ΑΝΩΤΑΤΟ ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ

ΘΕΣΣΑΛΟΝΙΚΗΣ

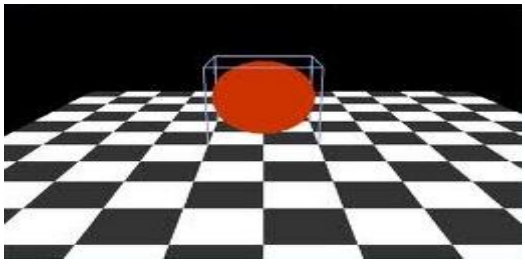
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Η Βιβλιοθήκη Γραφικών OpenGl

Με Χρήση C/C++ , JAVA και PYTHON



ΣΠΟΥΔΑΣΤΗΣ : Κουρδής Ανέστης

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ : Ράπτης Πασχάλης

ΘΕΣΣΑΛΟΝΙΚΗ 2012

Περιεχόμενα

1. ΕΙΣΑΓΩΓΗ

2. Η ΒΙΒΛΙΟΘΗΚΗ ΓΡΑΦΙΚΩΝ OpenGl

3. Η JoGl

4. Η PyOpenGl

5 . ΠΑΡΑΔΕΙΓΜΑΤΑ ΕΦΑΡΜΟΓΩΝ OPENGL

6. ΒΙΒΛΙΟΓΡΑΦΙΑ

1. ΕΙΣΑΓΩΓΗ

Οι σημειώσεις αυτές αποτελούν την παρουσίαση της βιβλιοθήκης γραφικών OpenGl,στο πλαίσιο της πτυχιακής εργασίας με τίτλο «Η βιβλιοθήκη γραφικών OpenGL με χρήση C/C++, Java και Python».

Στόχος αυτής της πτυχιακής εργασίας είναι η εκμάθηση και παρουσίαση της δομής και λειτουργίας του γραφικού API OpenGL. Η OpenGL είναι μια βιβλιοθήκη (αρχιτεκτονικής client – server) γραφικών ανοιχτού κώδικα (open source), ανεξάρτητης πλατφόρμας (cross-platform Windows, Unix-like), κατάλληλη για δημιουργία 2D και 3D γραφικών. Η γλώσσα C/C++ είναι απαραίτητη για την επικοινωνία/χρήση της OpenGL, η γλώσσα Java είναι απαραίτητη για την επικοινωνία/χρήση της JoGL (η Java εκδοχή της OpenGL) και η γλώσσα Python είναι απαραίτητη για την επικοινωνία/χρήση της PyOpenGL (η Python εκδοχή της OpenGL). Θα παρουσιαστούν προγράμματα σε C++, Java και Python,που θα αποτυπώνουν τις δυνατότητες της GL.

Τα εργαλεία που χρησιμοποιήθηκαν ,είναι η υπηρεσία Google Translate,ως υποστήριξη για τη μετάφραση ξενόγλωσσων βιβλίων και Documentations,τα Visual C++ 2008 Express Edition και Dev C++ για τη δημιουργία των Projects της C++ με χρήση της OpenGl ,το Eclipse Europa 3.3 για τη δημιουργία του Project της Java με χρήση της OpenGl,ενώ για το Project της Python με OpenGl,χρησιμοποιήθηκε η γραμμή εντολών των Microsoft Windows 7 και το Eclipse INDIGO,ρυθμισμένο σε λειτουργία PyDev για Python.

2.Η ΒΙΒΛΙΟΘΗΚΗ ΓΡΑΦΙΚΩΝ OpenGl

Τι είναι η OpenGl

Ξεκινώντας, είναι βασικό να αναφερθεί πως σε αυτές τις σημειώσεις, η θεωρητική παρουσίαση και εκμάθηση της βιβλιοθήκης OpenGl,θα γίνει σχεδόν αποκλειστικά με χρήση της γλώσσας C++,η οποία μοιάζει περισσότερο απο οποιαδήποτε με τη γλώσσα C, πάνω στην οποία γράφτηκε και χρησιμοποιήθηκε για πρώτη φορά η συγκεκριμένη βιβλιοθήκη γραφικών.

Η OpenGL είναι μια διασύνδεση λογισμικού σε υλικό γραφικών. Αυτή η διασύνδεση αποτελείται από περισσότερες από 700 διαφορετικές εντολές (περίπου 670 εντολές, όπως ορίζεται για την OpenGL έκδοση 3.0 και άλλες 50 στη Βιβλιοθήκη Utility OpenGL) που χρησιμοποιούνται ώστε να προσδιοριστούν τα αντικείμενα και οι πράξεις που απαιτούνται για την παραγωγή διαδραστικών τρισδιάστατων εφαρμογών.

Η OpenGL έχει σχεδιαστεί ως μία βελτιωμένη, ανεξάρτητη διεπαφή, που θα εφαρμοστεί σε πολλές διαφορετικές πλατφόρμες hardware και λειτουργικών συστημάτων. Για την επίτευξη αυτών των δυνατοτήτων, δεν χρησιμοποιούνται συγκεκριμένες εντολές για την εκτέλεση των καθηκόντων παραθύρων ή την απόκτηση εισόδου του χρήστη, αντ 'αυτού, στην OpenGl πρέπει να εργαστείτε με οποιοδήποτε σύστημα παραθύρων ελέγχει το συγκεκριμένο υλικό που χρησιμοποιείτε. Ομοίως, η OpenGL δεν παρέχει υψηλού επιπέδου εντολές για την περιγραφή των μοντέλων των τρισδιάστατών αντικειμένων. Οι εν λόγω εντολές θα μπορούσαν να επιτρέψουν σε σας να καθορίσετε σχετικά πολύπλοκα σχήματα, όπως μια αυτοκινητοβιομηχανία, τα μέρη του σώματος, ένα αεροπλάνο, ή μόρια. Με την OpenGL, θα πρέπει να "χτίσετε" το επιθυμητό μοντέλο σας από ένα μικρό σύνολο γεωμετρικών σημείων, γραμμών και πολυγώνων. Μια εκλεπτυσμένη εφαρμογή, η οποία παρέχει αυτά τα χαρακτηριστικά, θα μπορούσε σίγουρα να χτιστεί με χρήση της OpenGL. Η OpenGL Βιβλιοθήκη (GLU) παρέχει πολλά από τα χαρακτηριστικά μοντέλων, όπως quadric επιφάνειες και NURBS καμπύλες και επιφάνειες. Η GLU είναι ένα τυπικό μέρος της κάθε εφαρμογής OpenGL.

Τώρα που ξέρετε τι δεν μπορεί η OpenGL να κάνει, μπορείτε να κατανοήσετε τι είναι αυτό που μπορεί να κάνει.

Η OpenGl λειτουργεί με διάφορης πολυπλοκότητας τρόπους και στάδια, από υπολογιστή σε υπολογιστή. Κατά τη διάρκεια αυτών των σταδίων, η OpenGL μπορεί να εκτελεί άλλες εργασίες, όπως να εξαλείφει τα μέρη των αντικειμένων που είναι κρυμμένα από άλλα αντικείμενα. Επιπλέον, αφού η σκηνή είναι rasterized αλλά πριν εμφανιστεί κάτι στην οθόνη, μπορείτε να εκτελέσετε κάποιες λειτουργίες στα pixel των δεδομένων, αν θέλετε. Σε ορισμένες εφαρμογές (όπως με το σύστημα X Window), η OpenGL έχει σχεδιαστεί για να λειτουργεί ακόμα και αν ο υπολογιστής που εμφανίζει τα γραφικά που δημιουργείτε δεν είναι ο υπολογιστής που τρέχει το πρόγραμμα γραφικών σας. Αυτό θα μπορούσε να συμβεί

εάν εργάζεστε σε περιβάλλον δικτύου ηλεκτρονικών υπολογιστών, όπου πολλοί υπολογιστές συνδέονται μεταξύ τους σε ένα δίκτυο. Σε αυτήν την περίπτωση, ο υπολογιστής ο οποίος τρέχει το πρόγραμμά σας και το περιεχόμενο των OpenGL εντολών σχεδίασης, καλείται "Πελάτης", και ο υπολογιστής που δέχεται τις εντολές και εκτελεί το σχέδιο, ονομάζεται "Διακομιστής". Η μορφή για τη διαβίβαση των OpenGL εντολών (που ονομάζεται πρωτόκολλο) από τον πελάτη προς τον διακομιστή είναι πάντα η ίδια, έτσι τα OpenGL προγράμματα μπορούν να εργαστούν σε ένα δίκτυο, ακόμη και αν ο πελάτης και ο διακομιστής, είναι διαφορετικά είδη υπολογιστών. Εάν ένα πρόγραμμα OpenGL δεν λειτουργεί μέσω δικτύου, τότε υπάρχει μόνο ένας υπολογιστής και είναι ο πελάτης ή ο διακομιστής.

Μια μικρή δόση OpenGL Κώδικα

Επειδή μπορείτε να κάνετε τόσα πολλά πράγματα με το σύστημα γραφικών OpenGL, ένα OpenGL πρόγραμμα μπορεί να είναι περίπλοκο. Ωστόσο, η βασική δομή ενός χρήσιμου προγράμματος μπορεί να είναι απλή. Τα καθήκοντά του είναι να προετοιμάσει ορισμένα τμήματα που ελέγχουν τον τρόπο που η OpenGL θα τα αναπαραστήσει και να καθορίσει τα αντικείμενα που θα παρασχεθούν. Πριν εξετάσουμε αυτό τον μικρό κώδικα OpenGL,καλό θα ήταν να γίνει μια μικρή αναφορά κάποιων όρων. "Rendering", είναι η διαδικασία με την οποία ένας υπολογιστής δημιουργεί εικόνες από τα "μοντέλα". Αυτά τα "μοντέλα", ή "αντικείμενα", είναι κατασκευασμένα από τα γεωμετρικά πρότυπα, τις γραμμές και τα πολύγωνα, που ορίζονται από τις κορυφές τους. Το τελικό αποτέλεσμα στην οθόνη, αποτελείται από pixels που εμφανίζονται σ'αυτή. Ένα pixel είναι το μικρότερο ορατό στοιχείο που μπορεί να τεθεί στην οθόνη.

Οι πληροφορίες σχετικά με τα pixels (για παράδειγμα, τι χρώμα είναι) οργανώνονται στη μνήμη σε bitplanes. Ένα bitplane είναι μια περιοχή μνήμης που κατέχει ένα κομμάτι των πληροφοριών για κάθε pixel στην οθόνη. Το κομμάτι θα μπορούσε να αναφέρεται σε ένα συγκεκριμένο pixel . Τα bitplanes είναι οι πληροφορίες , οργανωμένες σε ένα framebuffer, ο οποίος κατέχει όλες τις πληροφορίες που χρειάζεται η οθόνη για τον έλεγχο των γραφικών της, όπως το χρώμα και η ένταση όλων των pixels στην οθόνη.

Τώρα ας δούμε αυτό το μικρό πρόγραμμα του παραδείγματος 1-1, το οποίο σχεδιάζει ένα λευκό ορθογώνιο σε μαύρο φόντο, όπως φαίνεται στο Σχήμα 1-1

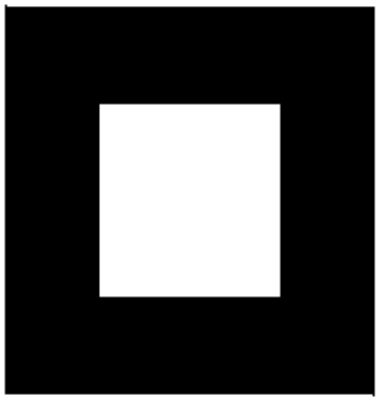


Figure 1-1 White Rectangle on a Black Background

Example 1-1 Chunk of OpenGL Code

```
#include <whateverYouNeed.h>
main() {
InitializeAWindowPlease();
glClearColor(0.0, 0.0, 0.0, 0.0);
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0, 1.0, 1.0);
glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
glBegin(GL_POLYGON);
glVertex3f(0.25, 0.25, 0.0);
glVertex3f(0.75, 0.25, 0.0);
glVertex3f(0.75, 0.75, 0.0);
glVertex3f(0.25, 0.75, 0.0);
glEnd();
```

```
glFlush();

UpdateTheWindowAndCheckForEvents();

}
```

Η main() αρχικοποιεί ένα παράθυρο στην οθόνη:

Η InitializeAWindowPlease () ρουτίνα νοείται ως ένα σύμβολο κράτησης θέσης για window-system , συγκεκριμένες ρουτίνες, οι οποίες συνήθως δεν είναι κλήσεις της OpenGL.

Οι επόμενες δύο γραμμές είναι OpenGL εντολές που ορίζουν το παράθυρο σε μαύρο: Η glClearColor () καθορίζει ποιο είναι το χρώμα του παραθύρου και η glClear () ανοίγει το παράθυρο. Μόλις το χρώμα οριστεί, το παράθυρο προβάλλεται με αυτό το χρώμα κάθε φορά που η glClear () καλείται. Αυτός ο ο ορισμός του χρώματος μπορεί να αλλάξει με μια νέα κλήση της glClearColor (). Ομοίως, η glColor3f () εντολή καθορίζει το χρώμα που θα χρησιμοποιηθεί για τα αντικείμενα σχεδίασης-σε αυτή την περίπτωση, το χρώμα είναι λευκό. Όλα τα αντικείμενα που μετά από αυτό το σημείο χρησιμοποίησαν αυτό το χρώμα, το διατηρούν, μέχρι να αλλάξει με μια άλλη κλήση για να οριστεί άλλο χρώμα γι' αυτά.Η επόμενη εντολή OpenGL που χρησιμοποιείται στο πρόγραμμα, glOrtho (), καθορίζει το σύστημα συντεταγμένων OpenGL και αναλαμβάνει την τελική θέση της εικόνας στην οθόνη. Οι επόμενες κλήσεις, που παρεμβάλλονται μεταξύ glBegin () και glEnd (), καθορίζουν τα αντικείμενα που πρέπει να συναχθούν-σε αυτό το παράδειγμα, ένα πολύγωνο με τέσσερις κορυφές. Το πολύγωνο είναι "γωνίες" που ορίζονται από τις glVertex3f () εντολές.Συγκεκριμένα,οι (x, y, z) είναι οι συντεταγμένες και το πολύγωνο είναι ένα ορθογώνιο στο z = 0.Τέλος,η glFlush () εξασφαλίζει ότι οι εντολές σχεδίασης πράγματι θα εκτελεστούν και αντί να αποθηκεύεται σε ένα buffer, αναμένει πρόσθετες εντολές OpenGL. Η UpdateTheWindowAndCheckForEvents () ρουτίνα κράτησης θέσης διαχειρίζεται το περιεχόμενο του παραθύρου και αρχίζει την επεξεργασία των γεγονότων. Στην πραγματικότητα, αυτό το κομμάτι του κώδικα OpenGL δεν είναι καλά δομημένο. Μπορεί να αναρωτηθείτε, "Τι θα συμβεί αν προσπαθήσω να μετακινήσω ή να αλλάξω το μέγεθος του παραθύρου;» ή «Χρειάζεται να κάνω επαναφορά του συστήματος συντεταγμένων κάθε φορά που θέλω να σχεδιάσω το ορθογώνιο; ".Αργότερα, θα παρουσιαστούν ξανά οι δύο InitializeAWindowPlease () και UpdateTheWindowAndCheckForEvents () που λειτουργούν πραγματικά και πιο αποτελεσματικά.

OpenGL σύνταξη μιας εντολής

Από το απλό πρόγραμμα που παρουσιάστηκε φαίνεται πως οι OpenGL εντολές χρησιμοποιούν το πρόθεμα gl και αρχικά κεφαλαία γράμματα για κάθε λέξη που συνθέτουν το όνομα της εντολής (κλήση glClearColor (), για παράδειγμα). Ομοίως, στην OpenGL ορίζονται σταθερές που ξεκινούν με GL_, με χρήση όλων των κεφαλαίων γραμμάτων, και τη χρήση υπογραμμίσεων για το διαχωρισμό λέξεων (για παράδειγμα, GL_COLOR_BUFFER_BIT). Μπορεί επίσης να παρατηρήθηκαν κάποια φαινομενικά ξένα γράμματα που προσαρτώνται σε μερικά ονόματα των εντολών (για παράδειγμα, η 3στ σε glColor3f () και glVertex3f ()).Είναι αλήθεια ότι το τμήμα Χρώματος του glColor3f ονόματος της εντολής () είναι αρκετό για να καθορίσει την εντολή ώστε μέσω αυτής να καθορίζεται και το τρέχον χρώμα. Ωστόσο, έχουν οριστεί περισσότερες από μια τέτοιες εντολές, έτσι ώστε να μπορείτε να χρησιμοποιήσετε μεγαλύτερη ποικιλία ορισμάτων. Ειδικότερα, το τρίτο μέρος του επιθήματος δείχνει ότι είναι δοσμένα τρία ορίσματα. Μια άλλη έκδοση της εντολής Color δέχεται τέσσερα ορίσματα. Το μέρος "f" του επιθήματος δείχνει ότι τα επιχειρήματα είναι floatingpoint αριθμοί. Έχοντας διαφορετικές μορφές OpenGL,με αυτό τον τρόπο επιτρέπονται τα δεδομένα του χρήστη σε δική του μορφή δεδομένων. Ορισμένες εντολές OpenGL δέχονται ως και οκτώ διαφορετικούς τύπους δεδομένων για τα ορίσματα τους. Τα γράμματα που χρησιμοποιούνται ως επιθήματα χρησιμοποιούνται για να προσδιορίσουν αυτούς τους τύπους δεδομένων. Οι ISO υλοποιήσεις της C OpenGL φαίνονται στον Πίνακα 1-1, μαζί με τον αντίστοιχο τύπο OpenGL. Η συγκεκριμένη εφαρμογή της OpenGL που χρησιμοποιείται μπορεί να μην ακολουθεί το συγκεκριμένο μοτίβο. Μια υλοποίηση σε C + + ή Ada που υποστηρίζει τη λειτουργία της υπερφόρτωσης, για παράδειγμα, δεν θα ήταν κατ 'ανάγκη απαραίτητη.

Suffix	Data Type	Typical Corresponding C-Language Type	OpenGL Type Definition
b	8-bit integer	signed char	GLbyte
s	16-bit integer	short	GLshort
i	32-bit integer	int or long	GLint, GLsizei
f	32-bit floating-point	float	GLfloat, GLclampf
d	64-bit floating-point	double	GLdouble, GLclampd
ub	8-bit unsigned integer	unsigned char	GLubyte, GLboolean
us	16-bit unsigned integer	unsigned short	GLushort
ui	32-bit unsigned integer	unsigned int or unsigned long	GLuint, GLenum, GLbitfield

Table 1-1 Command Suffixes and Argument Data Types

Έτσι, οι δύο εντολές

glVertex2i (1, 3)

glVertex2f (1.0, 3.0)

είναι ισοδύναμες, εκτός από το ότι η πρώτη καθορίζει τις συντεταγμένες στην κορυφή ως 32-bit ακέραιους, και η δεύτερη τους προσδιορίζει ως απλής ακρίβειας floatingpoint αριθμούς.

Σημείωση: Οι εφαρμογές στην OpenGL έχουν ελευθερία όταν τα δεδομένα C τύπου, χρησιμοποιούνται για να εκπροσωπήσουν OpenGL τύπους δεδομένων. Εάν χρησιμοποιείτε αναλυτικά την OpenGL για να ορίσετε τύπους δεδομένων σε όλη την εφαρμογή σας, θα αποφύγετε συγχύσεις σε τύπους όταν χρησιμοποιείται τον κώδικά σας μεταξύ διαφορετικών υλοποιήσεων.

Ορισμένες εντολές OpenGL μπορεί να πάρουν ένα τελικό γράμμα v, γεγονός που δείχνει ότι η εντολή παίρνει ένα δείκτη σε ένα διάνυσμα (ή array) , αντί για μια σειρά των επιμέρους ορισμάτων. Πολλές εντολές έχουν και οι vector και οι nonvector εκδόσεις, αλλά ορισμένες εντολές δέχονται μόνο μεμονωμένα επιχειρήματα ενώ άλλες απαιτούν ότι τουλάχιστον

ορισμένα από τα ορίσματα, θα καθορίζουν το διάνυσμα. Οι ακόλουθες γραμμές δείχνουν πώς μπορείτε να χρησιμοποιήσετε ένα διάνυσμα και μια nonvector έκδοση της εντολής που καθορίζει το τρέχον χρώμα:

glColor3f (1.0, 0.0, 0.0)?

GLfloat color_array [] = {1.0, 0.0, 0.0}?

glColor3fv (color_array)?

Τέλος, η OpenGL καθορίζει το είδος της GLvoid. Αυτή πιο συχνά χρησιμοποιείται για OpenGL εντολές που δέχονται δείκτες σε συστοιχίες τιμών. Στο υπόλοιπο αυτής της παρουσίασης (εκτός από πραγματικά παραδείγματα κώδικα),οι OpenGL εντολές αναφέρονται με βάση τα ονόματά τους και μόνο. Επίσης, ένας αστερίσκος συμπεριλαμβάνεται για να δείξει ότι μπορεί να υπάρχουν περισσότερα στο όνομα εντολής. Για παράδειγμα, η glColor * () αντιπροσωπεύει όλες τις παραλλαγές των εντολών που χρησιμοποιούνται για να οριστεί το τρέχον χρώμα. Αν θέλουμε να δείξουμε ένα συγκεκριμένο σημείο για μια εκδοχή μιας συγκεκριμένης εντολής, θα περιλαμβάνεται η κατάληξη ότι είναι αναγκαίο να οριστεί η έκδοση. Για παράδειγμα, glVertex * v () αναφέρεται σε όλες τις εκδόσεις vector της εντολής που χρησιμοποιείτε για να καθορίσετε κορυφές.

Η OpenGL ως μηχανή

Η OpenGL είναι μια μηχανή, ειδικά αν χρησιμοποιείτε τη σταθερή λειτουργία αγωγού. Μπορεί να τεθεί σε διάφορες καταστάσεις που στη συνέχεια παραμένουν σε ισχύ μέχρι να τις αλλάξετε. Όπως θα έχετε ήδη παρατηρήσει, το τρέχον χρώμα είναι μια μεταβλητή κατάσταση. Μπορείτε να ορίσετε το τρέχον χρώμα σε άσπρο, κόκκινο, ή οποιοδήποτε άλλο χρώμα, και εν συνεχεία, ένα αντικείμενο διατηρεί αυτό το χρώμα μέχρι να ρυθμίσετε το χρώμα σε κάποιο άλλο. Το τρέχον χρώμα είναι μόνο μία από τις πολλές μεταβλητές κατάστασης που διατηρεί η OpenGL. Άλλες μεταβλητές, ελέγχουν πράγματα όπως η τρέχουσα αναπαράσταση και μετασχηματισμούς αναπαράστασης , γραμμές και πολύγωνα μοτίβων, τρόπους σχεδίασης πολυγώνων, συμβάσεις πακέτων από pixels, τις θέσεις και τα χαρακτηριστικά του φωτισμού, τις ιδιότητες των στοιχείων και των αντικειμένων στο στάδιο της επεξεργασίας. Πολλές μεταβλητές κατάστασης αναφέρονται σε τρόπους που ενεργοποιούνται ή απενεργοποιούνται, με την εντολή glEnable () ή glDisable ().Εάν χρησιμοποιείτε προγραμματιζόμενη σκίαση, ανάλογα με την έκδοση της OpenGL που χρησιμοποιείτε, η ποσότητα που είναι εκτεθειμένη σε σκίαση , θα ποικίλλει. Κάθε

μεταβλητή κατάστασης ή λειτουργία, έχει μια προκαθορισμένη τιμή και σε οποιοδήποτε σημείο , μπορείτε να ρωτήσετε το σύστημα για την τρέχουσα τιμή κάθε μεταβλητής. Συνήθως, μπορείτε να χρησιμοποιήσετε μία από τις έξι ακόλουθες εντολές για να το κάνετε αυτό: glGetBooleanv (), glGetDoublev (), glGetFloatv (), glGetIntegerv (), glGetPointerv (), ή glIsEnabled (). Το ποιές από αυτές τις εντολές θα επιλέξετε, εξαρτάται από αυτό τον τύπο δεδομένων που θέλετε για απάντηση . Μερικές μεταβλητές διαθέτουν μια ειδικότερη εντολή-ερώτημα (Όπως glGetLight * (), glGetError (), ή glGetPolygonStipple ()). Επιπλέον, μπορείτε να αποθηκεύσετε μια συλλογή ιδιοτήτων των μεταβλητών σε μια στοίβα, με glPushAttrib () ή glPushClientAttrib (), να τις τροποποιήσετε προσωρινά, και να ανακτήσετε αργότερα τις αξίες, με τις glPopAttrib () ή glPopClientAttrib (). Για προσωρινές αλλαγές της κατάστασης, θα πρέπει να χρησιμοποιήσετε αυτές τις εντολές και όχι τις εντολές ερωτημάτων, δεδομένου ότι είναι πιθανό να είναι πιο αποτελεσματικές.

Η OpenGL ως αγωγός - OpenGL Rendering Pipeline

Οι περισσότερες εφαρμογές της OpenGL έχουν μια παρόμοια σειρά ενεργειών, μια σειρά από στάδια επεξεργασίας που ονομάζεται OpenGL Pipeline. Αυτή η λειτουργία, όπως φαίνεται στο Σχήμα 1-2, δεν είναι ένας αυστηρός κανόνας για το πώς η OpenGL υλοποιείται, αλλά παρέχει έναν αξιόπιστο οδηγό για την πρόβλεψη για το πως η OpenGL θα λειτουργήσει. Εάν είστε νέοι στα τρισδιάστατα γραφικά, η επόμενη περιγραφή μπορεί να φαίνεται όπως το νερό από τη μάνικα της πυροσβεστικής. Το παρακάτω διάγραμμα δείχνει την Henry Ford προσέγγιση γραμμή συναρμολόγησης, η οποία χρειάζεται η OpenGL για επεξεργασία των δεδομένων. Γεωμετρικά πρότυπα (κορυφές, γραμμές και πολύγωνα) ακολουθούν το μονοπάτι μέσα από την σειρά των κιβωτίων που περιλαμβάνει αξιολογητές και ανά-κορυφή πράξεις, ενώ τα δεδομένα εικονοστοιχείων (pixels, εικόνες, και bitmaps) αντιμετωπίζονται διαφορετικά για κάθε μέρος της διαδικασίας. Και οι δύο τύποι των δεδομένων ανάγονται στους ίδιους στα τελευταία βήματα (rasterization and per-fragment operations) πριν τα τελικά στοιχεία pixel μπουν στον framebuffer.

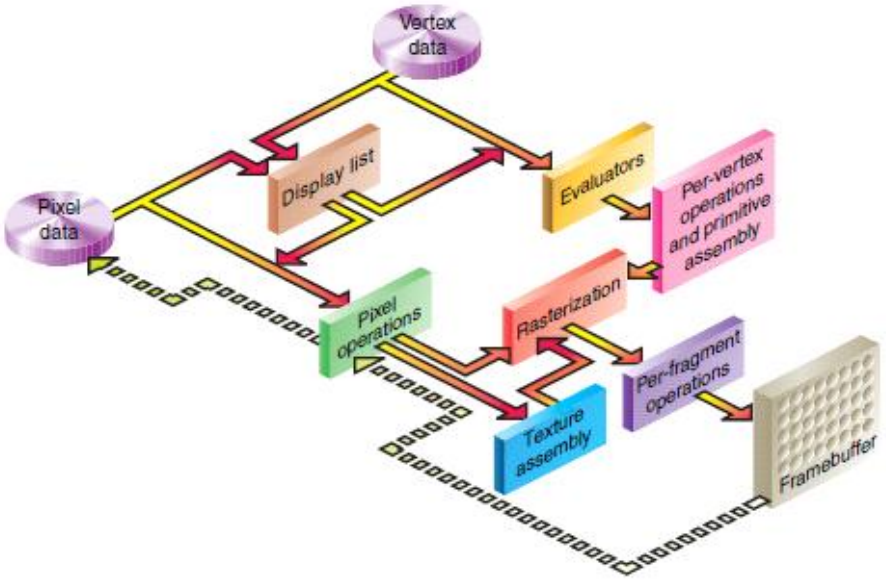


Figure 1-2 Order of Operations

Τώρα θα δείτε περισσότερες λεπτομέρειες σχετικά με τα βασικά στάδια του αγωγού OpenGL.

Λίστες Αναπαράστασης- Display Lists

Όλα τα στοιχεία, αν περιγράφουν τη γεωμετρία ή τα pixels, μπορούν να αποθηκευτούν σε μια λίστα αναπαράστασης για τρέχουσα ή μελλοντική χρήση. (Η εναλλακτική λύση για τη διατήρηση των δεδομένων σε μια λίστα αναπαράστασης είναι η επεξεργασία των δεδομένων άμεσα- γνωστή ως άμεση λειτουργία.) Όταν μια λίστα εμφανίζεται και εκτελείται, τα διατηρούμενα δεδομένα αποστέλλονται από τη λίστα στην οθόνη, ακριβώς όπως αν είχαν αποσταλεί από την εφαρμογή σε άμεση λειτουργία.

Οι αξιολογητές- Evaluators

Όλα τα γεωμετρικά πρότυπα περιγράφονται από κορυφές. Παραμετρικές καμπύλες και επιφάνειες, περιγράφοναι από τα σημεία ελέγχου, όπως και οι πολυωνυμικές συναρτήσεις που ονομάζονται basis functions. Οι αξιολογητές παρέχουν μια μέθοδο για την εξαγωγή των κορυφών και χρησιμοποιούνται για την αναπαράσταση της επιφάνειας από τα σημεία ελέγχου. Χρησιμοποιείται ένα πολυώνυμο χαρτογράφησης, που παράγει κανονική επιφάνεια, συντεταγμένες , χρώματα, αξίες και συντονισμό σημείων.

Λειτουργίες Per-Vertex

Οι λειτουργίες αυτές, μετατρέπουν τις κορυφές σε πρότυπα. Ορισμένοι τύποι δεδομένων κορυφής (για παράδειγμα, το χωροταξικό συντεταγμένων) μετατρέπονται από floating-point μήτρες. Οι συντεταγμένες ανακτώνται από μια θέση στον 3D «κόσμο» και μεταφέρονται σε μια θέση στον υπολογιστή σας στην οθόνη. Αν είναι ενεργοποιημένα τα προηγμένα χαρακτηριστικά, αυτό το στάδιο επεξεργασίας είναι ακόμα πιο φορτωμένο. Εάν χρησιμοποιείται Texturing, οι συντεταγμένες μπορούν να παραχθούν και να μετατραπούν εδώ. Αν ο φωτισμός είναι ενεργοποιημένος, οι υπολογισμοί φωτισμού με τη χρήση τους μπορούν να μετατραπούν , το φως ,την πηγή, ιδιότητες, καθώς και άλλες πληροφορίες φωτισμού για να παράγουν μια τιμή χρώματος. Από την OpenGL έκδοση 2.0, υπήρχε η δυνατότητα να χρησιμοποιούν σταθερής λειτουργίας κορυφές επεξεργασίας, όπως ακριβώς περιγράφηκε προηγουμένως, ή απόλυτο έλεγχο της λειτουργίας τους ανά κορυφή εργασιών με τη χρήση vertex shaders. Αν χρησιμοποιείτε shaders, όλες οι Per-Vertex προαναφερθείσες λειτουργίες, θα αντικατασταθούν από τον shader σας. Στην έκδοση 3.1, το σύνολο των σταθερών λειτουργιών ανα κορυφή ,έχουν αφαιρεθεί (εκτός και αν η εφαρμογή σας υποστηρίζει την GL_ARB_compatibility επέκταση) και η χρήση μιας κορυφής shader, είναι υποχρεωτική.

Primitive Assembly

Είναι η εξάλειψη των τμημάτων γεωμετρίας που δεν εμπίπτουν στο μισό χώρο, που ορίζεται από ένα plane.Η εξάλειψη ,απλά προσπερνά ή απορρίπτει .Σε ορισμένες περιπτώσεις, αυτό ακολουθείται από άποψη διαίρεσης, η οποία καθιστά μακρινά γεωμετρικά αντικείμενα να εμφανίζονται πιο κοντά από ό, τι πραγματικά πιο κοντινά αντικείμενα. Στη συνέχεια, προβάλλονται και εφαρμόζεται η συντεταγμένη βάθους (z-συντεταγμένη) . Αν είναι ενεργοποιημένη η απόρριψη και το πρότυπο είναι ένα πολύγωνο, τότε μπορεί να απορριφθεί από έναν έλεγχο απόρριψης. Ανάλογα με τη λειτουργία, ένα πολύγωνο μπορεί να εξαχθεί ως σημεία ή γραμμές. Τα αποτελέσματα αυτής της φάσης είναι πλήρη γεωμετρικά πρότυπα, τα οποία είναι μετασχηματισμένα και έχουν εξαλειφθεί οι κορυφές που σχετίζονται με το χρώμα, το βάθος, και μερικές φορές η υφή που συντονίζει τις αξίες και τις κατευθυντήριες γραμμές για το βήμα ραστεροποίησης.

Λειτουργίες Pixel

Ενώ γεωμετρικά δεδομένα περνούν από ένα μονοπάτι μέσα από την ροή του OpenGL αγωγού, τα δεδομένα pixel περνούν από μια διαφορετική διαδρομή. Τα Pixels περνούν από έναν πίνακα στο σύστημα μνήμης . Στη συνέχεια τα δεδομένα είναι κλιμακωτά και υποβάλλονται σε επεξεργασία σε ένα χάρτη pixel. Τα αποτελέσματα είτε μεταφέρονται στη μνήμη ή αποστέλλονται στο στάδιο ραστεροποίησης. Εάν τα δεδομένα pixel διαβάζονται από το framebuffer(scale,bias, mapping και clamping) εκτελούνται. Στη συνέχεια συσκευάζονται τα αποτελέσματα αυτά σε κατάλληλη μορφή και επιστρέφονται σε

ένα πίνακα στη μνήμη του συστήματος.Υπάρχουν ειδικές λειτουργίες αντιγραφής pixel για την αντιγραφή δεδομένων στο framebuffer σε άλλα μέρη του framebuffer ή στη μνήμη . Πολλές από τις πράξεις που περιγράφουν pixel, είναι μέρος της σταθερής λειτουργίας pixel αγωγού και συχνά μεταφέρονται, μεγάλες ποσότητες δεδομένων γύρω από το σύστημα. Σύγχρονες εφαρμογές γραφικών, τείνουν να βελτιστοποιήσουν την απόδοση, προσπαθώντας να επινοήσουν ενέργειες για την τοπική μνήμη με την κάρτα γραφικών (Η περιγραφή αυτή είναι μια γενίκευση, βεβαίως, αλλά είναι το πώς τα περισσότερα συστήματα εφαρμόζονται σήμερα).Η OpenGL έκδοση 3.0, η οποία υποστηρίζει το σύνολο αυτών των πράξεων, εισάγει επίσης αντικείμενα στους framebuffers ωστε να επιτευχθεί μικρότερος φόρτος. Αντικείμενα framebuffer, σε συνδυασμό με την προγραμματιζόμενη απο shaders μέθοδο,έχουν αντικαταστήσει πολλές από αυτές τις πράξεις (κυρίως, αυτών που σχετίζονται με μεταβιβάσεις pixel) και παρέχεται έτσι σημαντικά μεγαλύτερη ευελιξία.

Texture Assembly

Οι OpenGL εφαρμογές μπορούν να εφαρμόσουν τις εικόνες για γεωμετρικά αντικείμενα ώστε να κάνουν τα αντικείμενα να φαίνονται πιο ρεαλιστικά, η οποία είναι μία από τις πολυάριθμες τεχνικές mapping. Αν χρησιμοποιούνται διάφορες εικόνες , είναι καλή μέθοδος να διαιρούνται σε αντικείμενα ώστε να μπορείτε να μεταβείτε εύκολα μεταξύ τους. Σχεδόν όλες οι εφαρμογές OpenGL έχουν ειδικούς πόρους για την επιτάχυνση-απόδοση (η οποία μπορεί να χορηγηθεί από μια κοινή δεξαμενή πόρων στην υλοποίηση γραφικών),για να να σας βοηθήσει ωστε να διαχειριστείτε αυτούς τους πόρους αποτελεσματικά(μνήμη, προτεραιότητες)και να βοηθήσει στον έλεγχο πιθανών caching και maping προβλημάτων.

Ραστεροποίηση – Rasterization

Ραστεροποίηση είναι η μετατροπή των γεωμετρικών και των pixel δεδομένων σε κομμάτια. Κάθε τετράγωνο κομμάτι αντιστοιχεί σε ένα pixel στο framebuffer-γραμμές και κουκίδες που συνθέτουν πολύγωνα , πλάτος γραμμών, μέγεθος σημείων, το μοντέλο σκίασης, και η κάλυψη.

Λειτουργίες Fragment

Πριν οι τιμές αποθηκευτούν στο framebuffer, μια σειρά ενεργειών εκτελείται και είναι δυνατό να αλλάξουν ή ακόμα και να απορρίψουν κομμάτια. Όλες αυτές οι ενέργειες μπορεί

να ενεργοποιηθούν ή να απενεργοποιηθούν. Η πρώτη ενέργεια που μια τιμή μπορεί να συναντήσει είναι το texturing, όπου ένα Texel (texture element), δημιουργείται από τη μνήμη για κάθε κομμάτι και εφαρμόζεται το κομμάτι. Συνδυάζονται στη συνέχεια, πρωτοβάθμια και δευτεροβάθμια χρώματα, καθώς και ο υπολογισμός ομίχλης. Εάν η εφαρμογή σας απασχολεί κομμάτια shaders, οι προηγούμενες τρεις πράξεις μπορούν να γίνουν με shader. Μετά την τελική παραγωγή χρωμάτων και τη διεξαγωγή των προηγούμενων ενεργειών, ο έλεγχος για εξάλειψη, η δοκιμή άλφα, η stencil δοκιμή, και η depth-buffer δοκιμή (η δοκιμή αυτή δεν είναι hidden-surface αφαίρεση) αξιολογούνται, εφόσον είναι ενεργοποιημένες. Διαφορετικά μπορεί να τερματίσει η επεξεργασία ενός κομματιού . Στη συνέχεια, μπορεί να γίνει ανάμειξη, αναποφασιστικότητα, λογική λειτουργία, και συγκάλυψη από bitmask. Τέλος, τα επιτυχημένα κομμάτια μεταφέρονται στην τελική τους θέση και είναι πλέον ένα pixel.

OpenGL-Σχετικές Βιβλιοθήκες

Η OpenGL παρέχει ένα ισχυρό, αλλά προκαθορισμένο σύνολο εντολών και όλη η υψηλότερου επιπέδου σχεδίαση, πρέπει να γίνει από αυτές τις εντολές. Επίσης, τα OpenGL προγράμματα πρέπει να χρησιμοποιούν μηχανισμούς παραθύρων του συστήματος. Πολλές βιβλιοθήκες σας δίνουν τη δυνατότητα να απλοποιηθεί ο προγραμματισμός σας, με εργασίες ως εξής:

• Ο OpenGL Utility Βιβλιοθήκη (GLU) περιέχει αρκετές ρουτίνες που χρησιμοποιούν χαμηλότερου επιπέδου OpenGL εντολές για την εκτέλεση των καθηκόντων αυτών, όπως μήτρες για συγκεκριμένες κατευθύνσεις στην προβολή και τις προβλέψεις, την εκτέλεση tessellation πολύγωνου, και επιφάνειες . Αυτή η βιβλιοθήκη παρέχεται ως μέρος της κάθε εφαρμογής OpenGL.

• Για κάθε σύστημα παραθύρων, υπάρχει μια βιβλιοθήκη που επεκτείνει τη λειτουργικότητα αυτού του συστήματος , για την υποστήριξη OpenGL rendering, για τις συσκευές που χρησιμοποιούν το σύστημα X Window, όπως η επέκταση OpenGL στο X Window Σύστημα (GLX) που παρέχεται ως συμπλήρωμα της OpenGL .Οι GLX ρουτίνες χρησιμοποιούν το πρόθεμα GLX. Για τα Microsoft Windows, παρέχονται οι ρουτίνες WGL των Windows για OpenGL διεπαφή. Όλες οι ρουτίνες WGL κάνουν χρήση του προθέματος WGL. Για Mac OS, τρεις διεπαφές είναι διαθέσιμες: AGL (με πρόθεμα AGL), CGL (cgl) και το Cocoa (NSOpenGL κλάσεις).

• Ο OpenGL Utility Toolkit (GLUT) είναι ένα παραθυρικό σύστημα, ανεξάρτητο από την εργαλειοθήκη, αρχικά γράφτηκε από τον Mark Kilgard και κρύβει την πολυπλοκότητα των διαφορετικών παραθύρων APIs στο σύστημα. Υπάρχει επίσης μια έκδοση, με όνομα freeglut, η οποία επεκτείνει την αρχική λειτουργικότητα του GLUT. Η επόμενη ενότητα περιγράφει τις θεμελιώδεις ρουτίνες- αναγκαίες για προγράμματα ,χρησιμοποιώντας GLUT. Στα περισσότερα μέρη του κειμένου, θα χρησιμοποιείται ο όρος GLUT, με την σύμβαση ότι μπορεί να χρησιμοποιούμε και Freeglut εφαρμογή.

Συμπεριλάβετε Αρχεία

Για όλες τις εφαρμογές OpenGL, θα πρέπει να συμπεριλάβετε τα OpenGL αρχεία επικεφαλίδων σε κάθε αρχείο. Πολλές εφαρμογές OpenGL μπορεί να χρησιμοποιηθούν, όπως η GLU(η προαναφερθείσα OpenGL Utility Βιβλιοθήκη), η οποία απαιτεί την ένταξη του αρχείου header glu.h.

Έτσι σχεδόν κάθε αρχείο προέλευσης OpenGL ξεκινά με :

```
#include <GL/gl.h>

#include <GL/glu.h>
```

Σημείωση: Τα Microsoft Windows απαιτούν windows.h είτε gl.h ή glu.h, επειδή ορισμένες μακροεντολές που χρησιμοποιούνται στο εσωτερικό της Microsoft Windows έκδοσης του gl.h και glu.h ορίζονται στο windows.h. Η βιβλιοθήκη OpenGL αλλάζει συνεχώς. Οι διάφοροι προμηθευτές που καθιστούν υλικό γραφικών προσθέτουν νέα χαρακτηριστικά που μπορεί να είναι πολύ νέα για να μπορούν να ενσωματώνονται σε gl.h. Για να επωφεληθείτε από τις νέες αυτές επεκτάσεις OpenGL, ένα πρόσθετο αρχείο κεφαλίδας είναι διαθέσιμο, ονομάζεται glext.h. Αυτή η επικεφαλίδα περιέχει όλες τις τελευταίες εκδόσεις και τις λειτουργίες επέκτασης και tokens , και είναι διαθέσιμη για OpenGL στην τοποθεσία Web OpenGL (http://www.opengl.org/registry). Η τοποθεσία περιέχει επίσης τις προδιαγραφές για κάθε επέκταση OpenGL που δημοσιεύεται. Όπως και με κάθε επικεφαλίδα, θα μπορούσε να περιλαμβάνει την ακόλουθη δήλωση:

```
# include "glext.h"
```

Έχετε παρατηρήσει πιθανώς τα εισαγωγικά γύρω από το όνομα του αρχείου, σε σύγκριση με την φυσιολογική παρένθεση. Επειδή χρησιμοποιώντας glext.h υπάρχει θέμα για το αν επιτρέπουν οι πωλητές γραφικών πρόσβαση στις νέες επεκτάσεις, μάλλον θα χρειαστεί να κατεβάσετε εκδόσεις συχνά από το Internet, έχοντας έτσι ένα τοπικό αντίγραφο για να καταρτίσει το πρόγραμμά σας. Επιπλέον, μπορεί να μην έχετε την άδεια να τοποθετήσετε το glext.h αρχείο κεφαλίδας σε ένα σύστημα-header αρχείο που περιλαμβάνει κατάλογο (όπως / usr / include σε Unix συστήματα τύπου). Εάν αποκτάτε πρόσβαση σε μια βιβλιοθήκη με απευθείας διασύνδεση παραθύρου για την υποστήριξη OpenGL, όπως η GLX, WGL, ή το CGL, θα πρέπει να περιλαμβάνετε πρόσθετα αρχεία κεφαλίδας. Για παράδειγμα, αν καλείτε GLX, μπορεί να χρειαστεί να προσθέσετε αυτές τις γραμμές :

```
# include <X11/Xlib.h>
```

```
# include <GL/glx.h>
```

Στα Microsoft Windows, οι συνήθεις WGL μπορούν να καταστούν προσβάσιμες με :

```
# include <windows.h>
```

Αν χρησιμοποιείτε GLUT για τη διαχείριση των παραθύρων εργασιών του διαχειριστή, θα πρέπει να περιλαμβάνετε :

```
# include <freeglut.h>
```

Σημείωση: Το αρχικό αρχείο κεφαλίδας GLUT ονομάστηκε glut.h. Τόσο η glut.h όσο και η freeglut.h εγγυάται ότι gl.h και glu.h συμπεριλαμβάνονται σωστά για σας. Επιπλέον, σε αυτές τις κεφαλίδες βεβαιωθείτε ότι όλες οι εσωτερικές λειτουργικές διαδικασίες του συστήματος εξαρτώνται από μακροεντολές . Για να κάνετε GLUT σε φορητά προγράμματα , χρειάζεται glut.h ή freeglut.h. Οι περισσότερες OpenGL εφαρμογές χρησιμοποιούν επίσης τα τυποποιημένα στοιχεία C- κλήσεις του συστήματος βιβλιοθηκών, έτσι είναι ορθό να περιλάβουμε αρχεία κεφαλής που δεν έχουν σχέση με γραφικά, όπως :

```
# include <stdlib.h>
# include <stdio.h>
```

Σε αυτό το σημείο θα πρέπει να αναφερθεί πως είναι δυνατόν να υπάρχουν προβλήματα συμβατότητας μεταξύ παλαιότερων με καινούργιες εκδόσεις της OpenGl.

Για να κάνουμε αυτή τη μετάβαση ευκολότερη για δημιουργούς λογισμικού, η OpenGL Έκδοση 3.1 παρέχει μια ολόκληρη νέα σειρά κεφαλίδων αρχείων, και συνιστά μια τοποθεσία για τους πωλητές ώστε να τα εντάξουν στα αντίστοιχα λειτουργικά συστήματα. Μπορείτε ακόμα να χρησιμοποιήσετε τα gl.h και glext.h αρχεία, τα οποία θα συνεχίσουν να καταγράφουν όλα τα σημεία εισόδου OpenGL, ανεξάρτητα από έκδοση.

Ωστόσο, εάν έχετε κώδικα που πρέπει να χρησιμοποιείται μόνο με την έκδοση 3.1, μπορείτε να εξετάσετε τη χρήση της νέας έκδοσης OpenGL 3.1 επικεφαλίδες:

```
# include <GL3/gl3.h>

# include <GL3/gl3ext.h>
```

Περιλαμβάνουν τις λειτουργίες και τα tokens για την έκδοση 3.1 (για τις μελλοντικές εκδόσεις, το χαρακτηριστικά που θα πρέπει να περιορίζονται σε αυτή τη συγκεκριμένη έκδοση). Θα θα δείτε ότι αυτές οι κεφαλίδες απλοποιούν τη διαδικασία μετακίνησης υπάρχοντα κώδικα σε OpenGL νεότερες εκδόσεις. Όπως και για κάθε κεφαλίδα OpenGL, αυτά τα αρχεία είναι διαθέσιμα για λήψη από το OpenGL site (http://www.opengl.org/registry).

GLUT, το OpenGL Utility Toolkit

Όπως γνωρίζετε, η OpenGL περιέχει εντολές απόδοσης, αλλά έχει σχεδιαστεί για να είναι ανεξάρτητη από οποιοδήποτε σύστημα παραθύρων ή το λειτουργικό σύστημα. Κατά συνέπεια, δεν περιέχει εντολές για το άνοιγμα των παραθύρων διαβάζοντας τα γεγονότα από την πληκτρολόγιο ή το ποντίκι. Δυστυχώς, είναι αδύνατο να γράψει κάποιος ένα πλήρες πρόγραμμα χωρίς τουλάχιστον το άνοιγμα ενός παραθύρου, και πιο ενδιαφέρον τα προγράμματα απαιτούν ένα κομμάτι της εισόδου του χρήστη ή άλλες υπηρεσίες από το λειτουργικό σύστημα ή το σύστημα παραθύρων. Επιπλέον, δεδομένου ότι οι OpenGL εντολές σχεδίασης περιορίζονται σε αυτές που δημιουργούν απλά γεωμετρικά πρότυπα (σημεία, γραμμές και πολύγωνα), η GLUT περιλαμβάνει διάφορες ρουτίνες που δημιουργούν πιο περίπλοκα τρισδιάστατα αντικείμενα, όπως μια σφαίρα, ένα σωσίβιο, και μια τσαγιέρα (Σημειώστε, ότι η βοηθητική βιβλιοθήκη της OpenGl, έχει επίσης quadrics ρουτίνες που δημιουργούν μερικά από τα ίδια τρισδιάστατα αντικείμενα όπως της GLUT, όπως μια σφαίρα, κύλινδρος, ή κώνος)Η GLUT μπορεί να μην είναι ικανοποιητική για εφαρμογές με πλήρεις δυνατότητες OpenGL, αλλά μπορείτε να βρείτε ένα χρήσιμο σημείο εκκίνησης για την εκμάθηση της OpenGL.

Παράθυρο Διαχείρισης

Αρκετές ρουτίνες εκτελούν τις εργασίες που απαιτούνται για την προετοιμασία ενός παραθύρου:

• glutInit (int * argc, char ** argv) αρχικοποιεί τη GLUT). Η glutInit () θα πρέπει να καλείται πριν από κάθε άλλη ρουτίνα GLUT.

• glutInitDisplayMode (unsigned λειτουργία int) καθορίζει αν θα χρησιμοποιήσετε ένα RGBA ή color-index χρωματικό μοντέλο. Μπορείτε επίσης να καθορίσετε αν θέλετε ένα μονού ή διπλού-buffered παράθυρο. (Εάν εργάζεστε σε colorindex λειτουργία, θα θέλετε να φορτώσετε ορισμένα χρώματα στο χάρτη χρωμάτων. Η glutSetColor ()μπορεί να το κάνει αυτό. Τέλος, μπορείτε να χρησιμοποιήσετε αυτήν την ρουτίνα για να δείξετε ότι θέλετε το παράθυρο να έχει ένα σχετικό βάθος, stencil, multisampling, και / ή buffer καταχώρηση. Για παράδειγμα, εάν θέλετε ένα παράθυρο με διπλό buffering, το μοντέλο χρωμάτων RGBA, και το βάθος buffer, θα μπορούσαμε να πούμε glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH).

• glutInitWindowPosition (int x, int y) καθορίζει τη θέση της οθόνης για την επάνω αριστερή γωνία του παραθύρου σας.

• glutInitWindowSize (int πλάτος, ύψος int) καθορίζει το μέγεθος, σε pixels, από το παράθυρό σας.

• glutInitContextVersion (int majorVersion, int minorVersion) προσδιορίζει ποια έκδοση της OpenGL θέλετε να χρησιμοποιήσετε. (Αυτή είναι μια νέα προσθήκη ,διαθέσιμη μόνο όταν χρησιμοποιείτε Freeglut, και εισήχθη με την OpenGL έκδοση 3.0.

• glutInitContextFlags (int flags) καθορίζει τον τύπο του πλαισίου OpenGL που θέλετε να χρησιμοποιήσετε. Για κανονική λειτουργία OpenGL, μπορείτε να παραλείψετε την παρούσα κλήση από το πρόγραμμά σας. Ωστόσο, εάν θέλετε να χρησιμοποιήσετε ένα forward-compatible - OpenGL πλαίσιο, θα χρειαστεί να καλέσετε αυτήν την ρουτίνα. (Αυτό είναι επίσης ένα νέο επιπρόσθετο και διαθέσιμο μόνο σε Freeglut, και εισήχθη με την OpenGL Έκδοση 3.0.

• int glutCreateWindow (char * string) δημιουργεί ένα παράθυρο με OpenGL πλαίσιο. Επιστρέφει ένα μοναδικό αναγνωριστικό για το νέο παράθυρο. Προσοχή : μέχρι η glutMainLoop () να κληθεί, το παράθυρο δεν θα έχει ακόμη εμφανιστεί.

Η επανάκληση Display

Η glutDisplayFunc (void (* func) (void)) είναι το πρώτο και πιο σημαντικό γεγονός λειτουργίας επανάκλησης . Όποτε η GLUT αποφασίσει ότι η περιεχόμενα του παραθύρου πρέπει να επανεμφανίζονται, η λειτουργία επανάκλησης εκτελείται από την glutDisplayFunc () . Ως εκ τούτου, θα πρέπει να τεθούν όλες οι ρουτίνες ωστε να αναδιατυπώσουν τη σκηνή στη λειτουργία επανάκλησης οθόνης. Εάν το πρόγραμμά σας αλλάζει το περιεχόμενο του παραθύρου, μερικές φορές θα πρέπει να καλέσετε την glutPostRedisplay (), η οποία δίνει στη glutMainLoop () μια ώθηση για να καλέσετε τον καταχωρημένη επανάκληση οθόνης στην επόμενη ευκαιρία.

Εκτέλεση Προγράμματος

Το τελευταίο πράγμα που πρέπει να κάνετε είναι να καλέσετε τη glutMainLoop (). Όλα τα παράθυρα που έχουν δημιουργηθεί , τώρα θα πρέπει να φαίνονται. Η επεξεργασία Event ξεκινά, και η καταστατική επανάκληση οθόνης ενεργοποιείται. Μόλις ξεκινήσει αυτή η θηλιά , δεν σταματά ποτέ ! Το παράδειγμα 1-2 δείχνει πώς μπορείτε να χρησιμοποιήσετε τη GLUT για να δημιουργήσετε το απλό πρόγραμμα που φαίνεται στο Παράδειγμα 1-1. Σημειώστε την αναδιάρθρωση του κώδικα. Για τη μεγιστοποίηση της αποτελεσματικότητας, λειτουργίες που πρέπει να καλούνται μόνο μία φορά (καθορισμός του χρώματος του φόντου και το συντονισμό του συστήματος) ,βρίσκονται τώρα σε μια ρουτίνα που ονομάζεται init ().

Παράδειγμα 1-2 Απλό πρόγραμμα OpenGL Χρησιμοποιώντας GLUT: hello.c

```
void display(void)

{

/* clear all pixels */

glClear(GL_COLOR_BUFFER_BIT);

/* draw white polygon (rectangle) with corners at

* (0.25, 0.25, 0.0) and (0.75, 0.75, 0.0)

*/

glColor3f(1.0, 1.0, 1.0);
```

```
glBegin(GL_POLYGON);
glVertex3f(0.25, 0.25, 0.0);
glVertex3f(0.75, 0.25, 0.0);
glVertex3f(0.75, 0.75, 0.0);
glVertex3f(0.25, 0.75, 0.0);
glEnd();
/* don't wait!
* start processing buffered OpenGL routines
*/
glFlush();
}
void init(void)
{
/* select clearing (background) color */
glClearColor(0.0, 0.0, 0.0, 0.0);
/* initialize viewing values */
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}
/*
* Declare initial window size, position, and display mode
* (single buffer and RGBA). Open window with "hello"
* in its title bar. Call initialization routines.
* Register callback function to display graphics.
* Enter main loop and process events.
*/
```

```
int main(int argc, char** argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(250, 250);
glutInitWindowPosition(100, 100);
glutCreateWindow("hello");
init();
glutDisplayFunc(display);
glutMainLoop();
return 0; /* ISO C requires main to return int. */
}
```

Χειρισμός Γεγονότων Εισόδου

Μπορείτε να χρησιμοποιήσετε τις ακόλουθες ρουτίνες για να καταχωρήσετε εντολές επανάκλησης που καλούνται όταν συμβαίνουν συγκεκριμένα γεγονότα:

• glutReshapeFunc (void (* func) (int w, int h)) δείχνει σε ποιες ενέργειες θα πρέπει να γίνονται, όταν το παράθυρο αλλάξει μέγεθος.

• glutKeyboardFunc (void (* func) (unsigned char κλειδί, int x, int y)) και glutMouseFunc (void (* func) (int κουμπί, int κατάσταση, int x, int y)) επιτρέπουν να συνδεθεί ένα πλήκτρο του πληκτρολογίου ή ένα πλήκτρο του ποντικιού με μια ρουτίνα.

• glutMotionFunc (void (* func) (int x, int y)) είναι μια ρουτίνα που καλείται, όταν το ποντίκι μετακινηθεί ενώ ένα πλήκτρο του ποντικιού είναι πατημένο.

Η διαχείριση μιας διαδικασίας φόντου

Μπορείτε να καθορίσετε μια συνάρτηση που είναι να εκτελεστεί εάν δεν υπάρχουν άλλες εν αναμονή της-για παράδειγμα όταν ο βρόχος θα ήταν σε αδράνεια, με glutIdleFunc (void (* func) (κενό)). Αυτή η ρουτίνα παίρνει ένα δείκτη προς τη λειτουργία ως μόνο όρισμά του. Δίνοντας NULL (μηδέν) απενεργοποιείτε την εκτέλεση της συνάρτησης.

Σχεδίαση τρισδιάστατων αντικειμένων

Η GLUT περιλαμβάνει διάφορες ρουτίνες για τη σχεδίαση αυτών των τριών διαστάσεων αντικειμένων :

Cone	icosahedron	teapot
Cube	octahedron	tetrahedron
Dodecahedron	sphere	torus

Μπορείτε να σχεδιάσετε τα αντικείμενα αυτά ως wireframes ή ως shaded αντικείμενα. Για παράδειγμα, η ρουτίνα για ένα κύβο και μια σφαίρα, έχουν ως εξής:

void glutWireCube(GLdouble size);

void glutSolidCube(GLdouble size);

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

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

Όλα αυτά τα μοντέλα επικεντρώνονται στο σύστημα συντεταγμένων του πραγματικού κόσμου.

Animation

Ένα από τα πιο συναρπαστικά πράγματα που μπορείτε να κάνετε σε έναν υπολογιστή γραφικών είναι οι εικόνες που κινούνται. Είτε είστε ένας μηχανικός που προσπαθεί να δείξει όλες τις πλευρές ενός μηχανικού μέρους, είτε θέλετε να κατασκευάσετε έναν προσομοιωτή πτήσης για να πετάξει ένα αεροπλάνο , ή απλώς ένα παιχνίδι στον υπολογιστή , είναι σαφές ότι το animation είναι ένα σημαντικό μέρος των γραφικών υπολογιστών. Σε μια κινηματογραφική αίθουσα, η κίνηση επιτυγχάνεται με τη λήψη μιας ακολουθίας εικόνων και προβολή τους στα 24 καρέ ανά δευτερόλεπτο στην οθόνη. Κάθε πλαίσιο κινείται στη θέση του πίσω από το φακό, το κλείστρο είναι ανοιχτό, και το πλαίσιο εμφανίζεται στην οθόνη. Το κλείστρο είναι στιγμιαία κλειστό, ενώ η ταινία έχει προχωρήσει στην επόμενο πλαίσιο, τότε αυτό το πλαίσιο εμφανίζεται, και ούτω καθεξής. Παρόλο που βλέπετε 24 διαφορετικά καρέ ανά δευτερόλεπτο, το μυαλό σας να συνδυάζει όλα σε ένα ομαλή κίνηση. (Οι παλιές Τσάρλι Τσάπλιν ταινίες γυρίστηκαν στα 16 καρέ ανά δευτερόλεπτο και είναι αισθητό αυτό). Στα Computer-γραφικά γίνεται ανανέωση (επαναπροσδιορίσει την εικόνα), περίπου 60 με 76 φορές ανά δευτερόλεπτο, και μερικά

ακόμη λειτουργούν σε περίπου 120 ανά δευτερόλεπτο. Σαφώς, 60 ανά δευτερόλεπτο είναι καλύτερο από 30, και 120 είναι αντιληπτικά καλύτερο από ό, τι 60.

Ας υποθέσουμε ότι προσπαθείτε να κάνετε animation υπολογιστή με εκατομμύρια πλαίσια με ένα πρόγραμμα όπως αυτό:

open_window ()?

για (i = 0? i <1000000? i + +) {

clear_the_window ()?

draw_frame (i)?

wait_until_a_24th_of_a_second_is_over ()?

}

Αν προσθέσετε το χρόνο που απαιτείται για το σύστημά σας για να καθαρίσει την οθόνη και να επιστήσει ένα τυπικό πλαίσιο, το πρόγραμμα αυτό δίνει όλο και πιο φτωχά αποτελέσματα, ανάλογα με το πόσο κοντά στο 1 / 24 δευτερόλεπτα χρειάζεται για να καθαρίσει και να σχεδιάσει.

Οι περισσότερες OpenGL εφαρμογές παρέχουν διπλό buffer-υλικού ή λογισμικού που παρέχει δύο πλήρεις buffers χρώματος. Ο ένας εμφανίζει , όταν ο άλλος είναι στο στάδιο της επεξεργασίας. Όταν η σχεδίαση ενός πλαισίου είναι πλήρης, οι δύο buffers είναι αντίστροφοι, έτσι ώστε αυτό που ήταν, θα θεωρείται τώρα ότι χρησιμοποιείται για την σχεδίαση και το αντίστροφο. Αυτό είναι σαν μια μηχανή προβολής, με μόνο δύο καρέ σε έναν βρόχο. Με το διπλό buffering, κάθε καρέ εμφανίζεται μόνο όταν το σχέδιο έχει ολοκληρωθεί. Ο θεατής δεν βλέπει μια μερική σχεδίαση πλαισίου. Μια τροποποιημένη εκδοχή που εμφανίζει ομαλά κινούμενα γραφικά που χρησιμοποιούν doublebuffering μπορεί να μοιάζει κάπως έτσι:

open_window_in_double_buffer_mode();

for (i = 0; i < 1000000; i++) {

clear_the_window();

draw_frame(i);

swap_the_buffers();

}

Πάγωμα

Ορισμένες εφαρμογές OpenGL, εκτός από την εναλλαγή χρησιμοποιούν και τη swap_the_buffers () ρουτίνα η οποία περιμένει μέχρι την τρέχουσα περίοδο ανανέωσης της. Αυτή η τακτική επιτρέπει, επίσης στο το νέο buffer να εμφανίζεται ξεκινώντας από την αρχή. Υποθέτοντας ότι ανανεώνει το σύστημά σας την οθόνη 60 φορές το δευτερόλεπτο, αυτό σημαίνει ότι ο πιο γρήγορος ρυθμός καρέ που μπορεί να επιτύχει είναι 60 καρέ ανά δευτερόλεπτο (fps).

Motion = Redraw + Swap

Η δομή των πραγματκών προγραμμάτων κινουμένων σχεδίων δεν διαφέρει πολύ από αυτή την περιγραφή. Συνήθως, είναι πιο εύκολο να αναδιατυπωθεί το εσωτερικό του buffer από το μηδέν για κάθε καρέ, από το να καταλάβουμε ποιες περιοχές χρειάζονται επαναδιατύπωση. Αυτό ισχύει ιδιαίτερα σε εφαρμογές όπως τρισδιάστατοι εξομοιωτές πτήσης, όπου μια μικρή αλλαγή στον προσανατολισμό του αεροπλάνου αλλάζει τη θέση ολοκληρωτικά .Στις περισσότερες κινούμενες εικόνες, τα αντικείμενα σε μια σκηνή, απλά επανασχεδιάζονται με διαφορετικές μεταμορφώσεις-η άποψη των κινήσεων θεατή, ή ένα αυτοκίνητο κινείται κάτω από το δρόμο λίγο, ή ένα αντικείμενο που περιστρέφεται ελαφρώς. Να θυμάστε, ωστόσο, ότι στο χρόνο αδράνειας, η swap_the_buffers () ρουτίνα ,μπορεί συχνά να χρησιμοποιείται για τέτοιους υπολογισμούς. Η OpenGL δεν έχει swap_the_buffers () εντολή, επειδή αυτή η δυνατότητα μπορεί να μην είναι διαθέσιμη σε όλο το υλικό. Για παράδειγμα, εάν χρησιμοποιείτε το Σύστημα παραθύρων X και την πρόσβαση σε αυτό άμεσα, μπορείτε να χρησιμοποιήσετε την ακόλουθη GLX ρουτίνα:

void glXSwapBuffers(Display *dpy, Window window);

Εάν χρησιμοποιείτε τη βιβλιοθήκη GLUT, θα πρέπει να καλέσετε αυτή την ρουτίνα:

glutSwapBuffers void(void) ;

Το παράδειγμα 1-3 απεικονίζει τη χρήση των glutSwapBuffers () στην σχεδίαση ενός περιστρεφόμενου κύβου, όπως φαίνεται στο Σχήμα 1-3. Το παράδειγμα αυτό δείχνει επίσης πώς να χρησιμοποιείτε τη GLUT για τον έλεγχο μιας συσκευής εισόδου και να ενεργοποιήσετε και να απενεργοποιήσετε την αδρανή λειτουργία. Σε αυτό το παράδειγμα, τα κουμπιά του ποντικιού ενεργοποιούν και απενεργοποιούν την περιστροφή .

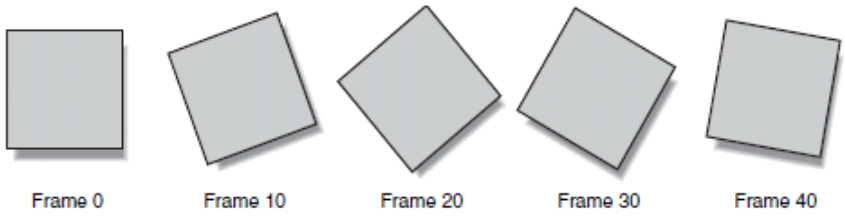


Figure 1-3 Double-Buffered Rotating Square

Example 1-3 Double-Buffered Program: double.c

```
static GLfloat spin = 0.0;

void init(void)

{

glClearColor(0.0, 0.0, 0.0, 0.0);

glShadeModel(GL_FLAT);

}

void display(void)

{

glClear(GL_COLOR_BUFFER_BIT);

glPushMatrix();

glRotatef(spin, 0.0, 0.0, 1.0);

glColor3f(1.0, 1.0, 1.0);
```

```
glRectf(-25.0, -25.0, 25.0, 25.0);
glPopMatrix();
glutSwapBuffers();
}



void spinDisplay(void)
{
spin = spin + 2.0;
if (spin > 360.0)
spin = spin - 360.0;
glutPostRedisplay();
}
void reshape(int w, int h)
{
glViewport(0, 0, (GLsizei) w, (GLsizei) h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-50.0, 50.0, -50.0, 50.0, -1.0, 1.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
}
void mouse(int button, int state, int x, int y)
{
switch (button) {
case GLUT_LEFT_BUTTON:
if (state == GLUT_DOWN)
```

```
glutIdleFunc(spinDisplay);
break;
case GLUT_MIDDLE_BUTTON:
if (state == GLUT_DOWN)
glutIdleFunc(NULL);
break;
default:
break;
}
}
/*
* Request double buffer display mode.
* Register mouse input callback functions
*/
int main(int argc, char** argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
glutInitWindowSize(250, 250);
glutInitWindowPosition(100, 100);
glutCreateWindow(argv[0]);
init();

glutDisplayFunc(display);
glutReshapeFunc(reshape);
glutMouseFunc(mouse);
glutMainLoop();
```

```
return 0;

}
```

Γενικά για την OpenGl

Προχωρημένα θέματα

Όπως αναφέρθηκε προηγουμένως, η OpenGL υποβάλλεται σε συνεχή βελτίωση και φινέτσα. Νέες μέθοδοι γραφικών εργασιών αναπτύχθηκαν, και αρκετά νέα πεδία, όπως το GPGPU (συντομογραφία του «γενικής χρήσης υπολογιστές» σε μονάδες επεξεργασίας γραφικών "),γεγονότα που φαίνεται ότι οδηγούν σε εξέλιξη των δυνατοτήτων του υλικού σε γραφικά. Νέες επεκτάσεις OpenGL προτείνονται από τους πωλητές, και τελικά ορισμένες από αυτές τις επεκτάσεις έχουν ενσωματωθεί ως μέρος μιας νέας αναθεώρησης πυρήνα της OpenGL. Με τα χρόνια, αυτή η διαδικασία ανάπτυξης έχει αποτρέψει πολλές περιττές μεθόδους για την υλοποίηση της ίδιας δραστηριότητας. Σε πολλές περιπτώσεις, ενώ η λειτουργικότητα ήταν παρόμοια, η απόδοση των εφαρμογών των μεθόδων »γενικά δεν ήταν, δίνοντας την εντύπωση ότι η OpenGL API ήταν αργή και δεν λειτούργησε καλά σε σύγχρονα συστήματα υλικού. Με την OpenGL έκδοση 3.0, η ομάδα εργασίας Khronos OpenGL ARB, καθιέρωσε ένα μοντέλο αποσβέσεων που αναφέρεται χαρακτηριστικά στο πως θα μπορούσαν να αφαιρεθούν από το API. Ωστόσο, για αυτή την αλλαγή, απαιτείται κάτι περισσότερο από αλλαγές στον πυρήνα του OpenGL API-αυτό επηρέαζε και το πώς τα

πλαίσια OpenGL δημιουργήθηκαν, και το » πώς» οι τύποι των πλαισίων γίνονταν διαθέσιμοι.

Τα OpenGL Πλαίσια

Ένα πλαίσιο OpenGL είναι η δομή των δεδομένων, όπου η OpenGL κρατά πληροφορίες που πρέπει να χρησιμοποιηθούν όταν σχεδιάζονται εικόνες. Περιλαμβάνει τα αντικείμενα όπως textures, server-side buffering αντικείμενα , τα σημεία εισόδου λειτουργίας, την ανάμειξη τμημάτων, και τη σχεδίαση shader αντικειμένων. Σε εκδόσεις της OpenGL πριν από την έκδοση 3.0, υπήρχε ένα μόνο είδος OpenGL πλαισίου-το πλήρες πλαίσιο, που περιείχε διαθέσιμες όλες τις πληροφορίες της εφαρμογής OpenGL, και υπήρχε μόνο ένας τρόπος να δημιουργήσεις ένα πλαίσιο (το οποίο ήταν window-system ανεξάρτητο). Με την έκδοση 3.0, ένα νέο είδος πλαισίου δημιουργήθηκε-η προς τα εμπρός-συμβατότητα πλαισίου-το οποίο κρύβει τα χαρακτηριστικά που προαναφέρθηκαν για τη μελλοντική απομάκρυνση από την OpenGL API, για να βοηθήσει τους προγραμματιστές εφαρμογών να ικανοποιήσουν τις απαιτήσεις τους, με τις μελλοντικές εκδόσεις της OpenGL.

Profiles

Εκτός από τους διάφορους τύπους περιβάλλοντος προστέθηκε στην έκδοση 3.0, η έννοια του προφίλ . Ένα προφίλ είναι ένα υποσύνολο των λειτουργιών OpenGL ειδικά για ένα πεδίο εφαρμογής, όπως παιχνίδια, computer-aided design (CAD), ή προγράμματα που έχουν γραφτεί για ενσωματωμένες πλατφόρμες. Επί του παρόντος, μόνο ένα προφίλ ορίζεται, το οποίο εκθέτει το σύνολο των λειτουργιών που υποστηρίζονται στο πλαίσιο που δημιουργήθηκε στην OpenGL. Νέοι τύποι προφίλ μπορούν να εισαχθούν σε μελλοντικές εκδόσεις της OpenGL. Κάθε σύστημα παραθύρων έχει το δικό του σύνολο λειτουργιών για την ενσωμάτωση της OpenGL κατά τη λειτουργία του (π.χ., WGL για τα Microsoft Windows), το οποίο είναι αυτό που πραγματικά δημιουργεί τον τύπο του πλαισίου OpenGL που ζητάτε (επίσης με βάση το προφίλ).Η GLUT κρύβει τις λεπτομέρειες αυτής της λειτουργίας. Σε κάποιο σημείο, μπορεί να πρέπει να είναι γνωστές οι λεπτομέρειες.

Καθορισμός OpenGL Εκδόσεων με GLUT

Η βιβλιοθήκη GLUT φροντίζει αυτόματα για τη δημιουργία ενός OpenGL πλαισίου όταν η glutCreateWindow () καλείται. Ως προεπιλογή, το αιτούμενο OpenGL πλαίσιο θα είναι συμβατό με την έκδοση OpenGL 2.1. Για ένα πλαίσιο της OpenGL 3.0 και μετά, θα χρειαστεί η glutInitContextVersion (). Ομοίως, εάν θέλετε να χρησιμοποιήσετε ένα forwardcompatible πλαίσιο για porting, θα πρέπει επίσης να διευκρινισθεί αυτό το χαρακτηριστικό, καλώντας glutInitContextFlags

Παράδειγμα 1-4 Δημιουργία Έκδοση OpenGL 3.0 Πλαίσιο Χρησιμοποιώντας GLUT

```
glutInit(&argc, argv);

glutInitDisplayMode(GLUT_RGBA | GLUT_DEPTH | GLUT_DOUBLE);

glutInitWindowSize(width, height);

glutInitWindowPosition(xPos, yPos);

glutInitContextVersion(3, 0);

glutInitContextFlags(GLUT_FORWARD_COMPATIBLE);

glutCreateWindow(argv[0]);
```

Πρόσβαση στις Λειτουργίες-Συναρτήσεις της OpenGL

Ανάλογα με το λειτουργικό σύστημα στο οποίο αναπτύσσετε εφαρμογές, μπορεί να χρειαστεί να κάνετε κάποιες επιπλέον εργασίες για την πρόσβαση σε ορισμένες OpenGL λειτουργίες. Μπορείτε να ξέρετε πότε αυτή η ανάγκη προκύπτει, επειδή σας αναφέρει ο compiler ότι οι διάφορες λειτουργίες είναι ακαθόριστες (φυσικά, κάθε compiler θα αναφέρεται σε αυτό το σφάλμα με διαφορετικό τρόπο, αλλά αυτό είναι η ουσία του θέματος). Σε αυτές τις περιπτώσεις, θα χρειαστεί να ανακτήσετε τη διεύθυνση της λειτουργίας του (σε ένα δείκτη λειτουργίας). Υπάρχουν διάφοροι τρόποι για να επιτευχθεί αυτό:

• Αν η εφαρμογή χρησιμοποιεί το εγγενές σύστημα παραθύρων για το άνοιγμα των παραθύρων και την επεξεργασία γεγονότων, χρησιμοποιήστε την κατάλληλη * GetProcAddress () συνάρτηση που το λειτουργικό σας σύστημα θα χρησιμοποιεί. Παραδείγματα τέτοιων λειτουργιών περιλαμβάνουν την wglGetProcAddress () και glXGetProcAddress ().

• Εάν χρησιμοποιείτε GLUT, στη συνέχεια, χρησιμοποιήστε τη λειτουργία ανάκτησης GLUT του δείκτη ρουτίνας, glutGetProcAddress ().

• Χρησιμοποιήστε το open-source έργο GLEW (συντομογραφία του «Επέκταση OpenGL Wrangler "). Το GLEW ορίζει κάθε λειτουργία OpenGL, ανάκτα τους δείκτες λειτουργιών και την επαλήθευση επεκτάσεων, αυτόματα, για εσάς. Πηγαίνετε στο http://glew.sourceforge.net/ να βρείτε περισσότερες λεπτομέρειες και για την απόκτηση του κώδικα ή των εκτελέσιμών.

Καθαρισμός του παραθύρου

Σχεδιάζοντας στην οθόνη ενός υπολογιστή είναι διαφορετικό από την σχεδίαση σε χαρτί, οπου το χαρτί είναι και λευκό, και το μόνο που έχετε να κάνετε σε αυτό είναι να ζωγραφίσετε την εικόνα. Σε έναν υπολογιστή, η μνήμη που κατέχει η εικόνα είναι συνήθως γεμάτη με την τελευταία εικόνα που σχεδιάστηκε, έτσι συνήθως χρειάζεται να διευκρινιστεί κάποιο χρώμα φόντου πριν ξεκινήσετε για να σχεδιάσετε την νέα σκηνή. Το χρώμα που χρησιμοποιείτε για το φόντο εξαρτάται από την εφαρμογή. Για έναν επεξεργαστή κειμένου, συνήθως είναι λευκό (το χρώμα του χαρτιού). Αν θέλετε φόντο εικόνας με ένα διαστημόπλοιο, θα ήταν σωστό να επιλέξετε το μαύρο του διαστήματος, πριν από την έναρξη για να συναγάγει τα αστέρια, τους πλανήτες, κτλ. Μερικές φορές ίσως να μην χρειαστεί να καθαρίσετε την οθόνη καθόλου. Για παράδειγμα, αν η εικόνα είναι το εσωτερικό ενός δωματίου, οπου καλύπτεται ολόκληρο το παράθυρο γραφικών ,καθώς σχεδιάζετε όλους τους τοίχους.

Μια ειδική εντολή για να διαγράψετε ένα παράθυρο μπορεί να είναι πολύ πιο αποτελεσματική από μια γενικής χρήσης εντολή σχεδίασης. Επιπλέον, η OpenGL σας επιτρέπει να ρυθμίσετε το σύστημα συντεταγμένων, που βλέπουν τη θέση και την προβολή κατεύθυνσης, αυθαίρετα, έτσι ενδέχεται να είναι δύσκολο να αντιληφθείτε το κατάλληλο μέγεθος και τη θέση για ένα ορθογώνιο παράθυρο-καθαρισμού. Τέλος, σε πολλά μηχανήματα, το υποσύστημα γραφικών αποτελείται από πολλαπλούς buffers εκτός από τον buffer που περιέχει τα χρώματα των pixel που εμφανίζονται. Αυτοί οι άλλοι buffers πρέπει να καθαρίζονται από στιγμή σε στιγμή, και είναι πιο βολικό να έχουν μία μόνο εντολή που μπορεί να καθαρίσει οποιοδήποτε συνδυασμό αυτών.

Πρέπει επίσης να αναφερθεί, πως είναι τα χρώματα των pixels που αποθηκεύονται στο υλικό γραφικών, στα bitplanes. Υπάρχουν δύο μέθοδοι αποθήκευσης. Είτε το κόκκινο,

πράσινο, μπλε, και το άλφα (RGBA)-τιμές ενός pixel που μπορεί να είναι άμεσα αποθηκευμένες στα bitplanes, ή μια ενιαία τιμή του δείκτη που αναφέρεται σε ένα πίνακα αναζήτησης χρωμάτων να είναι επίσης αποθηκευμένη. Η RGBA color-display κατάσταση είναι η πιο συχνά χρησιμοποιούμενη.

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

glClearColor (0.0, 0.0, 0.0, 0.0)?

glClear (GL_COLOR_BUFFER_BIT)?

Η πρώτη γραμμή ορίζει το χρώμα σε μαύρο, και η επόμενη εντολή καθαρίζει ολόκληρο το παράθυρο για το τρέχον χρώμα του καθορισμού χρώματος. Η μοναδική παράμετρος της glClear () , δείχνει ποιοί buffers πρέπει να καθαριστούν. Στην περίπτωση αυτή, το πρόγραμμα ανοίγει μόνο το buffer χρώματος, της εικόνας που εμφανίζεται στην οθόνη. Συνήθως, μπορείτε να ορίσετε το χρώμα μια φορά, στην αρχή της εφαρμογής, και στη συνέχεια θα καταργήσετε τους buffers όσο συχνά χρειάζεται. Η OpenGL παρακολουθεί το τρέχον χρώμα καθαρισμού ως μεταβλητή κατάσταση.

Για παράδειγμα, για να καθαρίσετε το χρώμα του buffer και το βάθος του, θα χρησιμοποιούσατε την παρακάτω ακολουθία εντολών:

glClearColor (0.0, 0.0, 0.0, 0.0)?

glClearDepth (1.0)?

glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)?

Σε αυτή την περίπτωση, η κλήση της glClearColor () είναι η ίδια όπως και πριν, η glClearDepth () εντολή καθορίζει την τιμή του βάθους κάθε pixel του buffer, και η παράμετρος της glClear () εντολής, αποτελείται τώρα από την bitwise OR λογική όλων των buffers που πρέπει να καθαριστούν. Η ακόλουθη σύντομη παρουσίαση των glClear () περιλαμβάνει έναν πίνακα που παραθέτει τους buffers που μπορούν να καθαριστούν και τα ονόματά τους.

void glClearColor (GLclampf red, green GLclampf, GLclampf blue,GLclampf alpha)

Ορίζει το τρέχον χρώμα του συμψηφισμού για χρήση σε εκκαθάριση buffer χρώματος σε RGBA λειτουργία. Το κόκκινο, πράσινο, μπλε, και η τιμή άλφα συμψηφίζονται εάν είναι απαραίτητο για την περιοχή [0, 1]. Το προεπιλεγμένο χρώμα εκκαθάρισης είναι (0, 0, 0, 0), το οποίο είναι το μαύρο.

void glClear (GLbitfield mask)?

Καθαρίζει τους buffers σε τρέχουσες τιμές εκκαθάρισης. Το όρισμα «μάσκα», είναι μια bitwise λογική ή ο συνδυασμός των τιμών που αναφέρονται στον Πίνακα 2-1.

Buffer	Name	Reference
Color buffer	GL_COLOR_BUFFER_BIT	Chapter 4
Depth buffer	GL_DEPTH_BUFFER_BIT	Chapter 10
Accumulation buffer	GL_ACCUM_BUFFER_BIT	Chapter 10
Stencil buffer	GL_STENCIL_BUFFER_BIT	Chapter 10

Table 2-1 Clearing Buffers

Πριν από την δήλωση μιας εντολής για να καθαρίσετε πολλαπλούς buffers, θα πρέπει να ορίσετε τις τιμές στις οποίες κάθε buffer πρέπει να καθαριστεί, αν θέλετε κάτι διαφορετικό από το προεπιλεγμένο RGBA χρώμα, τιμή βάθους, χρώμα , και stencil δείκτη. Εκτός από τις glClearColor () και glClearDepth () εντολές που καθορίζουν τις τρέχουσες τιμές για την

εκκαθάριση του χρώματος και το βάθος των buffers,οι glClearIndex (), glClearAccum (), και glClearStencil () προσδιορίζουν το χρώμα, το χρώμα των buffers, και το stencil δείκτη που χρησιμοποιείται για να καταργήσετε τον αντίστοιχο buffer.

Η OpenGL σας επιτρέπει να ορίσετε πολλαπλούς buffers επειδή ο καθαρισμός είναι γενικά μια αργή λειτουργία .Όταν το υλικό δεν υποστηρίζει ταυτόχρονο καθαρισμό, ο καθαρισμός γίνεται διαδοχικά. Η διαφορά μεταξύ

glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)?

και

glClear (GL_COLOR_BUFFER_BIT)?

glClear (GL_DEPTH_BUFFER_BIT)?

είναι ότι αν και οι δύο έχουν το ίδιο τελικό αποτέλεσμα, το πρώτο παράδειγμα θα μπορούσε να τρέχει πιο γρήγορα σε πολλά μηχανήματα. Σίγουρα δεν θα τρέξει πιο αργά.

Ο καθορισμός ενός χρώματος

Με την OpenGL, η περιγραφή του σχήματος ενός αντικειμένου στο στάδιο της επεξεργασίας είναι ανεξάρτητη από την περιγραφή του χρώματος του. Κάθε φορά που ένα συγκεκριμένο γεωμετρικό αντικείμενο σχεδιάζεται, καθορίζεται ο τρέχοντας χρωματισμός. Το σύστημα χρωματισμού μπορεί να είναι απλό ή αρκετά περίπλοκο. Σε γενικές γραμμές, ένας προγραμματιστής OpenGL θέτει πρώτα το χρώμα ή το σύστημα χρωματισμού και στη συνέχεια σχεδιάζει τα αντικείμενα. Μέχρι το χρώμα ή το σύστημα χρωματισμού να αλλάξει, όλα τα αντικείμενα έχουν αυτό το χρώμα ή χρησιμοποιούν το σύστημα με αυτό το χρωματισμό. Αυτή η μέθοδος βοηθά την OpenGL στην επίτευξη υψηλότερων επιδόσεων στη σχεδίαση από ό, τι θα προέκυπτε αν δεν παρακολουθείται το τρέχον χρώμα.

Για παράδειγμα, ο ψευδοκώδικας

set_current_color (κόκκινο)?

draw_object (A)?

draw_object (B)?

set_current_color (πράσινο)?

set_current_color (μπλε)?

draw_object (C)?

χρωματίζει τα αντικείμενα Α και Β στο κόκκινο, και το Γ αντικείμενο με μπλε χρώμα. Η εντολή για την τέταρτη γραμμή που ορίζει το τρέχον χρώμα σε πράσινο χάνεται.

Για να ορίσετε ένα χρώμα, χρησιμοποιήστε την εντολή glColor3f (). Χρειάζονται τρεις παράμετροι, οι οποίες είναι όλες αριθμοί κινητής υποδιαστολής μεταξύ 0,0 και 1,0. Οι παράμετροι είναι, κατά σειρά, το κόκκινο, πράσινο και μπλε. Μπορείτε να συνδυάσετε αυτές τις τρεις τιμές που καθορίζουν ένα "μίγμα" των τριών χρωμάτων: Η τιμή 0.0 ,σημαίνει οτι δεν χρησιμοποιείται κάποιο από αυτά τα στοιχεία, και η τιμή 1,0 ότι μπορούν να τα χρησιμοποιηθούν όλα.

Έτσι, ο κώδικας

glColor3f (1.0, 0.0, 0.0)?

δημιουργεί το πιο φωτεινό κόκκινο που το σύστημα μπορεί να χρωματίσει, χωρίς πράσινα ή μπλε συστατικά. Όλα τα μηδενικά κάνουν το μαύρο. Αντίθετα ,όλα μονάδες κάνουν το

λευκό. Αν ρυθμιστούν και οι τρεις συνιστώσες στο 0,5 ,αποδίδουν το γκρι (ενδιάμεσα μεταξύ μαύρου και λευκού). Εδώ υπάρχουν οκτώ εντολές και τα χρώματα που θα ορίσετε:

glColor3f (0.0, 0.0, 0.0)? / * black * /

glColor3f (1.0, 0.0, 0.0)? / * κόκκινο * /

glColor3f (0.0, 1.0, 0.0)? / * πράσινο * /

glColor3f (1.0, 1.0, 0.0)? / * κίτρινο * /

glColor3f (0.0, 0.0, 1.0)? / * μπλε * /

glColor3f (1.0, 0.0, 1.0)? / * ματζέντα * /

glColor3f (0.0, 1.0, 1.0)? / * κυανό * /

glColor3f (1,0, 1,0, 1,0)? / * λευκό * /

Μπορεί να έχετε παρατηρήσει νωρίτερα ότι η ρουτίνα για τον καθορισμό του χρώματος , η glClearColor (), παίρνει τέσσερις παραμέτρους, οι τρεις πρώτες εκ των οποίων ταιριάζουν με τις παραμέτρους για την glColor3f (). Η τέταρτη παράμετρος είναι η τιμή άλφα. Σε αυτό το παράδειγμα η τέταρτη παράμετρος της glClearColor () είναι η 0,0, η οποία είναι η προκαθορισμένη.

Ο εξαναγκασμός Ολοκλήρωσης της σχεδίασης

Όπως είδατε στο "Pipeline Rendering OpenGL" ,τα πιο σύγχρονα συστήματα γραφικών μπορούν να θεωρηθούν ως μια γραμμή assembly. Η κεντρική μονάδα επεξεργασίας (CPU) εκτελεί μια εντολή σχεδίασης. Πιθανόν το υπόλοιπο υλικό να εκτελεί γεωμετρικούς μετασχηματισμούς. Η απόρριψη εκτελείται, ακολουθούμενη από τη σκίαση και / ή την χαρτογράφηση. Τέλος, οι τιμές αναγράφονται στα bitplanes της οθόνης. Στις high-end αρχιτεκτονικές, κάθε μία από τις πράξεις αυτές εκτελούνται από ένα διαφορετικό κομμάτι του υλικού που είναι ήδη σχεδιασμένο για να εκτελεί συγκεκριμένα τμήματα του έργου πιο γρήγορα. Σε μια τέτοια αρχιτεκτονική, δεν υπάρχει καμία ανάγκη για την CPU να περιμένει κάθε σχέδιο εντολής για να ολοκληρωθεί πριν να εκτελέσει την επόμενη. Ενώ η CPU στέλνει ένα vertex διαμέσου του αγωγού, το υλικό μετασχηματισμού εργάζεται για τη

μετατροπή του τελευταίου που έχει δοθεί και τα προηγούμενα αποτρέπονται, και ούτω καθεξής. Σε ένα τέτοιο σύστημα, αν η CPU περίμενε για κάθε εντολή για να ολοκληρωθεί πριν από την έκδοση της επόμενης, θα μπορούσε να υπάρξει μια τεράστια καθυστέρηση .Επιπλέον, η εφαρμογή μπορεί να τρέχει σε περισσότερες από μία μηχανές. Για παράδειγμα, ας υποθέσουμε ότι το κύριο πρόγραμμα τρέχει αλλού (σε μηχάνημα του πελάτη) και ότι προβάλλονται τα αποτελέσματα του σχεδίου στο σταθμό εργασίας σας ή τερματικό σταθμό (server), ο οποίος συνδέεται με δίκτυο προς τον πελάτη. Στην περίπτωση αυτή, μπορεί να είναι τρομακτικά αργό για να στείλετε κάθε εντολή μέσω του δικτύου. Συνήθως, ο πελάτης συγκεντρώνει μια συλλογή εντολών σε ένα ενιαίο πακέτο δικτύου πριν από την αποστολή. Δυστυχώς, ο κώδικας δικτύου του πελάτη συνήθως δεν έχει κανένα τρόπο ωστε να καταλάβεις ότι το πρόγραμμα γραφικών έχει τελειώσει τη σχεδίαση ενός πλαισίου ή σκηνής. Στη χειρότερη περίπτωση, θα περιμένει για πάντα .

Για το λόγο αυτό, η OpenGL παρέχει τη glFlush εντολή (), η οποία αναγκάζει τον πελάτη να στείλει το πακέτο δικτύου ακόμα και αν δεν είναι πλήρες. Όπου δεν υπάρχει δίκτυο και όλες οι εντολές εκτελούντα στο διακομιστή, η glFlush () δεν θα είχε αποτέλεσμα. Ωστόσο, εάν γράφετε ένα πρόγραμμα που θέλετε να λειτουργήσει σωστά και και χωρίς δίκτυο, συμπεριλάβετε μια κλήση glFlush () στο τέλος κάθε πλαισίου ή σκηνής. Να σημειωθεί ότι η glFlush () δεν περιμένει την προηγούμενη εκτέλεση να ολοκληρωθεί- αλλά αναγκάζει για να αρχίσει αμέσως την εκτέλεση, διασφαλίζοντας έτσι ότι όλες οι προηγούμενες εντολές εκτελέστηκαν σε πεπερασμένο χρονικό διάστημα, ακόμη και αν δεν έχει εκτελεστεί καμία εντολή καθορισμού.

Υπάρχουν και άλλες καταστάσεις στις οποίες η glFlush () είναι χρήσιμη:

• Renderers λογισμικού που χτίζουν τις εικόνες στη μνήμη του συστήματος και δεν απαιτείται να ενημερώνεται συνεχώς η οθόνη.

• Εφαρμογές που συγκεντρώνουν σύνολα εντολών καθιστώντας την απόσβεση κόστους εκκίνησης. Το παραπάνω παράδειγμα μεταφοράς του δικτύου είναι μία παρουσίαση αυτών.

Void glFlush (void)?

Σε προηγούμενες εκδόσεις της OpenGL,οι εντολές που ξεκινούν την εκτέλεση με τρόπο ώστε να διασφαλίζουν πληρότητα σε πεπερασμένο χρόνο. Μερικές εντολές-για παράδειγμα, εντολές για buffer swap σε Doublebuffer αυτόματη λειτουργία ή flush-εν αναμονή εντολές στο δίκτυο.

Αν η glFlush () δεν είναι επαρκής για σας, προσπαθήστε με την glFinish (). Αυτή η εντολή αδειάζει το δίκτυο ως glFlush () και στη συνέχεια περιμένει για την κοινοποίηση από το υλικό γραφικών ή του δικτύου που αναφέρει ότι το σχέδιο έχει ολοκληρωθεί σ τον framebuffer. Ίσως χρειαστεί να χρησιμοποιήσετε την glFinish () αν θέλετε να συγχρονίσετε εργασίες-για παράδειγμα, για να βεβαιωθείτε ότι η τρισδιάστατη σχεδίασή σας είναι στην οθόνη προτού χρησιμοποιήσετε PostScript Display για να επιστήσει ετικέτες στην κορυφή της απόδοσης. Ένα άλλο παράδειγμα για να διασφαλιστεί ότι το σχέδιο έχει ολοκληρωθεί πριν αρχίσει να δέχεται σχόλια των χρηστών. Μετά την εκτέλεση της glFinish () εντολής, οι διαδικασίες γραφικών σας είναι αποκλεισμένες μέχρι να λάβετε ειδοποίηση από το hardware γραφικών ότι η σχεδίαση έχει ολοκληρωθεί. Λάβετε υπόψην ότι η υπερβολική χρήση της glFinish () μπορεί να μειώσει την απόδοση της αίτησής σας, ειδικά αν τρέχετε σε ένα δίκτυο, επειδή απαιτεί μετ 'επιστροφής επικοινωνία. Αν η glFlush () είναι επαρκής για τις ανάγκες σας, χρησιμοποιήστε την αντί της glFinish ().

Void glFinish (void).

Αυτή η εντολή δεν επιστρέφει τίποτα , μέχρι όλα τα αποτελέσματα από προηγούμενες εντολές να έχουν πλήρως υλοποιηθεί.

Coordinate System Survival Kit

Κάθε φορά που ανοίγετε αρχικά ένα παράθυρο ή αργότερα, για να μετακινήσετε ή να αλλάξετε το μέγεθος του ,το σύστημα παραθύρων θα στείλει ένα γεγονός για να σας ειδοποιήσει. Εάν χρησιμοποιείτε GLUT, η κοινοποίηση είναι αυτοματοποιημένη και η ρουτίνα που έχει καταχωρηθεί στο glutReshapeFunc () θα καλεστεί. Πρέπει να σιγουρευτείτε για μια λειτουργία επανάκλησης ότι θα :

• αποκατασταθεί η ορθογώνια περιοχή που θα είναι η νέα απόδοση του καμβά.

• Καθορίστε το σύστημα συντεταγμένων με το οποίο τα αντικείμενα θα σχεδιαστούν.

Αυτή τη στιγμή δημιουργούμε ένα απλό, βασικό, δύο διαστάσεων σύστημα συντεταγμένων στο οποίο μπορείτε να σχεδιάσετε μερικά αντικείμενα. Γίνεται κλήση της glutReshapeFunc (reshape), reshape() είναι η ακόλουθη συνάρτηση που φαίνεται στο Παράδειγμα 2-1.

Παράδειγμα 2-1 Ανασχηματισμός Λειτουργίας Επανάκλησης

```
Void reshape (int w, int h)
{
glViewport (0, 0, (GLsizei) w, (GLsizei), h)?
glMatrixMode (GL_PROJECTION)?
glLoadIdentity ()?
gluOrtho2D (0,0, (GLdouble) w, 0.0, (GLdouble), h)?
}
```

Στον πυρήνας της GLUT θα περάσει αυτή η λειτουργία δύο ορίσματα : το πλάτος και το ύψος, σε pixel, ενός νέου παραθύρου, για μετακίνηση παραθύρου ή αλλαγή μεγέθους του. Η glViewport () ρυθμίζει το pixel ορθογώνιο για την σχεδίαση ,ώστε να είναι ολόκληρο το νέο παράθυρο. Οι επόμενες τρείς ρουτίνες ρυθμίζουν το σύστημα συντεταγμένων για την σχεδίαση, έτσι ώστε η κάτω αριστερή γωνία να είναι (0, 0) και η πάνω δεξιά γωνία να είναι (w, h) (βλ. Σχήμα 2-1). Για να το εξηγήσουμε με έναν άλλο τρόπο, σκεφτείτε ένα κομμάτι χαρτί γραφικών παραστάσεων. Η w και h τιμές στη reshape() αντιπροσωπεύουν τις στήλες και τις γραμμές των τετραγώνων στο χαρτί . Στη συνέχεια θα πρέπει να θέσει τους άξονες για το χαρτί γραφικών παραστάσεων. Η gluOrtho2D () ρουτίνα βάζει (0, 0), για το χαμηλότερο, πιο αριστερό τετράγωνο, και κάνει κάθε τετράγωνο να αντιπροσωπεύει μία μονάδα.

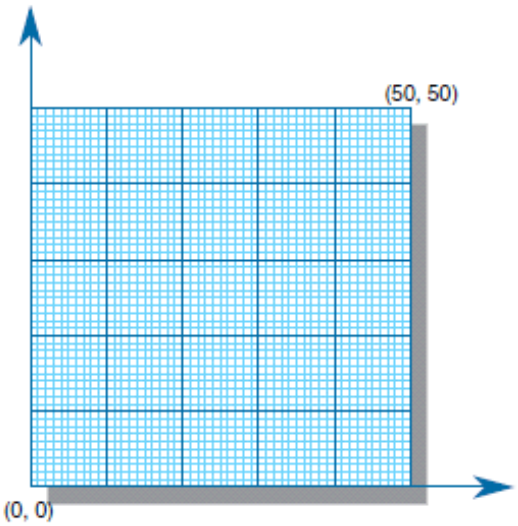


Figure 2-1 Coordinate System Defined by $w = 50$, $h = 50$

Περιγράφοντας τα σημεία, τις γραμμές και τα πολύγωνα

Έχετε πιθανώς μια αρκετά καλή ιδέα για το τι σημαίνουν μαθηματικά οι όροι σημείο, γραμμή, και πολύγωνο. Οι έννοιες αυτές στην OpenGL είναι παρόμοιες, αλλά όχι ακριβώς οι ίδιες. Μια διαφορά προέρχεται από τους περιορισμούς του υπολογιστή με βάση τους υπολογισμούς. Σε κάθε εφαρμογή OpenGL, οι floating-point υπολογισμοί είναι πεπερασμένης ακρίβειας, και έχουν στρογγυλοποίηση σφαλμάτων. Ως εκ τούτου, οι συντεταγμένες των OpenGL σημείων, γραμμών και πολύγωνων, υποφέρουν από τα ίδια προβλήματα. Μια πιο σημαντική διαφορά προκύπτει από τους περιορισμούς ενός raster γραφικού οθόνης. Σε μια τέτοια οθόνη, η μικρότερη προβαλλόμενη μονάδα είναι ένα pixel, και τα pixels μπορεί να είναι λιγότερο από το 1 / 100 της ίντσας σε εύρος, ή ακόμα πολύ μεγαλύτερο από ό, τι οι μαθηματικές έννοιες του απείρως μικρού (για σημεία) και του απείρως λεπτού (για γραμμές). Όταν η OpenGL εκτελεί υπολογισμούς, υποθέτει ότι τα σημεία απεικονίζονται ως φορείς αριθμών κινητής υποδιαστολής. Ωστόσο, ένα σημείο είναι συνήθως (αλλά όχι πάντα) ένα pixel, και πολλά διαφορετικά σημεία με ελαφρώς διαφορετικές συντεταγμένες θα μπορούσαν να αντληθούν από την OpenGL στο ίδιο pixel.

Σημεία

Ένα σημείο αντιπροσωπεύεται από μια σειρά από αριθμούς κινητής υποδιαστολής που ονομάζεται κορυφή. Όλοι οι υπολογισμοί γίνονται εσωτερικά σαν τρισδιάστατες κορυφές . Για κορυφές που καθορίζονται από το χρήστη ως δύο διαστάσεων (δηλαδή, με μόνο x-και y συντεταγμένες-coordinates), εφαρμόζεται από την OpenGL ως συντελεστής στάθμισης , z-συντεταγμένη ίση με το μηδέν.

Προηγμένα θέματα

Η OpenGL λειτουργεί με το ομοιογενές σύστημα συντεταγμένων των τρισδιάστατων σχημάτων της γεωμετρίας. Γ ια εσωτερικούς υπολογισμούς, όλες οι κορυφές εκπροσωπούνται με τέσσερις floating-point συντεταγμένες (x, y, z, w). Αν w είναι διαφορετικό από το μηδέν, αυτές οι συντεταγμένες αντιστοιχούν στο Ευκλείδειο, τρισδιάστατο σημείο (x / w, y / w, z / w). Μπορείτε να ορίσετε και τη w-συντεταγμένη στις OpenGL εντολές, αλλά αυτό γίνεται σπάνια. Αν η w-συντεταγμένη δεν προσδιορίζεται, τότε είναι 1.0 .

Γραμμές

Στην OpenGL, ο όρος γραμμή αναφέρεται σε ένα τμήμα της γραμμής, και όχι στη μαθηματική του εκδοχή που εκτείνεται στο άπειρο και στις δύο κατευθύνσεις. Υπάρχουν εύκολοι τρόποι για να καθορίσετε μια συνδεδεμένη σειρά τμημάτων γραμμής, ή ακόμα και μια κλειστή, συνδεδεμένη σειρά από τμήματα (βλ. Σχήμα 2-2). Σε όλες τις περιπτώσεις, όμως, οι γραμμές που αποτελούν συνδεδεμένη σειρά, καθορίζονται από την άποψη των κορυφών τους σε τελικά σημεία.

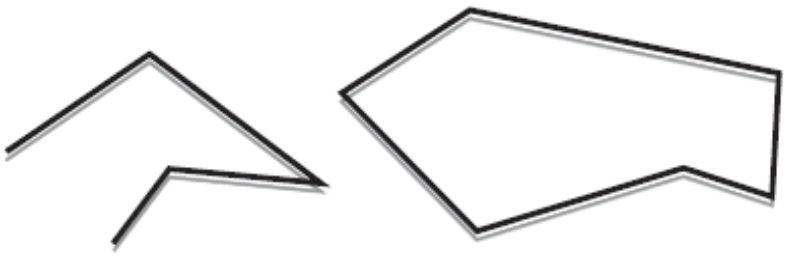


Figure 2-2 Two Connected Series of Line Segments

Πολύγωνα

Πολύγωνα είναι οι περιοχές που περικλείονται με ενιαίους κλειστούς βρόχους των τμημάτων γραμμών, οπου τα τμήματα της γραμμής, ορίζονται από τις κορυφές ,ως τελικά σημεία τους. Τα πολύγωνα συνήθως σχεδιάζονται με τα pixels , αλλά μπορούν επίσης να σχεδιαστούν από ένα σύνολο σημείων. Σε γενικές γραμμές, τα πολύγωνα μπορεί να είναι περίπλοκα, έτσι η OpenGL επιβάλλει κάποιους ισχυρούς περιορισμούς σχετικά με το οτι συνιστά ένα πρότυπο πολύγωνο. Πρώτον, τα άκρα των OpenGL πολυγώνων δεν μπορούν να τέμνονται (μαθηματικός θα μπορούσαμε να πούμε πως ένα απλό πολύγωνο πληροί την προϋπόθεση αυτή). Δεύτερον, τα OpenGL πολύγωνα πρέπει να να είναι κυρτά, που σημαίνει ότι δεν μπορούν να φέρουν οδοντώματα. Πιο συγκεκριμένα, μια περιοχή είναι κυρτή αν, δεδομένου ότι κάθε δύο σημεία στο εσωτερικό και ευθύγραμμο τμήμα που τα συνδέει είναι, επίσης, στο εσωτερικό. Βλ. Σχήμα 2-3 για μερικά παραδείγματα έγκυρων και μη έγκυρων πολυγώνων. Η OpenGL, όμως, δεν περιορίζει τον αριθμό των τμημάτων γραμμών που συνθέτουν τα όρια ενός κυρτού πολυγώνου. Να σημειωθεί επίσης ότι πολύγωνα με τρύπες, είναι μη αποδεκτά. Είναι nonconvex, και δεν μπορούν να να εξαχθούν με ένα όριο που αποτελείται από ένα μόνο κλειστό βρόχο. Θα πρέπει να γνωρίζετε ότι λόγος για τους OpenGL περιορισμούς στην εγκυρότητα τύπων πολυγώνων είναι η πιο απλή και γρήγορη σχεδίαση πολυγώνων από το υλικό . Απλά πολύγωνα μπορούν να αποδοθούν γρήγορα. Οι δύσκολες περιπτώσεις είναι δύσκολο να ανιχνευθούν γρήγορα.

Οι επιφάνειες στον πραγματικό κόσμο αποτελούνται από μη απλά πολύγωνα, nonconvex πολύγωνα, ή πολύγωνα με τρύπες. Από όλα αυτά τα πολύγωνα μπορούν να διαμορφωθούν τα συνδικάτα του απλού κυρτού πολυγώνου, κάποιες ρουτίνες για την κατασκευή είναι πιο πολύπλοκα αντικείμενα που παρέχονται στη βιβλιοθήκη GLU. Αυτές οι ρουτίνες περιέχουν σύνθετες περιγραφές και ταξιοθετήσεις τους, ή τα χωρίζουν σε ομάδες απλούστερων πολυγώνων OpenGL που μπορεί στη συνέχεια να σχεδιαστούν.

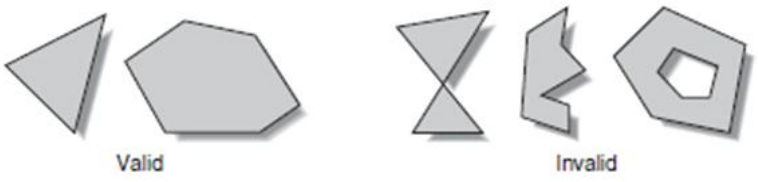


Figure 2-3 Valid and Invalid Polygons

Στην OpenGL οι κορυφές είναι πάντα τρισδιάστατες, τα σημεία που αποτελούν τα όρια ενός συγκεκριμένου πολυγώνου δεν πρέπει απαραίτητα να βρίσκονται στο ίδιο επίπεδο στο χώρο. (Φυσικά, το κάνουν σε πολλές περιπτώσεις-αν όλες οι z-συντεταγμένες είναι μηδέν, για παράδειγμα, ή αν το πολύγωνο είναι ένα τρίγωνο.) Εάν οι κορυφές ενός πολυγώνου δεν βρίσκονται στο ίδιο επίπεδο, στη συνέχεια, μετά από διάφορες περιστροφές στο διάστημα

,γίνονται αλλαγές στην οπτική γωνία, και προβολή, επί οθόνης, στα σημεία που θα μπορούσαν να μην αποτελούν ένα απλό κυρτό πολύγωνο. Για παράδειγμα, φανταστείτε ένα τεσσάρων σημείων τετράπλευρο, όπου τα σημεία είναι ελαφρώς έξω από το πλάνο. Μπορείτε να πάρετε ένα μη-απλό πολύγωνο που μοιάζει με παπιγιόν, όπως φαίνεται στο Σχήμα 2-4, το οποίο δεν είναι εγγυημένο οτι θα σχεδιαστεί σωστά. Η κατάσταση αυτή δεν είναι ασυνήθιστη, αν οι κατά προσέγγιση καμπύλες των επιφανειών από τετράπλευρα σημεία, προσεγγίζουν την αληθινή επιφάνεια. Μπορείτε πάντα να αποφύγετε το πρόβλημα με τη χρήση τριγώνων.

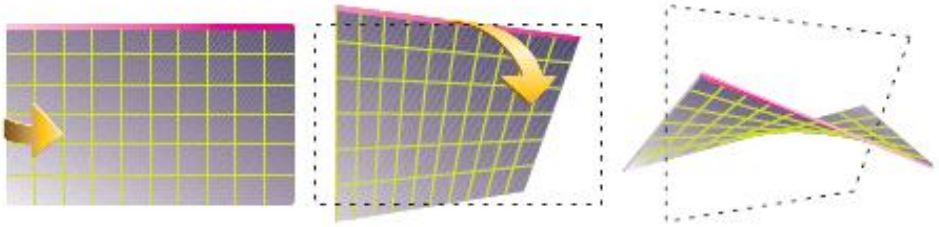


Figure 2-4 Nonplanar Polygon Transformed to Nonsimple Polygon

Ορθογώνια

Δεδομένου ότι τα ορθογώνια χρησιμοποιούνται πολύ συχνά σε εφαρμογές γραφικών,η OpenGL παρέχει ολοκληρωμένα πρότυπα σχεδίασης ορθογωνίων ,με τη glRect * ().

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

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

Σχεδιάζει το ορθογώνιο που ορίζεται από τα σημεία γωνιών (x1, y1) και (x2, y2). Το ορθογώνιο βρίσκεται στο επίπεδο z = 0 και έχει τις πλευρές παράλληλες με τους x-και y-άξονες. Αν χρησιμοποιείται μορφή διανύσματος στη συνάρτηση , οι γωνίες αποτελούν δύο δείκτες σε συστοιχίες, καθένας από τους οποίους περιέχει ένα ζεύγος (x, y).

Επέκταση Συμβατότητα

glRect

Σημειώστε ότι αν και το ορθογώνιο ξεκινά με ένα συγκεκριμένο προσανατολισμό στον τρισδιάστατο χώρο (στο xy-επίπεδο και τους άξονες), μπορείτε να αλλάξετε αυτό με την εφαρμογή περιστροφών ή άλλων μετατροπών.

Μπορεί να προσεγγιστεί οποιαδήποτε ομαλή καμπύλη γραμμής ή επιφάνειας , σε οποιαδήποτε αυθαίρετη ακρίβεια-από σύντομα τμήματα γραμμής ή μικρές πολυγωνικές περιοχές. Έτσι, υποδιαιρούνται καμπύλες γραμμές και επιφάνειες επαρκώς και στη συνέχεια να προσεγγίζονται με ευθύγραμμα τμήματα ή κατ 'αποκοπή πολυγώνου τα κάνει να φαίνονται καμπύλες (βλ. Σχήμα 2-5). Αν είστε δύσπιστοι ότι αυτό λειτουργεί πραγματικά, φανταστείτε ότι υποδιαιρώντας κάθε τμήμα γραμμής ή ενός πολυγώνου, γίνεται τόσο μικρό, που είναι μικρότερο από ένα pixel στην οθόνη.

Figure 2-5 Approximating Curves

Σχήμα 2-5. Προσέγγιση των Curves

Παρά το γεγονός ότι οι καμπύλες δεν είναι γεωμετρικά πρότυπα, η OpenGL παρέχει κάποια άμεση υποστήριξη για την κατάτμηση και την σχεδίασή τους.

Καθορισμός Κορυφών

Με την OpenGL, κάθε γεωμετρικό αντικείμενο, περιγράφεται ως ένα διατεταγμένο σύνολο κορυφών. Μπορείτε να χρησιμοποιήσετε την glVertex * () εντολή για να καθορίσετε μια κορυφή.

void glVertex [234]} {sifd (TYPE coords)?

void glVertex [234]} {sifd v (const TYPE * coords)?

Καθορίζει μια κορυφή για ένα γεωμετρικό αντικείμενο. Μπορείτε να ορίσετε έως και τέσσερις συντεταγμένες (x, y, z, w) για μία συγκεκριμένη κορυφή ή μόνο δύο (x, y), επιλέγοντας την κατάλληλη έκδοση της εντολής. Εάν χρησιμοποιείτε μια έκδοση που δεν διευκρινίζει ρητώς z ή w, z ,τότε θεωρείται ότι είναι 0, και w θεωρείται ότι είναι 1.Οι κλήσεις προς την glVertex * () είναι αποτελεσματικές μόνο μεταξύ ενός glBegin () και glEnd () ζεύγους.

Επέκταση Συμβατότητα

glVertex

glVertex2s (2, 3)?

glVertex3d (0.0, 0.0, 3.1415926535898)?

glVertex4f (2.3, 1.0, -2.2, 2.0)?

GLdouble dvect [3] = {5.0, 9.0, 1992.0}?

glVertex3dv (dvect)?

Το πρώτο παράδειγμα αποτελεί μια κορυφή με τρισδιάστατες συντεταγμένες (2, 3, 0). (Θυμηθείτε ότι, αν δεν έχει καθοριστεί, η z-συντεταγμένη τότε είναι 0.) Οι συντεταγμένες στο δεύτερο παράδειγμα είναι (0.0, 0.0, 3.1415926535898) (διπλής ακρίβειας αριθμοί κινητής υποδιαστολής). Το τρίτο παράδειγμα αποτελεί την κορυφή με τρισδιάστατες συντεταγμένες (1,15, 0,5, -1,1) σε μία ομοιογενή συντεταγμένη. (Θυμηθείτε ότι οι x-, y-και z-συντεταγμένες έχουν διαιρεθεί με τη w-συντεταγμένη.) Στο τελευταίο παράδειγμα, η dvect είναι ένας δείκτης σε μια σειρά από τρεις, διπλής-ακρίβειας, αριθμούς κινητής υποδιαστολής.

Σε κάποια συστήματα, η μορφή διανύσματος της glVertex * () είναι πιο αποτελεσματική, δεδομένου ότι μόνο μια παράμετρος χρειάζεται να περάσει στο υποσύστημα γραφικών. Ειδικό υλικό θα μπορούσε να είναι σε θέση να στείλει μια ολόκληρη σειρά από συντεταγμένες σε μία μόνο παρτίδα. Αν το μηχάνημά σας είναι σαν αυτό, είναι προς όφελός σας για να τακτοποιήσετε τα δεδομένα σας, ώστε να είναι στην κορυφή συντεταγμένες που συσκευάζονται διαδοχικά στη μνήμη. Στην περίπτωση αυτή, μπορεί να υπάρχουν κάποια οφέλη στην απόδοση.

Τώρα που έχετε δει τον τρόπο καθορισμού κορυφών, θα πρέπει ακόμα να ξέρετε πώς να πείτε στην OpenGL να δημιουργήσει ένα σύνολο σημείων, μια γραμμή ή ένα πολύγωνο από αυτές τις κορυφές. Για να το κάνετε αυτό, βασιστείτε σε κάθε σύνολο κορυφών μεταξύ μιας κλήσης της glBegin () και μιας κλήσης για glEnd (). Το όρισμα που πέρασε στην glBegin () καθορίζει τι είδους γεωμετρικά πρότυπα κατασκευάζονται από τις κορυφές. Για παράδειγμα, το παράδειγμα 2-3 προσδιορίζει τις κορυφές για το πολύγωνο που φαίνεται στο Σχήμα 2-6.

Παράδειγμα 2-3. Γεμισμένο Πολύγωνο

glBegin (GL_POLYGON)?

  glVertex2f (0.0, 0.0)?

  glVertex2f (0.0, 3.0)?

  glVertex2f (4.0, 3.0)?

  glVertex2f (6.0, 1.5)?

  glVertex2f (4.0, 0.0)?

glEnd ()?

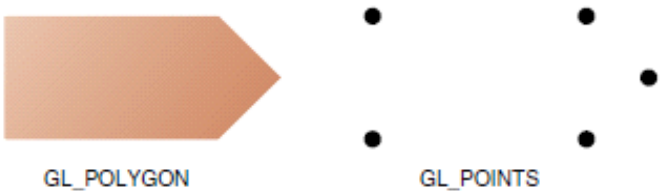


Figure 2-6 Drawing a Polygon or a Set of Points

Σχήμα 2-6. Αντλώντας ένα πολύγωνο ή ένα σύνολο σημείων

Αν είχε χρησιμοποιηθεί η GL_POINTS αντί της GL_POLYGON, το πρωτότυπο θα ήταν απλά τα πέντε σημεία που φαίνονται στο Σχήμα 2-6. Ο πίνακας 2-2 στην ακόλουθη δείχνει το πως η glBegin () απαριθμεί τα 10 πιθανά ορίσματα και τους αντίστοιχους τύπους των πρωτοτύπων.

Αξία	**Σημασία**
GL_POINTS	Ξεχωριστά σημεία
GL_LINES	Ζεύγη κορυφών ερμηνευμένες ως μεμονωμένα τμήματα γραμμής
GL_LINE_STRIP	Σειρά συνδεδεμένων ευθύγραμμων τμημάτων
GL_LINE_LOOP	Όπως το παραπάνω, με ένα τμήμα να προστίθεται μεταξύ της τελευταίας και της πρώτης κορυφής
GL_TRIANGLES	Τριπλές κορυφές που ερμηνεύονται ως τρίγωνα
GL_TRIANGLE_STRIP	Συνδεδεμένη λωρίδα τριγώνων
GL_TRIANGLE_FAN	Συνδεδεμένο fan τριγώνων
GL_QUADS	Τετράδες κορυφών που ερμηνεύονται ως τετράπλευρα πολύγωνα
GL_QUAD_STRIP	Συνδεδεμένη λωρίδα τετράπλευρών
GL_POLYGON	Boundary ενός απλού, κυρτού πολύγωνου

Πίνακας 2-2. Γεωμετρικά πρωτότυπα ,Ονόματα και έννοιες

```
void  glBegin (GLenum mode);
```

Σηματοδοτεί την έναρξη ενός κόμβου-δεδομένων της λίστας που περιγράφει ένα γεωμετρικό πρότυπο. Ο τύπος των προτύπων υποδεικνύεται από την κατάσταση, στην οποία μπορεί να είναι οποιαδήποτε από τις τιμές που φαίνονται στο πιο πάνω παράδειγμα.

glBegin

GL_QUADS

GL_QUAD_STRIP

GL_POLYGON

void glEnd (void)? -> Σηματοδοτεί το τέλος ενός κόμβου-στοιχείων λίστας.

Επέκταση Συμβατότητα

Το σχήμα 2-7 δείχνει παραδείγματα όλων των γεωμετρικών προτύπων που παρατίθενται στον Πίνακα 2-2, με περιγραφές των pixels που χρησιμοποιούνται για κάθε ένα από τα αντικείμενα. Σημειώστε ότι εκτός από τα σημεία, ορίζονται διάφοροι τύποι γραμμών και πολυγώνων. Προφανώς, μπορείτε να βρείτε πολλούς τρόπους για να σχεδιάσετε τα ίδια πρότυπα. Η μέθοδος που επιλέγεται εξαρτάται από την κορυφή των δεδομένων σας.

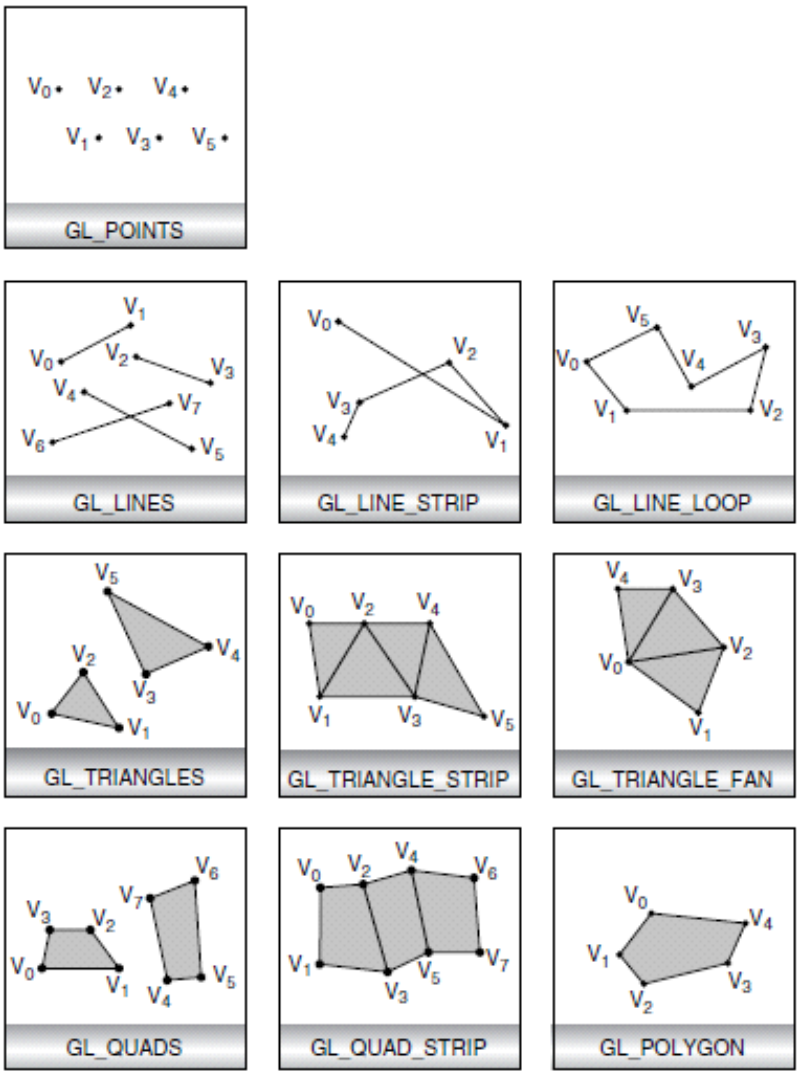


Figure 2-7 Geometric Primitive Types

Όπως μπορείτε να διαβάσετε στην ακόλουθη περιγραφή, ας υποθέσουμε ότι οι n κορυφές (v0, v1, v2, ..., VN-1) περιγράφονται από το glBegin () και glEnd () ζεύγος.

GL_POINTS Σχεδιάζει ένα σημείο σε κάθε μία από τις n κορυφές.

GL_LINES Σχεδιάζει μια σειρά ασύνδετων ευθύγραμμων τμημάτων. Τα τμήματα σχεδιάζονται μεταξύ v0 και v1, μεταξύ V2 και V3, και ούτω καθεξής. Εάν η n είναι περιττή, το τελευταίο τμήμα είναι η διάκριση μεταξύ VN-3 και VN-2, και το VN-1 δεν λαμβάνεται υπόψη.

GL_LINE_STRIP Σχεδιάζει μια γραμμή τμήματος από v0 να v1, έπειτα από V1 σε V2, και ούτω καθεξής, τελικά σχεδιάζει το τμήμα από το VN-2 στο VN-1. Έτσι, συνολικά n - σχεδιάζεται 1 ευθύγραμμο τμήμα. Τίποτα δεν σχεδιάζεται, αν το n είναι μεγαλύτερο από 1. Δεν υπάρχουν περιορισμοί σχετικά με τις κορυφές που περιγράφουν μια λωρίδα γραμμή (ή ένα βρόχο γραμμής). Οι γραμμές μπορούν να τέμνονται αυθαίρετα.

GL_LINE_LOOP Ίδια με την GL_LINE_STRIP, εκτός από το ότι το τελικό ευθύγραμμο τμήμα έχει ληφθεί από το VN-1 για το v0, συμπληρώνοντας ένα βρόχο.

GL_TRIANGLES Σχεδιάζει μια σειρά από τρίγωνα (τρίπλευρα πολύγωνα) με τη χρήση των κορυφών v0, v1, v2, μετά v3, v4, v5, και ούτω καθεξής. Αν το n δεν είναι πολλαπλάσιο του 3, η τελική μία ή δύο κορυφές, αγνοούνται.

GL_TRIANGLE_STRIP Σχεδιάζει μια σειρά από τρίγωνα (τρίπλευρα πολύγωνα) με τη χρήση των κορυφών, v0 v1, v2, τότε v2, v1, v3 (σημειώστε τη σειρά), στη συνέχεια, V2, V3, V4, και ούτω καθεξής. Ο σκοπός είναι να εξασφαλιστεί ότι τα τρίγωνα θα έχουν θα έχουν όλα τον ίδιο προσανατολισμό, έτσι ώστε η κατά τη σχεδίαση να μπορεί να γίνει σωστά ένα τμήμα μιας επιφάνειας. Η διατήρηση του προσανατολισμού είναι σημαντική για ορισμένες λειτουργίες, όπως η απόρριψη .

GL_TRIANGLE_FAN Ίδια με την GL_TRIANGLE_STRIP, εκτός του ότι οι κορυφές είναι οι v0, v1, v2, μετά v0, V2, V3, μετά v0, V3, V4, και ούτω καθ 'εξής (βλ. Σχήμα 2-7).

GL_QUADS Σχεδιάζει μια σειρά από τετράπλευρα (τετράπλευρα πολύγωνα) με τη χρήση των κορυφών v0, v1, v2, v3, v4 μετά, V5, V6, V7, και ούτω καθεξής. Αν το n δεν είναι πολλαπλάσιο του 4, οι τελικές μία, δύο, ή τρεις κορυφές, αγνοούνται.

GL_QUAD_STRIP Σχεδιάζει μια σειρά από τετράπλευρα (τετράπλευρα πολύγωνα) που αρχίζουν με v0, v1, v3, v2, στη συνέχεια, v2, v3, v5, v4, μετά v4, v5, V7, V6, και ούτω καθ 'εξής (βλ. Σχήμα 2-7) .Το n πρέπει να είναι τουλάχιστον 4 πριν σχεδιαστεί οτιδήποτε . Εάν το n είναι μονός αριθμός, η τελική κορυφή αγνοείται.

GL_POLYGON Σχεδιάζει ένα πολύγωνο με τα σημεία v0, ..., vn-1, όπως κορυφές. Το n πρέπει να είναι τουλάχιστον 3, ή να μην έχει σχεδιαστεί τίποτα. Επιπλέον, το συγκεκριμένο πολύγωνο δεν πρέπει να τέμνεται και πρέπει να είναι κυρτό. Εάν οι κορυφές δεν πληρούν αυτές τις προϋποθέσεις, τα αποτελέσματα είναι απρόβλεπτα.

Περιορισμοί στη χρήση της glBegin () και glEnd ()

Οι πιο σημαντικές πληροφορίες σχετικά με τις κορυφές είναι οι συντεταγμένες τους, οι οποίες καθορίζονται από την glVertex * () .Μπορεί να παρέχονται επίσης πρόσθετα στοιχεία, ειδικά για κάθε κορυφή-όπως ένα χρώμα, ένα κανονικό διάνυσμα, συντεταγμένες σχεδίασης, ή οποιοσδήποτε συνδυασμός αυτών, χρησιμοποιώντας ειδικές εντολές.

Επιπλέον, παρέχονται μερικές άλλες εντολές που χρησιμοποιούνται μεταξύ ενός ζεύγους glBegin () και glEnd (). Ο πίνακας 2-3 περιλαμβάνει μια πλήρη λίστα αυτών των έγκυρων εντολών.

Εντολή	Σκοπός εντολών
glVertex * ()	θέτει τις συντεταγμένες ενός κόμβου
glColor * ()	θέτει RGBA χρώμα
glIndex * (set)	θέτει χρώμα δείκτη
glSecondaryColor * ()	θέτει δευτερεύον χρώμα για post-texturing εφαρμογή
glNormal * ()	θέτει συντεταγμένες για κανονικό διάνυσμα
glMaterial * ()	θέτει ιδιότητες στοιχείων
glFogCoord * ()	θέτει τις συντεταγμένες της ομίχλης
glTexCoord * ()	θέτει texture συντεταγμένες
glMultiTexCoord * ()	θέτει texture συντεταγμένες για multitexturing
glVertexAttrib * ()	για γενικές ιδιότητες κόμβων
glEdgeFlag * ()	ελέγχει τις ακμές
glArrayElement ()	εξάγει τα δεδομένα του κόμβου(πίνακας)
glEvalCoord * (), glEvalPoint * ()	δημιουργεί συντεταγμένες
glCallList (), glCallLists ()	εκτελεί στην οθόνη λίστα (ες)

Πίνακας 2-3. Έγκυρες εντολές μεταξύ glBegin () και glEnd ()

Δεν υπάρχουν άλλες εντολές OpenGL που να ισχύουν μεταξύ ενός glBegin () και glEnd () ζεύγους, και κάνοντας άλλες κλήσεις η OpenGL παράγει ένα σφάλμα. Μερικές εντολές κορυφών πίνακα, όπως η glEnableClientState () και glVertexPointer (), όταν καλούνται μεταξύ glBegin () και glEnd (), έχουν απροσδιόριστη συμπεριφορά, αλλά δεν δημιουργούν κατ 'ανάγκη σφάλμα. (Επίσης, ρουτίνες που σχετίζονται με την OpenGL, όπως η GLX * () ,

έχουν απροσδιόριστη συμπεριφορά μεταξύ glBegin () και glEnd ().) Οι περιπτώσεις αυτές θα πρέπει να αποφεύγονται, καθώς ο εντοπισμός σφαλμάτων σ' αυτή την περίπτωση μπορεί να είναι πιο δύσκολος.

Σημειώστε ωστόσο ότι είναι οι μοναδικές εντολές OpenGL στις οποίες υπάρχουν περιορισμοί. Μπορείτε να συμπεριλάβετε σίγουρα άλλης γλώσσας προγραμματισμού-μεθόδους (εκτός απο κλήσεις, όπως στις προαναφερθείσες GLX * () ρουτίνες). Για παράδειγμα, το παράδειγμα 2-4 σχεδιάζει έναν κύκλο.

Παράδειγμα 2-4. Άλλες δομές μεταξύ glBegin () και glEnd ()

```
#define PI 3.1415926535898

GLint circle_points = 100;

glBegin(GL_LINE_LOOP);

for (i = 0; i < circle_points; i++) {

angle = 2*PI*i/circle_points;

glVertex2f(cos(angle), sin(angle));

}

glEnd();
```

Σημείωση

Το παράδειγμα αυτό δεν είναι ο πιο αποτελεσματικός τρόπος για να σχεδιάσετε έναν κύκλο, ειδικά εάν σκοπεύετε να το κάνετε κατ 'επανάληψη. Οι εντολές γραφικών που χρησιμοποιούνται είναι συνήθως πολύ γρήγορες, αλλά αυτός ο κώδικας υπολογίζει μια γωνία και καλεί τις sin() και cos () ρουτίνες για κάθε κορυφή. Επιπλέον, υπάρχει η επιβάρυνση βρόχου. (Ένας άλλος τρόπος για να υπολογιστούν οι κορυφές του κύκλου είναι να χρησιμοποιήσετε μια ρουτίνα GLU. Εάν πρέπει να σχεδιαστούν πολλοί κύκλοι, υπολογίστε τις συντεταγμένες των κορυφών μία φορά και αποθηκεύστε τις σε έναν πίνακα για να δημιουργηθεί μια λίστα οθόνης , ή να χρησιμοποιήσετε τους πίνακες κορυφών που τις παρέχουν.

Εκτός κι αν να συγκεντρώνονται σε μια λίστα οθόνης, όλες οι glVertex * () εντολές θα πρέπει να εμφανίζονται μεταξύ ενός glBegin () και glEnd () συνδυασμού. (Εάν εμφανίζονται

και αλλού, δεν έχει επιτευχθεί τίποτα). Σε περίπτωση που εμφανίζονται σε μια λίστα στην οθόνη, εκτελούνται μόνο εφόσον εμφανίζονται μεταξύ μιας glBegin () και μιας glEnd ().

Παρά το γεγονός ότι πολλές εντολές επιτρέπονται μεταξύ των glBegin () και glEnd (),οι κορυφές δημιουργούνται μόνο όταν η glVertex εντολή * () έχει εκδοθεί. Αυτή τη στιγμή η glVertex * () καλείται, η OpenGL εκχωρεί το τρέχον χρώμα για την κορυφή, τις texture συντεταγμένες, τις πληροφορίες του κανονικού διανύσματος, και ούτω καθεξής. Για να το δείτε αυτό, κοιτάξτε την ακόλουθη σειρά κώδικα. Το πρώτο σημείο είναι για αυτές με το κόκκινο, και το δεύτερο και τρίτο για αυτές με το μπλε χρώμα, παρά τις επιπλέον εντολές χρώματος:

```
glBegin (GL_POINTS)?
  glColor3f (0.0, 1.0, 0.0)? / * πράσινο * /
  glColor3f (1.0, 0.0, 0.0)? / * κόκκινο * /
  glVertex (...);
  glColor3f (1.0, 1.0, 0.0)? / * κίτρινο * /
  glColor3f (0.0, 0.0, 1.0)? / * μπλε * /
  glVertex (...);
  glVertex (...);
glEnd ()?
```

Μπορείτε να χρησιμοποιήσετε οποιοδήποτε συνδυασμό από τις 24 εκδόσεις της glVertex * () μεταξύ glBegin () και glEnd (), αν και σε πραγματικές εφαρμογές όλες οι κλήσεις σε κάθε συγκεκριμένη περίπτωση έχουν την τάση να είναι της ίδιας μορφής. Εάν τα δεδομένα της κορυφής για τις προδιαγραφές σας είναι συνεπή και επαναλαμβανόμενα (για παράδειγμα, glColor *, glVertex *, glColor *, glVertex *,...),αυτό μπορεί να βελτιώσει την απόδοση του προγράμματός σας, χρησιμοποιώντας πίνακες κορυφών.

Βασική Διαχείριση Καταστάσεων

Προηγουμένως, είδαμε ένα παράδειγμα μιας μεταβλητής κατάστασης, RGBA χρώματος, και πώς μπορεί να συσχετιστεί με ένα πρότυπο. Η OpenGL διατηρεί πολλές καταστάσεις και μεταβλητές κατάστασης. Ένα αντικείμενο μπορεί να αποδίδεται με φωτισμό, texturing, απόκρυψη, ομίχλη, και άλλες καταστάσεις που επηρεάζουν την εμφάνισή του. Από προεπιλογή, οι περισσότερες καταστάσεις είναι ανενεργές. Αυτές οι καταστάσεις μπορεί να είναι δαπανηρό να ενεργοποιηθούν. Για παράδειγμα, η ενεργοποίηση χαρτογράφησης θα είναι σχεδόν σίγουρα επιβραδυντική διαδικασία σχεδίασης προτύπου. Ωστόσο, η εικόνα θα βελτιωθεί σε ποιότητα και θα φαίνεται πιο ρεαλιστική, λόγω των ενισχυμένων δυνατοτήτων γραφικών.

Για να ενεργοποιήσετε και να απενεργοποιήσετε πολλές από αυτές τις καταστάσεις ,γίνεται η χρήση δύο απλών εντολών:

Μπορείτε επίσης να ελέγξετε αν μια κατάσταση μπορεί να ενεργοποιηθεί ή να απενεργοποιηθεί.

Οι καταστάσεις που θα δείτε διαθέτουν δύο ρυθμίσεις: on και off. Ωστόσο, οι περισσότερες OpenGL ρουτίνες καθορίζουν τιμές για τις μεταβλητές για πιο περίπλοκες καταστάσεις, για παράδειγμα, η glColor3f () θέτει τρεις τιμές, οι οποίες αποτελούν μέρος της GL_CURRENT_COLOR κατάστασης.

Υπάρχουν πέντε ερωτήματα ρουτίνες που χρησιμοποιούνται για την εξεύρεση των τιμών που έχουν οριστεί για πολλές καταστάσεις:

void glEnable (GLenum capability)

void glDisable (GLenum capability)

Η glEnable () ενεργοποιεί την ικανότητα, και η glDisable () την απενεργοποιεί. Περισσότερες από 60 ,απαριθμούνται οι τιμές που μπορούν να μεταβιβαστούν ως παράμετροι στις glEnable () και glDisable (). Μερικά παραδείγματα είναι η GL_BLEND (η οποία ελέγχει την ανάμειξη των RGBA τιμών), η GL_DEPTH_TEST (η οποία ελέγχει το βάθος, δηλαδή εκτελεί συγκρίσεις και ενημερώσεις για το βάθος buffer),η GL_FOG (η οποία ελέγχει την ομίχλη),η GL_LINE_STIPPLE (με σχέδια γραμμών), και η GL_LIGHTING (παίρνει την άποψη σε θέματα φωτισμού).

Η GLboolean glIsEnabled (GLenum capability)

Επιστρέφει GL_TRUE ή GL_FALSE, ανάλογα με το αν το ερώτημα έχει τη δυνατότητα αυτή τη στιγμή να ενεργοποιηθεί.

void glGetBooleanv (GLenum pname, GLboolean * params)?

void glGetIntegerv (GLenum pname, lampsi * params)?

void glGetFloatv (GLenum pname, GLfloat * params)?

void glGetDoublev (GLenum pname, GLdouble * params)?

void glGetPointerv (GLenum pname, GLvoid ** params)?

Οι παραπάνω, αποκτούν Boolean, integer, κινητής υποδιαστολής, διπλής ακρίβειας, ή δείκτη σε μεταβλητές κατάστασης. Το όρισμα pname είναι μια συμβολική σταθερά ένδειξης της μεταβλητής κατάστασης που επιστρέφεται, και params είναι ένας δείκτης σε μια σειρά από τον αναφερόμενο τύπο στον οποίο θα τοποθετήσετε τα δεδομένα που επιστρέφονται. Αυτές οι ρουτίνες, μπορούν να χειριστούν τα περισσότερα, αλλά όχι, όλα τα αιτήματα για τη λήψη πληροφοριών κατάστασης.

Εμφάνιση σημείων, γραμμών και πολυγώνων

Λεπτομέρειες Σημείων

Για να ελέγξετε το μέγεθος ενός σημείου, γίνεται χρήση της glPointSize () για την επιστροφή του επιθυμητού μεγέθους σε pixels.

void glPointSize (GLfloat size)

Ορίζει το πλάτος σε pixels για τα επιστρεφόμενα σημεία. Το μέγεθος πρέπει να είναι μεγαλύτερο από 0,0 και η προεπιλογή είναι 1,0 .Η πραγματική συλλογή από pixel στην οθόνη που είναι σχεδιασμένη για διάφορα σημεία πλάτους εξαρτάται από το αν είναι ενεργοποιημένο το antialiasing. (Antialiasing είναι μια τεχνική για την εξομάλυνση γραμμών και σημείων όπου και αν βρίσκονται. Εάν το antialiasing είναι απενεργοποιημένο (η προεπιλογή), το κλασματικό πλάτος, στρογγυλοποιείται σε ακέραιο πλάτος, και σχεδιάζεται στην οθόνη μια ευθυγραμμισμένη τετραγωνική περιοχή των pixel. Έτσι, εάν το πλάτος είναι 1,0, το τετράγωνο είναι 1 pixel από 1 pixel. εάν το πλάτος είναι 2,0, το τετράγωνο είναι 2 pixels από 2 pixels και ούτω καθεξής.

Με ενεργοποιημένα τα antialiasing ή multisampling ,σχεδιάζεται μια κυκλική ομάδα των pixels, και τα περιφερειακά pixels σχεδιάζονται συνήθως με χαμηλότερη πυκνότητα ώστε να δίνουν την αίσθηση ομαλότερης εμφάνισης στα άκρα. Σε αυτή τη λειτουργία, μη ακέραια πλάτη δεν στρογγυλεύονται. Οι περισσότερες εφαρμογές OpenGL υποστηρίζουν πολύ μεγάλου μεγέθους σημεία. Μπορείτε να κάνετε ερωτήματα για το ελάχιστο και το μέγιστο μέγεθος για aliased σημεία με τη χρήση της GL_ALIASED_POINT_SIZE_RANGE της glGetFloatv (). Ομοίως, μπορείτε να αποκτήσετε το εύρος των υποστηριζόμενων μεγεθών για να εξομαλύνονται τα σημεία, με την GL_SMOOTH_POINT_SIZE_RANGE της glGetFloatv (). Τα μεγέθη των υποστηριζόμενων antialising σημείων,κατανέμονται ομοιόμορφα ανάμεσα στο ελάχιστο και το μέγιστο μέγεθος για την περιοχή. Η κλήση της glGetFloatv () με την παράμετρο GL_SMOOTH_POINT_SIZE_GRANULARITY θα επιστρέψει με πόση ακρίβεια ένα συγκεκριμένο μέγεθος antialiased σημείο υποστηρίζεται. Για παράδειγμα, εάν ζητήσετε glPointSize (2.37) και επιστραφεί granularity ίσο με 0,1, τότε το μέγεθος του σημείου στρογγυλοποιείται σε 2.4.

Λεπτομέρειες γραμμής

Με την OpenGL, μπορείτε να καθορίσετε τις γραμμές με διαφορετικά πλάτη και να δημιουργήσετε γραμμές που είναι σχεδιασμένες με διάφορους τρόπους, διακεκομμένες, που εναλλάσσονται με κουκίδες και παύλες, και ούτω καθεξής.

Εύρος Γραμμών

Η πραγματική απόδοση των γραμμών επηρεάζεται εάν το antialiasing ή το multisampling είναι ενεργοποιημένα. Χωρίς antialiasing, με το πλάτος 1, 2, και 3 σχεδιάζονται γραμμές των 1, 2, και 3 pixels πλάτους. Με το antialiasing ενεργοποιημένο, μη ακέραια πλάτη γραμμών είναι πιθανά, και τα pixels στα περιθώρια είναι συνήθως σχεδιασμένα σε λιγότερη από την πλήρη ένταση. Όπως συμβαίνει και με το μέγεθος των σημείων, μια συγκεκριμένη εφαρμογή OpenGL μπορεί να περιορίζει το πλάτος των non antialiased γραμμών στο μέγιστο antialiased πλάτος της γραμμής, στρογγυλοποιώντας στον πλησιέστερο ακέραιο

αριθμό. Μπορείτε να αποκτήσετε το εύρος των υποστηριζόμενων πλατών γραμμών, με τη χρήση της GL_ALIASED_LINE_WIDTH_RANGE της glGetFloatv ().Για τον προσδιορισμό του ελάχιστου και του μέγιστου μεγέθους τών antialiased πλατών γραμμής,που η granularity εφαρμογή σας υποστηρίζει, καλέστε την glGetFloatv (), της GL_SMOOTH_LINE_WIDTH_RANGE και GL_ SMOOTH_LINE_WIDTH_GRANULARITY.

Σημείωση: Λάβετε υπόψη ότι, από προεπιλογή, οι γραμμές με 1 pixel πλάτος, εμφανίζονται γενικότερα με χαμηλότερη ανάλυση στις οθόνες. Για τις οθόνες υπολογιστών, αυτό δεν είναι συνήθως πρόβλημα, αλλα αν κάνετε χρήση OpenGL που χρησιμοποιεί highresolution plotter, 1-pixel γραμμές, μπορεί να είναι σχεδόν αόρατες. Για να αποκτήσετε ανάλυση-ανεξάρτητα πλάτους γραμμής, θα πρέπει να λαμβάνεται υπόψη τις φυσικές διαστάσεις των pixel.

Προηγμένα θέματα

Με μη antialiased εύρος γραμμών, το πλάτος της γραμμής δεν προσμετράται καθέτως στη γραμμή. Αντ' αυτού, είναι υπολογισμένη στην y-κατεύθυνση, εάν η απόλυτη τιμή της κλίσης είναι μικρότερη από 1,0.Διαφορετικά, είναι μετρημένη στη x-κατεύθυνση. Η απόδοση μιας antialiased γραμμής είναι ακριβώς ισοδύναμη με την παροχή ενός γεμάτου ορθογώνιο με συγκεκριμένο πλάτος, με επίκεντρο την ακριβή γραμμή.

Stripled Lines

Για να συνθέσετε γραμμές, με κουκκίδες ή με παύλες, μπορείτε να χρησιμοποιήσετε την εντολή glLineStipple () για να καθορίσει τη σχεδίαση με κουκίδες στο μοτίβο, και στη συνέχεια να ενεργοποιήσετε τη γραμμή με τη glEnable ().

glLineStipple (1, 0x3F07)?

glEnable (GL_LINE_STIPPLE)?

Η void glLineStipple(GLint factor, GLushort pattern),ορίζει το τρέχον striple για τις γραμμές. Το όρισμα μοτίβου, είναι μια 16-bit σειρά από 0 και 1, και χρησιμοποιείται όσες φορές χρειάζεται για να σχεδιαστεί με κουκίδες, μια δεδομένη γραμμή. Το 1 αναφέρει ότι σχέδιο γίνεται κανονικά, και το 0 ότι δεν γίνεται, σε pixel by pixel βάση, ξεκινώντας από τα χαμηλά σημεία του μοτίβου. Το μοτίβο μπορεί να αλλάζει-απλώνεται με τη χρήση συντελεστή, ο οποίος πολλαπλασιάζει κάθε υποσειρά των συνεχόμενων 1 και 0. Έτσι, αν τρία συνεχόμενα 1s εμφανίζονται στο σχέδιο, επεκτείνονται σε έξι αν ο συντελεστής είναι 2.Ο παράγοντας βρίσκεται μεταξύ 1 και 256. Τα σημεία στίξης-κουκίδες-παύλες, πρέπει να ενεργοποιούνται με κλήση της GL_LINE_STIPPLE της glEnable () και απενεργοποιούνται με το πέρασμα του ίδιου ορίσματος στη glDisable ().

Με το προηγούμενο παράδειγμα, στο 0x3F07 μοτίβο (το οποίο μεταφράζεται σε 0011111100000111 στο δυαδικό), θα πρέπει να δημιουργηθεί μια γραμμή με 3 pixels on, μετά 5 off, 6 On, και 2 off. (Αν αυτό φαίνεται οπισθοδρομικό, να θυμάστε ότι τα χαμηλής τάξης bit χρησιμοποιούνται για πρώτη φορά.) Αν ο συντελεστής ήταν 2, το σχέδιο θα είχε επιμήκυνση: 6 pixels on, 10 off, 12 on, και 4 off. Το σχήμα 2-8 δείχνει τις γραμμές που με διαφορετικά πρότυπα και παράγοντες επαναλαμβάνονται. Εάν δεν ενεργοποιήσετε τις γραμμές στίξης, το σχέδιο προχωρεί σαν πρότυπο 0xFFFF και παράγοντας είναι το 1. (Χρησιμοποιήστε τη glDisable () με GL_LINE_STIPPLE για να απενεργοποιήσετε τη στίξη.) Σημειώστε ότι στίξη μπορεί να χρησιμοποιηθεί σε συνδυασμό με ευρείες γραμμές και να παράγει ευρεία σχεδίαση στη χάραξη γραμμών.

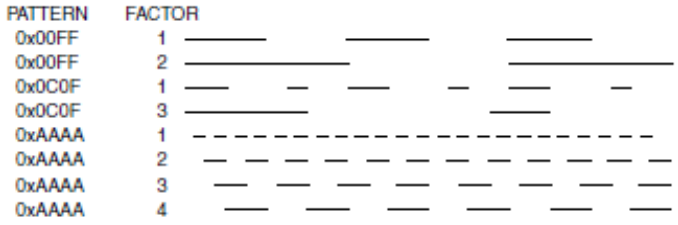


Figure 2-8 Stippled Lines

Ένας τρόπος για να περιγράψουμε την στίξη είναι ότι, καθώς η γραμμή είναι στο στάδιο της επεξεργασίας, το μοτίβο μετατίθεται κατά 1 bit κάθε φορά που ένα pixel έχει συντελεστή 1. Όταν υπάρχει μια σειρά από συνδεδεμένα ευθύγραμμα τμήματα που είναι μεταξύ ενός και μόνο glBegin () και glEnd (), το μοτίβο συνεχίζει μέχρι το τμήμα να μετατρέπεται στο επόμενο. Με αυτό τον τρόπο, μια στίξη μοτίβου συνεχίζεται σε μια σειρά από συνδεδεμένα ευθύγραμμα τμήματα. Όταν η glEnd () εκτελείται, το μοτίβο έχει μηδενιστεί, και αν οι περισσότερες γραμμές είχαν πριν τη στίξη απενεργοποιημένη ,η στίξη επανενεργοποιείται κατά την έναρξη του σχεδίου. Η σύνταξη γραμμών με τη GL_LINES, επαναφέρει το σχέδιο για κάθε ανεξάρτητη γραμμή. Το παράδειγμα 2-5 παρουσιάζει τα αποτελέσματα του σχεδίου με ένα ζευγάρι των διαφόρων μοτίβων σχεδιάζοντας με κουκίδες και παύλες. Καταδεικνύει, επίσης, τι θα συμβεί αν οι γραμμές σύρονται ως μια σειρά από επιμέρους τμήματα, αντί της συνδεδεμένης ενιαίας λωρίδας γραμμής. Τα αποτελέσματα της εκτέλεσης του προγράμματος φαίνονται στην Εικόνα 2-9.

Figure 2-9 Wide Stippled Lines

Example 2-5 Line Stipple Patterns: lines.c

```
#define drawOneLine(x1,y1,x2,y2) glBegin(GL_LINES); \
glVertex2f((x1),(y1)); glVertex2f((x2),(y2)); glEnd();
void init(void)
{
glClearColor(0.0, 0.0, 0.0, 0.0);
glShadeModel(GL_FLAT);
}
void display(void)
{
int i;
glClear(GL_COLOR_BUFFER_BIT);
/* select white for all lines */
glColor3f(1.0, 1.0, 1.0);
/* in 1st row, 3 lines, each with a different stipple */
glEnable(GL_LINE_STIPPLE);
glLineStipple(1, 0x0101); /* dotted */
drawOneLine(50.0, 125.0, 150.0, 125.0);
glLineStipple(1, 0x00FF); /* dashed */
drawOneLine(150.0, 125.0, 250.0, 125.0);
glLineStipple(1, 0x1C47); /* dash/dot/dash */
drawOneLine(250.0, 125.0, 350.0, 125.0);
/* in 2nd row, 3 wide lines, each with different stipple */
glLineWidth(5.0);
```

```
glLineStipple(1, 0x0101); /* dotted */
drawOneLine(50.0, 100.0, 150.0, 100.0);
glLineStipple(1, 0x00FF); /* dashed */
drawOneLine(150.0, 100.0, 250.0, 100.0);
glLineStipple(1, 0x1C47); /* dash/dot/dash */
drawOneLine(250.0, 100.0, 350.0, 100.0);
glLineWidth(1.0);
/* in 3rd row, 6 lines, with dash/dot/dash stipple */
/* as part of a single connected line strip */
glLineStipple(1, 0x1C47); /* dash/dot/dash */
glBegin(GL_LINE_STRIP);
for (i = 0; i < 7; i++)
glVertex2f(50.0 + ((GLfloat) i * 50.0), 75.0);
glEnd();


/* in 4th row, 6 independent lines with same stipple */
for (i = 0; i < 6; i++) {
drawOneLine(50.0 + ((GLfloat) i * 50.0), 50.0,
50.0 + ((GLfloat)(i+1) * 50.0), 50.0);
}
/* in 5th row, 1 line, with dash/dot/dash stipple */
/* and a stipple repeat factor of 5 */
glLineStipple(5, 0x1C47); /* dash/dot/dash */
drawOneLine(50.0, 25.0, 350.0, 25.0);
glDisable(GL_LINE_STIPPLE);
glFlush();
```

```
}
void reshape(int w, int h)
{
glViewport(0, 0, (GLsizei) w, (GLsizei) h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0, (GLdouble) w, 0.0, (GLdouble) h);
}
int main(int argc, char** argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(400, 150);
glutInitWindowPosition(100, 100);
glutCreateWindow(argv[0]);
init();
glutDisplayFunc(display);
glutReshapeFunc(reshape);
glutMainLoop();
return 0;
}
```

Η αναλογία της κάμερας

Η διαδικασία μετατροπής που χρησιμοποιείται για την παραγωγή της σκηνής που επιθυμείτε για την προβολή είναι ανάλογη με τη λήψη μιας φωτογραφίας με μια φωτογραφική μηχανή. Όπως φαίνεται στο Σχήμα 3-1, τα βήματα με μια φωτογραφική μηχανή (ή έναν υπολογιστή) μπορεί να είναι τα εξής:

1. Ρυθμίστε το τρίποδο και την κάμερα στο σημείο της σκηνής (για να δείτε μετασχηματισμούς).

2. Τακτοποιήστε τη σκηνή ώστε να φωτογραφηθεί η επιθυμητή θέση (μετασχηματισμός μοντελοποίηση).

3. Επιλέξτε ένα φακό της φωτογραφικής μηχανής ή ρυθμίσετε το ζουμ (μετασχηματισμό προβολής).

4. Καθορίστε πόσο μεγάλη θέλετε την τελική φωτογραφία που-για παράδειγμα, μπορεί να θέλετε να διευρυνθεί (viewport μετασχηματισμός).

Μετά από αυτά τα βήματα , η εικόνα μπορεί να ανακτηθεί και η σκηνή είναι έτοιμη για σχεδίαση. Σημειώστε ότι τα βήματα αυτά αντιστοιχούν στην σειρά με την οποία θα ορίσετε τους επιθυμητούς μετασχηματισμούς στο πρόγραμμά σας, όχι απαραίτητα με τη σειρά που οι σχετικές μαθηματικές πράξεις πραγματοποιούνται στις κορυφές ενός αντικειμένου. Για την προβολή μετασχηματισμού, πρέπει να προηγηθεί η μετατροπή μοντελοποίησης στον κώδικά σας, αλλά μπορείτε να καθορίσετε την προβολή και τους μετασχηματισμούς viewport σε οποιοδήποτε σημείο πριν από τη σχεδίαση. Η Εικόνα 3-2 δείχνει τη σειρά με την οποία οι λειτουργίες αυτές εμφανίζονται στον υπολογιστή σας.

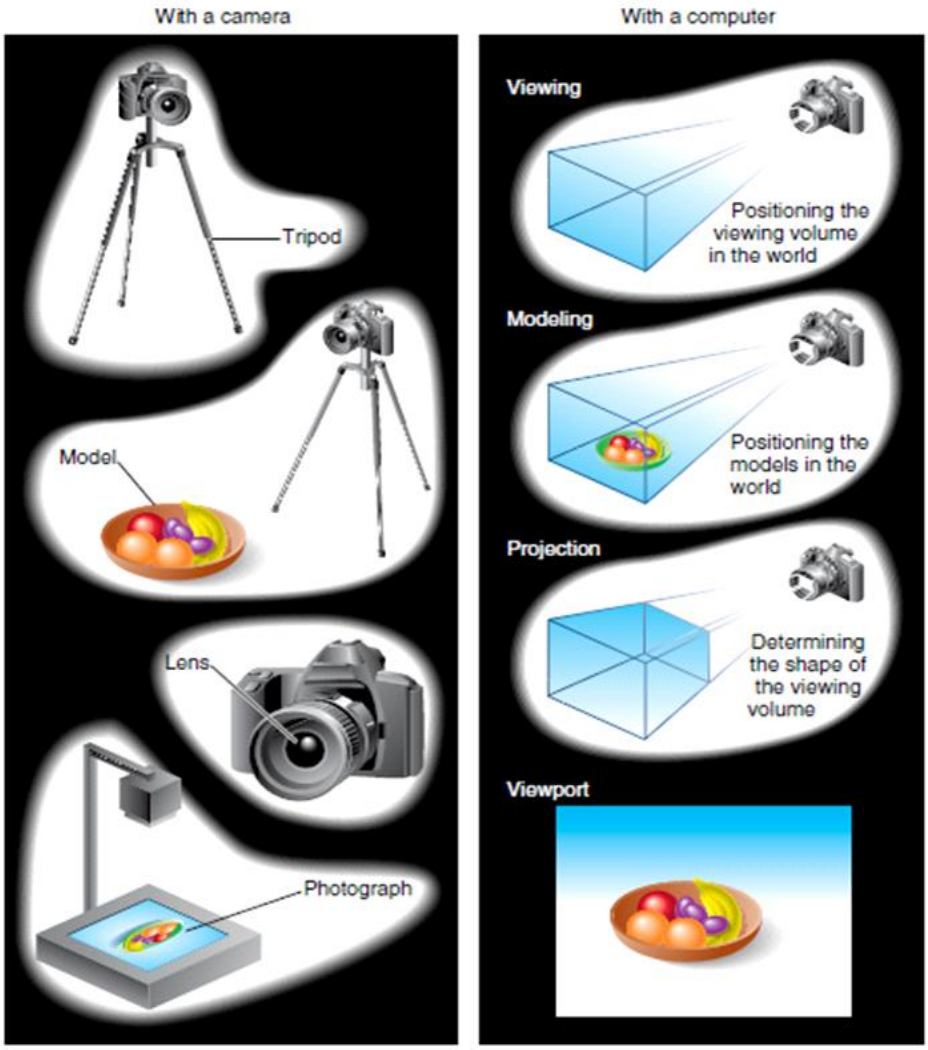


Figure 3-1 The Camera Analogy

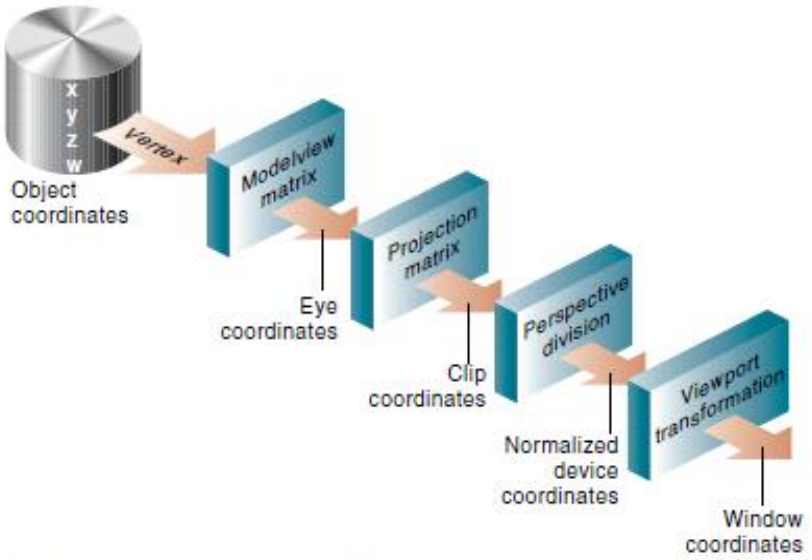


Figure 3-2 Stages of Vertex Transformation

Για να καθορίσετε την προβολή, μοντελοποίησης, και τους μετασχηματισμούς προβολής, θα πρέπει να κατασκευαστεί ένας πίνακας (μήτρα M) 4 χ 4, ο οποίος θα πολλαπλασιάζεται στη συνέχεια με τις συντεταγμένες της κάθε κορυφής v στη σκηνή, για να ολοκληρώσει το μετασχηματισμό:

v = "Mv

(Θυμηθείτε ότι οι κορυφές έχουν πάντα τέσσερις συντεταγμένες (x, y, z, w), αν και στις περισσότερες περιπτώσεις η w είναι 1, και για τις δύο διαστάσεις των δεδομένων, η z είναι 0.) Σημειώστε ότι η παρακολούθηση και οι μετασχηματισμοί μοντελοποίησης εφαρμόζονται αυτόματα στις κανονικές επιφάνειες, εκτός από τις κορυφές. (Τα κανονικά διανύσματα χρησιμοποιούνται μόνο σε συντεταγμένες των ματιών.) Αυτό εξασφαλίζει ότι η σχέση του κανονικού διανύσματος στην κορυφή είναι σωστά κρατημένη. Οι μετατροπές προβολής και μοντελοποίησης που έχετε καθορίσει ,συνδυάζονται για να σχηματίσουν τη μοντελοποιημένη μήτρα, η οποία εφαρμόζεται στα εισερχόμενα αντικείμενα συντεταγμένων για το μάτι. Στη συνέχεια, αν έχετε καθορίσει πρόσθετες απορρίψεις για ορισμένα αντικείμενα της σκηνής ,στη συνέχεια αυτά γίνονται αποδεκτά. Μέτα από αυτό, για την OpenGL ισχύει ο πίνακας προβολής με απόδοση συντεταγμένων. Αυτός ο μετασχηματισμός ορίζει μια προβολή όγκου. Αντικείμενα έξω από αυτό τον όγκο απορρίπτονται ώστε να μην είναι στην τελική σκηνή. Μετά από αυτό το σημείο, η διαίρεση προοπτικής γίνεται με τη διαίρεση τιμών συντεταγμένων w, για να παραχθούν οι κανονικές συντεταγμένες. Τέλος, οι μετατραπείς συντεταγμένων μετατρέπουν σε συντεταγμένες παραθύρου, εφαρμόζοντας το μετασχηματισμό viewport. Μπορείτε να χειριστείτε τις διαστάσεις του viewport για να κάνετε την τελική εικόνα να είναι διευρυμένη, συρρικνωμένη, ή να τεντώνεται. Ας υποθέσουμε ότι οι x-και y-συντεταγμένες είναι επαρκείς για να καθορίσουν ποια pixels πρέπει να σχεδιάζονται στην οθόνη. Ωστόσο, όλοι οι μετασχηματισμοί πραγματοποιούνται στις z-συντεταγμένες, καθώς με αυτό τον

τρόπο, κατά το τέλος αυτής της διαδικασίας μετασχηματισμού, οι z-τιμές , αντικατοπτρίζουν σωστά το βάθος ενός συγκεκριμένου κόμβου (που μετράται σε απόσταση από την οθόνη). Μια χρήση για αυτής της τιμής βάθους είναι να εξαλειφθούν τα περιττά σχέδια. Για παράδειγμα, ας υποθέσουμε οτι δύο κορυφές έχουν τις ίδιες x-και y-τιμές αλλά διαφορετικές z-τιμές. OpenGL μπορεί να χρησιμοποιήσει αυτές τις πληροφορίες για να καθορίσει ποιες επιφάνειες θα επικαλύπτονται από άλλες επιφάνειες και μπορεί στη συνέχεια να αποφύγει να σχεδιάσει τις κρυφές επιφάνειες. Θα πρέπει κανείς να γνωρίζει λίγα πράγματα για μήτρες από τη γραφική άλγεβρα των μαθηματικών ,για να μπορέσει να κατανοήσει στο μέγιστο τα προαναφερόμενα.

Ένα απλό παράδειγμα: Σχεδίαση Κύβου

Το παράδειγμα 3-1 σχεδιάζει ένα κύβο που σχεδιάζεται από μετασχηματισμό μοντέλων (βλ. Εικόνα 3-3), την προβολή μετασχηματισμού, την gluLookAt (), τις θέσεις και τους στόχους της κάμερας προς την κατεύθυνση όπου ο κύβος έχει συνταχθεί, τη μετατροπή προβολής και τους μετασχηματισμούς viewport.

Figure 3-3 Transformed Cube

Example 3-1 Transformed Cube: cube.c

```
void init(void)

{

glClearColor(0.0, 0.0, 0.0, 0.0);

glShadeModel(GL_FLAT);
```

```
}
void display(void)
{
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0, 1.0, 1.0);
glLoadIdentity(); /* clear the matrix */
/* viewing transformation */
gluLookAt(0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
glScalef(1.0, 2.0, 1.0); /* modeling transformation */
glutWireCube(1.0);
glFlush();
}
void reshape(int w, int h)
{
glViewport(0, 0, (GLsizei) w, (GLsizei) h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(-1.0, 1.0, -1.0, 1.0, 1.5, 20.0);
glMatrixMode(GL_MODELVIEW);
}
int main(int argc, char** argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(500, 500);
glutInitWindowPosition(100, 100);
glutCreateWindow(argv[0]);
```

```
init();
glutDisplayFunc(display);
glutReshapeFunc(reshape);
glutMainLoop();
return 0;
}
```

Το Χρώμα των Ηλεκτρονικών Υπολογιστών

Σε μια έγχρωμη οθόνη του υπολογιστή, το υλικό, κάνει κάθε pixel στην οθόνη να εκπέμπει διαφορετικές ποσότητες κόκκινου, πράσινου και μπλε φωτός. Αυτές ονομάζονται R, G, B τιμές. Συχνά συσκευάζονται μαζί (μερικές φορές με μια τέταρτη τιμή , που ονομάζεται άλφα, ή A), και ονομάζονται RGB (ή RGBA) τιμές. Οι πληροφορίες χρώματος σε κάθε pixel μπορεί να αποθηκευτούν είτε σε λειτουργία RGBA, στην οποία οι R, G, B τιμές ,διατηρούνται για κάθε pixel, ή σε colorindex λειτουργία, στην οποία σε μονό αριθμό (που ονομάζεται δείκτης χρώματος) αποθηκεύεται κάθε pixel. Κάθε δείκτης χρώματος δείχνει μια καταχώρηση σε έναν πίνακα που καθορίζει ένα συγκεκριμένο σύνολο R, G, B τιμών. Ένας τέτοιος πίνακας ονομάζεται χάρτης χρωμάτων. Σε λειτουργία Color-Index,είναι δυνατή η μεταβολή των τιμών του χάρτη χρωμάτων. Δεδομένου ότι για τους χάρτες που ελέγχονται από το σύστημα παραθύρων, δεν υπάρχουν OpenGL εντολές. Υπάρχει μεγάλη διακύμανση μεταξύ των διαφόρων υλικών. Σε οποιοδήποτε σύστημα γραφικών, κάθε pixel καταλαμβάνει την ίδια ποσότητα μνήμης για την αποθήκευση του χρώματος, και η μνήμη για όλα τα pixels ,ονομάζεται buffer χρωμάτων. Το μέγεθος του buffer , συνήθως μετριέται σε bits, οπότε ένας 8-bit buffer θα μπορούσε να αποθηκεύσει 8 bits δεδομένων (256 πιθανά διαφορετικά χρώματα) για κάθε pixel. Το μέγεθος των buffers, κυμαίνεται από μηχάνημα σε μηχάνημα. Οι R, G, B τιμές, μπορεί να κυμαίνονται από 0,0 (άδεια) έως 1,0 (πλήρης ένταση). Για παράδειγμα, R = 0,0, G = 0.0, και B = 1,0 αντιπροσωπεύουν το φωτεινότερο δυνατό μπλε. Αν R, G, και B είναι όλα 0.0, το pixel είναι μαύρο. Αν όλες είναι 1.0, το pixel τίθεται στο πιο φωτεινό λευκό που μπορεί να εμφανίζεται στην οθόνη. Συνδυάζοντας το πράσινο και το μπλε, δημιουργούνται αποχρώσεις του κυανού. Μπλε και κόκκινο συνδυάζονται για ματζέντα. Κόκκινο και πράσινο δημιουργούν κίτρινο. Παρακάτω, δημιουργούμε το χρώμα που θέλουμε από τα R, G, και B στοιχεία, για να πάρουμε έναν κύβο. Οι άξονες αυτού του κύβου, αντιπροσωπεύουν αποχρώσεις του κόκκινου, μπλε και πράσινου. Μια μαύρη και άσπρη έκδοση του κύβου, φαίνεται στο Σχήμα 4-1.

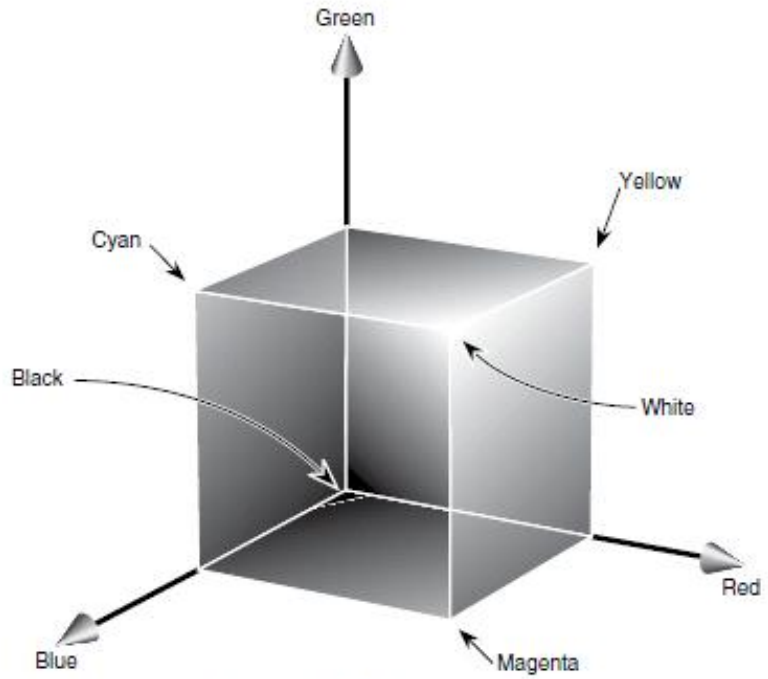


Figure 4-1 The Color Cube in Black and White

Οι εντολές που χρησιμοποιούνται για να καθοριστεί ένα χρώμα για ένα αντικείμενο (στην περίπτωση αυτή, ένα σημείο) μπορεί να είναι τόσο απλές όσο αυτές :

glColor3f (1.0, 0.0, 0.0)? / * το τρέχον χρώμα RGB είναι το κόκκινο: * /

/ * Πλήρες κόκκινο, χωρίς πράσινο, χωρίς μπλε. * /

glBegin (GL_POINTS);

glVertex3fv (point_array);

glEnd ();

Σε ορισμένες περιπτώσεις (για παράδειγμα, αν ο φωτισμός ή η σχεδίαση έχει εκτελεστεί), η ανάθεση χρωμάτων, θα μπορούσε να γίνει μέσω άλλων λειτουργιών πριν φθάσουν στον framebuffer ως τιμή που αντιπροσωπεύει ένα χρώμα για ένα pixel. Στην πραγματικότητα, το χρώμα ενός pixel προσδιορίζεται από μια μακρά ακολουθία πράξεων. Πριν την εκτέλεση ενός προγράμματος, το χρώμα-οθόνης αναμονής, έχει ρυθμιστεί με RGBA λειτουργία ή με color-index λειτουργία. Μόλις η color-index λειτουργία αρχικοποιηθεί, τότε δεν μπορεί να αλλάξει. Δεδομένου ότι το πρόγραμμα εκτελείται, ένα χρώμα (είτε ένας δείκτης χρώματος ή μια τιμή RGBA) καθορίζεται για κάθε μία κορυφή σε κάθε γεωμετρικό πρότυπο. Αυτό το χρώμα είναι είτε ένα χρώμα που έχει ρητά οριστεί για μια κορυφή ή, αν ο φωτισμός είναι ενεργοποιημένος, καθορίζεται από την αλληλεπίδραση του μετασχηματισμού πίνακα με τις φυσιολογικές επιφάνειες και άλλες ιδιότητες των στοιχείων. Μετά από τους σχετικούς

υπολογισμούς φωτισμού που έχουν πραγματοποιηθεί, το επιλεγμένο μοντέλο σκίασης εφαρμόζεται. Όπως προαναφέρθηκε, μπορείτε να επιλέξετε επίπεδη ή ομαλή σκίαση, καθέ μια από τις οποίες έχει διαφορετικές επιπτώσεις στο χρώμα ενός pixel. Στη συνέχεια, τα πρότυπα περνούν στη ραστεροποίηση ή μετατρέπονται σε μια δισδιάστατη εικόνα. Η ραστεροποίηση περιλαμβάνει τον καθορισμό του τετραγώνου του ακεραίου πλέγματος στο παράθυρο συντεταγμένων που καταλαμβάνεται από τα πρότυπα, και στη συνέχεια ανατίθενται χρώματα και άλλες τιμές σε κάθε τέτοιο τετράγωνο. Ένα τετράγωνο πλέγμα μαζί με τις συνδεδεμένες του τιμές χρώματος, z (βάθος) και υφή ,ονομάζεται Pixel. Μόλις ένα pixel έχει κατασκευαστεί, το texturing, η ομίχλη, και το antialiasing εφαρμόζονται. Μετά από αυτό, οποιαδήποτε συγκεκριμένη ανάμειξη άλφα, και bitwise λογικές πράξεις που πραγματοποιούνται με το τμήμα και τα pixel του, δεν είναι εφικτά, γιατί ήδη έχει αποθηκευτεί στον framebuffer. Τέλος, η τιμή του χρώματος του τμήματος (είτε το χρώμα δείκτη ή RGBA) είναι καταχωρημένο στα pixel και εμφανίζονται στο παράθυρο με λειτουργία έγχρωμης οθόνης παραθύρου.

RGBA vs Color-Index Λειτουργίας

Σε κάθε color-index ή RGBA λειτουργία, μια ορισμένη ποσότητα των δεδομένων , είναι αποθηκευμένη σε κάθε pixel. Η ποσότητα αυτή καθορίζεται από τον αριθμό των bitplanes στον framebuffer. Ένα bitplane περιέχει 1 bit δεδομένων για κάθε pixel. Εάν υπάρχουν 8 bitplanes χρώματων, υπάρχουν 8 bit χρώματος ανά pixel, και ως εκ τούτου, 28 = 256 διαφορετικές τιμές ή χρώματα που μπορούν να αποθηκευτούν σε ένα pixel. Τα Bitplanes συχνά είναι ομοιόμορφα κατανεμημένα για R, G,B στοιχεία (δηλαδή, ένα 24-bitplane σύστημα αφιερώνει 8 bits το κάθε ένα σε κόκκινο, πράσινο και μπλε), Αλλά αυτό δεν είναι πάντα αλήθεια. Για να μάθετε τον αριθμό των διαθέσιμων bitplanes στο σύστημά σας για το κόκκινο, πράσινο, μπλε, άλφα, ή το χρώμα δείκτη τιμών, γίνεται χρήση της glGetIntegerv () και των GL_RED_BITS, GL_GREEN_BITS, GL_BLUE_BITS, GL_ALPHA_BITS, και GL_INDEX_BITS.

Σημείωση: Η ένταση χρώματος στις περισσότερες οθόνες των υπολογιστών δεν θεωρείται ως γραμμική από το ανθρώπινο μάτι. Η χαρτογράφηση που χρησιμοποιείται γενικά, είναι εκθετική , με τον εκθέτη να αναφέρεται ως γάμμα (εξ ου και ο όρος διόρθωση γάμμα). Χρησιμοποιώντας το ίδιο γάμμα για το κόκκινο, πράσινο και μπλε στοιχείο ,δίνει αρκετά καλά αποτελέσματα, αλλά τρεις διαφορετικές τιμές γάμμα θα μπορούσαν να δώσουν ελαφρώς καλύτερα αποτελέσματα.

Ο πραγματικός κόσμος και ο φωτισμός στην OpenGl

Αν κοιτάξει κανείς σε μια φυσική επιφάνεια, η αντίληψη των ματιών στο χρώμα εξαρτάται από την κατανομή των φωτονίων και κάποιων άλλων λειτουργιών της ανθρώπινης όρασης .Τα φωτόνια προέρχονται από μια πηγή φωτός ή συνδυασμό των πηγών, μερικά απορροφώνται και μερικά αντανακλώνται από την επιφάνεια. Επιπλέον, διαφορετικές επιφάνειες μπορεί να έχουν πολύ διαφορετικές ιδιότητες-μερικά είναι λαμπερά και κατά προτίμηση αντανακλούν το φως σε συγκεκριμένες κατευθύνσεις. Οι περισσότερες επιφάνειες είναι κάπου στο ενδιάμεσο. Η OpenGL προσεγγίζει το φως και το φωτισμό σαν το φως να μπορεί να σπάσει σε κόκκινα, πράσινα, και μπλε στοιχεία. Έτσι, το χρώμα της φωτεινής πηγής χαρακτηρίζεται από την ποσότητα του κόκκινου, πράσινου και μπλε φωτός που εκπέμπεται, και το υλικό της επιφάνειας χαρακτηρίζεται από τα ποσοστά των εισερχόμενων κόκκινων, πράσινων, και μπλε στοιχείων τα οποία αντανακλώνται σε διάφορες κατευθύνσεις. Η OpenGL-εξισώσεις φωτισμού είναι μόνο προσεγγίσεις, αλλά και αυτές λειτουργούν αρκετά καλά και μπορούν να υπολογιστούν σχετικά γρήγορα. Αν θέλετε ένα πιο ακριβές (ή απλά διαφορετικό) μοντέλο φωτισμού, θα πρέπει να κάνετε τους υπολογισμούς σας σε λογισμικό. Το εν λόγω λογισμικό μπορεί να είναι πάρα πολύ περίπλοκο. Στο μοντέλο OpenGL φωτισμού, το φως σε μια σκηνή προέρχεται από διάφορες πηγές φωτός, οι οποίες μπορούν να ενεργοποιηθούν και να απενεργοποιηθούν. Μερικό από το φως προέρχεται από μια συγκεκριμένη κατεύθυνση ή θέση, και λίγο φως είναι συνήθως διάσπαρτο στη σκηνή. Για παράδειγμα, όταν ενεργοποιείτε μια λάμπα σε ένα δωμάτιο, το περισσότερο φως θα προέρχεται από τη λάμπα, αλλά μερικό από το φως των αποτελεσμάτων έχει ένα, δύο, τρία, ή περισσότερα τοιχώματα. Αυτή η διαφορά στο φως (που ονομάζεται φως του περιβάλλοντος) διασφαλίζει ότι είναι τόσο διάσπαρτο ,ωστε να μην υπάρχει κανένας τρόπος να φανεί η αρχική του κατεύθυνση, αλλά εξαφανίζεται αν μια συγκεκριμένη πηγή φωτός είναι απενεργοποιημένη. Τέλος, ίσως να υπάρξει ένα γενικό φως στη σκηνή που δεν προέρχεται από συγκεκριμένη πηγή. Στο μοντέλο OpenGL, οι πηγές φωτός έχουν επιπτώσεις μόνο όταν υπάρχουν επιφάνειες που απορροφούν και αντανακλούν το φως. Κάθε επιφάνεια θεωρείται ότι αποτελείται απο ένα υλικό με διάφορες ιδιότητες. Ένα υλικό που μπορεί να εκπέμψει το δικό του φως (όπως οι προβολείς σε ένα αυτοκίνητο), μπορεί να σκεδάσει κάποια εισερχόμενα φώτα προς όλες τις κατευθύνσεις, και μπορεί να αντανακλά κάποια μερίδα του εισερχόμενου φωτός σε μια επιθυμητή κατεύθυνση (όπως ένας καθρέφτης ή άλλη γυαλιστερή επιφάνεια).Ο OpenGL φωτισμός, θεωρεί πως ο φωτισμός πρέπει να χωριστεί σε τέσσερα ανεξάρτητα στοιχεία: αέρα, διάχυτο, κατοπτρική, και emissive. Και τα τέσσερα στοιχεία υπολογίζονται ανεξάρτητα και στη συνέχεια αθροίζονται.

Ένα απλό παράδειγμα: Μια αναμμένη Σφαίρα

Αυτά είναι τα βήματα που απαιτούνται για να προσθέσετε φωτισμό στη σκηνή:

1. Ορίστε κανονικά διανύσματα για κάθε κορυφή του κάθε αντικειμένου. Έτσι καθορίζεται ο προσανατολισμός του αντικειμένου σε σχέση με τις πηγές φωτός.

2. Δημιουργία, επιλογή, και θέση μίας ή περισσότερων πηγών φωτός.

3. Δημιουργήστε και επιλέξτε ένα μοντέλο φωτισμού, το οποίο καθορίζει το επίπεδο του πραγματικού φωτός περιβάλλοντος και την αποτελεσματική του άποψη (για τους σκοπούς των υπολογισμών φωτισμού).

4. Ορισμός ιδιοτήτων των υλικών με τα αντικείμενα στη σκηνή.

Το παράδειγμα 5-1 πετυχαίνει αυτά τα αποτελέσματα. Εμφανίζει μια σφαίρα που φωτίζεται από μία και μόνο πηγή φωτός, όπως φαίνεται και στο σχήμα .

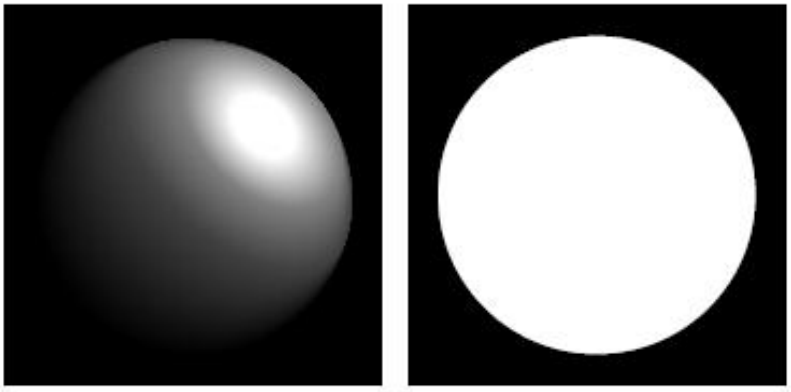


Figure 5-1 A Lit and an Unlit Sphere

```
void init(void)
{
GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat mat_shininess[] = { 50.0 };
GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };
GLfloat white_light[] = { 1.0, 1.0, 1.0, 1.0 };
```

```
GLfloat lmodel_ambient[] = { 0.1, 0.1, 0.1, 1.0 };
glClearColor(0.0, 0.0, 0.0, 0.0);
glShadeModel(GL_SMOOTH);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);

glLightfv(GL_LIGHT0, GL_POSITION, light_position);
glLightfv(GL_LIGHT0, GL_DIFFUSE, white_light);
glLightfv(GL_LIGHT0, GL_SPECULAR, white_light);
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lmodel_ambient);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glEnable(GL_DEPTH_TEST);
}
void display(void)
{
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glutSolidSphere(1.0, 20, 16);
glFlush();
}
void reshape(int w, int h)
{
glViewport(0, 0, (GLsizei) w, (GLsizei) h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
if (w <= h)
glOrtho(-1.5, 1.5, -1.5*(GLfloat)h/(GLfloat)w,
1.5*(GLfloat)h/(GLfloat)w, -10.0, 10.0);
else
glOrtho(-1.5*(GLfloat)w/(GLfloat)h,
1.5*(GLfloat)w/(GLfloat)h, -1.5, 1.5, -10.0, 10.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
}
int main(int argc, char** argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
glutInitWindowSize(500, 500);
glutInitWindowPosition(100, 100);
glutCreateWindow(argv[0]);
init();
glutDisplayFunc(display);
glutReshapeFunc(reshape);
glutMainLoop();
return 0;
}
```

Ένα πράγμα που πρέπει να θυμάστε σχετικά με το Παράδειγμα 5-1 είναι ότι χρησιμοποιεί RGBA λειτουργία χρώματος και όχι color-index λειτουργία. Ο υπολογισμός φωτισμού OpenGL είναι διαφορετικός για τους δύο τρόπους, και στην πραγματικότητα, οι δυνατότητες φωτισμού είναι πιο περιορισμένες σε color-index λειτουργία. Έτσι, η RGBA είναι ο προτιμώμενος τρόπος φωτισμού.

Ανάμειξη

Έχουν ήδη αναφερθεί οι τιμές άλφα (alpha = A στην RGBA), αλλά έχουν αγνοηθεί μέχρι τώρα. Οι τιμές που καθορίζονται ως Άλφα με την glColor * (), όταν χρησιμοποιεί την glClearColor () για να καθορίσει ένα χρώμα εκκαθάρισης και κατά τον καθορισμό ορισμένων παραμέτρων φωτισμού ,όπως η ένταση της πηγής φωτός. Όπως αναφέρθηκε, τα pixels σε μια οθόνη εκπέμπουν κόκκινο, πράσινο και μπλε φως, το οποίο ελέγχεται από τις τιμές red, green, και blue. Και πώς μια τιμή alpha επηρεάζει ό, τι έχει σχεδιαστεί σε ένα παράθυρο στην οθόνη; Όταν η ανάμειξη είναι ενεργοποιημένη, η τιμή άλφα χρησιμοποιείται συχνά για να συνδυάσει την τιμή χρώματος των τμημάτων που υποβάλλονται σε επεξεργασία με αυτή των pixel που έχουν ήδη αποθηκευτεί στον framebuffer. Η ανάμειξη συμβαίνει μετά τη ραστεροποίηση και τη μετατροπή σε τεμάχια, αλλά λίγο πριν τα τελικά pixels καταχωρηθούν στον framebuffer. Η Alpha τιμές μπορούν επίσης να χρησιμοποιηθούν για τη δοκιμή άλφα, για το αν μπορεί να γίνει αποδεκτό ή να απορριφθεί ένα τμήμα με βάση την τιμή άλφα. Χωρίς ανάμειξη, κάθε νέο τμήμα αντικαθιστά όλες τις υπάρχουσες τιμές χρώματος στον framebuffer, όπως κι αν το θραύσμα είναι αδιαφανές. Με την ανάμειξη, μπορείτε να ελέγξετε τον τρόπο (και την ποσότητα) της υπάρχουσας τιμής χρώματος σε συνδυασμό με την τιμή του νέου τμήματός της. Έτσι, μπορείτε να χρησιμοποιήσετε την "Alpha Ανάμειξη" για να δημιουργήσετε ένα διαφανές κομμάτι που σας επιτρέπει μερικές από τις παλαιότερες αποθηκευμένες τιμές χρώματος. Η ανάμειξη χρωμάτων βρίσκεται στην καρδιά των τεχνικών της ψηφιακής έκθεσης και της ζωγραφικής.

Σημείωση: Οι τιμές Άλφα δεν καθορίζονται σε λειτουργία color-index,όταν οι λειτουργίες ανάμειξης δεν εκτελούνται σε λειτουργία color-index. Ο πιο φυσικός τρόπος για να γίνει ανάμειξη λειτουργιών είναι να θεωρηθεί ότι τα RGB συστατικά ενός τμήματος , αντιπροσωπεύουν το χρώμα του, και το στοιχείο άλφα ότι αντιπροσωπεύει αδιαφάνεια. Διαφανείς ή ημιδιαφανείς επιφάνειες έχουν χαμηλότερη αδιαφάνεια από κανονικές αδιαφανείς επιφάνειες και κατά συνέπεια, χαμηλότερες τιμές άλφα. Για παράδειγμα, εάν προβληθεί ένα αντικείμενο μέσα από πράσινο γυαλί, το χρώμα που βλέπετε είναι εν μέρει πράσινο από το γυαλί και εν μέρει από το χρώμα του αντικειμένου. Το ποσοστό ποικίλλει ανάλογα με τις ιδιότητες μετάδοσης του γυαλιού: Εάν το ποτήρι μεταδίδει 80 τοις εκατό του φωτός που χτυπά (δηλαδή, έχει μια αδιαφάνεια του 20 τοις εκατό), το χρώμα που βλέπετε είναι ένας συνδυασμός από 20% έγχρωμες γυάλινες επιφάνειες και 80 τοις εκατό το χρώμα του αντικειμένου πίσω από αυτό. Μπορείτε να φανταστείτε εύκολα καταστάσεις με πολλαπλές διαφώτιστες επιφάνειες. Αν κοιτάξετε ένα αυτοκίνητο, για παράδειγμα, στο

εσωτερικό του έχει ένα κομμάτι γυαλί ανάμεσα σε αυτό και στην άποψή σας. Ορισμένα αντικείμενα πίσω από το αυτοκίνητο είναι ορατά μέσα από δύο κομμάτια γυαλιού.

Antialising – Εξομάλυνση

Μπορεί να έχετε παρατηρήσει σε ορισμένες από τις εικόνες OpenGL ότι οι γραμμές, ειδικά οι οριζόντιες αλλά και οι κάθετες, πως φαίνονται ακανόνιστα. Αυτά τα "Jaggies" εμφανίζονται επειδή η ιδανική γραμμή έχει προσεγγιστεί από μια σειρά από pixels που πρέπει να βρίσκονται στο πλέγμα pixel. Η jaggedness ονομάζεται aliasing, και αυτή τη στιγμή περιγράφουμε τις antialiasing τεχνικές για τη μείωσή της. Εικόνα 6-2 δείχνει δύο διασταυρούμενες γραμμές, που εξομαλύνονται. Οι εικόνες έχουν μεγεθυνθεί για να δείξουν καλύτερα την επίδραση.

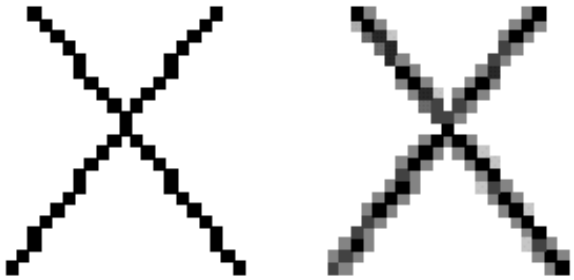


Figure 6-2 Aliased and Antialiased Lines

Το σχήμα 6-3 δείχνει πώς μια διαγώνια γραμμή 1 pixel, καλύπτει περισσότερα pixel από άλλες. Στην πραγματικότητα, κατά την εκτέλεση του antialiasing, η OpenGL υπολογίζει μια τιμή κάλυψης για κάθε τμήμα με βάση το κλάσμα του τετραγώνου του pixel στην οθόνη, ώστε να καλύπτεται. Το σχήμα 6-3 δείχνει τις τιμές αυτές για την κάλυψη της γραμμής. Σε RGBA λειτουργία, η OpenGL πολλαπλασιάζει την τιμή άλφα του τμήματος για την κάλυψή του. Μπορείτε να χρησιμοποιήσετε στη συνέχεια την προκύπτουσα τιμή άλφα για να συνδυαστεί το κομμάτι με τα αντίστοιχα pixel που είναι ήδη στον framebuffer. Σε color-index λειτουργία, η OpenGL θέτει τα λιγότερο σημαντικά 4 bits του δείκτη με βάση το χρώμα ,σχετικά με την κάλυψη του τμήματός του (0000 για καμία κάλυψη και 1111 για την πλήρη κάλυψη). Είναι στο χέρι μας να φορτώσουμε το χάρτη χρώματός και να τον εφαρμόσουμε κατάλληλα για να επωφεληθεί από αυτές τις πληροφορίες κάλυψης. Οι λεπτομέρειες του υπολογισμού των τιμών-κάλυψης είναι σύνθετες, είναι δύσκολο να προσδιοριστούν σε γενικές γραμμές, και στην πραγματικότητα μπορεί να διαφέρουν

ελαφρώς ανάλογα με την εφαρμογή της OpenGL. Μπορείτε να χρησιμοποιήσετε τη glHint () εντολή για την άσκηση ελέγχου πάνω από το trade-off μεταξύ της ποιότητας εικόνας και της ταχύτητας, αλλά όχι ότι όλες οι υλοποιήσεις θα πάρουν την υπόδειξη.

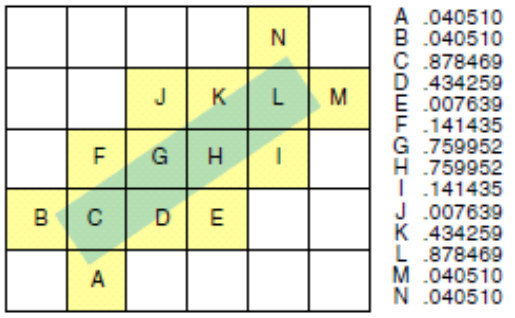


Figure 6-3 Determining Coverage Values

void glHint(GLenum target, GLenum hint);

Ελέγχει ορισμένες πτυχές της συμπεριφοράς OpenGL. Η παράμετρος-target , δείχνει ποια συμπεριφορά θα πρέπει να ελέγχεται και εμφανίζονται πιθανές τιμές της στον Πίνακα 6-3. Η παράμετρος hint, μπορεί να είναι η GL_FASTEST για να δείξει ότι μόνο η πιο αποτελεσματική λύση θα πρέπει να επιλεγεί, η GL_NICEST για να γίνει η υψηλής ποιότητας επιλογή, ή GL_DONT_CARE ώστε να μην ορίζεται καμία προτίμηση. Σε OpenGL εφαρμογές μπορούν να αγνοηθούν εντελώς. Η GL_PERSPECTIVE_CORRECTION_HINT παράμετρος αναφέρεται στις τιμές των χρωμάτων και συντονίζει την υφή παρεμβολής σε ένα πρότυπο: είτε γραμμικά στο χώρο της οθόνης (ένας σχετικά απλός υπολογισμός) ή σε μια perspective-correct λογική (η οποία απαιτεί περισσότερους υπολογισμούς).

Συχνά, τα συστήματα εκτελούν γραμμική παρεμβολή χρώματος, επειδή τα αποτελέσματα, ενώ δεν είναι τεχνικά σωστά, είναι οπτικά αποδεκτά. Ωστόσο, στις περισσότερες περιπτώσεις, οι υφές απαιτούν η perspective-correct παρεμβολή να είναι οπτικά αποδεκτή. Έτσι, μια εφαρμογή OpenGL μπορεί να επιλέξει να χρησιμοποιήσει αυτή την παράμετρο για τον έλεγχο της μεθόδου που χρησιμοποιείται για την παρεμβολή.

Parameter	Specifies
GL_POINT_SMOOTH_HINT, GL_LINE_ SMOOTH_HINT, GL_POLYGON_ SMOOTH_HINT	sampling quality of points, lines, or polygons during antialiasing operations
GL_FOG_HINT	whether fog calculations are done per pixel (GL_NICEST) or per vertex (GL_FASTEST)
GL_PERSPECTIVE_CORRECTION_HINT	quality of color and texture-coordinate interpolation
GL_GENERATE_MIPMAP_HINT	quality and performance of automatic mipmap level generation
GL_TEXTURE_COMPRESSION_HINT	quality and performance of compressing texture images
GL_FRAGMENT_SHADER_DERIVATIVE_ HINT	derivative accuracy for fragment processing for built-in GLSL shader functions dFdx, dFdy, and fwidth

Table 6-3 Values for Use with glHint()

Antialising σημείων και γραμμών

Ένας τρόπος για εξομάλυνση σε σημεία ή γραμμές είναι να ενεργοποιήσετε την antialiasing με την glEnable (), περνώντας την GL_POINT_SMOOTH ή την GL_LINE_SMOOTH, ανάλογα με την περίπτωση. Ίσως επίσης να είναι επιθυμητό να παρασχεθεί μια υπόδειξη ποιότητας με την glHint (). (Θυμηθείτε ότι μπορείτε να ρυθμίσετε το μέγεθος ενός σημείου ή το πλάτος μιας γραμμής. Μπορείτε να ζωγραφίσετε με κουκίδες και γραμμές.) Στη συνέχεια, ακολουθούν κάποιες διαδικασίες , ανάλογα με το αν είστε σε RGBA ή color-index λειτουργία. Ένας άλλος τρόπος για την εξομάλυνση σημείων ή γραμμών είναι να χρησιμοποιήσετε multisampling, με "Antialiasing Γεωμετρική πρωτύπων με Multisampling."

Antialiasing στην RGBA λειτουργία

Στη λειτουργία RGBA, χρειάζεται να ενεργοποιήσετε την ανάμειξη. Οι παράγοντες ανάμειξης που περισσότερο πιθανά πρέπει να χρησιμοποιηθούν είναι η GL_SRC_ALPHA (πηγή) και η GL_ONE_MINUS_SRC_ ALPHA (προορισμός). Εναλλακτικά, μπορείτε να χρησιμοποιήσετε την GL_ONE για να γίνουν οι γραμμές λίγο φωτεινότερες, όπου διασταυρώνονται .Όλα τα προαναφερόμενα, είναι αυτά που χρειάζονται για να

σχεδιαστούν σημεία ή γραμμές που πρέπει να εξομαλυνθούν. Η επίδραση του antialiased είναι πιο αισθητή, αν χρησιμοποιηθεί μια αρκετά υψηλή τιμή άλφα. Ωστόσο, στις περισσότερες περιπτώσεις, η κλήση μπορεί να αγνοηθεί χωρίς σημαντικές δυσμενείς επιπτώσεις. Το παράδειγμα 6-4 αρχικοποιεί τις απαραίτητες λειτουργίες για antialiasing και εφιστά έπειτα δύο τεμνόμενες διαγώνιες γραμμές. Όταν εκτελείτε αυτό το πρόγραμμα, πατήστε το πλήκτρο "R" για να περιστρέψετε τις γραμμές ,έτσι ώστε να μπορείτε να δείτε το αποτέλεσμα του antialiasing σε γραμμές των διαφόρων κλίσεων. Σημειώστε ότι το βάθος buffer δεν είναι ενεργοποιημένο σε αυτό το παράδειγμα.

Example 6-4 Antialiased Lines: aargb.c

```
static float rotAngle = 0.;

/* Initialize antialiasing for RGBA mode, including alpha

* blending, hint, and line width. Print out implementation-

* specific info on line width granularity and width.

*/

void init(void)

{

GLfloat values[2];

glGetFloatv(GL_LINE_WIDTH_GRANULARITY, values);

printf(“GL_LINE_WIDTH_GRANULARITY value is %3.1f\n”,

values[0]);

glGetFloatv(GL_LINE_WIDTH_RANGE, values);

printf(“GL_LINE_WIDTH_RANGE values are %3.1f %3.1f\n”,

values[0], values[1]);

glEnable(GL_LINE_SMOOTH);

glEnable(GL_BLEND);
```

```
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glHint(GL_LINE_SMOOTH_HINT, GL_DONT_CARE);
glLineWidth(1.5);
glClearColor(0.0, 0.0, 0.0, 0.0);
}
/* Draw 2 diagonal lines to form an X */
void display(void)
{
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0.0, 1.0, 0.0);
glPushMatrix();
glRotatef(-rotAngle, 0.0, 0.0, 0.1);
glBegin(GL_LINES);
glVertex2f(-0.5, 0.5);
glVertex2f(0.5, -0.5);
glEnd();
glPopMatrix();
glColor3f(0.0, 0.0, 1.0);
glPushMatrix();
glRotatef(rotAngle, 0.0, 0.0, 0.1);
glBegin(GL_LINES);
glVertex2f(0.5, 0.5);
glVertex2f(-0.5, -0.5);
glEnd();
glPopMatrix();
glFlush();
}
```

```
void reshape(int w, int h)
{
glViewport(0, 0, (GLint) w, (GLint) h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
if (w <= h)
gluOrtho2D(-1.0, 1.0,
-1.0*(GLfloat)h/(GLfloat)w, 1.0*(GLfloat)h/(GLfloat)w);
else
gluOrtho2D(-1.0*(GLfloat)w/(GLfloat)h,
1.0*(GLfloat)w/(GLfloat)h, -1.0, 1.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
}
void keyboard(unsigned char key, int x, int y)
{
switch (key) {
case ‘r’:
case ‘R’:
rotAngle += 20.;
if (rotAngle >= 360.) rotAngle = 0.;
glutPostRedisplay();
break;
case 27: /* Escape Key */
exit(0);
break;
default:
```

```
break;


}
}
int main(int argc, char** argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(200, 200);
glutCreateWindow(argv[0]);
init();
glutReshapeFunc(reshape);
glutKeyboardFunc(keyboard);
glutDisplayFunc(display);
glutMainLoop();
return 0;
}
```

Fog - Ομίχλη

Οι εικόνες υπολογιστών μερικές φορές φαίνονται εξωπραγματικά έντονες και σαφώς καθορισμένες. Η εξομάλυνση, κάνει ένα αντικείμενο να φαίνεται πιο ρεαλιστικό από την εξομάλυνση των άκρων του. Επιπλέον, μπορείτε να κάνετε μια ολόκληρη εικόνα να φαίνεται πιο φυσική, με την προσθήκη ομίχλης, γεγονός που κάνει τα αντικείμενα να

ξεθωριάζουν στην απόσταση. Η "Ομίχλη" είναι ένας γενικός όρος που περιγράφει παρόμοιες μορφές των ατμοσφαιρικών επιπτώσεων και μπορεί να χρησιμοποιηθεί για την προσομοίωση ομίχλης, καπνού ή ρύπανσης .Η ομίχλη είναι απαραίτητη σε visualsimulation εφαρμογές, όπου η ορατότητα είναι περιορισμένη και χρειάζεται να προσεγγιστούν. Συχνά ενσωματώνονται σε οθόνες προσομοίωσης flight-simulator.

Όταν η ομίχλη είναι ενεργοποιημένη, τα αντικείμενα που είναι μακρύτερα ,αρχίζουν να ξεθωριάζουν στο χρώμα στην ομίχλη. Μπορείτε να ελέγξετε την πυκνότητα της ομίχλης, η οποία καθορίζει το ρυθμό με τον οποίο τα αντικείμενα ξεθωριάζουν καθώς μεγαλώνει η απόσταση, καθώς και το χρώμα της ομίχλης τους. Μπορεί επίσης να γίνει ρητά, καθορισμός ώστε μια ομίχλη να συντονίζει ανά κορυφή τους υπολογισμούς αποστάσεων ομίχλης, αντί να χρησιμοποιείται και να υπολογίζεται αυτόματα η τιμή του βάθους.

Η ομίχλη είναι διαθέσιμη σε RGBA και σε color-index λειτουργία, αν και η υπολογισμοί είναι κάπως διαφορετικοί στις δυο αυτές περιπτώσεις. Δεδομένου ότι η ομίχλη εφαρμόζεται μετά τη μήτρα μετασχηματισμών, ο φωτισμός, και η χαρτογράφηση εκτελούνται, επηρεάζοντας, μετατρέποντας και ενεργοποιώντας την υφή των αντικειμένων. Σημειώστε ότι σε μια μεγάλη προσομοίωση προγραμμάτων, η ομίχλη μπορεί να βελτιώσει τις επιδόσεις, αφού μπορεί να επιλεγεί να μην γίνουν ορατά αντικείμενα που θα ήταν πολύ θολωμένα εάν ήταν ορατά. Όλοι οι τύποι των γεωμετρικών προτύπων μπορεί να είναι θολωμένοι, συμπεριλαμβανομένων των σημείων και γραμμών. Η χρήση της επίδρασης ομίχλης σε σημεία και γραμμές , ονομάζεται επίσης "depth-cuing" και είναι δημοφιλής στη μοριακή μοντελοποίηση και άλλες εφαρμογές.

Χρησιμοποιώντας ομίχλη

Η χρήση ομίχλης είναι εύκολη. Μπορείτε να την ενεργοποιήσετε με το πέρασμα της GL_FOG ,στην glEnable () , και να επιλέξετε το χρώμα και την εξίσωση που ελέγχει την πυκνότητα με την * glFog ().

Αν θέλετε, μπορείτε να δώσετε μια τιμή για GL_FOG_HINT με glHint ().

Το παράδειγμα 6-7 εμφανίζει πέντε κόκκινες σφαίρες, καθεμία από αυτές σε διαφορετικές αποστάσεις από την άποψη. Πατώντας το «f» πλήκτρο, επιλέγεται μια, μεταξύ των τριών διαφορετικών εξισώσεων ομίχλης.

Example 6-7 Five Fogged Spheres in RGBA Mode: fog.c

```
static GLint fogMode;
static void init(void)
{
GLfloat position[] = { 0.5, 0.5, 3.0, 0.0 };
glEnable(GL_DEPTH_TEST);
glLightfv(GL_LIGHT0, GL_POSITION, position);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
{
GLfloat mat[3] = {0.1745, 0.01175, 0.01175};
glMaterialfv(GL_FRONT, GL_AMBIENT, mat);
mat[0] = 0.61424; mat[1] = 0.04136; mat[2] = 0.04136;
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat);
mat[0] = 0.727811; mat[1] = 0.626959; mat[2] = 0.626959;
glMaterialfv(GL_FRONT, GL_SPECULAR, mat);
glMaterialf(GL_FRONT, GL_SHININESS, 0.6*128.0);
}
glEnable(GL_FOG);
{
GLfloat fogColor[4] = {0.5, 0.5, 0.5, 1.0};
fogMode = GL_EXP;
```

```
glFogi(GL_FOG_MODE, fogMode);
glFogfv(GL_FOG_COLOR, fogColor);
glFogf(GL_FOG_DENSITY, 0.35);
glHint(GL_FOG_HINT, GL_DONT_CARE);
glFogf(GL_FOG_START, 1.0);
glFogf(GL_FOG_END, 5.0);
}
glClearColor(0.5, 0.5, 0.5, 1.0); /* fog color */
}
static void renderSphere(GLfloat x, GLfloat y, GLfloat z)
{
glPushMatrix();
glTranslatef(x, y, z);
glutSolidSphere(0.4, 16, 16);
glPopMatrix();
}
/* display() draws 5 spheres at different z positions.
*/
void display(void)
{
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
renderSphere(-2., -0.5, -1.0);
renderSphere(-1., -0.5, -2.0);
renderSphere(0., -0.5, -3.0);
renderSphere(1., -0.5, -4.0);
renderSphere(2., -0.5, -5.0);
glFlush();
```

```
}
void reshape(int w, int h)
{
glViewport(0, 0, (GLsizei) w, (GLsizei) h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
if (w <= h)
glOrtho(-2.5, 2.5, -2.5*(GLfloat)h/(GLfloat)w,
2.5*(GLfloat)h/(GLfloat)w, -10.0, 10.0);
else
glOrtho(-2.5*(GLfloat)w/(GLfloat)h,
2.5*(GLfloat)w/(GLfloat)h, -2.5, 2.5, -10.0, 10.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
}
void keyboard(unsigned char key, int x, int y)
{
switch (key) {
case ‘f’:
case ‘F’:
if (fogMode == GL_EXP) {
fogMode = GL_EXP2;
printf(“Fog mode is GL_EXP2\n”);
}
else if (fogMode == GL_EXP2) {
fogMode = GL_LINEAR;
printf(“Fog mode is GL_LINEAR\n”);
```

```
}
else if (fogMode == GL_LINEAR) {
fogMode = GL_EXP;
printf("Fog mode is GL_EXP\n");
}
glFogi(GL_FOG_MODE, fogMode);
glutPostRedisplay();
break;
case 27:
exit(0);
break;
default:
break;
}
}
int main(int argc, char** argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
glutInitWindowSize(500, 500);
glutCreateWindow(argv[0]);
init();
glutReshapeFunc(reshape);
glutKeyboardFunc(keyboard);
glutDisplayFunc(display);
glutMainLoop();
return 0;}
```

Texturing - Χαρτογράφηση

Μέχρι στιγμής, όλα τα γεωμετρικά πρότυπα έχουν σχεδιαστεί είτε με ένα συμπαγές χρώμα ή μια ομαλή σκιά χρωμάτων στις κορυφές τους. Εάν θέλετε να σχεδιάσετε ένα μεγάλο τοίχο από τούβλα , για παράδειγμα, με το texturing (χαρτογράφηση)κάθε τούβλο πρέπει να εξαχθεί ως ξεχωριστό πολύγωνο (ένας μεγάλος επίπεδος τοίχος-που είναι στην πραγματικότητα ένα και μόνο ορθογώνιο-ίσως απαιτεί χιλιάδες ξεχωριστά τούβλα).Αυτή είναι μια πάρα πολύ καλή τακτική, εάν επιθυμούμε να είμαστε ρεαλιστές.

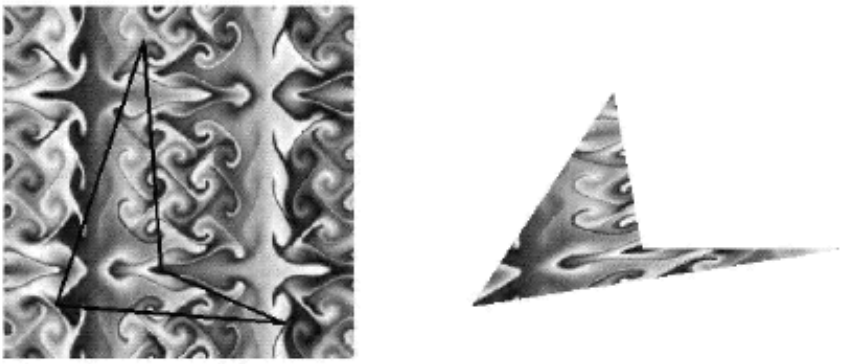


Figure 9-1 Texture-Mapping Process

Το texturing σας επιτρέπει να κολλήσετε μια εικόνα (που λαμβάνεται, ίσως, με τη λήψη μιας φωτογραφίας από ένα πραγματικό τείχος) σε ένα πολύγωνο για να δημιουργηθεί ένας ολόκληρος τοίχος ως ένα ενιαίο πολύγωνο. Το texturing εξασφαλίζει ότι όλα θα συμβούν σωστά, όπως το πολύγωνο θα μεταμορφώνεται, ενώ παρέχει κι άλλου είδους υπηρεσίες. Για παράδειγμα, όταν ο τοίχος προβάλλεται σε μια προοπτική, τα τούβλα ενδέχεται να εμφανιστούν μικρότερα όταν ο τοίχος είναι πιο μακριά από την οπτική γωνία. Άλλες χρήσεις για τη χαρτογράφηση περιλαμβάνουν παράσταση για μεγάλα πολύγωνα που αντιπροσωπεύουν το έδαφος σε προσομοίωση πτήσεων ,σε μοτίβα ταπετσαρίας και υφές που κάνουν τα πολύγωνα όπως τις φυσικές επιφάνειες, όπως μάρμαρο, ξύλο και πανί. Οι δυνατότητες είναι ατελείωτες. Αν και είναι πιο φυσικό να σκεφτούμε τις συστάσεις της εφαρμογής για τα πολύγωνα, το texturing μπορεί να εφαρμοστεί σε όλα τα πρότυπα-σημεία, γραμμές, πολύγωνα, bitmaps, και εικόνες.

Επειδή υπάρχουν τόσες πολλές δυνατότητες, το texturing είναι ένα αρκετά περίπλοκο ζήτημα και θα πρέπει να κάνετε διάφορες επιλογές προγραμματισμού κατά τη χρήση του. Για αρχή, οι περισσότεροι άνθρωποι καταλαβαίνουν ενστικτωδώς μια δισδιάστατη επιφάνεια, αλλά μια δομή μπορεί να είναι μονοδιάστατη ή ακόμα και τρισδιάστατη.

Μπορεί να γίνει χρήση του χάρτη δομών για επιφάνειες από ένα σύνολο πολυγώνων ή σε καμπύλες επιφάνειες, και μπορείτε να επαναλάβετε ένα texturing σε μία, δύο ή τρεις κατευθύνσεις (ανάλογα με το πόσες διαστάσεις περιγράφονται) για την κάλυψη από την επιφάνεια. Επιπλέον, μπορείτε να αντιστοιχίσετε αυτόματα ένα texturing πάνω σε ένα αντικείμενο με τέτοιο τρόπο ώστε η να φαίνονται περιγράμματα ή άλλες ιδιότητες του στοιχείου που προβάλλεται. Λαμπερά αντικείμενα μπορούν να έχουν texturing, ώστε να φαίνεται ότι είναι στο κέντρο του δωματίου ή άλλο περιβάλλον, γεγονός που αντικατοπτρίζει το περιβάλλον από τις επιφάνειες τους. Τέλος, ένα texturing μπορεί να εφαρμοστεί σε μια επιφάνεια με διαφορετικούς τρόπους. Μπορεί να είναι βαμμένη άμεσα (όπως μια πινακίδα τοποθετείται σε μια επιφάνεια), που χρησιμοποιείται για να ρυθμίσει το χρώμα της επιφάνειας θα έχει περαστεί με άλλο τρόπο, ή θα χρησιμοποιείται για να συνδυάσει ένα χρώμα texturing με το χρώμα της επιφάνειας.

Οι χαρτογραφημένες επιφάνειες είναι απλές ορθογώνιες συστοιχίες των δεδομένων-για παράδειγμα, τα δεδομένα χρώματος, στοιχεία φωτεινότητας, ή το χρώμα και τα δεδομένα άλφα ή μεμονωμένες τιμές σε μια επιφάνεια. Η αίτηση μπορεί να καθορίσει την επιφάνεια των αντικειμένων, με κάθε επιφάνεια αντικειμένου να αντιπροσωπεύει μια ενιαία επιφάνεια (και τα πιθανά συνακόλουθα mipmaps). Μερικές υλοποιήσεις της OpenGL μπορούν να υποστηρίξουν μια ειδική σειρά λειτουργιών της επιφάνειας αντικειμένων που έχουν καλύτερη απόδοση από ό, τι αντικείμενα έξω από το σύνολο εργασίας. Μπορείτε να χρησιμοποιήσετε την OpenGL για να δημιουργήσετε και να διαγράψετε επιφάνειες αντικειμένων και να καθορίσετε ποιες επιφάνειες δίνουν το αποτέλεσμα που επιθυμείται.

Framebuffer

Ένας σημαντικός στόχος σχεδόν κάθε προγράμματος γραφικών, είναι να σχεδιάσει εικόνες στην οθόνη. Η οθόνη αποτελείται από ένα ορθογώνιο πίνακα απο pixels,με κάθε θέση να εμφανίζει ένα μικρό τετράγωνο του χρώματος σε εκείνο το σημείο στην εικόνα. Μετά το στάδιο της ραστεροποίησης (συμπεριλαμβανομένων της χαρτογράφησης και της ομίχλης), τα δεδομένα δεν είναι ακόμη pixels-αλλα τμήματα. Κάθε τμήμα έχει τον συντονισμό των δεδομένων που αντιστοιχεί σε pixel, καθώς και το χρώμα και το βάθος των τιμών. Στη συνέχεια, κάθε τμήμα υποβάλλεται σε μια σειρά ελέγχων και υπολογισμών, ορισμένοι από τους οποίους έχουν προηγουμένως περιγραφεί, όπως π.χ η ανάμειξη.

Αν εγκριθούν από τις δοκιμές και τις εργασίες, οι τιμές και τα αντίστοιχα τμήματά τους, είναι έτοιμα να γίνουν pixels. Για να σχεδιάσετε τα pixels, θα πρέπει να γνωρίζετε το χρώμα τους, δηλαδή τις πληροφορίες που είναι αποθηκευμένες στον buffer χρώματος . Κάθε φορά που τα δεδομένα αποθηκεύονται για κάθε pixel, αυτή η καταχώρηση για όλα τα pixels καλείται από τον buffer. Διαφορετικές ζώνες ενδέχεται να περιέχουν διαφορετικές ποσότητες δεδομένων ανά pixel, αλλά μέσα σε συγκεκριμένο buffer, κάθε pixel έχει την ίδια ποσότητα δεδομένων.

Ένας buffer που αποθηκεύει ένα μόνο κομμάτι των πληροφοριών σχετικά με pixels καλείται bitplane, και ο αριθμός των bitplanes σε ένα buffer ονομάζεται συνήθως «βάθος» του. (Δεν πρέπει να συγχέεται με την τιμή του βάθους ενός pixel, το οποίο χρησιμοποιείται για λειτουργίες του buffer βάθους. Όπως φαίνεται στο Σχήμα 10-1,στα κάτω αριστερά pixel σε ένα παράθυρο OpenGL, είναι τα pixel(0, 0), που αντιστοιχούν στο παράθυρο συντεταγμένων της κάτω αριστερής γωνίας της περιφέρειας που καταλαμβάνει αυτό το pixel. Σε γενικές γραμμές, τα pixel (x, y) γεμίζουν την περιοχή που οριοθετείται από x στα αριστερά, x + 1 για το δικαίωμα, y στο κάτω μέρος, και y + 1 στην την κορυφή.

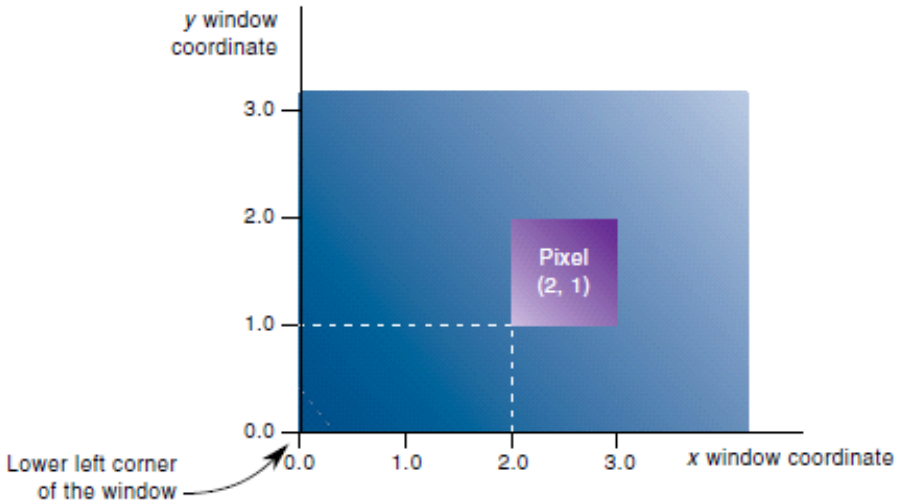


Figure 10-1 Region Occupied by a Pixel

Ως ένα παράδειγμα buffer, ας δούμε αυτόν του χρώματος, ο οποίος κατέχει τις πληροφορίες για το χρώμα που θα εμφανίζεται στην οθόνη. Υποθέστε ότι η οθόνη είναι 1280 pixels πλάτους και 1024 pixels ύψους και ότι είναι μια πλήρης 24-bit οθόνη-και όσον αφορά τα χρώματα, ότι υπάρχουν 224 (ή 16.777.216) διαφορετικά χρώματα που μπορούν να εμφανιστούν. Τα 24 bits μετατρέπονται σε 3 bytes (8 bits ανά byte)και ο buffer χρώματος σε αυτό το παράδειγμα έχει να αποθηκεύσει τουλάχιστον 3 bytes δεδομένων για καθένα από τα 1.310.720 (1280 * 1024) pixel στην οθόνη. Ένα ιδιαίτερο σύστημα υλικού

μπορεί να έχει περισσότερα ή λιγότερα pixel στην οθόνη καθώς και λιγότερα ή περισσότερα δεδομένα χρώματος ανά pixel. Οι buffer χρώματος, ωστόσο, έχουν την ίδια ποσότητα δεδομένων που αποθηκεύονται για κάθε pixel στην οθόνη. Οι buffer χρώματος είναι μόνο μία από τις πολλές κατηγορίες buffer που κατέχουν πληροφορίες σχετικά με ένα pixel. Μπορούν να αποτελούνται και από αρκετούς subbuffers. Ο framebuffer σε ένα σύστημα αποτελείται από το σύνολο αυτών των buffers. Αυτοί μπορούν να εκτελούν τα καθήκοντα για την εξάλειψη ή την απόκρυψη μιας επιφάνειας, για antialiasing ,stenciling ,κ.α .

Οι buffers και οι χρήσεις τους

Ένα σύστημα OpenGL μπορεί να χειριστεί τους ακόλουθους buffers :

•buffer χρώματος

• buffer βάθους

• Stencil buffer

• buffer συσωρευτής

Η OpenGL εφαρμογή προσδιορίζει τους buffers που οι διαθέσιμοι και πόσα bit ανά pixel κάθε buffer κρατά. Επιπλέον, μπορεί να υπάρχουν πολλαπλές εικόνες, ή τύποι παραθύρων, τα οποία έχουν διαφορετικές ζώνες διαθέσιμες. Ο πίνακας 10-1 παραθέτει τις παραμέτρους που χρησιμοποιούνται με την glGetIntegerv (), με το OpenGL ερώτημα προς το σύστημα σας, για προσωρινή αποθήκευση ανά-pixel, για μια συγκεκριμένη οπτική.

Parameter	Meaning
GL_RED_BITS, GL_GREEN_BITS, GL_BLUE_BITS, GL_ALPHA_BITS	number of bits per R, G, B, or A component in the color buffers
GL_INDEX_BITS	number of bits per index in the color buffers
GL_DEPTH_BITS	number of bits per pixel in the depth buffer
GL_STENCIL_BITS	number of bits per pixel in the stencil buffer
GL_ACCUM_RED_BITS, GL_ACCUM_GREEN_BITS, GL_ACCUM_BLUE_BITS, GL_ACCUM_ALPHA_BITS	number of bits per R, G, B, or A component in the accumulation buffer

Table 10-1 Query Parameters for Per-Pixel Buffer Storage

3. H JoGl

H OpenGl και η JAVA

Εδώ και μερικά χρόνια, ένας προγραμματιστής που ήθελε να δημιουργήσει προγράμματα γραφικών υψηλής ποιότητας που θα μπορούσαν να πωληθούν σε χρήστες διαφορετικών λειτουργικών συστημάτων ,είχε μία επιλογή-την OpenGL. Η GL είναι βιβλιοθήκη γραφικών. Η OpenGL είναι σήμα κατατεθέν του SGI. Η OpenGL φέρεται ως μια πλατφόρμα για C προγραμματισμό API. Στην πραγματικότητα όμως, πρόκειται για ένα υλικό ανεξάρτητο προδιαγραφών για μια διασύνδεση προγραμματισμού.

Η OpenGL χρησιμοποιείται για την κατασκευή γραφικών. Είναι γρήγορη. Τις περισσότερες φορές, είναι ανεξάρτητη υλικού. Φαίνεται ότι η OpenGL μπορεί να κάνει τα πάντα στον τομέα των γραφικών.

Δυστυχώς, η OpenGL είναι γραμμένη για τη γλώσσα C. Η C δεν είναι η πιο δημοφιλής γλώσσα για τον προγραμματισμό σύνθετων εφαρμογών. Ένα από τα μεγαλύτερα μειονεκτήματα της OpenGL είναι ότι δεν μπορείτε να κάνετε τίποτα χωρίς ένα παράθυρο για να θέτει τα γραφικά σας μέσα και η OpenGL δεν παρέχει ένα εύκολο μέσο για να δημιουργηθούν τα παράθυρα. Αυτό κάνει την OpenGL σκληρή για τους αρχάριους.

Ευτυχώς, η GLUT (OpenGL Utility Toolkit) εισήχθη και πρόσθεσε λειτουργίες παραθύρων , κουμπιά, και γεγονότα που δημιουργούνται από τους χρήστες πιο εύκολα. Ακόμα, η μάθηση της OpenGL σε C ή ακόμα και C + + μπορεί να είναι οδυνηρή για τους νέους προγραμματιστές ή προγραμματιστές που θέλουν να χρησιμοποιήσουν αληθινό αντικειμενοστρεφή προγραμματισμό.

Στη συνέχεια ήρθε η JOGL

Η Java είναι ίσως η πιο δημοφιλής αληθινή αντικειμενοστραφής γλώσσα προγραμματισμού. Έχουν γίνει πολλές προσπάθειες για να παντρευτεί η OpenGL με την Java, αλλά αυτό που κυρίως επετεύχθει και έγινε δημοφιλές, ήταν τα Java Bindings για OpenGL ή JOGL. Ο λόγος για αυτό είναι ότι η προσπάθεια αυτή υποστηρίζεται από την Sun Microsystems (οι δημιουργοί της Java) και τις υπηρεσίες κοινής ωφέλειας (οι δημιουργοί της OpenGL).

Σήμερα, η JOGL έχει αναπτυχθεί από την ομάδα τεχνολογίας παιχνιδιών της Sun. Αναπτύχθηκε από τον Ken Russel και τον Chris Kline. Ο Ράσελ είναι ένας υπάλληλος που εργάζεται για το HotSpot Virtual Machine με πολλά χρόνια εμπειρία στο χώρο του 3D.Ο Kline έχει αναλάβει έργα για Ανορθολογική Αγώνων και επίσης

είναι πολύ έμπειρος με τα 3D γραφικά.

Έχουν υπάρξει διάφορες προσπάθειες για την παροχή πρόσβασης σε OpenGL μέσα από ένα φιλικό Java API-μεταξύ αυτών ,ηJava 3D,η OpenGL για την τεχνολογία Java (gl4java), και η ελαφριά Game-Βιβλιοθήκη Java (LWJGL.

Η OpenGL χρησιμοποιείται για την προβολή 3D μοντέλων. Είναι ισχυρό, γρήγορο, και ίσως το πιο σπουδαίο πράγμα που μπορεί να συμβεί σε Java, από τότε που η JAVA Swing εισήχθη. Χρησιμοποιώντας OpenGL μέσω JOGL, υπάρχει δυνατότητα κατασκευής παιχνιδιών ή μοντέλων καταστάσεων που θα μπορούσαν να είναι πολύ ακριβά για να δημιουργηθούν.

Χρήση JOGL;

Αν θέλουμε να χρησιμοποιήσουμε JOGL, θα χρειαστεί να πάρουμε jogl.jar και τα συνοδευτικά αρχεία.

Το πρώτο τέχνασμα είναι να βρεθούν τα δυαδικά αρχεία για το λειτουργικό σας σύστημα και η εξαγωγή τους. Κάθε λειτουργικό σύστημα είναι διαφορετικό, αλλά υπάρχουν δύο μέρη κατα την εγκατάσταση. Υπάρχει η βιβλιοθήκη Java σε ένα αρχείο jar και ο πηγαίος κώδικας σε μια άλλη βιβλιοθήκη. Τα jogl.jar πρέπει να τοποθετούνται στο σύστημα CLASSPATH, και η δυαδική βιβλιοθήκη πρέπει να τοποθετηθεί οπουδήποτε οι βιβλιοθήκες αποθηκεύονται στο λειτουργικό σύστημά σας. Αν είστε τυχεροί, θα έχετε ένα πρόγραμμα εγκατάστασης να το κάνει για σας. Αν δεν έχετε ένα πρόγραμμα εγκατάστασης και δεν ξέρετε πού να ψάξετε για πληροφορίες σχετικά με τη διάθεση πάντα στον υπολογιστή σας, μπορείτε να ξεκινήσετε με τις συνδέσεις που παρουσιάζονται. Το πρώτο παράδειγμα κώδικα ,είναι γραμμένο ειδικά για να ελέγξει κανείς, αν έχει εγκαταστήσει τα πάντα σωστά.

JavaDocs για JOGL

Τα JavaDocs μπορούν να ληφθούν απο την ίδια τοποθεσία με τη διανομή της JOGL. Τα JavaDocs θα πρέπει να έχουν όνομα παρόμοιο με jogl-1.0-usrdoc.tar.

Εάν κάνετε περιήγηση στο net.java.games.jogl πακέτο, γρήγορα θα παρατηρήσετε ότι κάποιες από τις κατηγορίες είναι τεράστιες. Η GL είναι ένα τέλειο παράδειγμα γι' αυτό το σκοπό. Σημαντικά είναι και τα ακόλουθα :

GLDrawable
GLCanvas
GLJPanel
GLCapabilities
GLDrawableFactory

Αυτές είναι βασικές διεπαφές στον κόσμο των γραφικών. Αν θυμάστε, νωρίτερα αναφέρθηκε ότι ένα από τα μεγαλύτερα μειονεκτήματα για αρχάριους στην OpenGL ,είναι η έλλειψη ενός πρότυπου συστήματος παραθύρων. Η GLUT βοηθά σε μεγάλο βαθμό, αλλά υπάρχει και η Swing και η AWT (Abstract Window Toolkit). Είναι πολύ πιθανόν να έχετε ήδη χρησιμοποιήσει AWT και Swing, έτσι δεν πρόκειται να νιώθετε σαν να μαθαίνεται τα πάντα από το μηδέν. Μετά από μια πολύ σύντομη εισαγωγή, για να λάβει θέση ένα στοιχείο για JOGL επάνω στην οθόνη, δεν θα χρειαστεί πολύ δουλειά .

GlueGen ... σχεδόν τόσο ενδιαφέρουσα όπως η JOGL

Όπως είναι γνωστό, η OpenGL είναι γραμμένη για C προγραμματιστές. Αυτό σημαίνει ότι για Java, πρέπει να υπάρχει κάποια φυσική διασύνδεση. Αυτό σημαίνει JNI (Java Native Interface), το οποίο δεν είναι θέμα ψυχαγωγίας ή αισθητικής, αλλά επιβάλλεται να γραφτεί για να κάνει αυτή τη σύνδεση. Η OpenGL είναι αρκετά μεγάλη. Εγκαθιστώντας όλες αυτές τις συνδέσεις, παίρνει χρόνο. Για να γίνουν τα πράγματα λίγο πιο εύκολα, υπάρχουν πολλές δυνατότητες από τους προμηθευτές και η OpenGL συνεχίζει να βελτιώνεται, κάτι το οποίο σημαίνει ότι δεν υπάρχουν αλλαγές για να συμβαδίσει με αυτές. Με λίγα λόγια, ήταν αρκετά δύσκολο για τον «καθένα» να προσπαθεί να συμβαδίσει με την OpenGL και να γράψει ένα Java native interface που να τα καλύπτει όλα.

Η JOGL φρόντισε ωστε να επωφεληθεί από τα αρχεία κεφαλίδας C και να γράψει κάποιο κώδικα που θα κάνει όλη τη δουλειά JNI γι 'αυτά. Δημιούργησαν τη GlueGen. Η GlueGen αναλύει τα αρχεία κεφαλίδας C και στη συνέχεια με «μαγικό» τρόπο, δημιουργεί τους απαιτούμενους Java και JNI κώδικες για να συνδεθείτε σε αυτές τις εγγενείς βιβλιοθήκες. Αυτό σημαίνει ότι οι ανανεώσεις της OpenGL μπορούν να προστεθούν γρήγορα σε JOGL.

Hello World!

Αυτό το Hello World θα εξετάσει την εγκατάσταση μας και θα δείξει αν το σύνολο ή μέρος του, έχει εγκατασταθεί σωστά. Θυμηθείτε ότι υπάρχουν δύο μέρη στην JOGL εγκατάσταση. Υπάρχει η βιβλιοθήκη Java σε ένα αρχείο jar και ο πηγαίος κώδικας σε μια άλλη βιβλιοθήκη.

Εδώ είναι το πρόγραμμά μας:

```
import net.java.games.jogl.*;


public class HelloWorld {
   public static void main (String args[]) {
      try {
         System.loadLibrary("jogl");
         System.out.println(
            "Hello World! (The native libraries are
installed.)"
         );
        GLCapabilities caps = new GLCapabilities();
         System.out.println(
            "Hello JOGL! (The jar appears to be
available.)"
         );
      } catch (Exception e) {
         System.out.println(e);
      }
   }
}
```

Κατ 'αρχάς, αυτό το είναι ένα πρόγραμμα δοκιμών για να δείτε αν ο πηγαίος κώδικας και οι Java βιβλιοθήκες,έχουν εγκατασταθεί σωστά.Η JOGL έχει εγκατασταθεί σωστά μόνο όταν το jogl.jar και η μητρική βιβλιοθήκη, που ονομάζεται κάτι σαν libjogl.jnilib ή jogl.dll,είναι και οι δύο εγκατεστημένες. Αν η μητρική βιβλιοθήκη δεν είναι προσβάσιμη, το πρόγραμμα αυτό θα εμφανίσει μια εξαίρεση java.lang.UnsatisfiedLinkError. Αν το JAR δεν έχει εγκατασταθεί στο classpath, τότε το πρόγραμμα δεν θα κάνει compile. Ο μεταγλωττιστής javac θα αποφανθεί κάτι παρόμοιο με "net.java.games.jogl και οτι το πακέτο δεν υπάρχει."Όταν αυτή η κλάση μεταγλωττιστεί και εκτελεστεί χωρίς εξαιρέσεις,όλα είναι έτοιμα για χρήση της JOGL.

4. H PyOpenGl

PYTHON και OpenGl

Η Python είναι μια γλώσσα διερμηνέα, object-oriented, υψηλού επιπέδου, με δυναμική σημασιολογία. Οι υψηλού επιπέδου λειτουργίες που είναι κτισμένες στις δομές δεδομένων της ,σε συνδυασμό με τη δυναμική πληκτρολόγηση και δυναμική σύνδεση, την καθιστούν ιδιαίτερα ελκυστική για την ταχεία ανάπτυξη εφαρμογών, καθώς και για τη χρήση της ως scripting γλώσσα ή τρόπο διασύνδεσης για να συνδέσετε υπάρχοντα στοιχεία μαζί. Είναι απλή και είναι εύκολη στη σύνταξή, ενώ δίνει έμφαση στην αναγνωσιμότητα και κατά συνέπεια μειώνει το κόστος συντήρησης του προγράμματος. Η Python υποστηρίζει τις ενότητες και τα πακέτα, τα οποία ενθαρρύνουν τα τμηματοποιημένα προγράμματα και την επαναχρησιμοποίηση κώδικα. Ο διερμηνέας Python και η πλούσια στάνταρ βιβλιοθήκη είναι διαθέσιμα σε μορφή δυαδικού κώδικα και χωρίς χρέωση ,για όλες τις σημαντικές πλατφόρμες και διανέμονται ελεύθερα.

Συχνά, οι προγραμματιστές προτιμούν την Python, λόγω της αύξησης της παραγωγικότητας που παρέχει. Δεδομένου ότι δεν υπάρχει βήμα συλλογής και επεξεργασία-test-debug κύκλος ,είναι απίστευτα γρήγορη.

Η PYTHON χρησιμοποιείται με την OpenGl, όπως και οποιαδήποτε άλλη αντικειμενοστραφής γλώσσα προγραμματισμού (πέρα από τη γλωσσική σύνταξη και διαδικαστικά θέματα σύνδεσης βιβλιοθηκών), χρησιμοποιώντας τις ίδιες βιβλιοθήκες και λειτουργίες.

5. ΠΑΡΑΔΕΙΓΜΑΤΑ ΕΦΑΡΜΟΓΩΝ OPENGL

OpenGl με Χρήση C++

Χρήση του Visual C++ 2008 Express Edition

Αυτά που χρειαζόμαστε είναι τα εξής :

1. Τη **Visual C++ Express 2008**: Βλέπετε How to install Visual C++ Express. Το VC++ θα εγκατασταθεί στο "`C:\Program Files\Microsoft Visual Studio 9.0\VC`", με τα headers στο sub-directory "`include`" και τις βιβλιοθήκες στον "`lib`".

2. Το **Windows SDK** το οποίο περιλαμβάνει την **OpenGL** και την **GLU (OpenGL Utility)**. Η Visual C++ 2008 Express δεσμεύει το Microsoft Windows SDK, το οποίο θα φορτωθεί στο "`C:\Program Files\Microsoft SDKs\Windows\v6.0A`". (Σε διαφορετική περίπτωση, θα χρειαστεί download και εγκατάσταση του Windows SDK ξεχωριστά).

   Τα παρακάτω χρησιμοποιούνται απο το Windows SDK:
   - `gl.h`, `glu.h` : Header για την OpenGL και τη GLU,στο φάκελο "`C:\Program Files\Microsoft SDKs\Windows\v6.0A\include\gl`".Αυτός ο φάκελος θα περιέχεται στο περιβάλλον `INCLUDE` ή στην επιλογή `/I<dir>` της γραμμής εντολών.
   - `opengl32.lib`, `glu32.lib`: Βιβλιοθήκες για OpenGL και GLU στο φάκελο "`C:\Program Files\Microsoft SDKs\Windows\v6.0A\lib`". Αυτός ο φάκελος θα βρίσκεται στο περιβάλλον `LIB` ή στην επιλογή `/L<dir>` της γραμμής εντολών.Οι βιβλιοθήκες `opengl32.lib` και `glu32.lib` θα περιέχονται σαν διασυνδέσεις εισόδου ,μέσω της επιλογής `/link<lib>` or `/l<lib>` ,της γραμμής εντολών.
   - `opengl32.dll`, `glu32.dll`: Δυναμικής σύνδεσης βιβλιοθήκες για OpenGL και GLU στο φάκελο "`C:\Windows\System32`".".Αυτός ο φάκελος θα περιέχεται στο περιβάλλον `PATH`.

Εαν χρησιμοποιείται VC++ IDE, το header path και το lib path θα έχουν ρυθμιστεί σωστά. Εαν χρησιμοποιείται CMD shell, πρέπει να τρέξουμε το batch file "vcvars32.bat" (στο "C:\Program Files\Microsoft Visual Studio 9.0\VC\bin") ή αντίθετα να τρέξουμε το "Visual Studio 200X Command Prompt".

3. Τη **GLUT (OpenGL Utility Toolkit)**: Κατεβάζουμε τη Nate Robin's αυθεντική Win32 πόρτα της GLUT στο @ http://www.xmission.com/~nate/glut.html ή τη freeglut @ http://freeglut.sourceforge.net. Κάνουμε Unzip και μετά copy τη "glut.h"στο"C:\ProgramFiles\MicrosoftSDKs\Windows\v6.0A\include\gl",την "glut32.lib" στο "C:\Program Files\Microsoft SDKs\Windows\v6.0A\lib", και το αρχείο "glut32.dll" στο "C:\Windows\System32" (στο ίδιο location με την OpenGL και την GLU).

Για να γράψουμε το πρόγραμμά μας στη VISUAL C++ ,με χρήση της GLUT :

1. Ανοίγουμε το Visual C++ 2008 Express.
2. Δημιουργούμε ένα "Win32 Console Application" project (σημειώνεται οτι η GLUT τρέχει ως "Console Application"): Επιλέγουμε "File" menu ⇒ New ⇒ Project... ⇒ Στο "Project Types", επιλέγουμε "Visual C++", "Win32". Στο "Templates", επιλέγουμε "Win32 Console Application". Στο "Location",δηλώνουμε τον τρέχον κατάλογο εργασίας μας. Στο "Name", δίνουμε "FirstOpenGLProject" ⇒ Next ⇒ Επιλέγουμε "Empty Project" ⇒ Finish.
3. Δημιουργούμε ένα καινούργιο πηγαίο αρχείο : Right-click στο "Source Files" του project name ⇒ Add ⇒ New Item... ⇒ Στο "Categories", επιλέγουμε "Visual C++", "Code".Στο "Templates",επιλέγουμε "C++ File (.cpp)".Στο "Name", δίνουμε "GL01GettingStarted.cpp" ⇒ Add.
4. Στο panel του editor για "GL01GettingStarted.cpp",δίνουμε τον ακόλουθο κώδικα, ο οποίος δημιουργεί τρία σχήματα (κόκκινο τετράγωνο,μπλε πολύγωνο και πράσινο τρίγωνο) :

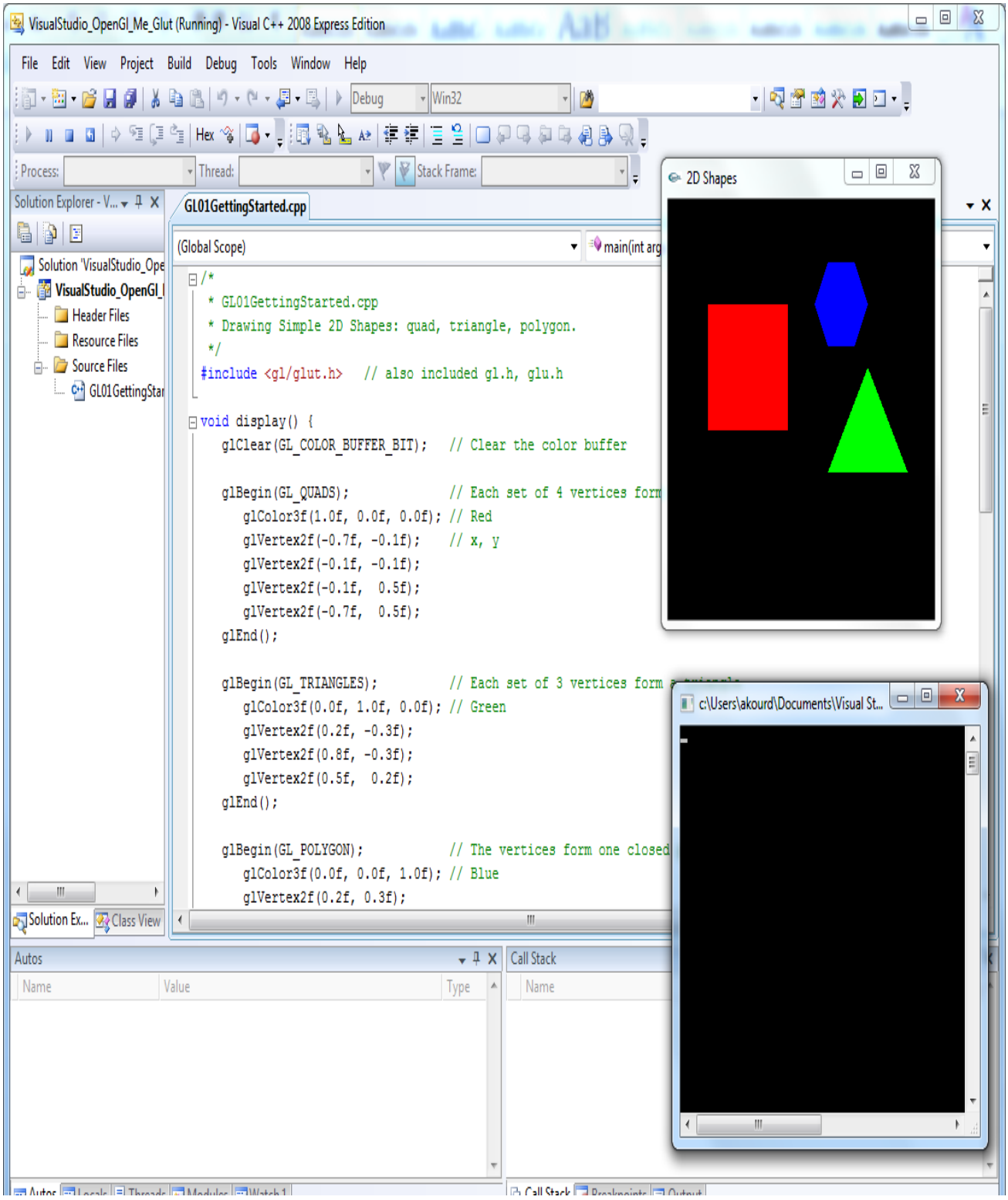
```
/*
 * GL01GettingStarted.cpp
 * Drawing Simple 2D Shapes: quad, triangle, polygon.
 */
#include <gl/glut.h>   // also included gl.h, glu.h

void display() {
   glClear(GL_COLOR_BUFFER_BIT);   // Clear the color buffer

   glBegin(GL_QUADS);              // Each set of 4 vertices form a quad
      glColor3f(1.0f, 0.0f, 0.0f); // Red
      glVertex2f(-0.7f, -0.1f);    // x, y
      glVertex2f(-0.1f, -0.1f);
      glVertex2f(-0.1f,  0.5f);
      glVertex2f(-0.7f,  0.5f);
   glEnd();
```

```
   glBegin(GL_TRIANGLES);          // Each set of 3 vertices form a
triangle
      glColor3f(0.0f, 1.0f, 0.0f); // Green
      glVertex2f(0.2f, -0.3f);
      glVertex2f(0.8f, -0.3f);
      glVertex2f(0.5f,  0.2f);
   glEnd();

   glBegin(GL_POLYGON);            // The vertices form one closed polygon
      glColor3f(0.0f, 0.0f, 1.0f); // Blue
      glVertex2f(0.2f, 0.3f);
      glVertex2f(0.4f, 0.3f);
      glVertex2f(0.5f, 0.5f);
      glVertex2f(0.4f, 0.7f);
      glVertex2f(0.2f, 0.7f);
      glVertex2f(0.1f, 0.5f);
   glEnd();

   glFlush();  // Render now
}

// GLUT runs as a Console Application
int main(int argc, char** argv) {
   glutInit(&argc, argv);          // Initialize GLUT
   glutCreateWindow("2D Shapes");  // Create a window with the given title
   glutDisplayFunc(display);       // Register callback handler for window
re-paint
   glutMainLoop();                 // Enter infinitely event-processing
loop
   return 0;
}
```

4.Κάνουμε Build το solution ("Build" menu ⇒ Build Solution) και τρέχουμε το πρόγραμμα ("Debug" menu ⇒ "Start Without Debugging").

Το αποτέλεσμα του παραπάνω προγράμματος μέσα στο περιβάλλον της Visual C++ θα είναι αυτό της παρακάτω εικόνας.

5. ΠΑΡΑΔΕΙΓΜΑΤΑ ΕΦΑΡΜΟΓΩΝ OpenGl



Χρήση του Dev C++

Dev C++ και OpenGL (και Pthreads)

Downloads

Χρειαζόμαστε τα εξής :

1) Κατεβάζουμε το Dev-C++ 5.0 beta 9.2 (4.9.9.2) με το Mingw/GCC 3.4.2 απο το www.bloodshed.net

2) Κατεβάζουμε τη glut (OpenGL) απο το devpaks.org

Αν χρησιμοποιείται threading:

3) Κατεβάζουμε το Pthreads απο το devpaks.org

4)Κατεβάζουμε το pthreadgc2.dll απο το www.dll-files.com

Εναλλακτικός τρόπος είναι να κατεβάσουμε όλα τα παραπάνω σε ένα αρχείο zip , εδώ : zip file.

Απαραίτητα :

-Εγκατάσταση του Dev C++ με click στον installer

-Μέσα στο Dev C++ ανοίγουμε τον package manager :

Tools->Package Manager

-Φορτώνουμε τη glut και το Pthreads (εαν χρησιμοποιούμε threading) package:

Package->Install Package

-Exit package manager

-Compiling and running

-Δημιουργούμε ένα καινούργιο project:

File->New->Project...

-Επιλέγουμε «Empty Project type»στη βασική καρτέλα παραθύρου

-Βεβαιωνόμαστε οτι είναι επιλεγμένο «C++ project»

-Δημιουργούμε ένα αντίγραφο του pthreadgc2.dll (?)που πιθανόν να χρειαστεί ,στον ίδιο φάκελο όταν τρέχουμε το πρόγραμμα , εάν χρησιμοποιείται threading.

-Κάνουμε Add το .cpp αρχείο στο project :

Project->add to project

-Επιλέγουμε το .cpp αρχείο

-Δηλώνουμε τα linker options:

Project->Project options

-Στο Parameters tab window προσθέτουμε την ακόλουθη γραμμή στο Linker pane:

« -lglut32 -lglu32 -lopengl32 -lwinmm -lgdi32 »

-Κάνουμε click στο ok

-Αν χρησιμοποιούμε Pthreads πρέπει επίσης να προσθέσουμε :

« -lpthreadGC2»

στο Linker pane

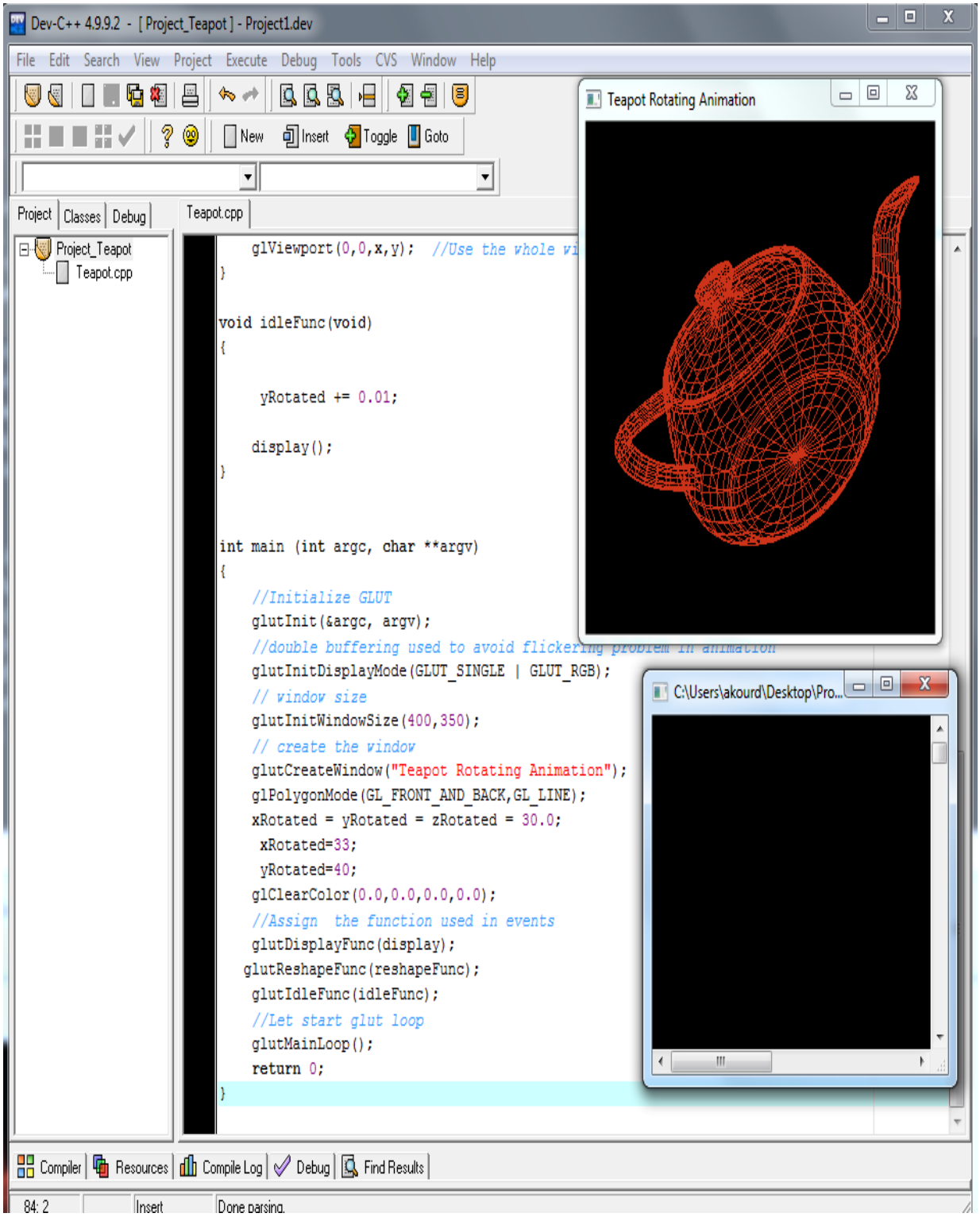
-Κάνουμε Compile και run το project:

Execute->Compile & run

Τρέχοντας τον ακόλουθο κώδικα,ο οποίος βρίσκεται στο πηγαίο αρχείο Teapot.cpp,του Project με όνομα Project_Teapot,παίρνουμε το αποτέλεσμα της παρακάτω εικόνας(μια περιστρεφόμενη τσαγιέρα).

```
#include <GL\glut.h>

GLfloat xRotated, yRotated, zRotated;
GLdouble size=1;


void display(void)
{

    glMatrixMode(GL_MODELVIEW);
    // clear the drawing buffer.
    glClear(GL_COLOR_BUFFER_BIT);
    // clear the identity matrix.
    glLoadIdentity();
    // traslate the draw by z = -4.0
    // Note this when you decrease z like -
8.0 the drawing will looks far , or smaller.
    glTranslatef(0.0,0.0,-4.5);
    // Red color used to draw.
    glColor3f(0.8, 0.2, 0.1);
    // changing in transformation matrix.
    // rotation about X axis
    glRotatef(xRotated,1.0,0.0,0.0);
    // rotation about Y axis
    glRotatef(yRotated,0.0,1.0,0.0);
    // rotation about Z axis
    glRotatef(zRotated,0.0,0.0,1.0);
    // scaling transfomation
    glScalef(1.0,1.0,1.0);
    // built-in (glut library) function , draw you a Teapot.
    glutSolidTeapot(size);
    // Flush buffers to screen

    glFlush();
    // sawp buffers called because we are using double buffering
   // glutSwapBuffers();
}

void reshapeFunc(int x, int y)
{
    if (y == 0 || x == 0) return;  //Nothing is visible then, so ret
urn
    //Set a new projection matrix
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    //Angle of view:40 degrees
    //Near clipping plane distance: 0.5
    //Far clipping plane distance: 20.0

    gluPerspective(40.0,(GLdouble)x/(GLdouble)y,0.5,20.0);
```

```
    glViewport(0,0,x,y);  //Use the whole window for rendering
}

void idleFunc(void)
{

     yRotated += 0.01;

    display();
}


int main (int argc, char **argv)
{
    //Initialize GLUT
    glutInit(&argc, argv);
    //double buffering used to avoid flickering problem in animation
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    // window size
    glutInitWindowSize(400,350);
    // create the window
    glutCreateWindow("Teapot Rotating Animation");
    glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
    xRotated = yRotated = zRotated = 30.0;
     xRotated=33;
     yRotated=40;
    glClearColor(0.0,0.0,0.0,0.0);
    //Assign  the function used in events
    glutDisplayFunc(display);
   glutReshapeFunc(reshapeFunc);
    glutIdleFunc(idleFunc);
    //Let start glut loop
    glutMainLoop();
    return 0;
}
```


Dev-C++ 4.9.9.2 - [Project_Teapot] - Project1.dev
File Edit Search View Project Execute Debug Tools CVS Window Help
New Insert Toggle Goto
Project Classes Debug
Project_Teapot
Teapot.cpp
Teapot.cpp
glViewport(0,0,x,y); //Use the whole wi
}
void idleFunc(void)
{
yRotated += 0.01;
display();
}
int main (int argc, char **argv)
{
//Initialize GLUT
glutInit(&argc, argv);
//double buffering used to avoid flickering problem in animation
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
// window size
glutInitWindowSize(400,350);
// create the window
glutCreateWindow("Teapot Rotating Animation");
glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
xRotated = yRotated = zRotated = 30.0;
xRotated=33;
yRotated=40;
glClearColor(0.0,0.0,0.0,0.0);
//Assign the function used in events
glutDisplayFunc(display);
glutReshapeFunc(reshapeFunc);
glutIdleFunc(idleFunc);
//Let start glut loop
glutMainLoop();
return 0;
}
Teapot Rotating Animation
C:\Users\akourd\Desktop\Pro...
Compiler Resources Compile Log Debug Find Results
84: 2 Insert Done parsing.

OpenGl me χρήση Java (JoGl)

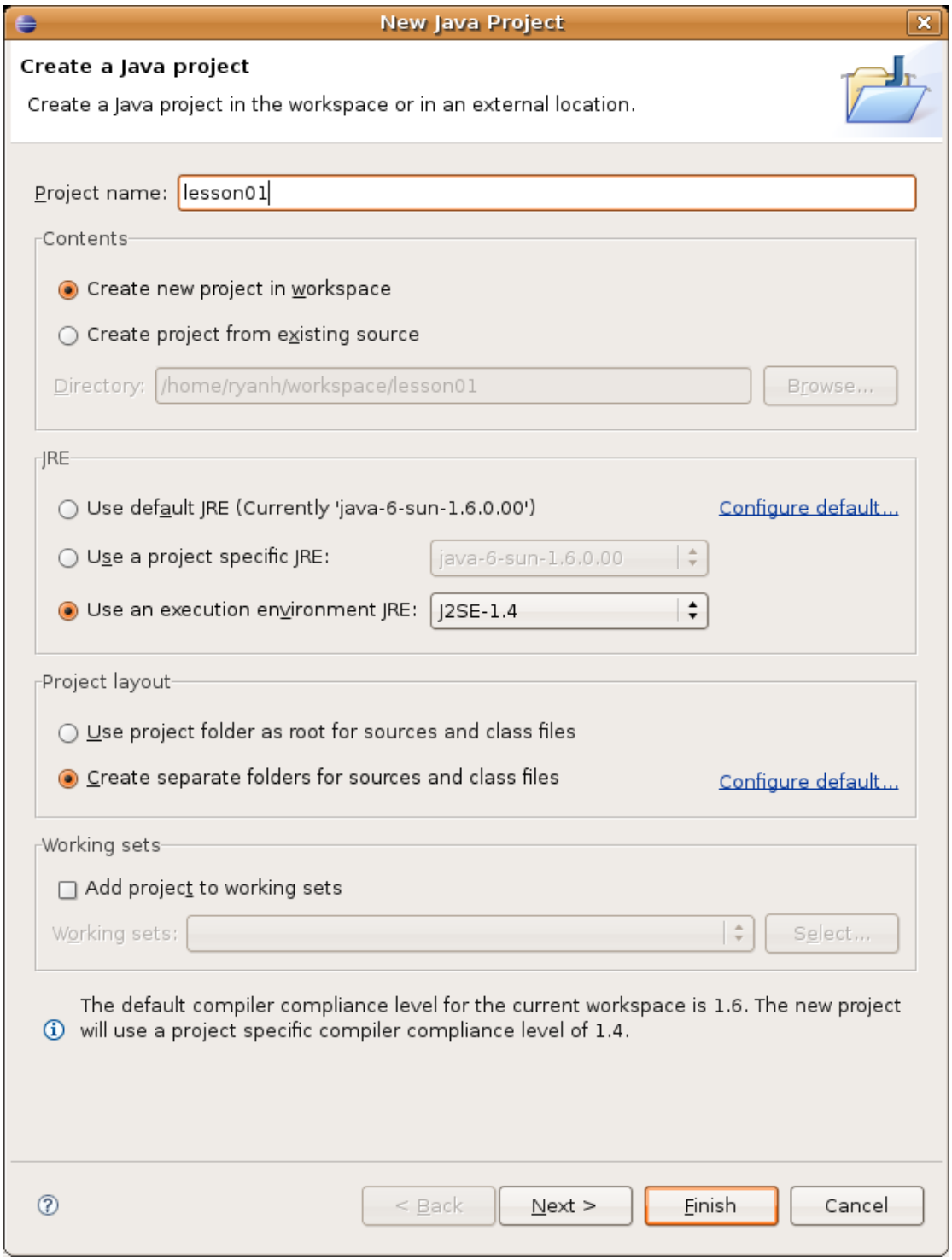
Αυτά που χρειαζόμαστε είναι :

- Το Eclipse Europa 3.3 (δική μας επιλογή)
- Το JAVA SDK
- Τα jogl binaries αρχεία για την πλατφόρμα μας.
- Το jogl src

Δημιουργία του Project

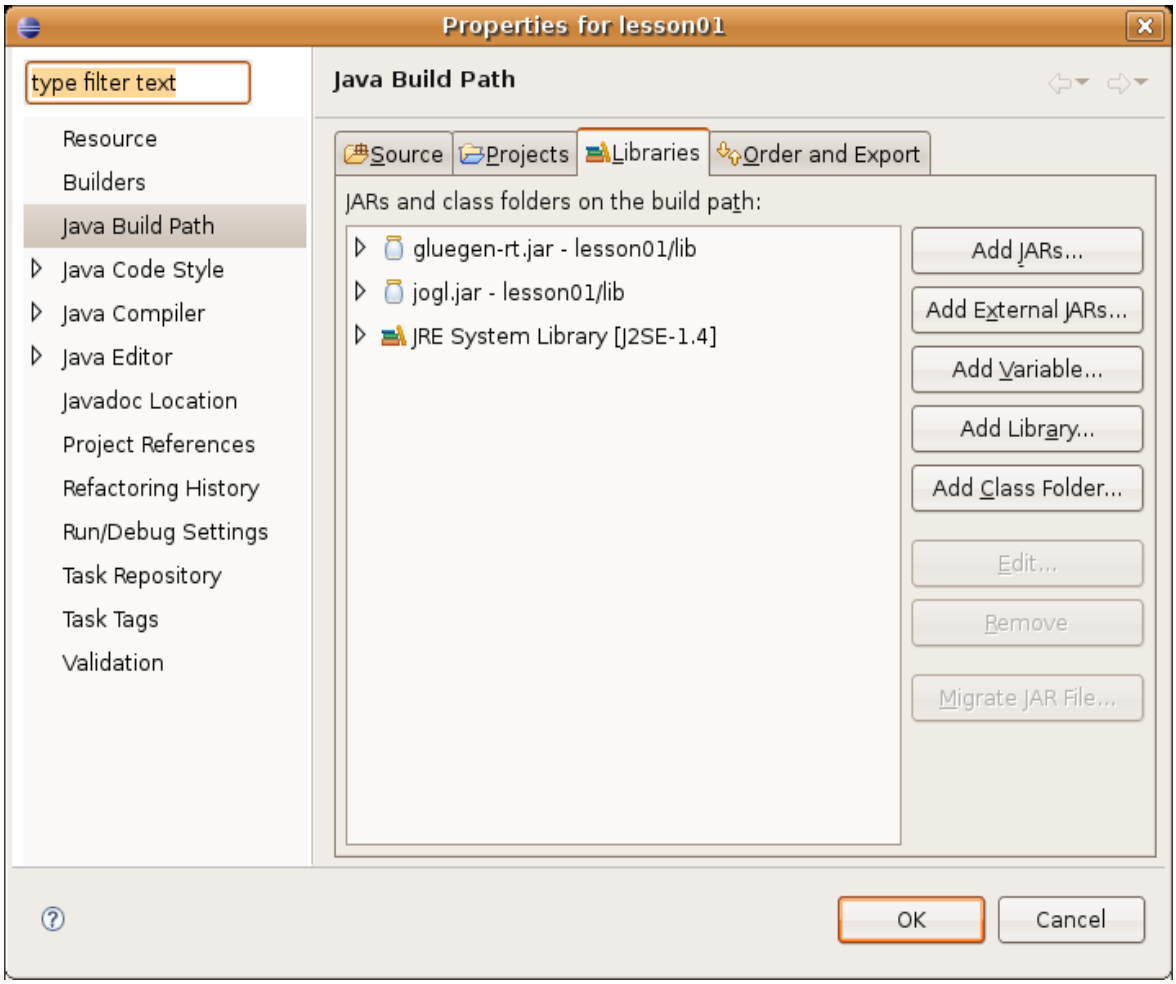
Τα παρακάτω, περιγράφουν την ενσωμάτωση της JoGl στο Eclipse Europa 3.3 και τη δημιουργία ενός νέου Java Project,του οποίου οι κλάσεις,θα κάνουν χρήση της.Το αποτέλεσμα αυτού του Project (3D χρωματιστά σχήματα), που θα παρουσιαστεί αργότερα σε εικόνα,είναι αυτό των κλάσεων Lesson01.java ,Lesson01Applet.java , Lesson01Window.java τις οποίες συμπεριλάβαμε στο τέλος,μέσα στον φάκελο src και αφού ολοκληρώσαμε τα παρακάτω βήματα.Τα αρχεία αυτά καθώς και όλα τα απαραίτητα αρχεία JoGl ,βρίσκονται επίσης μέσα στο ολοκληρωμένο Project που βρίσκεται στη διεύθυνση http://timelessname.com/jogl/lesson01/ και στο CD που συνοδεύει την παρούσα πτυχιακή εργασία.

Το πρώτο λοιπόν πράγμα που πρέπει να γίνει,είναι να δημιουργηθεί ενα καινούργιο Project. Επιλέγουμε File > New >Java Project .

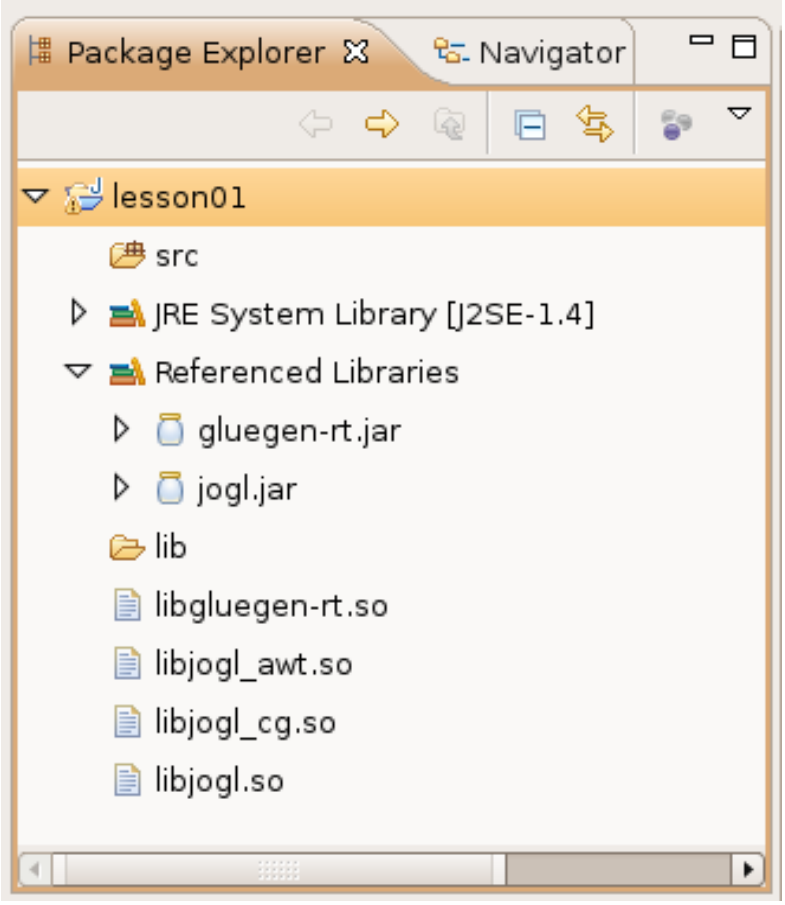


Προσθήκη απαραίτητων λειτουργιών

Δημιουργούμε ένα νέο φάκελο στο Project και του δίνουμε το όνομα *lib*. Προσθέτουμε το *jogl.jar* και το *gluegen-rt.jar* στο νέο μας φάκελο lib. Παίρνουμε τις ειδίκες βιβλιοθήκες πλατφόρμας τις οποίες συνήθως συναντούμε με ονόματα όπως libjogl/libjogl_cg/libjogl_awt/libgluegen-rt και τις τοποθετούμε στον υψηλοτέρου επιπέδου φάκελο του Project μας. Πηγαίνουμε *Project->Properties->Java Build Path* και πατάμε στην καρτέλα *Libraries*. Προσθέτουμε *jogl.jar* και *gluegen-rt.jar*.

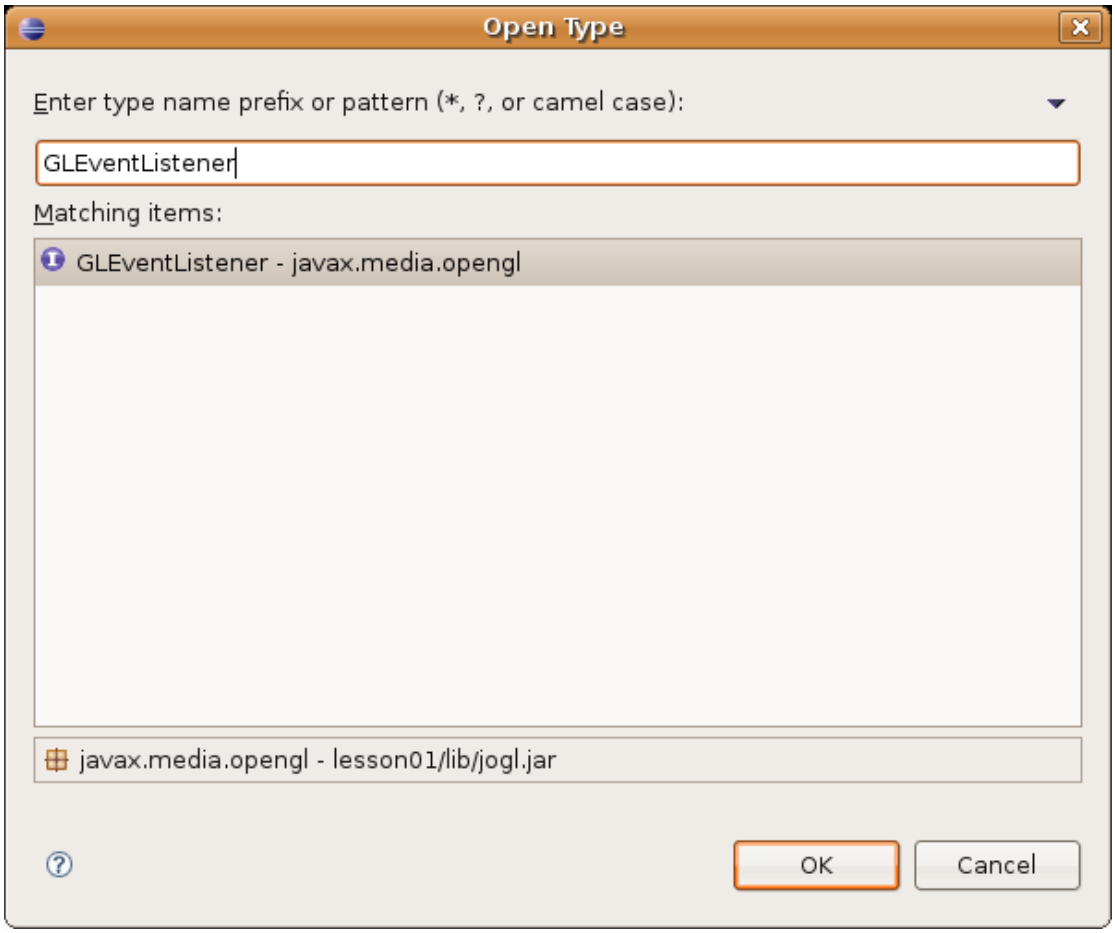


Το Project μας θα φαίνεται κάπως έτσι :



Τώρα βάζουμε το jogl source zip (jogl-1.1.0-src.zip) στο φάκελο *lib* . Πατάμε ctrl+shift+t και θα εμφανιστεί ειδοποίηση ωστε να δώσουμε class name. Δίνουμε όνομα GLEventListener και αμέσως εντοπίζεται μέσω της αναζήτησης του Eclipse.Την επιλέγουμε και πατάμε OK.

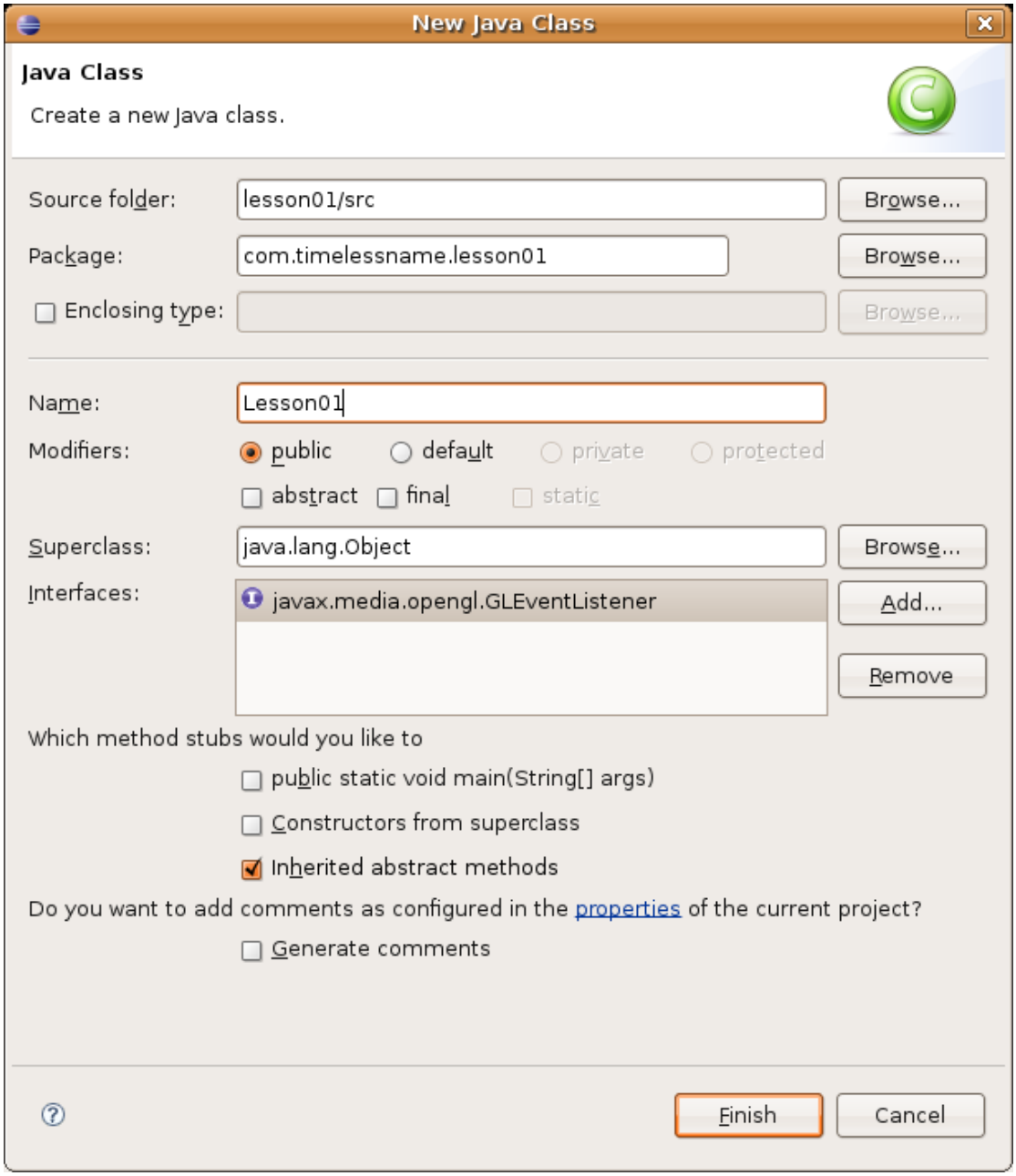
Αν παρατηρηθεί πως δεν υπάρχει source,κάνουμε Click στο Attach source και κάνουμε add το jogl source zip (jogl-1.1.0-src.zip).



Οι πρώτες κλάσεις

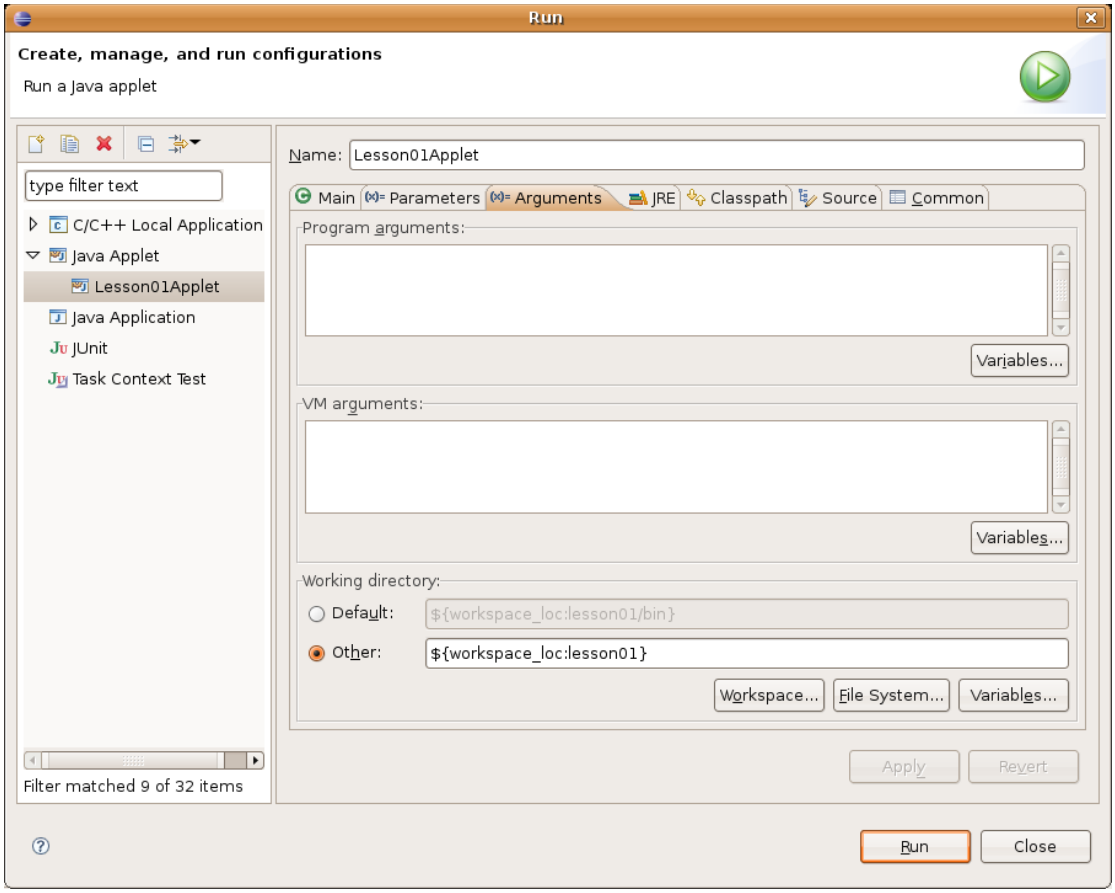
Κάνουμε δεξί click στον φάκελο source(src) και πηγαίνουμε *New->Class*. Αφήνουμε τα name και package όπως έχουν. Κάνουμε Add το interface της GLEventListener.

Το όνομα πακέτου com.timelessname.lesson01 , αναφέρεται στο όνομα που απο κατασκευής έδωσαν στο πακέτο που βρίσκονται τα αρχεία των κλάσεων java του Project μας (http://timelessname.com/jogl/lesson01/) και το οποίο βρίσκεται στο συνοδευτικό CD.



Έτσι θα δημιουργηθεί μια νέα κλάση και θα αντιστοιχιστούν σε αυτή οι interfaces methods με τα σωστά ονόματα μεταβλητών. Μπορούμε να συμπεριλάβουμε αυτές τις μεθόδους με τον κώδικα του αρχείου: Lesson01.java.

Τώρα θα χρειαστούμε κλάσεις για να χρησιμοποιήσουμε αυτό το αρχείο είτε ως applet ,είτε ως frame κι έτσι προσθέτουμε αυτά τα δύο αρχεία στο project: Lesson01Applet.java Lesson01Window.java. Ξεκινώντας,το παράθυρο πρέπει να είναι μπροστά. Ωστόσο,η applet πρέπει να έχει ως working directory ,το root του project.



Παρακάτω φαίνεται το αποτέλεσμα (3D χρωματιστά σχήματα) που θα έχουμε,εαν συμπεριλάβουμε τις κλάσεις Lesson01Applet.java

και Lesson01Window.java στο φάκελο src και έπειτα τρέξουμε το Project.Πριν το αποτέλεσμα,παρουσιάζεται ο κώδικας της κάθε κλάσης.

Lesson01.java

```
package com.timelessname.lesson01;

import javax.media.opengl.GL;
import javax.media.opengl.GLAutoDrawable;
import javax.media.opengl.GLEventListener;
import javax.media.opengl.glu.GLU;

public class Lesson01 implements GLEventListener {

	protected float pyramidRotation;
	protected float cubeRotation;

	public void display(GLAutoDrawable drawable) {
		final GL gl = drawable.getGL();
		gl.glClear(GL.GL_COLOR_BUFFER_BIT |
GL.GL_DEPTH_BUFFER_BIT);

		gl.glLoadIdentity();
		gl.glTranslatef(-1.5f,0.0f,-6.0f);
		gl.glRotatef(pyramidRotation,0.0f,1.0f,0.0f);
		drawPyramid(gl);
		pyramidRotation+=0.2f;

		gl.glLoadIdentity();
		gl.glTranslatef(1.5f,0.0f,-7.0f);

		gl.glRotatef(cubeRotation,1.0f,1.0f,1.0f);

		drawCube(gl);
		cubeRotation-=0.15f;
	}

	protected void drawPyramid(GL gl){
		gl.glBegin(GL.GL_TRIANGLES);

			gl.glColor3f(1.0f,0.0f,0.0f);
			gl.glVertex3f( 0.0f, 1.0f, 0.0f);

			gl.glColor3f(0.0f,1.0f,0.0f);
			gl.glVertex3f(-1.0f,-1.0f, 1.0f);

			gl.glColor3f(0.0f,0.0f,1.0f);
```

```
                gl.glVertex3f( 1.0f,-1.0f, 1.0f);

                gl.glColor3f(1.0f,0.0f,0.0f);
                gl.glVertex3f( 0.0f, 1.0f, 0.0f);

                gl.glColor3f(0.0f,0.0f,1.0f);
                gl.glVertex3f( 1.0f,-1.0f, 1.0f);

                gl.glColor3f(0.0f,1.0f,0.0f);
                gl.glVertex3f( 1.0f,-1.0f, -1.0f);

                gl.glColor3f(1.0f,0.0f,0.0f);
                gl.glVertex3f( 0.0f, 1.0f, 0.0f);

                gl.glColor3f(0.0f,1.0f,0.0f);
                gl.glVertex3f( 1.0f,-1.0f, -1.0f);

                gl.glColor3f(0.0f,0.0f,1.0f);
                gl.glVertex3f(-1.0f,-1.0f, -1.0f);

                gl.glColor3f(1.0f,0.0f,0.0f);
                gl.glVertex3f( 0.0f, 1.0f, 0.0f);

                gl.glColor3f(0.0f,0.0f,1.0f);
                gl.glVertex3f(-1.0f,-1.0f,-1.0f);

                gl.glColor3f(0.0f,1.0f,0.0f);
                gl.glVertex3f(-1.0f,-1.0f, 1.0f);

        gl.glEnd();
    }

    protected void drawCube(GL gl){
        gl.glBegin(GL.GL_QUADS);
                gl.glColor3f(0.0f,1.0f,0.0f);
                gl.glVertex3f( 1.0f, 1.0f,-1.0f);

                gl.glVertex3f(-1.0f, 1.0f,-1.0f);

                gl.glVertex3f(-1.0f, 1.0f, 1.0f);

                gl.glVertex3f( 1.0f, 1.0f, 1.0f);

                gl.glColor3f(1.0f,0.5f,0.0f);
                gl.glVertex3f( 1.0f,-1.0f, 1.0f);

                gl.glVertex3f(-1.0f,-1.0f, 1.0f);

                gl.glVertex3f(-1.0f,-1.0f,-1.0f);

                gl.glVertex3f( 1.0f,-1.0f,-1.0f);

                gl.glColor3f(1.0f,0.0f,0.0f);
                gl.glVertex3f( 1.0f, 1.0f, 1.0f);
```

```
                gl.glVertex3f(-1.0f, 1.0f, 1.0f);

                gl.glVertex3f(-1.0f,-1.0f, 1.0f);

                gl.glVertex3f( 1.0f,-1.0f, 1.0f);

                gl.glColor3f(1.0f,1.0f,0.0f);
                gl.glVertex3f( 1.0f,-1.0f,-1.0f);

                gl.glVertex3f(-1.0f,-1.0f,-1.0f);

                gl.glVertex3f(-1.0f, 1.0f,-1.0f);

                gl.glVertex3f( 1.0f, 1.0f,-1.0f);

                gl.glColor3f(0.0f,0.0f,1.0f);
                gl.glVertex3f(-1.0f, 1.0f, 1.0f);

                gl.glVertex3f(-1.0f, 1.0f,-1.0f);

                gl.glVertex3f(-1.0f,-1.0f,-1.0f);

                gl.glVertex3f(-1.0f,-1.0f, 1.0f);

                gl.glColor3f(1.0f,0.0f,1.0f);
                gl.glVertex3f( 1.0f, 1.0f,-1.0f);

                gl.glVertex3f( 1.0f, 1.0f, 1.0f);

                gl.glVertex3f( 1.0f,-1.0f, 1.0f);

                gl.glVertex3f( 1.0f,-1.0f,-1.0f);

          gl.glEnd();
     }

     public void displayChanged(GLAutoDrawable drawable,
boolean modeChanged, boolean deviceChanged) {

     }

     public void init(GLAutoDrawable drawable) {
          final GL gl = drawable.getGL();
          gl.glShadeModel(GL.GL_SMOOTH);
          gl.glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
          gl.glClearDepth(1.0f);
          gl.glEnable(GL.GL_DEPTH_TEST);
          gl.glDepthFunc(GL.GL_LEQUAL);
          gl.glHint(GL.GL_PERSPECTIVE_CORRECTION_HINT,
GL.GL_NICEST);
     }

     public void reshape(GLAutoDrawable drawable, int x, int
y, int width, int height) {
          final GL gl = drawable.getGL();
          final GLU glu = new GLU();
```

```
            gl.setSwapInterval(1);

            gl.glViewport(0, 0, width, height);
            gl.glMatrixMode(GL.GL_PROJECTION);
            gl.glLoadIdentity();

            glu.gluPerspective(
                        45.0f,
                        (double) width / (double) height,
                        0.1f,
                        1000.0f);

            gl.glMatrixMode(GL.GL_MODELVIEW);
            gl.glLoadIdentity();
      }

}
```

Lesson01Applet.java

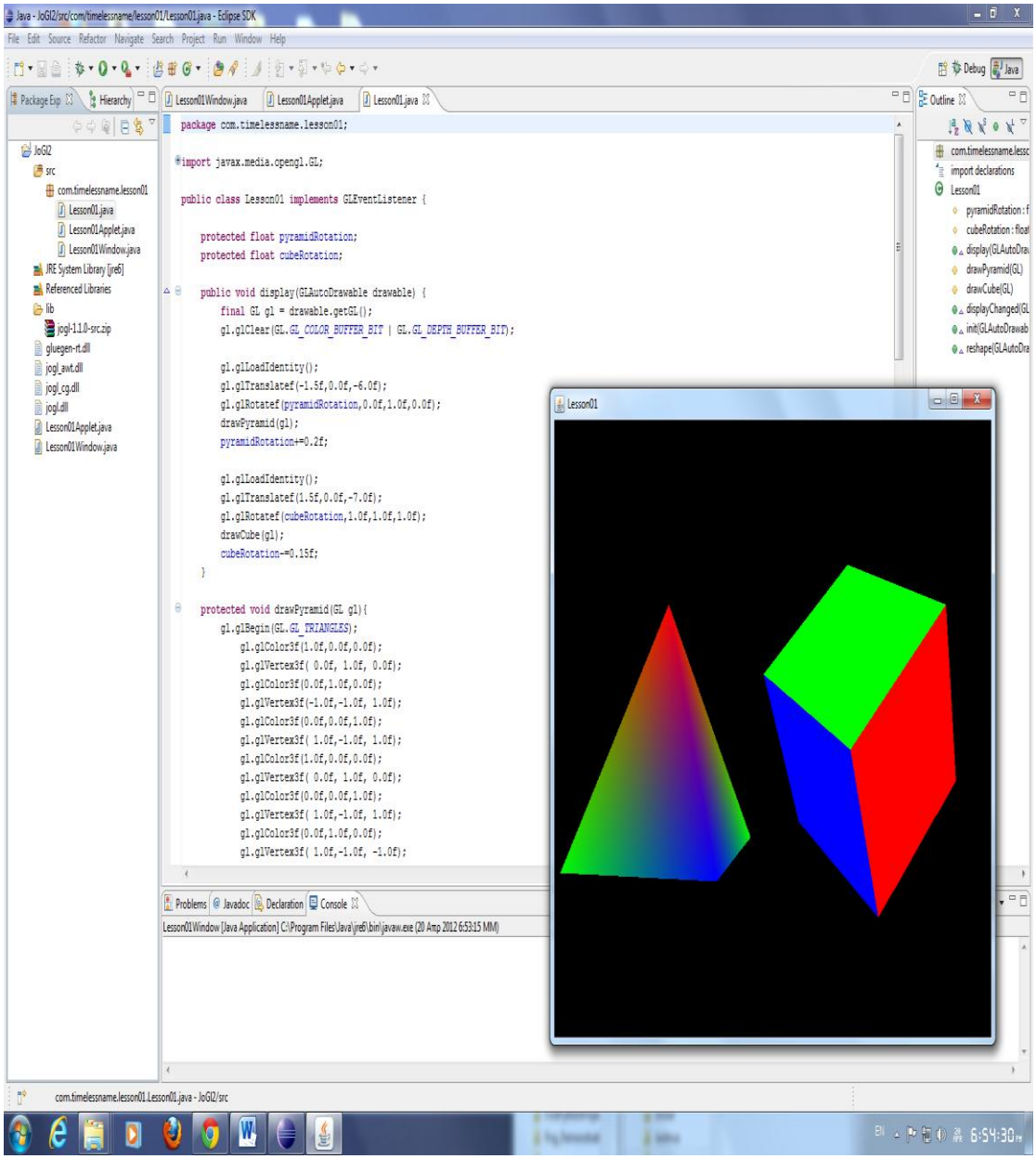
```
package com.timelessname.lesson01;

import java.applet.Applet;
import java.awt.BorderLayout;

import javax.media.opengl.GLCanvas;
import javax.media.opengl.GLCapabilities;

import com.sun.opengl.util.Animator;

public class Lesson01Applet extends Applet {

      private static final long serialVersionUID =
7403003329697278728L;

      protected GLCanvas canvas;
      protected Animator animator;

      public void init() {
            setLayout(new BorderLayout());
            GLCapabilities glCapabilities = new
GLCapabilities();

            glCapabilities.setDoubleBuffered(true);
            glCapabilities.setHardwareAccelerated(true);

            canvas = new GLCanvas(glCapabilities);
```

```
            animator = new Animator(canvas);

            canvas.addGLEventListener(new Lesson01());

            add(canvas, BorderLayout.CENTER);
      }

      public void start() {
            animator.setRunAsFastAsPossible(false);
            animator.start();
      }

      public void stop() {
            animator.stop();
      }

}
```

Lesson01Window.java

```
package com.timelessname.lesson01;

import java.awt.Frame;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.media.opengl.GLCanvas;

import com.sun.opengl.util.Animator;

public class Lesson01Window extends Frame {

      private static final long serialVersionUID =
7633042051769682994L;

      protected GLCanvas canvas;
      protected Animator animator;

      public static void main(String[] args) {
            Frame frame = new Frame("Lesson01");
          GLCanvas canvas = new GLCanvas();

          canvas.addGLEventListener(new Lesson01());
          frame.add(canvas);
          frame.setSize(800, 600);
          final Animator animator = new Animator(canvas);
          frame.addWindowListener(new WindowAdapter() {
              public void windowClosing(WindowEvent e) {
                new Thread(new Runnable() {
                    public void run() {
                      animator.stop();
```

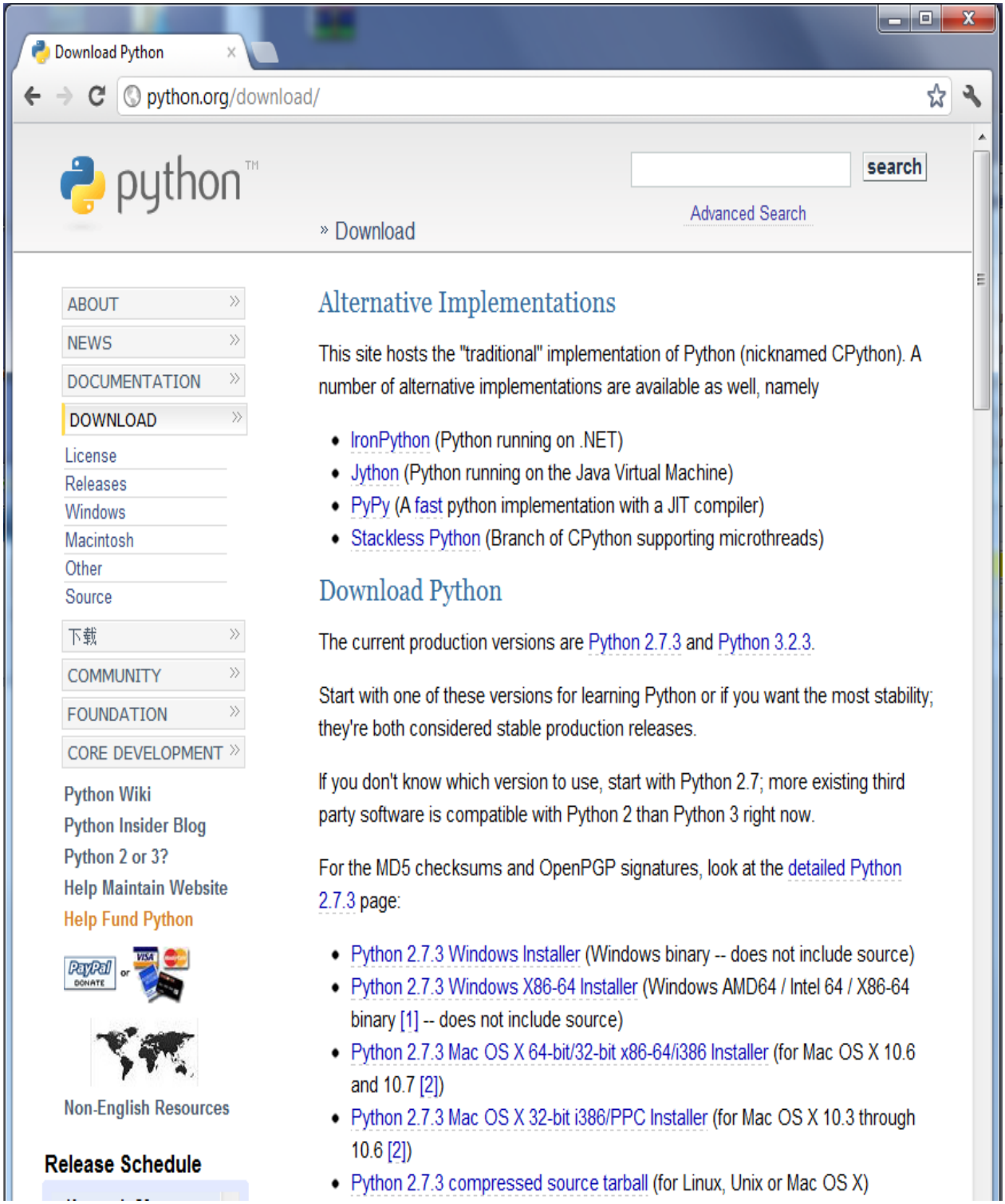
```
                    System.exit(0);
                }
            }).start();
        }
    });
    frame.setVisible(true);
    animator.start();
  }
}
```



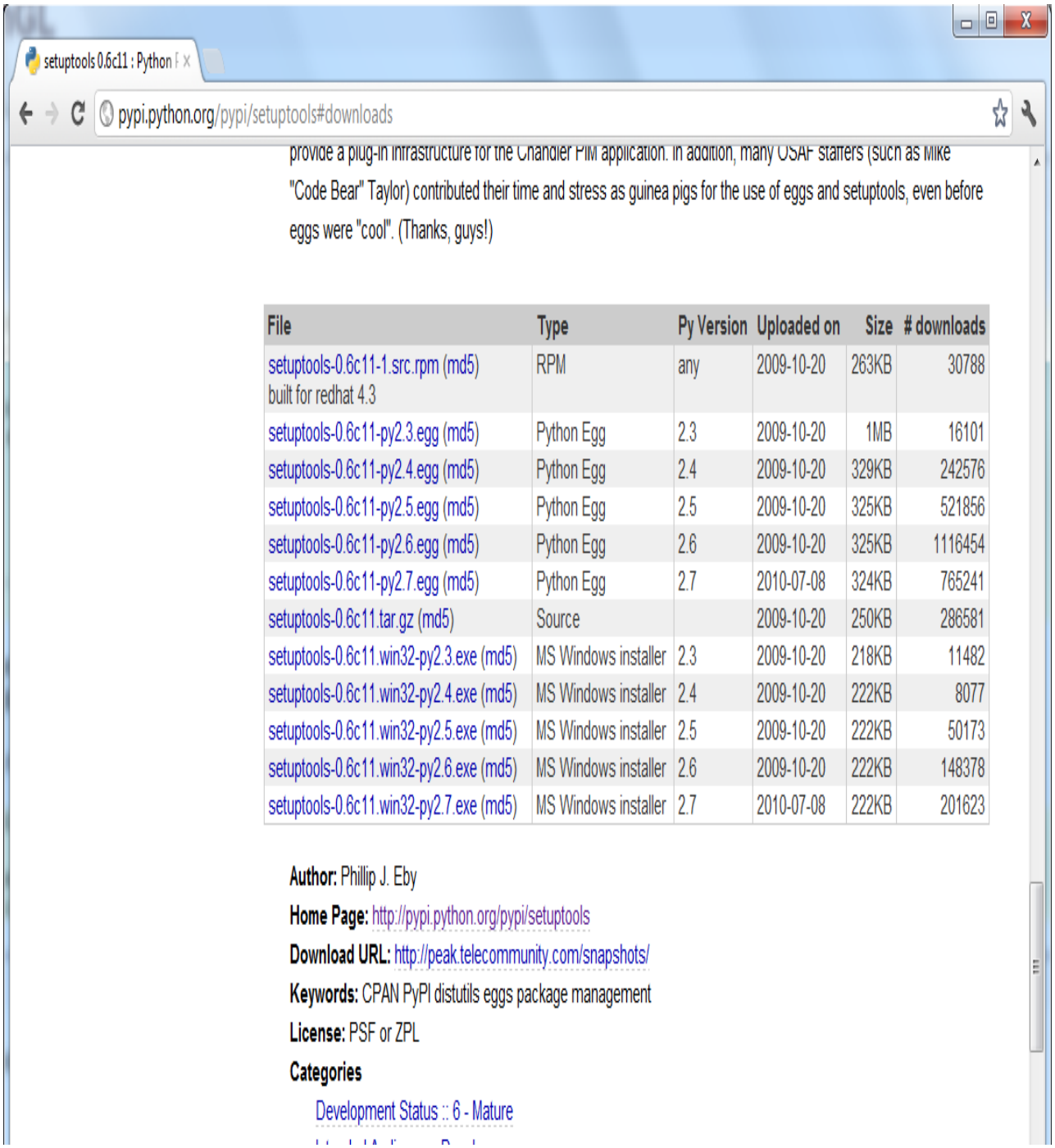
OpenGl με χρήση Python (PyGl)

Το πρώτο πράγμα που χρειάζεται να κάνουμε, είναι να εγκαταστήσουμε στον υπολογιστή μας την πλατφόρμα για τη γλώσσα προγραμματισμού Python. Αν και κυκλοφορούν και νεότερες εκδόσεις της,η έκδοση που επιλέχθηκε (τυχαία) είναι η Python 2.7 η οποία είναι ικανή να καλύψει τις ανάγκες της παρουσίασής μας σε περιβάλλον Microsoft Windows.

Επισκεπτόμαστε λοιπόν τη διεύθυνση http://python.org/download/ όπου είναι ο επίσημος ιστότοπος της Python και επιλέγουμε για κατέβασμα το αρχείο **Python 2.7.3 windows installer**,το οποίο τρέχει και εγκαθίσταται αυτόματα και το οποίο φαίνεται στην παρακάτω εικόνα.


Download Python
python.org/download/
python
search
Advanced Search
» Download
ABOUT
NEWS
DOCUMENTATION
DOWNLOAD
License
Releases
Windows
Macintosh
Other
Source
下载
COMMUNITY
FOUNDATION
CORE DEVELOPMENT
Python Wiki
Python Insider Blog
Python 2 or 3?
Help Maintain Website
Help Fund Python
Non-English Resources
Release Schedule
Alternative Implementations
This site hosts the "traditional" implementation of Python (nicknamed CPython). A number of alternative implementations are available as well, namely
IronPython (Python running on .NET)
Jython (Python running on the Java Virtual Machine)
PyPy (A fast python implementation with a JIT compiler)
Stackless Python (Branch of CPython supporting microthreads)
Download Python
The current production versions are Python 2.7.3 and Python 3.2.3.
Start with one of these versions for learning Python or if you want the most stability; they're both considered stable production releases.
If you don't know which version to use, start with Python 2.7; more existing third party software is compatible with Python 2 than Python 3 right now.
For the MD5 checksums and OpenPGP signatures, look at the detailed Python 2.7.3 page:
Python 2.7.3 Windows Installer (Windows binary -- does not include source)
Python 2.7.3 Windows X86-64 Installer (Windows AMD64 / Intel 64 / X86-64 binary [1] -- does not include source)
Python 2.7.3 Mac OS X 64-bit/32-bit x86-64/i386 Installer (for Mac OS X 10.6 and 10.7 [2])
Python 2.7.3 Mac OS X 32-bit i386/PPC Installer (for Mac OS X 10.3 through 10.6 [2])
Python 2.7.3 compressed source tarball (for Linux, Unix or Mac OS X)

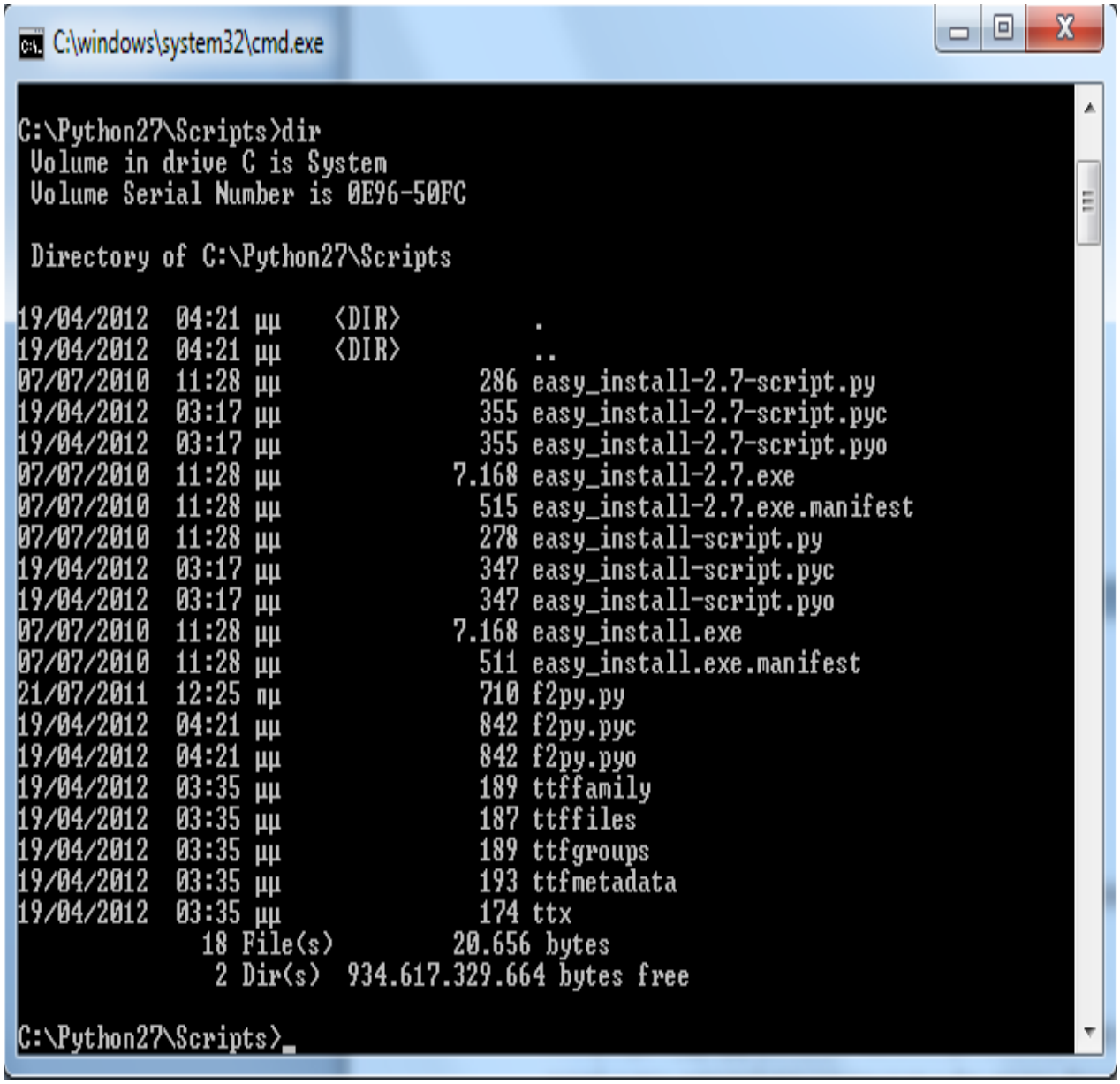
Στη συνέχεια κατεβάζουμε από το site http://pypi.python.org/pypi/setuptools#downloads το αρχείο setuptools-0.6c11.win32-py2.7.exe (md5),απαραίτητο για την σωστή λειτουργία της Python.



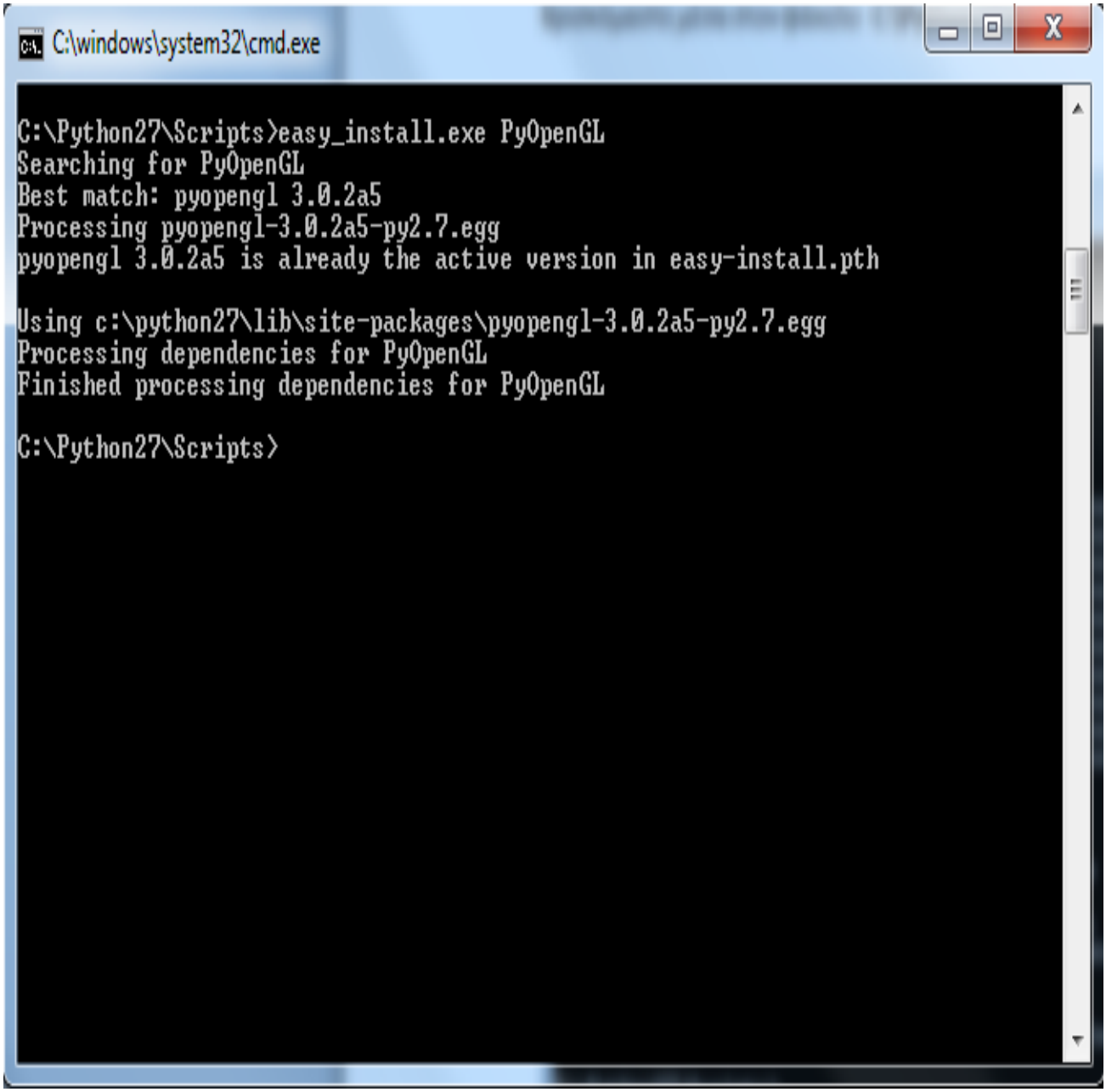
Κατα την προηγούμενη εγκατάσταση, αποθηκεύτηκε στον σκληρό δίσκο **C:** ,ο φάκελος **Python27.**

Ανοίγουμε τη γραμμή εντολών και δίνουμε εντολές ώστε να γίνει install των κατάλληλων αρχείων που απαιτούνται για την ενσωμάτωση της OpenGl στην Python .

Βρισκόμαστε μέσα στον φάκελο **C:\Python27\Scripts :**

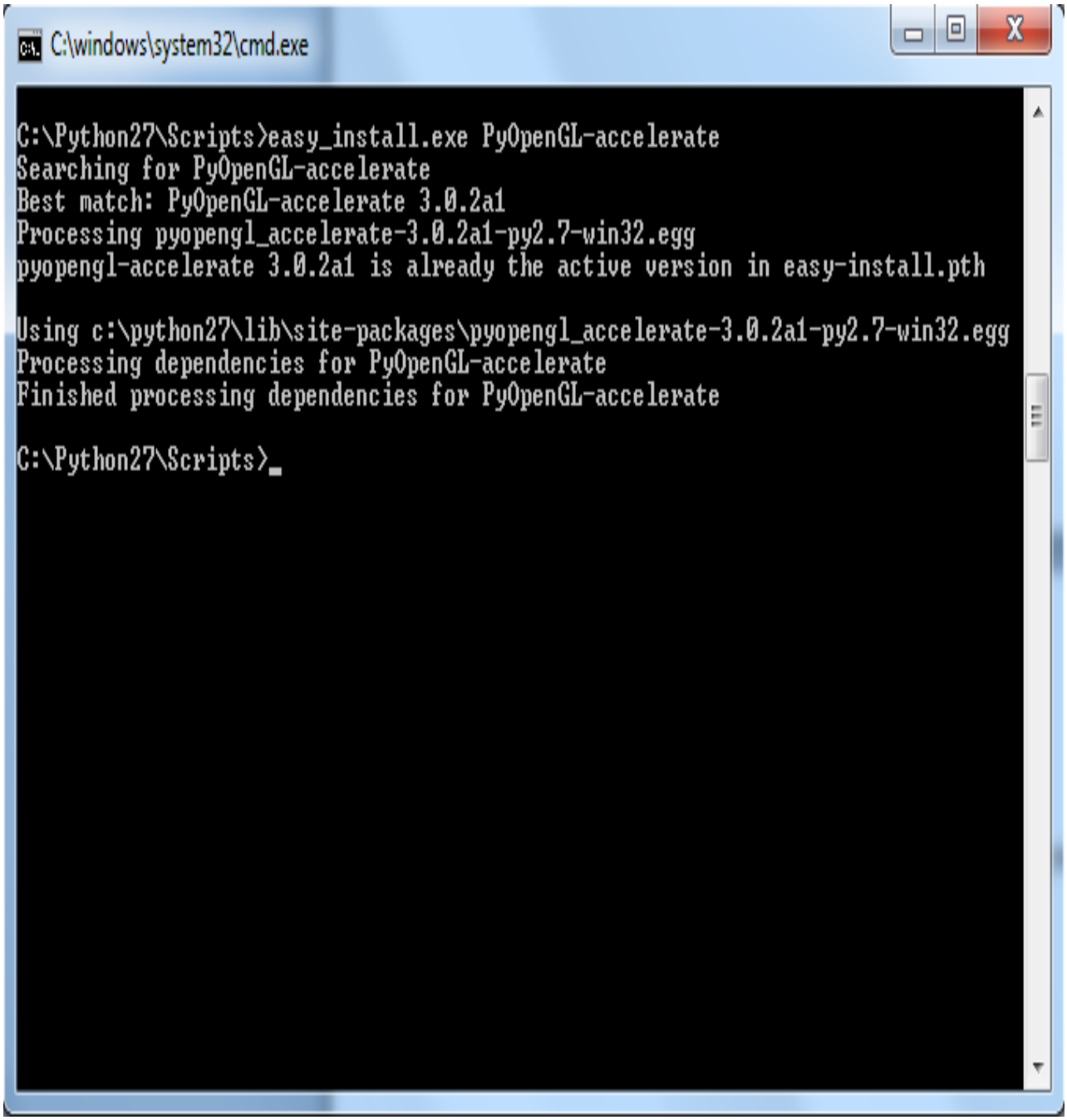


1) Τρέχουμε το αρχείο **easy_install.exe** ,δίνοντας του ως παράμετρο την τιμή **PyOpenGL**.



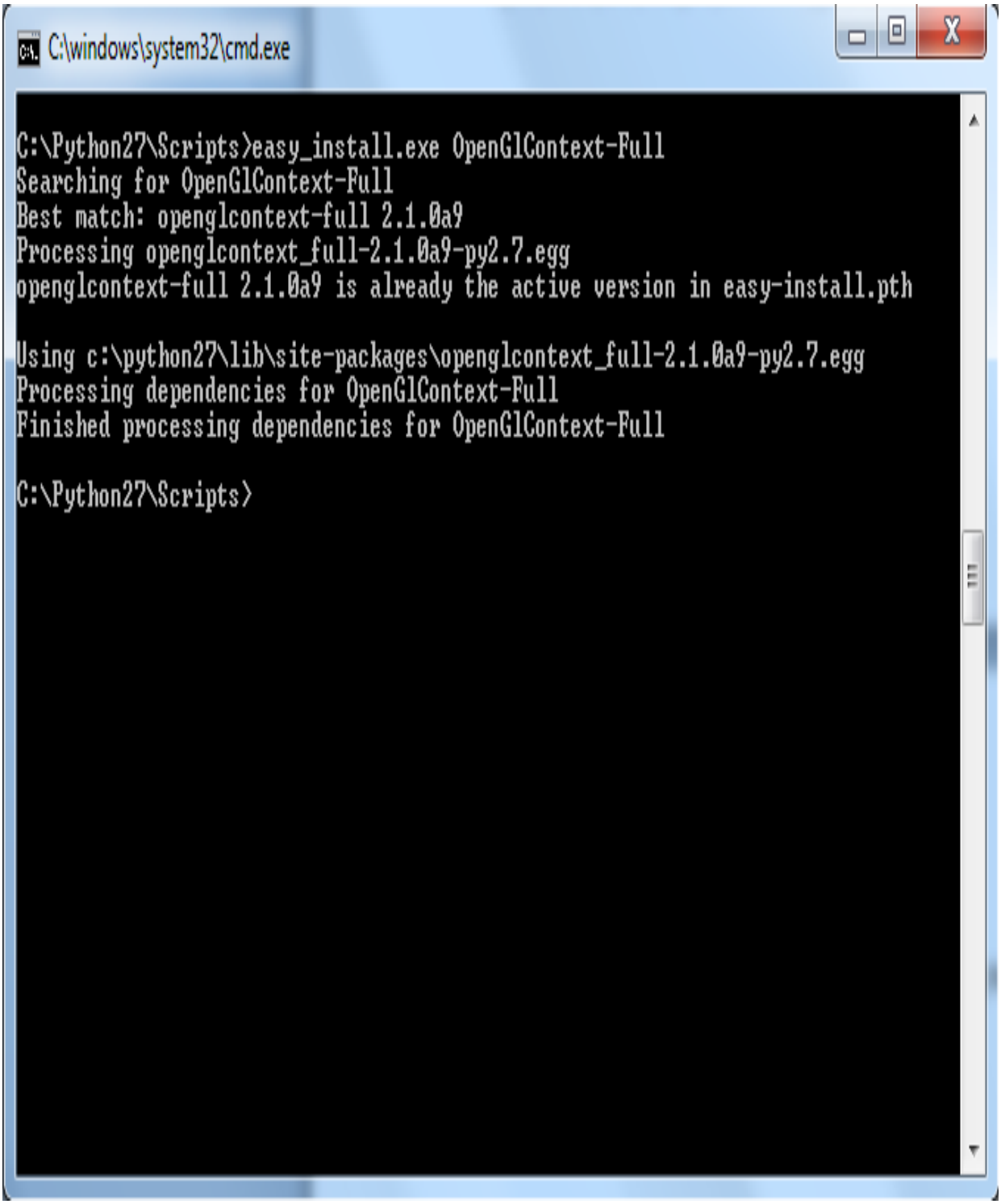
Όπως φαίνεται παραπάνω, το αρχείο **easy_install.exe**, εγκατέστησε τις απαραίτητες λειτουργίες για τη χρήση της OpenGl.

2) Τρέχουμε το αρχείο **easy_install.exe** ,δίνοντας του ως παράμετρο την τιμή **PyOpenGL-accelerate.**



Όπως φαίνεται παραπάνω, το αρχείο **easy_install.exe**, εγκατέστησε τις λειτουργίες που σχετίζονται με την ταχύτητα κατα τη χρήση της Python OpenGl.

3)Τρέχουμε το αρχείο **easy_install.exe** ,δίνοντας του ως παράμετρο την τιμή **PyOpenGLContext-Full**.



Όπως φαίνεται παραπάνω, το αρχείο **easy_install.exe**, εγκατέστησε κάποιες χρήσιμες αλλα προαιρετικές λειτουργίες για τη χρήση της Python OpenGl.

Ένα ακόμα βήμα είναι να συμπεριλάβουμε στο δυναμικό μας τη **GLUT :**

Από το site http://user.xmission.com/~nate/glut.html , κατεβάζουμε τον φάκελο **glut-3.7.6-bin.zip (117 KB)** και εν συνεχεία αντιγράφουμε το αρχείο **glut32.dll** που βρίσκεται μέσα σ'αυτόν, στο **C:\system.**

Επίσης μπορούμε να κατεβάσουμε τις βιβλιοθήκες **Numpy** και **Scipy**,οι οποίες χρησιμεύουν σε εφαρμογές που χρησιμοποιούν πίνακες και πιο σύνθετους μαθηματικούς υπολογισμούς :

Nympy (αρχείο **numpy-1.6.1-win32-superpack-python2.7.exe)**

http://sourceforge.net/projects/numpy/files/NumPy/1.6.1/

Scipy (αρχείο **scipy-0.9.0-win32-superpack-python2.7.exe**)

http://sourceforge.net/projects/scipy/files/scipy/0.9.0/

Από αυτό το σημείο και μετά,μπορούμε να τρέξουμε απλά παραδείγματα Python OpenGl,δημιουργώντας ένα πηγαίο αρχειό Python με κατάληξη "**py**",σε έναν editor για Python και στη συνέχεια γράφοντας μέσα σε αυτό,τον κώδικά μας.Υπάρχουν πολλοί editors για Python. Εδώ έχει χρησιμοποιηθεί το Eclipse INDIGO για Java,ρυθμισμένο σε λειτουργία Python.

Σημείωση : Από τη στιγμή που εγκαταστήσαμε την Python και τα απαραίτητα συνοδευτικά για την OpenGl από τη γραμμή εντολών, το compile και το run γίνονται αυτόματα εαν κάνουμε διπλό κλικ πάνω σε ένα αρχείο με κατάληξη "**py**".Έτσι το Eclipse χρησιμοποιήθηκε μόνο για τη δημιουργία του πηγαίου αρχείου. Θα μπορούσαμε επίσης να τρέξουμε μια εφαρμογή, γράφοντας τον κώδικα στην ειδική γραμμή εντολών της Python(shell).

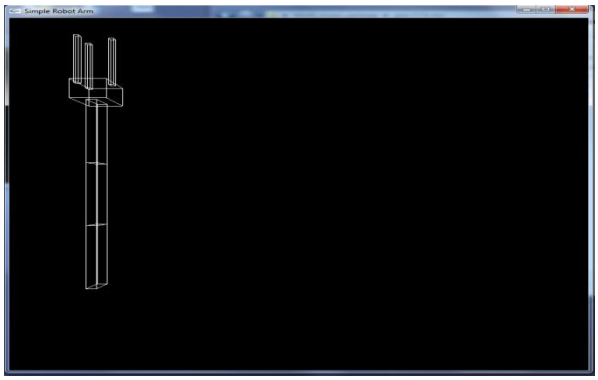
Ένα αρχείο Python,φαίνεται στην επόμενη εικόνα.

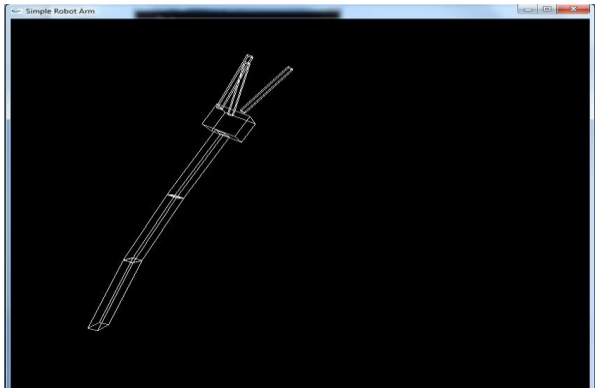
Ακολουθεί ένα παράδειγμα και συγκεκριμένα το αρχείο **RobotHand.py** ,στο οποίο γίνεται προσπάθεια δημιουργίας ενός χεριού από ανθρωποειδές ρομπότ ,όπου θα μπορούμε να κινούμε τα δάχτυλα, τον καρπό του ,ακόμα και ολόκληρο, πατώντας πλήκτρα στο πληκτρολόγιο.

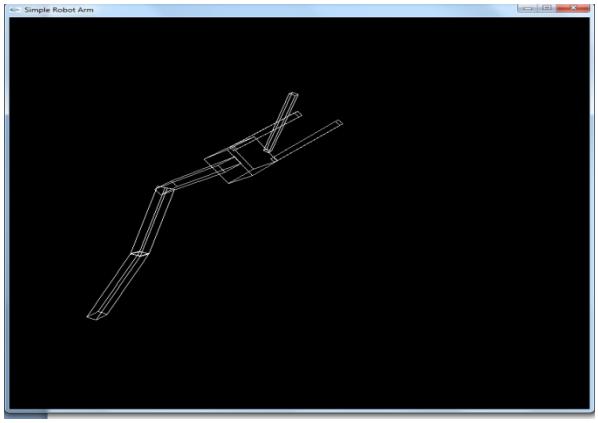
Ο κώδικας του αρχείου **RobotHand.py** είναι ο ακόλουθος :

```
from OpenGL.GLU import *
from OpenGL.GL import *
import sys

name = 'ball_glut'

def main():
    glutInit(sys.argv)
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH)
    glutInitWindowSize(400,400)
    glutCreateWindow(name)

    glClearColor(0.,0.,0.,1.)
    glShadeModel(GL_SMOOTH)
    glEnable(GL_CULL_FACE)
    glEnable(GL_DEPTH_TEST)
    glEnable(GL_LIGHTING)
    lightZeroPosition = [10.,4.,10.,1.]
    lightZeroColor = [0.8,1.0,0.8,1.0] #green tinged
    glLightfv(GL_LIGHT0, GL_POSITION, lightZeroPosition)
    glLightfv(GL_LIGHT0, GL_DIFFUSE, lightZeroColor)
    glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 0.1)
    glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 0.05)
    glEnable(GL_LIGHT0)
    glutDisplayFunc(display)
    glMatrixMode(GL_PROJECTION)
    gluPerspective(40.,1.,1.,40.)
    glMatrixMode(GL_MODELVIEW)
    gluLookAt(0,0,10,0,0,0, 0,1,0)
    glPushMatrix()
    glutMainLoop()
    return

def display():
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)
    glPushMatrix()
    color = [1.0,0.,0.,1.]
    glMaterialfv(GL_FRONT,GL_DIFFUSE,color)
    glutSolidSphere(2,20,20)
    glPopMatrix()
    glutSwapBuffers()
    return

if __name__ == '__main__': main()
```

Ο παραπάνω κώδικας υλοποιεί την εφαρμογή που φαίνεται στις επόμενες εικόνες.







6. ΒΙΒΛΙΟΓΡΑΦΙΑ-ΠΗΓΕΣ

Βιβλία

Dave Shreiner , The OpenGL®Programming Guide, Seventh Edition, July 31, 2009

Richard S. Wright, OpenGL® SuperBible: Comprehensive Tutorial and Reference (4th Edition) , June 28, 2007

Ronald Cohn Jesse Russell, Java OpenGL, January 1, 2012

Harvey M. Deitel, Python How to Program, February 14, 2002

Sites

http://www3.ntu.edu.sg/home/ehchua/programming/opengl/HowTo_VC_OpenGL.html

http://people.bath.ac.uk/abscjkw/ComputerPrograms/DevCpp/DevCppOpenGL.html

http://www.javaworld.com/javaworld/jw-02-2005/jw-0221-jogl.html?page=1

http://timelessname.com/jogl/lesson01/

http://www.python.org/doc/essays/blurb.html