Κεφάλαιο 1
Βασικές αρχές σχεδίασης

Εισαγωγή

Στο Κεφάλαιο αυτό επιχειρείται η εξοικείωση με τη φιλοσοφία και τον τρόπο λειτουργίας της μηχανής της OpenGL. Αρχικά παραθέτουμε τους βασικούς τύπους δεδομένων, καθώς και τις συμβάσεις σε ότι αφορά την ονομασία αριθμητικών παραμέτρων και συναρτήσεων. Επισημαίνουμε τη λειτουργία της OpenGL ως μηχανή καταστάσεων και περιγράφονται οι τρόποι με τους οποίους ο προγραμματιστής μπορεί να μεταβάλλει τη λειτουργία του συστήματος γραφικών, ρυθμίζοντας παραμέτρους και ενεργοποιώντας ή απενεργοποιώντας προσφερόμενες δυνατότητες.

Στην OpenGL, η σχεδίαση τόσο των απλών όσο και των πιό σύνθετων σημάτων εκτελείται βάσει κοινών κανόνων. Δίνονται οι τρόποι σχεδίασης ευρέως χρησιμοποιούμενων σχήματων, όπως είναι οι ευθείες, τα τρίγωνα και τα πολύγωνα, καθώς και επιπλέον πληροφορίες σχετικά με τις ιδιότητες πολυγώνων. Επιπλέον αναλύεται η δυνατότητα επαναχρησιμοποίησης κώδικα σχεδίασης μέσω των λιστών απεικόνισης .

1.1 Πρωτογενείς τύποι δεδομένων

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

Πίνακας 1: Τύποι δεδομένων στην OpenGL

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

1.2 Ονομασία συναρτήσεων - Συμβάσεις

Στην OpenGL, για ορισμένες εντολές ορίζονται πολλαπλές παραλλαγές, ανάλογα με:

- τον τύπο των ορισμάτων που δέχονται (π.χ. ακέραιοι ή πραγμάτικοί),
- τις διαστάσεις του χώρου (π.χ. σχεδίαση σε δύο ή τρείς διαστάσεις)
- τον αριθμό των συνιστωσών των χρωματικών τιμών (π.χ. τρεις στο μοντέλο RGB, τέσσερις στο μοντέλο RGB με μίξη χρωμάτων)
- τον τρόπο με τον οποίο επιλέγουμε να περάσουμε τις παραμέτρους στην εντολή (πέρασμα αριθμητικών τιμών (call by value) ή πέρασμα διανυσμάτων υπό τη μορφή μητρώων (call by reference)).

Ανάλογα λοιπόν με το συνδυασμό των παραπάνω παραμέτρων, ορισμένες εντολές της OpenGL παρουσιάζουν παράλλαγές. Τα ονόματα των παραλλαγών μιας εντολής καθορίζονται βάσει της εξής σύμβασης:

Όνομα εντολής + Άιάσταση χώρου/Πλήθος χρωματικών συνιστωσών + Πρωτογενής τύπος δεδομένων ορισμάτων + Τρόπος κλήσεως ορισμάτων

Στο τέλος δηλαδή του ονόματος της εντολής προσθέτονται επιθήματα που καθορίζουν πλήρως τις παραπάνω παραμέτρους.

Ως παράδειγμα θα επιλέξουμε την εντολη ***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. Έτσι λχ για τον καθορισμό ενός σημείου στο διδιάστατο χώρο που οι συντεταγμένες του δίνονται υπό τη μορφή πραγματικών αριθμών απλής ακρίβειας (*GLfloat*) με κλήση τιμής (call by value) η αντίστοιχη εντολή έχει τη μορφή:

glVertex2f (GLfloat x, GLfloat y);

ενώ η αντίστοιχη συνάρτηση στον τρισδιάστατο χώρο με ορίσματα ακεραίων θα έχει τη μορφή

glVertex3i(GLint x, GLint y, GLint z);

Επίσης οι συντεταγμένες ενός σημείου μπορουν να δοθούν στην εντολή ***glVertex**** δίνοντας το δείκτη ενός πίνακα που περιέχει τις τιμές (call by reference). Πχ ο πίνακας:

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

καθορίζει ένα σημείο στον τρισδιάστατο χώρο με συντεταγμένες $(x, y, z) = (1,2,3)$.

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

```
glVertex3fv(const GLfloat *coordArray);
```

Δίνοντας λοιπόν την εντολή

```
glVertex3fv(coord);
```

ορίζουμε το παραπάνω σημείο.

1.3 Η OpenGL ως μηχανή καταστάσεων

Το περιβάλλον της OpenGL μπορεί να χαρακτηριστεί ως μια μηχανή καταστάσεων. Με τον όρο "μηχανή καταστάσεων" αναφερόμαστε σε ένα περιβάλλον, το οποίο, σε κάθε χρονική στιγμή, λειτουργεί βάσει προκαθορισμένων **ιδιοτήτων (attributes)** ή αλλιώς **μεταβλητών κατάστασης (state variables)**. Οι μεταβλητές κατάστασης έχουν μια προκαθορισμένη αρχική τιμή η οποία μπορεί να μεταβληθεί κατά την πορεία της εκτέλεσης του κώδικα από τον προγραμματιστή. Επιπλέον, οι αρχικές τιμές τους ή οι τιμές που τους ανατέθηκαν την τελευταία φορά, παραμένουν ενεργές.

Δεδομένου ότι στην OpenGL οι αλγόριθμοι εκτελούνται επαναληπτικά, είναι σημαντικό ο προγραμματιστής να αρχικοποιεί τις μεταβλητές κατάστασης, όποτε αυτό είναι απαραίτητο, καθώς και να παρακολουθεί τις τιμές τους, ούτως ώστε να παράγει το επιθυμητό αποτέλεσμα σε κάθε κύκλο εκτέλεσης.

Ορισμένα παραδείγματα μεταβλητών κατάστασης που ορίζονται στην OpenGL είναι: το τρέχον χρώμα σχεδίασης, οι τιμές των μητρώων μετασχηματισμού και προβολής, το πάχος των σχεδιαζόμενων γραμμών, το χρώμα καθαρισμού της οθόνης κ.λ.π.

1.4 Ενεργοποίηση, απενεργοποίηση και επισκόπηση παραμέτρων κατάστασης

Η κατάσταση λειτουργίας της OpenGL, σε κάθε χρονική στιγμή, καθορίζεται από τις τιμές που αναθέτουμε σε ένα σύνολο προκαθορισμένων ιδιοτήτων. Μπορούμε να διακρίνουμε τις ιδιότητες σε δύο κατηγορίες:

<u>α) Ιδιότητες δύο καταστάσεων</u>:

Αυτές έχουν δύο πιθανές τιμές (*TRUE* ή *FALSE*) και καθορίζουν την υποστήριξη ή μη εξειδικευμένων

λειτουργιών, όπως λ.χ. τη μίξη χρωμάτων (blending), την καταστολή κρυμμένων επιφανειών (hidden surface removal), τη χρήση φωτορεαλισμού (shading) και την απόδοση υφής σε επιφάνειες (texture mapping).

Ο λόγος για τον οποίο πρέπει να δηλώνεται η ενεργοποίηση των δυνατοτήτων της OpenGL είναι καθαρά πρακτικός. Η ταυτόχρονη ενεργοποίηση όλων των δυνατών ιδιοτήτων, όταν αυτές δεν πρόκειται να αξιοποιηθούν από τον προγραμματιστή, θα προκαλούσε άσκοπη επιβάρυνση στο σύστημα. Συνεπώς, δίνοντας στον προγραμματιστή τη δυνατότητα επιλογής ως προς την ενεργοποίηση των προσφερομένων δυνατοτήτων, η OpenGL βελτιστοποιεί το υπολογιστικό φορτίο, καθώς υποστηρίζει μόνο τις εκείνες τις λειτουργίες που καλύπτουν τις εκάστοτε ανάγκες του προγραμματιστή.

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

void glEnable(GLenum cap);
void glDisable(Glenum cap);

όπου *cap* η ιδιότητα που επιθυμούμε να ενεργοποιήσουμε ή να απενεργοποιήσουμε.

Εάν επιθυμούμε λ.χ. να ενεργοποιήσουμε την καταστολή κρυμμένων επιφανειών, δίνουμε ως παράμετρο στην ***glEnable*** την προκαθορισμένη αριθμητική σταθερά *GL_DEPTH_TEST*.

```
glEnable(GL_DEPTH_TEST);
```

Παρομοίως, η απενεργοποίηση της παραμέτρου γίνεται με την εντολή

```
glDisable(GL_DEPTH_TEST);
```

Προκειμένου να ελέγξουμε αν μια ιδιότητα δύο καταστάσεων είναι ενεργοποιημένη ή απενεργοποιημένη χρησιμοποιούμε την εντολή:

GLboolean glIsEnabled(GLenum capability);

η οποία επιστρέφει *GL_TRUE* ή *GL_FALSE*, ανάλογα με το αν η εξεταζόμενη ιδιότητα είναι ενεργοποιημένη ή όχι.

β) Σύνθετες ιδιότητες κατάστασης:

Εκτός από τις μεταβλητές δύο καταστάσεων (ενεργοποιημένη ή απενεργοποιημένη) υπάρχουν μεταβλητές κατάστασης που μπορούν να πάρουν περισσότερες από δύο τιμές. Προφανώς στην περίπτωση αυτή δεν μπορούμε να χρησιμοποιήσουμε τις εντολές ***glEnable*** , ***glDisable*** και ***glIsEnabled*** για να αναθέσουμε ή να επιστρέψουμε την τιμή τους. Αντ' αυτού, στην OpenGL χρησιμοποείται ένα σύνολο εντολών ανάθεσης τιμών. Για τις μεταβλητές πολλών καταστάσεων, έχει καθοριστεί ένα σύνολο **εντολών επισκόπησης** (**query functions**) οι οποίες επιστρέφουν την τιμή ή τις τιμές που τις χαρακτηρίζουν:

void glGetBooleanv(Glenum parameterName, GLboolean *parameters);
void glGetIntegerv(Glenum parameterName, GLint *parameters);
void glGetFloatv(Glenum parameterName, GLfloat *parameters);
void glGetDoublev(Glenum parameterName, GLdouble *parameters);

όπου *parameterName* η εξεταζόμενη παράμετρος. Το όρισμα *parameters* είναι ένας δείκτης του μητρώου στο οποίο αποθηκεύονται οι τιμές που προσδιορίζουν την εκάστοτε παράμετρο.

Το ποιά παραλλαγή εντολής επισκόπησης θα χρησιμοποιήσουμε για να πάρουμε πληροφορία της τρέχουσας τιμής μιας μεταβλητής κατάστασης εξαρτάται από τον τύπο δεδομένων που θα επιλέξουμε για την αποθήκευση της τιμής της.

Π.χ. στην περίπτωση του τρέχοντος χρώματος σχεδίασης στο χρωματικό μοντέλο RGB το μητρώο *parameters* θα περιέχει τις τιμές των τριών χρωματικών συνιστωσών και θα πρέπει να έχει έχει διάσταση 3. Στην περίπτωση αυτή εισάγουμε ως όρισμα *parameterName* την αριθμητική σταθερά *GL_CURRENT_COLOR* και η εντολή επισκόπησης του τρέχοντος χρώματος συντάσσεται ως εξής:

glGetFloatv(GL_CURRENT_COLOR, parameters);

Επισημαίνουμε ότι σπάνια χρησιμοποιούμε τις παραπάνω εντολές για πρόσβαση στις τρέχουσες τιμές μιας ιδιότητας. Αντ' αυτού, είναι βολικότερη η αποθήκευση μιας **ομάδας ιδιοτήτων**, όπως θα δούμε και στην ενότητα "Ομάδες ιδιοτήτων".

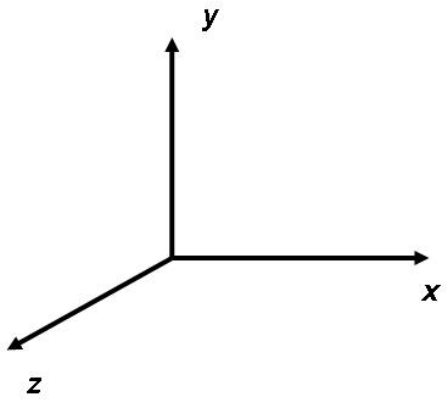
1.5 Σχεδίαση στην OpenGL

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. Επιπλέον το σύστημα συντεταγμένων είναι δεξιόστροφο δηλαδή τα μοναδιαία διανύσματα $\vec{x}\ \vec{y}\ \vec{z}$ συνδέονται με σχέσεις εξωτερικού γινομένου ως εξής:

$$\vec{x} \times \vec{y} = \vec{z}$$
$$\vec{y} \times \vec{z} = \vec{x}$$
$$\vec{z} \times \vec{x} = \vec{y}$$

Κάθε σημείο αναπαρίσταται με τη χρήση **ομογενών συντεταγμένων**. Με τη χρήση ομογενών συντεταγμένων όλα τα σημεία αναπαρίστανται με τέσσερις συντεταγμένες κινητής υποδιαστολής (*x*, *y*, *z*, *w*). Αν η παράμετρος *w* είναι διαφορετική από το 0, τότε αυτές οι συντεταγμένες ανταποκρίνονται στο τρισδιάστατο σημείο (*x/w, y/w, z/w*). Μπορούμε να καθορίσουμε την συντεταγμένη *w* στις εντολές OpenGL, αλλά αυτό γίνεται σπανίως. Αν η συντεταγμένη *w* δεν είναι καθορισμένη, το σύστημα θεωρεί την τιμή 1. (Η χρησιμότητα των ομογενών συντεταγμένων θα αναλυθεί στο κεφάλαιο "Μετασχηματισμοί συντεταγμένων").



Σχ. 1.1: Τρισδιάστατο δεξιόστροφο καρτεσιανό σύστημα συντεταγμένων

1.6 Καθαρισμός Οθόνης

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 τιμη. 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", παίζει το ρόλο "συντελεστή διαφάνειας". Θα αναφερθούμε στη χρήση της συνιστώσας alpha στο κεφάλαιο "Μίξη χρωμάτων".

Το χρώμα καθαρισμού, όντας μεταβλητή κατάστασης, διατηρεί την τελευταία τιμή που του ανατέθηκε.

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), πρεπει 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*.

```
glClear(GL_COLOR_BUFFER_BIT);
```

Στο παρακάτω παράδειγμα καθαρίσουμε τον ενταμιευτή χρωματικών τιμών χρησιμοποιώντας ως χρώμα φόντου το μαύρο:

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

Η πρώτη εντολή ορίζει ως χρώμα καθαρισμού το μαύρο και η δεύτερη εντολή εκτελεί τον καθαρισμό του παραθύρου με το χρώμα αυτό. Συνήθως, ορίζουμε το χρώμα καθαρισμού στην αρχή του προγράμματός μας και μετά καθαρίζουμε τους buffers όσο συχνά χρειάζεται

Η εντολή ***glClear*** επιτρέπει επίσης τον καθορισμό πολλαπλών ενταμιευτών με μία μόνο κλήση της. Στην περίπτωση αυτή δίνουμε ως όρισμα πολλαπλές παραμέτρους διαχωρισμένες με τελεστές OR (|).

Για να καθαρίσουμε λ.χ. τον ενταμιευτή χρωματικών τιμών (color buffer) και τον ενταμιευτή τιμών βάθους

(depth buffer) δίνουμε:

```
glClearColor(0.0, 0.0, 0.0, 0.0);
glClearDepth(0.0);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Η εντολή ***glClearColor()*** είναι ίδια με αυτήν του προηγούμενου παραδείγματος. ενώ εντολή ***glClearDepth()*** καθορίζει την τιμή την οποία θα δώσουμε σε κάθε pixel του ενταμιευτή βάθους (περισσότερα για τη χρήση ενταμιευτών βάθους στην ενότητα "Καταστολή κρυμμένων επιφανειών").

Ο Πίνακας 2 περιέχει τη λίστα των ενταμιευτών που μπορούμε να καθαρίσουμε χρησιμοποιώντας την εντολή ***glClear***.

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

Η OpenGL μας επιτρέπει να καθορίσουμε πολλαπλούς buffers διότι ο καθαρισμός γενικά είναι μία αργή διαδικασία μιας και υπάρχουν πάρα πολλά pixel στο παράθυρο (εκατομμύρια) και ορισμένες κάρτες γραφικών επιτρέπουν τον ταυτόχρονο καθαρισμό πολλαπλών ενταμιευτών. Για κάρτες που δεν υποστηρίζουν αυτή τη λειτουργία ισχύει ο διαδοχικός καθαρισμός ενταμιευτών. Η διαφορά μεταξύ της εντολής

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

και

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

είναι ότι, σε συστήματα υλικού (hardware) που υποστηρίζουν παράλληλο καθαρισμό ενταμιευτών, η πρώτη εκδοχή θα εκτελεστεί ταχύτερα.

1.7 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 glColor3ub(GLubyte red, GLubyte green, GLubyte blue); $(0 \le red, green, blue \le 255)$

void glColor3f(GLfloat red, GLfloat green, GLfloat blue); $(0 \le red, green, blue \le 1)$

void glColor3d(GLdouble red, GLdouble green, GLdouble blue); $(0 \le red, green, blue \le 1)$

Οι παράμετροι red, green και blue αντιστοιχούν στις τιμές των χρωματικών συνιστωσών του χρώματος στο μοντέλο RGB. Π.χ. η εντολή

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

επιστρέφει το πιο έντονο κόκκινο που μπορεί να αποδώσει το σύστημα. Στον πίνακα 3 ορίζονται ορισμένα βασικά χρώματα.

Πίνακας 3. Βασικά χρώματα

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

1.8 Καθορισμός κορυφών

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(),** όπως θα δούμε στη συνέχεια . Παρακάτω βλέπουμε μερικά παραδείγματα όπου χρησιμοποιείται η ***glVertex****:

```
glVertex2s(2,3);  // Δήλωση σημείου με συντεταγμένες (x,y)=(2,3)
glVertex3d(0,0,3.14);  // Δήλωση σημείου με συντεταγμένες (x,y,z)=(0,0,3.14)
glVertex4f(2.3, 1.0, -2.2, 2.0);

GLdouble dvect[3] = {5,9,1992};
glVertex3dv(dvect);  //Δήλωση σημείου που οι τιμές του βρίσκονται στο μητρώο
                     // dvect
```

Το πρώτο παράδειγμα αναπαριστά μία κορυφή με συντεταγμένες σε τρεις διαστάσεις (2, 3, 0). Όπως προαναφέραμε, αν η συντεταγμένη *z* δεν καθορίζεται το σύστημα θεωρεί την τιμή 0. Οι συντεταγμένες (0.0, 0.0, 3.14) στο δεύτερο παράδειγμα είναι αριθμοί κινητής υποδιαστολής διπλής ακρίβειας. Το τρίτο παράδειγμα αναπαριστά μία κορυφή με τρισδιάστατες συντεταγμένες (1.15, 0.5, -1.1) διότι οι συντεταγμένες *x, y*, και *z* διαιρούνται με την συντεταγμένη *w*. Στο τελευταίο παράδειγμα, το *dvect* είναι ένας δείκτης σε ένα πίνακα τριών αριθμών κινητής υποδιαστολής διπλής ακρίβειας.

1.9 Η δομή glBegin(), glEnd()

Ο ορισμός των κορυφών κάθε γεωμετρικού σχήματος περικλείεται μεταξύ δύο εντολών, των ***glBegin()*** και ***glEnd()***:

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

Το είδος του γεωμετρικού σχήματος που σχεδιάζεται, εξαρτάται από την παράμετρο που δίνουμε στο όρισμα *mode* της εντολής ***glBegin***. Έτσι, διακρίνουμε τις εξής περιπτώσεις:

1.9.1 Σημεία

Ένα σημείο αναπαρίσταται από ένα σύνολο αριθμών κινητής υποδιαστολής που ονομάζεται κορυφή. Όλοι οι εσωτερικοί υπολογισμοί γίνονται σαν οι κορυφές να είναι τρισδιάστατες. Κορυφές που καθορίζονται από το χρήστη ως δισδιάστατες (δηλαδή δίνονται μόνο οι τιμές για τις *x* και *y* συνιστώσες) παίρνουν από την OpenGL την τιμή *z=0*.

Η σχεδίαση σημείων επιτελείται δίνοντας ως όρισμα στην ***glBegin( )*** τη σταθερά *GL_POINTS*.

Ιδιότητες σημείων:

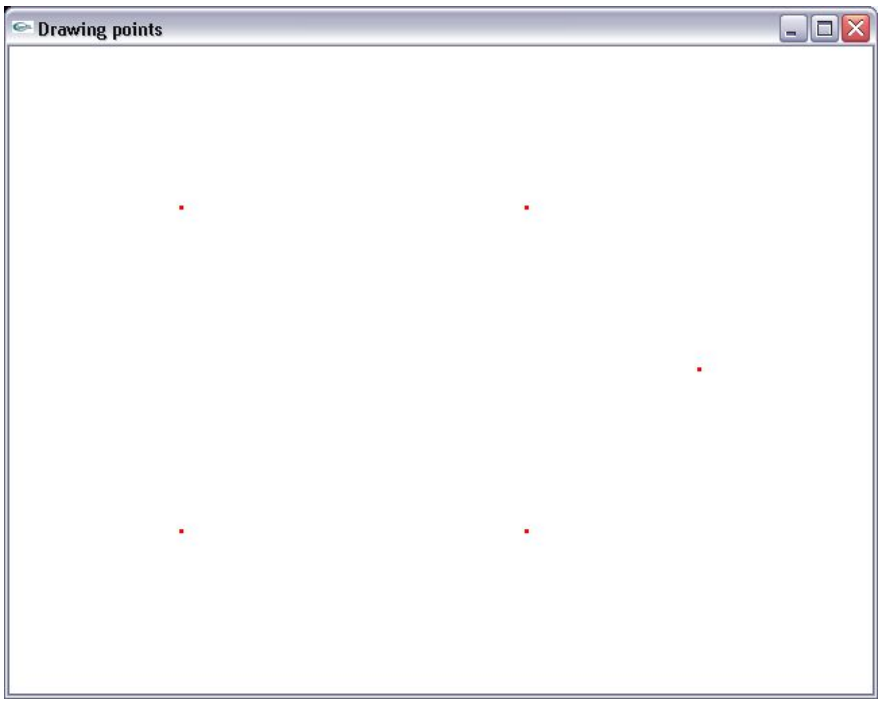
Μία χαρακτηριστική ιδιότητα των σημείων που ο προγραμματιστής μπορεί να μεταβάλλει είναι το πάχος τους. Για τη ρύθμιση του μεγέθους ενός σημείου, χρησιμοποιούμε την εντολή ***glPointSize( )***:

void glPointSize(GLfloat size);

όπου *size* το πλάτος του σημείου σε pixels. Προφανώς, πρέπει να είναι μεγαλύτερο από 0 και η προκαθορισμένη τιμή του είναι 1. Εξ'ορισμού δηλαδή, ένα σημείο απεικονίζεται στην οθόνη σαν ένα pixel.

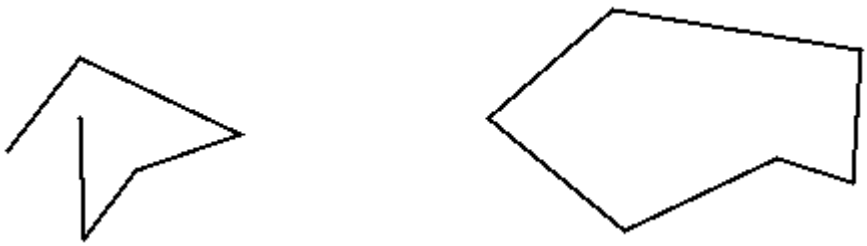
Παράδειγμα: Σχεδίαση σημείων

```
#include <glut.h>

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

	//Set point size to 3 pixels
	glPointSize(3);

	glBegin(GL_POINTS);
	glVertex2i(10,10);
	glVertex2i(20,10);
	glVertex2i(25,15);
	glVertex2i(20,20);
	glVertex2i(10,20);
	glEnd();

	glFlush();
}

int main(int argc, char** argv)
{
	glutInit(&argc,argv);
	glutInitWindowPosition(50,50);
	glutInitWindowSize(640,480);
	glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
	glutCreateWindow("Drawing points");
	glMatrixMode(GL_PROJECTION);
	gluOrtho2D(5,30,5,25);
	glutDisplayFunc(display);
	glutMainLoop();

	return 0;
}
```



1.9.2 Γραμμές

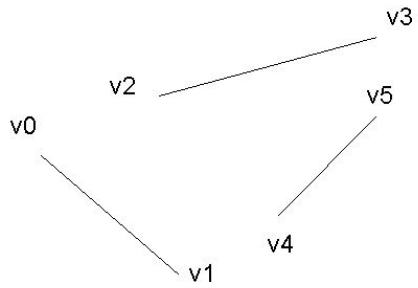
Στην OpenGL, γραμμή σημαίνει ευθύγραμμο τμήμα και όχι η μαθηματική έννοια που εκτείνει στο άπειρο και στις δύο κατευθύνσεις. Υπάρχουν εύκολοι τρόποι για να καθορίσουμε μία σειρά από συνδεμένα ευθύγραμμα τμήματα ή μία κλειστή σειρά από συνδεμένα ευθύγραμμα τμήματα (Σχ. 1.2). Σε όλες τις περιπτώσεις όμως, οι γραμμές που αποτελούν τις συνδεμένες σειρές καθορίζονται με τις συντεταγμένες των άκρων τους.



Σχήμα 1.2. Δύο σειρές από συνδεμένα ευθύγραμμα τμήματα.

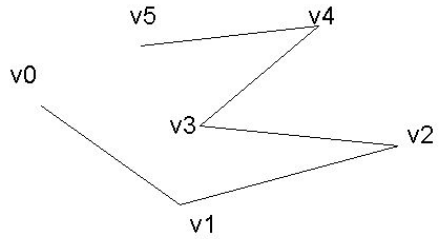
Αναλόγως της σταθεράς που δίνουμε στη glBegin μπορούμε να επιλέξουμε διαφορετικούς τρόπους σχεδίασης γραμμών. Υπάρχουν οι εξής επιλογές:

GL_LINES
Τα δοθέντα σημεία ορίζουν κατά ζεύγη ευθύγραμμα τμήματα. Δηλαδή για n σημεία $v_0, v_1, v_2, ..., v_{n-1}$ σχεδιάζονται τα τμήματα *(v0, v1) (v2,v3)* κοκ . Για περιττό αριθμό σημείων, το τελευταίο σημείο αγνοείται.



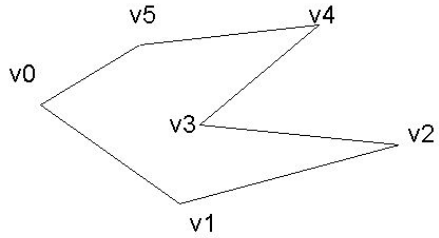
GL_LINE_STRIP

Δίνοντας τα σημεία $(v_0, v_1, ..., v_{n-1})$, σχεδιάζουμε τα διαδοχικά ευθύγραμμα τμήματα $(v_0, v_1), (v_1, v_2)$, …, (v_{n-2}, v_{n-1}).



GL_LINE_LOOP

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



Ιδιότητες γραμμών:

Όσον αφορά τις ιδιότητες των γραμμών, ο προγραμματιστής έχει την ευχέρεια να τροποποιήσει το πάχος και τη διάστιξή τους.

α) Πάχος γραμμών

Το πάχος γραμμής τροποποιείται με τη χρήση της εντολής:

void glLineWidth(GLfloat width);

όπου *width* το πάχος γραμμής σε pixels. Η παράμετρος width πρέπει να είναι μεγαλύτερη από 0 και εξ'ορισμού είναι ίση με 1. Το πάχος γραμμής είναι μεταβλητή κατάστασης, συνεπώς διατηρεί την τιμή που του ανατέθηκε την τελευταία φορά.

β) Διάστικτες γραμμές

Εξ' αρχής, στην OpenGL, οι γραμμές σχεδιάζονται ως συνεχή ευθύγραμμα τμήματα που ενώνουν τα δύο άκρα τους. Ωστόσο υπάρχουν περιπτώσεις που επιθυμούμε να σχεδιάσουμε διακεκομμένες γραμμές διαφόρων μορφών, να χρησιμοποιήσουμε δηλαδή διάστικτες γραμμές. Π.χ. η χρήση διακεκομμένων γραμμών απαιτείται σε εφαρμογές CAD.

Η δυνατότητα χρήσης διάστικτων γραμμών αποτελεί μια μεταβλητή κατάστασης της μηχανής της OpenGL. Αυτό σημαίνει ότι, προκειμένου να αξιοποιήσουμε αυτή τη δυνατότητα, θα πρέπει να το δηλώσουμε ρητά στο πρόγραμμά μας. Η κατάσταση ενεργοποίησης διάστιξης γραμμών ενεργοποιείται δίνοντας την παράμετρο *GL_LINE_STIPPLE* στην εντολή ***glEnable( )***:

glEnable(GL_LINE_STIPPLE);

Κατόπιν, η μορφή των διάστικτων γραμμών καθορίζεται με την εντολή ***glLineStipple( )***:

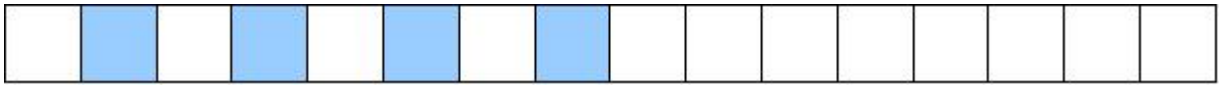
void glLineStipple (GLint factor, GLushort pattern);

Το όρισμα *pattern* καθορίζει το μοτίβο των ευθειών και συνήθως δίνεται δε δεκαεξαδική μορφή. Ο σχηματισμός του μοτίβου γίνεται ως εξής. Αναπαριστούμε την αριθμητική τιμή του ορίσματος pattern σε δυαδική μορφή. Π.χ. η τιμή 0x00AA αναπαρίσταται στο δυαδικό σύτημα ως: 0000 0000 1010 1010

Οι θέσεις των μονάδων καθορίζουν τα pixels της γραμμής που θα σχεδιαστούν, ενώ οι θέσεις των μηδενικών καθορίζουν τα pixels που θα παραμείνουν κενά. Έτσι για την τιμή 0x00AA του μοτίβου (για factor ίσο με 1) δίνοντας την εντολή:

```
glLineStipple(1,0x00AA);
```

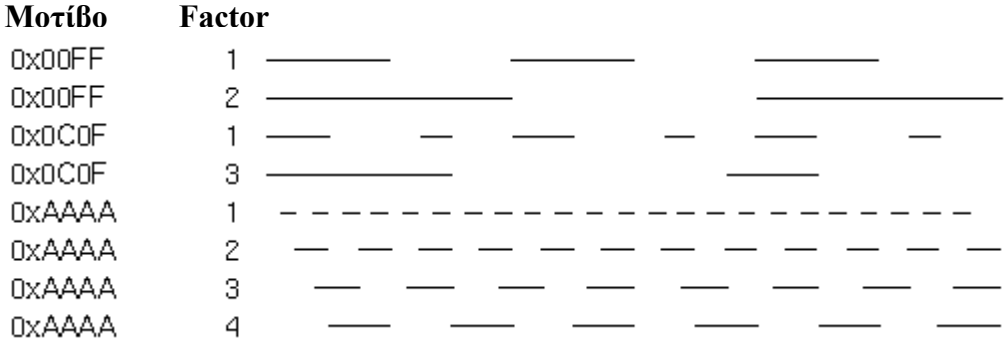
θα σχεδιάζεται το παρακάτω μοτίβο γραμμής



Η έναρξη του μοτίβου γίνεται από το λιγότερο σημαντικό ψηφίο του pattern.

Η ακέραια παράμετρος *factor* κλιμακώνει το μοτίβο, δηλαδή αναπαράγει κάθε δυαδικό ψηφίο του όσες φορές δηλώνει η τιμή της. Οπότε, στο παραπάνω παράδειγμα, για τιμή του factor ίση με δύο, θα δημιουργείται η ακόλουθη διάστιξη:

Το Σχ. 1.3 παρουσιάζει γραμμές σχεδιασμένες με διαφορετικά μοτίβα και επαναλαμβανόμενους factors. Αν δεν ενεργοποιήσουμε τις διάστικτες γραμμές, ο σχεδιασμός φίνεται σαν το μοτίβο να ήταν 0xFFFF και το factor ίσο με 1.



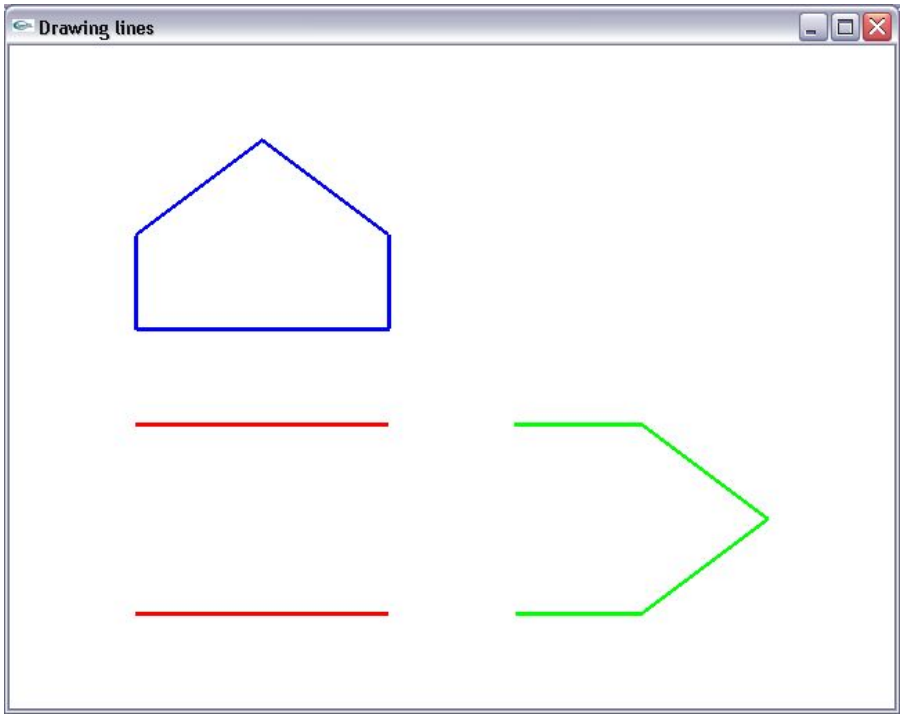
Σχ. 1.3: Παραδείγματα διαστίκτων γραμμών

Επισημαίνουμε η διάστιξη γραμμών, ως παράμετρος κατάστασης, παραμένει ενεργή έως ότου δηλωθεί μεταβολή της από τον προγραμματιστή.

Παράδειγμα: Σχεδίαση γραμμών

```
#include <glut.h>

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

	glLineWidth(3);

	glColor3f(1,0,0);

	glBegin(GL_LINES);
	glVertex2i(0,0);
	glVertex2i(10,0);
	glVertex2i(0,10);
	glVertex2i(10,10);
	glEnd();
```

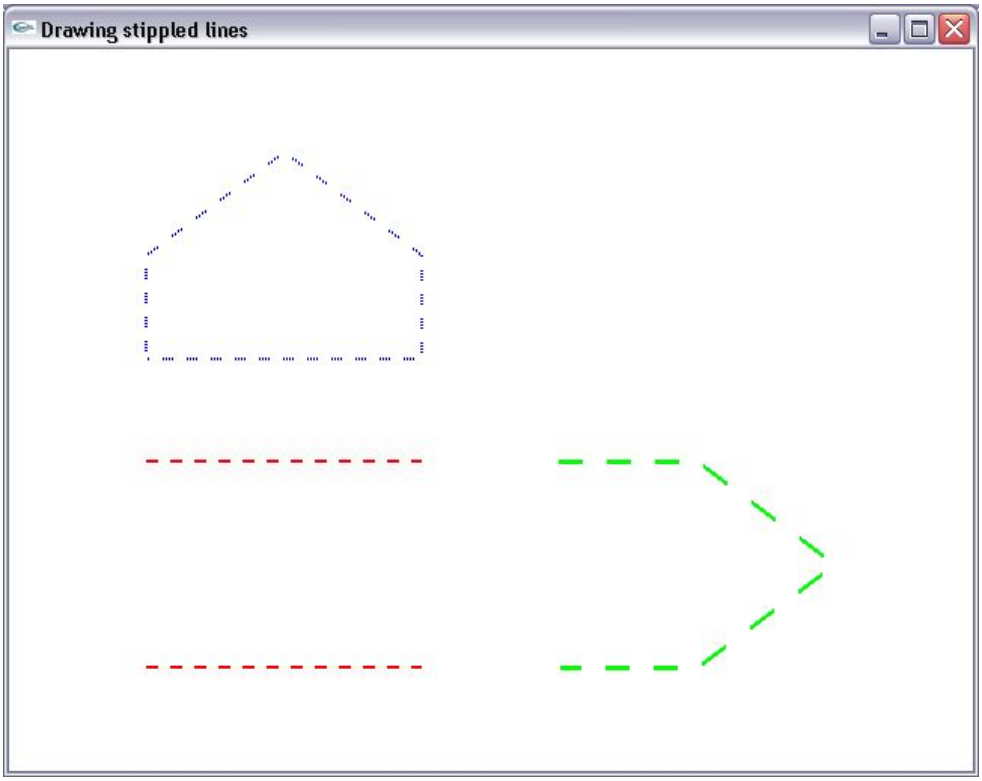
```
        glColor3f(0,1,0);

        glBegin(GL_LINE_STRIP);
        glVertex2i(15,10);
        glVertex2i(20,10);
        glVertex2i(25,5);
        glVertex2i(20,0);
        glVertex2i(15,0);
        glEnd();

        glColor3f(0,0,1);

        glBegin(GL_LINE_LOOP);
        glVertex2i(0,15);
        glVertex2i(0,20);
        glVertex2i(5,25);
        glVertex2i(10,20);
        glVertex2i(10,15);
        glEnd();

        glFlush();
}

int main(int argc, char** argv)
{
        glutInit(&argc,argv);
        glutInitWindowPosition(50,50);
        glutInitWindowSize(640,480);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGBA);
        glutCreateWindow("Drawing lines");

        glMatrixMode(GL_PROJECTION);
        gluOrtho2D(-5,30,-5,30);
        glutDisplayFunc(display);
        glutMainLoop();

        return 0;
}
```



Παράδειγμα: Σχεδίαση διάστικτων γραμμών

```
#include <glut.h>

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


	glColor3f(1,0,0);

	glLineWidth(2);
	glLineStipple(1,0x00FF);

	glBegin(GL_LINES);
	glVertex2i(0,0);
	glVertex2i(10,0);
	glVertex2i(0,10);
	glVertex2i(10,10);
	glEnd();

	glColor3f(0,1,0);

	glLineWidth(3);
	glLineStipple(2,0x00FF);

	glBegin(GL_LINE_STRIP);
	glVertex2i(15,10);
	glVertex2i(20,10);
	glVertex2i(25,5);
	glVertex2i(20,0);
	glVertex2i(15,0);
	glEnd();

	glColor3f(0,0,1);


	glLineWidth(2);
	glLineStipple(1,0x0A0A);

	glBegin(GL_LINE_LOOP);
	glVertex2i(0,15);
	glVertex2i(0,20);
	glVertex2i(5,25);
	glVertex2i(10,20);
	glVertex2i(10,15);
	glEnd();

	glFlush();
}

int main(int argc, char** argv)
{
	glutInit(&argc,argv);
	glutInitWindowPosition(50,50);
	glutInitWindowSize(640,480);
	glutInitDisplayMode(GLUT_SINGLE|GLUT_RGBA);
	glutCreateWindow("Drawing stippled lines");

	glMatrixMode(GL_PROJECTION);
```

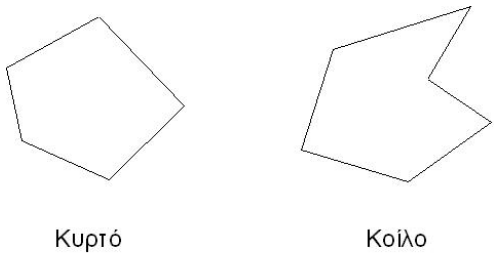
```
        gluOrtho2D(-5,30,-5,30);
        glEnable(GL_LINE_STIPPLE);

        glutDisplayFunc(display);
        glutMainLoop();

        return 0;
}
```



1.9.3 Πολύγωνα

Τα πολύγωνα είναι περιοχές που περικλείονται από απλούς κλειστούς βρόχους ευθύγραμμων τμημάτων, όπου τα ευθύγραμμα τμήματα καθορίζονται από τις κορυφές στα άκρα τους. Τα πολύγωνα σχεδιάζονται συμπαγή (με χρωματισμένα τα pixels στο εσωτερικό τους), ωστόσο έχουμε τη δυνατότητα να σχεδιάσουμε απλώς τα περιγράμματα ή τις κορυφές τους .

Τα πολύγωνα διακρίνονται σε δύο κατηγορίες: τα κυρτά και τα κοίλα. Στα κυρτά πολύγωνα όλες οι εσωτερικές γωνίες είναι μικρότερες των 180^0 ενώ ένα πολύγωνο είναι κοίλο όταν περιέχει τουλάχιστον μια εσωτερική γωνία μεγαλύτερη των 180^0 (Σχ. 1.4).



Σχ 1.4: Παραδείγματα κυρτού και κοίλου πολυγώνου

Γενικά τα πολύγωνα μπορεί να είναι πολύπλοκα και για το λόγο αυτό η OpenGL έχει θέσει τους εξής περιορισμούς στις ρουτίνες σχεδίασής τους.

α) Οι ρουτίνες σχεδιάζουν **κυρτά** πολύγωνα.

Αυτό συμβαίνει διότι οι ρουτίνες πλήρωσης του εσωτερικού ενός κοίλου πολυγώνου παρουσιάζουν αυξημένη πολυπλοκότητα. Για το λόγο αυτό οι βιβλιοθήκες γραφικών θέτουν τον παραπάνω περιορισμό.

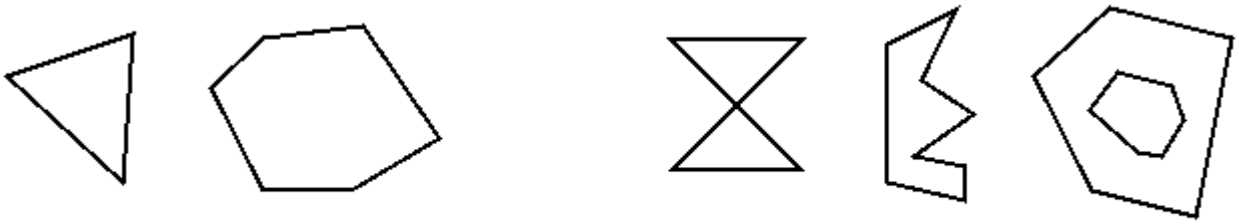
β) Οι πλευρές των πολυγώνων δεν μπορούν να τέμνονται .

Αυτός ο περιορισμός είναι άμεση απόρροια του πρώτου, καθώς, πολύγωνα με τεμνόμενες πλευρές είναι κοίλα.

γ) Οι ρουτίνες δε μπορούν να σχεδιάσουν πολύγωνα με οπές.

Πολύγωνα με οπές είναι κοίλα και δεν μπορούν να σχεδιαστούν με ένα όριο φτιαγμένο από ένα απλό κλειστό βρόχο. Σε αυτές τις περιπτώσεις, η σχεδίαση πολυγώνων με οπές είναι εφικτή με τον διαδοχικό ορισμό δύο κυρτών πολυγώνων.

Με βάση τους παραπάνω κανόνες, η OpenGL διαχωρίζει τα πολυγώνα σε “έγκυρα” και σε “μη έγκυρα”. Κριτήριο του διαχωρισμού αυτού είναι η αντιστοιχία του επιθυμητού με το τελικό αποτέλεσμα. Αυτό σημαίνει ότι, εάν δώσουμε εντολή σχεδίασης ενός “μη έγκυρου πολυγώνου”, το αποτέλεσμα θα είναι απρόβλεπτο. Στο Σχ. 1.5 παρουσιάζονται ορισμένα παραδείγματα.



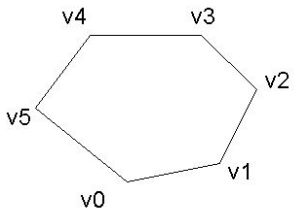
Έγκυρα Μη έγκυρα

Σχήμα 1.5: Έγκυρα και μη έγκυρα πολύγωνα.

Η δήλωση των κορυφών ενός ή περισσοτέρων πολυγώνων γίνεται μεταξύ των εντολών ***glBegin()*** και ***glEnd().*** Ανάλογα με το όρισμα της εντολής ***glBegin()*** υπάρχουν οι εξής διαφοροποιήσεις ως προς το σχηματισμό των πολυγώνων.

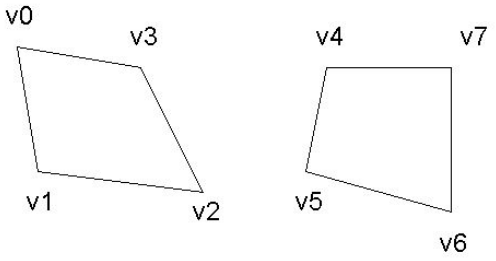
GL_POLYGON

Σχεδιάζει **ένα** πολύγωνο χρησιμοποιώντας τα σημεία $v_0, v_1, ..., v_{n-1}$ ως κορυφές. Το *n* πρέπει να είναι τουλάχιστον ίσο με 3, ειδάλλως δεν ορίζεται πολυγωνική επιφάνεια. Επιπλέον, το καθορισμένο πολύγωνο πρέπει να είναι κυρτό και να μην έχει τεμνόμενες πλευρές. Αν το πολύγωνο δεν πληρεί αυτές τις προϋποθέσεις, το αποτέλεσμα της σχεδίασής του θα είναι απρόβλεπτο.



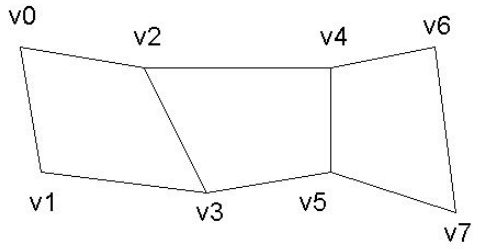
GL_QUADS

Σχεδιάζει διακεκριμένα τετράπλευρα. Για διατεταγμένο σύνολο κορυφών $v_0, v_1, ..., v_{n-1}$ και *n* πολλαπλάσιο του 4, τα τετράπλευρα καθορίζονται από τις τεράδες κορυφών $(v_0, v_1, v_2, v_3), (v_4, v_5, v_6, v_7)$ και ούτω καθεξής. Αν το *n* δεν είναι πολλαπλάσιο του 4, τότε η μία ή οι δύο ή οι τρεις τελευταίες κορυφές παραλείπονται.



GL_QUAD_STRIP

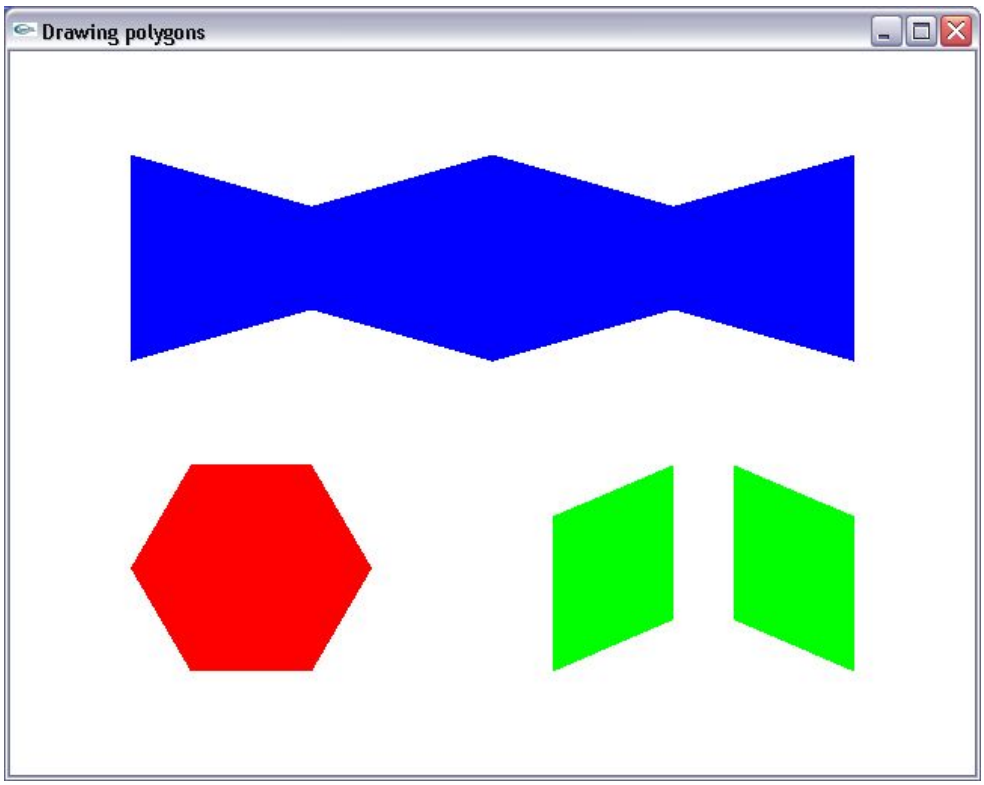
Σχεδιάζεται αλληλουχία τετραπλεύρων με μία κοινή πλευρά. Για διατεταγμένο σύνολο κορυφών $v_0, v_1, ..., v_{n-1}$ και n άρτιο, σχεδιάζονται τα τετράπλευρά με κορυφές από τα (v_0, v_1, v_3, v_2), συνεχίζοντας με (v_2, v_3, v_5, v_4), τα (v_4, v_5, v_7, v_6) και ούτω καθεξής. Το *n* πρέπει να είναι τουλάχιστον ίσο με 4 πριν σχεδιαστεί τετράπλευρο. Για *n* περιττό αριθμό, η τελευταία κορυφή παραλείπεται.



<u>Παράδειγμα:</u> Σχεδίαση πολυγώνων

```
#include <glut.h>

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

	glColor3f(1,0,0);

	glBegin(GL_POLYGON);

	glVertex2f(2.5,0);
	glVertex2f(7.5,0);
	glVertex2f(10,5);
	glVertex2f(7.5,10);
	glVertex2f(2.5,10);
	glVertex2f(0,5);

	glEnd();

	glColor3f(0,1,0);

	glBegin(GL_QUADS);

	glVertex2f(17.5,0);
	glVertex2f(17.5,7.5);
	glVertex2f(22.5,10);
	glVertex2f(22.5,2.5);

	glVertex2f(25,2.5);
	glVertex2f(30,0);
	glVertex2f(30,7.5);
	glVertex2f(25,10);

	glEnd();
```

```
	glColor3f(0,0,1);
	glBegin(GL_QUAD_STRIP);

	glVertex2f(0,15);
	glVertex2f(0,25);

	glVertex2f(7.5,17.5);
	glVertex2f(7.5,22.5);

	glVertex2f(15,15);
	glVertex2f(15,25);

	glVertex2f(22.5,17.5);
	glVertex2f(22.5,22.5);

	glVertex2f(30,15);
	glVertex2f(30,25);

	glEnd();

	glFlush();
}

int main(int argc, char** argv)
{
	glutInit(&argc,argv);
	glutInitWindowPosition(50,50);
	glutInitWindowSize(640,480);
	glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
	glutCreateWindow("Drawing polygons");

	glMatrixMode(GL_PROJECTION);
	gluOrtho2D(-5,35,-5,30);
	glutDisplayFunc(display);
	glutMainLoop();

	return 0;
}
```



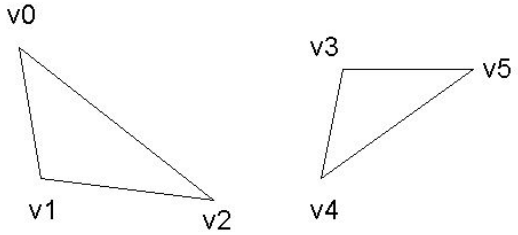
1.9.4 Τρίγωνα

Ειδικές περιπτώσεις των πολυγώνων αποτελούν τα τρίγωνα (n=3) για το οποία η OpenGL προσφέρει επιπρόσθετες παραμέτρους. Ως ειδικές περιπτώσεις πολυγώνων τα τρίγωνα διέπονται από τους ίδιους κανόνες απόδοσης επιφανειών που ισχύουν και για τα πολύγωνα. Αυτό όπως θα δούμε στην ενότητα «Όψεις πολυγώνων»

Ανάλογα με το όρισμα της εντολής ***glBegin***, διακρίνουμε τις εξής διαφοροποιήσεις στη σχεδίαση των τριγώνων.

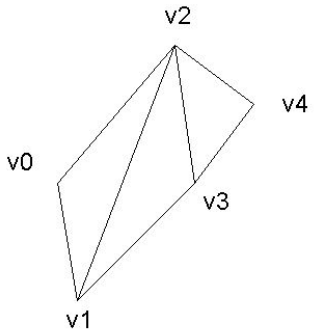
GL_TRIANGLES

Σχεδιάζει μια σειρά από σαφώς διακεκριμένα τρίγωνα. Για διατεταγμένο σύνολο κορυφών $v_0, v_1, v_2, ..., v_{n-1}$ σχεδιάζονται τα τρίγωνα με κορυφές (v_0, v_1, v_2), (v_3, v_4, v_5) και ούτω καθεξής. Αν το n δεν είναι ακέραιο πολλαπλάσιο του 3, η τελευταία ή οι δύο τελευταίες κορυφές παραλείπονται.



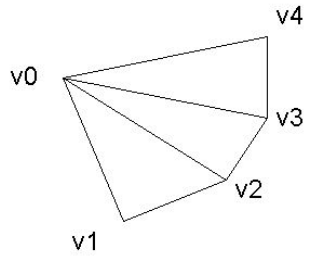
GL_TRIANGLE_STRIP

Σχεδιάζει μια αλληλουχία τριγώνων. Διαδοχικά τρίγωνα έχουν μία κοινή πλευρά. Δίνοντας ένα διατεταγμένο σύνολο κορυφών $(v_0, v_1, ..., v_{n-1})$ σχεδιάζονται τα τρίγωνα με κορυφές (v_0, v_1, v_2), (v_2, v_1, v_3), (v_2, v_3, v_4) και ούτω καθεξής. Η συγκεκριμένη σειρά εμφάνισης των κορυφών εξασφαλίζει ότι οι δήλωσεις των κορυφών των τριγώνων έχουν τον ίδιο προσανατολισμό. (αριστερόστροφο ή δεξιόστροφο) και παίζει ρόλο στον προσανατολισμό της επιφανείας των πολυγώνων, όπως θα δούμε στην ενότητα "Όψεις πολυγώνων". Το *n* πρέπει να είναι τουλάχιστο ίσο με 3, ειδάλλως δε θα σχεδιαστεί τίποτα.



GL_TRIANGLE_FAN

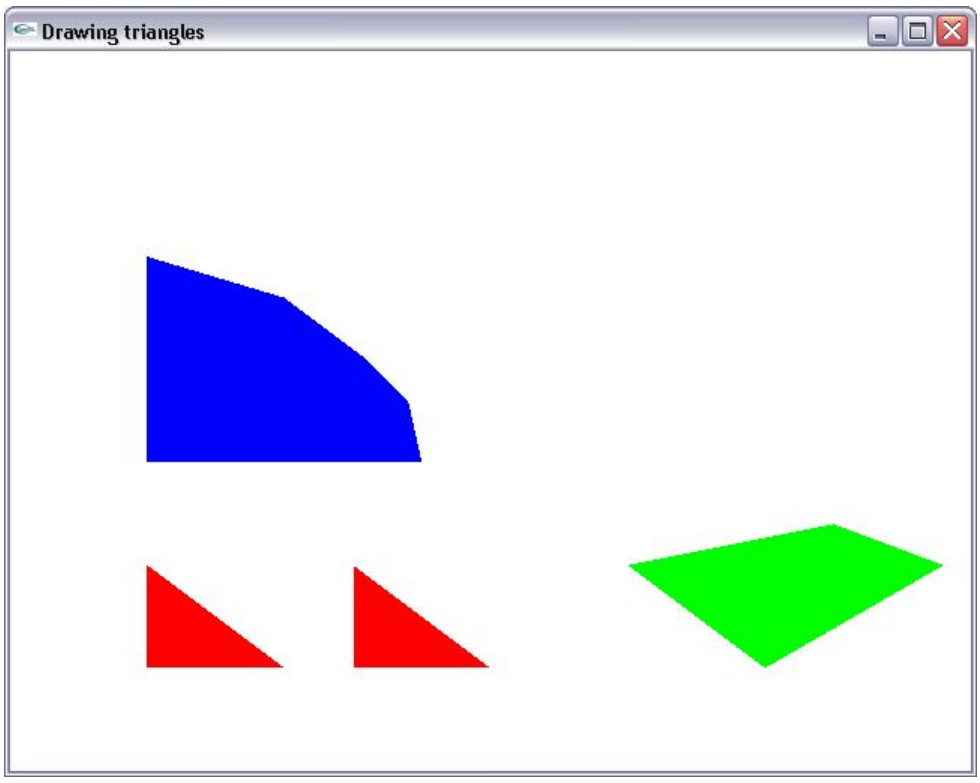
Σχεδιάζει τρίγωνα που έχουν το πρώτο σημείο κοινό και διαδοχικά τρίγωνα έχουν μία κοινή πλευρά. Επομένως τα τρίγωνα σχηματίζονται από τις κορυφές (v_0, v_1, v_2), (v_0, v_2, v_3), (v_0, v_3, v_4) και ούτω καθεξής.



<u>Παράδειγμα:</u> Σχεδίαση τριγώνων

```
#include <glut.h>

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

	glColor3f(1,0,0);

	glBegin(GL_TRIANGLES);
```

```
        glVertex2f(0,0);
        glVertex2f(5,0);
        glVertex2f(0,5);
        glVertex2f(7.5,0);
        glVertex2f(12.5,0);
        glVertex2f(7.5,5);

        glEnd();

        glColor3f(0,1,0);

        glBegin(GL_TRIANGLE_STRIP);

        glVertex2f(17.5,5);
        glVertex2f(22.5,0);
        glVertex2f(22.5,5);
        glVertex2f(25,5);

        glEnd();

        glColor3f(0,0,1);

        glBegin(GL_TRIANGLE_FAN);

        glVertex2f(0,10);
        glVertex2f(10,10);
        glVertex2f(9.5,13);
        glVertex2f(8,15);
        glVertex2f(5,18);
        glVertex2f(0,20);

        glEnd();

        glFlush();
}

int main(int argc, char** argv)
{
        glutInit(&argc,argv);
        glutInitWindowPosition(50,50);
        glutInitWindowSize(640,480);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutCreateWindow("Drawing triangles");

        glMatrixMode(GL_PROJECTION);
        gluOrtho2D(-5,30,-5,30);
        glutDisplayFunc(display);
        glutMainLoop();

        return 0;
}
```



1.9.5 Ορθογώνια

Εφόσον τα ορθογώνια χρησιμοποιούνται ευρέως στις εφαρμογές γραφικών, η OpenGL παρέχει τα μέσα για το σχεδιασμό ενός στοιχειώδους ορθογωνίου χρησιμοποιώντας την εντολή ***glRect*( )***. Μπορούμε να σχεδιάσουμε το ορθογώνιο σαν ένα πολύγωνο όπως είδαμε και πιο πριν, ωστόσο μπορεί να σχεδιαστεί και με την εντολή:

void glRect{sifd}(TYPE x1, TYPE y1, TYPE x2, TYPE y2);

η οποία σχεδιάζει ένα ορθογώνιο που ορίζεται από τα γωνιακά σημεία (*x1, y1*) και (*x2, y2*). Το ορθογώνιο βρίσκεται στο επίπεδο *z*=0 και έχει τις πλευρές του παράλληλες στους άξονες *x* και *y*. Αν χρησιμοποιείται η συνάρτηση με τον τύπο διανύσματος

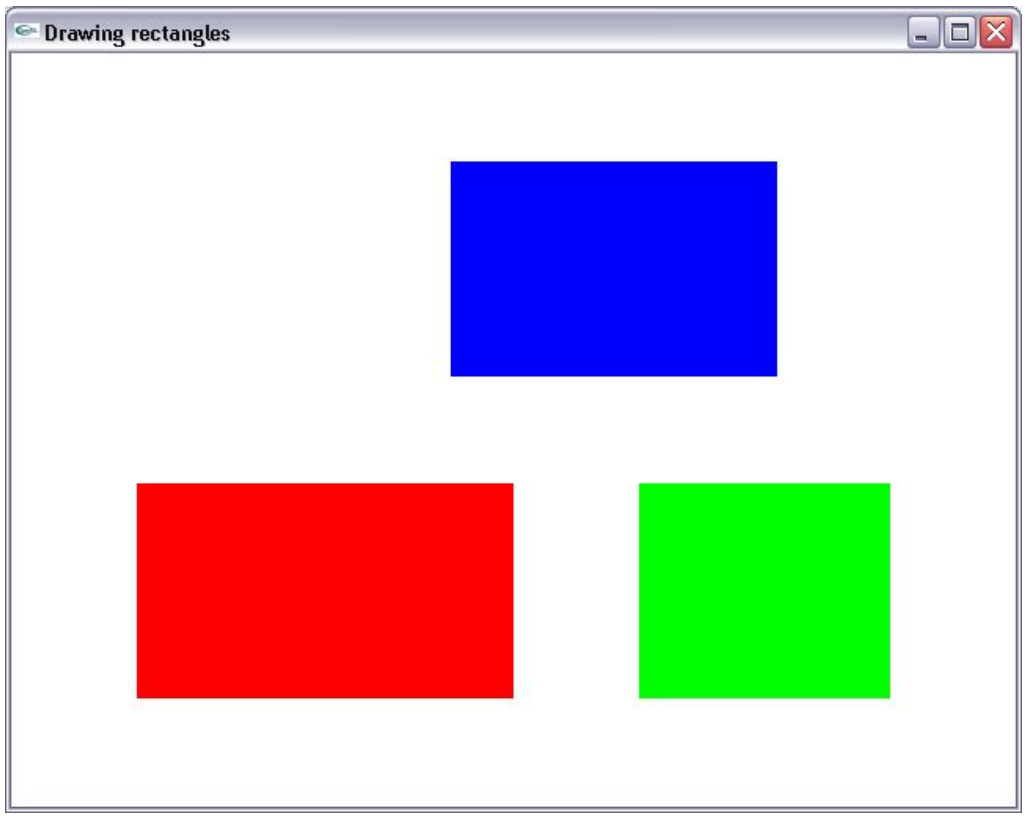
void glRect{sifd}v(TYPE *v1, TYPE *v2);

τότε οι γωνίες δίνονται από δύο δείκτες σε πίνακες, καθένας από τους οποίους περιέχει ένα ζεύγος συντεταγμένων (x, y).

Παρόλο που τα ορθογώνια αρχικοποιούνται με ένα συγκεκριμένο προσανατολισμό στον τρισδιάστατο χώρο, μπορούμε να τον μεταβάλλουμε, εφαρμόζοντας περιστροφές ή άλλους μετασχηματισμούς, τους οποίους θα αναλύσουμε στο Κεφάλαιο "Μετασχηματισμοί συντεταγμένων".

Παράδειγμα: Σχεδιάση ορθογωνίων

```
#include <glut.h>

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

	glColor3f(1,0,0);
	glRectf(0,0,15,10);

	glColor3f(0,1,0);
	glRectf(20,0,30,10);

	glColor3f(0,0,1);
	glRectf(12.5,15,25.5,25);

	glFlush();
}

int main(int argc, char** argv)
{
	glutInit(&argc,argv);
	glutInitWindowPosition(50,50);
	glutInitWindowSize(640,480);
	glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
	glutCreateWindow("Drawing rectangles");

	glMatrixMode(GL_PROJECTION);
	gluOrtho2D(-5,35,-5,30);
	glutDisplayFunc(display);
	glutMainLoop();

	return 0;
}
```



1.9.6 Καμπύλες

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 (Σχ. 1.6).

Σχήμα 1.6. Προσεγγίζοντας καμπύλες

Παρόλο που οι καμπύλες δεν είναι στοιχειώδεις γεωμετρικοί σχηματισμοί, οι βιβλιοθήκες GLU και GLUT προσφέρουν εντολές για τη σχεδιασή τους, όπως θα δούμε στο Κεφάλαιο "Προχωρημένα θέματα τρισδιάστατης σχεδίασης".

α) Σχεδίαση κύκλων:

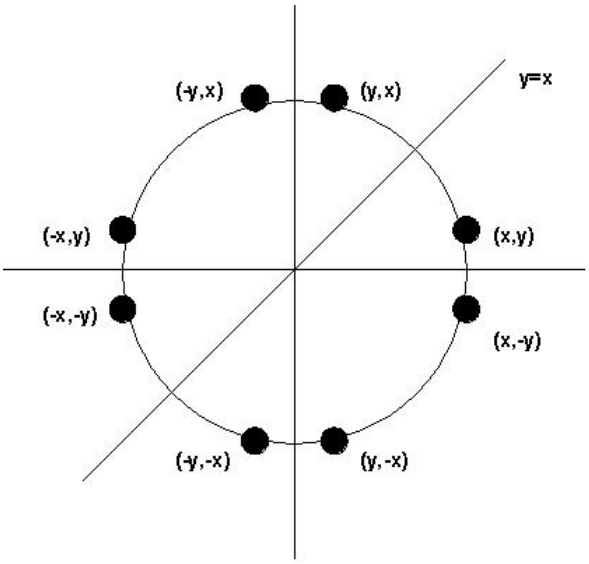
Η σχεδίαση κυκλικών σχημάτων επιτυγχάνεται χρησιμοποιώντας την παραμετρική εξίσωση κύκλου σε πολικές συντεταγμένες. Στην περίπτωση αυτή, κάθε συντεταγμένη της περιφέρειας ενός κύκλου προκύπτει από τις εξισώσεις

$$\begin{aligned} x &= x_c + r \cdot \cos\theta \\ y &= y_c + r \cdot \sin\theta \end{aligned} \qquad 0 \le \theta \le 2\pi$$

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

Θεωρώντας λοιπόν ένα στοιχειώδες γωνιακό βήμα $d\theta$, το οποίο εξασφαλίζει μια επαρκή δειγματοληψία της περιφέρειας του κύκλου, μπορούμε να σχεδιάσουμε κυκλικά σχήματα, συνενώνοντας τα σημεία με στοιχειώδη ευθύγραμμα τμήματα.

Για τον υπολογισμό των συντεταγμένων κάθε σημείου του κύκλου απαιτείται ο υπολογισμός δύο τριγωνομετρικών αριθμών, κάτι που εισάγει υπολογιστικό κόστος. Ένας τρόπος μείωσης του φορτίου αξιοποιεί τις συμμετρίες του κύκλου. Χάρη στις συμμετρίες αυτές, μπορούμε να περιορίσουμε τη χρήση των παραμετρικών εξισώσεων του κύκλου στο ένα όγδοο του και να αναπαραγάγουμε τα υπόλοιπα σημεία με τις σχέσεις συμμετρίας, όπως φαίνεται στο Σχ. 1.7.



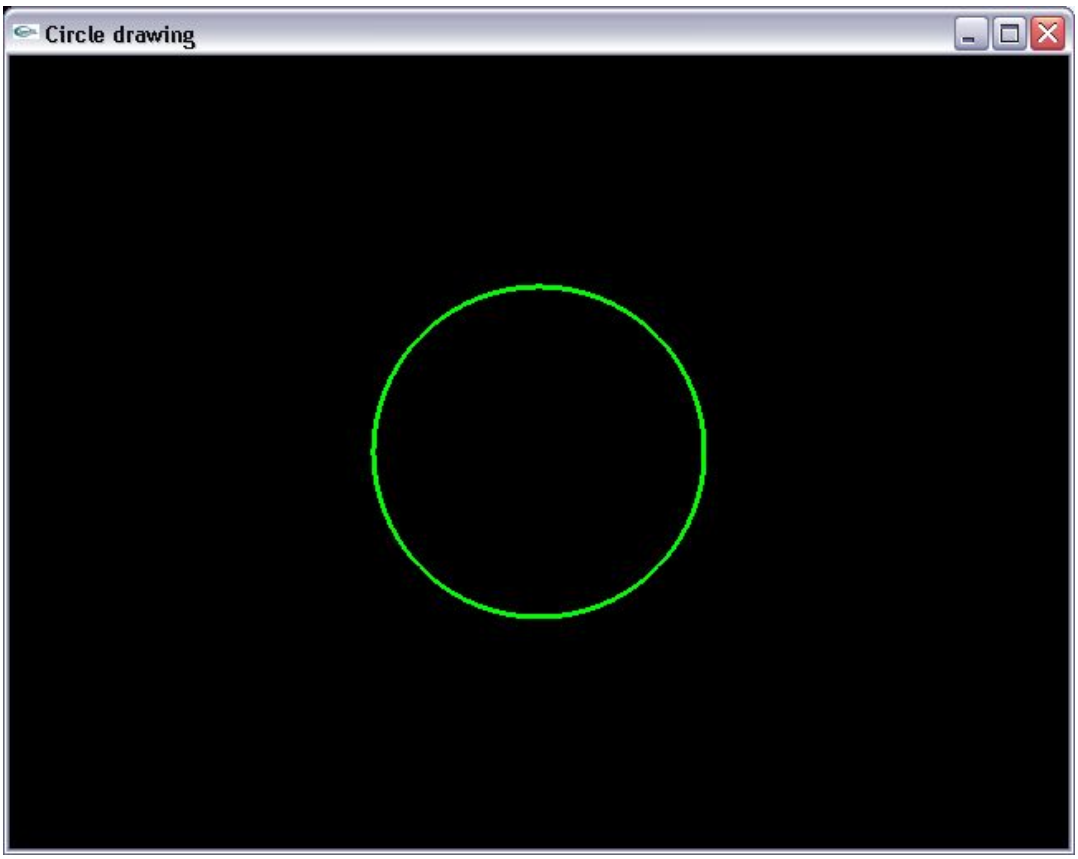
Σχ. 1.7: Συμμετρίες στις συνταταγμένες σημείων κύκλου

Παράδειγμα: Σχεδίαση κύκλου

```
#include <glut.h>
#include <math.h>

#define PI 3.14159
#define     circlePoints 256
int i;
```

```
void display()
{

	GLfloat angleStep=2*PI/(float)circlePoints;
	GLuint pointsPerQuarter=circlePoints/4;

	GLfloat x[circlePoints];
	GLfloat y[circlePoints];

	GLfloat radius=10;



	glClearColor(0,0,0,0);
	glClear(GL_COLOR_BUFFER_BIT);
	glColor3f(0,1,0);

	glLineWidth(3);

	for(i=0;i<pointsPerQuarter/2;i++)
	{
		//Define points in first quadrant
		x[i]=radius*cos(i*angleStep);
		y[i]=radius*sin(i*angleStep);

		x[pointsPerQuarter-1-i]=y[i];
		y[pointsPerQuarter-1-i]=x[i];

		//Define points in second quadrant
		x[pointsPerQuarter+i]=-y[i];
		y[pointsPerQuarter+i]=x[i];

		x[2*pointsPerQuarter-1-i]=-x[i];
		y[2*pointsPerQuarter-1-i]=y[i];

		//Define points in third quadrant
		x[2*pointsPerQuarter+i]=-x[i];
		y[2*pointsPerQuarter+i]=-y[i];

		x[3*pointsPerQuarter-1-i]=-y[i];
		y[3*pointsPerQuarter-1-i]=-x[i];

		//Define points in fourth quadrant
		x[3*pointsPerQuarter+i]=y[i];
		y[3*pointsPerQuarter+i]=-x[i];

		x[4*pointsPerQuarter-1-i]=x[i];
		y[4*pointsPerQuarter-1-i]=-y[i];
	}

	glBegin(GL_LINE_LOOP);



	for (i=0;i<circlePoints;i++)
	{
		glVertex2f(x[i],y[i]);
	}
	glEnd();

	glFlush();
}

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

```
{

	glutInit(&argc,argv);
	glutInitWindowPosition(50,50);
	glutInitWindowSize(640,480);
	glutInitDisplayMode(GLUT_SINGLE|GLUT_RGBA);
	glutCreateWindow("Circle drawing");

	glMatrixMode(GL_PROJECTION);
	gluOrtho2D(-32,32,-24,24);
	glutDisplayFunc(display);
	glutMainLoop();

	return 0;
}
```



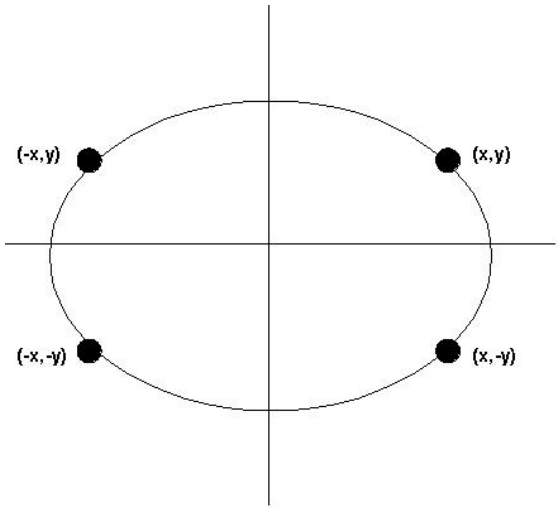
β) Σχεδίαση ελλείψεων:

Οι συντατεγμένες των σημείων της περιφέρειας μιας έλλειψης χαρακτηρίζονται από τις παραμετρικές εξισώσεις:

$$\begin{aligned} x &= x_c + r_x \cdot \cos\theta \\ y &= y_c + r_y \cdot \sin\theta \end{aligned} \qquad 0 \le \theta \le 2\pi$$

όπου r_x και r_y οι "ακτίνες" της έλλειψης στους δύο άξονές της (στο μικρό και μεγάλο άξονά της) Όπως και στην περίπτωση του κύκλου, μπορούμε να αξιοποιήσουμε συμμετρίες της έλλειψης για την αποδοτικότερη - από πλευράς ταχύτητας - δήλωση των σημείων της. Στην περίπτωση ελλείψεων, αρκεί η

δήλωση των συντεταγμένων του ενός τεταρτημορίου και η αναπαραγωγή τους σύμφωνα με τους κανόνες συμμετρίας που δίνονται στο Σχ. 1.8.

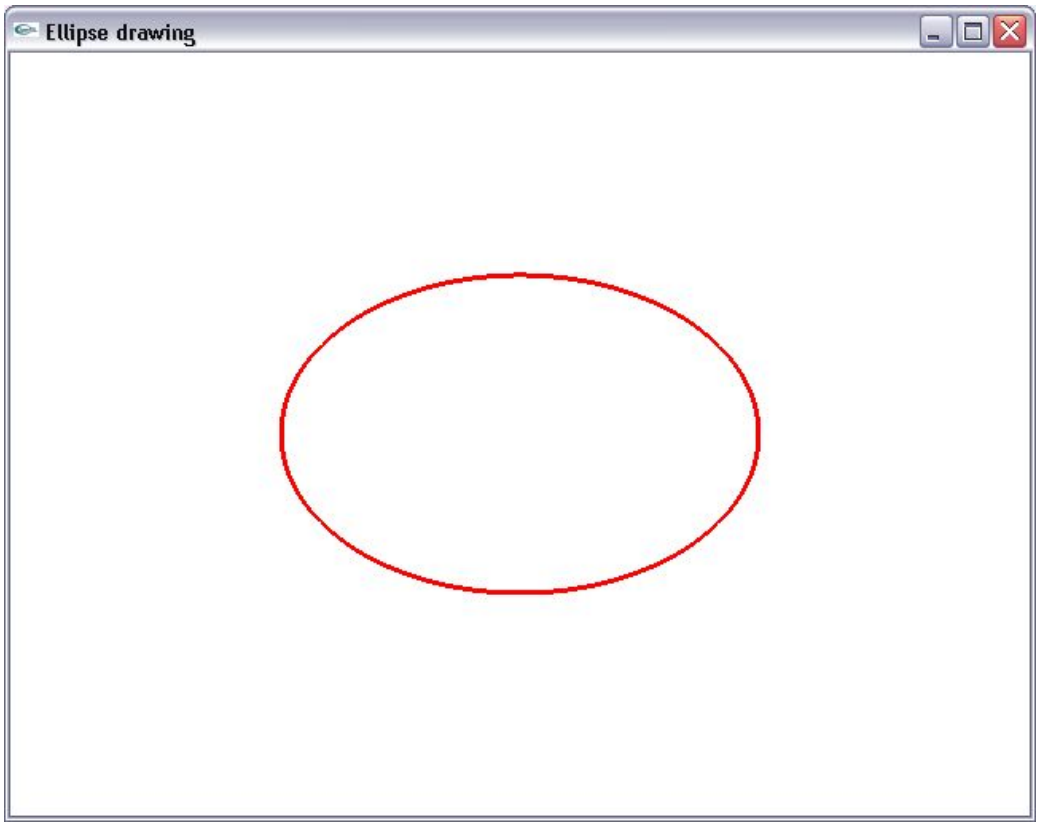


Σχ. 1.8: Συμμετρίες σε ελλείψεις

Παράδειγμα: Σχεδίαση έλλειψης

```
#include <glut.h>
#include <math.h>

#define PI 3.14159
#define ellipsePoints 256
int i;

void display()
{

	GLfloat angleStep=2*PI/(float)ellipsePoints;
	GLuint pointsPerQuarter=ellipsePoints/4;

	GLfloat x[ellipsePoints];
	GLfloat y[ellipsePoints];

	GLfloat rx=15;
	GLfloat ry=10;

	glClearColor(1,1,1,0);
	glClear(GL_COLOR_BUFFER_BIT);
	glColor3f(1,0,0);

	glLineWidth(3);

	glBegin(GL_LINE_LOOP);

	for(i=0;i<pointsPerQuarter;i++)
	{
		x[i]=rx*cos(i*angleStep);
		y[i]=ry*sin(i*angleStep);
	}

	for(i=0;i<pointsPerQuarter;i++)
	{
```

```
            x[pointsPerQuarter+i]=-x[(pointsPerQuarter-1)-i];
            y[pointsPerQuarter+i]=y[(pointsPerQuarter-1)-i];

            x[2*pointsPerQuarter+i]=-x[i];
            y[2*pointsPerQuarter+i]=-y[i];

            x[3*pointsPerQuarter+i]=x[(pointsPerQuarter-1)-i];
            y[3*pointsPerQuarter+i]=-y[(pointsPerQuarter-1)-i];
      }

      for (i=0;i<ellipsePoints;i++)
      {
            glVertex2f(x[i],y[i]);
      }
      glEnd();

      glFlush();
}

int main(int argc, char** argv)
{

      glutInit(&argc,argv);
      glutInitWindowPosition(50,50);
      glutInitWindowSize(640,480);
      glutInitDisplayMode(GLUT_SINGLE|GLUT_RGBA);
      glutCreateWindow("Ellipse drawing");

      glMatrixMode(GL_PROJECTION);
      gluOrtho2D(-32,32,-24,24);
      glutDisplayFunc(display);
      glutMainLoop();

      return 0;
}
```



1.10. Εξαναγκάζοντας την ολοκλήρωση σχεδιασμού

Σε ένα υπολογιστικό σύστημα οι εντολές εκτελούνται κατά ομάδες. Όταν ολοκληρωθεί η συλλογή ενός πλήθους εντολών (όταν γεμίσει ένας "ενταμιευτής εντολών"), αποστέλλονται όλες μαζί προς εκτέλεση. Η εκτέλεση εντολών μία προς μία θα ήταν ασύμφορη από πλευράς απόδοσης. Τη λογική αυτή υιοθετούν επίσης κατανεμημένα συστήματα στα οποία η αποστολή μίας και μόνο εντολής ανά πακέτο μηνύματος θα εισήγαγε υπερβολικό πλεονασμό στο δίκτυο.

Ωστόσο πριν από τη σχεδιάση ενός καρέ, ο προγραμματιστής θα πρέπει να είναι βέβαιος ότι, όσες εντολές έχουν δηλωθεί, προωθούνται προς εκτέλεση. Ένας τρόπος για να προωθήσουμε την εκτέλεση εντολών που εκκρεμούν, είναι η χρήση της εντολής ***glFlush( )***.

void glFlush( );

η οποία εξαναγκάζει την εκτέλεση όλων των δηλωμένων εντολών, ανεξαρτήτως του αν οι ενταμιευτές που τις περιέχουν είναι πλήρεις ή όχι.

Η εντολή ***glFlush( )*** πρέπει να χρησιμοποιείται κάθε φορά που ολοκληρώνουμε την περιγραφή του σκηνικού και προτού προβούμε στον πλήρη σχεδιασμό του. Την τοποθετούμε δηλαδή στο τέλος της συνάρτησης που εμπεριέχει τις εντολές σχεδίασης.

1.11. Σύνοψη

Ο Πίνακας 4 δείχνει τις παραμέτρους που δέχεται ως ορίσματα η ***glBegin( )***:

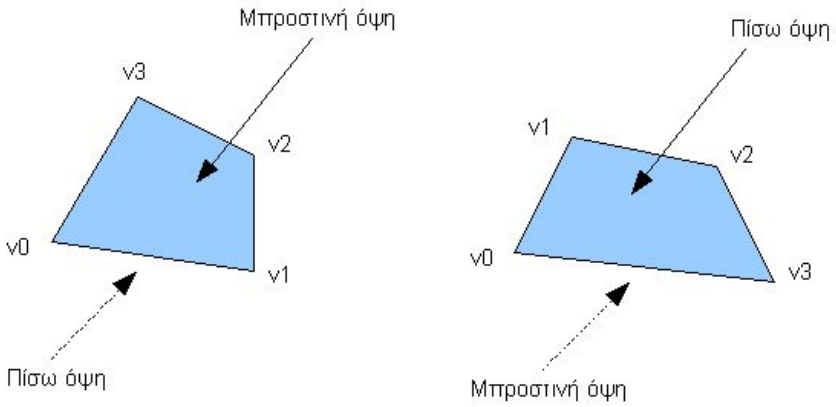
Παράμετρος	**Σημασία**
GL_POINTS	Ανεξάρτητα σημεία.
GL_LINES	Ζεύγη κορυφών που ερμηνεύονται ως ξεχωριστά ευθύγραμμα τμήματα.
GL_POLYGON	Όριο ενός απλού κυρτού πολυγώνου.
GL_TRIANGLES	Τριάδες κορυφών που ερμηνεύονται ως τρίγωνα.
GL_QUADS	Τετράδες κορυφών που ερμηνεύονται ως πολύγωνα τεσσάρων πλευρών.
GL_LINE_STRIP	Σειρές από ενωμένα ευθύγραμμα τμήματα.
GL_LINE_LOOP	Το ίδιο με το παραπάνω με τη διαφορά ότι προστίθεται ένα ευθύγραμμο τμήμα μεταξύ της τελευταίας και της πρώτης κορυφής.
GL_TRIANGLE_STRIP	Συνδεμένα τριγώνα σε σειρά.
GL_TRIANGLE_FAN	Συνδεμένα τρίγωνα σε έλικα.
GL_QUAD_STRIP	Συνδεμένα πολύγωνα τεσσάρων πλευρών σε σειρά.

Πίνακας 4. Παράμετροι της glBegin και η σημασία τους.

1.12 Όψεις πολυγώνων

Στην OpenGL κάθε πολύγωνο χαρακτηρίζεται από δύο όψεις: τη μπροστινή και την πίσω όψη. Η επιλογή του ποια όψη θα χαρακτηριστεί ως μπροστινή ή πίσω είναι αυθαίρετη. Σε περιπτώσεις που ορίζουμε σχήματα κλειστά στο χώρο, για τα επιμέρους πολύγωνα που ορίζουν τα σχήματα αυτά, συνήθως χαρακτηρίζουμε την εξωτερική τους όψη τους ως "μπροστινή" και την εσωτερική ως "πίσω" όψη. Ο διαχωρισμός της μπροστινής από την πίσω όψη είναι σημαντικός στην περίπτωση που θέλουμε να αποδώσουμε διαφορετική υφή σε κάθε επιφάνεια.

Στην OpenGL, ο προσανατολισμός της επιφάνειας ενός πολυγώνου καθορίζεται ανάλογα με τη φορά δήλωσης των κορυφών του. Εάν ο θεατής δηλώσει τις κορυφές του πολυγώνου με αριστερόστροφη φορά από τη δική του οπτική γωνία τότε, η ορατή σε αυτόν πλευρά θα είναι η μπροστινή Εάν σχηματίσει το πολύγωνο με δήλωση των κορυφών με τη φορά των δεικτών του ρολογιού – πάντα από τη δικη του οπτική γωνία - τότε στον θεατή θα είναι ορατή η πίσω όψη (Σχ. 1.9).



Σχ. 1.9 Καθορισμός όψεων πολυγώνων

Ο παραπάνω τρόπος δήλωσης του προσανατολισμού επιφανειών πολυγώνου είναι και η προκαθορισμένη σύμβαση στην OpenGL. Ωστόσο, αυτή μπορεί να μεταβληθεί με την εντολή ***glFrontFace***:

glFrontFace(Glenum mode);

όπου η παράμετρος *mod*e παίρνει δύο πιθανές τιμές:

GL_CCW:
Η ορατή στο θεατή όψη χαρακτηρίζεται ως μπροστινή όψη (front facing polygon) εάν οι κορυφές του πολυγώνου δηλωθούν κατά τη θετική φορα (counterclockwise) από την οπτική γωνία του θεατή. Αυτή είναι και η αρχικά καθορισμένη κατάσταση λειτουργίας.

GL_CW:

Η ορατή στο θεατή όψη χαρακτηρίζεται ως μπροστινή όψη εάν οι κορυφές του πολυγώνου δηλωθούν κατά

την αρνητική φορα (clockwise) από την οπτική γωνία του θεατή.

1.13 Τροποποίηση σχεδίασης πολυγώνων

Κάθε όψη ενός πολυγώνου (μπροστινή και πίσω) μπορεί να σχεδιαστεί με διαφορετικό τρόπο, ανάλογα με τις ρυθμίσεις που ισχύουν για την εκάστοτε πλευρά. Π.χ. μπορούμε να σχεδιάζουμε τις μπροστινές όψεις των πολυγώνων συμπαγείς και τις πίσω όψεις ως περιγράμματα. Η προκαθορισμένη κατάσταση λειτουργίας στην OpenGL ορίζει ότι και οι μπροστινές και οι πίσω όψεις σχεδιάζονται συμπαγείς, ωστόσο η συμπεριφορά αυτή μπορεί να τροποποιηθεί με την εντολή ***glPolygonMode()***:

void glPolygonMode(GLenum face, GLenum mode);

όπου η παράμετρος *face* δηλώνει την όψη για την οποία θέλουμε να επιβάλουμε την τροποποίηση. Η παράμετρος *mode* δηλώνει τον τρόπο απεικόνισης της εκάστοτε όψης

Η παράμετρος *face* δέχεται τρείς πιθανές τιμές:
GL_FRONT: Η επιβαλλόμενη τροποποίηση αφορά τις μπροστινές όψεις
GL_BACK: Η επιβαλλόμενη τροποποίηση αφορά τις πίσω όψεις
GL_FRONT_AND_BACK: Η επιβαλλόμενη τροποποίηση αφορά και τις δύο όψεις.

Για την παράμετρο *mode* ορίζονται οι επιλογές:
GL_FILL: Η όψη σχεδιάζεται σημπαγής. Αυτή είναι η προκαθορισμένη επιλογή για μπροστινές και πίσω όψεις.
*GL*_LINE: Σχεδιάζεται μόνο το περίγραμμα της όψης.
GL_POINT: Σχεδιάζονται μόνο οι κορυφές της όψης.

Προκειμένου λ.χ. να σχεδιάσουμε τις μπροστινές όψεις ως περιγράμματα και τις κορυφές των πίσω όψεων, δίνουμε τις εντολές:

```
glPolygonMode(GL_FRONT,GL_LINE);
glPolygonMode(GL_BACK,GL_POINT);
```

1.14 Καταστολή όψεων πολυγώνων

Σε μια εντελώς κλειστή επιφάνεια που σχηματίζεται από πολύγωνικές επιφάνειες με τον ίδιο προσανατολισμό (όλες οι εξωτερικές επιφάνειες δηλωμένες ως μπροστινές), δεν θα είναι ποτέ καμία από τις πίσω όψεις τους ορατή σε έναν εξωτερικό παρατηρητή, διότι πάντα θα κρύβονται από τα μπροστινές όψεις. Στην περίπτωση αυτή, μπορούμε να αυξήσουμε την ταχύτητα σχεδιασμού, αναθέτοντας την OpenGL να

απορρίπτει τη σχεδίαση όψεων μόλις διαπιστώνει ότι είναι "πίσω όψεις". Ομοίως, σε έναν εσωτερικό παρατηρητή, μόνο οι πίσω όψεις θα είναι ορατές. Για να πληροφορήσουμε την OpenGL να απορρίπτει τη σχεδίαση μπροστινών όψεων (front faces) ή πίσω όψεων (back faces), χρησιμοποιούμε την εντολή ***glCullFace()***. Αρχικά απαιτείται η ενεργοποίηση της καταστολής με την εντολή ***glEnable( )*:**

void glEnable(GL_CULL_FACE);

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

void glCullFace(GLenum mode);

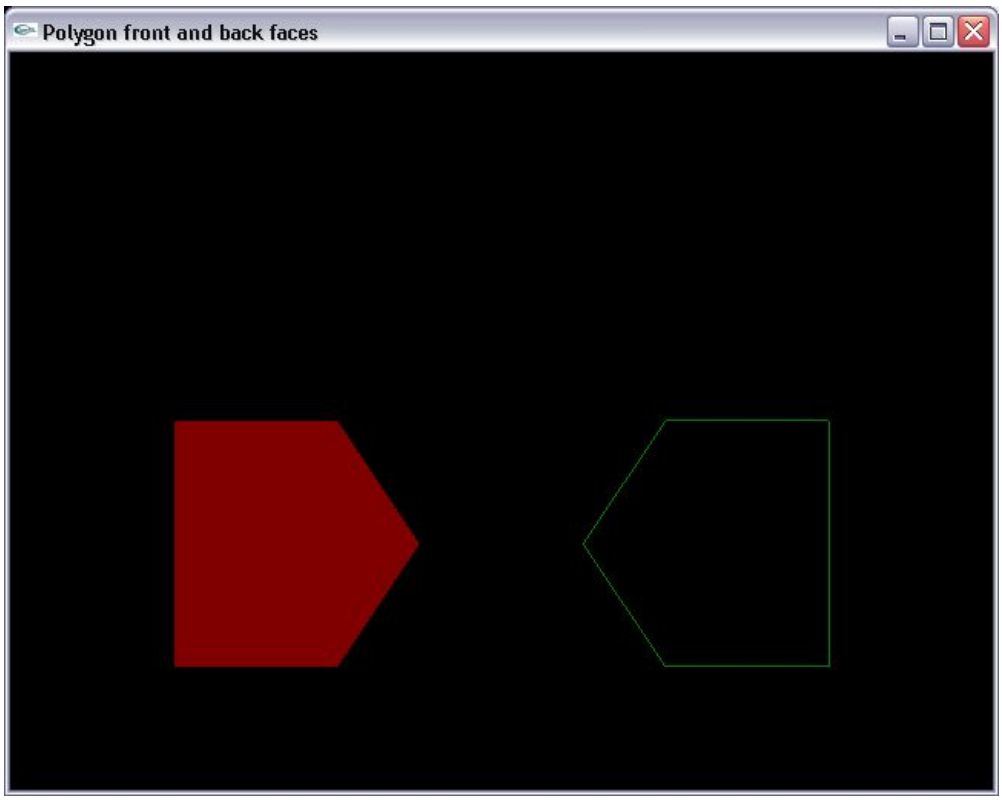
Η παραπάνω γραμμή κώδικα δηλώνει ποια πολύγωνα θα απορριφθούν. Το όρισμα *mode* δέχεται τις τιμές *GL_FRONT*, *GL_BACK* ή *GL_FRONT_AND_BACK* για να δηλώσει front-facing, back-facing ή όλα τα πολύγωνα.

Η ενεργοποίηση της καταστολής όψεων με την ***glEnable(GL_CULL_FACE)***, όταν δε συνοδεύεται από τη δήλωση εντολής ***glCullFace***, θεωρεί ως προκαθορισμένη λειτουργία την καταστολή των πίσω όψεων.

Παράδειγμα: Αλλαγή στον τρόπο σχεδίασης όψεων πολυγώνων

```
#include <glut.h>

void display()
{
	glClearColor(0,0,0,0);
	glClear(GL_COLOR_BUFFER_BIT);

	glColor3f(0.5,0,0);


	//Front face of a polygon - Vertices defined in counter clockwise order

	glBegin(GL_POLYGON);

	glVertex2i(0,0);
	glVertex2i(10,0);
	glVertex2i(15,10);
	glVertex2i(10,20);
	glVertex2i(0,20);

	glEnd();

	//Back face of a polygon - Vertices defined in clockwise order

	glColor3f(0,0.5,0);

	glBegin(GL_POLYGON);


	glVertex2i(30,0);
	glVertex2i(25,10);
```

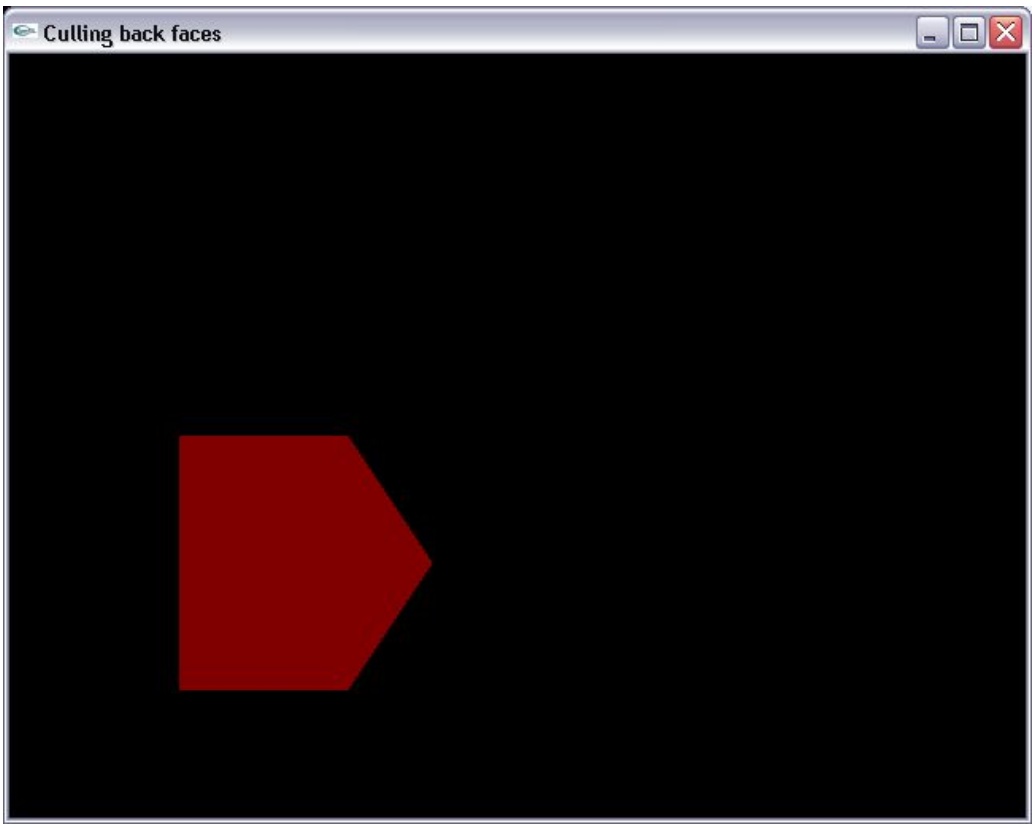
```
        glVertex2i(30,20);
        glVertex2i(40,20);
        glVertex2i(40,0);

        glEnd();

        glFlush();
}

int main(int argc, char** argv)
{
        glutInit(&argc,argv);
        glutInitWindowPosition(50,50);
        glutInitWindowSize(640,480);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGBA);
        glutCreateWindow("Polygon front and back faces");

        glMatrixMode(GL_PROJECTION);
        gluOrtho2D(-10,50,-10,50);

        //Polygon front faces are to be filled.

        glPolygonMode(GL_FRONT,GL_FILL);

        //Polygon back faces are to be drawn as lines

        glPolygonMode(GL_BACK,GL_LINE);


        glutDisplayFunc(display);

        glutMainLoop();

        return 0;
}
```



Παράδειγμα: Καταστολή πίσω όψεων

```
#include <glut.h>

void display()
{
	glClearColor(0,0,0,0);
	glClear(GL_COLOR_BUFFER_BIT);

	glColor3f(0.5,0,0);


	//Front face of a polygon - Vertices defined in counter clockwise order

	glBegin(GL_POLYGON);

	glVertex2i(0,0);
	glVertex2i(10,0);
	glVertex2i(15,10);
	glVertex2i(10,20);
	glVertex2i(0,20);

	glEnd();

	//Back face of a polygon - Vertices defined in clockwise order

	glColor3f(0,0.5,0);

	glBegin(GL_POLYGON);


	glVertex2i(30,0);
	glVertex2i(25,10);
	glVertex2i(30,20);
	glVertex2i(40,20);
	glVertex2i(40,0);

	glEnd();

	glFlush();
}

int main(int argc, char** argv)
{
	glutInit(&argc,argv);
	glutInitWindowPosition(50,50);
	glutInitWindowSize(640,480);
	glutInitDisplayMode(GLUT_SINGLE|GLUT_RGBA);
	glutCreateWindow("Culling back faces");

	glMatrixMode(GL_PROJECTION);
	gluOrtho2D(-10,50,-10,50);

	//Enable face culling. Now, back faces are culled by default.
	glEnable(GL_CULL_FACE);

	glutDisplayFunc(display);

	glutMainLoop();
```

```
        return 0;
}
```



1.15 Ομάδες ιδιοτήτων

Στην ενότητα «Ενεργοποίηση-απενεργοποίηση και επισκόπηση παραμέτρων κατάστασης» είδαμε ότι η επισκόπηση των μεταβλητών κατάστασης εκτελείται με τις εντολές glGetFloatv glGetDoublev κλπ. Ωστόσο, η χρήση πολλαπλών εντολών για την ανάκτηση ενός συνόλου ιδιοτήτων θα επιβάρυνε τον προγραμματιστή. Για να αποφευχθεί το φορτίο αυτό, στην OpenGL έχουν καθοριστεί οι ***ομάδες ιδιοτήτων*** (attribute groups). Δηλαδή, οι ιδιότητες της μηχανής κατάστασης έχουν ομαδοποιηθεί ούτως ώστε με μία μόνο εντολή να εκτελείται η αποθήκευση όλων των ιδιοτήτων μιας ομάδας σε μία θέση μνήμης για μελλοντική χρήση. Μια τέτοια αποθήκευση είναι χρήσιμη στην περίπτωση που θέλουμε να μεταβάλλουμε σε ένα ενδιάμεσο στάδιο του κώδικα κάποιες ιδιότητες και αργότερα να επαναφέρουμε τις αποθηκευμένες ρυθμίσεις.

Οι ομάδες ιδιοτήτων αποθηκεύονται στη ***στοίβα ιδιοτήτων***, μια περιοχή μνήμης που χρησιμοποιεί η μηχανή της OpenGL για το σκοπό αυτό. Η αποθήκευση γίνεται με τη χρήση της εντολής ***glPushAttrib()***:

void glPushAttrib(attributeGroup);

όπου *attributeGroup* μια αριθμητική σταθερά που καθορίζει την ομάδα ιδιοτήτων που θέλουμε να αποθηκεύσουμε. Στην OpenGL έχουν καθοριστεί οι εξής ομάδες ιδιοτήτων:

α) Ομάδα ιδιοτήτων σημείων (*GL_POINT_BIT*):
Περιέχει τις παραμέτρους που χαρακτηρίζουν την εμφάνιση ενός σημείου όπως λ.χ. το μέγεθος κουκίδας

β) Ομάδα ιδιοτήτων γραμμών (*GL_LINE_BIT*):
Περιέχει τις ιδιότητες που χαρακτηρίζουν την εμφάνιση μιας γραμμής όπως λ.χ. το πάχος της και η διάστιξή της.

γ) Ομάδα ιδιοτήτων πολυγώνων (*GL_POLYGON_BIT*):
Περιέχει τις ιδιότητες που χαρακτηρίζουν την εμφάνιση ενός πολυγώνου όπως λ.χ. τις συμβάσεις ως προς τον καθορισμό και τον τρόπο σχεδίασης της μπροστινής και πίσω όψης τους.

δ) «Ομάδα» ιδιοτήτων χρώματος (*GL_CURRENT_BIT*):
Το χρώμα εντάσσεται στην «ομάδα» ιδιοτήτων *GL_CURRENT_BIT*.

Η ανάκληση των τιμών ιδιοτήτων που αποθηκεύτηκαν στη στοίβα ιδιοτήτων γίνεται εκτελώντας την εντολή ***glPopAttrib()***:

void glPopAttrib( );

Επισημαίνουμε ότι η ***glPopAttrib*** ανακαλεί **μόνο** τις τιμές των ομάδων ιδιοτήτων που αποθηκεύτηκαν στο παρελθόν στη στοίβα ιδιοτήτων με εντολές ***glPushAttrib***.

1.16 Λίστες απεικόνισης (display lists)

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

Μια λίστα απεικόνισης περικλείεται μεταξύ δύο εντολών: των ***glNewList*** και ***glEndList***. Όταν σε ένα πρόγραμμα ορίζονται πολλαπλές λίστες, μια αυτόνομη λίστα απεικόνισης διακρίνεται από τις υπόλοιπες βάσει ενός ακεραίου αριθμού που παίζει το ρόλο του "αναγνωριστικού αριθμού" της **(list identifier)**. Προεκιμένου να αποδώσουμε αναγνωριστικούς αριθμούς σε λίστες απεικόνισης, πρέπει να δεσμεύσουμε το απαιτούμενο εύρος τιμών. Η δέσμευση αυτή γίνεται με την εντολή ***glGenLists()***.

GLuint glGenLists(GLint range);

όπου *range* το πλήθος των αναγνωριστικών που θέλουμε να χρησιμοποιήσουμε. Η συνάρτηση επιστρέφει μια ακέραιη τιμή που αντιστοιχεί στον πρώτο αναγνωριστικό αριθμό.

Π.χ. με τη σύνταξη

```
listID=glGenLists(2);
```

παράγουμε 2 identifiers. Ο πρώτος έχει ακέραιη τιμή *listID*, ο δεύτερος *listID+1* και ούτω καθ' εξής.

Η ανάθεση αναγνωριστικής τιμής σε μια λίστα απεικόνισης γίνεται κατά την έναρξη της δήλωσής της στην εντολή ***glNewList***:

void glNewList(GLuint listID, GLenum listMode);

όπου *listID* το αναγνωριστικό που θέλουμε να αποδώσουμε στη λίστα απεικόνισης. Η παράμετρος *listMode* έχει δύο πιθανές τιμές:

GL_COMPILE: Δηλώνουμε τον κώδικα σχεδιασμού του σύνθετου αντικειμένου που περιγράφεται στη λίστα απεικόνισης
GL_COMPILE_AND_EXECUTE: Δηλώνουμε **και εκτελούμε ταυτόχρονα** τον κώδικα σχεδιασμού που περιέχεται στη λίστα απεικόνισης.

Η δήλωση λοιπόν ενός σύνθετου σχήματος σε λίστα απεικόνισης έχει τη μορφή:

```
glNewList();
// Εντολές δήλωσης σχήματος
glEndList();
```

Μεταξύ των εντολών ***glNewList*** και ***glEndList*** ορίζουμε το σύνθετο γεωμετρικό σχήμα της λίστας απεικόνισης, χρησιμοποιώντας τις εντολές σχεδιάσης σχημάτων που αναφέραμε παραπάνω.

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

void glCallList(GLuint listID);

Μία σημαντική λεπτομέρεια που ο προγραμματιστής πρέπει να έχει υπόψη, αφορά τις πιθανές αλλαγές των μεταβλητών κατάστασης κατά την εκτέλεση μιας λίστας απεικόνισης. Πρέπει να έχουμε υπόψη ότι, εάν

μέσα σε μια display list μεταβάλλουμε την τιμή μιας μεταβλητής κατάστασης (όπως π.χ. του τρέχοντος χρώματος σχεδίασης), **η μεταβολή αυτή θα παραμείνει ενεργή και μετά το πέρας εκτέλεσης της λίστας**. Δηλαδή εάν μέσα στη display list έχει οριστεί την τελευταία φορά ως χρώμα σχεδίασης το κόκκινο μετά το πέρας της εκτέλεσης της λίστας η παράμετρος θα διατηρήσει την τελευταία τιμή και θα πρέπει να τη μεταβάλλει ο προγραμματιστής. Είναι χρήσιμο λοιπόν ο προγραμματιστής να αποθηκεύσει τις τιμές των ιδιοτήτων που θα μεταβληθούν στη στοίβα ιδιοτήτων, πριν από την κλήση της λίστας, ούτως ώστε να είναι σε θέση να τις επαναφέρει μετά το πέρας της εκτέλεσης της λίστας.

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

glDeleteLists(startId, nLists);

όπου *startId* ο αναγνωριστικός αριθμός της πρώτης λίστας απεικόνισης και *nLists* το πλήθος των λιστών που θέλουμε να διαγράψουμε.

Πχ. Δίνοντας

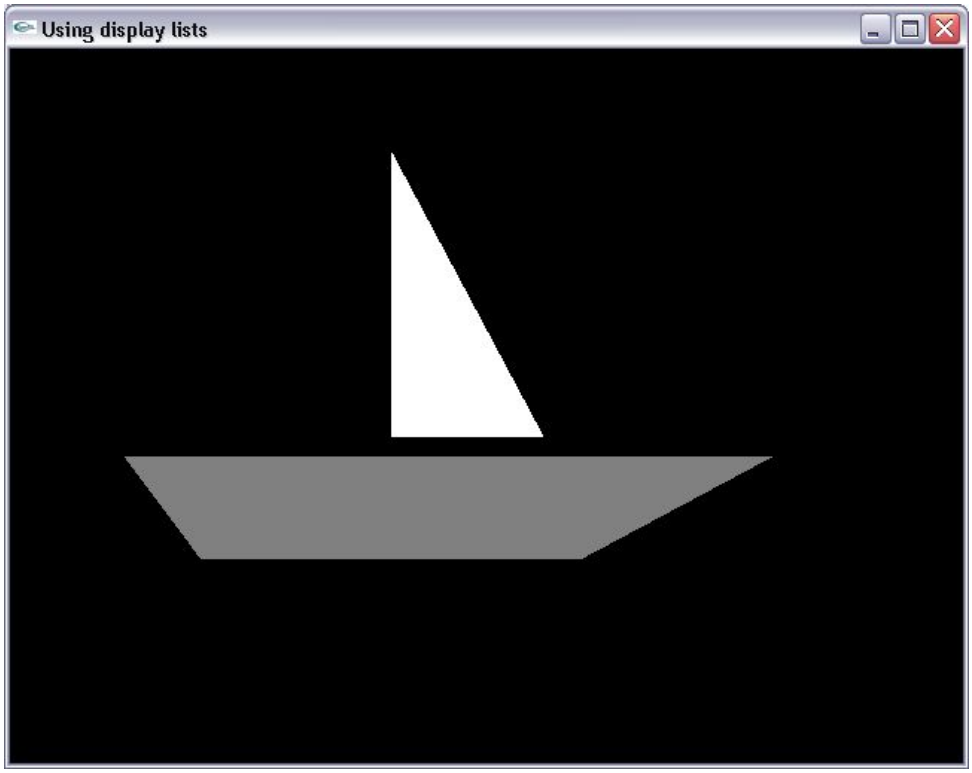
```
glDeleteLists(someListID,3);
```

Διαγράφουμε τις λίστες απεικόνισης με αναγνωριστικούς αριθμούς *someListID*, *someListId+1* και *someListId+2*.

Παράδειγμα: Ορισμός και εκτέλεση λίστας απεικόνισης

```
#include <glut.h>

GLuint listID;

void Boat(GLsizei displayListID)
{
      glNewList(displayListID,GL_COMPILE);

      glColor3f(0.5,0.5,0.5);
      glBegin(GL_POLYGON);
      glVertex2f(15,10);
      glVertex2f(20,15);
      glVertex2f(3,15);
      glVertex2f(5,10);
      glEnd();

      glColor3f(1,1,1);
      glBegin(GL_TRIANGLES);
      glVertex2f(14,16);
      glVertex2f(10,30);
      glVertex2f(10,16);
      glEnd();

      glEndList();
```

```
}

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

	glClear(GL_COLOR_BUFFER_BIT);

	glCallList(listID);

	glFlush();
}

int main(int argc, char** argv)
{
	glutInit(&argc,argv);
	glutInitWindowPosition(50,50);
	glutInitWindowSize(640,480);
	glutInitDisplayMode(GLUT_SINGLE|GLUT_RGBA);
	glutCreateWindow("Using display lists");

	glMatrixMode(GL_PROJECTION);
	gluOrtho2D(0,25,0,35);

	listID=glGenLists(1);

	Boat(listID);

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



1.17 Μητρώα σημείων - Μητρώα χρωμάτων

Στις προηγούμενες ενότητες αναφερθήκαμε στις εντολές σχεδίασης γεωμετρικών σχημάτων με τη χρήση της δομής glBegin/glEnd. Ωστόσο, αν επιχειρήσει κανείς να σχεδιάσει περισσότερο πολύπλοκες σκηνές, εάν βασιστεί αποκλειστικά στις προαναφερθείσες εντολές, θα χρειαστεί να συντάξει έναν υπερβολικά μεγάλο αριθμό γραμμών κώδικα γα την απόδοση των σχημάτων.

Για να διευκολύνει τη σύνθεση πολύπλοκων σκηνών, η OpenGL παρέχει την τεχνική των **μητρώων σημείων** (**vertex arrays**). Η χρησιμότητα των μητρώων σημείων αναδεικνύεται σε εφαρμογές μεγάλης κλίμακας, στις οποίες έχουμε ένα πολύ μεγάλο όγκο δεδομένων (δειγμάτων με τη μορφή σημείων) και θέλουμε να σχηματίσουμε μια γραφική απεικόνιση αυτών των δεδομένων με τον ελάχιστο δυνατό αριθμό εντολών. Με τα μητρώα σημείων μπορούμε να επιβάλλουμε ένα πρότυπο ή μεθοδολογία σχεδίασης δηλώνοντας απλώς ένα μητρώο σημείων και τον τρόπο με τον οποίο θέλουμε να χρησιμοποιηθεί στη σχεδίαση αυτό το μητρώο σημείων. Η ιδέα των μητρώων σημείων επεκτείνεται και στην απόδοση χρωματικών τιμών σε μεγάλο πλήθος σημείων οπότε στην περίπτωση αυτή ορίζουμε τα μητρώα χρωμάτων (color arrays).

Αρχικά απαιτείται η ενεργοποίηση της χρήσης μητρώων σημείων ή/και χρωμάτων με την εντολή ***glEnableClientState***:

void glEnableClientState(GLenum array);

όπου *array* ο η κατηγορία των δεδομένων που περιέχονται στο μητρώο. Η OpenGL μπορεί να υποστηρίξει τους εξής τύπους μητρώων:

GL_VERTEX_ARRAY: Ενεργοποιούμε τη χρήση μητρώων που περιέχουν συντεταγμένες σημείων. Χρησιμοποιείται όταν θέλουμε να ορίσουμε τις συντεταγμένες πολλαπλών κορυφών.
GL_COLOR_ARRAY: Ενεργοποιούμε τη χρήση μητρώων που περιέχουν συντελεστές χρωματικών τιμών. Χρησιμοποιείται όταν θέλουμε να δηλώσουμε χρωματικές τιμές σε πολλαπλές κορυφές.
GL_TEXTURE_ARRAY: Αναλύεται στο Κεφάλαιο "Απόδοση υφής"

Όταν ορίζουμε τις συντεταγμένες όλων των κορυφών σε ένα μητρώο έστω vertexArray και δηλώνουμε στην OpenGL το μητρώο σημείων με την εντολή glVertexPointer:

void glVertexPointer(GLint size, GLenum type, GLsizei stride, const GLvoid *vertexPointer);

- Η παράμετρος **vertexPointer* είναι δείκτης στο μητρώο με τις τιμές των συντεταγμένων.
- Με το όρισμα *size* ορίζουμε το πλήθος των τιμών που προσδιορίζουν τις συντεταγμένες καθεμιάς

κορυφής. Σε διδιάστατες σκηνές έχει την τιμή 2, ενώ για σημεία στον τρισδιάστατο χώρο έχει την τιμή 3.

- Η παράμετρος *type* καθορίζει τον πρωτογενή τύπο δεδομένων με τον οποίο αποθηκεύονται οι τιμές των συντεταγμένων στο μητρώο *pointer*. Οι διαθέσιμες αριθμητικές σταθερές είναι *GL_INT*, *GL_SHORT*, *GL_FLOAT*, *GL_DOUBLE*.
- Η παράμετρος *stride* καθορίζει την απόσταση μεταξύ των συντεταγμένων διαδοχικών σημείων στο μητρώο.
- Αυτή η παράμετρος χρησιμοποιείται μόνο στην περίπτωση που αποθηκεύονται τιμές πολλών ιδιοτήτων στο ίδιο μητρώο (π.χ. όταν για κάθε σημείο αποθηκεύονται οι συντεταγμένες και οι συντελεστές του χρώματός του στο ίδιο μητρώο). Όταν το μητρώο περιέχει τιμές ενός μόνο χαρακτηριστικού (π.χ περιέχει μόνο συντεταγμένες σημείων) χρησιμοποιείται η τιμή 0.

Εάν θέλουμε να ορίσουμε πολλαπλές χρωματικές τιμές δίνοντας τα βάρη τους σε ένα μητρώο, έστω colorArray, χρησιμοποιούμε την εντολή ***glColorPointer*** *με παρόμοια σύνταξη*:

void glColorPointer(GLint size, GLenum type, GLsizei stride, const GLvoid *vertexPointer);

Στην περίπτωση αυτή η παράμετρος *size* καθορίζει το πλήθος των βαρών που προσδιορίζουν μια χρωματική τιμή. Στο μοντέλο RGB η παράμετρος παίρνει την τιμή 3.

Επιπλέον χρειάζεται να ορίσουμε:

α) με ποια σειρά θα χρησιμοποιηθούν οι κορυφές που περιέχονται στο μητρώο vertexArray ή/και οι χρωματικές τιμές που περιέχονται στο μητρώο colorArray

β) τι σχήματα ορίζουν οι κορυφές.

Η διαδοχή με την οποία χρησιμοποιούνται τα σημεία ορίζεται σε ένα μητρώο το οποίο περιέχει τις θέσεις των σημείων στο μητρώο vertexPointer Στο δέυτερο αυτό μητρώο, έστω *vertexIndex*, τα σημεία δηλώνονται με τη σειρά χρήσης τους. Π.χ. με το μητρώο δεικτών

$$vertexIndex = \{i_1, i_2, ..., i_n\}$$

δηλώνουμε ότι θα χρησιμοποιηθεί πρώτο το σημείο ή/και χρώμα που δηλώνεται i1-στό στο μητρώο σημείων/χρωμάτων, θα χρησιμοποιηθεί δεύτερο το σημείο/χρώμα που δηλώνεται i2-στό στο μητρώο σημείων/χρωμάτων και ούτω καθεξής. Η δήλωση της σειράς χρήσης παίζει σημαντικό ρόλο κυρίως όταν δηλώνουμε πολυγωνικές επιφάνειες και μας ενδιαφέρει η απόδοση της σωστής όψης.

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

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

όπου **indices* δείκτης στο μητρώο που περιέχει τους δείκτες των σημείων ή/και χρωμάτων με τη σειρά χρήσης τους. Το όρισμα mode καθορίζει τι σχήματα ορίζουν τα σημεία και παίρνει τις ίδιες τιμές που χρησιμοποιούνται στη δομή glBegin/glEnd (*GL_LINES*, *GL_TRIANGLES*, *GL_QUADS* κ.λ.π.). Το όρισμα count προσδιορίζει το πλήθος των σημείων ή/και χρωματικών τιμών που περιέχονται στα μητρώα σημείων ή/και χρωμάτων. Το όρισμα type καθορίζει τον πρωτογενή τύπο δεδομένων με τον οποίο δίνονται οι δείκτες στο μητρώο indices. Οι επιτρεπόμενες αριθμητικές σταθερές είναι *GL_UNSIGNED_BYTE*, *GL_UNSIGNED_SHORT* και *GL_UNSIGNED_INT* για πρωτογενείς τύπους δεδομένων *GLubyte*, *GLushort* και *GLuint* αντίστοιχα.

Παράδειγμα: Σχεδίαση ταινίας τριγώνων

Ας πάρουμε ως παράδειγμα τη σχεδίαση μιας ταινίας τριγώνων (TRIANGLE_STRIP) που ορίζεται από πέντε σημεία $(v1, v2, v3, v4, v5)$. Για τη σχεδίαση, με τη συμβατική δομή glBegin/glEnd, απαιτούνται οι εντολές

```
glBegin(GL_TRIANGLE_STRIP);
glVertex3fv(v1);
glVertex3fv(v2);
glVertex3fv(v3);
glVertex3fv(v4);
glVertex3fv(v5);
glVertex3fv(v6);
glVertex3fv(v7);
glEnd();
```

Με τη χρήση ενός μητρώου vertArray που περιέχει τις συντεταγμένες των σημείων

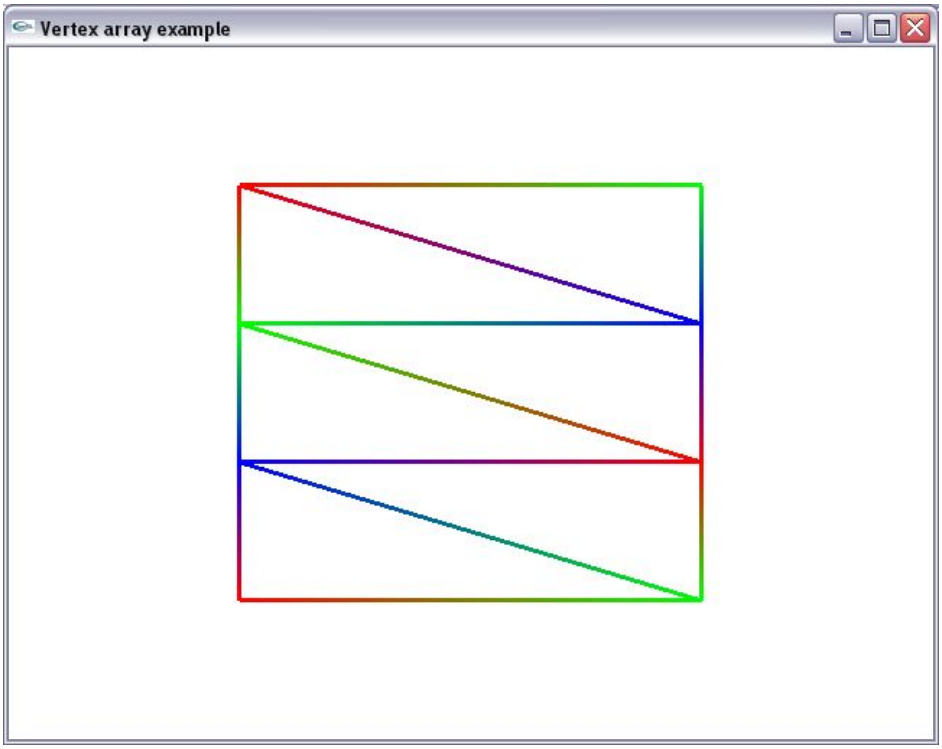
$$vertArray = \left(v1_x, v1_y, v1_z, v2_x, v2_y, v2_z, ..., v7_x, v7_y, v7_z\right)$$

Η σχεδίαση της αλληλουχίας εκτελείται με τις εντολές

```
GLfloat vertArray={v1x,v1y,v1z,v2x,v2y,v2z,...,v7x,v7y,v7z};
GLubyte vertIndex={0,1,2,3,4,5,6,7};
glVertexPointer(3,GL_FLOAT,0,vertexArray);
glDrawElements(GL_TRIANGLE_STRIP, 5, GL_UNSIGNED_BYTE,vertIndex);
```

χρησιμοποιώντας μικρότερο πλήθος εντολών κατά τη σύνταξη του κώδικα. Σε εφαρμογές σχεδίασης μεγαλύτερης κλιμακας, το κέρδος είναι ακόμη μεγαλύτερο.

```
#include <glut.h>

GLint vertexArray[]={0,0, 10,0, 0,5, 10,5, 0,10, 10,10, 0,15, 10,15};

GLfloat colorArray[]={1,0,0, 0,1,0, 0,0,1, 1,0,0, 0,1,0, 0,0,1, 1,0,0, 0,1,0};

GLuint vertexIndex[]={0,1,2,3,4,5,6,7};

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

	glEnableClientState(GL_VERTEX_ARRAY);
	glEnableClientState(GL_COLOR_ARRAY);

	glVertexPointer(2,GL_INT,0,vertexArray);
	glColorPointer(3,GL_FLOAT,0,colorArray);

	glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);

	glDrawElements(GL_TRIANGLE_STRIP,8,GL_UNSIGNED_INT,vertexIndex);

	glFlush();
}

int main(int argc, char** argv)
{
	glutInit(&argc,argv);
	glutInitWindowPosition(50,50);
	glutInitWindowSize(640,480);
	glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
	glutCreateWindow("Vertex array example");

	glMatrixMode(GL_PROJECTION);
	gluOrtho2D(-5,15,-5,20);

	glutDisplayFunc(display);
	glutMainLoop();

	return 0;
}
```



1.18 Περιορισμοί κατά τη χρήση των glBegin( ) και glEnd( )

Η πιο σημαντική πληροφορία γις τις κορυφές είναι οι συντεταγμένες τους, οι οποίες καθορίζονται από την εντολή ***glVertex*().*** Μπορούμε επίσης να προσθέσουμε επιπλέον εξειδικευμένες παραμέτρους για κάθε κορυφή, όπως το χρώμα, προσανατολισμό κανονικού διανύσματος, συντεταγμένες υφής ή και συνδυασμό αυτών χρησιμοποιώντας ειδικές εντολές. Υπάρχουν μερικές ακόμα εντολές που είναι έγκυρες μεταξύ του ζευγαριού ***glBegin()*** και ***glEnd()***. Ο Πίνακας 5 περιέχει μια πλήρη λίστα των έγκυρων εντολών. Ορισμένες από τις έγκυρες εντολές που περιέχονται στον πίνακα αναλύονται σε μεταγενέστερα κεφάλαια.

Πίνακας 4. Έγκυρες εντολές εντός της δομής glBegin( ) και glEnd( )

Εντολή	**Σημασία**
glVertex*()	Θέτει τις συντεταγμένες της κορυφής.
glColor*()	Θέτει το τρέχον χρώμα.
glIndex*()	Θέτει τον τρέχοντα χρωματικό πίνακα.
glNormal*()	Θέτει τις συντεταγμένες του κανονικού διανύσματος.
glEvalCoord*()	Δημιουργεί συντεταγμένες.
glCallList(), glCallLists()	Εκτελεί την/τις λίστα/λίστες απεικόνισης.
glTexCoord*()	Θέτει τις συντεταγμένες της υφής. (Κεφάλαιο 6)
glEdgeFlag*()	Ελέγχει το σχεδιασμό των άκρων.
glMaterial*()	Θέτει τις ιδιότητες του υλικού.

Καμμία άλλη εντολή εκτός από αυτές που προαναφέρθηκαν δεν είναι έγκυρη μεταξύ του ζευγαριού ***glBegin*** και ***glEnd*** και οποιαδήποτε άλλη κλήση σε OpenGL δημιουργεί σφάλμα. Ωστόσο, μπορούμε να συμπεριλάβουμε πρόσθετους τύπους σύνταξης εντολών, όπως λ.χ. επαναληπτικούς βρόχους δήλωσης

σημείων με δομές for().

Οι εντολές ***glVertex**** πρέπει να εμφανίζονται μεταξύ ενός συνδυασμού ***glBegin*** και ***glEnd***. Αν εμφανίζονται αλλού δεν έχουν αποτέλεσμα. Αν εμφανίσθουν σε μια display list, εκτελούνται μόνο μεταξύ των ***glBegin*** και ***glEnd***.

Παρόλο που πολλές εντολές επιτρέπονται μεταξύ των ***glBegin()*** και ***glEnd()***, οι κορυφές δημιουργούνται μόνο όταν ακολουθεί η εντολή ***glVertex*()***. Από τη στιγμή που κληθεί η εντολή ***glVertex*()***, η OpenGL αναθέτει στην κορυφή που προκύπτει τις τρέχουσες μεταβλητές κατάστασης (όπως λ.χ. το τρέχον χρώμα σχεδίασης). Η επίδραση των μεταβλητών κατάστασης φαίνεται στο ακόλουθο δείγμα κώδικα.

```
glBegin(GL_POINTS);
   glColor3f(0.0, 1.0, 0.0);    // Επιλογή πράσινου χρώματος
   glVertex(...);              //Ανάθεση πράσινου χρώματος
   glColor3f(1.0, 0.0, 0.0);    // Επιλογή κόκκινου χρώματος
   glVertex(...);              //Ανάθεση κόκκινου χρώματος
   glColor3f(0.0, 0.0, 1.0);   // Επιλογή μπλε χρώματος
   glVertex(...);              //Ανάθεση μπλέ χρώματος
glEnd();
```