Κεφάλαιο 5
Απόδοση τρισδιάστατων σκηνών – Κινούμενα γραφικά

Εισαγωγή

Στο Κεφάλαιο αυτό αναλύονται τεχνικές ορθής απόδοσης τριαδιάστατων σκηνών καθώς και κινουμένων σκηνών. Για την ορθή αναπαράσταση τρισδιάστατων σκηνών απαιτείται η τήρηση προφανών κανόνων απόκρυψης επιφανειών, κάτι που επιτυγχάνεται με την τεχνική καταστολής κρυμμένων επιφανειών. Επίσης αναλύονται τεχνικά ζητήματα σε ότι αφορά την υλοποίηση κινουμένων γραφικών και αναλύεται η τεχνική της διπλής ενταμίευσης για τη βέλτιστη απόδοσή τους. Επίσης, δίνονται οι εντολές σχεδίασης ευρέως χρησιμοποιούμενων τριασδιάστατων τετραγωνικών επιφανειών όπως λ.χ. σφαιρών, κυλίνδρων και κώνων. Επιπρόσθετα γίνεται αναφορά στη μίξη χρωμάτων και τη χρήση της για την προσωμοίωση διαφανών επιφανειών.

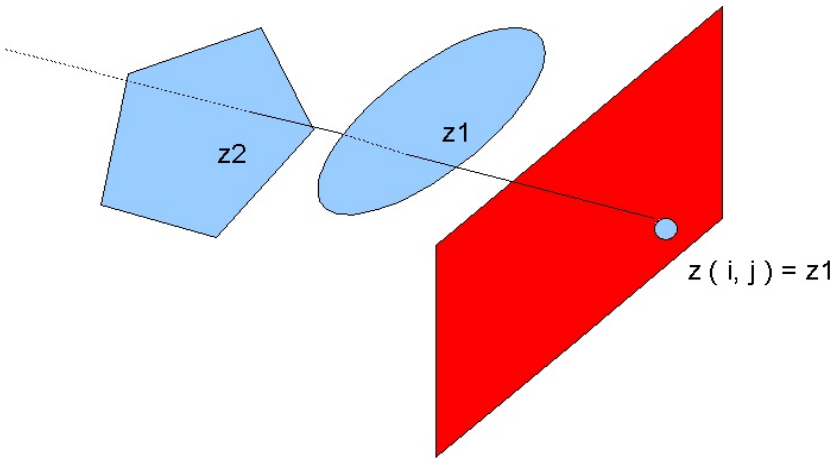
5.1 Καταστολή κρυμμένων επιφανειών

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 με 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 με 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 $z = 0$. 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. 5.1.



Σχ. 5.1: 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 στην εντολή ***glEnable*** το όρισμα *GL_DEPTH_TEST*.

glEnable(GL_DEPTH_TEST);

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

glutInitDisplayMode(GL_DEPTH);

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

```
glClear(GL_COLOR_BUFFER_BIT);
glClear(GL_DEPTH_BUFFER_BIT);
```

ή με μία εντολή καθαρισμού

```
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
```

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

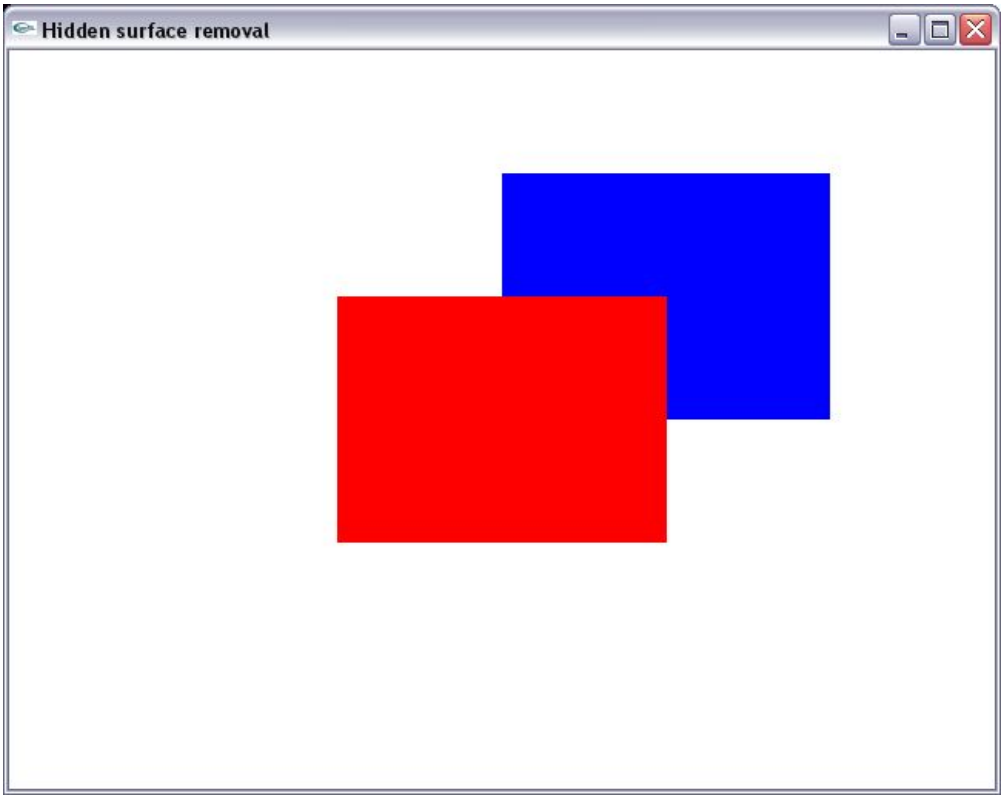
void glClearDepth(GLdouble maxDepth);

Το όρισμα *maxDepth* ορίζει τη μέγιστη τιμή βάθους που θα χρησιμοποιείται κατά τον καθαρισμό του ενταμιευτή βάθους)

Παράδειγμα: Καταστολή κρυμμένων επιφανειών

```
#include <glut.h>
GLdouble eyeX;
GLdouble eyeY;
GLdouble eyeZ;
GLdouble toX;
GLdouble toY;
GLdouble toZ;

void display()
{
	glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
	//Drawing two polygons with overlapping projections

	//The closest polygon is drawn in red
	glColor3f(1,0,0);
	glBegin(GL_POLYGON);
	glVertex3f(-10,-10,-10);
	glVertex3f(10,-10,-10);
	glVertex3f(10,10,-10);
	glVertex3f(-10,10,-10);
	glEnd();

	//The polygon lying further is drawn in blue
	glColor3f(0,0,1);
	glBegin(GL_POLYGON);
	glVertex3f(0,0,-20);
	glVertex3f(20,0,-20);
	glVertex3f(20,20,-20);
	glVertex3f(0,20,-20);
	glEnd();

	glFlush();
}

int main (int argc,char **argv)
{
	eyeX=0;
	eyeY=0;
```

```
        eyeZ=0;
        toX=0;
        toY=0;
        toZ=-1;

        glutInit(&argc,argv);
        glutInitWindowPosition(50,50);
        glutInitWindowSize(640,480);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutCreateWindow("Hidden surface removal");

        glClearColor(1,1,1,0);

        glEnable(GL_DEPTH_TEST);

        glMatrixMode(GL_MODELVIEW);
        gluLookAt(eyeX,eyeY,eyeZ,toX,toY,toZ,0,1,0);

        glMatrixMode(GL_PROJECTION);

        glOrtho(-30,30,-30,30,5,50);

        glutDisplayFunc(display);
        glutMainLoop();
        return 0;
}
```



5.2 Τρισδιάστατες επιφάνειες

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 και GLU.

α) 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****.

β) 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".

Στη GLU, 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;

Η αρχικοποίηση του αντικειμένου *qObj* γίνεται με την εντολή ***gluNewQuaric***
GLUquadric *gluNewQuadric( );

Η οποία επιστρέφει δείκτη σε αντικείμενο της κλάσης *GLUquadric*.

Η σύνταξη αρχικοποίησης μιας τετραγωνικής επιφανείας έχει λοιπόν τη μορφή

qObj = gluNewQuadric( );

Κατόπιν αποδίδουμε στο αντικείμενο *qObj* μια συγκεκριμένη επιφάνεια χρησιμοποιώντας εντολές που αναλύονται για κάθε σχήμα ξεχωριστά στη συνέχεια.

Για να διαγράψουμε ένα αντικείμενο χρησιμοποιούμε την εντολή ***gluDeleteQuadric***:

void GLUdeleteQuadric(GLUquadric *quadObject);

όπου *quadObject* το αντικείμενο προς διαγραφή.

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.

Στη συνέχεια αναφέρουμε τις πιο συχνά χρησιμοποιούμενες τρισδιάστατες τετραγωνικές επιφάνειες.

5.2.1 Κύβος

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

void glutWireCube (GLdouble edgeLength);

για τη σχεδίαση του περιγράμματος κύβου και

void glutSolidCube (GLdouble edgeLength);

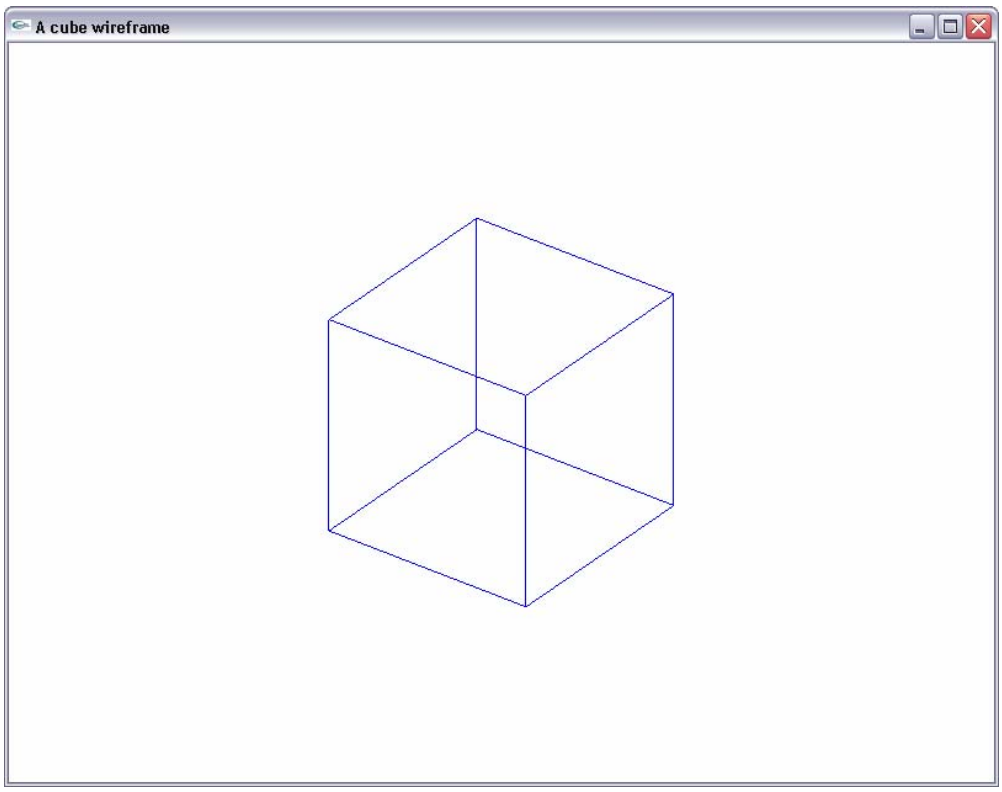
για τη σχεδίαση συμπαγούς κυβικού σχήματος,

Η παράμετρος *edgeLength* καθορίζει το μήκος των ακμών.

Εξ' ορισμού, ο κύβος σχεδιάζεται με το κέντρο του στην αρχή του συστήματος συντεταγμένων και με τις έδρες του παράλληλες προς τα επίπεδα $X-Y$, $X-Z$ και $Y-Z$ του συστήματος συντεταγμένων σκηνής. Για τη σχεδίαση του κύβου σε διαφορετική θέση της σκηνής, προηγείται ο κατάλληλος μετασχηματισμός μοντέλου.

Παράδειγμα: Σχεδίαση κυβικού πλέγματος

```
#include <glut.h>

void display()
{
	glColor3f(0,0,1);
	glClearColor(1,1,1,0);

	glClear(GL_COLOR_BUFFER_BIT);

	glutWireCube(40);

	glFlush();
}

int main(int argc, char **argv)
{
	glutInit(&argc,argv);
	glutInitWindowPosition(50,50);
	glutInitWindowSize(800,600);
	glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
	glutCreateWindow("A cube wireframe");

	glMatrixMode(GL_PROJECTION);
	glOrtho(-80,80,-60,60,0,100);

	glMatrixMode(GL_MODELVIEW);
	gluLookAt(-30,-30,40,0,0,0,0,1,0);

	glutDisplayFunc(display);
	glutMainLoop();

	return 0;
  }
```



5.2.2 Σφαίρα

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

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

Οι συντεταγμένες x, y, z των σημείων της σφαίρας δίνονται από τις παραμετρικές σχέσεις

$$\begin{aligned} x &= r \cdot \cos\theta \cdot \cos\phi \\ y &= r \cdot \cos\phi \cdot \sin\theta \\ z &= r \cdot \sin\theta \end{aligned} \qquad \phi \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right], \quad \theta \in [-\pi, \pi]$$

Η βιβλιοθήκη GLUT περιέχει για τη σχεδιάση σφαιρών την εντολή

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

για τη σχεδίαση σφαιρικού πλέγματος, καθώς και την εντολή

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

για τη σχεδιάση μιας συμπαγούς σφαιρικής επιφανείας.

Η παράμετρος *radius* δηλώνει την ακτίνα της σφαίρας. Το όρισμα *slices* δηλώνει το πλήθος των κατακορύφων υποδιαιρέσεων (μεσημβρινοί), των υποδιαιρέσεων δηλαδή που μετράει ο παρατηρητής στη σφαίρα όταν τη διατρέχει οριζόντια και οι πόλοι της βρίσκονται στην πάνω και κάτω θέση. Το όρισμα *stacks* δηλώνει το πλήθος των οριζοντίων υποδιαιρέσεων (γεωγραφικά πλάτη) που επιλέγουμε. Ουσιαστικά, η σφαίρα, όπως και όλες οι καμπύλες επιφάνειες, προσεγγίζεται από ένα πολυγωνικό πλέγμα. Οι διαστάσεις των στοιχειωδών πολυγώνων, άρα και η πιστότητα στην αναπαράστασή της σφαίρας, εξαρτάται από το πλήθος των οριζοντίων και κατακορύφων υποδιαιρέσεων (οι οποίες αποτελούνται από ευθύγραμμα τμήματα).

Οι εντολές ***glutWireSphere*** και ***glutSolidSphere*** θεωρούν το κέντρο της σφαίρας στην αρχή του καρτεσιανού συστήματος συντεταγμένων σκηνής. Επίσης, οι πόλοι της σφαίρας (δηλαδή τα σημεία από τα οποία αναχωρούν και στα οποία καταλήγουν οι μεσημβρινοί) βρίσκονται επί του άξονα z (στα σημεία $z=-r$ και $z=r$). Επομένως, για να σχεδιαστεί μια σφαίρα σε άλλη θέση της σκηνής ή με διαφορετικό προσανατολισμό, θα πρέπει να προηγηθεί ο κατάλληλος μετασχηματισμός μοντέλου.

Μπορούμε επίσης να σχεδιάσουμε μια σφαιρική επιφάνεια χρησιμοποιώντας την εντολή ***gluSphere*** της βιβλιοθήκης GLU:

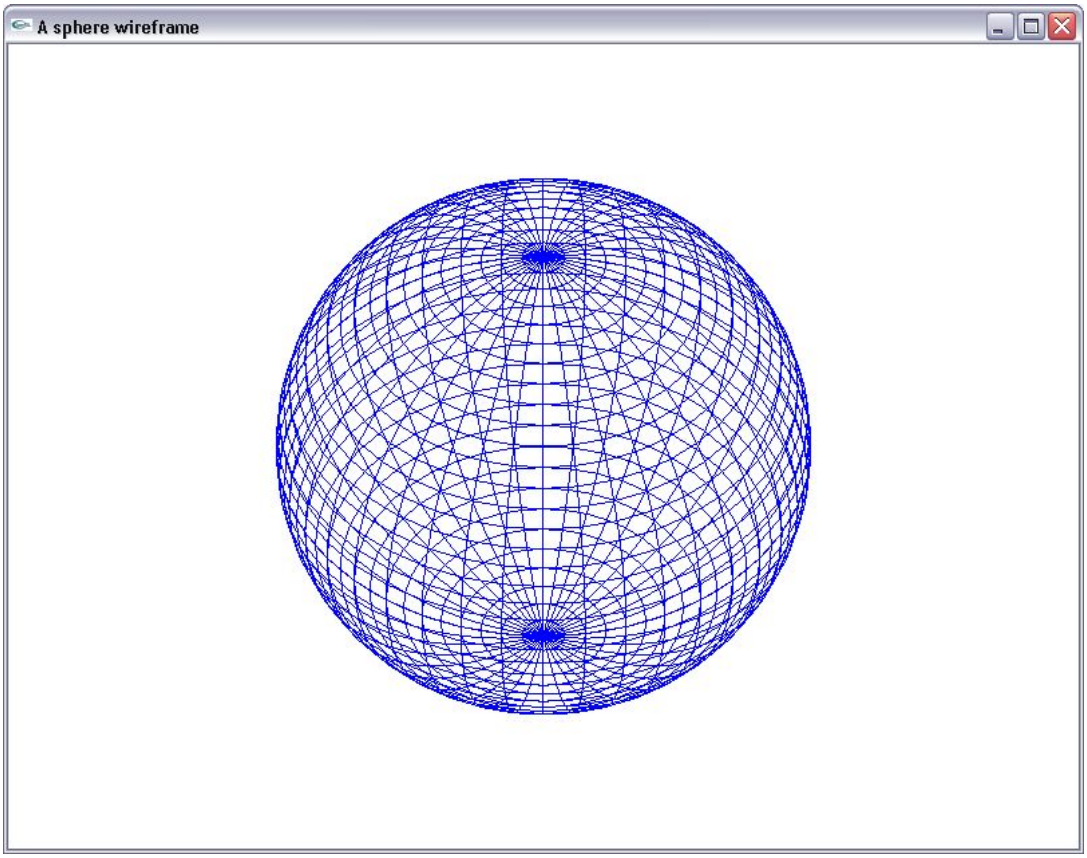
void gluSphere(GLUquadric *qObj, GLdouble radius, GLint slices, GLint stacks);

όπου *qObj* το αντικείμενο στο οποίο αντιστοιχίζουμε τη σφαιρική επιφάνεια.

Παράδειγμα: Σχεδίαση περιγράμματος σφαίρας με τη βιβλιοθήκη GLUT

```
#include <glut.h>

void display()
{
	glColor3f(0,0,1);
	glClearColor(1,1,1,0);

	glClear(GL_COLOR_BUFFER_BIT);

	glutWireSphere(40,40,40);

	glFlush();
}

int main(int argc, char **argv)
{
	glutInit(&argc,argv);
	glutInitWindowPosition(50,50);
	glutInitWindowSize(800,600);
	glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
	glutCreateWindow("A sphere wireframe");
```

```
        glMatrixMode(GL_PROJECTION);
        glOrtho(-80,80,-60,60,0,100);

        glMatrixMode(GL_MODELVIEW);
        gluLookAt(0,-40,40,0,0,0,0,1,0);

        glutDisplayFunc(display);
        glutMainLoop();

        return 0;
}
```



5.2.3 Κώνος

Για τη σχεδίαση κώνων η βιβλιοθήκη GLUT παρέχει την εντολή

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

για τη σχεδίαση κωνικού περιγράμματος και την εντολή

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

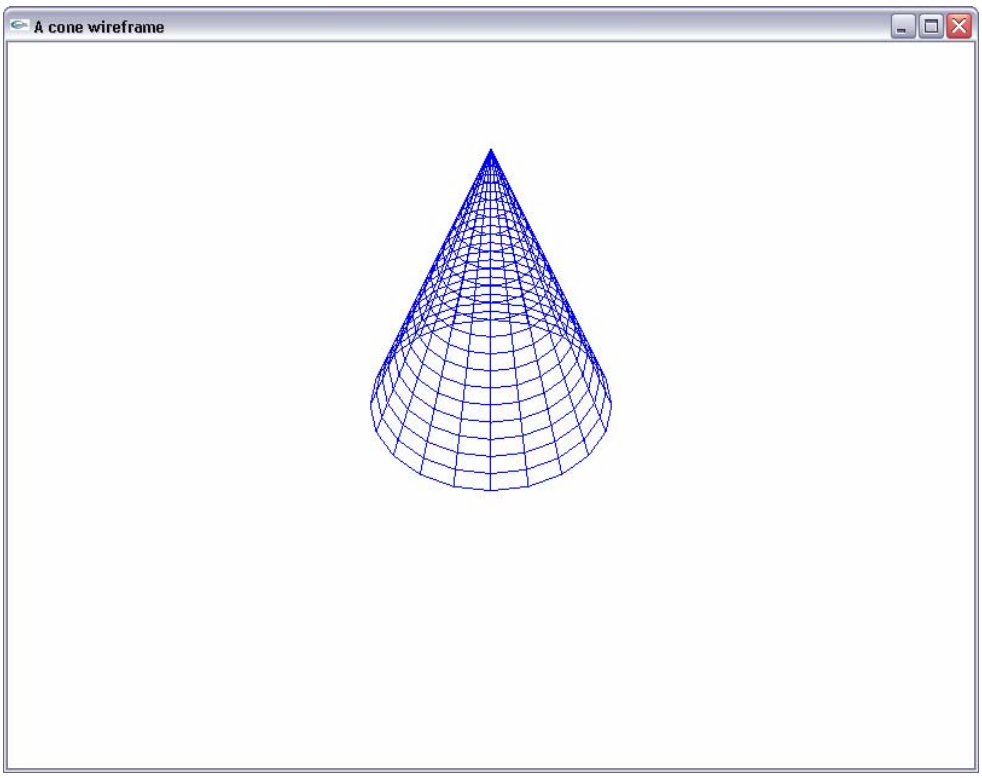
για τη σχεδιάση μιας συμπαγούς κωνικής επιφανείας. Η παράμετρος *base* ορίζει την ακτίνα της βάσης του κώνου και η παράμετρος *height* ορίζει το ύψος του. Η Η παράμετρος *slices* καθορίζει το πλήθος των

οριζοντίων υποδιαιρέσεων που χρησιμοποιούνται για την προσέγγιση του κώνου (μετρούνται διατρέχοντας την περιφέρεια μιας διατομής του κώνου) Η παράμετρος *stacks* καθορίζει το πλήθος των κατακόρυφων υποδιαιρέσεων που χρησιμοποιούνται για την προσέγγιση του κώνου (μετρούνται διατρέχοντας τον κώνο από τη βάση ως την κορυφή του).

Οι εντολές ***glutWireCone*** και ***glutSolidCone*** τοποθετούν το κέντρο της βάσης του κώνου στην αρχή του συστήματος συντεταγμένων σκηνής και ο άξονας του κώνου ακολουθεί τον άξονα Oz.

Παράδειγμα: Σχεδιάση περιγράμματος κωνικής επιφανείας

```
#include <glut.h>

void display()
{
	glColor3f(0,0,1);
	glClearColor(1,1,1,0);

	glClear(GL_COLOR_BUFFER_BIT);

	glutWireCone(20,60,20,20);

	glFlush();
}

int main(int argc, char **argv)
{
	glutInit(&argc,argv);
	glutInitWindowPosition(50,50);
	glutInitWindowSize(800,600);
	glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
	glutCreateWindow("A cone wireframe");

	glMatrixMode(GL_PROJECTION);
	glOrtho(-80,80,-60,60,0,100);

	glMatrixMode(GL_MODELVIEW);
	gluLookAt(0,-40,40,0,0,0,0,1,0);

	glutDisplayFunc(display);
	glutMainLoop();

	return 0;
}
```



5.2.4 Κύλινδρος

Η σχεδίαση κυλινδρικών επιφανειών εκτελείται με την εντολή ***gluCylinder*** της βιβλιοθήκης GLU:

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

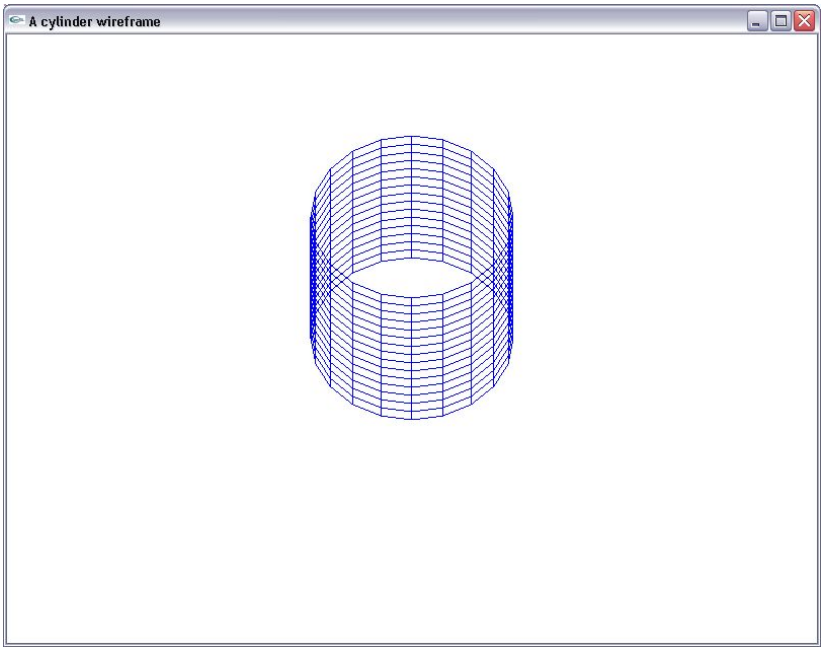
όπου *qObj* δείκτης στο αντικείμενο της κυλινδρικής επιφάνειας. Μπορούμε να ορίσουμε διαφορετική ακτίνα για την κυκλική επιφάνεια της βάσης και της κορυφής του κυλίνδρου μέσω των ορισμάτων *baseRadius* και *topRadius* αντίστοιχα. Η παράμετρος *height* εκφράζει το ύψος του κυλίνδου. Τα ορίσματα *slices* και *stacks* ορίζουν το πλήθος των οριζοντίων και κάθετων υποδαιρέσεων που προσεγγίζουν την επιφάνεια του κυλίνδρου.

Ο κύλινδρος τοποθετείται με τη βάση του στο επίπεδο XY. Το κέντρο της βάσης βρίσκεται στην αρχή των αξόνων και το σχήμα εκτείνεται προς το θετικό ημιάξονα z .

Παράδειγμα: Σχεδίαση κυλινδρικού πλέγματος

```
#include <glut.h>

GLUquadric *cylinder;

void display()
{
```

```
	glColor3f(0,0,1);
	glClearColor(1,1,1,0);

	glClear(GL_COLOR_BUFFER_BIT);

	gluQuadricDrawStyle(cylinder,GLU_LINE);
	gluCylinder(cylinder,20,20,40,20,15);

	glFlush();
}

int main(int argc, char **argv)
{
	glutInit(&argc,argv);
	glutInitWindowPosition(50,50);
	glutInitWindowSize(800,600);
	glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
	glutCreateWindow("A cylinder wireframe");

	glMatrixMode(GL_PROJECTION);
	glOrtho(-80,80,-60,60,0,100);

	glMatrixMode(GL_MODELVIEW);
	gluLookAt(0,-30,40,0,0,0,0,1,0);

	cylinder=gluNewQuadric();

	glutDisplayFunc(display);
	glutMainLoop();

	return 0;
}
```



5.2.5 Κυκλικός δίσκος – Δακτύλιος

Ο κυκλικός δίσκος, σε αντίθεση με τον κύλινδρο, είναι επίπεδος. Ορίζεται με την εντολή ***gluDisk*** της βιβλιοθήκης GLU. Με την ίδια εντολή, σχεδιάζεται επίσης και ο δακτύλιος, η επιφάνεια του οποίου

εκτείνεται μεταξύ μιας εσωτερικής και μιας εξωτερικής ακτίνας.

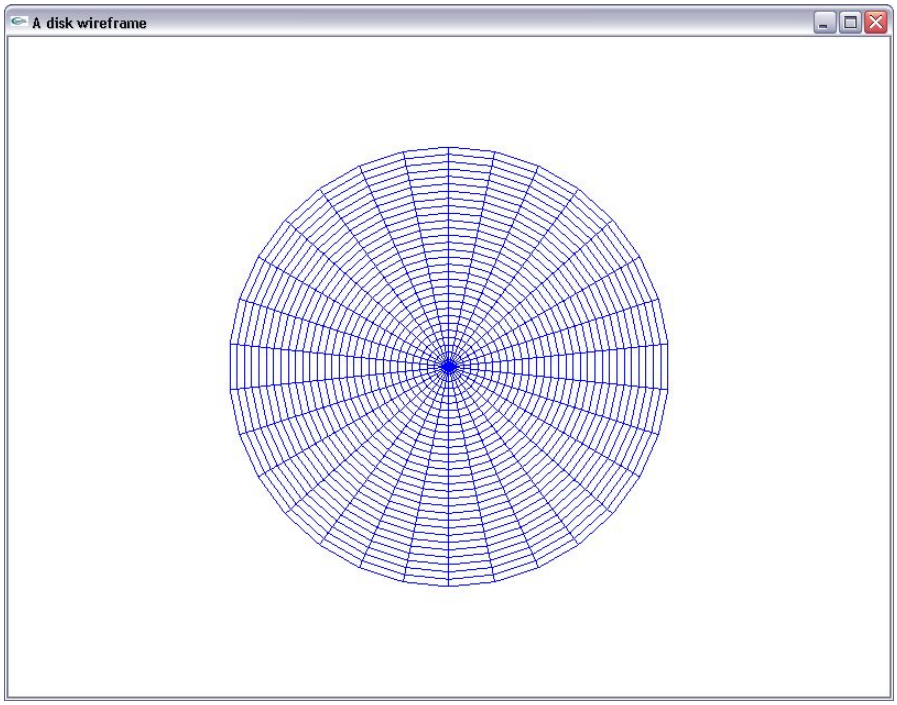
void gluDisk (GLUquadric *quadObject, GLdouble innerRadius, GLdouble outerRadius, Glint slices, Glint loops);

Το όρισμα *quadObject* εκφράζει το αντικείμενο στο οποίο αντιστοιχίζουμε την επιφάνεια. Η παράμετρος *innerRadius* δηλώνει την εσωτερική ακτίνα από την οποία ξεκινάει ο σχηματισμός του δακτυλίου. Για *innerRadius=0* σχεδιάζουμε έναν κυκλικό δίσκο. Η παράμετρος *outerRadius* ορίζει την ακτίνα μέχρι την οποία εκτείνεται ο δίσκος ή ο δακτύλιος. Η παράμετρος *slices* ορίζει το πλήθος των υποδιαιρέσεων (“φετών”) που μετράμε διαγράφοντας κυκλική πορεία σταθερής ακτίνας. Η παράμετρος *loops* ορίζει το πλήθος των ομόκεντρων κύκλων που χρησιμοποιούνται για την προσέγγιση της επιφάνειας και που μετρούνται αναχωρώντας από το κέντρο του κύκλου και με κίνηση προς την την περιφέρεια.

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

Παράδειγμα: Σχεδίαση πλέγματος κυκλικού δίσκου

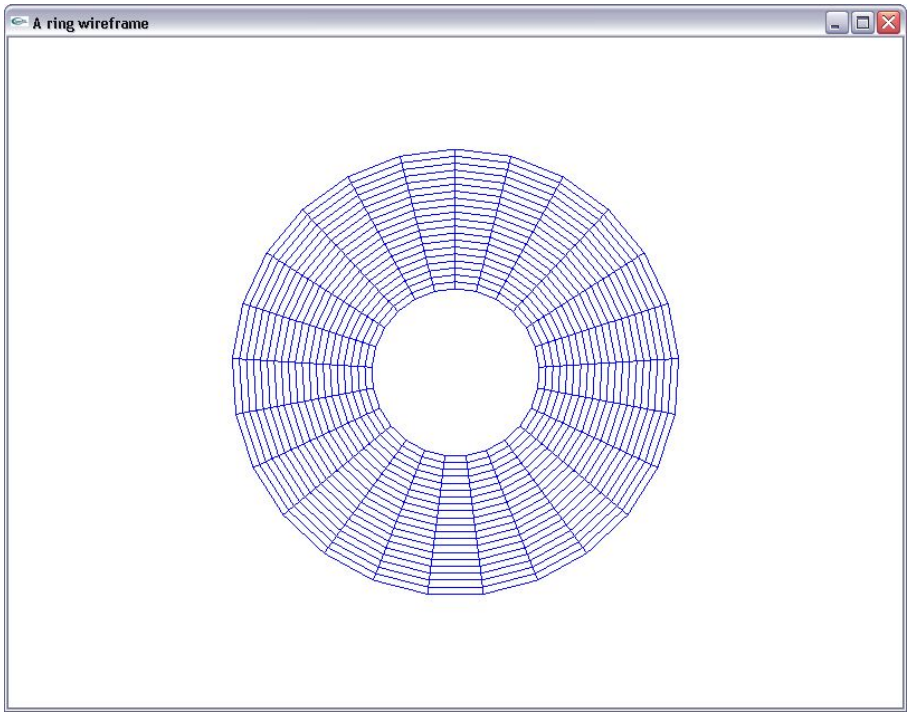
```
#include <glut.h>

GLUquadric *disk;

void display()
{
	glColor3f(0,0,1);
	glClearColor(1,1,1,0);

	glClear(GL_COLOR_BUFFER_BIT);

	gluQuadricDrawStyle(disk,GLU_LINE);
	gluDisk(disk,0,40,30,30);

	glFlush();
}

int main(int argc, char **argv)
{
	glutInit(&argc,argv);
	glutInitWindowPosition(50,50);
	glutInitWindowSize(800,600);
	glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
	glutCreateWindow("A disk wireframe");

	glMatrixMode(GL_PROJECTION);
	glOrtho(-80,80,-60,60,0,100);

	disk=gluNewQuadric();

	glutDisplayFunc(display);
	glutMainLoop();

	return 0;
}
```



Παράδειγμα: Σχεδίαση πλέγματος κυκλικού δακτυλίου

```
#include <glut.h>

GLUquadric *disk;

void display()
{
	glColor3f(0,0,1);
	glClearColor(1,1,1,0);

	glClear(GL_COLOR_BUFFER_BIT);

	gluQuadricDrawStyle(disk,GLU_LINE);
	gluDisk(disk,15,40,25,20);

	glFlush();
}

int main(int argc, char **argv)
{
	glutInit(&argc,argv);
	glutInitWindowPosition(50,50);
	glutInitWindowSize(800,600);
	glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
	glutCreateWindow("A ring wireframe");

	glMatrixMode(GL_PROJECTION);
	glOrtho(-80,80,-60,60,0,100);

	disk=gluNewQuadric();

	glutDisplayFunc(display);
	glutMainLoop();
```

```
        return 0;
}
```



5.2.6 Κυκλικός τομέας – Τομέας δακτυλίου

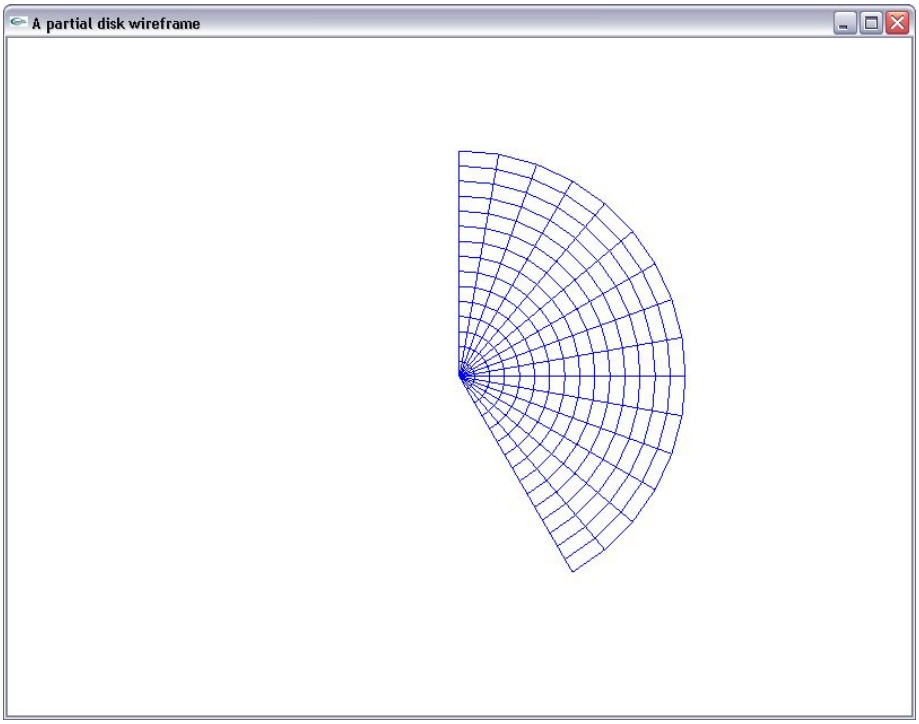
Αντί για ένα συμπαγή δίσκο ή δακτύλιο, έχουμε επίσης τη δυνατότητα σχεδίασης ενός μόνο γωνιακού τους τμήματος, δηλαδή ενός κυκλικού τομέα ή ενός τομέα δακτυλίου. Στην περίπτωση αυτη χρησιμοποιούμε την εντολή ***gluPartialDisk*** της GLU:

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

Η παράμετρος *startAngle* καθορίζει τη γωνία από την οποία ξεκινάει ο σχεδιασμός του σχήματος. Η παράμετρος *sweepAngle* καθορίζει το γωνιακό εύρος του δακτυλίου. Η γωνιακή θέση 0 αντιστοιχεί στην κατεύθυνση προς τα πάνω (προς το θετικό ημιάξονα y) και η τιμή της γωνιακής θέσης αυξάνεται κατά την αρνητική φορά.
Οι τομείς σχεδιάζονται επί του επιπέδου XY και με το κέντρο τους στην αρχή του συστήματος συντεταγμένων σκηνής.

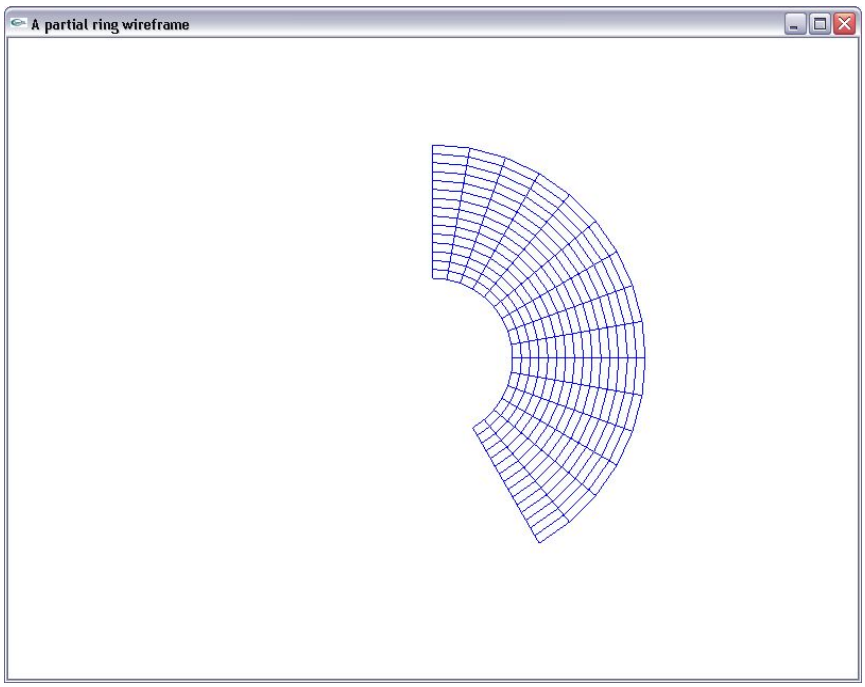
Παράδειγμα: Σχεδίαση πλέγματος κυκλικού τομέα

```
#include <glut.h>

GLUquadric *disk;

void display()
{
	glColor3f(0,0,1);
	glClearColor(1,1,1,0);

	glClear(GL_COLOR_BUFFER_BIT);

	gluQuadricDrawStyle(disk,GLU_LINE);
	gluPartialDisk(disk,0,40,15,15,0,150);

	glFlush();
}

int main(int argc, char **argv)
{
	glutInit(&argc,argv);
	glutInitWindowPosition(50,50);
	glutInitWindowSize(800,600);
	glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
	glutCreateWindow("A partial disk wireframe");

	glMatrixMode(GL_PROJECTION);
	glOrtho(-80,80,-60,60,0,100);

	disk=gluNewQuadric();

	glutDisplayFunc(display);
	glutMainLoop();

	return 0;
}
```



Παράδειγμα: Σχεδίαση πλέγματος τομέα δακτυλίου

```
#include <glut.h>

GLUquadric *disk;

void display()
{
	glColor3f(0,0,1);
	glClearColor(1,1,1,0);

	glClear(GL_COLOR_BUFFER_BIT);

	gluQuadricDrawStyle(disk,GLU_LINE);
	gluPartialDisk(disk,15,40,15,15,0,150);

	glFlush();
}

int main(int argc, char **argv)
{
	glutInit(&argc,argv);
	glutInitWindowPosition(50,50);
	glutInitWindowSize(800,600);
	glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
	glutCreateWindow("A partial ring wireframe");

	glMatrixMode(GL_PROJECTION);
	glOrtho(-80,80,-60,60,0,100);

	disk=gluNewQuadric();

	glutDisplayFunc(display);
	glutMainLoop();

	return 0;
}
```



5.2.7 Τόρος

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

Ο τόρος σχεδιάζεται με εντολές της βιβλιοθήκης GLUT. Ορίζονται δύο παραλλαγές. Η εντολή

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

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

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

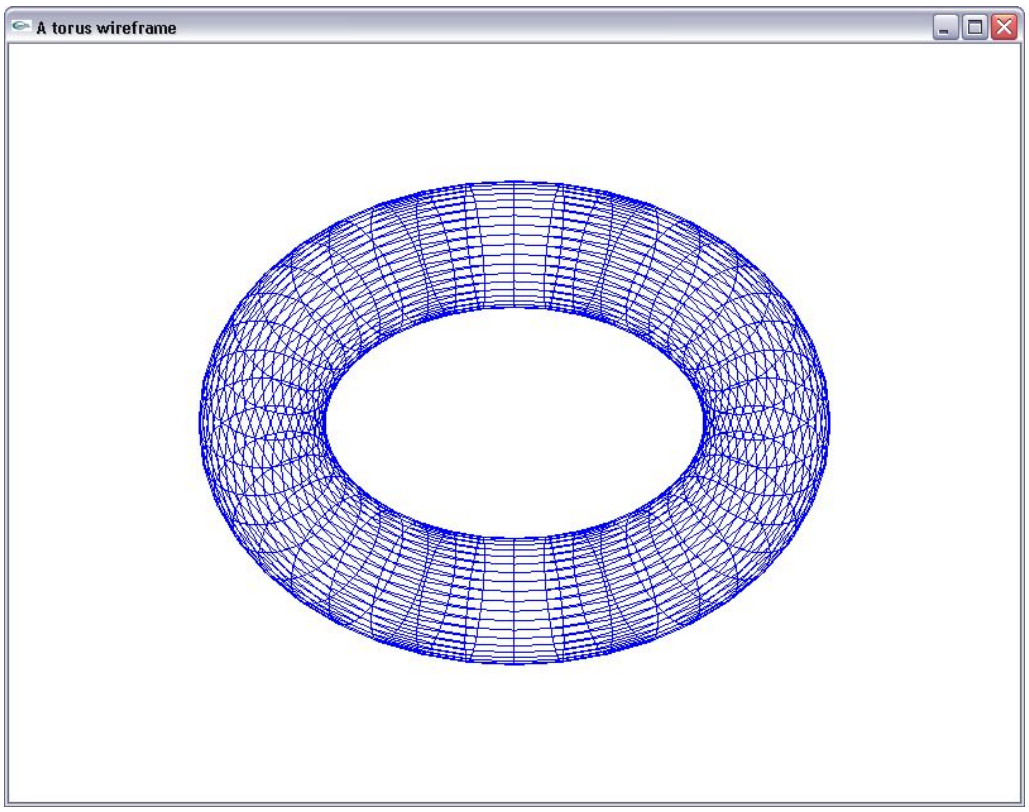
προσεγγίζει τον τόρο με συμπαγείς πολυγωνικές επιφάνειες.

Το όρισμα *innerRadius* εκφράζει την ακτίνα της κυκλικής διατομής του τόρου. Το όρισμα *outerRadius* είναι η απόσταση του κέντρου της διατομής του τόρου από τον άξονά του. Η παράμετρος *sides* καθορίζει το πλήθος των υποδιαιρέσεων που προσεγγίζουν τα όρια της διατομής του τόρου (μετρούνται καθώς διατρέχουμε την περιφέρεια μιας κυκλικής διατομής του τόρου). Η παράμετρος *rings* καθορίζει το πλήθος των κυκλικών διατομών που χρησιμοποιούμε για την προσέγγιση του τόρου (μετρούνται καθώς διατρέχουμε μια κλειστή διαδρομή κατά μήκος του τοροειδούς δακτυλίου).

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

Παράδειγμα: Σχεδίαση τοροειδούς περιγράμματος

```
#include <glut.h>

void display()
{
	glColor3f(0,0,1);
	glClearColor(1,1,1,0);

	glClear(GL_COLOR_BUFFER_BIT);

	glutWireTorus(10,40,40,40);

	glFlush();
}

int main(int argc, char **argv)
{
	glutInit(&argc,argv);
	glutInitWindowPosition(50,50);
	glutInitWindowSize(800,600);
	glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
	glutCreateWindow("A torus wireframe");

	glMatrixMode(GL_PROJECTION);
	glOrtho(-80,80,-60,60,0,100);

	glMatrixMode(GL_MODELVIEW);
	gluLookAt(0,-40,40,0,0,0,0,1,0);

	glutDisplayFunc(display);
	glutMainLoop();

	return 0;
}
```



5.3 Μίξη χρωμάτων

Κατά τη σχεδίαση επικαλυπτόμενων σχημάτων εάν οι συντεταγμένες των νέων σχημάτων επικαλύπτονται με τις συντεταγμένες προηγουμένων σχημάτων οι χρωματικές τιμές στις επικαλυπτόμενες θέσεις αντικαθίστανται με τις χρωματικές τιμές του νέου σχήματος. Ωστόσο στην περίπτωση επικαλυπτόμενων σχημάτων 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. Η χρωματική τιμή της πηγής ταυτίζεται με το χρώμα του σχήματος που σκοπεύουμε να συνδυάσουμε με τα υπάρχοντα χρώματα του ενταμιευτή χρώματος.
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. Έστω ότι για το στρώμα προορισμού έχουμε ορίσει τους συντελεστές μίξης $D = (D_r, D_g, D_b, D_a)$ και για το στρώμα πηγής τους συντελεστές μίξης $S = (S_r, S_g, S_b, S_a)$. Εάν το χρώμα του στρώματος προορισμού περιγράφεται με τις συνιστώσες (R_d, G_d, B_d, A_d) και το χρώμα του στρώματος πηγής περιγράφεται με τις συνιστώσες (R_s, G_s, B_s, A_s) τότε οι νέες αναμεμιγμένες χρωματικές τιμές RGBA που θα αποθηκευτούν στον ενταμιευτή χρώματος είναι:

$$(S_r \cdot R_s + D_r \cdot I_d, \quad S_g \cdot G_s + D_g \cdot G_d, \quad S_b \cdot B_s + D_b \cdot B_d, \quad S_a \cdot A_s + D_a \cdot A_d)$$

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:

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

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

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

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

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

GL_ONE_MINUS_DST_ALPHA: Επιλέγουμε ως συντελεστή μίξης για το εκάστοτε στρώμα το συμπλήρωμα

της συνιστώσας A_d ως προς τη μονάδα. $(1-A_d, 1-A_d, 1-A_d, 1-A_d)$.

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

Με τη μίξη χρωμάτων έχουμε τη δυνατότητα να προσομοιώσουμε φαινόμενα διαφάνειας. Στην περίπτωση αυτή, η διαφάνεια μιας επιφάνειας καθορίζεται με τη χρήση της alpha συνιστώσας της. Είναι σύνηθες να ορίζουμε μια επιφάνεια ως πλήρως διαφανή για τιμή alpha ίση με 1 και πλήρως αδιαφανή για τιμή alpha ίση με 0. Εάν θεωρήσουμε ότι η διαφανής επιφάνεια ορίζεται στο στρώμα πηγής και έχει συνιστώσες (R_s, G_s, B_s, A_s), προκειμένου να χρησιμοποιήσουμε τη συνιστώσα A_s 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 A_s (κοντά στη μονάδα) αναδεικνύουν σε μεγαλύτερο βαθμό το χρώμα του στρώματος προορισμού και υποβιβάζουν την απόδοση του στρώματος πηγής. Δηλαδή η τιμή A_s λειτουργεί ως συντελεστής διαφάνειας.

Προφανώς, όταν χρησιμοποιούνται συνιστώσες alpha για τον καθορισμό των συντελεστών μίξης, απαιτείται η δήλωση χρήσης του μοντέλου RGBA με την εντολή ***glutInitDisplayMode***:

glutInitDisplayMode(GLUT_RGBA);

Με το ακόλουθο παράδειγμα

```
glColor4f(1,0,0,0);
glBegin(GL_POLYGON);
……
glEnd();
glColor4f(0,1,0,0.7);
glBegin(GL_TRIANGLES);
…….....
glEnd();
```

αρχικά ορίζουμε ότι όλα τα σχήματα που θα δηλώνονται θα έχουν κόκκινο χρώμα και τιμή alpha ίση με 0 και σχεδιάζουμε μια πολυγωνική επιφάνεια με αυτά τα χαρακτηριστικά. Κατόπιν δηλώνουμε ότι τα σχήματα θα έχουν πράσινο χρώμα με τιμή alpha ίση με 0.7

Εάν ρυθμίσουμε το μοντέλο μίξης χρωμάτων με την εντολή

glBlendFunc(ONE_MINUS_SRC_ALPHA,GL_SRC_ALPHA);

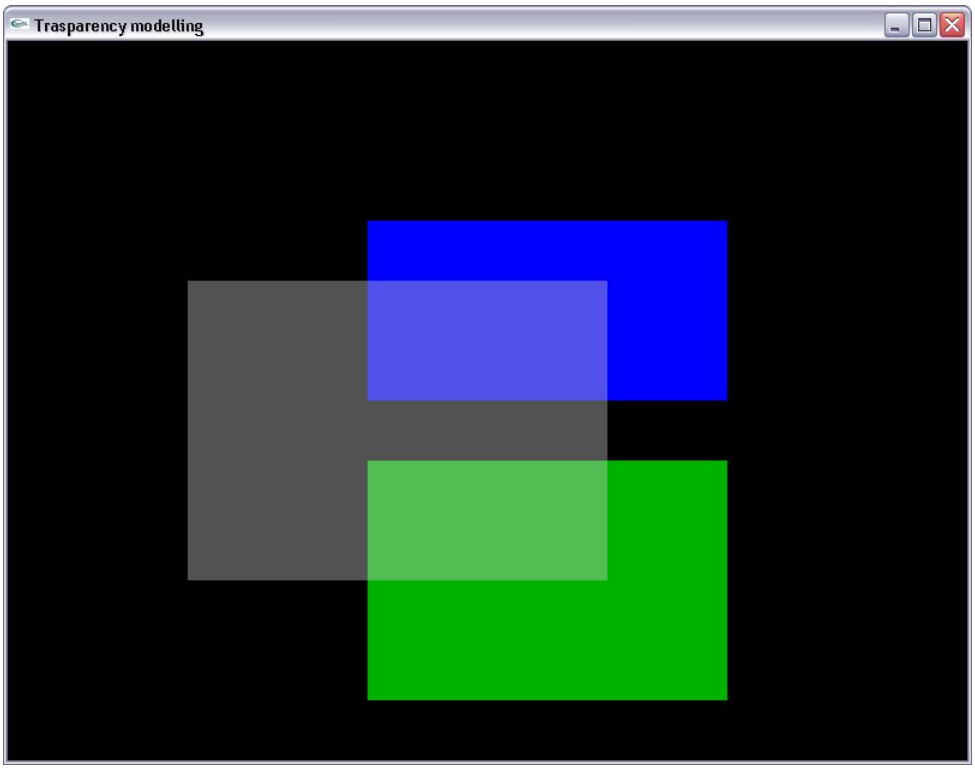
οι τιμές alpha λειτουργούν ως συντελεστές διαφάνειας. Επομένως το πράσινο τρίγωνο θα είναι ημιδιαφανές (συντελεστής διαφάνειας 0.7) και αν επικαλύψει το κόκκινο πολύγωνο, το χρώμα που θα αποδοθεί στις επικαλυπτόμενες περιοχές του ενταμιευτή είναι

$$(0.3\times 1+0.7\times 0,\quad 0.3\times 0+0.7\times 1,\ 0,\ 0.3\times 0+0.7\times 0,\ 0.7\times 0+0.3\times 0.7)$$

Παράδειγμα: Μοντελοποίηση διαφάνειας

```
#include <glut.h>

void init()
{
	glutInitWindowPosition(50,50);
	glutInitWindowSize(800,600);
	glutCreateWindow("Trasparency modelling");

	//Using the RGBA color apace
	glutInitDisplayMode(GLUT_SINGLE|GLUT_RGBA);

	glMatrixMode(GL_PROJECTION);
	glOrtho(-40,40,-30,30,0,50);

	glClearColor(0,0,0,0);

	//Enabling color blending
	glEnable(GL_BLEND);

	//Enabling hidden surface removal
	glEnable(GL_DEPTH_TEST);

	//Modelling transparency: S=(1-As,1-As,1-As,1-As) D=(As,As,As,As)
	glBlendFunc(GL_ONE_MINUS_SRC_ALPHA,GL_SRC_ALPHA);
}
void display()
{
	glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

	glBegin(GL_QUADS);

	//Drawing an blue opaque rectangle
	glColor4f(0,0,1,0);
	glVertex3f(-10,0,-20);
	glVertex3f(20,0,-20);
	glVertex3f(20,15,-20);
	glVertex3f(-10,15,-20);

	//Drawing a green opaque rectangle
	glColor4f(0,.7,0,0);
	glVertex3f(-10,-5,-20);
	glVertex3f(20,-5,-20);
	glVertex3f(20,-25,-20);
```

```
        glVertex3f(-10,-25,-20);

        //Drawing a transparent gray rectangle (As=0.6)
        glColor4f(0.8,0.8,0.8,0.6);
        glVertex3f(-25,-15,-10);
        glVertex3f(10,-15,-10);
        glVertex3f(10,10,-10);
        glVertex3f(-25,10,-10);

        glEnd();

        glFlush();
}
int main (int argc, char **argv)
{
        glutInit(&argc,argv);

        init();

        glutDisplayFunc(display);
        glutMainLoop();
}
```



5.4 Κινούμενα γραφικά

Στο Κεφάλαιο της διαχείρισης γεγονότων είδαμε ότι η δημιουργία κινούμενων γραφικών είναι εφικτή μεταβάλλοντας το σκηνικό και δίνοντας διαδοχικές εντολές επανασχεδιασμού της σκηνής. Όμως το αποτέλεσμα του αντίστοιχου παραδείγματος δεν ήταν ικανοποιητικό.

Ο λόγος στον οποίο οφειλόταν η χαμηλή ποιότητα του παραδείγματος είναι **οι ασύγχρονες διεργασίες**

εγγραφής και ανάγνωσης του ενταμιευτή χρωματικών τιμών (colour buffer). Ο ενταμιευτής χρωματικών τιμών περιέχει τις πληροφορίες χρώματος που χρειάζονται για την απεικόνιση της σκηνής. Οι πληροφορίες αυτές διαβάζονται από το μετατροπέα ψηφιακού σε αναλογικό σήμα (DAC) της οθόνης, προκειμένου να αναπαρασταθεί η σκηνή στην οθόνη. Η συχνότητα ανάγνωσης του ενταμιευτή χρωματικών τιμών κυμαίνεται μεταξύ 60-100Hz, ανάλογα με τη συχνότητα σάρωσης της οθόνης. Επίσης η μηχανή της OpenGL, κατά τη φάση σχηματισμού του επόμενου καρέ, μετά την εφαρμογή του μετασχηματισμού παρατήρησης, εγγράφει τις χρωματικές τιμές των pixels του καρέ **στον ίδιο ενταμιευτή**.

Όμως αυτές οι δύο διαδικασίες ανάγνωσης και εγγραφής δεν είναι συγχρονισμένες. Αυτό σημαίνει ότι κατά τη διάρκεια της σάρωσης του color buffer από τον DAC της οθόνης, ενδέχεται οι τιμές του να μεταβληθούν από την OpenGL, στα πλαίσια της σχεδίασης του επόμενου καρέ. Αποτέλεσμα αυτής της έλλειψης συγχρονισμού είναι το τρεμοπαίξιμο της σκηνής (flickering) και η υποβάθμιση της ποιότητας των κινούμενων γραφικών.

Για να αποφευχθεί το φαινόμενο αυτό, στις εφαρμογές κινουμένων γραφικών χρησιμοποιείται πάντα η τεχνική της **διπλής ενταμίευσης**. Στη διπλή ενταμίευση, οι εφαρμογές αντί για έναν, χρησιμοποιούν δύο ενταμιευτές χρωματικών τιμών: τον **ενταμιευτή προσκηνίου** και τον **ενταμιευτή παρασκηνίου**. Ο ενταμιευτής προσκηνίου περιέχει το τρέχον καρέ που σαρώνεται από τον DAC της οθόνης και απεικονίζεται στο χρήστη. Ο ενταμιευτής παρασκηνίου χρησιμοποιείται από την OpenGL ως αποθηκευτικός χώρος στον οποίο σχεδιάζεται το επόμενο καρέ. Όταν ολοκληρωθεί η σχεδίαση του επόμενου καρέ, κατά τη φάση της κάθετης επαναφοράς της δέσμης σάρωσης της οθόνης, οι δύο ενταμιευτές εναλλάσσουν τους ρόλους τους. Επομένως, ο πρώην ενταμιευτής παρασκηνίου λειτουργεί ως ενταμιευτής προσκηνίου, τα δεδομένα του σαρώνονται από τον DAC και σχεδιάζεται το επόμενο καρέ. Ο πρώην ενταμιευτής προσκηνίου λειτουργεί ως ενταμιευτής παρασκηνίου και είναι διαθέσιμος για το σχηματισμό του επόμενου καρέ. Η διαδικασία αυτή εκτελείται επαναληπτικά. Επομένως, **σε κάθε σάρωση, ο μετατροπέας DAC της οθόνης σαρώνει τα δεδομένα ενός και μόνο καρέ και αφού έχει ολοκληρωθεί η διαδικασία σχηματισμού του**. Το αποτέλεσμα της διπλής ενταμίευσης είναι η ~~δημιουργία~~ ομαλή μετάβαση μεταξύ των καρέ.

Στην OpenGL η διπλή ενταμίευση ενεργοποιείται δίνοντας στην εντολή ***glutInitDisplayMode*** το όρισμα *GLUT_DOUBLE*.

glutInitDisplayMode(GLUT_DOUBLE);

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

void glutSwapBuffers();

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

Παράδειγμα: Εφαρμογή διπλής ενταμίευσης

```
#include <glut.h>
GLuint x1=10;
GLuint y1=10;
GLuint x2=x1+50;
GLuint y2=y1+50;

GLuint angle;

void display()
{
	glClearColor(1,1,1,0);
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

	glColor3f(1,0,0);
	glRecti(x1,y1,x2,y2);
	glutSwapBuffers();
}

void animate()
{
	glMatrixMode(GL_MODELVIEW);
	//Increasing the angle of rotation in the modelview matrix.
	//If the angle exceeds 360 degrees it it adjusted in the range [0,360] (angle modulo 360).

	angle=(angle+1)%360;

	glLoadIdentity();

	//Define a rotation about the z-axis.
	glRotatef(angle,0,0,1);
	glutPostRedisplay();
}

int main(int argc, char** argv)
{
	angle=0;

	glutInit(&argc,argv);
	glutInitWindowPosition(50,50);
	glutInitWindowSize(640,480);
	glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
	glutCreateWindow("An animated square");

	glMatrixMode(GL_PROJECTION);
	gluOrtho2D(-160,160,-120,120);
```

```
	glutIdleFunc(animate);
	glutDisplayFunc(display);
	glutMainLoop();

	return 0;
}
```