ΓΡΑΦΙΚΑ ΥΠΟΛΟΓΙΣΤΩΝ

ΓΚΑΝΙΑΣ ΑΓΓΕΛΟΣ

ΕΙΣΑΓΩΓΗ

Τα γραφικά υπολογιστών πραγματικού χρόνου είναι αντικείμενα γραφικών (συντεταγμένες σημείων και επιφανειών, χρωματισμοί, φωτισμοί και υφές τους) τα οποία αποδίδονται οπτικά κατά τη στιγμή που εκτελείται ένα πρόγραμμα υπολογιστή, κάθε φορά που αυτό συμβαίνει, με εκ νέου εκτέλεση των κατάλληλων εντολών / υπολογισμών από τον επεξεργαστή. Για παράδειγμα, τα γραφικά που εμφανίζονται στην οθόνη ενός υπολογιστή ο οποίος εκτελεί ένα βιντεοπαιχνίδι, ανήκουν συνήθως σε αυτήν την κατηγορία. Για την προβολή τους απαιτείται κάποια μηχανή γραφικών πραγματικού χρόνου, όπως για παράδειγμα η Ogre3D, η Irrlich, το Crystal Space, μηχανές παιχνιδιών (π.χ. Source) κτλ. Τα γραφικά πραγματικού χρόνου μπορούν να είναι και αλληλεπιδραστικά, με τη μηχανή γραφικών να αποκρίνεται κατάλληλα σε εισόδους του χρήστη (π.χ. από περιφερειακά όπως το ποντίκι ή το πληκτρολόγιο), μα αυτό δεν είναι απαραίτητο. Για τον προγραμματισμό τους υπάρχουν διάφορες προτυποποιημένες βιβλιοθήκες, όπως η OpenGL και η Direct3D.

Το αντικείμενο αυτής της πτυχιακής εργασίας είναι η προσομοίωση ενός αλληλεπιδραστικού, τρισδιάστατου μοντέλου του Α.Τ.Ε.Ι. Άρτας σε πραγματικό χρόνο (real-time), φτιαγμένο από υπολογιστή, το οποίο δίνει τη δυνατότητα στο χρήστη για εμβύθιση στον μοντελοποιημένο κόσμο και τη δυνατότητα για απευθείας χειρισμό.

Ο χρήστης θα έχει την δυνατότητα ελεύθερης περιήγησης θέας πρώτου προσώπου (first person view) στους εξωτερικούς χώρους των Τ.Ε.Ι σχεδιασμένοι με μεγάλη λεπτομέρεια, μέσω μιας κάμερας προσφέροντας του μεγάλη ελευθέρια κίνησης αφού προσομοιώνει τις κινήσεις του κεφαλιού ενός πραγματικού ανθρώπου.

Οι εξωτερικοί χώροι του Α.Τ.Ε.Ι. απεικονίζουν κατά ένα αρκετά μεγάλο ποσοστό την πραγματικότητα, αφού έχουν σχεδιαστεί με μεγάλη ακρίβεια όσον αφορά τις διαστάσεις του, και με μεγάλη λεπτομέρεια αφού οι υφές που έχουν χρησιμοποιηθεί ξεπερνούν σε ανάλυση τα 2 Megapixels.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 12: ΠΑΡΑΔΕΙΓΜΑΤΑ ΚΩΔΙΚΑ

CHAPTER I

ΕΙΣΑΓΩΓΗ ΣΤΗΝ OPENGL

1. Τι είναι η OpenGl

Η OpenGL (***Open Graphics Library)*** δεν είναι απλά μια βιβλιοθήκη σχεδίασης γραφικών αλλά ένα πρότυπο υλοποίησης βιβλιοθηκών σχεδίασης γραφικών. Εμπεριέχει δηλαδή το σύνολο των συναρτήσεων που πρέπει να υλοποιεί μία βιβλιοθήκη γραφικών προκειμένου να είναι συμβατή με αυτό. Το πρότυπο αυτό λοιπόν καθορίζει μια προγραμματι-στική διεπιφάνεια ***(application programming interface ή API).*** Οι εντολές της OpenGl μπορούν να σας επιτρέψουν να καθορίσετε σχήματα ή συνδυασμό σχημάτων, όπως από το πιο απλό που είναι ένα δυσδιάστατο τετράγωνο μέχρι και πιο σύνθετα όπως ένα κτίριο, ένας βράχος ή μια κυρτή επιφάνια μέσα από ένα μικρό σύνολο γεωμετρικών σχημάτων όπως σημεία, γραμμές και πολύγωνα.

1. Σύνταξη εντολών της OpenGl

Οι εντολές της OpenGL χρησιμοποιούν το πρόθεμα gl και αρχικά κεφαλαία γράμματα για κάθε λέξη που συν-θέτουν το όνομα της εντολής όπως η glClearColor.Ομοίως, οι καθορισμένες σταθερές της OpenGL αρχίζουν με GL_, χρησιμοποιώντας μόνο κεφαλαία γράμματα και χωρίζοντας τις λέξεις με underscores όπως η GL_COLOR_BUFFER_BIT . Επίσης επισυνάπτονται κάποια ξένα γράμματα σε ορισμένες εντολές όπως οι glColor3f και glVertex3f. Ειδικότερα, ο αριθμός 3 της εντολής υποδεικνύει ότι η συγκεκριμένη εντολή χρησιμοποιεί τρία ορίσματα και το f υποδεικνύει ότι τα ορίσματα αυτά είναι αριθμοί floating-point. Μερικές εντολές της Openl δέχονται 8 διαφορετικούς τύπους δεδομένων για τα ορίσμτα τους. Τα γράμματα που χρησιμοποιούνται ως επιθήματα για να προσδιορίσουν αυτούς τους τύπους δεδομένων παρουσιάζονται στον Πίνακα 1-1, μαζί με τους αντίστοιχους ορισμούς τύπου της OpenGL

Πίνακας 1-1: Επιθέματα Εντολών και Τύποι Δεδομένων επιχειρημάτων

Suffix	Data types	Language Type	Type Definition
b	8-bit integer	signed char	GLbyte
s	16-bit integer	Short	GLshort
i	32-bit integer	int or long	GLint, GLsizei
f	32-bit floating-point	Float	GLfloat, GLclampf
d	64-bit floating-point	Double	GLdouble, GLclampd
ub	8-bit unsigned integer	unsigned char	GLubyte, GLboolean
us	16-bit unsigned integer	unsigned short	GLushort
ui	32-bit unsigned integer	unsigned int or unsigned long	GLuint, GLenum,

Ως παράδειγμα θα επιλέξουμε την εντολή glVertex*, με την οποία ορίζουμε τις συντεταγμένες ενός σημείου στο επίπεδο ή στον τρισδιάστατο χώρο. Το επίθημα καθορίζεται από τον τύπο δεδομένων, σύμφωνα με τον *Πίνακα 1-1*. Έτσι λχ για τον καθορισμό ενός σημείου οι συντεταγμένες του δίνονται υπό τη μορφή πραγματικών αριθμών απλής ακρίβειας (GLfloat) με κλήση τιμής η αντίστοιχη εντολή έχει τη μορφή ***glVertex2f (GLfloat x, GLfloat y)*** στον δυσσδιάστατο χώρο και ***glVertex3i(GLint x, GLint y, GLint z)*** στον τρισδιάστατο χώρο.

```
glVertex2i(1,  3);          κορυφές συντεταγμένων ως ακέραιους 32-bit αριθμούς
glVertex2f(1.1,  3.7);      κορυφές συντεταγμένων ως δεκαδικούς  32-bit floating-point  αριθμούς
glVertex3i(1,  3,  2);      κορυφές συντεταγμένων στον 3Δ χώρο ως ακέραιους 32-bit αριθμούς
```

Επίσης οι συντεταγμένες ενός σημείου μπορουν να δοθούν στην εντολή glVertex δίνοντας το δείκτη ενός πίνακα που περιέχει τις τιμές.
Ο παρακάτω πίνακας καθορίζει ένα σημείο στον τρισδιάστατο χώρο με συντεταγμένες x = 1, y = 2 και z = 3.

```
GLfloat coord[]={1,2,3};
```

Στην περίπτωση αυτή, στο όνομα της εντολής προστίθεται το επίθημα ***v (vector)***, το οποίο προσδιορίζει ότι περνάμε ως όρισμα έναν δείκτη σε μητρώο. Επομένως, η παραλλαγή της εντολής συντάσεται ως εξής:

```
GLfloat coord[]={1,2,3} ;
glVertex3fv(coord);
```

1.1. Οι βιβλιοθήκες της OpenGl

Η OpenGL παρέχει ένα ισχυρό σύνολο εντολών απόδοσης, καθώς όλα τα σχέδια υψηλότερου επιπέδου πρέπει να γίνουν βάσει αυτών των εντολών.Ένας αριθμός βιβλιοθηκών υπάρχουν για την απλοποίηση των εργασιών προγραμματισμού συμπεριλαμβανομένων των εξής:

- **Βασική Βιβλιοθήκη (OpenGL Core Library ή GL)**

Η βασική βιβλιοθήκη της OpenGL περιέχει τις κύριες εντολές σχεδίασης. Όλες οι εντολές της βιβλιοθήκης αυτής διακρίνονται από το πρόθεμα ***gl.*** Πολλές από τις συναρτήσεις της δέχονται προκαθορισμένα ορίσματα (συμβολικές στα-θερές) τα οποία έχουν οριστεί στη βιβλιοθήκη και αντιστοιχούν σε διάφορες παραμέτρους ή καταστάσεις λειτουργίας. Κατά σύμβαση, οι σταθερές αυτές ξεκινούν με το πρόθεμα ***GL_*** .

- **The OpenGL Utility Library (GLU)**

Η OpenGL Utility Library (GLU) είναι μια βιβλιοθήκη γραφικών υπολογιστή για OpenGL. Αποτελείται από μια σειρά από λειτουργίες που χρησιμοποιούν τη βασική βιβλιοθήκη OpenGL για την παράγωγη ρουτινών υψηλότερου επιπέδου σχεδίασης σε αντίθεση με τις πιο πρωτόγονες ρουτίνες που παρέχει το OpenGL.
Ανάμεσα σε αυτά τα χαρακτηριστικά υπάγεται και η παράγωγη των mipmap υφών, η κατασκευή quadric επιφανειών, παραγωγή κυρτών επιφανειών , η ερμηνεία σφαλμάτων κώδικα, μια εκτεταμένη σειρά από ρουτίνες για τη δημιουργία προβολής και απλή τοποθέτηση της κάμερας. Όλες οι εντολές της βιβλιοθήκης GLU ξεκινούν με το πρόθεμα ***glu*.**

- **The OpenGL Utility Toolkit (GLUT)**

Όπως γνωρίζετε, η OpenGL περιέχει εντολές απόδοσης αλλά είναι σχεδιασμένη να είναι ανεξάρτητη από οποιοδήποτε σύστημα παραθύρων ή λειτουργικό σύστημα. Κατά συνέπεια, δεν περιέχει εντολές για το άνοιγμα των παραθύρων ή ανάγνωση συμβάντων από το πληκτρολόγιο ή το ποντίκι. Δυστυχώς, είναι αδύνατο να γραφεί ένα πλήρες πρόγραμμα γραφικών χωρίς τουλάχιστον να ανοιχθεί ένα παράθυρο, και ακόμη τα πιο ενδιαφέροντα προγράμματα απαιτούν μια εισαγωγή δεδομένων από το χρήστη ή άλλες υπηρεσίες από το λειτουργικό σύστημα ή το σύστημα παραθύρων .Έτσι η OpenGl χρησιμοποιεί την βιβλιοθήκη GLUT για να απλοποιήσει το άνοιγμα των παραθύρων, την εισαγωγή δεδομένων από το χρηστή, και ούτω καθεξής. Επιπλέον, δεδομένου ότι οι εντολές σχεδίασης της OpenGL περιορίζονται σε εκείνα που δημιουργούν απλά γεωμετρικά σχήματα (σημεία, γραμμές και πολύγωνα), η βιβλιοθήκη GLUT περιλαμβάνει αρκετές ρουτίνες που δημιουργούν πιο περίπλοκα τρισδιάστατα αντικείμενα, όπως μια σφαίρα, μια σπείρα, και μια τσαγιέρα ή ακόμη έτοιμα γεωμετρικά σχήματα όπως σφαίρες, κώνους κυλίνδρους , τετράγωνα, και δίσκους. Όλες οι εντολές της βιβλιοθήκης GLU ξεκινούν με το πρόθεμα ***glut*** .

1.2. Σχηματισμός παραθύρων (Window Management)

Τέσσερις ρουτίνες εκτελούν καθήκοντα που είναι απαραίτητα για να προετοιμάσει ένα παράθυρο.

`glutInit(int *argc, char **argv)`

Η glutInit θα προετοιμάσει τη βιβλιοθήκη GLUT και θα διαπραγματευτεί μια συνεδρία με το σύστημα παραθύρων. Κατά τη διάρκεια αυτής της διαδικασίας, η glutInit μπορεί να προκαλέσει τον τερματισμό του προγράμματος της βιβλιοθήκης GLUT με ένα μήνυμα λάθους στο χρήστη αν η GLUT δεν έχει προετοιμαστεί σωστά.
όπου argc και argv το πλήθος και οι τιμές των ορισμάτων που περνάμε στο πρόγραμμα μέσω της γραμμής εντολών (command line arguments). Το όρισμα argv επιστρέφει το όνομα του εκτελέσιμου αρχείου.

`glutInitWindowPosition(int x, int y)`

καθορίζει τη θέση του παραθύρου σε σχέση με την επάνω αριστερή γωνία του οθόνης

`glutInitWindowSize(int width, int size)`

όπου width και height το πλάτος και ύψος της επιφάνειας σχεδίασης της εφαρμογής σε pixels.
Επισημαίνουμε ότι οι διαστάσεις της επιφάνειας σχεδίασης μπορούν να μεταβληθούν από το χρήστη σε κάθε χρονική στιγμή κατά την εκτέλεση του προγράμματος με τη χρήση του ποντικιού.

`glutCreateWindow(const char *title)`

η οποία επιστρέφει μια ακέραιη τιμή που λειτουργεί ως αναγνωριστικό του παραθύρου (χρήσιμη παράμετρος στην περίπτωση δημιουργίας πολλαπλών παραθύρων). Η παράμετρος title αντιστοιχεί στον τίτλο του παραθύρου.

1.2.1. Παράμετροι λειτουργίας παραθύρων (The Display Callback)

Κατά την αρχικοποίηση του παραθύρου ορίζουμε αν αυτό προορίζεται να περιέχει στατικές σκηνές ή κινούμενα γραφικά όπως και το χρωματικό μοντέλο που θα χρησιμοποιηθεί. Οι ρυθμίσεις αυτές δίνονται με την εντολή *glutInitDisplayMode:*

`glutInitDisplayMode(unsigned int mode);`
όπου mode προκαθορισμένες αριθμητικές σταθερές που παίρνουν τις παρακάτω τιμές:

`GLUT_SINGLE:` Για την απεικόνιση χρησιμοποιείται η τεχνική της απλής ενταμίευσης (χρησιμοποιείται ένας ενταμιευτής χρωματικών τιμών). Αυτή η ρύθμιση επιλέγεται για τη σχεδίαση στατικών σκηνών.

`GLUT_DOUBLE:` Χρησιμοποιείται η τεχνική της διπλής ενταμίευσης (double buffering) χρησιμοποιούνται δηλαδή δύο ενταμιευτές χρωματικών τιμών. Η επιλογή αυτή ενδείκνυται για την παρουσίαση κινούμενων γρα-φικών, για λόγους που θα εξηγηθούν στο Κεφάλαιο "Κινούμενα γραφικά".

`GLUT_RGB:` Τα χρώματα ορίζονται με την περιγραφή τους στο χρωματικό μοντέλο RGB. Κάθε χρώμα δηλαδή περιγράφεται από τους συντελεστές βάρους της κόκκινης , της πράσινης και μπλε χρωματικής συνιστώσας του.

`GLUT_RGBA:` Τα χρώματα ορίζονται στο μοντέλο RGBA.

`GLUT_INDEX:` Χρήση του μοντέλου χρωματικών πινάκων (colour tables).

1.3. Διαχείριση Γεγονότων (Handling Input Events)

Ως γεγονός ορίζουμε την καταγραφή κάποιας δραστηριότητας του συστήματος, συνήθως μιας δραστηριότητας από κάποια συσκευή εισόδου όπως ένα πληκτρολόγιο ή ένα ποντίκι. Ωστόσο υπάρχουν και γεγονότα που εγείρονται από το λειτουργικό σύστημα υπό ορισμένες συνθήκες. Μία καταγραφή γεγονότος περιέχει πληροφορίες που το προσδιορίζουν επακριβώς. Π.χ. μία καταγραφή γεγονότος από το πληκτρολόγιο περιέχει την ταυτότητα του πλήκτρου που πιέστηκε και τη θέση του δείκτη όταν αυτό πατήθηκε. Μία καταγραφή γεγονότος από το ποντίκι περιέχει πληροφορίες για το πλήκτρο του ποντικιού που πιέστηκε και την τοποθεσία του δείκτη στην οθόνη όταν πιέστηκε.

1.3.1. Γεγονός αλλαγής διαστάσεων παραθύρου

`void glutReshapeFunc (void (* FUNC) (int w, int h)`
Η συνάρτηση κλήσης reshape είναι χρήσιμη για να επαναπροσδιορίσουμε το εύρος της επιφάνειας σχεδίασης που θα αξιοποιηθεί μετά από αλλαγές των διαστάσεων του παραθύρου. Εάν δεν ορίσουμε συνάρτηση διαχείρισης του γεγονότος, η μηχανή της OpenGL αυτομάτως επεκτείνει τη σχεδίαση της σκηνής σε όλη τη διαθέσιμη επιφάνεια που προσφέρει το παράθυρο σχεδίασης. Αυτό σημαίνει ότι εάν θελήσουμε να επιβάλλουμε κάποιους περιορισμούς - όπως λ.χ. να χρησιμοποιήσουμε διαφορετική αναλογία πλάτους-ύψους από αυτή που έχει η επιφάνεια σχεδίασης στο σύνολό της – θα πρέπει να καταχωρήσουμε μια συνάρτηση κλήσης που ορίζει το παράθυρο σχεδίασης.

1.3.2. Αλληλεπίδραση από το ποντίκι

`void glutMouseFunc(void mouseClicked(int button, int state, int x, int y)):`
Καταχωρεί τη συνάρτηση mouseClicked ως συνάρτηση διαχείρισης γεγονότων πίεσης πλήκτρων ποντικιού.
Η τιμή του ορίσματος button που επιστρέφει το λειτουργικό σύστημα εξαρτάται από το κουμπί του ποντικιού που πιέστηκε:

- GL_LEFT_BUTTON: Πιέστηκε το αριστερό κουμπί του ποντικιού.
- GL_MIDDLE_BUTTON: Πιέστηκε το μεσαίο κουμπί του ποντικιού.
- GL_RIGHT_BUTTON: Πιέστηκε το το δεξί κουμπί του ποντικιού.
- GLUT_DOWN: Πίεση του πλήκτρου του ποντικιού.
- GLUT_UP: Απελευθέρωση του πλήκτρου του ποντικιού.

Τα ορίσματα x, y εκφράζουν τη θέση του δείκτη του ποντικιού όταν πιέστηκε το κουμπί (ακέραιες τιμές με τη θέση στην πάνω αριστερή γωνία της επιφάνειας σχεδίασης της εφαρμογής).

1.3.3. Αλληλεπίδραση από το πληκτρολόγιο

Η αλληλεπίδραση με το πληκτρολόγιο επιτρέπει την σύνδεση ενός πλήκτρου του με μια ρουτίνα που καλείται όταν το κουμπί του πληκτρολογίου πατιέται ή απελευθερώνεται.
Δεδομένου ότι τα πλήκτρα διαχωρίζονται σε δύο κατηγορίες, τα πλήκτρα χαρακτήρων και τα ειδικά πλήκτρα (Function keys, Insert, Page Up, Page Down κλπ.), ομοίως ορίζονται και δύο κατηγορίες γεγονότων πληκτρολογίου.
Η **`glutKeyboardFunc (void (* FUNC) ((unsigned char, int x, int y))`** αλληλεπιδρά με τα πλήκτρα χαρακτήρων και η **`void glutSpecialFunc(void specialKey(int key, int x, int y))`** με τα ειδικά πλήκτρα**.**
Example 1-5: Reshape Callback Function

```
GLvoid ReSizeGLScene(GLsizei width, GLsizei height)
        {
                if (height==0)
                {
                        height=1;
                }

                glViewport(0,0,width,height);
                glMatrixMode(GL_PROJECTION);
                glLoadIdentity();

                gluPerspective(45.0f,(GLfloat)width/(GLfloat)height,5.0f, 130000.0f);

                glMatrixMode(GL_MODELVIEW);
                glLoadIdentity();
        }
```

Example 1-6 : Mouse Callback Function

```
void mouseButtonClicked(int button,int state,int x, int y)
        {
                if (state==GLUT_DOWN)
                {
                        ....
                        if (button==GLUT_LEFT_BUTTON)
                        glColor3f(1,0,0);
                        ....
                }
        }

        int main(int argc, char ** argv)
        {
                ....
                glutMouseFunc(mouseButtonClicked);
                ....
        }
```

Example 1-7 : Keyboard Function

```
  void keyboard(unsigned char key,int x, int y)
  {
        if (key== 't' )
        {
                y1++;   y2++;
        }
  }

  int main(int argc, char** argv)
  {
  ....
  glutKeyboardFunc(keyboard);
  ....
  }
```

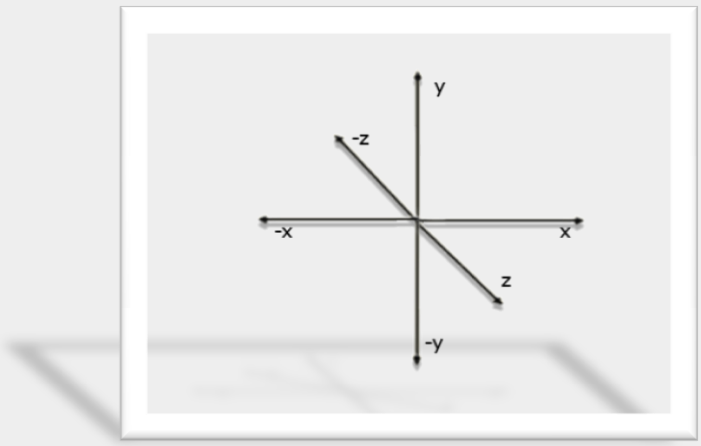
Example 1-8 : Special keys Keyboard Function

```
void specialKeys(int key,int x, int y)
{
        if (key==GLUT_KEY_UP)
        {
                y1++;   y2++;
        }
}

int main(int argc, char** argv)
{
        ....
        glutSpecialFunc(specialKeys);
        ....
}
```

CHAPTER II

ΒΑΣΙΚΕΣ ΑΡΧΕΣ ΣΧΕΔΙΑΣΗΣ

Όλα τα βασικά σχήματα στην OpenGL σχηματίζονται με δήλωση των κορυφών τους. Με τον όρο "βασικά σχήματα" αναφερόμαστε σε γραμμές, τρίγωνα και πολύγωνα. Πιο "σύνθετα" σχήματα, όπως κύκλοι και ελλείψεις προσεγγίζονται με τη σύνδεση πολλαπλών σημείων τους με ευθύγραμμα τμήματα.
Στην OpenGL η σκηνή αναπαρίσταται στη γενική περίπτωση σε τρισδιάστατο καρτεσιανό σύστημα συντεταγμένων.



Σχημα 2.1: Τρισδιάστατο σύστημα συντεταγμένων

2.1. Καθαρισμός Οθόνης (The Color Buffer)

Πριν αρχίσουμε το σχεδιασμό 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 εννοούμε την αρχικοποίηση των τιμών του ενταμιευτή με κάποια προκαθορισμένη τιμή. Ο καθαρισμός γίνεται με το χρώμα φόντου που επιλέγουμε εμείς.
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);
```

όπου red, green, blue τα βάρη του χρώματος στο χρωματικό μοντέλο RGBA. Η τέταρτη παράμετρος αποκαλείται "συνιστώσα alpha" και παίζει το ρόλο "συντελεστή διαφάνειας".

Ο καθαρισμός της οθόνης γίνεται με την εντολή glClear(): ***glClear(GLEnum buffer);***

όπου buffer ο ενταμιευτής που θέλουμε να καθαρίσουμε. Η εντολή glClear() αποδίδει σε όλες τις θέσεις μνήμης του ενταμιευτή την προκαθορισμένη τιμή καθαρισμού για το συγκεκριμένο ενταμιευτή. Π.χ. αποδίδει στον ενταμιευτή χρώματος τις χρωματικές τιμές του φόντου.
Δεδομένου ότι η μηχανή της OpenGL περιέχει πολλούς ενταμιευτές (*Πίνακας 2*), πρέπει να καθορίσουμε το είδος του ενταμιευτή που επιθυμούμε να καθαρίσουμε, περνώντας ως όρισμα την κατάλληλη σταθερά στην εντολή glClear( ). Προκειμένου λ.χ. να καθαρίσουμε τον ενταμιευτή χρωματικών τιμών, δίνουμε ως όρισμα τη σταθερά GL_COLOR_BUFFER_BIT.

Για παράδειγμα, αυτές οι γραμμές κώδικα καθαρίζουν ένα παράθυρο λειτουργίας σε μαύρο:

```
glClearColor(0.0,  0.0,  0.0, 0.0);
glClear(GL_COLOR_BUFFER_BIT);
```

Πίνακας 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

2.2. Καθορισμός Χρωμάτων

Με την OpenGL, η περιγραφή του σχήματος ενός αντικειμένου και ο χρωματισμός του είναι ανεξάρτητα. Κάθε φορά που δίνουμε εντολή σχεδιάσης ενός συγκεκριμένου γεωμετρικού σχήματος, το τρέχον επιλεγμένο χρώμα, όντας μεταβλητή κατάστασης, καθορίζει και το χρώμα με το οποίο το σχήμα σχεδιάζεται.
Για να καθορίσουμε ένα χρώμα, χρησιμοποιούμε την εντολή glColor3f(). Η εντολή αυτή παίρνει τρεις παραμέτρους, οι οποίες είναι όλες αριθμοί κινητής υποδιαστολής ή διπλής ακρίβειας (μεταξύ 0.0 και 1.0) ή ακέραιοι, ανάλογα με το ποια από τις παρακάτω τρείς μορφές επιλέγουμε:

```
void glColor3f(GLfloat r, GLfloat g, GLfloat b);
```
$0 \leq r, g, b \leq 1$

```
void glColor4f(GLfloat r, GLfloat g, GLfloat b, GLfloat a);
```
$0 \leq r, g, b \leq 1, \quad 0 = a \leq 1.$

Πίνακας 2.1 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

2.3. Δήλωση Κορύφων

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

```
void glVertex{234}{sifd}[v](TYPE coords);
```

Η εντολή αυτή καθορίζει μία κορυφή, η οποία θα χρησιμοποιηθεί για την περιγραφή ενός γεωμετρικού σχήματος. Μπορούμε να δώσουμε μέχρι τέσσερις συντεταγμένες (x, y, z, w) για μία συγκεκριμένη κορυφή ή και μέχρι δύο (x, y), χρησιμοποιώντας την κατάλληλη εκδοχή της εντολής. Η εντολή glVertex*() πρέπει να εκτελείται μεταξύ των εντολών glBegin() και glEnd().

```
glVertex2i(2,  3);
```
Δήλωση σημείου με συντεταγμένες (x,y)=(2,3)

```
glVertex3f(0,  0,  3.14);
```
Δήλωση σημείου με συντεταγμένες (x,y,z)=(0,0,3.14)

```
GLdouble dvect[3] = {5,  9,  7};
glVertex3dv(dvect);
```
Δήλωση σημείου που οι τιμές του βρίσκονται στο μητρώο dvect

Ο ορισμός των κορυφών κάθε γεωμετρικού σχήματος περικλείεται μεταξύ δύο εντολών, των glBegin() καιglEnd():
Το είδος του γεωμετρικού σχήματος που σχεδιάζεται, εξαρτάται από την παράμετρο που δίνουμε στο όρισμα mode της εντολής glBegin.

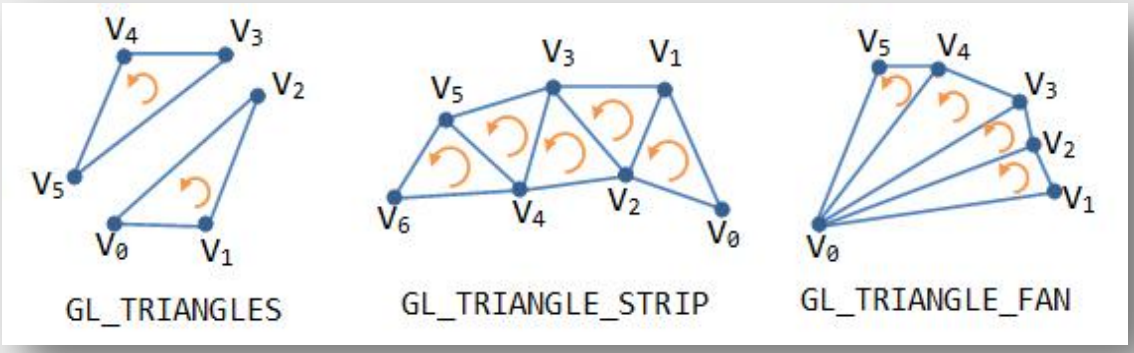
```
void glBegin(GLEnum mode);
...
void glEnd();
```

2.4. Σχεδιασμός γεωμετρικών σχημάτων

Αυτή η ενότητα εξηγεί πώς γίνεται ο σχεδιασμός γεωμετρικών σχημάτων στήν OpenGL. Όλα τα γεωμετρικά σχήματα περιγράφονται με βάση τις κορυφές τους ***(vertices)*** - συντεταγμένες που ορίζουν τα καθεαυτά σημεία, απολήξεις των τμημάτων της γραμμής, ή γωνίες των τριγώνων.

2.4.1. Τρίγωνα (Triangles)

Τριάδες κορυφών που ερμηνεύονται ως τρίγωνα με χρήση των παραμέτρων GL_TRIANGLES (απλά τρίγωνα), GL_TRIANGLE_STRIP (τριγώνα συνδεδεμένα σε σειρά) και GL_TRIANGLE_FAN (τριγώνα συνδεδεμένα σε έλικα)

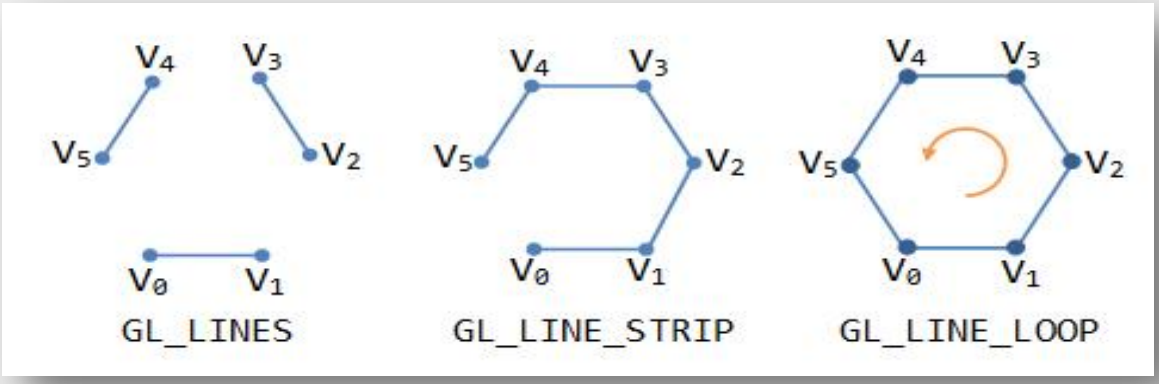


Ο παρακάτω κώδικας υλοποιεί τέσσερα τρίγωνα συνδεδεμένα σε σειρά με την GL_TRIANGLE_STRIP.

```
glBegin(GL_TRIANGLE_STRIP);
glVertex3f( 0.0f, 0.0f, 0.0f ); //vertex 0
glVertex3f( 0.5f, 1.0f, 0.0f ); //vertex 1
glVertex3f( 1.0f, 0.0f, 0.0f ); //vertex 2
glVertex3f( 1.5f, 1.0f, 0.0f ); //vertex 3
glVertex3f( 2.0f, 0.0f, 0.0f ); //vertex 4
glVertex3f( 2.5f, 1.0f, 0.0f ); //vertex 5
glEnd();
```

2.4.2. Γραμμές (Lines)

Ζεύγη κορυφών που ερμηνεύονται ως ξεχωριστά ευθύγραμμα τμήματα με χρήση της GL_LINE (ένα απλό ευθύγραμμο τμήμα), GL_LINE_STRIP (ενωμένα ευθύγραμμα τμήματα) και GL_LINE_LOOP (σειρές από ενώ-μένα ευθύγραμμα τμήματα με τη διαφορά ότι προστίθεται ένα ευθύγραμμο τμήμα μεταξύ της τελευταίας και της πρώ-της κορυφής).

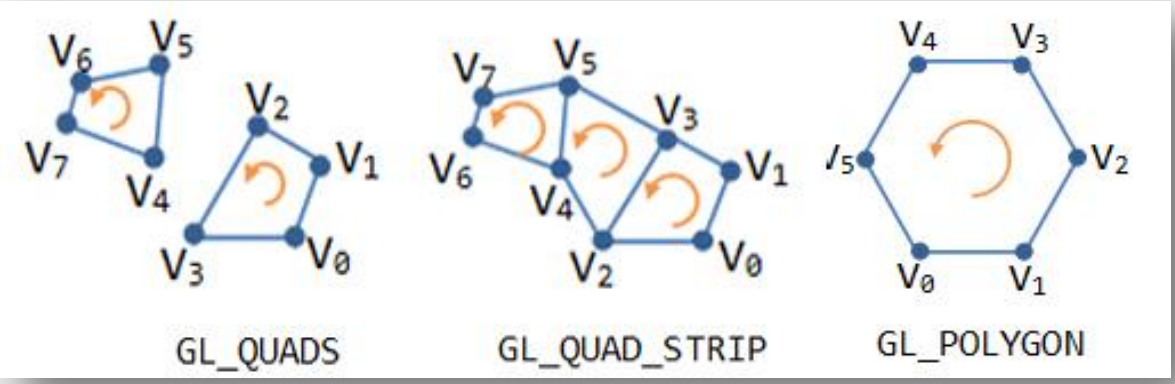


Ο παρακάτω κώδικας υλοποιεί έξι γραμμές συνδεδεμένες σε σειρά με την GL_LINE_STRIP και πάχος τρία pixel.

```
glLineWidth(3.0);

glBegin(GL_ LINE_STRIP);
glVertex3f( 0.0f, 0.0f, 0.0f ); //vertex 0
glVertex3f( 1.0f, 0.0f, 0.0f ); //vertex 1
glVertex3f( 1.5f, 1.0f, 0.0f ); //vertex 2
glVertex3f( 1.0f, 2.0f, 0.0f ); //vertex 3
glVertex3f( 0.0f, 2.0f, 0.0f ); //vertex 4
glVertex3f( 1.0f, 3.0f, 0.0f ); //vertex 5
glEnd();
```

2.4.3. Τετράπλευρα (Quads)

Κορυφές που ειρηνεύονται ως πολύγωνα τεσσάρων πλευρών με χρήση της GL_QUADS (ένα απλό τετράπλευρο) και GL_QUADS_STRIP (δύο ή περισσότερα πολύγονα συνδεδεμένα σε σειρά) και ως πολύγονα τριών ή και περισσοτέρων πευρών με χρήση της GL_POLYGON.



Ο παρακάτω κώδικας υλοποιεί ένα κύβο με την GL_QUADS ορίζοντας για κέθε πλεύρα τοτ διαφορετικό χρώμα.

```
glBegin(GL_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();
```

CHAPTER III

ΚΑΤΑΣΚΕΥΗ ΤΡΙΣΔΙΑΣΤΑΤΩΝ ΕΠΙΦΑΝΕΙΩΝ

Στην ενότητα αυτή παρουσιάζουμε τις εντολές μέσω των οποίων ορίζουμε στην OpenGL τρισδιάστατες επιφάνειες. Η πλειοψηφία των επιφανειών εντάσσεται στην κατηγορία των τετραγωνικών επιφανειών (quadrics), γιατί εκφράζονται με εξισώσεις δευτέρου βαθμού. Οι εντολές σχηματισμού τρισδιάστατων επιφανειών περιέχονται στις βιβλιοθήκες GLUT, GLU και GL.

3.1. Εντολές της βιβλιοθήκης GLUT

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**.

Κύβος

```
void glutWireCube ( GLdouble edgeLength );
void glutSolidCube ( GLdouble edgeLength );
```

Η παράμετρος edgeLength καθορίζει το μήκος των ακμών.

Σφαίρα

```
void glutWireSphere( GLdoule radius, GLint slices, GLint stacks );
void glutSolidSphere( GLdoule radius, GLint slices, GLint stacks );
```

Η παράμετρος radius δηλώνει την ακτίνα της σφαίρας.
Το όρισμα slices δηλώνει το πλήθος των κατακορύφων υποδιαιρέσεων (μεσημβρινοί).
Το όρισμα stacks δηλώνει το πλήθος των οριζοντίων υποδιαιρέσεων (γεωγραφικά πλάτη).

Κώνος

```
void glutWireCone(GLdouble base, GLdouble height, GLint slices, Glint  stacks);
void glutSolidCone (GLdouble base, GLdouble height, GLint slices, Glint  stacks);
```

Η παράμετρος base ορίζει την ακτίνα της βάσης του κώνου
Η παράμετρος height ορίζει το ύψος του.
Η παράμετρος slices καθορίζει το πλήθος των οριζοντίων υποδιαιρέσεων
Η παράμετρος stacks καθορίζει το πλήθος των κατακόρυφων υποδιαιρέσεων

3.2. Εντολές της βιβλιοθήκης GLU

Οι εντολές της βιβλιοθήκης GLU έχουν πιο πολύπλοκη σύνταξη σε σχέση με τις εντολές της GLUT, ωστόσο υποστηρίζουν περισσότερες δυνατότητες. Υποστηρίζουν λ.χ. τη δυνατότητα απόδοσης υφής στο σχήμα , όπως θα δούμε στο κεφάλαιο "Απόδοση υφής".

3.2.1. Επιφάνειες - Quadrics

Κάθε επιφάνεια χαρακτηρίζεται προγραμματιστικά ως ένα αντικείμενο της κλάσης ***GLUquadricObj***. Επομένως, η δημιουργία κάθε νέου σχήματος απαιτεί την αρχικοποίηση ενός νέου αντικειμένου, έστω ***qObj***:

```
GLUquadricObj * qObj;
qObj = gluNewQuadric( );
```

Με τις εντολές της βιβλιοθήκης GLU έχουμε τη δυνατότητα να αναπαραστήσουμε τις τρισδιάστατες επιφάνειες είτε ως συμπαγείς είτε με τη μορφή πλέγματος, είτε σχεδιάζοντας απλώς τις κορυφές τους. Η επιλογή του τρόπου αναπαράστασης γίνεται με την εντολή **gluQuadricDrawStyle:**

```
void gluQuadricDrawStyle(GLUquadric *quadObject, GLenum drawStyle);
```

όπου quadObject το αντικείμενο της επιφανείας για την οποία καθορίζουμε τον τρόπο αναπαράστασης. Η τιμή drawStyle παίρνει τις εξής τιμές:

GLU_POINT: Σχεδιάζονται μόνο οι κορυφές των επιφανειών
GLU_LINE: Σχεδιάζεται το πλέγμα της επιφανείας
GLU_FILL: Οι επιφάνειες του αντικειμένου σχεδιάζονται συμπαγείς.

Κατασκεύη σφαίρας

```
void gluSphere( gluNewQuadric(), GLdoule radius, GLint slices, Glint  stacks );
```

Η παράμετρος radius ορίζει την ακτίνα της βάσης της σφαίρας
Η παράμετρος slices καθορίζει το πλήθος των οριζοντίων υποδιαιρέσεων
Η παράμετρος stacks καθορίζει το πλήθος των κατακόρυφων υποδιαιρέσεων

```
GLUquadric * Sphere;
Sphere = gluNewQuadric();                       Αρχικοποίηση του αντικειμένου sphere
gluQuadricDrawStyle(Sphere, GLU_FILL);          Τρόπος  αναπαράστασης
gluSphere(Sphere,10,60,60);                     Κατασκεύη της σφαίρας
```

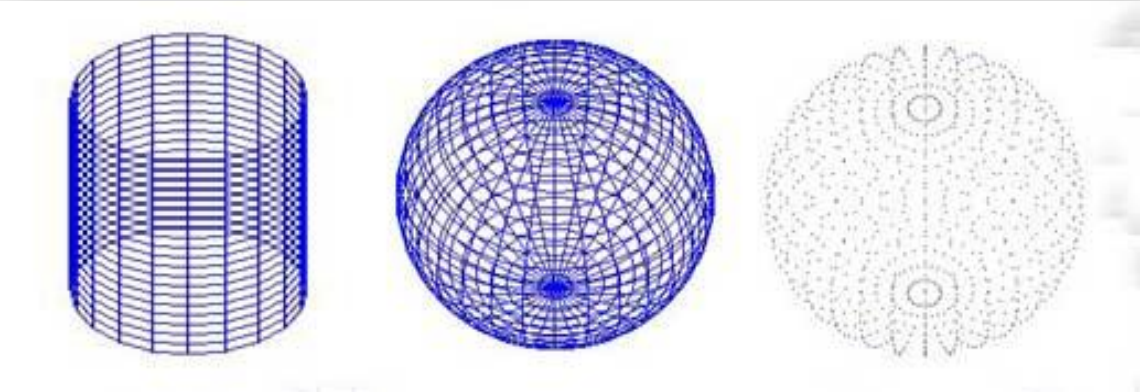
Κατασκεύη κύλινδρου

```
void gluCylinder( gluNewQuadric(),  GLdoule baseRadius, GLdoule topRadius, GLint
height, GLint slices, Glint  stacks);
```

Η παράμετρος baseRadius ορίζει την ακτίνα της βάσης του κυλίνδρου
Η παράμετρος topRadiusορίζει την ακτίνα της βάσης του κυλίνδρου
Η παράμετρος height ορίζει το υψος του κυλίνδρου
Η παράμετρος slices καθορίζει το πλήθος των οριζοντίων υποδιαιρέσεων
Η παράμετρος stacks καθορίζει το πλήθος των κατακόρυφων υποδιαιρέσεων

```
GLUquadric * cylinder;
cylinder = gluNewQuadric();                     Αρχικοποίηση του αντικειμένου cylinder
gluQuadricDrawStyle(cylinder, GLU_FILL);        Τρόπος  αναπαράστασης
gluCylinder (cylinder, 0.5, 0.5, 2, 20, 20);    Κατασκεύη του κυλίνδρου
```

Σχήμα 4.1. Η gluCylinder και gluSphere με τρόπο αναπαράστασης GLU_LINE και GLU_POINT



3.3. Εντολές της βιβλιοθήκης GL (Κατασκεύη επιφανειών στον τρισδιάστατο χώρο)

Οι εντολές της βιβλιοθήκης GL παρέχουν έναν εντελώς διαφορετικό και πιο σύνθετο τρόπο απεικόνισης τρισδιάστατων αντικειμένων σε σχέση με τις δύο παραπάνω βιβλιοθήκες.Η GLUT και GLU παρέχουν έναν αυτοματοποιημένο τρόπο αναπαράστασης των αντικειμένων κάτι που κάνει την ζώη του προγραμματιστή πιο εύκολη αφού ο σχεδιασμός ενός απλού γεωμετρικού σχήματος γίνεται απλά καλόντας την αντίστοιχη εντολή και το σχήμα αυτόματα προβάλεται στην οθόνη. Αντίθετα η GL βασίζεται στην δήλωση των συντεταγμένων των κορυφών της επιφάνειας (*βλέπε 2.5 Σχεδιασμός γεωμετρικών σχημάτων και 3.3.1.* Κατασκεύη επιφανειών *).*

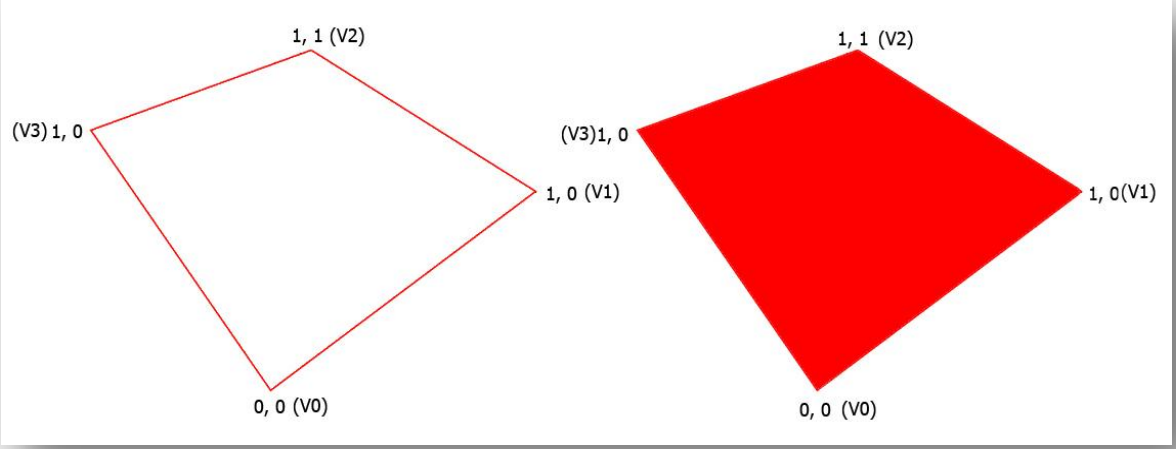
3.3.1. Κατασκεύη επιφανειών

Κατασκευή επιφάνειας με την GL_QUADS

```
glColor3i(1, 0, 0);             καθορισμός χρώματος ως κόκκινο
glBegin(GL_QUADS);              αρχικοποίηση της GL_QUADS
glVertex3f( 0,  0,  1);         κάτω αριστερή γωνία της επιφάνειας
glVertex3f( 0,  10, 1);         κάτω δεξιά γωνία της επιφάνειας
glVertex3f( 10, 10, 1);         πάνω δεξιά γωνία της επιφάνειας
glVertex3f( 10, 0,  1);         πάνω αριστερή γωνία της επιφάνειας
glEnd();                        τέλος της GL_QUADS
```

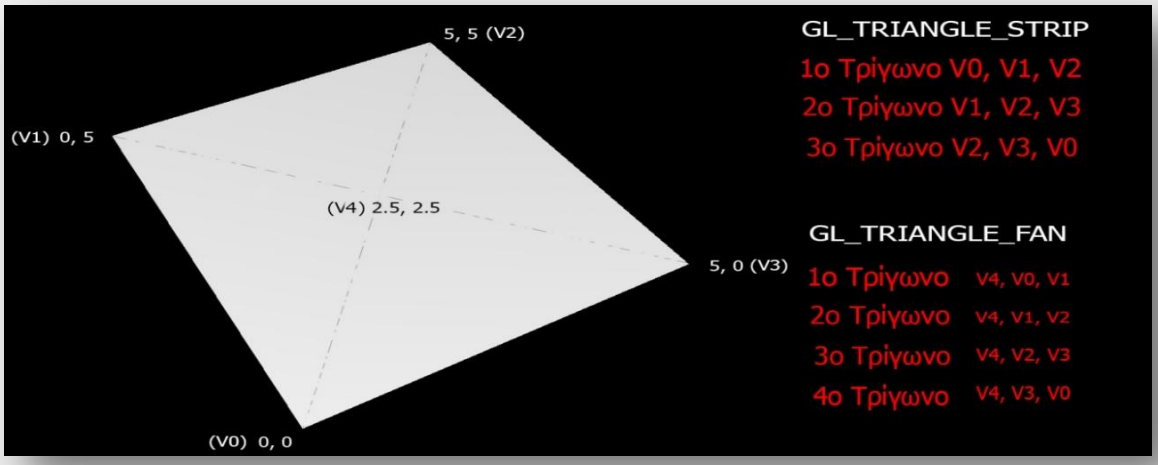
Κατασκευή του περιγράμματος της παραπάνω επιφάνειας με την GL_LINE_LOOP

```
glColor3i(1, 0, 0);                Καθορισμός χρώματος ως κόκκινο
glBegin(GL_LINE_LOOP);             αρχικοποίηση της GL_LINE_LOOP
glVertex3f( 0,  0,  1);            κάτω αριστερή γωνία της επιφάνειας --V0
glVertex3f( 0,  10, 1);            κάτω δεξιά γωνία της επιφάνειας --V1
glVertex3f( 10, 10, 1);            πάνω δεξιά γωνία της επιφάνειας --V2
glVertex3f( 10, 0,  1);            πάνω αριστερή γωνία της επιφάνειας --V3
glEnd();                           τέλος της GL_LINE_LOOP
```



Κατασκευή επιφάνειας με την GL_TRIANGLE_STRIP και GL_TRIANGLE_FAN

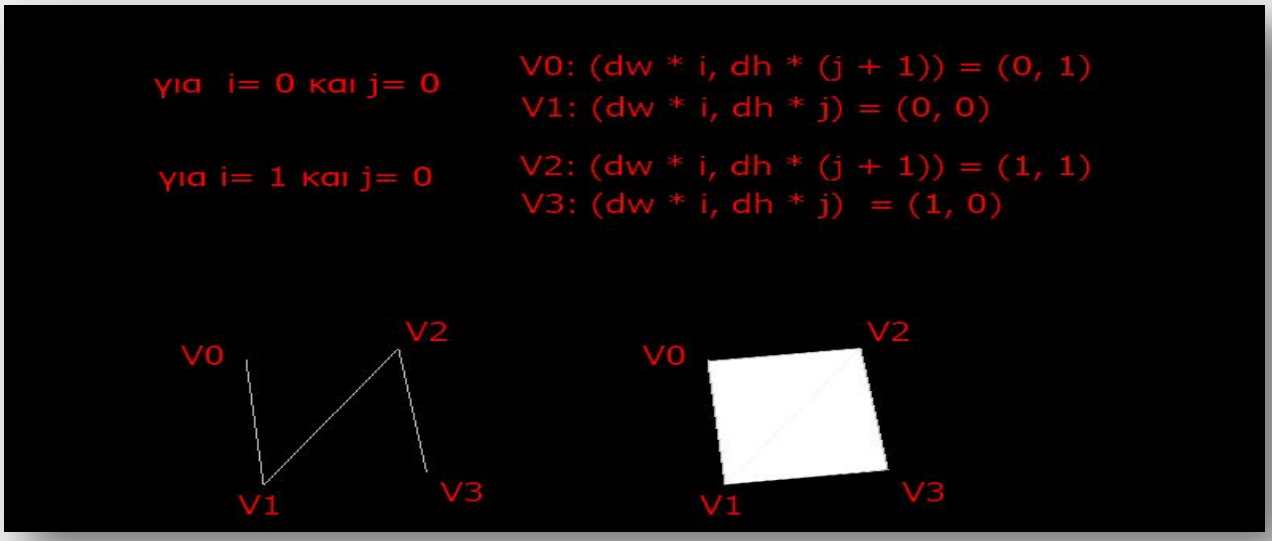
```
glColor3f(1, 1, 1);                  καθορισμός χρώματος ως λευκό
glBegin(GL_TRIANGLE_STRIP);          αρχικοποίηση της GL_TRIANGLE_STRIP
glVertex3f( 0,  0,  1);              1ο σημείο του τριγώνου – V0
glVertex3f( 0,  5,  1);              2ο σημείο του τριγώνου – V1
glVertex3f( 5,  5,  1);              3ο σημείο του τριγώνου – V2
glVertex3f( 5,  0,  1);              4ο σημείο του τριγώνου – V3
glVertex3f( 0,  0,  1);              5ο σημείο του τριγώνου – V4
glEnd();                             τέλος της GL_TRIANGLE_STRIP

glColor3f(1, 1, 1);                   καθορισμός χρώματος ως λευκό
glBegin(GL_TRIANGLE_FAN);             αρχικοποίηση της GL_TRIANGLE_FAN
glVertex3f( 2.5, 2.5, 1.0);           1ο σημείο του τριγώνου – V0
glVertex3f( 0,  0,  1);               2ο σημείο του τριγώνου – V1
glVertex3f( 0,  5,  1);               3ο σημείο του τριγώνου – V2
glVertex3f( 5,  5,  1);               4ο σημείο του τριγώνου – V3
glVertex3f( 5,  0,  1);               5ο σημείο του τριγώνου – V4
glVertex3f( 0,  0,  1);               6ο σημείο του τριγώνου – V5
glEnd();                              τέλος της GL_TRIANGLE_FAN
```



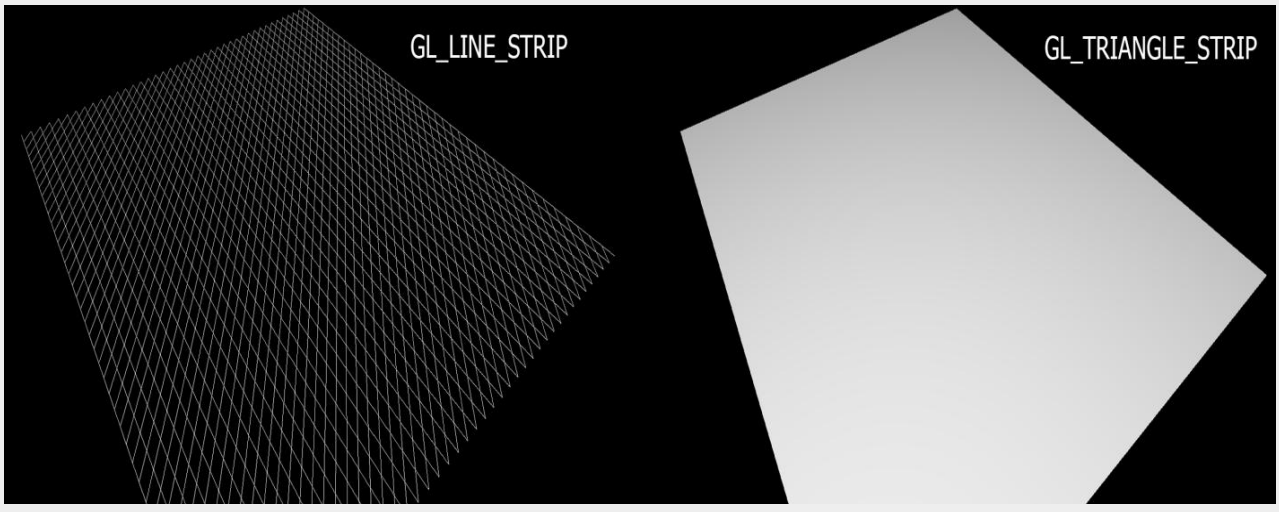
Αλγόριθμος κατασκευής επιφάνειας με την GL_TRIANGLE_STRIP και GL_LINE_STRIP

```
void b(int w, int h)    αρχικοποίηση αντικειμένου b με ορίσματα το πλάτος(w) και το ύψος(h) του κάθε τριγώνου
{
  int i, j;         στην i και j αποθηκεύονται οι συντεταγμένες των κορυφών του κάθε τριγώνου στον άξονα x, y
                                                                                                  αντίστοιχα

  int ve= 1;                                                  με την ve ορίζουμε το πλήθος των καθέτων τριγώνων
  int hor= 1;                                               με την hor ορίζουμε το πλήθος των οριζοντίων τριγώνων
  float dw = 1.0 / w;           το dw είναι το νέο πλάτος του κάθε τριγώνου όπως προκύπτει απο την πράξη 1/ w
  float dh = 1.0 / h;             το dh είναι το νέο ύψος του κάθε τριγώνου όπως προκύπτει απο την πράξη 1/ h

  for (j = 0; j < ve; ++j)
  {
    glBegin(GL_TRIANGLE_STRIP); ή glBegin(GL_LINE_STRIP);                                        αρχικοποίηση
    for (i = 0; i <= hor; ++i)
    {
      glVertex2f(dw * i, dh * (j + 1));
      glVertex2f(dw * i, dh * j);
    }
    glEnd();                                                                                        τέλος
  }
}
```

Σχήμα 4.2: Υλοποίηση του παραπάνω κώδικα με τις GL_LINE_STRIP και GL_TRIANGLE_STRIP



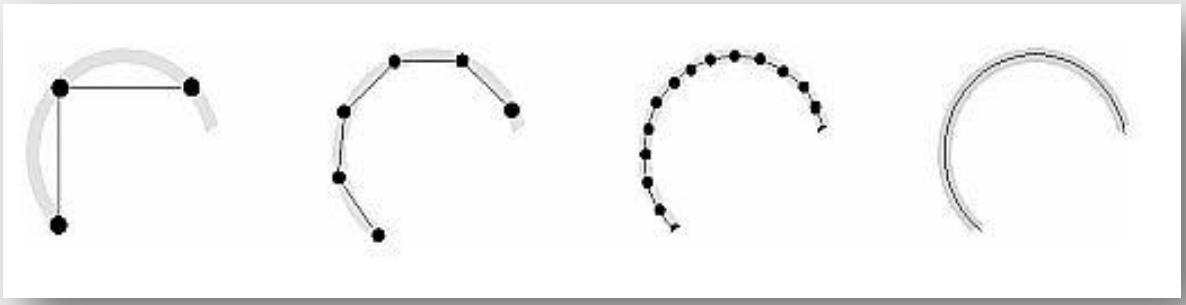
Σχήμα 4.3: Υλοποίηση του παραπάνω κώδικα για ve = 40 και hor = 40



3.3.1. Καμπύλες και σχεδίαση συμπαγών κύκλων

Οποιαδήποτε καμπύλη γραμμή ή επιφάνεια μπορεί να προσεγγιστεί από στοιχειώδη ευθύγραμμα τμήματα ή από ένα πολυγωνικό πλέγμα αντίστοιχα. Για το λόγο αυτό, με επαρκή δειγματοληψία, μπορούμε να υποδιαιρέσουμε καμπύλες γραμμές και επιφάνειες σε επιμέρους ευθύγραμμα τμήματα ή επίπεδα πολύγωνα (Σχήμα. 2.3)

Σχήμα 2.3 Προσεγγίζοντας καμπύλες



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 .

Εξισωση 1: Υπολογισμος των συντεταγμενων περιφερειας κυκλου

$$x = x_c + r * \cos\theta$$
$$y = y_c + r * \sin\theta$$
$$0 \le \theta \le 2\pi$$

όπου , x_c και y_c οι συντεταγμένες του κέντρου του κύκλου και r η ακτίνα του.

Κώδικας 1: Κατασκευή κύκλου

Αρχικά δηλώνουμε τις απαραίτητες μεταβλητές

```
double xc = 0.0;            // Το πρώτο σημείο του τρίγωνου κατά τον άξονα χ
double yc = 0.0;            // Το πρώτο σημείο του τρίγωνου κατά τον άξονα y
double y1s = 0.5;           // Το δεύτερο σημείο του τρίγωνου κατά τον άξονα χ
double x1s = 0.5;           // Το δεύτερο σημείο του τρίγωνου κατά τον άξονα y
double r = 0.5;             // Η ακτίνα του κύκλου
```

Η κατασκευή του κύκλου γίνεται με την κατασκεύη τριγώνων με κοινή κορυφή Α το κέντρο του κύκλου και έχοντας τις άλλες δυο πλευρές Β και Γ σε σημεία της περιφέρειας του κύκλου με την χρήση της GL_TRIANGLES.

```
glBegin(GL_TRIANGLES)
```

Δημιουργούμε 360 τρίγωνα με κοινή κορυφή την $A(\chi_c,y_c)$ (σχ. 2.4)

```
for(int i=0;i<=360;i++)
      {
```

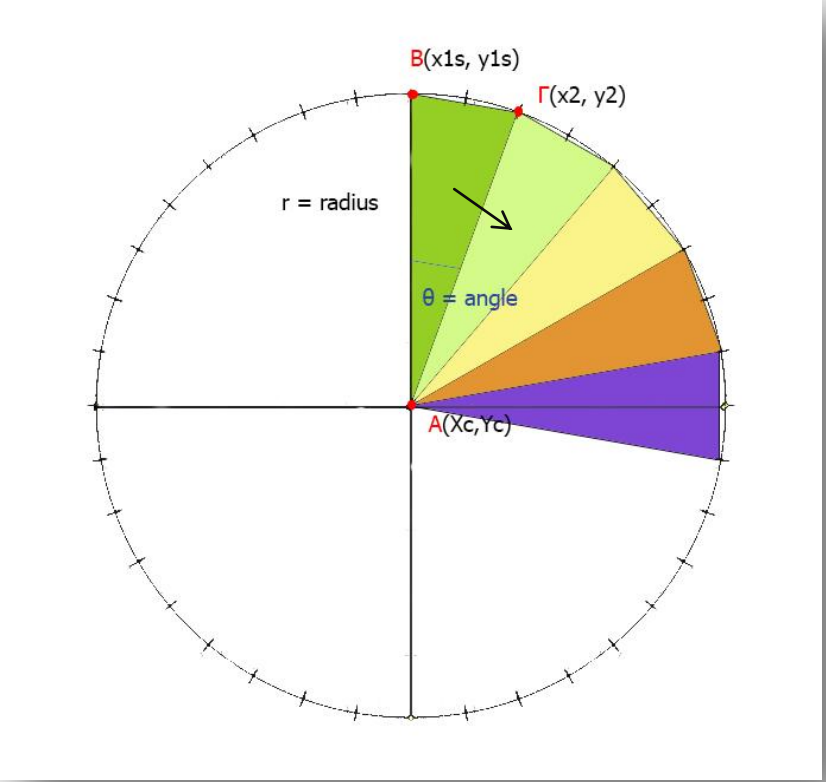
Υπολογισμός των συντεταγμένων της περιφέρειας του κύκλου με βάση τις παραπάνω δηλωθείσες μεταβλητές για εύρεση της πλευράς ΒΓ του τρίγωνου.

```
double angle=(float)(((double)i)/57.29577957795135);     Η γωνία θ της κορυφής Α ολων των τριγώνων
double y2= yc +(r *(float)cos((double)angle));        Το τρίτο σημείο του τρίγωνου κατά τον άξονα y
double x2= xc +(r *(float)sin((double)angle));        Το τρίτο σημείο του τρίγωνου κατά τον άξονα χ
```

Κατασκευή τριγώνου έχοντας και τις τρείς κορυφές του.

```
glVertex2d(xc, yc);                  Οι συντεταγμένες του κέντρου του κύκλου και κορυφή Α
glVertex2d(x1s,y1s);                 Η κορυφή Β
glVertex2d(x2,y2);                   Η κορυφή Γ

y1s=y2;                              Η κορυφή Β παίρνει τις τιμές της κορυφής Γ και η νέα Γ υπολογίζεται
x1s=x2;                              από την παραπάνω ρουτίνα

}
glEnd();
```

Σχήμα 2.4: Κατασκεύη Κύκλου



CHAPTER IV

ΑΠΟΔΟΣΗ ΤΡΙΣΔΙΑΣΤΑΤΩΝ ΣΚΗΝΩΝ

Όταν δηλώνουμε τη θέση ενός σημείου χρησιμοποιώντας τις εντολές σχεδίασης *glVertex*,οι συντεταγμένες που αποδίδουμε στα σημεία, δεν αντιστοιχούν στις θέσεις των pixels στην οθόνη. Αντίθετα ορίζονται σε ένα σύστημα με απεριόριστο εύρος , στο ***σύστημα συντεταγμένων σκηνής (world coordinate system).***
Ωστόσο, όταν τα δηλούμενα σημεία προωθούνται προς απεικόνιση στην οθόνη, λόγω της πεπερασμένης αναλυτικότητάς της τελευταίας, για κάθε σχεδιαζόμενο σημείο, θα πρέπει να υπολογιστούν οι ακέραιες συντεταγμένες που καθορίζουν την αντίστοιχή του θέση στην επιφάνεια σχεδίασης. Υπολογίζουμε δηλαδη τις λεγόμενες ***συντεταγμένες συσκευής (device coordinate system).***

4.1. Αποκοπή στις δύο διαστάσεις

Με τον όρο αποκοπή εννοούμε τη διαδικασία κατά την οποία απομονώνουμε ένα τμήμα της σκηνικού για την αναπαράστασή του στη συσκευή εξόδου. Ουσιαστικά η απομόνωση αυτή στις δύο διαστάσεις επιτελείται ορίζοντας ως προς το σύστημα συντεταγμένων σκηνής τα όρια του παραθύρου αποκοπής ***(clipping window).*** Με τον όρο παράθυρο αποκοπής εννοούμε ένα ορθογώνιο, μέσα στο οποίο περικλείεται το τμήμα της σκηνής που θέλουμε να απομονώσουμε. Ο καθορισμός του παραθύρου αποκοπής σε δύο διαστάσεις γίνεται χρησιμοποιώντας την εντολή *gluOrtho2D* της βιβλιοθήκης GLU:

```
void gluOrtho2D(GLdouble  left, GLdouble right, GLdouble bottom, GLdouble top);
```

Οι left, right καθορίζουν τις συντεταγμένες για το αριστερό και το δεξί κατακόρυφο επίπεδο αποκοπής.
Οι bottom, top καθορίζουν τις συντεταγμένες για το κάτω και το πάνω οριζόντιο επίπεδο αποκοπής.

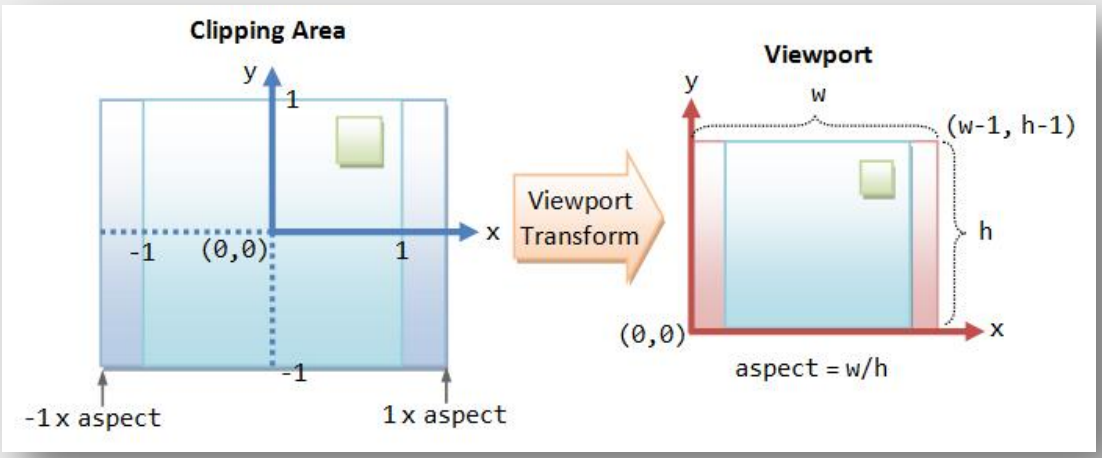
4.2. Παράθυρο παρατήρησης

Στην OpenGL, αντί να χρησιμοποιήσουμε ολόκληρη τη διαθέσιμη επιφάνεια σχεδίασης της συσκευής εξόδου για την αναπαράσταση μιας σκηνής, μπορούμε να καθορίσουμε το τμήμα της επιφάνειας σχεδίασης (τα όρια των ακεραίων συντεταγμένων συσκευής [ ] minmax , xuxu και) μέσα στο οποίο θέλουμε να σχεδιαστούν τα περιεχόμενα του παραθύρου αποκοπής Ο καθορισμός του εύρους των συντεταγμένων συσκευής που θα διατεθούν για τη σχεδιάση της αποκομμένης σκηνής γίνεται με τον καθορισμό ενός παραθύρου παρατήρησης (viewport). Ο καθορισμός των διαστάσεων ενός παραθύρου παρατήρησης γίνεται χρησιμοποιώντας την εντολή glViewport:

```
void glViewport(GLint  x, GLint  y ,GLsizei  width, GLsizei  height);
```

Οι x, y καθορίζουν την κάτω αριστερή γωνία του ορθογώνιου παραθύρου προβολής, σε εικονοστοιχεία. Η αρχική τιμή είναι (0,0)
Οι width, height καθορίζουν το πλάτος και το ύψος του παραθύρου προβολής. Όταν ένα πλαίσιο GL συνδέεται για πρώτη φορά με ένα παράθυρο,το πλάτος και το ύψος καθορίζονται με τις διαστάσεις του εν λόγω παράθυρου.

Σχήμα 3.2. Η glViewport



Η glViewport καθορίζει τη θέση του παραθύρου παρατήρησης στην επιφάνεια σχεδίασης και δηλώνεται πάντοτε μέσα στον κώδικα της συνάρτησης κλήσης που διαχειρίζεται το γεγονός reshape (που καταχωρείται στην glutReshapeFunc). Οι συντεταγμένες που δίνουμε ως ορίσματα στην glViewPort είναι ακέραιες συντεταγμένες συσκευής.

4.3. Απεικόνιση τρισδιάστατων σκηνών

Η απεικόνιση τρισδιάστατων σκηνών καθορίζεται, στη γενικότερη περίπτωση, από την οπτική γωνία από την οποία ο θεατής παρατηρεί τη σκηνή, καθώς και από τον τύπο προβολής που επιλέγουμε για την αναπαράσταση της σκηνής. Με αλλαγή της θέσης και του προσανατολισμού της κάμερας, προκύπτει διαφορετική αναπαράσταση της σκηνής στο διδιάστατο επίπεδο της συσκευής εξόδου. Η δυνατότητα αλλαγής της οπτικής γωνίας προσφέρεται με το μετασχηματισμό οπτικής γωνίας και μας επιτρέπει την επισκόπηση της σκηνής από πολλαπλές θέσεις. Επιπλέον, υπάρχει η δυνατότητα χρήσης διαφορετικών τύπων προβολής της τρισδιάστατης σκηνής, ανάλογα με την επιθυμητή απόδοση της σκηνής.

Η ***παράλληλη προβολή*** διατηρεί τις αναλογίες των σχημάτων και είναι χρήσιμη για την αναπαράσταση σχεδίων και γενικά για εφαρμογές που η διατήρηση της κλίμακας έχει σημασία.

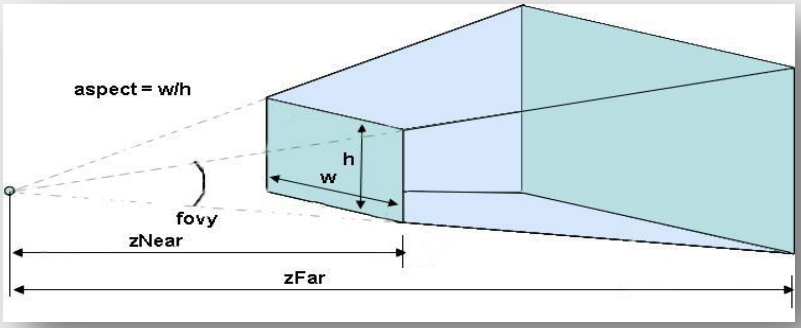
Η ***προοπτική προβολή*** αποσκοπεί στην απόδοση ρεαλιστικών σκηνών,ακολουθώντας τους κανόνες οπτικής που ακολουθούν οι κάμερες και το ανθρώπινο μάτι.

4.3.1. Προοπτική προβολή (Prespective View)

Η προοπτική απεικόνιση τρισδιάστατων σκηνών γίνεται με την χρήση της **gluPerspective** και της **glFrustrum**

```
void gluPerspective(GLdouble fovy, GLdouble aspect,GLdouble zNear,GLdouble zFar);
```

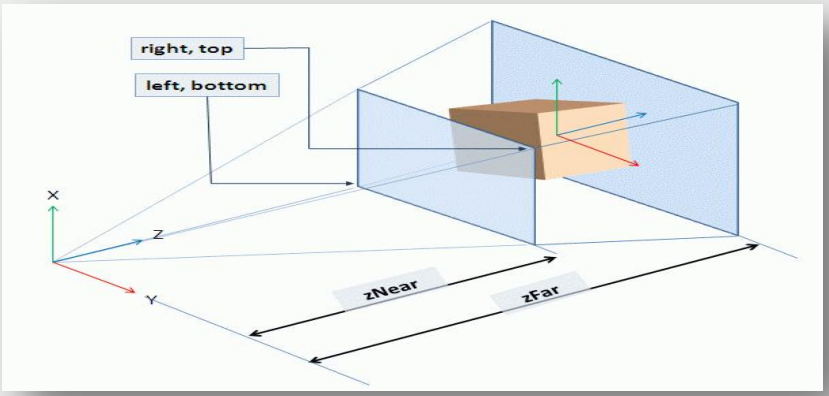
Σχήμα 3.1. Η gluPerspective (προοπτική προβολή)



Η *Fovy* καθορίζει το πεδίο της οπτική γωνία, σε μοίρες, στην κατεύθυνση Y.
Η *Aspect* καθορίζει την αναλογία διαστάσεων που καθορίζει το οπτικό πεδίο κατά τη κατεύθυνση X. Η αναλογία διαστάσεων είναι ο λόγος του X (πλάτος) προς y (ύψος).
Η *zNear* καθορίζει την απόσταση από το θεατή μέχρι το κοντινότερο επίπεδο αποκοπής (πάντα θετικό).
Η *zFar* καθορίζει την απόσταση από το θεατή μέχρι το μακρύτερο επίπεδο αποκοπής (πάντα θετικό).

```
void glFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble
nearVal,  GLdouble  farVal);
```

Σχήμα 3.2. Η glFrustum (προοπτική προβολή)



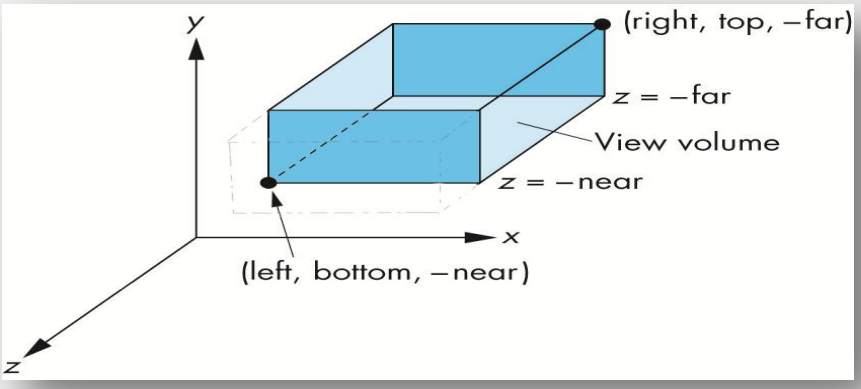
Η *left, right* καθορίζει τις συντεταγμένες για το αριστερό και το δεξί κατακόρυφο επίπεδο αποκοπής.
Η *bottom, top* καθορίζει τις συντεταγμένες για το κάτω και το πάνω οριζόντιο επίπεδο αποκοπής.
Η *nearVal, farVal* καθορίζει τις αποστάσεις απο την εγγύτερο στο μακρύτερο παράθυρο αποκοπής. Οι τιμές αυτές είναι αρνητικές εάν το παράθυρο είναι πίσω από το θεατή.

4.3.2. Παράλληλη προβολή (Orthographic View)

Η παράλληλη απεικόνιση τρισδιάστατων σκηνών γίνεται με την χρήση της **glOrtho**

```
void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble nearVal, GLdouble farVal);
```

Σχήμα 3.3. Η glOrtho (Παράλληλη προβολή)



Η *left, right* καθορίζει τις συντεταγμένες για το αριστερό και το δεξί κατακόρυφο επίπεδο αποκοπής.
Η *bottom, top* καθορίζει τις συντεταγμένες για το κάτω και το πάνω οριζόντιο επίπεδο αποκοπής.
Η *nearVal, farVal* καθορίζει τις αποστάσεις απο την εγγύτερο στο μακρύτερο παράθυρο αποκοπής. Οι τιμές αυτές είναι αρνητικές εάν το παράθυρο είναι πίσω από το θεατή.

4.4. Μετασχηματισμός οπτικής γωνίας

Σε δυσδιάστατες σκηνές κατά την εξέταση των αλγορίθμων αποκοπής και μετασχηματισμού παρατήρησης, το επίπεδο παρατήρησης ταυτίζεται με το επίπεδο XY. Ωστόσο, σε τρισδιάστατες σκηνές, η ταύτιση αυτή δεν είναι υποχρεωτική. Απεναντίας, υπάρχει η δυνατότητα αλλαγής της οπτικής γωνίας από την οποία παρατηρούμε τη σκηνή. Αυτή η δυνατότητα μας επιτρέπει την επισκόπηση της σκηνής από πολλαπλές θέσεις.
Στο μετασχηματισμό οπτικής γωνίας θεωρούμε το σύστημα ***συντεταγμένων παρατηρητή***. Η θέση αυτού του συστήματος συντεταγμένων καθορίζεται από δύο παράγοντες:

- Από τη θέση του παρατηρητή: Η αρχή του συστήματος συντεταγμένων παρατηρητή ταυτίζεται με το σημείο παρατήρησης.

- Από την κατεύθυνση παρατήρησης: Η κατεύθυνση παρατήρησης ταυτίζεται με τον αρνητικό ημιάξονα z του συστήματος συντεταγμένων παρατηρητή.

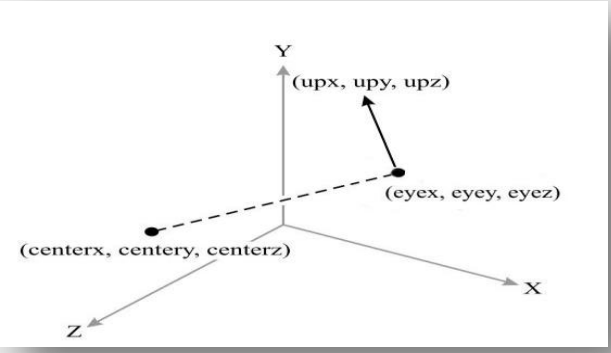
Η θέση της κάμερας στη σκηνή προσδιορίζεται με τις συντεταγμένες ενός σημείου, έστω ***(eyeX, eyeY, eyeZ).*** Ο προσανατολισμός της κάμερας προσδιορίζεται μ ε δύο διανύσματα: το ***διάνυσμα κατεύθυνσης παρατήρησης*** και το ***διάνυσμα άνω κατεύθυνσης (view-up vector).*** Το πρώτο διάνυσμα δηλώνει την κατεύθυνση προς την οποία είναι προσανατολισμένος ο παρατηρητής. Ταυτίζεται με τον αρνητικό ημιάξονα των z του συστήματος συντεταγμένων του παρατηρητή και η αρχή του βρίσκεται τη θέση του παρατηρητή. Το διάνυσμα άνω κατεύθυνσης δηλώνει την προς τα πάνω κατεύθυνση του επιπέδου προβολής.

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

```
void 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.

Σχήμα 3.4. Η gluLookAt



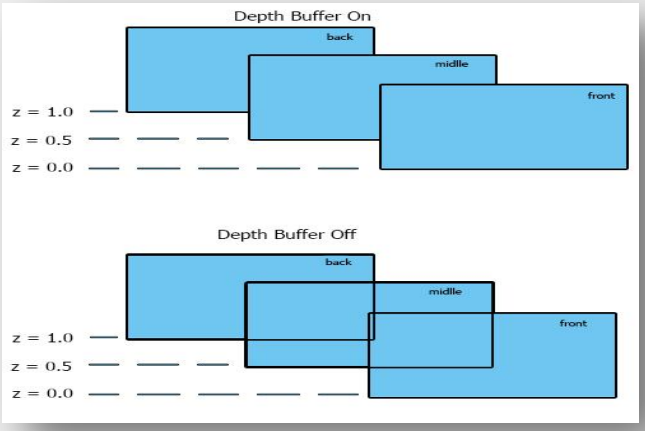
4.5. Καταστολή κρυμμένων επιφανειών (The depth Buffer)

Κατά τη σχεδίαση επιφανειών στον τρισδιάστατο χώρο, ένας προφανής κανόνας που πρέπει να τηρείται είναι το ότι, οι επιφάνειες που βρίσκονται πλησιέστερα στον παρατηρητή καλύπτουν τις επιφάνειες που βρίσκονται από πίσω τους. Ωστόσο, η OpenGL, στην προκαθορισμένη εξ' αρχής κατάσταση λειτουργίας, δε λαμβάνει υπόψη την πληροφορία βάθους, παρά μόνο εάν αυτό δηλωθεί ρητά από τον προγραμματιστή. Επομένως, εάν σχεδιαστούν δύο επιφάνειες που βρίσκονται σε διαφορετικό βάθος και οι προβολές τους επικαλύπτονται, υπάρχει η πιθανότητα, η επιφάνεια που βρίσκεται πλησιέστερα στον παρατηρητή να καλυφθεί από την επιφάνεια που βρίσκεται μακρύτερα. Αυτό εξαρτάται από τη διαδοχή μ ε την οποία δηλώνονται τα σχήματα στον κώδικα του προγράμματος. Εάν το μακρινότερο σχήμα δηλωθεί δεύτερο, η προβολή του θα επικαλύψει την προβολή του αρχικά σχεδιασμένου πλησιέστερου σχήματος, κάτι που φυσικά είναι ανεπιθύμητο. Η δήλωση των σχημάτων με τη σειρά, από το πιο απομακρυσμένο προς το πλησιέστερο, δεν αποτελεί λύση, γιατί στην περίπτωση που θα εφαρμοστούν μετασχηματισμοί οπτικής γωνίας, η ορατότητα ή μη των επιφανειών θα μεταβάλλεται.

Στην OpenGL, ο έλεγχος της ορατότητας επιφανειών γίνεται μ ε τη χρήση του ενταμιευτή βάθους (***depth buffer ή z-buffer)***. Πρόκειται για ένα μητρώο με διαστάσεις ίδιες με τις διαστάσεις της επιφάνειας σχεδίασης σε pixels. Σε κάθε στοιχείο του ενταμιευτή βάθους αποθηκεύεται η συντεταγμένη z της επιφανείας που βρίσκεται πλησιέστερα στον παρατηρητή στο αντίστοιχο pixel.

Δεδομένου ότι στο στάδιο του τρισδιάστατου μετασχηματισμού παρατήρησης οι τιμές βάθους κανονικοποιούνται στο εύρος τιμών [0,1], τα πιο μακρινά σημεία βρίσκονται επί του μακρινού επιπέδου αποκοπής στη σκηνή και μετά τον τρισδιάστατο μετασχηματισμό παρατήρησης έχουν συντεταγμένες βάθους z=1. Τα πιο κοντινά σημεία βρίσκονται στο εγγύς επίπεδο αποκοπής και έχουν τιμή . Συνεπώς, η OpenGL μπορεί να εντοπίσει την επιφάνεια που είναι ορατή σε κάθε pixel της επιφάνειας σχεδίασης, βρίσκοντας την επιφάνεια που έχει τη μικρότερη συντεταγμένη βάθους στο εκάστοτε pixel, όπως φαίνεται στο *Σχ. 3.5*

Σχήμα 3.5. Αρχή λειτουργίας του ενταμιευτή βάθους



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

Επιπλέον, θα πρέπει, στη συνάρτηση display, πριν το σχεδιασμό ή επανασχεδιασμό ενός καρέ, να αρχικοποιείται ο ενταμιευτής τιμών βάθους με την εντολή glClear. *glClear(GL_DEPTH_BUFFER_BIT);*

CHAPTER V

ΜΕΤΑΣΧΗΜΑΤΙΣΜΟΙ ΣΥΝΤΕΤΑΓΜΕΝΩΝ

Στο 2ο και 3ο Κεφάλαιο αναλύσαμε βασικές εντολές σχεδίασης, μέσω των οποίων ο προγραμματιστής μπορεί να σχεδιάσει γεωμετρικά σχήματα στις επιθυμητές συντεταγμένες της σκηνής. 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. Ουσιαστικά επιθυμούμε, με τον ορισμό ενός σημείου με συντεταγμένες x, y, z και βάσει ενός κανόνα αντιστοίχισης, το σημείο να αποδοθεί σε μια θέση της σκηνής με συντεταγμένες x', y', z'. Στο κεφάλαιο αυτό αναλύεται η λογική που ακολουθεί η μηχανή της OpenGL, σε ό,τι αφορά την εφαρμογή μετασχηματισμών συντεταγμένων.

5.1. Συντεταγμένες μοντέλου – Μετασχηματισμός μοντέλου

Η χρησιμότητα των μετασχηματισμών συντεταγμένων αναδεικνύεται στην περίπτωση που χρησιμοποιούμε ένα γεωμετρικό σχήμα που έχει οριστεί από τρίτο προγραμματιστή σε μια display list *(βλέπε κεφάλαιο 10)*.. Στην περίπτωση που ο τρίτος προγραμματιστής κατασκευάζει μια λίστα απεικόνισης, ορίζει ένα σύνθετο σχήμα σε ένα βολικό για αυτόν σύστημα συντεταγμένων. Συνήθως ορίζει το σχήμα κοντά στην αρχή των αξόνων. Το εύρος των συντεταγμένων στο οποίο εκτείνεται το σχήμα επιλέγεται αυθαίρετα ή βάσει της κοινής λογικής. Οι συντεταγμένες με τις οποίες δηλώνεται ένα προτύπο σχήμα σε μια λίστα απεικόνισης συχνά αναφέρονται ως ***συντεταγμένες μοντέλου.*** Επιπλέον, εάν ο χρήστης της λίστας επιθυμεί να επαναχρησιμοποιήσει τον κώδικά της για τη σχεδίαση πολλαπλών σχημάτων στην ίδια σκηνή, θα ήταν αδύνατο να το επιτύχει με απλή εκτέλεση της λίστας απεικόνισης. Κάθε εκτέλεση της λίστας απεικόνισης θα σχεδίαζε το σχήμα στην ίδια (αρχικά καθορισμένη) θέση και με τις ίδιες διαστάσεις που έχουν οριστεί στη λίστα απεικόνισης. Είναι λοιπόν εμφανές ότι σε ορισμένες περιπτώσεις επιθυμούμε να επιβάλλουμε μετασχηματισμούς που μετασχήματίζουν τις συντεταγμένες του πρότυπου σχήματος σε κατάλληλες θέσεις και μ ε κατάλληλες διαστάσεις στη σκηνή.
Ο μετασχηματισμός αυτός ονομάζεται μετασχηματισμός μοντέλου **(modelview transformation)** και εκτελείται ορίζοντας ένα μητρώο μετασχηματισμού το οποίο καθορίζει τις συντεταγμένες σκηνής (x', y', z') που θα αντιστοιχιστούν σε ένα σημείο που έχει δηλωθεί με συντεταγμένες μοντέλου (x, y, z) .

5.2. 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).**

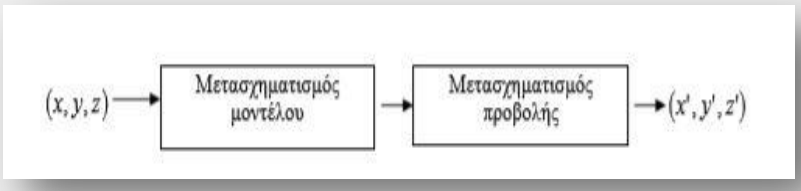
α. **Το μητρώο μετασχηματισμού μοντέλου** σχηματίζεται βάσει των μαθηματικών σχέσεων που καθορίζουν το μετασχηματισμό μοντέλου. Δηλαδή τα στοιχεία του μητρώου έχουν τιμές τέτοιες, ούτως ώστε η δήλωση ενός σημείου με συντεταγμένες (x, y, z) να οδηγεί στην απόδοση του σημείου (x', y', z') στη σχεδιαζόμενη σκηνή βάσει της σχέσης.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \mathbf{M}_{\mathbf{modelview}} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Επιπλέον, 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 αυτή ενσωματώνονται στο μητρώο μετασχηματισμού μοντέλου.

β. **Το μητρώο προβολής** σχηματίζεται από κανόνες αντιστοίχισης που καθορίζουν τον τρόπο με τον οποίο η παρατηρούμενη σκηνή θα προβληθεί στο επίπεδο παρατήρησης του θεατή (επίπεδο προβολής).
Η συνδυασμένη επίδραση των μητρώων μετασχηματισμού μοντέλου και προβολής στις συντεταγμένες των οριζόμενων σημείων, σχηματίζει την εικόνα που τελικά αποδίδεται στην οθόνη του υπολογιστή. Η διαδικασία φαίνεται στο *Σχήμα. 5.1.*

Σχήμα 5.1. Διάγραμμα μετασχηματισμού συντεταγμένων στην OpenGL



Επομένως, στις συντεταγμένες x, y, z πρώτα επιδρά το μητρώο μετασχηματισμού μοντέλου και κατόπιν το μητρώο προβολής .

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = M_{projection} * M_{modelview} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

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);

όπου matrixMode σταθερά που παίρνει μιά από τις τιμές:

GL_MODELVIEW: μετάβαση στην κατάσταση επεξεργασίας του μητρώου μετασχηματισμού μοντέλου
GL_PROJECTION: μετάβαση στην κατάσταση επεξεργασίας του μητρώου προβολής.

Η αρχικοποίηση των μητρώων προβολής και μετασχηματισμού μοντέλου στο μητρώο γίνεται με την εντολή *glLoadIdentity.*

```
void glLoadIdentity ( );
```

Ένα σύνηθες παράδειγμα εναλλαγής των μητρώων προβολής και μετασχηματισμού υπάρχει στον επαναπροσδιορισμό του παραθύρου.

Παράδειγμα 5.1. Η αρχικοποίηση των μητρώων προβολής και μετασχηματισμού

```
GLvoid ReSizeGLScene(GLsizei width, GLsizei height)
{
        if (height==0)
        {
                height=1;
        }
   glViewport(0,0,width,height);
   glMatrixMode(GL_PROJECTION);          μετάβαση στην κατάσταση επεξεργασίας του μητρώου προβολής.
   glLoadIdentity();                     αρχικοποίηση του μητρώου
   gluPerspective(45.0f,(GLfloat)width/(GLfloat)height,0.1f,100000.0f);      επεξεργασία μητρώου προβολής
   glMatrixMode(GL_MODELVIEW);           μετάβαση στην κατάσταση επεξεργασίας του μητρώου μετασχηματισμού.
   glLoadIdentity();                     αρχικοποίηση του μητρώου
}
```

5.3. Μετατόπιση

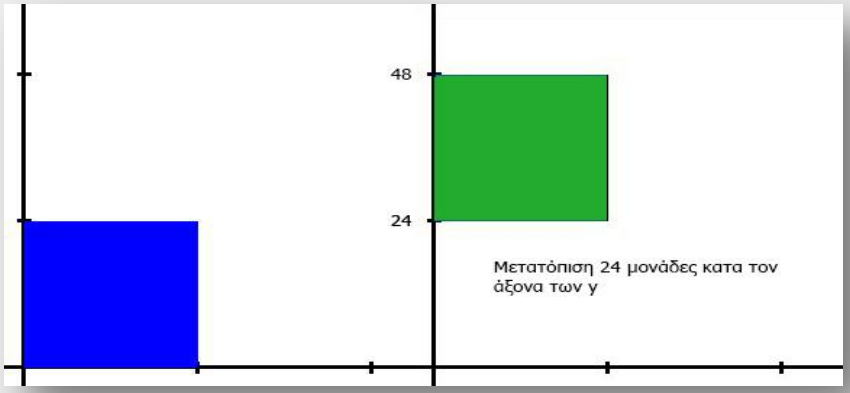
Στην OpenGL η μετατόπιση των συντεταγμένων της σκηνής στο χώρο κατά σταθερές x_{tr}, y_{tr}, z_{tr} εκτελείται με την εντολή ***glTranslate****

glTranslatef (GLfloat x, GLfloat y, GLfloat z);
glTranslated (GLdouble x, GLdouble y, GLdouble z);

Παράδειγμα 5.2. Ο κώδικας της παρακάτω Περιστροφής

```
void display()
{
  ...
  glColor3f(0,1,0);          ορισμός χρώματος το πράσινο
  glTranslatef(0, 24, 0);      μετατόπιση κατα 0,6 μονάδες
  glBegin(GL_QUADS);
  glVertex2f( 0, 0);
  glVertex2f( 0, 24);
  glVertex2f(24, 24);
  glVertex2f(24, 0);
  glEnd();
  ...
 }
```

Σχήμα 5.3 Μετατόπιση



5.4. Κλιμάκωση

Στην κλιμάκωση, οι συντεταγμένες πολλαπλασιάζονται με ένα σταθερό ανά διεύθυνση συντελεστή με σκοπό την αυξομείωση του αρχικού δηλωθέντος μεγέθους των αντικειμένων βάσει των σχέσεων:

$$x' = s_x * x$$

$$y' = s_y * y$$

$$z' = s_z * z$$

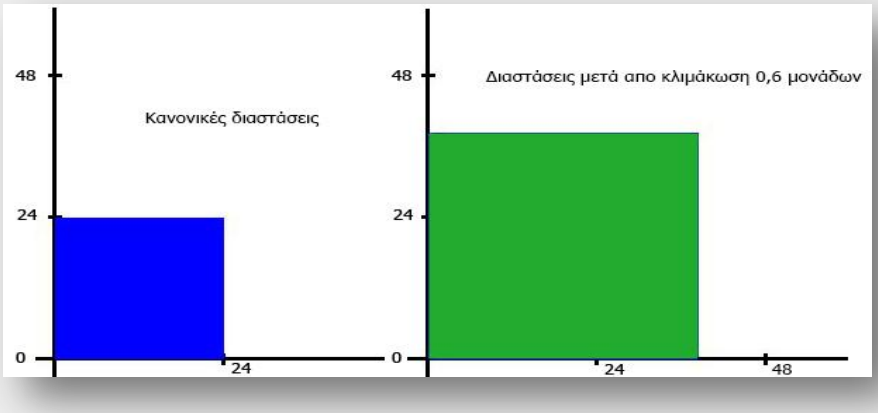
όπου S_x , S_y, S_z οι επιβαλλόμενοι συντελεστές κλιμάκωσης κατά τις διευθύνσεις x, y και z
Στην OpenGL, η κλιμάκωση εκτελείται με την εντολή ***glScale*:***

```
glScalef(GLfloat x, Glfloat  y, Glfloat  z );
glScaled(GLdouble  x, Gldouble  y Gldouble  z );
```

Παράδειγμα 5.3. Ο κώδικας της παρακάτω Κλιμάκωσης

```
void display()
{
  ...
  glColor3f(0,1,0);          ορισμός χρώματος το πράσινο
  glScalef(1.6, 1.6, 1.6);   κλιμάκωση κατα 0,6 μονάδες
  glBegin(GL_QUADS);
  glVertex2f( 0, 0);
  glVertex2f( 0, 24);
  glVertex2f(24, 24);
  glVertex2f(24, 0);
  glEnd();
  ...
 }
```

Σχήμα 5.4. Κλιμάκωση



5.5. Περιστροφή

Η περιστροφή ενός αντικειμένου κατά γωνία φ στις δύο διαστάσεις επί του επιπέδου XY και ως προς άξονα περιστροφής που διέρχεται την αρχή των αξόνων δίνεται από τις σχέσεις:

- $x' = \cos\varphi * x - \sin\varphi * y$
- $y = \sin\varphi * x - \cos\varphi * y$

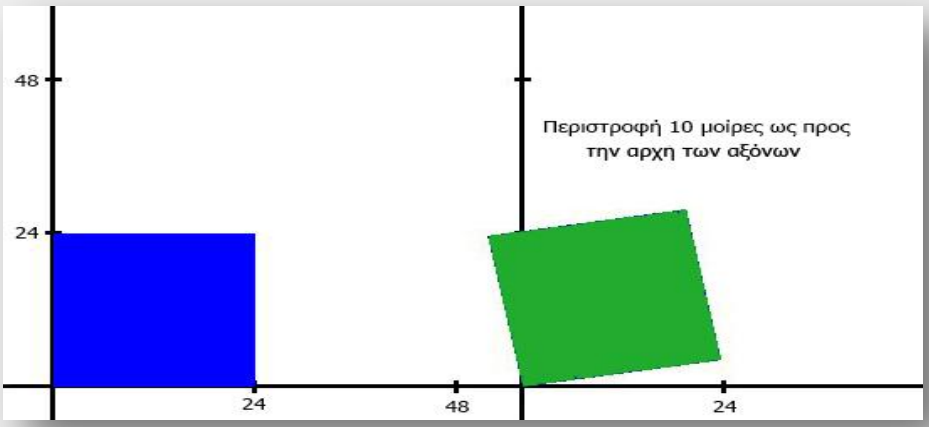
Στην OpenGL μπορούμε να εκτελέσουμε μετασχηματισμούς περιστροφής ως προς οποιαδήποτε άξονα περιστροφής. Αυτό επιτυγχάνεται ορίζοντας τις συνιστώσες του διανύσματος που ορίζει τη διεύθυνση ενός άξονα περιστροφής. Ο άξονας περιστροφής διέρχεται από την αρχή των αξόνων.
Μετασχηματισμοί περιστροφής εκτελούνται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

```
glRotatef (Glfloat angle, GLfloat vx, GLfloat vy, GLfloat vz );
glRotated (GLdouble angle, GLdouble vx, GLdouble vy, GLdouble vz );
```

angle: η γωνία περιστροφής σε μοίρες
vx,vy,vz : οι συνιστώσες του διανύσματος που εκφράζει τη διεύθυνση του άξονα περιστροφής

Παράδειγμα 5.4. Ο κώδικας της παρακάτω Περιστροφής

```
void display()
{
   ...
   glColor3f(0,1,0);          ορισμός χρώματος το πράσινο
   glRotatef(10, 0, 0, 1 );   περιστροφή 10 μοίρες

   glBegin(GL_QUADS);
   glVertex2f( 0, 0);
   glVertex2f( 0, 24);
   glVertex2f(24, 24);
   glVertex2f(24, 0);
   glEnd();
   ...
}
```



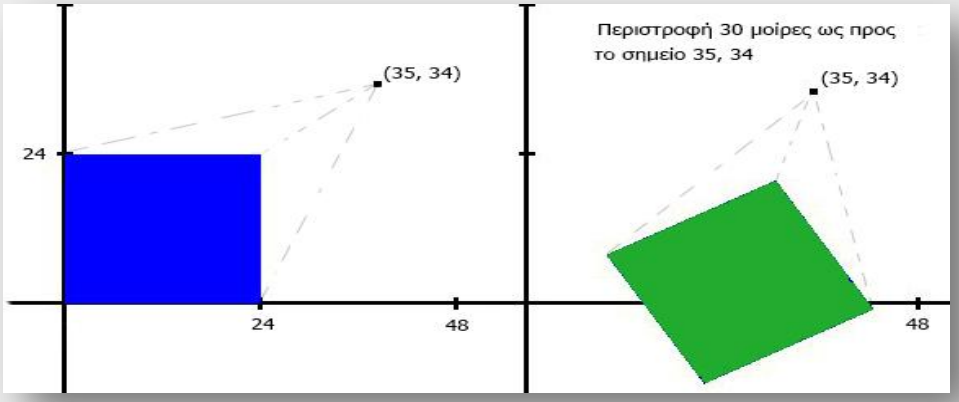
Σχήμα 5.4. Περιστροφή στις δύο διαστάσεις ως προς την αρχή των αξόνων

5.5.1. Περιστροφή ως προς ένα σταθερό σημείο

Η περιστροφή ενός αντικειμένου κατά γωνία φ στις δύο διαστάσεις επί του επιπέδου XY ως προς ένα σταθερό σημείο στο χώρο είναι πιο πολύπλοκο από μια απλή περιστροφή ως προς τους άξονες αφού πρώτα πρέπει να δηλώσουμε το σημείο και ύστερα να περιστρέψουμε το αντικείμενο ως προς το σημείο αυτό.
Στην OpenGl αυτό είναι εφικτό με τον συνδυασμό μετατόπισης – περιστροφής.
Αρχικά μετατοπίζουμε το αντικείμενο που θέλουμε να περιστρέψουμε στο σημείο περιστροφής , περιστρέφουμε το αντικείμενο και στην συνέχεια μετατοπίζουμε το αντικείμενο πίσω στην αρχική του θέση.

Παράδειγμα 5.4. Ο κώδικας της παρακάτω Περιστροφής

```
void display()
{
   ...
   glTranslatef(35, 34, 0);        μεττόπιση στο σημείο περιστροφής
   glRotatef(30, 0, 0, 1 );        Περιστροφή κατά 30 μοίρες
   glTranslatef(-35, -34, 0);      μεττόπιση πίσω στην αρχική θέση

   // do the drawing
   ...
       }
```



Σχήμα 5.5. Περιστροφή στις δύο διαστάσεις ως προς ένα σταθερό σημείο

5.6. Πολλαπλασιασμός τρέχοντος μητρώου μετασχηματισμού με αυθαίρετο μητρώο

Στην OpenGL ο προγραμματιστής έχει την ευχέρεια να παρακάμψει τις παραπάνω συναρτήσεις δήλωσης μετασχηματισμών και να τροποποιήσει απευθείας τις τιμές του τρέχοντος μητρώου μετασχηματισμού. Έτσι έχει τη δυνατότητα να συνθέσει μετασχηματισμούς που δε μπορούν να εκφραστούν από το συνδυασμό των ανωτέρω εντολών.

Στην περίπτωση που θέλουμε να πολλαπλασιάσουμε το τρέχον μητρώο με ένα αυθαίρετο μητρώο για την επίτευξη κάποιου μετασχηματισμού , χρησιμοποιούμε την εντολή **glMultMatrix***. Η εντολή αυτή χρησιμοποιείται στην απόδοση σκιών για τον μετασχηματισμό την προβαλλόμενης γεωμετρίας του αντικειμένου στο έδαφος.

```
void glMultMatrixf (GLfloat *  ´´ αυθαίρετο μητρώο´´  );

void glMultMatrixd (GLdouble *  ´´ αυθαίρετο μητρώο´´  );
```

Το νέο μητρώο μετασχηματισμού που προκύπτει μετά τη διαδικασία, είναι το εξής:
νέο μητρώο μετασχηματισμού = τρέχων μητρώο μετασχηματισμού * αυθαίρετο μητρώο

5.7. Σύνθετοι μετασχηματισμοί

Για κάθε κατηγορία μητρώου μετασχηματισμού (μοντέλου και προβολής) η μηχανή της OpenGL προβλέπει την ύπαρξη μιας στοίβας, η οποία προσφέρει τη δυνατότητα αποθήκευσης πολλαπλών προφίλ για κάθε μητρώο μετασχηματισμού, με σκοπό τη μελλοντική τους χρήση.

Κατά την έναρξη εκτέλεσης ενός προγράμματος, στη στοίβα κάθε μητρώου μετασχηματισμού ορίζεται μόνο ένα, από τα υπόλοιπα της στοίβας, μητρώο στην κορυφή της. Το μητρώο αυτό αναφέρεται ως το ενεργό μητρώο μετασχηματισμού και είναι αυτό που βρίσκεται σε ισχύ την εκάστοτε χρονική στιγμή. Οποιαδήποτε τροποποίηση εκτελούμε με τη

δήλωση εντολών στοιχειωδών μετασχηματισμών επιδρά στο ενεργό μητρώο μετασχηματισμού. Όσα μητρώα βρίσκονται χαμηλότερα στη στοίβα μένουν ανεπηρέαστα.

Τώρα στην περίπτωση που έχουμε στην σκηνή μας πολλά αντικείμενα και θέλουμε να μετασχηματίσουμε το καθένα από αυτά ξεχωριστά είναι προφανές ότι θα χρειαστούμε περισσότερα από ένα μητρώα μετασχηματισμού, δλδ ένα μητρώο για κάθε μετασχηματισμό.

Αυτό που χρειαζόμαστε είναι η δηλωση του τρέχοντος μητρώου μετασχηματισμού με τον εκάστοτε μετασχηματισμό του αντικειμένου ή των αντικειμένων που υφίσταντε ίδιο μετασχηματισμό, ως ενεργο ,την απελευθέρωση του και δήλωση του επομένου μητρώου της στοίβας ως ενεργό για υλοποίηση επομένων μετασχηματισμών. Η παραπάνω διαδικασία αποθήκευσης και δήλωσης του επομένου μητρώου εκτελείται με την εντολή ***glPushMatrix*** και η απελευθέρωση με την εντολή ***glPopMatrix.***

Παράδειγμα 5.5.a Εναλαγή μητρώων

```
glMatrixMode(GL_MODELVIEW);              //επιλογη μητρωου μετασχηματισμου

glPushMatrix();                  // δηλωση του μητρωου ως ενεργο
Rotate  object 1
glPopMatrix();                           //Απελευθέρωση του μητρώου

glPushMatrix();                  //δήλωση του επομένου μητρώου ως ενεργό
Translate  object 2
glPopMatrix();                           //Απελευθέρωση του μητρώου
```

Παράδειγμα 5.5.b Εναλαγή μητρώων

```
glMatrixMode(GL_MODELVIEW);              //επιλογη μητρωου μετασχηματισμου
glLoadIdentity();

Translate  object 1                     //Το object 1 θα μετατοπιστει

Translate  object 2                     //Το object 2 θα μετατοπιστει δυο φορες και κατά την
                                         μετατόπηση  του object 1 και κατά την δικια του
                                         μετατόπηση

glPopMatrix();                           //Απελευθέρωση του μητρώου
```

CHAPTER VI

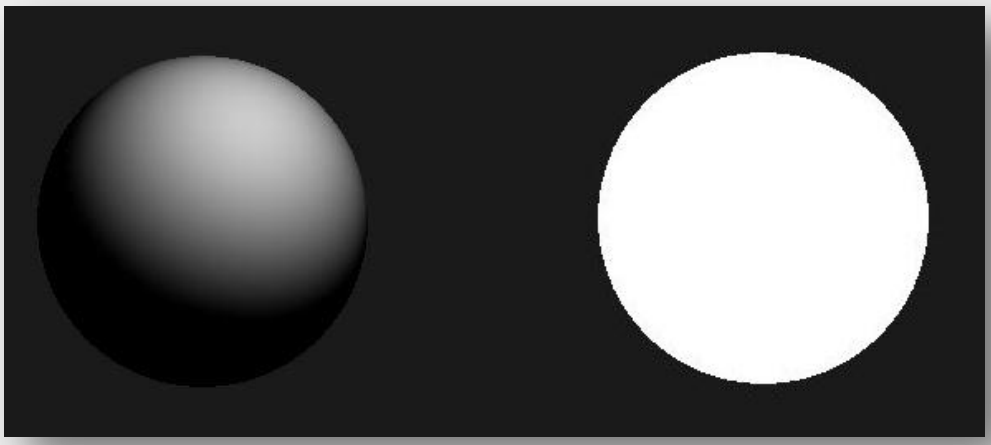
ΦΩΤΟΡΕΑΛΙΣΜΟΣ

Στο Κεφάλαιο 2 αναλύσαμε τις μεθόδους απόδοσης χρωμάτων σε επιφάνειες κατά τη σχεδίασή τους στη σκηνή. Ωστόσο, εάν ενδιαφερόμαστε για την απόδοση σκηνών που προσομοιώνουν τον πραγματικό κόσμο, τα εργαλεία που αναλύσαμε έως τώρα δεν επαρκούν. Προκειμένου να δημιουργήσουμε μια ρεαλιστική απεικόνιση, θα πρέπει να λάβουμε υπόψη και να προσομοιώσουμε τον τρόπο με τον οποίο παράγεται μια εικόνα στον πραγματικό κόσμο. Δεδομένου ότι στην πράξη μια εικόνα σχηματίζεται βάσει της επίδρασης των πηγών φωτισμού στις επιφάνειες των αντικειμένων, το πώς θα αποδοθεί η σκηνή καθορίζεται από τις ιδιότητες των πηγών φωτισμού, από τις ιδιότητες των επιφανειών αλλά και από την σκίαση των αντικειμένων.

6.1. Φωτορεαλισμός

Η OpenGL πριν την εμφάνιση της σκηνής στο παράθυρο υπολογίζει το χρώμα του κάθε pixel της τελικής σκηνής που έχει στην κατοχή του ο framebuffer. Μέρος αυτού του υπολογισμού εξαρτάται από το ίδος του φωτισμού που χρησιμοποιείται στη σκηνή και τον τρόπο που τα αντικείμενα στη σκηνή αντανακλούν ή απορροφούν αυτό το φως. Για παράδειγμα, ο ωκεανός έχει ένα διαφορετικό χρώμα σε μια φωτεινή, ηλιόλουστη μέρα από ό, τι σε μια γκρίζα, συννεφιασμένη μέρα. Η παρουσία του ηλιακού φωτός ή τα σύννεφα καθορίζει αν θα δείτε τον ωκεανό ως φωτεινό τυρκουάζ ή σκοτεινό γκρι-πράσινο.
Στην πραγματικότητα, τα περισσότερα αντικείμενα δεν φαίνονται καν τρισδιάστατα μέχρι να φωτιστούν.
Η Εικόνα 5.1 δείχνει δύο εκδόσεις της ίδιας ακριβώς σκηνής (ενιαία σφαίρα), μία με φωτισμό και μία χωρίς.

Εικόνα 5.1. A Lit and an Unlit Sphere

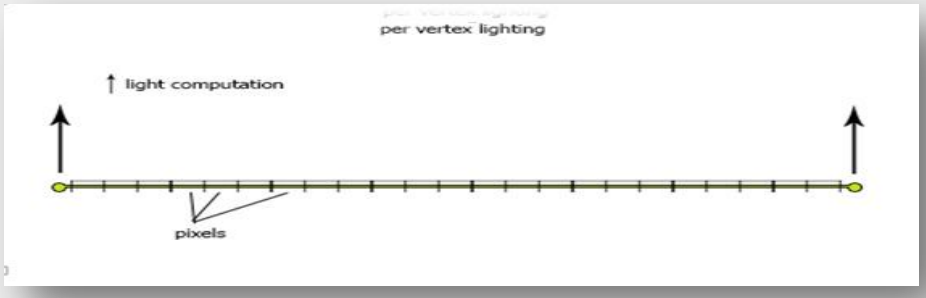


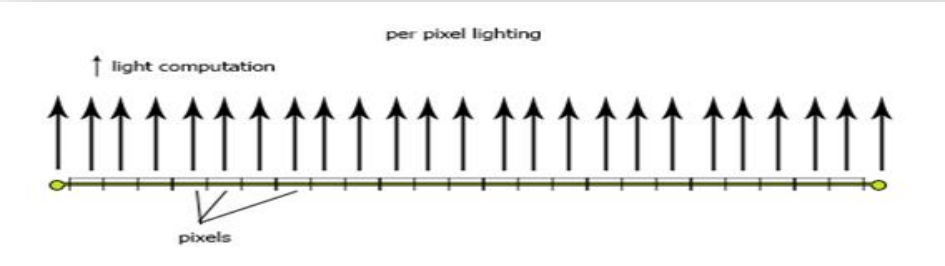
6.2. Τεχνίκες φωτισμού

Οι τεχνικές φωτισμού μπορούν να διακριθούν στις δυο παρακάτω κατηγορίες.

- Φωτισμός ανα εικονοστοιχειο (per pixel lighting)
- Φωτισμός ανα επιφάνεια (per vertex lighting)

Η διαφορά τους ότι περά από το ότι η πρώτη δίνει μια πιο ρεαλιστική απόδοση της σκηνής, είναι ότι στην per vertex lighting το χρώμα υπολογίζεται στις κορυφές της κάθε επιφάνειας ξεχωριστά και στη συνέχεια παρεμβάλλεται μεταξύ των κορυφών ενώ στην per pixel lighting ο προσανατολισμός επιφανείας *(βλέπε 6.4.)* παρεμβάλλεται μεταξύ των κορυφών και το χρώμα υπολογίζεται ξεχωριστά για κάθε οικονοστοιχειο της επιφάνειας.
Οι βιβλιοθήκες τις openGl δεν παρέχουν τις απαραιτιτες εντολες για την υλοποιηση του φωτισμου ανα εικονοστοιχείο Οι εντολες αυτές παρέχονται στις βιβλιοθήκες μιας νεότερης έκδοσης της OpenGl, την OpenGl SL (OpenGl Shading Language).





6.3. Πηγές φωτισμού

Από πλευράς φυσικής, μια εικόνα σχηματίζεται από την πρόσπτωση ακτινών φωτός σε μια φωτοευαίσθητη επιφάνεια (όπως λ.χ. στο ανθρώπινο μάτι ή στον αισθητήρα μιας κάμερας). Επομένως, προκειμένου να σχηματιστεί μια εικόνα απαιτείται η ύπαρξη φωτεινής ενέργειας, η οποία προέρχεται από μία ή περισσότερες πηγές φωτισμού. Χωρίς πηγές φωτισμού δεν είναι εφικτός ο σχηματισμός εικόνας.

Στην OpenGL ο φωτισμός ενεργοποιείται δίνοντας στην εντολή *glEnable* το όρισμα GL_LIGHTING και υποστηρίζει μέχρι και οκτώ διαφορετικές πήγες φωτισμού LIGHT1 έως LIGHT8.

```
glEnable( GL_LIGHTING );
glEnable( LIGHT1 );
...
glEnable( LIGHT8 );
```

Προκειμένου να οριστεί μια πηγή φωτισμού, πρέπει να προσδιορίσουμε ένα σύνολο παραμέτρων που χαρακτηρίζουν τη συμπεριφορά της. Τα χαρακτηριστικά μιας πηγής φωτισμού διακρίνονται σε δύο κατηγορίες:

α) Χαρακτηριστικά που προσδιορίζονται με μια αριθμητική τιμή

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

Το όρισμα *light* είναι μια συμβολική σταθερά που προσδιορίζει την πηγή φωτισμού.
Το όρισμα *parameterName* είναι μια συμβολική σταθερά που καθορίζει το χαρακτηριστικό που αποδίδουμε.
Το όρισμα *parameter* δηλώνει την αριθμητική τιμή που αποδίδουμε στην ιδιότητα parameterName.

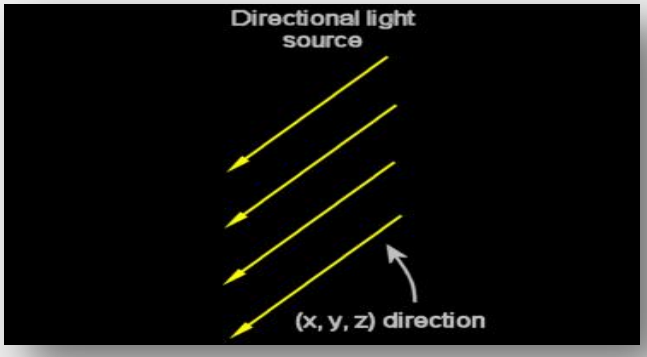
β) Χαρακτηριστικά που προσδιορίζονται από ένα σύνολο αριθμητικών τιμών

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

Το όρισμα parameterArray είναι δείκτης σε μητρώο. Το μητρώο αυτό περιέχει το σύνολο των τιμών που προσδιορίζουν το χαρακτηριστικό parameterName.

6.3.1. Κατευθυντικές πηγές (Πηγές σε άπειρη απόσταση)

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, ορίζουμε μια πηγή σε άπειρη απόσταση με την εντολή

glLightfv(GL_LIGHTX, GL_POSITION, positionVector);

με το τέταρτο στοιχείο του μητρώου positionVector ίσο με μηδέν. Στην περίπτωση αυτή, τα τρια πρώτα στοιχεία του μητρώου positionVector δε δηλώνουν τη θέση της πηγής (εφόσον βρίσκεται σε άπειρη απόσταση) αλλά το διάνυσμα κατεύθυνσης των ακτίνων που αναχωρούν από την πηγή.

```
GLfloat light0Position[]={1,0,1,0};
glLightfv(GL_LIGHT0, GL_POSITION, light0Position);
```

Το μοντέλο ακτινικής εξασθένησης δεν εφαρμόζεται σε πηγές φωτισμού που βρίσκονται σε άπειρη απόσταση, διοτί τότε η ένταση του φωτός στη σκηνή θα μηδενιζόταν. Στην περίπτωση αυτή, ο συντελεστής ακτινικής εξασθένησης είναι ίσος με 1. Επομένως στη γενική περίπτωση ισχύει:

$$f_{raddat}(d) = \begin{cases} 1 \\ \dfrac{1}{a_0 + a_1 * d + a_2 * d^2} \end{cases}$$

Πηγές σε άπειρη απόσταση μπορούν να χρησιμοποιηθούν για να μοντελοποιήσουμε τις ακτίνες του ήλιου.

6.3.2. Κατευθυντικές πηγές (σποτ)

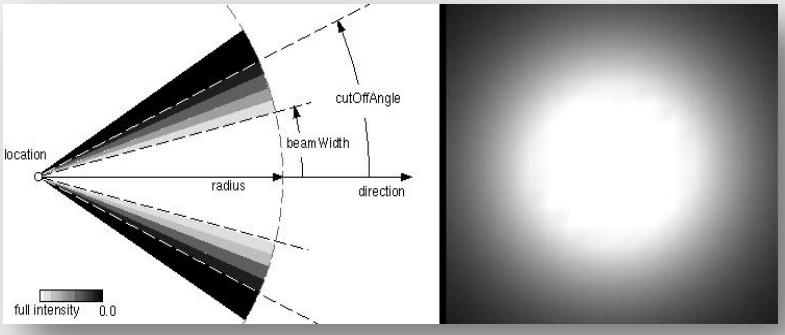
Μια σημειακή πηγή φωτισμού μπορεί να καθοριστεί ούτως ώστε να εκπέμπει σε ένα περιορισμένο γωνιακό εύρος, όπως ένα φωτιστικό σποτ. Σε αυτή την περίπτωση, το τμήμα της σκηνής που φωτίζεται από την κατευθυντική πηγή περικλείεται σε έναν κώνο. Το άνοιγμα του κώνου διάχυσης καθορίζεται από μια γωνία αποκοπής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

Όπως στις σημειακές, έτσι και στις κατευθυντικές πηγές φωτισμού, κάθε εκπεμπόμενη ακτίνα εξασθενεί συναρτήσει της απόστασης από την πηγή. Επιπλέον όμως, εξασθενεί και συναρτήσει της γωνιακής της απόκλισης από τον άξονα του κώνου φωτισμού. Επομένως, ένας παρατηρητής παρατηρεί τη μέγιστη ένταση όταν βρίσκεται επί του άξονα του κώνου ($\varphi = 0$) . Όσο αποκλίνει γωνιακά από τον άξονα εκπομπής, η παρατηρούμενη ένταση $I(\varphi)$ εξασθενεί, βάσει του ακόλουθου παράγοντα γωνιακής εξασθένησης (angular attenuation).

$$f_{angatt}(\varphi) = cos^n(\varphi)$$

Ο εκθέτης n καθορίζει κατά πόσο διαχέεται η εκπεμπόμενη ένταση γύρω από τη θέση μέγιστης ακτινοβολίας. Θέτοντας μεγάλες τιμές στον εκθέτη, ορίζουμε μια ισχυρά κατευθυντική πηγή και η εκπεμπόμενη ένταση μειώνεται ραγδαία συναρτήσει της γωνιακής απόκλισης φ. Μικρές τιμές του εκθέτη προκαλούν μικρότερη διακύμανση της εκπεμπόμενης έντασης στο γωνιακό έυρος που καλύπτει η πηγή.

Εικόνα 5.2 . Ο σποτ φωτισμος (κώνος διάχησης)



Κατεύθυνση μέγιστης εκπομπής

Η κατεύθυνση εκπομπής καθορίζει τον άξονα εκπομπής της πηγής και ορίζεται προσδιορίζοντας με την εντολή glLightfv την παράμετρο GL_SPOT_DIRECTION:

```
glLightfv(GL_LIGHTX, GL_SPOT_DIRECTION, directionVector);
```

Γωνία κάλυψης

Το γωνιακό εύρος που καλύπτει μια κατευθυντική πηγή ορίζεται προσδιορίζοντας με την εντολή glLightf την παράμετρο GL_SPOT_CUTOFF:

```
glLightf(GL_LIGHTX, GL_SPOT_CUTOFF, cutoffAngle);
```

όπου cutoffAngle η γωνία αποκοπής της πηγής . Αντικείμενα που αποκλίνουν από τον κύριο άξονα εκπομπής της πηγής κατά γωνίες μεγαλύτερες της cutoffAngle δε φωτίζονοται καθόλου από αυτήν. Εάν για μια σημειακή πηγή δεν ορίσουμε γωνία αποκοπής, η μηχανή της OpenGL θεωρεί ως προκαθορισμένη γωνία αποκοπής τις 180 μοίρες. Επομένως στην περίπτωση αυτή η πηγή εκπέμπει φως προς όλες τις κατευθύνσεις.

Γωνιακή εξασθένηση έντασης

Η γωνιακή εξασθένηση καθορίζεται δηλώνοντας τον εκθέτη εξασθένησης που ορίζεται στο μοντέλο Phong . Η δήλωσή του γίνεται προσδιορίζονταις με την εντολή glLightf την παράμετρο GL_SPOT_EXPONENT:

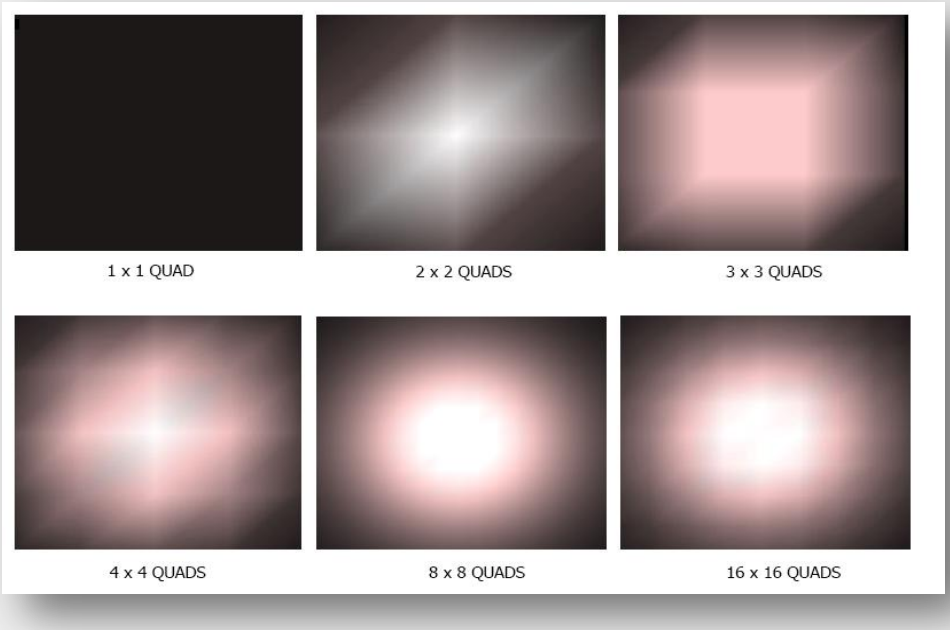
```
glLightf(GL_LIGHTX,GL_SPOT_EXPONENT, exponent);
```

όπου exponent η τιμή του εκθέτη, η οποία κυμαίνεται μεταξύ των τιμών 0 και 128. Εάν δεν καθορίσουμε την τιμή του εκθέτη, η μηχανή της OpenGL θεωρεί ως προκαθορισμένη τιμή το . Επομένως, στην περίπτωση αυτή, η κατευθυντική πηγή εκπέμει με ομοιόμορφη ένταση σε όλο το γωνιακό εύρος της.

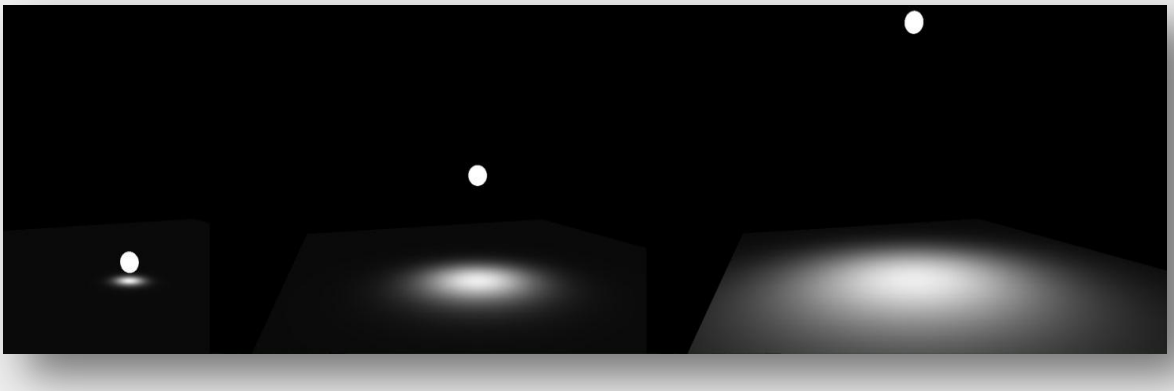
Υλοποίηση spot φωτισμού

Η υλοποίηση επιτυνχάνεται χρησιμοποιωντας την τεχνικη φωτισμου ανα εικονοστοιχείο αλλα υπαρχει η δυνατοτητα αναπαραστασης της χρησιμοποιοντας την τεχνικη ανα επιφανεια όπως θα δουμε στις παρακάτω εικόνες.

Η απεικόνιση μιας σποτ πηγής στην οθόνη απαιτεί δυο σταδία υλοποίησης.
Την υλοποίηση της πηγής που αναλύεται παραπάνω και την υλοποίηση της προσπίπτουσας επιφάνειας (vertex) ώστε να ‘‘δεχτεί’’ το φωτισμό.
Μια επιφανεια κατασκευασμενη από ένα quad ή ένα triangle (δλδ από 4 ή 3 κορυφές) δεν είναι δυνατο να φωτιστει από μια σποτ πηγη χρυσιμοποιοντας τον φωτισμο ανα επιφανεια γιατι όπως ειδαμε στην ενοτητα 6.2 το χρωμα (του φωτος) υπολογιζεται στις κορυφες της επιφάνειας και κατι τετοιο δεν μπορει να καταστησει δυνατο τον φωτισμο της επιφανειας βασει του κωνου διαχησης που εκπεμπεται από την πηγη. Η επιφανεια θα πρεπει να αποτελειται από πολλές μικρές επιφάνειες *(βλέπε ενότητα 3.3.3)* για τον πιο ακριβη υπλογισμο του φωτος.

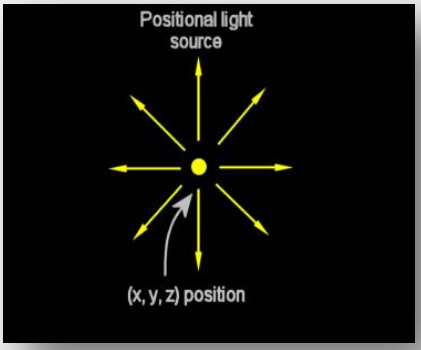


Το παρακάτω παράδειγμα υλοποιεί μία σπότ πηγή στο 3Δ χώρο. *(βλέπε κώδικα στο κεφάλαιο 12 παράδειγμα 1).*



6.3.3. Σημειακές πηγές

Οι σημειακές πηγές φωτισμού καταλαμβάνουν άπειρα μικρό χώρο και δηλώνονται ορίζοντας τη θέση τους στη σκηνή. Σε μια ομοιόμορφη σημειακή πηγή, οι ακτίνες διαδίδονται σφαιρικά προς όλες τις κατευθύνσεις και με την ίδια ένταση.



Για να δηλώσουμε μια σημειακή πηγή στη σκηνή, προσδιορίζουμε τη θέση της με την εντολή

```
glLightfv (lightName, GL_POSITION, positionVector);
```

όπου positionVector δείκτης σε μητρώο με τέσσερα στοιχεία. Τα τρία πρώτα στοιχεία του ορίζουν τη θέση και το τέταρτο στοιχείο ορίζει εάν η πηγή φωτός είναι σημειακή. Με μη μηδενική τιμή ορίζουμε μια σημειακή πηγή.

```
GLfloat light0Position[] = {5,5,0,1}
glLightfv (GL_LIGHT0, GL_POSITION, light0Position);
```

Σε πραγματικές συνθήκες η ένταση των σημειακών πηγών εξασθένεί κατά την απομάκρυνσή μας από τη πηγή. Αυτή η εξασθένηση ονομάζεται ακτινική εξασθένηση *(radial attenuation)* και οφείλεται στο διασκορπισμό της εκπεμπόμενης ισχύος σε συνεχώς μεγαλύτερες σφαιρικές επιφάνειες, καθώς αυξάνεται η απόσταση από την πηγή. Συγκεκριμένα, η εξασθένηση της έντασης μιας σημειακής πηγής συναρτήσει της απόστασης ακολουθεί το νόμο του αντιστρόφου τετραγώνου, δηλαδή σε ένα σημείο που απέχει απόσταση d από την πηγή για τον υπολογισμό της έντασης λαμβάνεται υπόψη ο παρακάτω παράγοντας εξασθένησης.

$$f_{raddat}(d) = \frac{1}{a_0 + a_1 * d + a_2 * d^2}$$

Οι τιμές των συντελεστών α_0, α_1, α_2 επιλέγονται αυθαίρετα από τον προγραμματιστή, ούτως ώστε να προκύψει το επιθυμητό αισθητικό αποτέλεσμα.
Οι συντελεστές εξασθένησης για την πηγή φωτισμού lightName (lightName=GL_LIGHT0,…,GL_LIGHT7) ορίζονται χρησιμοποιώντας τις εντολές.

- **glLightf (lightName, GL_CONSTANT_ATTENUATION, α_0); καθορισμός σταθερού όρου α_0**
- **glLightf (lightName, GL_LINEAR_ATTENUATION, α_1); καθορισμός γραμμικού όρου α_1**
- **glLightf (lightName, GL_QUADRATIC_ATTENUATION, α_2); καθορισμός τετραγωνικού όρου α_2**

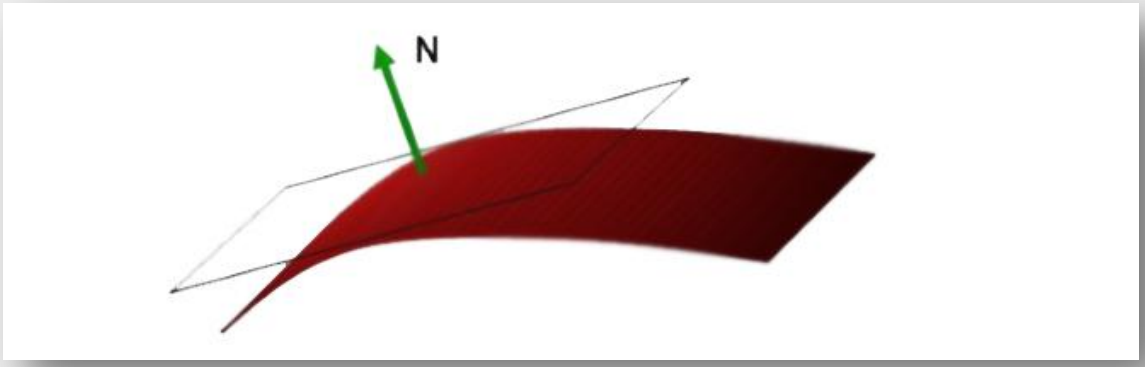
Σημειακές πηγές μπορούν να χρησιμοποιηθούν για να μοντελοποιήσουμε μια λάμπα σε ένα εξωτερικό ή εσωτερικό χώρο.

6.4. Ανακλώσες επιφάνειες - Χαρακτηριστικά επιφανειών

Στον πραγματικό κόσμο, ένας παρατηρητής αντιλαμβάνεται την παρουσία επιφανειών από τις ανακλάσεις που αυτές προκαλούν στις ακτίνες των πηγών φωτισμού. Κάθε επιφάνεια γίνεται αντιληπτή με διαφορετικά χαρακτηριστικά ανάλογα με τα χαρακτηριστικά της ανακλαστικότητάς της (χρώμα). Επιπλέον όμως, παίζει ρόλο και η σχετική θέση της κάθε επιφάνειας ως προς τις υπάρχουσες πηγές φωτισμού.

6.4.1 Προσανατολισμός επιφανείας (NORMALIZATION)

Προκειμένου να περιγράψουμε τον προσανατολισμό μιας επιφάνειας σε κάθε σημείο της, χρησιμοποιούμε την έννοια του κανονικού διανύσματος.
Εάν θεωρήσουμε μια καμπύλη επιφάνεια στο χώρο, τότε, σε κάθε σημείο της επιφανείας ορίζουμε ως κανονικό διάνυσμα το διάνυσμα που είναι κάθετο στο εφαπτόμενο επίπεδο του συγκεκριμένου σημείου



Εικόνα 6.1 . Κανονικό διάνυσμα σημείου επιφανείας

Δεδομένου ότι σε εφαρμογές γραφικών οι επιφάνειες προσεγγίζονται από πολυγωνικά πλέγματα, κάθε στοιχειωδης πολυγωνική επιφάνεια συμπίπτει με το εφαπτόμενο επίπεδό της. Επομένως, εξ΄ορισμού, το κανονικό διάνυσμα μιας πολυγωνικής επιφάνειας είναι κάθετο σε αυτήν. Εάν θεωρήσουμε λοιπόν τρεις κορυφές ενός του πολυγώνου x_1, y_1, z_1 , x_2, y_2, z_3 και x_3, y_3, z_3 , το κανονικό διάνυσμά του προκύπτει από το εξωτερικό γινόμενο των διανυσμάτων που ορίζουν οι κορυφές του, δηλαδή

$$\overrightarrow{N} = \begin{vmatrix} \vec{x} & \vec{y} & \vec{z} \\ (x_2 - x_1) & (y_2 - y_1) & (z_2 - z_1) \\ (x_3 - x_1) & (y_3 - y_1) & (z_3 - z_1) \end{vmatrix}$$

Ο παραπάνω τύπος του ορισμού κανονικού διανύσματος επιφανείας εφαρμόζεται από την OpenGL, όταν ο προγραμματιστής δε δηλώνει τιμές για αυτό. Ωστόσο ο προγραμματιστής μπορεί να ρυθμίσει τον προσανατολισμό μιας επιφάνειας, δηλώνοντας τις συνιστώσες του κανονικού διανύσματος για κάθε κορυφή της επιφάνειας. Μπορούμε να αναθέσουμε σε κάθε κορυφή ένα κανονικό διάνυσμα με την εντολή **glNormal3*:**

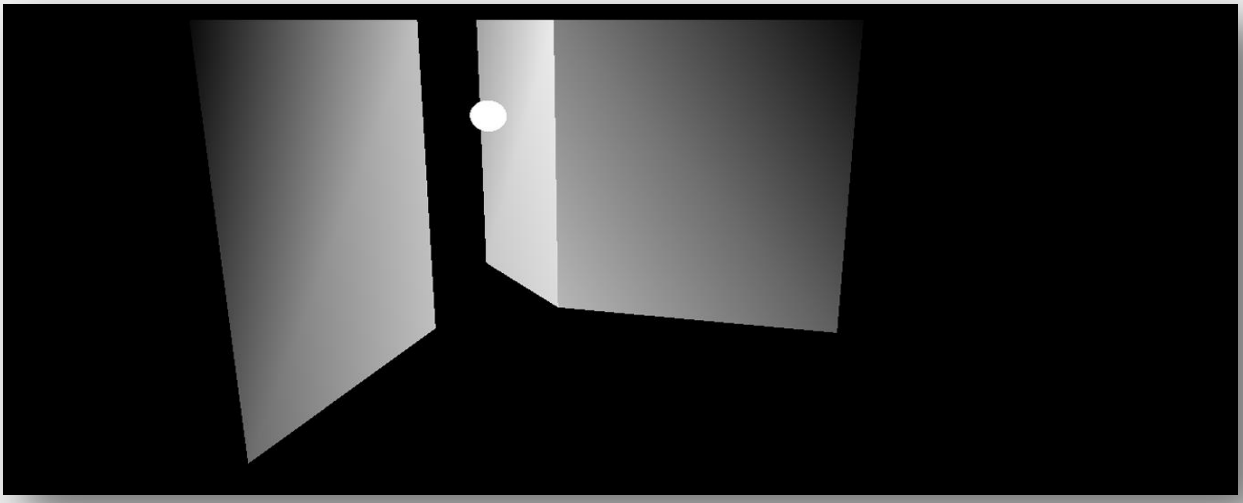
```
glNormal3{bsifd}(TYPE nx, TYPE ny TYPE nz );
```

όπου nx, ny και nz οι συνιστώσες του κανονικού διανύσματος.

Από τη στιγμή που θα καθορίσουμε τον προσανατολισμό του κανονικού διανύσματος, αυτός ανατίθεται σε κάθε κορυφή που δηλώνουμε στη συνέχεια του προγράμματος. Δηλαδή ο προσανατολισμός του κανονικού διανύσματος είναι μια μεταβλητή κατάστασης. Προκειμένου να ορίσουμε σε κάθε κορυφή μιας επιφάνειας διαφορετικό προσανατολισμό, απαιτείται η αλλαγή του κανονικού διανύσματος.

Για παράδειγμα έχουμε την παρακάτω σκηνή στον 3Δ χώρο.
Ο παρακάτω κώδικας σχεδιάζει τις πλευρές και τα κανονικά διανύσματα τους.

```
glBegin(GL_QUADS);

glNormal3f(-1.0, 0.0, 0.0);          η επιφάνεια που φωτίζεται κοιτάει προς τον αρνητικό άξονα του x
glVertex3f( 0,  0,  0);
glVertex3f( 0,  5,  0);
glVertex3f( 0,  5,  10);
glVertex3f( 0,  0,  10);

glNormal3f(0.0, -1.0, 0.0);          η επιφάνεια που φωτίζεται κοιτάει προς τον αρνητικό άξονα του y
glVertex3f( 0,  0,  0);
glVertex3f( 6, -5,  0);
glVertex3f( 6, -5,  10);
glVertex3f( 0,  0,  10);

glNormal3f(0.0, -1.0, 0.0);          η επιφάνεια που φωτίζεται κοιτάει προς τον αρνητικό άξονα του y
glVertex3f( -4,  0,  0);
glVertex3f( -10, -5,  0);
glVertex3f( -10, -5,  10);
glVertex3f( -4,  0,  10);

glEnd();
```



6.4.2. Σκίαση Gouraud

Οπως αναφέραμε στην παραπάνω ενότητα, κάθε στοιχειώδης πολυγωνική επιφάνεια χαρακτηρίζεται από το κανονικό της διάνυσμα, το οποίο χρησιμοποιείται για υπολογισμούς από το μοντέλο φωτισμού. Ωστόσο, μια πολυγωνική επιφάνεια δεν αποτελείται από ένα, αλλά από πολλά εσωτερικά σημεία. Δεδομένου ότι για τον υπολογισμό της φωτεινότητας έχουμε μόνο ένα διαθέσιμο κανονικό διάνυσμα ανά πολυγωνική επιφάνεια, το μοντέλο φωτισμού υπολογίζει μία τιμή φωτισμού για όλα τα pixels της πολυγωνικής επιφάνειας. Ωστόσο η αναπαράσταση αυτή έχει μειωμένη λεπτομέρεια και δεν αναπαριστά ρεαλιστικά τη φωτιζόμενη επιφάνεια.

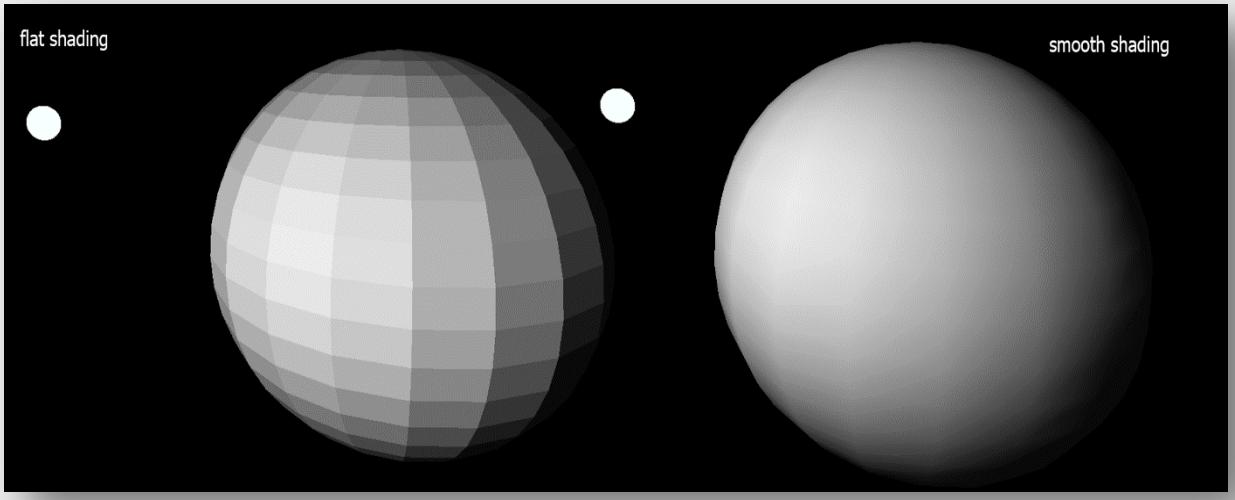
Μια πιο ρεαλιστική απόδοση της έντασης στα σημεία των επιφανειών αποδίδεται με τη μέθοδο σκίασης **Gouraud**. Στην τεχνική Gouraud, υπολογίζουμε ένα κανονικό διάνυσμα για κάθε μία κορυφή του πολυγώνου.

Η σκίαση Gouraud είναι ενεργοποιημένη εξ' αρχής στη μηχανή καταστάσεων της OpenGL. Ωστόσο, ο προγραμματιστής έχει τη δυνατότητα να επιλέξει την ενιαία απόδοση φωτεινότητας σε κάθε στοιχειώδη επιφάνεια με την εντολή **glShadeModel**. Δίνοντας την εντολή:

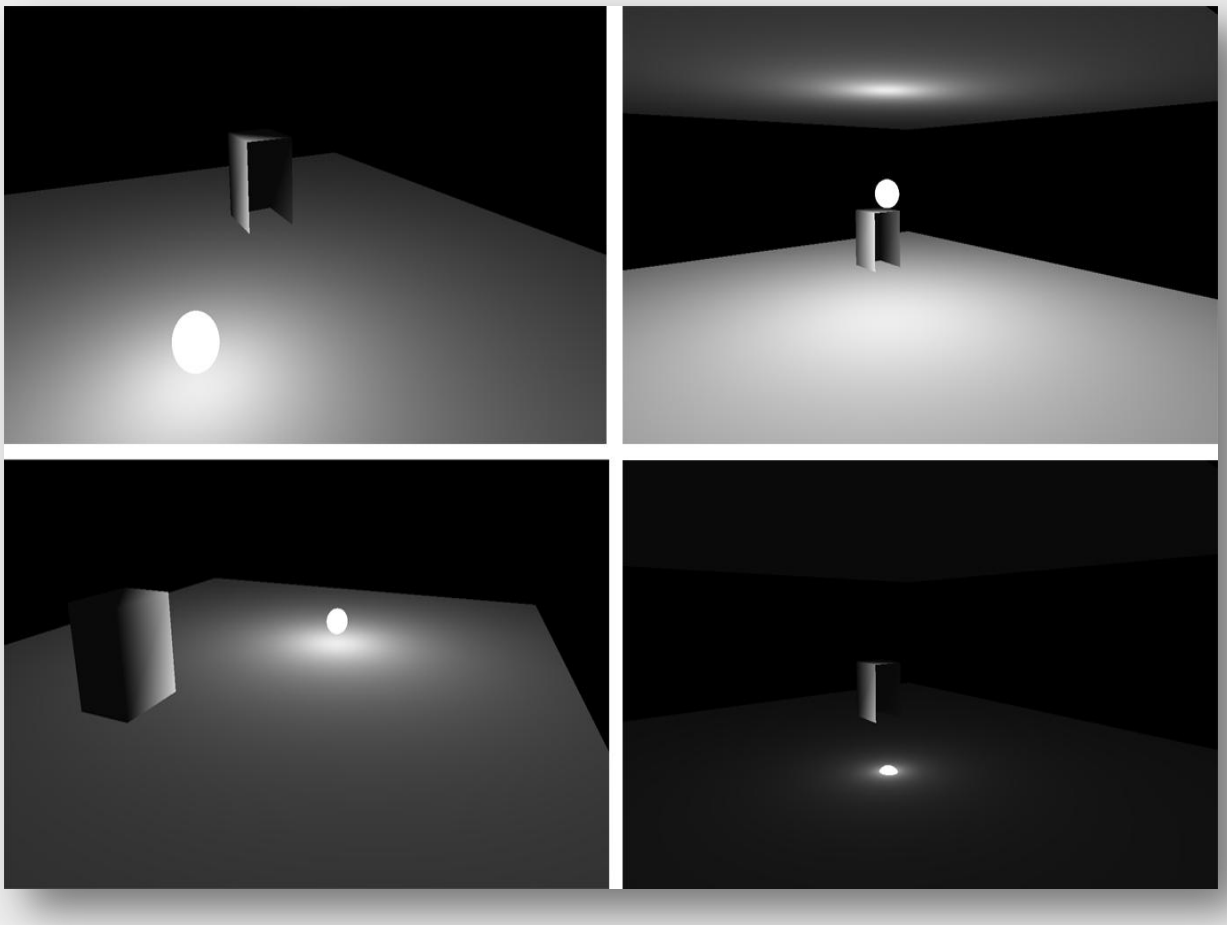
glShadeModel(GL_FLAT); **ενιαία σκίαση σε όλη την έκταση κάθε πολυγωνικής επιφανείας**

glShadeModel(GL_SMOOTH); **χρήση της σκίασης Gouraud**

Εικόνα 6.4: Σκίαση Gouraud



Για παράδειγμα έχουμε ένα τετράπλευρο και μία σημειακή πηγή φωτός με δυνατότητα μετατόπισης της από τον χρήστη στον τρισδιάστατο χώρο. *(βλέπε κώδικα στο κεφάλαιο 12 παράδειγμα 1).*Στο παράδειγμα υπολογίζονται τα κανονικά διανύσματα βάση της σκίασης Gouraud τόσο του τετραπλεύρου οσο του ταβανιού και του πατώματος.



6.5. Μοντελοποίηση πηγών φωτισμού στην OpenGL

Στο μοντέλο φωτισμού που υλοποιούν οι βιβλιοθήκες γραφικών, θεωρούμε ότι το φως που εκπέμπεται από μία πηγή μπορεί να διαχωριστεί σε τρεις συνιστώσες: μια συνιστώσα που συνεισφέρει αποκλειστικά στον περιβάλλοντα φωτισμό της σκηνής (περισσότερα για τον περιβάλλοντα φωτισμό στη συνέχεια), μια συνιστώσα που προκαλεί αποκλειστικά ανακλάσεις διάχυσης και μια συνιστώσα που προκαλεί αποκλειστικά κατοπτρικές ανακλάσεις.
Ο ***περιβάλλων φωτισμός (ambient light)*** εκτείνεται σε ολόκληρη τη σκηνή, έχει την ίδια ένταση σε όλη την έκταση της σκηνής και επιδρά σε όλες τις επιφάνειες. Ουσιαστικά, με τον περιβάλλοντα φωτισμό προσομοιώνουμε ένα γενικό επίπεδο φωτεινότητας που επιδρά ομοιόμορφα σε όλες τις επιφάνειες της σκηνής. Η ένταση και το χρώμα του περιβάλλοντος φωτισμού ορίζονται δηλώνοντας τις χρωματικές συνιστώσες του.

Ο ***Φωτισμός διάχυσης (diffuse light)*** προκαλεί αποκλειστικά ανακλάσεις διάχυσης. Επιδρά μόνο στις επιφάνειες που έχουν οπτική επαφή με την πηγή φωτισμού.

Ο ***Κατοπτρικός φωτισμός (specular light)*** συνεισφέρει αποκλειστικά στην εμφάνιση κατοπτρικών ανακλάσεων.

Στην OpenGL ορίζονται με την εντολές:

```
float emittedAmbient[4] = {r, g, b, a};
float emittedDiffuse[4] = {r, g, b, a};
float emittedSpecular[4] = {r, g, b, a};

glLightfv(GL_LIGHTX, GL_AMBIENT, emittedAmbient);          Ambient Light
glLightfv(GL_LIGHTX, GL_DIFFUSE, emittedDiffuse);          Diffuse Light
glLightfv(GL_LIGHTX, GL_SPECULAR, emittedSpecular);        Specular Light
```

6.5.1. 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 περιβάλλων φωτισμός, μια παράμετρος περιβάλλοντος φωτισμού που δεν εντάσσεται σε κάποια από τις πηγές φωτισμού (GL_LIGHT0, GL_LIGHT1, …) αλλά ρυθμίζεται ανεξάρτητα.
Η καθολική παράμετρος περιβάλλοντος φωτισμού ορίζεται με την χρήση της εντολής glLightModelfv ως εξής:

```
float globalAmbient[]={r,g,b};
glLightModelfv( GL_LIGHT_MODEL_AMBIENT, globalAmbient );
```

6.6. 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.
Ανάλογα με τον τύπο του χαρακτηριστικού, χρησιμοποιείται η εντολή:

- **`glMaterialf(GLenum face, GLenum property, GLfloat propertyValue);`**
- **`glMaterialfv(GLenum face, GLEnum property, GLfloat *propertyValues);`**

Το όρισμα face καθορίζει την όψη στην οποία αποδίδεται το εκάστοτε 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 property είναι ο τύπος φωτισμού που χρησιμοποιούμε.

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

6.6.1. Συντελεστές ανάκλασης επιφανειών στην OpenGL

Κάθε επιφάνεια χαρακτηρίζεται από τους συντελεστές ανάκλασής της. Οι συντελεστές ανάκλασης της επιφάνειας είναι αριθμητικές τιμές που καθορίζουν το ποσοστό της φωτεινής έντασης που επιστρέφει η επιφάνεια στη σκηνή. Δεδομένου ότι σε πραγματικές σκηνές η ανακλαστικότητα μιας επιφάνειας διαφέρει ανάλογα με τη συχνότητα (χρώμα) εκπομπής, ορίζουμε έναν συντελεστή ανάκλασης ανά βασικό χρώμα. Με αυτό τον τρόπο ουσιαστικά δηλώνουμε το χρώμα της επιφανείας.

Επιπρόσθετα, το μοντέλο φωτισμού της OpenGL θεωρεί ότι και η ανακλαστικότητα των επιφανειών περιγράφεται με τρεις "κατηγορίες" συντελεστών ανάκλασης: το "συντελεστή ανάκλασης περιβάλλοντος φωτισμού", το "συντελεστή ανάκλασης διάχυτου φωτισμού" και το "συντελεστή ανάκλασης της κατοπτρικής συνιστώσας". Κάθε συντελεστής ανάκλασης περιγράφεται από τις χρωματικές συνιστώσες του.

Στην OpenGL ορίζονται με την εντολές:

```
float ambientMat[4] = {r, g, b, a};
float diffuseMat[4] = {r, g, b, a};
float specularMat[4] = {r, g, b, a};

glMaterialfv ( face, GL_AMBIENT, ambientMat);        συντελεστή ανάκλασης περιβάλλοντος φωτισμού
glMaterialfv( face, GL_DIFFUSE, diffuseMat);            συντελεστή ανάκλασης διάχυτου φωτισμού
glMaterialfv( face, GL_SPECULAR, specularMat);     συντελεστή ανάκλασης του κατοπτρικου φωτισμού
```

Επιπλέον, στον συντελεστή ανάκλασης κατοπτρικου φωτισμού 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 );
```

όπου ο εκθέτης γωνιακής εξασθένησης, ο οποίος κυμαίνεται μεταξύ των τιμών 0 και 128.

Για τη ρεαλιστική απόδοση μιας ανακλώσας επιφάνειας οι συντελεστές διάχυτης ανάκλασης και οι συντελεστές ανάκλασης περιβάλλοντος φωτισμού πρέπει να είναι οι ίδιοι. Προκειμένου λοιπον να αποδώσουμε τις ιδιότητες αυτές με μία εντολή, μπορούμε να χρησιμοποιήσουμε την εντολή glMaterialfv με το όρισμα GL_AMBIENT_AND_DIFFUSE ως εξής:

```
float ambientAndDiffuse[]={r,g,b};
glMaterialfv(GL_FRONT_AND_BACK,GL_AMBIENT_AND_DIFFUSE,ambientAndDiffuse);
```

CHAPTER VII

ΥΦΕΣ

Στην ενότητα 6.1. αναφερθήκαμε 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 παρατηρούμε 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. Πχ. Είναι πολύ δύσκολο να προσομοιώσει κανείς τη μορφολογία φυσικών επιφανειών όπως την επιφάνεια δρόμου, 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. Στον κόσμο των γραφικών σε Η/Υ ένα τέτοιο πρότυπο απόδοσης αποκαλείται **υφή (texture)** και η διαδικασία της απόδοσης υφής σε μία επιφάνεια ονομάζεται **απόδοση υφής (texture mapping).**

Για απόδοση υφής σε επιφάνειες απαιτείται ο καθορισμός του προτύπου εικόνας-υφής που θα χρησιμοποιήσουμε. Υπάρχουν διάφορα πρότυπα εικόνων όπως JPEG, PNG, BMP, TGA κ.α., τα οποία έχουν διαφορετικό τρόπο φόρτωσης σε ένα project της Opengl. Το πρότυπο TGA χρησιμοποιείται για λόγους διαφάνειας υφών. *(βλέπε κεφάλαιο 7)*

7.1. Απόδοση υφής (Texture Mapping).

Η απόδοση μορφολογικών χαρακτηριστικών (απόδοση υφής) σε επιφάνειες επιτελείται με τη χρήση ενός προκαθορισμένου προτύπου, του μητρώου υφής. Το μητρώο υφής περιέχει χρωματικές τιμές, οι οποίες, στο σύνολό τους, καθορίζουν τη μορφολογία της επιφανείας ενός συγκεκριμένου υλικού, δηλαδή αποτελείται από εικονοστοιχεία. Ωστόσο, στη γλώσσα της Γραφικής, τα στοιχεία αυτά, προκειμένου να διακριθούν ως έννοια από τα pixels γεωμετρικών σχημάτων, αποκαλούνται στοιχεία υφής **(texture elements ή texels).**

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.

1) **Δημιουργία αναγνωριστικών αριθμών υφής.**

```
void glGenTextures(GLsizei n, GLuint *textureID);
```

GLsizei n: Καθορίζει τον αριθμό των υφών που πρόκειται να παραχθούν.
GLuint *textureID: το μητρώο που περιέχει τους αναγνωριστικούς αριθμούς

2) **Ανάθεση αναγνωριστικού αριθμού σε ένα μητρώο υφής.**

```
void glBindTexture(GLenum target, GLuint *textureID);
```

GLenum target: Καθορίζει το στόχο για τον οποίο δεσμεύεται η συγκεκριμένη υφή.
GLuint *textureID: Αντιστοιχεί στον αναγνωριστικό αριθμό που αναθέτουμε στην τρέχουσα υφή.

3) **Καθορισμός της υφής**

Για απόδοση υφής σε επιφάνειες απαιτείται η ενεργοποίηση της απόδοσης διδιάστατων υφών με την εντολή 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 );
```

Τα ορίσματα της αναλύονται ως εξής:

GLenum target: *προσδιορίζουμε το 2Δ μητρώο υφής.*

GLint level: *καθορίζει αν το μητρώο υφής χρησιμοποιείται ως μια βαθμίδα σε μια πυραμίδα μητρώων (mipmapping).*

GLint internalFormat: *καθορίζει το 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)*

GLsizei width: *καθορίζει το πλήθος των texel της εικόνας image58 οριζοντίως*

GLsizei height: *καθορίζει το πλήθος των texel της εικόνας image58 καθέτως*

GLint border: *καθορίζουμε το αν η υφή θα περιβάλλεται από όριο πάχους ενός pixel. Με την τιμή 0 ορίζουμε ότι δε χρησιμοποιείται όριο, ενώ με τιμή 1 ορίζουμε ότι χρησιμοποιούμε όριο.*

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

GL_RGB : *Ορίζουμε τη διαδοχή συνιστωσών κόκκινο-πράσινο-μπλε (χρησιμοποιείται σε αρχεία εικόνων τύπου BMP*
GL_BGR_EXT:*για διαδοχή συνιστωσών μπλε-πράσινο-κόκκινο (χρησιμοποιείται σε αρχεία εικόνων τύπου DIB*
GL_RGBA: *Ορίζουμε τη διαδοχή συνιστωσών κόκκινο-πράσινο-μπλε-alpha*
GL_BGRA_EXT: *Ορίζουμε τη διαδοχή συνιστωσών μπλε-πράσινο-κόκκινο*

GLenum type: *καθορίζει τον πρωτογενή τύπο δεδομένων με τον οποίο δίνονται τα texels στο μητρώο υφής. Δέχεται τις τιμές GL_UNSIGNED_BYTE (αρχεία εικόνων), GL_INT και GL_FLOAT.*

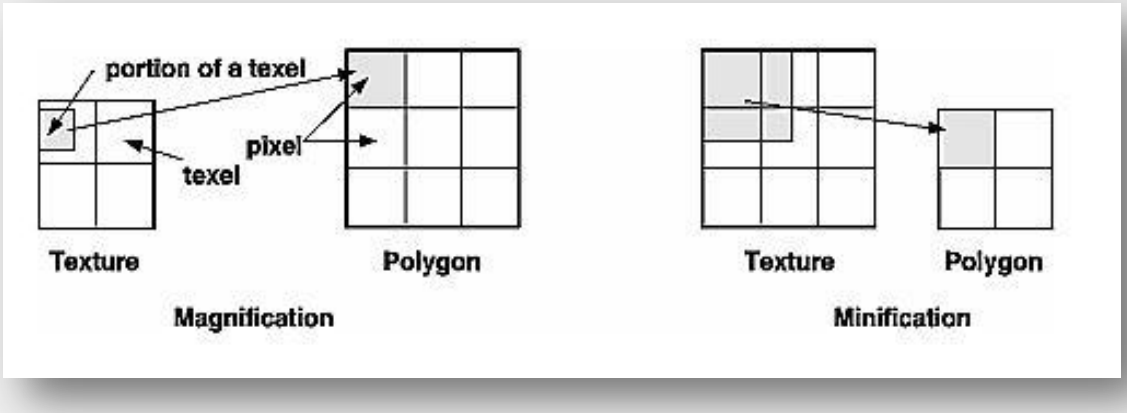
const GLvoid *texelArray: *Η παράμετρος texelArray είναι δείκτης στο μητρώο που περιέχει τις χρωματικές τιμές των texels. Στην περίπτωση που το μητρώο είναι μονοδιάστατο, τα στοιχεία του μητρώου προσδιορίζουν τις τιμές των texels ανά n-άδες όπου n το πλήθος των χρωματικών συνιστωσών που περιγράφουν κάθε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

7.2. Ρυθμίσεις απόδοσης υφής (Σμίκρυνση υφής - Μεγέθυνση υφής)

Οι χάρτες υφής (texture maps) είναι τετράγωνοι ή ορθογώνοι, αλλά μετά την χαρτογράφηση τους σε ένα πολύγωνο ή επιφάνεια και μετά τον μετασχηματισμό τους σε συντεταγμένες οθόνης, τα μεμονωμένα texels της υφής σπανίως αντιστοιχούν σε μεμονωμένα pixel της τελικής εικόνας στην οθόνη.

Ανάλογα με τους μετασχηματισμούς που χρησιμοποιούνται και τη χαρτογράφηση υφής που εφαρμόζεται ένα μοναδικό εικονοστοιχείο στην οθόνη μπορεί να αντιστοιχεί σε οτιδήποτε, από ένα μικρό τμήμα ενός texel (μεγέθυνση) μέχρι σε μια μεγάλη συλλογή από texels (σμίκρυνση), όπως δείχνεται στην εικόνα 6.1.

Σε κάθε περίπτωση, είναι ασαφές ποιες ακριβώς τιμές Texel θα πρέπει να χρησιμοποιούνται.Κατά συνέπεια, η OpenGL επιτρέπει τον ορισμό οποιαδήποτε από τις επιλογές φιλτραρίσματος για να καθορισμό αυτών των υπολογισμών. Επίσης, μπορείτε να καθορίσετε ανεξάρτητα τις μεθόδους φιλτραρίσματος για μεγέθυνση και σμίκρυνση.
Η επιλογή αυτή γίνεται με την εντολή glTexParameteri:



Εικόνα 6.1 : Texture Magnification and Minification

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

Με το όρισμα target αναφερόμαστε στο άν η ρύθμιση επιβάλλεται στην κατηγορία μονοδιάστατων ή διδιάστατων υφών. Παίρνει την τιμή GL_TEXTURE_1D όταν ρυθμίζουμε παραμέτρους απόδοσης που αφορούν μονοδιάστατες υφές ή την τιμή GL_TEXTURE_2D όταν ρυθμίζουμε παραμέτρους απόδοσης που αφορούν διδιάστατες υφές.

Το όρισμα parameterName παίρνει την τιμή GL_TEXTURE_MIN_FILTER ή GL_TEXTURE_MAG_FILTER, ανάλογα με το αν θα καθορίσουμε τη συμπεριφορά της απόδοσης υφής κατά τη σμίκρυνση ή τη μεγέθυνσή της αντίστοιχα.

Το όρισμα 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 παρεμβολή.

Οι παρακάτω τιμές του ορίσματος parameterValue αποδίδονται όταν χρησιμοποιούμε πυραμίδες υφής και θα αναλυθούν σε παρακάτω ενότητα.

GL_NEAREST_MIPMAP_NEAREST
GL_LINEAR_MIPMAP_NEAREST
GL_NEAREST_MIPMAP_LINEAR
GL_LINEAR_MAP_LINEAR

Εικόνα 7.1: Nearest and Linear filtering

7.3. Απόδοση συντεταγμένων υφής

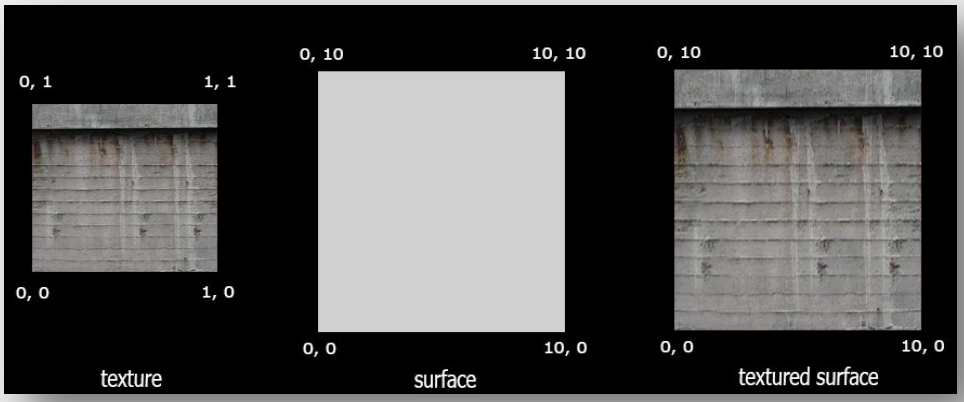
Στην OpenGL η απόδοση υφής σε γραμμές και επιφάνειες εκτελείται αναθέτοντας σε κάθε μία κορυφή τις συντεταγμένες της υφής. Με τον τρόπο αυτό η μηχανή της OpenGL αποδίδει τα στοιχεία του μητρώου υφής κατά μήκος της γραμμής ή στην έκταση της επιφάνειας ούτως ώστε να ικανοποιούνται οι οριακές συνθήκες που επιβάλλει ο προγραμματιστής. Η ανάθεση συντεταγμένων υφής γίνεται με τη χρήση της εντολής **glTexCoord{1,2}{i,s,f,d}.**

```
void glTexCoord1f (GLfloat s);                    για μονοδιάστατες υφές
void glTexCoord2f (GLfloat s, GLfloat t);               για διδιάστατες υφές
```

Με τα ορίσματα **s**, **t** ορίζουμε τις τρέχουσες συντεταγμένες υφής στους άξονες x και y και ουσιαστικά ορίζουν πόσες φορές θα αποδοθεί η υφή στην επιφάνεια.

Παράδειγμα 6.1: Απόδοση υφής σε επιφάνεια μία φορά κατά x και y.

```
glBindTexture(GL_TEXTURE_2D, _textureId57);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
glBegin(GL_QUADS);

glTexCoord2f(0, 0);glVertex3f( 0,  0,  0);
glTexCoord2f(1, 0);glVertex3f( 10, 0,  0);
glTexCoord2f(1, 1);glVertex3f( 10, 10, 0);
glTexCoord2f(0, 1);glVertex3f( 0,  10, 0);

glEnd();
```

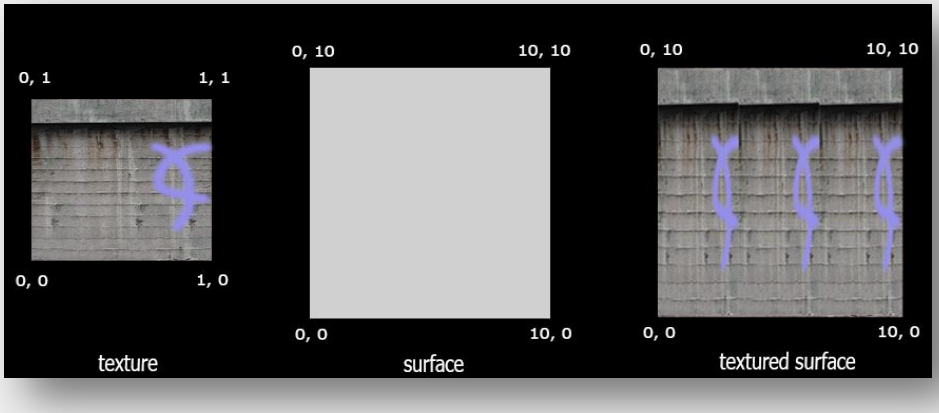


Όπως βλέπουμε η υφή με αυτόν τον τρόπο απόδοσης της τεντώνεται – μεγενθύνεται ώστε να καλύψει την επιφάνεια. Όμως αυτή η μεγένθυνση δεν προκάλει αισθητική αλλοίωση και αυτό γιατί η υφή είναι ηψυλής ανάλυσης (600 x 800). Τι θα γινόταν όμως αν έπρεπε να αποδόσουμε μια μικρής ανάλυσης υφή σε μία σχετικά μεγάλη επιφάνεια;

Το παρακάτω παράδειγμα απαντάει στο ερώτημα.

Παράδειγμα 6.2: Απόδοση υφής σε επιφάνεια τρείς φορές κατά x και μία κατά y.

```
glBindTexture(GL_TEXTURE_2D, _textureId57);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
glBegin(GL_QUADS);

glTexCoord2f(0, 0);glVertex3f( 0,  0,  0);
glTexCoord2f(3, 0);glVertex3f( 10, 0,  0);
glTexCoord2f(3, 1);glVertex3f( 10, 10, 0);
glTexCoord2f(0, 1);glVertex3f( 0,  10, 0);

glEnd();
```



Φόρτωση εικόνας προτύπου BMP από τον σκληρό δίσκο και απόδοση της σε επιφάνεια

```
#include "imageloader.h" header για BMP
#include "BMPLoader.cpp" C++ κώδικας για BMP


GLuint loadTexture58(Image* image58)                    κάλεσμα της κλάσης Image
{
        GLuint textureId58;                             Δήλωση textureId58;
        glGenTextures(1, &textureId58);          Δημιουργία αναγνωριστικών αριθμών υφής
        glBindTexture(GL_TEXTURE_2D, textureId58);      Ανάθεση αναγνωριστικού αριθμού
        glTexImage2D(GL_TEXTURE_2D,                     Καθορισμός υφής
        0,
        GL_RGB,
        image58->width, image58->height,
        0,
        GL_RGB,
        GL_UNSIGNED_BYTE,
        image58->pixels);
        return textureId58;
}

void loadTextures()                                     φόρτωση υφής από τον σκληρό δίσκο
{
        Image* image58 = loadBMP("Texures/58.bmp");
        _textureId58 = loadTexture58(image58);
        delete image58;
}

void drawSurface()                              Κατασκεύη επιφάνειας και απόδοση υφής στην επιφάνεια
{
        glBindTexture(GL_TEXTURE_2D, _textureId58);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);

        glBegin(GL_QUADS);
        glTexCoord2f(0, 0);glVertex3f( 0,  0,  0);
        glTexCoord2f(1, 0);glVertex3f( 10, 0,  0);
        glTexCoord2f(1, 1);glVertex3f( 10, 10, 0);
        glTexCoord2f(0, 1);glVertex3f( 0,  10, 0);
        glEnd();
}
```

Φόρτωση εικόνας προτύπου TGA από τον σκληρό δίσκο και απόδοση της σε επιφάνεια

```
#include "TGALoader.cpp"  C++  κώδικας για TGA
#include "Texture.h"      header για TGA
#include "Tga.h"          header για TGA


bool LoadTGA(Texture *, char *);

Texture textura[2];                                             two textures


int LoadTextures()
{
        int Status=FALSE;                                                       Status Indicator

        if (LoadTGA(&textura[0], "Texures/1.tga") &&
            LoadTGA(&textura[1], "Texures/2.tga"))
        {
                Status=TRUE;                                                    Set The Status To TRUE

                for (int loop=0; loop<2; loop++)                    Loop Through Both Textures
                {
                        glGenTextures(1, &textura[loop].texID);
                        glBindTexture(GL_TEXTURE_2D, textura[loop].texID);
                        glTexImage2D(GL_TEXTURE_2D,
                        0,
                        textura[loop].bpp / 8,
                        textura[loop].width,
                        textura[loop].height,
                        0,
                        textura[loop].type,
                        GL_UNSIGNED_BYTE,
                        textura[loop].imageData);
```

```
                        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
                        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);

                        if (textura[loop].imageData)                    If Texture Image Exists ( CHANGE )
                        {
                                free(textura[loop].imageData);              Free The Texture Image Memory
                        }}}
        return Status;                                                                  Return The Status


void drawSurface()                                      Κατασκεύη επιφάνειας και απόδοση υφής στην επιφάνεια
{
        glBindTexture(GL_TEXTURE_2D, textura[0].texID);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);

        glBegin(GL_QUADS);
        glTexCoord2f(0, 0);glVertex3f( 0,  0,  0);
        glTexCoord2f(1, 0);glVertex3f( 10, 0,  0);
        glTexCoord2f(1, 1);glVertex3f( 10, 10, 0);
        glTexCoord2f(0, 1);glVertex3f( 0,  10, 0);
        glEnd();}
```

Όπως βλέπουμε ο κώδικας για την ανάθεση της υφής σε επιφάνεια είναι ίδιος και για τα δυο διαφορετικά πρότυπα εικόνων (BMP και TGA).Αυτό που αλλάζει είναι ο τρόπος φόρτωσης των εικόνων από τον σκληρό δίσκο ώστε να είναι διαθέσιμη για την απόδοση της από τον προγραμματιστή σε επιφάνεια της σκηνής. *(βλέπε κώδικα στο κεφάλαιο 10 παράδειγμα 3).*

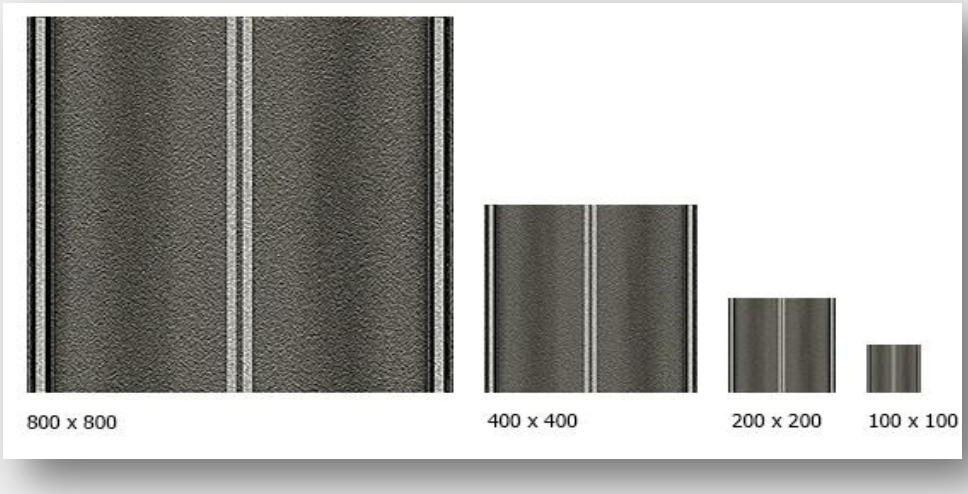
7.4. Πυραμίδες μητρώων υφής (MipMaps)

Όπως προαναφέραμε, όταν μια υφή αποδίδεται σε μια επιφάνεια που εκτείνεται σε μικρότερο εύρος pixels από τον αριθμό των texels που περιέχει το μητρώο στοιχείων υφής, τότε η υφή σμικρύνεται. Ωστόσο η απόδοση ενός μητρώου υφής ενός μητρώου υφής σε επιφάνειες μικρότερου εμβαδού οδηγεί στην παραμόρφωση της απεικόνισης ιδιαίτερα σε κινούμενες εικόνες που η επιφάνεια απομακρύνεται από τον παρατηρητή. Στην περίπτωση αυτή γίνεται ορατό ένα φαινόμενο τρεμοπαίγματος.

Τα φαινόμενα αλλοίωσης λόγω σμίκρυνσης υφής είναι ιδιαίτερα εμφανή σε επίπεδες επιφάνειες οι οποίες προβάλλονται στο επίπεδο του παρατηρητή βάσει προοπτικής προβολής (βλεπε ενότητα 4.3.1) , εκτείνονται από τη θέση του παρατηρητή προς το βάθος της σκηνής στις οποίες αποδίδονται υφές που περιέχουν ευθείες.

Για την απόδοση υφής σε επιφάνειες οι οποίες βρίσκονται μακρία από τον παρατηρητή, η μηχανή της OpenGL προβλέπει επιπρόσθετα την χρήση μιας πυραμίδας μητρώων υφής. Η πυραμίδα μητρώων υφής παράγεται εξάγοντας βαθμίδες (levels), δηλαδή προσεγγίσεις της αρχικής υφής που παράγονται με διαδοχικές υποδιαιρέσεις της αρχικής ανάλυσης.

Έτσι, για ένα μητρώο υφής με m x n texels, εξάγουμε μια πυραμίδα μητρώων υφής με αναλύσεις m/2 x n/2, m/4 x n/4 και ούτω καθεξής (Εικόνα 7.2).



Εικόνα.7.2: Πυραμίδα μητρώων υφής

Όταν η μηχανή της OpenGL κληθεί να αποδώσει σε μια επιφάνεια το μητρώο υφής, έχοντας ήδη εξαγάγει μια πυραμί-δα εξομαλυσμένων προσεγγίσεων, επιλέγει τη βαθμίδα με τις πλησιέστερες διαστάσεις. Η λειτουργία αυτή ονομάζεται **mipmapping** και εφαρμόζεται μόνο σε περιπτώσεις σμίκρυνσης (minification).
Η χρησιμότητά της έγκειται στο ότι, κατά την απόδοση υφής σε μικρές επιφάνειες, εφαρμογή των αλγορίθμων προσέγγισης σε εξομαλυσμένες βαθμίδες μικρότερης ανάλυσης είναι περισσότερο αποδοτική σε σχέση με την εφαρμογή τους στο αρχικό (μεγάλων διαστάσεων) μητρώο υφής.

7.4.1. Κατασκευή πυραμίδας μητρώων υφής

Η Κατασκευή πυραμίδας εκτελείται χρησιμοποιώντας την εντολή gluBuild1DMipmaps (για πυραμίδα μονοδιάστατης υφής) ή την εντολή gluBuild2DMipmaps (για πυραμίδα διδιάστατης υφής). Οι εντολές αυτές περιέχονται στη βιβλιοθήκη GLU.

```
int gluBuild1DMipmaps( GLenum target ,
                       GLint components,
                       GLint width,
                       GLenum format,
                       GLenum type,
                       const void *texelArray );

int gluBuild2DMipmaps( GLenum target ,
                       GLint components,
                       GLint width,
                       GLint height,
                       GLenum format,
                       GLenum type,
                       const void *texelArray );
```

όπου target η τιμή GL_TEXTURE_1D ή GL_TEXTURE_2D (ανάλογα με τη διάσταση της υφής), components το πλήθος των συνιστωσών που προσδιορίζουν τη χρωματική τιμή ενός texel, width το πλάτος της υφής σε texels, height το ύψος της υφής σε texels, format η διαδοχή με την οποία δίνονται οι χρωματικές συνιστώσες, type ο πρωτογενής τύπος δεδομένων που χρησιμοποιείται για την περιγραφή των χρωματικών τιμών και texelArray το μητρώο τιμών.

Η ενεργοποίηση της λειτουργίας γίνεται προσδιορίζοντας την παράμετρο GL_TEXTURE_MIN_FILTER με την εντολή glTexParameteri.

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

Οι προβλεπόμενες τιμές (parameterValue) των παραμέτρων GL_TEXTURE_MAG_FILTER για την εφαρμοφή mipmapping είναι οι εξής:

GL_LINEAR και
GL_NEAREST οι οποίες αναλύονται 7.2.

Οι προβλεπόμενες τιμές (parameterValue) των παραμέτρων GL_TEXTURE_MIN_FILTER για την εφαρμοφή mipmapping είναι οι εξής:

GL_NEAREST_MIPMAP_NEAREST: Επιλέγεται η βαθμίδα της πυραμίδας με τις πλησιέστερες διαστάσεις. Η προσέγγιση των νοητών στοιχείων υφής βασίζεται στην επιλογή του πλησιέστερου γείτονα.

GL_LINEAR_MIPMAP_NEAREST: Επιλέγεται η βαθμίδα της πυραμίδας με τις πλησιέστερες διαστάσεις. Τα νοητά στοιχεία υφής προσεγγίζονται βάσει γραμμικής παρεμβολής.

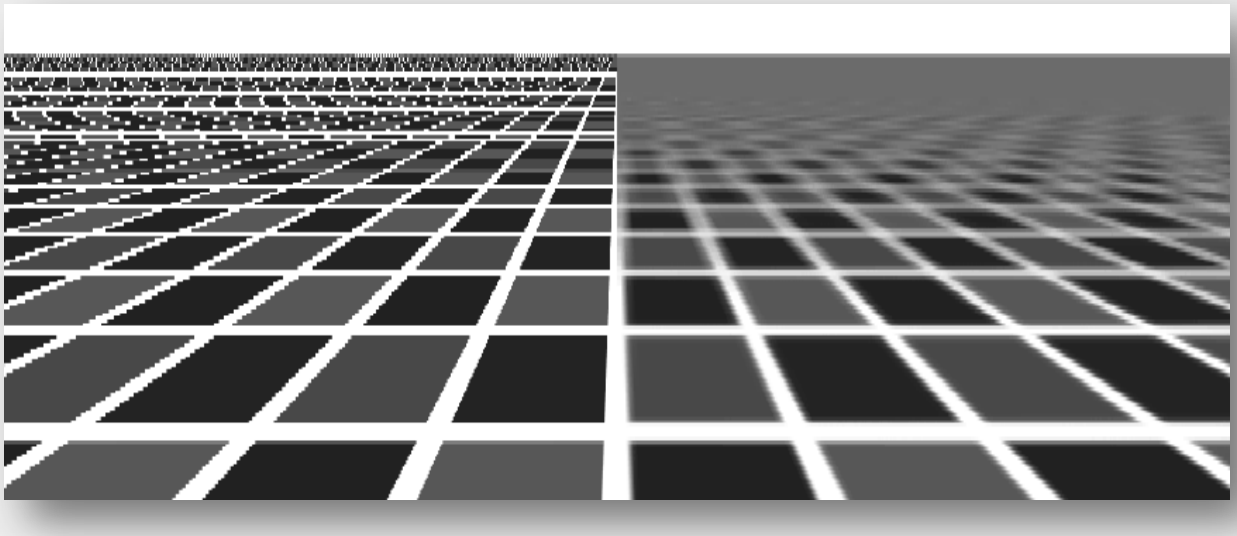
GL_NEAREST_MIPMAP_LINEAR: Επιλέγουμε τις δύο βαθμίδες της πυραμίδας με τις πλησιέστερες διαστάσεις. Για την προσέγγιση υφής, επιλέγουμε από κάθε μία βαθμίδα ξεχωριστά το πλησιέστερο texel. Η χρωματική τιμή που αποδίδουμε στο pixel της επιφάνειας προκύπτει με γραμμική παραμβολή από τις χρωματικές τιμές των δύο texels που επελέγησαν από κάθε μία βαθμίδα.

GL_LINEAR_MAP_LINEAR: Επιλέγουμε τις δύο βαθμίδες της πυραμίδας με τις πλησιέστερες διαστάσεις. Σε περιπτώσεις προγέγγισης υφής προσεγγίζουμε σε κάθε μία βαθμίδα ξεχωριστά την τιμή ενός νοητού texel βάσει γραμμικής παρεμβολής. Η χρωματική τιμή που αποδίδουμε στο pixel της επιφάνειας προκύπτει με γραμμική παραμβολή από τις χρωματικές τιμές των δύο νοητών texels που εκτιμήθηκαν σε κάθε βαθμίδα.

Στο παρακάτω παράδειγμα παρουσιάζεται ο κώδικας κατασκεύης, φόρτωσης και απόδοσης mipmap υφών.

```
GLuint loadMipmappedTexture3(Image* image3)
{
	GLuint textureId3;
	glGenTextures(1, &textureId3);
	glBindTexture(GL_TEXTURE_2D, textureId3);
	gluBuild2DMipmaps(GL_TEXTURE_2D,
			  GL_RGB,
			  image3->width, image3->height,
			  GL_RGB,
			  GL_UNSIGNED_BYTE,
			  image3->pixels);
	return textureId3;
}


GLuint _textureId3;


void  Loadtexture()
{
	Image* image3 = loadBMP("Texures/3.bmp");
	 _textureId3 = loadMipmappedTexture3(image3);
	delete image3;
	}

void  Surface()
{
	glBindTexture(GL_TEXTURE_2D, _textureId3);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
	glTexParameteri(GL_TEXTURE_2D,
			GL_TEXTURE_MIN_FILTER,
			GL_LINEAR_MIPMAP_LINEAR);

	glBegin(GL_QUADS);

	glTexCoord2f(0, 0);glVertex2f( 0, 0);
	glTexCoord2f(1, 0);glVertex2f( 0, 10);
	glTexCoord2f(1, 1);glVertex2f(10, 10);
	glTexCoord2f(0, 1);glVertex2f(10, 0);

	glEnd();
	}
```

Εικόνα.7.3: Left (mipmaps off), right (mipmaps on)



CHAPTER VIII

ΜΙΞΗ ΧΡΩΜΑΤΩΝ - ΔΙΑΦΑΝΕΙΑ

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 έχουμε τη δυνατότητα να αναμίξουμε τις χρωματικές τιμές τους και να παραγάγουμε έναν ενδιάμεσο χρωματισμό στα κοινά σημεία τους. Με τον τρόπο αυτό μπορούμε όπως λ.χ. να προσομοιώσουμε διαφανείς ή ημιδιαφανείς επιφάνειες. Για τη μίξη χρησιμοποιείται ευρέως το χρωματικό μοντέλο RGBA. Αυτό το μοντέλο ορίζουμε για κάθε χρώμα τις τρεις συνιστώσες του και μία τέταρτη συνιστώσα ,τον συντελεστή alpha, ο οποίος χρησιμοποιείται ως συντελεστής μίξης.

Στη μίξη χρωμάτων ορίζουμε δύο στρώματα: το στρώμα προορισμού (destination) και το στρώμα πηγής (source) Με τον όρο στρώμα προορισμού αναφερόμαστε 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.

Προκειμένου να εκτελεστεί η μίξη χρωμάτων ορίζουμε τους συντελεστές μίξης για την πηγή και τον προορισμό. Οι συντελεστές μίξης καθορίζουν σε τι ποσοστό θα συμμετάσχουν οι χρωματικές τιμές του στρώματος προορισμού και οι χρωματικές πηγές του στρώματος πηγής στη σκηνή που θα προκύψει μετά την υπέρθεσή τους.
Για τη μίξη χρωμάτων στην OpenGL, αρχικά απαιτείται η ενεργοποίηση της λειτουργίας με την εντολή:

```
glEnable(GL_BLEND);
```

Κατόπιν ορίζουμε τους συντελεστές μίξης που αντιστοιχίζονται στο στρώμα πηγής και στο στρώμα προορισμού με την εντολή glBlendFunc:

```
void glBlendFunc(GLenum sFactor, GLenum dFactor);
```

όπου sFactor και dFactor παράμετροι που καθορίζουν τους συντελεστές μίξης για το στρώμα πηγής και το στρώμα προορισμού αντίστοιχα. Καθορίζονται με τις εξής αριθμητικές σταθερές:

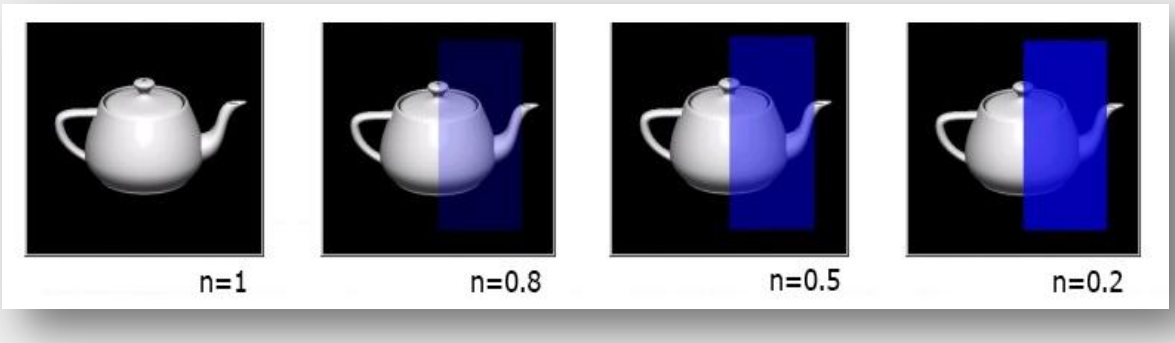
Constant	Relevant Factor	Computed Blend Factor
GL_ZERO	source or destination	(0, 0, 0, 0)
GL_ONE	source or destination	(1, 1, 1, 1)
GL_DST_COLOR	source	(Rd, Gd, Bd, Ad)
GL_SRC_COLOR	destination	(Rs, Gs, Bs, As)
GL_ONE_MINUS_DST_COLOR	source	(1, 1, 1, 1)-(Rd, Gd, Bd, Ad)
GL_ONE_MINUS_SRC_COLOR	destination	(1, 1, 1, 1)-(Rs, Gs, Bs, As)
GL_SRC_ALPHA	source or destination	(As, As, As, As)
GL_ONE_MINUS_SRC_ALPHA	source or destination	(1, 1, 1, 1)-(As, As, As, As)
GL_DST_ALPHA	source or destination	(Ad, Ad, Ad, Ad)
GL_ONE_MINUS_DST_ALPHA	source or destination	(1, 1, 1, 1)-(Ad, Ad, Ad, Ad)
GL_SRC_ALPHA_SATURATE	source	(f, f, f, 1); f=min(As, 1-Ad)

Με τη μίξη χρωμάτων έχουμε τη δυνατότητα να προσομοιώσουμε φαινόμενα διαφάνειας. Στην περίπτωση αυτή, η διαφάνεια μιας επιφάνειας καθορίζεται με τη χρήση της alpha συνιστώσας της. Ορίζουμε μια επιφάνεια ως πλήρως διαφανή για τιμή alpha ίση με 1 και πλήρως αδιαφανή για τιμή alpha ίση με 0 και προκειμένου να χρησιμοποιήσουμε τη συνιστώσα alpha ως συντελεστή διαφάνειας, καθορίζουμε το μοντέλο μίξης ως εξής

```
glBlendFunc(ONE_MINUS_SRC_ALPHA,GL_SRC_ALPHA);
```

Με τη ρύθμιση αυτή μεγάλες τιμές του συντελεστή (κοντά στη μονάδα) αναδεικνύουν σε μεγαλύτερο βαθμό το χρώμα του στρώματος προορισμού και υποβιβάζουν την απόδοση του στρώματος πηγής. Δηλαδή η τιμή alpha λειτουργεί ως συντελεστής διαφάνειας (n).

Εικόνα 8.1: Διαφάνεια



Στην εικόνα 8.1 απεικονίζεται η έτοιμη ρουτίνα της βιβλιοθήκης glut για υλοποίηση μιας τσαγιέρας και μπροστά από την τσαγιερά ένα μπλε ορθογώνιο με διαφορετικούς συντελεστές διαφάνειας n για την κάθε μια εικόνα ξεχωριστά. Όπως βλέπουμε για τιμές n κοντά στο 0 το ορθογώνιο γίνεται πλήρως αδιαφανές και για τιμές n κοντά στο 1 πλήρως διαφανές. Βλέπε κώδικα στο κεφάλαιο 12 παράδειγμα 2.

8.1. Ταξινόμιση βάθους (Depth shorting)

Όταν δοκιμάσουμε να χρησιμοποιήσουμε την τεχνική μείξης χρωμάτων σε μια πλήρη 3D σκηνή για να δημιουργήσουμε την αίσθηση της διαφάνειας, θα διαπιστώσουμε μερικά προβλήματα.

Καταρχάς αν θυμηθούμε πως δουλεύει ο ενταμιευτης βάθους (z-buffer), για κάθε pixel που θέλουμε να σχεδιάσουμε στον framebuffer ελέγχεται το βάθος του σε σχέση με το βάθος του pixel που είναι ήδη στον framebuffer.Αυτό σημαίνει ότι αν πάμε να σχεδιάσουμε ένα ημιδιάφανο ή πλήρως διάφανο πολύγωνο, και μετά ένα άλλο συμπαγές πολύγονο πίσω από αυτό, το δεύτερο δεν θα σχεδιαστεί, αφού θα το κρύβει το πρώτο άσχετα αν είναι διαφανές η όχι, και έτσι θα χαλάσει η ψευδαίσθηση τις διαφάνειας.

Για να αποφύγουμε αυτό το πρόβλημα, πρέπει να σχεδιάσουμε τα ημιδιάφανα αντικείμενα της σκηνής και μετα τα μη-διαφανή.

Ο παρακάτω κώδικας απεικονίζει δυο πανομοιότυπες σκηνές (σκηνή 1 και σκηνή 2) στον τρισδιάστατο χώρο αποτελούμενες από μια τσαγιέρα και ένα μπλε ορθογώνιο εκ τον οποίων το ορθογώνιο είναι διαφανές. Η διαφορά είναι ότι στην σκηνή 1 σχεδιάζουμε πρώτα την τσαγιέρα (correct blending) ενώ στην σκηνή 2 πρώτα το ορθογώνιο. Τα αποτελέσματα φαίνονται στην εικόνα 8.2.

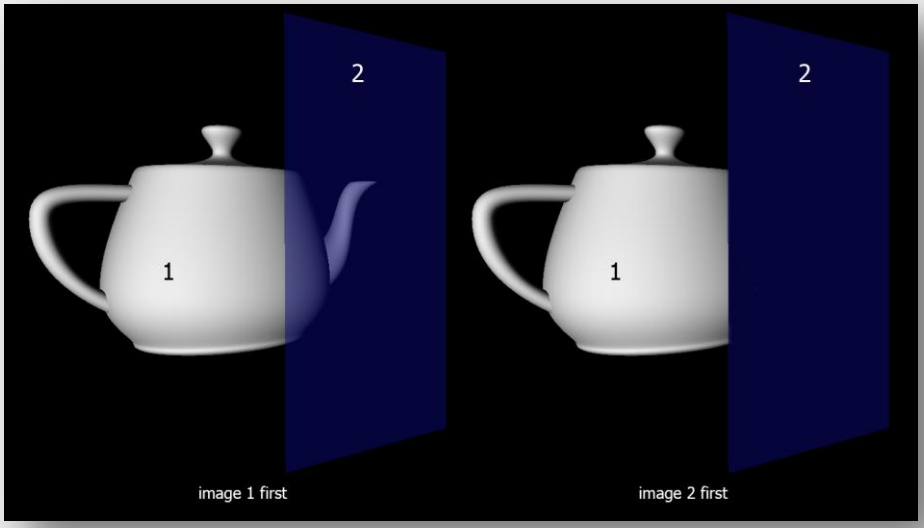
ΣΚΗΝΗ 1

```
glPushMatrix();
glTranslatef(0, 0, -50);                          μετατόπιση κατά -50 στον άξονα των z
GLfloat white[]={1.0, 1.0, 1.0, 1.0};             ορισμός χρώματος επιφάνειας τσαγιέρας
glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,white);
glutSolidTeapot(36);                              ρουτίνα της glut για σχεδίση της τσαγιέρας
glPopMatrix();


glPushMatrix();
glEnable(GL_BLEND);                               ενεργοποίηση της λειτουργίας μείξης χρωμάτων
glBlendFunc(GL_ONE_MINUS_SRC_ALPHA,GL_SRC_ALPHA);
GLfloat blue[]={0.0, 0.0, 1.0, 0.5};              ορισμός χρώματος επιφάνειας ορθογωνίου
glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,blue);
glBegin(GL_QUADS);                                ρουτίνα της gl για σχεδίση του ορθογωνίου
glVertex3f(0,  -50, 0);                           σχεδίση ορθογωνίου κατα 0 στον άξονα των z
glVertex3f(60, -50, 0);
glVertex3f(60,  50, 0);
glVertex3f(0,   50, 0);
glEnd();
glDisable(GL_BLEND);                              απενεργοποίηση της λειτουργίας μείξης χρωμάτων
glPopMatrix();
```

ΣΚΗΝΗ 2

```
glPushMatrix();
glEnable(GL_BLEND);                               ενεργοποίηση της λειτουργίας μείξης χρωμάτων
glBlendFunc(GL_ONE_MINUS_SRC_ALPHA,GL_SRC_ALPHA);
GLfloat blue[]={0.0, 0.0, 1.0, 0.5};              ορισμός χρώματος επιφάνειας ορθογωνίου
glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,blue);
glBegin(GL_QUADS);                                ρουτίνα της gl για σχεδίση του ορθογωνίου
glVertex3f(0,  -50, 0);                           σχεδίση ορθογωνίου κατα 0 στον άξονα των z
glVertex3f(60, -50, 0);
glVertex3f(60,  50, 0);
glVertex3f(0,   50, 0);
glEnd();
glDisable(GL_BLEND);                              απενεργοποίηση της λειτουργίας μείξης χρωμάτων
glPopMatrix();


glPushMatrix();
glTranslatef(0, 0, -50);                          μετατόπιση κατά -50 στον άξονα των z
GLfloat white[]={1.0, 1.0, 1.0, 1.0};             ορισμός χρώματος επιφάνειας τσαγιέρας
glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,white);
glutSolidTeapot(36);                              ρουτίνα της glut για σχεδίση της τσαγιέρας
glPopMatrix();
```

Εικόνα 8.2: Depth shorting (Σκηνή 1 αριστερά και σκηνή 2 δεξιά)

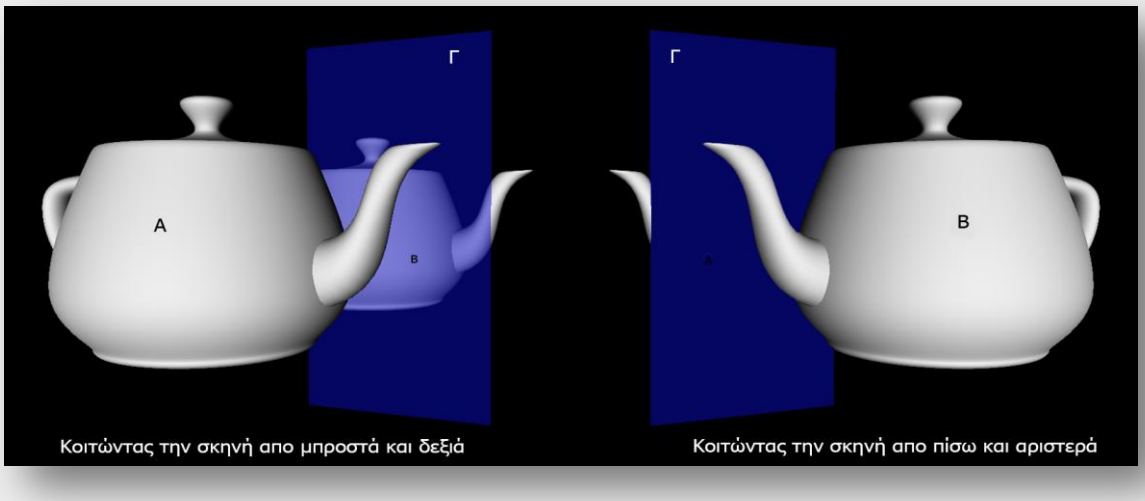


Αλλά αυτό δεν μας λύνει το πρόβλημα όταν έχουμε την δυνατότητα ελεύθερης περιήγησης στην σκηνή. Το επιθυμητό αποτέλεσμα της διαφάνειας παράγεται στην σκηνή 1 μόνο όταν κοιτάμε την σκηνή από μπροστά. Τι θα γινόταν όμως αν κοιτούσαμε την σκηνή από άλλες οπτικές γωνίες;

Στο παρακάτω παράδειγμα τροποποιούμε το κώδικα της σκηνής 1 ώστε να απεικονίζει δυο τσαγιέρες και ανάμεσα το διαφανές μπλε ορθογώνιο. Επίσης έχουμε την δυνατότητα ελεύθερης περιήγησης στο 3δ χώρο. Η σειρά με την οποία είναι αποθηκευμένα στο depth buffer είναι Β, Γ, Α.

Το αποτέλεσμα φαίνεται στην εικόνα 8.3.

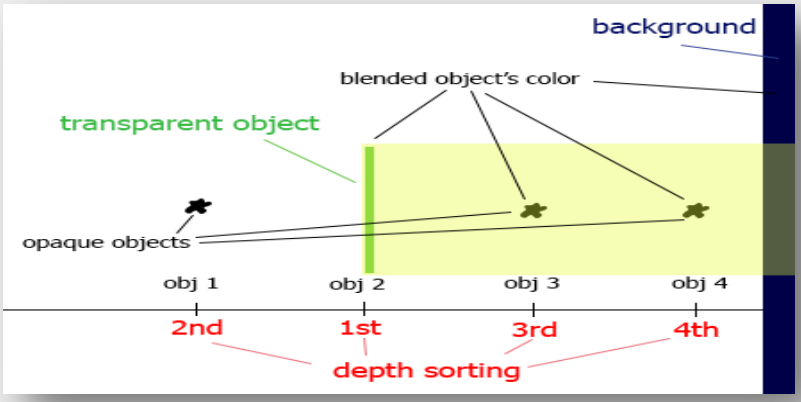
Εικόνα 8.3: Depth shorting με ελεύθερη περιήγηση στο 3δ χώρο.



Όπως βλέπουμε στην εικόνα 8.3 στην αριστερή αναπαράσταση της σκηνής έχουμε τις δυο τσαγιέρες Α και Β και ανάμεσα τους το διαφανές αντικείμενο Γ μέσα από το οποίο μπορούμε να δούμε την τσαγιέρα Β και όλα τα αντικείμενα που έχουν ή θα σχεδιαστούν μετά από αυτό. Κοιτάζοντας όμως την σκηνή από άλλη οπτική γωνία όπως φαίνεται στην δεξιά αναπαράσταση της σκηνής η διαφανής μπλε επιφάνεια καλύπτει την τσαγιέρα Β γιατί τα αντικείμενα αυτά έχουν ταξινομηθεί με τέτοια σειρά στον depth buffer ώστε η επιφάνεια Γ να κάνει μίξη χρωμάτων μόνο με την τσαγιέρα Β και όχι με την Α.
Οπότε για να καλύψουμε αυτή την περίπτωση πρέπει να ταξινομήσουμε σωστά τα αντικείμενα ώστε από οποιαδήποτε οπτική γωνία και να κοιτάξουμε την σκηνή, η μπλε επιφάνεια να έχει εφέ διαφάνειας.

Η σωστή ταξινόμηση βάθους είναι Β, Α, Γ..

Σχήμα 8.4: Blending and Depth shorting



Σύμφωνα με το παραπάνω σχήμα 8.4 από όποια μεριά και να δούμε την σκηνή, τα μαύρου χρώματος opaque objects θα φαίνονται μέσα από το πράσινο transparent object.

8.2. Η Συνιστώσα Άλφα

Όπως έχουμε δει σε προηγούμενη ενότητα η συνιστώσα άλφα η οποία είναι το τέταρτο όρισμα για τον καθορισμό του χρώματος μιας επιφάνειας έχει τον ρόλο του συντελεστή διαφανειας. Επιφάνειες με άλφα τιμή ίση με 1 καθίστανται ως πλήρως διάφανες και με άλφα τιμή ίση με 0 πλήρως αδιάφανες. Αλλά παρόλο την διαφάνεια τους ο depth buffer γνωρίζει την υπαρξη τους και καταγραφεί το βάθος τους στο χωρο.
Αυτό σε μια σταθερή σκηνή χωρίς ελεύθερη περιήγηση δεν αποτελεί πρόβλημα εφόσον έχει γίνει σωστή ταξινόμηση βαθους. Αλλα σε μια ρεαλιστικότερη σκηνή με πολλά διάφανα και αδιάφανα αντικείμενα ή ακόμη και με επιφάνειες αποτελούμενες και από διάφανα και από αδιάφανα τμήματα, μια σωστή ταξινόμηση βάθους δεν μπορεί να γίνει.

Για να καλύψουμε και αυτή την περίπτωση πρέπει να φροντίσουμε να μην ζωγραφίζουμε καν τα σχεδόν ή εντελώς διαφανή αντικείμενα ή κομμάτια αντικειμένων, ώστε να μην γράφεται το βάθος τους στον z buffer. Αυτό το πετυχαίνουμε με το alpha testing.

```
void glAlphaFunc(GLenum  func,  GLclampf  ref);
```

Func	Ref	Description
GL_NEVER	0 έως 1	Pixels never passes the test
GL_LESS	0 έως 1	Only pixels with value less than the ref value passes the test
GL_EQUAL	0 έως 1	Only pixels with value equal to the ref value passes the test
GL_LEQUAL	0 έως 1	Only pixels with value less or equal to the ref value passes the test
GL_GREATER	0 έως 1	Only pixels with value greater from the ref value passes the test
GL_NOTEQUAL	0 έως 1	Only pixels with value not equal to the ref value passes the test
GL_GEQUAL	0 έως 1	Only pixels with value grater or equal to the ref value passes
GL_ALWAYS	0 έως 1	Pixels always passes the test

```
glEnable(GL_ALPHA_TEST);
glAlphaFunc(GL_GREATER, 0.5);
```

Με τον παραπάνω κώδικα λέμε στην OpenGL να απορρίψει οποιοδήποτε pixel πάμε να ζωγραφίσουμε που έχει τιμή alpha κάτω από 0.5.

Η τεχνική alpha testing σε συνδυασμό με την μείξη χρωμάτων και την απόδοση υφής χρησιμοποιείται κυρίως σε επιφάνειες οι οποίες αποτελούνται και από διάφανα και από αδιάφανα pixels.
Στην OpenGl και γενικώς στον κόσμο των 3Δ γραφικών ο παραπάνω συνδυασμός χρησιμοποιείται για την απόδοση πολύπλοκων μορφολογικά αντικειμένων όπως ένα δέντρο.

8.3. Κατασκευή ημιδιαφάνους επιφάνειας με υφή δέντρου

Η διαδικασία σχετικά με την προετοιμασία και απόδοση ενός δέντρου με την χρήση του παραπάνω συνδυασμού φαίνεται στα παρακάτω βήματα.

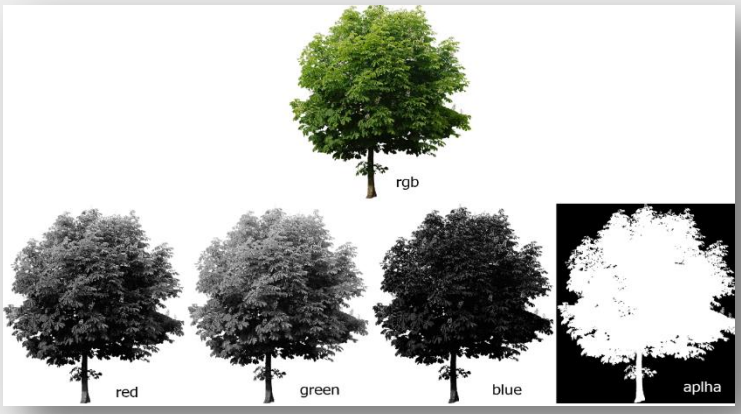
Βήμα 1ο: Κατασκευή ορθογωνίου χρησιμοποιώντας την glQuads.

Βήμα 2ο: Δημιουργία υφής δέντρου με δυνατότητα διαφάνειας.

Υφές με δυνατότητα διαφάνειας στην ουσία είναι πρότυπα εικόνων στα οποία έχει ορισθεί, εκτός των τριών βασικών χρωματικών συνιστωσών (κόκκινο, κίτρινο, μπλε) και η τέταρτη συνιστώσα άλφα. Τέτοια πρότυπα εικόνων είναι της μορφής TGA (βλέπε tga loader στο κεφάλαιο 10) και είναι δυνατό να επεξεργαστούμε την άλφα συνιστώσα της εικόνας ανοίγοντας την με διάφορους επεξεργαστές εικόνας όπως το Photoshop. (βλέπε photoshop tutorial για ορισμό της άλφα συνιστώσας στο κεφάλαιο 12).

Στην εικόνα 8.4 αναλύουμε μια υφή δέντρου στης τέσσερις χρωματικές συνιστώσες της.

Εικόνα 8.4: Υφή δέντρου



Όπως βλέπουμε στην εικόνα η alpha συνιστώσα της υφής αποτελείται από άσπρα και μαύρα pixels. Τα μαύρα pixels έχουν συντελεστή διαφάνειας ίσο με 1 οπότε κατά την απόδοση τους θα είναι πλήρως διάφανες και τα άσπρα συντελεστή διαφάνειας ίσο με 0, δλδ πλήρως αδιάφανες.

Βήμα 3ο: Απόδοση της υφής στο παραπάνω ορθογώνιο.

Βήμα 4ο: Χρήση του alpha test για την απόρριψη των pixels που έχουν ορισθεί ως διαφανές.

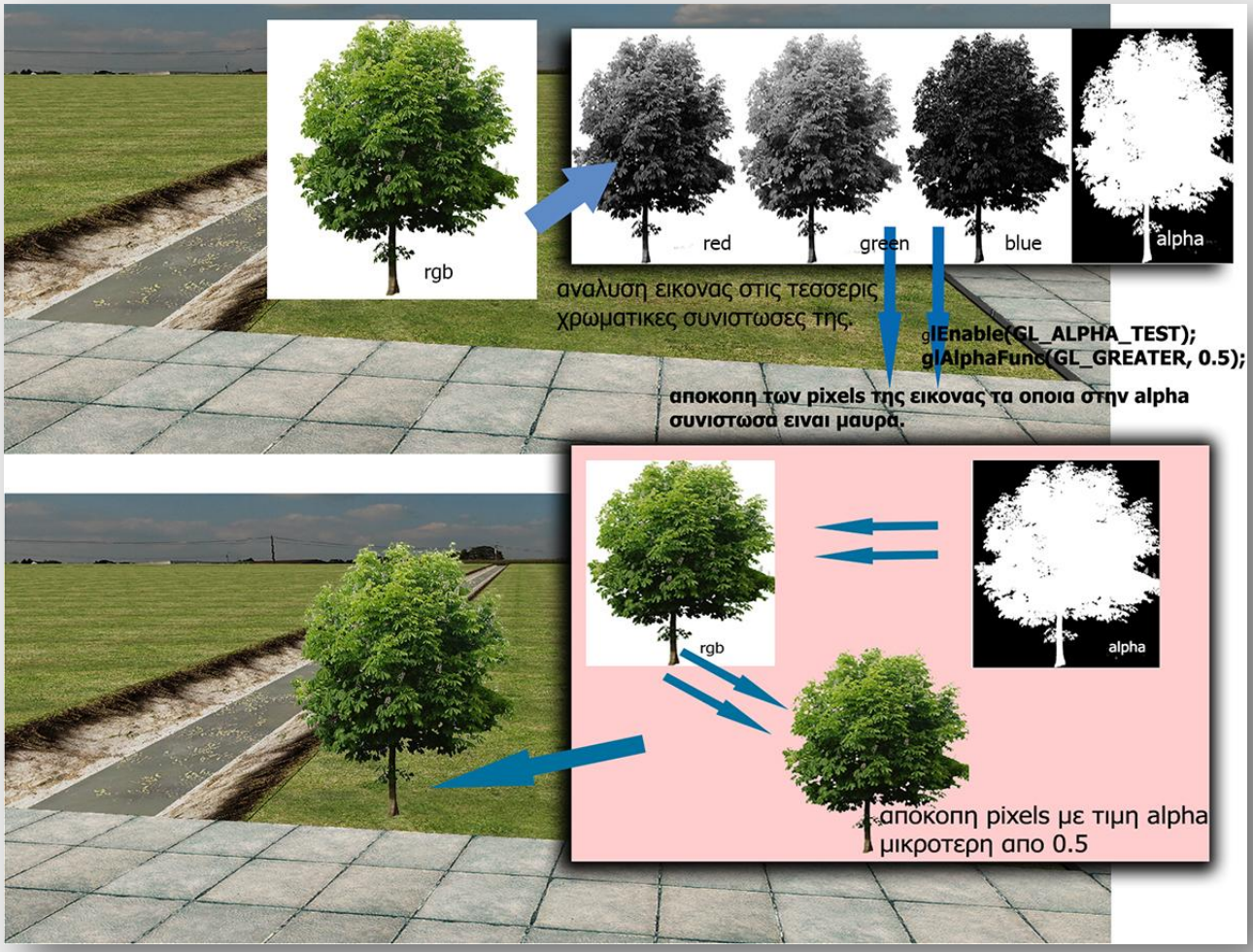
```
glEnable(GL_ALPHA_TEST);
glAlphaFunc(GL_GREATER, 0.5);
```

Ο παραπάνω κώδικάς θα απορρίψει όσα pixels έχουν συντελεστή διαφάνειας 1, δλδ τα μαύρα pixels της άλφα συνιστώσας

Ο παρακάτω κώδικας υλοποιεί τα τέσσερα παραπάνω βήματα.

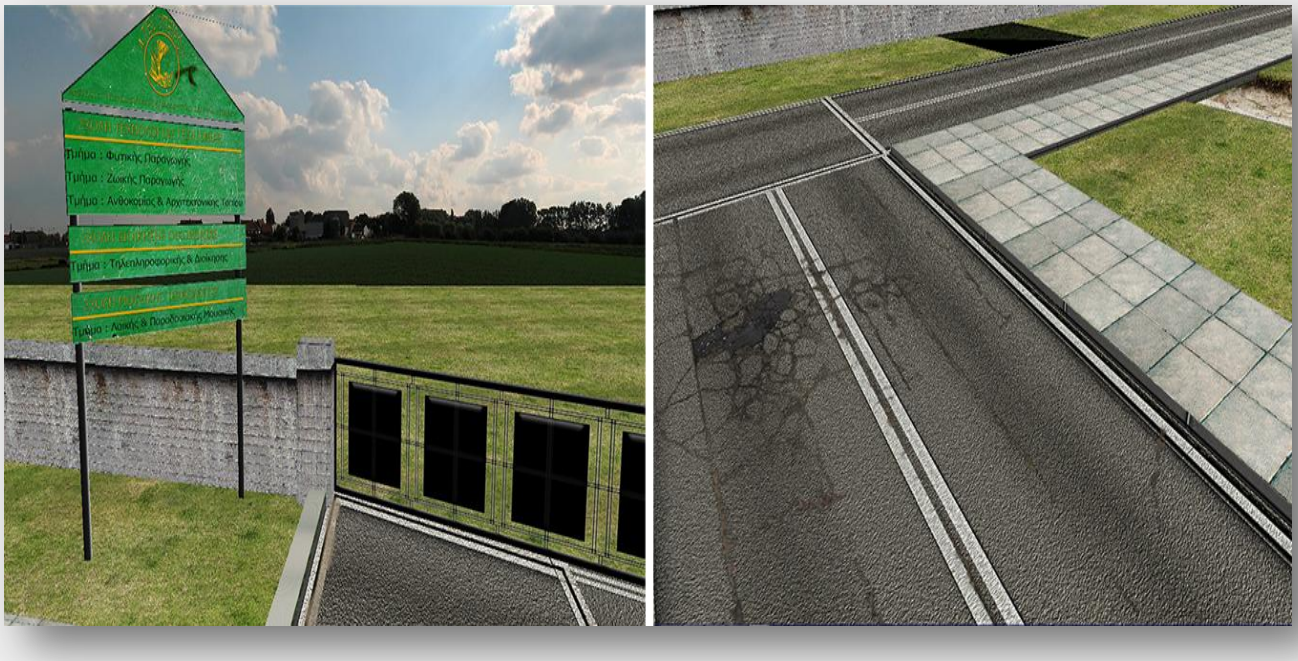
```
glPushMatrix();
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glEnable(GL_ALPHA_TEST);
glAlphaFunc(GL_GREATER, 0.5);
glBindTexture(GL_TEXTURE_2D, textura[1].texID);

glBegin(GL_QUADS);
glNormal3f(1, 1, 1);glTexCoord2f(0,0);glVertex3f( 0.0f,  0.0f,  0.0f);
glNormal3f(1, 1, 1);glTexCoord2f(1,0);glVertex3f(10.0f,  0.0f,  0.0f);
glNormal3f(1, 1, 1);glTexCoord2f(1,1);glVertex3f(10.0f,  0.0f, 10.0f);
glNormal3f(1, 1, 1);glTexCoord2f(0,1);glVertex3f( 0.0f,  0.0f, 10.0f);
glEnd();

glBindTexture(GL_TEXTURE_2D,0);
glDisable(GL_BLEND);
glDisable(GL_ALPHA_TEST);
glPopMatrix();
```

Εικόνα 8.5: alpha blended tree



Τα αντικείμενα των παρακάτω εικόνων έχουν υλοποιηθεί βάση της ίδιας τεχνικής.

Εικόνα 8.6: alpha blended objects (sign, black gate and cracks on street)



CHAPTER IX

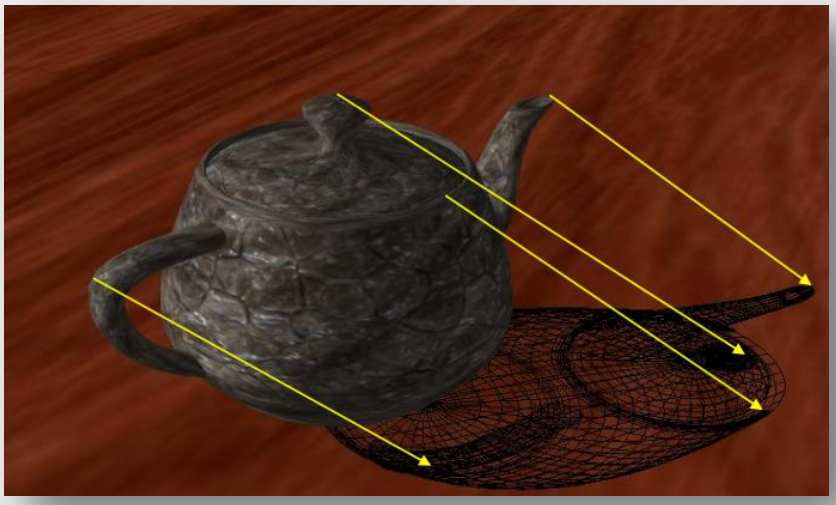
ΣΚΙΑΣΗ ΑΝΤΙΚΕΙΜΕΝΩΝ

Ο Frank Crow παρουσίασε για πρώτη φορά την ιδέα χρησιμοποίησης τομέων σκίασης (shadow volumes) για την απόδοση σκιών (shadow casting) των αντικείμενων το 1977. Ο Tim Heidmann της Silicon Graphics εφάρμοσε τους τομείς σκίασης του Crow αξιοποιώντας τον **stencill buffer** για την καλύτερη υλοποίηση των τομέων σκίασης.
Ας ρίξουμε μια ματιά στο πώς λειτουργεί η πρωτότυπη τεχνική τομέων σκίασης του Crow.

Για να κατασκευάσει ένας τομέας σκίασης, το βασικό προαπαιτούμενο είναι η υλοποίηση μιας πηγής φωτός. Οι ακτίνες της πηγής αντανακλώνται από κάθε κορυφή του αντικειμένου και προβάλουν στην επιφάνεια πίσω από το αντικείμενο την γεωμετρία του. Η προβαλλόμενη γεωμετρία αυτή του αντικειμένου είναι η σκιά του αντικειμένου και ονομάζεται τομέας σκίασης (shadow volume) και έχει τόσα πολύγωνα όσα και το αρχικό αντικείμενο.

Στην εικόνα 9.1 έχουμε ένα αντικείμενο πάνω από μια επιφάνεια με υφή ξύλου και μια πηγή φωτός αριστερά του αντικειμένου όπως φαίνεται από τα κίτρινα βέλη. Η σκιά του αντικειμένου ή αλλιώς η προβαλλομένη γεωμετρία του αναπαριστάται ως wireframe.

Εικόνα 9.1: Shadow Casting



9.1. Ο Stencill Buffer

Οι δοκιμές Stencil (Stencil testing) προϋποθέτουν την ύπαρξη του stencil buffer έτσι οποιαδήποτε εξήγηση για το πώς λειτουργει το stencil test στηρίζεται στην κατανόηση για το τι ειναι ο stencil buffer και ποιες οι λειτουργίες του.

Ο stencil buffer είναι μια ομάδα από bits (π.χ. 0 ή 1) τα οποία αντιστοιχούν σε pixels των προβαλλομένων αντικειμένων και ανάλογα το αποτέλεσμα όσα pixels έχουν stencil value 1 θα προβάλλονται ενώ τα υπολοιπα θα απορρίπτονται. Στην ουσια καλυπτουμε τα pixels που δεν θελουμε να προβληθουν στην τελικη σκηνη (masking) με σκοπο την επιτευξη διαφορων εφε, κατι που θα μας βοηθησει αργοτερα στην αποδοση σκιων.

Το προαπαιτούμενο για την εφαρμογή του stencil buffer είναι η υλοποίηση ενός παραθύρου που υποστηρίζει τον stencil buffer. Συνήθως, πρέπει να καθορίσουμε τον αριθμό των stencil bits που χρειαζόμαστε, επειδή μια εφαρμογή μπορεί να είναι περιορισμένη σε σχέση με τον αριθμό των stencil bits που υποστηρίζει. Σε ένα Win32 OpenGL project, αυτό ΑΠΑΙΤΕΙ τη χρήση της ChoosePixelFormat και PIXELFORMATDESCRIPTOR *(βλέπε κώδικα στο κεφάλαιο 10 παράδειγμα 7)* για να βρείτε μία μορφή εικονοστοιχείου (pixel format) που υποστηρίζει τον stencil buffer.

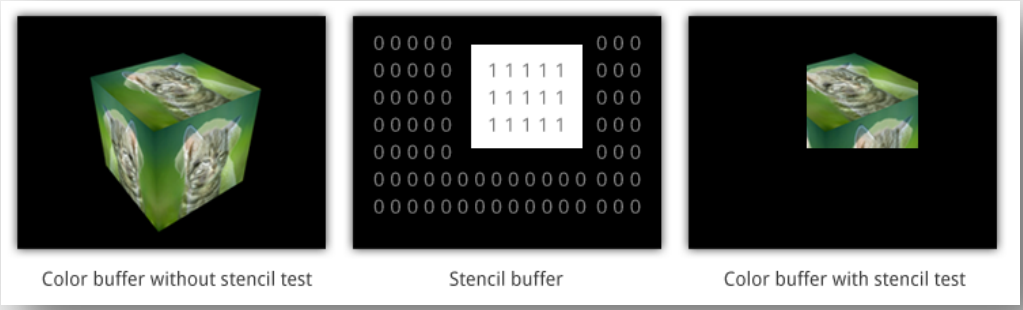
Στην OpenGL, για να ορίσουμε την τιμή καθαρισμού παραθύρου για τον stencil buffer και για να καθαρίσουμε τους color, depth και stencil buffer χρησιμοποιούμε τον παρακάτω κώδικα.

```
glClearStencil(0);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT |GL_STENCIL_BUFFER_BIT);
```

Έπειτα, όπως το depth test έτσι και το stencil test μπορεί είτε να ενεργοποιηθεί, είτε να απενεργοποιηθεί και το αποτέλεσμα του test καθορίζει αν τα pixels θα αποκοπούν ή θα αποδοθούν.
Η ενεργοποίηση και απενεργοποίηση του stencil test γίνεται με τον παρακάτω κώδικα.

```
glEnable(GL_STENCIL_TEST);
glDisable(GL_STENCIL_TEST);
```

Εικόνα 9.2: Ο Stencil Buffer



Το παραδειγμα της εικονας 9.2 είναι ένα απλο παραδειγμα του stencil test. Ο τροπος αποδοσης της τελικης εικονας (δεξια εικονα) φαιναιται στην παρακατω διαδικασια. *(βλέπε κώδικα στο κεφάλαιο 12 παράδειγμα 3).*

- Ενεργοποιουμε το stencil test
- Σχεδιαζουμε, στην προκειμενη περιπτωση, ένα απλο δυσδιαστατο τετραγωνο και το αποθηκευουμε στον stencil buffer με stencil value 1 (μεσαια εικονα).
- Λεμε στο stencil test να αποριψει οσα pixels εχουν stencil value μικροτερη του 1 και σχεδιαζουμε τον κυβο (αριστερη εικονα).
- Απενεργοποιουμε το stencil test.
- Τρεχουμε το project.

Το stencil test ουσιαστικά είναι μια σύγκριση μεταξύ της τιμής που είναι αποθηκευμένη στον stencil buffer για κάθε pixel μιας επιφάνειας και μίας τιμής αναφοράς (stencil reference value) που ορίζουμε κάθε φορά πριν το test και υλοποιείται με τις παρακάτω εντολές.

```
void glStencilFunc(GLenum func, GLint ref, GLuint mask);
```

Η *func* είναι μια συμβολική σταθερά που καθορίζει ποια συνάρτηση θα χρησιμοποιηθεί για το stencil test και η *ref* είναι μια ακέραια τιμή αναφοράς που χρησιμοποιείται για το test. Στον παρακάτω πίνακα φαίνεται αναλυτικά το είδος της σύγκρισης του stencil test για κάθε σταθερά της func.

Πίνακας 9.1: Η glStencilFunc

Func	Ref	Description
GL_NEVER	0 έως 1	Pixels never passes the test
GL_LESS	0 έως 1	Only pixels with value less than the ref value passes the test
GL_EQUAL	0 έως 1	Only pixels with value equal to the ref value passes the test
GL_LEQUAL	0 έως 1	Only pixels with value less or equal to the ref value passes the test
GL_GREATER	0 έως 1	Only pixels with value greater from the ref value passes the test
GL_NOTEQUAL	0 έως 1	Only pixels with value not equal to the ref value passes the test
GL_GEQUAL	0 έως 1	Only pixels with value grater or equal to the ref value passes
GL_ALWAYS	0 έως 1	Pixels always passes the test

```
void StencilOp(GLenum sfail, GLenum dpfail, GLenum dppass);
```

Η sfail καθορίζει την ενέργεια που θα παρθεί όταν το stencil test αποτύχει. Η dpfail καθορίζει την ενέργεια που θα παρθεί όταν το stencil test πέτυχει αλλά το depth test αποτύχει. Το dppass καθορίζει την ενέργεια που θα παρθεί όταν περάσουν και το stencil test και το depth test ή όταν περάσει μόνο το stencil test και είτε δεν υπάρχει ο depth buffer είτε το depth test δεν είναι ενεργοποιημένο. Και τα τρία ορίσματα της glStencilOp δέχονται τις παρακάτω συμβολικές σταθερές και η αρχική σταθερά για όλα είναι η GL_KEEP.

Πίνακας 9.2: Η StencilOp

Func	Description
GL_KEEP	Keeps the current value.
GL_ZERO	Sets the stencil buffer value to 0.
GL_REPLACE	Sets the stencil buffer value to ref, specified by glStencilFunc.
GL_INCR	Increments the current stencil buffer value. Clamps to the maximum representable unsigned value.
GL_DECR	Decrements the current stencil buffer value. to 0.
GL_INVERT	Bitwise inverts the current stencil buffer value.

9.2. Απόδοση σκιών χωρίς την χρήση του Stencill Buffer

Ο αλγόριθμος για την απόδοση σκιών είναι ένα πολύ γνωστό, εύκολο στην εφαρμογή τέχνασμα γραφικών προβάλλοντας σκιές των αυθαιρέτων αντικείμενων πάνω σε επίπεδες επιφάνειες. Η τεχνική και η μαθηματική του βάση περιγράφονται από τον Blinn σε μια κλασική στήλη με τίτλο «Me and My (Fake) Shadow.
Έχοντας δεδομένη την εξίσωση για την απόδοση επιφάνειας που δέχεται σκιές (**Ground plane equation**), και την θέση της πηγής του φωτός (Light position), ένα 4x4 matrix (**Shadow matrix**) μπορεί να κατασκευαστεί το οποίο θα προβάλει την γεωμετρία των αντικειμένων στο προκαθορισμένο έδαφος (ground plane) βασισμένο στην θέση της πηγής φωτός.

Σε ένα απλό μοντέλο σκιών χρησιμοποιούμε ως ground plane equation το κάθετο κανονικό διάνυσμα του εδάφους (ground plane) όπως φαίνεται παρακάτω. Η ground plane equation για πολύπλοκες μορφολογικές επιφάνειες καθώς και η shadowMatrix equation θα αναλυθεί παρακάτω (βλεπε ενότητα 9.3.1).

```
float groundplane[] = {0.0f, 0.0f, 1.0f, 0.0f};  Το κανονικό διάνυσμα του εδάφους
```

Ο αλγόριθμος έχει ως εξής.

Φόρτωσε το modelview matrix με τον μετασχηματισμό προβολής. Για παράδειγμα, δεδομένης της τοποθεσίας που κοιτάμε την σκηνή (eye location), το κέντρο της προβολής (center of viewing) και το διάνυσμα της κάθετης κατεύθυνσης (up direction vector) της gluLookAt, ο μετασχηματισμός μπορεί να ορισθεί ως εξής.

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(eye[0], eye[1], eye[2],
          center[0], center[1], center[2],
          up[0], up[1], up[2]);
```

Δεδομένης της πηγής φωτός και υποθέτοντας ότι το έδαφος που θα προβληθεί η σκιά είναι επίπεδο, ενεργοποιούμε την πηγή φωτός και σχεδιάζουμε το έδαφος και το αντικείμενο με τον depth buffer ενεργό.

```
float lightPosition[4] = { 10, 10, 10, 0 };
float groundPlaneEquation[4] = { 0, 0, 1, 0 };
```
(ενότητα 9.3.2 ανάληση για μη επίπεδες επιφάνειες)

```
glLightfv(GL_LIGHT0, GL_POSITION, lightPosition);
glEnable(GL_LIGHT0);
glEnable(GL_LIGHTING);
drawObject();
drawGroundPlane();
```

Έχοντας δεδομένη την θέση της πηγής φωτός και την εξίσωση του εδάφους (ground plane equation), κατασκευάζουμε το μητρώο σκιών (shadow matrix) χρησιμοποιώντας την ρουτίνα shadowMatrix.

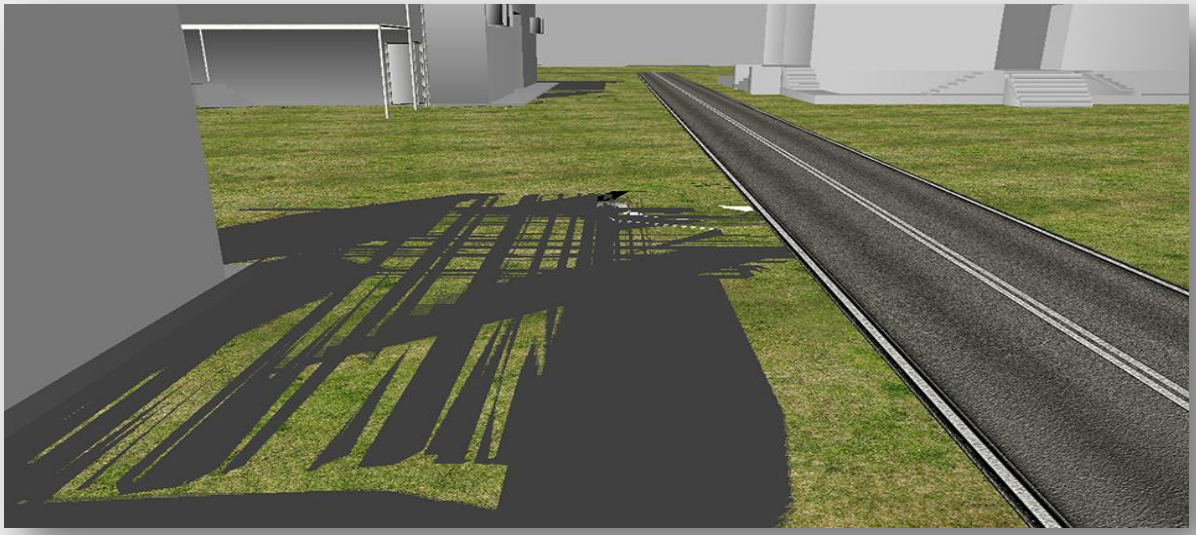
```
glPushMatrix();
GLfloat matrix [4][4];
shadowMatrix(&matrix[0][0], lightPosition, groundPlaneEquation);
```
(ενότητα 9.3.1 ανάληση)
```
glMultMatrixf(&matrix[0][0]);
```

Απενεργοποιούμε το φωτισμό, θέτουμε το χρώμα της σκιάς ως ανοιχτό γκρι και σχεδιάζουμε το αντικείμενο ξανά.

```
glDisable(GL_LIGHTING);
glColor3f(0.25, 0.25, 0.25);
drawObject();
glPopMatrix();
```

Αν δεν κάνουμε κάτι τα πολύγονα του εδάφους και τα πολύγονα των προβαλλομένων σκιών θα είναι συν επίπεδα. Αυτό σημαίνει πως όταν ενεργοποιήσουμε τον depth buffer όλα τα προαναφερόμενα πολύγονα θα έχουν σχεδόν την ιδιά τιμή βάθους προκαλώντας εικονική παραμόρφωση μεταξύ σκιών και εδάφους όπως φαίνεται στην εικόνα 9.3.

Εικόνα 9.3: Εικονικη παραμορφωση σκιων-εδαφους



Η λύση στο πρόβλημα είναι να σηκώσουμε ελαφρώς από το έδαφος την σκιά, τόσο ώστε να μη γίνετε αντιληπτό με ορατό μάτι, ώστε τα πολύγωνα της σκιάς και εδάφους να έχουν διαφορετική τιμή βάθους.
Αυτό είναι δυνατό με την χρήση μιας λειτουργίας της OpenGl, την Polygon offset.

```
void glPolygonOffset(GLfloat factor, GLfloat units);
```

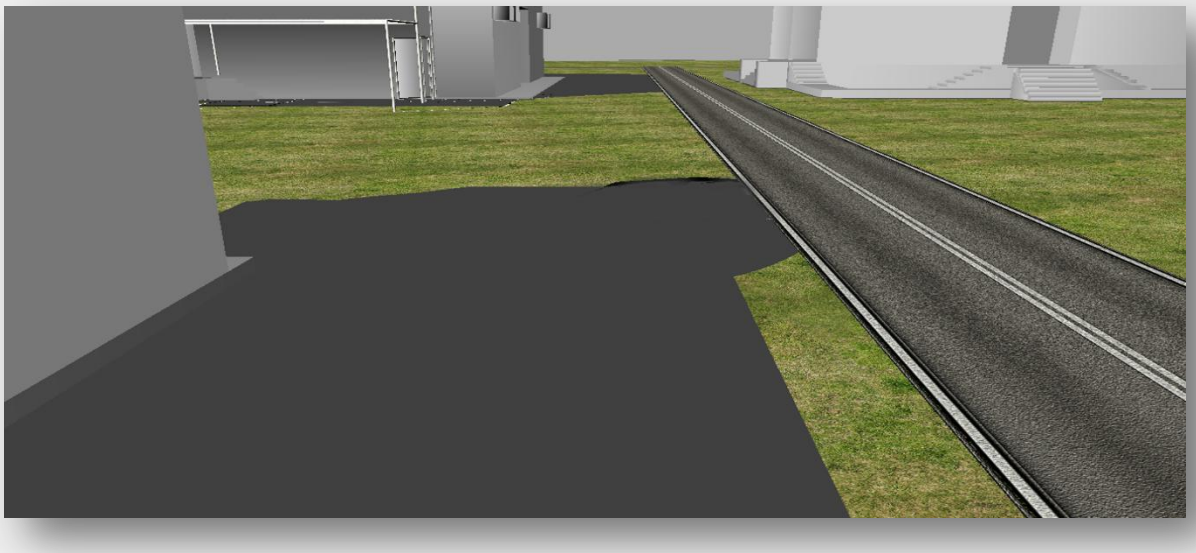
Το όρισμα factor προσδιορίζει μια τιμή κλιμάκωσης για να δημιουργήσει ένα μεταβλητό αντιστάθμισμα βάθους για κάθε πολύγωνο και το όρισμα units για να δημιουργήσει ένα σταθερό αντιστάθμισμα βάθους για κάθε πολύγωνο.

Τοποθετώντας λοιπόν τον παρακάτω κώδικα πριν τη δεύτερη κλήση του αντικειμένου έχουμε το παρακάτω αποτέλεσμα, όπως φαίνεται στην εικόνα 9.4.

```
glEnable(GL_POLYGON_OFFSET_FILL);
glPolygonOffset(-1, -30);
```

Με τον παραπάνω κώδικα ενεργοποιούμε την Polygon offset και ορίζουμε ως factor το -1 και units το -30. Δεν υπάρχει κάποιος κανόνας για την δήλωση των δυο ορισμάτων της Polygon offset, απλά βρίσκεται στην κρίση του προγραμματιστή προς την υλοποίηση του επιθυμητού αποτελέσματος.

Εικόνα 9.4: Απόδοση σκιών με την Polygon offset



Η επίπεδη απόδοση σκιών λειτουργεί αρκετά καλά, αλλά έχει αρκετούς περιορισμούς. Προφανώς, αυτές οι σκιές μπορούν να προβληθούν μόνο σε οριζόντιες και επίπεδες επιφάνειες και όχι σε διάφορά μη επίπεδα αντικείμενα της

σκηνής εξαιτίας της groundplane equation η οποία έχει ορισθεί ως το κανονικό κάθετο διάνυσμα του εδάφους. Αυτά είναι πολύπλοκα θέματα και θα αναλυθούν σε επόμενη ενότητα.
Αποδέχοντας τις σκιές όπως είναι, η τεχνική ακόμη πάσχει ακόμη από μερικές ατελειες. Όπως ορίσαμε προηγούμενος οι σκιές θα αποδίδονται ως γκρι χρώμα κάτι που όπως φαίνεται στις παραπάνω εικόνες δεν είναι καθόλου ρεαλιστικό. Η λύση στο πρόβλημα είναι η μείξη χρωμάτων (βλέπε κεφάλαιο 8).

Με την μίξη χρωμάτων μπορούμε να αναμείξουμε τον χρώμα της σκιάς με οποιαδήποτε υφή έχει ορισθεί για το έδαφος. Δυστυχώς αυτό δεν είναι καθόλου εύκολο. Το πρόβλημα είναι όταν το μητρώο σκιών (shadow matrix) προβάλει τα πολύγωνα ενώς αντικειμένου σε μια επιφάνεια, τα pixels ίσως ανανεωθούν περισσότερο από μια φορά. Αυτό σημαίνει ότι ένα συγκεκριμένο pixel ίσως αναμειχθεί πολλές φορές, σκουραίνοντας το χρώμα του κάθε φορά, με αποτέλεσμα μια σκιά η οποία να είναι πολύ σκοτεινή. Το πρόβλημα αυτό είναι γνωστό ως double blending.

Τοποθετώντας λοιπόν τον παρακάτω κώδικα μετά τον ορισμό του polygon offset έχουμε το παραπάνω αποτέλεσμα, όπως φαίνεται στην εικόνα 9.5.

```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glColor4f(0.0, 0.0, 0.0, 0.5);
```

9.3. Βελτίωση της απόδοσης σκιών με την χρήση του Stencill Buffer

Η τεχνική απόδοσης επίπεδων σκιών μπορεί να βελτιωθεί χρησιμοποιώντας τον stencil buffer με σκοπό το πρόβλημα του double blending θα εξαλειφθεί εντελώς. Η βελτιωμένη τεχνική φαίνεται στα παρακάτω βήματα.

Βήμα 1: Αποδίδουμε στα pixel του εδάφους μια μοναδική stencil value πχ. 3 και το αποθηκεύουμε στον stencil buffer.
Βήμα 2: Προβάλουμε την γεωμετρία, δλδ την σκιά του αντικειμένου στο έδαφος.
Βήμα 3: Αποδίδουμε στα pixels της σκιάς μια μοναδική stencil value πχ. 2 και το αποθηκεύουμε στον stencil buffer.
Βήμα 4: Λέμε στο stencil test να απορρίψει όσα pixels έχουν stencil value 3 και πάνω.Τελικά προβάλλονται μόνο τα pixel με stencil value 2 δλδ τα pixels της σκιάς.
Βήμα 5: Απενεργοποιούμε τον stencil buffer και ξανασχεδιάζουμε το έδαφος και το αντικείμενο.

Ενεργοποιούμε στο stencil test.

```
glEnable(GL_STENCIL_TEST);
```

Με την χρήση της glStencilFunc δηλώνουμε πως το test δεν θα αποτυγχάνει ποτέ και ορίζουμε ως ref value την τιμή 3.

```
glStencilFunc(GL_ALWAYS, 3, 0xffffffff);
```

Θέτουμε στο όρισμα dppass την GL_REPLACE για να αντικαταστήσουμε την τιμή του stencil buffer με την τιμή 3 όταν το depth και stencil test περάσουν.

```
glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE);
```

Σχεδιάζουμε το έδαφος έχοντας από το βήμα 2 stencil value την τιμή 3.

```
Grass();
```

Ξανακαλούμε την glStencilFunc δηλώνοντας πως το test θα είναι επιτυχής μόνο για τιμές μικρότερες του 2.

```
glStencilFunc(GL_LESS, 2, 0xffffffff);
```

Θέτουμε στα ορίσματα της glStencilOp την GL_REPLACE για να αντικαταστήσουμε την τιμή του stencil buffer με την τιμή 2 όταν το depth και stencil test περάσουν.

```
glStencilOp(GL_REPLACE, GL_REPLACE, GL_REPLACE);
```

Ορίζουμε την polygon offset, επιλέγουμε την κατάλληλη blending συνάρτηση για την μίξη χρωμάτων των pixel σκιάς-εδάφους, απενεργοποιούμε το φωτισμό και με την glColor4f ορίζουμε ως ποσοστό μίξης το 50% δίνοντας στην alpha συνιστώσα της glColor4f την τιμή 0.5.

```
glEnable(GL_POLYGON_OFFSET_FILL);
glPolygonOffset(-1, -30);
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glDisable(GL_LIGHTING);
glColor4f(0.0, 0.0, 0.0, 0.5);
```

Χρησιμοποιούμε την glMultMatrixf για τον μετασχηματισμό της γεωμετρίας του αντικειμένου στο επίπεδο του εδάφους και καλούμε το αντικείμενο έχοντας ήδη από το βήμα 5 για stencil value την τιμή 2.

```
glPushMatrix();
glMultMatrixf((GLfloat *) grassShadow);
Build();
glPopMatrix();
```

Απενεργοποιούμε την μείξη χρωμάτων, την polygon offset και το stecil test, και ενεργοποιούμε τω φωτισμό.

```
glDisable(GL_BLEND);
glEnable(GL_LIGHTING);
glDisable(GL_POLYGON_OFFSET_FILL);
glDisable(GL_STENCIL_TEST);
glPopMatrix();
```

Έπειτα αφού απενεργοποιήθηκε ο stencil buffer ξανασχεδιάζουμε το έδαφος και το αντικείμενο.

```
Grass();
Build();
```

Εικόνα 9.5: Shadows with double blending

Εικόνα 9.6: Shadows with double blending eliminated

9.3.1. Κατασκευάζοντάς ένα μητρώο σκιών (shadowMatrix)

Η ακόλουθη ρουτίνα shadowMatrix αποθηκεύει τις συντεταγμένες των κορυφών των αντικειμένων που βρίσκονται μεταξύ της πηγής φωτός (light position) και του εδάφους (ground plane) σε ένα 4x4 μητρώο, κατάλληλο για επεξεργασία από την glMultMatrixf η οποία μετασχηματίζει και τελικά προβάλει τις συντεταγμένες των αντικειμένων στο έδαφος (ground plane).

Έχουμε ως ground plane equation το κάθετο κανονικό διάνυσμα του εδάφους και η πηγή φωτός να είναι ένα τυχαίο σημείο στο χώρο.

```
void shadowMatrix(GLfloat shadowMat[4][4], groundplane[4], GLfloat lightpos[4])
{
 GLfloat dot;

  dot = groundplane[X] * lightpos[X] + groundplane[Y] * lightpos[Y] +
        groundplane[Z] * lightpos[Z] + groundplane[W] * lightpos[W];

    shadowMat[0][0] = dot - lightpos[X] * groundplane[X];
    shadowMat[1][0] = 0.f - lightpos[X] * groundplane[Y];
    shadowMat[2][0] = 0.f - lightpos[X] * groundplane[Z];
    shadowMat[3][0] = 0.f - lightpos[X] * groundplane[W];

    shadowMat[X][1] = 0.f - lightpos[Y] * groundplane[X];
    shadowMat[1][1] = dot - lightpos[Y] * groundplane[Y];
    shadowMat[2][1] = 0.f - lightpos[Y] * groundplane[Z];
    shadowMat[3][1] = 0.f - lightpos[Y] * groundplane[W];

    shadowMat[X][2] = 0.f - lightpos[Z] * groundplane[X];
    shadowMat[1][2] = 0.f - lightpos[Z] * groundplane[Y];
    shadowMat[2][2] = dot - lightpos[Z] * groundplane[Z];
    shadowMat[3][2] = 0.f - lightpos[Z] * groundplane[W];

    shadowMat[X][3] = 0.f - lightpos[W] * groundplane[X];
    shadowMat[1][3] = 0.f - lightpos[W] * groundplane[Y];
    shadowMat[2][3] = 0.f - lightpos[W] * groundplane[Z];
    shadowMat[3][3] = dot - lightpos[W] * groundplane[W];
}
```

9.3.2. Κατασκευάζοντάς μια εξίσωση εδάφους για μη επίπεδες επιφάνειες.

Μέχρι τώρα θεωρούσαμε ως ground plane equation το κάθετο κανονικό διάνυσμα του εδάφους με σκοπό την απόδοση επίπεδων σκιών σε επίπεδο έδαφος. Σε μια ρεαλιστική σκηνή όμως το έδαφος δεν θα είναι πάντα επίπεδο και επομένως αυτό που χρειαζόμαστε είναι μια διαφορετική προσέγγιση για την απόδοση σκιών σε μη επίπεδες επιφάνειες. Η παρακάτω ρουτίνα findPlane παίρνει τις συντεταγμένες των γωνιών του εδάφους (Vertices) και ουσιαστικά υπολογίζει την κλίση και την μορφολογία τους έχοντας ως στόχο την απόδοση σκιών οι οποίες θα εφάπτονται ομαλά γύρο από οποιαδήποτε επιφάνεια όσο πολύπλοκη και αν είναι.

```
void findPlane(GLfloat plane[4],
               GLfloat v0[3], GLfloat v1[3], GLfloat v2[3])
{
   GLfloat vec0[3], vec1[3];

   vec0[X] = v1[X] - v0[X];
   vec0[Y] = v1[Y] - v0[Y];
   vec0[Z] = v1[Z] - v0[Z];

   vec1[X] = v2[X] - v0[X];
   vec1[Y] = v2[Y] - v0[Y];
   vec1[Z] = v2[Z] - v0[Z];

   /* find cross product to get A, B, and C of plane equation */
   plane[A] = vec0[Y] * vec1[Z] - vec0[Z] * vec1[Y];
   plane[B] = -(vec0[X] * vec1[Z] - vec0[Z] * vec1[X]);
   plane[C] = vec0[X] * vec1[Y] - vec0[Y] * vec1[X];

   plane[D] = -(plane[A] * v0[X] + plane[B] * v0[Y] + plane[C] * v0[Z]);
}
```

Έχουμε το παρακάτω παράδειγμα υλοποίησης των shadowMatrix και findPlane.

Δήλωση κορυφών επιφάνειας εδάφους

```
static GLfloat grassVertices[4][3] =
{
   { -20, 23.955, 0 },
   { 300, 23.955, 0 },
   { 300,    300, 0 },
   { -20,    300, 0 },
};
```

Κατασκευή επιφάνειας εδάφους

```
glBegin(GL_QUADS);
glNormal3f( -1,  -1, 3); glTexCoord2f( 0,  0); glVertex3fv(grassVertices[0]);
glNormal3f(0.5,  -1, 3); glTexCoord2f(45,  0); glVertex3fv(grassVertices[1]);
glNormal3f(0.5, 0.5, 3); glTexCoord2f(45, 45); glVertex3fv(grassVertices[2]);
glNormal3f( -1, 0.5, 3); glTexCoord2f( 0, 45); glVertex3fv(grassVertices[3]);
glEnd();
```

Κάλεσμα της ρουτίνας findplane με ορίσματα τις κορυφές του εδάφους και αποθήκευση τους στο 1^{ο} όρισμα της finPlane το grassPlane.

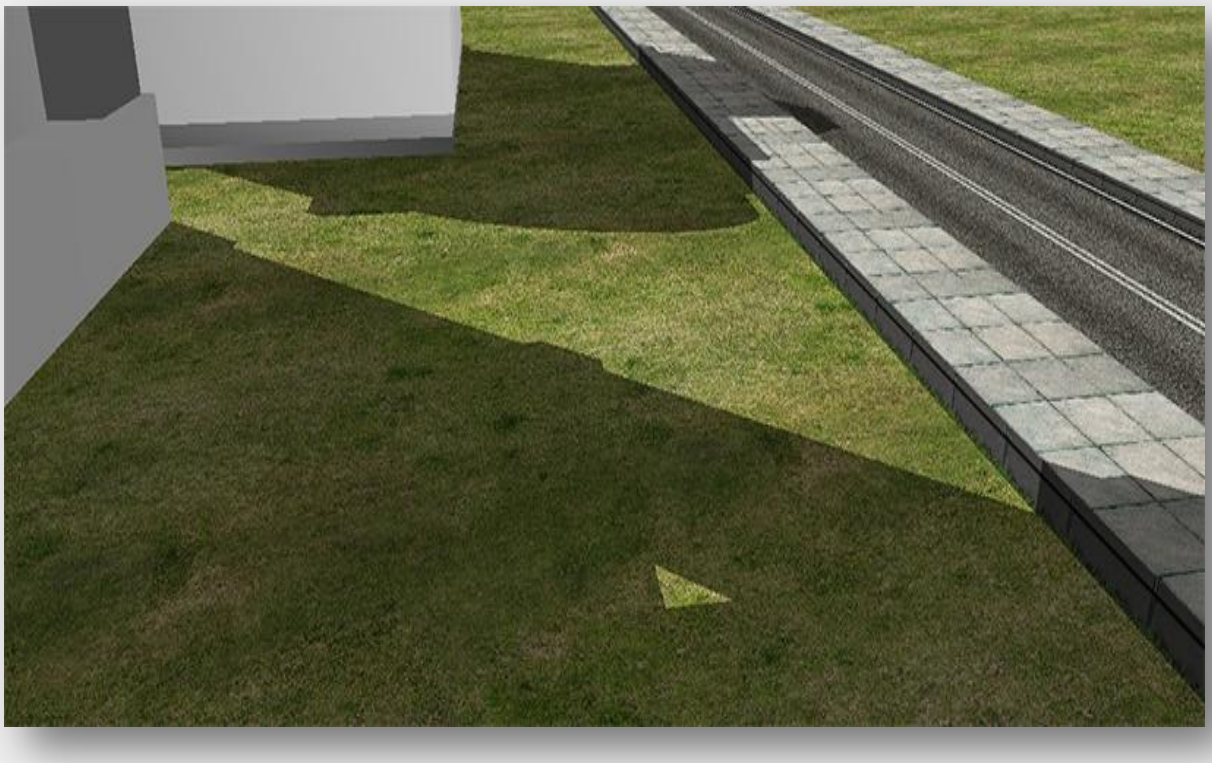
```
findPlane(grassPlane, grassVertices[1], grassVertices[2], grassVertices[3]);
```

Κάλεσμα της ρουτίνας shadowMatrix με ορίσματα το grassPlane της findPlane και το lp που είναι η θέση της πηγής του φωτός. Το αποτέλεσμα της ρουτίνας αποθηκεύεται στο 1^{ο} όρισμα της shadowMatrix το grassShadow το οποίο θα χρησιμοποιηθεί αργότερα από την glMultMatrixf.

```
shadowMatrix(grassShadow, grassPlane, lp);
```

Ενεργοποίηση του stencil buffer και απόδοση των σκιών των αντικειμένων.

Εικόνα 9.7: Απόδοση σκιάς σε μη επίπεδες επιφάνειες



CHAPTER X

ΛΙΣΤΕΣ ΑΠΕΙΚΟΝΙΣΗΣ

Οι λίστες απεικόνισης (display lists) είναι μια ομάδα από εντολές τις OpenGl, τις οποίες μετρά από τον ορισμό τους τις αποθηκεύουμε με σκοπό να τις χρησιμοποιήσουμε αργότερα. Όταν καλούμε μια display list, οι εντολές που είναι αποθηκευμένες μέσα σε αυτήν εκτελούνται με την σειρά με την οποία δηλωθήκαν. Οι περισσότερες εντολές της OpenGl μπορούν είτε να αποθηκευτούν σε μια display list είτε να δηλωθούν σε άμεσο τρόπο, όπως τα παραδείγματα κώδικα που έχουμε δει μέχρι τώρα, με τον οποίο οι εντολές εκτελούνται αμέσως.

10.1. Η χρήση μιας display list

Οι display lists από την στιγμή που μας δίνουν την δυνατότητα να αποθηκεύσουμε εντολές μέσα τους και να τις καλέσουμε αργότερα, βελτιώνουν κατά πολύ την απόδοση. Είναι συχνά μια καλή ιδέα να 'κρύψουμε' εντολές μέσα σε μια display list αν σχεδιάζουμε να σχεδιάσουμε την ίδια γεωμετρία πολλές φορές.

Για να δούμε πώς μπορούμε να χρησιμοποιήσουμε τις display lists για την αποθήκευση της γεωμετρίας ενός αντικειμένου μόνο μία φορά, θεωρήστε την κατασκευή ενός τρίκυκλου. Οι δύο τροχοί στο πίσω μέρος είναι το ίδιο μέγεθος, αλλά είναι μετατοπισμένες η μία από την άλλη. Ο μπροστινός τροχός είναι μεγαλύτερος από τους πίσω τροχούς καθώς επίσης και σε μια διαφορετική θέση. Ένας αποτελεσματικός τρόπος για να καταστήσει τους τροχούς στο τρίκυκλο θα είναι να αποθηκεύσετε τη γεωμετρία για έναν τροχό σε μια display list, και στη συνέχεια, εκτελέσετε την λίστα τρεις φορές. Έπειτα με τους κατάλληλους μετασχηματισμούς τοποθετήστε τις ρόδες στη θέση τους και στο κατάλληλο μεγεθος.Οι μετασχηματισμοί θα γίνουν πριν το κάλεσμα της λίστας.

10.2. Δημιουργία και εκτέλεση μιας display list

Η διαδικασία υλοποίησης μιας λίστας απεικόνισης αποτελείται από τον καθορισμό των λιστών προς δημιουργία, αρχικοποίηση, κατασκευή και τερματισμός της συγκεκριμένης λίστας και κάλεσμα της.

ΔΗΜΙΟΥΡΓΙΑ ΕΝΟΣ ΣΥΝΕΧΟΜΕΝΟΥ ΣΥΝΟΛΟΥ ΚΕΝΩΝ DISPLAY LIST.

```
GLuint  list = glGenLists(GLsizei  range);
```

- ListName είναι το όνομα της λίστας και range ο αριθμός των λιστών προς δημιουργία.

ΔΗΜΙΟΥΡΓΙΑ Η ΑΝΤΙΚΑΤΑΣΤΑΣΗ ΜΙΑΣ DISPLAY LIST.

```
void glNewList(GLuint  list,  GLenum  mode);
```

- Το list καθορίζει το όνομα της display list και mode τον τρόπο μεταγλώττισης ο οποίος μπορεί να είναι GL_COMPILE ή GL_COMPILE_AND_EXECUTE.

- GL_COMPILE : οι εντολές απλά συντάσονται και εκτελούνται αργότερα.
- GL_COMPILE_AND_EXECUTE : οι εντολές εκτελούνται κατά την σύνταξη τους στη λίστα.

ΤΕΛΟΣ ΤΗΣ ΛΙΣΤΑΣ

```
glEndList();
```

ΚΑΛΕΣΜΑ ΤΗΣ ΛΙΣΤΑΣ

```
void glCallList(GLuint  list);
```

- Το list καθορίζει το όνομα της λίστας προς εκτέλεση.

Παρακάτω στο παράδειγμα 1 αποθηκεύουμε σε μια λίστα την ρουτίνα απόδοσης ενός κύβου και την καλούμε δυο φορές αποδίδοντας διαφορετικό χρώμα, μέγεθος και θέση κάθε φορά.

ΠΑΡΑΔΕΙΓΜΑ 1

```
GLuint Square;

GLvoid Lists()
{
        Square=glGenLists(1);                          creates room for 1 list
        glNewList(Square,GL_COMPILE_AND_EXECUTE);      create the list and set the mode
        glutSolidCube(4);                              solid cube routine of the glut library
        glEndList();                                   end of the list
}

int InitGL(GLvoid)
{
        ...
        Lists();                                       build the display list
        ...
}

void display()
{
        glTranslatef(0.0, 5.0, 0.0);          set the position
        glScalef(1.0, 1.0, 1.0);              set the scaling
        glColor3f(1.0, 0.0, 0.0);             set the color
        glCallList(Square);                   call list square after transformations

        glTranslatef(3.0, 7.0, 0.0);          set the position
        glScalef(2.0, 2.0, 2.0);              set the scaling
        glColor3f(0.0, 0.5, 0.5);             set the color
        glCallList(Square);                    call again list square after transformations
}
```

ΠΑΡΑΔΕΙΓΜΑ 2

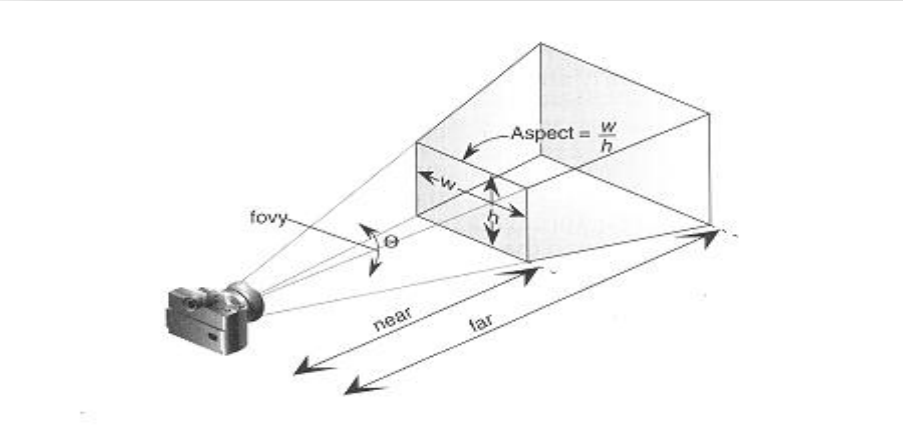
```
GLuint shapes;
GLuint sphere;
GLuint cube;
GLuint teapot;

GLvoid Lists()
{
        shapes=glGenLists(3);          creates room for 3 lists

        sphere = shapes;
        glNewList(sphere ,GL_COMPILE_AND_EXECUTE);
        glutSolidSphere(4, 20, 20);
        glEndList();

        cube = shapes +1;
        glNewList(cube ,GL_COMPILE_AND_EXECUTE);
        glutSoliCube(4);
        glEndList();

        teapot = shapes +2;
        glNewList(teapot ,GL_COMPILE_AND_EXECUTE);
        glutSolidTeapot(6);
        glEndList();
}

int InitGL(GLvoid)
{
        ...
        Lists();                                       build the display list
        ...
}

void display()
{
        glCallList(sphere);
        glCallList(cube);
        glCallList(teapot);
}
```

CHAPTER XI

ΕΛΕΥΘΕΡΗ ΠΕΡΙΗΓΗΣΗ ΣΤΟΝ 3Δ ΧΩΡΟ

Η ελεύθερη περιήγηση υπάρχει έτσι ώστε να έχουμε την δυνατότητα της εξερεύνησης του 3δ χώρου που κατασκευάζουμε. Για την επίτευξη αυτού, πρέπει, να τροποποιήσουμε την αρχική σταθερή κάμερα που δημιουργούμε σε ένα σταθερό σημείο στον χώρο ώστε να έχει την δυνατότητα μετακίνησης της θέσης της από το αρχικό σταθερό σημείο προς όλα τα σημεία και όλες τις κατευθύνσεις στον χώρο.

Υπάρχουν περισσότεροι από έναν τρόποι επίτευξης της ελεύθερης περιήγησης, αλλά οποιονδήποτε και να επιλέξουμε οπωσδήποτε θα υπάρχει το κάλεσμα της gluPerspective. Αυτή η εντολή ορίζει ένα οπτικό πεδίο σε σχήμα πυραμίδας με τετράγωνη βάση, έχοντας την βάση στο μακρύτερο οπτικό πεδίο του θεατή και ως κορυφή το κοντινότερο, και μόνο τα αντικείμενα που βρίσκονται μέσα στην πυραμίδα θα είναι θεατά.



Εικόνα 11.1: Camera pyramid

Η κάμερα όπως φαίνεται στην εικόνα 11.1 καθορίζεται από την γωνιά fov(y) του οπτικό πεδίου της (filed of view), από τον λόγο του μήκους (w) προς το ύψος (h) του κοντινότερου οπτικού πεδίου (aspect ratio) και από τις τιμές του κοντινότερου (znear) και μακρύτερου (zfar) οπτικού πεδίου.
Έτσι η 'όραση' του θεατή περνά μέσα από την κορυφή της πυραμίδας και εκτείνεται προς την βάση της.

Οι καλύτερες και πιο ρεαλιστικές τιμές για τον ορισμό του οπτικού πεδίου της κάμερας είναι αυτές με γωνιά οπτικού πεδίου (fovy) 45^0, λόγο μήκους-ύψους (w/h) οι διαστάσεις σε pixels της οθόνης που προβάλλεται η σκηνή και το κοντινότερο (znear) και μακρύτερο (zfar) οπτικό πεδίο ποικίλει ανάλογα την κρίση του προγραμματιστή.

```
gluPerspective(45.0, 1366/768, 5.0, 130000.0);
```

Με την gluPerspective υλοποιήσαμε μια σταθερή σε ένα σημείο κάμερα, κοιτώντας πάντα προς μια συγκεκριμένη κατεύθυνση αλλά χωρίς την δυνατότητα κίνησης. Η κάμερα θα πρέπει να κινείται εμπρός ή πίσω, να κοιτάει προς τα πάνω ή κάτω με ένα φυσικό και ρεαλιστικό τρόπο σαν η κάμερα ουσιαστικά να είναι ένας άνθρωπος που περπατάει στον δρόμο και κοιτάει τριγύρω (θέα πρώτου προσώπου). Όμως, η σταθερή κάμερα που υλοποιήσαμε με την gluPerspective δεν είναι δυνατό να τροποποιηθεί. Πάντα θα είναι στο ίδιο μέγεθος και στην ιδιά θέση. Έτσι αντί για την μετακίνηση της κάμερας, θα πρέπει να μετακινήσουμε ολόκληρη την σκηνή. Για παράδειγμα αν θέλουμε να περιστρέψουμε την κάμερα προς τα δεξιά, θα πρέπει στην ουσία να περιστρέψουμε την σκηνή προς τα αριστερά κοκ.

Όμως πως μπορούμε να μετασχηματίσουμε ολόκληρη την σκηνή; Χρησιμοποιώντας κλασεις (Class) και διανύσματα (vectors).

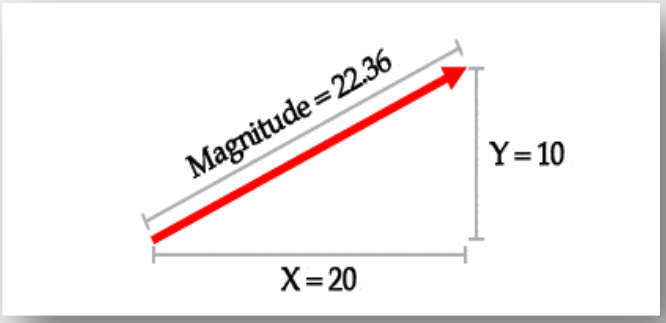
11.1. Διανύσματα (Vectors)

Μια αξιοπρεπής κατανόηση για το τι είναι διάνυσμα είναι θεμελιώδους σημασίας στον κόσμο του 3Δ προγραμματισμού. Όταν φτάσουμε στον κώδικα αργότερα, θα χρησιμοποιήσουμε διανύσματα για να μετακινήσουμε την κάμερα σε διάφορες κατευθύνσεις χρησιμοποιώντας το πληκτρολόγιο και το ποντίκι.

Σε τρισδιάστατες και δυσδιάστατες σκηνές, τα διανύσματα χρησιμοποιούνται για να αναπαραστήσουν έννοιες όπως:

- **Position** - Θέση της κάμερας σε συντεταγμένες.
- **Displacement** - Μετακίνηση της κάμερας.
- **Direction** - Κατεύθυνση κατά την οποία μετακινείται η κάμερα.
- **Velocity** - Ταχύτητα μετακίνησης της κάμερας.

Χρησιμοποιώντας λοιπόν έναν ψευδο-μαθηματικό ορισμό, διάνυσμα είναι μια κατεύθυνση (direction) με έκταση (magnitute) *(σχ.11.1)*. Ένα διάνυσμα μπορεί να έχει οποιαδήποτε κατεύθυνση. Πάνω, κάτω, δεξιά, αριστερά, βόρια, ανατολικά κοκ. Οποιαδήποτε κατεύθυνση που μπορείς να δείξεις με το δάχτυλο είναι μια έγκυρη κατεύθυνση για ένα 3Δ διάνυσμα. Το άλλο μέρος ενός διανύσματος, η έκταση, είναι το μήκος ή αλλιώς το μέγεθος του διανύσματος. Ο καλύτερος τρόπος κατανόησης ενός διανύσματος είναι η σχεδίαση του. Τα διανύσματα συνήθως σχεδιάζονται ως βέλη. Η μύτη του βέλους δείχνει την κατεύθυνση και το μήκος του την έκταση.



Σχήμα 11.1: Εικονική απεικόνιση ενός διανύσματος

Όσον αφορά τον προγραμματισμό, ένα διάνυσμα είναι απλά μια διάταξη από αριθμούς. Κάθε αριθμός είναι η έκταση του διανύσματος. Για παράδειγμα, ένα τρισδιάστατο διάνυσμα είναι μια διάταξη από τρεις αριθμούς ενώ ένα δυσδιάστατο διάνυσμα είναι μια διάταξη από δυο αριθμούς.

Είναι εύκολο να δούμε πως ένα 3δ διάνυσμα αναπαριστά την θέση του σε ένα σύστημα συντεταγμένων. Οι τρεις διαστάσεις ενός 3δ διανύσματος είναι οι x, y και z τιμές και η κατεύθυνση και έκταση του μετριούνται από την αρχή των αξόνων. Για παράδειγμα, αν ένα αντικείμενο βρίσκεται σε μια τυχαία θέση στον 3Δ χώρο (0, 2, 0), τότε εκτείνεται δυο μονάδες κατά τον θετικό άξονα των y.

```
tVector3(float x, float y, float z);
```

Όταν αντιστρέφουμε ένα διάνυσμα – που σημαίνει, όταν κάνουμε ένα υπάρχον διάνυσμα αρνητικό, η έκταση του παραμένει η ίδια αλλά η κατεύθυνση του γίνετε η αντίθετη από αυτήν που ήταν.

```
tVector3(float -x, float -y, float -z);
```

Ένας χειριστής (operator) χρησιμοποιείται για την κλιμάκωση των διανυσμάτων βάση ενός αριθμού. Ο αριθμός ονομάζεται κλιμακωτής (scalar) και γιαυτο τον λόγο η διαδικασία ονομάζεται χειριστές κλιμάκωσης (scalar operators). Όταν κλιμακώνουμε ένα διάνυσμα βάση ενός αριθμού και ανάλογα το είδος του operator (+, -, *, /), το αποτέλεσμα είναι ένα νέο διάνυσμα με την ίδια κατεύθυνση, αλλά με κλιμακούμενη έκταση, κάτι που θα μας χρησιμεύσει αργότερα στην μετακίνηση της κάμερας.

```
tVector3(float x*number, float y*number, float z*number);
tVector3(float x/number, float y/number, float z/number);
tVector3(float x-number, float y-number, float z-number);
tVector3(float x+number, float y+number, float z+number);
```

Για την ευκολότερη πρόσβαση από εμάς στους παραπάνω χειριστές κλιμάκωσης, τους ομαδοποιούμε σε ένα struct. Τι είναι ένα struct και πως κατασκευάζεται θα το δούμε στην επόμενη ενότητα.

11.2. Κατασκευή ενός vector struct

Ένα struct στη γλώσσα προγραμματισμού C και C++ είναι μια σύνθετη δήλωση τύπων δεδομένων που ορίζουν ένα φυσικά ομαδοποιημένο κατάλογο μεταβλητών που πρέπει να τοποθετηθούν συγκεντρωμένες κάτω από ένα όνομα σε ένα μπλοκ της μνήμης, επιτρέποντας τες, να είναι εύκολα προσβάσιμες μέσω ενός και μόνο δείκτη.
Έτσι ένα struct διανυσμάτων μας επιτρέπει την ομαδοποιημένη δήλωση των χειριστών κλιμάκωσης διανυσμάτων για την εύκολη πρόσβαση τους κατά την υλοποίηση της κάμερας.

```
typedef struct tVector3
{
 tVector3() {}                                                              constructor

 tVector3 (float new_x, float new_y, float new_z)                    initialize constructor

 {x = new_x; y = new_y; z = new_z;}

 tVector3 operator+(tVector3 vVector)
 {
        return tVector3(vVector.x+x, vVector.y+y, vVector.z+z);
 }

 tVector3 operator-(tVector3 vVector)
 {
        return tVector3(x-vVector.x, y-vVector.y, z-vVector.z);
 }

 tVector3 operator*(float number)
 {
        return tVector3(x*number, y*number, z*number);
 }

 tVector3 operator/(float number)
 {
        return tVector3(x/number, y/number, z/number);
 }

 float x, y, z;                                                          3D vector coordinates

}tVector3;
```

11.3. Ορισμός ιδιοτήτων της κάμερας με διανύσματα και κατασκεύη της κλάσης Ccamera.

Όπως είδαμε σε παραπάνω ενότητα σε τρισδιάστατες και δυσδιάστατες σκηνές, τα διανύσματα χρησιμοποιούνται για να αναπαραστήσουν ιδιότητες όπως:

- Θέση της κάμερας
- Μετακίνηση της κάμερας
- Περιστροφή της κάμερας
- Περιστροφή της κάμερας από το ποντίκι

Έχοντας κατασκευάσει το struct διανυσμάτων αυτό που μας μένει είναι να κατασκευάσουμε μια κλάση η οποία θα ομαδοποιεί τις παραπάνω ιδιότητες της κάμερας.

```
 class CCamera
 {
       public:

       tVector3 mPos;
       tVector3 mView;
       tVector3 mUp;

       void Mouse_Move(int wndWidth, int wndHeight);     function to rotate the camera with the mouse
       void Move_Camera(float speed);                  function to move the camera given the speed
       void Rotate_View(float speed);                function to rotate the camera given the speed
       void Position_Camera(        function to set the camera initial position and view direction

                    float pos_x, float pos_y,float pos_z,            position in x, y, z coordinates
                    float view_x, float view_y, float view_z,    view direction in x, y, z coordinates
                     float up_x,   float up_y,   float up_z );                        the up vector
 };
```

Συνάρτηση αρχικής θέσης της κάμερας

```
void CCamera::Position_Camera(float pos_x,  float pos_y,  float pos_z,
                              float view_x, float view_y, float view_z,
                              float up_x,   float up_y,   float up_z)
{
       mPos    = tVector3(pos_x,  pos_y,  pos_z );              set position
       mView   = tVector3(view_x, view_y, view_z);      set view
       mUp     = tVector3(up_x,   up_y,   up_z  );             set the up vector
}
```

Τα ορίσματα της παραπάνω συνάρτησής καθορίζονται από τον χρήστη και τοποθετούν την κάμερα σε ένα αρχικό σημείο στον χώρο. Το mPos καθορίζει την θέση της κάμερας στις x, y, z συντεταγμένες, το mView καθορίζει προς τα πού θα κοιτάει η κάμερα και το mUp καθορίζει τον κατακόρυφο άξονα συντεταγμένων.

Συνάρτηση μετακίνησης της κάμερας

```
void CCamera::Move_Camera(float speed)
{
       tVector3 vVector = mView - mPos;

       mPos.x  = mPos.x  + vVector.x * speed;
       mPos.z  = mPos.z  + vVector.z * speed;
       mView.x = mView.x + vVector.x * speed;
       mView.z = mView.z + vVector.z * speed;
}
```

Η συνάρτηση μετακίνησης της κάμερας από ένα αρχικό σημείο προς μια τυχαία κατεύθυνση θέτει σε χρήση τις κλιμακώσεις διανυσμάτων όπως έχουν ορισθεί στο struct **`tVector3.`** Όπως έχουμε δει σε προηγούμενη ενότητα ένα διάνυσμα είναι μια κατεύθυνση με έκταση, η οποία έκταση (ή μήκος) βάσει ενός χειριστή κλιμάκωσης μπορεί να μεγαλώνει ή να μικραίνει.
Οπότε με την **`tVector3 vVector = mView – mPos`** δηλώνουμε ένα διάνυσμα με κατεύθυνση την **mView**, με έκταση η οποία υπολογίζεται μετα από κλιμάκωση της με την σταθερά speed **`(vVector.x * speed`**) και με νέα θέση την **`mView – mPos.`**
Το νέο διάνυσμα καλείται από την keyboard function το οποίο πατώντας ένα κουμπί από το πληκτρολόγιο πολλαπλασιάζεται με μια μεταβλητή και εκτείνετε σταδιακά μέχρι την νέα θέση.

Συνάρτηση περιστροφής της κάμερας με το ποντίκι

```
void CCamera::Mouse_Move(int wndWidth, int wndHeight)
{
      POINT mousePos;
      int mid_x = wndWidth  >> 1;
      int mid_y = wndHeight >> 1;
      float angle_y  = 0.0f;
      float angle_z  = 0.0f;

      GetCursorPos(&mousePos); // Get the 2D mouse cursor (x,y) position

      if( (mousePos.x == mid_x) && (mousePos.y == mid_y) ) return;

      SetCursorPos(mid_x, mid_y);      // Set the mouse cursor in the center of the window

      // Get the direction from the mouse cursor, set a resonable maneuvering speed
      angle_y = (float)( (mid_x - mousePos.x) ) / 1000;
      angle_z = (float)( (mid_y - mousePos.y) ) / 1000;

      // The higher the value is the faster the camera looks around.
      mView.y += angle_z * 2;

      // limit the rotation around the x-axis
      if((mView.y - mPos.y) > 8)  mView.y = mPos.y + 8;
      if((mView.y - mPos.y) <-8)  mView.y = mPos.y - 8;

      Rotate_View(-angle_y);
}
```

Συνάρτηση περιστροφής της κάμερας με το πληκτρολόγιο

```
void CCamera::Rotate_View(float speed)
{
	tVector3 vVector = mView - mPos; // Get the view vector

	mView.z = (float)(mPos.z + sin(speed)*vVector.x + cos(speed)*vVector.z);
	mView.x = (float)(mPos.x + cos(speed)*vVector.x - sin(speed)*vVector.z);
}
```

11.4. Αρχικοποίηση της Κάμερας και αλληλεπίδραση από τον χρήστη.

Ονομάζουμε την κλάση Ccamera σε objCamera.

```
CCamera objCamera; Rename the CCamera class to objCamera
```

Δηλώνουμε την αρχική θέση και οπτική κατεύθυνση της κάμερας.

```
Given that we are in x,y,z coordinate system with y set as the height, we set the initial camera position
in 0,6,5 degrees and looking at y direction at 5.5 degrees.

objCamera.Position_Camera(0, 6, 5,   0, 5.5f, 0,    0, 1, 0);
```

Συνδέουμε τις ιδιότητες της κάμερας με το πληκτρολόγιο.

```
#define CAMERASPEED     0.006f                  The Camera Speed

void Keyboard_Input()
{

     if (GetKeyState('3') & 0x80)
            {
              Move straight given the cameraspeed multiplying it by 155 (fast)
                 objCamera.Move_Camera( CAMERASPEED * 155 );
            }
     if (GetKeyState('W') & 0x80)
                {
              Move straight given the cameraspeed multiplying it by 25 (slow)
                 objCamera.Move_Camera( CAMERASPEED  * 25 );

              walkbiasangle += 11;
                 walkbias = (float)cos(walkbiasangle * piover180)/15.0f; Walk effect

            }
     if (GetKeyState(VK_DOWN) & 0x80)
           {
              Move back given the negative cameraspeed multiplying it by 5 (very slow)
                 objCamera.Move_Camera(-CAMERASPEED * 5);
           }
     if (GetKeyState('S') & 0x80)
       {
                Move back given the negative cameraspeed multiplying it by 825 (very fast)
                 objCamera.Move_Camera(-CAMERASPEED  * 825 );
           }
     if((GetKeyState(VK_LEFT) & 0x80) || (GetKeyState('A') & 0x80))
           {
                Look left given the negative cameraspeed
                objCamera.Rotate_View(-CAMERASPEED);
           }

     if((GetKeyState(VK_RIGHT) & 0x80) || (GetKeyState('D') & 0x80))
           {
                Look right given the cameraspeed
                objCamera.Rotate_View( CAMERASPEED);
           }
}
```

Καλούμε στην Main() τις δυο ρουτίνες για αλληλεπίδραση από το πληκτρολόγιο και το ποντίκι.

```
Keyboard_Input();
objCamera.Mouse_Move(1366,768); Call function with arguments the maximum size of screen reosolution
```

CHAPTER XII

ΠΑΡΑΔΕΙΓΜΑΤΑ ΚΩΔΙΚΑ

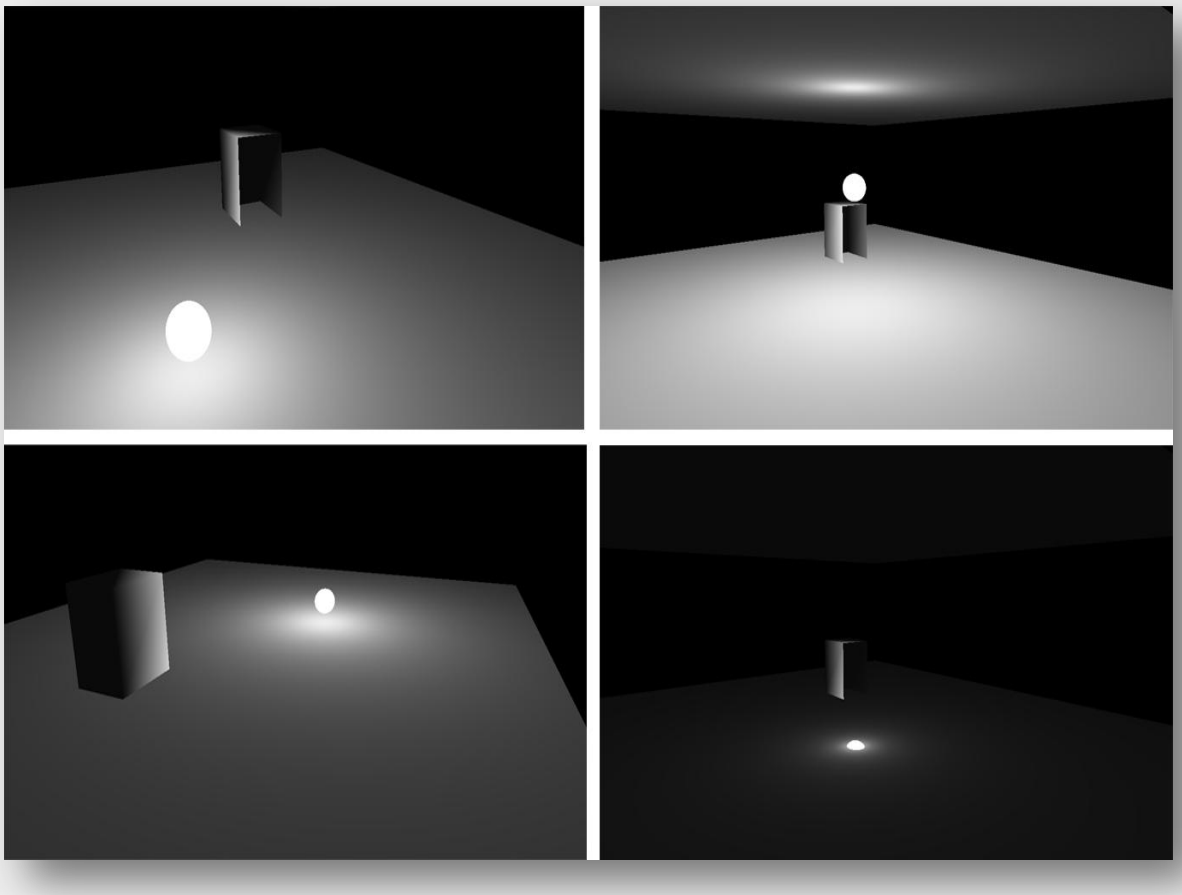
ΠΑΡΑΔΕΙΓΜΑ 1

Εχουμε ένα τετράπλευρο και μία σημειακή πηγή φωτός με δυνατότητα μετατόπισης της από τον χρήστη στον τρισδιάστατο χώρο.Στο παράδειγμα υπολογίζονται τα κανονικά διανύσματα βάση της σκίασης Gouraud τόσο του τετραπλεύρου οσο του ταβανιού και του πατώματος.

```
float light1Position[] = {0, 0 , 10, 1}; //1 positional, 0 directional

void LightSource() //draw the light source
{
        glPushMatrix();

        float NoEmission[4] = {0.0, 0.0, 0.0, 1.0};
        float Emission[4] = {1.0, 1.0, 1.0, 1.0};
        float Lt1ambient[4] = {0.0, 0.0, 0.0, 1.0};
```

```
        float Lt1diffuse[4] = {1.0, 1.0, 1.0, 1.0};
        float Lt1specular[4] = {1.0, 1.0, 1.0, 1.0};

        glLightfv(GL_LIGHT1, GL_AMBIENT, Lt1ambient);
        glLightfv(GL_LIGHT1, GL_DIFFUSE, Lt1diffuse);
        glLightfv(GL_LIGHT1, GL_SPECULAR,Lt1specular);

        glTranslatef(light1Position[0],light1Position[1],light1Position[2]); //translate the sphere
        glMaterialfv(GL_FRONT_AND_BACK, GL_EMISSION, Emission);
        glutSolidSphere(0.5, 60, 60);                                        //the light source
        glMaterialfv(GL_FRONT_AND_BACK, GL_EMISSION, NoEmission);
        glLightfv(GL_LIGHT1, GL_POSITION, light1Position );

        glPopMatrix();
}

static void Surface(int w, int h)  //draw the surface with triangles
{
  int i, j;
  float dw = 1.0 / w;
  float dh = 1.0 / h;

  glNormal3f(0.0, 0.0, 1.0);
  for (j = 0; j < 400; ++j)             // the next conditions and code creates a
  {                                     // surface of 160000 triangles.Replace
    glBegin(GL_TRIANGLE_STRIP);         // the GL_TRIANGLE_STRIP with
    for (i = 0; i <= 400; ++i)          // GL_LINE_STRIP to see the triangles.
        {
      glVertex2f(dw * i, dh * (j + 1));
      glVertex2f(dw * i, dh * j);
    }
    glEnd();
  }
}

void Object()                           //draw a simple quad object
{
        glBegin(GL_QUADS);

        glNormal3f(-1, -1,  1); glVertex3f(-1, -1,  3);
        glNormal3f( 1, -1,  1); glVertex3f( 1, -1,  3);
        glNormal3f( 1,  1,  1); glVertex3f( 1,  1,  3);
        glNormal3f(-1,  1,  1); glVertex3f(-1,  1,  3);

        glNormal3f(-1,  1,  1); glVertex3f(-1,  1,  0);
        glNormal3f(-1,  1,  1); glVertex3f(-1,  1,  3);
        glNormal3f( 1,  1,  1); glVertex3f( 1,  1,  3);
        glNormal3f( 1,  1,  1); glVertex3f( 1,  1,  0);

        glNormal3f( 1, -1,  1); glVertex3f( 1, -1,  0);
        glNormal3f( 1,  1,  1); glVertex3f( 1,  1,  0);
        glNormal3f( 1,  1,  1); glVertex3f( 1,  1,  3);
        glNormal3f( 1, -1,  1); glVertex3f( 1, -1,  3);

        glNormal3f(-1, -1,  1); glVertex3f(-1, -1,  0);
        glNormal3f(-1, -1,  1); glVertex3f(-1, -1,  3);
        glNormal3f(-1,  1,  1); glVertex3f(-1,  1,  3);
        glNormal3f(-1,  1,  1); glVertex3f(-1,  1,  0);

        glEnd();
}


void Keyboard_Input()                   // keyboard interaction of the light source
{
        if(GetKeyState('1') & 0x80)
        {
                light1Position[2] -= 0.3;
        }
        if(GetKeyState('2') & 0x80)
        {
                light1Position[2] += 0.3;
        }
        if(GetKeyState('4') & 0x80)
        {
                light1Position[0] -= 0.3;
        }
        if(GetKeyState('5') & 0x80)
        {
                light1Position[0] += 0.3;
```

```
        }
        if(GetKeyState('9') & 0x80)
        {
                light1Position[1] += 0.3;
        }
        if(GetKeyState('0') & 0x80)
        {
                light1Position[1] -= 0.3;
        }
}
```

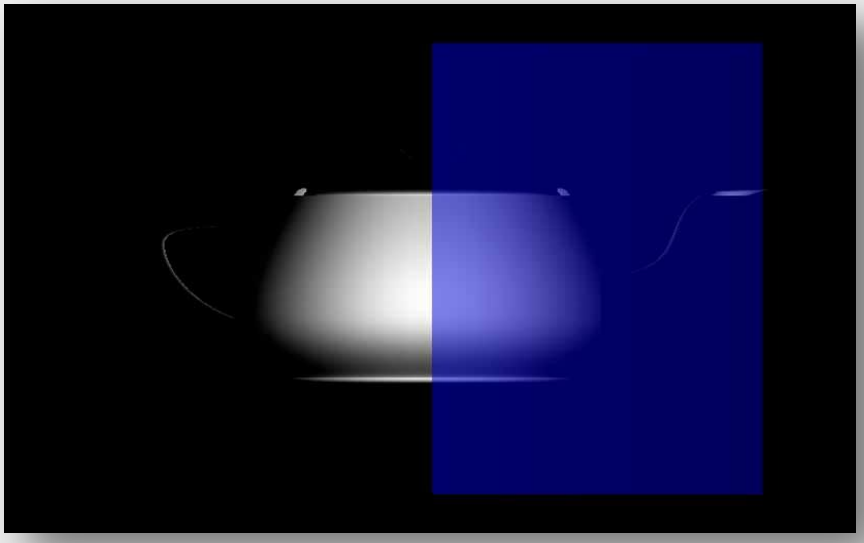


ΠΑΡΑΔΕΙΓΜΑ 2

Στο παράδειγμα 2 εφαρμόζουμε το εφέ διαφάνειας. Η σκηνή αποτελείται από μια τσαγιέρα και από μια διαφανής μπλε επιφάνεια.

```
#include <glut.h>

void init()
{
        glutInitWindowPosition(50,50);                  // the window position
        glutInitWindowSize(800,600);                    // the window size
        glutCreateWindow("Diffuse reflection");         // the name of the window
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glClearColor(0,0,0,0);                          // clear the screen with black color

        glMatrixMode(GL_PROJECTION);
```

```
        glOrtho(-80,80,-60,60,0,50);
        glEnable(GL_LIGHTING);                          // Enable lighting and
        glEnable(GL_LIGHT0);                            // the light source 0

        GLfloat globalAmbient[]={0,0,0};                        // set the light and material properties
        glLightModelfv(GL_LIGHT_MODEL_AMBIENT,globalAmbient);     // of the sceen
        GLfloat light0Position[]={0,0,40,1};
        glLightfv(GL_LIGHT0,GL_POSITION,light0Position);
        GLfloat light0Diffuse[]={1,1,1};
        glLightfv(GL_LIGHT0,GL_DIFFUSE,light0Diffuse);
        GLfloat diffuseMat[]={1.0, 1.0, 1.0};
        glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,diffuseMat);

        glMatrixMode(GL_MODELVIEW);
        glTranslatef(0,0,-40);                  // translate the scene away from the screen at 40 degrees
}

void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glutSolidTeapot(36);                                    // draw teapot first

        glPushMatrix();
        glEnable(GL_BLEND);                                     // enable blending
        glBlendFunc(GL_ONE_MINUS_SRC_ALPHA,GL_SRC_ALPHA);       // and set the blending functions
        GLfloat blue[]={0.0, 0.0, 1.0, 0.5};                    // set surface color to blue with
        glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,blue);        // alpha value (transparency) to half.
        glBegin(GL_QUADS);                                      // draw surface second
        glVertex2f(0,  -50);
        glVertex2f(60, -50);
        glVertex2f(60,  50);
        glVertex2f(0,   50);
        glEnd();
        glPopMatrix();
        glFlush();
}
```



ΠΑΡΑΔΕΙΓΜΑ 3

Το παραδειγμα 3 είναι μια απλη υλοποιηση αποκοπης pixel του stencil buffer. Στην σκηνη εχουμε ένα απλο κυβο και μια 2Δ επιφανεια. Η διαδικασια αποκοπης αναλυεεται στα παρακατω βηματα.

Βημα 1. Ενεργοποιουμε το stencil test

```
glPushMatrix();
glColorMask(GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE);  // clear color mask and disable writing to color buffer
glDepthMask(GL_FALSE);          // clear depth mask and disable writing to depth buffer
glEnable(GL_STENCIL_TEST);              // enable stencilling and write only to stencil buffer
```

Βημα 2. Σχεδιαζουμε, στην προκειμενη περιπτωση, ένα απλο δυσδιαστατο τετραγωνο και το αποθηκευουμε στον stencil buffer με stencil value 1 .

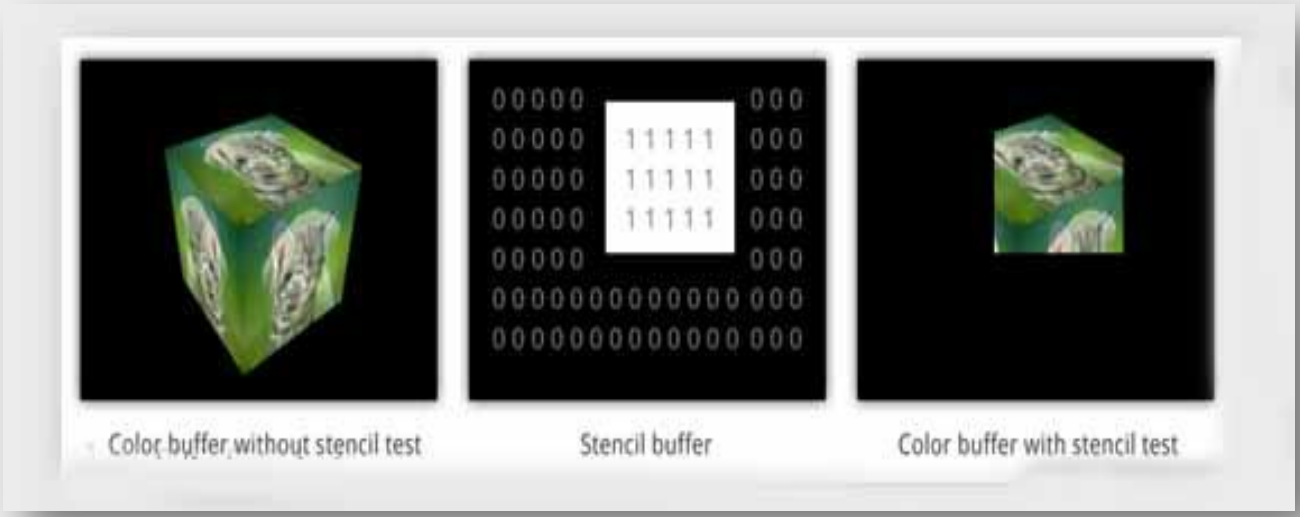
```
glStencilFunc(GL_ALWAYS, 1, 0xFFFFFFFF);                // set reference stencil value 1.
glStencilOp(GL_REPLACE, GL_REPLACE, GL_REPLACE);
glBegin(GL_QUADS);                                      // draw the 2d surface
glVertex3f( 10, 10,  0);
glVertex3f( 20, 10,  0);
glVertex3f( 20, 10, 10);
glVertex3f( 10, 10, 10);
glEnd();
```

Βημα 3. Λεμε στο stencil test να αποριψει οσα pixels εχουν stencil value ιση του 1 και σχεδιαζουμε τον κυβο.

```
glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);        // enable writing to color buffer
glDepthMask(GL_TRUE);                                   // enable writing to depth buffer
glStencilFunc(GL_EQUAL, 1, 0xFFFFFFFF);
glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);
glDisable(GL_LIGHTING);
glTranslatef(15, 25, 6);
glutSolidCube(20);
```

Βημα 4. Απενεργοποιουμε το stencil test.

```
glEnable(GL_LIGHTING);
glDisable(GL_STENCIL_TEST);
glPopMatrix();
```



ΠΑΡΑΔΕΙΓΜΑ 4

Διάβασμα δεδομένων και απόδοση επιφάνειας από αρχείο κειμένου.

Βήμα 1: Αποθήκευση δεδομένων επιφάνειας σε αρχείο .txt

Στην opengl είναι δυνατή η κατασκευή επιφανειών φορτώνοντας τα δεδομένα τους από αρχείο κειμένου αποθηκευμένο στον σκληρό δίσκο. Ένα τετράγωνο το οποίο είναι η πιο απλή ως προς την υλοποίηση της επιφάνεια αποτελείται από τέσσερις κορυφές $\mathbf{v_0}$ **(x, y, z)** $\mathbf{v_1}$ **(x, y, z)** $\mathbf{v_2}$ **(x, y, z)** $\mathbf{v_3}$ **(x, y, z),** μια υφή $\mathbf{vt_0}$ (u, v) $\mathbf{vt_1}$ (u, v) $\mathbf{vt_2}$ (u, v) $\mathbf{vt_3}$ (u, v) και ένα κανονικό διάνυσμα **vn (x, y, z)**.

Έστω μια τετράγωνη επιφάνεια στον 3Δ χώρο με κορυφές, τις $\mathbf{v_0\,(0,\,0,\,0)}$ $\mathbf{v_1\,(1,\,0,\,0)}$ $\mathbf{v_2\,(1,\,0,\,1)}$ $\mathbf{v_3\,(0,\,0,\,1)}$, με συντεταγμένες υφής τις $\mathbf{vt_0\,(0,\,0)}$ $\mathbf{vt_1\,(1,\,0)}$ $\mathbf{vt_2\,(1,\,1)}$ $\mathbf{vt_3\,(0,\,1)}$ και με κανονικό διάνυσμα **vn (0, 1, 0)** επειδή η επιφάνεια 'κοιτάει' προς τον άξονα y.

```
       x    y    z
1     0.0  0.0  0.0  -> v0
2     1.0  0.0  0.0  -> v1
3     1.0  0.0  1.0  -> v2
4     0.0  0.0  1.0  -> v3

       u    v
1     0.0  0.0  -> vt0
2     1.0  0.0  -> vt1
3     1.0  1.0  -> vt2
4     0.0  1.0  -> vt3

       x    y    z
1     0.0  1.0  0.0  -> vn0
```

Με τα παραπάνω δεδομένα η επιφάνεια δεν είναι ακόμη δυνατό να υλοποιηθεί γιατί δεν έχει ακόμη δηλωθεί η σύνδεση μεταξύ κορυφών, συντεταγμένων υφής και συντεταγμένων του κανονικού διανύσματος. Στην OpenGl αυτή η σύνδεση καταστεί μια επιφάνεια δυνατή. Η δήλωση είναι της μορφής. f0 v0/ vt0/ vn0 v1/ vt1/ vn0 v2/ vt2/ vn0 v3/ vt3/ vn0

```
1  1/ 1/ 1    2/ 2/ 1    3/ 3/ 1    4/ 4/ 1   -> οι αριθμοί δηλώνουν τις γραμμές
```

```
4                                   -> πληθος κορυφων
0.0  0.0  0.0                       -> 1
1.0  0.0  0.0                       -> 2
1.0  0.0  1.0                       -> 3
0.0  0.0  1.0                       -> 4

4                                   -> πληθος συντεταγμενων υφης
0.0  0.0                            -> 1
1.0  0.0                            -> 2
1.0  1.0                            -> 3
0.0  1.0                            -> 4

1                                   -> πληθος συντεταγμενων κανονικου δυανισματος
0.0  1.0  0.0                       -> 1

1                                   -> πληθος επιφανειων
1/ 1/ 1  2/  2/ 1  3/ 3/ 1  4/ 4/ 1    -> 1
```

Βήμα 2: Διάβασμα δεδομένων από την OpenGl.

Κατασκευη structs για αποθηκευση των δεδομενων.

```
struct sPoint
{
        float x, y, z; };           the face indices of object
struct sTexture
{
        float u, v; };          the texture coordinates of faces
struct sNormal
{
        float x, y, z;  };              the normals of object faces
struct sPlane
```

```
{
        unsigned int points[3];
        unsigned int normals[3];
        unsigned int textures[3];  };

struct glObject
{
        GLuint nPlanes, nPoints, nTextures, nNormals;        the data multitude of each struct

        sPoint points[2500];            max points of objects faces
        sPlane planes[2500];            max faces of objects
        sTexture uv[3200];              max texture coordinates of objects faces
        sNormal norm[2000];             Max normals of objects faces
};
```

Κατασκευή του loader για το διάβασμα των δεδομένων.

```
inline int ReadObject(char *st, glObject *o){

  FILE *file;
  unsigned int i;

  file = fopen(st, "r");
  if (!file) return FALSE;

  fscanf(file, "%d", &(o->nPoints));                    "%d"→ expects to see an integer
  for (i=1;i<=o->nPoints;i++)
  {
    fscanf(file, "%f", &(o->points[i].x));              "%f"→ expects to see a float
    fscanf(file, "%f", &(o->points[i].y));
    fscanf(file, "%f", &(o->points[i].z)); }

  fscanf(file, "%d", &(o->nTextures));
  for (i=0;i<o->nTextures;i++)
  {
        fscanf(file, "%f", &(o->uv[i].u));
        fscanf(file, "%f", &(o->uv[i].v));  }

  fscanf(file, "%d", &(o->nNormals));

  for (i=0;i<o->nNormals;i++)
  {
        fscanf(file, "%f", &(o->norm[i].x));
        fscanf(file, "%f", &(o->norm[i].y));
        fscanf(file, "%f", &(o->norm[i].z));  }

  fscanf(file, "%d", &(o->nPlanes));

  for (i=0;i<o->nPlanes;i++)
  {
       fscanf(file, "%d/", &(o->planes[i].points[0]));
        fscanf(file, "%d/", &(o->planes[i].textures[0]));
        fscanf(file, "%d", &(o->planes[i].normals[0]));

       fscanf(file, "%d/", &(o->planes[i]. points[1]));
        fscanf(file, "%d/", &(o->planes[i].textures[1]));
        fscanf(file, "%d", &(o->planes[i].normals[1]));

       fscanf(file, "%d/", &(o->planes[i]. points[2]));
        fscanf(file, "%d/", &(o->planes[i].textures[2]));
        fscanf(file, "%d", &(o->planes[i].normals[2]));   }

  return TRUE;
}
```

Βήμα 3 : Αρχικοποίηση του αντικειμένου

```
glObject CUBE;

if (!ReadObject("Data/moor.txt", &CUBE))
{
        return FALSE;
}
```

Βήμα 4 : Κατασκευή επιφάνειας με την GL_TRIANGLES **της OpenGl.**

```
void DrawGLObject(glObject o)
{
        unsigned int i, j;

        glBegin(GL_TRIANGLES);
```

```
        for (i=0; i<o.nPlanes; i++)
        {
                for (j=0; j<3; j++)
                {
                         glNormal3f(o.norm[o.planes[i].normals[j]].x,
                                    o.norm[o.planes[i].normals[j]].y,
                                    o.norm[o.planes[i].normals[j]].z);

                        glTexCoord2f(o.uv[o.planes[i].textures[j]].u,
                                     o.uv[o.planes[i].textures[j]].v);

                        glVertex3f(o.points[o.planes[i].p[j]].x,
                                   o.points[o.planes[i].p[j]].y,
                                  o.points[o.planes[i].p[j]].z);
                }
        }
        glEnd();
}
```

Βήμα 5 : Καλεσμα της συναρτησης DrawGLObject

```
DrawGLObject(CUBE);
```