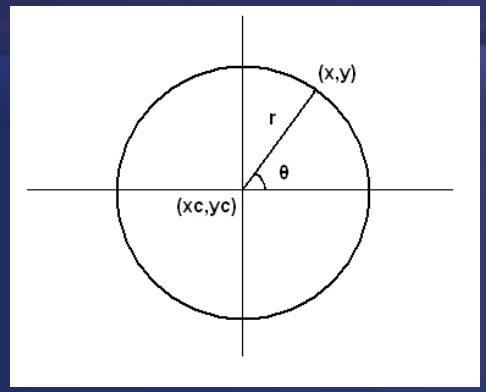
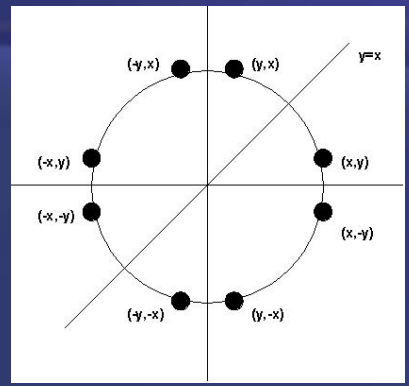
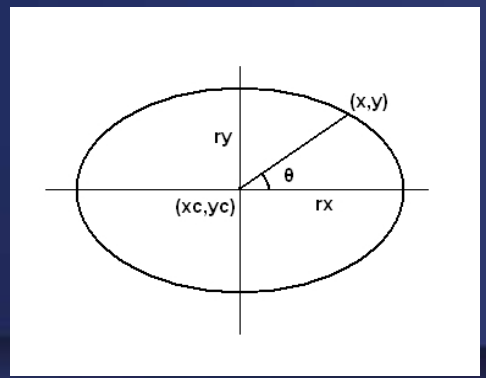
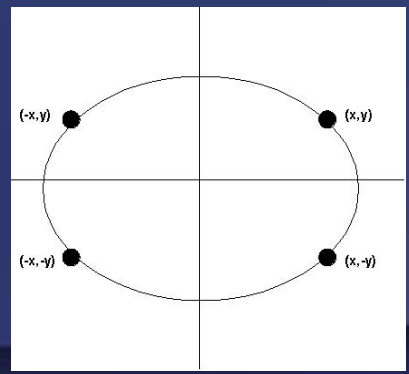
Βασικές αρχές σχεδίασης (β)









Περιεχόμενα ενότητας

- Σχεδίαση πολυγώνων
- Σχεδίαση τριγώνων
- Σχεδίαση καμπυλών
- Όψεις πολυγωνικών επιφανειών - Ρύθμιση σχεδίασης όψεων
- Ομάδες ιδιοτήτων
- Λίστες απεικόνισης
- Μητρώα σημειών
- Μητρώα χρωμάτων

Πολύγωνα

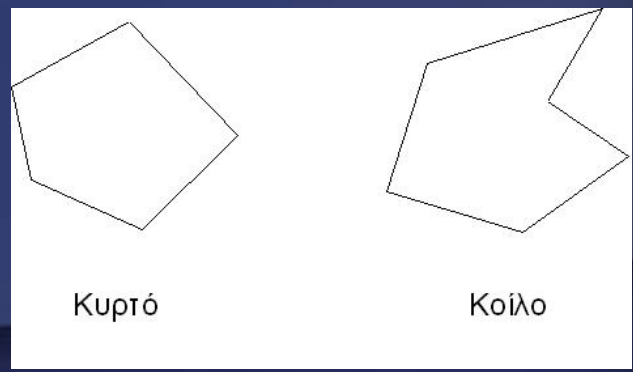
•Πολύγωνα: περιοχές που περικλείονται από βρόχους ευθύγραμμων τμημάτων.

•Τα πολύγωνα σχεδιάζονται συμπαγή ή τα περιγράμματά τους ή οι κορυφές τους.

•Κατηγορίες πολυγώνων: κυρτά και κοίλα

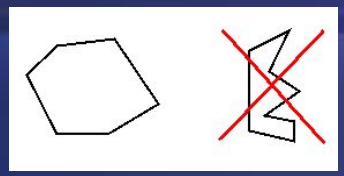
Κυρτά πολύγωνα:
Όλες οι εσωτερικές γωνίες τους είναι μικρότερες των 180 μοιρών

Κοίλα πολύγωνα:
Περιέχουν τουλάχιστον μια εσωτερική γωνία μεγαλύτερη των 180 μοιρών.



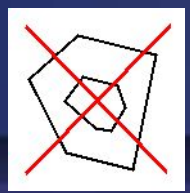
Περιορισμοί στη σχεδίαση πολυγώνων

α) Οι ρουτίνες της OpenGL σχεδιάζουν κυρτά πολύγωνα.



β) Οι πλευρές των πολυγώνων δεν μπορούν να τέμνονται.

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



Εάν δώσουμε εντολή σχεδίασης ενός “μη έγκυρου πολυγώνου”, το αποτέλεσμα θα είναι απρόβλεπτο.

Σχεδίαση πολυγώνων

Δηλώνουμε τις κορυφές ενός ή περισσοτέρων πολυγώνων μεταξύ των εντολών glBegin και glEnd.

Τρεις καταστάσης σχεδίασης:

α) Σχεδίαση πολύγωνων

β) Σχεδίαση τετραπλεύρων

γ) Σχεδίαση αλυσίδας τετραπλεύρων

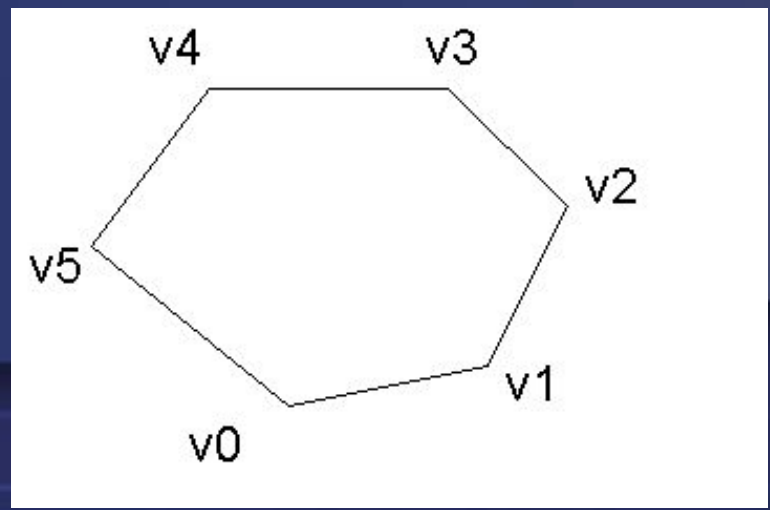
Σχεδίαση πολυγώνου (GL_POLYGON)

glBegin(GL_POLYGON);

•Τα σημεία ορίζουν τις διαδοχικές κορυφές ενός μόνο πολυγώνου (*πλήθος κορυφών* ≥ 3)

•Το πολύγωνο πρέπει να είναι κυρτό και να μην έχει τεμνόμενες πλευρές.

```
glBegin(GL_POLYGON);
glVertex*(v0);
glVertex*(v1);
glVertex*(v2);
glVertex*(v3);
glVertex*(v4);
glVertex*(v5);
glEnd();
```



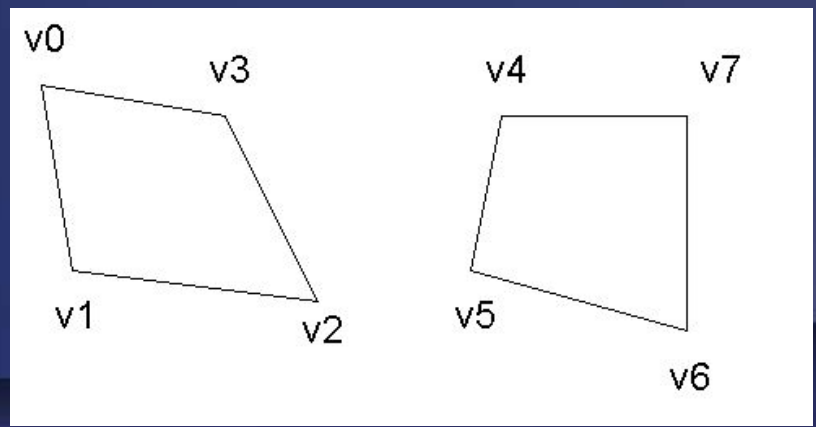
Σχεδίαση τετραπλεύρων (GL_QUADS)

glBegin(GL_QUADS):Οι κορυφές ορίζουν ανά τέσσερις διακεκριμένα τετράπλευρα.

$(v_0, v_1, v_2, \ldots, v_{n-1}) \longrightarrow (v_0, v_1, v_2, v_3), (v_4, v_5, v_6, v_7)$ κοκ

Αν το πλήθος κορυφών δεν είναι πολλαπλάσιο του 4, τότε η μία, δύο ή τρεις τελευταίες κορυφές παραλείπονται.

```
glBegin(GL_QUADS);
glVertex*(v0);
glVertex*(v1);
glVertex*(v2);
glVertex*(v3);
glVertex*(v4);
glVertex*(v5);
glVertex*(v6);
glVertex*(v7);
glEnd();
```



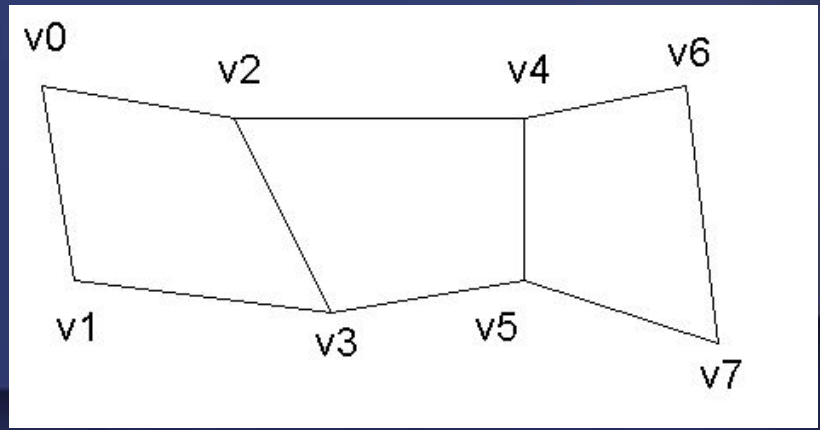
Σχεδίαση αλυσίδας τετραπλεύρων (GL_QUAD_STRIP)

glBegin(GL_QUAD_STRIP);
Ορισμός αλυσίδας τετραπλεύρων με μία κοινη πλευρά.

$(v_0,v_1,v_2,\ldots,v_n) \longrightarrow (v_0,v_1,v_3,v_2),\ (v_2,v_3,v_5,v_4)\ (v_4,v_5,v_7,v_6)$ κοκ

•Εάν το πλήθος των κορυφών είναι περιττό, η τελευταία κορυφή παραλείπεται.

```
glBegin(GL_QUAD_STRIP);
glVertex*(v0);
glVertex*(v1);
glVertex*(v2);
glVertex*(v3);
glVertex*(v4);
glVertex*(v5);
glVertex*(v6);
glVertex*(v7);
glEnd();
```



Σχεδίαση τριγώνων

•Ειδικές περιπτώσεις των πολυγώνων

Τρεις παραλλαγές σχεδίασης:

α) Σχεδίαση ανεξαρτήτων τριγώνων

β) Σχεδίαση αλυσίδας τριγώνων με κοινή πλευρά

γ) Σχεδίαση αλυσίδας τριγώνων με κοινή κορυφή

Σχεδίαση ανεξαρτήτων τριγώνων (GL_TRIANGLES)

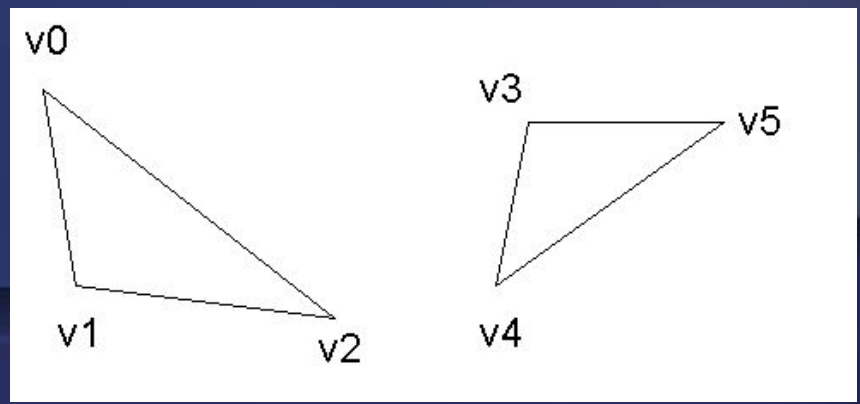
glBegin(GL_TRIANGLES);

Τα σημεία ορίζουν ανά τριάδες ανεξάρτητα τρίγωνα.

$(v_0,v_1,v_2,\ldots,v_{n-1}) \longrightarrow (v_0,v_1,v_2),\ (v_3,v_4,v_5)$ κοκ

•Αν το πλήθος κορυφών δεν είναι ακέραιο πολλαπλάσιο του 3, η τελευταία ή οι δύο τελευταίες κορυφές παραλείπονται.

```
glBegin(GL_TRIANLGES);
glVertex*(v0);
glVertex*(v1);
glVertex*(v2);
glVertex*(v3);
glVertex*(v4);
glVertex*(v5);
glEnd();
```



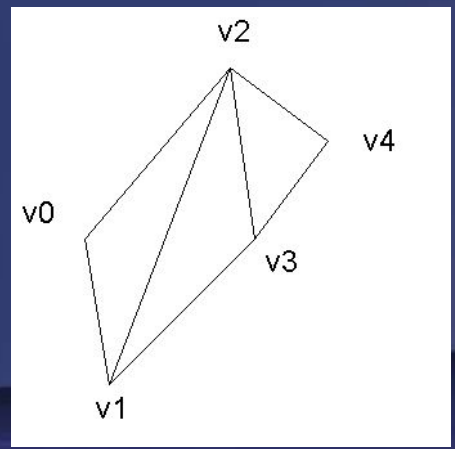
Σχεδίαση αλυσίδας τριγώνων (GL_TRIANGLE_STRIP)

glBegin(GL_TRIANGLE_STRIP):

Οι κορυφές ορίζουν μια αλυσίδα τριγώνων. Διαδοχικά τρίγωνα της αλυσίδας έχουν μία κοινή πλευρά.

$(v_0,v_1,v_2,\ldots,v_n) \longrightarrow (v_0,v_1,v_2),\ (v_2,v_1,v_3),\ (v_2,v_3,v_4)$ κοκ

```
glBegin(GL_TRIANLGE_STRIP);
glVertex*(v0);
glVertex*(v1);
glVertex*(v2);
glVertex*(v3);
glVertex*(v4);
glEnd();
```



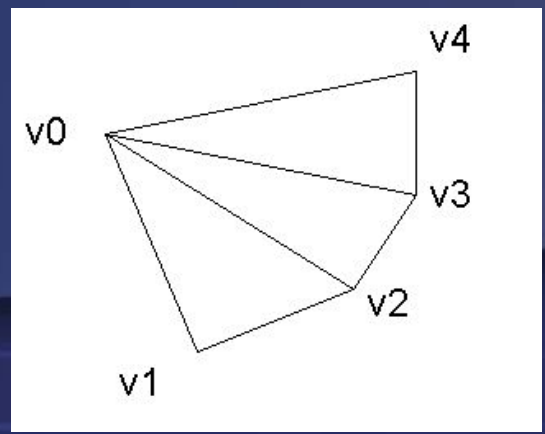
Σχεδίαση αλυσίδας τριγώνων (GL_TRIANGLE_FAN)

glBegin(GL_TRIANGLE_FAN):

Οι κορυφές ορίζουν μια αλυσίδα τριγώνων. Διαδοχικά τρίγωνα της αλυσίδας έχουν μία κοινή πλευρά και όλα τα τρίγωνα έχουν την πρώτη κορυφή κοινή

$(v_0,v_1,v_2,\ldots,v_n) \longrightarrow (v_0,v_1,v_2),\ (v_0,v_2,v_3),\ (v_0,v_3,v_4)$ κοκ

```
glBegin(GL_TRIANGLE_FAN);
glVertex*(v0);
glVertex*(v1);
glVertex*(v2);
glVertex*(v3);
glVertex*(v4);
glEnd();
```



Σχεδίαση ορθογωνίων

•Η OpenGL επιτρέπει το σχεδιασμό ορθογωνίων με την εντολή glRect*.

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

TYPE: τύπος δεδομένος που καθορίζεται από την παραλλαγή της glRect* που επιλέγουμε (GLfloat, GLint κλπ)

(x1,y1) (x2,y2): συντεταγμένες των κορυφών της μίας διαγωνίου του ορθογωνίου

•Χρησιμοποιώντας την glRect τα ορθογώνια σχεδιάζονται στο επίπεδο z=0 με τις ακμές τους παράλληλες στους άξονες x,y.

•Οι παραλλαγές glRect*v δέχονται τις συντεταγμένες των κορυφών της διαγωνίου με τη μορφή μητρώων

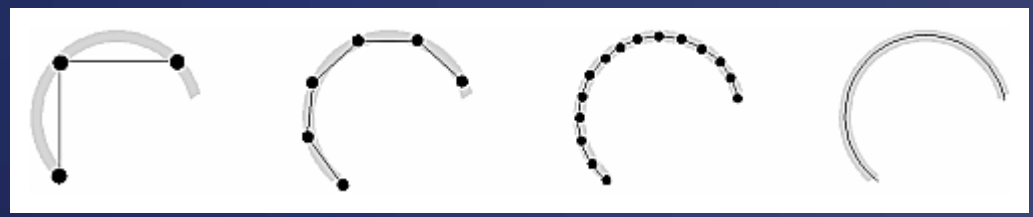
void glRect{sifd}v(TYPE *v1, TYPE *v2);

Σχεδίαση καμπυλών

•Οποιαδήποτε καμπύλη γραμμή μπορεί να προσεγγιστεί από στοιχειώδη ευθύγραμμα τμήματα

•Οποιαδήποτε επιφάνεια μπορεί να προσεγγιστεί ή από ένα πολυγωνικό πλέγμα.

•Με επαρκή δειγματοληψία, μπορούμε να υποδιαιρέσουμε καμπύλες γραμμές και επιφάνειες σε επιμέρους ευθύγραμμα τμήματα ή επίπεδα πολύγωνα.



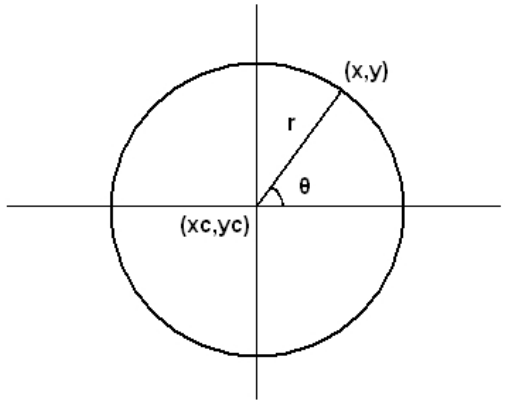
•Οι βιβλιοθήκες GLU και GLUT προσφέρουν εντολές για τη σχεδιασή ορισμένων σύνθετων γεωμετρικών σχημάτων.

Σχεδίαση κύκλου

•Η σχεδίαση κυκλικών σχημάτων επιτυγχάνεται χρησιμοποιώντας την παραμετρική εξίσωση κύκλου σε πολικές συντεταγμένες.

$$x = x_c + r \cdot \cos\theta$$
$$y = y_c + r \cdot \sin\theta$$

$$0 \le \theta \le 2\pi$$



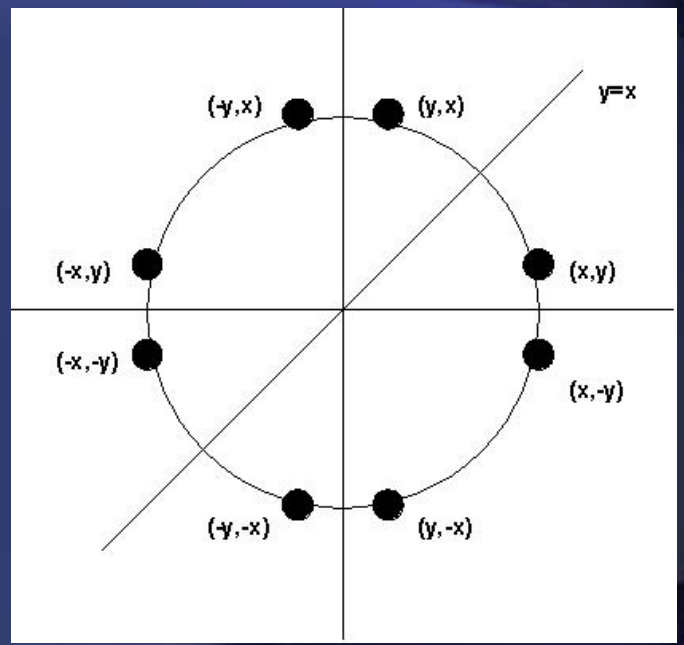
•Με ένα στοιχειώδες γωνιακό βήμα dθ, σχεδιάζουμε κυκλικά σχήματα, συνενώνοντας τα σημεία της περιφέρειάς του με στοιχειώδη ευθύγραμμα τμήματα.

•Για τον υπολογισμό των συντεταγμένων κάθε σημείου του κύκλου απαιτείται ο υπολογισμός δύο τριγωνομετρικών αριθμών, κάτι που εισάγει υπολογιστικό κόστος.

Συμμετρίες κύκλου

•Συμμετρίες ως προς την ευθεία y=x και ως προς τους άξονες x,y.

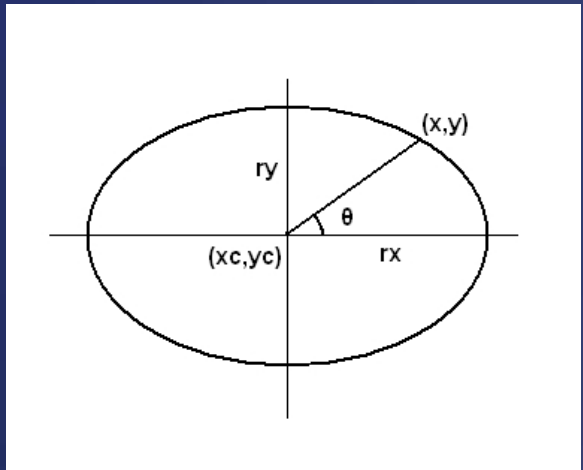
•Μπορούμε να περιορίσουμε τη χρήση των παραμετρικών εξισώσεων στο ένα όγδοο του κύκλου.Αναπαραγάγουμε τα υπόλοιπα σημεία με σχέσεις συμμετρίας.



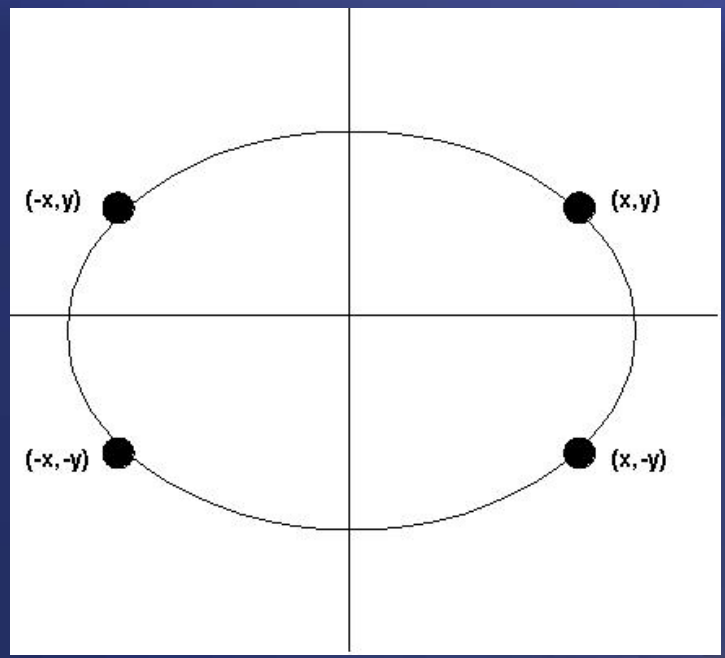
Σχεδίαση έλλειψης

$$x = x_c + r_x \cdot \cos\theta$$
$$y = y_c + r_y \cdot \sin\theta$$
$$0 \le \theta \le 2\pi$$

r_x, r_y: μήκος των αξόνων της έλλειψης



Συμμετρίες έλλειψης



Προώθηση εντολών προς εκτέλεση (flushing)

•Σε ένα υπολογιστικό σύστημα, οι εντολές εκτελούνται κατά ομάδες, αφού ολοκληρωθεί η συλλογή ενός πλήθους εντολών.

•Ωστόσο, για να σχεδιάστει ένα καρέ, ο προγραμματιστής θα πρέπει να είναι βέβαιος ότι, όσες εντολές έχουν δηλωθεί, προωθούνται προς εκτέλεση.

•Στην OpenGL ια να προωθήσουμε την εκτέλεση εντολών που εκκρεμούν, χρησιμοποιούμε την εντολής glFlush.

```
void glFlush( );
```

•Η εντολή glFlush πρέπει να εκτελείται κάθε φορά που ολοκληρώνουμε την περιγραφή του σκηνικού. Την τοποθετούμε στο τέλος της συνάρτησης που περιέχει τις εντολές σχεδίασης (display).

Όψεις πολυγώνων

•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.

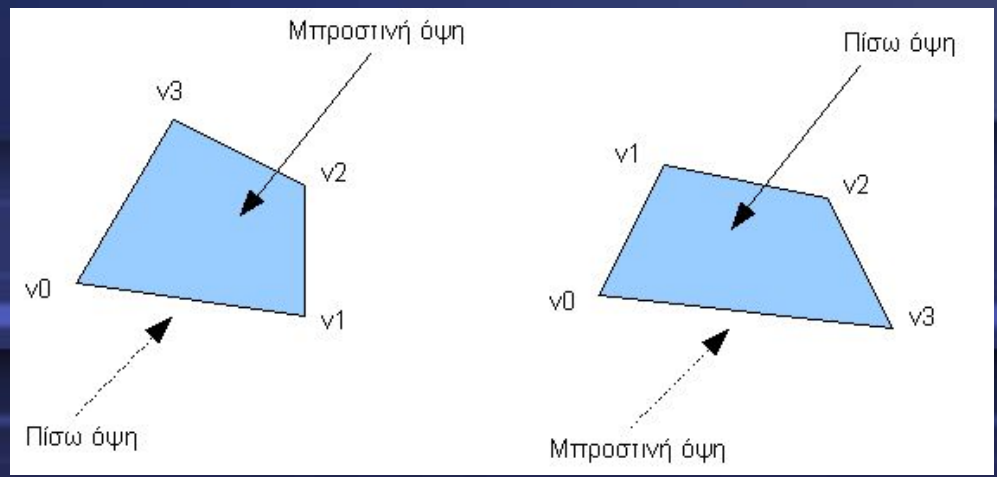
•Η διάκριση αυτή είναι χρήσιμη στην περίπτωση που θέλουμε να αποδώσουμε διαφορετικά χαρακτηριστικά σε κάθε μία όψη μιας επιφάνειας (π.χ. διαφορετική υφή).

Όψεις πολυγώνων

•Ο προσανατολισμός της επιφάνειας ενός πολυγώνου καθορίζεται ανάλογα με τη φορά δήλωσης των κορυφών του:

α) 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.

β) Εάν ο θεατής δηλώσει τις κορυφές του πολυγώνου με δεξιόστροφη φορά από τη δική του οπτική γωνία τότε, η ορατή σε αυτόν πλευρά θα είναι η πίσω.



Ρύθμιση όψεων πολυγώνων

•Η σύμβαση στη δήλωση όψεων πολυγώνων μπορεί να μεταβληθεί από τον προγραμματιστή με την εντολή glFrontFace:

void glFrontFace(GLenum mode);

mode δέχεται τις συμβολικές σταθερές:

GL_CCW: Η ορατή στο θεατή όψη χαρακτηρίζεται ως μπροστινή όψη (front facing polygon) εάν οι κορυφές του πολυγώνου δηλωθούν κατά τη θετική φορα από την οπτική γωνία του θεατή (counterclockwise).

GL_CW: Η ορατή στο θεατή όψη χαρακτηρίζεται ως μπροστινή όψη εάν οι κορυφές του πολυγώνου δηλωθούν κατά την αρνητική φορα από την οπτική γωνία του θεατή (clockwise).

Τροποποίηση σχεδίασης πολυγώνων

•Κάθε όψη ενός πολυγώνου (μπροστινή και πίσω) μπορεί να σχεδιαστεί με διαφορετικό τρόπο.

•Με την αρχική ρύθμιση της OpenGL, και οι μπροστινές και οι πίσω όψεις σχεδιάζονται συμπαγείς.

•Η ρύθμιση του τρόπου σχεδίασης κάθε όψης δηλώνεται με την εντολή glPolygonMode.

```
void glPolygonMode(GLenum face, GLenum mode);
```

Αλλαγή σχεδίασης όψεων πολυγώνων

void glPolygonMode(GLenum face, GLenum mode);

face – η όψη της οποίας το σχεδιασμό τροποποιούμε:
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.

mode – τρόπος σχεδιασμού της επιλεγόμενης όψης:
GL_FILL: Η όψη σχεδιάζεται συμπαγής.
GL_LINE: Σχεδιάζεται μόνο το περίγραμμα της όψης.
GL_POINT: Σχεδιάζονται μόνο οι κορυφές της όψης.

Παραδείγματα σχεδίασης όψεων πολυγώνων

glPolygonMode(GL_FRONT,GL_LINE);
glPolygonMode(GL_BACK,GL_POINT);
Σχεδίαση των μπροστινών όψεων ως περιγράμματα
Σχεδίαση κορυφών των πίσω όψεων

glPolygonMode(GL_FRONT,GL_LINE);
Σχεδίαση των μπροστινών όψεων ως περιγράμματα
Σχεδίαση των πίσω όψεων κατά την προηγουμένως ισχύουσα ρύθμιση

Καταστολή όψεων πολυγώνων (1)

•Όταν σε μια σκηνή είναι όρατό μόνο ένα από τα δύο είδη όψεων πολυγώνων, συμφέρει να αγνοήσουνε το σχεδιασμό των μή ορατών όψεων.

•Η δυνατότητα καταστολής όψεων πολυγώνων αρχικά πρέπει να ενεργοποιηθεί με την εντολή glEnable

```
glEnable(GL_CULL_FACE);
```

•Οι όψεις που καταστέλλονται δηλώνονται με την εντολή glCullFace.

```
void  glCullFace(GLenum mode);
```

Καταστολή όψεων πολυγώνων (2)

void glCullFace(GLenum mode);

mode: δέχεται τις συμβολικές σταθερές
GL_FRONT: καταστολή μπροστινών όψεων
GL_BACK: καταστολή πίσω όψεων
GL_FRONT_AND_BACK: καταστολή και των δύο ειδών όψεων

Πχ
glCullFace(GL_BACK); Καταστολή πίσω όψεων

•Ενεργοποίηση της καταστολής όψεων με την glEnable χωρίς χρήση της glCullFace αυτομάτως ενεργοποιεί την καταστολή των πίσω όψεων.

Ομάδες ιδιοτήτων (attribute groups)

Η χρήση πολλαπλών εντολών επισκόπησης μεταβλητών κατάστασης (glGetFloatv glGetDoublev κοκ) για την ανάκτηση ενός συνόλου τους είναι κουραστική για τον προγραμματιστή.

Στην OpenGL οι ιδιότητες κατάστασης έχουν ομαδοποιηθεί σε ομάδες.

Με τη χρήση μίας μόνο εντολή εκτελείται η αποθήκευση όλων των ιδιοτήτων μιας ομάδας σε μία θέση μνήμης για μελλοντική χρήση

Οι ομάδες ιδιοτήτων αποθηκεύονται στη στοίβα ιδιοτήτων (attribute stack).

Ορισμένες ομάδες ιδιοτήτων

α) Ομάδα ιδιοτήτων σημείου:

Περιέχει παραμέτρους που καθορίζουν την εμφάνιση ενός σημείου (π.χ. Πάχος σημείου)

β) Ομάδα ιδιοτήτων γραμμών:

Περιέχει παραμέτρους που καθορίζουν την εμφάνιση γραμμών (π.χ. Πάχος γραμμής, διάστιξη γραμμής)

γ) Ομάδα ιδιοτήτων πολυγώνων:

Εμπεριέχει όλες τις παραμέτρους που επηρεάζουν τη σχεδίαση πολυγώνων.

δ) Χρώμα:

Το χρώμα εντάσσεται στη δική του "ομάδα ιδιοτήτων".

Αποθήκευση ομάδων ιδιοτήτων

Οι μεταβλητές μιας ομάδας αποθηκεύονται στη στοίβα ιδιοτήτων με την εντολή glPushAttrib:

void glPushAttrib(attributeGroup);

attributeGroup καθορίζει την ομάδα ιδιοτήτων
GL_POINT_BIT: ομάδα ιδιοτήτων σημείου
GL_LINE_BIT: ομάδα ιδιοτήτων γραμμής
GL_POLYGON_BIT: ομάδα ιδιοτήτων πολυγώνου
GL_CURRENT_BIT: "ομάδα ιδιοτήτων" χρώματος

Πχ
glPushAttrib(GL_CURRENT_BIT);
Αποθήκευση τρέχοντος χρώματος σχεδίασης στη στοίβα ιδιοτήτων.

Ανάκληση ομάδων ιδιοτήτων

Οι τιμές ιδιοτήτων που αποθηκεύτηκαν στη στοίβα ιδιοτήτων ανακαλούνται με την εντολή glPopAttrib:

void glPopAttrib( );

Η glPopAttrib:

α) ανακαλεί όλες τις τιμές των μεταβλητών κατάστασης που αποθηκεύτηκαν στο παρελθόν στη στοίβα ιδιοτήτων με εντολές glPushAttrib.

β) ανακαλεί μόνο τις τιμές των μεταβλητών κατάστασης που έχουν αποθηκευτεί στο παρελθόν στη στοίβα ιδιοτήτων.

Λίστες απεικόνισης (display lists)

•Η περιγραφή ενός σύνθετου γεωμετρικού σήματος είναι βολικό να περικλείεται σε μια αυτόνομη ενότητα κώδικα, τη λίστα απεικόνισης (display list).

•Η λίστα απεικόνισης εκτελείται κάθε φορά που θέλουμε να σχεδιάσουμε το σύνθετο σχήμα. Δίνει λοιπόν τη δυνατότητα επεναχρησιμοποίησης κώδικα.

Αναγνωριστικός λίστας απεικόνισης

•Σε κάθε λίστα απεικόνισης καταχωρείται ένας ακέραιος αναγνωριστικός αριθμός (list identifier)

•Δημιουργούμε αναγνωριστικούς αριθμούς με την εντολή glGenLists.

•GLuint glGenLists(GLint range);

•range: το πλήθος των αναγνωριστικών ακεραίων που θα δημιουργήσουμε

Π.χ.
ListID = glGenLists(3);
Δημιουργία τριών αναγνωριστικών με τιμές listID, listID+1, listID+2

Δημιουργία λίστας απεικόνισης

•Μια λίστα απεικόνισης ορίζεται μεταξύ δύο εντολών: των glNewList και glEndList.

void glNewList(GLuint listID, GLenum listMode);

listID: ο αναγνωριστικός που αποδίδουμε στη λίστα απεικόνισης

listMode: δέχεται τις σταθερές
GL_COMPILE: για απλή δήλωση του κώδικα σχεδιασμού
GL_COMPILE_AND_EXECUTE: για δήλωση και εκτέλεση του κώδικα σχεδιασμού

glNewList(...);
Εντολές δήλωσης σχήματος
glEndList();

Μεταξύ των εντολών glNewList και glEndList ορίζουμε το σύνθετο γεωμετρικό σχήμα χρησιμοποιώντας τις εντολές σχεδιάσης σχημάτων που προαναφέρθησαν.

Εκτέλεση/Διαγραφή λίστας απεικόνισης

•Μια λίστα απεικόνισης εκτελείται δίνοντας το αναγνωριστικό της στην εντολή glCallList:

•void glCallList(GLuint listID);

•Οι λίστες απεικόνισης διαγράφονται με την εντολή glDeleteLists. (Οι αναγνωριστικοί αριθμοί τους αποδεσμεύονται.)

glDeleteLists(startId, nLists);

•startID: ο αναγνωριστικός της πρώτης λίστας που διαγράφεται
•nLists: το πλήθος των λιστών που διαγράφονται.

Π.χ.
glDeleteLists(someListID,3);
Διαγράφουμε τις λίστες απεικόνισης με αναγνωριστικούς αριθμούς someListID, someListId+1 και someListId+2.

Λίστες απεικόνισης και μεταβλητές κατάστασης

•Εάν μεταβάλλουμε μέσα σε μια display list 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.

•Είναι χρήσιμο ο προγραμματιστής να αποθηκεύσει τις τιμές των ιδιοτήτων που θα μεταβληθούν στη στοίβα ιδιοτήτων, πριν από την κλήση της λίστας, ούτως ώστε να τις επαναφέρει μετά το πέρας της εκτέλεσης της λίστας.

Μητρώα κορυφών - Μητρώα χρωμάτων

•Ο σχεδιάσμός μεγάλου πλήθους σχημάτων αποκλειστικά με τη χρήση της δομής glBegin/glEnd απαιτεί έναν υπερβολικά μεγάλο αριθμό εντολών

•Για να διευκολύνει τη σχεδίαση πολλαπλών σχημάτων, η OpenGL παρέχει την τεχνική των μητρώων κορυφών (vertex arrays).

•Με τα μητρώα σημείων σχεδιάζουμε ένα μεγάλο αριθμό σχημάτων δηλώνοντας απλώς ένα μητρώο κορυφών και τον είδος των σχηματων που αυτές ορίζουν.

•Η ίδια ιδέα επεκτείνεται και στην απόδοση χρωματικών τιμών σε μεγάλο πλήθος κορυφών. Στην περίπτωση αυτή ορίζουμε τα μητρώα χρωμάτων (color arrays).

Ενεργοποίηση χρήσης μητρώου κορυφών/χρωμάτων

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

void glEnableClientState(GLenum array);

array: δηλώνει την κατηγορία των δεδομένων που ομαδοποιούνται στο μητρώο

GL_VERTEX_ARRAY: ομαδοποίηση συντεταγμένων πολλαπλών κορυφών

GL_COLOR_ARRAY: ομαδοποίηση χρωματικών τιμών πολλαπλών κορυφών.

GL_TEXTURE_ARRAY: αναλύεται στο Κεφάλαιο "Απόδοση υφής"

Ομαδοποίηση κορυφών (1)

Βήμα 1ο: Ενεργοποιούμε τη χρήση μητρώων κορυφών με την glEnableClientState.

glEnableClientState(GL_VERTEX_ARRAY);

Βήμα 2ο: Καταχωρούμε το μητρώο κορυφών με την εντολή glVertexPointer.

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

vertexPointer: δείκτης στο μητρώο κορυφών

size: το πλήθος των τιμών που προσδιορίζουν τις συντεταγμένες καθεμιάς κορυφής.
Για διδιάστατες κορυφές: 2
Για τρισδιάστατες κορυφές: 3

Ομαδοποίηση κορυφών (2)

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

type: ο πρωτογενής τύπος δεδομένων με τον οποίο αποθηκεύονται οι συντεταγμένων στο μητρώο vertexPointer.
Αριθμητικές σταθερές: GL_INT, GL_SHORT, GL_FLOAT, GL_DOUBLE

stride: η απόσταση μεταξύ των συντεταγμένων διαδοχικών σημείων στο μητρώο. Χρησιμοποιείται μόνο σε όταν αποθηκεύονται τιμές πολλών ιδιοτήτων στο ίδιο μητρώο (πχ διαδοχική αποθήκευση συντεταγμένων και χρώμάτων σε ένα μητρώο) Για μητρώα που περιέχουν μόνο συντεταγμένες κορυφών χρησιμοποιείται η τιμή 0.

Ομαδοποίηση κορυφών (2)

Βήμα 2ο: Ορίζουμε τη διαδοχή με την οποία χρησιμοποιούνται τα σημεία του μητρώου κορυφών. Η διαδοχή ορίζεται σε ένα μητρώο δεικτών.

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

•Το μητρώο δεικτών ορίζει τη σειρά χρήσης των σημείων του μητρώου κορυφών. Η κορυφή που δηλώνεται στη θέση i_1 του μητρώου κορυφών χρησιμοποιείται πρώτη, η κορυφή που δηλώνεται στη θέση i_2 του μητρώου κορυφών χρησιμοποιείται δεύτερη κ.ο.κ.

Ομαδοποίηση κορυφών (3)

Βήμα 3ο: Εκτελούμε τη σχεδίαση των σχημάτων με την εντολή glDrawElements:

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

indices: δείκτης στο μητρώο δεικτών

mode: καθορίζει τι σχήματα ορίζουν οι κορυφές. (GL_LINES, GL_TRIANGLES, GL_QUADS κ.λ.π.).

count: το πλήθος των κορυφών που περιέχονται στο μητρώο κορυφών.

type: ο πρωτογενής τύπος δεδομένων των τιμών στο μητρώο δεικτών. Δεκτές αριθμητικές σταθερές: GL_UNSIGNED_BYTE, GL_UNSIGNED_SHORT, GL_UNSIGNED_INT.

Ομαδοποίηση χρωματικών τιμών

•Η ομαδοποίηση χρωματικών τιμών εκτελείται με ακριβώς τον ίδιο τρόπο.

glEnableClientState(GL_COLOR_ARRAY);

•Το μητρώο χρωματικών τιμών καταχωρείται με την εντολή glColorPointer.

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

size:το πλήθος των χρωματικών συνιστωσών
Για το μοντέλο RGB: 3
Για το μοντέλο RGBA: 4

•Ορίζουμε ένα μητρώο δεικτών και δίνουμε την εντολή σχεδίασης με την glDrawElements όπως προηγουμένως.

Τέλος ενότητας!