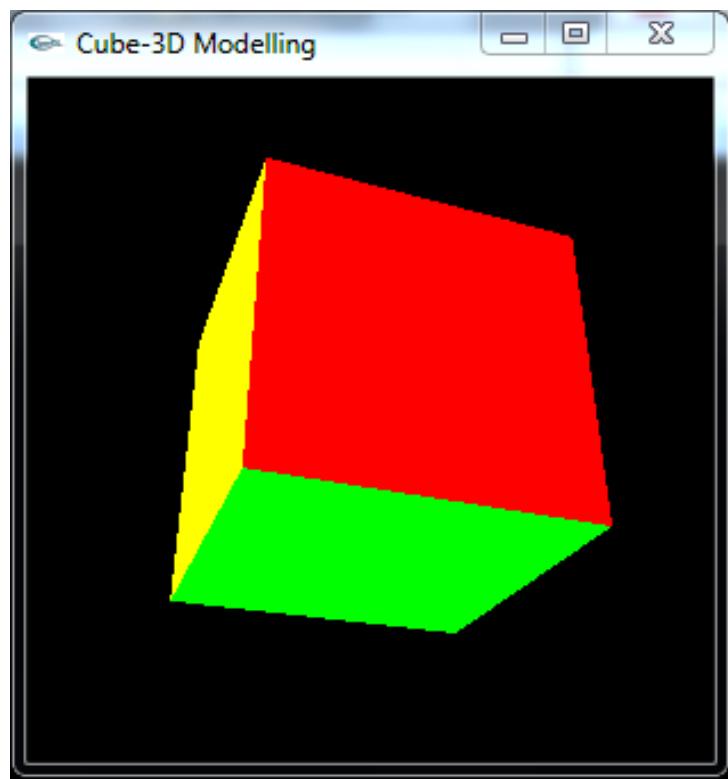
glColor3f(0.0f, 1.0f, 0.0f);
glVertex3fv(&v[faces[5][0]][0]);
glVertex3fv(&v[faces[5][1]][0]);
glVertex3fv(&v[faces[5][2]][0]);
glVertex3fv(&v[faces[5][3]][0]);
glEnd();

glBegin(GL_QUADS);
    glNormal3fv(&n[6][0]);
    glColor3f(0.0f, 0.0f, 1.0f);
    glVertex3fv(&v[faces[6][0]][0]);
    glVertex3fv(&v[faces[6][1]][0]);
    glVertex3fv(&v[faces[6][2]][0]);
    glVertex3fv(&v[faces[6][3]][0]);
glEnd();
glutSwapBuffers();
}

void init(void)
{
/* Setup cube vertex data . */
v[0][0] = v[1][0] = v[2][0] = v[3][0] = -1;
v[4][0] = v[5][0] = v[6][0] = v[7][0] = 1;
v[0][1] = v[1][1] = v[4][1] = v[5][1] = -1;
v[2][1] = v[3][1] = v[6][1] = v[7][1] = 1;
v[0][2] = v[3][2] = v[4][2] = v[7][2] = 1;
v[1][2] = v[2][2] = v[5][2] = v[6][2] = -1;
/* Use depth buffering for hidden surface elimination . */
 glEnable(GL_DEPTH_TEST);
/* Setup the view of the cube . */
 glMatrixMode(GL_PROJECTION);
 gluPerspective( /* field of view in degree */ 40.0,
 /* aspect ratio */ 1.0,
 /* Z near */ 1.0, /* Z far */ 10.0);
 glMatrixMode(GL_MODELVIEW);
 gluLookAt(0.0, 0.0, 5.0, /* eye is at (0,0,5) */
 0.0, 0.0, 0.0, /* center is at (0,0,0) */
 0.0, 1.0, 0.); /* up is in positive Y direction */
/* Adjust cube position to be asthetic angle . */
 glTranslatef(0.0, 0.0, -1.0);
 glRotatef(60, 1.0, 0.0, 0.0);
 glRotatef(-20, 0.0, 0.0, 1.0);
}
int main(int argc, char **argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
glutCreateWindow("Cube-3D Modelling");
glutDisplayFunc(drawScene);
init();
glutMainLoop();
return 0; /* ANSI C requires main to return int . */
}

```

Αποτέλεσμα απεικόνισης



Σχήμα 4.9: 3D απεικόνιση κύβου με χρωματισμένες τις πλευρές του.

4.1.9 Cylinders (Δημιουργία κυλίνδρων)

Κώδικας σε OpenGl

```
#include <math.h>           // For math routines (such as sqrt & trig) .
#include <stdio.h>
#include <glut.h> // OpenGL Graphics Utility Library
#include "cylinders.h"

// The next global variable controls the animation's state and speed.
float RotateAngle = 0.0f; // Angle in degrees of rotation around y-axis
float Azimuth = 20.0; // Rotated up or down by this amount
float AngleStepSize = 3.0f; // Step three degrees at a time
const float AngleStepMax = 10.0f;
const float AngleStepMin = 0.1f;
int WireFrameOn = 1; // == 1 for wire frame mode
// glutKeyboardFunc is called below to set this function to handle
// all "normal" key presses .
void myKeyboardFunc( unsigned char key, int x, int y )
{
    switch ( key ) {
        case 'w':
            WireFrameOn = 1-WireFrameOn;
            if ( WireFrameOn ) {
                glPolygonMode ( GL_FRONT_AND_BACK, GL_LINE); // Just show
wireframes
            }
            else {
                glPolygonMode ( GL_FRONT_AND_BACK, GL_FILL ); // Show solid
polygons
            }
            glutPostRedisplay();
            break;
        case 'R':
            AngleStepSize *= 1.5;
            if (AngleStepSize>AngleStepMax ) {
                AngleStepSize = AngleStepMax;
            }
            break;
        case 'r':
            AngleStepSize /= 1.5;
            if (AngleStepSize<AngleStepMin ) {
                AngleStepSize = AngleStepMin;
            }
            break;
        case 27: // Escape key
            exit(1);
    }
}
// glutSpecialFunc is called below to set this function to handle
// all "special" key presses . See glut.h for the names of
// special keys .
void mySpecialKeyFunc( int key, int x, int y )
{
    switch ( key ) {
        case GLUT_KEY_UP:
            Azimuth += AngleStepSize;
```

```

        if ( Azimuth>80.0f ) {
            Azimuth = 80.0f;
        }
        break;
    case GLUT_KEY_DOWN:
        Azimuth -= AngleStepSize;
        if ( Azimuth < -80.0f ) {
            Azimuth = -80.0f;
        }
        break;
    case GLUT_KEY_LEFT:
        RotateAngle += AngleStepSize;
        if ( RotateAngle > 180.0f ) {
            RotateAngle -= 360.0f;
        }
        break;
    case GLUT_KEY_RIGHT:
        RotateAngle -= AngleStepSize;
        if ( RotateAngle < -180.0f ) {
            RotateAngle += 360.0f;
        }
        break;
    }
    glutPostRedisplay();
}
/*
 * drawScene() handles the animation and the redrawing of the
 * graphics window contents .
*/
void drawScene(void)
{
    // Clear the rendering window
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    // Rotate the image
    glMatrixMode( GL_MODELVIEW );           //Current matrix affects objects positions
    glLoadIdentity(); // Initialize to the identity
    glTranslatef( -0.5, 0.0, -35.0 ); // Translate from origin (in front of viewer)
    glRotatef( RotateAngle, 0.0, 1.0, 0.0 ); // Rotate around y-axis
    glRotatef( Azimuth, 1.0, 0.0, 0.0 ); // Set Azimuth angle
    glDisable( GL_CULL_FACE );
    glPushMatrix();
    glTranslatef( 1.5, 0.0, 0.0 );
    glRotatef( -90.0, 1.0, 0.0, 0.0 );
    glColor3f( 1.0, 0.2, 0.2 ); // Reddish color
    // Parameters : height , radius , slices , stacks
    drawGluCylinder(2.0, 0.75, 8, 10 );
    glPopMatrix();
    glEnable( GL_CULL_FACE );
    glPushMatrix();
    glTranslatef( -1.5, 0.0, 0.0 );
    glRotatef( -90.0, 1.0, 0.0, 0.0 );
    glColor3f( 0.2, 1.0, 0.2 ); // Greenish color
    // Parameters : height , base radius , top radius , slices , stacks .
    drawGluSlantCylinderWithCaps( 2.0, 1.0, 0.4, 8, 8 );
    glPopMatrix();
    // Flush the pipeline , swap the buffers
    glFlush();
    glutSwapBuffers();
}
// Initialize OpenGL's rendering modes

```

```

void initRendering()
{
    glEnable( GL_DEPTH_TEST ); // Depth testing must be turned on
        glCullFace( GL_BACK );
        glPolygonMode ( GL_FRONT_AND_BACK, GL_LINE ); // Just show wireframes at first
    }
// Called when the window is resized
// w, h - width and height of the window in pixels .
void resizeWindow(int w, int h)
{
    double aspectRatio;
    // Define the portion of the window used for OpenGL rendering .
    glViewport( 0, 0, w, h ); // View port uses whole window
    // Set up the projection view matrix : perspective projection
    // Determine the min and max values for x and y that should appear in the window .
    // The complication is that the aspect ratio of the window may not match the
    // aspect ratio of the scene we want to view .
    w = (w==0) ? 1 : w;
    h = (h==0) ? 1 : h;
    aspectRatio = (double)w / (double)h;
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    gluPerspective( 15.0, aspectRatio, 25.0, 45.0 );
}
// Main routine
// Set up OpenGL , define the callbacks and start the main loop
int main( int argc, char** argv )
{
    glutInit(&argc, argv);
    // We're going to animate it , so double buffer
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH );
        // Window position (from top corner) , and size (width% and height)
    glutInitWindowPosition( 10, 60 );
    glutInitWindowSize( 360, 360 );
    glutCreateWindow( "GluCylinders" );
        // Initialize OpenGL as we like it..
    initRendering();
        // Set up callback functions for key presses
    glutKeyboardFunc( myKeyboardFunc ); // Handles "normal" ascii symbols
    glutSpecialFunc( mySpecialKeyFunc ); // Handles "special" keyboard keys
        // Set up the callback function for resizing windows
    glutReshapeFunc( resizeWindow );
        // Call this for background processing
        // glutIdleFunc( myIdleFunction ) ;
        // call this whenever window needs redrawing
    glutDisplayFunc( drawScene );
        fprintf(stdout, "Arrow keys control viewpoint.\n");
        fprintf(stdout, "Press \'w\' to toggle wireframe mode.\n");
        fprintf(stdout, "Press \'R\' or \'r\' to increase or decrease rate of movement (respectively).\n");
        // Start the main loop . glutMainLoop never returns .
    glutMainLoop();
    return(0); // This line is never reached .
}
// ****
//These are four general purpose routines for generating
//cylinders , with or without caps .
//See the OpenGL redbook or other glu documentation for more information .
//These generate normal , but not texture coordinates .
//To generate texture coordinates , you need to modify the code to
//call gluQuadricTexture() ;

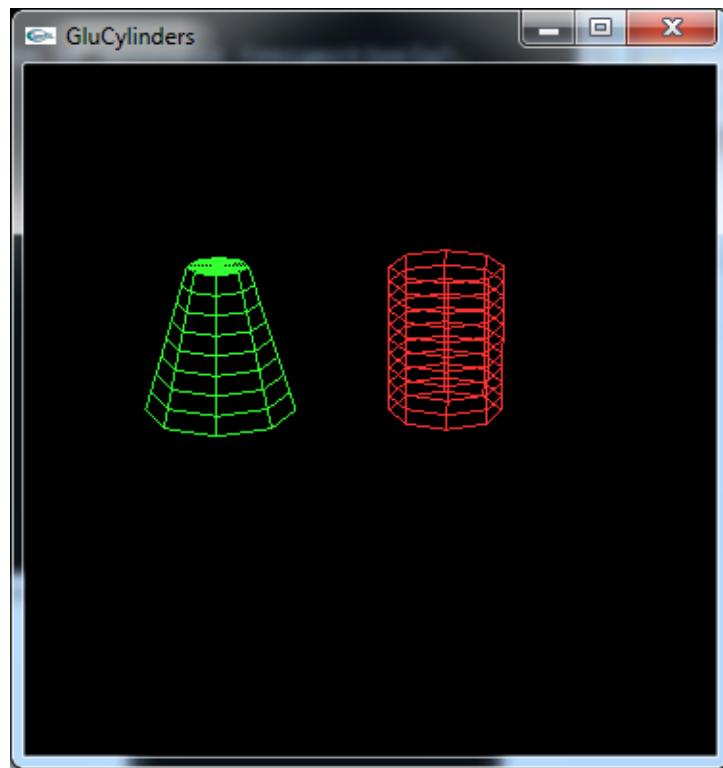
```

```

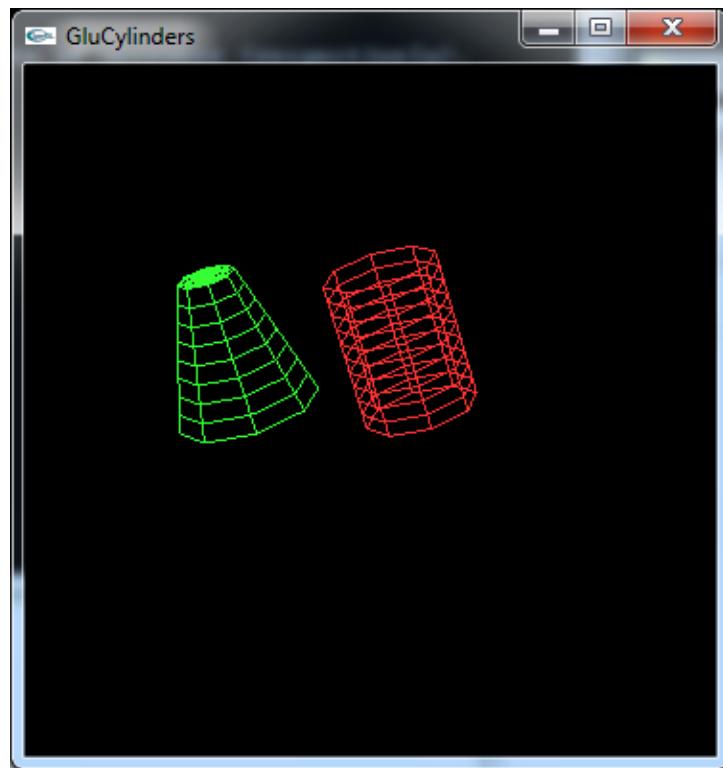
//For higher performance , you should consider putting your cylinders into
//a display list .
//Please note these routines do not do all possible error checking , and
//thus should not be used in a production or other critical environment .
// *****
// A Reusable gluQuadric object :
GLUquadricObj* myReusableQuadric = 0;
void drawGluCylinder( double height, double radius, int slices, int stacks ) {
    drawGluSlantCylinder( height, radius, radius, slices, stacks );
}
void drawGluSlantCylinder( double height, double radiusBase, double radiusTop, int slices, int stacks )
{
    if ( ! myReusableQuadric ) {
        myReusableQuadric = gluNewQuadric();
        // Should (but don't) check if pointer is still null --- to catch memory allocation errors .
        gluQuadricNormals( myReusableQuadric, GL_TRUE );
    }
    // Draw the cylinder .
    gluCylinder( myReusableQuadric, radiusBase, radiusTop, height, slices, stacks );
}
void drawGluCylinderWithCaps( double height, double radius, int slices, int stacks ) {
    drawGluSlantCylinderWithCaps( height, radius, radius, slices, stacks );
}
void drawGluSlantCylinderWithCaps( double height, double radiusBase, double radiusTop, int slices, int
stacks )
{
    // First draw the cylinder
    drawGluSlantCylinder( height, radiusBase, radiusTop, slices, stacks );
    // Draw the top disk cap
    glPushMatrix();
    glTranslated(0.0, 0.0, height);
    gluDisk( myReusableQuadric, 0.0, radiusTop, slices, stacks );
    glPopMatrix();
    // Draw the bottom disk cap
    glPushMatrix();
    glRotated(180.0, 1.0, 0.0, 0.0);
    gluDisk( myReusableQuadric, 0.0, radiusBase, slices, stacks );
    glPopMatrix();
}

```

Αποτέλεσμα απεικόνισης



Σχήμα 4.10: 3D Απεικόνιση κυλίνδρων



Σχήμα 4.11: 3D Απεικόνιση κυλίνδρων με κλίση. (Εμφανίζεται με το πάτημα του πλήκτρου w)

4.1.10 Cone (Δημιουργία κινούμενου κώνου)

```
#include <glut.h>

GLfloat xRotated, yRotated, zRotated;
// Cone
GLdouble base=1;
GLdouble height=1.5;
GLint slices =50;
GLint stacks =50;

void displayCone(void)
{
    glMatrixMode(GL_MODELVIEW);
    // clear the drawing buffer .
    glClear(GL_COLOR_BUFFER_BIT);
    // clear the identity matrix .
    glLoadIdentity();
    // traslate the draw by z = -4.0
    // Note this when you decrease z like -8.0 the drawing will looks far , or smaller .
    glTranslatef(0.0,0.0,-4.5);
    // Red color used to draw .
    glColor3f(0.8, 0.2, 0.1);
    // changing in transformation matrix .
    // rotation about X axis
    glRotatef(xRotated,1.0,0.0,0.0);
    // rotation about Y axis
    glRotatef(yRotated,0.0,1.0,0.0);
    // rotation about Z axis
    glRotatef(zRotated,0.0,0.0,1.0);
    // scaling transfomation
    glScalef(1.0,1.0,1.0);
    // built-in (glut library) function , draw you a Cone .
    glutSolidCone(base,height,slices,stacks);
    // Flush buffers to screen
    glFlush();
    // swap buffers called because we are using double buffering
    // glutSwapBuffers() ;
}

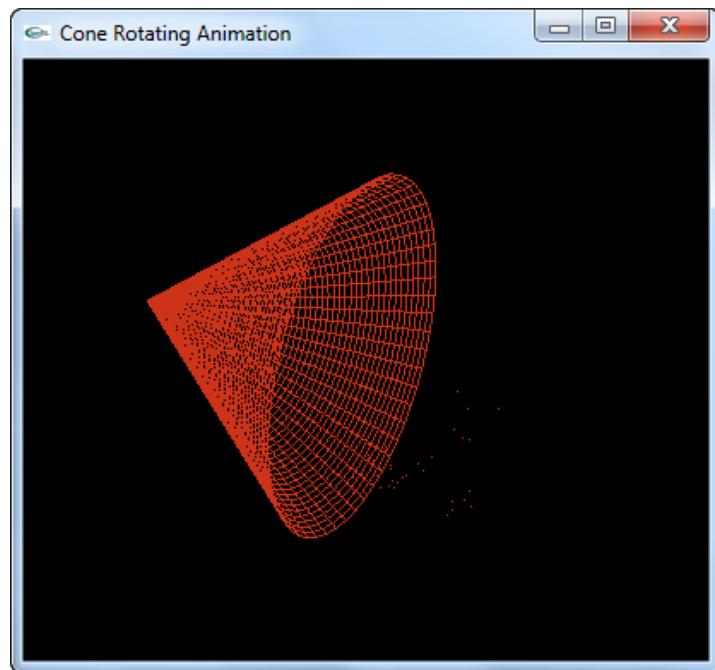
void reshapeCone(int x, int y)
{
    if (y == 0 || x == 0) return; //Nothing is visible then , so return
    //Set a new projection matrix
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    //Angle of view:40 degrees
    //Near clipping plane distance : 0.5
    //Far clipping plane distance : 20.0
    gluPerspective(40.0,(GLdouble)x/(GLdouble)y,0.5,20.0);
    glViewport(0,0,x,y); //Use the whole window for rendering
}

void idleCone(void)
{
    yRotated += 0.01;
    displayCone();
}

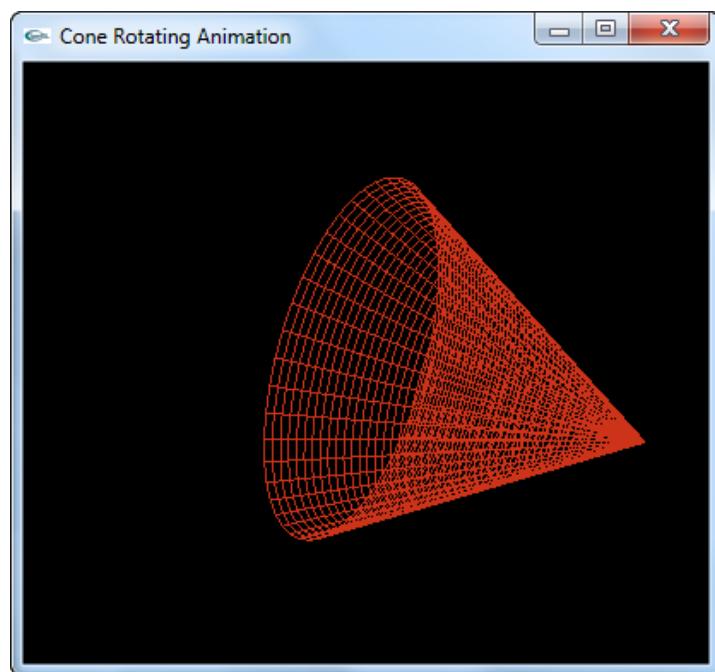
int main (int argc, char **argv)
{
```

```
//Initialize GLUT
glutInit(&argc, argv);
//double buffering used to avoid flickering problem in animation
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
// window size
glutInitWindowSize(400,350);
// create the window
glutCreateWindow("Cone Rotating Animation");
glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
xRotated = yRotated = zRotated = 30.0;
xRotated=33;
yRotated=40;
glClearColor(0.0,0.0,0.0,0.0);
//Assign the function used in events
glutDisplayFunc(displayCone);
glutReshapeFunc(reshapeCone);
glutIdleFunc(idleCone);
//Let start glut loop
glutMainLoop();
return 0;
}
```

Αποτέλεσμα απεικόνισης



Σχήμα 4.12: 3D Cone (Δημιουργία κινούμενου κώνου)



Σχήμα 4.13: 3D Cone (Δημιουργία κινούμενου κώνου)

4.1.11 Octahedron (Δημιουργία Οκτάπλευρου)

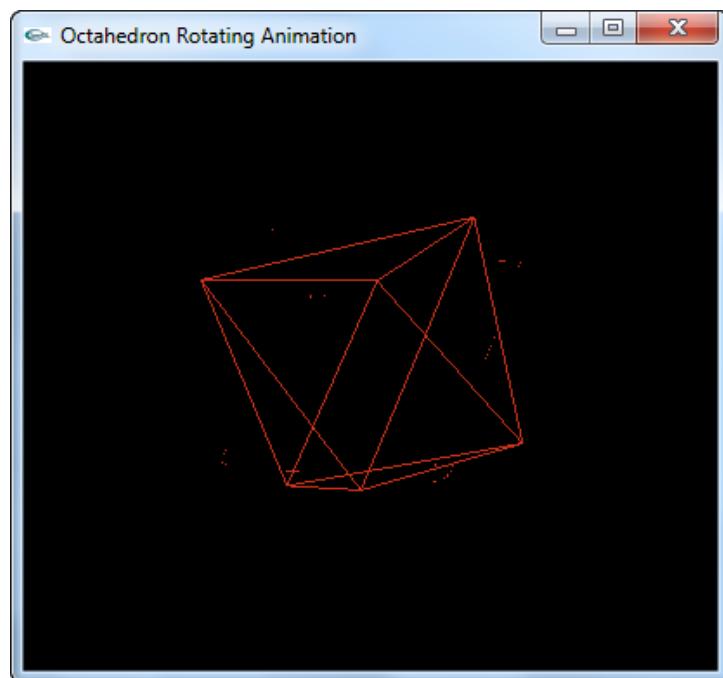
```
#include <glut.h>
GLfloat xRotated, yRotated, zRotated;
// Octahedron
void displayOctahedron(void)
{
    glMatrixMode(GL_MODELVIEW);
    // clear the drawing buffer .
    glClear(GL_COLOR_BUFFER_BIT);
    // clear the identity matrix .
    glLoadIdentity();
    // traslate the draw by z = -4.0
    // Note this when you decrease z like -8.0 the drawing will looks far , or smaller .
    glTranslatef(0.0,0.0,-4.5);
    // Red color used to draw .
    glColor3f(0.8, 0.2, 0.1);
    // changing in transformation matrix.
    // rotation about X axis
    glRotatef(xRotated,1.0,0.0,0.0);
    // rotation about Y axis
    glRotatef(yRotated,0.0,1.0,0.0);
    // rotation about Z axis
    glRotatef(zRotated,0.0,0.0,1.0);
    // scaling transfomation
    glScalef(1.0,1.0,1.0);
    // built-in (glut library) function , draw you a Octahedron .
    glutSolidOctahedron();
    // Flush buffers to screen
    glFlush();
    // swap buffers called because we are using double buffering
    // glutSwapBuffers();
}
void reshapeOctahedron(int x, int y)
{
    if (y == 0 || x == 0) return; //Nothing is visible then , so return
    //Set a new projection matrix
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    //Angle of view : 40 degrees
    //Near clipping plane distance : 0.5
    //Far clipping plane distance : 20.0
    gluPerspective(40.0,(GLdouble)x/(GLdouble)y,0.5,20.0);
    glViewport(0,0,x,y); //Use the whole window for rendering
}
void idleOctahedron(void)
{
    yRotated += 0.03;
    displayOctahedron();
}
int main (int argc, char **argv)
{
    //Initialize GLUT
    glutInit(&argc, argv);
    //double buffering used to avoid flickering problem in animation
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    // window size
    glutInitWindowSize(400,350);
```

```

// create the window
glutCreateWindow("Octahedron Rotating Animation");
glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
xRotated = yRotated = zRotated = 30.0;
xRotated=33;
yRotated=40;
glClearColor(0.0,0.0,0.0,0.0);
// Assign the function used in events
glutDisplayFunc(displayOctahedron);
glutReshapeFunc(reshapeOctahedron);
glutIdleFunc(idleOctahedron);
//Let start glut loop
glutMainLoop();
return 0;
}

```

Αποτέλεσμα απεικόνισης



Σχήμα 4.14: Octahedron (Δημιουργία Οκτάπλευρου)

4.1.12 Rotating Cube (Δημιουργία Περιστρεφόμενου Κύβου)

```
#include <glut.h>

GLfloat xRotated, yRotated, zRotated;
void init(void)
{
    glClearColor(0,0,0,0);
}

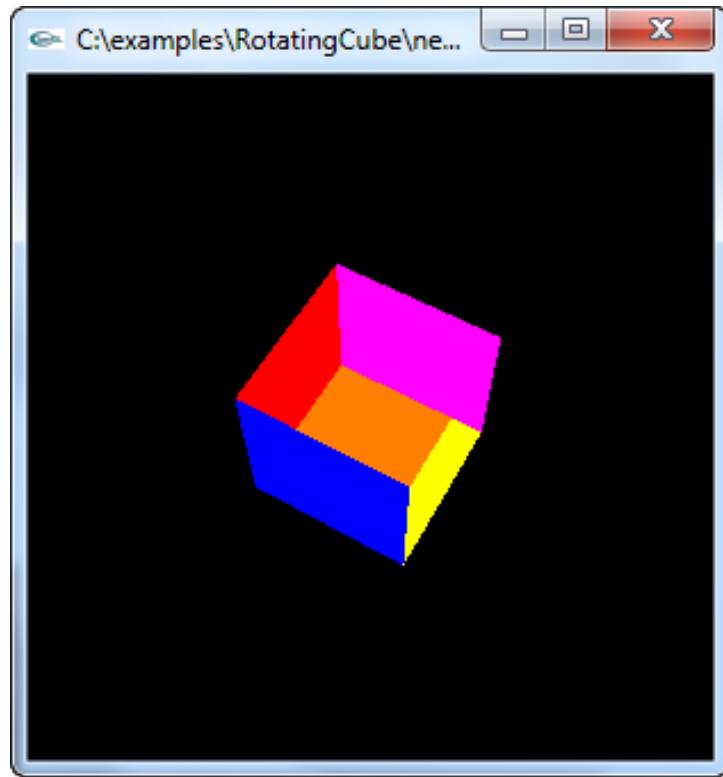
void DrawCube(void)
{
    glMatrixMode(GL_MODELVIEW);
    // clear the drawing buffer .
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glTranslatef(0.0,0.0,-10.5);
    glRotatef(xRotated,1.0,0.0,0.0);
    // rotation about Y axis
    glRotatef(yRotated,0.0,1.0,0.0);
    // rotation about Z axis
    glRotatef(zRotated,0.0,0.0,1.0);
    glBegin(GL_QUADS); // Draw The Cube Using quads
    glColor3f(0.0f,1.0f,0.0f); // Color Blue
    glVertex3f( 1.0f, 1.0f,-1.0f); // Top Right Of The Quad (Top)
    glVertex3f(-1.0f, 1.0f,-1.0f); // Top Left Of The Quad (Top)
    glVertex3f(-1.0f, 1.0f, 1.0f); // Bottom Left Of The Quad (Top)
    glVertex3f( 1.0f, 1.0f, 1.0f); // Bottom Right Of The Quad (Top)
    glColor3f(1.0f,0.5f,0.0f); // Color Orange
    glVertex3f( 1.0f,-1.0f, 1.0f); // Top Right Of The Quad (Bottom)
    glVertex3f(-1.0f,-1.0f, 1.0f); // Top Left Of The Quad (Bottom)
    glVertex3f(-1.0f,-1.0f,-1.0f); // Bottom Left Of The Quad (Bottom)
    glVertex3f( 1.0f,-1.0f,-1.0f); // Bottom Right Of The Quad (Bottom)
    glColor3f(1.0f,0.0f,0.0f); // Color Red
    glVertex3f( 1.0f, 1.0f, 1.0f); // Top Right Of The Quad (Front)
    glVertex3f(-1.0f, 1.0f, 1.0f); // Top Left Of The Quad (Front)
    glVertex3f(-1.0f,-1.0f, 1.0f); // Bottom Left Of The Quad (Front)
    glVertex3f( 1.0f,-1.0f, 1.0f); // Bottom Right Of The Quad (Front)
    glColor3f(1.0f,1.0f,0.0f); // Color Yellow
    glVertex3f( 1.0f,-1.0f,-1.0f); // Top Right Of The Quad (Back)
    glVertex3f(-1.0f,-1.0f,-1.0f); // Top Left Of The Quad (Back)
    glVertex3f(-1.0f, 1.0f,-1.0f); // Bottom Left Of The Quad (Back)
    glVertex3f( 1.0f, 1.0f,-1.0f); // Bottom Right Of The Quad (Back)
    glColor3f(0.0f,0.0f,1.0f); // Color Blue
    glVertex3f(-1.0f, 1.0f, 1.0f); // Top Right Of The Quad (Left)
    glVertex3f(-1.0f, 1.0f,-1.0f); // Top Left Of The Quad (Left)
    glVertex3f(-1.0f,-1.0f,-1.0f); // Bottom Left Of The Quad (Left)
    glVertex3f(-1.0f,-1.0f, 1.0f); // Bottom Right Of The Quad (Left)
    glColor3f(1.0f,0.0f,1.0f); // Color Violet
    glVertex3f( 1.0f, 1.0f,-1.0f); // Top Right Of The Quad (Right)
    glVertex3f( 1.0f, 1.0f, 1.0f); // Top Left Of The Quad (Right)
    glVertex3f( 1.0f,-1.0f,-1.0f); // Bottom Left Of The Quad (Right)
    glVertex3f( 1.0f,-1.0f, 1.0f); // Bottom Right Of The Quad (Right)
    glEnd(); // End Drawing The Cube
    glFlush();
}
void animation(void)
{
```

```

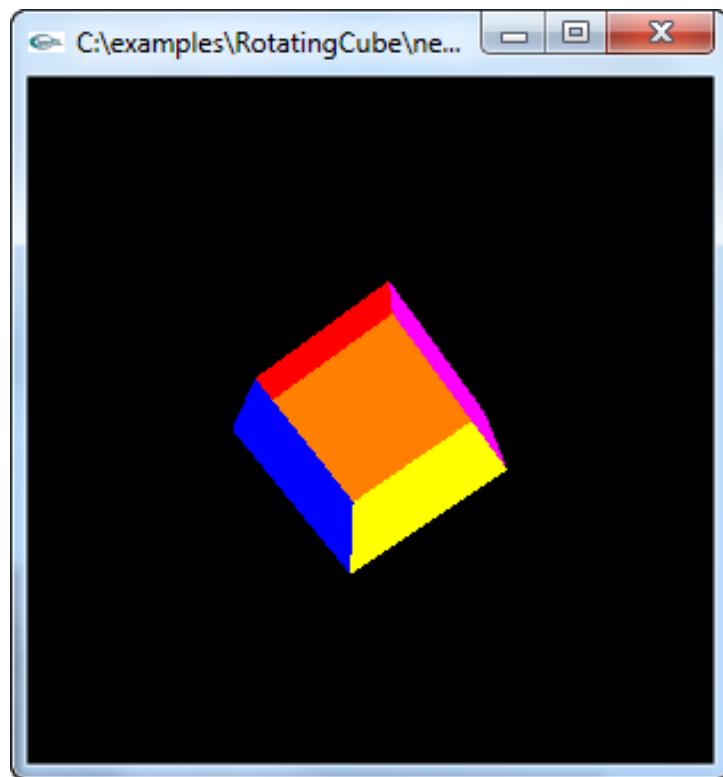
yRotated += 0.01;
xRotated += 0.02;
DrawCube();
}
void reshape(int x, int y)
{
if (y == 0 || x == 0) return; //Nothing is visible then , so return
// Set a new projection matrix
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
// Angle of view:40 degrees
// Near clipping plane distance : 0.5
// Far clipping plane distance : 20.0
gluPerspective(40.0,(GLdouble)x/(GLdouble)y,0.5,20.0);
glMatrixMode(GL_MODELVIEW);
glViewport(0,0,x,y); // Use the whole window for rendering
}
int main(int argc, char** argv){
glutInit(&argc, argv);
//we initizilize the glut. functions
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowPosition(100, 100);
glutCreateWindow(argv[0]);
init();
glutDisplayFunc(DrawCube);
glutReshapeFunc(reshape);
//Set the function for the animation .
glutIdleFunc(animation);
glutMainLoop();
return 0;
}

```

Αποτέλεσμα απεικόνισης



Σχήμα 4.15: 3D Rotating Cube (Δημιουργία Περιστρεφόμενου Κύβου)



Σχήμα 4.16: 3D Rotating Cube (Δημιουργία Περιστρεφόμενου Κύβου)

4.1.13 SolidShpereLight (Δημιουργία Συμπαγούς Φωτιζόμενης Σφαίρας)

//In this example we rotate 3D sphere with light in a circle shape like solar system

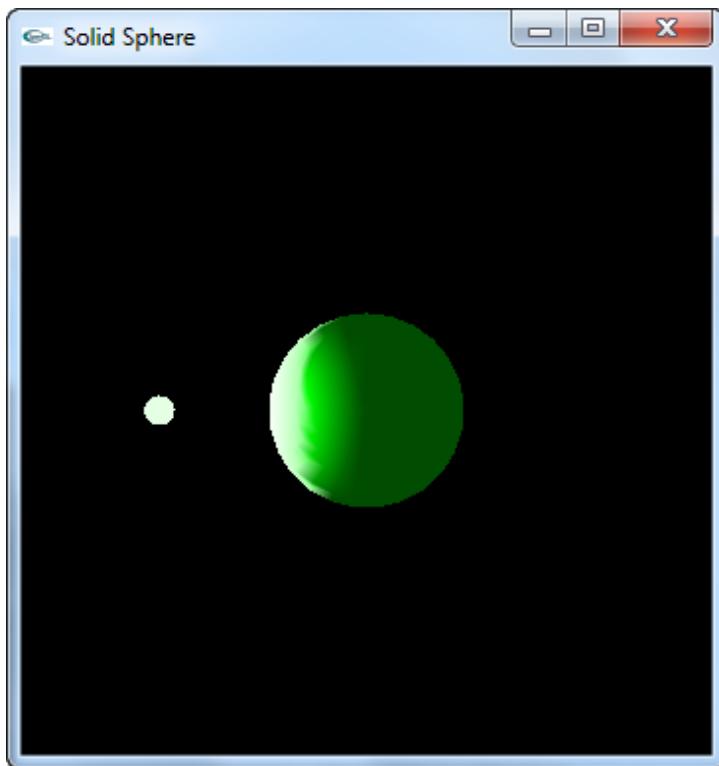
```
#include <glut.h>
#include <math.h> // For math routines (such as sqrt & trig).
GLfloat xRotated, yRotated, zRotated;
GLdouble radius=2;
GLfloat qaBlack[] = {0.0, 0.0, 0.0, 1.0}; //Black Color
GLfloat qaGreen[] = {0.0, 1.0, 0.0, 1.0}; //Green Color
GLfloat qaWhite[] = {1.0, 1.0, 1.0, 1.0}; //White Color
GLfloat qaRed[] = {1.0, 0.0, 0.0, 1.0}; //White Color
    // Set lighting intensity and color
GLfloat qaAmbientLight[] = {0.1, 0.1, 0.1, 1.0};
GLfloat qaDiffuseLight[] = {1, 1, 1, 1.0};
GLfloat qaSpecularLight[] = {1.0, 1.0, 1.0, 1.0};
GLfloat emitLight[] = {0.9, 0.9, 0.9, 0.01};
GLfloat Noemit[] = {0.0, 0.0, 0.0, 1.0};
    // Light source position
GLfloat qaLightPosition[] = {0, 0, 0, 1};
void display(void);
void reshape(int x, int y);
void idleFunc(void)
{
    if ( zRotated > 360.0 ) {
        zRotated -= 360.0*floor(zRotated/360.0); // Don't allow overflow
    }
    if ( yRotated > 360.0 ) {
        yRotated -= 360.0*floor(yRotated/360.0); // Don't allow overflow
    }
    zRotated += 0.01;
    yRotated +=0.01;
    display();
}
void initLighting()
{
    // Enable lighting
 glEnable(GL_LIGHTING);
 glEnable(GL_LIGHT0);
    // Set lighting intensity and color
    glLightfv(GL_LIGHT0, GL_AMBIENT, qaAmbientLight);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, qaDiffuseLight);
    glLightfv(GL_LIGHT0, GL_SPECULAR, qaSpecularLight);
}
int main (int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH );
    glutInitWindowSize(350,350);
    glutCreateWindow("Solid Sphere");
    initLighting();
    xRotated = yRotated = zRotated = 0.0;
    glutIdleFunc(idleFunc);
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutMainLoop();
    return 0;
}
```

```

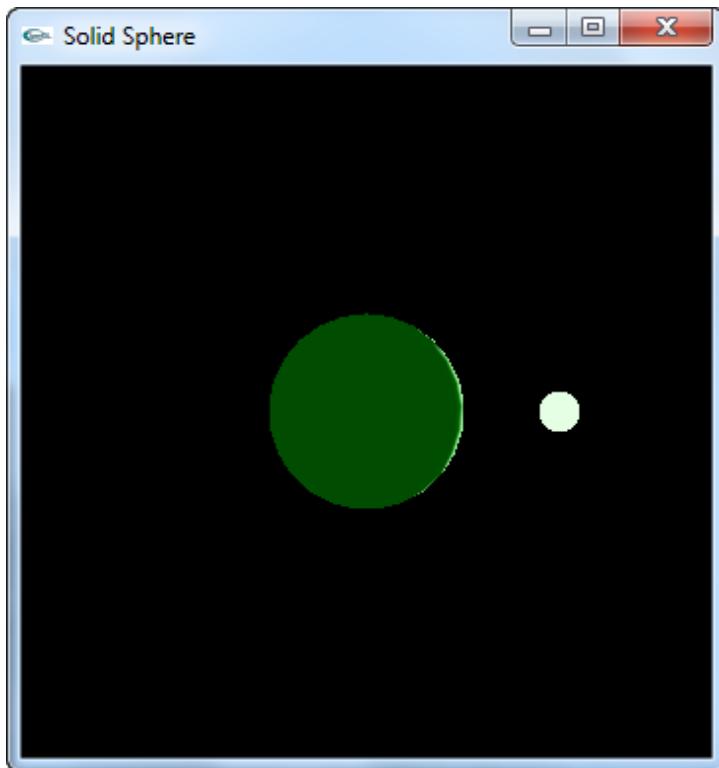
void display(void)
{
    glMatrixMode(GL_MODELVIEW);
    // clear the drawing buffer.
    glClear(GL_COLOR_BUFFER_BIT);
    // clear the identity matrix .
    glLoadIdentity();
    glTranslatef(0.0,0.0,-20.0);
    glPushMatrix();
    glTranslatef(0.0,0.0,0);
    // Set material properties
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, qaGreen);
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, qaGreen);
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, qaWhite);
    glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 0.2);
    glutSolidSphere(radius,25,25);
    glPopMatrix();
    glPushMatrix();
    glRotatef(yRotated,0.0,1.0,0.0);
    glTranslatef(5.0,0.0,0.0);
    // Set the light position
    glLightfv(GL_LIGHT0, GL_POSITION, qaLightPosition);
    glMaterialfv(GL_FRONT_AND_BACK, GL_EMISSION, emitLight); // Make sphere glow
(emissive)
    glutSolidSphere(radius/6,25,25);
    glMaterialfv(GL_FRONT_AND_BACK, GL_EMISSION, Noemit);
    glPopMatrix();
    glFlush();
    glutSwapBuffers();
}
void reshape(int x, int y)
{
    if (y == 0 || x == 0) return;
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(39.0,(GLdouble)x/(GLdouble)y,0.6,40.0);
    glMatrixMode(GL_MODELVIEW);
    glViewport(0,0,x,y); //Use the whole window for rendering
}

```

Αποτέλεσμα απεικόνισης



Σχήμα 4.17: SolidSphereLight (Δημιουργία Συμπαγούς Φωτιζόμενης Σφαίρας)



Σχήμα 4.18: SolidSphereLight (Δημιουργία Συμπαγούς Φωτιζόμενης Σφαίρας)

4.1.14 AddSpotLight (Προσθήκη Φωτισμού Σε Κύβο)

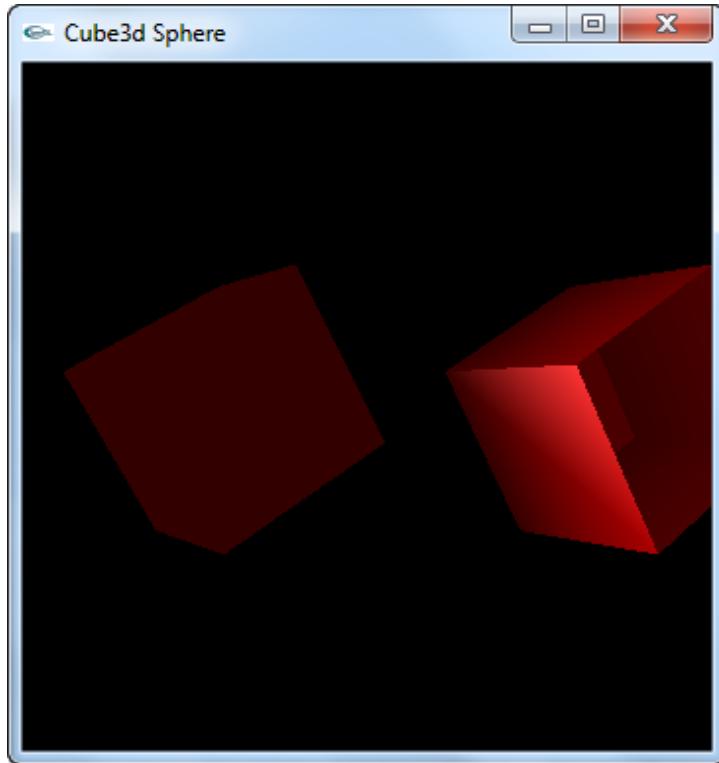
```
#include <glut.h>
#include <math.h> // For math routines (such as sqrt & trig) .
GLfloat xRotated, yRotated, zRotated;
GLdouble radius=4;
GLfloat qaBlack[] = {0.0, 0.0, 0.0, 1.0}; //Black Color
GLfloat qaGreen[] = {1.0, 0.0, 0.0, 1.0}; //Green Color
GLfloat qaWhite[] = {1.0, 1.0, 1.0, 1.0}; //White Color
GLfloat qaRed[] = {1.0, 0.0, 0.0, 1.0}; //Red Color
    // Set lighting intensity and color
GLfloat qaAmbientLight[] = {0.1, 0.1, 0.1, 1.0};
GLfloat qaDiffuseLight[] = {1, 1, 1, 1.0};
GLfloat qaSpecularLight[] = {1.0, 1.0, 1.0, 1.0};
GLfloat emitLight[] = {0.9, 0.9, 0.9, 0.01};
GLfloat Noemit[] = {0.0, 0.0, 0.0, 1.0};
    // Light source position
GLfloat qaLightPosition[] = {0, 0, 2, 1};
GLfloat qaLightDirection[] = {1, 1, 1, 0};
GLfloat dirVector0[] = {1.0, 0.0, 0.0, 0.0};
void display(void);
void reshape(int x, int y);
void idleFunc(void)
{
    display();
}
void initLighting()
{
    // Enable lighting
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    // Set lighting intensity and color
    glLightfv(GL_LIGHT0, GL_AMBIENT, qaAmbientLight);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, qaDiffuseLight);
    glLightfv(GL_LIGHT0, GL_POSITION, qaLightPosition);
    glLightfv(GL_LIGHT0, GL_SPECULAR, qaSpecularLight);
    /////////////////////////////////
    glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 30.0); // set cutoff angle
    glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, dirVector0);
    glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, 1); // set focusing strength
}
void keyboard(unsigned char key, int x, int y)
{
    if (key == 'T' || key == 'L')
    {
        glEnable(GL_LIGHTING);
    }
    else if (key == 'd' || key == 'D')
    {
        glDisable(GL_LIGHTING);
    }
}
int main (int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH );
    glutInitWindowSize(350,350);
    glutCreateWindow("Cube3d Sphere");
    initLighting();
    xRotated = yRotated = zRotated = 0.0;
```

```

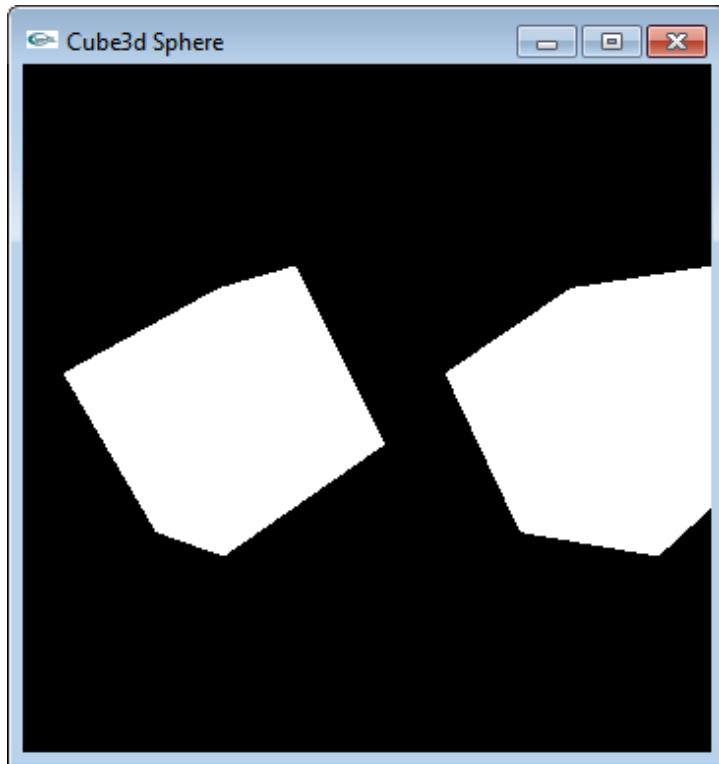
glutIdleFunc(idleFunc);
glutDisplayFunc(display);
glutKeyboardFunc(keyboard); // Register keyboard callback
glutReshapeFunc(reshape);
glutMainLoop();
return 0;
}
void display(void)
{
    glMatrixMode(GL_MODELVIEW);
    // clear the drawing buffer .
    glClear(GL_COLOR_BUFFER_BIT);
    // clear the identity matrix .
    glLoadIdentity();
    glTranslatef(0.0,0.0,-20.0);
    glPushMatrix();
    glTranslatef(-3.0,0.0,0);
    // Set material properties
    glRotatef(30,0.0,1.0,0.0);
    glRotatef(30,0.0,0.0,1.0);
        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, qaRed);
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, qaRed);
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, qaWhite);
    glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 20);
    glutSolidCube(radius);
    glPopMatrix();
    glPushMatrix();
    glTranslatef(5.0,0.0,0);
    // Set material properties
    glRotatef(30,0.0,1.0,0.0);
    glRotatef(30,0.0,0.0,1.0);
        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, qaRed);
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, qaRed);
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, qaWhite);
    glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 20);
    glutSolidCube(radius);
    glPopMatrix();
    glPushMatrix();
    glRotatef(yRotated,0.0,1.0,0.0);
    glLightfv(GL_LIGHT0, GL_POSITION, qaLightPosition);
    glPopMatrix();
    glFlush();
    glutSwapBuffers();
}
void reshape(int x, int y)
{
    if (y == 0 || x == 0) return;
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(39.0,(GLdouble)x/(GLdouble)y,0.6,40.0);
    glMatrixMode(GL_MODELVIEW);
    glViewport(0,0,x,y); // Use the whole window for rendering
}

```

Αποτέλεσμα απεικόνισης



Σχήμα 4.19: AddSpotLight (Προσθήκη Φωτισμού Σε Κύβο) (Εμφανίζεται με το πάτημα του πλήκτρου L)

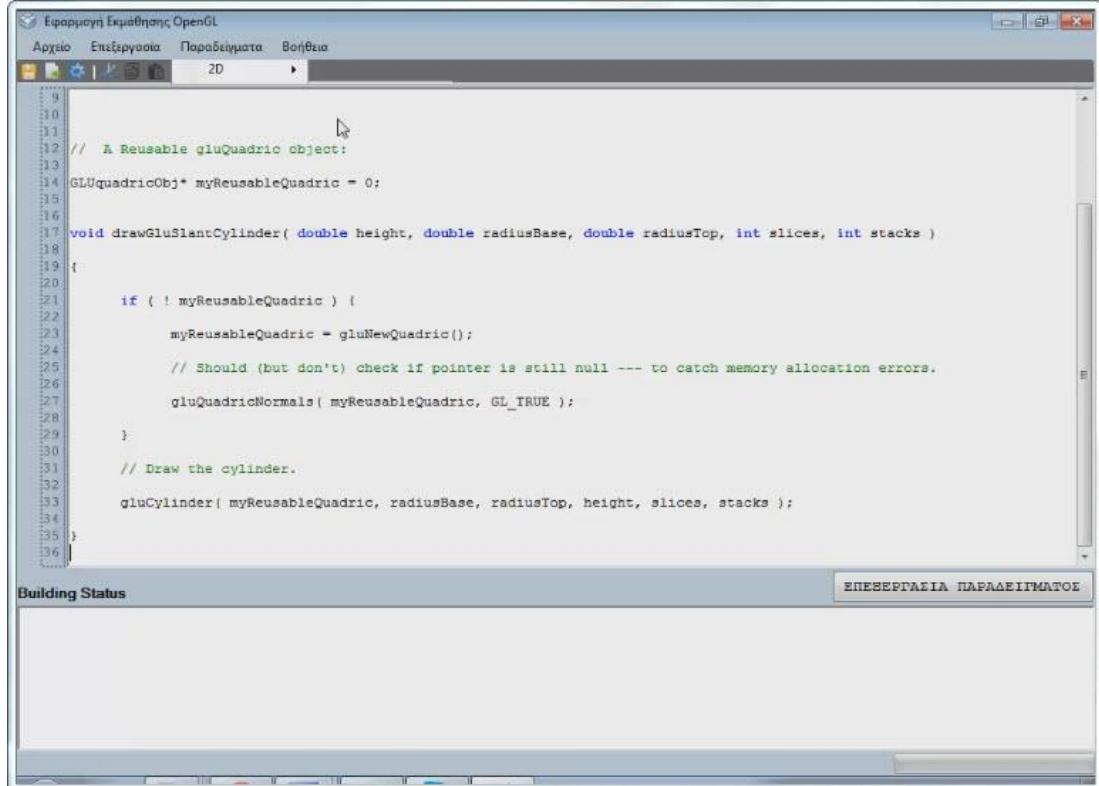


Σχήμα 4.20 AddSpotLight (Προσθήκη Φωτισμού Σε Κύβο) (Εμφανίζεται με το πάτημα του πλήκτρου D)

4.2 Ενδεικτικά Παραδείγματα

4.2.1 Λειτουργία κώδικα χωρίς λάθη

Εμφάνιση κώδικα παραδείγματος κύλινδρου:



The screenshot shows a Windows application window titled "Εφαρμογή Εκμάθησης OpenGL". The menu bar includes "Αρχικό", "Επιζήργαση", "Παραδείγματα", and "Βοήθεια". A toolbar below the menu has icons for file operations and a "2D" button. The main area contains C code for drawing a cylinder:

```
9
10
11 // A Reusable gluQuadric object:
12
13 GLUquadricObj* myReusableQuadric = 0;
14
15
16 void drawGluSlantCylinder( double height, double radiusBase, double radiusTop, int slices, int stacks )
17 {
18
19     if ( ! myReusableQuadric ) {
20
21         myReusableQuadric = gluNewQuadric();
22
23         // Should (but don't) check if pointer is still null --- to catch memory allocation errors.
24
25         gluQuadricNormals( myReusableQuadric, GL_TRUE );
26
27     }
28
29     // Draw the cylinder.
30
31     gluCylinder( myReusableQuadric, radiusBase, radiusTop, height, slices, stacks );
32
33 }
34
35 }
```

Below the code editor, there is a "Building Status" panel which is currently empty.

Σχήμα 4.21 Εμφάνιση κώδικα κυλίνδρου

Εκτέλεση κώδικα και εμφάνιση 0 προειδοποιήσεων και 0 λαθών:

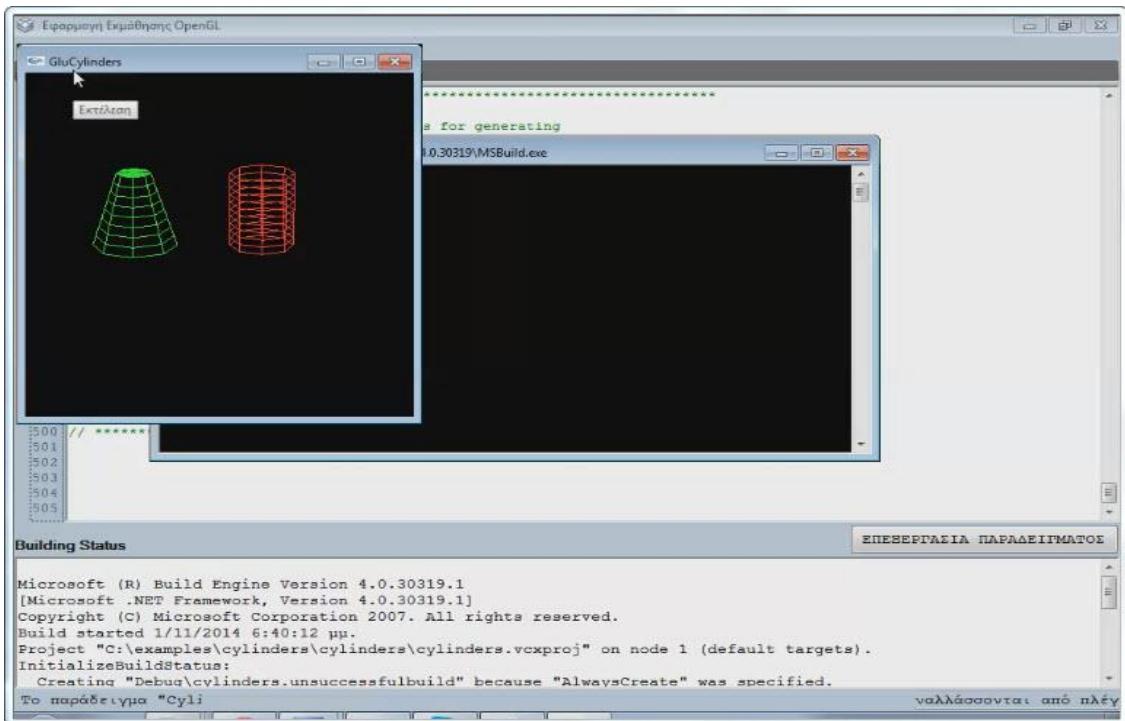
The screenshot shows the OpenGL application window. The main area displays the source code for generating cylinders. The code includes comments explaining the purpose of various routines and how to generate normals and texture coordinates. The build status window at the bottom shows the following output:

```
Building Status
FinalizeBuildStatus:
  Deleting file "Debug\cylinders.unsuccessfulbuild".
  Touching "Debug\cylinders.lastbuildstate".
Done Building Project "C:\examples\cylinders\cylinders.vcxproj" (default targets)...
Build succeeded.
  0 Warning(s)
  0 Error(s)
Time Elapsed 00:00:00.24
```

ΣΗΜΕΙΩΣΗ: Η λέξη "ΕΠΕΞΕΓΓΑΛΙΑ ΠΑΡΑΔΕΙΓΜΑΤΟΣ" στην επάνω δεξιά γωνία του παραθύρου δεν είναι στα ελληνικά.

Σχήμα 4.22 Εκτέλεση κώδικα

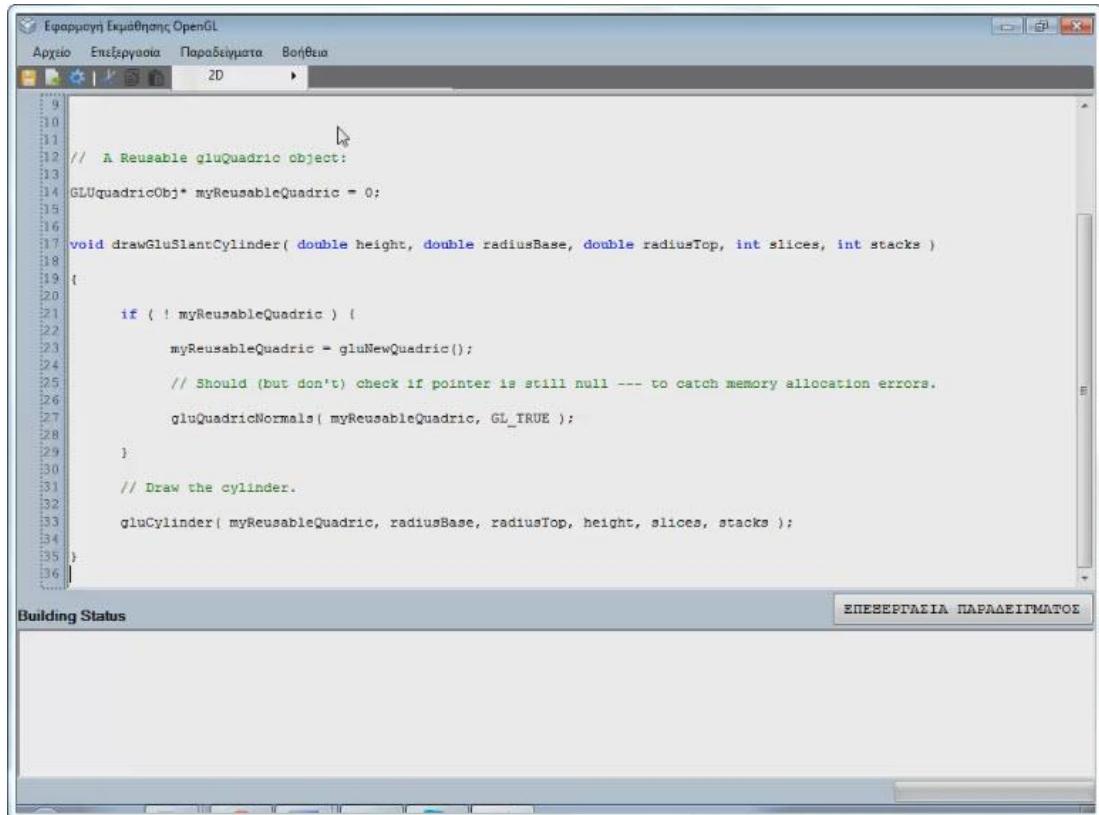
Εμφάνιση απεικόνισης γραφήματος μετά την εκτέλεση του κώδικα:



Σχήμα 4.23 Εμφάνιση γραφήματο

4.2.2 Λειτουργία κώδικα με λάθη

Εμφάνιση κώδικα παραδείγματος κυλίνδρου:



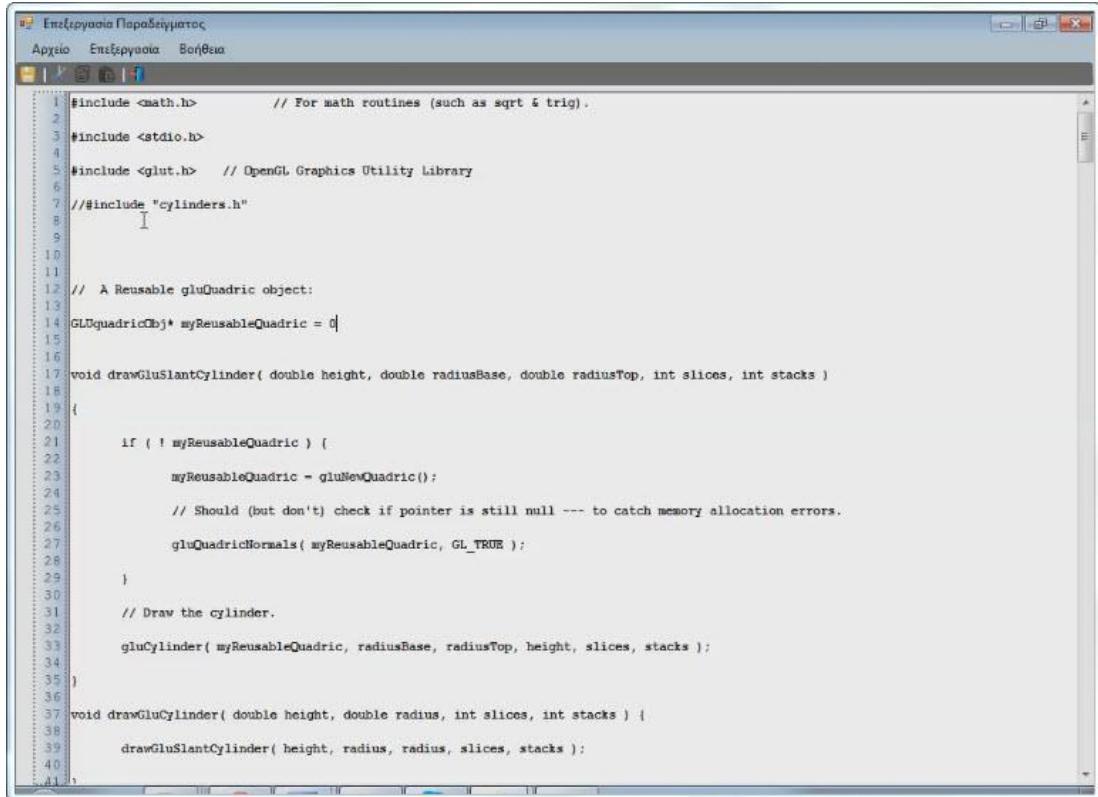
The screenshot shows a window titled "Εφαρμογή Εκμάθησης OpenGL" (OpenGL Learning Application). The menu bar includes "Αρχείο", "Επεξεργασία", "Πορεδιώματα", and "Βοήθεια". A toolbar below the menu has icons for file operations and a "2D" button. The main area contains C code for drawing a cylinder:

```
9
10
11
12 // A Reusable gluQuadric object:
13
14 GLUquadricObj* myReusableQuadric = 0;
15
16
17 void drawGluSlantCylinder( double height, double radiusBase, double radiusTop, int slices, int stacks )
18 {
19
20     if ( ! myReusableQuadric ) {
21
22         myReusableQuadric = gluNewQuadric();
23
24         // Should (but don't) check if pointer is still null --- to catch memory allocation errors.
25
26         gluQuadricNormals( myReusableQuadric, GL_TRUE );
27
28     }
29
30
31     // Draw the cylinder.
32
33     gluCylinder( myReusableQuadric, radiusBase, radiusTop, height, slices, stacks );
34
35 }
```

Below the code editor, there are two status bars: "Building Status" and "ΕΠΕΞΕΡΓΑΣΙΑ ΠΑΡΑΔΕΙΓΜΑΤΟΣ".

Σχήμα 4.24 Εμφάνιση κώδικα κυλίνδρου

Εκτέλεση κώδικα και εμφάνιση 0 προειδοποιήσεων και 1 λαθών. Στη συγκεκριμένη περίπτωση όπως φαίνεται στην παρακάτω εικόνα έχει αφαιρεθεί από τον κώδικα στην γραμμή 14 το ερωτηματικό στο τέλος της γραμμής:



```
Επεξεργασία Παραδίγματος
Αρχείο Επεξεργασία Βοήθεια

1 #include <math.h>           // For math routines (such as sqrt & trig).
2
3 #include <stdio.h>
4
5 #include <glut.h>          // OpenGL Graphics Utility Library
6
7 //##include "cylinders.h"
8
9
10
11
12 // A Reusable gluQuadric object:
13
14 GLUquadricObj* myReusableQuadric = 0;
15
16
17 void drawGluSlantCylinder( double height, double radiusBase, double radiusTop, int slices, int stacks )
18 {
19
20     if ( ! myReusableQuadric ) {
21
22         myReusableQuadric = gluNewQuadric();
23
24         // Should (but don't) check if pointer is still null --- to catch memory allocation errors.
25
26         gluQuadricNormals( myReusableQuadric, GL_TRUE );
27
28     }
29
30     // Draw the cylinder.
31
32     gluCylinder( myReusableQuadric, radiusBase, radiusTop, height, slices, stacks );
33
34 }
35
36
37 void drawGluCylinder( double height, double radius, int slices, int stacks ) {
38
39     drawGluSlantCylinder( height, radius, radius, slices, stacks );
40
41 }
```

Σχήμα 4.25 Εκτέλεση κώδικα

Αποθηκεύεται η επεξεργασία του παραδείγματος:

```
#include <math.h>
// For math routines (such as sqrt & trig).

#include <stdio.h>
// OpenGL Graphics Utility Library

#include "cylinders.h"

// A Reusable gluQuadric object:
GLUquadricObj* myReusableQuadric = 0;

void drawGluSlantCylinder( double height, double radiusBase, double radiusTop, int slices, int stacks )
{
    if ( ! myReusableQuadric ) {
        myReusableQuadric = gluNewQuadric();
        // Should (but don't) check if pointer is still null --- to catch memory allocation errors.
        gluQuadricNormals( myReusableQuadric, GL_TRUE );
    }
    // Draw the cylinder.
    gluCylinder( myReusableQuadric, radiusBase, radiusTop, height, slices, stacks );
}

void drawGluCylinder( double height, double radius, int slices, int stacks ) {
    drawGluSlantCylinder( height, radius, radius, slices, stacks );
}
```

Σχήμα 4.26 Αποθήκευση Επεξεργασίας Παραδείγματος

Εισάγεται η ονομασία του παραδείγματος και επιλέγεται επόμενο:

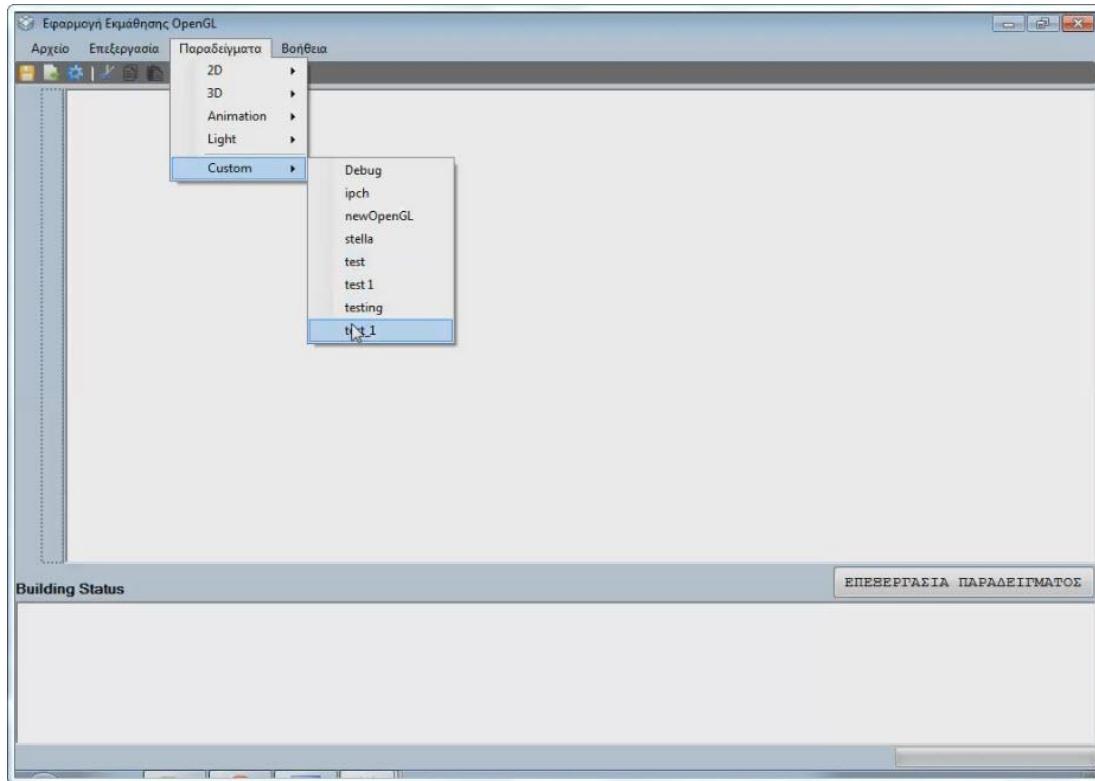
```
#include <math.h>
// For math routines (such as sqrt & trig).

#include <stdio.h>
// OpenGL Graphics Utility Library

#include "cylinders.h"
Όνομα παραδείγματος
test_1
Επόμενο
```

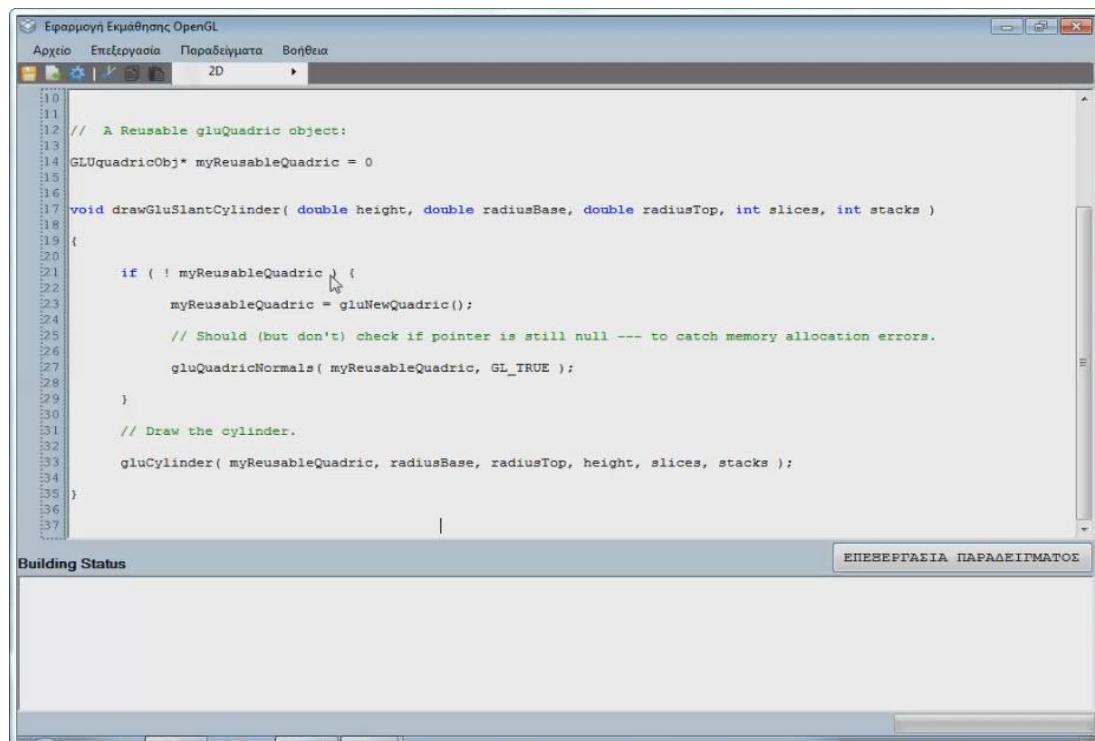
Σχήμα 4.27 Εισαγωγή ονόματος παραδείγματος

Επιλογή φακέλου custom για εύρεση παραδείγματος που επεξεργάστηκε:



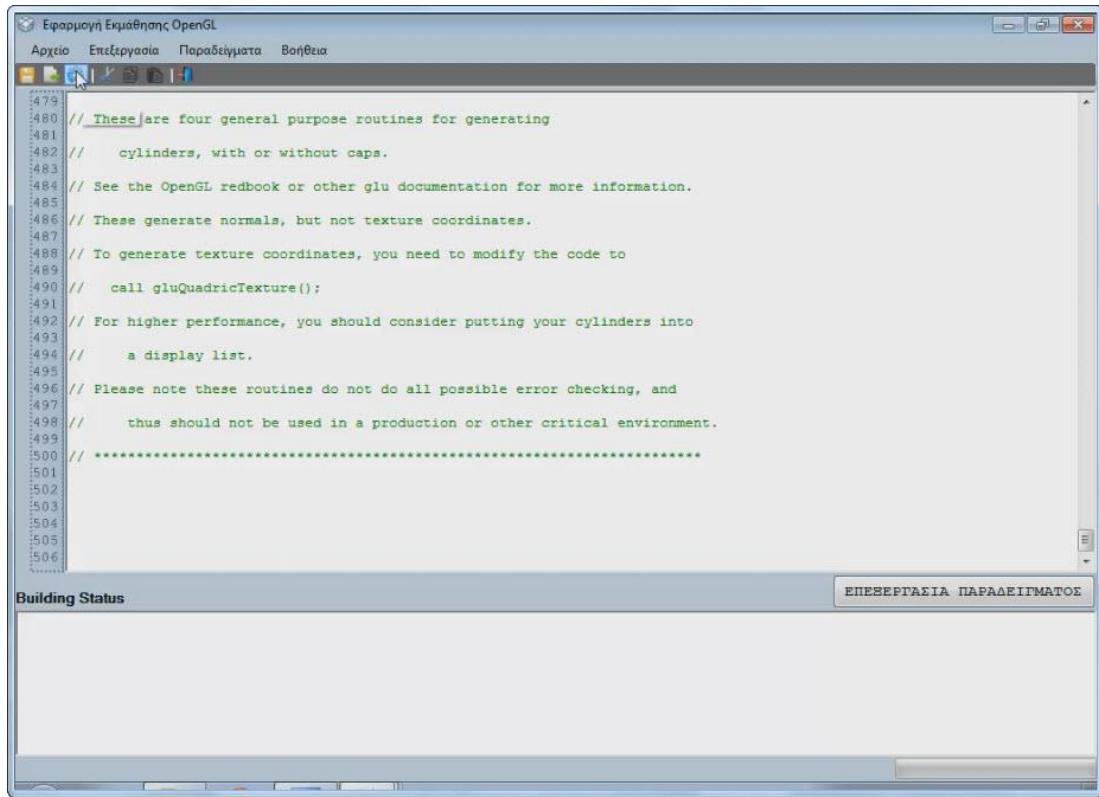
Σχήμα 4.28 Επιλογή φακέλου custom και παραδείγματος

Φόρτωση επεξεργασμένου παραδείγματος:



Σχήμα 4.29 Φόρτωση επεξεργασμένου παραδείγματος

Εκτέλεση παραδείγματος:



The screenshot shows a Windows application window titled "Εφαρμογή Εκμάθησης OpenGL". The menu bar includes "Αρχείο", "Επεξεργασία", "Παραδείγματα", and "Βοήθεια". The main window contains a code editor with the following C code:

```
479 // These are four general purpose routines for generating
480 // cylinders, with or without caps.
481 // See the OpenGL redbook or other glu documentation for more information.
482 // These generate normals, but not texture coordinates.
483 // To generate texture coordinates, you need to modify the code to
484 // call gluQuadricTexture();
485 // For higher performance, you should consider putting your cylinders into
486 // a display list.
487 // Please note these routines do not do all possible error checking, and
488 // thus should not be used in a production or other critical environment.
489 // ****
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
```

The status bar at the bottom left says "Building Status" and the right side says "ΕΠΕΞΕΡΓΑΣΙΑ ΠΑΡΑΔΕΙΓΜΑΤΟΣ".

Σχήμα 4.30 Φόρτωση παραδείγματος και εκτέλεση

Εκτέλεση κώδικα και εμφάνιση 1 λάθους. Όπως φαίνεται στο Building Status, έχει εμφανιστεί ένα λάθος που προειδοποιεί ότι έχει ξεχαστεί ένα ερωτηματικό στη σειρά 14

```
// These are four general purpose routines for generating
// cylinders, with or without caps.
// See the OpenGL redbook or other glu documentation for more information.
// These generate normals, but not texture coordinates.
// To generate texture coordinates, you need to modify the code to
// call gluQuadricTexture();
// For higher performance, you should consider putting your cylinders into
// a display list.
// Please note these routines do not do all possible error checking, and
// thus should not be used in a production or other critical environment.
// *****
```

Building Status

```
"C:\examples\Custom\test_1\newOpenGL\newOpenGL.vcxproj" (default target) (1) ->
(ClCompile target) -> [redacted]
  c:\examples\custom\test_1\newopengl\newopengl.cpp(17): error C2144: syntax error : 'void' should be preceded
by ';' [C:\examples\Custom\test_1\newOpenGL\newOpenGL.vcxproj]
  0 Warning(s)
  1 Error(s)
```

Time Elapsed 00:00:02.34

Σχήμα 4.31 Εκτέλεση κώδικα και εμφάνιση λάθους

Παρακάτω φαίνεται ότι προστίθεται το ερωτηματικό:

```
#include <math.h>           // For math routines (such as sqrt & trig).
#include <stdio.h>
#include <glut.h>            // OpenGL Graphics Utility Library
//#include "cylinders.h"

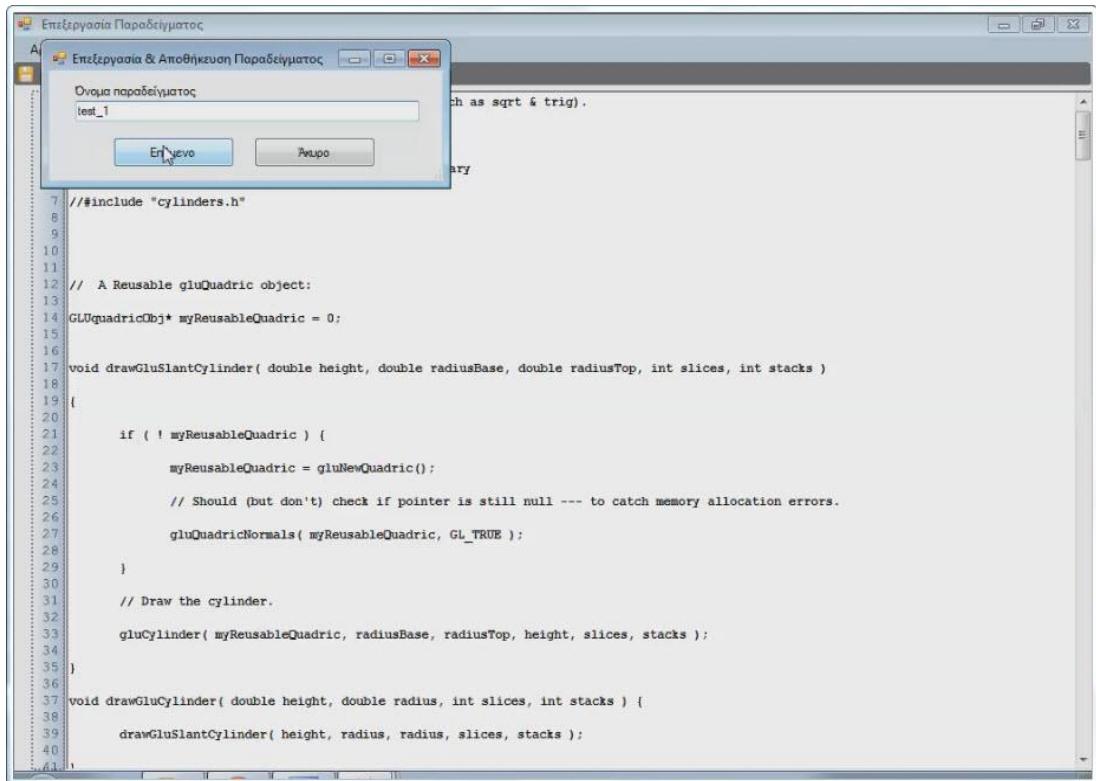
// A Reusable gluQuadric object:
GLUquadricObj* myReusableQuadric = 0; [redacted]

void drawGluSlantCylinder( double height, double radiusBase, double radiusTop, int slices, int stacks )
{
    if ( ! myReusableQuadric ) {
        myReusableQuadric = gluNewQuadric();
        // Should (but don't) check if pointer is still null --- to catch memory allocation errors.
        gluQuadricNormals( myReusableQuadric, GL_TRUE );
    }
    // Draw the cylinder.
    gluCylinder( myReusableQuadric, radiusBase, radiusTop, height, slices, stacks );
}

void drawGluCylinder( double height, double radius, int slices, int stacks ) {
    drawGluSlantCylinder( height, radius, radius, slices, stacks );
}
```

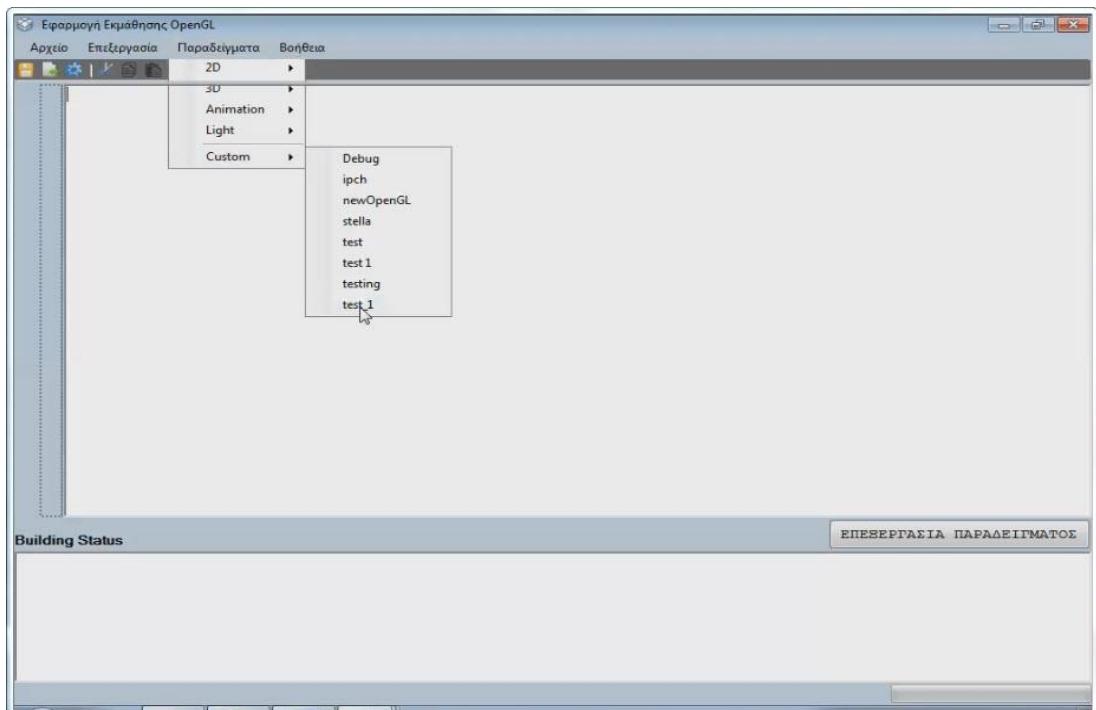
Σχήμα 4.32 Διόρθωση λάθους

Γίνεται αποθήκευση των αλλαγών:



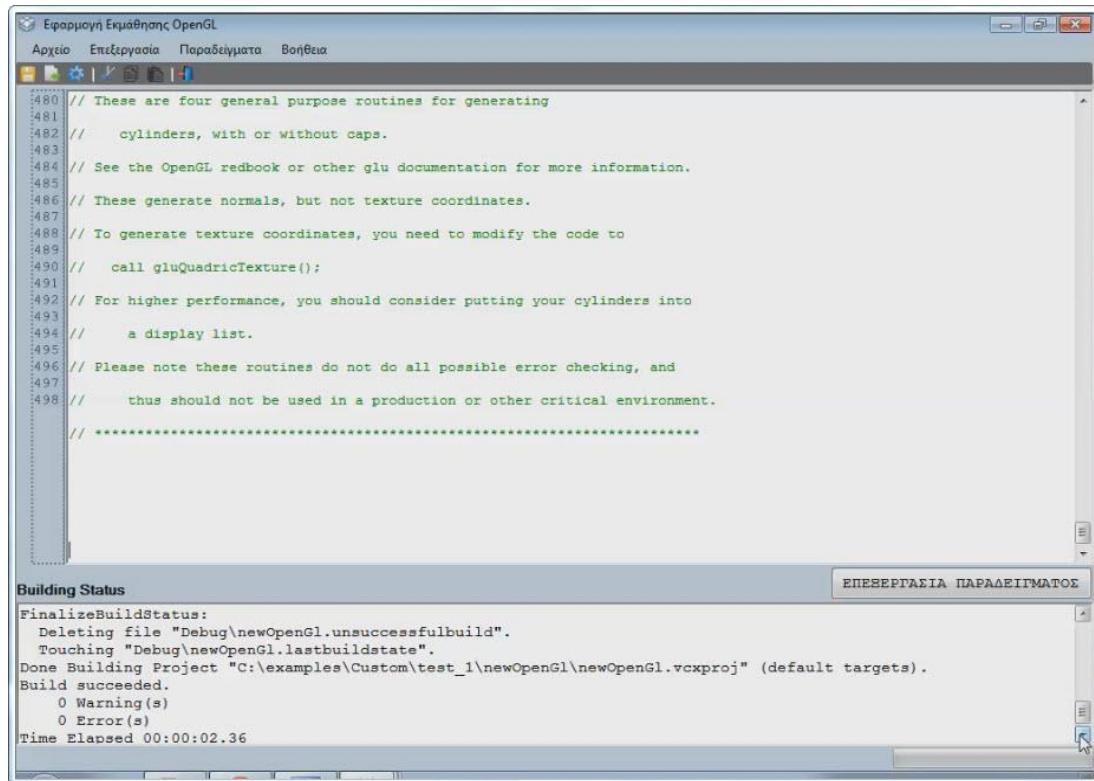
Σχήμα 4.33 Αποθήκευση αλλαγών

Επιλογή φακέλου custom και παραδείγματος που επεξεργάστηκε:



Σχήμα 4.34 Επιλογή παραδείγματος που επεξεργάστηκε

Εκτελείται ξανά ο κώδικας και εμφανίζει μηδενικά λάθη:



The screenshot shows the OpenGL application window with the title "Εφαρμογή Εκμάθησης OpenGL". The menu bar includes "Αρχείο", "Επεξεργασία", "Παραδείγματα", and "Βοήθεια". The main area displays a portion of the C++ code:

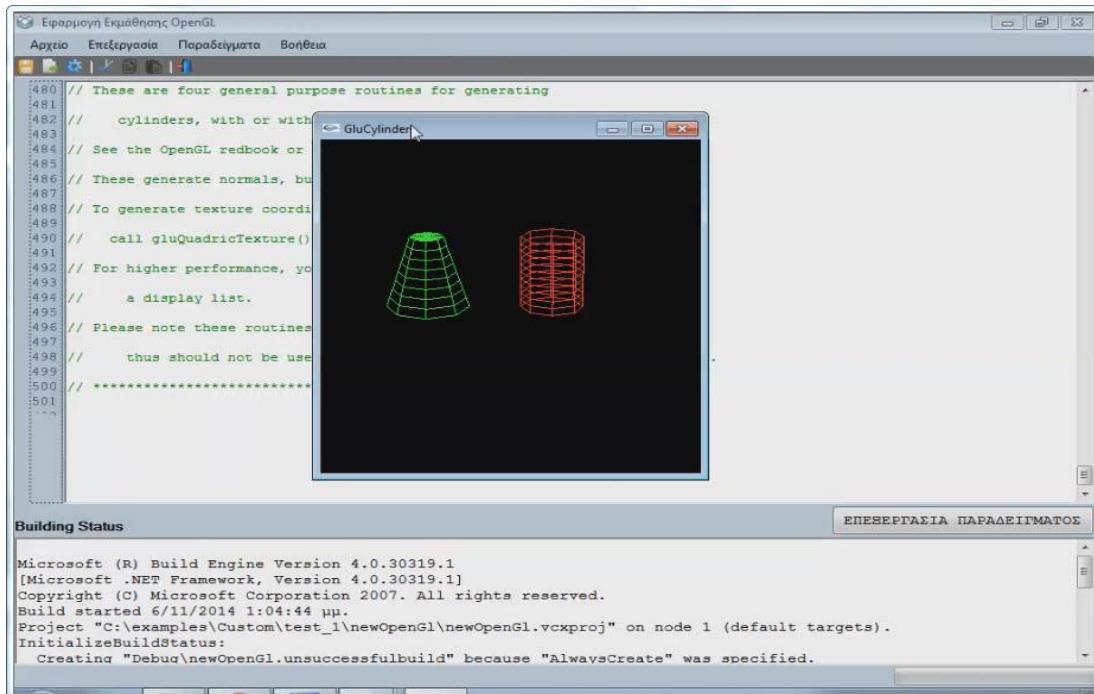
```
// These are four general purpose routines for generating
// cylinders, with or without caps.
// See the OpenGL redbook or other glu documentation for more information.
// These generate normals, but not texture coordinates.
// To generate texture coordinates, you need to modify the code to
// call gluQuadricTexture();
// For higher performance, you should consider putting your cylinders into
// a display list.
// Please note these routines do not do all possible error checking, and
// thus should not be used in a production or other critical environment.
// *****
```

The "Building Status" panel at the bottom shows the build process:

```
FinalizeBuildStatus:
Deleting file "Debug\newOpenGL.unsuccessfulbuild".
Touching "Debug\newOpenGL.lastbuildstate".
Done Building Project "C:\examples\Custom\test_1\newOpenGL\newOpenGL.vcxproj" (default targets).
Build succeeded.
  0 Warning(s)
  0 Error(s)
Time Elapsed 00:00:02.36
```

Σχήμα 4.35 Μη εμφάνιση λαθών

Εμφάνιση γραφήματος:



The screenshot shows the OpenGL application window with the title "Εφαρμογή Εκμάθησης OpenGL". The menu bar includes "Αρχείο", "Επεξεργασία", "Παραδείγματα", and "Βοήθεια". The main area displays a portion of the C++ code:

```
// These are four general purpose routines for generating
// cylinders, with or with
// See the OpenGL redbook or
// These generate normals, bu
// To generate texture coordi
// call gluQuadricTexture()
// For higher performance, yo
// a display list.
// Please note these routines
// thus should not be use
// *****
```

A small window titled "GluCylinder" is displayed in the center, showing two wireframe cylinders: one green and one red.

The "Building Status" panel at the bottom shows the build process:

```
Microsoft (R) Build Engine Version 4.0.30319.1
[Microsoft .NET Framework, Version 4.0.30319.1]
Copyright (C) Microsoft Corporation 2007. All rights reserved.
Build started 6/11/2014 1:04:44 μμ.
Project "C:\examples\Custom\test_1\newOpenGL\newOpenGL.vcxproj" on node 1 (default targets).
InitializeBuildStatus:
Creating "Debug\newOpenGL.unsuccessfulbuild" because "AlwaysCreate" was specified.
```

Σχήμα 4.36 Εμφάνιση γραφήματος

ΚΕΦΑΛΑΙΟ 5

ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΕΣ ΠΡΟΤΑΣΕΙΣ ΓΙΑ ΒΕΛΤΙΩΣΗ

5.1 Συμπεράσματα

Η εφαρμογή που αναπτύχθηκε κατά τη διάρκεια της πτυχιακής είναι μια εκπαιδευτική εφαρμογή για την εκμάθηση της OpenGL και τη δημιουργία γραφικών απεικονίσεων. Η υλοποίηση της έχει ως σκοπό να βοηθήσει τους φοιτητές στο μάθημα γραφικών υπολογιστών ως εικονικό εργαστήριο γιατί δεν υπάρχει η δυνατότητα παρακολούθησης κανονικού εργαστηρίου.

Η εφαρμογή που υλοποιήθηκε κατά τη διάρκεια της πτυχιακής εργασίας θεωρείται ότι ανταποκρίνεται ικανοποιητικά στις απαιτήσεις που ετέθησαν από τον επιβλέποντα καθηγητή. Αντιμετωπίστηκαν σε σημαντικό βαθμό οι δυσκολίες που εμφανίστηκαν κατά τη διάρκεια ανάπτυξης και δόθηκαν οι απαραίτητες λύσεις. Η εφαρμογή είναι εύχρηστη και σε απλοποιημένη μορφή για την καλύτερη χρήση της από το φοιτητή.

Η εφαρμογή που αναπτύχθηκε κατά τη διάρκεια αυτής της πτυχιακής εργασίας συνοδεύεται από ένα πλήρες set up, ώστε να μπορεί να εγκατασταθεί σε οποιονδήποτε υπολογιστή με λειτουργικό σύστημα Windows (εξαιρείται η έκδοση Windows Home). Ο χρήστης θα αναπτύσσει τον κώδικα του αποκλειστικά με την OpenGL και θα βασίζεται στις βιβλιοθήκες της γλώσσας προγραμματισμού C++.

Η εφαρμογή δίνει τη δυνατότητα στο χρήστη, να αναπτύσσει δικό του κώδικα, να ενσωματώνει τα δικά του παραδείγματα στην εφαρμογή και γενικά να πειραματίζεται με την OpenGL. Συγκεκριμένα στην εφαρμογή υπάρχουν έτοιμα ομαδοποιημένα παραδείγματα όπου ο χρήστης μπορεί να δει (κώδικας και γράφημα), αλλά δεν μπορεί να τα τροποποιήσει. Ωστόσο έχει τη δυνατότητα να πάρει κομμάτια κώδικα και να τα ενσωματώσει στο δικό του, με τη βοήθεια της λειτουργίας «Επεξεργασία Παραδείγματος». Στο νέο παράθυρο που δημιουργείται μπορεί να κάνει οποιαδήποτε τροποποίηση και προσθήκη κώδικα θέλει και να αποθηκεύσει το νέο αρχείο με τον κώδικα που δημιουργείται και με το όνομα που επιθυμεί.

Επιπλέον, ο χρήστης μπορεί να γράψει δικό του κώδικα επιλέγοντας αρχικά τη λειτουργία «Εισαγωγή Παραδείγματος» από τη γραμμή εργαλείων. Στη συνέχεια ανοίγει ένα παράθυρο, όπου ο χρήστης πληκτρολογεί το όνομα που επιθυμεί και έτσι δημιουργείται μια νέα φόρμα για τη συγγραφή κώδικα. Η φόρμα αυτή περιέχει ήδη τις απαραίτητες βιβλιοθήκες και τη συνάρτηση main. Έτσι ο χρήστης το μόνο που κάνει είναι να γράψει τον κώδικα του σε OpenGL. Αφού αναπτύξει τον κώδικα του, μπορεί να τον εκτελέσει με τη βοήθεια των σχετικών λειτουργιών. Στο κάτω μέρος της εφαρμογής υπάρχει το παράθυρο «building status», μέσα από το οποίο ο χρήστης παρατηρεί όλη τη διαδικασία εκτέλεσης του κώδικα του. Μπορεί να παρατηρήσει τυχόν σφάλματα που έχει κάνει και σε ποια γραμμή βρίσκονται και να τα διορθώσει. Εάν δεν υπάρχουν σφάλματα μπορεί να δει το γράφημα σε ένα νέο παράθυρο απεικόνισης.

5.2 Μελλοντικές προτάσεις για βελτίωση

- Στον κώδικα που γράφει ο φοιτητής σε OpenGL να γίνεται συντακτικός έλεγχος, δηλαδή εάν γράψει κάποια εντολή και υπάρχει κάποιο λάθος να κοκκινίζει η λέξη, ώστε να κατανοηθεί από το φοιτητή ότι έχει γράψει λάθος κάποια εντολή. Το ίδιο μπορεί να συμβεί και με άλλα σύμβολα, παραδείγματος χάριν, αν ο φοιτητής

δημιουργήσει μια συνάρτηση, την «ανοίξει» με άγκιστρο και στο τέλος της συνάρτησης ξεχάσει να την «κλείσει», το άγκιστρο που ανοίγει η συνάρτηση θα κοκκινίσει ώστε ο χρήστης να αντιληφθεί ότι το έχει ξεχάσει. Επίσης μια ακόμη λειτουργία που μπορεί να προστεθεί είναι όταν μαρκάρεται π.χ. μια παρένθεση να επισημάνεται με κάποιο τρόπο η αντίστοιχη παρένθεση που κλείνει ή ανοίγει.

2. Δυνατότητα ενσωμάτωσης browser, ώστε ο χρήστης να μπορεί να πλοηγηθεί στο διαδίκτυο απευθείας από την εφαρμογή για ανάγνωση κάποιων εγχειριδίων χρήσης της OpenGL ή την χρήση κάποιων παραδειγμάτων κώδικα.
3. Να εξελιχθεί σε μια πολυγλωσσική εφαρμογή, δηλαδή να μπορεί ο χρήστης να επιλέξει την γλώσσα που θέλει να εμφανίζεται η εφαρμογή π.χ. στα αγγλικά, γερμανικά κ.ο.κ.
4. Επίσης θα μπορούν να γίνουν κάποιες επιπλέον ενέργειες για τη διευκόλυνση του χρήστη. Όπως να μπορεί ο χρήστης να χρησιμοποιεί τις λειτουργίες copy, paste, cut. Ακόμη μια λειτουργία που θα μπορούσε να προστεθεί είναι στο πλαίσιο που θα γράφει ο φοιτητής τον κώδικα του να μπορεί να ανοίξει ακόμη ένα πλαίσιο για να μπορεί αν θέλει να πραγματοποιήσει κάποια ταυτόχρονη ή επιπλέον εργασία.
5. Μια επιπλέον λειτουργία είναι να υπάρχει η δυνατότητα «save as» – «αποθήκευση ως» που υπάρχει και σε άλλα προγράμματα. Δηλαδή ο χρήστης να μπορεί να αποθηκεύσει τον κώδικα του και σε άλλες μορφές εκτός από .cpp που αποθηκεύεται στην τρέχουσα εφαρμογή αυτόματα.
6. Να υπάρχει η δυνατότητα αποθήκευσης του γραφήματος που θα εμφανιστεί μετά την εκτέλεση του προγράμματος σε κάποια μορφή που επιθυμεί ο χρήστης όπως .bmp, .jpg, .gif και άλλα.
7. Να υπάρχει μορφοποίηση του κώδικα, δηλαδή να μπορεί να επιλέξει την γραμματοσειρά που επιθυμεί, το μέγεθος των γραμμάτων το χρώμα και άλλες παρόμοιες μορφοποιήσεις.
8. Η ύπαρξη δυνατότητας εκτύπωσης της γραφικής απεικόνισης, να μπορεί δηλαδή εάν θέλει ο χρήστης να εκτυπώσει το γράφημα.
9. Η προσθήκη πλαισίου μέσα στην εφαρμογή που να εμφανίζει το γράφημα μετά την εκτέλεση και όχι σε νέο παράδειγμα.
10. Η δυνατότητα λειτουργίας autocomplete. Δηλαδή σε κάποιες συγκεκριμένες λέξεις-εντολές όταν θα γράφει ο χρήστης τα πρώτα γράμματα θα εμφανίζεται από κάτω μια λίστα με λέξεις που θα ταιριάζουν για να επιλέγει αυτή που επιθυμεί, για περισσότερη διευκόλυνση του χρήστη στην σύνταξη του κώδικα.

ΠΑΡΑΡΤΗΜΑ 1

ΕΓΧΕΙΡΙΔΙΟ ΕΓΚΑΤΑΣΤΑΣΗΣ ΕΦΑΡΜΟΓΗΣ

Βήμα 1^ο

Από το φάκελο LearningGraphics → LearningGraphics → Debug της εφαρμογής εκτελείται το αρχείο LearningGraphics

Όνομα	Ημερομηνία τροπ...	Τύπος	Μέγεθος
DotNetFX40Client	9/11/2014 2:25 μμ	Φάκελος α...	
vcredist_x64	9/11/2014 2:25 μμ	Φάκελος α...	
vcredist_x86	9/11/2014 2:25 μμ	Φάκελος α...	
WindowsInstaller3_1	9/11/2014 2:25 μμ	Φάκελος α...	
LearningGraphics	9/11/2014 11:59 πμ	Πακέτο τ...	506,480 KB

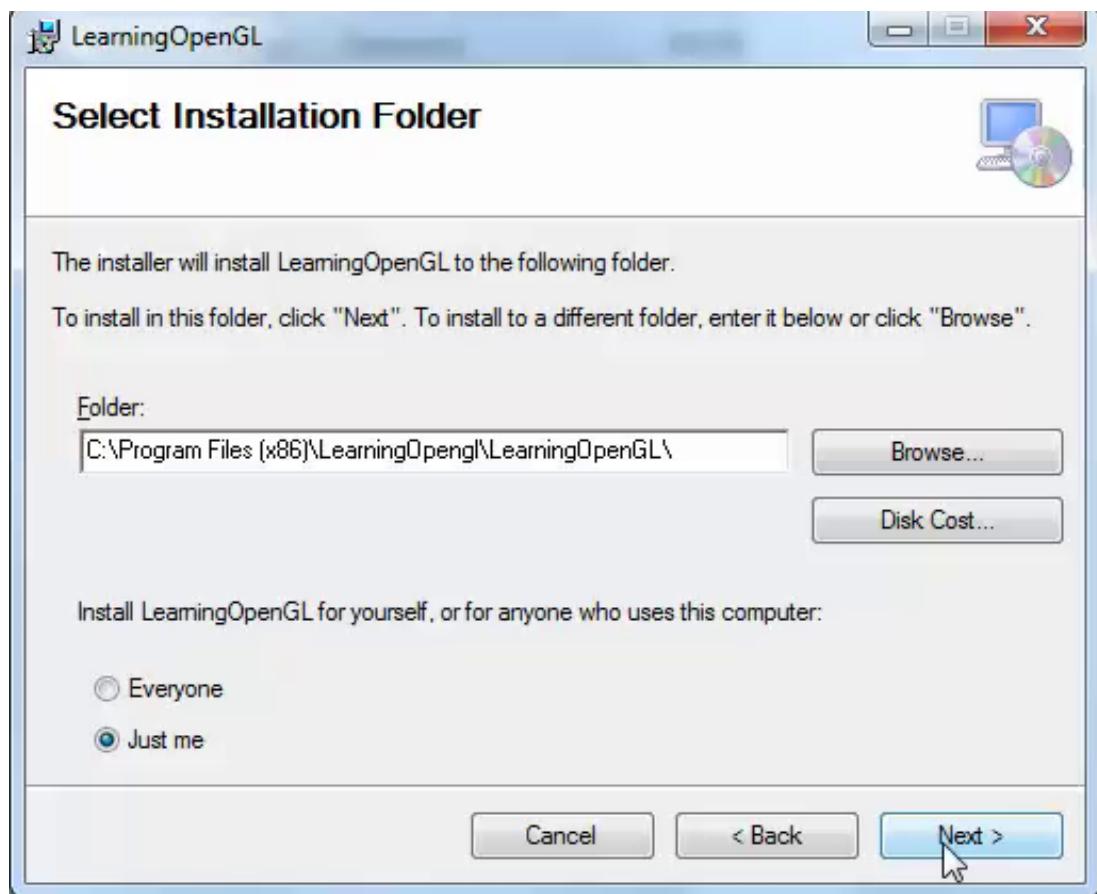
Βήμα 2^ο

Στη συνέχεια επιλέγεται το κουμπί Next.



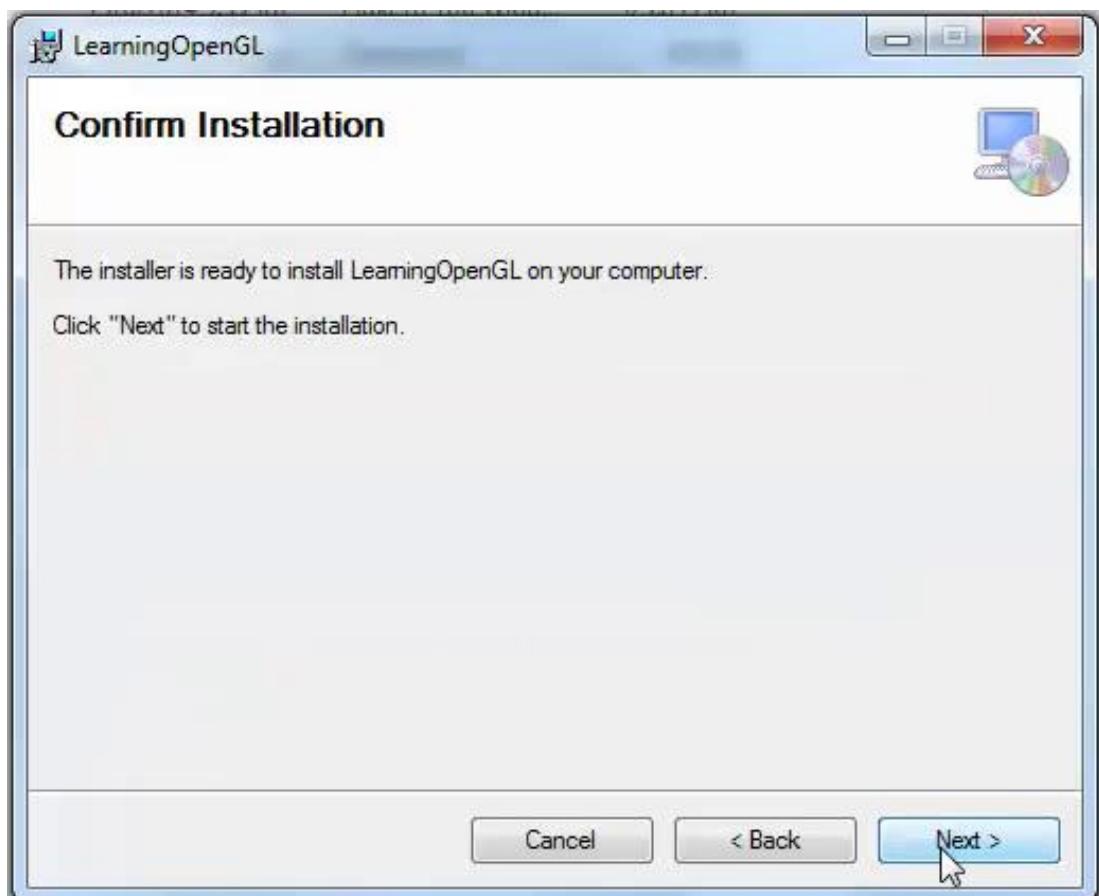
Βήμα 3^ο

Σε αυτό το παράθυρο ζητείται το path όπου ο χρήστης επιθυμεί να εγκαταστήσει την εφαρμογή. Εδώ το path που προτείνεται είναι C:\Program Files(x86)\ LearningOpengl\LearningOpenGL\. Ο χρήστης φυσικά θα μπορεί να επιλέξει διαφορετικό path (μονοπάτι) με τη βοήθεια του κουμπιού Browse. Επιπλέον, πιο κάτω υπάρχουν οι επιλογές “Everyone” και “Just Me”, ανάλογα ποιοι χρήστες του υπολογιστή επιτρέπεται να έχουν πρόσβαση στην εφαρμογή. Όποια επιλογή και να πατήσει ο χρήστης δεν επηρεάζει τις βασικές παραμέτρους της εφαρμογής. Στη συνέχεια επιλέγεται το κουμπί Next.



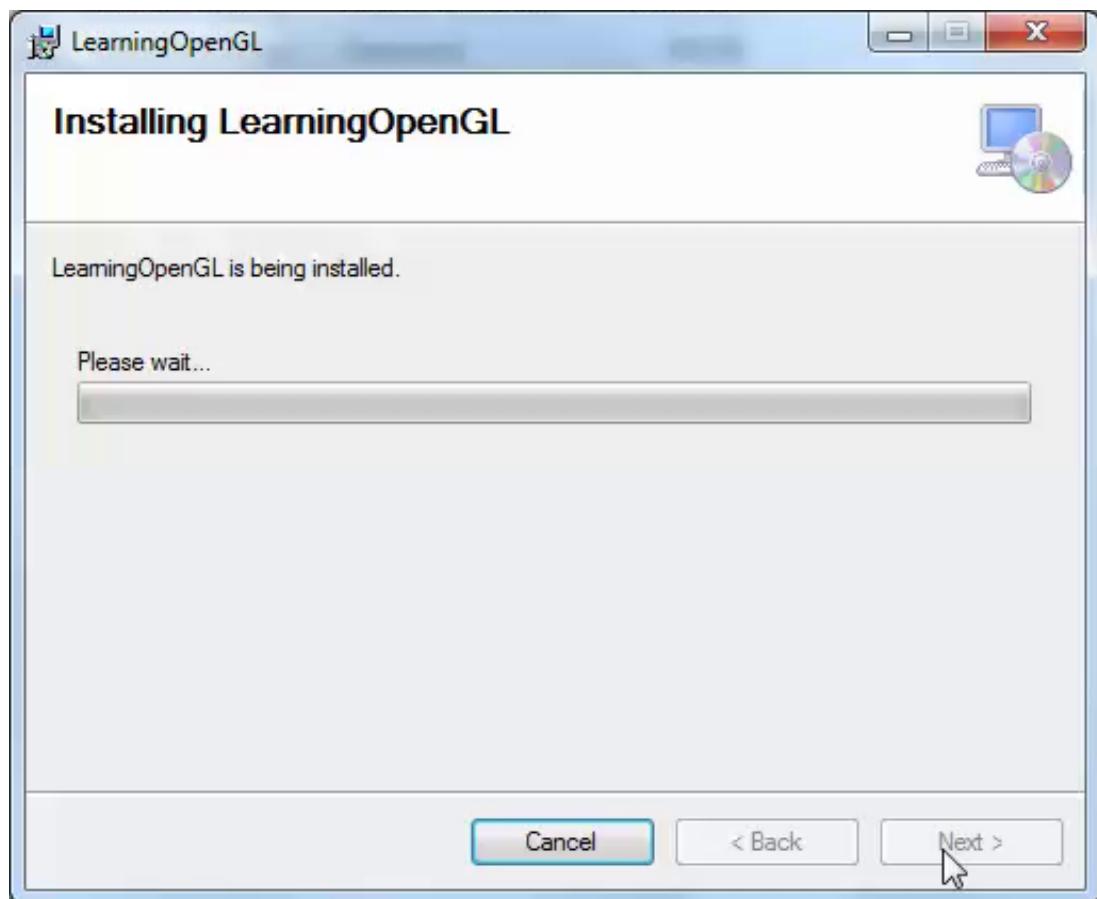
Βήμα 4^ο

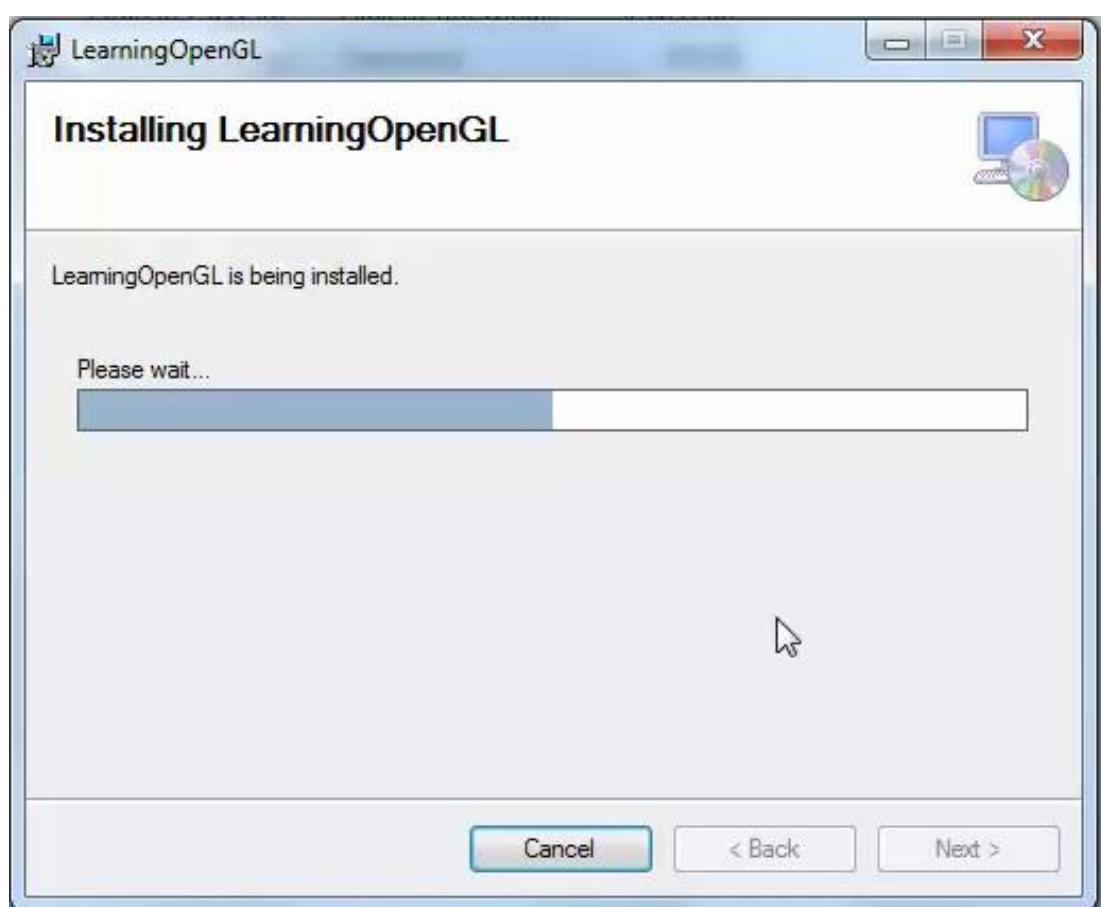
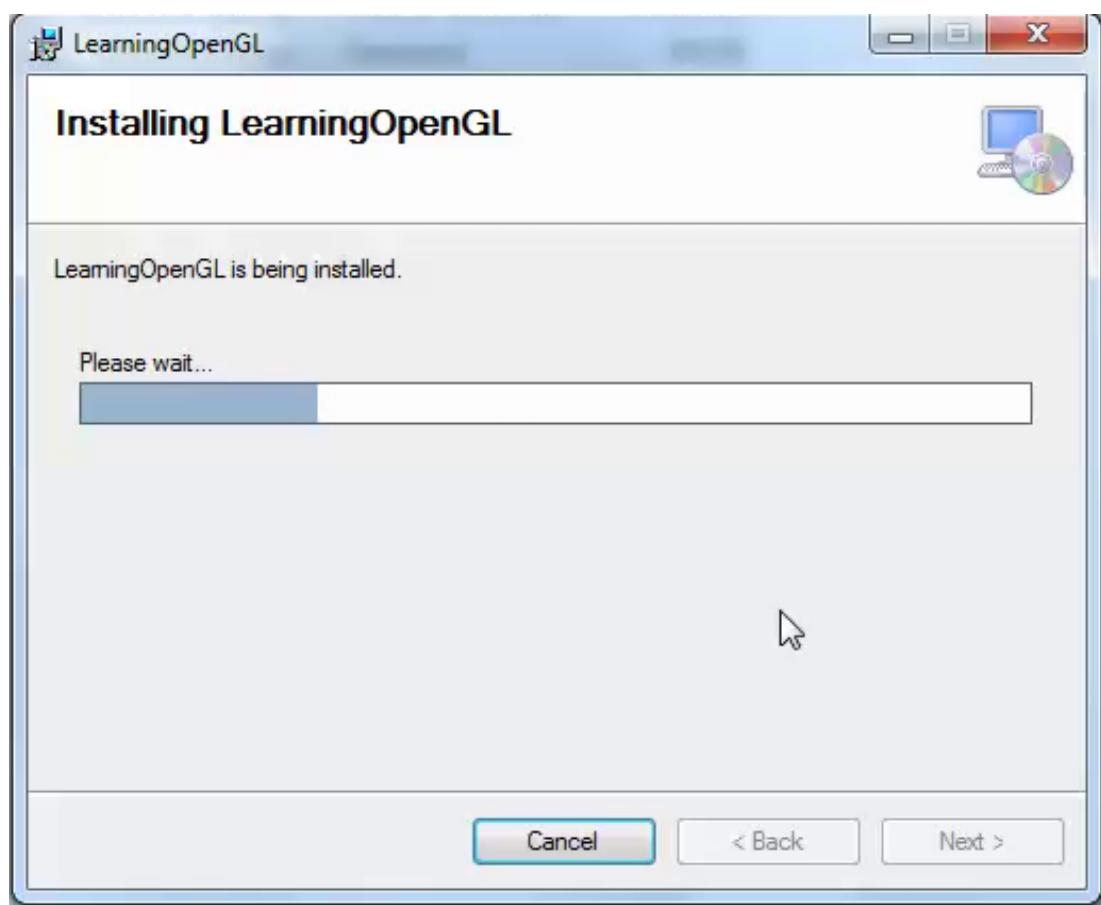
Εφόσον ο χρήστης είναι σύγουρος για τα παραπάνω βήματα που πραγματοποίησε, του ζητείται μια επιβεβαίωση ώστε να ξεκινήσει η εγκατάσταση της εφαρμογής. Αν θέλει να ξεκινήσει η εγκατάσταση πατάει το κουμπί Next, αν θέλει να αλλάξει κάτι από τα προηγούμενα βήματα πατάει Back και αν θέλει να ακυρώσει την εγκατάσταση πατάει Cancel.

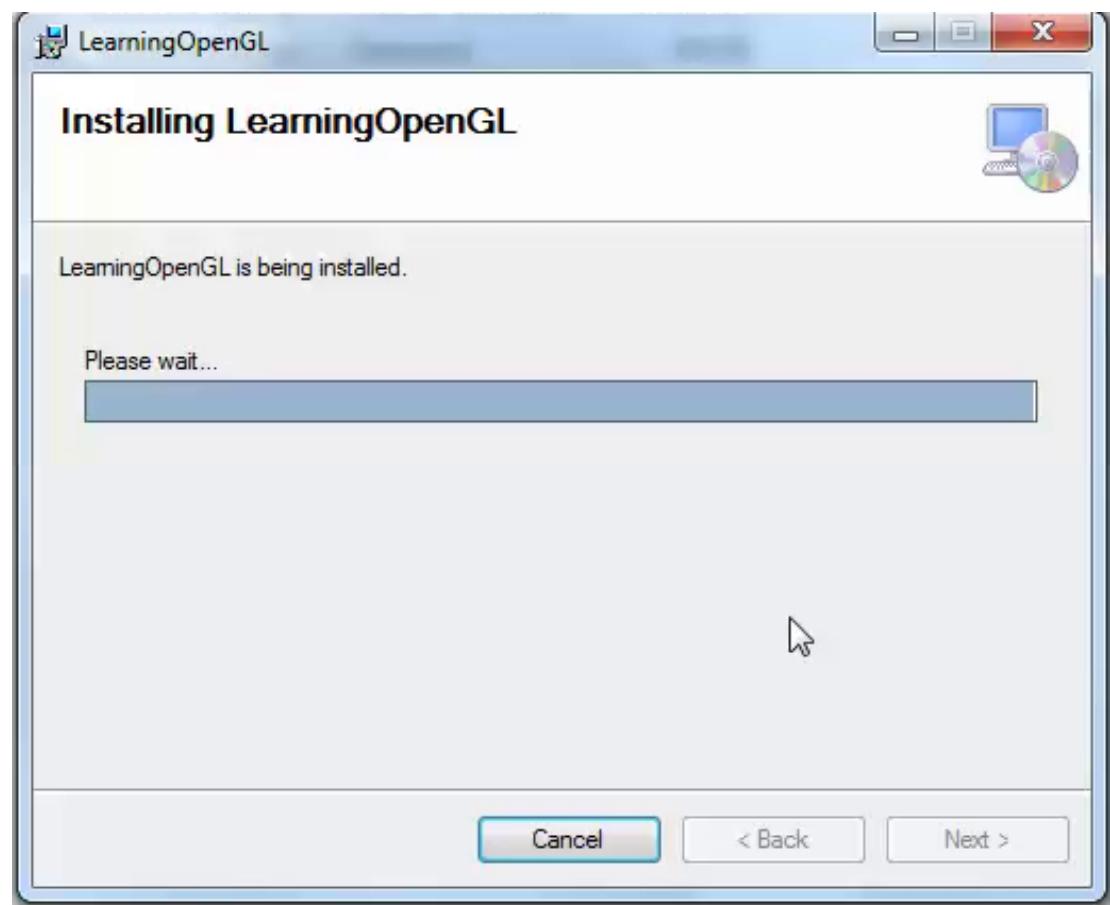


Βήμα 5^ο

Εφόσον ο χρήστης επιβεβαιώσει ότι επιθυμεί να ξεκινήσει η εγκατάσταση, εμφανίζεται το παρακάτω παράθυρο με την πρόοδο της εγκατάστασης.

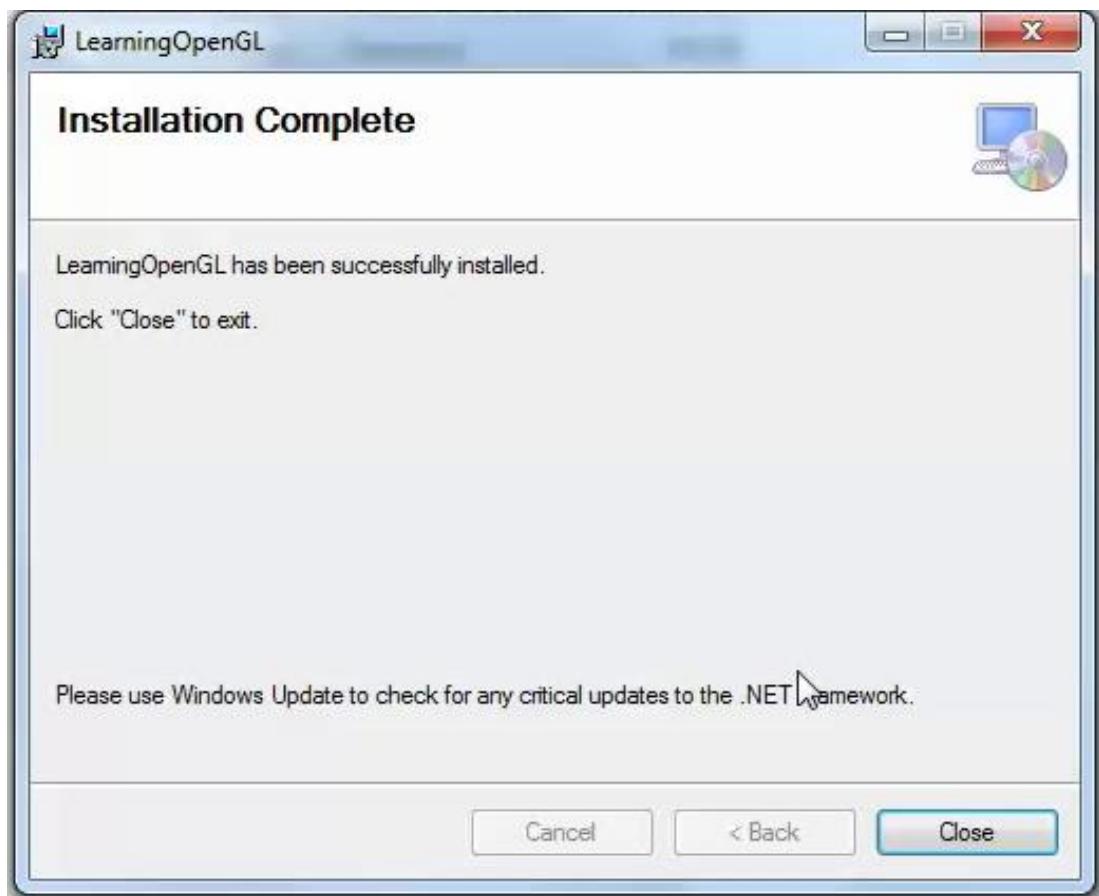






Βήμα 6^ο

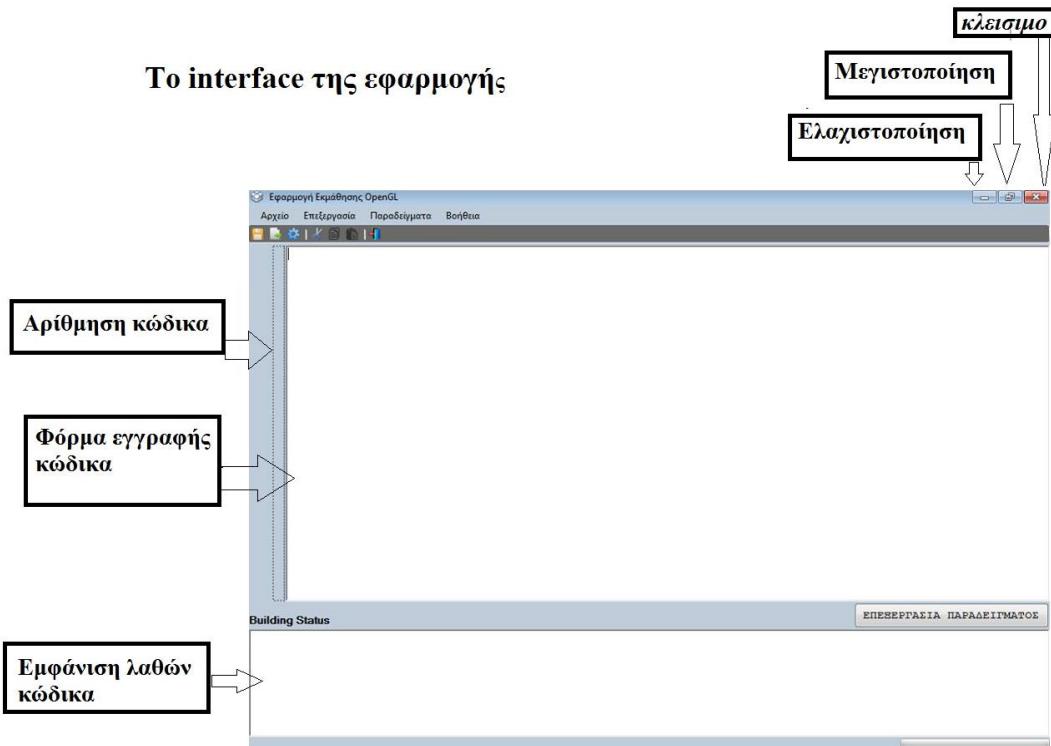
Όταν η εγκατάσταση φτάσει στο τέλος και είναι επιτυχής εμφανίζεται το παρακάτω παράθυρο.



Ο χρήστης επιλέγει το κουμπί Close για να βγει από την εγκατάσταση. Έπειτα πηγαίνει στο φάκελο όπου είχε επιλέξει να εγκατασταθεί η εφαρμογή, επιλέγει το αρχείο της εγκατεστημένης πλέον εφαρμογής με την ονομασία Graphics.exe και ανοίγει την εφαρμογή (εκτέλεση). Μπορεί επίσης να εκτελέσει την εφαρμογή από τη συντόμευση που εμφανίζεται στην επιφάνεια εργασίας.

ΠΑΡΑΡΤΗΜΑ 2

ΕΓΧΕΙΡΙΔΙΟ ΧΡΗΣΗΣ



Σχήμα Π2.1 To interface της εφαρμογής

Επεξήγηση εικονιδίων

- → Αποθήκευση εργασίας
- → Εισαγωγή παραδείγματος
- → Εκτέλεση κώδικα
- → Έξοδος
- → Αποκοπή
- → Αντιγραφή
- → Επικόλληση

Επεξήγηση μενού

Το μενού **Αρχείο** περιλαμβάνει τις λειτουργίες: Αποθήκευση και Έξοδος, Έξοδος.

Το μενού **Επεξεργασία** περιλαμβάνει τις λειτουργίες: Αποκοπή, Αντιγραφή, Επικόλληση, Αναίρεση.

Το μενού **Παραδείγματα** περιλαμβάνει τις λειτουργίες: 2D, 3D, Animation, Light και Custom.

Η λειτουργία **2D** περιλαμβάνει κώδικα και την απεικόνιση δισδιάστατων παραδειγμάτων.

Η λειτουργία **3D** περιλαμβάνει κώδικα και την απεικόνιση τρισδιάστατων παραδειγμάτων.

Η λειτουργία **Animation** περιλαμβάνει κώδικα και την απεικόνιση κινούμενων παραδειγμάτων.

Η λειτουργία **Light** περιλαμβάνει κώδικα και την απεικόνιση παραδειγμάτων με φωτισμό.

Η λειτουργία **Custom** περιλαμβάνει τα παραδείγματα που πρόσθεσε ο χρήστης για να μπορεί να τα βρει και να τα επεξεργαστεί.

Το μενού **Βοήθεια** περιλαμβάνει τις λειτουργίες: Εγχειρίδιο χρήσης εφαρμογής, Εγχειρίδιο χρήσης παραδειγμάτων, Εγχειρίδιο χρήσης OpenGL.

Λειτουργία εφαρμογής

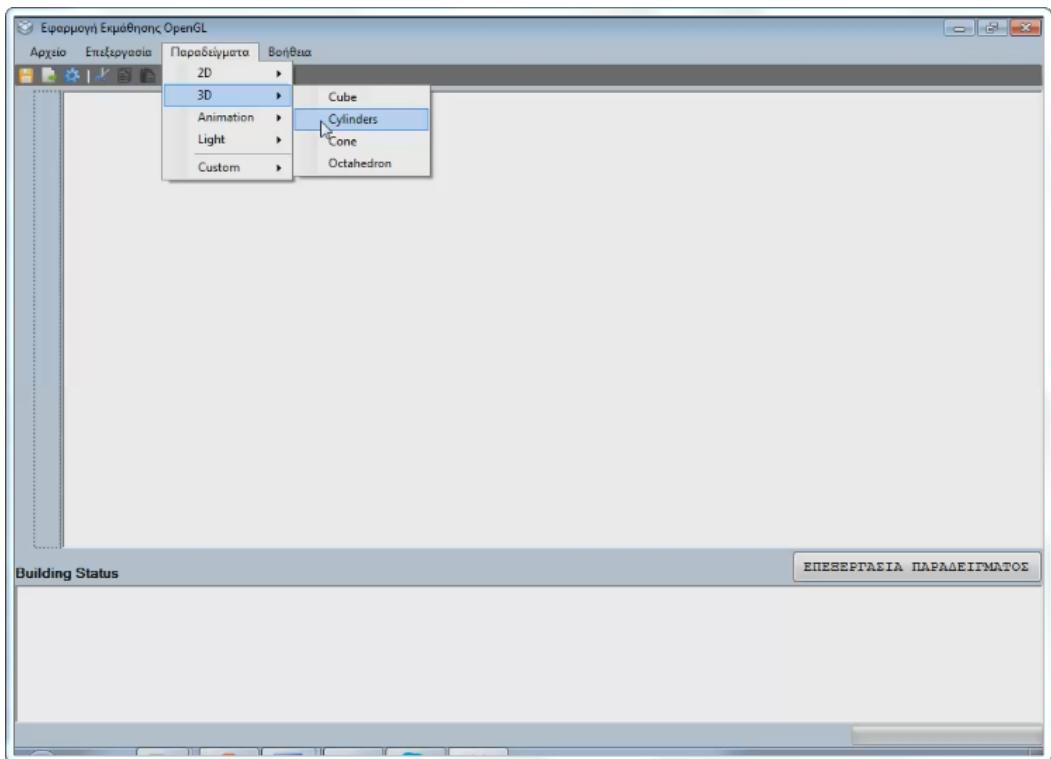
1^ο βήμα

Κατά την **πρώτη εκκίνηση** της εφαρμογής πρέπει χειροκίνητα να εξάγει ο χρήστης τα παραδείγματα για να λειτουργήσουν στην εφαρμογή. Μαζί με την εγκατάσταση της εφαρμογής θα υπάρχει και ένας συμπιεσμένος φάκελος με τα παραδείγματα που ο χρήστης τον τοποθετεί στον τοπικό του δίσκο και τα παραδείγματα είναι έτοιμα για χρήση στην εφαρμογή.

2^ο βήμα

Ανοιγμα ενός παραδείγματος

α) Επιλογή παραδείγματος



Σχήμα Π2.2 Παράδειγμα κύλινδροι

β) Εμφάνιση κώδικα

A screenshot of the same OpenGL application window. The code editor displays the following C code:

```
1 // A Reusable gluQuadric object:
2
3 GLUquadricObj* myReusableQuadric = 0;
4
5
6 void drawGluSlantCylinder( double height, double radiusBase, double radiusTop, int slices, int stacks )
7 {
8
9     if ( ! myReusableQuadric ) {
10
11         myReusableQuadric = gluNewQuadric();
12
13         // Should (but don't) check if pointer is still null --- to catch memory allocation errors.
14
15         gluQuadricNormals( myReusableQuadric, GL_TRUE );
16
17     }
18
19     // Draw the cylinder.
20
21     gluCylinder( myReusableQuadric, radiusBase, radiusTop, height, slices, stacks );
22
23 }
```

The code is intended to draw a slanted cylinder using the gluCylinder function with a reusable gluQuadric object.

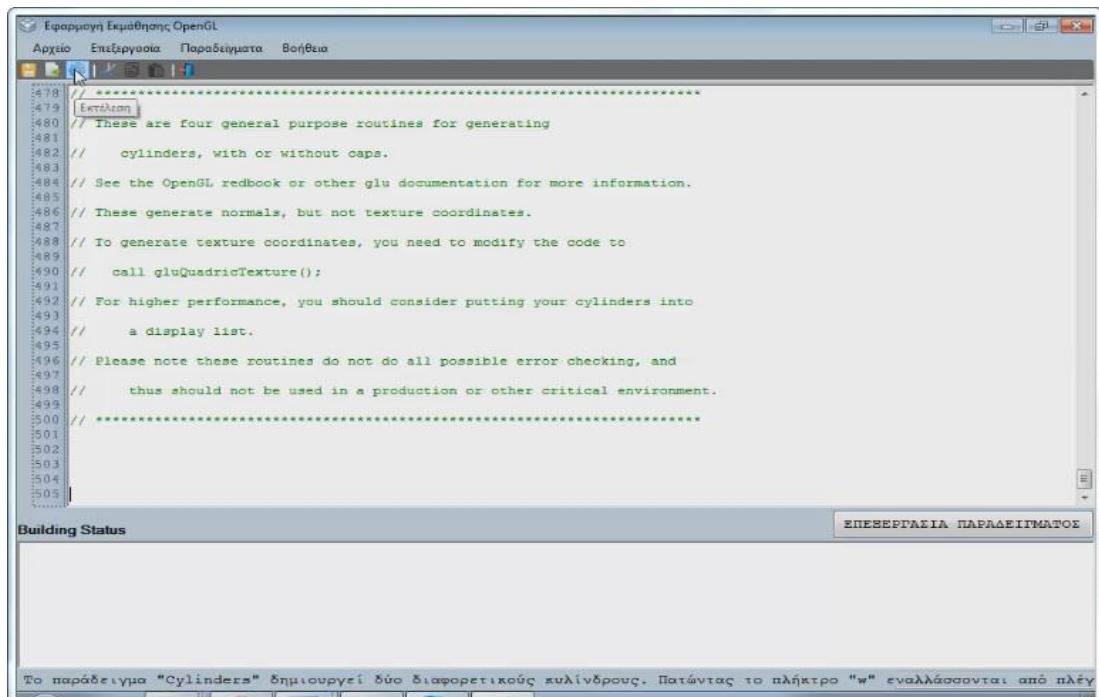
Σχήμα Π2.3 Εμφάνιση κώδικα κυλίνδρου

Π2.3

3^ο βήμα

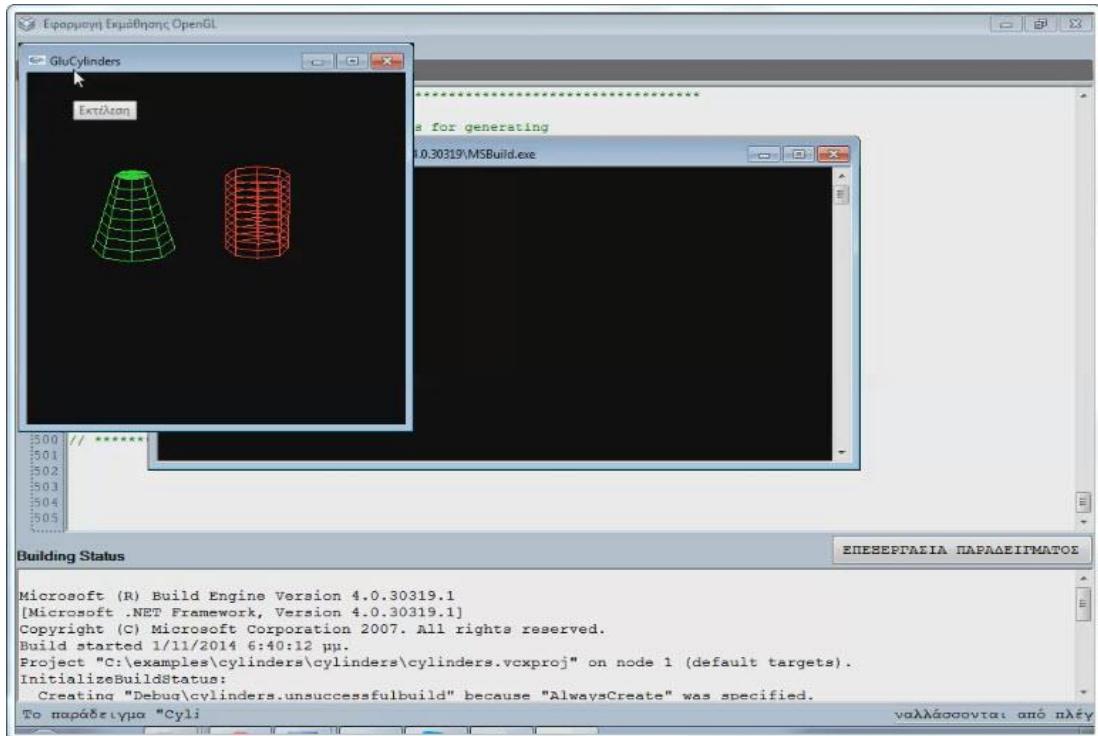
Εκτέλεση κώδικα

α) Επιλογή κουμπιού



Σχήμα Π2.4 Επιλογή λειτουργίας εκτέλεσης κώδικα

β)Εμφάνιση γραφήματος



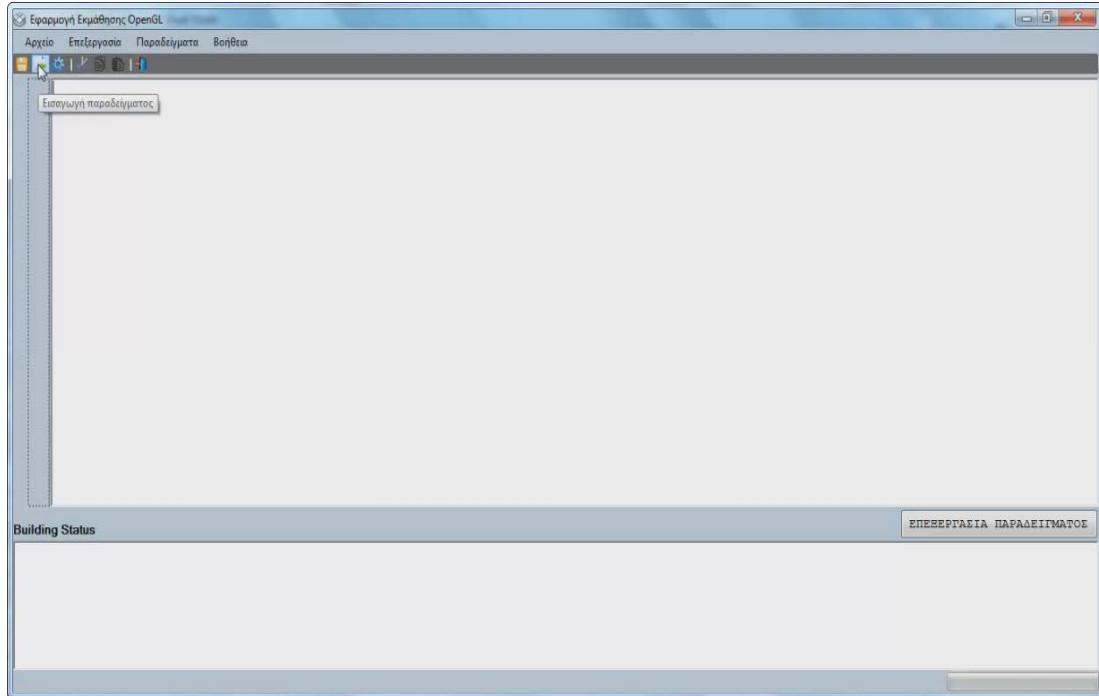
Σχήμα Π2.5 Εμφάνιση γραφήματος

Σημείωση: Εάν ο χρήστης πατήσει εκτέλεση κώδικα και εμφανιστεί 0 warnings 0 Errors χωρίς να εμφανιστεί το γράφημα πρέπει να ξαναπατήσει ακόμη μια φορά το κουμπί εκτέλέση και θα εμφανιστεί το γράφημα.

4^ο βήμα

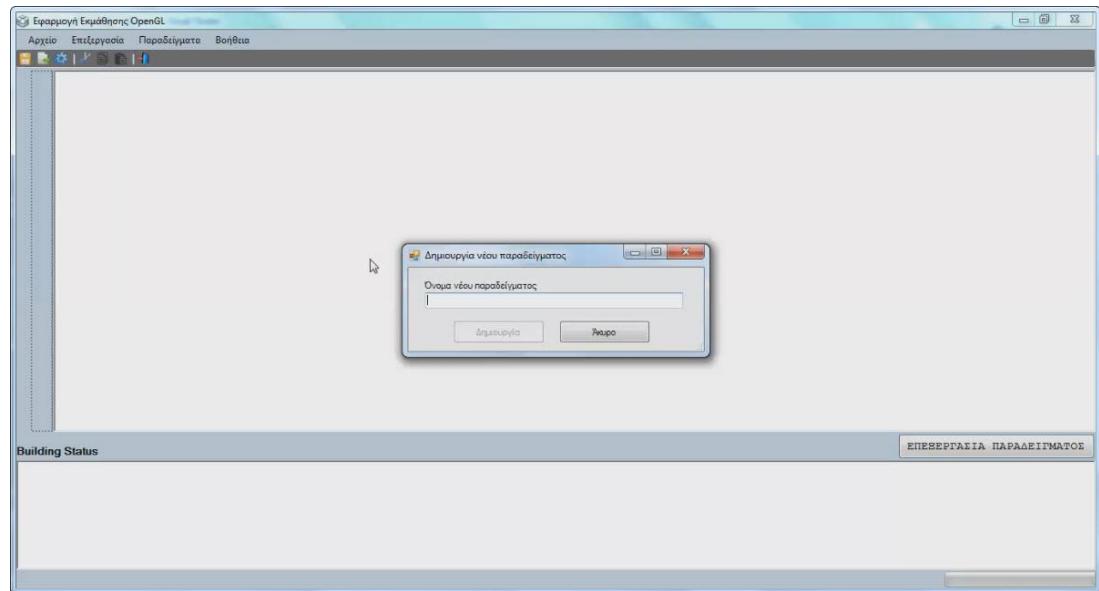
Εισαγωγή παραδείγματος

α) Επιλογή κουμπιού



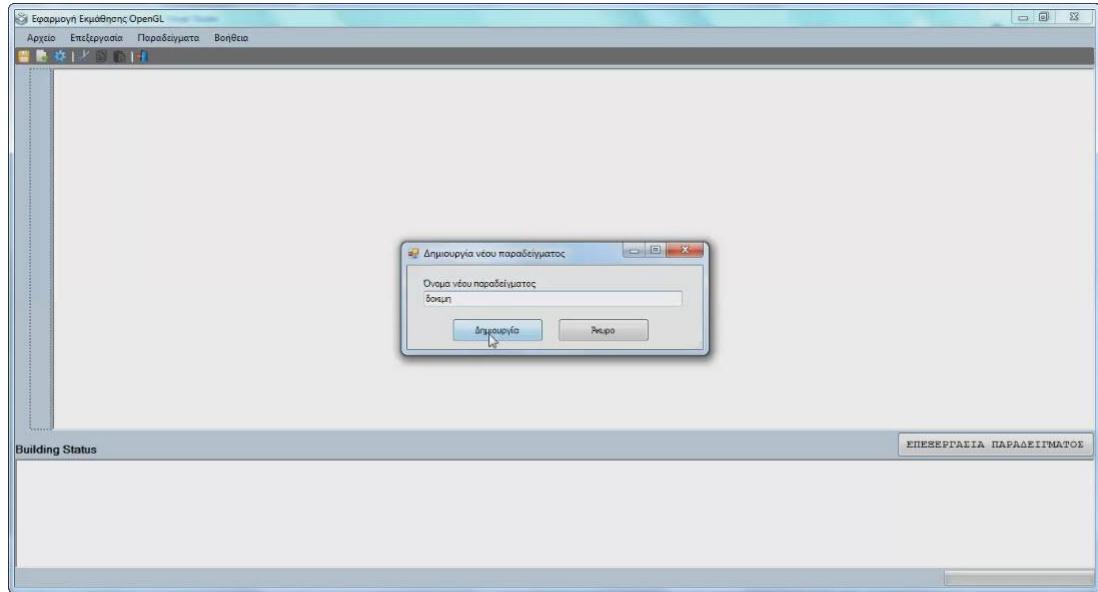
Σχήμα Π2.6 Επιλογή κουμπιού για εισαγωγή παραδείγματος

β) Εμφάνιση παραθύρου για εισαγωγή ονόματος εργασίας



Σχήμα Π2.7 Άνοιγμα νέου παραθύρου για εισαγωγή παραδείγματος

γ) Εισαγωγή ονόματος εργασίας και δημιουργία



Σχήμα Π2.8 Δημιουργία νέου παραδείγματος

δ) Εμφάνιση βασικών εντολών για εγγραφή κώδικα

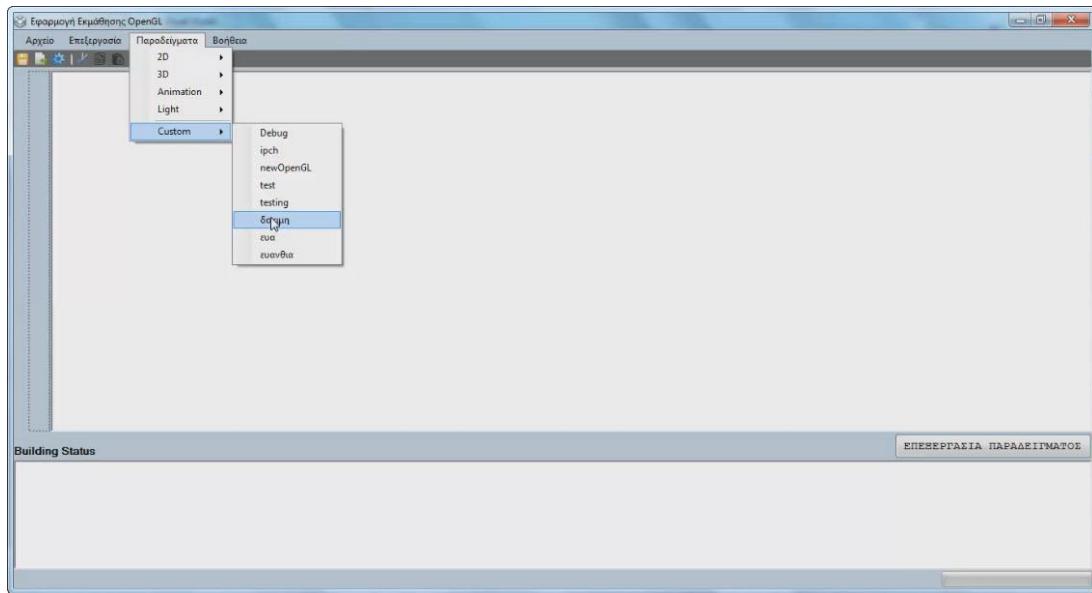
A screenshot of a Windows application window titled "Εφαρμογή Εκμάθησης OpenGL". The menu bar and toolbar are identical to the previous screenshot. The main area is a code editor displaying the following C code:

```
1 #include <stdio.h>
2
3 #include <glut.h> // OpenGL Graphics Utility Library
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18 // Main routine
19
20 int main( int argc, char** argv )
21 {
22
23
24
25
26
27
28 }
```

A cursor is visible in the code editor. The status bar at the bottom shows "Building Status" and "ΕΠΕΞΕΡΓΑΣΙΑ ΠΑΡΑΔΕΙΓΜΑΤΟΣ".

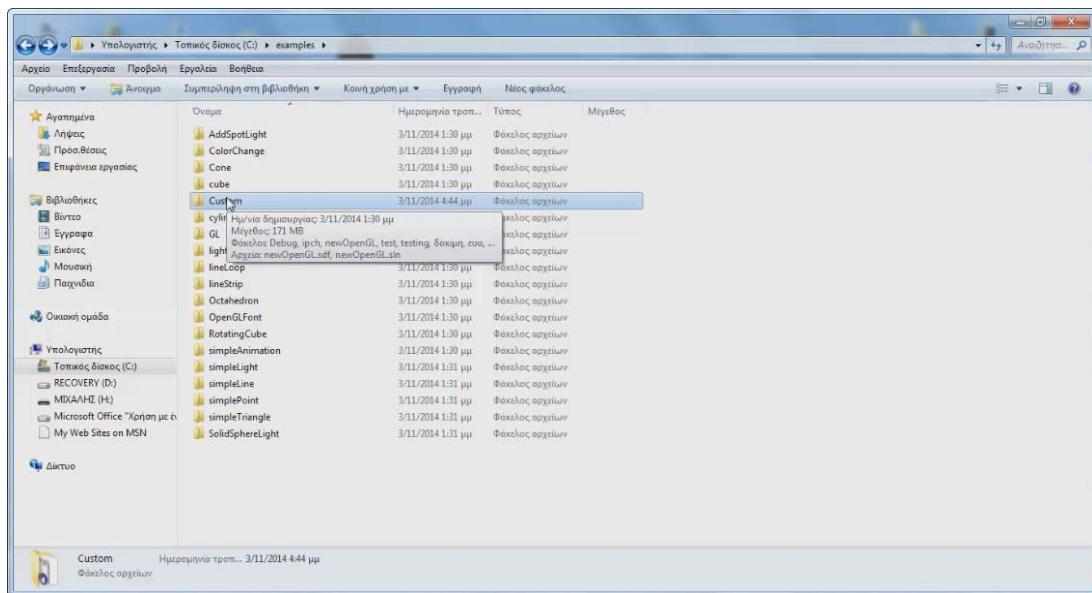
Σχήμα Π2.9 Βασικές εντολές για εγγραφή κώδικα

ε) Εμφάνιση νέου παραδείγματος στο μενού custom

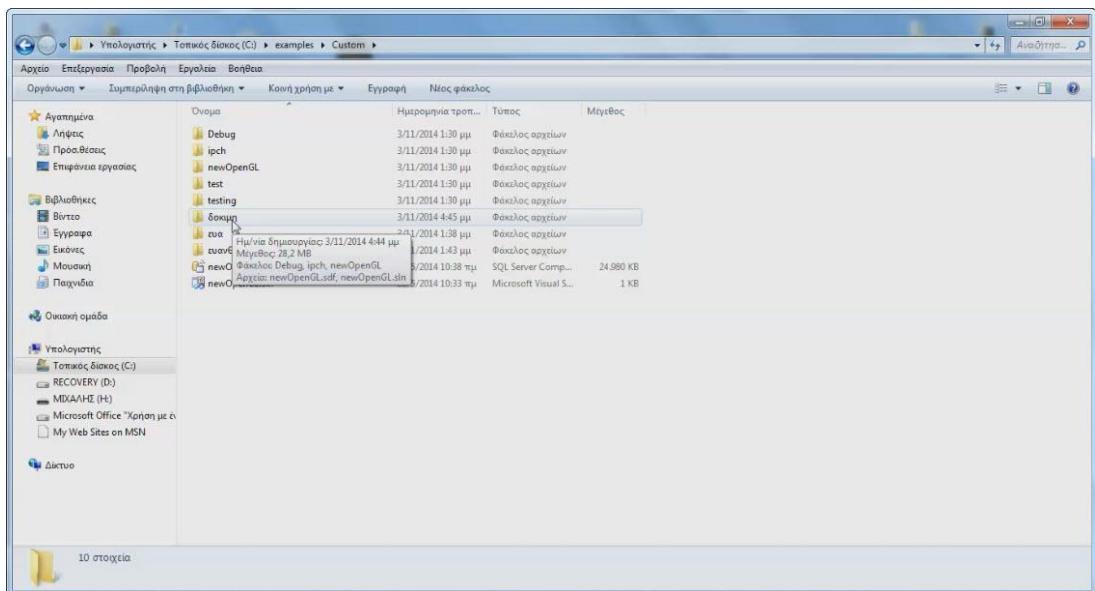


Σχήμα Π2.10 Εμφάνιση νέου παραδείγματος στο μενού custom

ζ) Η εμφάνιση και η δημιουργία του νέου παραδείγματος στον τοπικό δίσκο στο φάκελο custom.



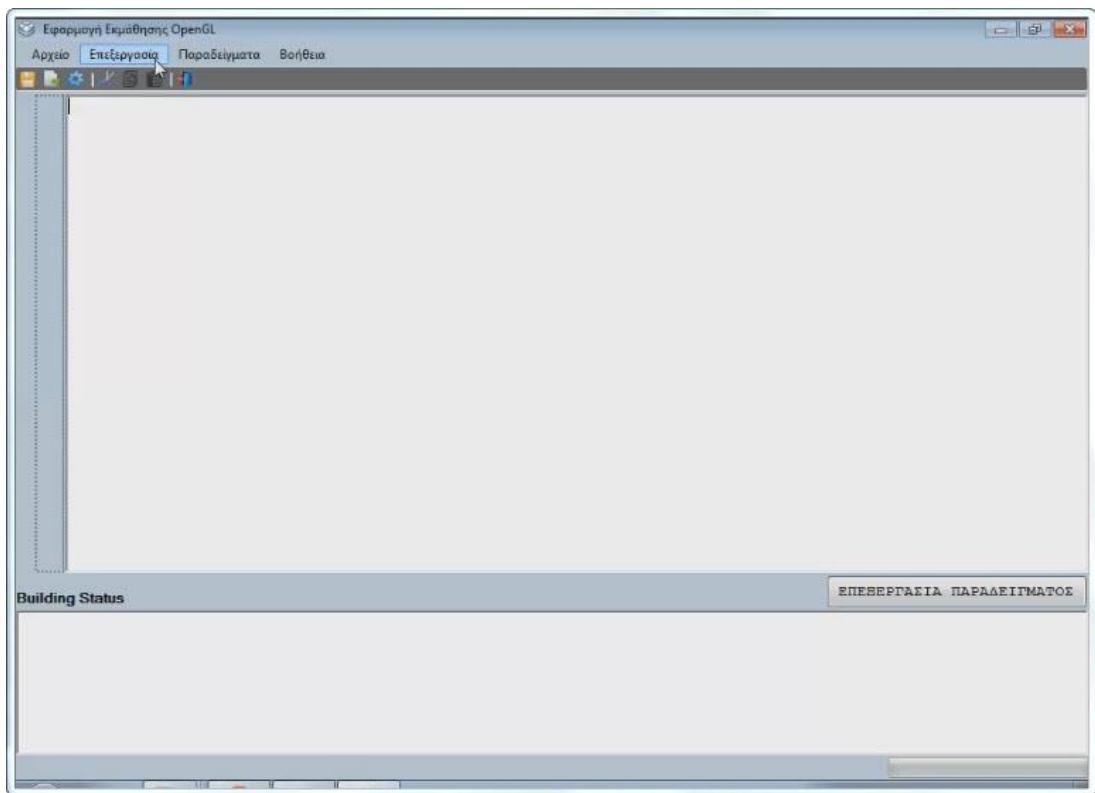
Σχήμα Π2.11 Το παράδειγμα εμφανίζεται στο τοπικό δίσκο στο φάκελο των παραδειγμάτων στο φάκελο custom



Σχήμα Π2.12 Το παράδειγμα εμφανίζεται στο τοπικό δίσκο στο φάκελο των παραδειγμάτων στο φάκελο custom

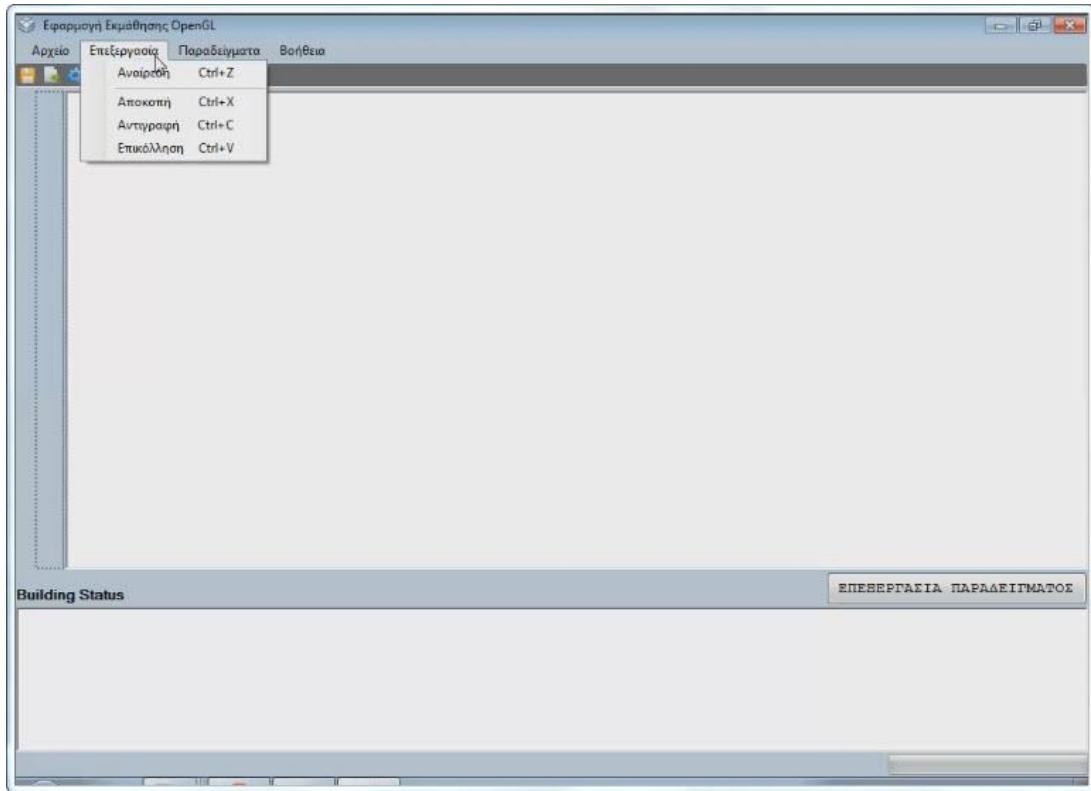
5^ο βήμα

α) Επιλογή μενού Επεξεργασία



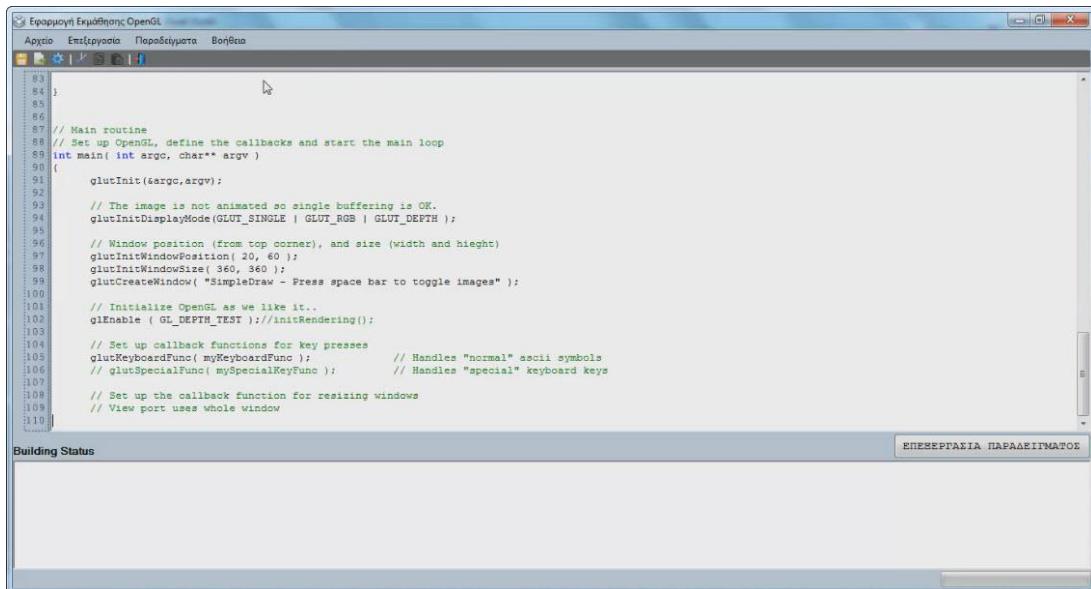
Σχήμα Π2.13 Επιλογή μενού Επεξεργασία

β) Εμφάνιση λειτουργιών μενού Επεξεργασία



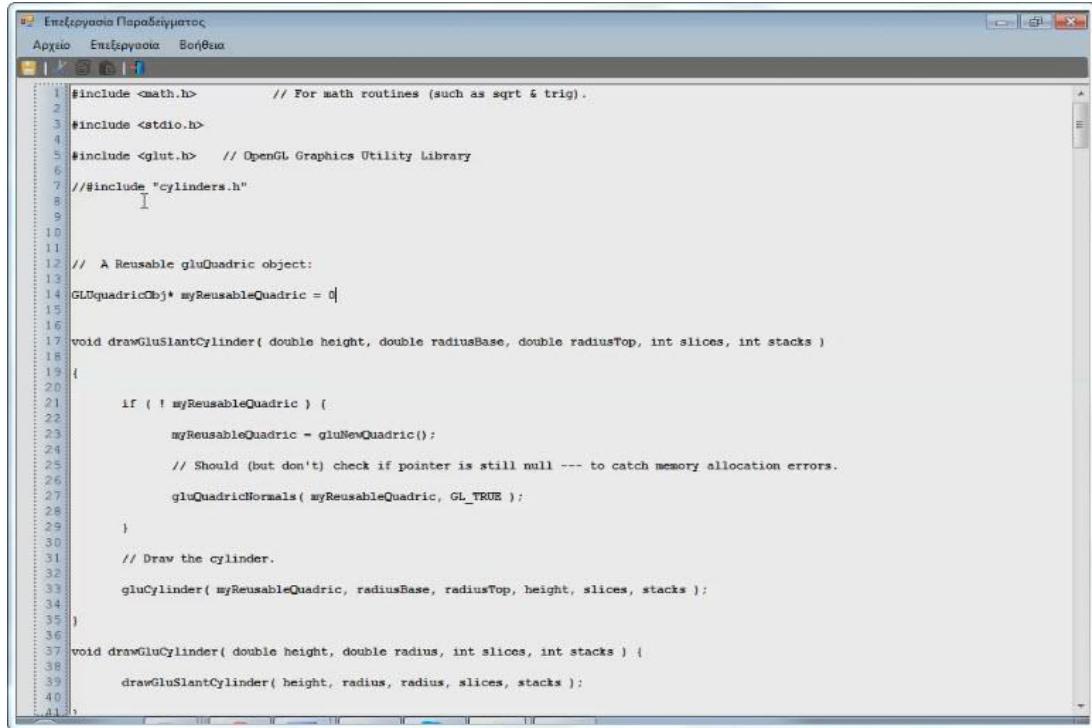
Σχήμα Π2.14 Εμφάνιση περιεχομένων μενού Επεξεργασία

γ) Κουμπί επεξεργασία παραδείγματος και νέα σελίδα



Σχήμα Π2.15 Εμφάνιση ενός παραδείγματος

δ) Εμφάνιση νέας φόρμας επεξεργασίας παραδείγματος. Επεξεργασία παραδείγματος, αφαιρούμε ένα ερωτηματικό.

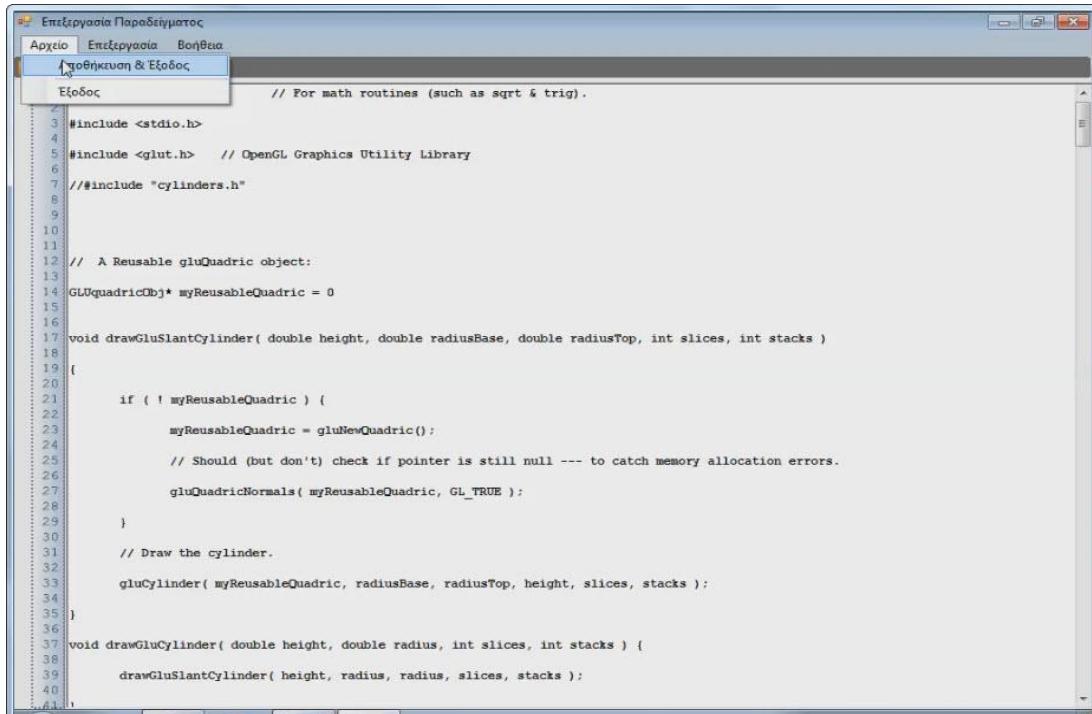


```
#include <math.h>           // For math routines (such as sqrt & trig).
#include <stdio.h>
#include <glut.h>      // OpenGL Graphics Utility Library
//#include "cylinders.h"
I

10
11
12 // A Reusable gluQuadric object:
13
14 GLUquadricObj* myReusableQuadric = 0;
15
16
17 void drawGluSlantCylinder( double height, double radiusBase, double radiusTop, int slices, int stacks )
18 {
19
20     if ( ! myReusableQuadric ) {
21
22         myReusableQuadric = gluNewQuadric();
23
24         // Should (but don't) check if pointer is still null --- to catch memory allocation errors.
25
26         gluQuadricNormals( myReusableQuadric, GL_TRUE );
27
28     }
29
30     // Draw the cylinder.
31
32     gluCylinder( myReusableQuadric, radiusBase, radiusTop, height, slices, stacks );
33
34 }
35
36 void drawGluCylinder( double height, double radius, int slices, int stacks ) {
37
38     drawGluSlantCylinder( height, radius, radius, slices, stacks );
39
40 }
41
```

Σχήμα Π2.16 Εμφάνιση νέας φόρμας επεξεργασίας παραδείγματος

ε) Αποθηκεύεται η επεξεργασία του παραδείγματος



Αρχείο Επεξεργασία Βοήθεια

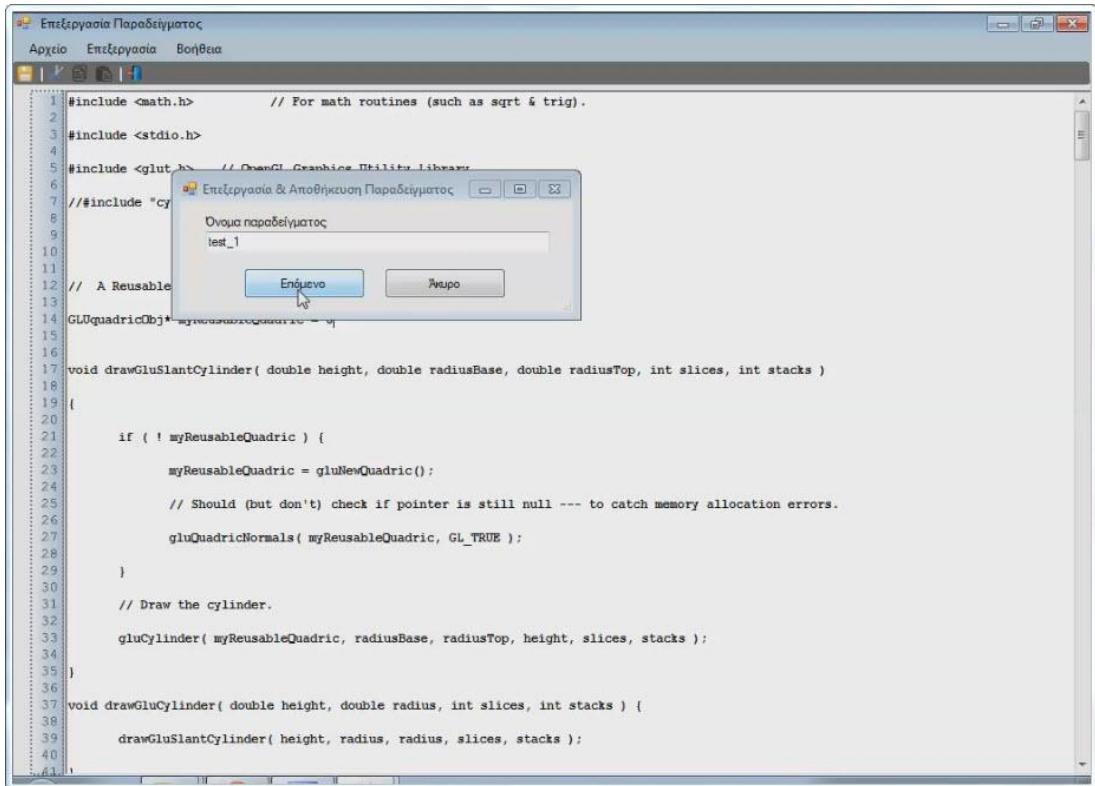
Αρχείο

```
#include <stdio.h>
#include <glut.h>      // OpenGL Graphics Utility Library
//#include "cylinders.h"

10
11
12 // A Reusable gluQuadric object:
13
14 GLUquadricObj* myReusableQuadric = 0;
15
16
17 void drawGluSlantCylinder( double height, double radiusBase, double radiusTop, int slices, int stacks )
18 {
19
20     if ( ! myReusableQuadric ) {
21
22         myReusableQuadric = gluNewQuadric();
23
24         // Should (but don't) check if pointer is still null --- to catch memory allocation errors.
25
26         gluQuadricNormals( myReusableQuadric, GL_TRUE );
27
28     }
29
30     // Draw the cylinder.
31
32     gluCylinder( myReusableQuadric, radiusBase, radiusTop, height, slices, stacks );
33
34 }
35
36 void drawGluCylinder( double height, double radius, int slices, int stacks ) {
37
38     drawGluSlantCylinder( height, radius, radius, slices, stacks );
39
40 }
```

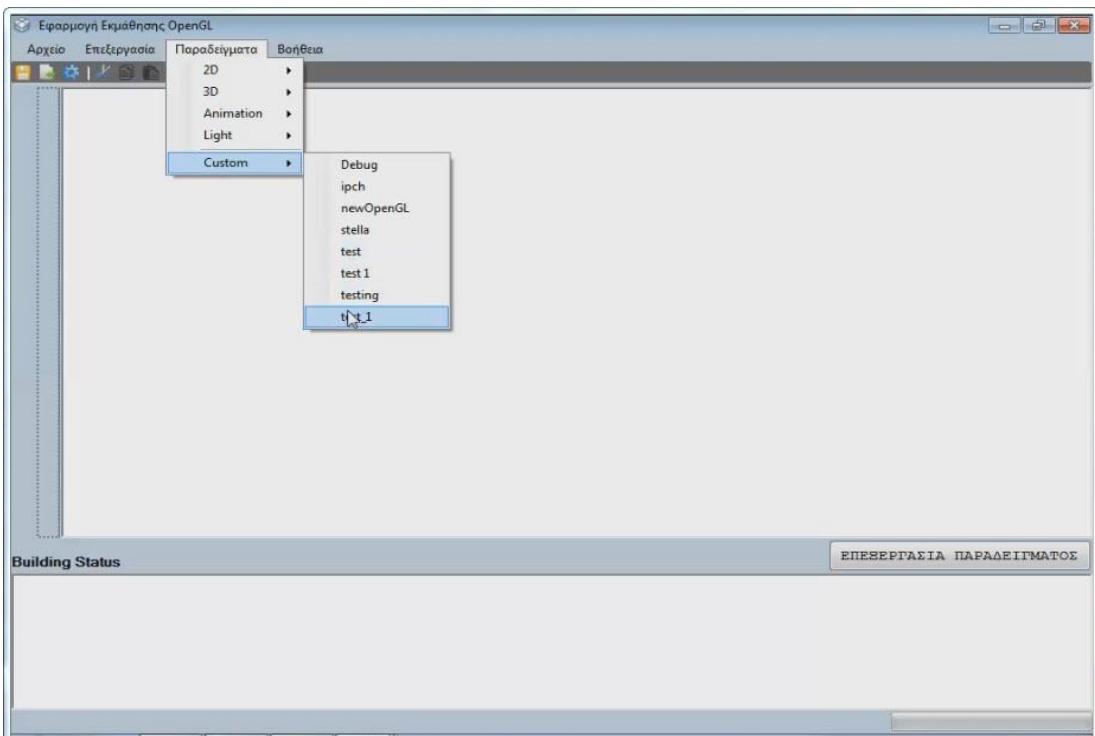
Σχήμα Π2.17 Αποθήκευση Επεξεργασίας Παραδείγματος

ζ) Εισάγεται η ονομασία του παραδείγματος και επιλέγεται επόμενο



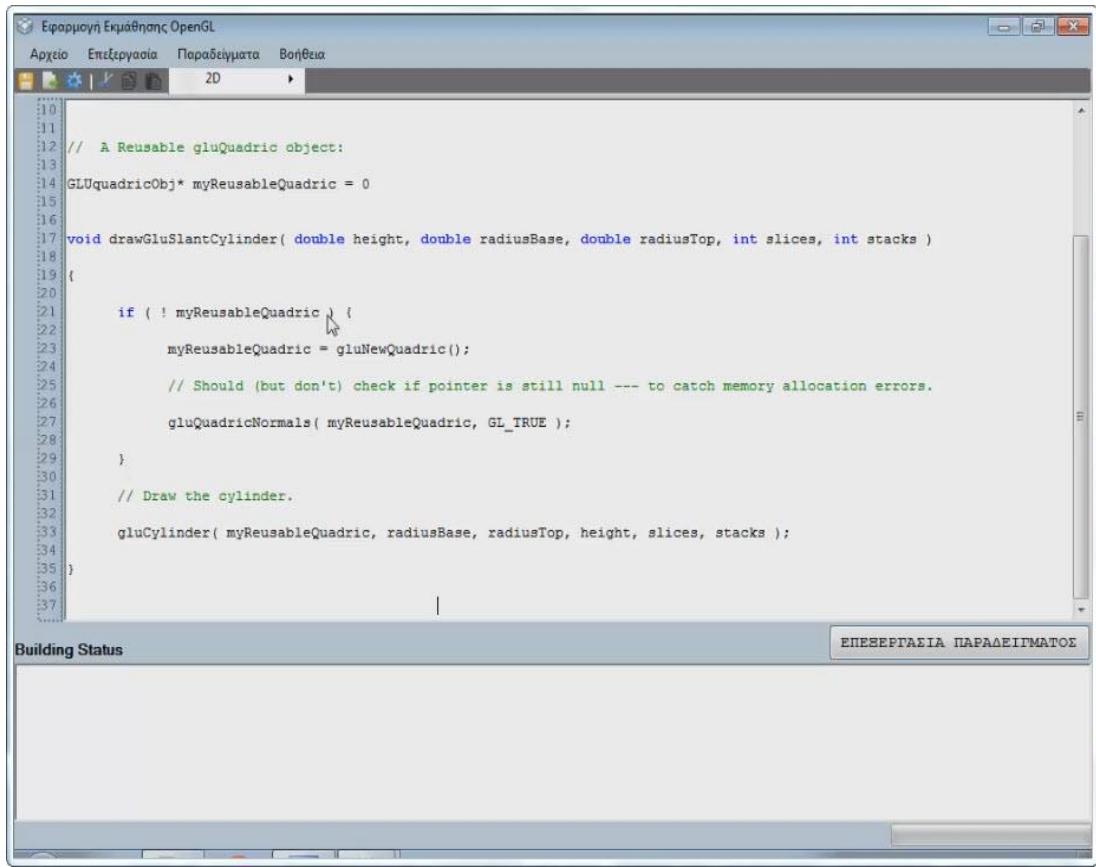
Σχήμα Π2.18 Εισαγωγή ονόματος παραδείγματος

η) Επιλογή φακέλου custom για εύρεση παραδείγματος που επεξεργάστηκε



Σχήμα Π2.19 Επιλογή φακέλου custom και παραδείγματος

θ)Φόρτωση επεξεργασμένου παραδείγματος



The screenshot shows a Windows application window titled "Εφαρμογή Εκμάθησης OpenGL". The menu bar includes "Αρχείο", "Επεξεργασία", "Παραδείγματα", and "Βοήθεια". A toolbar below the menu has icons for file operations and a "2D" button. The main area contains the following C code:

```
10
11
12 // A Reusable gluQuadric object:
13
14 GLUquadricObj* myReusableQuadric = 0
15
16
17 void drawGluSlantCylinder( double height, double radiusBase, double radiusTop, int slices, int stacks )
18
19 {
20
21     if ( ! myReusableQuadric ) {
22         myReusableQuadric = gluNewQuadric();
23
24         // Should (but don't) check if pointer is still null --- to catch memory allocation errors.
25
26         gluQuadricNormals( myReusableQuadric, GL_TRUE );
27
28     }
29
30
31     // Draw the cylinder.
32
33     gluCylinder( myReusableQuadric, radiusBase, radiusTop, height, slices, stacks );
34
35 }
36
37 }
```

Below the code, there are two status bars: "Building Status" and "ΕΠΕΞΕΡΓΑΣΙΑ ΠΑΡΑΔΕΙΓΜΑΤΟΣ".

Σχήμα Π2.20 Φόρτωση επεξεργασμένου παραδείγματος

Προσοχή: Η εφαρμογή δεν λειτουργεί σωστά σε λειτουργικό σύστημα Windows Home Premium.

ΠΑΡΑΡΤΗΜΑ 3

ΠΗΓΑΙΟΣ ΚΩΔΙΚΑΣ

Κλάση FileManager

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Windows.Forms;

namespace Graphics
{
    class FileManager
    {
        /*****
         *          class characteristics
         ****/
        string Filename { get; set; }
        FileStream fRead;
        FileStream fWrite;
        public string textRead { get; set; }
        public byte[] bufRead;
        public byte[] byfWrite;
        /*****
         *          constructor
         ****/
        public FileManager(string filename)
        {
            Filename = filename;
        }
        /*****
         *          writeFile(string text)
         ****/
        public void writeFile(string text)
        {
            //Σβήσιμο όλων των περιεχομένων του αρχείου πριν τη νέα εγγραφή
            System.IO.File.WriteAllText(Filename, string.Empty);
            fWrite = new FileStream(Filename, FileMode.OpenOrCreate, FileAccess.Write);
            try
            {
                if (fWrite.CanWrite)
                {
                    //Γράψιμο των δεδομένων στο αρχείο
                    byfWrite = Encoding.UTF8.GetBytes(text);
                    fWrite.Write(byfWrite, 0, byfWrite.Length);
                }
            }
            finally
            {
                fWrite.Flush();
                fWrite.Close();
            }
        }
    }
}
```

```

        }
    /*****************************************************************/
    public string readFile()
    {
        fRead = new FileStream(Filename, FileMode.Open, FileAccess.Read);
        int bytesRead;
        try
        {
            if (fRead.CanRead)
            {
                bufRead = new byte[fRead.Length];
                bytesRead = fRead.Read(bufRead, 0, bufRead.Length);
                textRead = Encoding.UTF8.GetString(bufRead);
            }
            else
            {
                MessageBox.Show("File could not be read");
            }
        }
        finally
        {
            fRead.Close();
        }
        return textRead;
    }
    /*****************************************************************/
}

```

Κλάση OutputRedirect

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Diagnostics;
using System.Threading;

namespace Graphics
{
    class OutputRedirect
    {
        // Define static variables shared by class methods .
        private static StringBuilder regMsgOutput = null;
        private static int numOutputLines = 0; //sum of lines we get from process output
        public static void InputText(string projPath, string exePath)
        {
            // Get access to mainForm public variables
            var form = Form.ActiveForm as mainForm;
            //Get paths for current example
            //string projPath = "C:\\examples\\\" + form.exampleName + "\\\" + form.exampleName +"\\\" +
            //form.exampleName + ".vcxproj";
            //string exePath = "C:\\examples\\\" + form.exampleName + "\\Debug\\\" + form.exampleName +
            ".exe";
            //Set encoding to windows language
            System.Text.Encoding systemencoding =
System.Text.Encoding.GetEncoding(System.Globalization.CultureInfo.CurrentCulture.TextInfo.OEM
CodePage);

```

```

// Initialize the process and its StartInfo properties .
Process msbuildProcess;
msbuildProcess = new Process();
msbuildProcess.StartInfo.FileName
"C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe";
msbuildProcess.StartInfo.Arguments = projPath;
// Set UseShellExecute to false for redirection .
msbuildProcess.StartInfo.UseShellExecute = false;
// Redirect the standard output of the sort command .
// This stream is read asynchronously using an event handler .
msbuildProcess.StartInfo.RedirectStandardOutput = true;
msbuildProcess.StartInfo.StandardOutputEncoding = systemencoding;
regMsgOutput = new StringBuilder("");
// Set our event handler to asynchronously read the sort output.
msbuildProcess.OutputDataReceived += new DataReceivedEventHandler(OutputStreamHandler);
// Redirect standard input as well . This stream
// is used synchronously .
msbuildProcess.StartInfo.RedirectStandardInput = true;
// Start the process .
msbuildProcess.Start();
// Start the asynchronous read of the output stream .
msbuildProcess.BeginOutputReadLine();
// Wait for the process to write output .
msbuildProcess.WaitForExit();
if (numOutputLines > 0)
{
    // Write the output to the regular messages text box .
    form.regularMsg.Text += regMsgOutput.ToString();
}
msbuildProcess.Close();
if (!regMsgOutput.ToString().Contains("error"))
{
    System.Diagnostics.Process.Start(exePath);
}
}

private static string PrintCPBytes(string str, int codePage)
{
    Encoding targetEncoding;
    byte[] encodedChars;
    // Gets the encoding for the specified code page .
    targetEncoding = Encoding.GetEncoding(codePage);
    // Gets the byte representation of the specified string .
    encodedChars = targetEncoding.GetBytes(str);
    // Prints the bytes .
    string message = "";
    for (int i = 0; i < encodedChars.Length; i++)
        message += encodedChars[i].ToString();
    return message;
}

private static void OutputStreamHandler(object sendingProcess, DataReceivedEventArgs outLine)
{
    // Collect the msbuild command output .
    if (!String.IsNullOrEmpty(outLine.Data))
    {
        numOutputLines++;
        // Add the text to the collected output .
        regMsgOutput.Append(Environment.NewLine + outLine.Data);
    }
}

```

Κλάση Program

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace Graphics
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application .
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new mainForm());
        }
    }
}
```

Κλάση ZipStorer

```
// ZipStorer, by Jaime Olivares
// Website: zipstorer.codeplex.com
// Version: 2.35 (March 14, 2010)
```

```
using System.Collections.Generic;
using System.Text;

namespace System.IO.Compression
{
    // Unique class for compression/decompression file . Represents a Zip file .
    ///
    public class ZipStorer : IDisposable
    {
        // Compression method enumeration
        public enum Compression : ushort {
            // Uncompressed storage
            Store = 0,
            // Deflate compression method
            Deflate = 8 }

        // Represents an entry in Zip file directory
        public struct ZipFileEntry
        {
            // Compression method
            public Compression Method;
            // Full path and filename as stored in Zip
            public string FilenameInZip;
            // Original file size
            public uint FileSize;
            // Compressed file size
            public uint CompressedSize;
            // Offset of header information inside Zip storage
            public uint HeaderOffset;
            // Offset of file inside Zip storage
        }
    }
}
```

```

public uint FileOffset;
// Size of header information
public uint HeaderSize;
// 32-bit checksum of entire file
public uint Crc32;
// Last modification time of file
public DateTime ModifyTime;
// User comment for file
public string Comment;
// True if UTF8 encoding for filename and comments , false if default (CP 437)
public bool EncodeUTF8;

// Overriden method
// Filename in Zip
public override string ToString()
{
    return this.FilenameInZip;
}
public bool IsDirectory()
{
    return this.FilenameInZip.EndsWith("/");
}
#endregion Public fields
// True if UTF8 encoding for filename and comments , false if default (CP 437)
public bool EncodeUTF8 = false;
// Force deflate algotithm even if it inflates the stored file . Off by default .
public bool ForceDeflating = false;
#endregion
#region Private fields
// List of files to store
private List<ZipFileEntry> Files = new List<ZipFileEntry>();
// Filename of storage file
private string FileName;
// Stream object of storage file
private Stream ZipFileStream;
// General comment
private string Comment = "";
// Central dir image
private byte[] CentralDirImage = null;
// Existing files in zip
private ushort ExistingFiles = 0;
// File access for Open method
private FileAccess Access;
// Static CRC32 Table
private static UInt32[] CrcTable = null;
// Default filename encoder
private static Encoding DefaultEncoding = Encoding.GetEncoding(437);
#endregion

#region Public methods
// Static constructor . Just invoked once in order to create the CRC32 lookup table .
static ZipStorer()
{
    // Generate CRC32 table
    CrcTable = new UInt32[256];
    for (int i = 0; i < CrcTable.Length; i++)
    {
        UInt32 c = (UInt32)i;
        for (int j = 0; j < 8; j++)
        {

```

```

        if ((c & 1) != 0)
            c = 3988292384 ^ (c >> 1);
        else
            c >= 1;
    }
    CrcTable[i] = c;
}
// Method to create a new storage file
public static ZipStorer Create(string _filename, string _comment)
{
    Stream stream = new FileStream(_filename, FileMode.Create, FileAccess.ReadWrite);

    ZipStorer zip = Create(stream, _comment);
    zip.Comment = _comment;
    zip.FileName = _filename;
    return zip;
}
// Method to create a new zip storage in a stream
public static ZipStorer Create(Stream _stream, string _comment)
{
    ZipStorer zip = new ZipStorer();
    zip.Comment = _comment;
    zip.ZipFileStream = _stream;
    zip.Access = FileAccess.Write;
    return zip;
}
// Method to open an existing storage file
public static ZipStorer Open(string _filename, FileAccess _access)
{
    Stream stream = (Stream)new FileStream(_filename, FileMode.Open, _access ==
FileAccess.Read ? FileAccess.Read : FileAccess.ReadWrite);
    ZipStorer zip = Open(stream, _access);
    zip.FileName = _filename;
    return zip;
}
// Method to open an existing storage from stream
public static ZipStorer Open(Stream _stream, FileAccess _access)
{
    if (!_stream.CanSeek && _access != FileAccess.Read)
        throw new InvalidOperationException("Stream cannot seek");
    ZipStorer zip = new ZipStorer();
    zip.ZipFileStream = _stream;
    zip.Access = _access;
    if (zip.ReadFileInfo())
        return zip;
    throw new System.IO.InvalidDataException();
}
// Add full contents of a file into the Zip storage
public void AddFile(Compression _method, string _pathname, string _filenameInZip, string
_comment)
{
    if (Access == FileAccess.Read)
        throw new InvalidOperationException("Writing is not allowed");
    FileStream stream = new FileStream(_pathname, FileMode.Open, FileAccess.Read);
    AddStream(_method, _filenameInZip, stream, File.GetLastWriteTime(_pathname), _comment);
    stream.Close();
}
// Add full contents of a stream into the Zip storage

```

```

public void AddStream(Compression _method, string _filenameInZip, Stream _source, DateTime
_modTime, string _comment)
{
    if (Access == FileAccess.Read)
        throw new InvalidOperationException("Writing is not allowed");
    long offset;
    if (this.Files.Count==0)
        offset = 0;
    else
    {
        ZipFileEntry last = this.Files[this.Files.Count-1];
        offset = last.HeaderOffset + last.HeaderSize;
    }
    // Prepare the fileinfo
    ZipFileEntry zfe = new ZipFileEntry();
    zfe.Method = _method;
    zfe.EncodeUTF8 = this.EncodeUTF8;
    zfe.FilenameInZip = NormalizedFilename(_filenameInZip);
    zfe.Comment = (_comment == null ? "" : _comment);
    // Even though we write the header now , it will have to be rewritten , since we don't know
    compressed size or crc .
    zfe.Crc32 = 0; // to be updated later
    zfe.HeaderOffset = (uint)this.ZipFileStream.Position; // offset within file of the start of this local
record
    zfe.ModifyTime = _modTime;
    // Write local header
    WriteLocalHeader(ref zfe);
    zfe.FileOffset = (uint)this.ZipFileStream.Position;
    // Write file to zip (store)
    Store(ref zfe, _source);
    _source.Close();
    this.UpdateCrcAndSizes(ref zfe);
    Files.Add(zfe);
}
// Updates central directory (if pertinent) and close the Zip storage
public void Close()
{
    if (this.Access != FileAccess.Read)
    {
        uint centralOffset = (uint)this.ZipFileStream.Position;
        uint centralSize = 0;
        if (this.CentralDirImage != null)
            this.ZipFileStream.Write(CentralDirImage, 0, CentralDirImage.Length);
        for (int i = 0; i < Files.Count; i++)
        {
            long pos = this.ZipFileStream.Position;
            this.WriteCentralDirRecord(Files[i]);
            centralSize += (uint)(this.ZipFileStream.Position - pos);
        }
        if (this.CentralDirImage != null)
            this.WriteEndRecord(centralSize + (uint)CentralDirImage.Length, centralOffset);
        else
            this.WriteEndRecord(centralSize, centralOffset);
    }
    if (this.ZipFileStream != null)
    {
        this.ZipFileStream.Flush();
        this.ZipFileStream.Dispose();
        this.ZipFileStream = null;
    }
}

```

```

        }

    // Read all the file records in the central directory
    public List<ZipFileEntry> ReadCentralDir()
    {
        if (this.CentralDirImage == null)
            throw new InvalidOperationException("Central directory currently does not exist");
        List<ZipFileEntry> result = new List<ZipFileEntry>();
        for (int pointer = 0; pointer < this.CentralDirImage.Length; )
        {
            uint signature = BitConverter.ToUInt32(CentralDirImage, pointer);
            if (signature != 0x02014b50)
                break;
            bool encodeUTF8 = (BitConverter.ToInt16(CentralDirImage, pointer + 8) & 0x0800) != 0;
            ushort method = BitConverter.ToInt16(CentralDirImage, pointer + 10);
            uint modifyTime = BitConverter.ToInt32(CentralDirImage, pointer + 12);
            uint crc32 = BitConverter.ToInt32(CentralDirImage, pointer + 16);
            uint comprSize = BitConverter.ToInt32(CentralDirImage, pointer + 20);
            uint fileSize = BitConverter.ToInt32(CentralDirImage, pointer + 24);
            ushort filenameSize = BitConverter.ToInt16(CentralDirImage, pointer + 28);
            ushort extraSize = BitConverter.ToInt16(CentralDirImage, pointer + 30);
            ushort commentSize = BitConverter.ToInt16(CentralDirImage, pointer + 32);
            uint headerOffset = BitConverter.ToInt32(CentralDirImage, pointer + 42);
            uint headerSize = (uint)(46 + filenameSize + extraSize + commentSize);
            Encoding encoder = encodeUTF8 ? Encoding.UTF8 : DefaultEncoding;
            ZipFileEntry zfe = new ZipFileEntry();
            zfe.Method = (Compression)method;
            zfe.FilenameInZip = encoder.GetString(CentralDirImage, pointer + 46, filenameSize);
            zfe.FileOffset = GetFileOffset(headerOffset);
            zfe.FileSize = fileSize;
            zfe.CompressedSize = comprSize;
            zfe.HeaderOffset = headerOffset;
            zfe.HeaderSize = headerSize;
            zfe.Crc32 = crc32;
            zfe.ModifyTime = DosTimeToDateTIme(modifyTime);
            if (commentSize > 0)
                zfe.Comment = encoder.GetString(CentralDirImage, pointer + 46 + filenameSize + extraSize, commentSize);
            result.Add(zfe);
            pointer += (46 + filenameSize + extraSize + commentSize);
        }
        return result;
    }

    // Copy the contents of a stored file into a physical file
    public bool ExtractFile(ZipFileEntry _zfe, string _filename)
    {
        // Make sure the parent directory exist
        string path = System.IO.Path.GetDirectoryName(_filename);
        if (!Directory.Exists(path))
            Directory.CreateDirectory(path);
        // Check it is directory. If so, do nothing
        if (Directory.Exists(_filename))
            return true;
        Stream output = new FileStream(_filename, FileMode.Create, FileAccess.Write);
        bool result = ExtractFile(_zfe, output);
        if (result)
            output.Close();
        File.SetCreationTime(_filename, _zfe.ModifyTime);
        File.SetLastWriteTime(_filename, _zfe.ModifyTime);
        return result;
    }
}

```

```

// Copy the contents of a stored file into an opened stream
public bool ExtractFile(ZipFileEntry _zfe, Stream _stream)
{
    if (!_stream.CanWrite)
        throw new InvalidOperationException("Stream cannot be written");
    // check signature
    byte[] signature = new byte[4];
    this.ZipFileStream.Seek(_zfe.HeaderOffset, SeekOrigin.Begin);
    this.ZipFileStream.Read(signature, 0, 4);
    if (BitConverter.ToInt32(signature, 0) != 0x04034b50)
        return false;
    // Select input stream for inflating or just reading
    Stream inStream;
    if (_zfe.Method == Compression.Store)
        inStream = this.ZipFileStream;
    else if (_zfe.Method == Compression.Deflate)
        inStream = new DeflateStream(this.ZipFileStream, CompressionMode.Decompress, true);
    else
        return false;
    // Buffered copy
    byte[] buffer = new byte[16384];
    this.ZipFileStream.Seek(_zfe.FileOffset, SeekOrigin.Begin);
    uint bytesPending = _zfe.FileSize;
    while (bytesPending > 0)
    {
        int bytesRead = inStream.Read(buffer, 0, (int)Math.Min(bytesPending, buffer.Length));
        _stream.Write(buffer, 0, bytesRead);
        bytesPending -= (uint)bytesRead;
    }
    _stream.Flush();
    if (_zfe.Method == Compression.Deflate)
        inStream.Dispose();
    return true;
}
// Removes one of many files in storage . It creates a new Zip file .
public static bool RemoveEntries(ref ZipStorer _zip, List<ZipFileEntry> _zifes)
{
    if (!(_zip.ZipFileStream is FileStream))
        throw new InvalidOperationException("RemoveEntries is allowed just over streams of type FileStream");
    //Get full list of entries
    List<ZipFileEntry> fullList = _zip.ReadCentralDir();
    //In order to delete we need to create a copy of the zip file excluding the selected items
    string tempZipName = Path.GetTempFileName();
    string tempEntryName = Path.GetTempFileName();
    try
    {
        ZipStorer tempZip = ZipStorer.Create(tempZipName, string.Empty);
        foreach (ZipFileEntry zfe in fullList)
        {
            if (!_zifes.Contains(zfe))
            {
                if (_zip.ExtractFile(zfe, tempEntryName))
                {
                    tempZip.AddFile(zfe.Method, tempEntryName, zfe.FilenameInZip, zfe.Comment);
                }
            }
        }
        _zip.Close();
        tempZip.Close();
    }
}

```

```

        File.Delete(_zip.FileName);
        File.Move(tempZipName, _zip.FileName);
        _zip = ZipStorer.Open(_zip.FileName, _zip.Access);
    }
    catch
    {
        return false;
    }
    finally
    {
        if (File.Exists(tempZipName))
            File.Delete(tempZipName);
        if (File.Exists(tempEntryName))
            File.Delete(tempEntryName);
    }
    return true;
}
#endregion

#region Private methods
// Calculate the file offset by reading the corresponding local header
private uint GetFileOffset(uint _headerOffset)
{
    byte[] buffer = new byte[2];
    this.ZipFileStream.Seek(_headerOffset + 26, SeekOrigin.Begin);
    this.ZipFileStream.Read(buffer, 0, 2);
    ushort filenameSize = BitConverter.ToInt16(buffer, 0);
    this.ZipFileStream.Read(buffer, 0, 2);
    ushort extraSize = BitConverter.ToInt16(buffer, 0);
    return (uint)(30 + filenameSize + extraSize + _headerOffset);
}
/* Local file header :
local file header signature 4 bytes (0x04034b50)
version needed to extract 2 bytes
general purpose bit flag 2 bytes
compression method 2 bytes
last mod file time 2 bytes
last mod file date 2 bytes
crc-32 4 bytes
compressed size 4 bytes
uncompressed size 4 bytes
filename length 2 bytes
extra field length 2 bytes
filename (variable size)
extra field (variable size)
*/
private void WriteLocalHeader(ref ZipFileEntry _zfe)
{
    long pos = this.ZipFileStream.Position;
    Encoding encoder = _zfe.EncodeUTF8 ? Encoding.UTF8 : DefaultEncoding;
    byte[] encodedFilename = encoder.GetBytes(_zfe.FilenameInZip);
    this.ZipFileStream.Write(new byte[] { 80, 75, 3, 4, 20, 0 }, 0, 6); // No extra header
    this.ZipFileStream.Write(BitConverter.GetBytes((ushort)(_zfe.EncodeUTF8 ? 0x0800 : 0)), 0, 2); // filename and comment encoding
    this.ZipFileStream.Write(BitConverter.GetBytes(_zfe.Method), 0, 2); // zipping method
    this.ZipFileStream.Write(BitConverter.GetBytes(DateTimeToDosTime(_zfe.ModifyTime)), 0, 4);
    // zipping date and time
    this.ZipFileStream.Write(new byte[] { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }, 0, 12); // unused CRC ,
    un/compressed           size           ,           updated           later
}

```

```

this.ZipFileStream.Write(BitConverter.GetBytes((ushort)encodedFilename.Length), 0, 2); // filename
length
    this.ZipFileStream.Write(BitConverter.GetBytes((ushort)0), 0, 2); // extra length
    this.ZipFileStream.Write(encodedFilename, 0, encodedFilename.Length);
    _zfe.HeaderSize = (uint)(this.ZipFileStream.Position - pos);
}
/* Central directory's File header :
   central file header signature 4 bytes (0x02014b50)
   version made by      2 bytes
   version needed to extract 2 bytes
   general purpose bit flag 2 bytes
   compression method 2 bytes
   last mod file time 2 bytes
   last mod file date 2 bytes
   crc-32      4 bytes
   compressed size 4 bytes
   uncompressed size 4 bytes
   filename length 2 bytes
   extra field length 2 bytes
   file comment length 2 bytes
   disk number start 2 bytes
   internal file attributes 2 bytes
   external file attributes 4 bytes
   relative offset of local header 4 bytes
   filename (variable size)
   extra field (variable size)
   file comment (variable size)
*/
private void WriteCentralDirRecord(ZipFileEntry _zfe)
{
    Encoding encoder = _zfe.EncodeUTF8 ? Encoding.UTF8 : DefaultEncoding;
    byte[] encodedFilename = encoder.GetBytes(_zfe.FilenameInZip);
    byte[] encodedComment = encoder.GetBytes(_zfe.Comment);
    this.ZipFileStream.Write(new byte[] { 80, 75, 1, 2, 23, 0xB, 20, 0 }, 0, 8);
    this.ZipFileStream.Write(BitConverter.GetBytes((ushort)(_zfe.EncodeUTF8 ? 0x0800 : 0)), 0, 2); // filename and comment encoding
    this.ZipFileStream.Write(BitConverter.GetBytes((ushort)_zfe.Method), 0, 2); // zipping method
    this.ZipFileStream.Write(BitConverter.GetBytes(DateTimeToDosTime(_zfe.ModifyTime)), 0, 4); // zipping date and time
    this.ZipFileStream.Write(BitConverter.GetBytes(_zfe.Crc32), 0, 4); // file CRC
    this.ZipFileStream.Write(BitConverter.GetBytes(_zfe.CompressedSize), 0, 4); // compressed file size
    this.ZipFileStream.Write(BitConverter.GetBytes(_zfe.FileSize), 0, 4); // uncompressed file size
    this.ZipFileStream.Write(BitConverter.GetBytes((ushort)encodedFilename.Length), 0, 2); // Filename in zip
    this.ZipFileStream.Write(BitConverter.GetBytes((ushort)0), 0, 2); // extra length
    this.ZipFileStream.Write(BitConverter.GetBytes((ushort)encodedComment.Length), 0, 2);
    this.ZipFileStream.Write(BitConverter.GetBytes((ushort)0), 0, 2); // disk=0
    this.ZipFileStream.Write(BitConverter.GetBytes((ushort)0), 0, 2); // file type: binary
    this.ZipFileStream.Write(BitConverter.GetBytes((ushort)0), 0, 2); // Internal file attributes
    this.ZipFileStream.Write(BitConverter.GetBytes((ushort)0x8100), 0, 2); // External file attributes
(normal/readable)      this.ZipFileStream.Write(BitConverter.GetBytes(_zfe.HeaderOffset), 0, 4); // Offset of header
    this.ZipFileStream.Write(encodedFilename, 0, encodedFilename.Length);
    this.ZipFileStream.Write(encodedComment, 0, encodedComment.Length);
}
/* End of central dir record:
   end of central dir signature 4 bytes (0x06054b50)
   number of this disk      2 bytes
   number of the disk with the
   start of the central directory 2 bytes

```

```

total number of entries in
the central dir on this disk 2 bytes
total number of entries in
the central dir 2 bytes
size of the central directory 4 bytes
offset of start of central
directory with respect to
the starting disk number 4 bytes
zipfile comment length 2 bytes
zipfile comment (variable size)
*/
private void WriteEndRecord(uint _size, uint _offset)
{
    Encoding encoder = this.EncodeUTF8 ? Encoding.UTF8 : DefaultEncoding;
    byte[] encodedComment = encoder.GetBytes(this.Comment);

    this.ZipFileStream.Write(new byte[] { 80, 75, 5, 6, 0, 0, 0, 0 }, 0, 8);
    this.ZipFileStream.Write(BitConverter.GetBytes((ushort)Files.Count+ExistingFiles), 0, 2);
    this.ZipFileStream.Write(BitConverter.GetBytes((ushort)Files.Count+ExistingFiles), 0, 2);
    this.ZipFileStream.Write(BitConverter.GetBytes(_size), 0, 4);
    this.ZipFileStream.Write(BitConverter.GetBytes(_offset), 0, 4);
    this.ZipFileStream.Write(BitConverter.GetBytes((ushort)encodedComment.Length), 0, 2);
    this.ZipFileStream.Write(encodedComment, 0, encodedComment.Length);
}
// Copies all source file into storage file
private void Store(ref ZipFileEntry _zfe, Stream _source)
{
    byte[] buffer = new byte[16384];
    int bytesRead;
    uint totalRead = 0;
    Stream outStream;
    long posStart = this.ZipFileStream.Position;
    long sourceStart = _source.Position;

    if (_zfe.Method == Compression.Store)
        outStream = this.ZipFileStream;
    else
        outStream = new DeflateStream(this.ZipFileStream, CompressionMode.Compress, true);
    _zfe.Crc32 = 0 ^ 0xffffffff;
    do
    {
        bytesRead = _source.Read(buffer, 0, buffer.Length);
        totalRead += (uint)bytesRead;
        if (bytesRead > 0)
        {
            outStream.Write(buffer, 0, bytesRead);
            for (uint i = 0; i < bytesRead; i++)
            {
                _zfe.Crc32 = ZipStorer.CrcTable[( _zfe.Crc32 ^ buffer[i]) & 0xFF] ^ (_zfe.Crc32 >> 8);
            }
        }
    } while (bytesRead == buffer.Length);
    outStream.Flush();
    if (_zfe.Method == Compression.Deflate)
        outStream.Dispose();
    _zfe.Crc32 ^= 0xffffffff;
    _zfe.FileSize = totalRead;
    _zfe.CompressedSize = (uint)(this.ZipFileStream.Position - posStart);
    // Verify for real compression
}

```

```

        if (_zfe.Method == Compression.Deflate && !this.ForceDeflating && _source.CanSeek &&
_zfe.CompressedSize > _zfe.FileSize)
    {
        // Start operation again with Store algorithm
        _zfe.Method = Compression.Store;
        this.ZipFileStream.Position = posStart;
        this.ZipFileStream.SetLength(posStart);
        _source.Position = sourceStart;
        this.Store(ref _zfe, _source);
    }
}

/* DOS Date and time :
MS-DOS date . The date is a packed value with the following format. Bits Description
0-4 Day of the month (1-31)
5-8 Month (1 = January, 2 = February, and so on)
9-15 Year offset from 1980 (add 1980 to get actual year)
MS-DOS time . The time is a packed value with the following format . Bits Description
0-4 Second divided by 2
5-10 Minute (0-59)
11-15 Hour (0-23 on a 24-hour clock)
*/
private uint DateTimeToDosTime(DateTime _dt)
{
    return (uint)(
        (_dt.Second / 2) | (_dt.Minute << 5) | (_dt.Hour << 11) |
        (_dt.Day << 16) | (_dt.Month << 21) | ((_dt.Year - 1980) << 25));
}

private DateTime DosTimeToDateTime(uint _dt)
{
    return new DateTime(
        (int)(_dt >> 25) + 1980,
        (int)(_dt >> 21) & 15,
        (int)(_dt >> 16) & 31,
        (int)(_dt >> 11) & 31,
        (int)(_dt >> 5) & 63,
        (int)(_dt & 31) * 2);
}

/* CRC32 algorithm
The 'magic number' for the CRC is 0xdebb20e3.
The proper CRC pre and post conditioning
is used, meaning that the CRC register is
pre-conditioned with all ones (a starting value
of 0xffffffff) and the value is post-conditioned by
taking the one's complement of the CRC residual .
If bit 3 of the general purpose flag is set , this
field is set to zero in the local header and the correct
value is put in the data descriptor and in the central
directory .
*/
private void UpdateCrcAndSizes(ref ZipFileEntry _zfe)
{
    long lastPos = this.ZipFileStream.Position; // remember position n
    this.ZipFileStream.Position = _zfe.HeaderOffset + 8;
    this.ZipFileStream.Write(BitConverter.GetBytes((ushort)_zfe.Method), 0, 2); // zipping method
    this.ZipFileStream.Position = _zfe.HeaderOffset + 14;
    this.ZipFileStream.Write(BitConverter.GetBytes(_zfe.Crc32), 0, 4); // Update CRC
    this.ZipFileStream.Write(BitConverter.GetBytes(_zfe.CompressedSize), 0, 4); // Compressed size
    this.ZipFileStream.Write(BitConverter.GetBytes(_zfe.FileSize), 0, 4); // Uncompressed size
    this.ZipFileStream.Position = lastPos; // restore position
}

```

```

// Replaces backslashes with slashes to store in zip header
private string NormalizedFilename(string _filename)
{
    string filename = _filename.Replace("\\", "/");
    int pos = filename.IndexOf(':');
    if (pos >= 0)
        filename = filename.Remove(0, pos + 1);
    return filename.Trim('/');
}
// Reads the end-of-central-directory record
private bool ReadFileInfo()
{
    if (this.ZipFileStream.Length < 22)
        return false;
    try
    {
        this.ZipFileStream.Seek(-17, SeekOrigin.End);
        BinaryReader br = new BinaryReader(this.ZipFileStream);
        do
        {
            this.ZipFileStream.Seek(-5, SeekOrigin.Current);
            UInt32 sig = br.ReadUInt32();
            if (sig == 0x06054b50)
            {
                this.ZipFileStream.Seek(6, SeekOrigin.Current);
                UInt16 entries = br.ReadUInt16();
                Int32 centralSize = br.ReadInt32();
                UInt32 centralDirOffset = br.ReadUInt32();
                UInt16 commentSize = br.ReadUInt16();
                // check if comment field is the very last data in file
                if (this.ZipFileStream.Position + commentSize != this.ZipFileStream.Length)
                    return false;
                // Copy entire central directory to a memory buffer
                this.ExistingFiles = entries;
                this.CentralDirImage = new byte[centralSize];
                this.ZipFileStream.Seek(centralDirOffset, SeekOrigin.Begin);
                this.ZipFileStream.Read(this.CentralDirImage, 0, centralSize);
                // Leave the pointer at the begining of central dir, to append new files
                this.ZipFileStream.Seek(centralDirOffset, SeekOrigin.Begin);
                return true;
            }
        } while (this.ZipFileStream.Position > 0);
    }
    catch { }
    return false;
}
#endif
#region IDisposable Members
// Closes the Zip file stream
public void Dispose()
{
    this.Close();
}
#endregion
}
}

```

Κόδικας newExampleForm για την φόρμα της εισαγωγής νέου παραδείγματος

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Graphics
{
    public partial class newExampleForm : Form
    {
        public string exampleName;
        public newExampleForm()
        {
            InitializeComponent();
        }
        private void createBtn_Click(object sender, EventArgs e)
        {
            exampleName = nameTxtBox.Text;
            Close();
        }
        private void nameTxtBox_TextChanged(object sender, EventArgs e)
        {
            if (nameTxtBox.Text != "")
            {
                createBtn.Enabled = true;
            }
            else
            {
                createBtn.Enabled = false;
            }
        }
        private void cancelBtn_Click(object sender, EventArgs e)
        {
            Close();
        }
    }
}
```

Κόδικας της κύριας φόρμας της εφαρμογής

```
using System;
using System.Collections.Generic; // Required to content the central directory
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;
using System.Threading;
using System.Threading.Tasks;
using System.IO;
using System.IO.Compression; // Required to use ZipStorer class
using System.Resources;
```

```

using System.Reflection;
using System.Text.RegularExpressions;

namespace Graphics
{
    public partial class mainForm : Form
    {
        public String code;
        public string exampleName;
        public string examplePath;
        public string projPath;
        public string exePath;
        public ToolStripMenuItem newProject;
        private bool textChangedEventFlag;
        // Define static variables shared by class methods.
        public static StringBuilder prOutput = null;
        private static int numOutputLines = 0;
        public mainForm()
        {
            InitializeComponent();
        }
        private void mainForm_Load(object sender, EventArgs e)
        {
            addCustomExamples();
        }
        /*
         * Η συνάρτηση addCustomExamples διαβάζει τα ονόματα των παραδειγμάτων που έχουν
         δημιουργηθεί στο φάκελο Custom
         * και δημιουργεί το αντίστοιχο μενού στα "Παραδείγματα"
        */
        private void addCustomExamples(){
            if(Directory.Exists(@"C:\examples\Custom"))
            {
                string[] subdirectoryEntries = Directory.GetDirectories(@"C:\examples\Custom");
                string projectName;
                if(subdirectoryEntries != null)
                {
                    foreach(string subdirectory in subdirectoryEntries)
                    {
                        projectName = subdirectory.Substring(subdirectory.LastIndexOf("\\") + 1);
                        newProject = new ToolStripMenuItem();
                        examplesCustomMenuItem.DropDownItems.Add(newProject);
                        newProject.Text = projectName;
                        newProject.Name = projectName + "MenuItem";
                        newProject.Click += new EventHandler(newProject_Click);
                    }
                }
            }
        }
        private static void prOutputHandler(object sendingProcess, DataReceivedEventArgs outLine)
        {
            // Collect the sort command output .
            if (!String.IsNullOrEmpty(outLine.Data))
            {
                numOutputLines++;
                // Add the text to the collected output .
                prOutput.Append(Environment.NewLine + "[" + numOutputLines.ToString() + "] - " +
outLine.Data);
            }
        }
    }
}

```

```

private void fileSaveMenuItem_Click(object sender, EventArgs e)
{
    saveFileDialog.FileName = examplePath;
    FileManager fm = new FileManager(saveFileDialog.FileName);
    fm.writeFile(textBox1.Text);
    statusLabel.Text = "Το αρχείο αποθηκεύτηκε .";
    Thread.Sleep(3000);
    statusLabel.Text = "";
}
private void simplePointMenuItem_Click(object sender, EventArgs e)
{
    textChangedEventFlag = false;
    exampleName = "simplePoint";
    examplePath = "C:\\examples\\\" + exampleName + "\\\" + exampleName + "\\\" + exampleName + ".cpp";
    projPath = "C:\\examples\\\" + exampleName + "\\\" + exampleName + "\\\" + exampleName + "\\\" + exampleName + ".vcxproj";
    //καλώ την κλάση FileManager για να διαβάσω το αρχείο που επέλεξα
    FileManager fm = new FileManager(examplePath);
    code = fm.readFile();
    //φορτώνω τα περιεχόμενα του αρχείου στο codeTextBox μέσω της συνάρτησης Parse
    statusLabel.Text = "Το παράδειγμα \"Simple Point\" δημιουργεί ένα σημείο";
    Parse();
    textChangedEventFlag = true;
}
private void fileExitMenuItem_Click(object sender, EventArgs e)
{
    Close();
}
private void simpleLineToolStripMenuItem_Click(object sender, EventArgs e)
{
    textChangedEventFlag = false;
    exampleName = "simpleLine";
    examplePath = "C:\\examples\\\" + exampleName + "\\\" + exampleName + "\\\" + exampleName + ".cpp";
    projPath = "C:\\examples\\\" + exampleName + "\\\" + exampleName + "\\\" + exampleName + "\\\" + exampleName + ".vcxproj";
    //καλώ την κλάση FileManager για να διαβάσω το αρχείο που επέλεξα
    FileManager fm = new FileManager(examplePath);
    code = fm.readFile();
    //φορτώνω τα περιεχόμενα του αρχείου στο codeTextBox
    statusLabel.Text = "Το παράδειγμα \"Simple Line\" δημιουργεί μια γραμμή, η οποία δημιουργείται
    από δύο σημεία";
    Parse();
    textChangedEventFlag = true;
}
private void lineStripToolStripMenuItem_Click(object sender, EventArgs e)
{
    textChangedEventFlag = false;
    exampleName = "lineStrip";
    examplePath = "C:\\examples\\\" + exampleName + "\\\" + exampleName + "\\\" + exampleName + ".cpp";
    projPath = "C:\\examples\\\" + exampleName + "\\\" + exampleName + "\\\" + exampleName + "\\\" + exampleName + ".vcxproj";
    exePath = "C:\\examples\\\" + exampleName + "\\\"Debug\\\" + exampleName + ".exe";
    //καλώ την κλάση FileManager για να διαβάσω το αρχείο που επέλεξα
    FileManager fm = new FileManager(examplePath);
    code = fm.readFile();
    //φορτώνω τα περιεχόμενα του αρχείου στο codeTextBox
    statusLabel.Text = "Το παράδειγμα \"Line Strip\" δημιουργεί μια τεθλασμένη ανοιχτή γραμμή, η
    οποία δημιουργείται από την ακολουθία πολλών σημείων";
}

```

```

Parse();
textChangedEventFlag = true;
}
private void lineLoopToolStripMenuItem_Click(object sender, EventArgs e)
{
    textChangedEventFlag = false;
exampleName = "lineLoop";
examplePath = "C:\\examples\\\" + exampleName + "\\\" + exampleName + "\\\" + exampleName +
".cpp";
projPath = "C:\\examples\\\" + exampleName + "\\\" + exampleName + "\\\" + exampleName +
".vcxproj";
exePath = "C:\\examples\\\" + exampleName + "\\Debug\\\" + exampleName + ".exe";
//καλώ την κλάση FileManager για να διαβάσω το αρχείο που επέλεξα
FileManager fm = new FileManager(examplePath);
code = fm.readFile();
//φορτώνω τα περιεχόμενα του αρχείου στο codeTxtBox
statusLbl.Text = "Το παράδειγμα \"Line Loop\" δημιουργεί μια τεθλασμένη κλειστή γραμμή , η
οποία δημιουργείται από την ακολουθία πολλών σημείων";
Parse();
textChangedEventFlag = true;
}
private void singleTriangleToolStripMenuItem_Click(object sender, EventArgs e)
{
    textChangedEventFlag = false;
exampleName = "simpleTriangle";
examplePath = "C:\\examples\\\" + exampleName + "\\\" + exampleName + "\\\" + exampleName +
".cpp";
projPath = "C:\\examples\\\" + exampleName + "\\\" + exampleName + "\\\" + exampleName +
".vcxproj";
exePath = "C:\\examples\\\" + exampleName + "\\Debug\\\" + exampleName + ".exe";

//καλώ την κλάση FileManager για να διαβάσω το αρχείο που επέλεξα
FileManager fm = new FileManager(examplePath);
code = fm.readFile();
//φορτώνω τα περιεχόμενα του αρχείου στο codeTxtBox
statusLbl.Text = "Το παράδειγμα \"Simple Triangle\" δημιουργεί ένα τρίγωνο";
Parse();
textChangedEventFlag = true;
}
private void cubeToolStripMenuItem_Click(object sender, EventArgs e)
{
    textChangedEventFlag = false;
exampleName = "cube";
examplePath = "C:\\examples\\\" + exampleName + "\\\" + exampleName + "\\\" + exampleName +
".cpp";
projPath = "C:\\examples\\\" + exampleName + "\\\" + exampleName + "\\\" + exampleName +
".vcxproj";
exePath = "C:\\examples\\\" + exampleName + "\\Debug\\\" + exampleName + ".exe";
//καλώ την κλάση FileManager για να διαβάσω το αρχείο που επέλεξα
FileManager fm = new FileManager(examplePath);
code = fm.readFile();
//φορτώνω τα περιεχόμενα του αρχείου στο codeTxtBox
statusLbl.Text = "Το παράδειγμα \"Cube\" δημιουργεί έναν πολύχρωμο κύβο";
Parse();
textChangedEventFlag = true;
}
private void cylindersToolStripMenuItem_Click(object sender, EventArgs e)
{
    textChangedEventFlag = false;
exampleName = "cylinders";

```

```

examplePath = "C:\\examples\\\" + exampleName + "\\\" + exampleName + "\\\" + exampleName +
".cpp";
projPath = "C:\\examples\\\" + exampleName + "\\\" + exampleName + "\\\" + exampleName +
".vcxproj";
exePath = "C:\\examples\\\" + exampleName + "\\Debug\\\" + exampleName + ".exe";
//καλώ την κλάση FileManager για να διαβάσω το αρχείο που επέλεξα
FileManager fm = new FileManager(examplePath);
code = fm.readFile();
//φορτώνω τα περιεχόμενα του αρχείου στο codeTextBox
statusLbl.Text = "Το παράδειγμα \"Cylinders\" δημιουργεί δύο διαφορετικούς κυλίνδρους.
Πατώντας το πλήκτρο \"w\" εναλλάσσονται από πλέγμα σε συμπαγείς";
Parse();
textChangedEventFlag = true;
}
private void exeToolBarBtn_Click(object sender, EventArgs e)
{
    regularMsg.Clear();
    try
    {
        OutputRedirect.InputText(projPath, exePath);
        statusLbl.Text = "Εκτέλεση εφαρμογής";
        Thread.Sleep(3000);
        statusLbl.Text = "";
    }
    catch (InvalidOperationException er)
    {
        regularMsg.Text += "Exception:";
        regularMsg.Text += er.ToString();
        statusLbl.Text = "";
    }
}
public void newProject_Click(object sender, EventArgs e)
{
    textChangedEventFlag = false;
    //Άνοιγμα παραδείγματος
    exampleName = "newOpenGL";
    examplePath = "C:\\examples\\Custom\\\" + sender.ToString() + "\\\" + exampleName + "\\\" +
exampleName + ".cpp";
    projPath = "C:\\examples\\Custom\\\" + sender.ToString() + "\\\" + exampleName + "\\\" +
exampleName + ".vcxproj";
    exePath = "C:\\examples\\Custom\\\" + sender.ToString() + "\\\" + exampleName + "\\Debug\\\" +
exampleName + ".exe";
    //καλώ την κλάση FileManager για να διαβάσω το αρχείο που επέλεξα
    FileManager fm = new FileManager(examplePath);
    code = fm.readFile();
    //φορτώνω τα περιεχόμενα του αρχείου στο codeTextBox
    Parse();
    textChangedEventFlag = true;
}
private void importToolBarBtn_Click(object sender, EventArgs e)
{
    newExampleForm newForm = new newExampleForm();
    newForm.ShowDialog();
    string projectName = newForm.exampleName;
    showProgress(5000);
    if (projectName != "")
    {
        try
        {
            Assembly assembly;

```

```

Stream zipStream;
assembly = Assembly.GetExecutingAssembly();
zipStream = assembly.GetManifestResourceStream("Graphics.Resources.newOpenGL.zip");
string newPath = @"C:\examples\Custom\" + projectName;
extract(zipStream, newPath);
statusLbl.Text = "Δημιουργία νέου παραδείγματος...";
}
catch
{
    statusLbl.Text = "";
    MessageBox.Show("Σφάλμα : Το αρχείο δεν μπόρεσε να δημιουργηθεί.", "Δημιουργία νέου παραδείγματος", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
//Προσθήκη στο μενού
newProject = new ToolStripMenuItem();
examplesCustomMenuItem.DropDownItems.Add(newProject);
newProject.Text = projectName;
newProject.Name = projectName + "MenuItem";
newProject.Click += new EventHandler(newProject_Click);
//Άνοιγμα παραδείγματος
exampleName = "newOpenGL";
examplePath = "C:\examples\Custom\" + projectName + "\\\" + exampleName + "\\\" +
exampleName + ".cpp";
//καλώ την κλάση FileManager για να διαβάσω το αρχείο που επέλεξα
FileManager fm = new FileManager(examplePath);
code = fm.readFile();
//φορτώνω τα περιεχόμενα του αρχείου στο codeTextBox
textChangedEventFlag = false;
codeTextBox.Clear();
Parse();
textChangedEventFlag = true;
}
}
void Parse()
{
    codeTextBox.Clear();
    // Fforeach line in input,
    // identify key words and format them when adding to the rich text box.
    Regex r = new Regex("\n");
    String[] lines = r.Split(code);
    foreach (string l in lines)
    {
        ParseLine(l);
    }
}
void ParseLine(string line)
{
    Regex r = new Regex("[ \t{}();]");
    String[] tokens = r.Split(line);
    foreach (string token in tokens)
    {
        // Set the token's default color and font .
        codeTextBox.SelectionColor = Color.Black;
        codeTextBox.SelectionFont = new Font("Courier New", 10, FontStyle.Regular);
        // Check for a comment.
        if (token == "//" || token.StartsWith("//"))
        {
            // Find the start of the comment and then extract the whole comment .
            int index = line.IndexOf("//");
            string comment;

```

```

        comment = line.Substring(index, line.Length - index);
        codeTxtBox.SelectionColor = Color.Green;
        codeTxtBox.SelectionFont = new Font("Courier New", 10, FontStyle.Regular);
        codeTxtBox.SelectedText = comment;
        break;
    }
    // Check whether the token is a keyword .
    String[] keywords = {

"abstract", "as", "base", "bool", "break", "byte", "case", "catch", "char", "checked", "class", "const", "continue", "decimal",

"default", "delegate", "do", "double", "else", "enum", "event", "explicit", "extern", "false", "finally", "fixed", "float",

"for", "foreach", "goto", "if", "implicit", "in", "int", "namespace", "new", "null", "object", "operator", "out", "private",

"protected", "public", "return", "short", "sizeof", "stackalloc", "static", "string", "struct", "switch", "this", "throw",

"true", "try", "typeof", "uint", "ulong", "unchecked", "unsafe", "ushort", "using", "virtual", "void", "volatile", "while"

};

        for (int i = 0; i < keywords.Length; i++)
        {
            if (keywords[i] == token)
            {
                //Apply alternative color and font to highlight keyword .
                codeTxtBox.SelectionColor = Color.Blue;
                codeTxtBox.SelectionFont = new Font("Courier New", 10, FontStyle.Bold);
                break;
            }
        }
        codeTxtBox.SelectedText = token;
    }
    codeTxtBox.SelectedText = "\n";
}
private void TextChangedEvent(object sender, EventArgs e)
{
    if (textChangedEventFlag)
    {
        // Calculate the starting position of the current line.
        int start = 0, end = 0;
        for (start = codeTxtBox.SelectionStart - 1; start > 0; start--)
        {
            if (codeTxtBox.Text[start] == '\n') { start++; break; }
        }
        // Calculate the end position of the current line .
        for (end = codeTxtBox.SelectionStart; end < codeTxtBox.Text.Length; end++)
        {
            if (codeTxtBox.Text[end] == '\n') break;
        }
        // Extract the current line that is being edited .
        String line;
        if ((start != end) && (start != -1))
        {
            line = codeTxtBox.Text.Substring(start, end - start);
        }
    }
}

```

```

else
{
    line = "";
}
// Backup the users current selection point .
int selectionStart = codeTxtBox.SelectionStart;
int selectionLength = codeTxtBox.SelectionLength;
// Split the line into tokens .
Regex r = new Regex("([ \t{}();])");
string[] tokens = r.Split(line);
int index = start;
foreach (string token in tokens)
{
    // Set the token's default color and font .
    codeTxtBox.SelectionStart = index;
    codeTxtBox.SelectionLength = token.Length;
    codeTxtBox.SelectionColor = Color.Black;
    codeTxtBox.SelectionFont = new Font("Courier New", 10, FontStyle.Regular);
    // Check for a comment .
    if (token == "//" || token.StartsWith("//"))
    {
        // Find the start of the comment and then extract the whole comment .
        int length = line.Length - (index - start);
        string commentText = codeTxtBox.Text.Substring(index, length);
        codeTxtBox.SelectionStart = index;
        codeTxtBox.SelectionLength = length;
        codeTxtBox.SelectionColor = Color.LightGreen;
        codeTxtBox.SelectionFont = new Font("Courier New", 10, FontStyle.Regular);
        break;
    }
    // Check whether the token is a keyword .
    String[] keywords = {

"abstract", "as", "base", "bool", "break", "byte", "case", "catch", "char", "checked", "class", "const", "continue", "decimal",

"default", "delegate", "do", "double", "else", "enum", "event", "explicit", "extern", "false", "finally", "fixed", "float",

"for", "foreach", "goto", "if", "implicit", "in", "int", "namespace", "new", "null", "object", "operator", "out", "private",

"protected", "public", "return", "short", "sizeof", "stackalloc", "static", "string", "struct", "switch", "this", "throw",

"true", "try", "typeof", "uint", "ulong", "unchecked", "unsafe", "ushort", "using", "virtual", "void", "volatile", "while"

};

    for (int i = 0; i < keywords.Length; i++)
    {
        if (keywords[i] == token)
        {
            // Apply alternative color and font to highlight keyword .
            codeTxtBox.SelectionColor = Color.Blue;
            codeTxtBox.SelectionFont = new Font("Courier New", 10, FontStyle.Bold);
            break;
        }
    }
    index += token.Length;
}

```

```

// Restore the users current selection point .
codeTxtBox.SelectionStart = selectionStart;
codeTxtBox.SelectionLength = selectionLength;
}
}

private void showProgress(int ms) {
progressBar.Visible = true;
progressBar.Maximum = ms;
for (int i = 0; i < ms; i++)
{
    progressBar.PerformStep();
}
progressBar.Visible = false;
}

private void extract(Stream zipStream, string dest)
{
// Opens existing zip file
ZipStorer zip = ZipStorer.Open(zipStream, FileAccess.Read);
// Read all directory contents
List<ZipStorer.ZipFileEntry> dir = zip.ReadCentralDir();
// Extract all files in target directory
string path;
bool result;
result = false;
foreach (ZipStorer.ZipFileEntry entry in dir)
{
    path = Path.Combine(dest, entry.FilenameInZip);
    result = zip.ExtractFile(entry, path);
    if (entry.IsDirectory())
    {
        Directory.CreateDirectory(path.ToString());
    }
    else
    {
        result = zip.ExtractFile(entry, path);
    }
}
zip.Close();
statusLbl.Text = "";
}

private void extractExamplesToolBarBtn_Click(object sender, EventArgs e)
{
try
{
    Assembly assembly;
    Stream zipStream;
    assembly = Assembly.GetExecutingAssembly();
    zipStream = assembly.GetManifestResourceStream("Graphics.Resources.examples.zip");
    extract(zipStream, @"C:\examples");
    statusLbl.Text = "Περιμένετε όσο διαρκεί η αποσυμπίεση των αρχείων...";
    showProgress(5000);
    MessageBox.Show("Η εξαγωγή των παραδειγμάτων ολοκληρώθηκε επιτυχώς.", "Εκμάθηση OpenGL", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
catch (InvalidDataException)
{
    statusLbl.Text = "";
    MessageBox.Show("Σφάλμα: Το αρχείο zip δεν υποστηρίζεται ή περιέχει σφάλματα.", "Εκμάθηση OpenGL", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
}

```

```

        }
        catch(Exception ex)
        {
            statusLbl.Text = "";
            //MessageBox.Show("Δημιουργήθηκε άγνωστο σφάλμα κατά τη διάρκεια προσπέλασης του αρχείου.", "Εκμάθηση OpenGL", MessageBoxButtons.OK, MessageBoxIcon.Error);
            MessageBox.Show(ex.Message, "Εκμάθηση OpenGL", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

    private void codeTxtBox_KeyPress(object sender, KeyPressEventArgs e)
    {
        statusLbl.Text = "Το αρχείο δεν έχει αποθηκευτεί";
    }

    private void button1_Click(object sender, EventArgs e)
    {
        addCustomExamples();
    }

    private void coneToolStripMenuItem_Click(object sender, EventArgs e)
    {
        textChangedEventFlag = false;
        //Άνοιγμα παραδείγματος
        exampleName = "newOpenGL";
        examplePath = "C:\\examples\\\" + sender.ToString() + "\\\" + exampleName + "\\\" +
exampleName + ".cpp";
        projPath = "C:\\examples\\\" + sender.ToString() + "\\\" + exampleName + "\\\" + exampleName +
".vcxproj";
        exePath = "C:\\examples\\\" + sender.ToString() + "\\\" + exampleName + "\\Debug\\\" +
exampleName + ".exe";
        //καλώ την κλάση FileManager για να διαβάσω το αρχείο που επέλεξα
        FileManager fm = new FileManager(examplePath);
        code = fm.readFile();
        //φορτώνω τα περιεχόμενα του αρχείου στο codeTxtBox
        Parse();
        textChangedEventFlag = true;
    }

    private void octahedronToolStripMenuItem_Click(object sender, EventArgs e)
    {
        textChangedEventFlag = false;
        //Άνοιγμα παραδείγματος
        exampleName = "newOpenGL";
        examplePath = "C:\\examples\\\" + sender.ToString() + "\\\" + exampleName + "\\\" +
exampleName + ".cpp";
        projPath = "C:\\examples\\\" + sender.ToString() + "\\\" + exampleName + "\\\" + exampleName +
".vcxproj";
        exePath = "C:\\examples\\\" + sender.ToString() + "\\\" + exampleName + "\\Debug\\\" +
exampleName + ".exe";
        //καλώ την κλάση FileManager για να διαβάσω το αρχείο που επέλεξα
        FileManager fm = new FileManager(examplePath);
        code = fm.readFile();
        //φορτώνω τα περιεχόμενα του αρχείου στο codeTxtBox
        Parse();
        textChangedEventFlag = true;
    }

    private void openGLFontToolStripMenuItem_Click(object sender, EventArgs e)
    {
        textChangedEventFlag = false;
        //Άνοιγμα παραδείγματος
        exampleName = "newOpenGL";

```

```

examplePath = "C:\\examples\\\" + sender.ToString() + "\\\" + exampleName + "\\\" +
exampleName + ".cpp";
projPath = "C:\\examples\\\" + sender.ToString() + "\\\" + exampleName + "\\\" + exampleName +
".vcxproj";
exePath = "C:\\examples\\\" + sender.ToString() + "\\\" + exampleName + "\\Debug\\\" +
exampleName + ".exe";
//καλώ την κλάση FileManager για να διαβάσω το αρχείο που επέλεξα
FileManager fm = new FileManager(examplePath);
code = fm.readFile();
//φορτώνω τα περιεχόμενα του αρχείου στο codeTextBox
Parse();
textChangedEventFlag = true;
}
private void rotatingCubeToolStripMenuItem_Click(object sender, EventArgs e)
{
    textChangedEventFlag = false;
    //Άνοιγμα παραδείγματος
    exampleName = "newOpenGL";
    examplePath = "C:\\examples\\\" + sender.ToString() + "\\\" + exampleName + "\\\" +
exampleName + ".cpp";
    projPath = "C:\\examples\\\" + sender.ToString() + "\\\" + exampleName + "\\\" + exampleName +
".vcxproj";
    exePath = "C:\\examples\\\" + sender.ToString() + "\\\" + exampleName + "\\Debug\\\" +
exampleName + ".exe";
    //καλώ την κλάση FileManager για να διαβάσω το αρχείο που επέλεξα
    FileManager fm = new FileManager(examplePath);
    code = fm.readFile();
    //φορτώνω τα περιεχόμενα του αρχείου στο codeTextBox
    Parse();
    textChangedEventFlag = true;
}
private void colorChangeToolStripMenuItem_Click(object sender, EventArgs e)
{
    textChangedEventFlag = false;
    //Άνοιγμα παραδείγματος
    exampleName = "newOpenGL";
    examplePath = "C:\\examples\\\" + sender.ToString() + "\\\" + exampleName + "\\\" +
exampleName + ".cpp";
    projPath = "C:\\examples\\\" + sender.ToString() + "\\\" + exampleName + "\\\" + exampleName +
".vcxproj";
    exePath = "C:\\examples\\\" + sender.ToString() + "\\\" + exampleName + "\\Debug\\\" +
exampleName + ".exe";
    //καλώ την κλάση FileManager για να διαβάσω το αρχείο που επέλεξα
    FileManager fm = new FileManager(examplePath);
    code = fm.readFile();
    //φορτώνω τα περιεχόμενα του αρχείου στο codeTextBox
    Parse();
    textChangedEventFlag = true;
}
private void solidSphereLightToolStripMenuItem_Click(object sender, EventArgs e)
{
    textChangedEventFlag = false;
    //Άνοιγμα παραδείγματος
    exampleName = "newOpenGL";
    examplePath = "C:\\examples\\\" + sender.ToString() + "\\\" + exampleName + "\\\" +
exampleName + ".cpp";
    projPath = "C:\\examples\\\" + sender.ToString() + "\\\" + exampleName + "\\\" + exampleName +
".vcxproj";
    exePath = "C:\\examples\\\" + sender.ToString() + "\\\" + exampleName + "\\Debug\\\" +
exampleName + ".exe";
}

```

```

//καλώ την κλάση FileManager για να διαβάσω το αρχείο που επέλεξα
FileManager fm = new FileManager(examplePath);
statusLbl.Text = "ΣτοIn this example we rotate 3D sphere with light in a circle shape like solar
system.";
code = fm.readFile();
//φορτώνω τα περιεχόμενα του αρχείου στο codeTextBox
Parse();
textChangedEventFlag = true;
}
private void addSpotLightToolStripMenuItem_Click(object sender, EventArgs e)
{
    textChangedEventFlag = false;
//Άνοιγμα παραδείγματος
exampleName = "newOpenGL";
examplePath = "C:\\examples\\" + sender.ToString() + "\\\" + exampleName + "\\\" +
exampleName + ".cpp";
projPath = "C:\\examples\\" + sender.ToString() + "\\\" + exampleName + "\\\" + exampleName +
".vcxproj";
exePath = "C:\\examples\\" + sender.ToString() + "\\\" + exampleName + "\\\" + exampleName + "\\Debug\\\" +
exampleName + ".exe";
//καλώ την κλάση FileManager για να διαβάσω το αρχείο που επέλεξα
FileManager fm = new FileManager(examplePath);
statusLbl.Text = "Πατώντας το πλήκτρο \"L\" ή \"D\" ενεργοποιείται και απενεργοποείται το
φως.";
code = fm.readFile();
//φορτώνω τα περιεχόμενα του αρχείου στο codeTextBox
Parse();
textChangedEventFlag = true;
}
private void cutMenuItem_Click(object sender, EventArgs e)
{
    if (codeTextBox.SelectionLength > 0)
        codeTextBox.Cut();
}
private void copyMenuItem_Click(object sender, EventArgs e)
{
    if (codeTextBox.SelectionLength > 0)
        codeTextBox.Copy();
}
private void pasteMenuItem_Click(object sender, EventArgs e)
{
    if (Clipboard.GetDataObject().GetDataPresent(DataFormats.Text))
    {
        codeTextBox.Paste();
    }
}
private void undoMenuItem_Click(object sender, EventArgs e)
{
    if (codeTextBox.CanUndo)
    {
        codeTextBox.Undo();
        codeTextBox.ClearUndo();
    }
}
private void editBtn_Click(object sender, EventArgs e)
{
    editForm eForm = new editForm();
    eForm.editCodeTextBox.Lines = codeTextBox.Lines;
    eForm.ShowDialog();
    Hide();
}

```

```

        }
        private void mainForm_Shown(object sender, EventArgs e)
        {
            addCustomExamples();
        }
        private void appUseMenuItem_Click(object sender, EventArgs e)
        {
            System.Diagnostics.Process.Start(@"Resources\help.pdf");
        }
        private void examplesHelp_Click(object sender, EventArgs e)
        {
            System.Diagnostics.Process.Start(@"Resources\examples.pdf");
        }
        private void ToolStripMenuItem1_Click(object sender, EventArgs e)
        {
            System.Diagnostics.Process.Start(@"Resources\basic.pdf");
        }
        private void ToolStripMenuItem2_Click(object sender, EventArgs e)
        {
            System.Diagnostics.Process.Start(@"Resources\2.pdf");
        }
        private void ToolStripMenuItem3_Click(object sender, EventArgs e)
        {
            System.Diagnostics.Process.Start(@"Resources\3.pdf");
        }
        private void ToolStripMenuItem4_Click(object sender, EventArgs e)
        {
            System.Diagnostics.Process.Start(@"Resources\4.pdf");
        }
        private void ToolStripMenuItem5_Click(object sender, EventArgs e)
        {
            System.Diagnostics.Process.Start(@"Resources\5.pdf");
        }
        private void ToolStripMenuItem6_Click(object sender, EventArgs e)
        {
            System.Diagnostics.Process.Start(@"Resources\6.pdf");
        }
    }
}

```

Κώδικας φόρμας επεξεργασίας παραδείγματος

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
//using System.Diagnostics;
using System.Threading;
using System.Threading.Tasks;
using System.IO;
using System.IO.Compression; // Required to use ZipStorer class
using System.Resources;
using System.Reflection;
using System.Text.RegularExpressions;

namespace Graphics

```

```

{
    public partial class editForm : Form
    {
        // public string code;
        public string projectName;
        public string exampleName;
        public string examplePath;
        public string projPath;

        public editForm()
        {
            InitializeComponent();
        }

        private void fileExitMenuItem_Click(object sender, EventArgs e)
        {
            Close();
        }

        private void fileBackMenuItem_Click(object sender, EventArgs e)
        {
            editExampleNameForm newForm = new editExampleNameForm();
            mainForm mForm = new mainForm();
            newForm.ShowDialog();
            string projectName = newForm.exampleName;
            if (projectName != "")
            {
                try
                {
                    Assembly assembly;
                    Stream zipStream;
                    assembly = Assembly.GetExecutingAssembly();
                    zipStream = assembly.GetManifestResourceStream("Graphics.Resources.newOpenGL.zip");
                    string newPath = @"C:\examples\Custom\" + projectName;
                    mForm.extract(zipStream, newPath);
                    statusLbl.Text = "Δημιουργία νέου παραδείγματος...";
                }
                catch
                {
                    statusLbl.Text = "";
                    MessageBox.Show("Σφάλμα: Το αρχείο δεν μπόρεσε να δημιουργηθεί.", "Δημιουργία νέου παραδείγματος", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
                //Άνοιγμα παραδείγματος
                exampleName = "newOpenGL";
                examplePath = "C:\examples\Custom\" + projectName + "\\\" + exampleName + "\\\" +
                exampleName + ".cpp";
                FileManager fm = new FileManager(examplePath);
                fm.writeFile(editCodeTextBox.Text);
                mForm.ShowDialog();
            }
            Close();
        }

        private void cutMenuItem_Click(object sender, EventArgs e)
        {
            if (editCodeTextBox.SelectionLength > 0)

                editCodeTextBox.Cut();
        }

        private void copyMenuItem_Click(object sender, EventArgs e)
        {
            if (editCodeTextBox.SelectionLength > 0)

```

```

        editCodeTextBox.Copy();
    }
    private void pasteMenuItem_Click(object sender, EventArgs e)
    {
        if (Clipboard.GetDataObject().GetDataPresent(DataFormats.Text))
        {
            editCodeTextBox.Paste();
        }
    }
    private void undoMenuItem_Click(object sender, EventArgs e)
    {
        if (editCodeTextBox.CanUndo)
        {
            editCodeTextBox.Undo();
            editCodeTextBox.ClearUndo();
        }
    }
    private void appUseMenuItem_Click(object sender, EventArgs e)
    {
        System.Diagnostics.Process.Start(@"help.pdf");
    }
    private void toolStripMenuItem11_Click(object sender, EventArgs e)
    {
        System.Diagnostics.Process.Start(@"Resources\examples.pdf");
    }
    private void toolStripMenuItem13_Click(object sender, EventArgs e)
    {
        System.Diagnostics.Process.Start(@"Resources\basic.pdf");
    }
    private void toolStripMenuItem14_Click(object sender, EventArgs e)
    {
        System.Diagnostics.Process.Start(@"Resources\2.pdf");
    }
    private void toolStripMenuItem15_Click(object sender, EventArgs e)
    {
        System.Diagnostics.Process.Start(@"Resources\3.pdf");
    }
    private void toolStripMenuItem16_Click(object sender, EventArgs e)
    {
        System.Diagnostics.Process.Start(@"Resources\4.pdf");
    }
    private void toolStripMenuItem17_Click(object sender, EventArgs e)
    {
        System.Diagnostics.Process.Start(@"Resources\5.pdf");
    }
    private void toolStripMenuItem18_Click(object sender, EventArgs e)
    {
        System.Diagnostics.Process.Start(@"Resources\6.pdf");
    }
}
}

```

Κώδικας φόρμας αποθήκευσης επεξεργασμένου παραδείγματος

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;

```

```
using System.Text;
using System.Windows.Forms;
using System.Threading;
using System.Threading.Tasks;
using System.IO;
using System.IO.Compression; // Required to use ZipStorer class
using System.Resources;
using System.Reflection;
using System.Text.RegularExpressions;

namespace Graphics
{
    public partial class editExampleNameForm : Form
    {
        public string exampleName;
        public editExampleNameForm()
        {
            InitializeComponent();
        }

        private void NextBtn_Click(object sender, EventArgs e)
        {
            exampleName = nameTxtBox.Text;
            Close();
        }

        private void nameTxtBox_TextChanged(object sender, EventArgs e)
        {
            if (nameTxtBox.Text != "")
            {
                NextBtn.Enabled = true;
            }
            else
            {
                NextBtn.Enabled = false;
            }
        }
    }
}
```


ΒΙΒΛΙΟΓΡΑΦΙΑ

- [1] Γραφικά υπολογιστών (2014). Από τη Βικιπαίδεια, την ελεύθερη εγκυκλοπαίδεια: http://el.wikipedia.org/wiki/%CE%93%CF%81%CE%B1%CF%86%CE%B9%CE%BA%CE%AC_%CF%85%CF%80%CE%BF%CE%BB%CE%BF%CE%B3%CE%B9%CF%83%CF%84%CF%8E%CE%BD
- [2] Μπεκιάρης Γεώργιος (2008). Μηχανές Γραφικών και Γραφικά Υπολογιστών (Πτυχιακή εργασία, Τεχνολογικό Εκπαιδευτικό Ίδρυμα Πειραιά- Τμήμα Ηλεκτρονικών Υπολογιστικών Συστημάτων, (2008): <http://www.mutantstargoat.com/bekos/files/doc/gbekiaris.thesis.bsc.pdf>
- [3] Ράπτης, Πασχάλης. Γραφικά Υπολογιστών: Συσκευές στο ΤΕΙ Θεσσαλονίκης Τμήμα Πληροφορικής: (<http://aetos.it.teithe.gr/~praptis/CG/Graphics01a-Display.pdf>)
- [4] Τυμπακιανάκης, Αποστόλης. Τοπ Λάθη που κάνουμε όταν επιλέγουμε το υλικό του Υπολογιστή (Μέρος 1ο): <http://techblog.gr/computers/tech-how-to-choose-hardware-components-52211/>
- [5] Prvulovic, M. , Durdevic, D. , Tortalja, I. (2012, November). SeeGL: Software tool for learning the OpenGL graphics library: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6419530&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6419530
- [6] Processing (Γλώσσα Προγραμματισμού) (2014). Από τη Βικιπαίδεια, την ελεύθερη εγκυκλοπαίδεια: [http://el.wikipedia.org/wiki/Processing_\(%CE%93%CE%BB%CF%8E%CF%83%CF%83%CE%B1_%CE%A0%CF%81%CE%BF%CE%B3%CF%81%CE%B1%CE%BC%CE%BC%CE%B1%CF%84%CE%B9%CF%83%CE%BC%CE%BF%CF%8D\)](http://el.wikipedia.org/wiki/Processing_(%CE%93%CE%BB%CF%8E%CF%83%CF%83%CE%B1_%CE%A0%CF%81%CE%BF%CE%B3%CF%81%CE%B1%CE%BC%CE%BC%CE%B1%CF%84%CE%B9%CF%83%CE%BC%CE%BF%CF%8D))
- [7] Χατζηκυριάκος, Γιώργος (2012). Μια εισαγωγή στο περιβάλλον και τις δυνατότητες της Processing: <http://www.linuxinside.gr/forum/%CE%BC%CE%B9%CE%B1-%CE%B5%CE%B9%CF%83%CE%B1%CE%B3%CF%89%CE%B3%CE%AE-%CF%83%CF%84%CE%BF-%CF%80%CE%B5%CF%81%CE%B9%CE%B2%CE%AC%CE%BB%CE%BB%CE%BF%CE%BD-%CE%BA%CE%B1%CE%B9-%CF%84%CE%B9%CF%82-%CE%B4%CF%85%CE%BD%CE%B1%CF%84%CF%8C%CF%84%CE%B7%CF%84%CE%AD%CF%82-%CF%84%CE%B7%CF%82-processing>

- [8] Vera, Lucia. , Campos, Ruben. , Herrera, Gerardo. , Romero, Cristina. , (2007, Μάρτιος). Computer graphics applications in the education process of people with learning difficulties:
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.103.6139&rep=rep1&type=pdf>
- [9] Programa SEDEA (2014): <http://www.ondaeduca.com/en/products/programa-sede>
- [10] Stellarium: <http://www.stellarium.org/el/>
- [11] 3D Graphics with OpenGL Basic Theory (2012, Ιούλιος):
http://www.ntu.edu.sg/home/ehchua/programming/opengl/cg_basicstheory.html
- [12] Graphics library (2014). Από τη Βικιπαίδεια, την ελεύθερη εγκυκλοπαίδεια:
http://en.wikipedia.org/wiki/Graphics_library
- [13] DirectX (2014). Από τη Βικιπαίδεια, την ελεύθερη εγκυκλοπαίδεια:
<http://en.wikipedia.org/wiki/DirectX>
- [14] DirectX Graphics and Gaming (2014): <http://msdn.microsoft.com/en-us/library/windows/desktop/ee663274%28v=vs.85%29.aspx>)
- [15] Τι είναι Direct3D εξόδου: <http://el.wingwit.com/Software/other-computer-software/144746.html - .U2eSrHbDunk>
- [16] Γεώργιος Ε. Πάνος (Δεκέμβριος 2009). Τεχνικές ανάπτυξης 3D παιχνιδιών με την Java 3D. (Μεταπτυχιακή εργασία, Πανεπιστήμιο Μακεδονίας, Θεσσαλονίκη-Τμήμα Εφαρμοσμένης Πληροφορικής, 2009):
https://dspace.lib.uom.gr/bitstream/2159/13730/1/Panos_Msc2010.pdf
- [17] QuickDraw 3D (2013). Από τη Βικιπαίδεια, την ελεύθερη εγκυκλοπαίδεια:
http://en.wikipedia.org/wiki/QuickDraw_3D
- [18] Mantle (API) (2014). Από τη Βικιπαίδεια, την ελεύθερη εγκυκλοπαίδεια:
http://en.wikipedia.org/wiki/Mantle_%28API%29
- [19] WebGL(2014). Από τη Βικιπαίδεια, την ελεύθερη εγκυκλοπαίδεια:
<http://en.wikipedia.org/wiki/WebGL>
- [20] Java 3D (2014). Από τη Βικιπαίδεια, την ελεύθερη εγκυκλοπαίδεια:
http://en.wikipedia.org/wiki/Java_3D
- [21] QSDK (2012). Από τη Βικιπαίδεια, την ελεύθερη εγκυκλοπαίδεια:
<http://en.wikipedia.org/wiki/QSDK>
- [22] Simple Direct Media Layer (2014). Από τη Βικιπαίδεια, την ελεύθερη εγκυκλοπαίδεια: http://en.wikipedia.org/wiki/Simple_DirectMedia_Layer
- [23] Simple and Fast Multimedia Library: <http://www.sfml-dev.org/>

- [24] Simple and Fast Multimedia Library (2014). Από τη Βικιπαίδεια, την ελεύθερη εγκυκλοπαίδεια: <http://en.wikipedia.org/wiki/SFML>
- [25] Βιβλιοθήκες Ανοικτού Κώδικα για Γραφικό Σχεδιασμό στα Windows στη C/C++ για αρχάριους και μη:
https://www.google.gr/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0CC8QFjAA&url=http%3A%2F%2Fwww.myprotia.gr%2Fcomm unity%2Fviewtopic%2Cmode%2Cattach%2Cid%2C772.html&ei=2WCHU6KW O-mp4gSToYGQCQ&usg=AFQjCNHkxqhDpr8RISM51Eb8sfTz2qs3JA&sig2=cU5tb5ZqBQCrW2z_DAIIRA&bvm=bv.67720277,d.bGE
- [26] Allegro (software) (2014). Από τη Βικιπαίδεια, την ελεύθερη εγκυκλοπαίδεια: [http://en.wikipedia.org/wiki/Allegro_\(software\)](http://en.wikipedia.org/wiki/Allegro_(software))
- [27] Cairo (graphics) (2014). Από τη Βικιπαίδεια, την ελεύθερη εγκυκλοπαίδεια: [http://en.wikipedia.org/wiki/Cairo_\(graphics\)](http://en.wikipedia.org/wiki/Cairo_(graphics))
- [28] Mesa (computer graphics) (2014). Από τη Βικιπαίδεια, την ελεύθερη εγκυκλοπαίδεια: http://en.wikipedia.org/wiki/Mesa_3D
- [29] Skia Graphics Engine (2014). Από τη Βικιπαίδεια, την ελεύθερη εγκυκλοπαίδεια: <http://en.wikipedia.org/wiki/Openskia>
- [30] Free Graphics Libraries, 2D & 3D Engines, Image Drawing (2011): <http://www.thefreecountry.com/sourcecode/graphics.shtml>
- [31] The website programmer's dynamic graphics generation tool: <http://www.boutell.com/gd/>
- [32] Levien, Raph. Libart: <http://www.levien.com/libart/>
- [33] Knapp, Michael. What is GraphiX? (2001): http://www.cg.tuwien.ac.at/~knapp/graphix_old/gxframe.html
- [34] National Technical University of Athens, Multimedia Technology Laboratory. (“Εισαγωγή στην OpenGL”,): http://www.medialab.ntua.gr/education/ComputerGraphics/OpenGL_Lectures/01-Introduction.pdf
- [35] Cohen, Jonathan. OpenGL: A Practical Introduction (2005): <http://www.cs.jhu.edu/~cohen/IGG/Lectures/OpenGL.pdf>
- [36] Τσιομπίκας , Γιάννης. Εισαγωγή στον προγραμματισμό 3D γραφικών με το OpenGL: <http://nuclear.mutantstargoat.com/articles/gltut/gl02/>
- [37] National Technical University of Athens, Multimedia Technology Laboratory. (“Βασικές αρχές σχεδίασης”): http://www.medialab.ntua.gr/education/ComputerGraphics/OpenGL_Lectures/02-Chapter1.pdf

- [38] Framebuffer Object (2014). Από τη Βικιπαίδεια, την ελεύθερη εγκυκλοπαίδεια: http://en.wikipedia.org/wiki/Framebuffer_Object
- [39] Java (2014). Από τη Βικιπαίδεια, την ελεύθερη εγκυκλοπαίδεια: <http://el.wikipedia.org/wiki/Java>
- [40] .NET Framework (2014). Από τη Βικιπαίδεια, την ελεύθερη εγκυκλοπαίδεια: http://en.wikipedia.org/wiki/.NET_Framework
- [41] Εισαγωγή στη C#: <http://studentguru.gr/w/tutorials/01-c>
- [42] C++ (2014). Από τη Βικιπαίδεια, την ελεύθερη εγκυκλοπαίδεια: <http://el.wikipedia.org/wiki/C%2B%2B>
- [43] GLUT-like Windowing, GUI, and Media Control toolkits: <https://www.opengl.org/resources/libraries/windowtoolkits/>
- [44] Cpw for OpenGL: <http://mathies.com/cpw/about.html>
- [45] Spitzak, Bill. (1998-2012). Fast Light Toolkit: <http://www.fltk.org/index.php>
- [46] GLFW: <http://www.glfw.org/>
- [47] Azuma, Daniel. (2000) The GLOW Toolkit: <http://glow.sourceforge.net/>
- [48] Stewart, Nigel. (2010) GLT OpenGL C++ Toolkit: <http://www.nigels.com/glt/>
- [49] Rademacher, Paul. Glui user interface library: <http://glui.sourceforge.net/>
- [50] Libraries: <https://www.opengl.org/sdk/libs/>
- [51] GLUT and OpenGL Utility Libraries: <https://www.opengl.org/resources/libraries/>
- [52] Mesa 3D and Direct Rendering Infrastructure wiki (2014): <http://dri.freedesktop.org/wiki/>
- [53] Open Inventor (2013): <http://www.vsg3d.com/open-inventor/sdk>
- [54] Coin / IntroductionToCoin3D (2014): <https://bitbucket.org/Coin3D/coin/wiki/IntroductionToCoin3D>
- [55] The OpenSceneGraph Project Website: <http://www.openscenegraph.org/>
- [56] National Technical University of Athens, Multimedia Technology Laboratory. Index of/education/ComputerGraphics/OpenGL_Lectures: http://www.medialab.ntua.gr/education/ComputerGraphics/OpenGL_Lectures/
- [57] About SDL: <http://www.libsdl.org/>
- [58] Αρχεία obj: <http://www.reviversoft.com/el/file-extensions/obj>
- [59] Computer Graphics Hardware: (2012, July): http://www.ntu.edu.sg/home/ehchua/programming/OpenGL/cg_basicstheory.html