ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΣΧΕΔΙΑΣΜΟΣ ΚΑΙ ΥΛΟΠΟΙΗΣΗ ΕΦΑΡΜΟΓΗΣ ΓΙΑ ΤΗΝ ΕΚΜΑΘΗΣΗ ΓΡΑΦΙΚΩΝ Η/Υ ΣΕ JAVA 2D ΚΑΙ JAVA 3D

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

της

ΜΑΚΕΛΗ ΑΙΚΑΤΕΡΙΝΗΣ

Βόλος, Ιούλιος 2013

Η σελίδα αυτή είναι σκόπιμα λευκή.

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΣΧΕΔΙΑΣΜΟΣ ΚΑΙ ΥΛΟΠΟΙΗΣΗ ΕΦΑΡΜΟΓΗΣ ΓΙΑ ΤΗΝ ΕΚΜΑΘΗΣΗ ΓΡΑΦΙΚΩΝ Η/Υ ΣΕ JAVA 2D ΚΑΙ JAVA 3D

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

της

ΜΑΚΕΛΗ ΑΙΚΑΤΕΡΙΝΗΣ

Επιβλεποντες :

ΤΣΟΜΠΑΝΟΠΟΥΛΟΥ ΠΑΝΑΓΙΩΤΑ	ΜΠΟΖΑΝΗΣ ΠΑΝΑΓΙΩΤΗΣ
ΕΠΙΚΟΥΡΟΣ ΚΑΘΗΓΗΤΡΙΑ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ	ΑΝΑΠΛΗΡΩΤΗΣ ΚΑΘΗΓΗΤΗΣ – ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

Εγκρίθηκε από την διμελή εξεταστική επιτροπή την …….

(Υπογραφή)	*(Υπογραφή)*
.....	.....
ΤΣΟΜΠΑΝΟΠΟΥΛΟΥ ΠΑΝΑΓΙΩΤΑ	ΜΠΟΖΑΝΗΣ ΠΑΝΑΓΙΩΤΗΣ
ΕΠΙΚΟΥΡΟΣ ΚΑΘΗΓΗΤΡΙΑ	ΑΝΑΠΛΗΡΩΤΗΣ ΚΑΘΗΓΗΤΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ	ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

(Υπογραφή)

.....

ΜΑΚΕΛΗ ΑΙΚΑΤΕΡΙΝΗ

Διπλωματούχος Μηχανικός Ηλεκτρονικών Υπολογιστών, Τηλεπικοινωνιών και Δικτύων του Τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Πανεπιστημίου Θεσσαλίας

Η σελίδα αυτή είναι σκόπιμα λευκή.

Στην μητέρα μου Γεωργία και στην αδερφή μου Μαρία

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

Ευχαριστίες 13

Περίληψη 14

Abstract 15

1. Εισαγωγή 16

1.1 Αντικείμενο Εργασίας 16

2. Αρχές των Γραφικών η/υ 17

2.1 Κατηγορίες και Εφαρμογές 17

2.2 Σύστημα Συντεταγμένων 19

2.3 Πλέγμα από Pixel 19

3. Περιβάλλον Εκτέλεσης – Δομή της Εφαρμογής 22

3.1 Γλώσσα Υλοποίησης και Περιβάλλον Εκτέλεσης 22

3.2 Δομή της Εφαρμογής 24

4. Εξώφυλλο 27

4.1 Το Παράθυρο 27

4.2 Το Εξώφυλλο 27

5. Java 2D – 1η Επιλογή (Point / Line / Rectangle / Round Rectangle / Ellipse) 29

5.1 Java.awt.geom Πακέτο και Κλάση Shape 29

5.2 Η Μέθοδος Paint Component() kai το Double Buffering 29

5.3 Παρουσίαση 31

5.4 Λειτουργικότητα 32

5.4.1 Point – Σημείο 32

5.4.2 Line – Γραμμή 32

5.4.3 Rectangle – Ορθογώνιο 38

5.4.4 Round Rectangle – Στρογγυλεμένο Ορθογώνιο 40

5.4.5 Ellipse – Έλλειψη 42

5.4.6 Colors – Χρώματα 44

5.4.7 Line Thickness – Πάχος Γραμμής 46

5.4.8 Antialiasing – Ομαλοποίηση 49

5.4.9 Clear and Back– Καθαρισμός και Επιστροφή 51

5.4.10 Save and Load – Αποθήκευση και Φόρτωση 53

5.4.11 Command Line – Γραμμή Εντολών 56

6. Java 2D – $2^{η}$ Επιλογή (Polygon / Polyline / Arc / Quad Curve / Cubic Curve) 59

6.1 Παρουσίαση 59
6.2 Λειτουργικότητα 59
6.2.1 Arcs – Τόξα 59
6.2.2 Polylines – Πολυγωνικές Γραμμές 63
6.2.3 Polygons – Πολύγωνα 64
6.2.4 Quadratic Curve – Τετραγωνική Καμπύλη 69
6.2.5 Cubic Curve – Κυβική Καμπύλη 73
7. Java 2D – 3η Επιλογή (Areas) 76
7.1 Παρουσίαση 76
7.2 Λειτουργικότητα 76
7.2.1 Union – Ένωση 76
7.2.2 Difference – Διαφορά 78
7.2.3 Symmetric Difference – Αποκλειστικό ή (XOR) 79
7.2.4 Intersection – Τομή 81
7.2.5 Πράξεις άλλων Σχημάτων 82
8. Java 2D – 4η Επιλογή (Geometric Transformations) 85
8.1 Παρουσίαση 85
8.2 Λειτουργικότητα 85
8.2.1 Scaling – Αλλαγή Μεγέθους Δισδιάστατου Σχήματος 85
8.2.2 Rotate – Αλλαγή Προσανατολισμού Δισδιάστατου Σχήματος 87
8.2.3 Shear – Αλλαγή Σχήματος Δισδιάστατου Σχήματος 88
8.2.4 Translate – Αλλαγή Θέσης Δισδιάστατου Σχήματος 90
8.3 Ομογενείς Συντεταγμένες 91
8.4 Σειρά Μετασχηματισμών 93
8.5 Μετατροπή Συντεταγμένων Παράθυρου σε Καρτεσιανές Συντεταγμένες 94
9. Java 2D – 5η Επιλογή (Crop and Textures) 96
9.1 Παρουσίαση 96
9.2 Λειτουργικότητα 96
9.2.1 Textures – Υφές 96
9.2.2 Load Texture – Φόρτωση Υφής 99
9.2.3 Crop – Περικοπή Εικόνας 100
9.2.4 Gradient Color – Διαβαθμισμένο Χρώμα 102
9.2.5 Lookup Operation– Lookup Λειτουργιά 105
9.2.6 Filters – Φίλτρα 107

10. Java 2D – 6η Επιλογή (Fonts ant Text) 114
10.1 Παρουσίαση 114
10.2 Λειτουργικότητα 114
10.2.1 Fonts and Text – Γραμματοσειρές και Κείμενο 114
11. Java 3D – 7η Επιλογή (Geometric Transformations) 119
11.1 Java 3D 119
11.2 Περιγραφή Τρισδιάστατου Κόσμου 119
11.3 Java 3D Concept 120
11.3.1 Scene Graph – Γράφημα Σκηνής 120
11.3.2 Bounds – Όρια 122
11.3.3 Nodes - Κόμβοι 123
11.3.4 View Model – Μοντέλο Προβολής 123
11.4 Γενική Δομή Επίλογων Java 3D 124
11.5 Παρουσίαση 126
11.6 Λειτουργικότητα 126
11.6.1 Right-Hand Coordinate System – Σύστημα Συντεταγμένων Δεξιού Χεριού 126
11.6.2 Ομογενείς Συντεταγμένες στον Τρισδιάστατο Χώρο 127
11.6.3 Αντικείμενα στην Java 3D 127
11.6.4 Scaling – Αλλαγή Μεγέθους Τρισδιάστατου Σχήματος 129
11.6.5 Rotate – Αλλαγή Προσανατολισμού Δισδιάστατου Σχήματος 130
11.6.6 Translate – Αλλαγή Θέσης Τρισδιάστατου Σχήματος 132
11.6.7 Σύνθετοι Μετασχηματισμοί 133
11.6.8 Σύστημα Αξόνων 135
11.6.9 Τρισδιάστατο Κείμενο 138
11.6.10 Το Scene graph της Επιλογής 140
11.6.11 Command Line – Γραμμή Εντολών 141
11.6.12 Clear and Back– Επαναφορά και Επιστροφή 143
12. Java 3D – 8η Επιλογή (Appearance) 145
12.1 Παρουσίαση 145
12.2 Λειτουργικότητα 145
12.2.1 Προβολές 145
12.2.2 Orbit Behavior και Επιστροφή στην Αρχική Θέση 151
12.2.3 Normal Vectors για Επιφάνειες 152
12.2.4 Ψηφιοποίηση και Polygon Attributes 154
12.2.5 Ψαλίδισμα του Όγκου 163

12.2.6 Καθορισμός Ορατών Επιφανειών 164
12.2.7 Πηγές Φωτός και Απόσβεση Φωτός 172
12.2.8 Αντανάκλαση Φωτός 175
12.2.9 Υλικό – Material 181
12.2.10 Διαφάνεια – Transparency 183
12.2.11 Το Scene graph της Επιλογής 185
2.2.12 Clear and Back– Επαναφορά και Επιστροφή 188
13. Java 3D – 9η επιλογή (Lighting and Shading) 190
13.1 Παρουσίαση 190
13.2 Λειτουργικότητα 190
13.2.1 Πηγές Φωτός στην Java 3D 190
13.2.2 Προσαρμοσμένα Υλικά – Custom Materials 195
13.2.3 Σκίαση 197
13.2.4 Σκιές 202
13.2.5 Το Scene Graph της Επιλογής 203
13.2.6 Clear and Back – Επαναφορά και Επιστροφή 204
14. Java 3D – 10η Επιλογή (Fog) 207
14.1 Παρουσίαση 207
14.2 Λειτουργικότητα 207
14.2.1Εισαγωγή Έτοιμων Γεωμετρικών αντικειμένων 207
14.2.2 Φόντο – Background 210
14.2.3 Ομίχλη - Fog 210
14.2.4 Το Scene Graph της Επιλογής 213
14.2.5 Clear and Back – Επαναφορά και Επιστροφή 215
15. Επίλογος 217
15.1 Συμπεράσματα 217
15.2 Προοπτικές Επέκτασης / Εξέλιξης 217
16. Βιβλιογραφία 218
17. Παράρτημα Α 220
17.1 Συνάρτηση draw Simple Coordinate System() 220
17.2 Συνάρτηση set To My Default Appearance() 221
18. Παράρτημα Β 222

ΕΥΧΑΡΙΣΤΙΕΣ

Φτάνοντας στο τέλος των προπτυχιακών μου σπουδών θα ήθελα να ευχαριστήσω όλους όσους στάθηκαν δίπλα μου τόσο σε ακαδημαϊκό όσο και σε προσωπικό επίπεδο.

Πάνω από όλους θα ήθελα να ευχαριστήσω την οικογένεια μου που παρά τις οποιεσδήποτε δυσκολίες πάντα βρίσκονταν στο πλευρό μου για να με στηρίξει και να με βοηθήσει με κάθε δυνατό τρόπο.

Στη συνέχεια θα ήθελα να ευχαριστήσω την επιβλέπουσα καθηγήτριά μου κ. Παναγιώτα Τσομπανοπούλου η οποία με καθοδήγησε και με βοήθησε στην περάτωση της παρούσας εργασίας. Επίσης την ευχαριστώ πολύ για τις πολύτιμες συμβουλές της, για τον χρόνο που ξόδεψε για να ακούσει τα προβλήματα μου, για την υπομονή της αλλά και την κατανόηση που έδειξε γενικά κατά την διάρκεια της συνεργασίας μας.

Νιώθω την ανάγκη να ευχαριστήσω τον κ. Παναγιώτη Μποζάνη για τις γνώσεις που μου προσέφερε όλα αυτά τα χρόνια όπως επίσης για την τιμή που μου έκανε να είναι ο δεύτερος επιβλέπων καθηγητής της διπλωματικής μου.

Επίσης ευχαριστώ όλους τους φίλους μου οι οποίοι πίστεψαν σε εμένα και με υποστήριξαν. Για τις ατελείωτες ώρες διαβάσματος που περάσαμε μαζί. Για τα άγχη που μοιραστήκαμε, για όλες τις καλές αλλά και κακές στιγμές που μας βρήκαν πάντα ενωμένους.

Θέλω ιδιαίτερα να ευχαριστήσω τον αδερφό μου Νίκο για την βοήθειά του όπως επίσης για τις παρατηρήσεις του και την επιμέλεια του κειμένου, και το φίλο μου Θανάση για την συμβολή του στην περάτωση αυτής της διπλωματικής εργασίας.

Κλείνοντας θα ήθελα να ευχαριστήσω τον Δημήτρη που πάντα ήταν δίπλα μου, που είχε πάντα εμπιστοσύνη σε εμένα και στις ικανότητές μου, που μου έδινε κουράγιο . Χωρίς εκείνον δεν θα είχα καταφέρει να ολοκληρώσω τις σπουδές μου.

Μακέλη Κατερίνα
Αθήνα Ιούλιος 2013

ΠΕΡΙΛΗΨΗ

Τα γραφικά η/υ παρέχουν στους χρήστες μεθόδους για να κατασκευάζουν εικόνες χρησιμοποιώντας τον υπολογιστή. Τα γραφικά η/υ χρησιμοποιούνται σε πλήθος εφαρμογών όπως στην κατασκευή GUI (Graphical User Interface – Γραφικό Περιβάλλον Χρήστη), στην αναπαράσταση δεδομένων, στο σχέδιο με την βοήθεια η/υ (CAD – Computer Aided Design), στην κατασκευή με την βοήθεια η/υ (CAM – Computer Aided Manufacturing), στην προσομοίωση εικόνων.

Στην παρούσα διπλωματική εργασία παρουσιάζεται μία εφαρμογή για την εκμάθηση γραφικών η/υ σε 2 και 3 διαστάσεις υλοποιημένη σε γλώσσα προγραμματισμού Java στο ολοκληρωμένο περιβάλλον εκτέλεσης (IDE – Integrated Development Environment) Netbeans . Η εφαρμογή χρησιμοποιεί τις διεπαφές προγραμματισμού εφαρμογών (API – Application Programming Interface) Java 2D και Java 3D. Η διεπαφή της εφαρμογής έχει σχεδιαστεί με απλό και κατανοητό τρόπο έτσι ώστε οποιοσδήποτε χωρίς υπόβαθρο προγραμματισμού να μπορεί να την χρησιμοποιήσει και να μάθει τις αρχές των γραφικών η/υ. Η εφαρμογή προσφέρει στον χρήστη 10 επιλογές πιο συγκεκριμένα 6 επιλογές για γραφικά σε 2 διαστάσεις και 4 επιλογές για γραφικά σε 3 διαστάσεις.

Στόχος της εργασίας αυτής είναι ο χρήστης της εφαρμογής να έρθει σε πρώτη επαφή με πλήθος μεθόδων που είναι απαραίτητες για την παραγωγή γραφικών η/υ να τις χρησιμοποιήσει μέσω της διεπαφής της εφαρμογής και να δει τα αποτελέσματά τους , χωρίς να είναι γνώστης αυτών. Μπορεί να αλλάξει παραμέτρους στις μεθόδους βλέποντας σε πραγματικό χρόνο τις αλλαγές που επιφέρουν. Τέλος κάθε φορά που χρησιμοποιείται μία μέθοδος ο χρήστης έχει την δυνατότητα να βλέπει τις εντολές που εκτελούνται, με σκοπό να μπορεί και ο ίδιος γράφοντας τον κώδικά αυτό, να αναπαράγει το αποτέλεσμα.

ABSTRACT

Computer graphics provide users with methods to generate images using a computer. Computer graphics are used in numerous applications such as GUI (graphical User Interface) , data representation, computer aided design (CAD), computer aided manufacturing (CAM) and picture simulation.

This thesis presents a tutorial for computer graphics in 2 and 3 dimensions implemented on Java programming language in the integrated execution environment (IDE) Netbeans. This tutorial uses the application programming interfaces (API) Java 2D and Java 3D. The interface is designed in a simple and understandable way so that anyone without a programming background can use it and learn the principles of computer graphics. The tutorial offers 10 options for the user, in particular 6 options for 2 dimensional graphics and 4 options for 3 dimensional graphics.

The aim of this thesis is to enable the user of this tutorial to come in first touch with a number of methods that are needed to generate computer graphics. The user can use these methods through the application interface and see the results, without originally knowing computer graphics; he can also change parameters of the methods and watch in real time the changes that occur. Eventually every time a method is used, the user has the ability to see the commands that are executed in order to write the same code so as to reproduce the result.

1. Εισαγωγη

1.1 Αντικειμενο Εργασιας

Αντικείμενο της διπλωματικής εργασίας αποτελεί η ανάπτυξη μίας εφαρμογής για την εκμάθηση γραφικών η/υ στην γλώσσα προγραμματισμού Java. Η εφαρμογή διαθέτει 10 επιλογές που περιέχουν γραφικά σε 2 (Java2D) αλλά και 3 διαστάσεις (Java 3D). Τα γραφικά υλοποιούνται με μεθόδους των οποίων ο κώδικας εμφανίζεται στην οθόνη του η/υ με σκοπό ο χρήστης να μπορέσει να προγραμματίσει δικές του εφαρμογές και να αναπαράγει τα ίδια αποτελέσματα χρησιμοποιώντας τον αναγραφόμενο κώδικα. Ο κύριος σκοπός της παρούσας εργασίας είναι ο χρήστης να μπορέσει να χρησιμοποιήσει βασικές μεθόδους των γραφικών η/υ χωρίς να κατέχει το απαιτούμενο προγραμματιστικό και μαθηματικό υπόβαθρο. Η διεπαφή της εφαρμογής είναι απλά σχεδιασμένη, με εύκολη μετάβαση ανάμεσα στις διάφορες επιλογές, απλότητα και σαφήνεια στην αλληλεπίδραση με τον χρήστη.

Αρχικά θα γίνεται μικρή μελέτη μερικών ενδιαφερόντων αλγορίθμων που χρησιμοποιούνται για την υλοποίηση των μεθόδων που περιλαμβάνει η εφαρμογή όπως επίσης και αναφορά στο μαθηματικό υπόβαθρο που απαιτείται για την κατανόηση αυτών των αλγορίθμων. Σε επόμενο στάδιο θα παρουσιάζεται ο τρόπος με τον οποίο μπορούν να υλοποιηθούν οι μέθοδοι σε προγραμματιστικό επίπεδο και θα παρουσιάζονται τα αντίστοιχα στιγμιότυπα από την εφαρμογή.

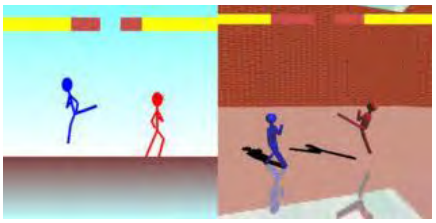
2. Αρχες των Γραφικων η/υ

2.1 Κατηγοριες και Εφαρμογες

Τα γραφικά η/υ είναι ένας κλάδος της επιστήμης των υπολογιστών που ασχολείται με τη θεωρία και την τεχνολογία σύνθεσης εικόνων σε η/υ. Για την εισαγωγή, δημιουργία και επεξεργασία των γραφικών η/υ χρησιμοποιείται ειδικό λογισμικό. Τα στοιχεία μπορούν να εισαχθούν στον η/υ μέσω διάφορων συσκευών (σαρωτής – scanner, ψηφιακή φωτογραφική μηχανή) με την μορφή κουκκίδων, γραμμών είτε μέσω του πληκτρολογίου. Αφού τα δεδομένα εμφανιστούν στην οθόνη μπορεί να γίνει επεξεργασία αυτών , που είναι αποκλειστικό αντικείμενο του λογισμικού επεξεργασίας εικόνας.

Τα γραφικά η/υ μπορούν να διακριθούν σε διάφορες κατηγορίες ανάλογα με κάποιο κριτήριο κατηγοριοποίησης. Ανάλογα με το πλήθος των διαστάσεων οι οποίες συμμετέχουν στην απεικόνιση διακρίνονται σε :

- Δισδιάστατα (2D) γραφικά η/υ
- Τρισδιάστατα (3D) γραφικά η/υ



Εικονα 1 ιδια εικονα σε 2D και 3D

Ανάλογα με τον χρόνο στον οποίο γίνεται η απόδοσή τους (Rendering) :

- Στατικά γραφικά η/υ
- Γραφικά η/υ πραγματικού χρόνου

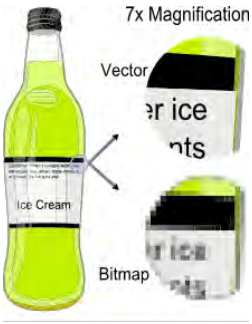
Δισδιάστατα (2D) γραφικά η/υ

Τα δισδιάστατα γραφικά η/υ αποτελούν προσπάθειες απεικόνισης γραφικών 2 διαστάσεων στην οθόνη μιας ψηφιακής συσκευής. Συνήθως τέτοιου είδους γραφικά χρησιμοποιούνται για την δημιουργία GUI αλλά και για εικονογραφήσεις βιβλίων, περιοδικών και άλλων εντύπων.

Στα γραφικά η/υ συμπεριλαμβάνεται και η επεξεργασία εικόνας (στατικής ή κινούμενης), η οποία γίνεται με τη βοήθεια ειδικού λογισμικού. Η επεξεργασία εικόνας είναι μία από τις δημοφιλέστερες χρήσεις του η/υ σήμερα, καθώς επιτρέπει τη βελτίωση μιας φωτογραφίας με την εφαρμογή ειδικών φίλτρων , τα οποία μπορούν όχι μόνο να βελτιώσουν , αλλά και να παραλλάξουν την εικόνα.

Τα 2D γραφικά μπορούν να καταταγούν σε 2 μεγάλες κατηγορίες:

- Διανυσματικά γραφικά (vector graphics) : χρησιμοποιούνται για τη δημιουργία εικόνων όπως λογότυποι, προοπτικές σχημάτων κ.α.
- Γραφικά ψηφίδων (raster graphics) : όλα τα γραφικά που δημιουργούνται από ψηφιοποίηση υπαρκτών αντικειμένων ανήκουν σε αυτή την κατηγορία



ΕΙΚΟΝΑ 2 ΔΙΑΦΟΡΑ ΑΠΟ VECTOR ΣΕ BITMAP GRAPHICS

Η βασική διαφορά των δύο κατηγοριών είναι το χαρακτηριστικό της ανάλυσης (resolution). Τα διανυσματικά γραφικά περιγράφουν μία εικόνα με τη βοήθεια της αναλυτικής γεωμετρίας και κατά συνέπεια με τη βοήθεια εξισώσεων, ενώ τα γραφικά ψηφίδων λειτουργούν όπως ακριβώς ένα ψηφιδωτό : όσο μικρότερες και περισσότερες ψηφίδες χρησιμοποιούνται , τόσο πιο ευκρινές είναι το τελικό αποτέλεσμα. Η ανάλυση μετράται σε κουκκίδες (ψηφίδες) ανά ίντσα (dots per inch - dpi). Για μία οθόνη η ανάλυση των 72 ή 96 dpi είναι επαρκής, αν όμως η εικόνα προορίζεται για επαγγελματική εκτύπωση, το ελάχιστο απαιτούμενο είναι οι 300 dpi. Τα διανυσματικά γραφικά είναι ανεξάρτητα ανάλυσης (resolution free), γιατί δεν χρησιμοποιούν ψηφίδες για τα σχηματισμό της εικόνας.

Ένα ακόμα χαρακτηριστικό των γραφικών είναι το βάθος χρώματος, δηλαδή το πλήθος των δυαδικών ψηφίων (bits) που χρησιμοποιούνται για την περιγραφή χρώματος κάθε ψηφίδας (ή κάθε περιοχής στα διανυσματικά γραφικά). Το σημερινό στάνταρ είναι για τις οθόνες βάθος χρώματος 24 bits ενώ για τις εκτυπώσεις 32 bits.

Για τις εφαρμογές του διαδικτύου χρησιμοποιούνται συνήθως γραφικά ψηφίδων, γιατί τα διανυσματικά γραφικά δεν υποστηρίζονται από παλαιότερες εκδόσεις φυλλομετρητών (browsers) που χρησιμοποιούνται ακόμα από σχετικά μεγάλο ποσοστό των χρηστών του διαδικτύου.

Ο τύπος των γραφικών αναγνωρίζεται συνήθως από την επέκταση του ονόματος του αρχείου, στο οποίο είναι αποθηκευμένα. Οι πλέον συνήθεις τύποι είναι :

Διανυσματικά γραφικά : .svg, .cdr, .ai

Γραφικά ψηφίδων : .tif, .bmp, .jpg, .gif, .png (οι 3 τελευταίοι τύποι είναι κατάλληλοι για το διαδίκτυο)

Τρισδιάστατα (3D) γραφικά η/υ

Τα τρισδιάστατα γραφικά η/υ αποτελούν προσπάθειες απεικόνισης γραφικών τριών διαστάσεων στην οθόνη μιας ψηφιακής συσκευής. Το γεγονός ότι η απεικόνιση χρησιμοποιεί 3 διαστάσεις τα καθιστά ιδιαίτερα ρεαλιστικά. Τέτοιου είδους γραφικά χρησιμοποιούνται συνήθως από προγράμματα όπως τα παιχνίδια η/υ και οι εικονικοί κόσμοι. Τα τρισδιάστατα γραφικά βρίσκουν επίσης εφαρμογή στον κινηματογράφο, για τη δημιουργία εικονικών κόσμων αλλά και ειδικών εφέ, που είναι αδύνατο να γυριστούν ως πραγματικές σκηνές.

Στατικά Γραφικά η/υ

Τα στατικά γραφικά η/υ αποτελούν αντικείμενα γραφικών τα οποία δεν αποδίδονται την στιγμή που εκτελούνται αλλά έχουν αποδοθεί μία φορά κατά τη δημιουργία τους. Παράδειγμα τέτοιων

γραφικών είναι τα μικρά βίντεο, τα οποία εμφανίζονται σε διάφορα παιχνίδια και έχουν γυριστεί μία φορά και κάθε φορά που θα τα παρακολουθήσουμε παραμένουν ίδια. Για την δημιουργία τους χρησιμοποιείται κάποιο πρόγραμμα δημιουργίας γραφικών και κίνησης (animation).

Γραφικά η/υ πραγματικού χρόνου

Τα γραφικά υπολογιστών πραγματικού χρόνου αποτελούν αντικείμενα γραφικών τα οποία αποδίδονται την στιγμή που εκτελούνται. Για παράδειγμα τα γραφικά που εμφανίζονται στην οθόνη ενός υπολογιστή , ο οποίος εκτελεί ένα παιχνίδι, ανήκουν συνήθως σε αυτή την κατηγορία. Για τη δημιουργία τους απαιτείται κάποια μηχανή απόδοσης γραφικών (graphics rendering engine) πραγματικού χρόνου.

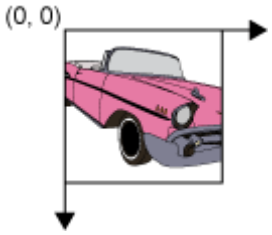
Εφαρμογές των γραφικών η/υ

Όπως έχει ήδη αναφερθεί τα γραφικά η/υ μπορούν να χρησιμοποιηθούν στην κατασκευή GUI, στα παιχνίδια η/υ, στην δημιουργία εικονικών κόσμων, στην εικονογράφηση εντύπων, στην αναπαράσταση δεδομένων, στην αρχιτεκτονική σχεδίαση, στην υπολογιστική βιολογία και φυσική, στις προσομοιώσεις σε η/υ, στις ψηφιακές τέχνες, στην εκπαίδευση, στον σχεδιασμό ιστοσελίδων, στην γραφιστική κ.τ.λ. .

2.2 Σύστημα Συντεταγμένων

Το Java 2D API διατηρεί δύο χώρους συντεταγμένων τον χώρο του χρήστη και τον χώρο της συσκευής. Ο χώρος του χρήστη είναι ανεξάρτητος της συσκευής και είναι ένα λογικό σύστημα συντεταγμένων. Οι εφαρμογές χρησιμοποιούν αυτό το σύστημα συντεταγμένων αποκλειστικά. Όλα τα σχήματα που δημιουργούνται μέσα από Java 2D μεθόδους είναι ορισμένα σε αυτό το σύστημα. Ο χώρος της συσκευής εξαρτάται από την συσκευή και ποικίλει ανάλογα με το που θα αποδοθούν (rendering) τα σχήματα.

Η αρχή του χώρου του χρήστη είναι πάνω αριστερά, τα x αυξάνονται προς τα δεξιά και τα y προς τα κάτω.

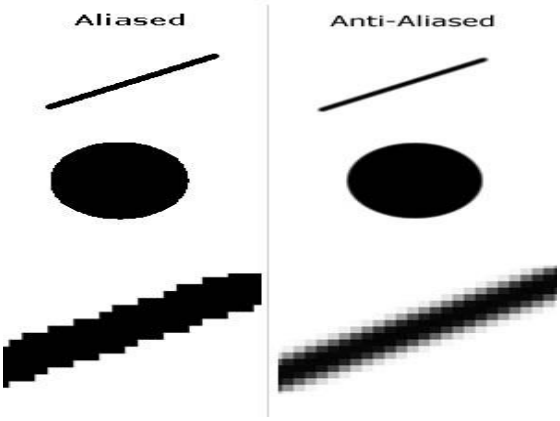


Εικονα 3 αρχη του συστηματος συντεταγμενων του χρηστη

Ο χώρος του χρήστη είναι μία ομοιόμορφη αφαίρεση όλων των δυνατών χώρων συσκευής. Αυτόματα οι συντεταγμένες του χώρου του χρήστη μετατρέπονται σε συντεταγμένες του χώρου συσκευής όταν κάποιο αντικείμενο αποδίδεται στην οθόνη.

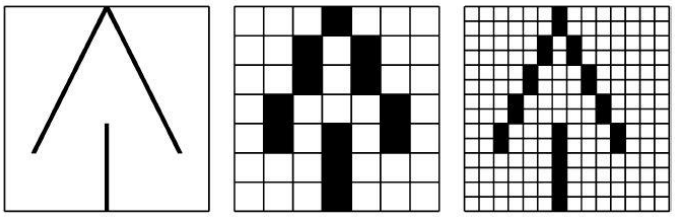
2.3 Πλεγμα απο Pixel

Όπως ήδη αναφέρθηκε [2.1] τα δισδιάστατα γραφικά χωρίζονται σε δύο κατηγορίες, στα διανυσματικά γραφικά και στα γραφικά ψηφίδων. Οι οθόνες των η/υ όπως επίσης οι εκτυπωτές κτλ βασίζουν την απεικόνιση των γραφικών στα γραφικά ψηφίδων ή αλλιώς raster oriented graphics / pixel oriented graphics. Τα raster graphics χρησιμοποιούν έναν πλέγμα από pixel συγκεκριμένου μεγέθους. Μπορεί ένα pixel να βαφτεί με ένα χρώμα, στην απλούστερη περίπτωση θεωρούμε διαθέσιμα χρώματα το άσπρο και το μαύρο. Για να ζωγραφίσουμε σχήματα τα οποία έχουν δημιουργηθεί με διανύσματα πάνω στο πλέγμα από pixel πρέπει αυτά τα σχήματα να μετατραπούν σε pixels, η διαδικασία αυτή ονομάζεται scan conversion (μετατροπή μέσω σάρωσης) . Αυτό μπορεί να οδηγήσει σε πολύ μεγάλο υπολογιστικό κόστος. Μία standard οθόνη έχει πάνω από 1.000.000 pixels και για καθένα από αυτά πρέπει να αποφασιστεί τι χρώμα θα βαφτεί για το κάθε σχήμα που θέλουμε να αποδώσουμε. Κατά τη διάρκεια της διαδικασίας αυτής μπορούν να δημιουργηθούν προβλήματα στην ομαλοποίηση (aliasing - antialiasing) των σχημάτων. Για παράδειγμα οδοντωτές ακμές (stair casing).



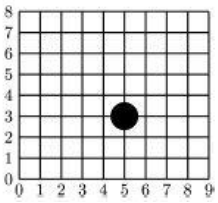
Εικονα 4 προβλημα aliasing

Τα pixel oriented graphics υπόκεινται σε μία συγκεκριμένη ανάλυση (resolution). Όταν έχει αποφασιστεί η ανάλυση για τα γραφικά που θα αποδοθούν τότε όλα τα σχήματα υπόκεινται στην συγκεκριμένη ανάλυση και δεν μπορούμε να ανακτήσουμε τα αρχικά διανυσματικά σχήματα με αποτέλεσμα να υπάρχουν πάρα πολλά μειονεκτήματα αν αυτά τα σχήματα αποδοθούν σε οθόνη με διαφορετική ανάλυση, μεγιστοποιηθούν ή ελαχιστοποιηθούν.



Εικονα 5 δυο διαφορετικες αναλυσεις για το βελος της αριστερης εικονας

Στην εργασία αυτή όταν αναφερόμαστε σε pixel θα θεωρούμε το τετράγωνο μεταξύ των γραμμών του πλέγματος όπως στην εικόνα 6.



Εικονα 6 pixel

3. Περιβαλλον Εκτελεσης – Δομη της Εφαρμογης

3.1 Γλωσσα Υλοποιησης και Περιβαλλον Εκτελεσης

Η υλοποίηση της εφαρμογής έχει γίνει στην γλώσσα προγραμματισμού Java. Η έκδοση που έχει χρησιμοποιηθεί είναι η Java 7. Η Java είναι αντικειμενοστραφής γλώσσα προγραμματισμού και χαρακτηριστικό της είναι η ανεξαρτησία λειτουργικού συστήματος και πλατφόρμας. Τα προγράμματα που είναι γραμμένα σε Java τρέχουν ακριβώς το ίδιο σε όλα τα λειτουργικά συστήματα χωρίς να χρειαστεί να γίνει ξανά μεταγλώττιση (compile) ή να αλλάξει ο πηγαίος κώδικας.

Εικονα 7 Java icon

Για να γράψει κάποιος κώδικα Java δεν χρειάζεται τίποτα άλλο παρά έναν επεξεργαστή κειμένου παρόλα αυτά ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) είναι πολύ χρήσιμο ιδιαιτέρως στον εντοπισμό σφαλμάτων. Η εφαρμογή αυτή έχει υλοποιηθεί στο IDE Netbeans 7.3 . Το Netbeans είναι ένα περιβάλλον εκτέλεσης για τον προγραμματισμό κυρίως σε Java αλλά και σε άλλες γλώσσες προγραμματισμού όπως PHP, C/C++ και HTML. Είναι επίσης ένα πλαίσιο για πλατφόρμες εφαρμογών για Java desktop και άλλα. Το Netbeans έχει γραφεί σε Java και μπορεί να τρέξει σε Windows, OS X, Linux, Solaris και σε άλλες πλατφόρμες που μπορούν να υποστηρίξουν συμβατό JVM (Java Virtual Machine – Εικονική Μηχανή Java).

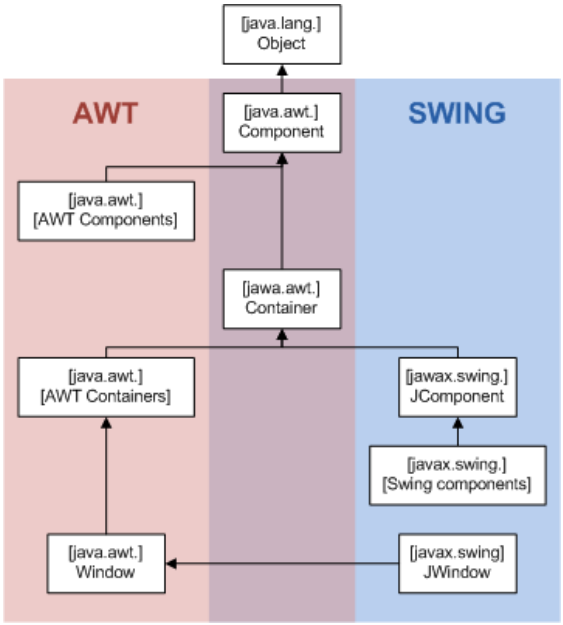
Εικονα 8 Netbeans icon

Στην συγκεκριμένη εφαρμογή χρησιμοποιήθηκαν οι εργαλειοθήκες AWT και Swing καθώς επίσης οι διεπαφές προγραμματισμού Java 2D και Java 3D. Οι AWT, Swing και Java 2D απαρτίζουν τις Java Foundation Classes (θεμελιώδεις κλάσεις της Java) της Oracle.

Η εργαλειοθήκη AWT (Abstract Window Toolkit) είναι η αρχική εργαλειοθήκη γραφικών στοιχείων της Java. Είναι η βασική διεπαφή προγραμματισμού για να παρέχει ένα GUI για προγράμματα σε Java, είναι επίσης η μόνη εργαλειοθήκη για GUI για έναν μεγάλο αριθμό από Java ME (Micro Edition) προφίλ όπως το CDC (Connected Device Configuration) προφίλ για τα κινητά τηλέφωνα. Η AWT περιέχει την διεπαφή μεταξύ του συστήματος παραθύρων και της εφαρμογής, τα GUI events (γεγονότα), τους διαχειριστές διάταξης (layout managers), την διεπαφή για την αλληλεπίδραση με το ποντίκι και το πληκτρολόγιο, ένα πακέτο για χρήση μαζί με το Clipboard και το Drag and Drop, ένα βασικό σύνολο GUI components (στοιχείων) όπως κουμπιά, κουτιά κειμένου και μενού. Επίσης έχει την δυνατότητα να παρέχει κάποιες υψηλού επιπέδου λειτουργίες όπως πρόσβαση στο system tray (taskbar) σε συστήματα που το υποστηρίζουν καθώς και έναρξη κάποιων

desktop εφαρμογών όπως φυλλομετρητές (browsers) και email clients.

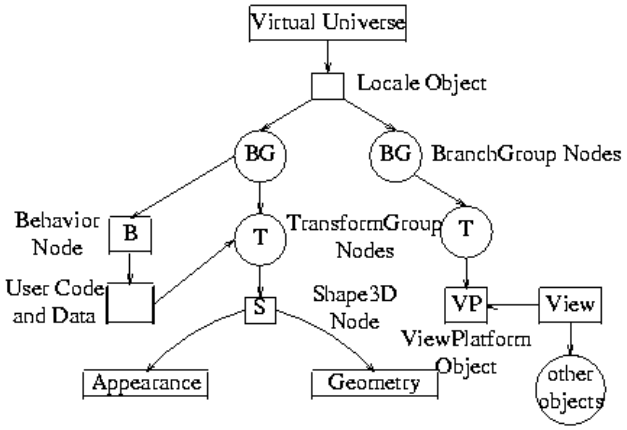
Η εργαλειοθήκη Swing είναι η κύρια εργαλειοθήκη γραφικών στοιχείων της Java. Η εργαλειοθήκη Swing σχεδιάστηκε για να παρέχει ένα πιο εξελιγμένο σύνολο GUI στοιχείων από την AWT. Η εργαλειοθήκη Swing παρέχει μία φυσική εμφάνιση και αίσθηση (look and feel) που μιμείται αυτή των διάφορων πλατφόρμων, έχει πιο ισχυρά και ευέλικτα components από την AWT. Εκτός από τα γνωστά components παρέχει πολλά προηγμένα components όπως κουμπιά, check boxes, ετικέτες, tabbed panel, scroll panes, δέντρα, πίνακες και λίστες.



ΕΙΚΟΝΑ 9 AWT ΚΑΙ SWING ΙΕΡΑΡΧΙΑ ΚΛΑΣΕΩΝ

Η διεπαφή προγραμματισμού Java 2D χρησιμοποιείται για τον σχεδιασμό γραφικών 2 διαστάσεων στην γλώσσα προγραμματισμού Java. Κάθε Java 2D λειτουργία σχεδίασης μπορεί τελικά να αντιμετωπιστεί ως γέμισμα ενός σχήματος με χρήση ενός χρώματος και ως σύνθεσης του αποτελέσματος στην οθόνη. Οι κλάσεις της διεπαφής Java 2D είναι οργανωμένες ως ακολούθως : στο κύριο πακέτο AWT, στην βασική βιβλιοθήκη της Java για σχήματα 2 διαστάσεων , στην βιβλιοθήκη για χειρισμό ανάγλυφων, στην βιβλιοθήκη για τον χειρισμό των χρωμάτων, στην βιβλιοθήκη για τις γραφικές εικόνες και στην βιβλιοθήκη για την εκτύπωση.

Η διεπαφή προγραμματισμού Java 3D βασίζεται σε ένα γράφημα σκηνής (scene graph). Τρέχει πάνω από OpenGL ή Direct3D. Το scene graph είναι ένα κατευθυνόμενο μη κυκλικό γράφημα . Η Java 3D είναι μία διεπαφή που συμπυκνώνει τον προγραμματισμό γραφικών με τη χρήση μιας πραγματικής αντικειμενοστραφούς προσέγγισης. Μία σκηνή κατασκευάζεται χρησιμοποιώντας ένα γράφημα σκηνής που είναι μία αναπαράσταση των αντικειμένων που πρέπει να εμφανιστούν. Αυτό το γράφημα σκηνής είναι δομημένο ως ένα δέντρο που περιέχει πολλά στοιχεία απαραίτητα για την εμφάνιση των αντικειμένων. Επιπλέον η Java 3D υποστηρίζει ήχο.

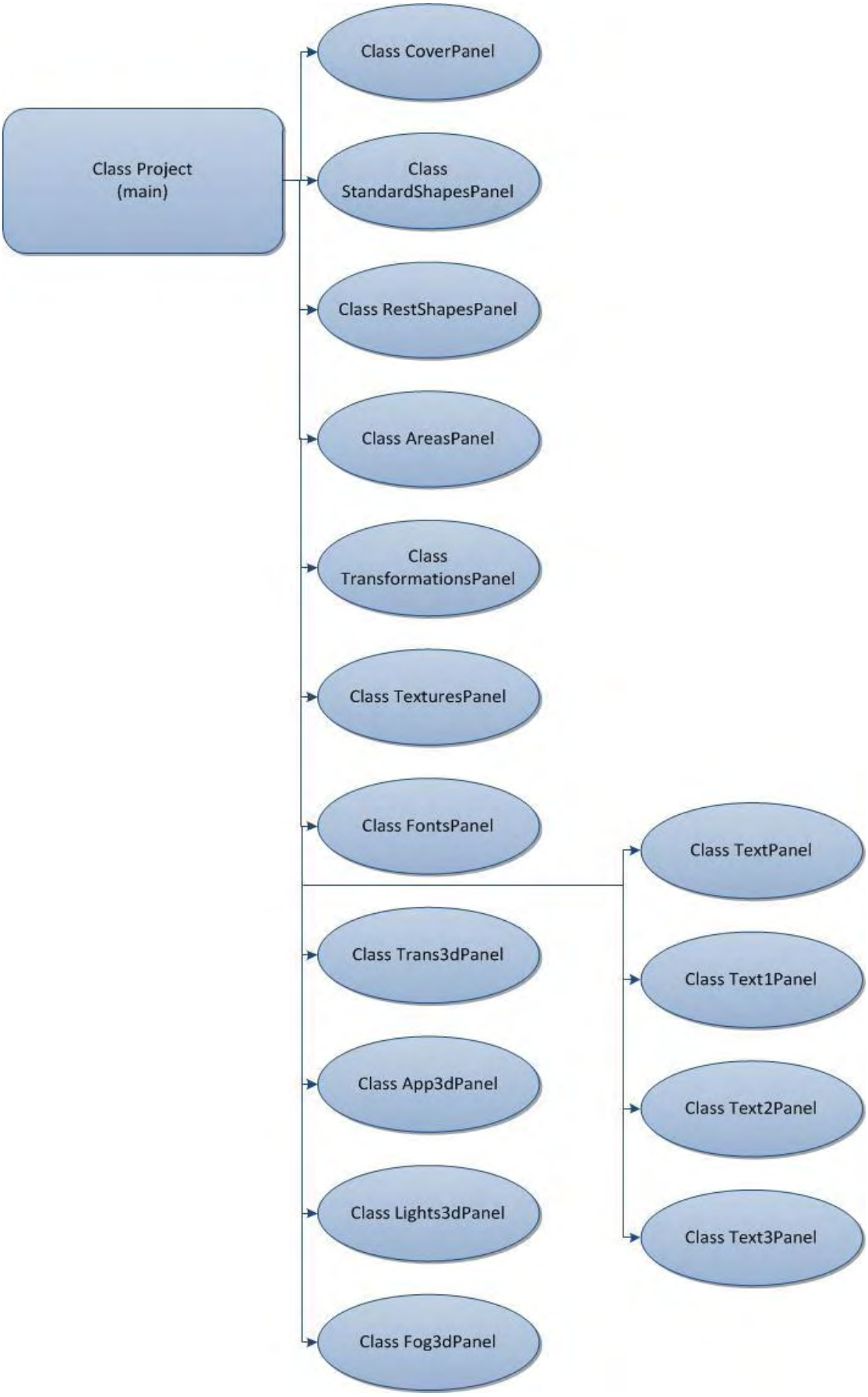


ΕΙΚΟΝΑ 10 JAVA 3D SCENE GRAPH

3.2 Δομη της Εφαρμογης

Η εφαρμογή αποτελείται από την βασική κλάση Project η οποία είναι παραγόμενη κλάση της κλάσης JFrame . Αυτό σημαίνει ότι κληρονομεί όλες τις μεθόδους της JFrame. Στην συγκεκριμένη περίπτωση επιλέχθηκε η κλάση JFrame για να δημιουργήσει το παράθυρο της εφαρμογής μιας που ένα αντικείμενο JFrame περιλαμβάνει ένα περίγραμμα και τα συνήθη κουμπιά ελαχιστοποίησης, αλλαγής μεγέθους και κλεισίματος παραθύρου. Η κλάση Project περιέχει 15 εσωτερικές κλάσεις. Οι κλάσεις που περιέχει είναι οι :

- CoverPanel παραγόμενη κλάση της κλάσης JPanel
- StandardShapesPanel παραγόμενη κλάση της κλάσης JPanel που υλοποιεί επίσης τα interfaces (διεπαφές) MouseListener και MouseMotionListener
- RestShapesPanel παραγόμενη κλάση της κλάσης JPanel που υλοποιεί επίσης τα interfaces MouseListener και MouseMotionListener
- AreasPanel παραγόμενη κλάση της κλάσης JPanel
- TransformationsPanel παραγόμενη κλάση της κλάσης JPanel
- TexturesPanel παραγόμενη κλάση της κλάσης JPanel που υλοποιεί επίσης τα interfaces MouseListener και MouseMotionListener
- FontsPanel παραγόμενη κλάση της κλάσης JPanel
- Trans3dPanel παραγόμενη κλάση της κλάσης JPanel
- App3dPanel παραγόμενη κλάση της κλάσης JPanel
- Lights2dPanel παραγόμενη κλάση της κλάσης JPanel
- Fog3dPanel παραγόμενη κλάση της κλάσης JPanel
- TextPanel παραγόμενη κλάση της κλάσης JPanel
- Text1Panel παραγόμενη κλάση της κλάσης JPanel
- Text2Panel παραγόμενη κλάση της κλάσης JPanel
- Text3Panel παραγόμενη κλάση της κλάσης JPanel



ΕΙΚΟΝΑ 11 ΣΧΗΜΑ ΚΛΑΣΕΩΝ

Όλες οι εσωτερικές κλάσεις είναι παραγόμενες της κλάσης JPanel γιατί αποτελούν τα containers μέσα στα οποία έχουν προστεθεί πολλά components για τη δημιουργία των επιλογών της εφαρμογής υποδιαιρώντας το JFrame. Επιλέχθηκε να δημιουργηθούν εσωτερικές κλάσεις γιατί σκοπός ήταν η βασική κλάση να είναι αυτοπεριεχόμενη και επίσης ήταν πιο εύκολο στον προγραμματισμό γιατί οι μέθοδοι τους έχουν πρόσβαση στις instance variables της άλλης κλάσης.

Το σχήμα που έχει υλοποιηθεί, σύμφωνα με τα παραπάνω, αποτελείται από το κύριο παράθυρο που δημιουργείται με το αντικείμενο της κλάσης Project που περιέχει το JPanel που δημιουργείται με το αντικείμενο της κλάσης CoverPanel που περιέχει το «εξώφυλλο» της εφαρμογής με τα κουμπιά που σε μεταφέρουν στις διαθέσιμες επιλογές. Έπειτα δημιουργούνται 10 αντικείμενα, 1 από την κάθε μία από τις ακόλουθες κλάσεις : StandardShapesPanel, RestShapesPanel, AreasPanel, TransformationsPanel, TexturesPanel, FontsPanel, Trans3dPanel, App3dPanel, Lights2dPanel και Fog3dPanel. Το καθένα από τα παραπάνω JPanel είναι τα containers που περιέχουν τις επιλογές της εφαρμογής. Επίσης δημιουργούνται 4 ακόμα αντικείμενα των κλάσεων : TextPanel, Text1Panel, Text2Panel, Text3Panel που είναι μέρος των επιλογών για την Java 3D και είναι τα JPanels που περιέχουν τις εντολές του κώδικα. Η δημιουργία των 4 τελευταίων panel ήταν απαραίτητη λόγω της ιδιαίτερης διαδικασίας παραγωγής των 3D γραφικών μέσω του API Java 3D αλλά και για την ομοιομορφία της εφαρμογής.

Κάποιες από τις εσωτερικές κλάσεις υλοποιούν interfaces με σκοπό να μπορούν να χειριστούν γεγονότα (events). Αυτό γίνεται γιατί τα GUI που υλοποιούνται με Swing χρησιμοποιούν μία μορφή του αντικειμενοστραφούς προγραμματισμού που καλείται προγραμματισμός με γεγονότα (event-driven programming) . Ένα γεγονός είναι ένα αντικείμενο που ενεργεί ως σήμα σε ένα άλλο αντικείμενο το οποίο καλείται ακουστής (listener). Η αποστολή ενός event καλείται ενεργοποίηση του event (firing the event). Το αντικείμενο που ενεργοποιεί το event είναι συχνά ένα GUI component π.χ. ένα κουμπί που έχει γίνει clicked. Ένα αντικείμενο listener εκτελεί κάποια ενέργεια σε απάντηση στο event. Στην εφαρμογή αυτή τα events που θέλουμε να «ακουστούν» έχουν να κάνουν με το ποντίκι όπως αν το κουμπί του ποντικιού έκανε click, αν είναι πατημένο, αν σταμάτησε να είναι πατημένο καθώς και με την κίνηση του ποντικιού.

Method	Description
mousePressed	This event is called when the mouse button is pressed and held.
mouseReleased	This event is called when a mouse button that is being held is released.
mouseClicked	This event is called when a mouse button is pressed then released.
mouseEntered	This method is called when the mouse pointer enters the bounds of a particular component.
mouseExited	This method is called when the mouse pointer exits the bounds of a particular component.

Εικονα 12 Mouse Listener interface

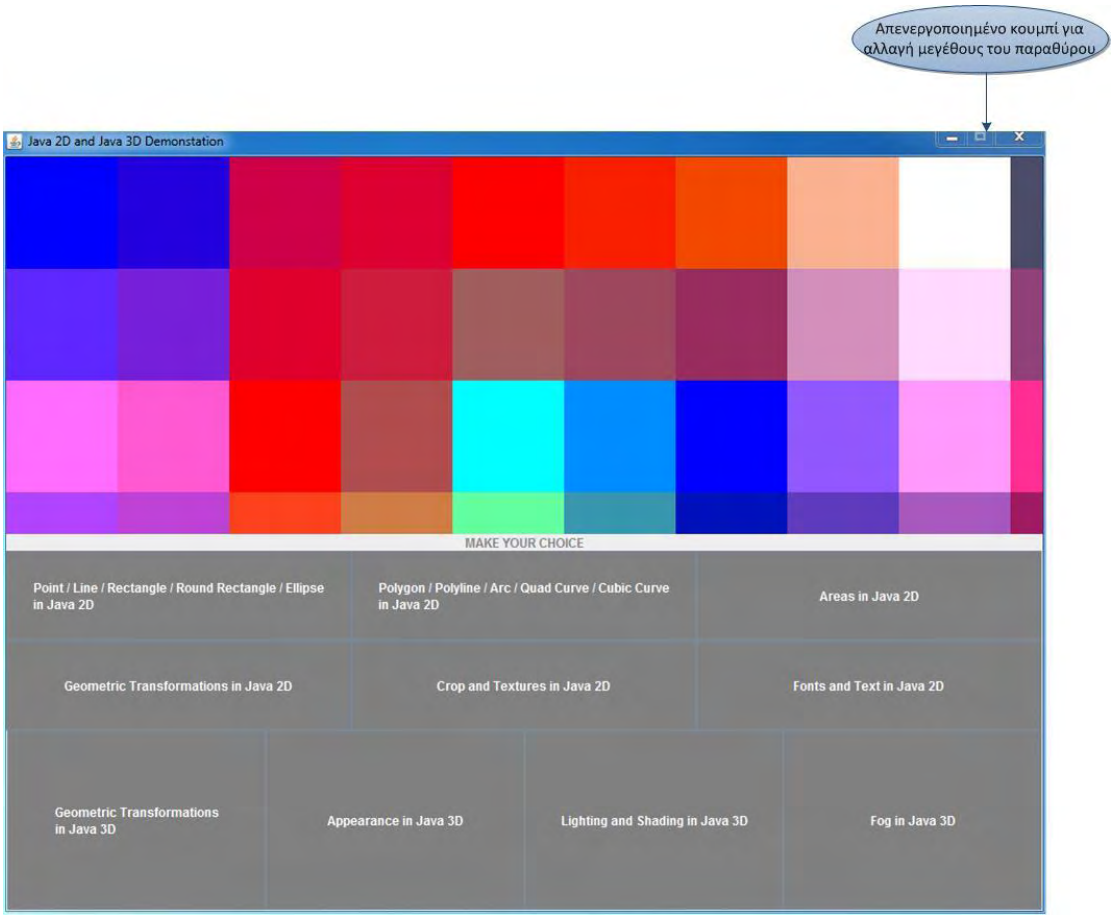
Method	Description
mouseMoved	This is triggered any time the mouse is moved within the bounds of any given applet/application
mouseDragged	This is triggered when the mouse is moved while a button is held down,

Εικονα 13 Mouse Motion Listener interface

4. Εξωφυλλο

4.1 Το Παραθυρο

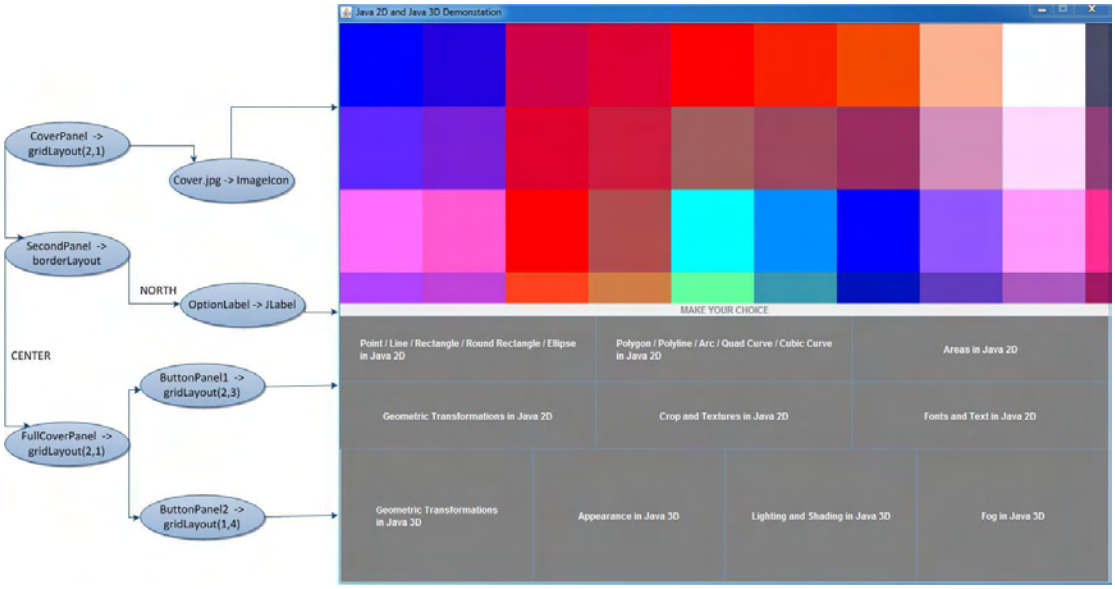
Το παράθυρο της εφαρμογής έχει διαστάσεις 1000x750 pixels (πλάτος x μήκος). Αυτό σημαίνει ότι το σημείο (0, 0) βρίσκεται στην πάνω αριστερά γωνία και το παράθυρο εκτείνεται 1000 pixels προς τα δεξιά και 750 pixels προς τα κάτω. Ο διαθέσιμος χώρος για να αποδοθούν τα σχήματα είναι λίγο μικρότερος από τις διαστάσεις του παραθύρου λόγω της μπάρας που περιέχει τα 3 βασικά κουμπιά (ελαχιστοποίηση, αλλαγή μεγέθους και κλείσιμο παραθύρου) και λόγω του περιγράμματος του παραθύρου. Έχει απενεργοποιηθεί η λειτουργία αλλαγής μεγέθους γιατί πολλά σχήματα ζωγραφίζονται σε ορισμένες θέσεις και υπάρχει πρόβλημα με την ανάλυσή των σχημάτων.



Εικονα 14 το εξωφυλλο της εφαρμογης

4.2 Το Εξωφυλλο

Το εξώφυλλο της εφαρμογής είναι αντικείμενο της κλάσης CoverPanel . Αποτελείται από μία σύνθετη διάταξη (layout). Η δομή του βασίζεται στην μείξη grid Layout και border Layout.



ΕΙΚΟΝΑ 15 CUSTOM LAYOUT

Το CoverPanel παρουσιάζει τις διαθέσιμες επιλογές στον χρήστη. Ο χρήστης μπορεί να μεταβεί από την μία επιλογή στην άλλη μέσω αυτού του panel. Κάθε φορά που ο χρήστης βρίσκεται στο CoverPanel αφού είχε επιλέξει κάποια επιλογή οτιδήποτε είχε ζωγραφιστεί σε αυτή την επιλογή σβήνεται. Επίσης με την βοήθεια του tool Tip Text (μέθοδος που παρουσιάζει βοηθητικό κείμενο όταν το βέλος του ποντικιού δείξει πάνω σε ένα component) γίνεται μία σύντομη περιγραφή του περιεχομένου κάθε επιλογή.

5. Java 2D – 1η Επιλογη (Point / Line / Rectangle / Round Rectangle / Ellipse)

5.1 Java.awt.geom Πακετο και Κλαση Shape

Το Java 2D API διαθέτει πολλές κλάσεις οι οποίες ορίζουν βασικά γεωμετρικά σχήματα όπως σημεία, γραμμές και ορθογώνια. Αυτές οι κλάσεις είναι μέρος του πακέτου java.awt.geom.

Η κλάση Shape διαθέτει διάφορες υποκλάσεις που μας επιτρέπουν την κατασκευή πολλών δισδιάστατων γεωμετρικών σχημάτων. Οι συντεταγμένες των σχημάτων αυτών δίνονται σε double ή float γιατί τα αντικείμενα της Shape κλάσης είναι διανυσματικά γραφικά (vector graphics). Αφού οριστεί ένα αντικείμενο για να εμφανιστεί στην οθόνη του η/υ πρέπει είτε να γίνει draw είτε fill. Η μέθοδος draw σχεδιάζει το περίγραμμα των αντίστοιχων σχημάτων ενώ η μέθοδος fill ζωγραφίζει όλη την περιοχή που περικλείει το σχήμα.

5.2 Η Μεθοδος Paint Component() και το Double Buffering

Τα AWT components περιέχουν μία μέθοδο paint() η οποία παίρνει ως όρισμα ένα αντικείμενο τύπου Graphics. Η paint() χρησιμοποιείται για να εμφανίσει τα components που έχουν σχεδιαστεί στην οθόνη του η/υ. Η μέθοδος αυτή είναι ήδη ορισμένη και καλείται αυτόματα όταν το σχήμα απεικονίζεται στην οθόνη. Για να ορίσουμε νέα σχήματα πρέπει να ορίσουμε ξανά την paint() και να περικλείουμε την δήλωση super.paint() μέσα στην μέθοδο. Κάθε component που μπορεί να σχεδιαστεί στην οθόνη έχει συσχετισμένο ένα αντικείμενο τύπου Graphics. Αυτό το αντικείμενο έχει δεδομένα που καθορίζουν ποια περιοχή της οθόνης καλύπτεται από το component. Π.χ. το αντικείμενο για ένα JFrame καθορίζει ότι η σχεδίαση γίνεται μέσα στα όρια του αντικειμένου JFrame. Έτσι το καλούν αντικείμενο τύπου Graphics έστω g όταν ενεργοποιείται η μέθοδος paint() αντικαθίσταται από το αντικείμενο Graphics που είναι συσχετισμένο με το JFrame επομένως τα σχήματα σχεδιάζονται εντός του JFrame.

Στην εφαρμογή αυτή ζωγραφίζουμε πάνω σε JComponents και όχι σε components, πιο συγκεκριμένα σε JPanels . Για να ζωγραφίσουμε σε JComponents χρησιμοποιούμαι διαφορετική μέθοδο σχεδίασης την paint Component(). Οι λεπτομέρειες για την paint Component() είναι ίδιες με αυτές για την paint(). Ζωγραφίζουμε πάνω στα ξεχωριστά panels και έπειτα τα τοποθετούμε όλα στο ίδιο JFrame που είναι το παράθυρο της εφαρμογής.

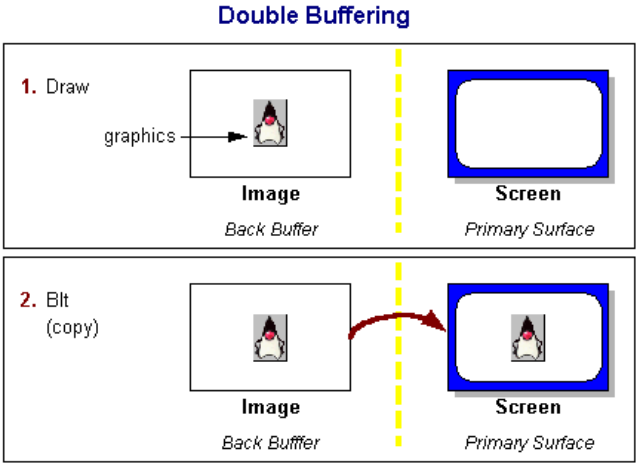
Στις πρώτες 6 επιλογές που διαθέτει η εφαρμογή παρουσιάζονται μέθοδοι της Java 2D. Για να σχεδιαστεί ένα αντικείμενο χρησιμοποιώντας τις δυνατότητες της Java 2D θα πρέπει το αντικείμενο αυτό να είναι τύπου Graphics2D. Εφόσον η paint Component() παίρνει ως όρισμα αντικείμενο τύπου Graphics πρέπει να γίνει casting του αντικειμένου από Graphics σε Graphics2D.

Για να αλλάξουμε τα γραφικά περιεχόμενα του παραθύρου καλούμε την μέθοδο repaint(). Η μέθοδος repaint() κάνει κάποια προεργασία και κατόπιν ενεργοποιεί τη μέθοδο paint Component() η οποία σχεδιάζει ξανά την οθόνη. Η repaint() είναι ήδη ορισμένη αλλά πρέπει να ενεργοποιείται ρητά εν αντιθέσει με την paint Component() που πρέπει να ορίζεται αλλά δεν ενεργοποιείται ρητά.

Οι πρώτες 6 επιλογές της εφαρμογής χωρίζονται σε 2 κατηγορίες : σε αυτές που δημιουργούν σχήματα σύμφωνα με την αλληλεπίδραση με τον χρήστη (μέσω του ποντικιού) και σε αυτές που ζωγραφίζουν σχήματα σε προκαθορισμένες θέσεις. Στην πρώτη κατηγορία ανήκουν οι κλάσεις : StandardShapesPanel, RestShapesPanel, TexturesPanel ενώ στην δεύτερη κατηγορία οι : AreasPanel, TransformationsPanel, FontsPanel. Και στις 2 κατηγορίες χρησιμοποιείται η τεχνική του

Double Buffering. Ζωγραφίζονται σχήματα σε μία εικόνα , αντικείμενο της κλάσης buffered Image, η οποία εικόνα βρίσκεται στην μνήμη του η/υ και έπειτα η εικόνα αυτή αντιγράφεται στην οθόνη του η/υ. Αυτό γίνεται γιατί αν ζωγραφιστούν σχήματα με αρκετά στοιχεία για παράδειγμα γραμμές κατευθείαν στην οθόνη του η/υ είναι εύκολο να παρατηρήσει κανείς ότι παίρνει αρκετό χρόνο η απόδοσή τους, αν όμως ζωγραφιστούν τα ίδια σχήματα σε μία buffered Image και έπειτα αντιγραφεί η εικόνα αυτή στην οθόνη του η/υ τότε η δημιουργία σχημάτων γίνεται πολύ πιο γρήγορα. Στις πρώτες 6 επιλογές της εφαρμογής ζωγραφίζεται συνεχώς μία buffered Image που ανανεώνεται και αυτή είναι που αντιγράφεται στην οθόνη του η/υ και συνεπώς αυτή που βλέπουμε.

Εκτός από την ταχύτητα δημιουργίας των σχημάτων οι buffered Images είναι απαραίτητες για την δημιουργία των σχημάτων με την βοήθεια του ποντικιού. Για παράδειγμα στη δημιουργία μίας γραμμής το σημείο που ξεκινά η γραμμή βρίσκεται εκεί που κάνει click το ποντίκι για πρώτη φορά και έπειτα η γραμμή φαίνεται να σχηματίζεται ανάλογα με την κατεύθυνση του ποντικιού και όσο κρατείται πατημένο το κουμπί του ποντικιού. Όταν το κουμπί σταματήσει να είναι πατημένο η γραμμή εμφανίζεται σύμφωνα με την τελική της θέση χωρίς να είναι ορατές οι γραμμές που δημιουργήθηκαν κατά την διάρκεια αλλαγής των κατευθύνσεων του ποντικιού. Αυτό είναι δυνατό με την βοήθεια των buffered Images. Οι μη τελικές γραμμές ζωγραφίζονται κατευθείαν στην οθόνη του η/υ δημιουργώντας το εφέ με την κίνηση του ποντικιού και όταν το κουμπί σταματήσει να είναι πατημένο η τελική γραμμή ζωγραφίζεται σε μία buffered Image η οποία κατευθείαν αντιγράφεται στην οθόνη του η/υ «καλύπτοντας» τις υπόλοιπες γραμμές. Στο τέλος η μόνη ορατή γραμμή είναι η τελική γραμμή που βρίσκεται στην buffered Image. Με αυτή την τεχνική έχουν δημιουργηθεί όλα τα σχήματα που αλληλεπιδρούν δυναμικά με το ποντίκι.



ΕΙΚΟΝΑ 16 DOUBLE BUFFERING

Για την buffered image γράφουμε :

BufferedImage in_bi=(BufferedImage)this.createImage(this.getWidth(),this.getHeight())

Έτσι δημιουργείται η buffered image η οποία έχει διαστάσεις όσο και το panel στο οποίο βρίσκεται. Έπειτα δημιουργείται ένα αντικείμενο Graphics2D για την buffered image

Graphics2D g2bi = in_bi.createGraphics()

Χρησιμοποιούμε αυτό το αντικείμενο με τις μεθόδους draw, fill για να σχεδιάσουμε σχήματα πάνω στην buffered image και έπειτα για να κάνουμε ορατά αυτά τα σχήματα πρέπει να μεταφέρουμε / αντιγράψουμε την buffered image στην οθόνη του η/υ. Για να το κάνουμε αυτό γράφουμε :

g2.drawImage(in_bi, x , y , null)

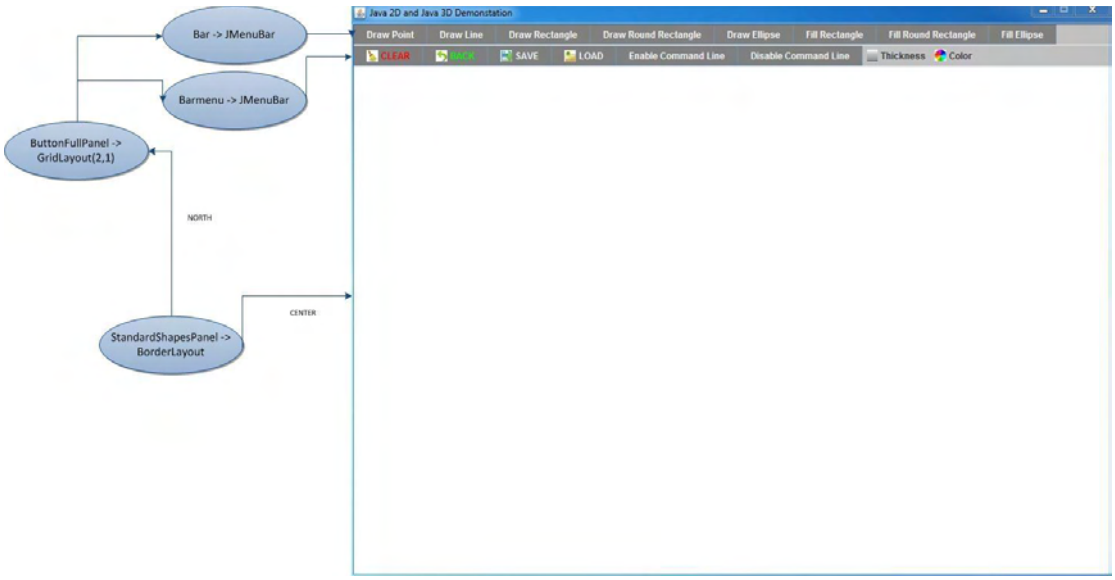
όπου g2 το αντικείμενο που παίρνει ως όρισμα η μέθοδος paint Component(). In_bi είναι η buffered image που έχουμε δημιουργήσει η οποία θα ζωγραφιστεί στην οθόνη στο ορθογώνιο που αντιστοιχεί η πάνω αριστερή γωνία του στις συντεταγμένες (x , y) και η κάτω δεξιά του γωνία στις συντεταγμένες (x + this.getWidth(), y + this.getHeight()) .

Θα πρέπει να τονίσουμε ότι τα σχήματα που αποδίδονται δεν ζωγραφίζονται μέσα στην μέθοδο paint Component() αλλά σε μία άλλη μέθοδο που καλείται μέσω της paint Component() . Αυτό γίνεται γιατί σε όλες τις επιλογές της εφαρμογής δημιουργούνται σχήματα που αλλάζουν συχνά με αποτέλεσμα αν δημιουργούνταν (ορίζονταν και τοποθετούνταν στον χώρο) μέσα στην paint Component() θα ήταν εύκολο να υπάρξουν φαινόμενα flickering (να τρεμοπαίζει η οθόνη) και αργής απόκρισης του παραθύρου. Η Java δίνει προτεραιότητα στην μέθοδο paint Component() με αποτέλεσμα όταν πυροδοτηθούν άλλα γεγονότα όπως για παράδειγμα το κλείσιμο του παραθύρου να μην μπορεί να πραγματοποιηθούν άμεσα.

5.3 Παρουσιαση

Η πρώτη επιλογή της εφαρμογής είναι το αντικείμενο της κλάσης StandradShapesPanel. Είναι ένα panel που περιέχει διάφορα Swing components όπως κουμπιά και drop down menus όπως επίσης και μία περιοχή σχεδίασης μέσα στην οποία ο χρήστης μπορεί να σχεδιάσει σημεία, γραμμές, ορθογώνια, στρογγυλεμένα ορθογώνια και ελλείψεις. Ο χρήστης μπορεί να σχεδιάσει το περίγραμμα των παραπάνω σχημάτων ή να τα γεμίσει με χρώμα. Ο χρήστης έχει ακόμη τη δυνατότητα να αλλάξει το μέγεθος της γραμμής, να αποθηκεύσει τη ζωγραφιά του, να φορτώσει μία άλλη ζωγραφιά και ύστερα να σχεδιάσει πάνω της νέα σχήματα. Μπορεί να καθαρίσει την περιοχή σχεδίασης. Τέλος σε ένα κίτρινο πλαίσιο εμφανίζονται οι εντολές που δημιουργούν τα σχήματα. Το πλαίσιο αυτό , σε αυτήν την εργασία, ονομάζεται command line και μπορεί να γίνει ορατό και μη ανάλογα με την επιλογή του χρήστη.

Το StandardShapesPanel έχει custom δομή που δημιουργείται με border Layout και grid Layout.



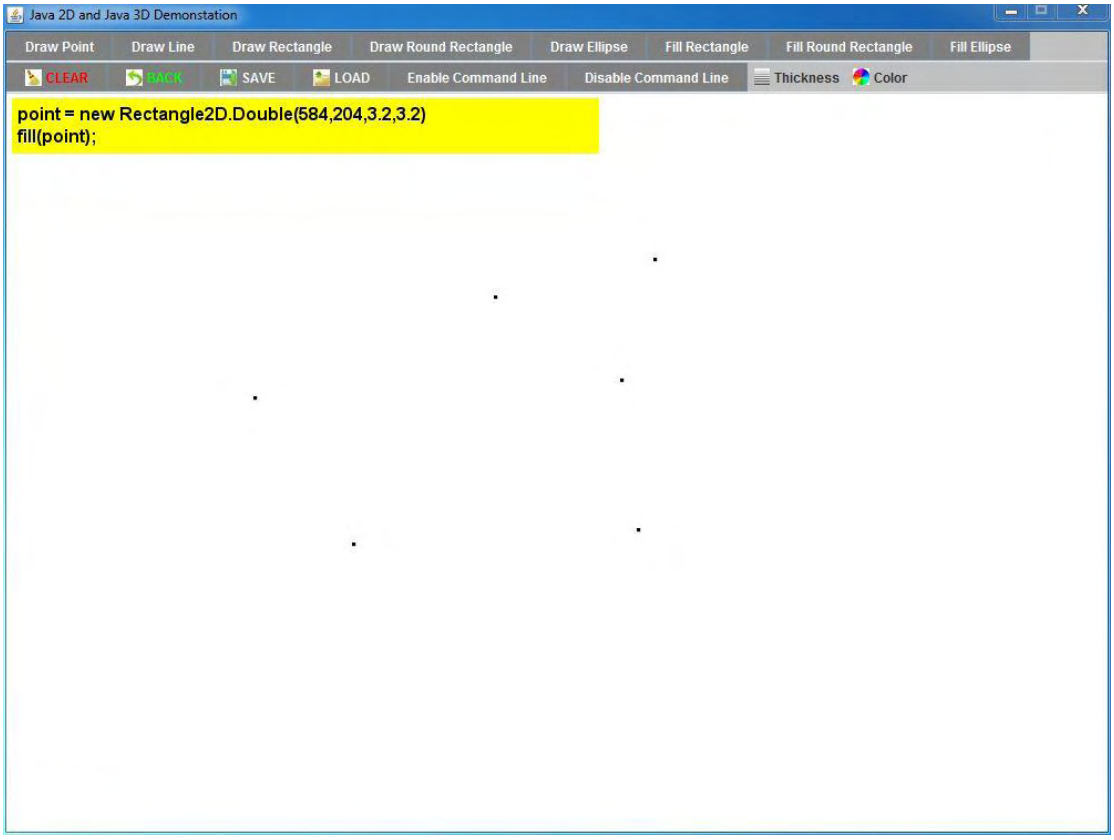
Εικονα 17 StandardShapesPanel layout

5.4 Λειτουργικοτητα

5.4.1 Point – Σημειο

Η αφηρημένη κλάση Point2D δεν είναι υποκλάση της κλάσης Shape και για αυτό το λόγο ένα σημείο δεν μπορεί να δημιουργηθεί απευθείας από την κλάση Point2D. Αντικείμενα της κλάσης Point2D χρησιμοποιούνται για να ορίζουν συντεταγμένες για άλλα γεωμετρικά σχήματα και όχι για να ζωγραφίζονται σημεία στην οθόνη του η/υ. Η κλάση Point2D είναι βασική κλάση για τις κλάσεις Point2D.Double και Point2D.Float οι οποίες περιγράφουν την ακρίβεια των παραμέτρων τους, αν θα είναι float ή double. Για να ζωγραφίσει κανείς ένα σημείο στην οθόνη του η/υ μπορεί να ζωγραφίσει μία γραμμή που να ξεκινά και να τελειώνει στις ίδιες συντεταγμένες ή ένα ορθογώνιο με ίδιο μήκος και πλάτος ίσα με ένα pixel.

Στην εφαρμογή αυτή έχει επιλεχθεί ο δεύτερος τρόπος για να ζωγραφίσουμε σημεία δηλαδή ζωγραφίζοντας ορθογώνια. Ο κώδικας με τον οποίο υλοποιείται το ορθογώνιο θα περιγραφεί σε επόμενη παράγραφο [5.4.3]. Για να δημιουργήσουμε σημεία πατάμε το κουμπί «Draw Point» και απλά κάνουμε click στην περιοχή σχεδίασης.



Εικονα 18 δημιουργια σημειων

5.4.2 Line – Γραμμη

Αλγόριθμος σχεδίασης γραμμής – DDA (Ψηφιακός Διαφορικός Αναλυτής)

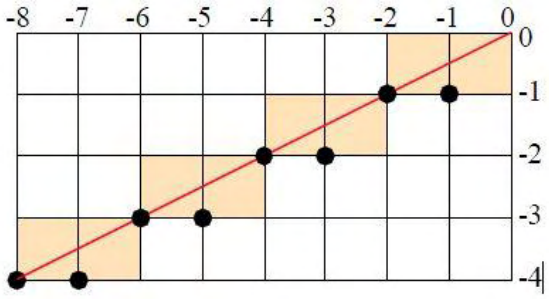
Σκοπός είναι η σχεδίαση μίας γραμμής από τις συντεταγμένες (x_L, y_L) στις συντεταγμένες (x_H , y_H) όπου το x_L< x_H και m= $\frac{y_H - y_L}{x_H - x_L}$ η κλίση 0<m≤1. Ο αλγόριθμος προχωράει κατά τον άξονα των x, καθορίζει τι συμβαίνει με τα y αποφασίζοντας έτσι ποιο pixel θα ζωγραφίσει.

1. (x_0, y_0) = (x_L, y_L)
2. For (i=0 ; i<= x_H-x_L ; i++) // για όλα τα διαστήματα μεταξύ του x_L και x_H
3. DrawPixel(x_i, Round(y_i)) // ζωγράφισε το κοντινότερο pixel κοντά στο y_i
4. x_{i+1} = x_i+1 // ανανέωσε το x
5. y_{i+1} = m x_{i+1} + b // ανανέωσε το y

Η γραμμή 5 μπορεί να γραφεί ως y_{i+1} = m (x_i+1) + b λόγω της γραμμής 4. Αν κάνουμε την πράξη έχουμε y_{i+1} = m x_i+ m + b . Όπου y_i = m x_i+ b άρα η γραμμή 5 γίνεται y_{i+1} = y_i + m .
Συνεπώς η βελτιωμένη έκδοση του κώδικα είναι :

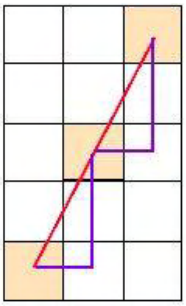
1. (x_0, y_0) = (x_L, y_L)
2. For (i=0 ; i<= x_H-x_L ; i++) // για όλα τα διαστήματα μεταξύ του x_L και x_H
3. DrawPixel(x_i, Round(y_i)) // ζωγράφισε το κοντινότερο pixel κοντά στο y_i
4. x_{i+1} = x_i+1 // ανανέωσε το x
5. y_{i+1} = y_i + m // ανανέωσε το y

Χρειάζεται να υπολογίσουμε το m μόνο μία φορά. Ο DDA χρησιμοποιεί συνεχείς προσθέσεις και στρογγυλοποιήσεις. Είναι σχετικά αργός γιατί περιέχει πράξεις κινητής υποδιαστολής.

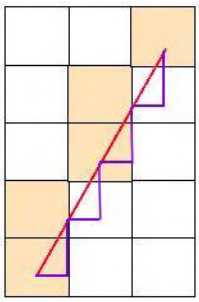


Εικονα 19 DDA για 0<m≤1

Όλες οι υπόλοιπες κλίσεις αντιμετωπίζονται με συμμετρία. Στον DDA παρατηρούνται κενά στην δημιουργία της γραμμής για m>1 , όμως το αποτέλεσμα βελτιώνεται αν αλλάξουν θέση τα x με τα y χρησιμοποιώντας στην συνθήκη για αύξηση του x 1/m αντί για m.



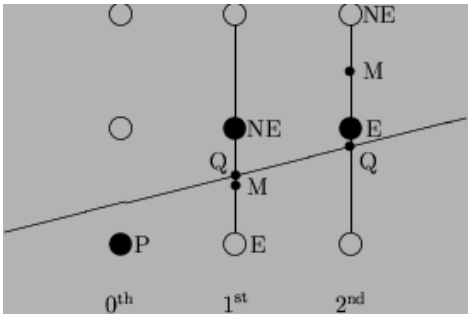
Εικονα 20 DDA m>1



ΕΙΚΟΝΑ 21 DDA ΜΕ 1/Μ ΑΝΤΙ ΓΙΑ Μ ΟΤΑΝ ΑΥΞΑΝΕΤΑΙ ΤΟ Χ

Αλγόριθμος σχεδίασης γραμμής – Midpoint Algorithm (Αλγόριθμος μέσου σημείου)

Σκοπός είναι ,όπως και πριν, η σχεδίαση μίας γραμμής από τις συντεταγμένες (x_L, y_L) στις συντεταγμένες (x_H , y_H) όπου το x_L< x_H και m= $\frac{y_H - y_L}{x_H - x_L}$ η κλίση 0<m≤1. Η λογική του αλγορίθμου βασίζεται στην επιλογή του επόμενου pixel από αυτό που βρισκόμαστε σχετικά με το αν η γραμμή περνάει πάνω ή κάτω από το (x+1 , y+1/2). Αν βρίσκεται πάνω επιλέγουμε προς σχεδίαση το ΒΑ (βορειοανατολικό – NE) pixel αλλιώς επιλέγουμε το Α (ανατολικό – Ε).



ΕΙΚΟΝΑ 22 Ε ΚΑΙ NE PIXEL

Πώς καταλαβαίνουμε ότι ένα σημείο είναι πάνω ή κάτω από τη γραμμή?
Έχουμε την εξίσωση της γραμμής y = mx + b . Αντικαθιστούμε το m= $\frac{y_H - y_L}{x_H - x_L}$ και η εξίσωση γίνεται y = $\frac{y_H - y_L}{x_H - x_L}$ x + b. Πολλαπλασιάζουμε και τα δύο μέλη με τον όρο $x_H - x_L$ και έχουμε $(x_H - x_L)$ y = ($\frac{y_H - y_L}{x_H - x_L}$ x + b) $(x_H - x_L)$. Κάνοντας τις απλοποιήσεις και μετακινώντας τους όρους στο αριστερό μέλος έχουμε την εξίσωση $(x_H - x_L)$ y + $(y_L - y_H)$ x + $(x_L - x_H)$ b = 0 (εξίσωση Α). Μία γραμμή μπορεί να γραφεί στην έμμεση μορφή της (implicit form)ως f(x , y) = Cx + Dy + E . Μπορούμε οπότε να πούμε ότι η εξίσωση Α είναι της μορφής f(x , y) = Cx + Dy + E με :

- C = $y_L - y_H$
- D = $x_H - x_L$
- E = $(x_L - x_H)$ b

Η f(x , y) = Cx + Dy + E έχει τις ιδιότητες :

- Αν το (x , y) βρίσκεται πάνω στη γραμμή τότε το f(x , y) = 0
- Αν το (x , y) βρίσκεται κάτω από τη γραμμή τότε το f(x , y) < 0
- Αν το (x , y) βρίσκεται πάνω από τη γραμμή τότε το f(x , y) > 0

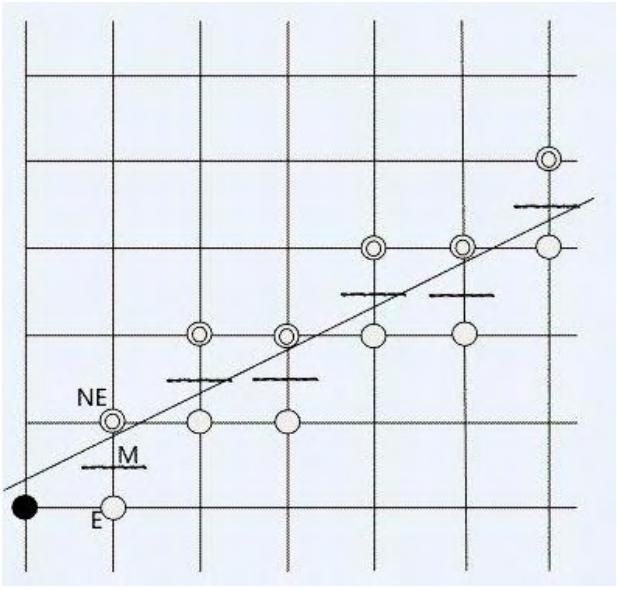
Άρα και η εξίσωση Α εφόσον μπορεί να γραφεί σύμφωνα με την implicit form έχει και τις παραπάνω ιδιότητες.

Ποιο είναι κάθε φορά το μέσο?
Το αρχικό σημείο είναι το (x_L, y_L). Το πρώτο μέσο που πρέπει να υπολογίσουμε ε το f(x_L + 1, $y_L + 1/2$).

f(x_L + 1, $y_L + 1/2$) = C (x_L + 1) +D ($y_L + 1/2$) + E (εξίσωση Β) , εφόσον f(x_L, y_L) = Cx_L + Dy_L + E η εξίσωση Β μπορεί να γίνει f(x_L + 1, $y_L + 1/2$) = f(x_L, y_L) + C + 1/2D . Όμως το f(x_L, y_L) βρίσκεται πάνω στην γραμμή εφόσον είναι το αρχικό μας σημείο οπότε το f(x_L, y_L) = 0. Άρα f(x_L + 1, $y_L + 1/2$) = C + 1/2D ή f(x_L + 1, $y_L + 1/2$) = 2C + D.

Για την σταδιακή κατασκευή του αλγορίθμου πρέπει να υπολογίσουμε :

- Το f(x+2 , y + 1/2) αν έχει επιλεγεί το E pixel (γιατί επιλέχθηκε το (x+1,y))
  f(x+2 , y + 1/2) = 2C(x+2) + 2D(y+1/2) +2E
  f(x+2 , y + 1/2) = 2C + f(x+1,y+1/2) // το f(x+1,y+1/2) έχει ήδη υπολογιστεί
- Το f(x+2 , y + 3/2) αν έχει επιλεγεί το NE pixel (γιατί επιλέχθηκε το (x+1,y+1))
  f(x+2 , y + 3/2) = 2C (x+2) + 2D(y+3/2) +2E
  f(x+2 , y + 3/2) = 2C +2D + f(x+1,y+1/2) // το f(x+1,y+1/2) έχει ήδη υπολογιστεί



Εικονα 23 midpoint αλγοριθμος

1. x=x_L
2. y=y_L
3. D = $x_H - x_L$
4. C = $y_L - y_H$
5. SUM = 2C + D // μεταβλητή απόφασης
6. Draw(x , y)
7. While (x < x_H) // για όλα τα σημεία μέχρι το x_H
8. If (SUM < 0) // αν το ενδιάμεσο σημείο είναι κάτω από τη γραμμή επέλεξε το (x+1 , y+1)
9. SUM += 2D
10. y++
11. x++ // αλλιώς επέλεξε το (x+1 , y)

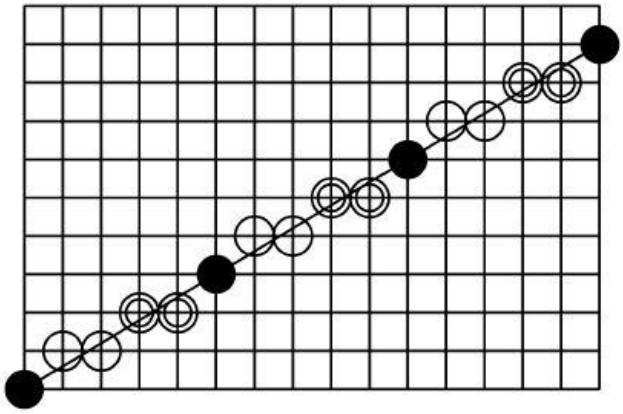
12. SUM += 2C // η f στο επόμενο σημείο
13. Draw(x , y)

Ο αλγόριθμος αυτός χρησιμοποιεί πράξεις μεταξύ ακεραίων και είναι πολύ πιο γρήγορος από τον DDA. Επίσης μπορεί να επεκταθεί και σε άλλα σχήματα όπως κύκλος, έλλειψη.

Όπως και στον DDA οι γραμμές με διαφορετικές κλίσεις αντιμετωπίζονται συμμετρικά. Για παράδειγμα για γραμμές με κλίση -1<m<0 αντί να επιλέξουμε μεταξύ του E και NE pixel επιλέγουμε μεταξύ του S (south – νότιο) και SE (south east – νοτιοανατολικό) pixel. Επίσης για m>1 αλλάζουν θέση τα x με τα y , το y αυξάνεται κατά 1 σε κάθε επανάληψη και αποφασίζεται ποιο x θα επιλεγεί σύμφωνα με την μεταβλητή απόφασης SUM.

Αλγόριθμος σχεδίασης γραμμής - Brons' Structural Algorithm (Διαρθρωτικός Αλγόριθμος)

Οι structural αλγόριθμοι προσπαθούν να μειώσουν το υπολογιστικό κόστος των προηγούμενων αλγορίθμων προσπαθώντας να αναλύσουν την γραμμή που πρέπει να σχεδιαστεί σε επαναλαμβανόμενα patterns (μοτίβα).



ΕΙΚΟΝΑ 24 STRUCTURAL ΑΛΓΟΡΙΘΜΟΣ

Όπως φαίνεται στην εικόνα 24 το μοτίβο απαρτίζεται από το pixel που έχει μαρκαριστεί με ένα μαύρο κύκλο, από τα 2 συνεχόμενα pixels με κύκλο με μαύρο περίγραμμα και από το 2 επόμενα pixels με 2 ομόκεντρους μαύρους κύκλους. Έστω D (diagonal – διαγώνιο / βορειοανατολικό) pixel και H (horizontal – οριζόντιο / ανατολικό) pixel. Έτσι η γραμμή μπορεί να εκφραστεί ως επανάληψη του μοτίβου DHDHD. Οι γραμμές όμως μπορεί να περιλαμβάνουν διαφορετικά μοτίβα με διαφορετικό ρυθμό επανάληψης.

Υποθέτοντας ότι τα δύο άκρα της γραμμής είναι τα (x_0, y_0) και (x_1, y_1) οι αριθμοί x_0, y_0 , x_1, y_1 πρέπει να είναι ακέραιοι όπως και οι τιμές dx = $x_1 - x_0$ και
dy = $y_1 - y_0$. Η κλίση είναι ρητός αριθμός. Οι τιμές του y είναι ίσες με $\frac{d_x}{d_y}$ x + b . Με το b ρητό αριθμό και τις τιμές του x ακεραίους τα y πρέπει να είναι στρογγυλοποιημένα. Ένας πεπερασμένος αριθμός από διαφορετικά υπόλοιπα είναι δυνατά για τους υπολογισμούς του y. Έτσι οποιαδήποτε γραμμή πάνω σε πλέγμα από pixels πρέπει να βασίζεται σε επαναλαμβανόμενο μοτίβο. Το μοτίβο αυτό μπορεί να είναι μεγάλο και στην χειρότερη περίπτωση μπορεί να αρχίζει στην αρχή της γραμμής και να τελειώνει στο τέλος της.

Έστω ότι θέλουμε να πετύχουμε τη σχεδίαση μίας γραμμής από τις συντεταγμένες (x_0, y_0) στις συντεταγμένες (x_1 , y_1) όπου το $x_0 < x_1$ και m= $\frac{y_1 - y_0}{x_1 - x_0}$ η κλίση 0<m≤1. Υπολογίζουμε τις τιμές dx =

$x_1 - x_0$ και dy = $y_1 - y_0$. Εκτός από το αρχικό pixel άλλα dx pixels πρέπει να σχεδιαστούν. Για αυτά τα dx pixels απαιτούνται dy διαγώνια βήματα. Τα υπόλοιπα dx –dy πρέπει να είναι οριζόντια βήματα. Πρέπει να βρούμε τη σωστή ακολουθία οριζοντίων και διαγώνιων βημάτων. Η ακολουθία $H^{dx-dy}D^{dy}$ περιέχει τον σωστό αριθμό από οριζόντια και διαγώνια βήματα αλλά πιθανών όχι με τη σωστή σειρά. Η ακολουθία αυτή χρησιμοποιείται σαν αρχική ακολουθία και με μεταθέσεις φτάνουμε στην τελική ακολουθία.
Αλγόριθμος Bron :

1. Αρχική ακολουθία $H^{dx-dy}D^{dy}$
2. Αν ο μέγιστος κοινός διαιρέτης των dx, dy είναι μεγαλύτερος του 1 (g = gcd(dx,dy)) τότε η γραμμή μπορεί να ζωγραφιστεί από g επαναλήψεις μίας ακολουθίας μήκους dx/g
3. Επομένως μπορεί να θεωρηθεί ότι τα dx, dy δεν έχουν κοινό διαιρέτη.
4. Έστω P, Q δύο ακολουθίες του αλφαβήτου {D,H}
5. Ξεκινώντας από μία ακολουθία P^pQ^q με συχνότητες p και q που δεν έχουν κοινό διαιρέτη και θεωρώντας 1< q < p η ακέραια διαίρεση p = k q + r με
   0< r < q μας οδηγεί στην αντιμετατιθεμένη ακολουθία
   - Αν (q - r) > r , $(P^kQ)^{q-r}(P^{k+1}Q)^r$
   - Αν (q - r) < r , $(P^{k+1}Q)^r(P^kQ)^{q-r}$
6. Εφαρμόζουμε την ίδια διαδικασία επαναληπτικά στις υποακολουθίες μήκους r και q-r μέχρι r = 1 ή q-r =1 .

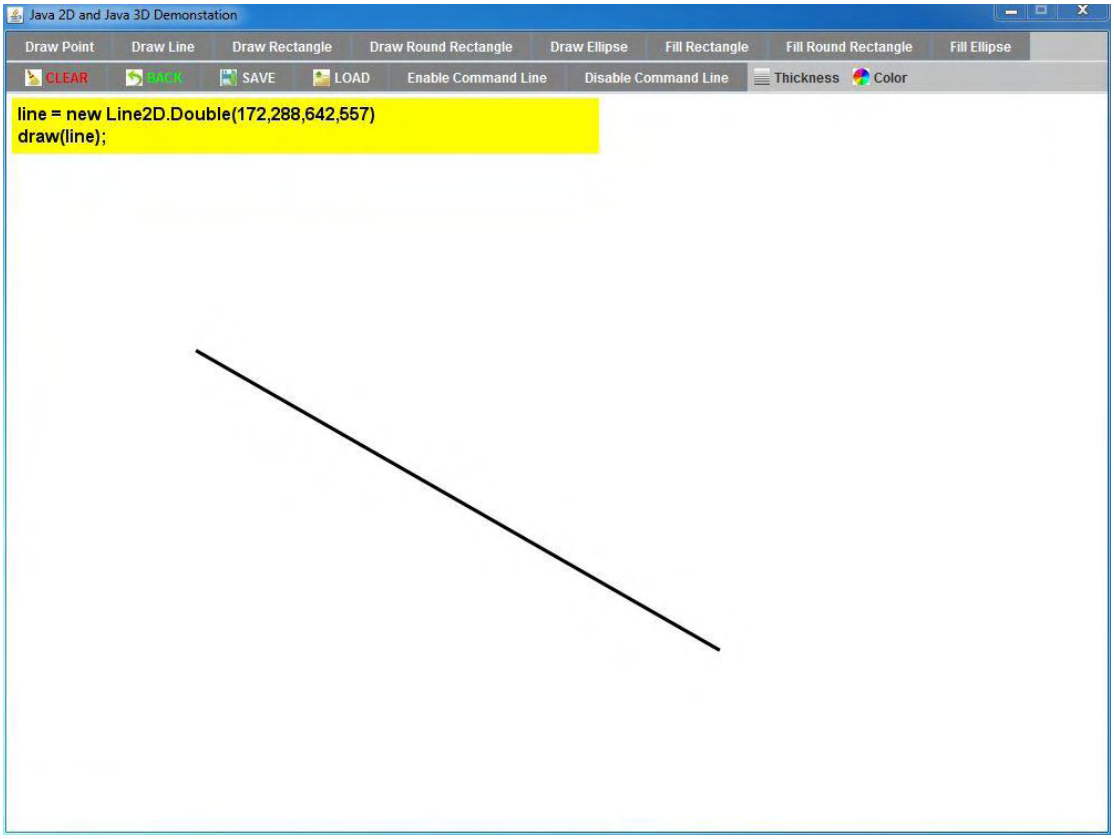
Όπως στον midpoint και στον DDA οι γραμμές με διαφορετικές κλίσεις αντιμετωπίζονται συμμετρικά.

Η αφηρημένη κλάση Line2D είναι υποκλάση της κλάσης Shape και έχει 2 παραγόμενες κλάσεις τις Line2D.Double, Line2D.Float. Για να ορίσουμε μία γραμμή χρησιμοποιούμε την εντολή :

Line2D.Double line = new Line2D.Double(x1,y1,x2,y2)

Όπου δημιουργείτε ένα ευθύγραμμο τμήμα, αντικείμενο της κλάσης Line2D.Double , με αρχή στις συντεταγμένες (x1 , y1) και τέλος στις συντεταγμένες (x2 , y2). Για να σχεδιαστεί στην οθόνη η γραμμή γράφουμε :

draw(line)



ΕΙΚΟΝΑ 25 ΔΗΜΙΟΥΡΓΙΑ ΓΡΑΜΜΗΣ

Επιλέγουμε το κουμπί «Draw Line» και κάνουμε click μέσα στην περιοχή σχεδίασης δημιουργώντας το ένα άκρο του ευθύγραμμου τμήματος και κρατώντας πατημένο το κουμπί του ποντικιού εκτείνουμε το ευθύγραμμο τμήμα προς όποια κατεύθυνση θέλουμε. Το ευθύγραμμο τμήμα σταθεροποιείται όταν σταματήσουμε να έχουμε πατημένο το κουμπί του ποντικιού, δημιουργώντας το άλλο άκρο του ευθύγραμμου τμήματος σε εκείνες τις συντεταγμένες.

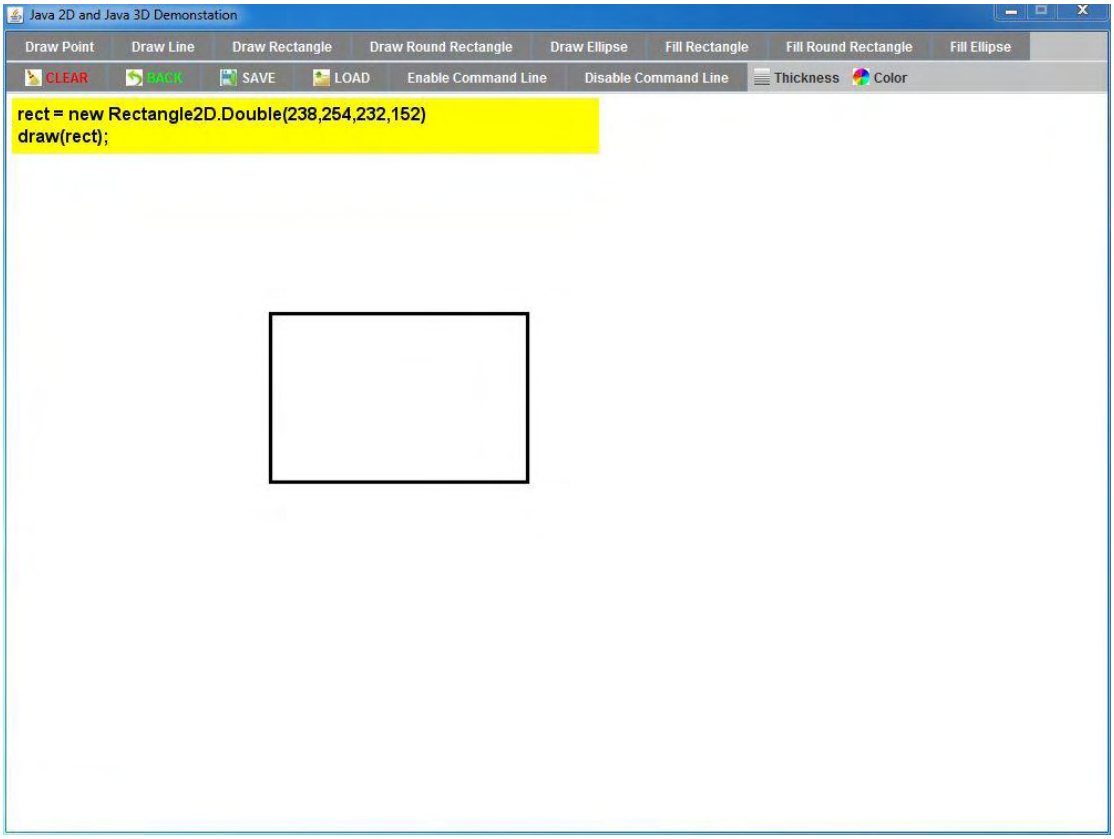
5.4.3 RECTANGLE – ΟΡΘΟΓΩΝΙΟ

Η αφηρημένη κλάση Rectangle2D είναι υποκλάση της κλάσης Shape και έχει 2 παραγόμενες κλάσεις τις Rectangle2D.Double, Rectangle2D.Float. Για να ορίσουμε ένα ορθογώνιο παράλληλο με τους άξονες χρησιμοποιούμε την εντολή :

Rectangle2D.Double rect = new Rectangle2D.Double(x,y,width,height)

Όπου δημιουργείτε ένα ορθογώνιο, αντικείμενο της κλάσης Rectangle2D.Double ,όπου η αριστερή πάνω γωνία του βρίσκεται στις συντεταγμένες (x , y) και η κάτω δεξιά γωνία του στις συντεταγμένες (x+width , y+height). Παίρνοντας υπόψη μας ότι ο άξονας των x εκτείνεται προς τα δεξιά και ο άξονας των y προς τα κάτω γνωρίζουμε ότι το ορθογώνιο θα εκτείνεται προς τα κάτω και δεξιά. Για να σχεδιαστεί το περίγραμμα του ορθογωνίου γράφουμε :

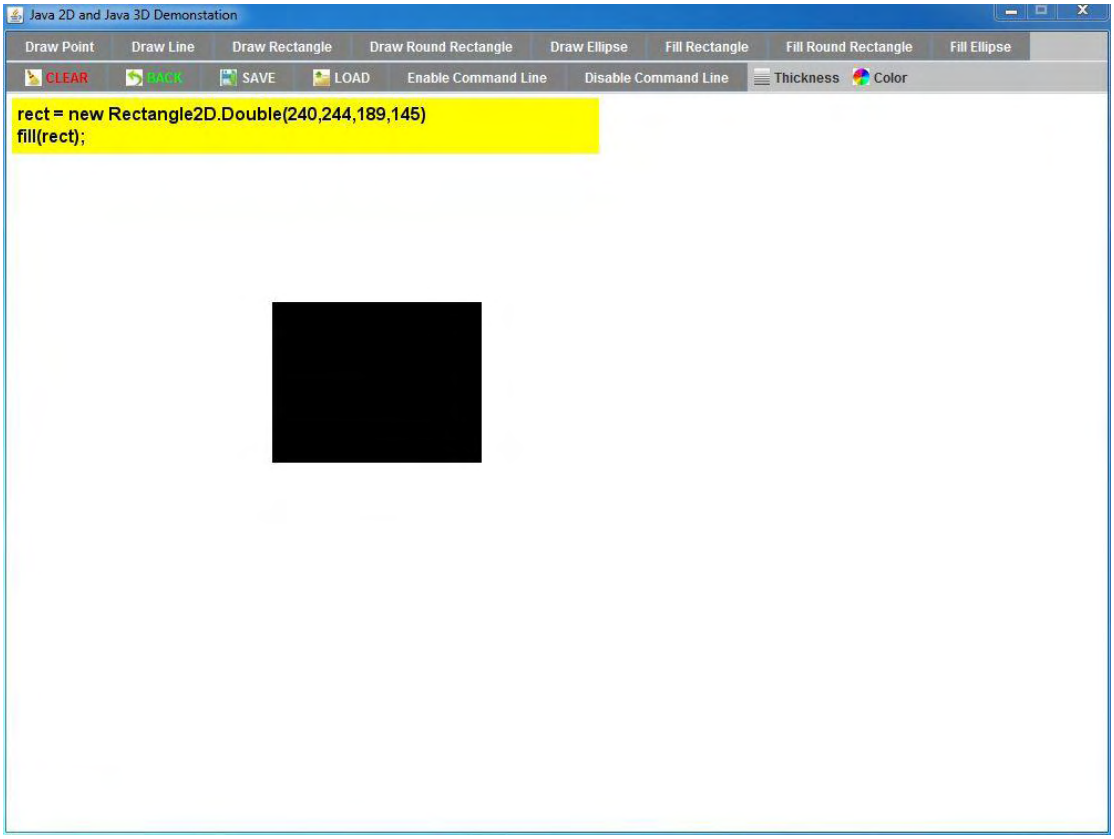
draw(rect)



ΕΙΚΟΝΑ 26 ΔΗΜΙΟΥΡΓΙΑ ΠΕΡΙΓΡΑΜΜΑΤΟΣ ΟΡΘΟΓΩΝΙΟΥ

Επιλέγουμε το κουμπί «Draw Rectangle» και κάνουμε click μέσα στην περιοχή σχεδίασης στην θέση όπου θέλουμε να βρίσκεται η πάνω αριστερά γωνία του ορθογωνίου και κρατώντας πατημένο το κουμπί του ποντικιού δημιουργούμε το ορθογώνιο και το σχήμα σταθεροποιείται εκεί που σταματάμε να πατάμε το κουμπί του ποντικιού.
Για να γεμίσουμε το ορθογώνιο με χρώμα γράφουμε :

fill(rect)



ΕΙΚΟΝΑ 27 ΔΗΜΙΟΥΡΓΙΑ ΟΡΘΟΓΩΝΙΟΥ ΓΕΜΙΣΜΕΝΟΥ ΜΕ ΧΡΩΜΑ

Επιλέγουμε το κουμπί «Fill Rectangle» και ακολουθούμε την ίδια διαδικασία με αυτή για τη δημιουργία του περιγράμματος. Το πλάτος αλλά και το ύψος του ορθογωνίου αλλάζουν δυναμικά και ανάλογα με την θέση που βρίσκεται το ποντίκι.

Για την δημιουργία των σημείων γράφουμε :

Rectangle2D.Double point = new Rectangle2D.Double(x,y,thickness_value,thickness_value)
fill(point)

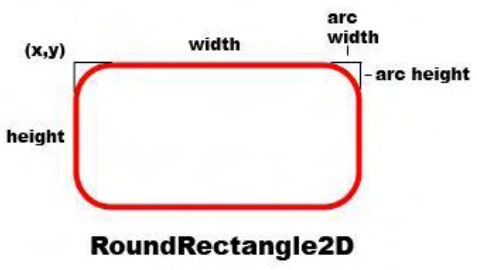
Το ορθογώνιο αυτό δημιουργείται με σταθερό πλάτος και ύψος. Το πλάτος και το ύψος είναι ίσα με την τιμή thickness_value η οποία αλλάζει σύμφωνα με το πάχος της γραμμής έτσι ώστε να δημιουργούνται μεγαλύτερα και μικρότερα σημεία. Τα ορθογώνια αυτά γεμίζονται με χρώμα με την μέθοδο fill(). Για το πάχος της γραμμής θα μιλήσουμε σε επόμενη παράγραφο [5.4.7].

5.4.4 ROUND RECTANGLE – ΣΤΡΟΓΓΥΛΕΜΕΝΟ ΟΡΘΟΓΩΝΙΟ

Η αφηρημένη κλάση Round Rectangle2D είναι υποκλάση της κλάσης Shape και έχει 2 παραγόμενες κλάσεις τις Round Rectangle2D.Double, Round Rectangle2D.Float. Για να ορίσουμε ένα στρογγυλεμένο ορθογώνιο παράλληλο με τους άξονες χρησιμοποιούμε την εντολή :

RoundRectangle2D.Double roundrect= new RoundRectangle2D.Double(x,y,width,height,arcw,arch)

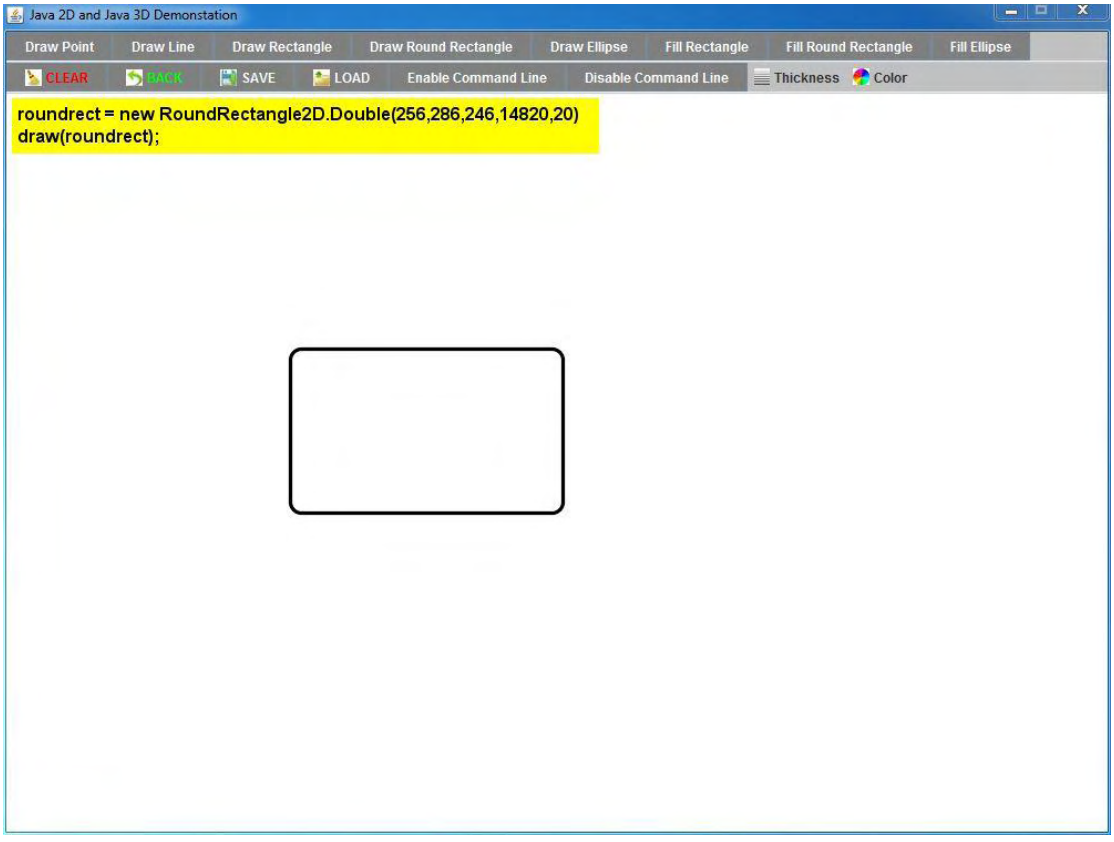
Όπου δημιουργείτε ένα στρογγυλεμένο ορθογώνιο, αντικείμενο της κλάσης Round Rectangle2D.Double , όπου η αριστερή πάνω γωνία του βρίσκεται στις συντεταγμένες (x , y) και η κάτω δεξιά γωνία του στις συντεταγμένες (x+width , y+height). Παίρνοντας υπόψη μας ότι ο άξονας των x εκτείνεται προς τα δεξιά και ο άξονας των y προς τα κάτω γνωρίζουμε ότι το στρογγυλεμένο ορθογώνιο θα εκτείνεται προς τα κάτω και δεξιά. Η μεταβλητή arcw περιγράφει το πλάτος και η μεταβλητή arch το ύψος του τόξου που χρησιμοποιείται για τις στρογγυλεμένες γωνίες. Τα arcw και arch έχουν σταθερές τιμές.



ΕΙΚΟΝΑ 28 ΣΤΡΟΓΓΥΛΕΜΕΝΟ ΟΡΘΟΓΩΝΙΟ

Για να σχεδιαστεί το περίγραμμα του στρογγυλεμένου ορθογωνίου γράφουμε :

draw(roundrect)

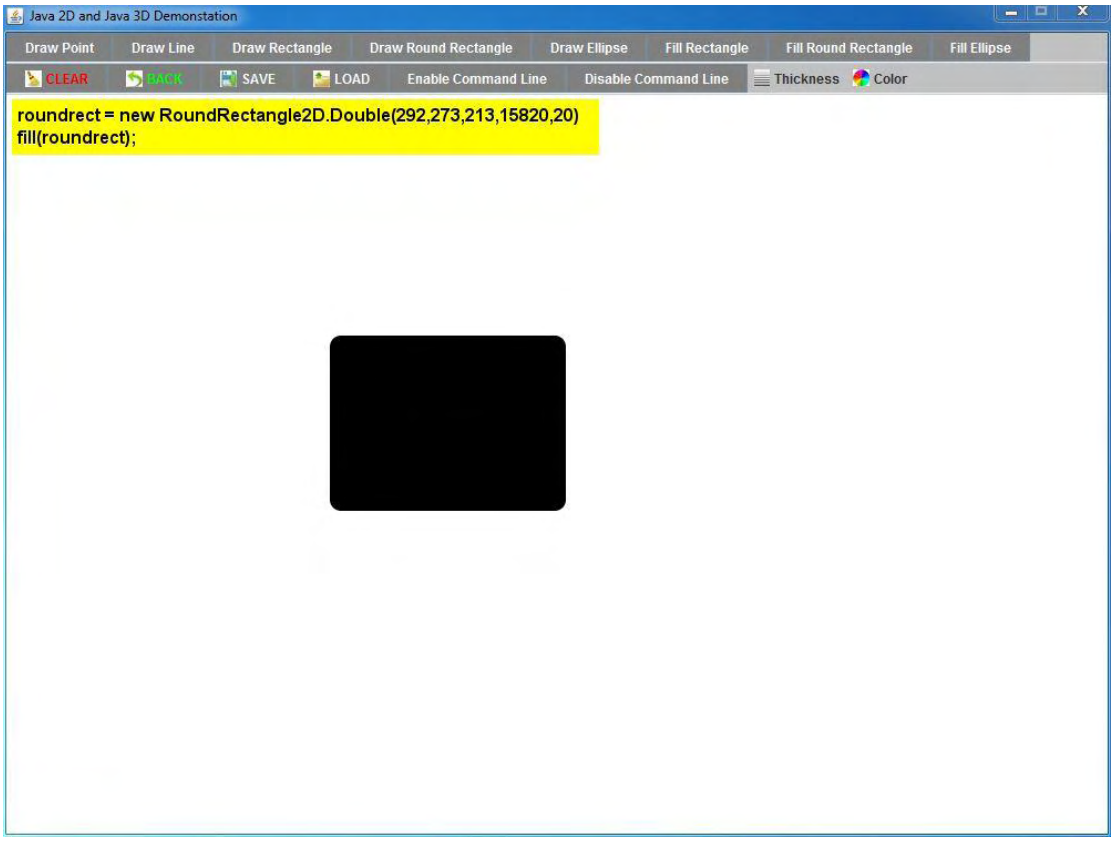


ΕΙΚΟΝΑ 29 ΔΗΜΙΟΥΡΓΙΑ ΠΕΡΙΓΡΑΜΜΑΤΟΣ ΣΤΡΟΓΓΥΛΕΜΕΝΟΥ ΟΡΘΟΓΩΝΙΟΥ

Επιλέγουμε το κουμπί «Draw Round Rectangle» και με την ίδια διαδικασία με αυτή του ορθογωνίου δημιουργείται το περίγραμμα του στρογγυλεμένου ορθογωνίου.

Για να γεμίσουμε το στρογγυλεμένο ορθογώνιο με χρώμα γράφουμε :

fill(roundrect)



ΕΙΚΟΝΑ 30 ΔΗΜΙΟΥΡΓΙΑ ΣΤΡΟΓΓΥΛΕΜΕΝΟΥ ΟΡΘΟΓΩΝΙΟΥ ΓΕΜΙΣΜΕΝΟΥ ΜΕ ΧΡΩΜΑ

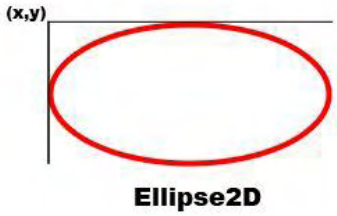
Επιλέγουμε το κουμπί «Fill Round Rectangle» και ακολουθούμε την ίδια διαδικασία με αυτή για τη δημιουργία του περιγράμματος.

5.4.5 ELLIPSE – ΈΛΛΕΙΨΗ

Η αφηρημένη κλάση Ellipse2D είναι υποκλάση της κλάσης Shape και έχει 2 παραγόμενες κλάσεις τις Ellipse2D.Double, Ellipse2D.Float. Για να ορίσουμε μία έλλειψη παράλληλη με τους άξονες χρησιμοποιούμε την εντολή :

Ellipse2D.Double elli = new Ellipse2D.Double(x,y,width,height)

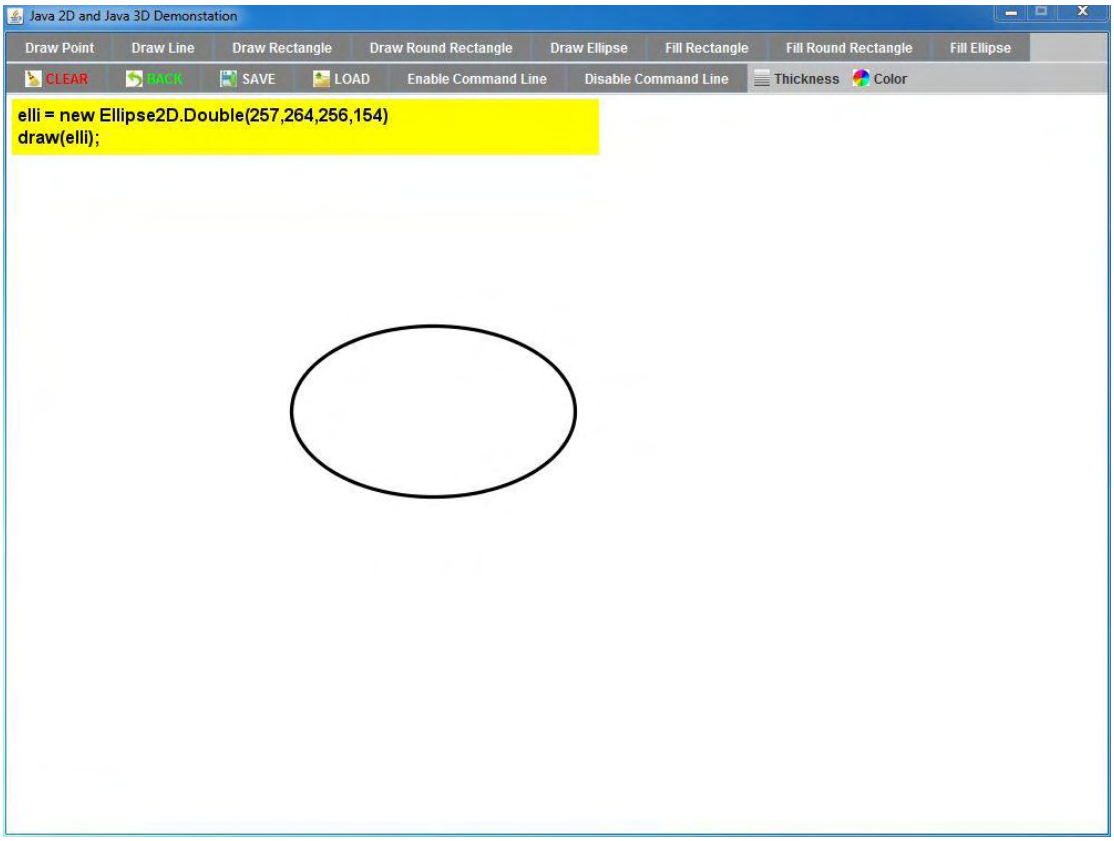
Όπου δημιουργείτε μία έλλειψη, αντικείμενο της κλάσης Ellipse2D.Double ,η οποία ορίζεται από το ορθογώνιο που την περικλείει. Έτσι αντί να ορίσουμε την έλλειψη ορίζουμε το εξωτερικό ορθογώνιο σύμφωνα με τις παραμέτρους για το αντικείμενο Rectangle2D.



ΕΙΚΟΝΑ 31 ΕΛΛΕΙΨΗ

Για να σχεδιαστεί το περίγραμμα της έλλειψης γράφουμε :

draw(elli)

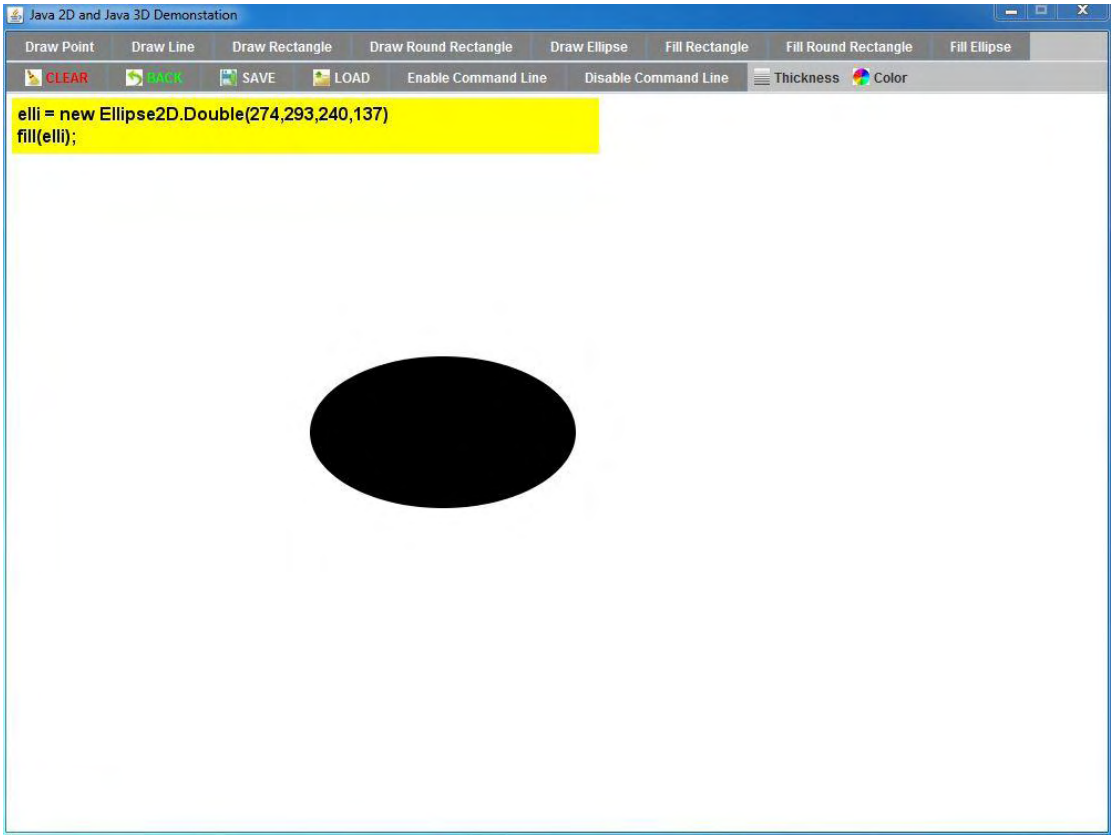


ΕΙΚΟΝΑ 32 ΔΗΜΙΟΥΡΓΙΑ ΠΕΡΙΓΡΑΜΜΑΤΟΣ ΕΛΛΕΙΨΗΣ

Επιλέγουμε το κουμπί «Draw Ellipse» και κάνουμε click μέσα στην περιοχή σχεδίασης στην θέση όπου θέλουμε να βρίσκεται η πάνω αριστερά γωνία του ορθογωνίου που περικλείει την έλλειψη και κρατώντας πατημένο το κουμπί του ποντικιού δημιουργούμε την έλλειψη μέσα στο «φανταστικό» ορθογώνιο που την περικλείει. Το σχήμα σταθεροποιείται εκεί που σταματάμε να πατάμε το κουμπί του ποντικιού.

Για να γεμίσουμε την έλλειψη με χρώμα γράφουμε :

fill(rect)

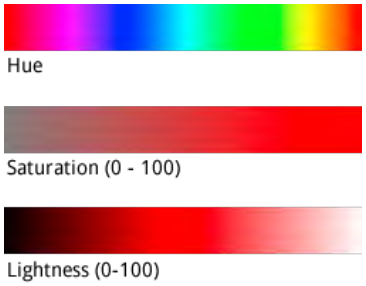


Εικονα 33 δημιουργια ελλειψης γεμισμενης με χρωμα

Επιλέγουμε το κουμπί «Fill Ellipse» και ακολουθούμε την ίδια διαδικασία με αυτή για τη δημιουργία του περιγράμματος. Το πλάτος αλλά και το ύψος του ορθογωνίου που περικλείουν την έλλειψη αλλάζουν δυναμικά και ανάλογα με την θέση που βρίσκεται το ποντίκι άρα αλλάζει δυναμικά και η έλλειψη.

5.4.6 Colors – Χρωματα

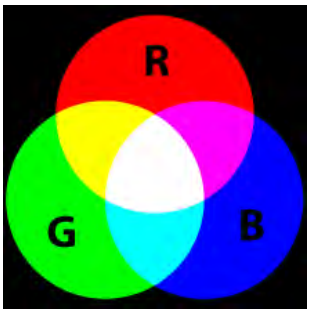
Υπάρχουν διάφορα μοντέλα χρωμάτων όπως το RGB, το CMY, το HLS. Το ανθρώπινο μάτι έχει τρεις υποδοχείς χρώματος που ο κάθε υποδοχέας είναι ευαίσθητος σε ένα μικρό κομμάτι του φάσματος του φωτός. Οι υποδοχείς αυτοί ονομάζονται κόκκινος, πράσινος και μπλε. Η αντίληψη των χρωμάτων για το ανθρώπινο μάτι στηρίζεται σε 3 συστατικά ή υποδοχείς. Τα συστατικά αυτά είναι η απόχρωση (hue – ο βαθμός στον οποίο μπορεί να περιγραφεί ένα ερέθισμα ως παρόμοιο ή διαφορετικό από τα ερεθίσματα που περιγράφονται ως κόκκινο, πράσινο, μπλε και κίτρινο), ο κορεσμός (saturation – η πολυχρωμία του χρώματος σε σχέση με τη δική του φωτεινότητα) και η ένταση (intensity / lightness – το συνολικό ποσό φωτός που περνά μέσα από μία περιοχή). Όλα τα μοντέλα χρωμάτων δημιουργούνται χρησιμοποιώντας αυτά τα 3 συστατικά ή τους υποδοχείς.



ΕΙΚΟΝΑ 34 ΑΠΟΧΡΩΣΗ, ΚΟΡΕΣΜΟΣ ΚΑΙ ΕΝΤΑΣΗ

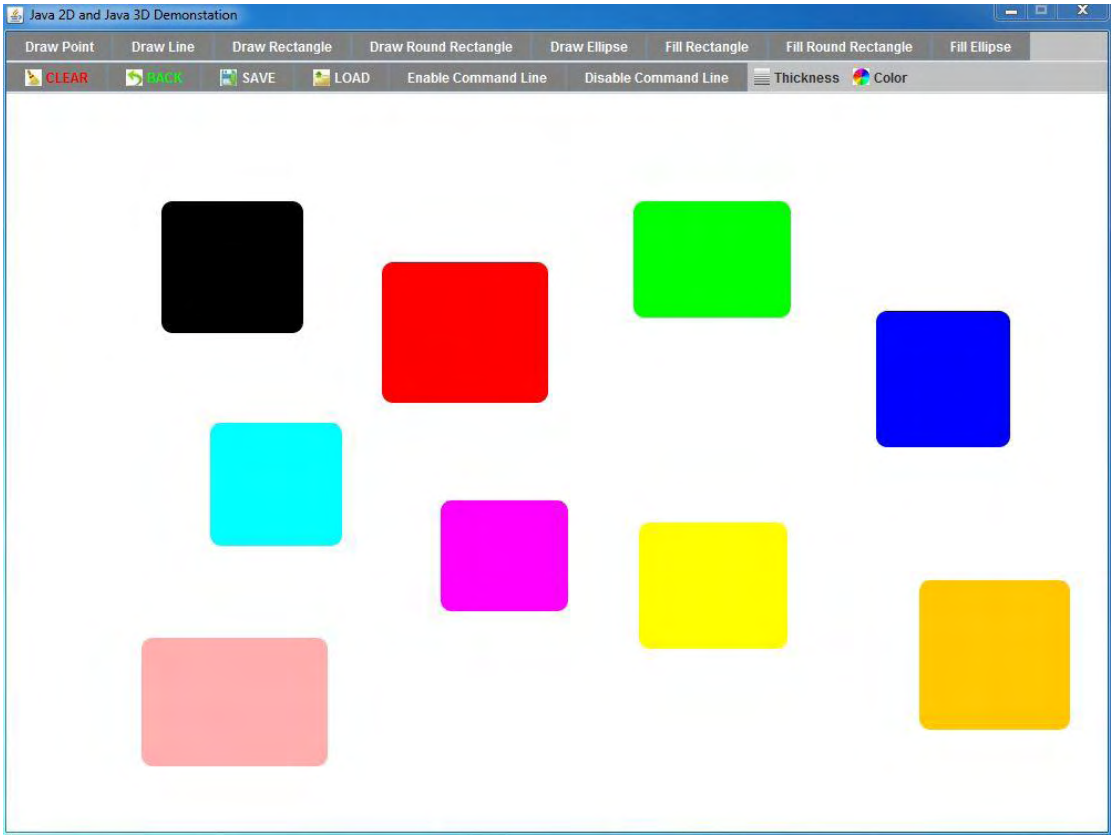
Στα πλαίσια αυτής της εργασίας χρησιμοποιείται το RGB μοντέλο μιας που είναι το πιο ευρέως διαδεδομένο μοντέλο για τις περισσότερες εφαρμογές των γραφικών η/υ.

Το μοντέλο χρωμάτων RGB είναι ένα προσθετικό χρωματικό μοντέλο στο οποίο το κόκκινο, το πράσινο και το μπλε φως προστίθενται με διάφορους τρόπους έτσι ώστε να αναπαράγουν ένα ευρύ φάσμα χρωμάτων. Το όνομα του μοντέλου προέρχεται από τα αρχικά των ονομάτων των τριών βασικών χρωμάτων και στηρίζεται στους 3 υποδοχείς που έχει το ανθρώπινο μάτι. Το RGB μοντέλο έχει 3 τιμές R, G, B ∈ [0 , 1]. Η μέγιστη ένταση αντιστοιχεί στη τιμή 1 και η ελάχιστη στην τιμή 0. Το (0 , 0 , 0) αντιστοιχεί στο μαύρο, το (1 , 1 , 1) στο άσπρο, το (1 , 0 , 0) στο κόκκινο, το (0 , 1 , 0) στο πράσινο και το (0 , 0 , 1) στο μπλε. Συνήθως στην κωδικοποίηση των χρωμάτων χρησιμοποιείται 1 byte για κάθε ένα από τα αρχικά χρώματα έτσι ώστε το καθένα από αυτά να έχει 256 διαφορετικά επίπεδα έντασης.



ΕΙΚΟΝΑ 35 ΜΕΙΞΕΙΣ ΧΡΩΜΑΤΩΝ – RGB ΜΟΝΤΕΛΟ

Στην εφαρμογή πηγαίνοντας στο μενού Color μπορείς να επιλεχθεί το χρώμα για το περίγραμμα ή για το γέμισμα του επιλεγμένου σχήματος.



ΕΙΚΟΝΑ 36 ΤΑ ΧΡΩΜΑΤΑ ΤΗΣ ΕΦΑΡΜΟΓΗΣ

Για να δημιουργήσουμε ένα αντικείμενο της κλάσης Color γράφουμε :

Color in_col = new Color(reg,green,blue)

Όπου τα red, green, blue παίρνουν τιμές σε ακέραιους από 0 έως 255 ή σε float από 0 έως 1. Επίσης η κλάση Color περιέχει κάποιες σταθερές που περιγράφουν τα βασικά χρώματα και καλούνται ως Color.BLUE, Color.MAGENTA κ.τ.λ. . Για να οριστεί το χρώμα του σχήματος ως το χρώμα col που δημιουργήσαμε καλούμε τη μέθοδο για το Graphics2D αντικείμενο (μπορεί να χρησιμοποιήσει και τις 2 μεθόδους – αντικείμενα της κλάσης Graphics μπορούν να χρησιμοποιήσουν μόνο την setColor()).:

setPaint(in_col) // μέθοδος κλάσης Graphics2D

ή την

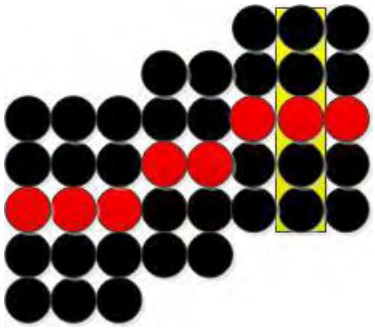
setColor(in_col) // μέθοδος κλάσης Graphics – βασικής κλάσης της Graphics2D

Για να αλλάξει το χρώμα πρέπει να καλέσουμε ξανά την παραπάνω μέθοδο με άλλο χρώμα. Μέχρι τότε όλα τα σχήματα που δημιουργούνται θα βάφονται με το προεπιλεγμένο χρώμα.

5.4.7 LINE THICKNESS – ΠΑΧΟΣ ΓΡΑΜΜΗΣ

Οι συσκευές εξόδου σήμερα διαθέτουν πολύ μεγάλες αναλύσεις με αποτέλεσμα οι γραμμές που ζωγραφίζονται με πλάτος 1 pixel να φαίνονται πάρα πολύ λεπτές. Υπάρχουν διάφορες τεχνικές για να ζωγραφίσεις πιο παχιές γραμμές, μία από αυτές είναι η **αντιγραφή pixel** (pixel replication).

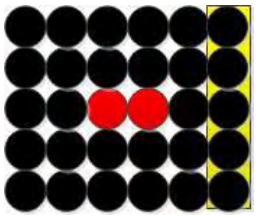
Στην αντιγραφή pixel για κάθε ένα pixel που ζωγραφίζεται ζωγραφίζονται επίσης n pixels πάνω και n pixels κάτω από το αρχικό pixel. Έτσι το πάχος από 1 pixel γίνεται 2n + 1 pixel. Η αντιγραφή pixel μπορεί να γίνει είτε στον άξονα των x είτε στον άξονα των y ανάλογα με την κλίση της γραμμής που θέλουμε να ζωγραφίσουμε.



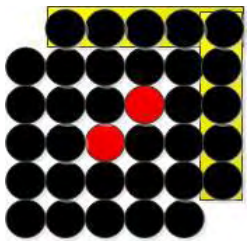
Εικονα 37 pixel replication

Μία άλλη τεχνική είναι η τεχνική **moving pen** (κινούμενο στυλό). Στην τεχνική αυτή για κάθε ένα pixel που ζωγραφίζεται , ζωγραφίζεται ένα τετράγωνο 5x5 pixel με κεντρικό pixel το αρχικό, λειτουργώντας σαν την μύτη ενός στυλό. Για να μην ζωγραφίζονται συνεχώς τα ίδια pixels πρέπει να ληφθεί αρχικά υπόψη το σχήμα της μύτης του στυλό. Πιο συγκεκριμένα :

- Για το αρχικό pixel ζωγραφίζονται και τα 25 pixel
- Αν το επόμενο pixel προς σχεδίαση είναι το E (eastern - ανατολικό) τότε ζωγραφίζονται μόνο τα 5 pixel στα δεξιά.
- Αν το επόμενο pixel προς σχεδίαση είναι το NE (north eastern – βόρειο ανατολικό) τότε ζωγραφίζονται τα 9 pixels πάνω και δεξιά.

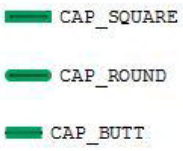


Εικονα 38 moving pen - E pixel



Εικονα 39 moving pen - NE pixel

Όταν ζωγραφίζονται παχιές γραμμές οι άκρες των γραμμών όπως και οι ενώσεις μεταξύ των γραμμών μπορούν να μοντελοποιηθούν με πολλούς τρόπους. Οι άκρες των γραμμών μπορεί να καταλήγουν σε ορθογώνιο σχήμα, σε ημικύκλιο ή να έχουν αποκοπεί.

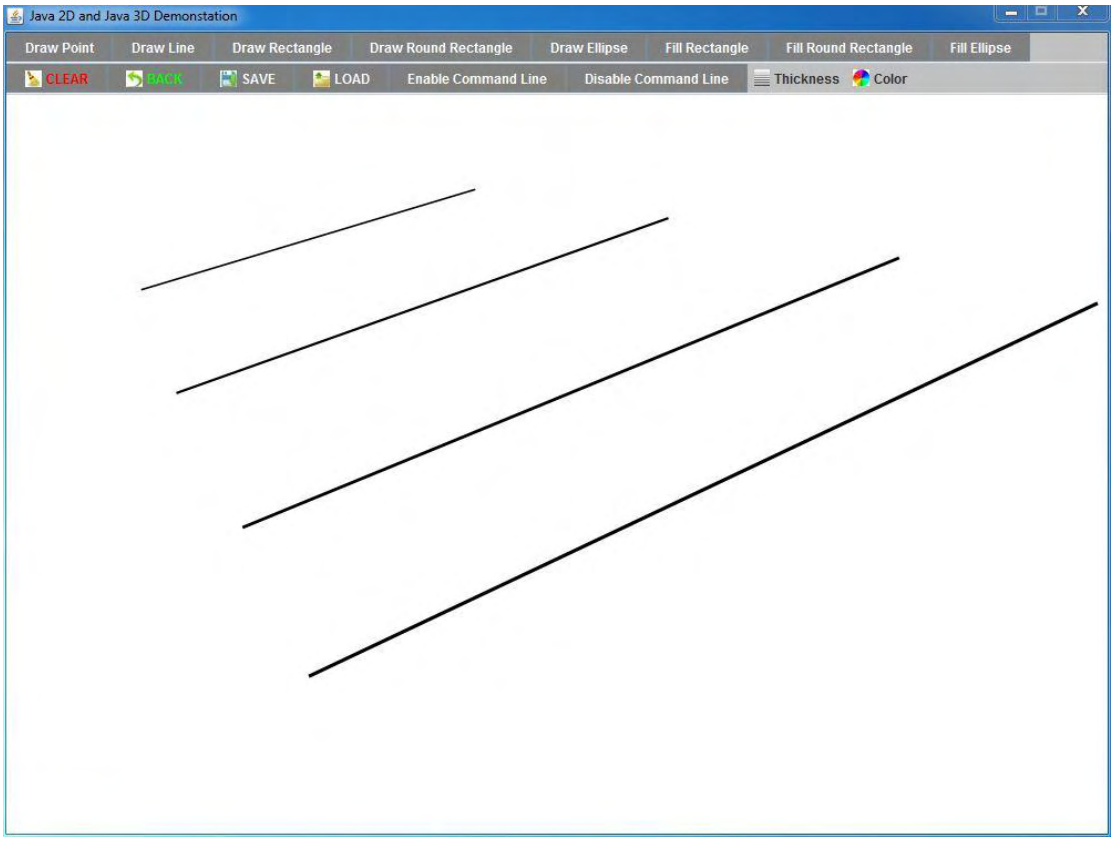


Εικονα 40 ακρες γραμμων

Οι ενώσεις των γραμμών μπορεί να είναι είτε μυτερές είτε αποκομμένες είτε στρογγυλεμένες.

Εικονα 41 ενωσεις γραμμων

Στην εφαρμογή πηγαίνοντας στο μενού Thickness μπορείς να επιλέξεις το πάχος της γραμμής.



Εικονα 42 line thickness

Για να καθοριστούν το πάχος, οι καταλήξεις (άκρες) και οι ενώσεις των γραμμών πρέπει να δημιουργηθεί ένα αντικείμενο της κλάσης BasicStroke.

BasicStroke bs = new BasicStroke(thickness , cap , join)

Η παράμετρος thickness καθορίζει το πάχος της γραμμής, η παράμετρος cap τις καταλήξεις των γραμμών και η παράμετρος join τις ενώσεις. Η παράμετρος thickness παίρνει τιμές float από 0.0 και πάνω. Η τιμή 0.0 αντιστοιχεί στην πιο λεπτή γραμμή. Η παράμετρος cap παίρνει μία από τις τιμές : CAP_SQUARE, CAP_ROUND, CAP_BUTT με αποτελέσματα που φαίνονται στην εικόνα 40. Η παράμετρος join παίρνει μία από τις τιμές : JOIN_MITTER, JOIN_BEVEL, JOIN_ROUND με αποτελέσματα που φαίνονται στην εικόνα 41.

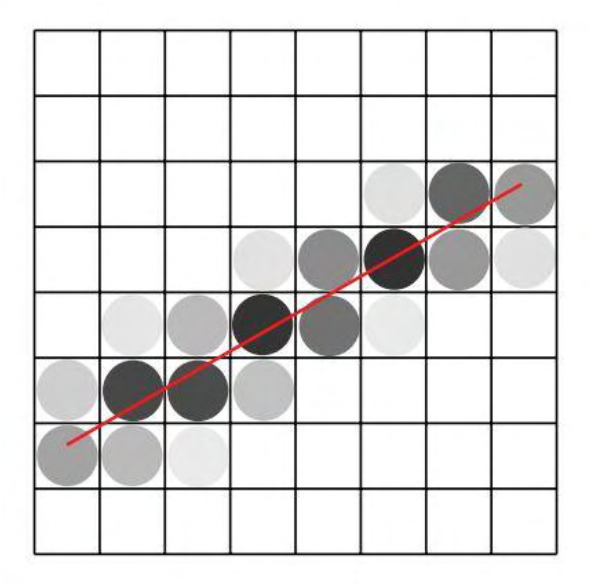
Στην εφαρμογή οι τιμές που χρησιμοποιήθηκαν για τις παραμέτρους είναι CAP_SQUARE και JOIN_MITER που είναι και οι default τιμές για αυτές τις παραμέτρους και για την παράμετρο thickness 1.6, 2.2, 2.8, 3.2 με αποτελέσματα που φαίνονται στην εικόνα 42.

5.4.8 Antialiasing – Ομαλοποιηση

Όπως αναφέρθηκε σε προηγούμενη παράγραφο [2.3] κατά τη δημιουργία των γραμμών ειδικά όταν το πάχος τους είναι 1 pixel μπορούν να δημιουργηθούν φαινόμενα aliasing δηλαδή μη ομαλοποιημένες γραμμές. Αν υποθέσουμε ότι τα pixels μπορούν να χρωματιστούν μόνο με μαύρο και άσπρο τότε τίποτα δεν μπορεί να γίνει για το φαινόμενο aliasing. Αν υποθέσουμε όμως ότι τα pixel μπορούν να χρωματιστούν με την κλίμακα του γκρι ή γενικά με χρώματα τότε τα φαινόμενα aliasing μπορούν να μειωθούν.

Η βασική ιδέα είναι να μειωθεί το φαινόμενο «μαλακώνοντας» το περίγραμμα των γραμμών μειώνοντας την ένταση αυτών των pixel. Η τεχνική αυτή καλείται antialiasing. Υπάρχουν αρκετές τεχνικές που ακολουθούνται για να εφαρμοστεί το antialiasing παρακάτω θα αναφερθούν δύο από αυτές.

Στην τεχνική **Unweighted Area Sampling** (αστάθμητη περιοχή δειγματοληψίας) η γραμμή ερμηνεύεται σαν ένα μακρύ και λεπτό ορθογώνιο. Το pixel ερμηνεύεται σαν ένα τετράγωνο που μπορεί να γεμίσει με χρώμα και όχι σαν ένα σημείο. Η ένταση του κάθε pixel επιλέγεται ανάλογα με το ποσό της επιφάνειας του pixel που καλύπτεται από το ορθογώνιο της γραμμής. Το μειονέκτημα της τεχνικής αυτής είναι ότι για κάθε pixel πρέπει να προσδιορίζεται και να υπολογίζεται η τομή του με το τετράγωνο της γραμμής. Η διαδικασία αυτή απαιτεί μεγάλο υπολογιστικό κόστος.

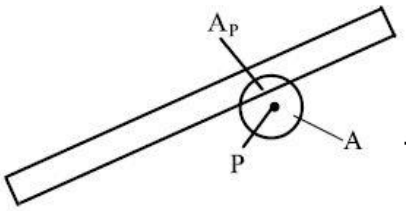


Εικονα 43 Unweighted Area Sampling

Στην τεχνική **Weighted Area Sampling** (σταθμισμένη περιοχή δειγματοληψίας) η γραμμή ερμηνεύεται πάλι σαν ένα μακρύ και λεπτό ορθογώνιο και τα pixel σαν τετράγωνα επίσης λαμβάνει υπόψη και μία συνάρτηση βαρών w(x,y) η οποία έχει την μεγαλύτερη τιμή στο κέντρο των pixels και η τιμή μειώνεται καθώς μεγαλώνει η απόσταση από το κέντρο. Μία συνήθης συνάρτηση βαρών μπορεί να περιγραφεί ως ένας κύκλος A γύρω από ένα pixel P. Η ένταση για κάθε pixel δίνεται από τον τύπο

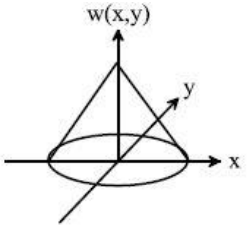
$$\frac{\int_{AREA_P} w(x,y)dx\,dy}{\int_{AREA} w(x,y)dx\,dy}$$

Όπου $AREA_P$ είναι η τομή των περιοχών του pixel και της γραμμής.



Εικονα 44 περιοχεσ pixel, γραμμησ και η τομη τουσ

Με μία συνάρτηση βαρών όπως της εικόνας 45 μπορούμε να πούμε ότι τελικά η ένταση του pixel εξαρτάται μόνο από την απόστασή του από την γραμμή.



ΕΙΚΟΝΑ 45 ΣΥΝΑΡΤΗΣΗ ΒΑΡΩΝ

Έτσι η ένταση μπορεί να γραφεί σαν συνάρτηση I(dp). Όπου dp η απόσταση του pixel P από την γραμμή. Οι τιμές της έντασης περιορίζονται σύμφωνα με την ένταση που μπορεί να αποδοθεί , για παράδειγμα σε μία οθόνη η/υ η τιμή φτάνει μέχρι το 256. Έτσι δημιουργείται ένα πεπερασμένο και διακριτό σύνολο τιμών για την ένταση I : {0 , … , imax}. Για κάθε pixel στην γειτονία της γραμμής θα πρέπει να βρεθεί η τιμή της έντασης που του αναλογεί.

Το να δημιουργηθεί μία γραμμή είναι ανάλογη διαδικασία με το να βρεθούν οι τιμές της έντασης στα γειτονικά pixel της γραμμής. Η διαδικασία για την δημιουργία της γραμμής είναι να προχωράμε κατά x σαρώνοντας το πλέγμα των pixel και να υπολογίζουμε την τιμή του y. Τώρα αντί να σαρώνεται μόνο ο άξονας των x πρέπει να σαρώνεται η περιοχή γύρω από την γραμμή και επίσης αντί να υπολογίζεται η τιμή του y πρέπει να υπολογίζεται η τιμή της έντασης. Βάση αυτής της αναλογίας ο αλγόριθμος Midpoint επεκτάθηκε για να μπορέσει να δημιουργήσει το φαινόμενο του antialiasing.

Στην εφαρμογή για την δημιουργία ομαλών γραμμών έχει χρησιμοποιηθεί σε όλη την σχεδίαση η μέθοδος :

setRenderingHint(RenderingHints.KEY_ANTIALIASING,RenderingHints.VALUE_ANTIALIAS_ON);

για το Graphics2D αντικείμενο. Η μέθοδος αυτή κάνει δυνατή την σχεδίαση όλων των σχημάτων και των περιοχών με την τεχνική antialiasing.

5.4.9 CLEAR AND BACK– ΚΑΘΑΡΙΣΜΟΣ ΚΑΙ ΕΠΙΣΤΡΟΦΗ

CLEAR

Κάθε φορά που θέλουμε να καθαρίσουμε την περιοχή σχεδίασης μία Boolean μεταβλητή, η clear, αλλάζει από false σε true και καλείται η μέθοδος repaint() για το panel στο οποίο είμαστε. Η clear είναι αρχικοποιημένη σε false. Για να γίνει καθαρισμός της περιοχής σχεδίασης γράφουμε :

clear=true

standardShapesPanel.repaint()

Η μέθοδος paint Component() καλείται λόγω της repaint() και εκεί εφόσον η clear είναι true ανανεώνεται η in_bi buffered image (εικόνα στην μνήμη του η/υ). Η in_bi βάφεται άσπρη καλύπτοντας όλα τα σχήματα που είχαν σχεδιαστεί και αντιγράφεται στην οθόνη του η/υ. Η τιμή της μεταβλητής clear γίνεται πάλι false.

```
if (in_bi == null || clear==true) {
        in_bi = (BufferedImage)this.createImage(this.getWidth(),this.getHeight());
        Graphics2D g2bi = in_bi.createGraphics();
        g2bi.setColor(Color.WHITE);
        g2bi.fillRect(0,0,this.getWidth(),this.getHeight());
        clear = false;
}
```

Ο έλεγχος μέσα στο if για in_bi == null αναφέρεται στην πρώτη φορά που δημιουργείται η buffered image εφόσον είναι αρχικοποιημένη σε null.

Στην εφαρμογή για να καθαριστεί η περιοχή σχεδίασης πατάμε το κουμπί «CLEAR».

BACK

Η εφαρμογή είναι χωρισμένη σε επιλογές που επικοινωνούν μέσω του κοινού εξωφύλλου (CoverPanel). Είναι δυνατή η επιστροφή μέσα από μία εφαρμογή στο εξώφυλλο για να μας δοθεί η δυνατότητα να επιλέξουμε μία άλλη επιλογή. Για να πραγματοποιηθεί αυτό θα πρέπει όλες οι απαραίτητες μεταβλητές να πάρουν τις αρχικές τους τιμές

```
standardShapesPanel.in_shape = Shape.NOTHING;  // default επιλογή για το σχήμα
standardShapesPanel.in_thickness = Thickness.T1;  // default πάχος γραμμής
standardShapesPanel.in_color = Color.BLACK;   // default χρώμα
```

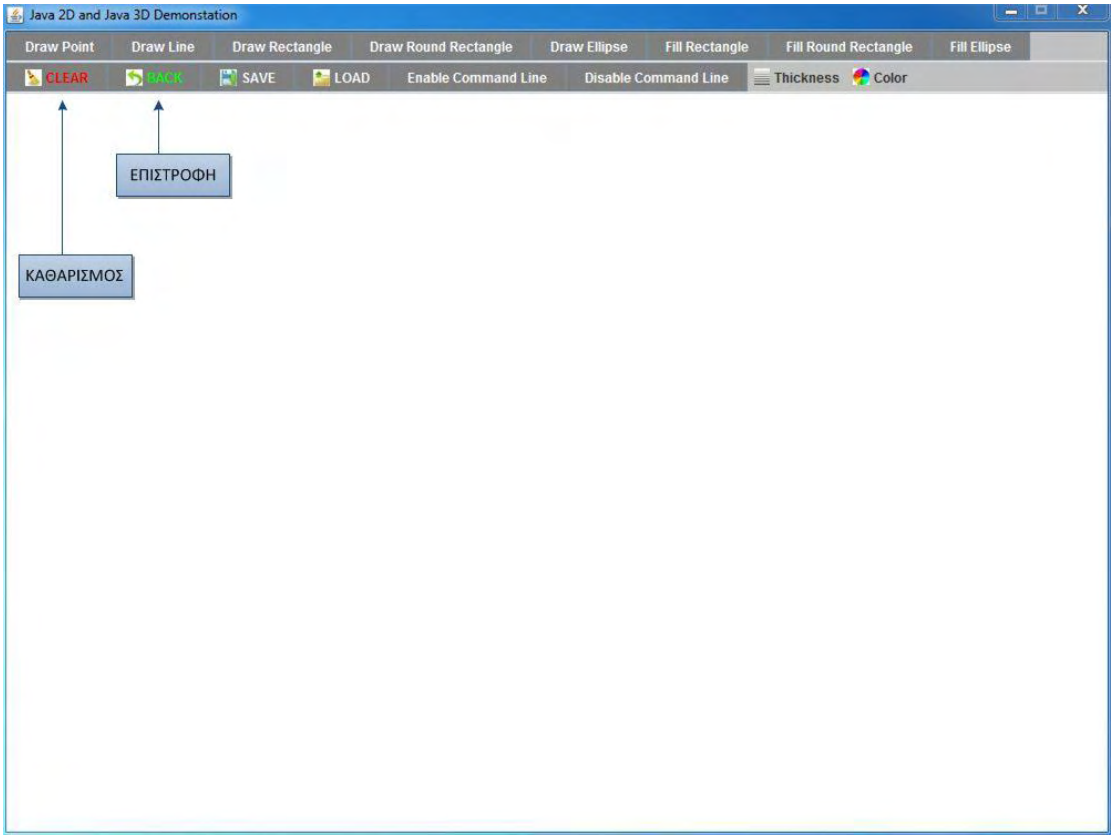
να ανανεωθεί η buffered image στην αρχική της μορφή για να είναι η επιλογή έτοιμη για χρήση την επόμενη φορά

```
clear=true;
standardShapesPanel.repaint();
```

και θα πρέπει να γίνει ορατό το CoverPanel και να μην είναι πλέον ορατό το panel της επιλογής που βρισκόμαστε , StandardShapesPanel.

```
standardShapesPanel.setVisible(false);
setContentPane(coverPanel);
coverPanel.setVisible(true);
```

Στην εφαρμογή για να επιστρέψουμε στο εξώφυλλο πατάμε το κουμπί «BACK».



ΕΙΚΟΝΑ 46 ΚΑΘΑΡΙΣΜΟΣ ΠΕΡΙΟΧΗΣ ΣΧΕΔΙΑΣΗΣ ΚΑΙ ΕΠΙΣΤΡΟΦΗ ΣΤΟ ΕΞΩΦΥΛΛΟ

5.4.10 SAVE AND LOAD – ΑΠΟΘΗΚΕΥΣΗ ΚΑΙ ΦΟΡΤΩΣΗ

Σε κάποιες επιλογές της εφαρμογής υπάρχει η δυνατότητα αποθήκευσης της εικόνας που έχει δημιουργηθεί και φόρτωσης οποιασδήποτε εικόνας.

Όλη η σχεδίαση γίνεται σε μία εικόνα στην μνήμη του η/υ η οποία αντιγράφεται στην οθόνη του η/υ οπότε αρκεί να αποθηκεύσουμε την εικόνα που βρίσκεται στην μνήμη που είναι αντικείμενο της κλάσης Buffered Image ή να φορτώσουμε μία εικόνα να την αντιγράψουμε στην εικόνα στην μνήμη και να την εμφανίσουμε στην οθόνη του η/υ. Για να κάνουμε αυτές τις διαδικασίες χρησιμοποιούμε ένα Swing component το JFileChooser που εμφανίζει στην οθόνη ένα παράθυρο μέσω του οποίου μας δίνεται η δυνατότητα να επιλέξουμε ένα αρχείο ή έναν κατάλογο για να το ανοίξουμε ή να το αποθηκεύσουμε.

SAVE

Για να δημιουργήσουμε ένα αντικείμενο της κλάσης JFileChooser γράφουμε :

JFileChooser save = new JFileChooser()

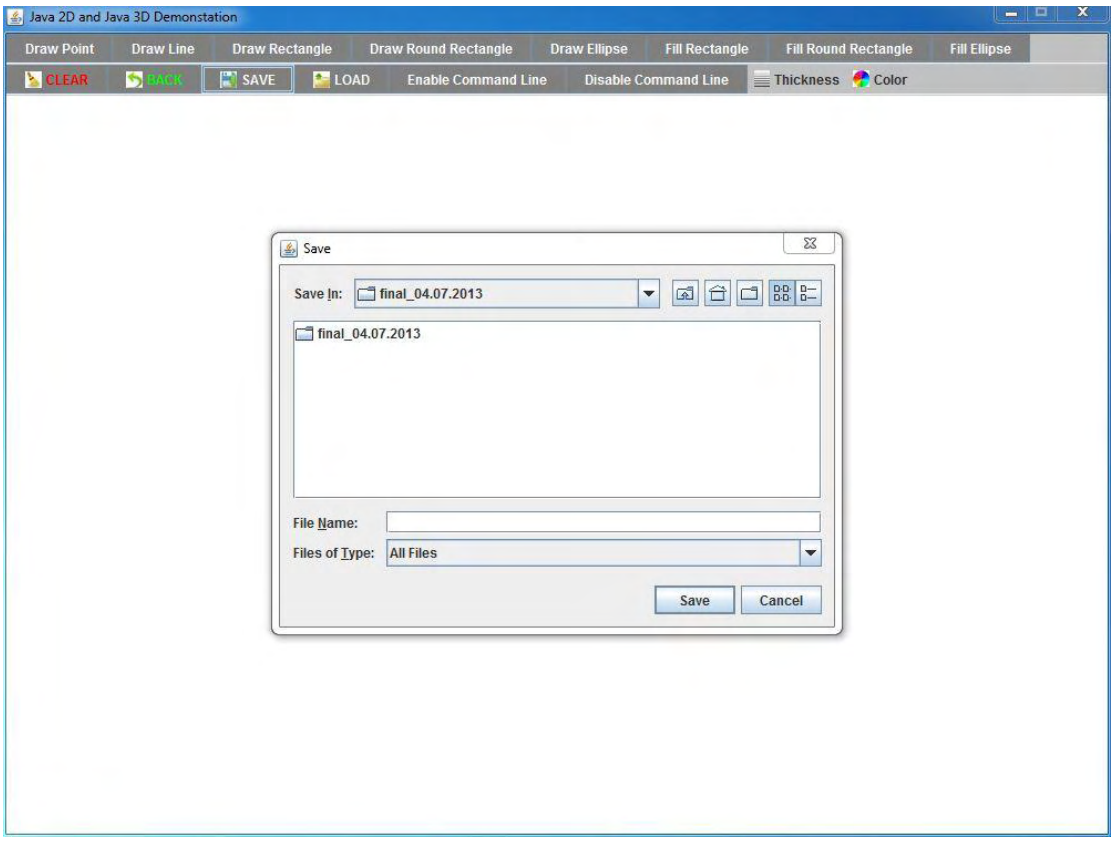
Για να ελέγξουμε αν ο χρήστης επέλεξε ένα αρχείο ή έναν κατάλογο και δεν πάτησε cancel (ακύρωση) πρέπει να ελέγξουμε την επιστρεφόμενη τιμή αν είναι ίση με την τιμή JFileChooser.APPROVE_OPTION .
Έπειτα αποθηκεύσουμε σε μία μεταβλητή τύπου string το path στο οποίο βρίσκεται το αρχείο και το συνενώνουμε με το επιλεγμένο όνομα και με την κατάληξη «.jpg» γιατί έχουμε επιλέξει οι εικόνες

μας να αποθηκεύονται σε αυτόν τον τύπο αρχείων. Ύστερα δημιουργούμε ένα αρχείο χρησιμοποιώντας την παραπάνω μεταβλητή string και αποθηκεύουμε τη ήδη δημιουργημένη buffered image στην αντίστοιχη θέση. Ο κώδικας φαίνεται παρακάτω.

```
if(save.showSaveDialog(standardShapesPanel)==JFileChooser.APPROVE_OPTION){
            String path = save.getSelectedFile().getAbsolutePath();
            path=path + ".jpg";
            try {
               File fileSave = new File(path);
               ImageIO.write(in_bi,"jpg", fileSave);
            } catch (IOException ex) {
               System.out.println(ex);
               JOptionPane.showMessageDialog(standardShapesPanel, ex);
            }
}
```

Σε περίπτωση που γίνει κάτι λάθος κατά τη διάρκεια της αποθήκευσης εμφανίζεται το αντίστοιχο μήνυμα στην οθόνη.

Στην εφαρμογή για να αποθηκεύσουμε την εικόνα πατάμε το κουμπί «SAVE».



ΕΙΚΟΝΑ 47 ΑΠΟΘΗΚΕΥΣΗ ΕΙΚΟΝΑΣ

LOAD

Για να φορτώσουμε μία εικόνα δημιουργούμε πάλι ένα αντικείμενο της κλάσης JFileChooser προσδιορίζοντας αυτή τη φορά φίλτρα για τα αρχεία μιας που θέλουμε να εμφανίζονται μόνο μερικοί τύποι αρχείων που υποστηρίζονται από την εφαρμογή. Οι τύποι αυτοί είναι .jpg, .png, .gif.

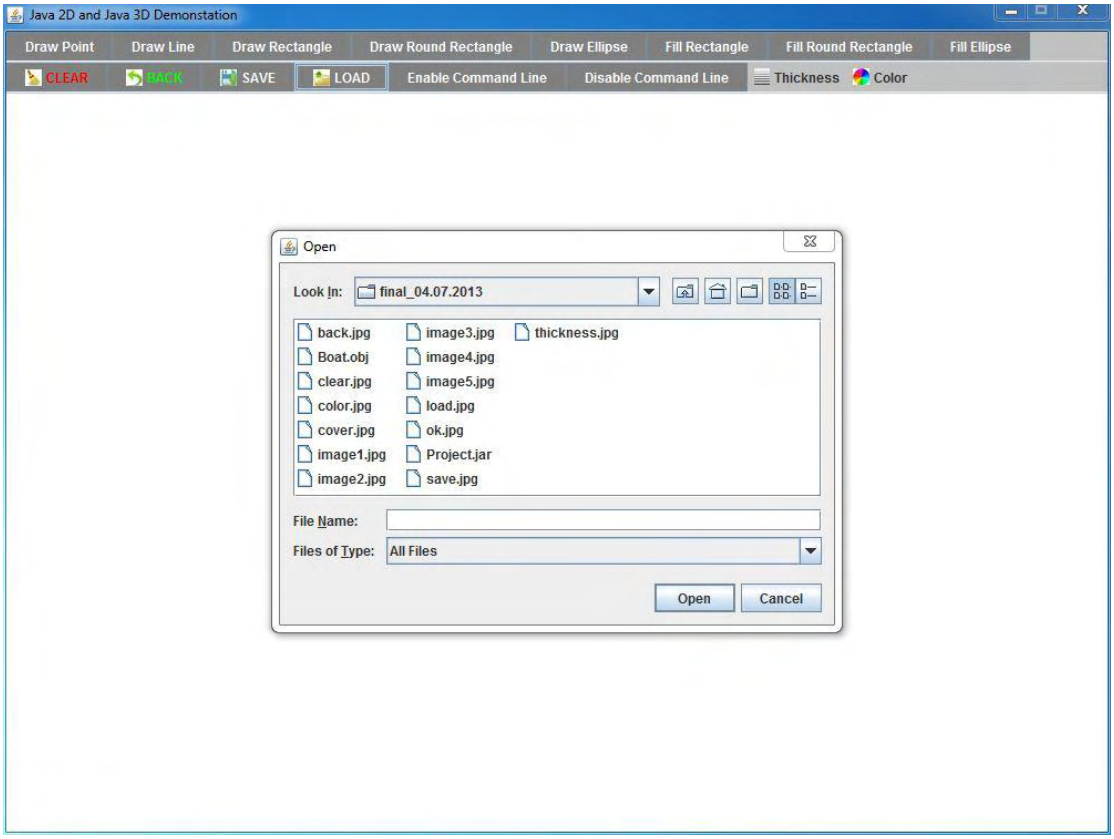
```
JFileChooser load = new JFileChooser()

load.setFileFilter(new FileNameExtensionFilter("JPEG File", "jpg"))

load.setFileFilter(new FileNameExtensionFilter("PNG File", "png"))

load.setFileFilter(new FileNameExtensionFilter("GIF File", "gif"))
```

Όπως και πριν ελέγχουμε την επιστρεφόμενη τιμή από το JFileChooser και δημιουργούμε ένα αρχείο στο οποίο φορτώνουμε το αντίστοιχο αρχείο που επιλέχθηκε μέσω του JFileChooser. Έπειτα δημιουργούμε μία καινούρια buffered image (img) στην οποία αποθηκεύουμε την εικόνα που φορτώθηκε. Δημιουργούμε ένα νέο αντικείμενο για την ήδη υπάρχουσα buffered image (την εικόνα στην οποία γίνεται η σχεδίαση – in_bi) το οποίο το χρησιμοποιούμε για να αντιγράψουμε την καινούρια buffered image (img) στην παλιά (in_bi). Αυτό γίνεται γιατί δεν γνωρίζουμε τις διαστάσεις της εικόνας που φορτώνεται με αποτέλεσμα αν την αντιγράψουμε κατευθείαν την φορτωμένη εικόνα στην buffered image της εικόνας σχεδίασης να εμφανιστεί με τις κανονικές της διαστάσεις δηλαδή πολύ μικρή ή πολύ μεγάλη. Έτσι δημιουργούμε μία καινούρια buffered image (img) με τις διαστάσεις της εικόνας που φορτώθηκε την οποία την τροποποιούμε στις διαστάσεις της περιοχής σχεδίασης (buffered image – in_bi).Τέλος καλούμε την μέθοδο repaint() για το συγκεκριμένο panel για να ανανεώσει την οθόνη γράφοντας την ανανεωμένη buffered image (in_bi) στην οθόνη.

```
if(load.showOpenDialog(standardShapesPanel)==JFileChooser.APPROVE_OPTION){
            try {
               File fileLoad = load.getSelectedFile();
               BufferedImage img = ImageIO.read(fileLoad);
               in_bi = new BufferedImage(WIDTH,HEIGHT, BufferedImage.TYPE_INT_RGB);
               Graphics2D df = in_bi.createGraphics();
               df.drawImage(img,0,0,WIDTH,HEIGHT,null);
               standardShapesPanel.repaint();

            } catch (IOException ex) {
               System.out.println(ex);
               JOptionPane.showMessageDialog(standardShapesPanel, ex);
            }
         }
```

Σε περίπτωση που γίνει κάτι λάθος κατά τη διάρκεια της φόρτωσης της εικόνας εμφανίζεται το αντίστοιχο μήνυμα στην οθόνη.

Στην εφαρμογή για να φορτώσουμε μία εικόνα πατάμε το κουμπί «LOAD».



ΕΙΚΟΝΑ 48 ΦΟΡΤΩΣΗ ΕΙΚΟΝΑΣ

5.4.11 COMMAND LINE – ΓΡΑΜΜΗ ΕΝΤΟΛΩΝ

Η γραμμή εντολών είναι το κίτρινο ορθογώνιο που βρίσκεται στην πάνω αριστερή γωνία της περιοχής σχεδίασης και περιέχει τις εντολές που εκτελούνται για την δημιουργία των σχημάτων.

Το ορθογώνιο αυτό ανανεώνεται κάθε φορά ανάλογα με τα σχήματα που ζωγραφίζονται και επίσης ανάλογα με την επιλογή του χρήστη μπορεί να είναι ορατό ή όχι.

Για να σχεδιαστεί το ορθογώνιο γράφουμε τις μεθόδους:

setColor(Color.YELLOW) // μέθοδος επιλογής χρώματος

fillRect(5,60,530,50) // σχεδίαση ορθογωνίου γεμισμένου με χρώμα - οι διαστάσεις του ορθογωνίου αλλάζουν σύμφωνα με τις ανάγκες τις κάθε επιλογής καλούμενες για το αντικείμενο Graphics2D.

Για να εμφανιστεί το κείμενο χρησιμοποιούμε τη μέθοδο

drawstring(“text”, x , y)

Η drawstring εμφανίζει το κείμενο που περικλείεται μέσα στα εισαγωγικά στην θέση (x, y). Περισσότερα για το κείμενο θα αναφερθούν σε επόμενο κεφάλαιο μιας που υπάρχει ξεχωριστή επιλογή της εφαρμογής που αναφέρεται σε αυτό [10].

Όταν ο χρήστης επιλέξει να μην εμφανίζεται η γραμμή εντολών τότε ζωγραφίζεται ένα ορθογώνιο χρώματος άσπρου πάνω από το κίτρινο ορθογώνιο καλύπτοντας έτσι το προηγούμενο ορθογώνιο. Το μόνο που αλλάζει στον παραπάνω κώδικα είναι το χρώμα του ορθογωνίου.

Για να ελέγχουμε κάθε φορά αν πρέπει να εμφανιστεί η γραμμή εντολών ανάλογα με τις επιλογές του χρήστη, χρησιμοποιείται μία Boolean μεταβλητή, η cl, που είναι αρχικοποιημένη σε true εφόσον η default επιλογή είναι η γραμμή εντολών να εμφανίζεται.

cl=true

Πριν σχεδιαστεί το κίτρινο ορθογώνιο ελέγχουμε αν η cl είναι true και αν ναι τότε μόνο σχεδιάζεται το κίτρινο ορθογώνιο. Η cl γίνεται false όταν ο χρήστης επιλέξει η γραμμή εντολών να μην είναι ορατή.

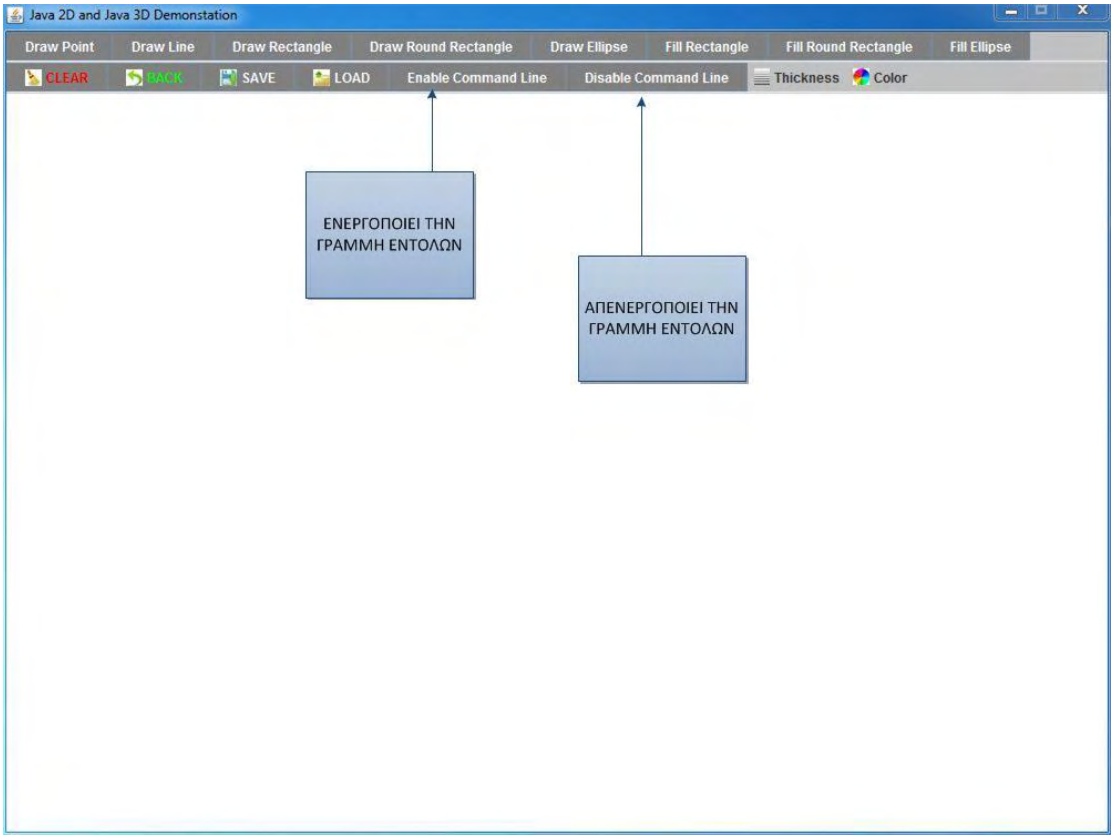
cl=false

Χρησιμοποιούμε και άλλη μία Boolean μεταβλητή, την fr, η οποία είναι αρχικοποιημένη σε false. Την πρώτη φορά που ο χρήστης επιλέγει να μην είναι ορατή η γραμμή εντολών η fr γίνεται true και μόνο τότε ζωγραφίζεται το λευκό ορθογώνιο και η fr επαναφέρεται στην αρχική τιμή της που είναι false. Αυτός ο έλεγχος θα μπορούσε να πραγματοποιηθεί και με την μεταβλητή cl αλλά εφόσον η cl θα ήταν false συνεχώς θα δημιουργούνταν συνεχώς ένα λευκό ορθογώνιο το οποίο μπορεί να κάλυπτε τυχόν σχήματα που θα είχαν δημιουργηθεί σε εκείνον τον χώρο μετά την πρώτη φορά που η γραμμή εντολών απενεργοποιήθηκε.

Για την fr γράφουμε :

```
if(fr == true){
        Graphics2D fr_bi = in_bi.createGraphics();
        fr_bi.setColor(Color.WHITE);
        fr_bi.fillRect(5,60,530,50);
        fr=false;
  }
```

Στην εφαρμογή για να ενεργοποιήσουμε την γραμμή εντολών πατάμε το κουμπί «Enable Command Line» και για να την απενεργοποιήσουμε πατάμε το κουμπί «Disable Command Line».



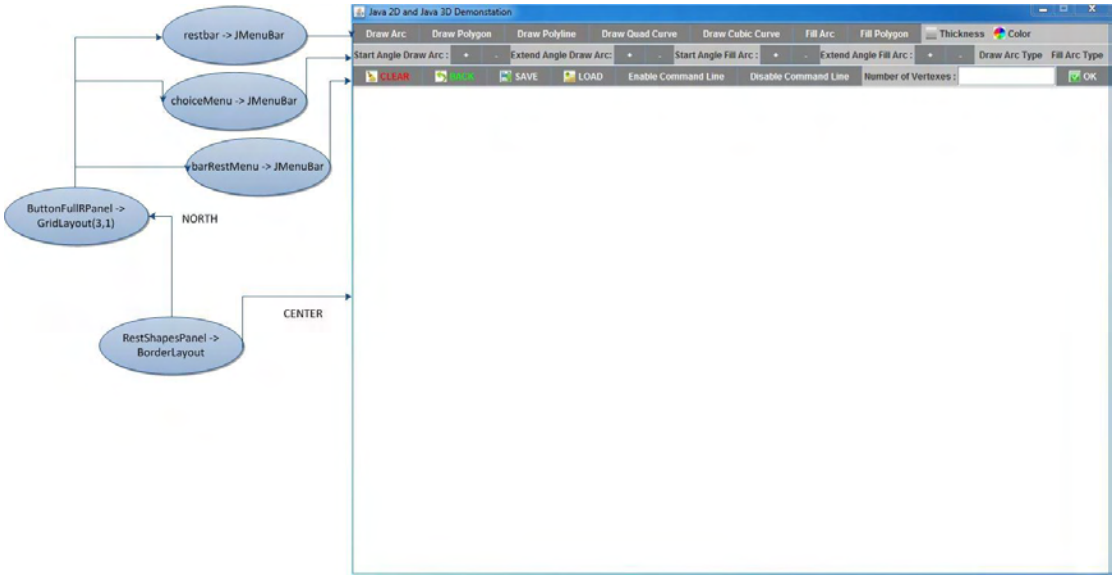
ΕΙΚΟΝΑ 49 ΕΝΕΡΓΟΠΟΙΗΣΗ ΚΑΙ ΑΠΕΝΕΡΓΟΠΟΙΗΣΗ ΓΡΑΜΜΗΣ ΕΝΤΟΛΩΝ

6. Java 2D – 2η Επιλογη (Polygon / Polyline / Arc / Quad Curve / Cubic Curve)

6.1 Παρουσιαση

Η δεύτερη επιλογή της εφαρμογής είναι το αντικείμενο της κλάσης RestShapesPanel. Είναι ένα panel που περιέχει διάφορα Swing components όπως κουμπιά και drop down menus όπως επίσης και μία περιοχή σχεδίασης μέσα στην οποία ο χρήστης μπορεί να σχεδιάσει πολύγωνα, πολυγωνικές γραμμές, τόξα, τετραγωνικές και κυβικές καμπύλες. Μπορεί να σχεδιάσει το περίγραμμα των παραπάνω σχημάτων ή να γεμίσει με χρώμα τα πολύγωνα και τα τόξα. Επίσης μπορεί να αλλάξει τις παραμέτρους των τόξων (γωνίες και είδη) και να σχεδιάσει πολύγωνα και πολυγωνικές γραμμές μέχρι 50 κορυφών. Τέλος παρέχονται οι λειτουργίες που περιγράφηκαν στο προηγούμενο κεφάλαιο : καθαρισμός της περιοχής σχεδίασης και επιστροφή στα εξώφυλλο [5.4.9], η γραμμή εντολών [5.4.11], αποθήκευση και φόρτωση εικόνας [5.4.10].

Το RestShapesPanel έχει custom δομή που δημιουργείται με border Layout και grid Layout.



Εικονα 50 RestShapesPanel layout

6.2 Λειτουργικοτητα

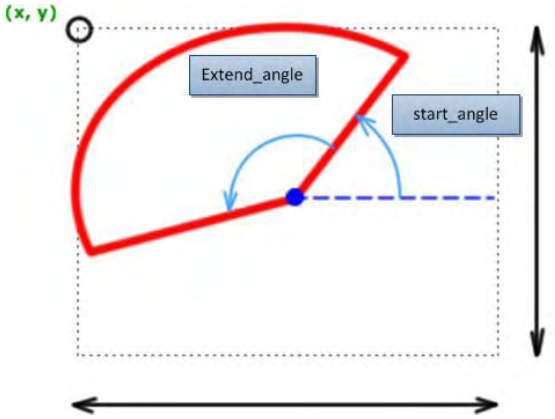
6.2.1 Arcs – Τοξα

Η αφηρημένη κλάση Arc2D είναι υποκλάση της κλάσης Shape και έχει 2 παραγόμενες κλάσεις τις Arc2D.Double, Arc2D.Float. Για να ορίσουμε ένα τόξο χρησιμοποιούμε την εντολή :

Arc2D.Double arc = new Arc2D.Double(rect,start_angle,extend_angle,type)

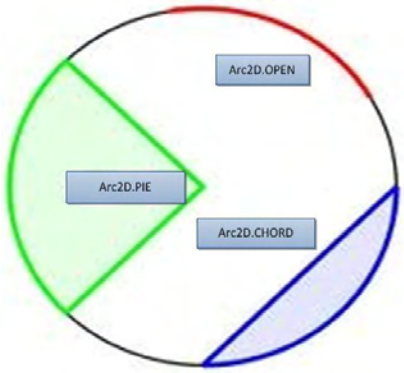
Όπου δημιουργείται ένα τόξο που περικλείεται εντός των ορίων του ορθογωνίου rect. Το τόξο είναι μέρος της έλλειψης που περικλείει το ίδιο ορθογώνιο. Η παράμετρος start_angle περιγράφει την γωνία όπου ξεκινάει το τόξο. Η τιμή της είναι float και δίνεται σε μοίρες. Η παράμετρος extend_angle

είναι η γωνία «ανοίγματος» του τόξου. Το τόξο εκτείνεται από την start_angle και φθάνει μέχρι την start_angle+extend_angle. Η τιμή της extend_angle είναι τύπου float και δίνεται σε μοίρες.



ΕΙΚΟΝΑ 51 START_ANGLE ΚΑΙ EXTEND_ANGLE

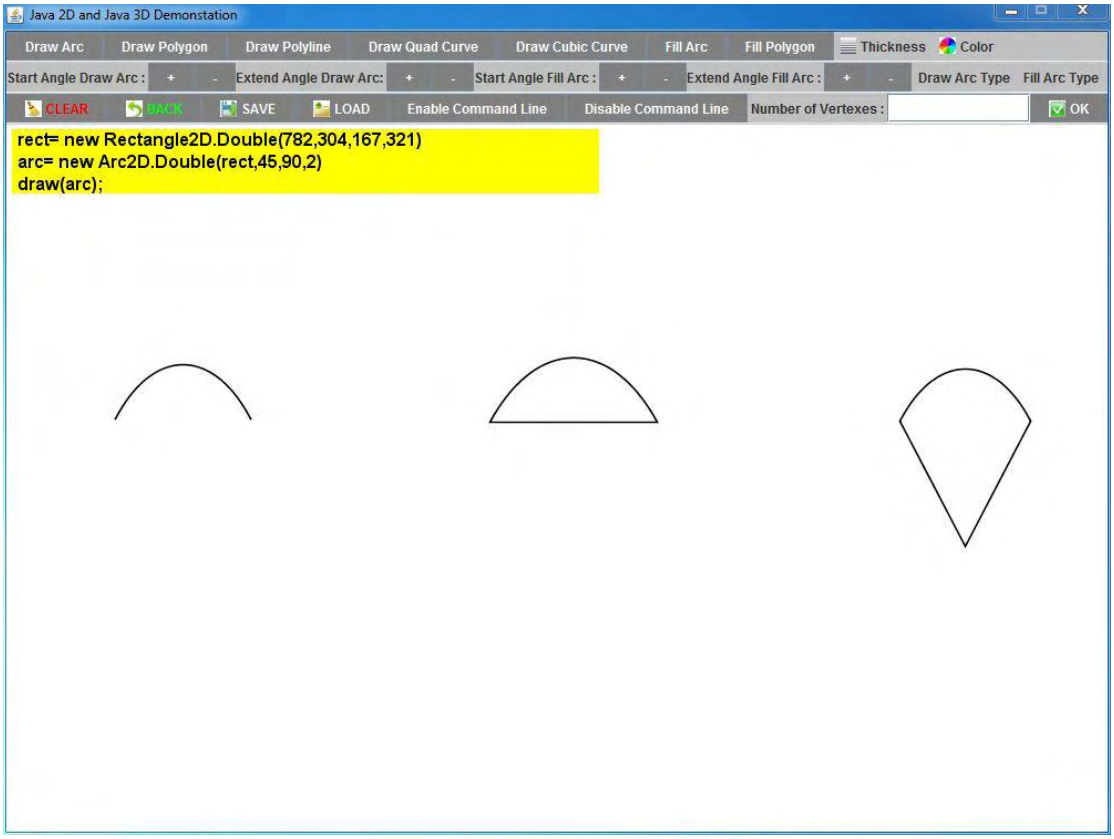
Η παράμετρος type περιγράφει τον τύπο του τόξου και μπορεί να πάρει 3 τιμές : Arc2D.OPEN, Arc2D.CHORD, Arc2D.PIE. Το Arc2D.OPEN αντιστοιχεί στο τόξο, το Arc2D.CHORD στο τόξο μαζί με την αντίστοιχη χορδή και το Arc2D.PIE στο κομμάτι της έλλειψης που αντιστοιχεί το συγκεκριμένο τόξο.



ΕΙΚΟΝΑ 52 CHORD, OPEN, PIE

Για να ζωγραφίσουμε το περίγραμμα του τόξου γράφουμε :

draw(arc)

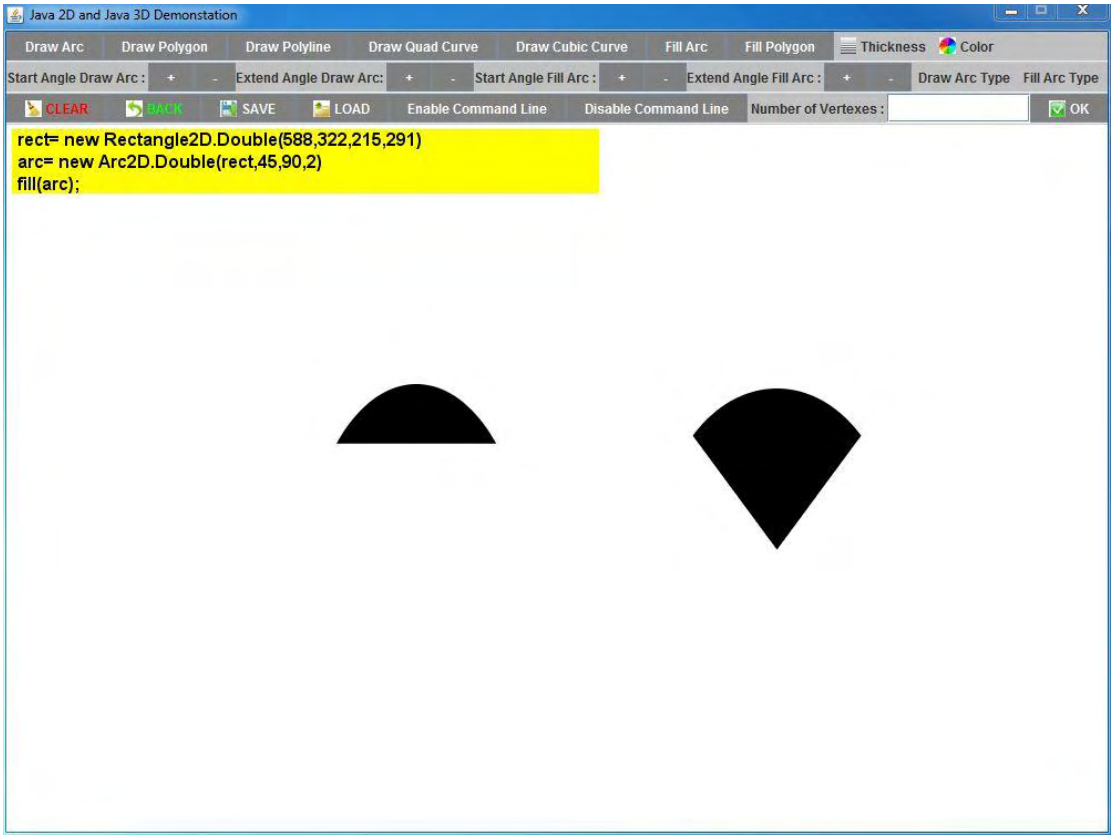


ΕΙΚΟΝΑ 53 ΠΕΡΙΓΡΑΜΜΑΤΑ ΤΥΠΩΝ ΤΟΞΟΥ

Επιλέγουμε το κουμπί «Draw Arc» και κάνουμε click μέσα στην περιοχή σχεδίασης στην θέση όπου θέλουμε να βρίσκεται η πάνω αριστερά γωνία του ορθογωνίου που περικλείει την τόξο και κρατώντας πατημένο το κουμπί του ποντικιού δημιουργούμε το τόξο. Το σχήμα σταθεροποιείται εκεί που σταματάμε να πατάμε το κουμπί του ποντικιού.

Για να γεμίσουμε το τόξο με χρώμα γράφουμε :

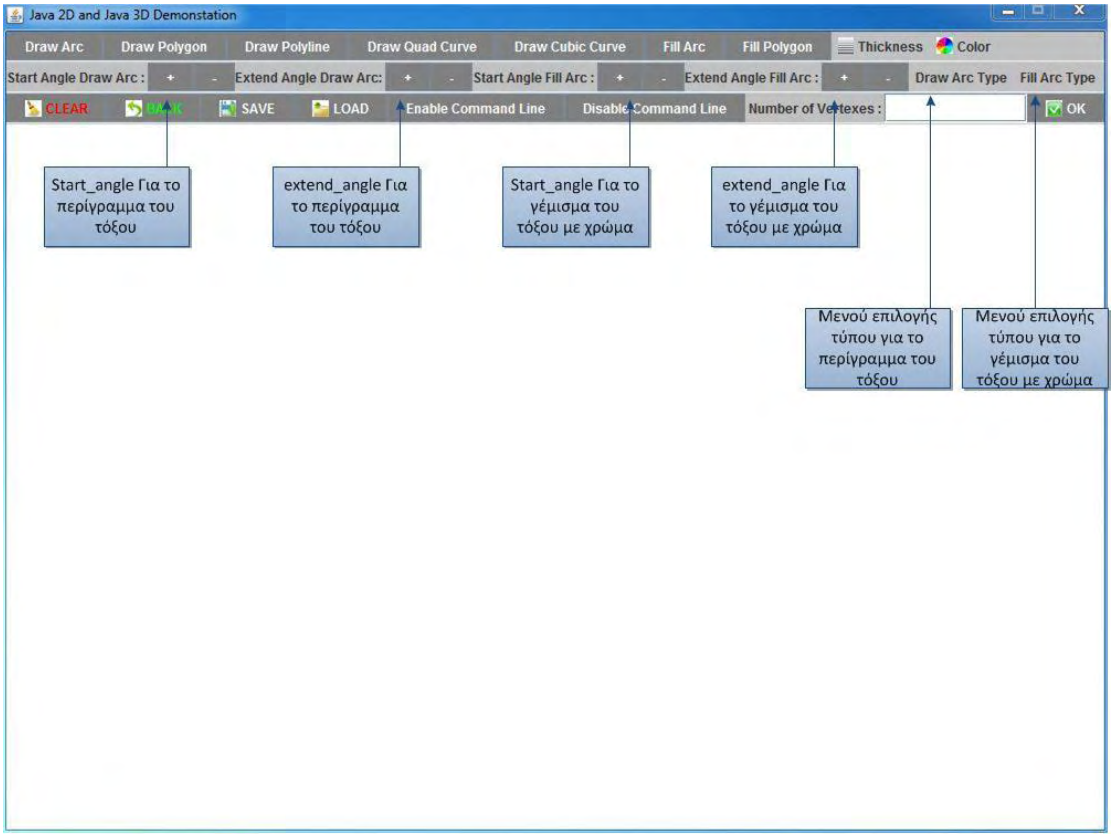
fill(arc)



ΕΙΚΟΝΑ 54 ΤΥΠΟΙ ΤΟΞΩΝ ΓΕΜΙΣΜΕΝΑ ΜΕ ΧΡΩΜΑ

Επιλέγουμε το κουμπί «Fill Arc» και ακολουθούμε την ίδια διαδικασία με αυτή για τη δημιουργία του περιγράμματος. Το πλάτος αλλά και το ύψος του ορθογωνίου που περικλείουν την έλλειψη αλλάζουν δυναμικά και ανάλογα με την θέση που βρίσκεται το ποντίκι άρα αλλάζει δυναμικά και το τόξο.

Η εφαρμογή δίνει επίσης την δυνατότητα στον χρήστη να αλλάξει τις γωνίες των τόξων κατά τη σχεδίαση του περιγράμματος αλλά και του γεμίσματος με χρώμα πατώντας τα κουμπιά «+» και «-» δίπλα από τις αντίστοιχες ετικέτες. Επίσης μπορείς να επιλεχθεί ο τύπος του τόξου όπως φαίνεται στις εικόνες 53, 54. Για το περίγραμμα είναι διαθέσιμοι και οι τρεις τύποι (open, chord, pie)ενώ για το γέμισμα με χρώμα μόνο οι δύο (chord, pie).



ΕΙΚΟΝΑ 55 ΕΠΙΛΟΓΕΣ ΓΙΑ ΓΩΝΙΕΣ ΚΑΙ ΤΥΠΟΥΣ ΤΩΝ ΤΟΞΩΝ

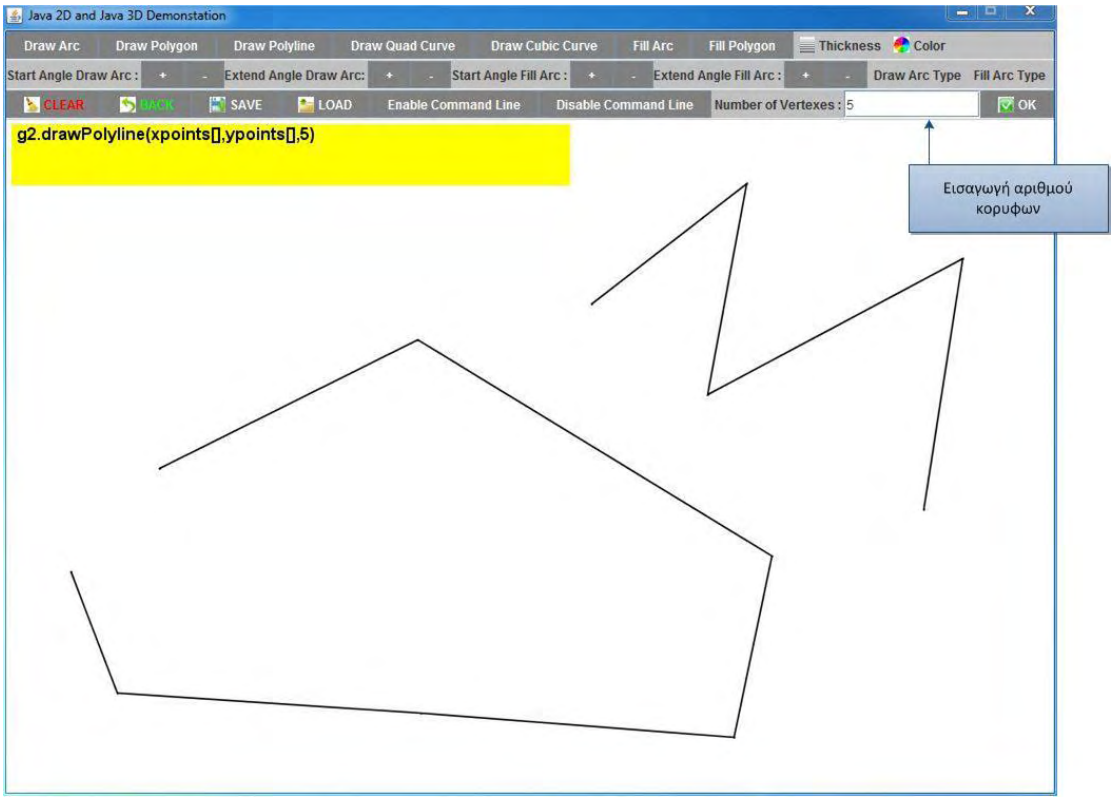
6.2.2 POLYLINES – ΠΟΛΥΓΩΝΙΚΕΣ ΓΡΑΜΜΕΣ

Πολυγωνική γραμμή χαρακτηρίζεται το ευθύγραμμο σχήμα που αποτελείται από πεπερασμένου πλήθους ευθύγραμμα τμήματα τα οποία δεν αποτελούν ευθεία γραμμή. Τα ευθύγραμμα τμήματα λέγονται πλευρές και τα άκρα τους κορυφές. Η πρώτη και η τελευταία κορυφή λέγονται άκρα της πολυγωνικής γραμμής.

Για να σχεδιάσουμε μία πολυγωνική γραμμή χρησιμοποιούμε την μέθοδο :

drawPolyline(int[] xpoints,int[] ypoints,int npoints)

καλούμενη για ένα αντικείμενο της κλάσης Graphics2D. Η μέθοδος ανήκει στην κλάση Graphics αλλά η Graphics2D ως παραγόμενη κλάση μπορεί να την καλέσει. Οι πίνακες xpoints και ypoints περιέχουν τις x και y συντεταγμένες των σημείων που αντιστοιχούν στα άκρα των ευθύγραμμων τμημάτων της πολυγωνικής γραμμής. Ο ακέραιος npoints αντιστοιχεί στο συνολικό αριθμό σημείων. Αν το πρώτο σημείο διαφέρει από το τελευταίο τότε η πολυγωνική γραμμή είναι ανοιχτή εάν δεν διαφέρει τότε είναι κλειστή.

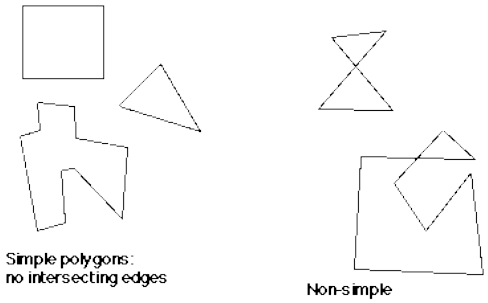


ΕΙΚΟΝΑ 56 ΠΟΛΥΓΩΝΙΚΗ ΓΡΑΜΜΗ

Στην εφαρμογή πατώντας το κουμπί «Draw Polyline» και κάνοντας click με το ποντίκι εμφανίζονται τα σημεία που έχουμε επιλέξει ως κορυφές της πολυγωνικής γραμμής. Μόλις τα σημεία γίνουν ίσα με τον αριθμό κορυφών τότε εμφανίζεται αυτόματα η πολυγωνική γραμμή στην οθόνη. Ο default αριθμός κορυφών είναι 3. Για να αλλάξουμε τον αριθμό των κορυφών γράφουμε τον επιλεγμένο αριθμό στην περιοχή που φαίνεται στην εικόνα 56 και πατάμε το κουμπί «OK» για να εισάγουμε τον αριθμό.

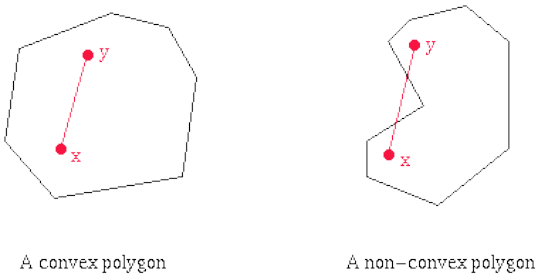
6.2.3 POLYGONS – ΠΟΛΥΓΩΝΑ

Πολύγωνο είναι κάθε κλειστή πολυγωνική γραμμή. Αν n ο αριθμός των πλευρών του πολυγώνου πρέπει n ≥ 3. Ένα πολύγωνο λέγεται απλό όταν οι πλευρές του δεν τέμνονται.



ΕΙΚΟΝΑ 57 ΑΠΛΑ ΚΑΙ ΟΧΙ ΑΠΛΑ ΠΟΛΥΓΩΝΑ

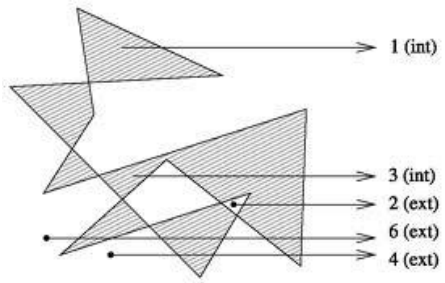
Κυρτά (convex) πολύγωνα ονομάζονται τα πολύγωνα που για κάθε ζευγάρι σημείων του πολυγώνου το ευθύγραμμο τμήμα που τα ενώνει βρίσκεται ολόκληρο εντός του πολυγώνου. Αν για κάποιο ζευγάρι στοιχείων το ευθύγραμμο τμήμα δεν είναι ολόκληρο μέσα στο πολύγωνο τότε αυτό το πολύγωνο ονομάζεται κοίλο (concave).



ΕΙΚΟΝΑ 58 CONVEX ΚΑΙ CONCAVE ΠΟΛΥΓΩΝΑ

Αλγόριθμος γεμίσματος πολυγώνου – Odd parity rule

Ο κανόνας odd parity καθορίζει αν ένα σημείο είναι εντός η εκτός του πολυγώνου. Αν αρχίσουμε να κινούμαστε κατά μήκος μίας γραμμής από ένα εσωτερικό σημείο του πολυγώνου προς μία κατεύθυνση τότε κάποια στιγμή θα συναντήσουμε πλευρά του πολυγώνου. Αν το πολύγωνο είναι concave τότε μπορεί να συναντήσουμε και άλλες πλευρές του πολυγώνου. Εφόσον ξεκινήσαμε από εσωτερικό σημείο όταν συναντήσουμε μία πλευρά τότε βρισκόμαστε εκτός του σχήματος αν συναντήσουμε και άλλη πλευρά θα βρισκόμαστε εντός του σχήματος μέχρι την επόμενη πλευρά που θα συναντήσουμε. Άρα κάθε φορά που συναντούμε μία γραμμή βρισκόμαστε σε διαφορετικό μέρος (εντός ή εκτός) του σχήματος. Αν ο αριθμός των πλευρών που συναντάμε είναι μονός (odd) τότε το σημείο είναι εντός του σχήματος, αν είναι ζυγός (even) τότε είναι εκτός. Η τεχνική αυτή δεν είναι κατάλληλη για υλοποίηση γιατί θα πρέπει να εφαρμοστεί σε κάθε pixel με αποτέλεσμα τεράστιο κόστος υπολογισμού.



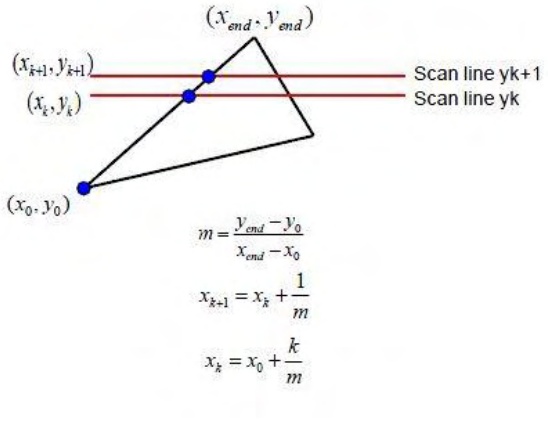
ΕΙΚΟΝΑ 59 ΚΑΝΟΝΑΣ ODD PARITY – INT : ΕΣΩΤΕΡΙΚΟ ΣΗΜΕΙΟ, EXT : ΕΞΩΤΕΡΙΚΟ ΣΗΜΕΙΟ

Αλγόριθμος γεμίσματος πολυγώνου – Scan line technique (Τεχνική με γραμμές σάρωσης)

Η ιδέα αυτής της τεχνικής βασίζεται στη σάρωση του πλέγματος των pixel κατά γραμμές.

- Βρες τα σημεία που τέμνονται οι scan lines με τις πλευρές του πολυγώνου
- Ταξινόμησε τα σημεία τομής ξεκινώντας από αυτό με το μικρότερο x
- Ζωγράφισε τα pixel μεταξύ ζευγαριών σημείων τομής που είναι εσωτερικά του πολυγώνου, χρησιμοποιώντας τον κανόνα odd parity για να καθορίσεις αν ένα από αυτά τα pixel που ανήκουν στην περιοχή αυτή είναι εντός ή εκτός πολυγώνου.

Ειδικές περιπτώσεις : κάθετες, οριζόντιες γραμμές, λεπτά πολύγωνα. Μπορούμε να χρησιμοποιήσουμε τις παρακάτω παραδοχές για να αυξήσουμε την απόδοση του αλγορίθμου: οι πλευρές που τέμνουν μία scan line είναι κυρίως ίδιες με εκείνες που τέμνουν την προηγούμενη scan line, οι τιμές του x για μία scan line είναι πολύ κοντά στις τιμές του x για την προηγούμενη scan line.



ΕΙΚΟΝΑ 60 SCAN LINE ALGORITHM

Για την υλοποίηση του αλγορίθμου χρησιμοποιούμε έναν πίνακα που περιέχει τις scan lines, όπου το κάθε κελί του πίνακα αποθηκεύει μία συνδεδεμένη λίστα που περιέχει τις πλευρές του πολυγώνου που τέμνει η scan line που αντιστοιχεί σε αυτό το κελί. Ο πίνακας ονομάζεται active edge table. Η λίστα της τρέχουσας scan line ονομάζεται active edge list. Η κάθε πλευρά αποτελείται από 3 μεταβλητές : maxY (μέγιστη τιμή του y), current (η τιμή x του σημείου με την μικρότερη τιμή y), xIncr (ίση με 1/m – m σύμφωνα με την εικόνα 59).

1. line = 0
2. While(line<height)
3. Πρόσθεσε τις πλευρές στην active edge list από το active edge table Ξεκινώντας από τη line
4. Αφαίρεσε τα pixel που τελειώνουν στην line
5. Ζωγράφισε τα pixel
6. Αύξησε τις τιμές των x των πλευρών της active edge list
7. line = line + 1

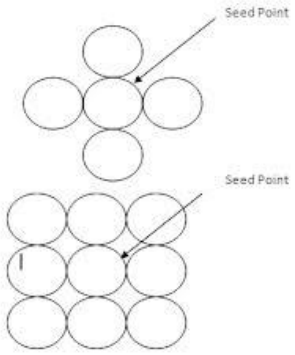
Αλγόριθμος γεμίσματος πολυγώνου – Seed Fill

Στην μέθοδο Seed Fill ένα συγκεκριμένο seed point (pixel έναρξης) διαλέγεται και αρχίζουμε να ζωγραφίζουμε τα πάνω και κάτω pixel μέχρι να φτάσουμε στο περίγραμμα. Η μέθοδος αυτή έχει 2 τύπους : Boundary fill και Flood fill.

- Boundary Fill
  Στο boundary fill αρχικά ζωγραφίζεται το περίγραμμα και το seed point εσωτερικά του σχήματος. Ελέγχουμε τα γειτονικά pixel του seed point αν έχουν το ίδιο χρώμα με το περίγραμμα. Αν έχουν το ίδιο χρώμα προχωρούμε στα επόμενα pixel αν όχι τα βάφουμε στο χρώμα του περιγράμματος και έπειτα προχωρούμε στα επόμενα pixel.
- Flood Fill

Υπάρχουν περιπτώσεις που το χρώμα περιγράμματος είναι διαφορετικό από το χρώμα που θέλουμε να βάψουμε το εσωτερικό του σχήματος. Ο Αλγόριθμος Flood Fill χρησιμοποιείται για τέτοιες περιπτώσεις. Ο Flood Fill είναι ίδιος με τον αλγόριθμο Boundary Fill μόνο που αντί να ζωγραφίζει τα pixel στο χρώμα του περιγράμματος τους αναθέτει κάποιο άλλο χρώμα.

Για να ορίσουμε ποιο είναι το γειτονικό pixel χρησιμοποιούνται περιοχές από pixel που μπορεί να αποτελούνται από 4 ή 8 pixel.

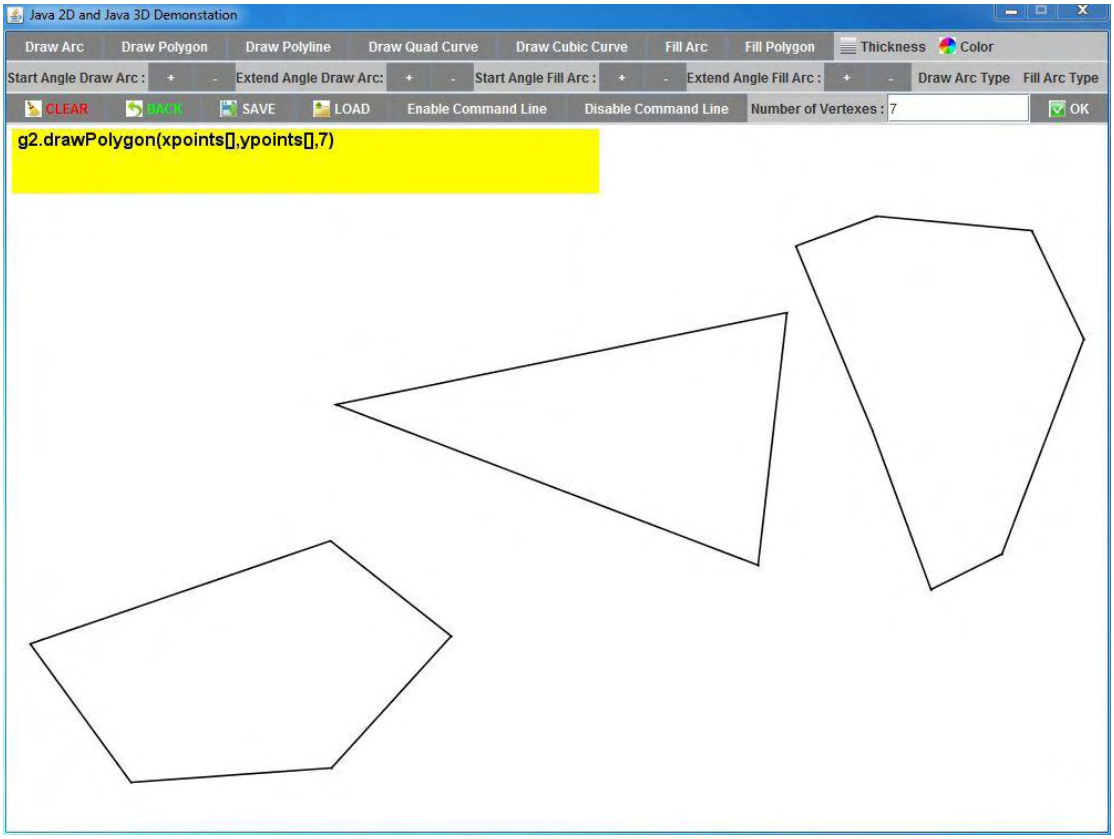


ΕΙΚΟΝΑ 61 SEED PIXEL ΚΑΙ ΟΙ ΠΕΡΙΟΧΕΣ ΤΩΝ ΓΕΙΤΟΝΙΚΩΝ PIXEL

Για να σχεδιάσουμε το περίγραμμα ενός πολυγώνου χρησιμοποιούμε την μέθοδο :

drawPolygon(int[] xpoints,int[] ypoints,int npoints)

καλούμενη για ένα αντικείμενο της κλάσης Graphics2D. Η μέθοδος αυτή όπως και η draw Polyline() ανήκει στην κλάση Graphics αλλά η Graphics2D ως παραγόμενη κλάση μπορεί να την καλέσει. Οι παράμετροι της μεθόδους είναι ίδιες με αυτές της draw Polyline(). Η διαφορά είναι ότι η μέθοδος αυτή δημιουργεί το πολύγωνο κλείνοντας αυτόματα το σχήμα ενώνοντας το πρώτο με το τελευταίο σημείο.



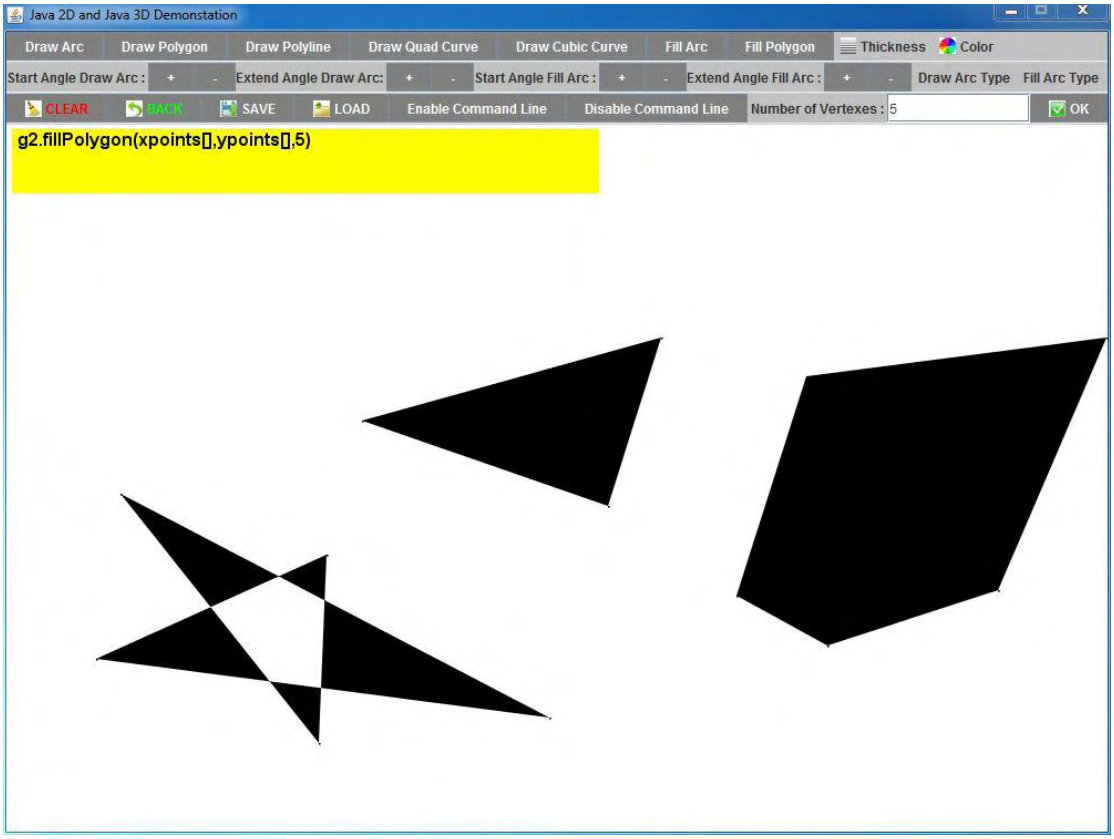
ΕΙΚΟΝΑ 62 ΠΕΡΙΓΡΑΜΜΑ ΠΟΛΥΓΩΝΟΥ

Στην εφαρμογή πατώντας το κουμπί «Draw Polygon» και κάνοντας click με το ποντίκι εμφανίζονται τα σημεία που έχουμε επιλέξει ως κορυφές του πολυγώνου. Μόλις τα σημεία γίνουν ίσα με τον προεπιλεγμένο αριθμό κορυφών τότε εμφανίζεται αυτόματα το πολύγωνο στην οθόνη. Όπως και για την πολυγωνική γραμμή ο default αριθμός κορυφών είναι 3 και για να αλλάξουμε τον αριθμό αυτό ακολουθούμε τη διαδικασία που αναφέρθηκε στην προηγούμενη παράγραφο [6.2.2].

Για να γεμίσουμε ένα πολύγωνο με χρώμα γράφουμε τη μέθοδο :

fillPolygon(int[] xpoints,int[] ypoints,int npoints)

Η fill Polygon() έχει ίδιες παραμέτρους με αυτές της draw Polygon() και draw Polyline(). Η διαφορά της με την draw Polygon() είναι ότι αφού σχεδιάσει το περίγραμμα γεμίζει το σχήμα με χρώμα.



ΕΙΚΟΝΑ 63 ΠΟΛΥΓΩΝΟ ΓΕΜΙΣΜΕΝΟ ΜΕ ΧΡΩΜΑ

Στην εφαρμογή πατώντας το κουμπί «Fill Polygon» και ακολουθώντας την ίδια διαδικασία με αυτή για το περίγραμμα πολυγώνου και πολυγωνικής γραμμής δημιουργούμε το πολύγωνο γεμισμένο με χρώμα.

6.2.4 QUADRATIC CURVE – ΤΕΤΡΑΓΩΝΙΚΗ ΚΑΜΠΥΛΗ

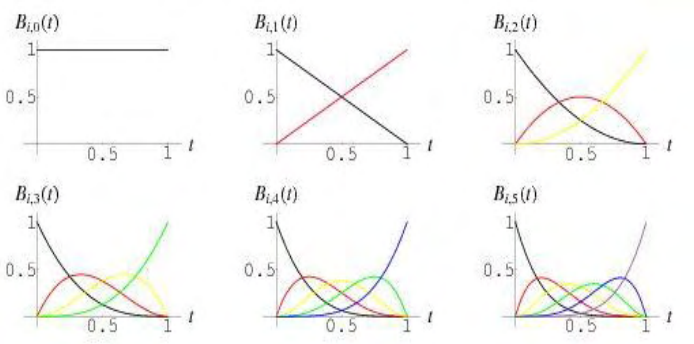
Bernstein Πολυώνυμα Βαθμού n

Το i Bernstein πολυώνυμο βαθμού n με n ∈ {0, 1, 2,…., n} ορίζεται ως :

$$B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i},$$

Με t ∈ [0,1].

Μερικά από τα πρώτα πολυώνυμα φαίνονται στην εικόνα 64.



ΕΙΚΟΝΑ 64 ΓΡΑΦΙΚΕΣ ΠΑΡΑΣΤΑΣΕΙΣ

$$B_{0,0}(t) = 1$$
$$B_{0,1}(t) = 1 - t$$
$$B_{1,1}(t) = t$$
$$B_{0,2}(t) = (1-t)^2$$
$$B_{1,2}(t) = 2(1-t)t$$
$$B_{2,2}(t) = t^2$$
$$B_{0,3}(t) = (1-t)^3$$
$$B_{1,3}(t) = 3(1-t)^2 t$$
$$B_{2,3}(t) = 3(1-t)t^2$$
$$B_{3,3}(t) = t^3.$$

ΕΙΚΟΝΑ 65 BERNSTEIN ΠΟΛΥΩΝΥΜΑ

Τα Bernstein πολυώνυμα έχουν 2 πολύ χρήσιμες ιδιότητες:

1. Σε κάθε σημείο του [0,1] διαστήματος όλα τα πολυώνυμα αθροίζονται στο 1.

$$\sum_{i=0}^{n} B_{i,n}(t) = 1,$$

Για t ∈ [0,1].

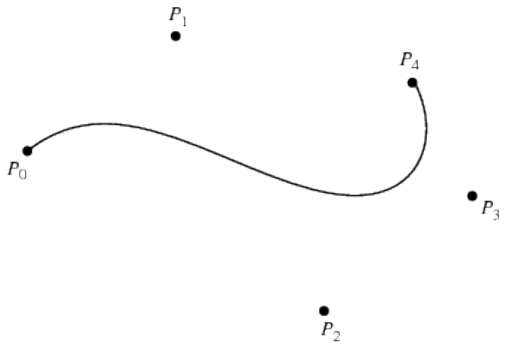
2. Η τιμή του πολυωνύμου μέσα στο διάστημα [0,1] θα είναι από 0 μέχρι 1.
$B_i^{(n)}(t)$ ∈ [0,1] για όλα τα t ∈ [0,1].

Καμπύλη Bezier

Οι καμπύλες Bezier χρησιμοποιούν Bernstein πολυώνυμα βαθμού n για να προσεγγίσουν n+1 σημεία ελέγχου P0, . . . , Pn ∈ R^p. Για τα γραφικά η/υ μας ενδιαφέρουν μόνο οι περιπτώσεις για p=2 (επίπεδο) και p=3 (τρισδιάστατος χώρος). Τα σημεία ελέγχου επίσης ονομάζονται Bezier points. Η καμπύλη

$$x(t) = \sum_{i=0}^{n} P_i \, B_i^{(n)}(t)$$

Με t ∈ [0,1] που ορίζεται από τα P0, . . . ,Pn ∈ R^p σημεία ονομάζεται Bezier καμπύλη βαθμού n.



ΕΙΚΟΝΑ 66 BEZIER ΚΑΜΠΥΛΗ

Ιδιότητες της Bezier καμπύλης :

Η Bezier καμπύλη παρεμβάλει το πρώτο και το τελευταίο σημείο αυτό σημαίνει ότι : x(1) = Pn και x(0) = P0 . Γενικά η καμπύλη δεν περνάει από τα άλλα σημεία ελέγχου.

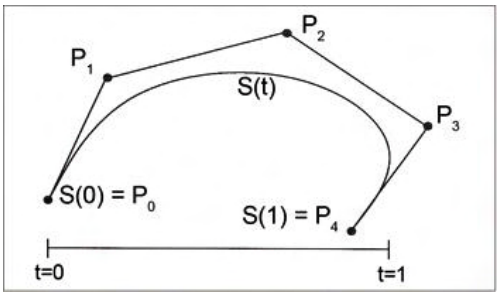
Η εφαπτόμενη στο πρώτο και στο τελευταίο σημείο υπολογίζεται ως :

$$x(0) = n \cdot (P1 - P0),$$

$$x(1) = n \cdot (Pn - Pn-1)$$

Αυτό σημαίνει ότι η εφαπτομένη στο πρώτο σημείο P0 δείχνει στην διεύθυνση του P1, και η εφαπτομένη στο τελευταίο σημείο Pn δείχνει στο Pn-1.

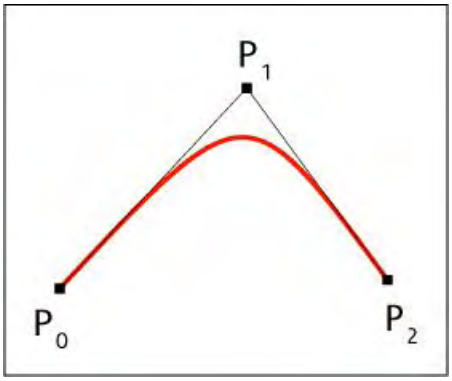
Ενώνοντας όλα τα σημεία ελέγχου φτιάχνουμε το Bezier πολύγωνο. Το convex hull (κυρτό περίβλημα – το κυρτό περίβλημα συνόλου σημείων στο επίπεδο είναι το κυρτό πολύγωνο με το ελάχιστο εμβαδό που περιλαμβάνει όλα τα δεδομένα σημεία) του Bezier πολυγώνου περιέχει πλήρως την καμπύλη Bezier.



Εικονα 67 Bezier πολυγωνο

Επίσης οι Bezier καμπύλες είναι συμμετρικές , δηλαδή τα σημεία έλεγχου P0, . . . ,Pn και Pn, . . . ,P0 δημιουργούν την ίδια καμπύλη η οποία όμως περνάει από τα σημεία με την αντίθετη κατεύθυνση.

Τετραγωνική καμπύλη (Bezier καμπύλη) είναι η Bezier καμπύλη βαθμού 2 (p=2) η οποία ορίζεται από 3 σημεία ελέγχου P0, P1, P2. Η καμπύλη ξεκινά από το P0 και τελειώνει στο P2.



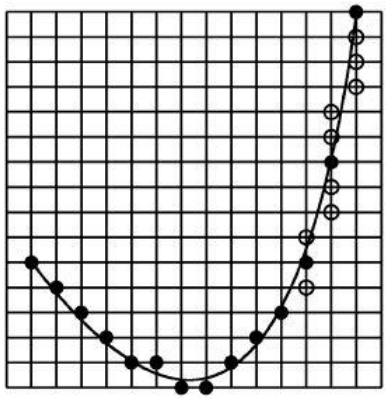
Εικονα 68 quadratic curve - τετραγωνικη καμπυλη

Αλγόριθμος Σχεδίασης Καμπύλων – Midpoint Algorithm (Αλγόριθμος Μέσου Σημείου)

Ο αλγόριθμος Μέσου σημείου μπορεί να γενικευθεί για τις καμπύλες. Για τον σχεδιασμό αυθαίρετων καμπυλών ή συνεχών καμπυλών ο αλγόριθμος βασίζεται σε τμηματικές προσεγγίσεις

της καμπύλης από σύντομες γραμμές. Για να σχεδιάσουμε την y = f(x) δεν είναι αρκετό να προχωρούμε κατά χ και να βρίσκουμε το y. Σε περιπτώσεις γραμμών με κλίση μεγαλύτερη του 1 ο αλγόριθμος αυτός δημιουργεί γραμμές με μεγάλα κενά. Για να αντιμετωπιστεί αυτό το πρόβλημα ανταλλάσουμε τους ρόλους των x με των y, προχωρώντας κατά y και βρίσκοντας το x. Αυτό σημαίνει ότι ζωγραφίζεται η αντίστροφη της f(x) στον y άξονα. Οι τυχαίες εξισώσεις δεν έχουν ούτε σταθερή κλίση ούτε η κλίση μπορεί να υπολογιστεί εύκολα. Επίσης δεν είναι εύκολο να υπολογιστεί και η αντίστροφη εξίσωση. Για αυτό το λόγο ζωγραφίζοντας τυχαίες καμπύλες προχωρούμε κατά x βρίσκοντας το y και ζωγραφίζοντας τα αντίστοιχα pixel, ακολούθως ζωγραφίζουμε την γραμμή που ενώνει δύο pixel με γειτονικές τιμές x, με τον αλγόριθμο Midpoint για γραμμές.

```
1. int yRound1, yRound2
2. yRound1 = round( f(x0) )
3. for ( int x=x0 ; x<x1 ; x++)
4.         yRound2 = round( f(x+1) )
5.         drawLine( x, yRound1, x+1, yRound2)
6.         yRound1 = yRound2
```



ΕΙΚΟΝΑ 69 ΑΛΓΟΡΙΘΜΟΣ ΓΙΑ ΣΧΕΔΙΑΣΗ ΚΑΜΠΥΛΩΝ

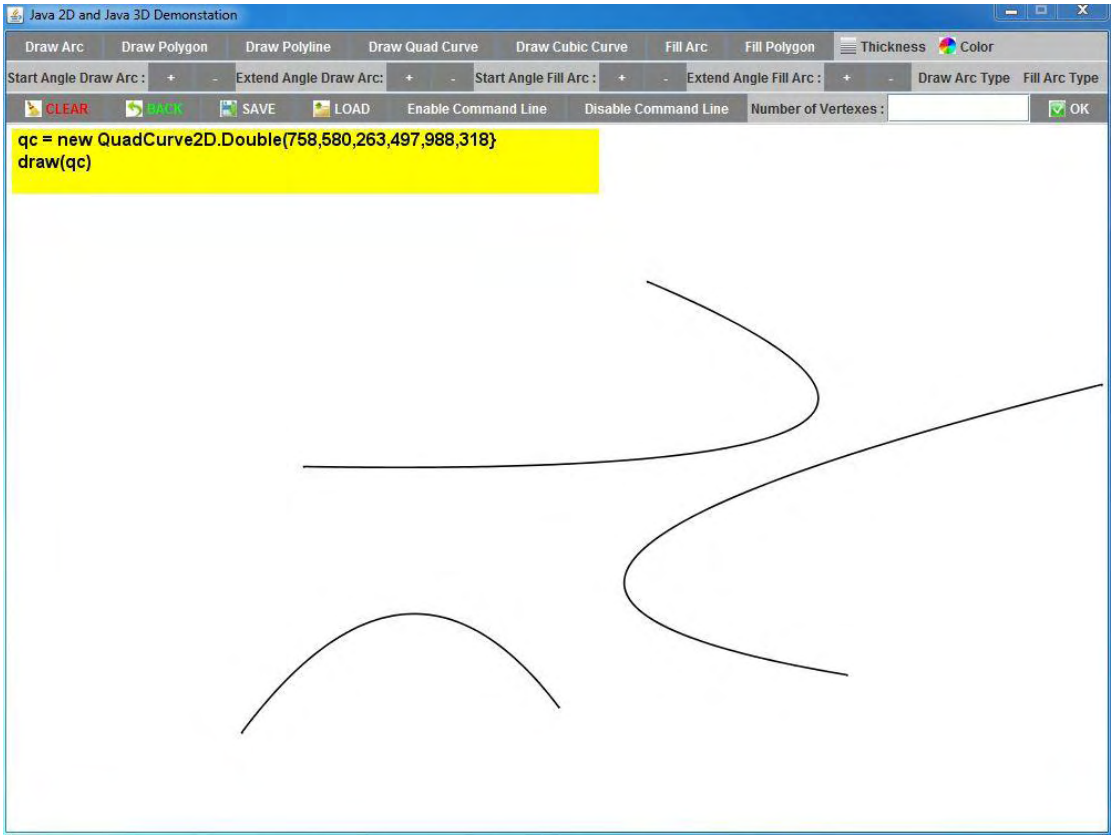
Στην εικόνα 69 ζωγραφίζεται μία συνεχής εξίσωση y = f(x) στο διάστημα [x0, x1] με x0, x1 ∈ N. Οι ζωγραφισμένοι με μαύρο χρώμα κύκλοι αντιστοιχούν στα (x, round(f(x))) pixel για ακέραιες τιμές στον άξονα των x. Οι κενοί κύκλοι αντιστοιχούν στα pixel που ζωγραφίζονται όταν δύο από τα pixel που έχουν υπολογιστεί με γειτονικά x ενώνονται με μία γραμμή.

Ζωγραφίζοντας με αυτόν τον τρόπο την καμπύλη δεν οδηγεί πάντα στην επιλογή των ίδιων pixel με αυτά που θα επιλέγαμε αν ζωγραφίζαμε τα πιο κοντινά pixel στην καμπύλη, όμως η διαφορά είναι το πολύ ένα pixel.

Η αφηρημένη κλάση Quad Curve2D είναι υποκλάση της κλάσης Shape και έχει 2 παραγόμενες κλάσεις τις Quad Curve2D.Double, Quad Curve2D.Float. Για να ορίσουμε μία τετραγωνική καμπύλη χρησιμοποιούμε την εντολή :

QuadCurve2D.Double qc =new QuadCurve2D.Double(x1,y1,ctrlx,ctrly,x2,y2)

Όπου x1, y1 το σημείο έναρξης της καμπύλης x2,y2 το σημείο τερματισμού της καμπύλης και ctrlx, ctrly το σημείο ελέγχου της καμπύλης.

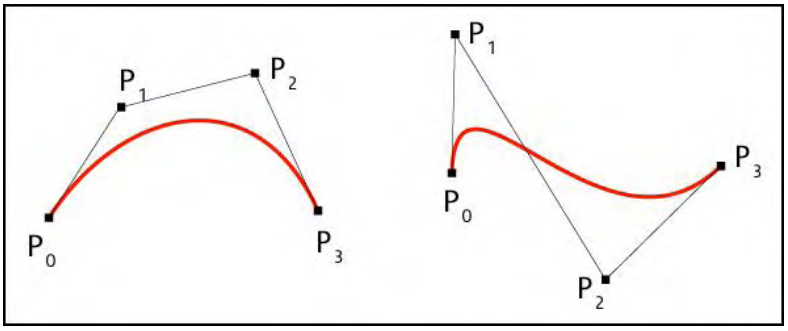


ΕΙΚΟΝΑ 70 ΤΕΤΡΑΓΩΝΙΚΗ ΚΑΜΠΥΛΗ

Στην εφαρμογή πατώντας το κουμπί «Draw Quad Curve» και κάνοντας 2 click με το ποντίκι εμφανίζονται τα σημεία έναρξης και τερματισμού της καμπύλης. Στο επόμενο click αντιστοιχεί το σημείο ελέγχου της καμπύλης το οποίο αλλάζει σύμφωνα με την κατεύθυνση του ποντικιού. Κατά τη διάρκεια που έχουμε πατημένο το κουμπί του ποντικιού σχεδιάζονται οι καμπύλες που αντιστοιχούν στο τρέχον σημείο ελέγχου. Όταν σταματήσουμε να πατάμε το κουμπί του ποντικιού το σχήμα σταθεροποιείται στο τελευταίο σημείο ελέγχου.

6.2.5 CUBIC CURVE – ΚΥΒΙΚΗ ΚΑΜΠΥΛΗ

Κυβική καμπύλη είναι η Bezier καμπύλη βαθμού 3 (p=3) η οποία ορίζεται από 4 σημεία ελέγχου P0, P1, P2, P3. Η καμπύλη ξεκινά στο P0 και τελειώνει στο P3. Η γραμμή P0P1 είναι η εφαπτομένη της καμπύλης στο P0 και η γραμμή P2P3 είναι η εφαπτομένη της καμπύλης στο P3. Η καμπύλη δεν περνά από τα P1,P2. Η απόσταση μεταξύ του P0 και του P1 καθορίζει πόσο «μακριά» η καμπύλη εκτείνεται στην κατεύθυνση του P1 πριν «στρίψει» κατά το P3.

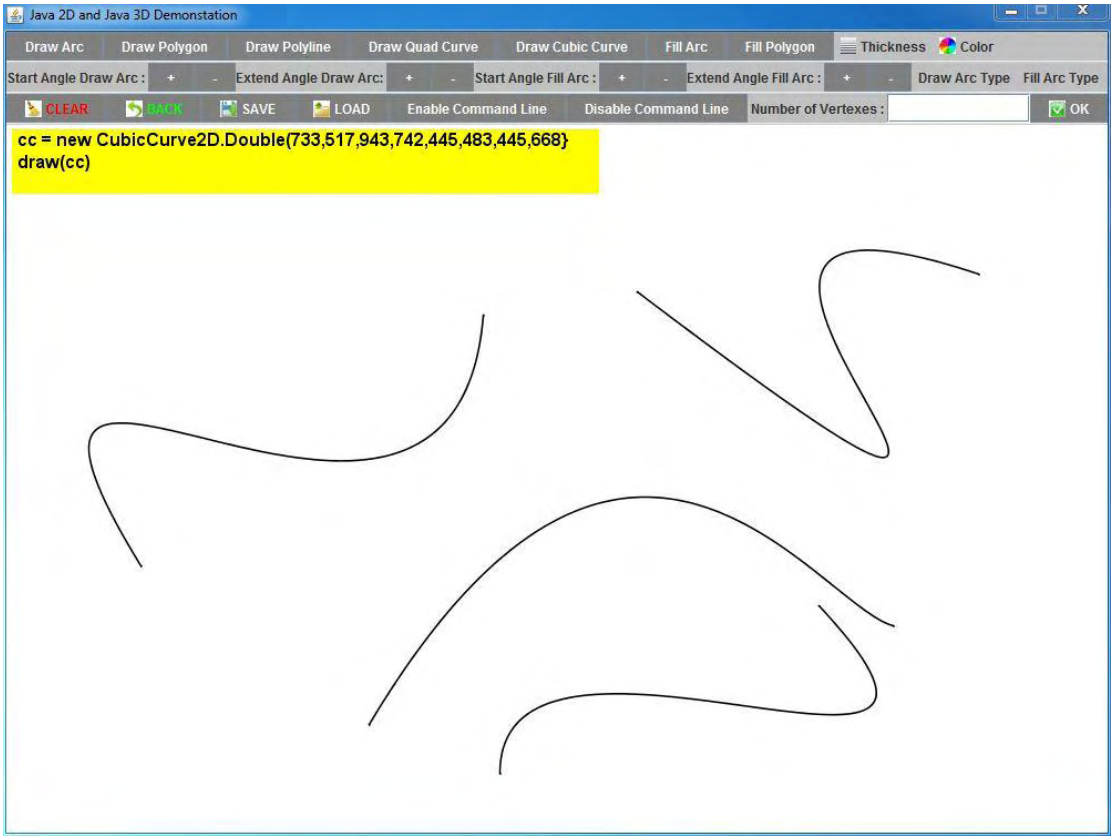


ΕΙΚΟΝΑ 71 CUBIC CURVE - ΚΥΒΙΚΗ ΚΑΜΠΥΛΗ

Η αφηρημένη κλάση Cubic Curve2D είναι υποκλάση της κλάσης Shape και έχει 2 παραγόμενες κλάσεις τις Cubic Curve2D.Double, Cubic Curve2D.Float. Για να ορίσουμε μία κυβική καμπύλη χρησιμοποιούμε την εντολή :

CubicCurve2D.Double cc = new CubicCurve2D.Double(x1,y1,ctrlx1,ctrly1,ctrlx2,ctrly2,x2,y2)

Όπου x1, y1 το σημείο έναρξης της καμπύλης x2,y2 το σημείο τερματισμού της καμπύλης και ctrlx1, ctrly1, ctrlx2, ctrly2 τα σημεία ελέγχου της καμπύλης.



ΕΙΚΟΝΑ 72 ΚΥΒΙΚΗ ΚΑΜΠΥΛΗ

Στην εφαρμογή πατώντας το κουμπί «Draw Cubic Curve» και κάνοντας 2 click με το ποντίκι εμφανίζονται τα σημεία έναρξης και τερματισμού της καμπύλης. Στο επόμενο click αντιστοιχεί το πρώτο σημείο ελέγχου της καμπύλης το οποίο αλλάζει σύμφωνα με την κατεύθυνση του ποντικιού.

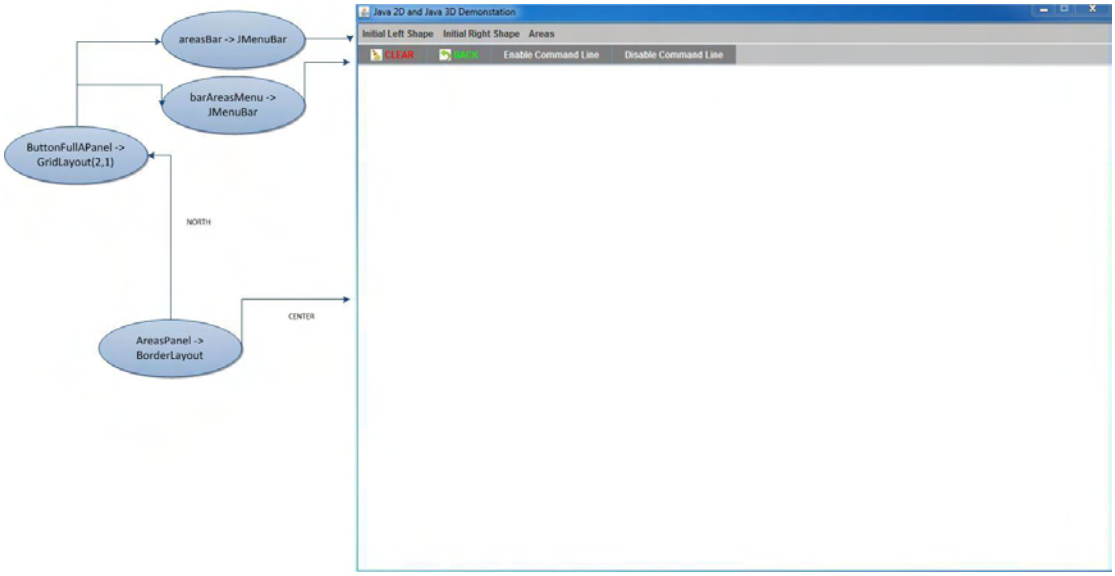
Κατά τη διάρκεια που έχουμε πατημένο το κουμπί του ποντικιού σχεδιάζονται οι καμπύλες που αντιστοιχούν στο τρέχον σημείο ελέγχου. Όταν σταματήσουμε να πατάμε το κουμπί του ποντικιού το σχήμα σταθεροποιείται έχοντας σχηματίσει τα άκρα της καμπύλης και το πρώτο σημείο ελέγχου. Στο επόμενο click αντιστοιχεί το δεύτερο σημείο ελέγχου της καμπύλης το οποίο αλλάζει σύμφωνα πάλι με την κατεύθυνση του ποντικιού. Κατά τη διάρκεια που έχουμε πατημένο το κουμπί του ποντικιού σχεδιάζονται οι καμπύλες που αντιστοιχούν στο τρέχον δεύτερο σημείο ελέγχου, δεδομένου του ήδη σχεδιασμένου πρώτου σημείου ελέγχου. Όταν σταματήσουμε να πατάμε το κουμπί του ποντικιού το σχήμα σταθεροποιείται έχοντας σχηματίσει και το δεύτερο σημείο ελέγχου. Η καμπύλη έχει πάρει το τελικό της σχήμα.

7. Java 2D – 3η Επιλογη (Areas)

7.1 Παρουσιαση

Η τρίτη επιλογή της εφαρμογής είναι το αντικείμενο της κλάσης AreasPanel. Είναι ένα panel που περιέχει διάφορα Swing components όπως κουμπιά και drop down menus όπως επίσης και μία περιοχή σχεδίασης. Ο χρήστης μπορεί να επιλέξει ανάμεσα από τα σχήματα : κύκλο, ορθογώνιο και τόξο να εμφανιστούν στην περιοχή σχεδίασης ορισμένα σαν περιοχές και έπειτα να σχεδιαστούν πατώντας ένα κουμπί η τομή, η ένωση, η διαφορά και το αποκλειστικό ή (XOR) τους. Τέλος παρέχονται οι λειτουργίες που περιγράφηκαν στο πέμπτο κεφάλαιο : καθαρισμός της περιοχής σχεδίασης και επιστροφή στα εξώφυλλο [5.4.9] και η γραμμή εντολών [5.4.11].

Το AreasPanel έχει custom δομή που δημιουργείται με border Layout και grid Layout.



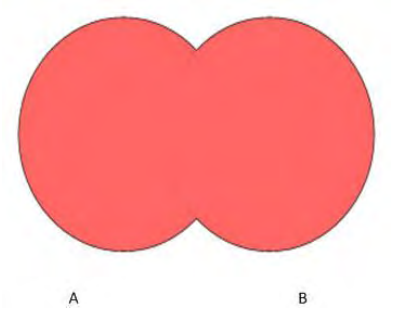
Εικονα 73 AreasPanel layout

7.2 Λειτουργικοτητα

7.2.1 Union – Ένωση

Μερικά από τα σχήματα για τα οποία έχουμε μιλήσει στα προηγούμενα κεφάλαια όπως η έλλειψη και το πολύγωνο μπορούμε να πούμε ότι ορίζουν περιοχές. Εν αντιθέσει με την γραμμή και τις τετραγωνικές και κυβικές καμπύλες οι περιοχές μπορούν να γεμιστούν με χρώμα ή επίσης και με κάποια υφή (texture). Για να ορίσουμε μία περιοχή μπορούμε να χρησιμοποιήσουμε ένα πολύγωνο ή το περίγραμμα κάποιου σχήματος, πολλές φορές όμως για να ορίσουμε πολύπλοκες περιοχές δημιουργούμε μίξεις ήδη καθορισμένων περιοχών χρησιμοποιώντας την θεωρία των συνόλων (Set Theory).

Μία από τις σημαντικότερες πράξεις στην θεωρία συνόλων είναι η Ένωση η οποία συνενώνει δύο περιοχές σε μία μεγαλύτερη ενιαία περιοχή. Έστω δύο περιοχές στο σχήμα του κύκλου η οποίες έχουν κοινά σημεία. Η ένωση τους A U B φαίνεται στην εικόνα 74. Η ένωση εμφανίζεται με ροζ χρώμα.



ΕΙΚΟΝΑ 74 A U B

Για να δημιουργήσουμε μία περιοχή σχήματος ορθογωνίου αρχικά πρέπει να δημιουργήσουμε το ορθογώνιο ,όπως στην παράγραφο 5.3.4 , χρησιμοποιώντας ένα αντικείμενο της κλάσης Rectangle2D παραγόμενη κλάση της κλάσης Shape.

Rectangle2D.Double rect_left = new Rectangle2D.Double(x,y,width,height)

Για να ορίσουμε ένα σχήμα ως περιοχή χρησιμοποιούμε την κλάση Area η οποία δέχεται ως όρισμα αντικείμενα της κλάσης Shape.

Area a = new Area(Shape s)

Στην περίπτωση του ορθογωνίου rect_left γράφουμε :

Area c1 = new Area(rect_left)

Επίσης ορίζουμε ακόμη ένα τόξο arc_right ,όπως στην παράγραφο 6.2.1 [6.2.1], και έπειτα την αντίστοιχη περιοχή r1.

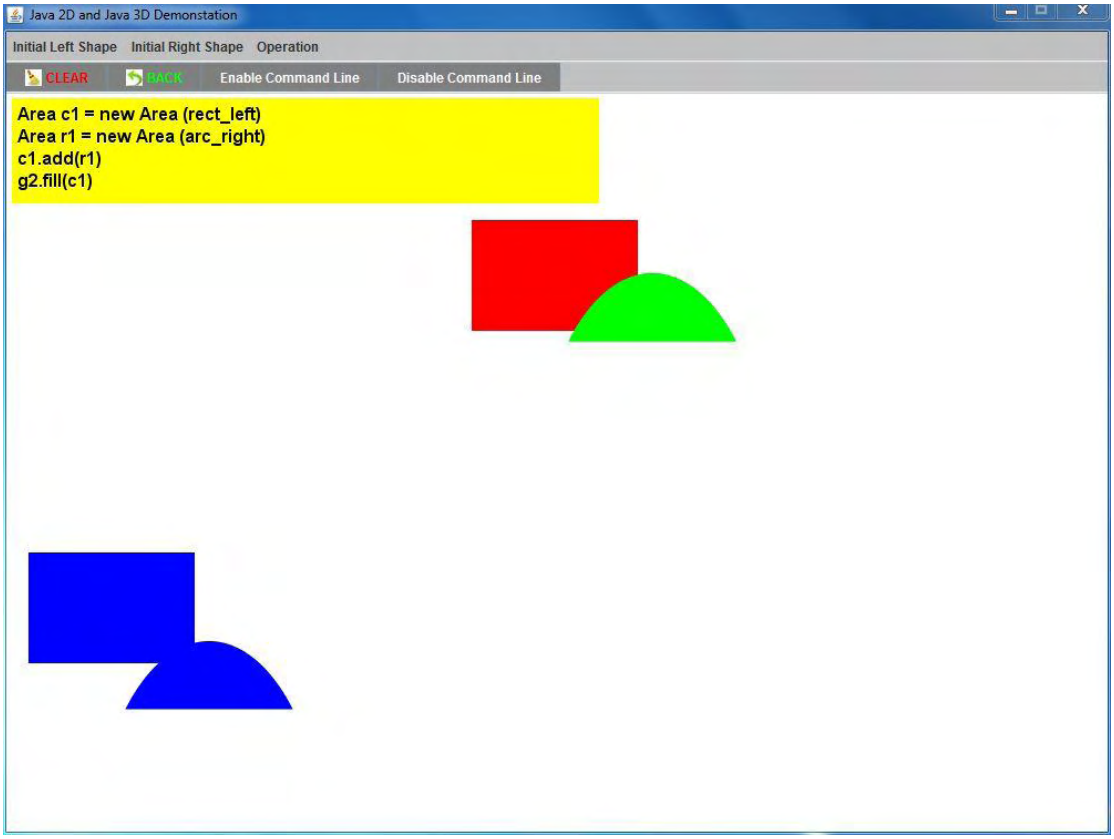
Area r1 = new Area(arc_right)

Η ένωση δύο περιοχών δημιουργείται γράφοντας :

c1.add(r1)

Η περιοχή c1 περιέχει το αποτέλεσμα της πράξης. Για να εμφανίσουμε το περίγραμμα της τελικής περιοχής χρησιμοποιούμε τη μέθοδο draw() με παράμετρο την περιοχή c1. Για να εμφανίζουμε την τελική περιοχή γεμισμένη με χρώμα χρησιμοποιούμε τη μέθοδο fill() ξανά με παράμετρο την περιοχή c1.

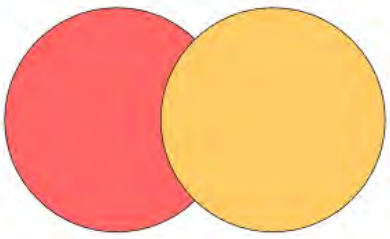
Η εφαρμογή στην επιλογή «Areas» δίνει την δυνατότητα στον χρήστη να σχηματίσει πιο περίπλοκες περιοχές χρησιμοποιώντας ως αρχικές προκαθορισμένες περιοχές τα σχήματα του ορθογωνίου, του κύκλου και του τόξου. Μπορεί να επιλεχθεί από το μενού «Initial Left Shape» το αριστερό σχήμα και από το μενού «Initial Right Shape» το δεξί σχήμα. Έπειτα για να εμφανιστεί η ένωσή τους γεμισμένη με μπλε χρώμα επιλέγεται από το μενού «Operation» η πράξη «Union». Σε προκαθορισμένη θέση στην περιοχή σχεδίασης εμφανίζεται η ένωση των επιλεγμένων σχημάτων. Η επιλογή αυτή δεν περιέχει αλληλεπίδραση με το ποντίκι.



ΕΙΚΟΝΑ 75 ΕΝΩΣΗ ΟΡΘΟΓΩΝΙΟΥ ΚΑΙ ΤΟΞΟΥ

7.2.2 DIFFERENCE – ΔΙΑΦΟΡΑ

Μία άλλη πράξη είναι η διαφορά. Έστω η διαφορά του κύκλου Β από τον κύκλο Α. Η διαφορά τους Α-Β θα εμφανίσει την περιοχή του κύκλου Α εκτός της περιοχής που ορίζει ο κύκλος Β. Η διαφορά εμφανίζεται με ροζ χρώμα.



ΕΙΚΟΝΑ 76 A-B

Για να ορίσουμε τη διαφορά της περιοχής του τόξου arc_right από την περιοχή του ορθογωνίου rect_left που δημιουργήθηκαν στην προηγούμενη παράγραφο αρχικά δημιουργούμε δύο νέες περιοχές γιατί στην περιοχή του ορθογωνίου c1 πλέον βρίσκεται η ένωση των δύο σχημάτων. Έτσι έχουμε :

Area c2 = new Area(rect_left)

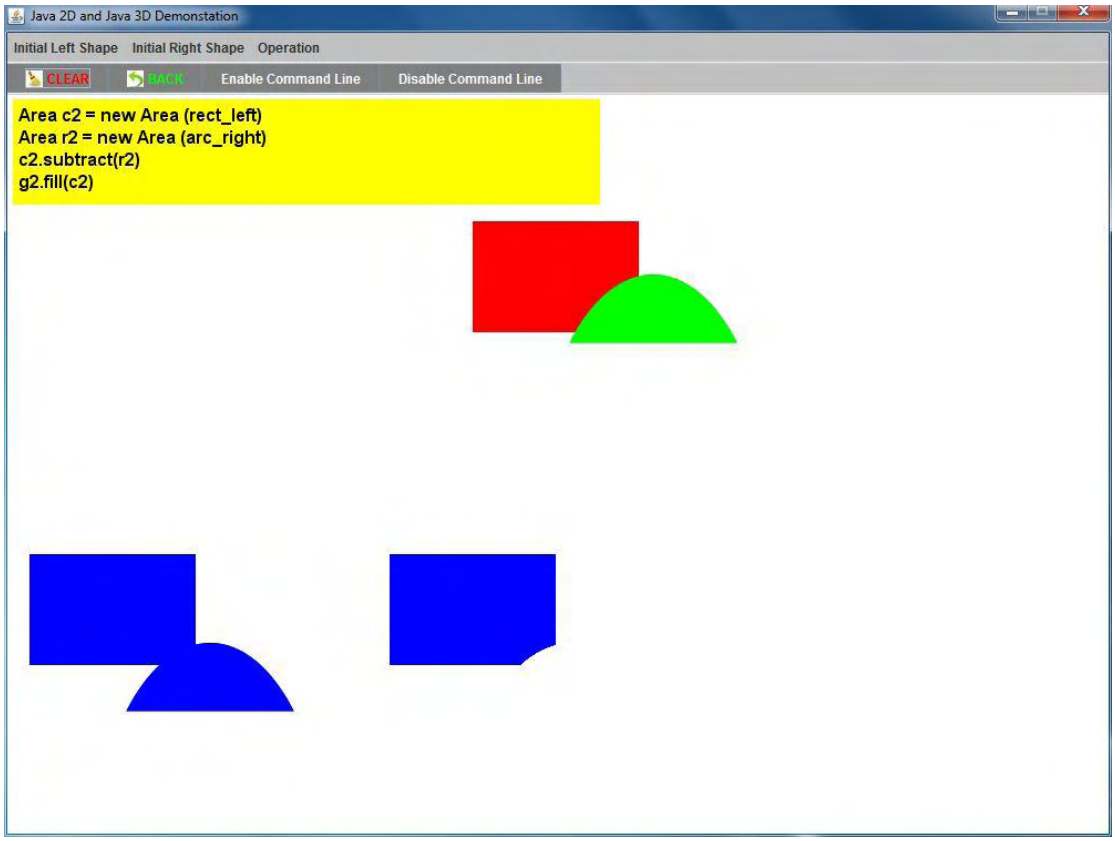
Area r2 = new Area(arc_right)

Για την διαφορά τους γράφουμε :

c2.subtract(r2)

Η περιοχή c2 περιέχει το αποτέλεσμα της πράξης. Όπως αναφέρθηκε και στην προηγούμενη παράγραφο μπορούμε να εμφανίσουμε το περίγραμμα του τελικού σχήματος με την μέθοδο draw() καλούμενη για την περιοχή c2. Επίσης μπορούμε να εμφανίσουμε την διαφορά των δύο σχημάτων γεμισμένη με χρώμα με τη βοήθεια της μεθόδου fill() καλούμενη για την τελική περιοχή c2.

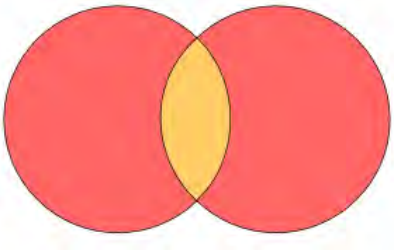
Στην εφαρμογή αφού επιλέξουμε αριστερό και δεξί σχήμα με τον τρόπο που περιγράφεται στην προηγούμενη παράγραφο μπορούμε να εμφανίσουμε την διαφορά αυτών των σχημάτων επιλέγοντας από το μενού «Operation» την επιλογή «Difference». Η διαφορά εμφανίζεται σε προκαθορισμένη θέση στην περιοχή σχεδίασης.



ΕΙΚΟΝΑ 77 ΑΦΑΙΡΕΣΗ ΤΟΞΟΥ ΑΠΟ ΟΡΘΟΓΩΝΙΟ

7.2.3 SYMMETRIC DIFFERENCE – ΑΠΟΚΛΕΙΣΤΙΚΟ Η (XOR)

Η πράξη αποκλειστικό ή των κύκλων A και B εμφανίζει τους δύο κύκλους εκτός από την κοινή περιοχή τους όπως φαίνεται στην εικόνα 78. Το αποτέλεσμα της πράξης αποκλειστικό ή A Δ B εμφανίζεται με ροζ.



ΕΙΚΟΝΑ 78 Α Δ Β

Για να ορίσουμε την πράξη αποκλειστικό ή του ορθογωνίου rect_left και του τόξου arc_right ορίζουμε όπως και πριν τις περιοχές τους.

Area c3 = new Area(rect_left)

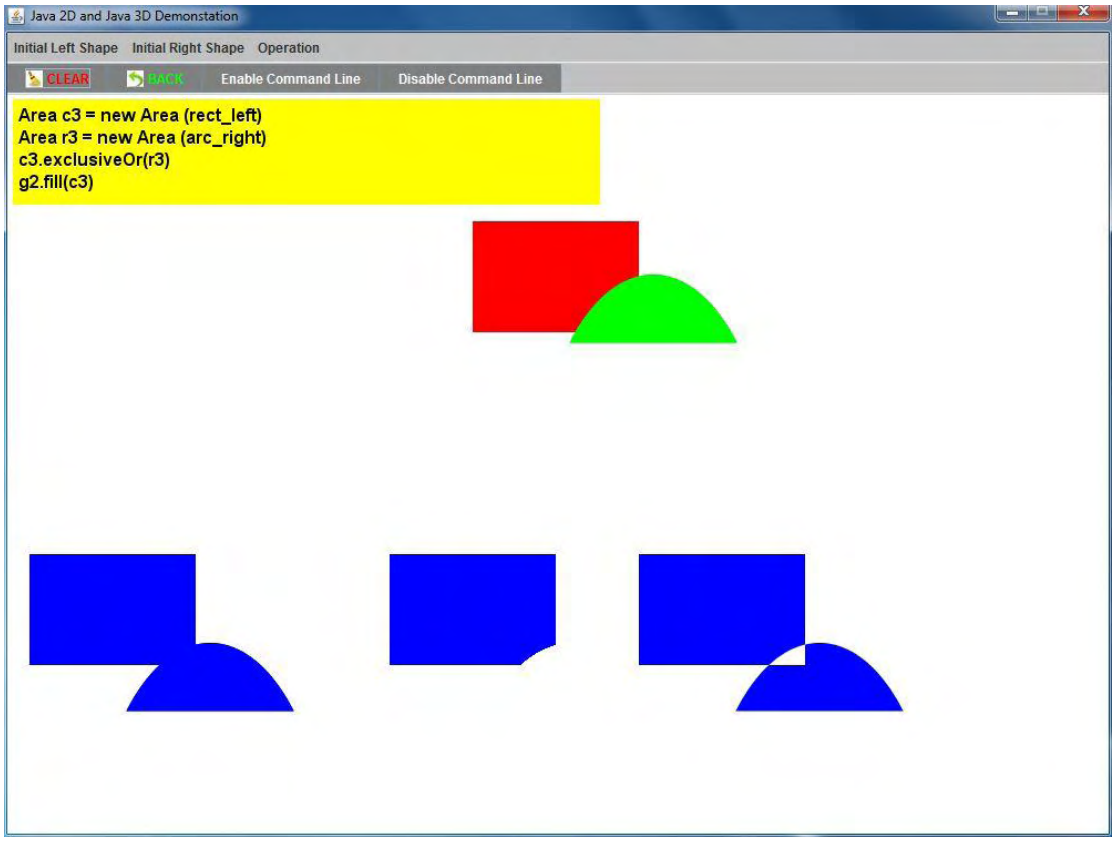
Area r3 = new Area(arc_right)

Για την πράξη γράφουμε :

c3.exclusiveOr(r3)

Η περιοχή c3 περιέχει το αποτέλεσμα της πράξης. Μπορούμε να χρησιμοποιήσουμε τις μεθόδους draw() και fill() με παράμετρο την περιοχή c3.

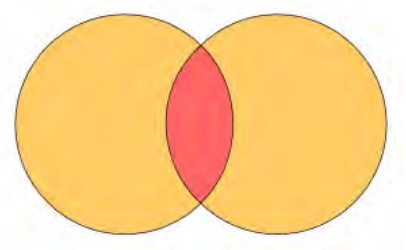
Στην εφαρμογή αφού επιλέξουμε το αριστερό και το δεξί σχήμα έπειτα επιλέγουμε την επιλογή «Symmetric Difference» από το μενού «Operation». Η πράξη αποκλειστικό ή εμφανίζεται σε καθορισμένη θέση στην περιοχή σχεδίασης.



ΕΙΚΟΝΑ 79 ΑΠΟΚΛΕΙΣΤΙΚΟ Η ΟΡΘΟΓΩΝΙΟΥ ΜΕ ΤΟΞΟ

7.2.4 Intersection – Τομη

Έστω οι κύκλοι A, B των προηγούμενων παραγράφων. Η τομή των A,B περιέχει την κοινή περιοχή τους. Η τομή A ∩ B των δύο κύκλων εμφανίζεται στην εικόνα 80 με ροζ χρώμα.



Εικονα 80 A ∩ B

Για να εμφανίσουμε την τομή του ορθογωνίου rect_left με το τόξο arc_right ορίζουμε τις περιοχές τους γράφοντας :

Area c4 = new Area(rect_left)

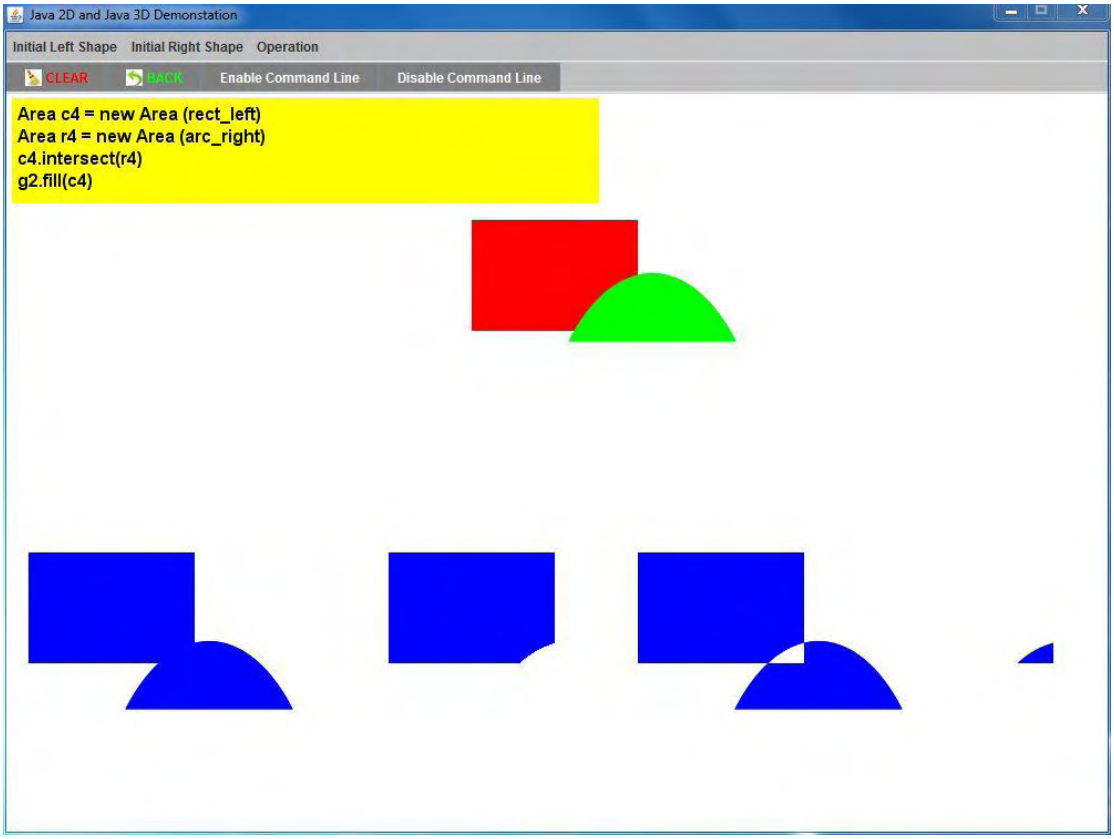
Area r4 = new Area(arc_right)

Για την πράξη της τομής γράφουμε την εντολή :

c4.intersect(r4)

Η περιοχή c4 περιέχει το αποτέλεσμα της πράξης. Για να εμφανίσουμε το περίγραμμα όπως και στις προηγούμενες πράξεις χρησιμοποιούμε την μέθοδο draw() με παράμετρο την περιοχή c4. Για να γεμίσουμε την περιοχή c4 που περιέχει την τομή με χρώμα χρησιμοποιούμε την μέθοδο fill().

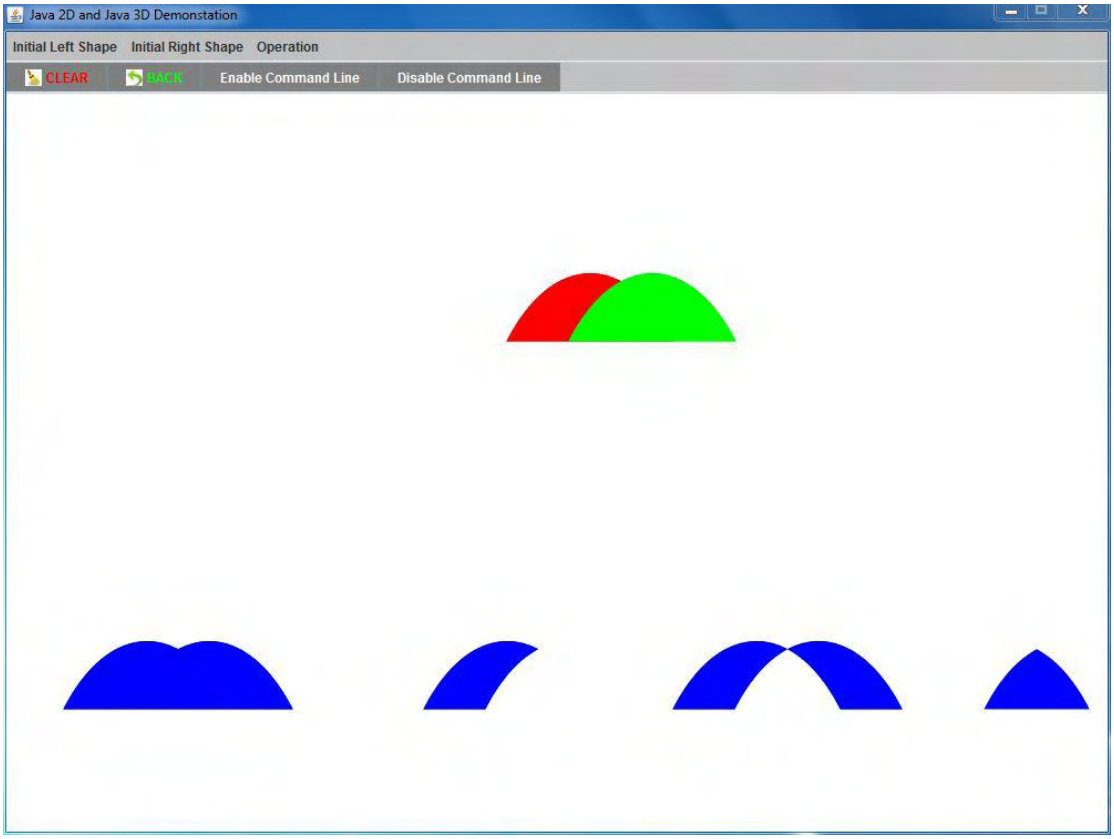
Στην εφαρμογή επιλέγουμε αριστερό και δεξί σχήμα και έπειτα για να εμφανίσουμε τη τομή των επιλεγμένων σχημάτων επιλέγουμε την επιλογή «Intersection» από το μενού «Operation». Η τομή εμφανίζεται σε καθορισμένο σημείο της περιοχής σχεδίασης.



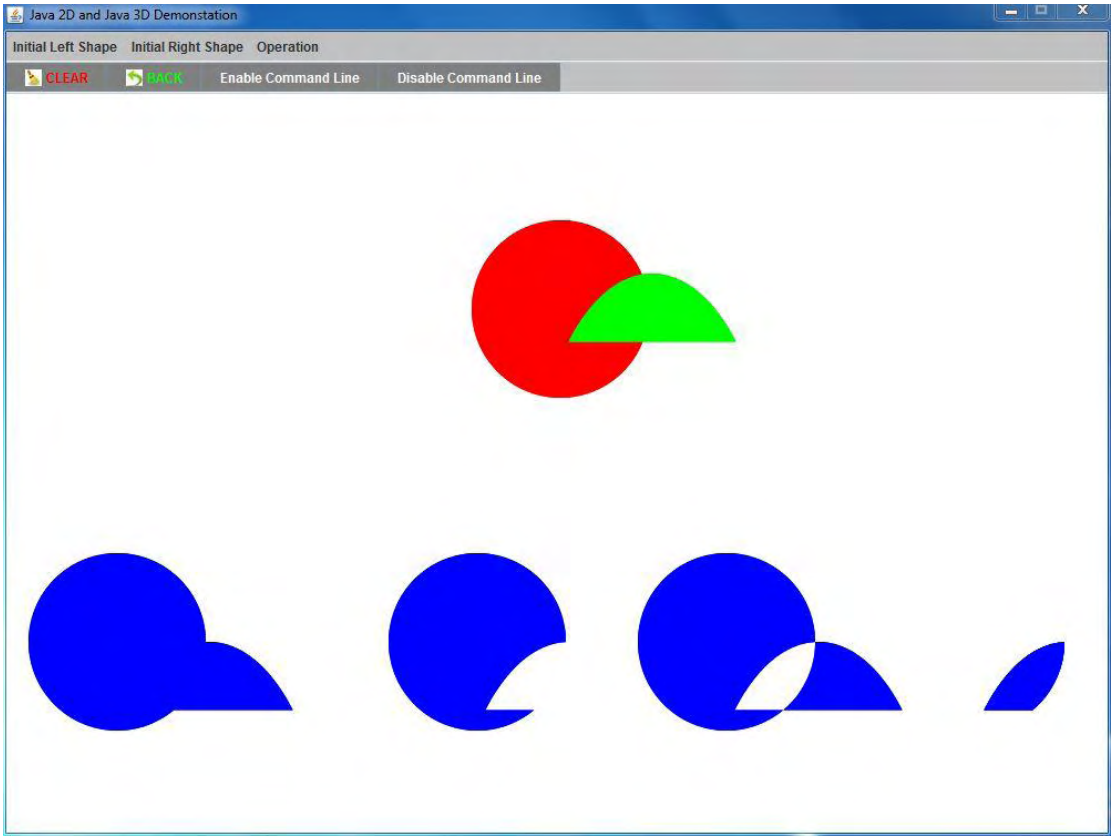
ΕΙΚΟΝΑ 81 ΤΟΜΗ ΟΡΘΟΓΩΝΙΟΥ ΜΕ ΤΟΞΟ

7.2.5 ΠΡΑΞΕΙΣ ΑΛΛΩΝ ΣΧΗΜΑΤΩΝ

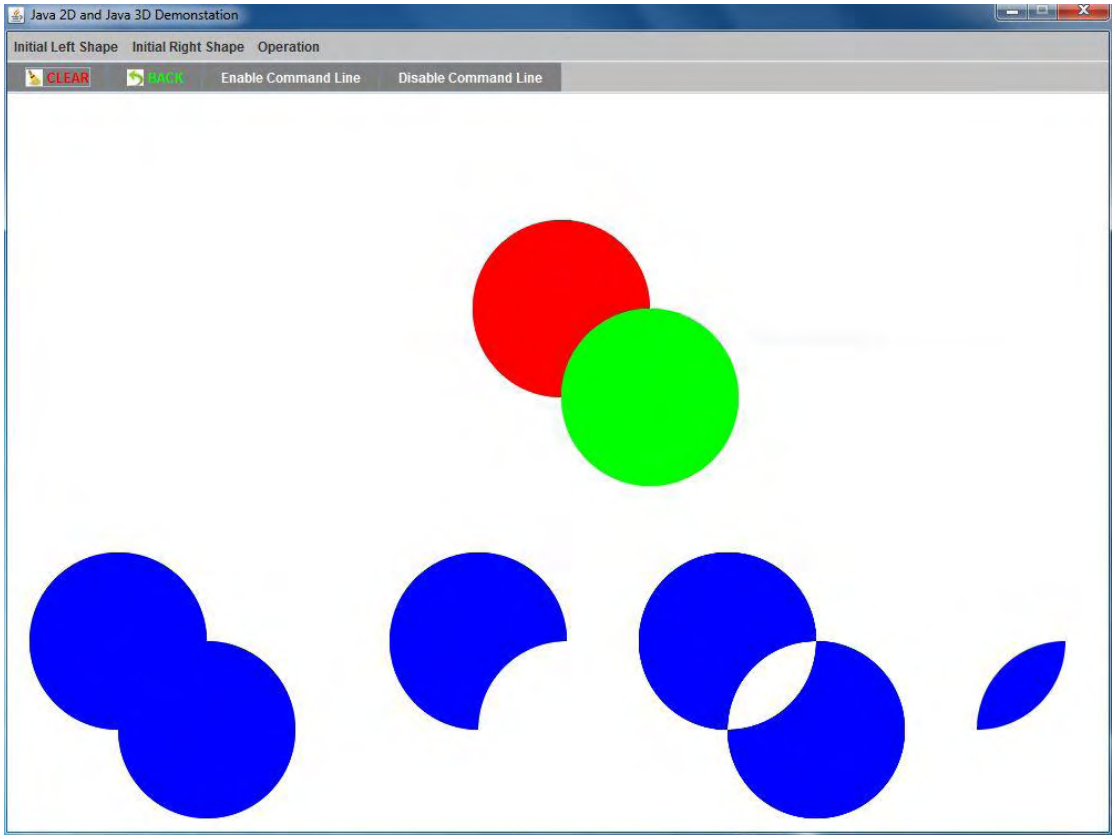
Έχει αναφερθεί ότι μπορούμε να επιλέξουμε για αρχικά σχήματα μεταξύ των σχημάτων κύκλος, ορθογώνιο και τόξο. Παρακάτω εμφανίζονται οι πράξεις συνόλων μερικών συνδυασμών των διαθέσιμων σχημάτων.



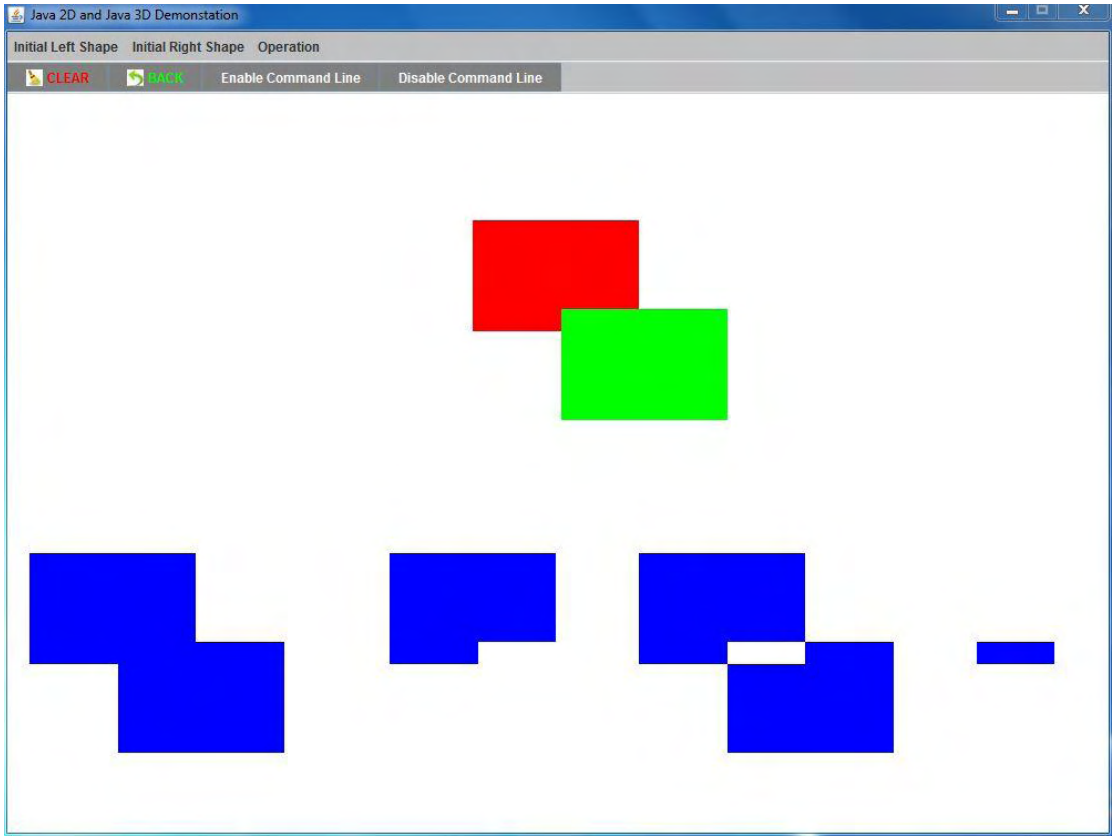
ΕΙΚΟΝΑ 82 ΤΟΞΟ ΜΕ ΤΟΞΟ



ΕΙΚΟΝΑ 83 ΚΥΚΛΟΣ ΜΕ ΤΟΞΟ



ΕΙΚΟΝΑ 84 ΚΥΚΛΟΣ ΜΕ ΚΥΚΛΟ



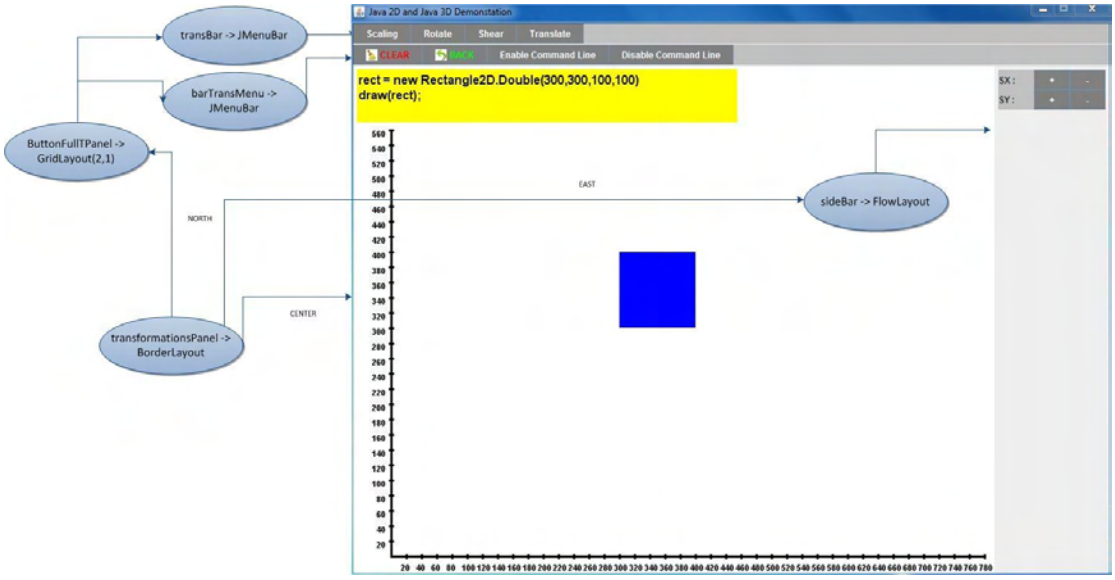
ΕΙΚΟΝΑ 85 ΟΡΘΟΓΩΝΙΟ ΜΕ ΟΡΘΟΓΩΝΙΟ

8. Java 2D – 4η Επιλογη (Geometric Transformations)

8.1 Παρουσιαση

Η τέταρτη επιλογή της εφαρμογής είναι το αντικείμενο της κλάσης TransformationsPanel. Είναι ένα panel που περιέχει Swing components όπως κουμπιά και ετικέτες και εμφανίζει panels με επιλογές ανάλογα με το κουμπί που έχει πατηθεί. Επίσης υπάρχει μία περιοχή η οποία περιέχει ένα καρτεσιανό σύστημα αξόνων σχεδίασης και ένα μπλε τετράγωνο. Ο χρήστης μπορεί να επιλέξει ανάμεσα σε 4 διαφορετικούς μετασχηματισμούς για το τετράγωνο αλλάζοντάς του το μέγεθος (scaling), τον προσανατολισμό (rotation), το σχήμα (shear) και την θέση (translation). Τέλος παρέχονται οι λειτουργίες που περιγράφηκαν στο πέμπτο κεφάλαιο : καθαρισμός της περιοχής σχεδίασης και επιστροφή στα εξώφυλλο [5.4.9] και η γραμμή εντολών [5.4.11].

Το TransformationsPanel έχει custom δομή που δημιουργείται με border Layout, grid Layout και flow Layout.



Εικονα 86 transformationsPanel layout

8.2 Λειτουργικοτητα

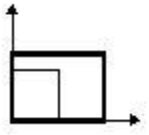
8.2.1 Scaling – Αλλαγη Μεγεθους Δισδιαστατου Σχηματος

Ο μετασχηματισμός scaling οδηγεί σε ένα τέντωμα ή σε μία σμίκρυνση του σχήματος στον x ή/και στον y άξονα. Ένας μετασχηματισμός scaling S(sx , sy) προβάλει το σημείο (x , y) στο σημείο (x' , y'). Το σημείο (x' , y') δίνεται από τον τύπο :

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} sx \cdot x \\ sy \cdot y \end{pmatrix} = \begin{pmatrix} sx & 0 \\ 0 & sy \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$

Sx είναι ο παράγοντας για το scaling στον x άξονα. Αν |sx|>1 γίνεται τέντωμα του σχήματος στον άξονα των x, αν |sx|<1 γίνεται σμίκρυνση του σχήματος. Αν ο παράγοντας sx είναι αρνητικός τότε

μαζί με το τέντωμα ή την σμίκρυνση του σχήματος γίνεται και αντανάκλαση του σχήματος ως προς τον y άξονα. Ο sy παράγοντας οδηγεί σε τέντωμα ή σμίκρυνση του σχήματος στον άξονα των y, αναλόγως με το sx. Αρνητικό sy οδηγεί εκτός από τέντωμα ή σμίκρυνση οδηγεί και σε αντανάκλαση του σχήματος ως προς τον x άξονα.



ΕΙΚΟΝΑ 87 SCALING – ΤΕΝΤΩΜΑ ΤΕΤΡΑΓΩΝΟΥ ΩΣ ΠΡΟΣ Χ ΚΑΙ Υ ΑΞΟΝΑ

Για να κάνουμε τον μετασχηματισμό scaling αρχικά δημιουργούμε ένα αντικείμενο της κλάσης Affine Transform το οποίο δημιουργεί τον μοναδιαίο πίνακα. Ο πολλαπλασιασμός οπουδήποτε σημείου με τον μοναδιαίο πίνακα οδηγεί στην προβολή του σημείου στην ίδια θέση χωρίς καμία αλλαγή.

AffineTransform scaling = new AffineTransform()

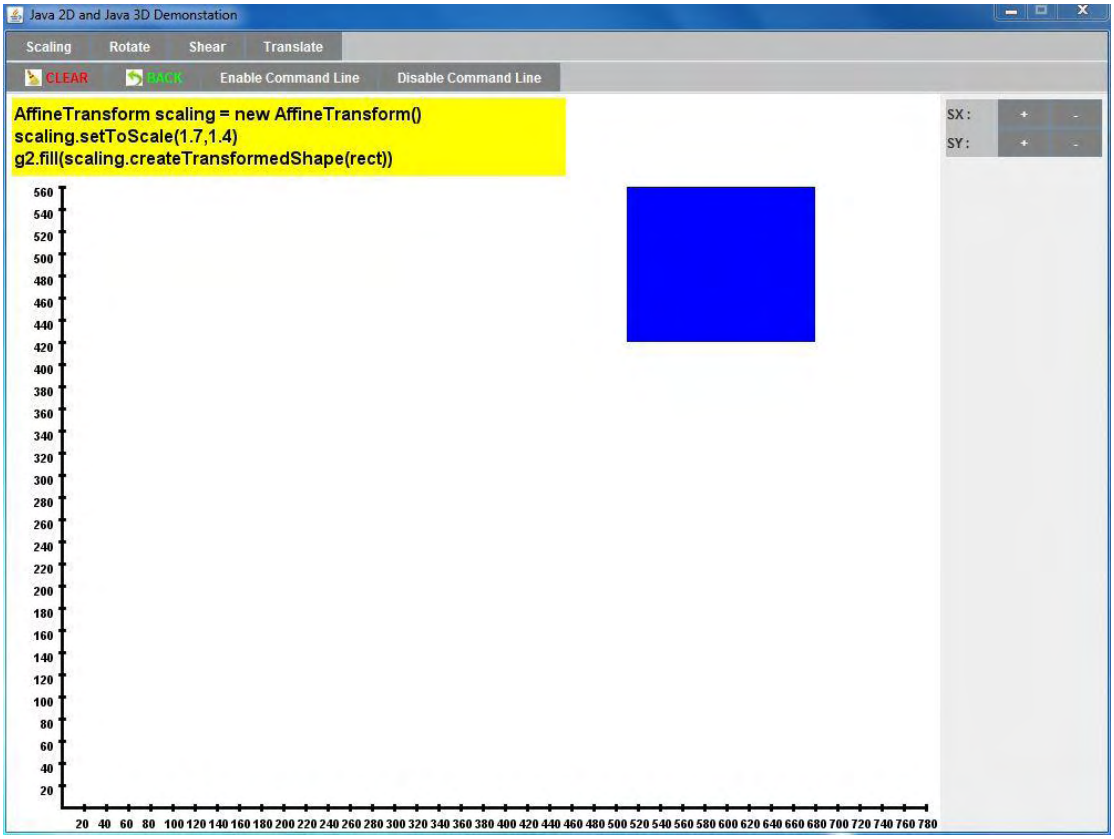
Έπειτα ορίζουμε το αντικείμενο scaling της κλάσης Affine Transform ως μετασχηματισμό τύπου scaling και εισάγουμε τους παράγοντες sx και sy.

Scaling.setToScale(sx , sy)

Για να εφαρμοστεί ο παραπάνω μετασχηματισμός σε ένα αντικείμενο της κλάσης Shape γράφουμε :

Shape transformedShape = scaling.createTransformedShape(s)

Το αντικείμενο transformedShape μπορεί να εμφανιστεί στην οθόνη καλώντας τις μεθόδους draw() και fill() με παράμετρο το αντικείμενο transformedShape ανάλογα με το αν θέλουμε να εμφανιστεί το περίγραμμά του ή να είναι γεμισμένο με χρώμα.
Στην εφαρμογή καθώς επιλέγεται από το εξώφυλλο η επιλογή «Geometric Transformations in Java 2D» εμφανίζεται ένα τετράγωνο μπλε χρώματος σχεδιασμένο σε συγκεκριμένες συντεταγμένες. Πατώντας το κουμπί «Scaling» εμφανίζεται ένα panel στα δεξιά της περιοχής σχεδίασης. Το panel αυτό περιέχει κουμπιά με τα οποία αυξάνονται και μειώνονται οι παράγοντες scaling sx και sy. Με κάθε αλλαγή των sx ή sy το σχήμα εμφανίζεται ανανεωμένο.



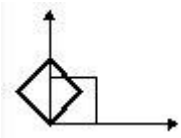
ΕΙΚΟΝΑ 88 ΣΤΙΓΜΙΟΤΥΠΟ SCALING

8.2.2 ROTATE – ΑΛΛΑΓΗ ΠΡΟΣΑΝΑΤΟΛΙΣΜΟΥ ΔΙΣΔΙΑΣΤΑΤΟΥ ΣΧΗΜΑΤΟΣ

Ο μετασχηματισμός rotation οδηγεί στην αλλαγή προσανατολισμού του σχήματος κατά τον παράγοντα θ δηλαδή την γωνία περιστροφής. Ένας μετασχηματισμός rotation R(θ) κατά γωνία θ προβάλει το σημείο (x , y) στο σημείο (x' , y'). Το σημείο (x' , y') δίνεται από τον τύπο :

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \cdot \cos(\theta) - y \cdot \sin(\theta) \\ x \cdot \sin(\theta) + y \cdot \cos(\theta) \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$

Η περιστροφή γίνεται αντίθετα από τη φορά των δεικτών του ρολογιού γύρω από την αρχή των αξόνων αν η γωνία περιστροφής είναι θετική. Αν η γωνία περιστροφής είναι αρνητική η περιστροφή γίνεται σύμφωνα με τη φορά των δεικτών του ρολογιού.



ΕΙΚΟΝΑ 89 ROTATION - ΠΕΡΙΣΤΡΟΦΗ ΤΕΤΡΑΓΩΝΟΥ ΚΑΤΑ ΓΩΝΙΑ Θ

Για να κάνουμε τον μετασχηματισμό rotation, όπως και στο scaling, δημιουργούμε ένα αντικείμενο της κλάσης Affine Transform το οποίο δημιουργεί τον μοναδιαίο πίνακα.

AffineTransform rotation = new AffineTransform()

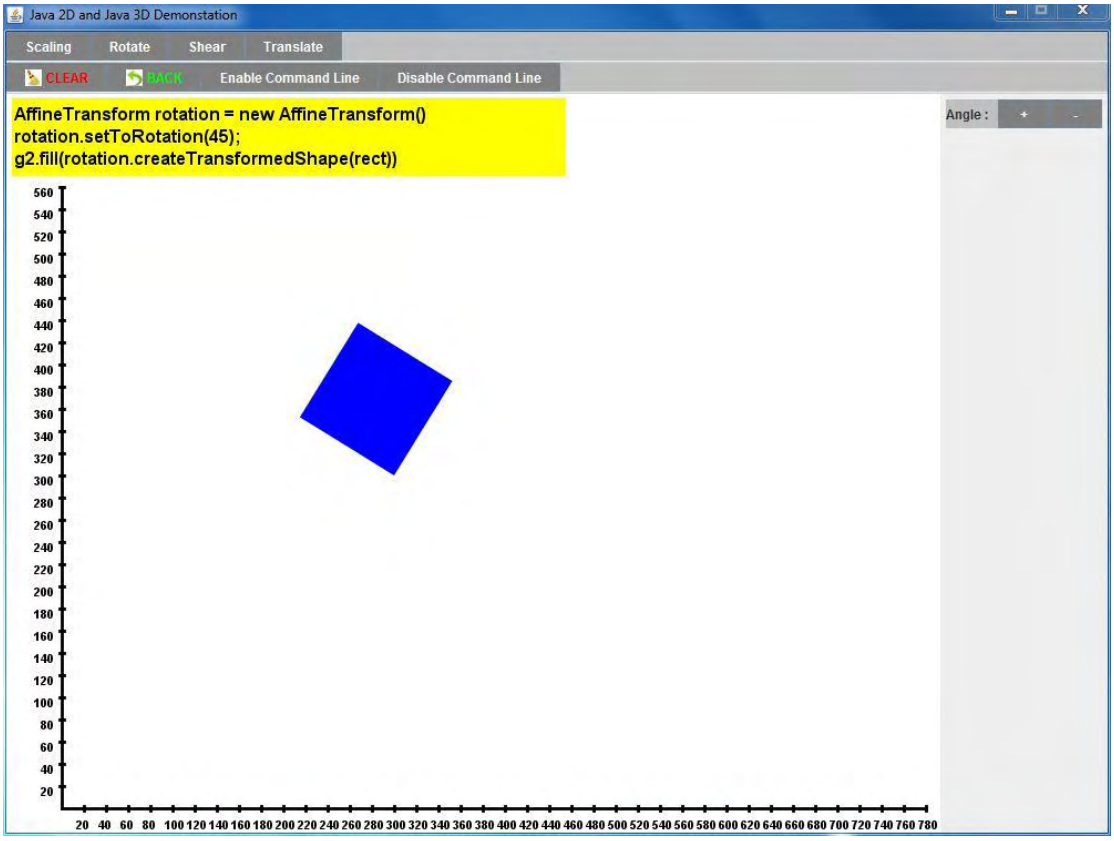
Έπειτα ορίζουμε το αντικείμενο rotation της κλάσης Affine Transform ως μετασχηματισμό τύπου περιστροφής (rotation) και εισάγουμε τον παράγοντα που καθορίζει την περιστροφή δηλαδή την γωνία περιστροφής θ. Η περιστροφή γίνεται γύρω από την αρχή των αξόνων.

Rotation.setToRotation(θ)

Για να εφαρμοστεί ο παραπάνω μετασχηματισμός σε ένα αντικείμενο της κλάσης Shape γράφουμε :

Shape transformedShape = rotation.createTransformedShape(s)

Το αντικείμενο transformedShape μπορεί να εμφανιστεί στην οθόνη καλώντας τις μεθόδους draw() και fill() με παράμετρο το αντικείμενο transformedShaep ανάλογα με το αν θέλουμε να εμφανιστεί το περίγραμμά του ή να είναι γεμισμένο με χρώμα.
Στην εφαρμογή πατώντας το κουμπί «Rotate» εμφανίζεται ένα panel στα δεξιά της περιοχής σχεδίασης. Το panel αυτό περιέχει κουμπιά με τα οποία αυξάνεται και μειώνεται η γωνία περιστροφής θ. Με κάθε αλλαγή της γωνίας θ το σχήμα εμφανίζεται ανανεωμένο.



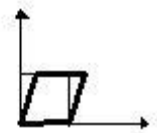
Εικονα 90 στιγμιοτυπο rotation

8.2.3 Shear – Αλλαγη Σχηματος Δισδιαστατου Σχηματος

Ο μετασχηματισμός shearing οδηγεί σε παραμόρφωση του σχήματος. Ένας μετασχηματισμός shearing Sh(sx , sy) του σημείου (x , y) δίνει το σημείο (x' , y') με νέες συντεταγμένες. Το σημείο (x' , y') δίνεται από τον τύπο :

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x + sx \cdot y \\ y + sy \cdot x \end{pmatrix} = \begin{pmatrix} 1 & sx \\ sy & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$

Ο μετασχηματισμός shearing γίνεται γύρω από την αρχή των αξόνων. Αν ένα αντικείμενο δεν είναι κεντραρισμένο γύρω από την αρχή των αξόνων θα παραμορφωθεί από το shearing αλλά και θα μετατοπιστεί. Αν επιλεγεί ο παράγοντας sx να είναι 0 τότε το shearing γίνεται μόνο ως προς τον άξονα των x. Αν ο παράγοντας sy είναι 0 τότε το shearing γίνεται μόνο ως προς τον άξονα των y.



ΕΙΚΟΝΑ 91 ΑΛΛΑΓΗ ΣΧΗΜΑΤΟΣ ΤΕΤΡΑΓΩΝΟΥ ΩΣ ΠΡΟΣ ΤΟΝ ΑΞΟΝΑ ΤΩΝ Y

Για να κάνουμε τον μετασχηματισμό shearing δημιουργούμε ένα αντικείμενο της κλάσης Affine Transform.

AffineTransform shear = new AffineTransform()

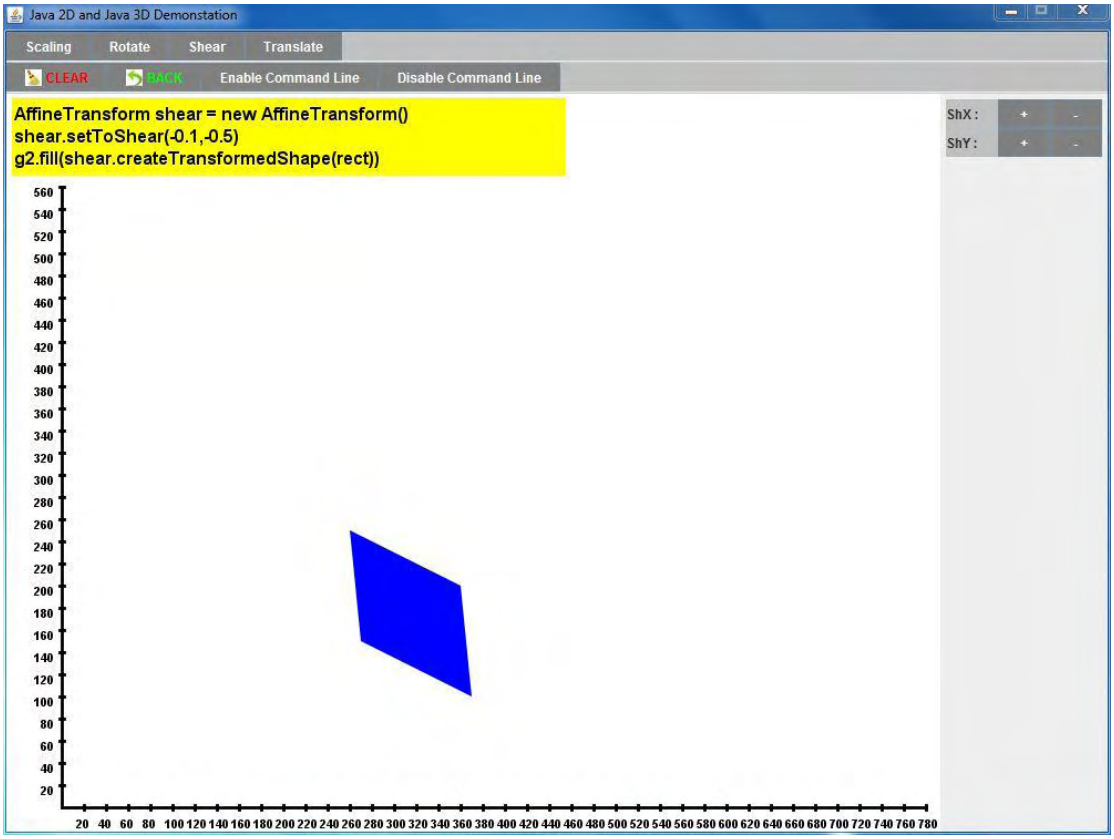
Έπειτα ορίζουμε το αντικείμενο shear της κλάσης Affine Transform ως μετασχηματισμό τύπου shearing και εισάγουμε τους παράγοντες sx και sy.

Shear.setToShear(sx , sy)

Για να εφαρμοστεί ο παραπάνω μετασχηματισμός σε ένα αντικείμενο της κλάσης Shape γράφουμε :

Shape transformedShape = shear.createTransformedShape(s)

Το αντικείμενο transformedShape μπορεί να εμφανιστεί στην οθόνη με τις μεθόδους draw() και fill() έχοντας ως παράμετρο το αντικείμενο transformedShape ανάλογα με το αν θέλουμε να εμφανιστεί το περίγραμμά του ή να είναι γεμισμένο με χρώμα.
Στην εφαρμογή πατώντας το κουμπί «Shear» εμφανίζεται ένα panel στα δεξιά της περιοχής σχεδίασης. Το panel αυτό περιέχει κουμπιά με τα οποία αυξάνονται και μειώνονται οι παράγοντες shear sx και sy. Με κάθε αλλαγή των sx ή sy το σχήμα εμφανίζεται ανανεωμένο.

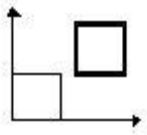


ΕΙΚΟΝΑ 92 ΣΤΙΓΜΙΟΤΥΠΟ SHEAR

8.2.4 TRANSLATE – ΑΛΛΑΓΗ ΘΕΣΗΣ ΔΙΣΔΙΑΣΤΑΤΟΥ ΣΧΗΜΑΤΟΣ

Ο μετασχηματισμός translation οδηγεί σε αλλαγή της θέσης του σχήματος. Ένας μετασχηματισμός translation T(dx , dy) οδηγεί σε μετατόπιση κατά το διάνυσμα d=(dx , dy) ^{T} . Ο μετασχηματισμός translation προβάλει το σημείο (x , y) στο σημείο (x' , y') που δίνεται από τον τύπο :

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x + dx \\ y + dy \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} dx \\ dy \end{pmatrix}$$



ΕΙΚΟΝΑ 93 ΑΛΛΑΓΗ ΘΕΣΗΣ ΤΕΤΡΑΓΩΝΟΥ

Για να κάνουμε τον μετασχηματισμό translation δημιουργούμε ένα αντικείμενο της κλάσης Affine Transform.

AffineTransform translation = new AffineTransform()

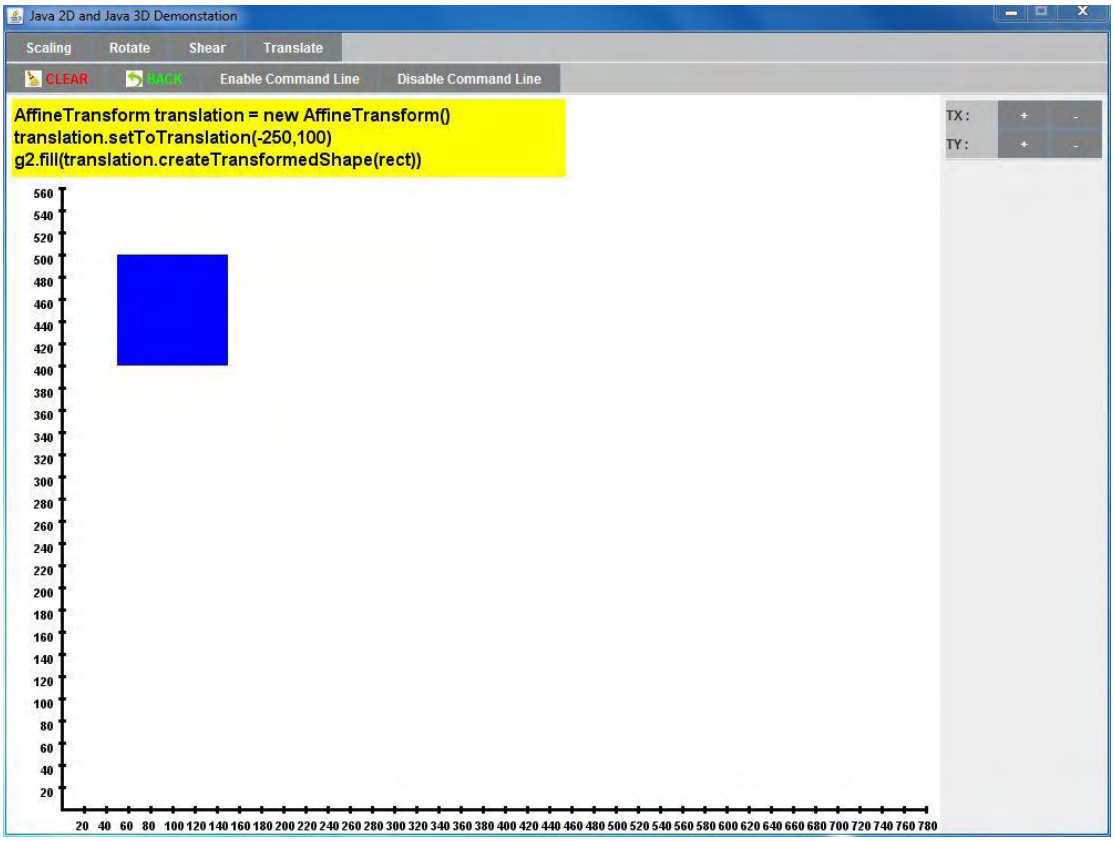
Έπειτα ορίζουμε το αντικείμενο translation της κλάσης Affine Transform ως μετασχηματισμό τύπου translation και εισάγουμε τους παράγοντες dx και dy.

translation.setToTrnaslation(dx , dy)

Για να εφαρμοστεί ο παραπάνω μετασχηματισμός σε ένα αντικείμενο της κλάσης Shape γράφουμε :

Shape transformedShape = trnaslation.createTransformedShape(s)

Το αντικείμενο transformedShape μπορεί να εμφανιστεί στην οθόνη με τις μεθόδους draw() και fill() και παράμετρο το αντικείμενο transformedShape ανάλογα με το αν θέλουμε να εμφανιστεί το περίγραμμά του ή να είναι γεμισμένο με χρώμα.
Στην εφαρμογή πατώντας το κουμπί «Translate» εμφανίζεται ένα panel στα δεξιά της περιοχής σχεδίασης. Το panel αυτό περιέχει κουμπιά με τα οποία αυξάνονται και μειώνονται οι παράγοντες translation dx και dy. Με κάθε αλλαγή των dx ή dy το σχήμα εμφανίζεται ανανεωμένο.



ΕΙΚΟΝΑ 94 ΣΤΙΓΜΙΟΤΥΠΟ TRANSLATION

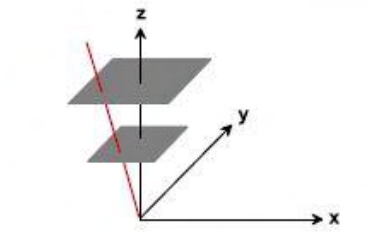
8.3 ΟΜΟΓΕΝΕΙΣ ΣΥΝΤΕΤΑΓΜΕΝΕΣ

Οι μετασχηματισμοί scaling, rotation, shear είναι γραμμικοί μετασχηματισμοί. Ο μετασχηματισμός translation δεν είναι γραμμικός έτσι δεν μπορεί να εκφραστεί ως πολλαπλασιασμός πινάκων. Ο πολλαπλασιασμός πινάκων αφήνει πάντα το μηδενικό διάνυσμα χωρίς αλλαγή, όμως ένας μετασχηματισμός translation θα οδηγήσει σε αλλαγή της θέσης του μηδενικού διανύσματος , άρα και της αρχής των αξόνων, κατά διάνυσμα d.

Στα γραφικά η/υ πολύπλοκοι μετασχηματισμοί εκφράζονται ως συνθέσεις απλών μετασχηματισμών. Ένας μετασχηματισμός αποτελούμενους από scaling, shears, rotations μπορεί να εκφραστεί ως ένας πίνακας, δηλαδή το γινόμενο των πινάκων που αντιστοιχούν σε αυτούς τους

μετασχηματισμούς. Αν πρέπει να συμπεριλάβουμε στον μετασχηματισμό αυτό και translations τότε ο μετασχηματισμός δεν μπορεί να εκφραστεί ως ένας μόνο πίνακας. Αν μπορούσαμε να αναπαραστήσουμε κάθε πιθανό σύνθετο μετασχηματισμό σε έναν μόνο πίνακα θα ήταν μεγάλο πλεονέκτημα από άποψης εξοικονόμησης μνήμης και μείωσης του χρόνου υπολογισμού. Για να το πετύχουμε αυτό αναπαριστούμε τις συντεταγμένες των σημείων με διαφορετικό τρόπο χρησιμοποιώντας τις ομογενείς συντεταγμένες.

Οι ομογενείς συντεταγμένες χρησιμοποιούν άλλη μία διάσταση για την αναπαράσταση συντεταγμένων στον επίπεδο. Το σημείο (x,y,z) σε ομογενείς συντεταγμένες αντιστοιχεί στο σημείο ($\frac{x}{z}$, $\frac{y}{z}$) σε καρτεσιανές συντεταγμένες με z≠0. Για να εκφραστεί το σημείο (x,y) σε ομογενείς συντεταγμένες μπορούμε να χρησιμοποιήσουμε το (x,y,1). Όποια άλλη αναπαράσταση της μορφής (z ·x0 , z·y0,z) με z≠0 κωδικοποιεί το ίδιο σημείο. Το σημείο {(x,y,z) $\epsilon\mathbb{R}^3$ | (x,y,z) = (z ·x0 , z·y0,z)} βρίσκεται πάνω στη γραμμή στον $\mathbb{R}^3$ και περνάει από την αρχή των αξόνων. Η γραμμή δίνεται από τις εξισώσεις : x-x0·z=0 και y-y0·z=0 . Οποιοδήποτε σημείο στην γραμμή αυτή εκτός από την αρχή των αξόνων αντιπροσωπεύει το σημείο (x0,y0) σε ομογενείς συντεταγμένες. Αποφασίζοντας την τιμή του z π.χ. z=1 τότε το καρτεσιανό επίπεδο αναπαρίσταται από ένα παράλληλο σε αυτό επίπεδο με την αντίστοιχη τιμή του z.



ΕΙΚΟΝΑ 95 ΟΜΟΓΕΝΕΙΣ ΣΥΝΤΕΤΑΓΜΕΝΕΣ

Όλα τα σημεία πάνω στην κόκκινή γραμμή αναπαριστούν το ίδιο σημείο με διαφορετικές τιμές του z ανάλογα σε ποιο γκρι επίπεδο βρίσκονται. Αποφασίζοντας τιμή για το z τότε το καρτεσιανό επίπεδο αναπαριστάται πλήρως από το γκρι επίπεδο σε ομογενείς συντεταγμένες που αντιστοιχεί στην επιλεγμένη τιμή του z.

Σε ομογενείς συντεταγμένες ο μετασχηματισμός translation μπορεί να γραφεί σαν πολλαπλασιασμός πινάκων :

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} x+dx \\ y+dy \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Οι υπόλοιποι μετασχηματισμοί μπορούν να γραφούν επεκτείνοντας τους αντίστοιχους πίνακες όπως φαίνεται παρακάτω.

- Scaling – S(sx,sy)

$$\begin{pmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- Rotation – R(θ)

$$\begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- Shear – Sh(sx,sy)

$$\begin{pmatrix} 1 & sx & 0 \\ sy & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- Translation - T(dx , dy)

$$\begin{pmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{pmatrix}$$

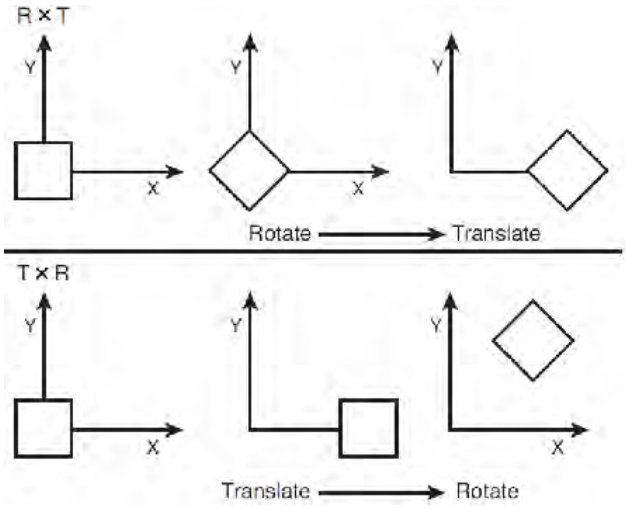
Με τις παραπάνω αναπαραστάσεις των μετασχηματισμών πλέον όλοι οι σύνθετοι μετασχηματισμοί μπορούν να γραφούν ως πολλαπλασιασμός πινάκων και να αναπαρασταθούν από έναν μόνο πίνακα. Όλοι οι παραπάνω πίνακες είναι της μορφής

$$\begin{pmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{pmatrix}.$$

Ο πολλαπλασιασμός πινάκων της παραπάνω μορφής έχει ως αποτέλεσμα πίνακα της ίδιας μορφής. Για αυτό το λόγο στα γραφικά η/υ οι μετασχηματισμοί αποθηκεύονται με αυτόν τον τρόπο. Η αναπαράσταση αυτή εκτός από τους μετασχηματισμούς στο επίπεδο είναι κατάλληλη και για τους μετασχηματισμούς στον χώρο, οι οποίοι θα αναλυθούν σε επόμενο κεφάλαιο [11]. Έτσι είναι φανερό ότι οι κάρτες γραφικών θα πρέπει εκτός των άλλων να μπορούν να κάνουν πολλαπλασιασμούς πινάκων πάρα πολύ γρήγορα.

8.4 Σειρα Μετασχηματισμων

Θα πρέπει να λάβουμε υπόψη ότι η σειρά με την οποία γίνονται οι μετασχηματισμοί έχει σημασία γιατί αλλάζει το τελικό αποτέλεσμα. Ο πολλαπλασιασμός πινάκων είναι μη αντιμεταθετική πράξη. Γενικά όταν συνδυάζουμε μετασχηματισμούς διαφορετικού τύπου παρατηρούμε διαφορετικά αποτελέσματα. Μόνο όταν εφαρμόζουμε μετασχηματισμούς ιδίου τύπου η σειρά δεν έχει σημασία για το αποτέλεσμα.

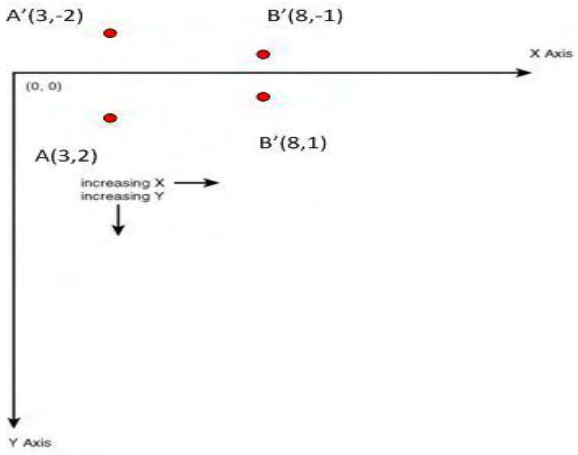


Εικονα 96 πανω - rotation και μετα translation
κατω - translation και μετα rotation

8.5 Μετατροπη Συντεταγμενων Παραθυρου σε Καρτεσιανες Συντεταγμενες

Στην επιλογή «Geometric Transformations in Java 2D» στην περιοχή σχεδίασης υπάρχει μόνιμα ζωγραφισμένο ένα σύστημα καρτεσιανών συντεταγμένων. Αυτό συμβαίνει γιατί όλα τα σχήματα ζωγραφίζονται βάση αυτού του συστήματος (το (0,0) σημείο βρίσκεται στην κάτω αριστερά γωνία με τον άξονα των y να εκτείνεται προς τα πάνω και τον άξονα των x προς τα δεξιά) και όχι βάση του συστήματος συντεταγμένων του παραθύρου (το (0,0) σημείο βρίσκεται στην πάνω αριστερά γωνία με τον άξονα των y να εκτείνεται προς τα κάτω και τον άξονα των x προς τα δεξιά) που είναι το default σύστημα σχεδίασης στα γραφικά η/υ.

Αυτό επιτυγχάνεται αν εφαρμόσουμε τον κατάλληλο μετασχηματισμό στο Graphics2D αντικείμενο το οποίο έχει δημιουργηθεί με casting του Graphics αντικειμένου που παίρνει ως παράμετρο η μέθοδος paint Component(). Έτσι κάθε σχήμα αφού δημιουργηθεί στα πλαίσια του Graphics2D αντικειμένου μετασχηματίζεται κατάλληλα και έπειτα ζωγραφίζεται στην οθόνη του η/υ. Πρώτα γίνεται ένα scaling αφήνοντας ίδια τα x και αλλάζοντας τα y. Τα y σχεδιάζονται συμμετρικά ως προς τον άξονα των x στο τεταρτημόριο που τα x είναι θετικά και τα y αρνητικά.



Εικονα 97 σημεια A, B μετα το scaling

AffineTransform flip = new AffineTransform()

flip.setToScale(1,-1)

Το επόμενο βήμα είναι να μεταφέρουμε τους άξονες έτσι ώστε η αρχή των αξόνων να βρίσκεται στην κάτω αριστερά γωνία του παραθύρου. Χρησιμοποιούμε τον μετασχηματισμό translate γράφοντας :

AffineTransform lift = new AffineTransform()

lift.setToTranslation(xOffset,windowHeight - yOffset)

Τα x μετακινούνται κατά τον παράγοντα xOffset ενώ τα y κατά τον παράγοντα windowHeight-yOffset. Η μεταβλητή windowHeight αντιπροσωπεύει το ύψος του παραθύρου. Τα xOffset και yOffset χρησιμοποιούνται έτσι ώστε να μην συμπέσει το (0,0) σημείο με την κάτω αριστερά γωνία του παραθύρου. Μετακινούμε το (0,0) προς τα δεξιά και πάνω για να υπάρχει χώρος να σχεδιαστούν οι άξονες και τα νούμερα. Για να συνδυάσουμε τους 2 μετασχηματισμούς γράφουμε :

flip.preConcatenate(lift)

με αυτή την εντολή ο μετασχηματισμός lift συνδυάζεται με τον μετασχηματισμό flip σε επίπεδο πολλαπλασιασμού πινάκων. Η προτεραιότητα δίνεται στην πράξη από αριστερά. Αυτό σημαίνει ότι πρώτα γίνεται ο μετασχηματισμός flip και μετά ο μετασχηματισμός lift. Το τελικό αποτέλεσμα αποθηκεύεται στον μεταβλητή flip.
Για να εφαρμόσουμε τον σύνθετο μετασχηματισμό στο αντικείμενο Graphics2D γράφουμε :

g2.transform(flip)

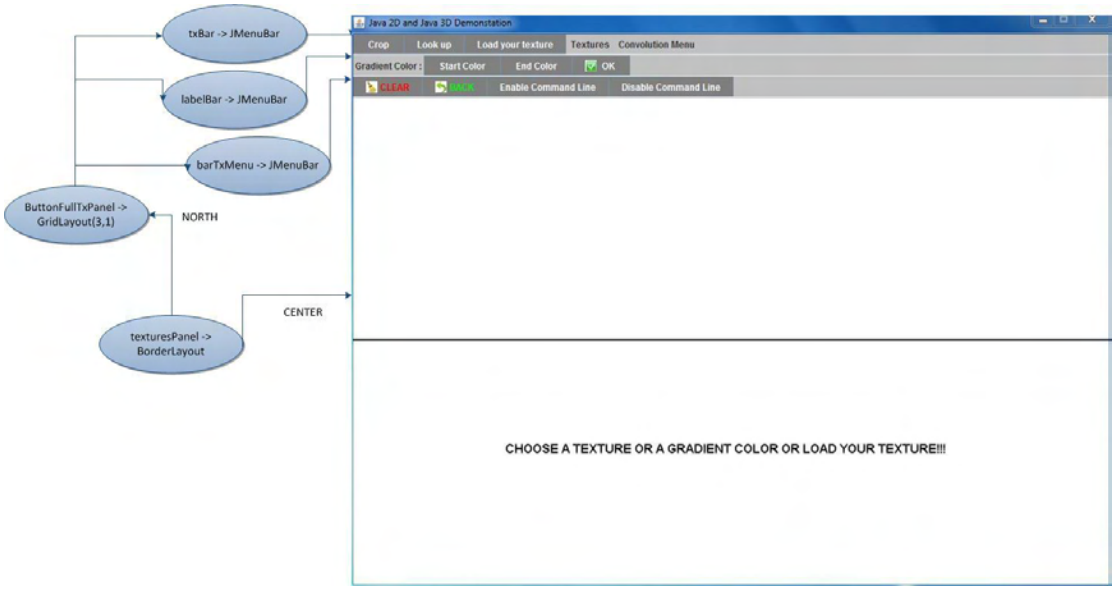
με αυτό τον τρόπο σε όλα τα σχήματα που δημιουργούνται εφαρμόζεται ο μετασχηματισμός flip και έπειτα σχεδιάζονται στην οθόνη του η/υ δίνοντας την αίσθηση ότι τα σχήματα σχεδιάζονται βάση του καρτεσιανού συστήματος συντεταγμένων.
Οι άξονες όπως και οι αριθμοί που εμφανίζονται στους άξονες έχουν σχεδιαστεί με την βοήθεια της συνάρτησης drawSimpleCoordinateSystem() που μπορεί να βρεθεί στα παραδείγματα του βιβλίου «Intoduction to Computer Graphics Using java 2D and 3D» του συγγραφέα Frank Klawonn όπως και στο Παράρτημα Α [Παράρτημα Α].

9. Java 2D – 5η Επιλογη (Crop and Textures)

9.1 Παρουσιαση

Η πέμπτη επιλογή της εφαρμογής είναι το αντικείμενο της κλάσης TexturesPanel. Είναι ένα panel που περιέχει Swing components όπως κουμπιά και ετικέτες. Περιέχει επίσης την περιοχή σχεδίασης που είναι χωρισμένη σε δύο μέρη. Στο κάτω μέρος της περιοχής σχεδίασης εφαρμόζεται η επιλογή του χρήστη που μπορεί να είναι μία υφή από αυτές που παρέχονται από την εφαρμογή, μία υφή που μπορεί να την φορτώσει ο ίδιος ή ένα διαβαθμισμένο χρώμα. Τις υφές όπως και το διαβαθμισμένο χρώμα μπορεί να τα αλλοιώσει εφαρμόζοντας πάνω τους διάφορα φίλτρα όπως εύρεση των ακμών, θόλωμα, όξυνση των χρωμάτων, αλλαγή της τιμής των χρωμάτων, αντιστροφή των χρωμάτων. Επίσης ο χρήστης μπορεί να περικόψει μέρος της εικόνας που έχει δημιουργήσει σε σχήμα ορθογωνίου. Η περικομμένη εικόνα εμφανίζεται στο πάνω μέρος της περιοχής σχεδίασης. Τέλος παρέχονται οι λειτουργίες που περιγράφηκαν στο πέμπτο κεφάλαιο : καθαρισμός της περιοχής σχεδίασης και επιστροφή στα εξώφυλλο [5.4.9] και η γραμμή εντολών [5.4.11].

Το TexturesPanel έχει custom δομή που δημιουργείται με border Layout, και grid Layout.



Εικονα 98 texturesPanel layout

9.2 Λειτουργικοτητα

9.2.1 Textures – Υφες

Εικόνες μπορούν να χρησιμοποιηθούν σαν υφές για να «γεμίσουν» ένα σχήμα, μία περιοχή, μία buffered Image ή πιο γενικά ένα μέρος της οθόνης. Στην εφαρμογή ο χρήστης μπορεί να επιλέξει ανάμεσα σε πέντε έτοιμες υφές που χρησιμοποιούνται για να γεμίσουν μία buffered Image. Όπως έχει ήδη αναφερθεί όλη η σχεδίαση γίνεται σε μία buffered Image την in_bi η οποία ανανεώνεται συνεχώς. Σε αυτή την επιλογή δημιουργείται μία νέα buffered Image η top_bi η οποία

χρησιμοποιείται ως η περιοχή που θα γεμίσει με την ανάλογη υφή. Η top_bi δημιουργείται περίπου στη μέση της in_bi έχοντας διαστάσεις ενός ορθογωνίου που καλύπτει το κάτω μέρος της in_bi. Γίνεται μερική επικάλυψη της in_bi από την top_bi κάθε φορά που ανανεώνεται μία από τις 2 buffered Images. Για να δημιουργηθεί η top_bi γράφουμε:

BufferedImage top_bi = (BufferedImage)this.createImage(1000,350)

Η πρώτη παράμετρος (1000) είναι το πλάτος της top_bi που αντιστοιχεί στο πλάτος της περιοχής σχεδίασης ενώ η δεύτερη παράμετρος (350) είναι το ύψος της top_bi που αντιστοιχεί στα 7/15 του συνολικού ύψους της περιοχής σχεδίασης που είναι 750. Έτσι έχουμε δημιουργήσει μία εικόνα στην μνήμη του η/υ σε σχήμα ορθογώνιο με διαστάσεις 1000x350. Έπειτα δημιουργείται ένα αντικείμενο Graphics2D για την εικόνα αυτή γράφοντας :

Graphics2D gbi = top_bi.createGraphics()

Χρησιμοποιούμε αυτό το αντικείμενο με τις μεθόδους set Color() και fill() για να χρωματίσουμε και να σχεδιάσουμε σχήματα πάνω στην buffered image. Στην συγκεκριμένη εικόνα δημιουργούμε ένα λευκό ορθογώνιο στις διαστάσεις της top_bi το οποίο λειτουργεί ως λευκό φόντο.

gbi.setColor(Color.WHITE)

gbi.fillRect(0,400,this.getWidth(),350)

Προσθέτουμε επίσης μία γραμμή για να ξεχωρίζει τις δύο buffered Images και κείμενο για να γίνει κατανοητό στον χρήστη που θα εμφανιστεί η υφή που θα επιλέξει.

gbi.setColor(Color.BLACK)

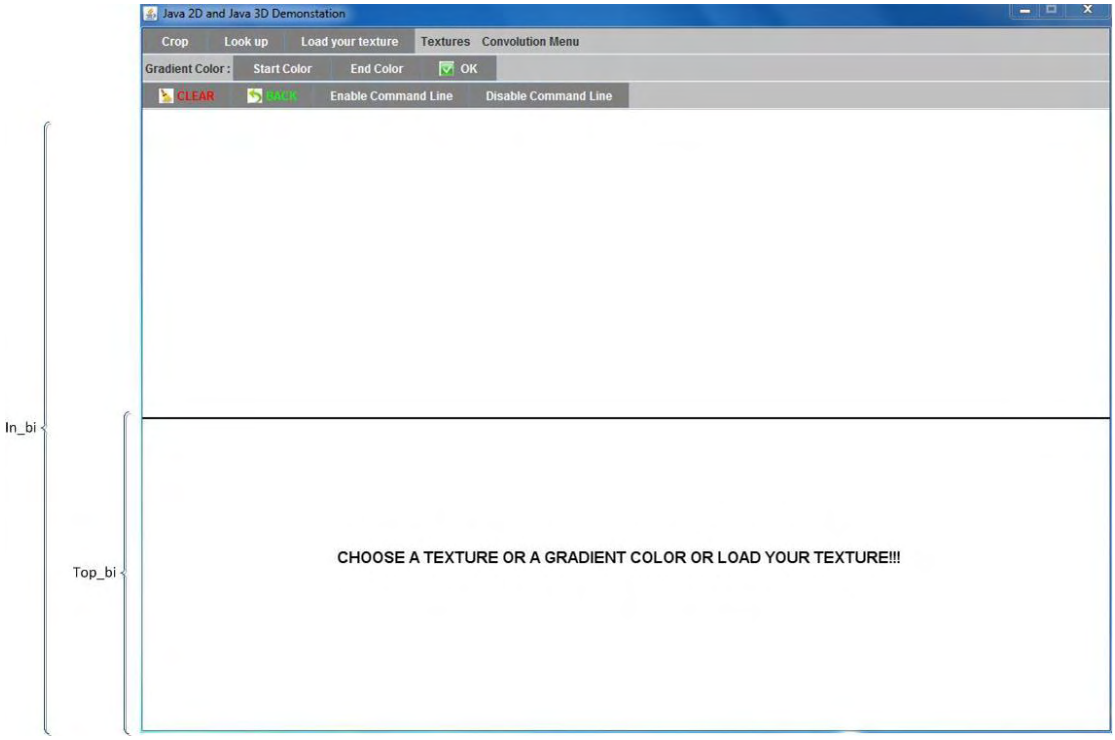
drawLine(0,0,1000,0,gbi)

gbi.drawString("CHOOSE A TEXTURE OR A GRADIENT COLOR OR LOAD YOUR TEXTURE!!!",200,150)

έπειτα για να κάνουμε ορατό το λευκό ορθογώνιο με την γραμμή και το κείμενο μεταφέρουμε / αντιγράφουμε την buffered Image στην οθόνη του η/υ με την εντολή :

g2.drawImage (top_bi,null,0,400)

όπου g2 το αντικείμενο που παίρνει ως όρισμα η μέθοδος paint Component(). Η top_bi θα ζωγραφιστεί στην οθόνη στο ορθογώνιο που αντιστοιχεί η πάνω αριστερή γωνία του στις συντεταγμένες (0 , 400) και η κάτω δεξιά του γωνία στις συντεταγμένες (0 + 1000 , 400 + 350).



ΕΙΚΟΝΑ 99 IN_BI ΕΠΙΚΑΛΥΠΤΟΜΕΝΗ BUFFERED IMAGE ΑΠΟ ΤΗΝ TOP_BI

Για να δημιουργήσουμε μία υφή αρχικά φορτώνουμε την εικόνα σε ένα αντικείμενο της κλάσης Image με όνομα texture1 γράφοντας :

Image texture1 = new javax.swing.ImageIcon("image1.jpg").getImage()

Δημιουργούμε ένα νέο αντικείμενο τύπου Graphics2D για την top_bi έτσι ώστε να ανανεωθεί ολόκληρη η εικόνα:

Graphics2D text1 = top_bi.createGraphics()

Σχεδιάζουμε την υφή texture1 πάνω στην top_bi με την βοήθεια του αντικειμένου text1 και την εμφανίζουμε στην οθόνη του η/υ γράφοντας :

text1.drawImage(texture1,0,0,WIDTH,350,null)

Στο panel αυτό κάθε φορά που γίνεται μία πράξη πρέπει να ανανεωθούν και οι 2 ή η 1 από τις 2 buffered Images. Για να ξέρουμε ανάλογα με την κάθε πράξη ποιες buffered Images πρέπει να ανανεώσουμε καλούμε τη μέθοδο get Stat() η οποία ανάλογα με το όρισμά της ανανεώνει μία μεταβλητή που καθορίζει την παραπάνω λειτουργία. Στην συγκεκριμένη περίπτωση η μεταβλητή είναι η Stat.NOTHING.

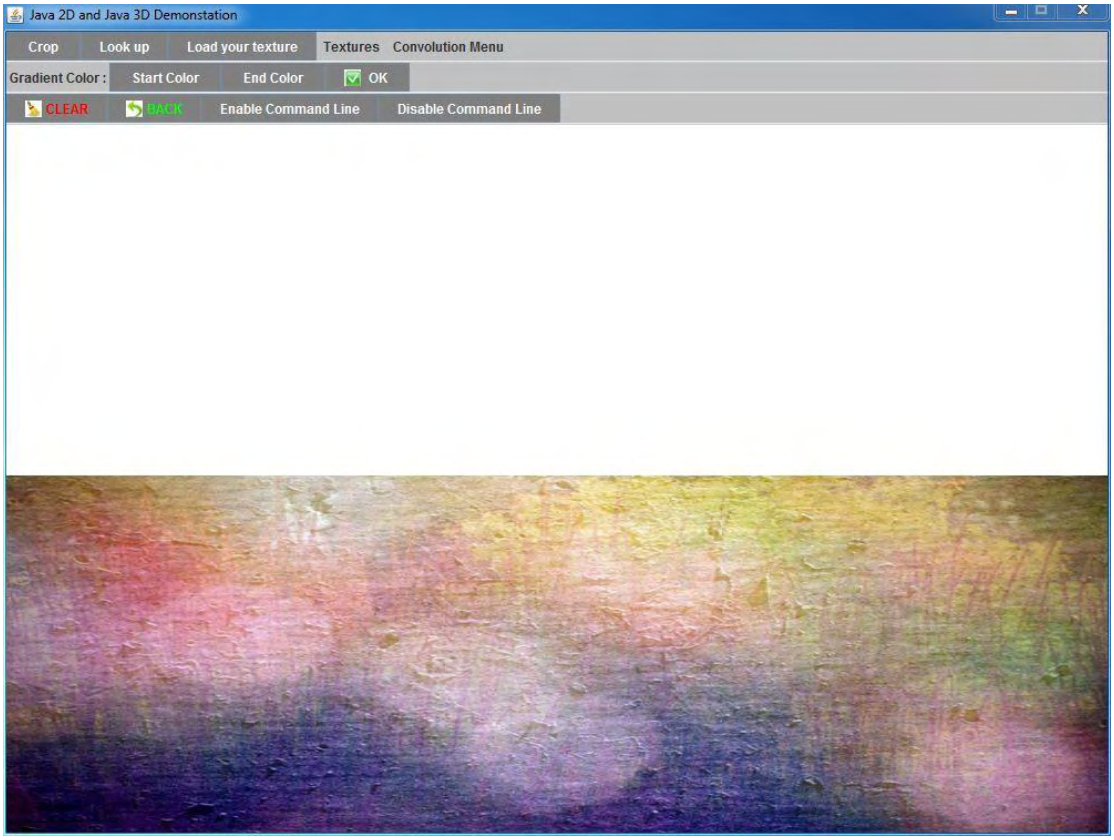
texturesPanel.getStat(Stat.NOTHING)

Αφού καθορίσαμε ποιες buffered Images πρέπει να ανανεωθούν καλούμε την μέθοδο repaint() για το panel texturesPanel η οποία εκτελεί την συνάρτηση paint Component() για να γίνουν οι αλλαγές.

texturesPanel.repaint()

Με την μεταβλητή Stat.NOTHING δεν γίνεται καμία αλλαγή στην in_bi.

Στην εφαρμογή ο χρήστης μπορεί να επιλέξει από το μενού «Textures» πέντε διαθέσιμες υφές. Οι υφές εμφανίζονται στον κάτω ορθογώνιο (top_bi buffered Image).



ΕΙΚΟΝΑ 100 TEXTURE ΝΟ 4

9.2.2 LOAD TEXTURE – ΦΟΡΤΩΣΗ ΥΦΗΣ

Εκτός από τις προκαθορισμένες υφές ο χρήστης μπορεί να φορτώσει μία εικόνα η οποία να λειτουργήσει ως υφή από τον δικό του η/υ. Για να φορτώσουμε μία εικόνα ακολουθούμε την διαδικασία όπως έχει αναφερθεί στην παράγραφο 5.4.10 [5.4.10]. Αρχικά προσδιορίζουμε το φίλτρα για τα αρχεία μιας που θέλουμε να εμφανίζονται μόνο τύπος αρχείων .jpg.

JFileChooser load = new JFileChooser()

load.setFileFilter(new FileNameExtensionFilter("JPEG File", "jpg"))

Αφού ελέγξουμε την επιστρεφόμενη τιμή από το JFileChooser δημιουργούμε ένα αρχείο στο οποίο φορτώνουμε το αντίστοιχο αρχείο που επιλέχθηκε μέσω του JFileChooser. Έπειτα δημιουργούμε μία καινούρια buffered image (img) στην οποία αποθηκεύουμε την εικόνα που φορτώθηκε. Δημιουργούμε ένα νέο αντικείμενο για την ήδη υπάρχουσα buffered image (την εικόνα στην οποία εμφανίζεται η υφή – top_bi) το οποίο το χρησιμοποιούμε για να αντιγράψουμε την καινούρια buffered image (img) στην παλιά (top_bi). Αυτό γίνεται γιατί δεν γνωρίζουμε τις διαστάσεις της εικόνας που φορτώνεται με αποτέλεσμα αν την αντιγράψουμε κατευθείαν την φορτωμένη εικόνα στην buffered image της εικόνας σχεδίασης να εμφανιστεί με τις κανονικές της διαστάσεις δηλαδή πολύ μικρή ή πολύ μεγάλη. Έτσι δημιουργούμε μία καινούρια buffered image (img) με τις

διαστάσεις της εικόνας που φορτώθηκε την οποία την τροποποιούμε στις διαστάσεις της top_bi (buffered image – top_bi).Τέλος καλούμε την μέθοδο repaint() για το συγκεκριμένο panel για να ανανεώσει την οθόνη γράφοντας την ανανεωμένη buffered image (top_bi) στην οθόνη.

```
if(loadtx.showOpenDialog(texturesPanel)==JFileChooser.APPROVE_OPTION){
  try {

        File fileLoad = loadtx.getSelectedFile();
        BufferedImage img = ImageIO.read(fileLoad);
        Graphics2D df = top_bi.createGraphics();
        df.drawImage(img,0,0,WIDTH,350,null);
        texturesPanel.getStat(Stat.NOTHING);
        texturesPanel.repaint();

  } catch (IOException ex) {
        System.out.println(ex);
        JOptionPane.showMessageDialog(texturesPanel, ex);
  }
}
```

Καλούμε τη μέθοδο getStat με την μεταβλητή Stat.NOTHING για να γνωρίζουμε ποια buffered image πρέπει να ανανεωθεί και έπειτα πραγματοποιείται η ανανέωση με την εντολή repaint().Σε περίπτωση που γίνει κάτι λάθος κατά τη διάρκεια της φόρτωσης της εικόνας εμφανίζεται το αντίστοιχο μήνυμα στην οθόνη.

Στην εφαρμογή για να φορτώσουμε μία εικόνα πατάμε το κουμπί «Load your Texture».

9.2.3 Crop – Περικοπη Εικονας

Η περικοπή μίας εικόνας αναφέρεται στην απομάκρυνση των εξωτερικών τμημάτων της εικόνας. Στην εφαρμογή γίνεται περικοπή της υφής που έχει εφαρμοστεί στην top_bi buffered Image σε ορθογώνιο σχήμα με σκοπό να απομονώσουμε τμήμα της εικόνας το οποίο εμφανίζεται στην in_bi buffered Image ακριβώς πάνω από την top_bi buffered Image. Η υφή είναι μία bufferedImage. Για να εφαρμόσουμε περικοπή μίας buffered Image γράφουμε :

BufferedImage croppedImage = originalImage.getSubimage(x , y , width , height)

Όπου croppedImage το όνομα της νέας buffered Image που δημιουργείται μετά την περικοπή της originalImage. Οι συντεταγμένες x,y ανήκουν στην πάνω αριστερή γωνία του ορθογωνίου στου

οποίου το σχήμα περικόπτουμε την εικόνα. Το πλάτος του ορθογωνίου είναι x + width και το ύψος είναι y + height. Στην εφαρμογή η originalImage είναι η top_bi και η croppedImage είναι η dest.

BufferedImage dest= top_bi.getSubimage(x, y, width, height)

Ζωγραφίζουμε την dest buffered Image με την μέθοδο draw Image(). Την μέθοδο draw Image() την καλούμε για το αντικείμενο Graphics2D που έχει δημιουργηθεί για την in_bi buffered Image με αποτέλεσμα ότι ζωγραφίζουμε να εμφανίζεται πάνω στην in_bi και όχι πάνω στην οθόνη όπως κάναμε μέχρι τώρα.

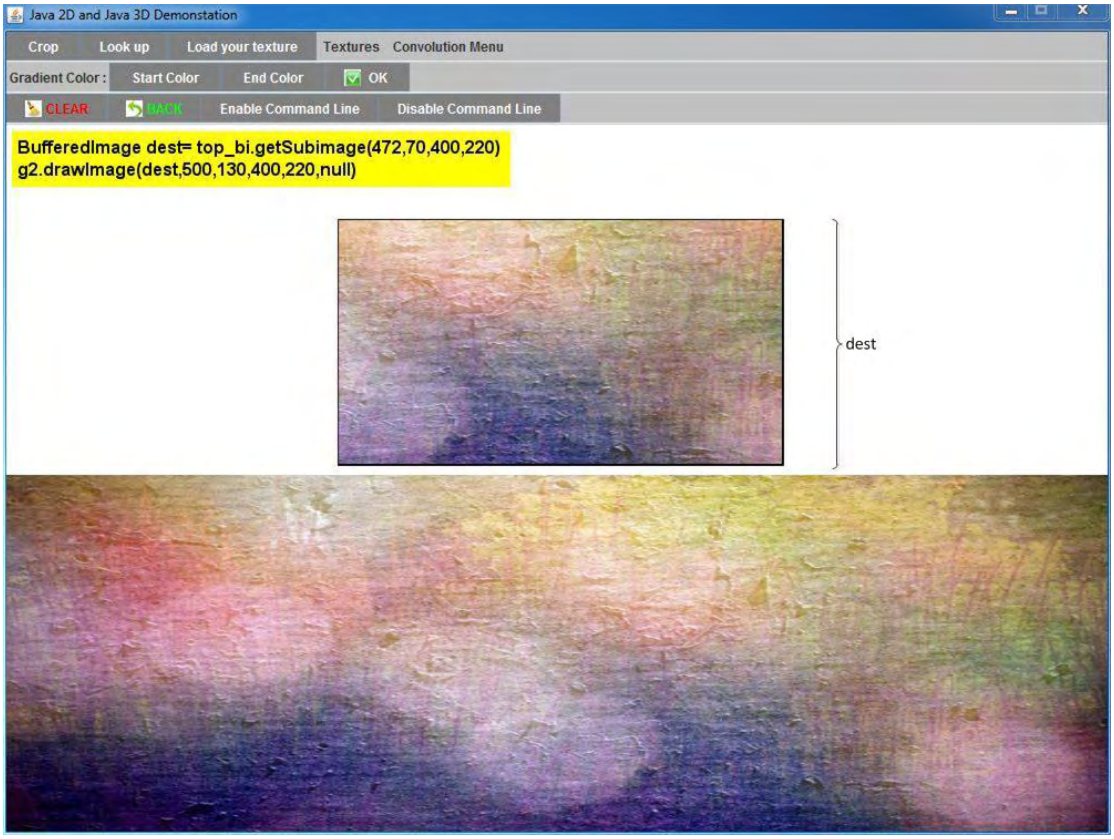
Graphics2D g2 = in_bi.createGraphics()

g2.drawImage(dest,xdraw,ydraw,w,z,null)

Το ορθογώνιο που περιέχει την εικόνα που έχει περικοπεί ζωγραφίζεται έτσι ώστε η πάνω αριστερή γωνία του να βρίσκεται στις συντεταγμένες xdraw, ydraw και να έχει πλάτος xdraw + w και ύψος ydraw + z. Για να γνωρίζουμε ποια buffered image πρέπει να ανανεώσουμε καλούμε τη μέθοδο get Stat() με την μεταβλητή Stat.CROP που δείχνει ότι πρέπει να ανανεωθεί η in_bi.

texturesPanel.getStat(Stat.CROP)

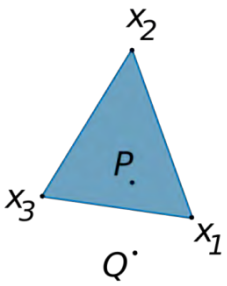
Στην εφαρμογή για να περικόψουμε την εικόνα πατάμε το κουμπί «Crop» και πηγαίνουμε πάνω από την top_bi buffered Image. Κάνοντας click εμφανίζεται ένα βοηθητικό ορθογώνιο στο μέγεθος της εικόνας που θέλουμε να περικόψουμε. Το μέγεθος του ορθογωνίου παραμένει ίδιο ανανεώνοντας όμως τις συντεταγμένες του σύμφωνα με την κίνηση του ποντικιού. Όταν σταματήσουμε να πατάμε το κουμπί του ποντικιού τότε εμφανίζεται η εικόνα που έχουμε επιλέξει να περικόψουμε σύμφωνα με τις τελευταίες συντεταγμένες. Το ορθογώνιο εφόσον είναι βοηθητικό δεν ζωγραφίζεται πάνω στην top_bi για να μην αλλοιώσει την υφή. Η διαδικασία για να ζωγραφιστεί δυναμικά το ορθογώνιο είναι ίδια με την διαδικασία δημιουργίας της γραμμής με αλληλεπίδραση με τον χρήστη με μία μικρή διαφορά. Κατά την διάρκεια που έχουμε πατημένο το κουμπί του ποντικιού το ορθογώνιο ζωγραφίζεται στην οθόνη του η/υ και μόλις σταματάμε να πατάμε το κουμπί του ποντικιού αντί να ζωγραφίζεται στην top_bi buffered Image, όπως γίνεται στην περίπτωση της γραμμής, ζωγραφίζεται πάλι στην οθόνη του η/υ με αποτέλεσμα να επικαλύπτεται από την top_bi buffered Image που ανανεώνεται. Η διαδικασία έχει τροποποιηθεί για να μην γίνεται ορατό το τελικό ορθογώνιο.



Εικονα 101 dest εικονα που εχει περικοπει – texture νο 4

9.2.4 Gradient Color – Διαβαθμισμενο Χρωμα

Στην συγκεκριμένη εφαρμογή έχει χρησιμοποιηθεί το μοντέλο χρωμάτων RGB όπως έχει παρουσιαστεί στην παράγραφο 5.4.6 [5.4.6]. Στο μοντέλο αυτό ένα χρώμα μπορεί να σχετιστεί με ένα διάνυσμα (r,g,b) ∈ $[0,1]^3$. Αυτή η αναπαράσταση επιτρέπει τον ορισμό του κυρτού συνδυασμού χρωμάτων. Κυρτός συνδυασμός σημείων (μπορεί να είναι και διανυσμάτων) $x1, x2, \ldots, xn$ είναι ένα σημείο της μορφής $a1x1 + a2x2 + \cdots + anxn$ που οι αριθμοί $ai \geq 0$ $και$ $a1 + a2 + \cdots + an = 1$. Αν έχουμε μόνο δύο σημεία όλοι οι κυρτοί συνδυασμοί τους βρίσκονται πάνω στο ευθύγραμμο τμήμα που τα ενώνει. Πιο γενικά όλοι οι κυρτοί συνδυασμοί των σημείων $x1, x2, \ldots, xn$ βρίσκονται στο convex hull [6.2.4] των δεδομένων σημείων.



Εικονα 102 το σημειο P ειναι κυρτος συνδυασμος των x1, x2, x3 (βρισκεται εντος του convex hull των x1,x2,x3) ενω το Q δεν ειναι κυρτος συνδυασμος των x1,x2,x3 (εκτος του convex hull)

Μία εφαρμογή του κυρτού συνδυασμού χρωμάτων είναι το διαβαθμισμένο χρώμα. Με τον όρο διαβαθμισμένο χρώμα εννοούμε όταν θέλουμε να ζωγραφίσουμε μία περιοχή με χρώμα που αλλάζει και όχι με ομογενές χρώμα. Πρέπει να οριστούν δύο χρώματα (r0,g0,b0) και (r1,g1,b1) για δύο σημεία p0, p1. Το χρώμα (r0,g0,b0) χρησιμοποιείται για το σημείο p0 και το χρώμα (r1,g1,b1) για το σημείο p1. Για τα σημεία πάνω στην γραμμή που ενώνει τα p0, p1 χρησιμοποιείται ο αντίστοιχος κυρτός συνδυασμός. Για το σημείο p = (1-a)p1 + a p0 χρησιμοποιείται το χρώμα (1-α)·(r0,g0,b0) + α · (r1,g1,b1).
Για τα διαβαθμισμένα χρώματα χρησιμοποιούμε την κλάση Gradient Paint.

GradientPaint gradPaint = new GradientPaint(x0,y0,color0,x1,y1,color1,repeat)

Όπου δημιουργείται ένα διαβαθμισμένο χρώμα μεταξύ των σημείων (x0,y0) και (x1,y1) που αντιστοιχούν στα χρώματα color0 και color1. Όπως αναφέρθηκε παραπάνω τα σημεία μεταξύ του ευθυγράμμου τμήματος που ενώνει τα p0,p1 αποκτούν το χρώμα τους βάση του κυρτού συνδυασμού των χρωμάτων color0 και color1. Το ίδιο διαβαθμισμένο χρώμα εφαρμόζεται και στις παράλληλες γραμμές με την γραμμή που ενώνει τα (x0,y0) και (x1,y1). Η Boolean μεταβλητή repeat καθορίζει αν το διαβαθμισμένο χρώμα θα επαναληφθεί πριν το (x0,y0) και μετά το (x1,y1). Αν η repeat είναι false τότε πριν το (x0,y0) τα pixel βάφονται με το χρώμα color0 και τα pixel μετά το (x1,y1) βάφονται με το χρώμα color1. Αν η repeat είναι true τότε το διαβαθμισμένο χρώμα επαναλαμβάνεται. Για να ενεργοποιηθεί το διαβαθμισμένο χρώμα καλούμε την μέθοδο set Paint() για ένα αντικείμενο Graphics2D, στην συγκεκριμένη περίπτωση καλούμε τη μέθοδο για το αντικείμενο Graphics2D που χρησιμοποιεί η top_bi buffered Image μιας που εκείνη θέλουμε να βάψουμε με το διαβαθμισμένο χρώμα.

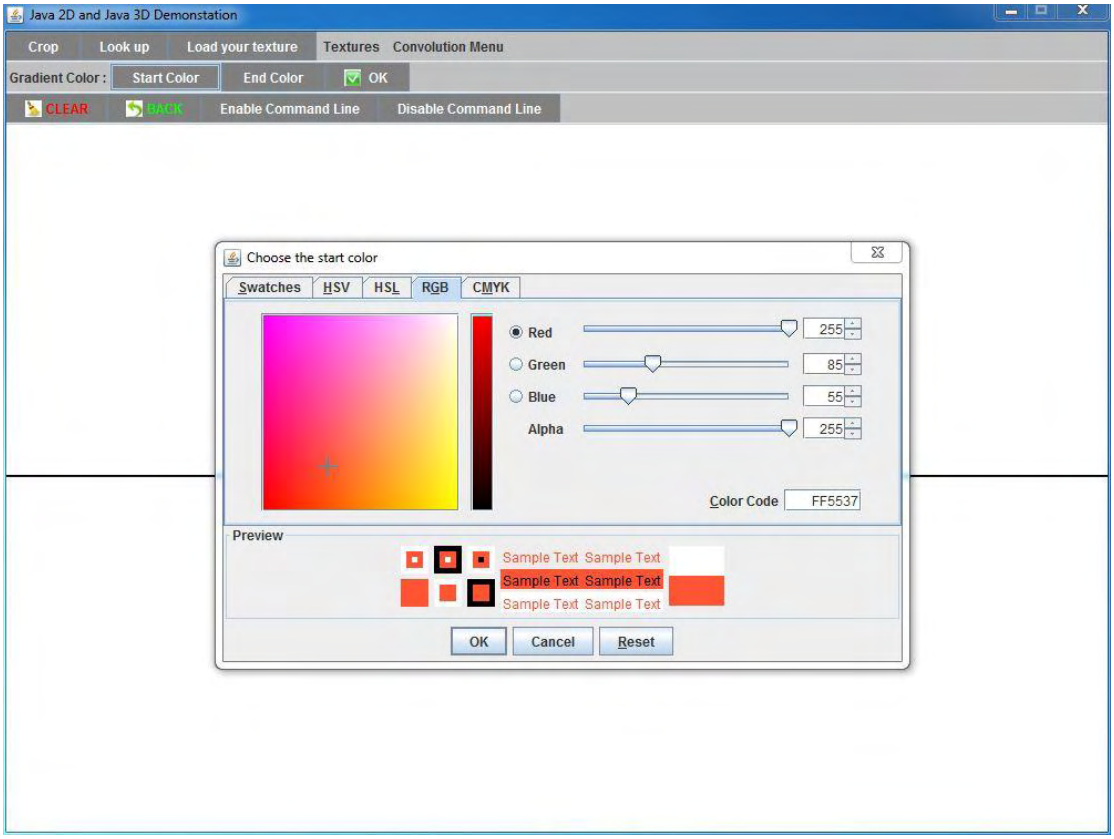
g2.setpaint(gradpaint)

Για να ξέρουμε ποια buffered Image πρέπει να ανανεώσουμε καλούμε τη μέθοδο get Stat() με την μεταβλητή Stat.GRADIENT όπου δείχνει ότι πρέπει να ανανεωθεί μόνο η top_bi. Έπειτα πραγματοποιείται η ανανέωση με την εντολή repaint().

texturesPanel.getStat(Stat.GRADIENT)

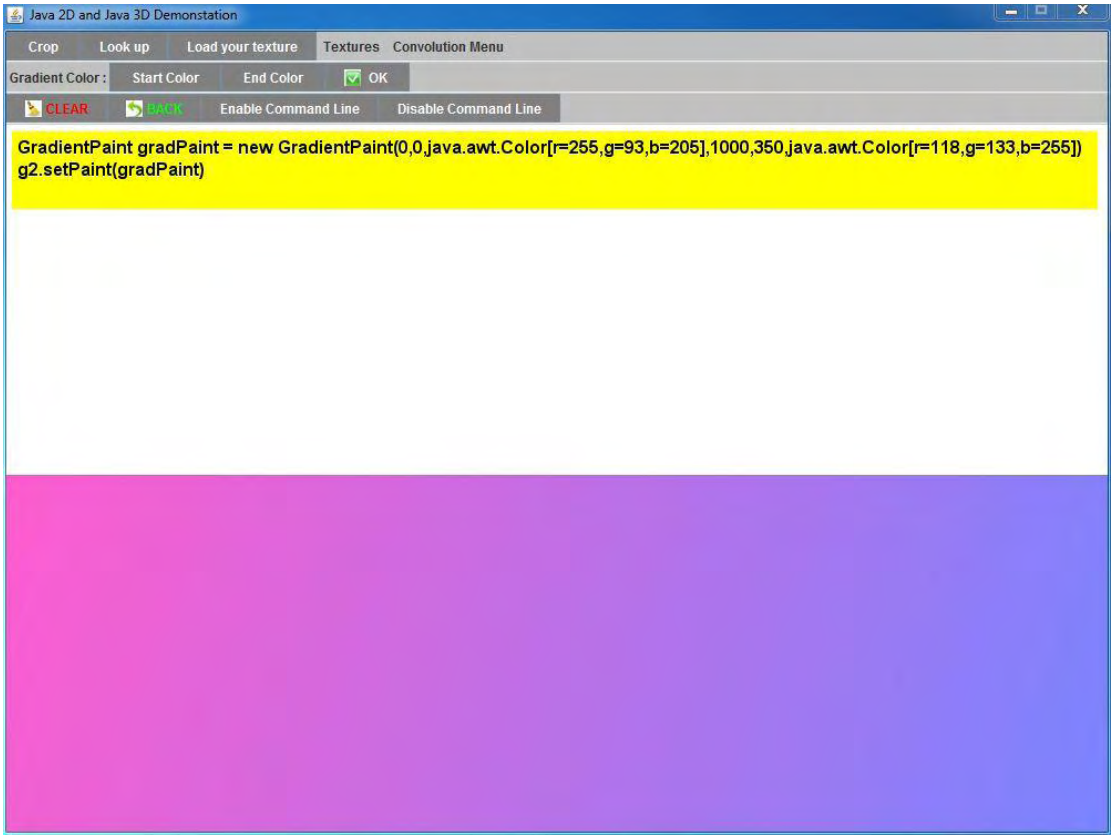
texturesPanel.repaint()

Στην εφαρμογή ο χρήστης πατώντας τα κουμπιά «Start Color» και «End Color» μπορεί να διαλέξει από το παράθυρο επιλογής χρώματος που εμφανίζεται το color0 και το color1 αντίστοιχα. Το παράθυρο επιλογής χρώματος δημιουργείται με την βοήθεια της κλάσης JColorChooser.



ΕΙΚΟΝΑ 103 ΠΑΡΑΘΥΡΟ ΕΠΙΛΟΓΗΣ ΧΡΩΜΑΤΟΣ

Έπειτα πατώντας το κουμπί «OK» μπορεί να εφαρμόσει το διαβαθμισμένο χρώμα που δημιούργησε στην top_bi εικόνα.



ΕΙΚΟΝΑ 104 ΔΙΑΒΑΘΜΙΣΜΕΝΟ ΧΡΩΜΑ

9.2.5 LOOKUP OPERATION– LOOKUP ΛΕΙΤΟΥΡΓΙΑ

Μπορούμε να υλοποιήσουμε μία λειτουργία Lookup για να τροποποιήσουμε μία buffered image με την βοήθεια της κλάσης Lookup Op. Για τις buffered images η λειτουργία Lookup ενεργεί στο χρώμα και στην παράμετρο alpha, η οποία παραμένει σταθερή στα πλαίσια αυτής της επιλογής της εφαρμογής. Πιο συγκεκριμένα χρησιμοποιείται η κλάση Lookup Op για να αντιστρέψει τις τιμές των χρωμάτων όλων των pixel. Η λειτουργία αυτή πραγματοποιείται με την βοήθεια ενός Lookup table αντικειμένου, που μπορεί να είναι ένας μόνο πίνακας ή πολλαπλοί πίνακες ανάλογα με τις απαιτήσεις της λειτουργίας που θέλουμε να υλοποιήσουμε. Ο Lookup table μπορεί να είναι αντικείμενο της κλάσης Byte Lookup Table ή Short Lookup Table ανάλογα με τους αριθμούς που χρησιμοποιούμε. Στην εφαρμογή χρησιμοποιείται η κλάση Byte Lookup Table. Το αντικείμενο της κλάσης Lookup Op καλείται για να αντικαταστήσει τις τιμές των χρωμάτων των pixel με τις αντίστοιχες τιμές που έχουν αποθηκευτεί στον πίνακα Lookup Table στον ίδιο δείκτη. Είναι δυνατό να αλλάξεις τις τιμές των χρωμάτων χρησιμοποιώντας οποιονδήποτε αλγόριθμο αντικατάστασης. Στην εφαρμογή χρησιμοποιείται ένας μόνο πίνακας Lookup Table ο οποίος εφαρμόζεται και στις τρεις συνιστώσες του χρώματος (red, green, blue). Όπως αναφέρθηκε θα μπορούσαμε να χρησιμοποιήσουμε περισσότερους πίνακες έτσι ώστε οι τιμές του χρώματος κάθε συνιστώσας να αντικατασταθούν με διαφορετικό τρόπο. Για να δημιουργήσουμε αυτήν την λειτουργία αρχικά δημιουργούμε ένα πίνακα από byte μεγέθους 256 όσο και το εύρος κάθε συνιστώσας του χρώματος:

byte lut[] = new byte[256]

Έπειτα μετατρέπουμε τις τιμές του πίνακα lut. Η τιμή του κάθε κελιού του πίνακα είναι ίση με 255 μείον τον δείκτη του πίνακα. Αυτός είναι ο αλγόριθμος αντικατάστασης χρώματος που θα εφαρμόσουμε.

for (int j=0; j<256; j++) {

lut[j] = (byte)(255-j) }

Δημιουργούμε ένα νέο αντικείμενο της κλάσης Byte Lookup Table περνώντας ως παράμετρο τον πίνακα lut που δημιουργήσαμε.

ByteLookupTable blut = new ByteLookupTable(0, lut)

Η πρώτη παράμετρος είναι το offset που καθιστά δυνατόν να προσδιοριστεί ένα σύνολο 256 διαδοχικών στοιχείων στον πίνακα lut όταν το μέγεθος του πίνακα είναι μεγαλύτερο από 256. Στην συγκεκριμένη περίπτωση είναι 0 γιατί ο πίνακας lut έχει μέγεθος 256 στοιχείων. Ύστερα δημιουργούμε ένα αντικείμενο Lookup Op γράφοντας :

LookupOp lop = new LookupOp(blut, null)

Η πρώτη παράμετρος είναι ο πίνακας αντικατάστασης blut και η δεύτερη παράμετρος αντιστοιχεί στην απόδοση της εικόνας (αν θέλουμε ή όχι να δημιουργηθεί η νέα εικόνα με την τεχνική antialiasing για παράδειγμα [5.4.8]). Στην συγκεκριμένη περίπτωση η δεύτερη μεταβλητή είναι null. Η λειτουργία που έχουμε δημιουργήσει εφαρμόζεται στην top_bi buffered image. Η αρχική εικόνα παύει να υπάρχει και πλέον είναι ορατή μόνο η εικόνα μετά την εφαρμογή του Lookup Op. Για να γίνει αυτό πρέπει να δημιουργήσουμε ένα νέο αντικείμενο Graphics2D για την ήδη υπάρχουσα top_bi buffered image γράφοντας :

Graphics2D lu = top_bi.createGraphics()

Για το αντικείμενο lu καλούμε την μέθοδο draw Image() για να εμφανιστεί στην οθόνη η εικόνα top_bi αφού έχει εφαρμοστεί σε αυτή η λειτουργία Lookup Op (δηλαδή η lop).

lu.drawImage(top_bi, lop, 0, 0)

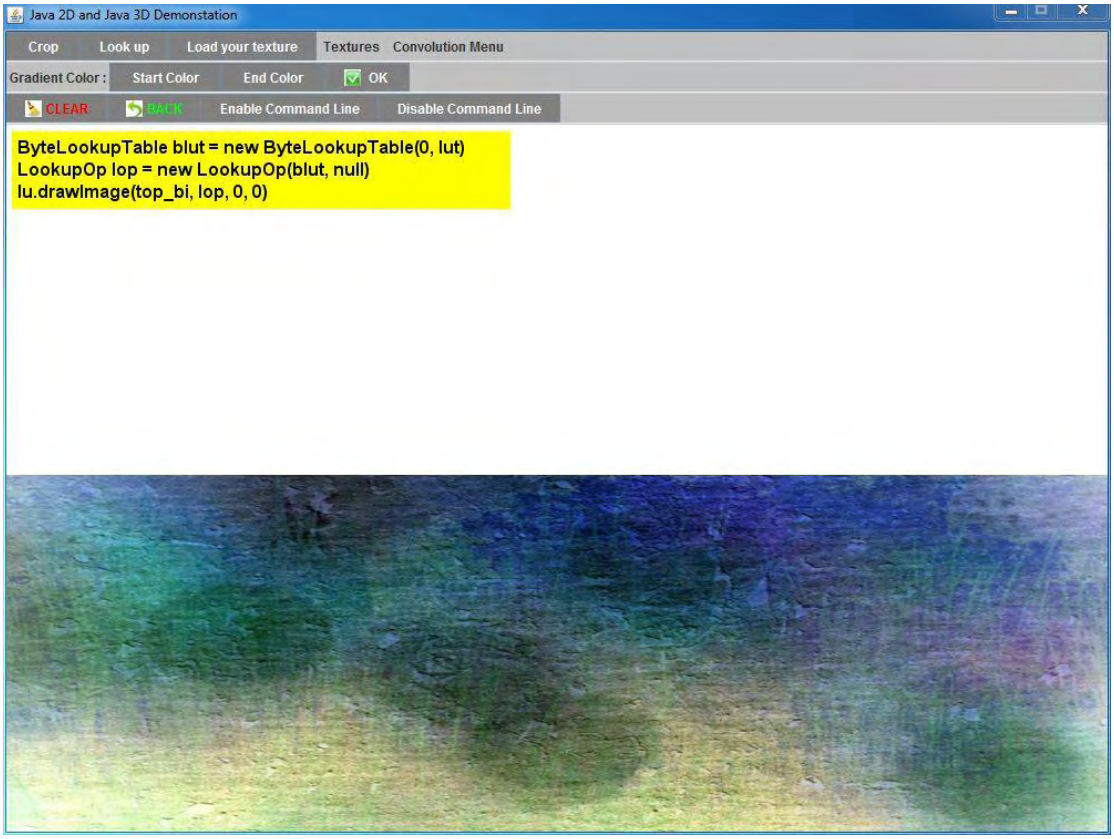
Η πρώτη παράμετρος είναι η εικόνα η οποία θα εμφανιστεί στη οθόνη. Η δεύτερη παράμετρος είναι τύπου Buffered Image Op το οποίο είναι interface το οποίο υλοποίει και τη λειτουργία Lookup Op. Το (0, 0) είναι το σημείο που θα σχεδιαστεί η πάνω αριστερή γωνία της εικόνας top_bi.

Για να ξέρουμε ποια buffered Image πρέπει να ανανεώσουμε καλούμε τη μέθοδο get Stat() με την μεταβλητή Stat.LOOKUP όπου δείχνει ότι πρέπει να ανανεωθεί μόνο η top_bi. Έπειτα πραγματοποιείται η ανανέωση με την μέθοδο repaint().

texturesPanel.getStat(Stat.LOOKUP)

texturesPanel.repaint()

Στην εφαρμογή πατώντας το κουμπί «Look up» εφαρμόζεται η λειτουργία Lookup Op που περιγράφηκε παραπάνω στην top_bi buffered Image αντιστρέφοντας τις τιμές των χρωμάτων. Αν πατήσουμε ξανά το κουμπί «Look up» (χωρίς να έχουμε εφαρμόσει κάποιο από τα φίλτρα που θα συζητηθούν παρακάτω) λόγω του συγκεκριμένου αλγορίθμου αντικατάστασης που χρησιμοποιήθηκε εμφανίζεται η αρχική εικόνα χωρίς καμία αλλαγή.



ΕΙΚΟΝΑ 105 LOOKUP ΛΕΙΤΟΥΡΓΙΑ – TEXTURE ΝΟ 4

9.2.6 FILTERS – ΦΙΛΤΡΑ

Η εφαρμογή αυτή δίνει την δυνατότητα στον χρήστη να χρησιμοποιήσει κάποια φίλτρα σε μία εικόνα και να δει τι αλλαγές, βελτιώσεις επιφέρουν. Με τα φίλτρα αυτά μπορεί να επιτευχθεί όξυνση των χρωμάτων, εύρεση ακμών, θόλωμα, αλλαγή της τιμής των χρωμάτων. Οι πρώτες 3 λειτουργίες γίνονται με την βοήθεια της κλάσης Convolve Op η οποία υλοποιεί μία συνέλιξη (convolution) μεταξύ των pixel της αρχικής εικόνας. Ως συνέλιξη μπορούμε να ονομάσουμε την διαδικασία η οποία αντικαθιστά κάθε pixel σε μία εικόνα με έναν συνδυασμό του αρχικού pixel με τα γειτονικά pixels. Ο συνδυασμός των pixels καθορίζεται από έναν πίνακα με float τιμές. Οι τιμές αυτές είναι πολλαπλάσια των pixel. Από αυτόν τον πίνακα δημιουργείται ένα αντικείμενο της κλάσης Kernel που χρησιμοποιείται για την εκτέλεση της συνέλιξης. Για παράδειγμα αν ο πίνακας είναι 3x3 όπως ο παρακάτω :

$$\begin{matrix} a & b & c \\ d & e & f \\ g & h & i \end{matrix}$$

Τότε η αντίστοιχη συνέλιξη αντικαθιστά το κεντρικό pixel με το άθροισμα :

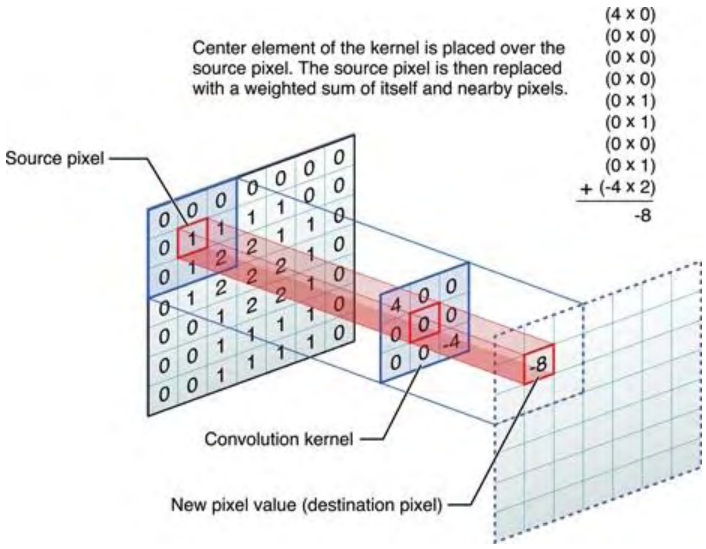
- a φορές την τιμή του pixel στην πάνω αριστερή γωνία από το κεντρικό pixel
- b φορές την τιμή του pixel πάνω από το κεντρικό pixel
- c φορές την τιμή του pixel στην πάνω δεξιά γωνία από το κεντρικό pixel
- κτλ

Αν το άθροισμα των συντελεστών του πίνακα είναι 1 τότε ο kernel δεν επιφέρει καμία αλλαγή στο pixel. Αν το άθροισμα είναι μικρότερο του 1 τότε η νέα εικόνα είναι πιο σκοτεινή από την αρχική αν είναι μεγαλύτερο του 1 τότε είναι πιο φωτεινή.

Ο αντίστοιχος constructor για την λειτουργία Convolve Op είναι :

public ConvolveOp(Kernel kenrnel, int edgeCondition, RenderingHints hints)

Για κάθε pixel έστω p στην αρχική εικόνα εφαρμόζεται ο πίνακας kernel όπως περιγράφηκε παραπάνω και υπολογίζεται η τιμή του pixel έστω p’ στην εικόνα αφού εφαρμοστεί το φίλτρο.



Εικονα 106 Convolve Op λειτουργια για kernel $\begin{matrix} 4 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -4 \end{matrix}$

Η τέταρτη λειτουργία γίνεται με την βοήθεια της κλάσης Rescale Op η οποία χρησιμοποιείται για να πολλαπλασιάσει την τιμή του χρώματος κάθε pixel με έναν παράγοντα που ορίζει ο χρήστης που μπορεί να αυξήσει ή να μειώσει την τιμή του χρώματος (scale factor) και μετά για να προσθέσει μία σταθερά στο γινόμενο αυτό. Μπορούν να υπάρξουν ξεχωριστοί παράγοντες και σταθερές για κάθε μία από τις 3 συνιστώσες του χρώματος (red, green, blue) ή ένας παράγοντας και μία σταθερά και για τις 3 συνιστώσες. Στην εφαρμογή αυτή χρησιμοποιείται ενιαίος παράγοντας και μία σταθερά και για τις 3 συνιστώσες του χρώματος. Ο αντίστοιχος constructor για την λειτουργία Rescale Op είναι :

public RescaleOp(float scaleFactor, float offset, RenderingHints hints)

Για κάθε pixel στην εικόνα έστω p’ μετά την εφαρμογή του φίλτρου θα ισχύει
p’ = (p*scaleFactor) + offset
όπου p το αντίστοιχο pixel στην αρχική εικόνα, scaleFactor ο παράγοντας για την αύξηση ή την μείωση της τιμής του χρώματος και offset η σταθερά.

1. **Edge Detection – Εύρεση Ακμών**
   Η εύρεση ακμών είναι η διαδικασία κατά την οποία γίνεται αναγνώριση των σημείων σε μία εικόνα όπου αλλάζει απότομα η φωτεινότητα ή πιο επίσημα δημιουργούνται ασυνέχειες. Μπορούμε να πούμε ότι ψάχνουμε αυτές τις ακμές και προσπαθούμε να τις επισημάνουμε. Χρειαζόμαστε η νέα

εικόνα να είναι πιο σκοτεινή από την αρχική για αυτό πρέπει το αλγεβρικό άθροισμα των συντελεστών του kernel να είναι μηδενικό. Ο πίνακας που χρησιμοποιήθηκε είναι ο

```
public static final float[] EDGE3x3= {  //  0+(-1)+0+(-1)+4+(-1)+0+(-1)+0=0
    0.0f, -1.0f, 0.0f,
    -1.0f, 4.f, -1.0f,
    0.0f, -1.0f, 0.0f}
```

Για να ορίσουμε την λειτουργία Convolve Op γράφουμε :

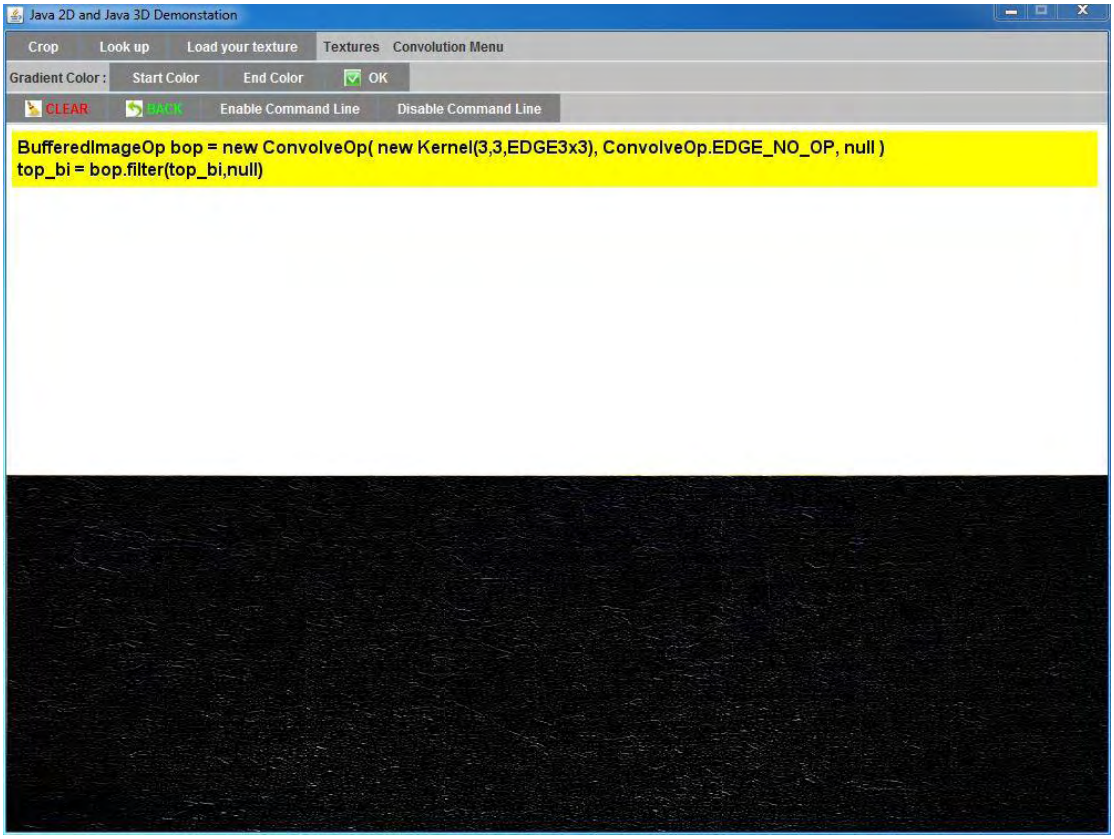
```
BufferedImageOp bop = new ConvolveOp( new Kernel(3,3,EDGE3x3), ConvolveOp.EDGE_NO_OP,
null )
```

Η Convolve Op υλοποιείται από το interface Buffered Image Op. Η πρώτη παράμετρος είναι ο kernel (ο πίνακας) που ορίστηκε παραπάνω, η δεύτερη παράμετρος αντιστοιχεί στην επιλογή να αντιγράφονται τα pixels από τις άκρες της αρχικής εικόνας στην τελική χωρίς να γίνονται αλλαγές σε αυτά. Η τρίτη παράμετρος αναφέρεται στην απόδοση της εικόνας (αν θέλουμε ή όχι να δημιουργηθεί η νέα εικόνα με την τεχνική antialiasing για παράδειγμα [5.4.8]) και είναι null. Χρησιμοποιούμε την μέθοδο filter καλούμενη για την λειτουργία Convolve Op , την bop, που δημιουργήσαμε γράφοντας :

```
top_bi = bop.filter(top_bi,null)
```

Η παραπάνω εντολή εφαρμόζει το φίλτρο που δημιουργήσαμε στην top_bi εικόνα και λόγω της δεύτερης παραμέτρου null δημιουργεί μία νέα buffered image για να αποθηκευτεί το αποτέλεσμα. Η νέα εικόνα μέσω της ανάθεσης γίνεται πάλι η top_bi. Δηλαδή η τελική εικόνα μετά την εφαρμογή του φίλτρου αποθηκεύεται στην ίδια μεταβλητή τύπου buffered image την top_bi. Για να ξέρουμε ποια buffered image πρέπει να ανανεώσουμε καλούμε τη μέθοδο get Stat() με την μεταβλητή Stat.EDGE όπου δείχνει ότι πρέπει να ανανεωθεί μόνο η top_bi. Έπειτα πραγματοποιείται η ανανέωση με την μέθοδο repaint().

```
texturesPanel.getStat(Stat.EDGE)
texturesPanel.repaint()
```



ΕΙΚΟΝΑ 107 ΕΥΡΕΣΗ ΑΚΜΩΝ – TEXTURE NO 4

2. **Sharpen – Όξυνση Χρωμάτων**

Η διαδικασία όξυνσης των χρωμάτων κάνει πιο ζωηρή την εικόνα. Πιο συγκεκριμένα αυξάνει την αντίθεση μεταξύ φωτεινών και σκοτεινών περιοχών για να προβάλει τα σημεία αυτά. Πρέπει να προσέξουμε να μην αλλάξουμε το χρώμα της εικόνας όπως γίνεται με την εύρεση ακμών έτσι πρέπει το αλγεβρικό άθροισμα των συντελεστών του kernel να είναι 1. Ο πίνακας που χρησιμοποιήθηκε είναι ο

public static final float[] SHARPEN3x3 = {

-1.f, -1.f, -1.f, // (-1)+(-1)+(-1)+(-1)+(9)+(-1)+ (-1)+(-1)+(-1) = 1

-1.f, 9.f, -1.f,

-1.f, -1.f, -1.f};

Για να ορίσουμε την λειτουργία Convolve Op γράφουμε :

BufferedImageOp cop = new ConvolveOp(new Kernel(3,3,SHARPEN3x3), ConvolveOp.EDGE_NO_OP, null)

Η Convolve Op υλοποιείται από το interface Buffered Image Op. Η πρώτη παράμετρος είναι ο kernel (ο πίνακας) που ορίστηκε παραπάνω, η δεύτερη παράμετρος αντιστοιχεί στην επιλογή να αντιγράφονται τα pixels από τις άκρες της αρχικής εικόνας στην τελική χωρίς να γίνονται αλλαγές σε αυτά. Η τρίτη παράμετρος αναφέρεται στην απόδοση της εικόνας και είναι null. Χρησιμοποιούμε την μέθοδο filter καλούμενη για την λειτουργία Convolve Op , την cop, που δημιουργήσαμε γράφοντας :

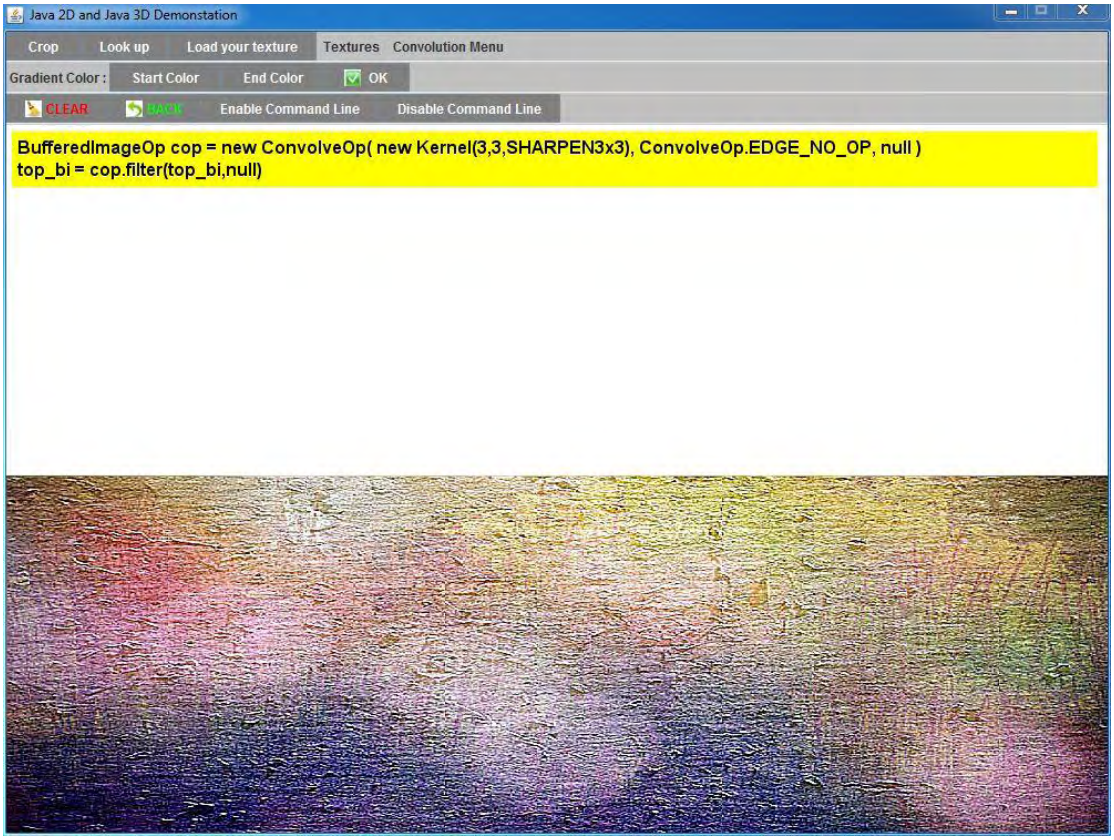
top_bi = cop.filter(top_bi,null)

Η παραπάνω εντολή εφαρμόζει το φίλτρο που δημιουργήσαμε στην top_bi εικόνα και λόγω της δεύτερης παραμέτρου null δημιουργεί μία νέα buffered image για να αποθηκευτεί το αποτέλεσμα. Η νέα εικόνα αυτή μέσω της ανάθεσης γίνεται πάλι η top_bi. Δηλαδή η τελική εικόνα μετά την εφαρμογή του φίλτρου αποθηκεύεται στην ίδια μεταβλητή τύπου buffered image την top_bi. Για να

ξέρουμε ποια buffered image πρέπει να ανανεώσουμε καλούμε τη μέθοδο get Stat() με την μεταβλητή Stat.SHARPEN όπου δείχνει ότι πρέπει να ανανεωθεί μόνο η top_bi. Έπειτα πραγματοποιείται η ανανέωση με την μέθοδο repaint().

texturesPanel.getStat(Stat.SHARPEN)

texturesPanel.repaint()



ΕΙΚΟΝΑ 108 ΟΞΥΝΣΗ ΧΡΩΜΑΤΩΝ - TEXTURE ΝΟ 4

3. **Blur – Θάμπωμα**

Ως θάμπωμα της εικόνας μπορούμε να περιγράψουμε το αποτέλεσμα της λήψης μίας φωτογραφίας όταν ο φακός δεν έχει εστιάσει σωστά. Πιο συγκεκριμένα κάθε pixel «απλώνεται» και «αναμειγνύεται» με τα γειτονικά pixel με αποτέλεσμα να εμφανίζεται η εικόνα πιο θαμπή. Με τον πίνακα που χρησιμοποιούμε το τελικό pixel είναι ο μέσος όρος του αρχικού pixel και των γειτονικών του pixel. Θέλουμε το αλγεβρικό άθροισμα των συντελεστών του kernel να είναι 1 γιατί δεν πρέπει να αλλάξει η φωτεινότητα της εικόνας.

public static final float[] BLUR3x3 = {

0.111f, 0.111f, 0.111f,

0.111f, 0.111f, 0.111f,

0.111f, 0.111f, 0.111f};

Για να ορίσουμε την λειτουργία Convolve Op γράφουμε :

BufferedImageOp op = new ConvolveOp(new Kernel(3,3,BLUR3x3), ConvolveOp.EDGE_NO_OP, null)

Η Convolve Op υλοποιείται από το interface Buffered Image Op. Η πρώτη παράμετρος είναι ο kernel (ο πίνακας) που ορίστηκε παραπάνω, η δεύτερη παράμετρος αντιστοιχεί στην επιλογή να αντιγράφονται τα pixels από τις άκρες της αρχικής εικόνας στην τελική χωρίς να γίνονται αλλαγές σε

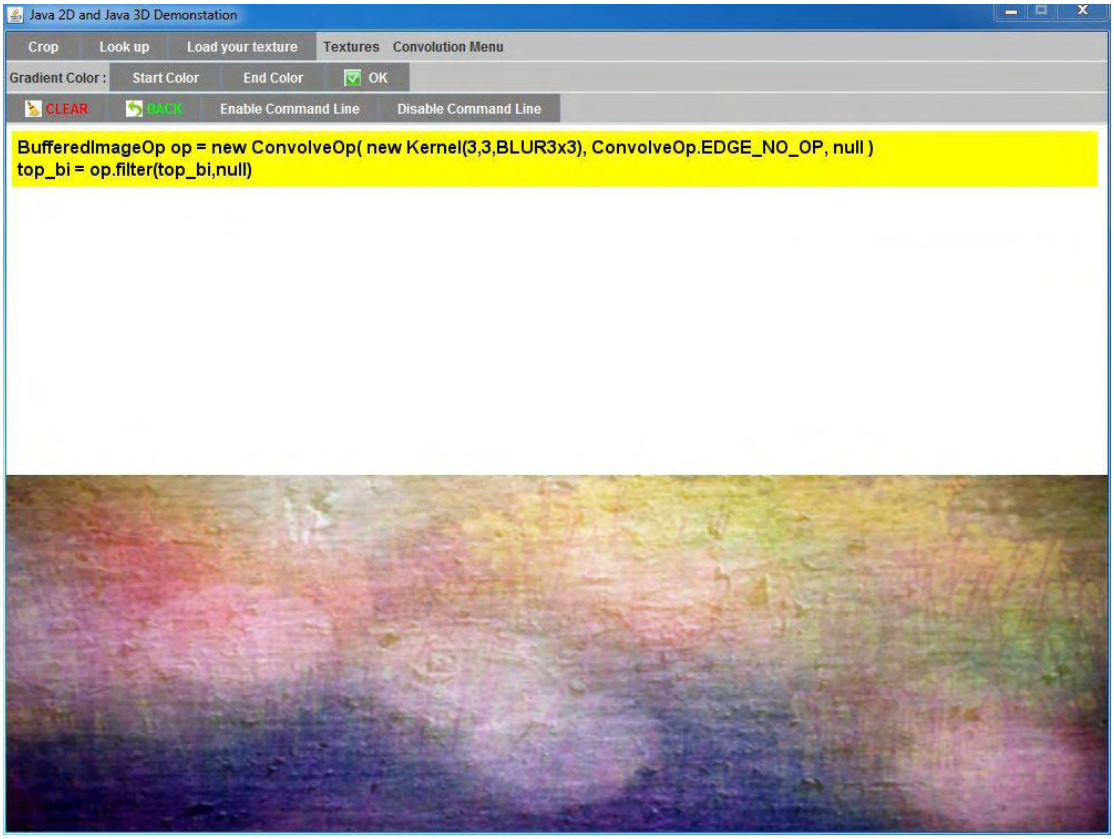
αυτά. Η τρίτη παράμετρος αναφέρεται στην απόδοση της εικόνας και είναι null. Χρησιμοποιούμε την μέθοδο filter καλούμενη για την λειτουργία Convolve Op , την op, που δημιουργήσαμε γράφοντας :

top_bi = op.filter(top_bi,null)

Η παραπάνω εντολή εφαρμόζει το φίλτρο που δημιουργήσαμε στην top_bi εικόνα και λόγω της δεύτερης παραμέτρου null δημιουργεί μία νέα buffered image για να αποθηκευτεί το αποτέλεσμα. Η νέα εικόνα αυτή μέσω της ανάθεσης γίνεται πάλι η top_bi. Δηλαδή η τελική εικόνα μετά την εφαρμογή του φίλτρου αποθηκεύεται στην ίδια μεταβλητή τύπου buffered image την top_bi. Για να ξέρουμε ποια buffered Image πρέπει να ανανεώσουμε καλούμε τη μέθοδο get Stat() με την μεταβλητή Stat.BLUR όπου δείχνει ότι πρέπει να ανανεωθεί μόνο η top_bi. Έπειτα πραγματοποιείται η ανανέωση με την μέθοδο repaint().

texturesPanel.getStat(Stat.BLUR)

texturesPanel.repaint()



ΕΙΚΟΝΑ 109 ΘΑΜΠΩΜΑ - TEXTURE ΝΟ 4

4. **Rescale – Αλλαγή Τιμής Χρωμάτων**

Ο όρος «αλλαγή τιμής χρωμάτων» περιέχει την αλλαγή στα χρώματα αλλά και στην φωτεινότητα της εικόνας. Για να ορίσουμε την λειτουργία Rescale Op γράφουμε :

BufferedImageOp rop = new RescaleOp(1.5f, 1.0f, null)

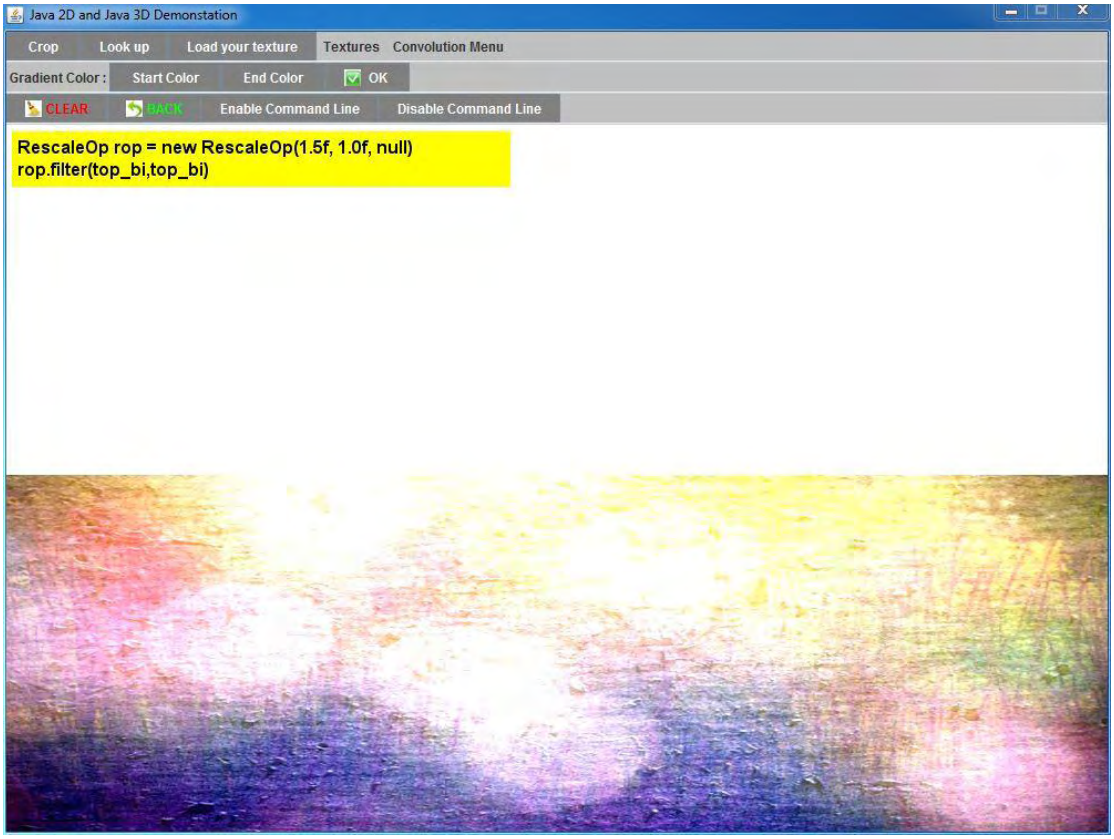
Η Rescale Op υλοποιείται από το interface Buffered Image Op. Η πρώτη παράμετρος είναι ο παράγοντας με τον οποίο πολλαπλασιάζεται η τιμή του χρώματος κάθε pixel, η δεύτερη παράμετρος αντιστοιχεί στην σταθερά που προστίθεται στο προηγούμενο γινόμενο. Η τρίτη παράμετρος αναφέρεται στην απόδοση της εικόνας και είναι null. Χρησιμοποιούμε την μέθοδο filter καλούμενη για την λειτουργία Rescale Op , την rop, που δημιουργήσαμε γράφοντας :

rop.filter(top_bi,top_bi)

Η παραπάνω εντολή εφαρμόζει το φίλτρο που δημιουργήσαμε στην top_bi εικόνα και λόγω της δεύτερης παραμέτρου null δημιουργεί μία νέα buffered image για να αποθηκευτεί το αποτέλεσμα. Η νέα εικόνα αυτή μέσω της ανάθεσης γίνεται πάλι η top_bi. Δηλαδή η τελική εικόνα μετά την εφαρμογή του φίλτρου αποθηκεύεται στην ίδια μεταβλητή τύπου buffered image την top_bi. Για να ξέρουμε ποια buffered image πρέπει να ανανεώσουμε καλούμε τη μέθοδο get Stat() με την μεταβλητή Stat.RESCALE όπου δείχνει ότι πρέπει να ανανεωθεί μόνο η top_bi. Έπειτα πραγματοποιείται η ανανέωση με την μέθοδο repaint().

texturesPanel.getStat(Stat. RESCALE)

texturesPanel.repaint()



ΕΙΚΟΝΑ 110 ΑΛΛΑΓΗ ΤΙΜΗΣ ΧΡΩΜΑΤΩΝ - TEXTURE ΝΟ 4

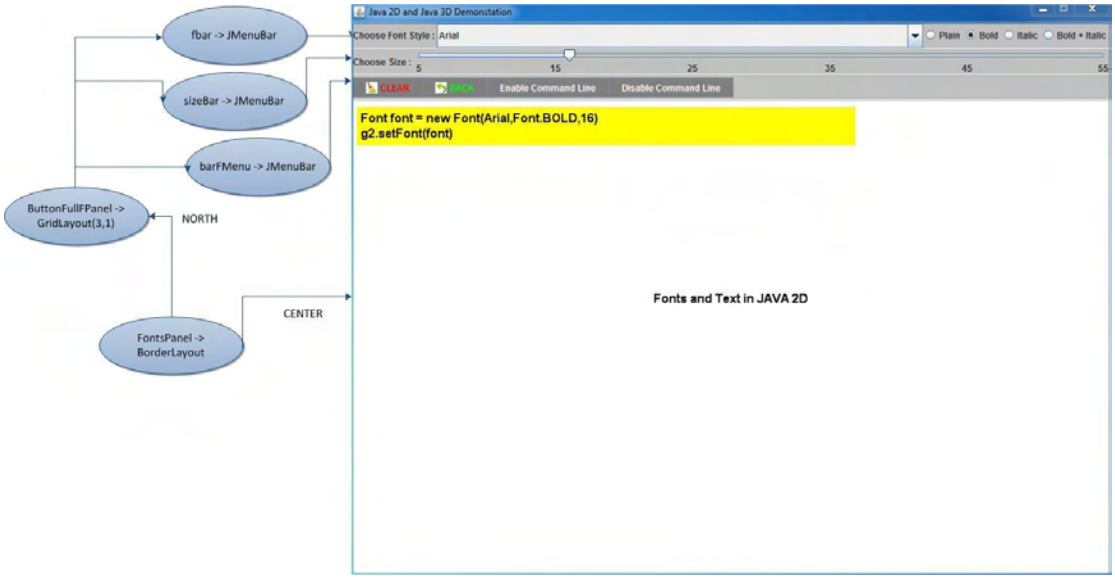
Στην εφαρμογή επιλέγουμε το φίλτρο που επιθυμούμε να εφαρμόσουμε στην εικόνα από το μενού «Convolution Menu». Οι επιλογές φέρουν τα ονόματα των αντίστοιχων φίλτρων : Sharpen, Edge Detect, Blur, Rescale.

10. Java 2D – 6η Επιλογη (Fonts ant Text)

10.1 Παρουσιαση

Η έκτη επιλογή της εφαρμογής είναι το αντικείμενο της κλάσης FontsPanel. Είναι ένα panel που περιέχει Swing components όπως κουμπιά, ετικέτες, sliders και άλλα όπως επίσης και μία περιοχή σχεδίασης. Ο χρήστης μπορεί να επιλέξει να εμφανίσει την φράση «Fonts and Text in JAVA 2D» με διαφορετικές γραμματοσειρές, με διαφορετικά στυλ όπως απλό, bold, italic, bold and italic και με διαφορετικό μέγεθος γραμμάτων. Τέλος παρέχονται οι λειτουργίες που περιγράφηκαν στο πέμπτο κεφάλαιο : καθαρισμός της περιοχής σχεδίασης και επιστροφή στα εξώφυλλο [5.4.9] και η γραμμή εντολών [5.4.11].

Το FontsPanel έχει custom δομή που δημιουργείται με border Layout, και grid Layout.

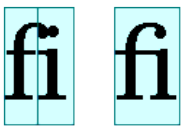


Εικονα 111 FontsPanel layout

10.2 Λειτουργικοτητα

10.2.1 Fonts and Text – Γραμματοσειρες και Κειμενο

Τα σχήματα που μία γραμματοσειρά χρησιμοποίει για να εκφράσει τους χαρακτήρες σε μία συμβολοσειρά ονομάζονται ιερογλυφικά. Ένας χαρακτήρας ή ένας συνδυασμός χαρακτήρων θα μπορούσε να αντιπροσωπευτεί από ένα ή περισσότερα ιερογλυφικά. Για παράδειγμα το «á» μπορεί να αντιπροσωπευτεί από δύο ιερογλυφικά ενώ το σύμπλεγμα «fi» (σύμπλεγμα – το τυπογραφικό στοιχείο που προέρχεται από την ένωση δύο ή περισσότερων γραμμάτων για οικονομία χώρου ή για αισθητικούς λόγους) μπορεί να αντιπροσωπευτεί από ένα ιερογλυφικό.

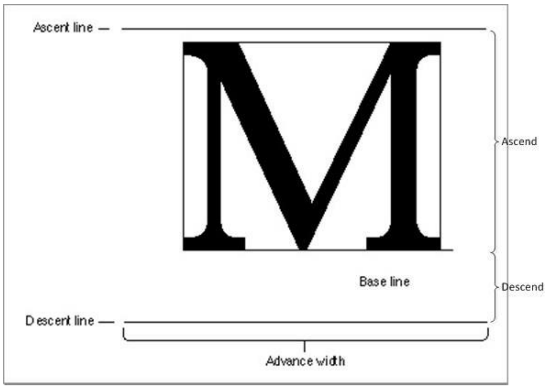


Εικονα 112 fi με δυο ιερογλυφικα και fi με ενα ιερογλυφικο

Μία γραμματοσειρά μπορεί να θεωρηθεί ως μία συλλογή από χαρακτήρες. Μία ενιαία γραμματοσειρά μπορεί να έχει πολλές πλευρές όπως πλάγια, κανονική ή έντονη γραφή. Όλες αυτές οι πλευρές σε μία γραμματοσειρά έχουν παρόμοια τυπογραφικά χαρακτηριστικά και μπορούν να αναγνωριστούν ως μέλη της ίδιας οικογένειας. Δηλαδή μία συλλογή από ιερογλυφικά με συγκεκριμένα χαρακτηριστικά σχηματίζουν μία πλευρά της γραμματοσειράς (font face). Μία συλλογή από διάφορες πλευρές της ίδιας γραμματοσειράς σχηματίζει την οικογένεια της γραμματοσειράς (font family). Η συλλογή των οικογενειών των γραμματοσειρών σχηματίζει το σύνολο των διαθέσιμων γραμματοσειρών στο σύστημα.
Υπάρχουν δύο είδη γραμματοσειρών οι φυσικές και οι λογικές. Οι φυσικές γραμματοσειρές απαρτίζουν τις πραγματικές βιβλιοθήκες γραμματοσειρών. Οι λογικές γραμματοσειρές είναι οι πέντε οικογένειες γραμματοσειρών που έχουν οριστεί από την Java και είναι οι : Serif, SansSerif, Monospaced, Dialog, DialogInput.Οι λογικές γραμματοσειρές δεν είναι πραγματικές βιβλιοθήκες γραμματοσειρών. Τα λογικά ονόματα των γραμματοσειρών αντιστοιχίζονται στις φυσικές γραμματοσειρές από το Java Runtime Environment.
Το μέγεθος της γραμματοσειράς δίνεται σε pt (points) με 1 pt ≈ 0.3515 mm. Η κάθε γραμματοσειρά δεν περιέχει μόνο την περιγραφή των ιερογλυφικών και το μέγεθος της. Κάθε φορά που τοποθετείται ένα ιερογλυφικό στο σημείο (x,y) , τοποθετείται το σημείο αναφοράς του στο σημείο (x,y). Το σημείο αναφοράς ορίζει μία οριζόντια γραμμή την γραμμή βάσης (baseline) του κάθε ιερογλυφικού. Στις συνήθεις γραφές οι baselines των ιερογλυφικών ευθυγραμμίζονται. Υπάρχουν περιπτώσεις που ιερογλυφικά βρίσκονται και κάτω από την baseline.

ΕΙΚΟΝΑ 113 ΤΟ G ΒΡΙΣΚΕΤΑΙ ΚΑΤΩ ΑΠΟ ΤΗΝ BASELINE

Κάθε ιερογλυφικό έχει advance πλάτος, ascend και descend (πλάτος προόδου, άνοδος, κάθοδος). Το ascend είναι η απόσταση πάνω από την baseline μέχρι εκεί που φτάνει το ιερογλυφικό. Το descent είναι η απόσταση κάτω από την baseline μέχρι εκεί που τελειώνει το ιερογλυφικό. Το advance πλάτος δείχνει την θέση που πρέπει να τοποθετηθεί το επόμενο ιερογλυφικό. Ένας πίνακας χαρακτήρων αλλά και μία συμβολοσειρά μπορεί να έχει ascend, descend και advance πλάτος. Το ascend θα είναι το μέγιστο ascend των χαρακτήρων του πίνακα ή της συμβολοσειράς, το descend θα είναι επίσης το μέγιστο descend των χαρακτήρων του πίνακα ή της συμβολοσειράς. Το advance πλάτος θα είναι το άθροισμα των advance πλατών κάθε χαρακτήρα του πίνακα ενώ ια τη συμβολοσειρά θα είναι η απόσταση κατά μήκος της baseline της συμβολοσειράς. Αυτή η απόσταση χρησιμοποιείται για την ευθυγράμμιση της συμβολοσειράς.

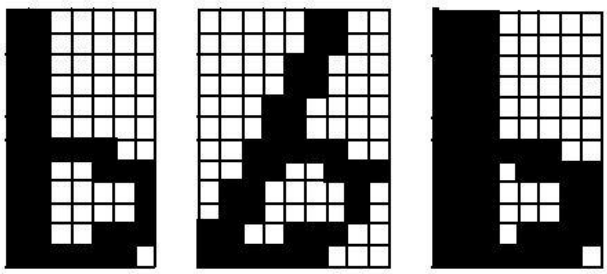


Εικονα 114 ascend, descend, advance πλατος

Επίσης τα ιερογλυφικά σε μία γραμματοσειρά μπορεί να έχουν διαφορετικά πλάτη. Τέτοιες γραμματοσειρές ονομάζονται αναλογικές (proportional). Στις αναλογικές γραμματοσειρές μπορεί ακόμη να ποικίλει η απόσταση μεταξύ των γραμμάτων ανάλογα με τον συνδυασμό τους.

Εικονα 115 αναλογικη γραμματοσειρα και γραμματοσειρα με ιδιο πλατος σε ολα τα γραμματα

Οι γραμματοσειρές μπορούν να αποδοθούν είτε με διανυσματικά γραφικά είτε με γραφικά ψηφίδων. Τα γραφικά ψηφίδων έχουν το πλεονέκτημα ότι δεν χρειάζεται επιπλέον επεξεργασία για την απόδοσή τους όταν σχεδιάζονται τα ιερογλυφικά. Κατά την απόδοσή τους σχεδιάζονται συγκεκριμένα pixel σύμφωνα με την προσχεδιασμένη εικόνα από pixel κάθε ιερογλυφικού. Το μειονέκτημα είναι ότι χρειάζονται ξεχωριστές εικόνες από pixel για κάθε ιερογλυφικό της ίδια γραμματοσειράς για διαφορετικό μέγεθος και διαφορετικό είδος γραφής (πλάγια, κανονική, έντονη). Βέβαια μπορούν να δημιουργούνται διαφορετικά μεγέθη ιερογλυφικών με την λειτουργία του scaling, η διαδικασία αυτή όμως δεν προτείνεται.
Για να αλλάξει το είδος της γραφής από κανονικό σε πλάγιο πρέπει να μετατοπιστούν τα ιερογλυφικά προς τα δεξιά. Για την έντονη γραφή ολόκληρο το ιερογλυφικό αντιγράφεται και μετατοπίζεται προς τα δεξιά.



Εικονα 116 κανονικη γραφη, πλαγια γραφη, εντονη γραφη

Οι τεχνικές για την πλάγια, την έντονη γραφή αλλά και το scaling των γραμματοσειρών για αναπαράσταση διαφορετικών μεγεθών οδηγούν σε μη αποδεκτά αποτελέσματα για αυτό και οι γραμματοσειρές συνήθως αποθηκεύονται ως διανυσματικά γραφικά.

Για να εμφανιστεί μία συμβολοσειρά στην Java 2D χρησιμοποιείται η μέθοδος :

drawstring(“text”, posx, posy)

καλούμενο για ένα αντικείμενο Graphics2D. Η μέθοδος αυτή εμφανίζει την συμβολοσειρά που βρίσκεται μέσα σε αυτάκια στην πρώτη παράμετρο της συνάρτησης , την text, στην θέση (xpos,ypos) για την προεπιλεγμένη γραμματοσειρά (τύπος : “Dialog”, είδος γραφής : Font.PLAIN - κανονικό, μέγεθος : 12).
Για να αλλάξουμε γραμματοσειρά χρησιμοποιούμε ένα νέο αντικείμενο της κλάσης Font γράφοντας :

Font f = new Font(“type”, Font.STYLE, size)

Όπου δημιουργείται μία νέα γραμματοσειρά με το όνομα f τύπου “type”, με είδος γραφής Font.STYLE και μεγέθους γραμμάτων size. Μπορεί να εμφανίστεί μία λίστα με όλες τους διαθέσιμους τύπους γραμματοσειρών σε έναν η/υ γράφοντας :

```
Font[] fl = GraphicsEnvironment.getLocalGraphicsEnvironment().getAllFonts();

for (int i=0; i<fl.length; i++)

{

System.out.println(fl[i].getName());

}
```

Όλα τα ονόματα που εμφανίζονται σε αυτή τη λίστα μπορούν να χρησιμοποιηθούν στην πρώτη παράμετρο της κλάσης Font για παράδειγμα : Arial, Calibri, Times New Roman, Tahoma κτλ. Η δεύτερη παράμετρος μπορεί να πάρει τις τιμές Font.BOLD για έντονη γραφή, Font.ITALIC για πλάγια γραφή, Font.PLAIN για κανονική γραφή και Font.BOLD + Font.ITALIC για έντονη και πλάγια γραφή.

Για να αλλάξουμε κάποια χαρακτηριστικά σχετικά με το μέγεθος της γραμματοσειράς χρησιμοποιούμε ένα στιγμιότυπο της Font Metrics καλούμενο για ένα αντικείμενο Graphics2D με παράμετρο το όνομα της γραμματοσειράς που έχουμε δημιουργήσει:

FontMetrics fm = g2.getFontMetrics(f)

Η μεταβλητή fm περιέχει τις μετρήσεις των γραφικών που έχουμε δημιουργήσει με την γραμματοσειρά f. Γράφοντας :

Rectangle2D rect = fm.getStringBounds(text, g2)

Δημιουργούμε ένα ορθογώνιο αντικείμενο της κλάσης Rectangle2D με διαστάσεις όσο το πλάτος και το ύψος της συμβολοσειράς text που έχει δημιουργηθεί μέσω του αντικειμένου Graphics2D g2. Το ορθογώνιο αυτό είναι το κουτί που περικλείει την συμβολοσειρά text. Στην εφαρμογή χρειαζόμαστε την θέση του ορθογωνίου αυτού γιατί αλλάζοντας το μέγεθος των γραμμάτων θέλουμε η συμβολοσειρά που εμφανίζεται να είναι στο κέντρο της περιοχής σχεδίασης. Για να βρούμε το πλάτος και το ύψος του ορθογωνίου καλούμε τις μεθόδους getWidth() και getHeight() :

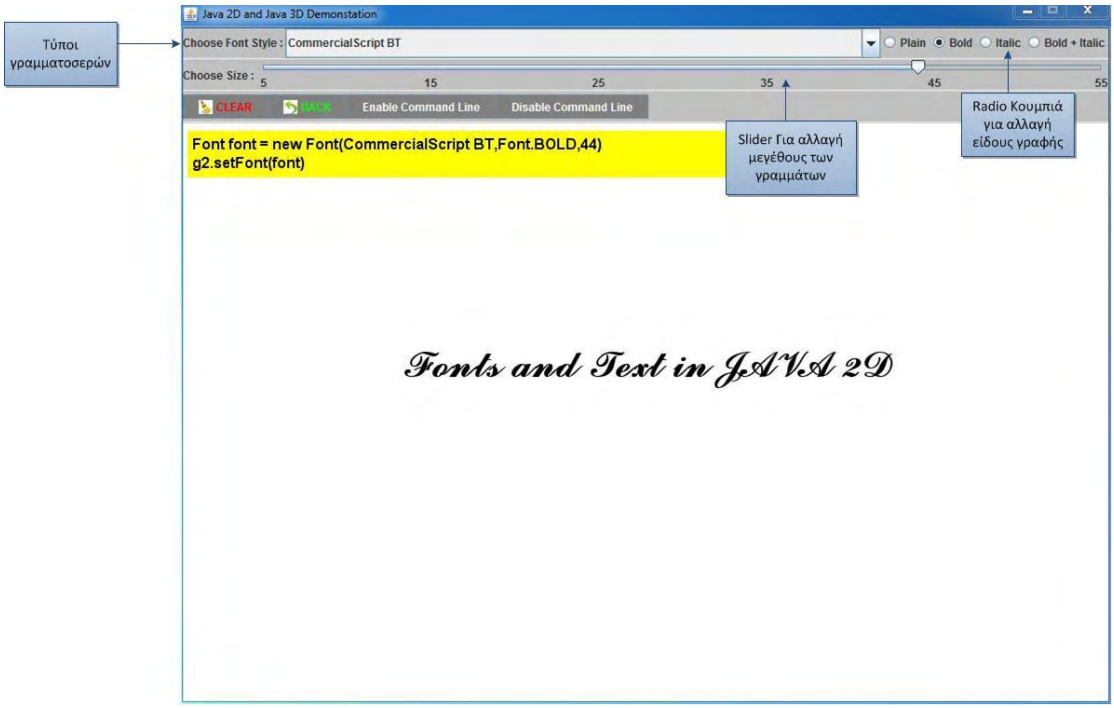
```
int textWidth   = (int)(rect.getWidth())

int textHeight  = (int)(rect.getHeight())
```

Κάθε φορά που αλλάζει μέγεθος η συμβολοσειρά οι μεταβλητές textWidth, textHeight παίρνουν νέες τιμές. Για να ξέρουμε σε ποια θέση πρέπει να εμφανιστεί η συμβολοσειρά υπολογίζουμε κάθε φορά το (x,y) σημείο αναφοράς σύμφωνα με τις τρέχουσες τιμές των textWidth, textHeight. Βρίσκουμε το x αφαιρώντας από το πλάτος του Panel που είναι σταθερό την τιμή textWidth και διαιρώντας δια 2. Βρίσκουμε το y αφαιρώντας από το ύψος του Panel που είναι σταθερό την τιμή textHeight, διαιρούμε δια 2 και προσθέτουμε την ascend απόσταση της συμβολοσειράς.

int x = (this.getWidth() - textWidth) / 2

int y = (this.getHeight() - textHeight) / 2 + fm.getAscent()

Τέλος ζωγραφίζουμε την συμβολοσειρά με την μέθοδο draw String() στην θέση x,y.
Η εφαρμογή εμφανίζει την φράση «Fonts and Text in JAVA 2D» με όλους τους διαθέσιμους τύπους γραμματοσειράς που διαθέτει ο η/υ που τρέχει την εφαρμογή. Ο χρήστης μπορεί επίσης να αλλάξει το μέγεθος των γραμμάτων μέσω ενός slider και να επιλέξει διαφορετικές γραφές (κανονική, έντονη, πλάγια, έντονη και πλάγια) από τα radio κουμπιά που βρίσκονται πάνω δεξιά. Η φράση εμφανίζεται πάντα στο κέντρο της περιοχής σχεδίασης .



ΕΙΚΟΝΑ 117 ΛΕΙΤΟΥΡΓΙΕΣ ΤΗΣ ΕΠΙΛΟΓΗΣ "FONTS AND TEXT"

11. JAVA 3D – 7Η ΕΠΙΛΟΓΗ (GEOMETRIC TRANSFORMATIONS)

11.1 JAVA 3D

Η διεπαφή προγραμματισμού Java 3D έκδοση 1.5.2 περιέχει τα jar αρχεία j3dcore (ο πυρήνας της διεπαφής Java 3D), j3dutils (οι λειτουργίες του πυρήνα) και vecmath (τα διανυσματικά τρισδιάστατα γραφικά). Μπορούμε να κατεβάζουμε τη Java 3D δωρεάν από το διαδίκτυο μέσω του συνδέσμου που δίνεται στο Παράρτημα Β [Παράρτημα Β]. Για να μπορέσουμε να δημιουργήσουμε προγράμματα με την Java 3D πρέπει εγκαταστήσουμε την Java 3D στο μηχάνημά μας και να εισάγουμε τα jar αρχεία στο πρόγραμμά μας μέσω του IDE που χρησιμοποιούμε σύμφωνα με τις οδηγίες που δίνονται στο Παράρτημα Β [Παράρτημα Β].

11.2 ΠΕΡΙΓΡΑΦΗ ΤΡΙΣΔΙΑΣΤΑΤΟΥ ΚΟΣΜΟΥ

Πριν ζωγραφίσουμε οτιδήποτε στην οθόνη του η/υ ένας εικονικός τρισδιάστατος κόσμος αντικειμένων πρέπει να οριστεί και να αποθηκευτεί στον η/υ. Ο τρισδιάστατος κόσμος μπορεί να περιέχει πολύ απλές σκηνές π.χ. ένα δέντρο ή και πιο πολύπλοκες όπως ένα δάσος.

Το πρώτο βήμα για την κατασκευή του εικονικού κόσμου είναι ο ορισμός της γεωμετρία των αντικείμενων. Τα αντικείμενα μπορεί να είναι φανταστικά ή και αντικείμενα που ήδη είναι κατασκευασμένα. Και στις δύο περιπτώσεις τα αντικείμενα πρέπει να μοντελοποιηθούν και να κατασκευαστούν. Μπορεί η κατασκευή και η μοντελοποίηση των αντικειμένων να είναι απλή λαμβάνοντας κάποιες μετρήσεις όπως πλάτος και ύψος αλλά πολλές φορές δεν είναι αρκετή για την κατασκευή αληθοφανών εικονικών κόσμων. Για την δημιουργία αληθοφανών αναπαραστάσεων αντικειμένων χρειάζονται πολύ περισσότερα στοιχεία όπως λεπτομέρειες για την επιφάνειά τους και την δομή τους. Ένας άλλος τρόπος για την μοντελοποίηση εικονικών αντικειμένων είναι οι 3D σαρωτές που παρέχουν πληροφορίες για την γεωμετρία των επιφανειών τους. Οι 3D σαρωτές μπορούν επίσης να παρέχουν στοιχεία και για την εσωτερική δομή αντικειμένων π.χ. γέφυρες, κτήρια. Άλλος ένα τρόπος μοντελοποίησης είναι οι τεχνικές x-ray, ultrasonic και τομογραφίας που μπορούν να παρέχουν πληροφορίες για την σκελετική δομή και τους ιστούς του ανθρωπίνου σώματος με σκοπό να δημιουργηθούν μοντέλα των οστών και των οργάνων.

Το δεύτερο βήμα για την αναπαράσταση του εικονικού κόσμου είναι ο ορισμός της θέσης του παρατηρητή και της κατεύθυνσης του για να προσδιοριστεί το συγκεκριμένο κομμάτι του εικονικού κόσμου που είναι ορατό στον παρατηρητή. Επίσης πρέπει να οριστεί το πεδίο όρασης του παρατηρητή, η γωνία θέασης και η μέγιστη απόσταση που μπορεί να δει ο παρατηρητής. Με αυτό τον τρόπο δημιουργείται μία τρισδιάστατη περιοχή η οποία είναι η μόνη ορατή περιοχή για τον παρατηρητή και η μόνη που πρέπει να αποδοθεί στην οθόνη.

Το τρίτο βήμα για την δημιουργία του εικονικού κόσμου είναι η πρόσθεση φωτισμού. Ο εικονικός κόσμος είναι μία μαύρη εικόνα αν δεν προσθέσουμε φως. Για να προσθέσουμε φως πρέπει να ορίσουμε τις πηγές φωτός, την θέση τους όπως επίσης και τα χαρακτηριστικά τους π.χ. το χρώμα, την φωτεινότητα κτλ.. Προσθέτοντας φως μπορούμε να δημιουργήσουμε και σκιάσεις.

Το τελευταίο βήμα είναι η προσθήκη κάποιον ειδικών εφέ όπως η ομίχλη και άλλα.

11.3 Java 3D Concept

11.3.1 Scene Graph – Γραφημα Σκηνης

Η διεπαφή προγραμματισμού Java 3D χρησιμοποιεί έναν απλό και ευέλικτο μηχανισμό για να αναπαραστήσει και να αποδώσει σκηνές, το scene graph. Το scene graph έχει την μορφή δέντρου και περιέχει δεδομένα διατεταγμένα με ιεραρχικό τρόπο. Το scene graph αποτελείται από πατρικούς κόμβους, κόμβους παιδιά και αντικείμενα δεδομένων. Οι πατρικοί κόμβοι καλούνται Group nodes και σε μερικές περιπτώσεις ελέγχουν πως η Java 3D ερμηνεύει τους απογόνους τους. Οι group nodes είναι η κόλλα που κρατάει ενωμένο το scene graph. Οι κόμβοι παιδιά μπορούν να είναι είτε Group nodes είτε leaf nodes. Οι leaf nodes δεν έχουν παιδιά. Οι κόμβοι παιδιά αποτελούν τα βασικά στοιχεία του scene graph για παράδειγμα την γεωμετρία, τον φωτισμό, τον ήχο, τις συμπεριφορές. Οι leaf nodes αναφέρονται σε αντικείμενα δεδομένων που ονομάζονται node components. Τα node components αντικείμενα δεδομένων δεν είναι κόμβοι του scene graph αλλά περιέχουν δεδομένα που οι leaf nodes χρειάζονται. Η δομή του scene graph καθορίζει τις σχέσεις μεταξύ των αντικειμένων του γραφήματος και ποια αντικείμενα μπορεί να αντιμετωπίσει ο προγραμματιστής σαν μία ενότητα.

Έστω ότι θέλουμε να ζωγραφίσουμε δύο σχήματα. Τα σχήματα αυτά ορίζονται από την γεωμετρία τους αλλά και την εμφάνιση τους. Σε αναπαράσταση scene graph τα δύο σχήματα είναι οι leaf nodes, η γεωμετρία και η εμφάνιση του κάθε σχήματος είναι τα node components κάθε leaf node και οι δύο leaf nodes είναι τα παιδιά του group node πατρικού κόμβου.

Shape3D myShape1 = new Shape3D(myGeometry1, myAppearance1)

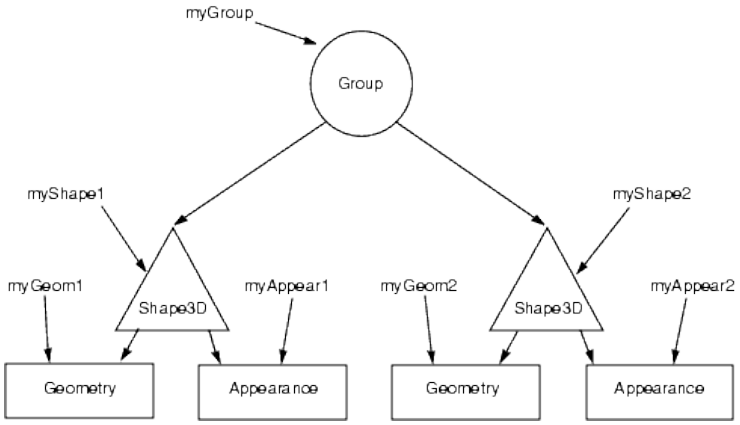
Shape3D myShape2 = new Shape3D(myGeometry2)

myShape2.setAppearance(myAppearance2)

Group myGroup = new Group()

myGroup.addChild(myShape1)

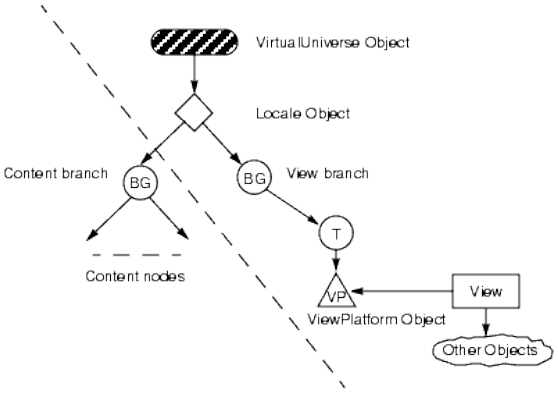
myGroup.addChild(myShape2)



Εικονα 118 παραδειγμα scene graph

Αφού έχει δημιουργηθεί το scene graph για να αποδοθεί στην οθόνη από την Java 3D πρέπει πρώτα να εισαχθεί σε ένα εικονικό κόσμο. Η Java 3D ακολουθεί ένα συγκεκριμένο πρότυπο για το πώς

μπορεί να εισαχθεί ένα scene graph στον εικονικό κόσμο. Ένα Java 3D περιβάλλον αποτελείται από δύο δομικά αντικείμενα το Virtual Universe (εικονικός κόσμος) και το Locale (τοποθεσία συμβάντος) και από ένα ή περισσότερα sub graphs τα οποία έχουν ως πατέρα έναν ειδικό κόμβο που ονομάζεται branch group (ομάδα διακλάδωσης).



ΕΙΚΟΝΑ 119 VIRTUAL UNIVERSE, LOCALE, BRANCH GROUPS

Το Virtual Universe αντικείμενο ορίζει τον εικονικό κόσμο. Το αντικείμενο αυτό επιτρέπει στην Java 3D να δημιουργεί έναν ξεχωριστό πεδίο για να ορίζει αντικείμενα και τις σχέσεις τους. Τυπικά ένα Java 3D πρόγραμμα έχει μόνο ένα Virtual Universe αντικείμενο. Προγράμματα με περισσότερα από ένα Virtual Universe αντικείμενα μπορούν να μοιράζονται node components αντικείμενα αλλά όχι scene graphs.

Το Locale αντικείμενο καθορίζει μία σταθερή θέση μέσα στον εικονικό κόσμο. Αυτή η θέση καθορίζει το κέντρο για όλα τα scene graphs που υπάρχουν στον συγκεκριμένο εικονικό κόσμο. Το αντικείμενο Locale επιτρέπει στον προγραμματιστή να καθορίζει το κέντρο με μεγάλη ακρίβεια οπουδήποτε μέσα στον γνωστό φυσικό κόσμο. Τυπικά τα Java 3D προγράμματα έχουν μόνο ένα αντικείμενο Locale με ένα προκαθορισμένη κέντρο το (0, 0, 0). Τα προγράμματα με περισσότερα από ένα Locale αντικείμενα θέτουν το κέντρο κάθε αντικειμένου Locale και των αντικειμένων που ανήκουν στο συγκεκριμένο αντικείμενο Locale σε διαφορετικές θέσεις μέσα στον ίδιο εικονικό κόσμο.

Το scene graph ξεκινά με έναν κόμβο branch group. Οι branch group κόμβοι χρησιμεύουν ως ρίζα ενός επιμέρους γραφήματος του branch graph (γράφημα διακλάδωσης). Υπάρχουν δύο είδη branch graphs τα content branches (διακλαδώσεις περιεχομένου) και τα view branches (διακλαδώσεις προβολής). Ένα content branch περιέχει μόνο leaf nodes σχετικούς με το περιεχόμενο ενώ ένα view branch περιέχει έναν κόμβο View Platform και μπορεί να περιέχει και άλλους leaf nodes σχετικούς με το περιεχόμενο. Τυπικά ένας εικονικός κόσμος περιέχει περισσότερα από ένα branch graphs (ένα view branch και οποιοδήποτε αριθμό από content branch). Ο branch group κόμβος έχει δύο ιδιότητες : μπορεί από μόνος του να εισαχθεί σε έναν Locale αντικείμενο και μπορεί να γίνει compiled. Οι branch group κόμβοι που δεν έχουν κάνει compile αντιμετωπίζονται από το σύστημα όπως και οι branch group κόμβοι που έχουν κάνει compile.

Για να εισάγουμε το scene graph που φαίνεται στην εικόνα 118 στο Locale αντικείμενο πρέπει πρώτα να δημιουργήσουμε τη ρίζα του δηλαδή έναν branch group κόμβο. Έτσι με τα ίδια δομικά στοιχεία δημιουργείται έναν content branch. Πρέπει να σημειώσουμε ότι τα αντικείμενα Locale δεν είναι μέρος του scene graph και δεν έχουν παιδιά.

Shape3D myShape1 = new Shape3D(myGeometry1, myAppearance1)

Shape3D myShape2 = new Shape3D(myGeometry2, myAppearance2)

BranchGroup myBranch = new BranchGroup()

myBranch.addChild(myShape1)

myBranch.addChild(myShape2)

myBranch.compile()

VirtualUniverse myUniverse = new VirtualUniverse()

Locale myLocale = new Locale(myUniverse)

myLocale.addBranchGraph(myBranch)

Όταν η Java3D επεξεργάζεται ένα scene graph δεν ακολουθεί κάποια σειρά για την απόδοση των αντικειμένων που ορίζονται στο scene graph. Μία υλοποίηση μπορεί να αποδώσει πρώτα το πρώτο σχήμα μετά το δεύτερο μία άλλη υλοποίηση πρώτα το δεύτερο και μετά το πρώτο και μία άλλη υλοποίηση και τα δύο σχήματα μαζί.

Όλα τα scene graph, όπως επίσης οι κόμβοι αλλά και τα node components που τα αποτελούν, είναι μέρη ενός ενεργού εικονικού κόσμου ή ενός μη ενεργού. Ένα αντικείμενο λέγεται live (ενεργό) αν είναι μέρος ενός ενεργού εικονικού κόσμου. Επιπλέον όπως ήδη αναφέρθηκε τα branch groups και κατ'επέκταση τα branch graphs έχουν κάνει compile ή όχι. Γενικά ένας κόμβος αν είναι live ή έχει κάνει compile η Java 3D επιβάλει περιορισμούς πρόσβασης για τους κόμβους αλλά και για τα node components αντικείμενα. Η Java 3D επιτρέπει εκείνες τις ενέργειες που έχουν οριστεί για το συγκεκριμένο κόμβο πριν ο κόμβος αυτός γίνει live ή κάνει compile.

11.3.2 Bounds – Όρια

Τα αντικείμενα Bound επιτρέπουν στον προγραμματιστή να ορίζει έναν όγκο μέσα στον εικονικό κόσμο. Υπάρχουν τρεις τρόποι να ορίσεις έναν όγκο : ως ένα κουτί, ως μία σφαίρα ή σαν ένα σύνολο από επίπεδα που περιέχουν κάποιο χώρο.

Τα αντικείμενα Bound ορίζουν έναν όγκο μέσα στον οποίο γίνονται κάποιες λειτουργίες. Φαινόμενα όπως ο φωτισμός, η ομίχλη, διαφορετικές εμφανίσεις και η μοντελοποίηση αντικειμένων χρησιμοποιούν αντικείμενα Bound για να καθορίσουν την περιοχή επιρροής τους. Οποιοδήποτε αντικείμενο υπάρχει μέσα στον όγκο που ορίζει το ορισμένο αντικείμενο Bound μπορούν να του εφαρμοστούν τα αντίστοιχα φαινόμενα. Η σωστή χρήση των αντικειμένων Bound εξασφαλίζει ότι αυτά τα φαινόμενα θα εφαρμοστούν σε έναν συγκεκριμένο όγκο για παράδειγμα ένα φωτιστικό να φωτίζει το δωμάτιο στο οποίο βρίσκεται.

Επίσης τα αντικείμενα Bound χρησιμοποιούνται για να ορίσουν μία περιοχή δράσης για τον ήχο ή για μία συμπεριφορά. Η Java 3D μπορεί να «ξεθωριάσει» κάποιον ήχο ή κάποια συμπεριφορά που πραγματοποιείται μακριά από τον χρήστη και δεν μπορεί να τον επηρεάσει (π.χ. να ακούσει τον ήχο).

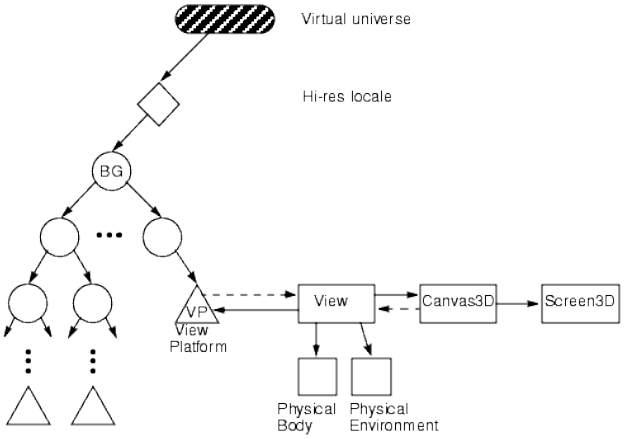
Τα αντικείμενα αυτά χρησιμοποιούνται για να ορίσουν περιοχές της εφαρμογής για λειτουργίες που γίνονται σε συγκεκριμένες προβολές (per-view operations) όπως η επιλογή του φόντου ή του ήχου του τοπίου.

11.3.3 Nodes - Κομβοι

Είναι σημαντικό να επισημάνουμε ότι όλοι οι κόμβοι του scene graph που δημιουργούνται τοποθετούνται σε μία ορισμένη θέση μέσο στον εικονικό κόσμο το κέντρο (0, 0, 0). Το κέντρο αυτό παρέχει έναν τοπικό σύστημα συντεταγμένων για κάθε αντικείμενο δηλαδή ένα σημείο αναφοράς. Ακόμα και αντικείμενα με αφηρημένες έννοιες όπως κάποια συμπεριφορά ή τα φώτα χαρακτηρίζονται από αυτό το σημείο αναφοράς. Η θέση ενός αντικειμένου παρέχει το κέντρο του τοπικού συστήματος συντεταγμένων και είναι το κέντρο του όγκου που οριοθετεί το συγκεκριμένο αντικείμενο. Τα αντικείμενα μπορούν να μετακινηθούν μέσα στον εικονικό κόσμο εφαρμόζοντας σε αυτά γεωμετρικούς μετασχηματισμούς. Οι γεωμετρικοί μετασχηματισμοί παίζουν πρωτεύον ρόλο στην δημιουργία τρισδιάστατων γραφικών και θα είναι το αντικείμενο αυτού του κεφαλαίου και της 7ης επιλογής της εφαρμογής.

11.3.4 View Model – Μοντελο Προβολης

Όπως περιγράφηκε στην προηγούμενη παράγραφο το branch graph χωρίζονται σε content branch και view branch. Τα view branch γραφήματα χρειάζονται για να ορίσουν τις λεπτομέρειες προβολής. Η Java 3D δίνει την δυνατότητα στον προγραμματιστή να μπορεί να αλλάξει τις παραμέτρους τις προβολής χωρίς να αλλάξει το scene graph. Αυτό γίνεται γιατί ένα πρόγραμμα σε Java 3D μπορεί να αποδοθεί σε διάφορα μέσα όπως η οθόνη ενός η/υ, σε αίθουσες προβολής και άλλα που έχουν διαφορετικές απαιτήσεις προβολής. Το μοντέλο προβολής πετυχαίνει αυτή την ευκολία στην χρήση διαχωρίζοντας τον εικονικό με τον φυσικό κόσμο. Το μοντέλο αυτό διαχωρίζει το πώς μία εφαρμογή τοποθετεί, προσανατολίζει και αλλάζει το μέγεθος ενός αντικειμένου View Platform σε έναν εικονικό κόσμο και πως η Java 3D κατασκευάζει την τελική προβολή (view) από την θέση παρατηρητή και τον αντίστοιχο προσανατολισμό. Η εφαρμογή ελέγχει την θέση και τον προσανατολισμό του View Platform αντικειμένου και η Java 3D υπολογίζει ποια προβολή πρέπει να αποδώσει σύμφωνα με την θέση και τον προσανατολισμό του φυσικού περιβάλλοντος του χρήστη αλλά και της θέσης και του προσανατολισμού του χρήστη μέσα στον φυσικό περιβάλλον.



Εικονα 120 View Model

Το View Model διανέμει τις παραμέτρους όρασης ανάμεσα σε διάφορα αντικείμενα: το αντικείμενο View και στα component αντικείμενα του Physical Body και Environment, στο Canvas3D αντικείμενο και στο Screen3D αντικείμενο.

Ο κόμβος View Platform καθορίζει ένα σύστημα συντεταγμένων άρα και ένα πλαίσιο αναφοράς μαζί με το κέντρο και το σημείο αναφοράς μέσα στον εικονικό κόσμο. Ο View Platform κόμβος χρησιμεύει ως σημείο σύνδεσης για το αντικείμενο View και ως βάση για τον καθορισμό της προβολής για την διαδικασία της απόδοσης από την Java 3D.

Το View αντικείμενο είναι το κύριο αντικείμενο του μοντέλου προβολής και περιέχει πολλά κομμάτια που συναρμολογούν την ολοκληρωμένη προβολή όπως τον τρόπο που αποδίδεται η εικόνα στα παράθυρα που αντιπροσωπεύονται από τα Canvas3D αντικείμενα. Επίσης περιέχει ένα σύνολο από καμβάδες που αντιπροσωπεύουν διάφορα παράθυρα σε μία προβολή.

Το Canvas3D αντικείμενο αντιπροσωπεύει ένα παράθυρο μέσα στο οποίο η Java 3D ζωγραφίζει εικόνες. Περιέχει μία αναφορά σε ένα Screen3D αντικείμενο και πληροφορίες που περιγράφουν το μέγεθος, το σχήμα και την θέση του Canvas3D αντικειμένου μέσα στο Screen3D αντικείμενο.

Το Screen3D αντικείμενο περιέχει πληροφορίες που αφορούν τις φυσικές ιδιότητες της οθόνης προβολής. Η Java 3D τοποθετεί τις πληροφορίες για την οθόνη προβολής σε ένα ξεχωριστό αντικείμενο για να αποφύγει την επικάλυψη των πληροφοριών της οθόνης για κάθε Canvas3D αντικείμενο που μοιράζεται μία κοινή οθόνη.

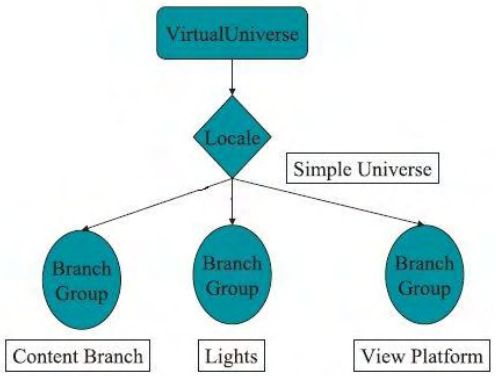
Το Physical Body αντικείμενο περιέχει μετρήσεις που περιγράφουν τα φυσικά χαρακτηριστικά του χρήστη όπως τη θέση των ματιών, την απόσταση μεταξύ των κορών των ματιών.

Το Physical Environment αντικείμενο περιέχει πληροφορίες για το φυσικό περιβάλλον του χρήστη όπως οι συσκευές εξόδου του ήχου και αν υπάρχουν αισθητήρες ανίχνευσης κίνησης και άλλα.

Τα αντικείμενα Canvas3D, Physical Environment, Physical Body μαζί με άλλα αντικείμενα αποτελούν περιεχόμενο του αντικείμενου Viewer που περιέχει τις πληροφορίες που περιγράφουν την φυσική αλλά και την εικονική παρουσία του 3D εικονικού κόσμου.

11.4 Γενικη Δομη Επιλογων Java 3D

Κάθε επιλογή Java 3D της εφαρμογής είναι ένα JPanel που ακολουθεί το concept που παρουσιάστηκε στην προηγούμενη παράγραφο [11.3]. Κάθε πρόγραμμα χωρίζεται όπως αναφέρθηκε στην 11.2 παράγραφο [12.2] σε 3 διακλαδώσεις (όχι 4) την διακλάδωση του περιεχομένου σε συνδυασμό με την διακλάδωση των ειδικών εφέ (content branch), την διακλάδωση της προβολής (View Platform) και την διακλάδωση του φωτισμού (Lights).



Εικονα 121 προγραμμα σε Java 3D χωρισμενο σε 3 διακλαδωσεις

Κάθε JPanel περιέχει ένα αντικείμενο της κλάσης Canvas3D το οποίο περνάει ως παράμετρος στο αντικείμενο Simple Universe κατά την δημιουργία του αντικειμένου Simple Universe. Έτσι το αντικείμενο Canvas3D περιέχει τον εικονικό κόσμο που θα δημιουργήσουμε. Χρησιμοποιούνται οι προεπιλεγμένες τιμές για τις παραμέτρους του Simple Universe με την βοήθεια της μεθόδου get Preferred Configuration(). Θα υπάρξουν αλλαγές που θα συζητηθούν σε επόμενο κεφάλαιο [12].

myCanvas3D = new Canvas3D(SimpleUniverse.getPreferredConfiguration())

SimpleUniverse simpUniv = new SimpleUniverse(myCanvas3D)

Με την δημιουργία του αντικειμένου Simple Universe δημιουργούνται και τα απαραίτητα αντικείμενα για την διακλάδωση της προβολής. Δηλαδή ένα αντικείμενα Viewing Platform και ένα αντικείμενο Viewer. Για να προσδιορίσουμε τις παραμέτρους της προβολής γράφουμε την παρακάτω πρόταση. Σε επόμενο κεφάλαιο θα μελετηθούν οι λεπτομέρειες για την προβολή [12].

simpUniv.getViewingPlatform().setNominalViewingTransform()

Για να δημιουργήσουμε την διακλάδωση του περιεχομένου (Content Branch) δημιουργούμε ένα branch group για το οποίο καλούμε τη μέθοδο create Scene Graph(). Η μέθοδος αυτή υλοποιείται ξεχωριστά για κάθε επιλογή της εφαρμογής και περιέχει όλες τις πληροφορίες για τα αντικείμενα του scene graph όπως την εμφάνισή τους, την γεωμετρία τους, την θέση τους και άλλα. Για να προσθέσουμε στο Simple Universe την διακλάδωση περιεχομένου χρησιμοποιούμε την μέθοδο add Branch graph() καλούμενη για το αντικείμενο Simple Universe.

BranchGroup theScene = new BranchGroup()

theScene = createSceneGraph()

simpUniv.addBranchGraph(theScene)

Για να δημιουργήσουμε την διακλάδωση του φωτισμού χρησιμοποιούμε την μέθοδο add Light() η οποία ορίζει τον φωτισμό της σκηνής και μπορεί να υλοποιηθεί ξεχωριστά για κάθε επιλογή της εφαρμογής. Ο φωτισμός εξαρτάται από το που βρίσκεται η σκηνή δηλαδή αν βρίσκεται εξωτερικά και χρειάζεται το φως του ήλιου ή αν βρίσκεται σε δωμάτιο και χρειάζεται τεχνητό φως. Οι λεπτομέρειες αυτής της μεθόδου μελετώνται σε ξεχωριστό κεφάλαιο της εργασίας αυτής [13] και παρουσιάζονται σε ξεχωριστή επιλογή της εφαρμογής.

addLight(simpUniv)

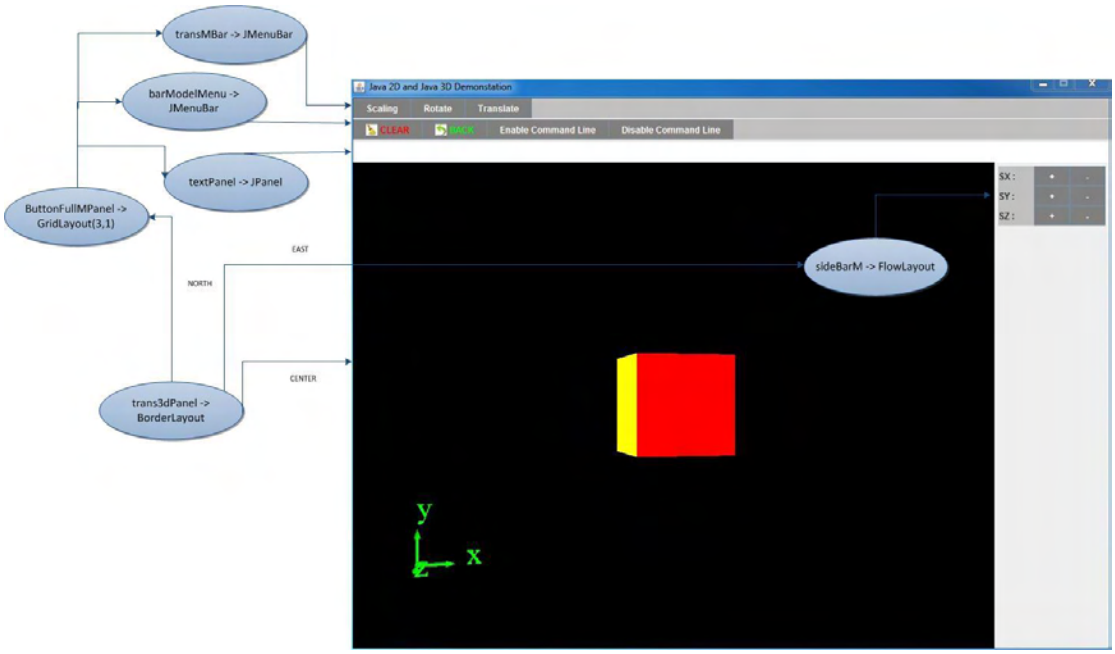
Στις 4 τελευταίες επιλογές της εφαρμογής που σχετίζονται με την Java 3D χρησιμοποιείται μία μίξη Swing και AWT components μαζί με το Simple Universe αντικείμενο που περιέχει τα τρισδιάστατα γραφικά. Τα Swing και AWT components ορίζουν λειτουργίες που εκτελούν τα αντικείμενα που βρίσκονται ορισμένα μέσα στο Simple Universe.

Κάθε φορά που εκτελούνται λειτουργίες εμφανίζονται οι αντίστοιχες εντολές στην γραμμή εντολών για την οποία έχουμε μιλήσει στην παράγραφο 5.4.11 [5.4.11]. Η λειτουργία είναι η ίδια όπως και στις επιλογές της εφαρμογής για την Java 2D με την μόνη διαφορά ότι αντί η γραμμή εντολών να είναι μέρος της περιοχής σχεδίασης πλέον αποτελεί ξεχωριστό Panel το οποίο προστίθεται στο παράθυρο. Η νέα μορφή της γραμμής εντολών θα μελετηθεί σε επόμενη παράγραφο [11.6.11].

11.5 Παρουσιαση

Η έβδομη επιλογή της εφαρμογής είναι το αντικείμενο της κλάσης Trans3dPanel. Είναι ένα panel που περιέχει Swing components όπως κουμπιά και ετικέτες. Περιέχει επίσης ένα αντικείμενο της κλάσης textPanel που είναι ένα JPanel που αποτελεί την γραμμή εντολών. Ακόμα κύριο συστατικό αυτής της επιλογής είναι ο εικονικός κόσμος που έχει δημιουργηθεί ο οποίος περιέχει έναν κύβο με πλευρές χρωματισμένες με διαφορετικά χρώματα. Σκοπός της επιλογής αυτής είναι ο χρήστης να μπορεί να επιλέξει ανάμεσα σε τρεις διαφορετικούς μετασχηματισμούς για τον κύβο στον οποίο μπορεί να αλλάξει το μέγεθος (scaling), τον προσανατολισμό (rotation) και την θέση (translation). Τέλος παρέχονται οι λειτουργίες επιστροφή στα εξώφυλλο και επαναφοράς του εικονικού κόσμου στην αρχική του κατάσταση.

Το Trans3dPanel έχει custom δομή που δημιουργείται με border Layout, Flow Layout και grid Layout.



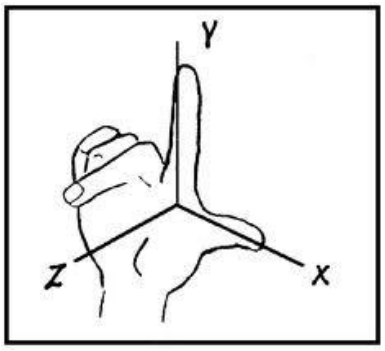
Εικονα 122 trans3dPanel layout

11.6 Λειτουργικοτητα

11.6.1 Right-Hand Coordinate System – Συστημα Συντεταγμενων Δεξιου Χεριου

Οι συντεταγμένες τριών διαστάσεων στα πλαίσια της εφαρμογής αυτής θα αναφέρονται πάντα στο σύστημα συντεταγμένων δεξιού χεριού. Για να γίνει σωστά ο προσανατολισμός χρησιμοποιούμε τον αντίχειρα ως τον x άξονα, τον δείκτη ως τον y άξονα και τον μέσο ως τον z άξονα. Στο σύστημα συντεταγμένων δεξιού χεριού ο x άξονας συμπίπτει με τον y άξονα αν περιστραφεί κατά γωνία 90 μοιρών αντίστροφα με την φορά του ρολογιού ως προς τον z άξονα, ο z άξονας με τον x επίσης συμπίπτουν αν ο z περιστραφεί κατά γωνία 90 μοιρών αντίστροφα με την φορά του ρολογιού ως προς τον y άξονα όπως και ο y άξονας συμπίπτει με τον z αν ο y περιστραφεί

κατά γωνία 90 μοιρών αντίστροφα με την φορά του ρολογιού ως προς τον x άξονα . Η περιστροφή των αξόνων αντίστροφα με την φορά του ρολογιού αναλογούν σε περιστροφή με θετική γωνία.



Εικονα 123 συστημα συντεταγμενων δεξιου χεριου

11.6.2 Ομογενείς Συντεταγμένες στον Τρισδιάστατο Χώρο

Στην παράγραφο 8.3 [8.3] παρουσιάστηκαν οι ομογενείς συντεταγμένες για να μπορέσουμε να εκφράσουμε τους μετασχηματισμούς στο επίπεδο με τη μορφή πολλαπλασιασμού πινάκων. Οι ομογενείς συντεταγμένες επεκτείνονται κατά μία διάσταση έτσι ώστε να μπορούν να εκφραστούν και οι μετασχηματισμοί στον τρισδιάστατο χώρο. Έτσι ένα σημείο στον τρισδιάστατο χώρο $\mathbb{R}^3$ αντιπροσωπεύεται από τέσσερις συντεταγμένες (x,y,z,w) όπου w≠0. Το σημείο (x,y,z,w) σε ομογενείς συντεταγμένες αντιπροσωπεύει το σημείο $(\frac{x}{w},\frac{y}{w},\frac{z}{w}) \in \mathbb{R}^3$ σε καρτεσιανές συντεταγμένες. Το σημείο $(x,y,z) \in \mathbb{R}^3$ μπορεί να αντιπροσωπευτεί από το σημείο $(x,y,z,1)$ σε ομογενείς συντεταγμένες. Βέβαια αυτός δεν είναι ο μοναδικός τρόπος. Οποιαδήποτε αναπαράσταση της μορφής $(x \cdot w, y \cdot w, z \cdot w, w)$ με w≠0 κωδικοποιεί το ίδιο σημείο επίσης. Οποιοσδήποτε πολύπλοκος μετασχηματισμός μπορεί να γραφεί ως πολλαπλασιασμός πινάκων απλών μετασχηματισμών. Για τέσσερα σημεία που δεν βρίσκονται στο ίδιο επίπεδο έστω p1, p2, p3, p4 $\in \mathbb{R}^3$ υπολογίζουμε τα σημεία p1', p2', p3' p4' μετά τον μετασχηματισμό λύνοντας το σύστημα γραμμικών εξισώσεων

$$pi' = M \cdot pi \;\; με \; i = 1,2,3,4$$

Με M τον πίνακα που περιέχει τον πολύπλοκο μετασχηματισμό που είναι της μορφής :

$$M = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

11.6.3 Αντικειμενα στην Java 3D

Η Java 3D για να μπορέσει να αποδώσει ένα τρισδιάστατο αντικείμενο μέσα σε έναν εικονικό κόσμο πρέπει να ορίσει εκτός από την γεωμετρία του και την εμφάνισή του. Με τον όρο εμφάνιση εννοούμε να προσδιορίσει ένα χρώμα ή μία υφή για την επιφάνειά του. Επίσης να προσδιορίσει πόσο λαμπερό ή μουντό είναι το χρώμα της επιφάνειάς του. Η εμφάνιση της επιφάνειας του αντικειμένου είναι αντικείμενο της κλάσης Appearance. Λεπτομέρειες για τις

διάφορες παραμέτρους της κλάσης αυτής θα συζητηθούν σε επόμενο κεφάλαιο [12]. Για να δημιουργήσουμε ένα στιγμιότυπο της κλάσης Appearance γράφουμε :

Appearance App = new Appearance()

setToMyDefaultAppearance(App,new Color3f(red, green, blue))

Στα πλαίσια αυτής της επιλογής χρησιμοποιούμε την μέθοδο set To My Default Appearance() η οποία αναθέτει ένα χρώμα, στιγμιότυπο της κλάσης Color3f, στην εμφάνιση App. Το χρώμα ορίζεται από τρεις float τιμές τις red, green, blue ∈ [0,1] που αντιστοιχούν στις εντάσεις του κόκκινου, πράσινου και μπλε χρώματος για το μοντέλο χρωμάτων RGB. Η συνάρτηση set To My Default Appearance () μπορεί να βρεθεί στα παραδείγματα του βιβλίου «Introduction to Computer Graphics Using java 2D and 3D» του συγγραφέα Frank Klawonn όπως και στο Παράρτημα Α [Παράρτημα Α]. Εφόσον έχει δημιουργηθεί μία εμφάνιση για την επιφάνεια των αντικειμένων μπορούμε να ορίσουμε αντικείμενα χρησιμοποιώντας την εμφάνιση αυτή. Τα βασικά αντικείμενα στην Java 3D είναι το κουτί, η σφαίρα, ο κώνος και ο κύλινδρος. Όλες οι τιμές των παραμέτρων θα είναι float.
Για το κουτί γράφουμε :

Box box = new Box(x,y,z,App)

Όπου δημιουργείται ένα κουτί με διαστάσεις (2x) x (2y) x (2z) στο κέντρο του συστήματος συντεταγμένων με την επιφάνειά του να έχει χρωματιστεί σύμφωνα με το στιγμιότυπο App της κλάσης Appearance.
Για την σφαίρα γράφουμε :

Sphere sphere = new Sphere(radius,App)

Όπου δημιουργείται μία σφαίρα ακτίνας radius με κέντρο το κέντρο του συστήματος συντεταγμένων χρωματισμένη σύμφωνα με το στιγμιότυπο App της κλάσης Appearance.
Για τον κώνο γράφουμε :

Cone cone = new Cone(radius, height,App)

"Όπου δημιουργείται έναν κώνος με ακτίνα radius και ύψος height με κέντρο το κέντρο του συστήματος συντεταγμένων χρωματισμένος σύμφωνα με το στιγμιότυπο App της κλάσης Appearance. Ο κώνος έχει τοποθετηθεί έτσι ώστε ο άξονας κατά μήκος του ύψους του να συμπίπτει με τον άξονα των y. Αυτό σημαίνει ότι ο κώνος είναι κεντραρισμένος γύρω από τον y άξονα με height/2 του κώνου πάνω από το επίπεδο x/y και height/2 του κώνου κάτω από το επίπεδο x/y.
Για τον κύλινδρο γράφουμε :

Cylinder cylinder = new Cylinder(radius, height, App)

"Όπου δημιουργείται έναν κύλινδρος με ακτίνα radius και ύψος height με κέντρο το κέντρο του συστήματος συντεταγμένων χρωματισμένος σύμφωνα με το στιγμιότυπο App της κλάσης Appearance. Ο κύλινδρος όπως και ο κώνος έχει τοποθετηθεί έτσι ώστε ο άξονας κατά μήκος του ύψους του να συμπίπτει με τον άξονα των y. Αυτό σημαίνει ότι ο κύλινδρος είναι κεντραρισμένος γύρω από τον y άξονα με height/2 του κυλίνδρου πάνω από το επίπεδο x/y και height/2 του κυλίνδρου κάτω από το επίπεδο x/y.

Στην επιλογή αυτή χρησιμοποιούνται όλα τα παραπάνω αντικείμενα για να δημιουργηθεί το σύστημα αξόνων που υπάρχει στην κάτω αριστερή γωνία του εικονικού κόσμου. Για το σύστημα αξόνων θα αναφερθούμε σε ξεχωριστή παράγραφο [11.6.8]. Όμως το κύριο συστατικό αυτής της

επιλογής είναι ο χρωματισμένος κύβος στον οποίο εφαρμόζονται όλοι οι μετασχηματισμοί. Ο χρωματισμένος κύβος είναι αντικείμενο της κλάσης Color Cube για την οποία γράφουμε :

ColorCube cube = new ColorCube(double scale)

Όπου δημιουργείται ένας κύβος με πλευρές χρωματισμένες με διαφορετικά χρώματα κεντραρισμένος ως προς το κέντρο του συστήματος συντεταγμένων με τις γωνίες του να είναι οι [-scale, -scale, -scale] και [scale, scale, scale]. Η default τιμή της μεταβλητής scale είναι 1 ενώ στην εφαρμογή η τιμή της μεταβλητής scale είναι 0.15 (ο κύβος εμφανίζεται μικρότερος).

11.6.4 Scaling – Αλλαγη Μεγεθους Τρισδιαστατου Σχηματος

Όπως έχει αναφερθεί αναλυτικά στην παράγραφο 8.2.1 [8.2.1] ένας μετασχηματισμός scaling οδηγεί σε ένα τέντωμα ή σε μία σμίκρυνση του σχήματος στον x ή/και στον y άξονα. Πλέον εφόσον μιλάμε για σχήματα στον τρισδιάστατο χώρο ένας μετασχηματισμός scaling οδηγεί σε τέντωμα ή σε σμίκρυνση και στον z άξονα. Ένας μετασχηματισμός scaling S(sx , sy, sz) προβάλει το σημείο (x , y , z) στο σημείο (x' , y' , z'). Το σημείο (x' , y' ,z') δίνεται από τον τύπο σε ομογενείς συντεταγμένες :

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} sx \cdot x \\ sy \cdot y \\ sz \cdot z \\ 1 \end{pmatrix} \text{ με } S(sx, sy, sz) = \begin{pmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

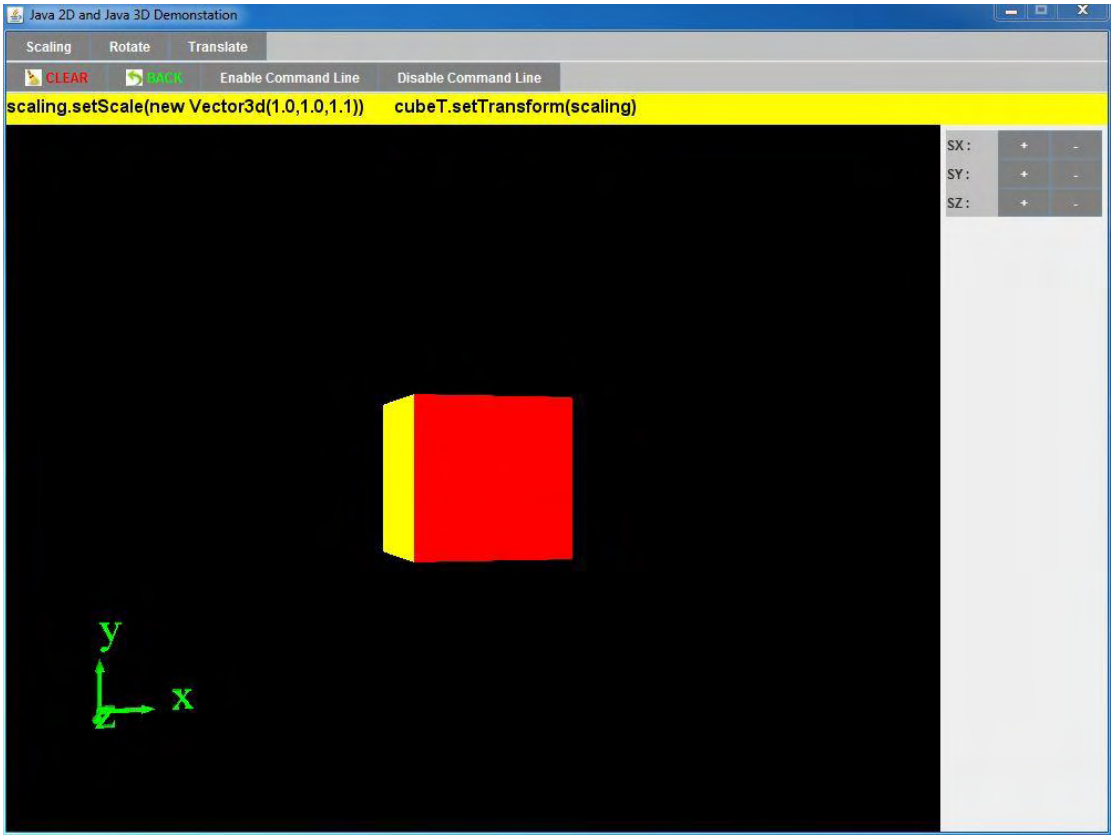
Για να δημιουργήσουμε μετασχηματισμούς χρησιμοποιούμε την κλάση Transform3D η οποία αποθηκεύει τρισδιάστατους μετασχηματισμούς με την μορφή πίνακα σε ομογενείς συντεταγμένες και δημιουργεί τον μοναδιαίο πίνακα.

Transform3D scaling = new Transform3D()

Έπειτα χρησιμοποιώντας την μέθοδο set Scale() εισάγοντας τους παράγοντες sx, sy και sz οδηγούμαστε σε ένα scale S(sx,sy,sz) :

scaling.setScale(new Vector3f(sx, sy, sz))

Στην εφαρμογή για έναν μετασχηματισμό scaling πατάμε το κουμπί «Scaling» όπου εμφανίζεται ένα νέο panel στα δεξιά του εικονικού κόσμου. Το panel αυτό περιέχει κουμπιά με τα οποία γίνεται scaling κατά 0.1 ή -0.1 στους άξονες x, y, z. Με κάθε αλλαγή των sx ή sy ή sz το σχήμα εμφανίζεται ανανεωμένο.



ΕΙΚΟΝΑ 124 SCALING

11.6.5 ROTATE – ΑΛΛΑΓΗ ΠΡΟΣΑΝΑΤΟΛΙΣΜΟΥ ΔΙΣΔΙΑΣΤΑΤΟΥ ΣΧΗΜΑΤΟΣ

Όπως έχει αναφερθεί αναλυτικά στην παράγραφο 8.2.2 [8.2.2] ένας μετασχηματισμός rotate οδηγεί στην αλλαγή προσανατολισμού του σχήματος κατά τον παράγοντα θ δηλαδή την γωνία περιστροφής. Ένας μετασχηματισμός rotation R(θ) κατά γωνία θ προβάλει το σημείο (x , y, z) στο σημείο (x’ , y’, z’). Στον δισδιάστατο χώρο θεωρούσαμε περιστροφές γύρω από την αρχή των αξόνων πλέον εφόσον μιλάμε για τον τρισδιάστατο χώρο ο μετασχηματισμός rotate μπορεί να γίνει γύρω από τον άξονα των x, των y ή/και των z. Το σημείο (x’ , y’, z’) για περιστροφή γύρω από τον άξονα των x δίνεται από τον τύπο :

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sin\theta & 0 \\ 0 & sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \text{ με } Rx(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sin\theta & 0 \\ 0 & sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Μία περιστροφή γύρω από τον άξονα x αφήνει την x συντεταγμένη του σημείου ίδια. Αντίστοιχα συμβαίνει και στις περιστροφές γύρω από τους άξονες z και y.Το σημείο (x’ , y’, z’) για περιστροφή γύρω από τον άξονα των z δίνεται από τον τύπο :

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \text{ με } Rz(\theta) = \begin{pmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Το σημείο (x’ , y’, z’) για περιστροφή γύρω από τον άξονα των y δίνεται από τον τύπο :

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \text{με } Ry(\theta) = \begin{pmatrix} cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Όπως και στον μετασχηματισμό scaling χρησιμοποιούμε την κλάση Transform3D η οποία αποθηκεύει τρισδιάστατους μετασχηματισμούς με την μορφή πίνακα σε ομογενείς συντεταγμένες και δημιουργεί τον μοναδιαίο πίνακα.

Transform3D rotation = new Transform3D()

Η μέθοδος rot X() ορίζει μία περιστροφή ως προς τον άξονα των x υπό τη γωνία theta.

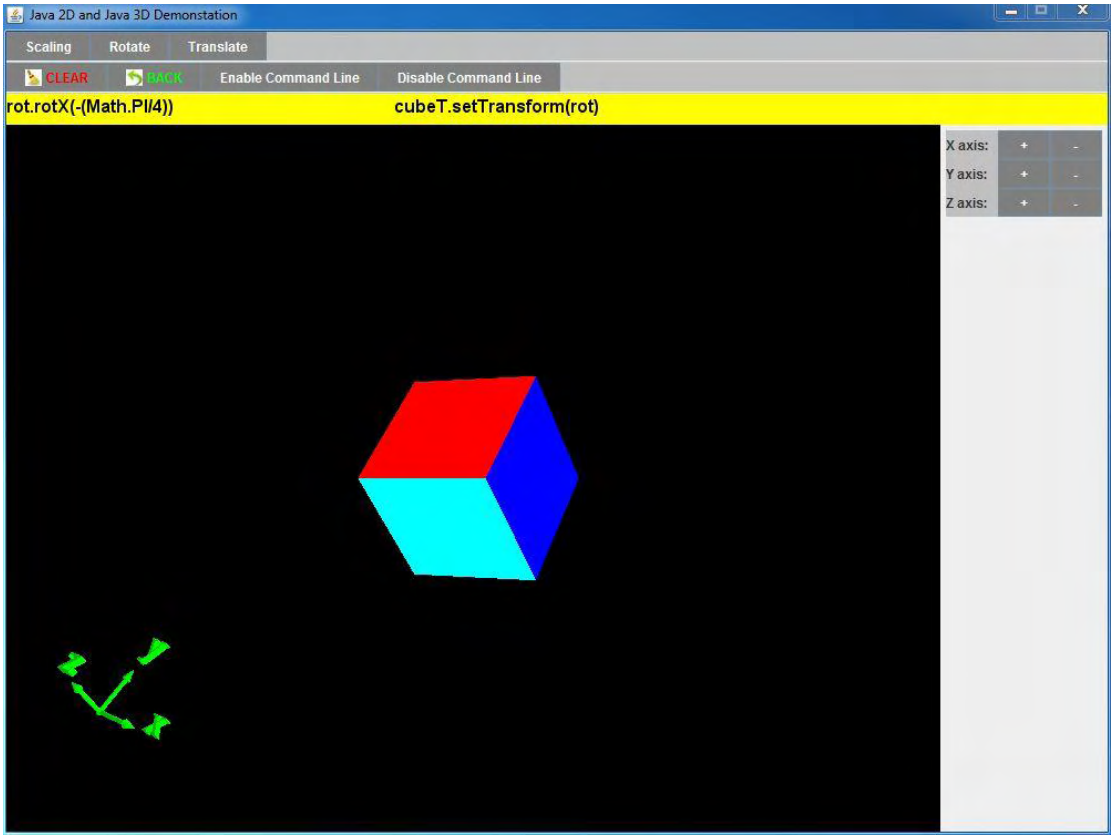
rotation.rotX(theta)

Για τους άξονες y και z χρησιμοποιούμε τις μεθόδους rot Y() και rot Z().

rotation.rotY(theta)

rotation.rotZ(theta)

Στην εφαρμογή για έναν μετασχηματισμό rotate πατάμε το κουμπί «Rotate» όπου εμφανίζεται ένα νέο panel στα δεξιά του εικονικού κόσμου. Το panel αυτό περιέχει κουμπιά με τα οποία γίνεται περιστροφή κατά γωνία 45 ή -45 μοιρών ως προς τους άξονες x,y,z. Μετά από κάθε περιστροφή το σχήμα εμφανίζεται ανανεωμένο.



ΕΙΚΟΝΑ 125 ROTATION

11.6.6 TRANSLATE – ΑΛΛΑΓΗ ΘΕΣΗΣ ΤΡΙΣΔΙΑΣΤΑΤΟΥ ΣΧΗΜΑΤΟΣ

Όπως έχει αναφερθεί αναλυτικά στην παράγραφο 8.2.4 [8.2.4] ένας μετασχηματισμός translation οδηγεί σε αλλαγή της θέσης του σχήματος. Ένας μετασχηματισμός translation T(dx , dy , dz) οδηγεί σε μετατόπιση κατά το διάνυσμα d=(dx , dy , dz) ^{T} . Ο μετασχηματισμός translation προβάλει το σημείο (x , y, z) στο σημείο (x' , y', z') που δίνεται από τον τύπο :

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x + dx \\ y + dy \\ z + dz \\ 1 \end{pmatrix} \text{ με } T(dx, dy, dz) = \begin{pmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

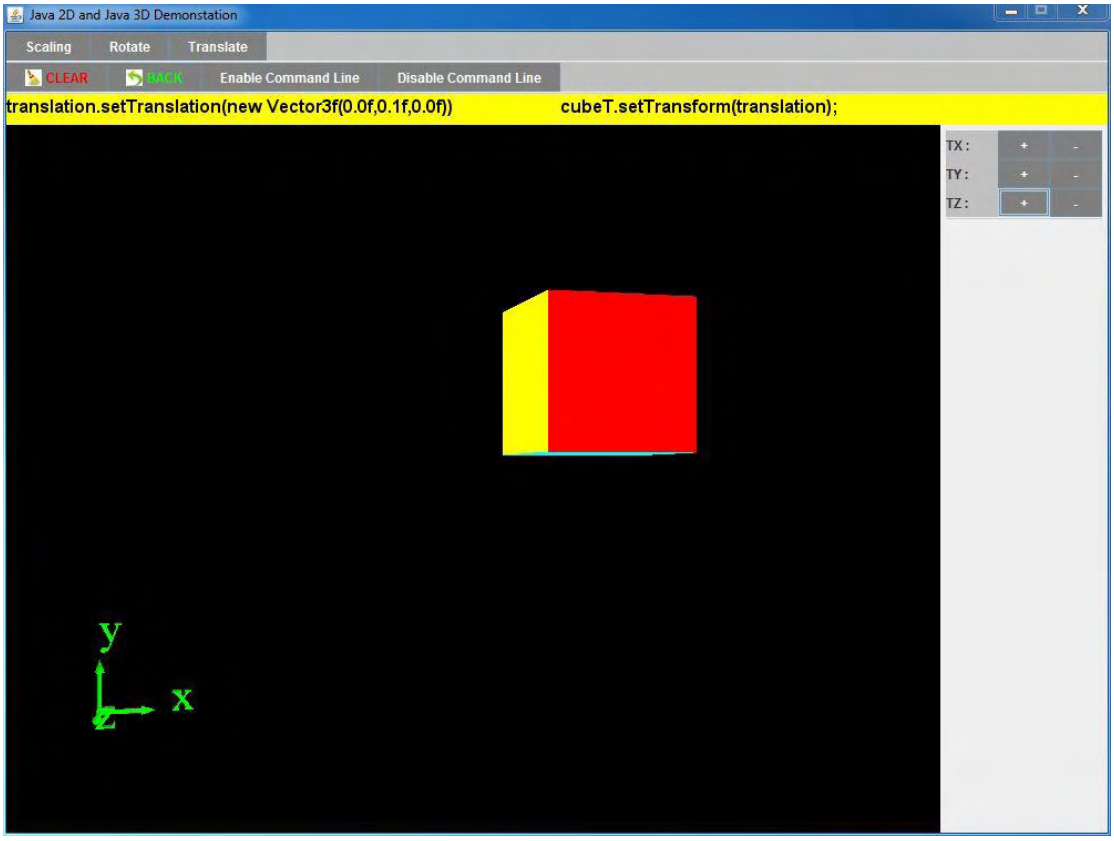
Χρησιμοποιούμε την κλάση Transform3D γράφοντας :

Transform3D translation = new Transform3D()

Έπειτα χρησιμοποιώντας την μέθοδο set Translation() εισάγοντας τους παράγοντες dx, dy και dz οδηγούμαστε σε ένα translation $T(dx, dy, dz)$:

translation.setTranslation(new Vector3f(dx, dy, dz))

Στην εφαρμογή για έναν μετασχηματισμό translation πατάμε το κουμπί «Translate» όπου εμφανίζεται ένα νέο panel στα δεξιά του εικονικού κόσμου. Το panel αυτό περιέχει κουμπιά με τα οποία γίνεται translation κατά 0.1 ή -0.1 σύμφωνα με τους παράγοντες dx, dy και dz. Με κάθε αλλαγή των dx ή dy ή dz το σχήμα εμφανίζεται ανανεωμένο.



Εικονα 126 Translation

11.6.7 Συνθετοι Μετασχηματισμοι

Η επιλογή αυτή της εφαρμογής δίνει την δυνατότητα στον χρήστη να εφαρμόσει συνδυασμούς απλών μετασχηματισμών έτσι ώστε να δημιουργηθούν πιο πολύπλοκοι μετασχηματισμοί. Είναι σύνηθες να αναθέτουμε σε ομάδες που ονομάζονται transformation groups αντικείμενα τα οποία θέλουμε να μετακινήσουμε στον τρισδιάστατο χώρο εφαρμόζοντάς τους μετασχηματισμούς. Το αντικείμενο της κλάσης Color Cube, το cube, ανατίθεται στο transformation group CubeT για να μπορούμε να το μετακινήσουμε. Το transformation group είναι κόμβος του scene graph. Το αντικείμενο cube για να φαίνεται πιο καλά η τρισδιάστατη μορφή του εμφανίζεται μόλις ανοίγει το παράθυρο της επιλογής «Geometric Transformations in Java 3D» περιστρεμμένο ως προς τον άξονα των y κατά γωνία 15 μοιρών. Ο μετασχηματισμός είναι ο :

Transform3D tfPlatform = new Transform3D()

tfPlatform.rotY(Math.PI/12)

Θέλουμε να εφαρμόσουμε τον παραπάνω μετασχηματισμό στο transformation group στο οποίο θα αναθέσουμε ως παιδί τον κύβο. Κάθε transformation Group δημιουργείται βάση ενός

μετασχηματισμού. Έτσι για να δημιουργήσουμε το transformation group για τον μετασχηματισμό tfPlatform γράφουμε :

TransformGroup cubeT = new TransformGroup(tfPlatform)

Για να συνδέσουμε το transformation group με τον κύβο cube ορίζουμε τον κύβο ως παιδί του transformation group γράφοντας :

cubeT.addChild(cube)

Θα μπορούσαμε να αναθέσουμε και άλλα παιδιά στο cubeT με αποτέλεσμα οποιοσδήποτε μετασχηματισμός εφαρμοστεί στον cubeT να εφαρμοστεί και στα παιδιά του. Ο κύβος εμφανίζεται ελαφρά περιστρεμμένος ως προς άξονα y, για να συνδυάσουμε την περιστροφή αυτή με κάποιον άλλο μετασχηματισμό για παράδειγμα με ένα scaling κατά παράγοντα sx θα πρέπει αρχικά να δημιουργήσουμε ένα καινούριο αντικείμενο Transform3D για το scaling γράφοντας :

Transform3D tempscaling = new Transform3D()

tempscaling.setScale(new Vector3f(1.1, 1.0, 1.0))

Αν εφαρμόσουμε αυτόν τον μετασχηματισμό κατευθείαν στο cubeT transformation group όπως κάναμε με την περιστροφή ο κύβος θα κάνει scale κατά παράγοντα sx χωρίς όμως να υπολογιστεί ή αρχική περιστροφή, δηλαδή θα γίνει scale στον αρχικό κύβο. Για να μπορέσουμε να κάνουμε το scaling στον ήδη περιστρεμμένο κύβο πρέπει πρώτα να αποθηκεύσουμε τα δεδομένα της περιστροφής σε μία προσωρινή μεταβλητή. Η αποθήκευση γίνεται με την βοήθεια της μεθόδου get Transform() που αποθηκεύει την Transform3D μεταβλητή, δηλαδή τον πίνακα της περιστροφής που ανήκει στο transformation group για το οποίο καλείται η μέθοδος.

Transform3D temp = new Transform3D()

cubeT .getTransform(temp)

Έπειτα δημιουργούμε τον νέο πίνακα του μετασχηματισμού με την μέθοδο mul() γράφοντας :

temp.mul(tempscaling)

Η μέθοδος mul() δημιουργεί τον συνδυασμό των 2 μετασχηματισμών με την έννοια του πολλαπλασιασμού πινάκων των Transform3D μεταβλητών temp και tempscaling. Το αποτέλεσμα των 2 μετασχηματισμών αποθηκεύεται στην μεταβλητή temp. Η μεταβλητή temp πλέον περιέχει έναν πίνακα που αποτελεί μία περιστροφή και ένα scaling. Για να εφαρμόσουμε τον νέο μετασχηματισμό temp στο transformation group cubeT χρησιμοποιούμε την μέθοδο set Transform() :

cubeT.setTransform(temp)

Στην επιλογή αυτή της εφαρμογής ακολουθείται η παραπάνω διαδικασία για την δημιουργία σύνθετων μετασχηματισμών.

Είναι σημαντικό να αναφέρουμε ότι για να είναι δυνατό το διάβασμα αλλά και το γράψιμο (αλλαγή) των πληροφοριών των αντικειμένων που ανήκουν στο transformation group πρέπει να αναθέσουμε «ικανότητες» (capabilities) στο transformation group. Γενικά η ανάθεση των capabilities είναι πολύ σημαντικό κομμάτι του προγράμματος γιατί διασφαλίζει τη σωστή λειτουργία του προγράμματος αλλά επίσης δίνει την δυνατότητα αλλαγής πολλών χαρακτηριστικών των αντικειμένων που θα χρειαστεί να τροποποιήσουμε αφού γίνουν live τα αντικείμενα. Για το transformation group γράφουμε :

cubeT.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE)

cubeT.setCapability(TransformGroup.ALLOW_TRANSFORM_READ)

11.6.8 Σύστημα Αξονων

Μέσα στον εικονικό κόσμο που έχει δημιουργηθεί για αυτή την επιλογή της εφαρμογής υπάρχει και ένα σύστημα αξόνων το οποίο βρίσκεται στην κάτω αριστερή γωνία του παραθύρου. Το σύστημα αξόνων υπάρχει για να βοηθά τον χρήστη να εντοπίζει σωστά τους άξονας κατά την διάρκεια εφαρμογής των μετασχηματισμών. Οι άξονες μετακινούνται σύμφωνα με τις περιστροφές που εφαρμόζονται στον κύβο και μένουν σταθεροί για τα scaling και τα translation μιας που εκείνοι οι μετασχηματισμοί μεταβάλουν το μέγεθος και την θέση του αντικειμένου και όχι τον προσανατολισμό του.

Το σύστημα αξόνων είναι μία δομή που αποτελείται από μία σφαίρα, τρεις κυλίνδρους και τρεις κώνους όλα χρωματισμένα σε πράσινο χρώμα. Όλα αυτά τα αντικείμενα καθώς δημιουργούνται τοποθετούνται στην αρχή του συστήματος συντεταγμένων δηλαδή στο κέντρο του παραθύρου. Για να σχηματιστεί σωστά το σύστημα αξόνων εφαρμόζουμε μετασχηματισμούς σε κάθε αντικείμενο ξεχωριστά. Έτσι δημιουργούμε το σύστημα αξόνων στην αρχή του συστήματος συντεταγμένων. Για να περιστρέψουμε κατά 15 μοίρες ως προς τον άξονα των y το σύστημα αξόνων (γιατί και ο κύβος είναι περιστρεμμένος κατά αυτή την γωνία κατά την εκκίνηση της επιλογής) και να το μεταφέρουμε στην κάτω αριστερή γωνία χρησιμοποιούμε transformation groups από αυτά τα αντικείμενα έτσι ώστε να τα αντιμετωπίσουμε σαν μία ενότητα. Πιο αναλυτικά για την εμφάνιση γράφουμε :

Appearance greenApp = new Appearance()

setToMyDefaultAppearance(greenApp,new Color3f(0.0f,0.7f,0.0f))

Για την σφαίρα :

float sphereR = 0.008f

Sphere sphere = new Sphere(sphereR,greenApp)

Για τους κυλίνδρους που λειτουργούν ως άξονες του συστήματος :

float axisHeight = 0.1f

Cylinder yaxis = new Cylinder(0.005f,axisHeight,greenApp) // ορισμός κυλίνδρου για τον άξονα y

Transform3D yaxis1 = new Transform3D()

yaxis1.setTranslation(new Vector3f(0.0f,0.05f,0.0f)) // μετακίνηση κυλίνδρου ως προς 0.05 παράγοντα στον y άξονα

yaxisT = new TransformGroup(yaxis1) // transformation group για τον άξονα y

yaxisT.addChild(yaxis)

Cylinder zaxis = new Cylinder(0.005f,axisHeight,greenApp) // ορισμός κυλίνδρου για τον άξονα z

Transform3D zaxis1 = new Transform3D()

zaxis1.setTranslation(new Vector3f(0.0f,0.0f,0.05f)) // μετακίνηση κυλίνδρου ως προς 0.05 παράγοντα στον z άξονα

zaxis2 = new Transform3D()

zaxis2.rotX(Math.PI/2) // περιστροφή κυλίνδρου κατά γωνία 90 μοιρών ως προς τον άξονα x

zaxis1.mul(zaxis2) // συνδυασμός μετασχηματισμών

zaxisT = new TransformGroup(zaxis1) // transformation group για τον άξονα z

zaxisT.addChild(zaxis)

Cylinder xaxis = new Cylinder(0.005f,axisHeight,greenApp) // ορισμός κυλίνδρου για τον άξονα x

Transform3D xaxis1 = new Transform3D()

xaxis1.setTranslation(new Vector3f(0.05f,0.0f,0.0f)) // μετακίνηση κυλίνδρου ως προς 0.05 παράγοντα στον x άξονα

xaxis2 = new Transform3D()

xaxis2.rotZ(-Math.PI/2) // περιστροφή κυλίνδρου κατά γωνία -90 μοιρών ως προς τον άξονα z

xaxis1.mul(xaxis2) // συνδυασμός μετασχηματισμών

xaxisT = new TransformGroup(xaxis1) // transformation group για τον άξονα x

xaxisT.addChild(xaxis)

Για τους κώνους που λειτουργούν ως τα βέλη των αξόνων :

float arrowHeight = 0.03f

Cone xarrow = new Cone(0.01f,arrowHeight,greenApp) // ορισμός κώνου για τον άξονα x

Transform3D xarrow1 = new Transform3D()

xarrow1.setTranslation(new Vector3f(0.1f,0.0f,0.0f)) // μετακίνηση κώνου ως προς 0.1 παράγοντα στον x άξονα

xarrow2 = new Transform3D()

xarrow2.rotZ(-Math.PI/2) // περιστροφή κώνου κατά γωνία -90 μοιρών ως προς τον άξονα z

xarrow1.mul(xarrow2) // συνδυασμός μετασχηματισμών

xarrowT = new TransformGroup(xarrow1) // transformation group για το βέλος του άξονα x

xarrowT.addChild(xarrow)

Cone yarrow = new Cone(0.01f,arrowHeight,greenApp) // ορισμός κώνου για τον άξονα y

Transform3D yarrow1 = new Transform3D()

yarrow1.setTranslation(new Vector3f(0.0f,0.1f,0.0f)) // μετακίνηση κώνου ως προς 0.1 παράγοντα στον y άξονα

yarrowT = new TransformGroup(yarrow1) // transformation group για το βέλος του άξονα y

yarrowT.addChild(yarrow)

Cone zarrow = new Cone(0.01f,arrowHeight,greenApp) // ορισμός κώνου για τον άξονα z

Transform3D zarrow1 = new Transform3D()

zarrow1.setTranslation(new Vector3f(0.0f,0.0f,0.1f)) // μετακίνηση κώνου ως προς 0.1 παράγοντα στον z άξονα

zarrow2 = new Transform3D()

zarrow2.rotX(Math.PI/2) // περιστροφή κώνου κατά γωνία 90 μοιρών ως προς τον άξονα x

zarrow1.mul(zarrow2) // συνδυασμός μετασχηματισμών

zarrowT = new TransformGroup(zarrow1) // transformation group για το βέλος του άξονα z

zarrowT.addChild(zarrow)

Εφόσον έχουμε δημιουργήσει και συνθέσει σωστά το σύστημα αξόνων στην αρχή του συστήματος συντεταγμένων δημιουργούμε έναν μετασχηματισμό για περιστροφή του συστήματος αξόνων κατά γωνία 15 μοιρών ως προς άξονα y και αναθέτουμε όλα τα αντικείμενα που δημιουργήσαμε ως παιδιά σε έναν νέο transformation group το οποίο παίρνει ως όρισμα την παραπάνω περιστροφή και την εφαρμόζει σε όλα τα μέλη του.

Transform3D coordSystemF = new Transform3D()

coordSystemF.rotY(Math.PI/12)

TransformGroup coordSystemTF = new TransformGroup(coordSystemF) // το transformation group

coordSystemTF.addChild(sphere)

coordSystemTF.addChild(yaxisT)

coordSystemTF.addChild(zaxisT)

coordSystemTF.addChild(xaxisT)

coordSystemTF.addChild(xarrowT)

coordSystemTF.addChild(yarrowT)

coordSystemTF.addChild(zarrowT)

Αφού έχουμε περιστρέψει το σύστημα αξόνων πλέον το τοποθετούμε στην κάτω αριστερή γωνία σαν μία ενότητα εφαρμόζοντας ένα translation και δημιουργώντας το τελικό transformation group.

Transform3D coordSystem = new Transform3D()

coordSystem.setTranslation(new Vector3f(-0.8f,-0.5f,0.0f))

coordSystemT = new TransformGroup(coordSystem) // τελικό transformation group

Αναθέτουμε στο transformation group coordSystemT το προηγούμενο transformation group coordSystemTF ως παιδί.

coordSystemT.addChild(coordSystemTF)

Κάθε φορά που γίνεται μία περιστροφή στον κύβο περιστρέφουμε με την ίδια γωνία και ως προς τον ίδιο άξονα το transformation group coordSystemT με αποτέλεσμα να περιστρέφεται και όλη η ενότητα του συστήματος αξόνων. Για να μπορέσουμε να κάνουμε αλλαγές στις πληροφορίες των αντικειμένων που έχει ως μέλη το transformation group πρέπει να ορίσουμε τις αντίστοιχες capabilities για παροχή άδειας για διάβασμα και γράψιμο.

coordSystemT.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE)

coordSystemT.setCapability(TransformGroup.ALLOW_TRANSFORM_READ)

11.6.9 Τρισδιαστατο Κειμενο

Το κείμενο σε έναν τρισδιάστατο κόσμο μπορεί να είναι δύο ή τριών διαστάσεων. Η Java 3D παρέχει δύο κλάσεις την Text2D και Text3D για αναπαράσταση δισδιάστατου και τρισδιάστατου κειμένου σε μία σκηνή. Η κλάση Text2D προβάλει το κείμενο ως υφή (texture) πάνω σε ένα εικονικό διάφανο ορθογώνιο. Το κείμενο αυτό μπορεί να είναι ορατό μόνο από μπροστά. Η κλάση Text3D μπορεί να δημιουργήσει κείμενο που είναι ορατό από όλες τις γωνίες αλλά μπορεί να μην είναι ορατό από μία «κακή» γωνία.

Στην επιλογή αυτή της εφαρμογής έχουμε χρησιμοποιήσει τα γράμματα x, y, z για να προσδιορίσουμε τους άξονες του συστήματος αξόνων της προηγούμενης παραγράφου. Για να δημιουργήσουμε τρισδιάστατα γράμματα πρέπει πρώτα να ορίσουμε μία τρισδιάστατη γραμματοσειρά η οποία παράγεται από μία δισδιάστατη γραμματοσειρά.

Font3D f3d = new Font3D(new Font("serif", Font.PLAIN, 1), new FontExtrusion())

Η τρισδιάστατη γραμματοσειρά είναι η f3d ενώ για την δισδιάστατη χρησιμοποιείται απευθείας ένα αντικείμενο της κλάση Font το new Font("serif", Font.PLAIN, 1). Το αντικείμενο Font Extrusion καθορίζει πως η τρισδιάστατη γραμματοσειρά εξάγεται από την δισδιάστατη. Για να δημιουργήσουμε τα γράμματα χρησιμοποιούμε την κλάση Text3D και την κλάση Shape3D:

Text3D t3dx = new Text3D(f3d,new String("x"),new Point3f(0.0f,0.0f,0.0f)) // το γράμμα «x» του άξονα x

Shape3D textX = new Shape3D(t3dx,greenApp)

Text3D t3dy = new Text3D(f3d,new String("y"),new Point3f(0.0f,0.0f,0.0f)) // το γράμμα «y» του άξονα y

Shape3D textY = new Shape3D(t3dy,greenApp)

Text3D t3dz = new Text3D(f3d,new String("z"),new Point3f(0.0f,0.0f,0.0f)) // το γράμμα «z» του άξονα z

Shape3D textZ = new Shape3D(t3dz,greenApp)

Το κείμενο που εμφανίζεται είναι στην μορφή συμβολοσειράς. Το κείμενο έχει δημιουργηθεί με την τρισδιάστατη γραμματοσειρά που είναι αντικείμενο της κλάσης Font3D και τοποθετείται στο σημείο που καθορίζει το αντικείμενο της κλάσης Point3f. Για κάθε ένα γράμμα που δημιουργήθηκε , δημιουργείται και ένα αντικείμενο Shape3D. Το Shape3D αντικείμενο παίρνει ως παραμέτρους το κείμενο και ένα αντικείμενο της κλάσης Appearance για να καθορίσει την εμφάνιση του κειμένου. Στο συγκεκριμένο παράδειγμα χρησιμοποιείται το αντικείμενο greenApp που δημιουργήθηκε στην προηγούμενη παράγραφο [11.6.8] για να χρωματίσει το κείμενο με το πράσινο χρώμα που έχει χρωματιστεί και το σύστημα αξόνων. Τα Shape3D αντικείμενα μπορούν να προστεθούν στο scene graph ή σε transformation groups.

Τα γράμματα όταν δημιουργούνται τοποθετούνται στην αρχή του συστήματος συντεταγμένων και πρέπει για να τους δώσουμε τη σωστή θέση (στην κάτω αριστερά γωνία του παραθύρου) και το σωστό μέγεθος να χρησιμοποιήσουμε μετασχηματισμούς. Αρχικά ένα scaling για να μικρύνουμε το μέγεθος.

Transform3D tfLetterScaling = new Transform3D()

tfLetterScaling.setScale(0.09)

έπειτα δημιουργούμε ένα transformation group για κάθε γράμμα. Εφαρμόζουμε τον παραπάνω μετασχηματισμό αναθέτοντας ως παιδιά τα Shape3D αντικείμενα που περιέχουν τα γράμματα στο αντίστοιχο transformation group.

TransformGroup tgTextX = new TransformGroup(tfLetterScaling)

tgTextX.addChild(textX)

TransformGroup tgTextY = new TransformGroup(tfLetterScaling)

tgTextY.addChild(textY)

TransformGroup tgTextZ = new TransformGroup(tfLetterScaling)

addChild(textZ)

Ύστερα εφαρμόζουμε ένα translation στα γράμματα για να τα μεταφέρουμε στη σωστή τους θέση (λίγο μετά τα βέλη των αξόνων). Έπειτα αναθέτουμε τα γράμματα ως παιδιά στο transformation group coordSystemTF που έχει οριστεί έτσι ώστε να εφαρμόζει στα μέλη του περιστροφή κατά γωνία 15 μοιρών ως προς άξονα y. Το coordSystemTF όπως και ο αντίστοιχος μετασχηματισμός έχουν οριστεί στην προηγούμενη παράγραφο [11.6.8].

Transform3D textX1 = new Transform3D()

textX1.setTranslation(new Vector3f(0.15f,0.0f,0.0f)) // μετακίνηση γράμματος «x» ως προς 0.15 παράγοντα στον x άξονα

TransformGroup textXT = new TransformGroup(textX1) // transformation group για εφαρμοστεί το translation

textXT.addChild(tgTextX)

coordSystemTF.addChild(textXT) // transformation group για εφαρμοστεί η περιστροφή

Transform3D textY1 = new Transform3D()

textY1.setTranslation(new Vector3f(0.0f,0.15f,0.0f)) // μετακίνηση γράμματος «y»ως προς 0.15 παράγοντα στον y άξονα

TransformGroup textYT = new TransformGroup(textY1) // transformation group για εφαρμοστεί το translation

textYT.addChild(tgTextY)

coordSystemTF.addChild(textYT) // transformation group για εφαρμοστεί η περιστροφή

Transform3D textZ1 = new Transform3D()

textZ1.setTranslation(new Vector3f(0.0f,0.0f,0.15f)) // μετακίνηση γράμματος «z» ως προς 0.15 παράγοντα στον z άξονα

TransformGroup textZT = new TransformGroup(textZ1) // transformation group για εφαρμοστεί το translation

textZT.addChild(tgTextZ)

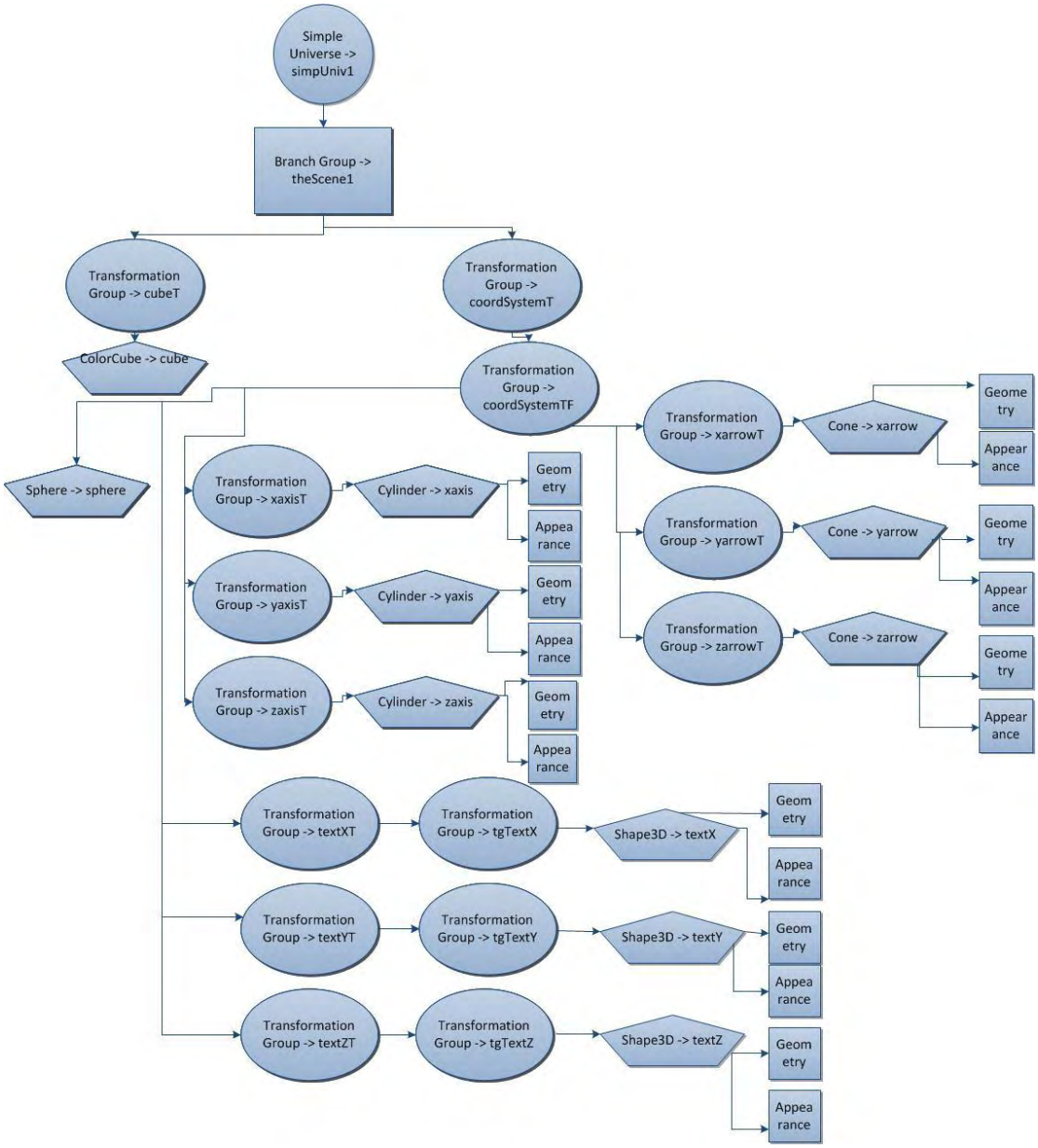
coordSystemTF.addChild(textZT) // transformation group για εφαρμοστεί η περιστροφή

Όπως έγινε και στην προηγούμενη παράγραφο με το σύστημα αξόνων τοποθετούμε τα γράμματα (μαζί με το σύστημα αξόνων εφόσον ανήκουν στον ίδιο transformation group) στην κάτω αριστερή γωνία εφαρμόζοντας ένα translation και αναθέτοντας ως παιδί του τελικού transformation group coordSystemT το transformation group coordSystemTF.

Κάθε φορά που γίνεται μία περιστροφή στον κύβο περιστρέφουμε με την ίδια γωνία και ως προς τον ίδιο άξονα το transformation group coordSystemT με αποτέλεσμα να περιστρέφεται όλη η ενότητα του συστήματος αξόνων μαζί με τα τρισδιάστατα γράμματα.

11.6.10 Το Scene graph της Επιλογης

Πλέον έχοντας μιλήσει για μετασχηματισμούς, transformation groups αλλά και για όλα τα αντικείμενα του εικονικού κόσμου που έχει δημιουργηθεί μπορούμε να παρουσιάσουμε το scene graph της επιλογής αυτής.



Εικονα 127 scene graph Trans3DPanel

Με κύκλο έχει σχεδιαστεί το αντικείμενο Simple Universe ενώ με ορθογώνιο το Branch Group. Με ελλείψεις έχουν σχεδιαστεί τα Transformation Groups που είναι οι κόμβοι του δέντρου, με πεντάγωνα τα φύλλα που είναι αντικείμενα Sphere, Cone, Cylinder, Color Cube και Shape3D και με τετράγωνα τα node components που αναφέρονται στα φύλλα του δέντρου.

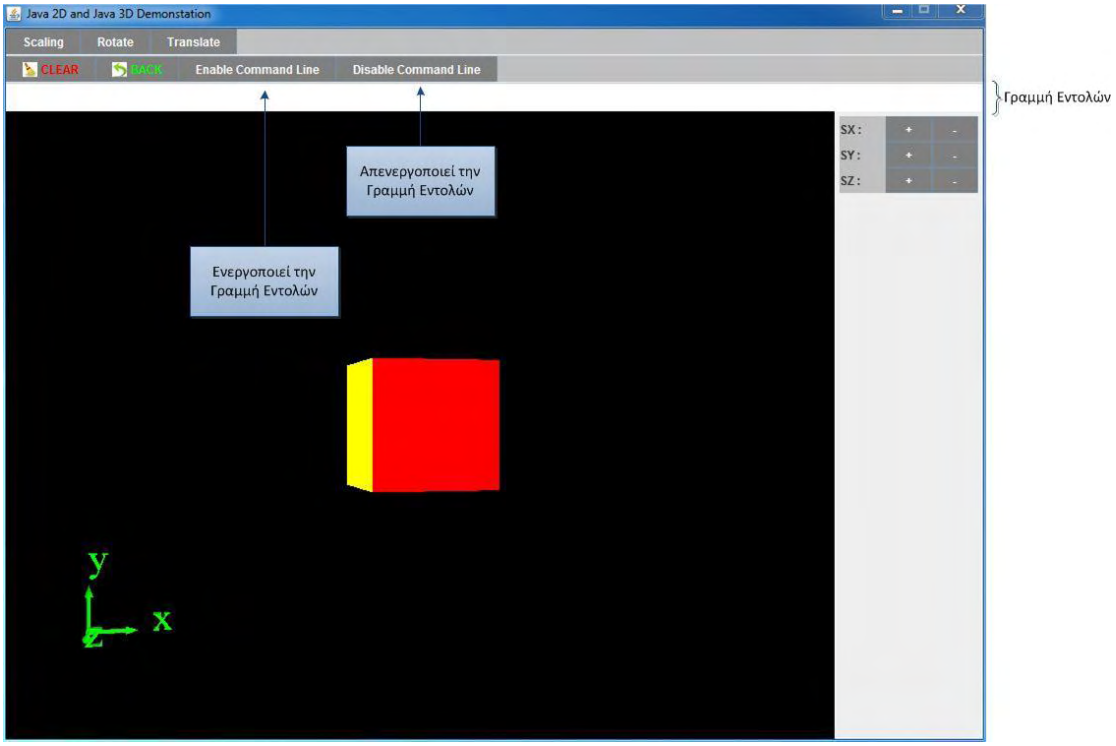
11.6.11 Command Line – Γραμμη Εντολων

Η γραμμή εντολών για την Java 2D ήταν το κίτρινο ορθογώνιο που εμφανιζόταν πάνω αριστερά στην περιοχή σχεδίασης και περιείχε τις εντολές που εκτελούσαν οι λειτουργίες της κάθε επιλογής της εφαρμογής. Για την Java 3D η γραμμή εντολών έχει την ίδια λειτουργικότητα αλλά διαφορετικό σχεδιασμό. Η δημιουργία τρισδιάστατων γραφικών απαιτεί την δημιουργία ενός εικονικού κόσμου μέσα στον οποίο σου δίνεται η δυνατότητα να περιηγηθείς και να τον δεις από

διαφορετικές γωνίες. Αν μέσα σε αυτόν τον κόσμο δημιουργούσαμε ένα κίτρινο ορθογώνιο (κουτί) που να εμφάνιζε τις εντολές δεν θα ήταν πάντα ορατό ανάλογα με το σημείο του εικονικού κόσμου στο οποίο βρισκόμασταν. Για αυτό το λόγο προτιμήθηκε να δημιουργηθεί η γραμμή εντολών μέσα σε ένα JPanel το οποίο αποτελεί κομμάτι κάθε επιλογής Java 3D της εφαρμογής. Ένας ακόμη λόγος δημιουργίας της γραμμής εντολών με αυτό τον τρόπο είναι η ομοιομορφία της εφαρμογής. Η γραμμή εντολών είναι μία ξεχωριστή εσωτερική κλάση της εφαρμογής η οποία παράγεται από την κλάση JPanel για αυτό έχει και τα χαρακτηριστικά του JPanel . Για κάθε μια από τις γραμμές εντολών που εμφανίζονται σε κάθε επιλογή για την Java 3D έχει δημιουργηθεί και μία νέα εσωτερική κλάση. Από τις κλάσεις αυτές δημιουργούνται 4 αντικείμενα (όσες και οι γραμμές εντολών) τα οποία προστίθενται στο παράθυρο σαν δομικά στοιχεία της κάθε επιλογής. Η γραμμή εντολών τοποθετείται στο πάνω μέρος του παραθύρου.

Το panel της γραμμής εντολών λειτουργεί σαν την περιοχή σχεδίασης μίας από τις Java 2D επιλογές. Δηλαδή αποτελείται από μία buffered image η οποία είναι χρωματισμένη με λευκό χρώμα και κάθε φορά που πρέπει να εμφανιστεί κάποια εντολή ζωγραφίζεται ένα κίτρινο ορθογώνιο με την αντίστοιχη εντολή με μαύρα γράμματα όπως έχει περιγραφεί στην παράγραφο 5.4.11 [5.4.11] . Υπάρχουν τα κουμπιά «Enable Command Line» και «Disable Command Line» τα οποία ενεργοποιούν και απενεργοποιούν αντίστοιχα την γραμμή εντολών. Κατά την απενεργοποίηση η γραμμή εντολών εμφανίζεται λευκή ενώ κατά την ενεργοποίηση εμφανίζεται το γνωστό κίτρινο ορθογώνιο με τις εντολές. Το JPanel της γραμμής εντολών είναι πάντα ορατό.

Για την επιλογή «Geometric Transformations in Java 3D» την γραμμή εντολών αποτελεί το αντικείμενο της κλάσης TextPanel. Για την επιλογή «Appearance in Java 3D» την γραμμή εντολών αποτελεί το αντικείμενο της κλάσης Text1Panel. Για την επιλογή «Lighting and Shading in Java 3D» την γραμμή εντολών αποτελεί το αντικείμενο της κλάσης Text2Panel ενώ για την επιλογή «Fog in Java 3D» το αντικείμενο της κλάσης Text3Panel.



Εικονα 128 γραμμη εντολων στην επιλογη «Geometric Transformations in Java 3D»- κουμπια για ενεργοποιηση και απενεργοποιηση

11.6.12 CLEAR AND BACK– ΕΠΑΝΑΦΟΡΑ ΚΑΙ ΕΠΙΣΤΡΟΦΗ

CLEAR

Με τον όρο «clear» στην επιλογή αυτή εννοούμε να μεταφέρουμε τον κύβο και το σύστημα αξόνων στην αρχική τους θέση και να μην είναι ορατό το πλευρικό panel με τα κουμπιά για το scaling, rotate και translate. Για να μεταφέρουμε τον κύβο όπως και το σύστημα αξόνων στην αρχική τους θέση εφαρμόζουμε στα transformation groups που ανήκουν τα αντικείμενα αυτά τους μετασχηματισμούς με τους οποίους τα τοποθετήσαμε κατά την δημιουργία του εικονικού κόσμου στις αρχικές τους θέσεις (ο κύβος στην αρχή του συστήματος συντεταγμένων και το σύστημα αξόνων στην κάτω αριστερή γωνία). Οι μετασχηματισμοί αυτοί είναι : για τον κύβο ο tfPlatform και για το σύστημα αξόνων ο coordSystem. Το transformation group για τον κύβο είναι το cubeT ενώ το transformation group για το σύστημα αξόνων είναι το coordSystemT. Με την βοήθεια της μεθόδου set Transform() χρησιμοποιούμε τους πίνακες που έχουν ήδη οριστεί για τους συγκεκριμένους μετασχηματισμούς και επαναφέρουμε τον κύβο και το σύστημα αξόνων στην σωστή τους θέση.

cubeT.setTransform(trans3dPanel.tfPlatform)

coordSystemT.setTransform(trans3dPanel.coordSystem)

Για να μην είναι ορατό το πλευρικό panel καλούμε την μέθοδο set Visible() με παράμετρο false για τα αντικείμενα που διαμορφώνουν το πλευρικό panel.

SMMenu.setVisible(false) // JPanel που περιέχει τα κουμπιά για το scaling

RMMenu.setVisible(false) // JPanel που περιέχει τα κουμπιά για το rotate

TMMenu.setVisible(false) // JPanel που περιέχει τα κουμπιά για το translate

Στην εφαρμογή για να επαναφέρουμε τον εικονικό κόσμο στην αρχική του κατάσταση πατάμε το κουμπί «Clear».

BACK

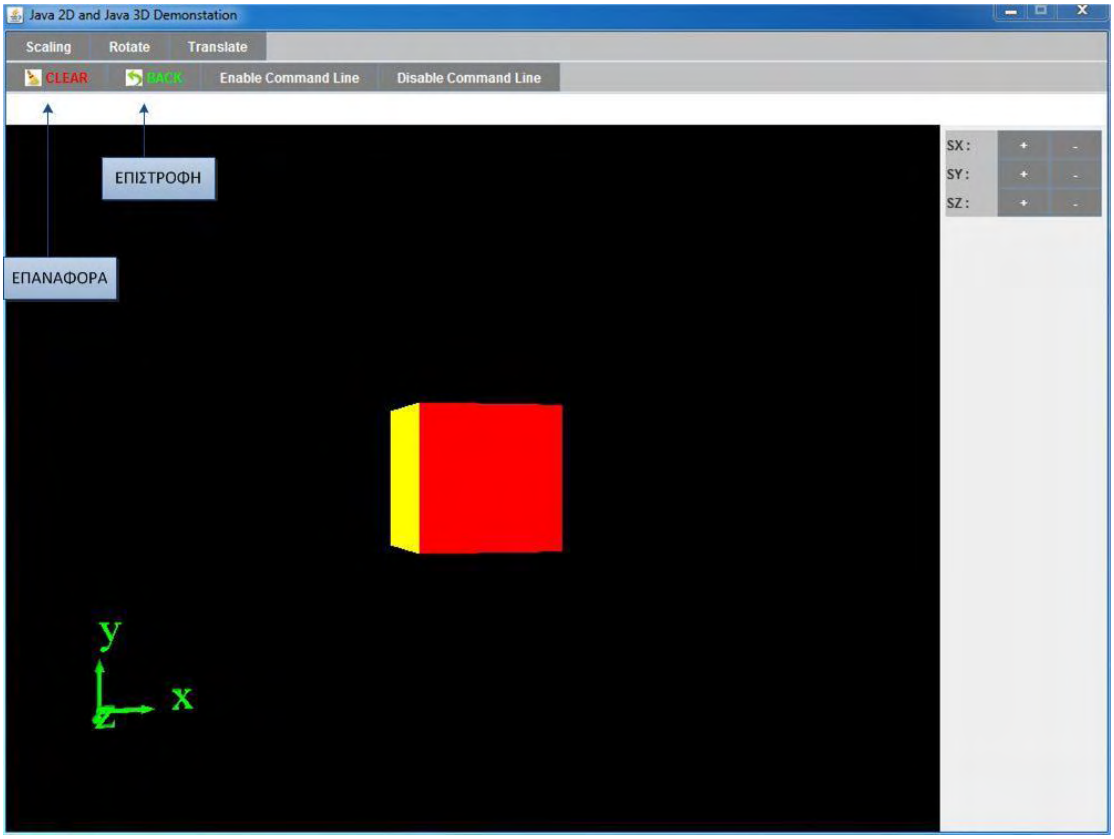
Η εφαρμογή είναι χωρισμένη σε επιλογές που επικοινωνούν μέσω του κοινού εξωφύλλου (CoverPanel). Είναι δυνατή η επιστροφή μέσα από μία εφαρμογή στο εξώφυλλο για να μας δοθεί η δυνατότητα να επιλέξουμε μία άλλη επιλογή. Για να πραγματοποιηθεί αυτό θα πρέπει όπως και στην διαδικασία της επαναφοράς (clear) να επαναφέρουμε τον εικονικό κόσμο στην αρχική του κατάσταση. Έτσι επαναφέρουμε τον κύβο και το σύστημα αξόνων και επίσης κάνουμε αόρατο το πλευρικό panel με τον κώδικα που αναφέρθηκε στο «Clear». Ακόμη πρέπει να γίνει ορατό το CoverPanel της εφαρμογής και να μην είναι πλέον ορατό το panel στο οποίο βρισκόμαστε. Για την επιλογή αυτή το panel είναι το Trans3dPanel.

trans3dPanel.setVisible(false)

setContentPane(coverPanel)

coverPanel.setVisible(true)

Στην εφαρμογή για να επιστρέψουμε στο εξώφυλλο πατάμε το κουμπί «BACK».



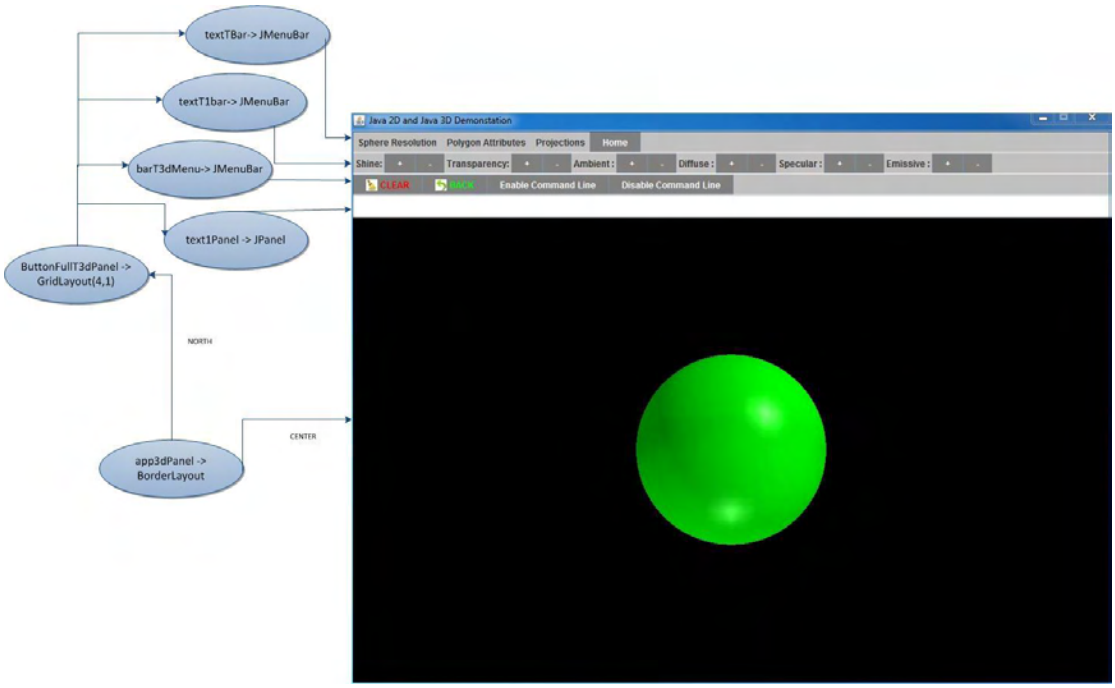
ΕΙΚΟΝΑ 129 ΚΟΥΜΠΙΑ ΓΙΑ ΕΠΑΝΑΦΟΡΑ ΚΑΙ ΕΠΙΣΤΡΟΦΗ

12. Java 3D – 8η Επιλογη (Appearance)

12.1 Παρουσιαση

Η όγδοη επιλογή της εφαρμογής είναι το αντικείμενο της κλάσης App3dPanel. Είναι ένα panel που περιέχει Swing components όπως κουμπιά και ετικέτες. Περιέχει επίσης ένα αντικείμενο της κλάσης text1Panel που είναι ένα JPanel που αποτελεί την γραμμή εντολών. Ακόμη κύριο συστατικό αυτής της επιλογής είναι ο εικονικός κόσμος που έχει δημιουργηθεί ο οποίος περιέχει μία σφαίρα. Σκοπός της επιλογής αυτής είναι ο χρήστης να μπορεί να αλλάξει τα γνωρίσματα που συνθέτουν την εμφάνιση της επιφανείας της σφαίρας. Μπορεί επίσης να αλλάξει τον τύπο προβολής και έχει την δυνατότητα να περιηγηθεί μέσα στον εικονικό κόσμο. Τέλος παρέχονται οι λειτουργίες επιστροφή στα εξώφυλλο και επαναφοράς του εικονικού κόσμου στην αρχική του κατάσταση.

Το App3dPanel έχει custom δομή που δημιουργείται με border Layout και grid Layout.



Εικονα 130 app3dPanel layout

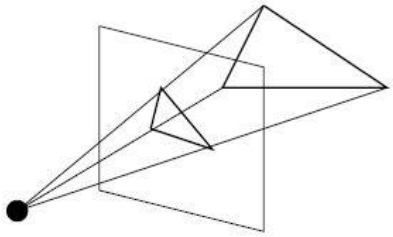
12.2 Λειτουργικοτητα

12.2.1 Προβολες

Οι γεωμετρικοί μετασχηματισμοί μέχρι τώρα χρησιμοποιήθηκαν για να τοποθετήσουν αντικείμενα μέσα στον εικονικό κόσμο. Για την αναπαράσταση μίας τρισδιάστατης σκηνής σε μία επίπεδη οθόνη υπολογιστή απαιτείται προβολή στο δισδιάστατο επίπεδο. Τέτοιες προβολές μπορούν να περιγραφούν με γεωμετρικούς μετασχηματισμούς. Για την αναπαράσταση μίας τρισδιάστατης σκηνής πρέπει να οριστούν η θέση του παρατηρητή και το επίπεδο προβολής. Ο παρατηρητής κοιτάζει στην κατεύθυνση του επιπέδου προβολής που μπορεί ναι ερμηνευτεί ως ένα παράθυρο πίσω από το οποίο βρίσκεται ο εικονικός κόσμος.

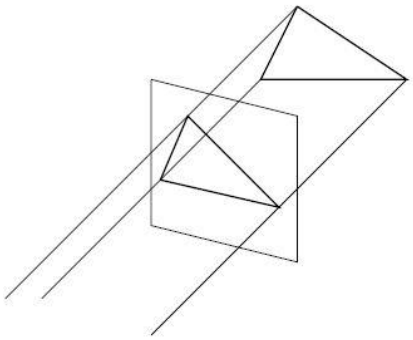
Η προβολή ενός αντικειμένου πάνω στο επίπεδο προβολής βρίσκεται ενώνοντας τα σημεία

που βρίσκεται το αντικείμενο με το κέντρο της προβολής και υπολογίζοντας τα σημεία τομής των γραμμών που ενώνουν το αντικείμενο με το επίπεδο προβολής. Η μέθοδος αυτή ονομάζεται προοπτική προβολή (perspective projection).



ΕΙΚΟΝΑ 131 ΠΡΟΟΠΤΙΚΗ ΠΡΟΒΟΛΗ

Όταν το κέντρο της προβολής μετακινείται όλο και πιο μακριά από το επίπεδο προβολής και εντέλει μετακινείται στο άπειρο οι γραμμές που ενώνουν το αντικείμενο με το επίπεδο προβολής γίνονται παράλληλες. Σε αυτή την περίπτωση δεν χρειάζεται να οριστεί το κέντρο της προβολής παρά μόνο η κατεύθυνση της προβολής. Για την παράλληλη προβολή οι γραμμές είναι παράλληλες ως προς την κατεύθυνση της προβολής. Συνήθως θεωρείται ότι η κατεύθυνση της προβολής είναι κάθετη στο επίπεδο προβολής.



ΕΙΚΟΝΑ 132 ΠΑΡΑΛΛΗΛΗ ΠΡΟΒΟΛΗ

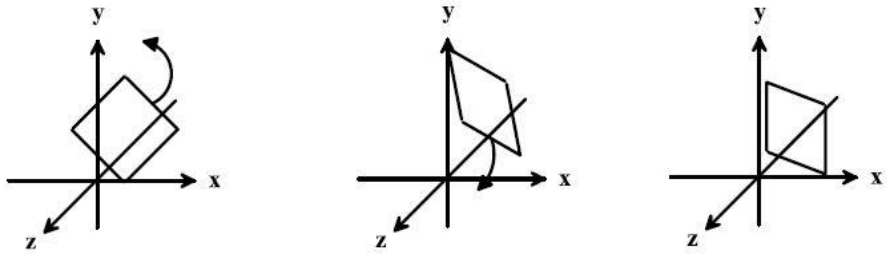
ΠΑΡΑΛΛΗΛΗ ΠΡΟΒΟΛΗ

Έστω η ειδική περίπτωση παράλληλης προβολής με επίπεδο προβολής το z = z0 (επίπεδο παράλληλο στο επίπεδο x/y). Αυτή η προβολή προβάλει το σημείο (x,y,z) στο σημείο (x,y,z0) με τον μετασχηματισμό που δίνεται από τον πίνακα:

$$\begin{pmatrix} x \\ y \\ z0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & z0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Βάση αυτής της αναπαράστασης οποιαδήποτε παράλληλη προβολή μπορεί να περιγραφεί με πολλαπλασιασμό πινάκων σε ομογενείς συντεταγμένες. Αν το επίπεδο προβολής δεν είναι παράλληλο στο επίπεδο που ορίζουν οι x/y άξονες, πριν από την εφαρμογή του παραπάνω πίνακα πρέπει να εφαρμοστεί κάποιου άλλου είδους μετασχηματισμός που θα προβάλει το επίπεδο

προβολής πάνω στο επίπεδο x/y. Αυτό μπορεί να επιτευχθεί με μία περιστροφή γύρω από τον άξονα των y και μία περιστροφή γύρω από τον άξονα των x.

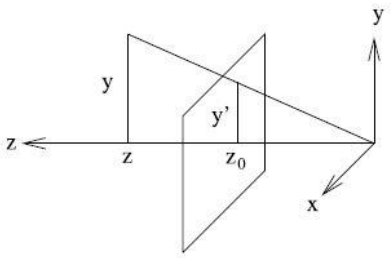


ΕΙΚΟΝΑ 133 ΠΡΟΒΟΛΗ ΕΠΙΠΕΔΟΥ ΠΡΟΒΟΛΗΣ ΠΑΝΩ ΣΤΟ ΕΠΙΠΕΔΟ ΠΟΥ ΟΡΙΖΟΥΝ ΟΙ ΑΞΟΝΕΣ Χ/Υ

Εφόσον με τους κατάλληλους μετασχηματισμούς όλα τα επίπεδα προβολής μπορούν να γίνουν παράλληλα στο επίπεδο x/y μπορούμε να χρησιμοποιούμε τον παραπάνω πίνακα για την εφαρμογή της παράλληλης προβολής. Ακόμη ο παραπάνω πίνακας μπορεί να δημιουργήσει παράλληλη προβολή για όλα τα επίπεδα προβολής πάνω στον άξονα των z εφόσον τους εφαρμοστεί ένας μετασχηματισμό translation έτσι ώστε να συμπέσουν με το επίπεδο προβολής z= z0.

ΠΡΟΟΠΤΙΚΗ ΠΡΟΒΟΛΗ

Οι προοπτικές προβολές μπορούν επίσης να εκφραστούν ως πολλαπλασιασμός πινάκων. Έστω η ειδική περίπτωση προοπτικής προβολής όπου το κέντρο της προβολής είναι το κέντρο του συστήματος συντεταγμένων και το επίπεδο προβολής είναι το επίπεδο z=z0 παράλληλο στο επίπεδο x/y.



ΕΙΚΟΝΑ 134 ΠΡΟΟΠΤΙΚΗ ΠΡΟΒΟΛΗ ΜΕ ΕΠΙΠΕΔΟ ΠΡΟΒΟΛΗΣ ΤΟ Z=Z0

Από την εικόνα 134 και σύμφωνα με τις αναλογίες μπορούμε να γράψουμε :

$\frac{x'}{x} = \frac{z0}{z}$ και $\frac{y'}{y} = \frac{z0}{z}$ οπότε και $x' = \frac{z0}{z} \cdot x$ και $y' = \frac{z0}{z} \cdot y$. Η προοπτική προβολή προβάλει το σημείο (x,y,z) στο σημείο (x',y',z0)=$(\frac{z0}{z} \cdot x, \frac{z0}{z} \cdot y, z0)$. Ο πίνακας προβολής μπορεί να γραφεί

$$\begin{pmatrix} x \\ y \\ z \\ \frac{z}{z0} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{z0} & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Αν διαιρέσουμε το σημείο (x,y,z, $\frac{z}{z0}$) με $\frac{z}{z0}$ τότε μας εμφανίζεται το σημείο (x',y',z0)=$(\frac{z0}{z} \cdot x, \frac{z0}{z} \cdot y, z0)$ σε καρτεσιανές συντεταγμένες. Οποιαδήποτε προοπτική προβολή μπορεί εντέλει να αναπαρασταθεί από την προοπτική προβολή με κέντρο προβολής το κέντρο του συστήματος συντεταγμένων και με επίπεδο προβολής παράλληλο στο x/y επίπεδο. Το κέντρο της προβολής

μπορεί να συμπέσει με το κέντρο του συστήματος συντεταγμένων αν εφαρμόσουμε σε αυτό ένα μετασχηματισμό translation. Το επίπεδο προβολής μπορεί να συμπέσει με ένα επίπεδο παράλληλο στο x/y επίπεδο με τους ίδιους μετασχηματισμούς που εφαρμόστηκαν και στην παράλληλη προβολή (περιστροφή γύρω από τον άξονα των y ακολουθούμενη από περιστροφή γύρω από τον άξονα των x).

Μία άλλη ειδική περίπτωση προοπτικής προβολής είναι η προβολή όπου αντί το κέντρο προβολής να είναι το κέντρο του συστήματος συντεταγμένων το κέντρο έχει μετατοπιστεί πάνω στον άξονα των z βάση του μετασχηματισμού translation (0,0,-z0) έτσι ώστε το επίπεδο προβολής να γίνει το x/y επίπεδο. Βάση των αναλογιών όπως στην εικόνα 134 υπολογίζοντας την προοπτική προβολή του σημείου (x,y,z) στο σημείο (x',y',0) παίρνουμε τις σχέσεις : $x' = \frac{z0}{z0+z} \cdot x = \frac{x}{1+\frac{z}{z0}}$ και $y' = \frac{z0}{z0+z} \cdot y = \frac{y}{1+\frac{z}{z0}}$. Η αντιστοίχιση αυτή μπορεί να γραφεί σε μορφή πίνακα σε ομογενείς συντεταγμένες :

$$\begin{pmatrix} x \\ y \\ 0 \\ 1+\frac{z}{z0} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{z0} & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Το σημείο (x,y,0,1+$\frac{z}{z0}$) σε ομογενείς συντεταγμένες αντιστοιχεί στο προβαλλόμενο σημείο σε καρτεσιανές συντεταγμένες ($\frac{x}{1+\frac{z}{z0}}, \frac{y}{1+\frac{z}{z0}}$, 0). Όπως αναφέρθηκε όλες οι προοπτικές προβολές μπορούν να αναπαρασταθούν από την προοπτική προβολή με κέντρο προβολής το κέντρο του συστήματος συντεταγμένων όμως επίσης η προβολή με κέντρο προβολής το κέντρο του συστήματος συντεταγμένων μπορεί να αναπαρασταθεί από την προοπτική προβολή με κέντρο πάνω στον z άξονα με την εφαρμογή ενός μετασχηματισμού translation. Έτσι μπορούμε να μελετήσουμε μόνο την περίπτωση όπου το κέντρο προβολής βρίσκεται πάνω στον άξονα των z εφαρμόζοντας τους απαραίτητους μετασχηματισμούς στον εικονικό κόσμο πριν από την προβολή. Ο πίνακας της προβολής μπορεί να αποσυντεθεί σε ένα γινόμενο δύο πινάκων.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{z0} & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{z0} & 1 \end{pmatrix}$$

Ο αριστερός πίνακας στο δεξί μέλος της εξίσωσης αντιστοιχεί στον πίνακα της παράλληλης προβολής με z=0, δηλαδή παράλληλη προβολή στο επίπεδο x/y. Δείξαμε ότι κάθε προοπτική προβολή $A_{perspective}$ μπορεί να αναπαρασταθεί από μία προοπτική προβολή $A_{perspective,z0}$ εφαρμόζοντας πρώτα τον κατάλληλο μετασχηματισμό.

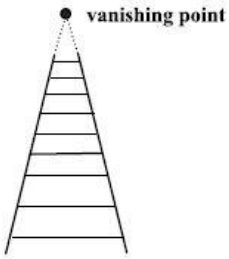
$$A_{perspective} = A_{perspective,z0} \cdot T$$

Βάση της αποσύνθεσης του πίνακα της προοπτικής προβολής $A_{perspective,z0}$ έχουμε :

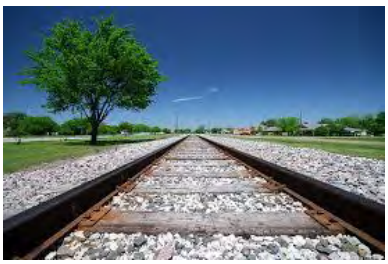
$$A_{perspective} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \tilde{T} \text{ όπου } \tilde{T} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{z0} & 1 \end{pmatrix} \cdot T$$

Αυτό σημαίνει ότι οποιαδήποτε προοπτική προβολή μπορεί να θεωρηθεί σαν το γινόμενο του κατάλληλου μετασχηματισμού $\tilde{T}$ με την παράλληλη προβολή στο x/y επίπεδο. Και οι παράλληλες προβολές μπορούν να αναπαρασταθούν με αυτή την ειδική παράλληλη προβολή. Έτσι γενικά οι προβολές μπορούν να αντιμετωπιστούν εφαρμόζοντας μετασχηματισμούς στον εικονικό κόσμο και έπειτα εφαρμόζοντας μία παράλληλη προβολή στο x/y επίπεδο.

Η παράλληλη προβολή στο x/y επίπεδο δίνει την τιμή 0 στην συντεταγμένη z. Σύμφωνα με τον πίνακα μετασχηματισμού $\tilde{T}$ το x/y επίπεδο δηλαδή όλα τα σημεία με z=0 δεν αλλάζουν εφόσον το επίπεδο x/y είναι το επίπεδο προβολής. Αν θεωρήσουμε ένα σημείο της μορφής (0,0,$\frac{1}{w}$) με w ∈ R με w≠0 μπορεί να γραφεί σε ομογενείς συντεταγμένες ως (0,0,1,w) . Ο πίνακας του μετασχηματισμού $\tilde{T}$ προβάλει αυτό το σημείο στο σημείο (0,0,1,$\frac{1}{z0} + w$) σε ομογενείς συντεταγμένες. Σε καρτεσιανές συντεταγμένες : $\left(0,0,\frac{1}{w}\right) \rightarrow \left(0,0,\frac{z0}{1+z0\cdot w}\right)$. Αν η παράμετρος w τείνει στο 0 το σημείο (0,0,$\frac{1}{w}$) κινείται πάνω στον z άξονα προς το άπειρο όπου η εικόνα του (0,0,$\frac{z0}{1+z0\cdot w}$) συγκλίνει στο σημείο (0,0,z0). Αυτό σημαίνει ότι το υποθετικό σημείο στο άπειρο πάνω στον z άξονα προβάλλεται σε έναν πεπερασμένο σημείο. Αν υποθέσουμε ότι όλες οι γραμμές περνάνε από το $\left(0,0,\frac{1}{w}\right)$ οι εικόνες αυτών των γραμμών τις οποίες παίρνουμε από τον πίνακα $\tilde{T}$ συναντώνται στο σημείο $\left(0,0,\frac{z0}{1+z0\cdot w}\right)$. Αν το w τείνει στο 0 οι γραμμές που περνούν από το σημείο $\left(0,0,\frac{1}{w}\right)$ γίνονται παράλληλες ως προς τον άξονα των z. Ο πίνακας προβάλει αυτές τις παράλληλες γραμμές να συναντώνται σε ένα υποθετικό σημείο στο άπειρο πάνω στον άξονα των z μέσω του σημείου $(0,0,z0)$. Αυτό το σημείο ονομάζεται vanishing point.

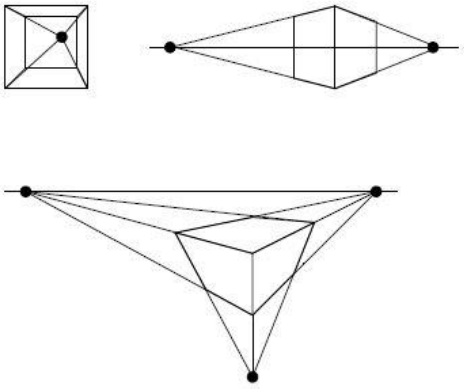


ΕΙΚΟΝΑ 135 VANISHING POINT



ΕΙΚΟΝΑ 136 ΓΡΑΜΜΕΣ ΤΡΑΙΝΟΥ - VANISHING POINT

Η παραπάνω θεωρία αποδεικνύει το φαινόμενο ότι οι παράλληλες γραμμές οι οποίες απομακρύνονται από τον παρατηρητή δεν φαίνονται παράλληλες σε προοπτική προβολή. Οι γραμμές αυτές συναντώνται στο vanishing point. Οι οριζόντιες και οι κάθετες γραμμές μένουν ως έχουν στην προοπτική προβολή. Επίσης αν επιλεγεί κάποιο άλλο επίπεδο προβολής για την ίδια προοπτική προβολή τότε μπορεί να υπάρχουν 2 ή και 3 vanishing points. Ο αριθμός των vanishing points είναι ίσος με τον αριθμό των αξόνων του συστήματος συντεταγμένων που το επίπεδο προβολής τέμνει.



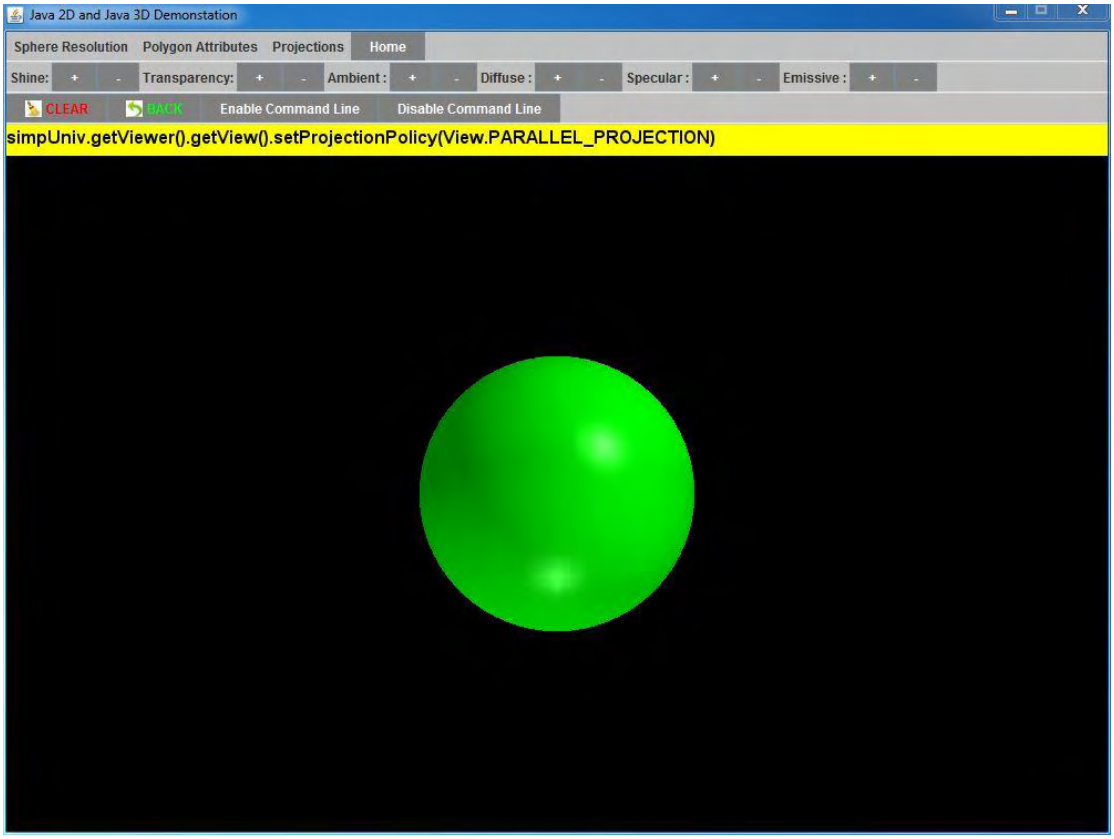
ΕΙΚΟΝΑ 137 ONE, TWO, THREE POINT PERSPECTIVE PROJECTIONS

Στην εφαρμογή εμφανίζεται μία σφαίρα πάνω στην οποία εφαρμόζονται τα δύο είδη προβολών που αναφέρθηκαν. Η προεπιλογή για την προβολή είναι η προοπτική προβολή, για να αλλάξουμε την προβολή χρησιμοποιούμε την εντολή :

simpUniv2.getViewer().getView().setProjectionPolicy(View.PARALLEL_PROJECTION)

Η μεταβλητή simpleUniv2 αναφέρεται στο αντικείμενο της κλάσης Simple Universe που έχει δημιουργηθεί για τον εικονικό κόσμο της δεύτερης επιλογής της εφαρμογής. Μαζί με την δημιουργία του αντικείμενου Simple Universe δημιουργούνται και όλα τα απαραίτητα αντικείμενα για την διακλάδωση της προβολής του scene graph δηλαδή τα αντικείμενο Viewing Platform και Viewer. Με την μέθοδο get Viewer() επιστρέφεται το αντικείμενο Viewer που είναι σχετικό με το scene graph και το αντικείμενο Simple Universe. Για το αντικείμενο Viewer επιστρέφεται το αντικείμενο View. Για το αντικείμενο View καλούμε τη μέθοδο set Projection Policy(). Με την τελευταία μέθοδο καθορίζεται ποιο μοντέλο προβολής θα χρησιμοποιηθεί βάση της παραμέτρου View.PARALLEL_PROJECTION για παράλληλη προβολή ή View.PERSPECTIVE_PROJECTION για προοπτική προβολή.

Στην εφαρμογή για να μεταβούμε σε διαφορετική προβολή από την προεπιλεγμένη (προοπτική) επιλέγουμε από το μενού «Projections» την επιλογή «Parallel», έπειτα αν θέλουμε να μεταβούμε πάλι σε προοπτική προβολή επιλέγουμε την επιλογή «Perspective».



ΕΙΚΟΝΑ 138 PARALLEL ΚΑΙ PERSPECTIVE PROJECTION

12.2.2 ORBIT BEHAVIOR ΚΑΙ ΕΠΙΣΤΡΟΦΗ ΣΤΗΝ ΑΡΧΙΚΗ ΘΕΣΗ

Στον προσδιορισμό των παραμέτρων της προβολής έχει χρησιμοποιηθεί σε κάθε επιλογή της εφαρμογής η παρακάτω εντολή :

simpUniv.getViewingPlatform().setNominalViewingTransform()

Για το εκάστοτε αντικείμενο simpUniv της κλάσης Simple Universe καλείται η μέθοδος get Viewing Platform() όπου επιστρέφεται το αντικείμενο Viewing Platform που είναι σχετικό με το scene graph και το αντικείμενο Simple Universe. Για το αντικείμενο Viewing Platform καλείται η μέθοδος set Nominal Viewing Transform() με την οποία ορίζεται η αρχική θέση του παρατηρητή τέτοια ώστε να μπορεί να δει το εύρος από -1 έως 1 στον x και y άξονα στο x/y επίπεδο. Αυτή είναι η προεπιλεγμένη θέση για τον παρατηρητή. Σε περίπτωση που χρειαστεί να αλλάξει η θέση του παρατηρητή εφαρμόζουμε έναν μετασχηματισμό έστω vt αντικείμενο της κλάσης Transform3D , στο αντικείμενο Viewing Platform του Simple Universe γράφοντας :

simpUniv.getViewingPlatform().getViewPlatformTransform().setTrnasform(vt)

Στην επιλογή αυτή έχει χρησιμοποιηθεί διαδραστική πλοήγηση μέσα στην σκηνή ελεγχόμενη από το ποντίκι με την βοήθεια της κλάσης Orbit Behavior. Η κλάση αυτή επιτρέπει πατώντας το αριστερό κουμπί του ποντικιού να περιστρέφεται η σκηνή, πατώντας το δεξί κουμπί του ποντικιού ο χρήστης να μπορεί να κινηθεί μέσα στην σκηνή και να κάνει zoom με την ροδέλα του ποντικιού. Ο κώδικας για την orbit behavior είναι :

OrbitBehavior ob = new OrbitBehavior(myCanvas3D ,OrbitBehavior.STOP_ZOOM)

ob2.setMinRadius(WIDTH)

ob.setSchedulingBounds(newBoundingSphere(new Point3d(0.0,0.0,0.0),Double.MAX_VALUE))

simpUniv.getViewingPlatform().setViewPlatformBehavior(ob)

Με την πρώτη εντολή δημιουργείται έναν αντικείμενο της κλάσης Orbit behavior το οποίο εφαρμόζεται στο αντικείμενο Canvas3D. Με την παράμετρο OrbitBehavior.STOP_ZOOM ορίζεται ότι η διαδικασία του zoom θα σταματήσει όταν φτάσει την ελάχιστη τροχιά που ορίζεται από την μέθοδο στην δεύτερη εντολή set Min Radius() και είναι ίση με το πλάτος του JPanel. Έπειτα ορίζεται η περιοχή στην οποία έχει επίδραση αυτή η συμπεριφορά. Η περιοχή αυτή έχει σχήμα σφαίρας. Στην τέταρτη εντολή με την μέθοδο get Viewing Platform() επιστρέφεται το αντικείμενο Viewing Platform για το αντικείμενο Simple Universe. Με την μέθοδο set View Platform Behavior() ορίζεται για το αντικείμενο Viewing Platform ποια συμπεριφορά θα ακολουθήσει δηλαδή ποιοί μετασχηματισμοί θα του εφαρμοστούν σύμφωνα με την μεταβλητή που παίρνει ως παράμετρο, στην συγκεκριμένη περίπτωση το αντικείμενο της κλάσης orbit behavior.

Εφόσον χρησιμοποιείται η orbit behavior για πλοήγηση μέσα στον εικονικό κόσμο η σφαίρα που έχει δημιουργηθεί αλλάζει θέση (λόγω της αλλαγής της προβολής) ανάλογα με τον χρήστη. Η προβολή πολλές φορές μπορεί να μην είναι ιδανική για να δούμε ολόκληρη την σφαίρα ή και τις αλλαγές που γίνονται πάνω σε αυτή. Για τον λόγο αυτό υπάρχει η ανάγκη η σφαίρα να μπορεί να τοποθετείται στην αρχική της θέση επαναφέροντας την αρχική προβολή. Έτσι αποθηκεύουμε σε μία μεταβλητή τύπου Transform3D τον μετασχηματισμό που μεταφέρει την προβολή στην αρχική της θέση :

home = new Transform3D()

simpUniv2.getViewingPlatform().getViewPlatformTransform().getTransform(home)

αμέσως μετά τις εντολές για την orbit behavior. Η μέθοδος get Transform() αποθηκεύει τον μετασχηματισμό για την Viewing Platform για τη συγκεκριμένη συμπεριφορά στην μεταβλητή Transform3D home. Για επαναφέρουμε την προβολή πρσοθέτουμε στο κατάλληλο σημείο π.χ. στο κουμπί «Home» τον κώδικα :

ob.setHomeTransform(home)

ob.goHome()

Με την πρώτη εντολή ορίζεται ως Home Transform δηλαδή ένα γνωστό σημείο ενδιαφέροντος για την επανατοποθέτηση και προσανατολισμό της Viewing Platform ο μετασχηματισμός home. Με την δεύτερη εντολή εφαρμόζεται στην συμπεριφορά για το Viewing Platform που είναι το αντικείμενο ob της κλάσης orbit behavior ο μετασχηματισμός home.
Στην εφαρμογή για να τοποθετηθεί η σφαίρα στην αρχική της θέση (να επαναφέρουμε την προβολή) πατάμε το κουμπί «Home».

12.2.3 Normal Vectors για Επιφανειες

Θέματα φωτισμού, σκίασης όπως επίσης και ανάκλασης φωτός είναι απαραίτητα για την δημιουργία ρεαλιστικών τρισδιάστατων γραφικών η/υ. Οι αντανακλάσεις του φωτός εξαρτώνται από

την γωνία του φωτός ως προς την επιφάνεια. Normal vectors για την επιφάνεια χρειάζονται για τον υπολογισμό αυτών των γωνιών.

Ένα τρίγωνο πάντα ορίζει ένα επίπεδο και όλα τα normal vectors ενός τέτοιου επίπεδου τριγώνου δείχνουν προς την ίδια κατεύθυνση. Αν το επίπεδο που ορίζει το τρίγωνο δίνεται από την εξίσωση :

$$Ax + By + Cz + D = 0$$

Τότε το διάνυσμα $(\mathrm{A}, \mathrm{B}, \mathrm{C})^T$ είναι ένα μη κανονικοποιημένο normal vector στο επίπεδο. Αυτό είναι αληθές για τον ακόλουθο λόγο. Αν n=$(\mathrm{nx}, \mathrm{ny}, \mathrm{nz})^T$ δεν είναι απαραίτητα ένα κανονικοποιημένο normal vector στο επίπεδο και v=$(\mathrm{vx}, \mathrm{vy}, \mathrm{vz})^T$ είναι ένα σημείο στο επίπεδο, τότε το σημείο $(x, y, z)^T$ βρίσκεται επίσης στο επίπεδο αν και μόνο αν το διάνυσμα που ενώνει τα v και $(x, y, z)^T$ βρίσκεται στο επίπεδο. Αυτό σημαίνει ότι το διάνυσμα που ενώνει τα v και $(x, y, z)^T$ πρέπει να είναι κάθετα στο normal vector.

$$0 = n^T \cdot ((x, y, z)^T - v) = nx \cdot x + ny \cdot y + nz \cdot z - n^T \cdot v$$

Επιλέγοντας A=nx, B=ny, C=nz, D=$\mathrm{n}^{\mathrm{T}} \cdot$ v παίρνουμε την εξίσωση για ο επίπεδο που ορίζει το τρίγωνο. Όταν ένα τρίγωνο δίνεται από τρία μη συγγραμμικά σημεία P1, P2, P3 το normal vector μπορεί να υπολογιστεί από το εξωτερικό γινόμενο :

$$n = (P2 - P1) \times (P3 - P1)$$

Το εξωτερικό γινόμενο δύο διανυσμάτων $(x1, y1, z1)^T$ και $(x2, y2, z2)^T$ ορίζεται ως το διάνυσμα :

$$\begin{pmatrix} x1 \\ y1 \\ z1 \end{pmatrix} \times \begin{pmatrix} x2 \\ y2 \\ z2 \end{pmatrix} = \begin{pmatrix} y1 \cdot z2 - y2 \cdot z1 \\ z1 \cdot x2 - z2 \cdot x1 \\ x1 \cdot y2 - x1 \cdot y1 \end{pmatrix}$$

Το εξωτερικό γινόμενο είναι 0 όταν τα δύο διανύσματα είναι συγγραμμικά.
Η εξίσωση του επιπέδου που ορίζει το τρίγωνο παρέχει ένα μη κανονικοποιημένο normal vector στο επίπεδο. Η τιμή D υπολογίζεται με την εισαγωγή ενός από τα σημεία που ορίζουν το τρίγωνο για παράδειγμα το σημείο P1:

$$D = n^T \cdot P1$$

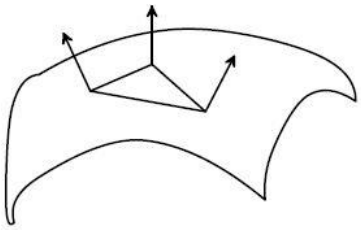
Το normal vector στο σημείο x(s0,t0) της επιφάνειας ελευθέρου σχήματος είναι το normal vector στο εφαπτόμενο επίπεδο στο συγκεκριμένο σημείο. Το εφαπτόμενο επίπεδο καθορίζεται από τα εφαπτόμενα διανύσματα στο x(s0,t0) με τις δύο παραμετρικές καμπύλες p(s)=x(s,t0) και q(t)=x(s0,t).

$$\left(\frac{\partial}{\partial s} x(s, t0)\right)_{s=s0} = \left(\frac{\partial}{\partial s} \sum_{i=0}^{n} \sum_{j=0}^{m} b_{ij} \cdot B_i^{(n)}(s) \cdot B_j^{(m)}(t0)\right)_{s=s0}$$
$$= \sum_{j=0}^{m} B_j^{(m)}(t0) \cdot \sum_{i=0}^{n} b_{ij} \cdot \left(\frac{\partial B_i^{(n)}(s)}{\partial s}\right)_{s=s0}$$

$$\left(\frac{\partial}{\partial t} x(s0, t)\right)_{t=t0} = \left(\frac{\partial}{\partial t} \sum_{i=0}^{n} \sum_{j=0}^{m} b_{ij} \cdot B_i^{(n)}(s0) \cdot B_j^{(m)}(t)\right)_{t=t0}$$
$$= \sum_{i=0}^{n} B_i^{(n)}(s0) \cdot \sum_{j=0}^{m} b_{ij} \cdot \left(\frac{\partial B_j^{(m)}(t)}{\partial t}\right)_{t=t0}$$

Αυτά τα διανύσματα εφαπτόμενων είναι παράλληλα στην επιφάνεια στο σημείο (s0,t0) και φέρουν το εφαπτόμενο επίπεδο σε αυτό το σημείο. Το εξωτερικό γινόμενο αυτών των εφαπτόμενων διανυσμάτων είναι το normal vector της επιφάνειας στο σημείο x(s0,t0).

Όταν μία επιφάνεια ελευθέρου σχήματος προσεγγίζεται από τρίγωνα τα normal vectors των τριγώνων δεν μπορούν να παραχθούν από τα τρίγωνα αλλά από την επιφάνεια κατευθείαν. Φυσικά είναι αδύνατο να αποθηκευτεί ένα normal vector για κάθε σημείο των τριγώνων, αλλά θα πρέπει τουλάχιστον να υπολογίζονται και να αποθηκεύονται τα normal vectors των τριών κορυφών των τριγώνων που προσεγγίζουν την επιφάνεια. Με αυτό τον τρόπο ένα επίπεδο που ορίζει ένα τρίγωνο μπορεί να έχει τρία διαφορετικά normal vectors που κληρονομούνται από την αρχική επιφάνεια.



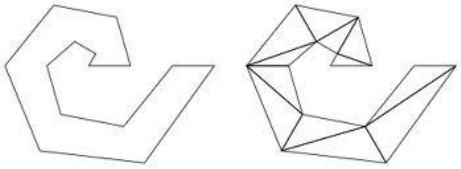
ΕΙΚΟΝΑ 139 NORMAL VECTORS ΤΩΝ ΚΟΡΥΦΩΝ ΤΟΥ ΤΡΙΓΩΝΟΥ ΠΟΥ ΠΡΟΣΕΓΓΙΖΕΙ ΤΗΝ ΑΡΧΙΚΗ ΕΠΙΦΑΝΕΙΑ ΕΛΕΥΘΕΡΟΥ ΣΧΗΜΑΤΟΣ

Τα normal vectors για τα βασικά γεωμετρικά αντικείμενο όπως κύβος, σφαίρα, κύλινδρος και κώνος ορίζονται αυτόματα από την Java 3D. Για αντικείμενα που έχουν φορτωθεί από κάποιο αρχείο , για παράδειγμα μορφή αρχείου Wavefront, τα normal vectors συνήθως περιέχονται στο αρχείο μαζί με τις συντεταγμένες του αντικειμένου. Όταν τα αντικείμενα μοντελοποιούνται κατευθείαν από τρίγωνα στην Java 3D, τα normal vectors μπορούν να οριστούν ρητά. Αυτό θα χρειαστεί σπάνια εφόσον πολύπλοκα αντικείμενα δεν σχεδιάζονται απευθείας σε Java 3D. Κατασκευάζονται με ένα κατάλληλο εργαλείο σχεδίασης και φορτώνονται στην Java 3D με αρχεία τύπου Wavefront.

12.2.4 Ψηφιοποιηση και Polygon Attributes

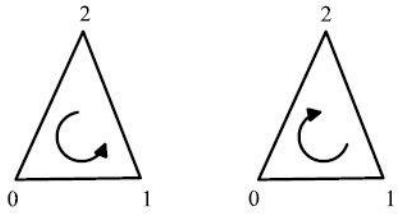
Μέχρι αυτό το σημείο της εφαρμογής έχουμε αναφερθεί σε βασικά γεωμετρικά σχήματα όμως στα γραφικά η/υ υπάρχει η ανάγκη για την δημιουργία πολύ πιο πολύπλοκων σχημάτων. Για την μοντελοποίηση τέτοιων σχημάτων αναφερόμαστε συχνά στις επιφάνειές τους και όχι στα σημεία τα οποία αποτελούν τον όγκο τους. Βέβαια υπάρχουν και περιπτώσεις όπως οι τρισδιάστατοι σαρωτές που πρώτα περιγράφουν το τρισδιάστατο σύνολο των σημείων των σχημάτων και μετά τις επιφάνειές τους.

Οι επιφάνειες των σχημάτων συχνά προσεγγίζονται από έναν μεγάλο αριθμό από πολύγωνα, τρίγωνα στις περισσότερες περιπτώσεις, με σκοπό να απλοποιηθούν οι υπολογισμοί για τον φωτισμό και τις προβολές. Για αυθαίρετες επιφάνειες μπορεί να είναι αδύνατο να βρεθεί μία αναλυτική έκφραση της αναπαράστασης της προβολής. Αποτελεσματικοί και γρήγοροι υπολογισμοί προβολών είναι αδύνατοι. Με τα πολύγωνα όμως ο υπολογισμός των προβολών είναι πολύ πιο εύκολος γιατί η εύρεση του σημείου τομής ενός πολυγώνου με μία γραμμή που αναπαριστά μία από τις γραμμές με τις οποίες υπολογίζονται οι προβολές είναι πολύ πιο απλή και γρήγορη στον υπολογισμό. Η προσέγγιση μίας καμπύλης επιφάνειας από πολύγωνα λέγεται ψηφιοποίηση (tessellation). Χρησιμοποιώντας μόνο τρίγωνα για τα πολύγωνα δεν αποτελεί περιορισμό εφόσον οποιοδήποτε πολύγωνο μπορεί να διασπαστεί σε τρίγωνα.



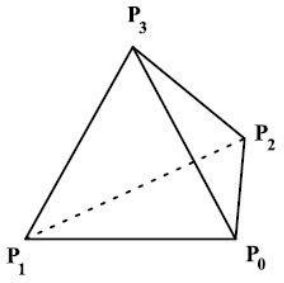
Εικονα 140 τριγωνοποιηση ενος πολυγωνου

Τα τρίγωνα έχουν το πλεονέκτημα ότι υπάρχουν πολύ γρήγοροι αλγόριθμοι για αυτά που μπορούν να υλοποιηθούν στις κάρτες γραφικών. Τα τρίγωνα ή τα πολύγωνα που χρησιμοποιούνται για την μοντελοποίηση επιφανειών συνήθως προσανατολίζονται έτσι ώστε να μπορεί να καθορίζεται ποια πλευρά του πολυγώνου είναι από την εξωτερική πλευρά της επιφάνειας. Ο προσανατολισμός καθορίζεται από την σειρά των κορυφών του πολυγώνου. Οι κορυφές αριθμούνται αντίθετα με τη φορά του ρολογιού όταν κοιτάμε την επιφάνεια του αντικειμένου από την μπροστινή του πλευρά.



Εικονα 141 προσανατολισμενα τριγωνα

Στην εικόνα 141 το τρίγωνο με τις κορυφές 0,1,2 είναι προσανατολισμένο στην κατεύθυνση του παρατηρητή, δηλαδή ο παρατηρητής μπορεί να δει το μπροστινό μέρος της επιφάνειας. Το ίδιο τρίγωνο με κορυφές 0,2,1 παραμένει αόρατο για τον παρατηρητή γιατί μπορεί να δει αυτό το μέρος της επιφάνειας μόνο από το πίσω μέρος. Όταν τα πολύγωνα είναι προσανατολισμένα η απόδοσή τους μπορεί να επιταχυνθεί σημαντικά εφόσον εκείνες οι επιφάνειες των οποίων οι κορυφές δεν βρίσκονται στην κατεύθυνση του παρατηρητή από το μπροστινό μέρος μπορούν να αγνοηθούν. Στο τετράεδρο της εικόνα 142 οι τέσσερις πλευρές του μπορούν να οριστούν σε μορφή τριγώνων ως : $\mathbf{P}_0$-$\mathbf{P}_3$- $\mathbf{P}_1$,$\mathbf{P}_0$- $\mathbf{P}_2$-$\mathbf{P}_3$,$\mathbf{P}_0$- $\mathbf{P}_1$- $\mathbf{P}_2$,$\mathbf{P}_1$- $\mathbf{P}_3$- $\mathbf{P}_2$. Για τον ορισμό των τριγώνων οι κορυφές κάθε τριγώνου έχουν καταγραφεί αντίθετα από την φορά του ρολογιού όταν βλέπεις κάθε τρίγωνο από την εξωτερική πλευρά του τετράεδρου.



Εικονα 142 τετραεδρο με κορυφες P0, P1, P2,P3

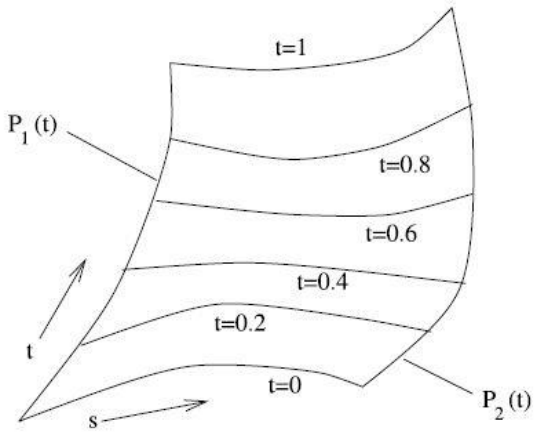
Πιο γενικά η ψηφιοποίηση είναι ίσως η σημαντικότερη τεχνική για την μοντελοποίηση και στηρίζεται στις επιφάνειες ελεύθερου σχήματος που ορίζονται από παραμετρικές καμπύλες. Για την περιγραφή μίας επιφάνειας ενός αντικειμένου με πολύγωνα χρειάζεται μία λίστα με σημεία δηλαδή τις κορυφές των πολυγώνων και μία λίστα με τα πολύγωνα που δημιουργούν αυτά τα σημεία. Εκτός από αυτή τη γεωμετρική δομή χρειάζονται πληροφορίες για το χρώμα ή την υφή της επιφάνειας όπως επίσης και

τα normal vectors τα οποία έχουν ανατεθεί στα πολύγωνα ή στις κορυφές των πολυγώνων που είναι απαραίτητα για τον υπολογισμό του φωτισμού και της σκιάς .Έτσι εκτός από τις λίστες με τις κορυφές ή και τα πολύγωνα που αποτελούν τις επιφάνειες πρέπει να αποθηκεύονται και τα normal vectors.

Στην ψηφιοποίηση όσο μεγαλύτερος ο αριθμός των τριγώνων τόσο καλύτερα προσεγγίζεται η επιφάνεια. Η εικόνα 143 δείχνει μία σφαίρα με διαφορετικό αριθμό από τρίγωνα. Η πρώτη σφαίρα έχει ψηφιοποιηθεί μόνο από 8 τρίγωνα. Το υπολογιστικό κόστος αυξάνεται με τον αριθμό των τριγώνων. Η προσέγγιση της επιφάνειας με τρίγωνα μπορεί να υπολογιστεί πριν την απόδοση του σχήματος. Συνήθως μεγαλύτερη ψηφιοποίηση οδηγεί σε τετραγωνική αύξηση του υπολογιστικού κόστους. Για παράδειγμα ο διπλασιασμός της ψηφιοποίησης κάθε δισδιάστατης επιφάνειας απαιτεί τέσσερις φορές περισσότερα τρίγωνα.

ΕΙΚΟΝΑ 143 ΣΦΑΙΡΑ ΜΕ ΔΙΑΦΟΡΕΤΙΚΗ ΨΗΦΙΟΠΟΙΗΣΗ

Οι επιφάνειες ελεύθερου σχήματος είναι στενά συνδεδεμένες με τις παραμετρικές καμπύλες. Οι επιφάνειες ελεύθερου σχήματος έχουν δύο παραμέτρους για να περιγράψουν την δισδιάστατη επιφάνεια ενώ μόνο μία παράμετρος η t χρειάζεται για τις καμπύλες. Όταν η μία παράμετρος της επιφάνειας θεωρείται σταθερή, η διαφοροποίηση της άλλη παραμέτρου αποδίδει μία καμπύλη πάνω στην επιφάνεια όπως στην εικόνα 144.

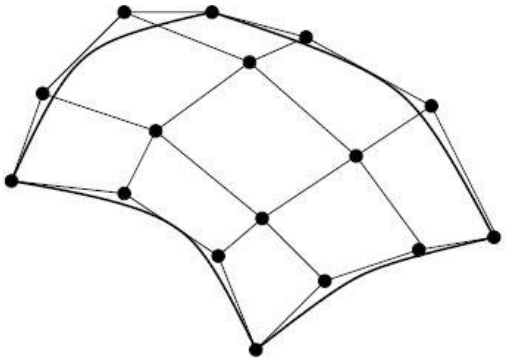


ΕΙΚΟΝΑ 144 ΕΠΙΦΑΝΕΙΑ ΕΛΕΥΘΕΡΗΣ ΜΟΡΦΗΣ

Οι επιφάνειες Bezier αποτελούνται από καμπύλες Bezier με παραμέτρους s και t. Οι καμπύλες Bezier μελετήθηκαν στην παράγραφο 6.2.4 [6.2.4].

$$x(s,t) = \sum_{i=0}^{n}\sum_{j=0}^{m} b_{ij} \cdot B_i^{(n)}(s) \cdot B_j^{(m)}(t) \qquad (s,t \in [0,1])$$

Πιο συνηθισμένες είναι οι Bezier καμπύλες βαθμού 3, δηλαδή m=n=3. Για τον ορισμό μίας Bezier επιφάνειας πρέπει να οριστούν (m+1)·(n+1) Bezier σημεία b_{ij} για παράδειγμα 16 στην περίπτωση της κυβικής επιφάνειας Bezier. Η εικόνα 145 δείχνει πως ένα δίχτυ από σημεία Bezier ορίζει μία επιφάνεια Bezier.



Εικονα 145 δικτυ απο Bezier σημεια για τον ορισμο μιας Bezier επιφανειας

Οι επιφάνειες Bezier έχουν παρόμοιες ιδιότητες με αυτές των καμπυλών Bezier. Τα τέσσερα σημεία στις κορυφές $b_{00}, b_{0m}, b_{n0}, b_{nm}$ βρίσκονται πάνω στην ίδια επιφάνεια. Γενικά αυτό δεν συμβαίνει και με τα άλλα σημεία ελέγχου. Η επιφάνεια μένει εντός του convex hull των σημείων ελέγχου. Καμπύλες με σταθερή τιμή s=s0 είναι Bezier καμπύλες με σεβασμό στα σημεία :

$$b_j = \sum_{i=0}^{n} b_{ij} \cdot B_j^{(n)}(s0)$$

Αναλογικά ισχύει για καμπύλες με σταθερή τιμή t=t0.

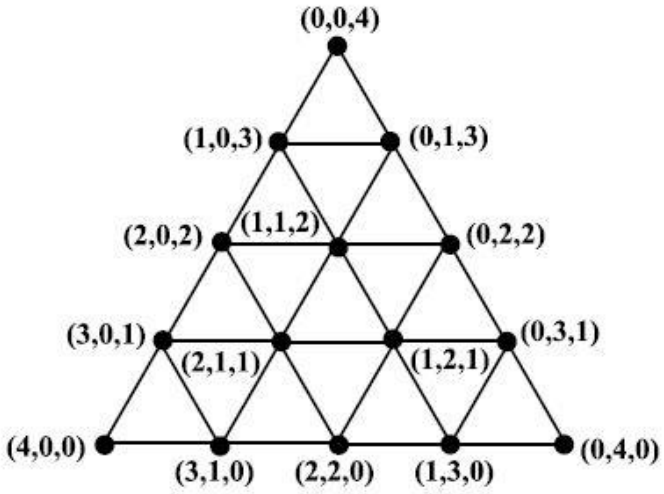
Εφόσον η ψηφιοποίηση όπως απαιτείται στα γραφικά η/υ προσεγγίζει τις επιφάνειες με τρίγωνα και όχι με τετράγωνα οι Bezier καμπύλες βαθμού n=3 μερικές φορές ορίζονται πάνω σε δίχτυ από τρίγωνα με τον ακόλουθο τρόπο :

$$x(t1, t2, t3) = \sum_{i,j,k \geq 0 : i+j+k=n} b_{ijk} \cdot B_{ijk}^{(n)}(t1, t2, t3)$$

Τα αντίστοιχα Bernstein πολυώνυμα [6.2.4] δίνονται από τον τύπο :

$$B_{ijk}^{(n)}(t1, t2, t3) = \frac{n!}{i!\, j!\, k!} \cdot t_1^i \cdot t_2^j \cdot t_3^k$$

Όπου t1+t2+t3=1, t1,t2,t3≥0 και i+j+k=n για I,j,k ∈ N.



Εικονα 146 δικτυ τριγωνοποιησης

Στην εφαρμογή η σφαίρα δημιουργείται με 5 διαφορετικές ψηφιοποιήσεις. Η ψηφιοποίηση όπως αναφέρθηκε διαφέρει ανάλογα με τον αριθμό των τριγώνων που χρησιμοποιούνται. Η ψηφιοποίηση καθορίζεται κατά την δημιουργία του αντικειμένου και δεν μπορεί να αλλάξει μετά την διαδικασία της απόδοσης για αυτό και έχουν δημιουργηθεί 5 διαφορετικές σφαίρες με 5 διαφορετικές ψηφιοποίησεις στις οποίες μεταβαίνουμε ανάλογα με την ψηφιοποίηση που επιθυμούμε. Για την δημιουργία της σφαίρας γράφουμε :

Sphere sphere = new Sphere(sphereR,Sphere.GENERATE_NORMALS,division ,greenApp)

Όπου sphereR είναι η ακτίνα, Sphere.GENERATE_NORMALS είναι το primitive flag για να δημιουργηθούν τα normal vectors που είναι απαραίτητα για την αλλαγή των παραμέτρων του υλικού (material) που θα μελετηθεί σε επόμενη παράγραφο [12.2.9]. Division είναι ο αριθμός των τριγώνων για την ψηφιοποίηση και greenApp είναι το αντικείμενο Appearance που χρησιμοποιήθηκε για το πράσινο χρώμα της σφαίρας. Η μεταβλητή division παίρνει τις τιμές : 4, 5, 12, 31 και 68. Η προεπιλεγμένη τιμή για την μεταβλητή division είναι 16. Όσο μεγαλύτερη η τιμή της μεταβλητής division τόσο καλύτερη προσέγγιση έχει το σχήμα μας.
Επίσης είναι δυνατόν να ορίσουμε τις ιδιότητες για την απόδοση των πολυγώνων που χρησιμοποιούνται στην ψηφιοποίηση με την κλάση Polygon Attributes. Στην εφαρμογή χρησιμοποιούνται μόνο τρίγωνα τα οποία μπορούν να σχεδιαστούν ως σημεία, με περιγράμματα ή γεμισμένα με χρώμα.

PolygonAttributes pa=new PolygonAttributes()

pa.setCapability(PolygonAttributes.ALLOW_MODE_WRITE)

greenApp.setPolygonAttributes(pa)

Στην πρώτη εντολή δημιουργείται ένα αντικείμενο της κλάσης Polygon Attributes. Για αυτό το αντικείμενο ορίζεται η δυνατότητα να επιτρέπεται η αλλαγή των παραμέτρων Polygon Attributes αφού έχει γίνει live το συγκεκριμένο αντικείμενο στο οποίο θα χρησιμοποιηθεί. Στην τρίτη εντολή ορίζουμε ότι κάθε σχήμα που θα ζωγραφιστεί με αυτήν την εμφάνιση (greenApp) θα μπορεί να εμφανίσει τις ιδιότητες που ορίζει η κλάση Polygon Attributes. Για να εμφανίσουμε τα περιγράμματα των τριγώνων γράφουμε:

pa.setPolygonMode(PolygonAttributes.POLYGON_LINE)

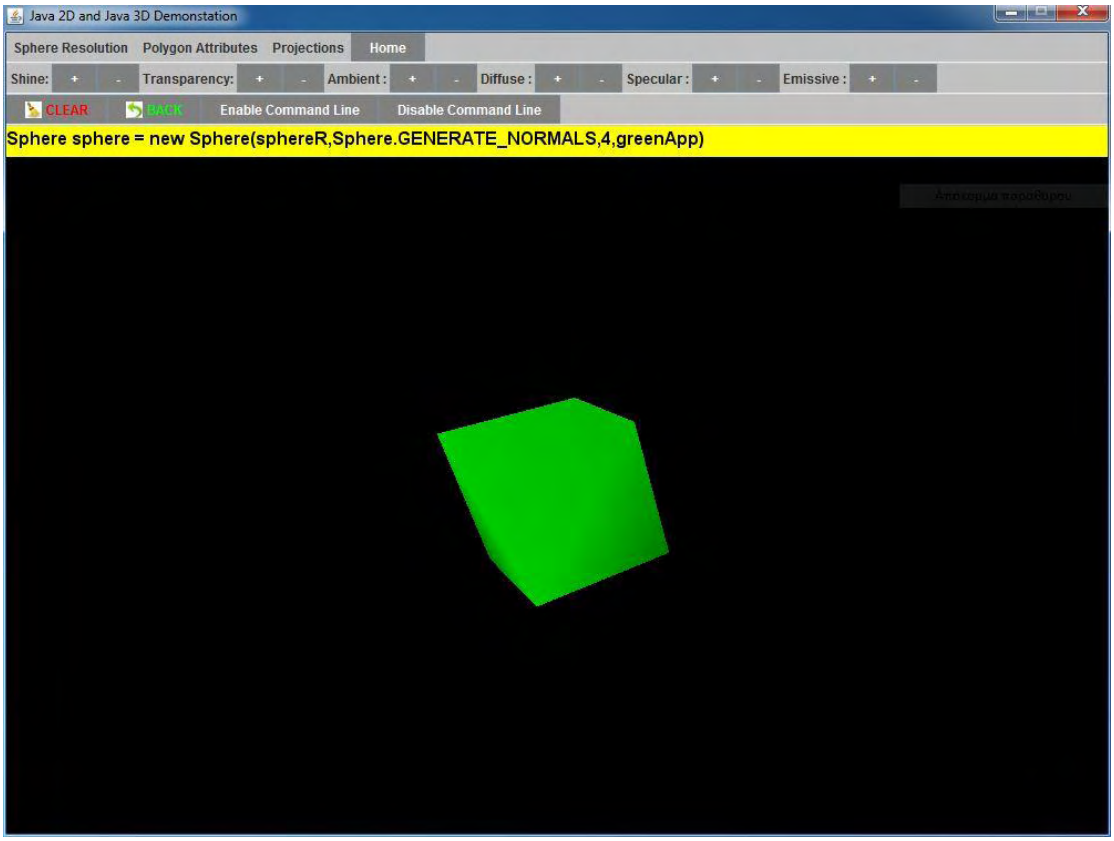
για να εμφανίσουμε τα τρίγωνα ως σημεία :

pa.setPolygonMode(PolygonAttributes.POLYGON_POINT)

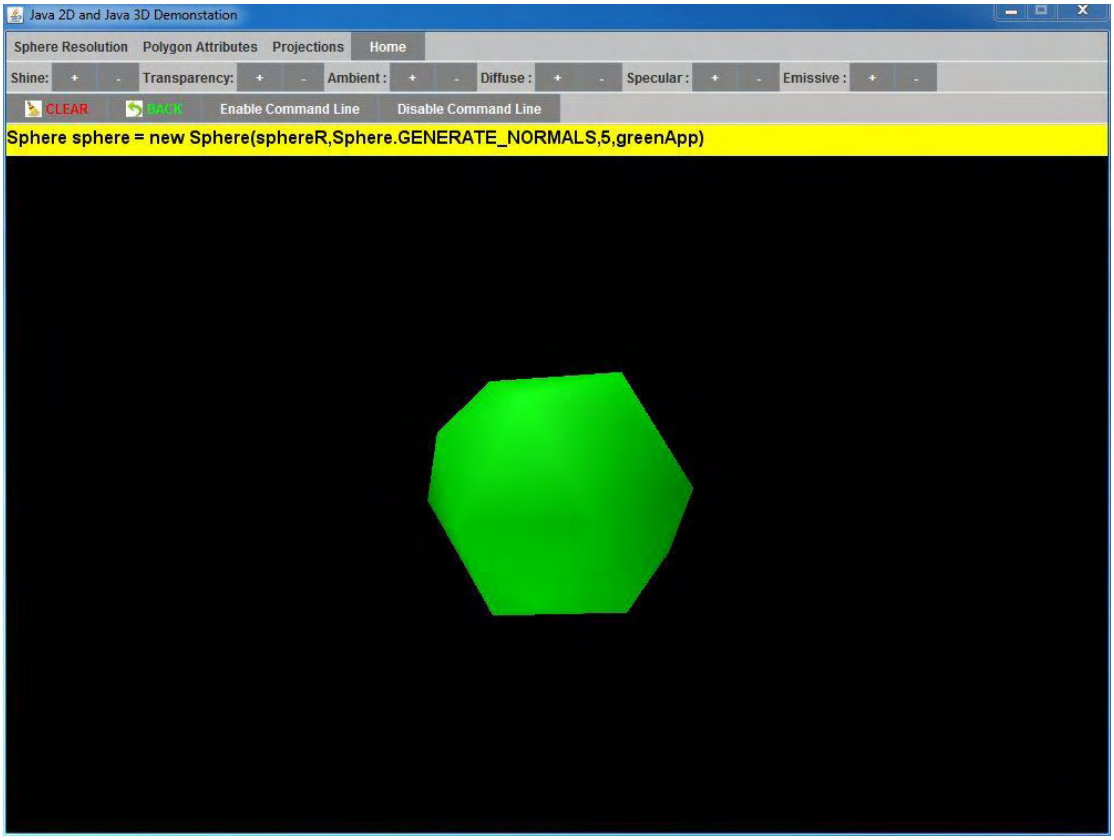
και για να εμφανίσουμε τα τρίγωνα γεμισμένα με χρώμα:

pa.setPolygonMode(PolygonAttributes.POLYGON_FILL)

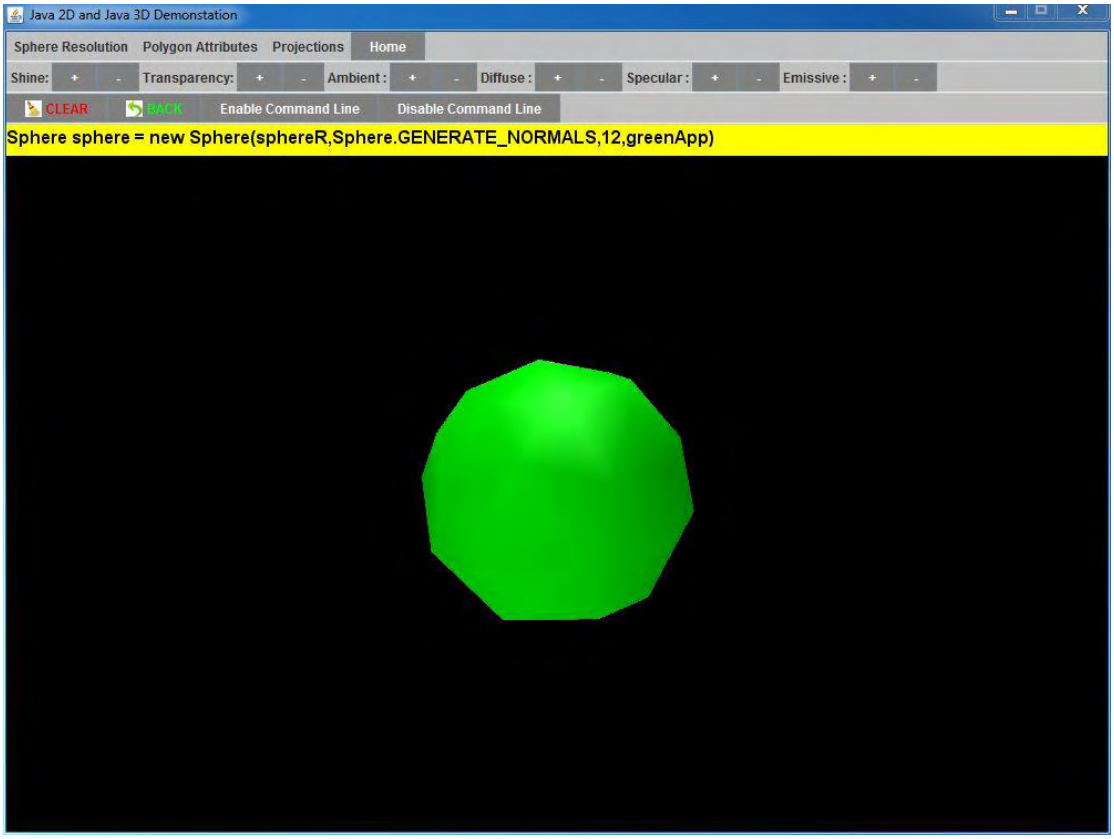
Στην εφαρμογή στο μενού «Sphere Tesselation» μπορούμε να αλλάξουμε την ψηφιοποίηση της σφαίρας. Στο μενού «Polygon Attributes» ανάλογα με την επιλογή μας εμφανίζονται τα τρίγωνα για την ψηφιοποίηση της σφαίρας είτε με περίγραμμα είτε γεμισμένα με χρώμα είτε ως τελείες.



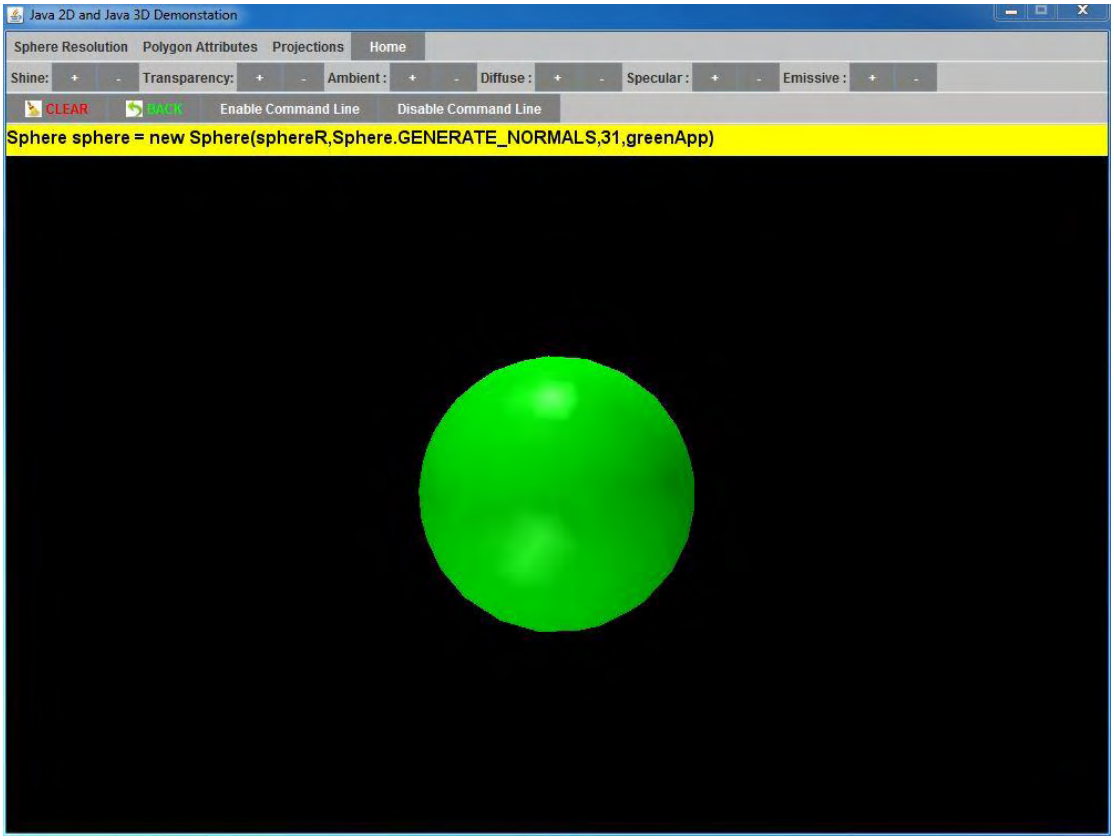
ΕΙΚΟΝΑ 147 RESS 0 - 4 DIVISIONS



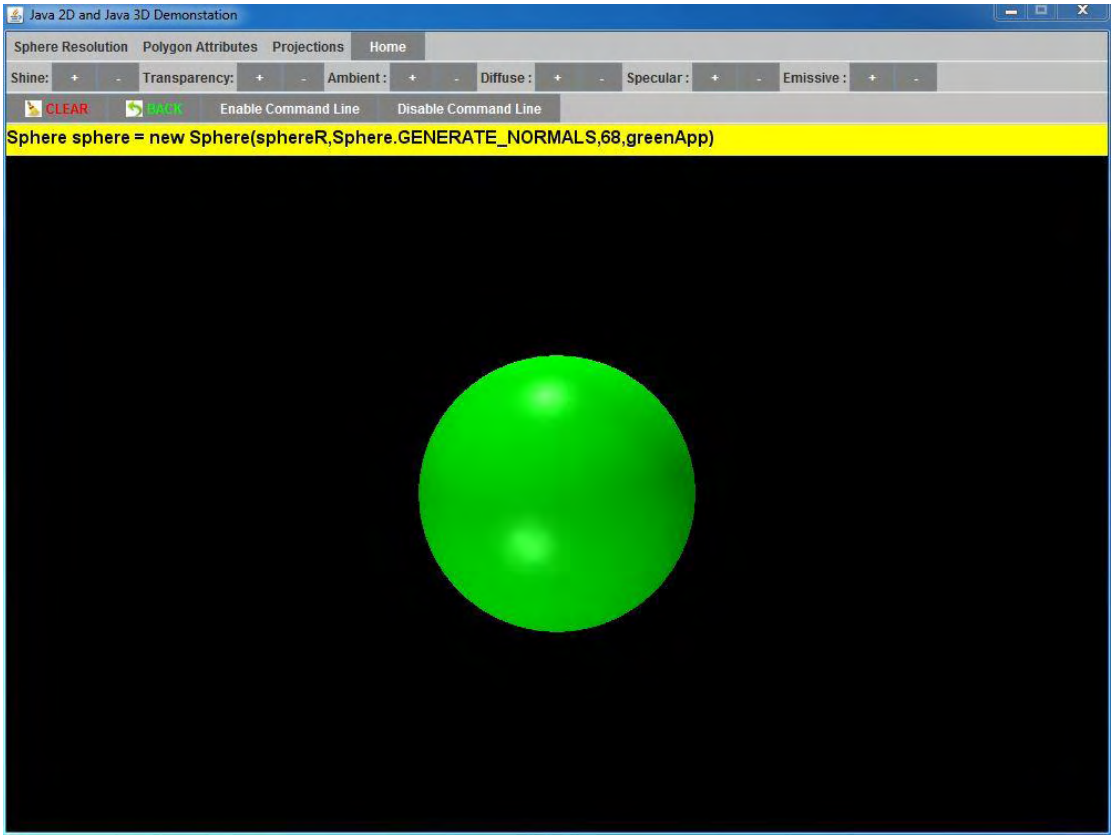
ΕΙΚΟΝΑ 148 RESS 1 - 5 DIVISIONS



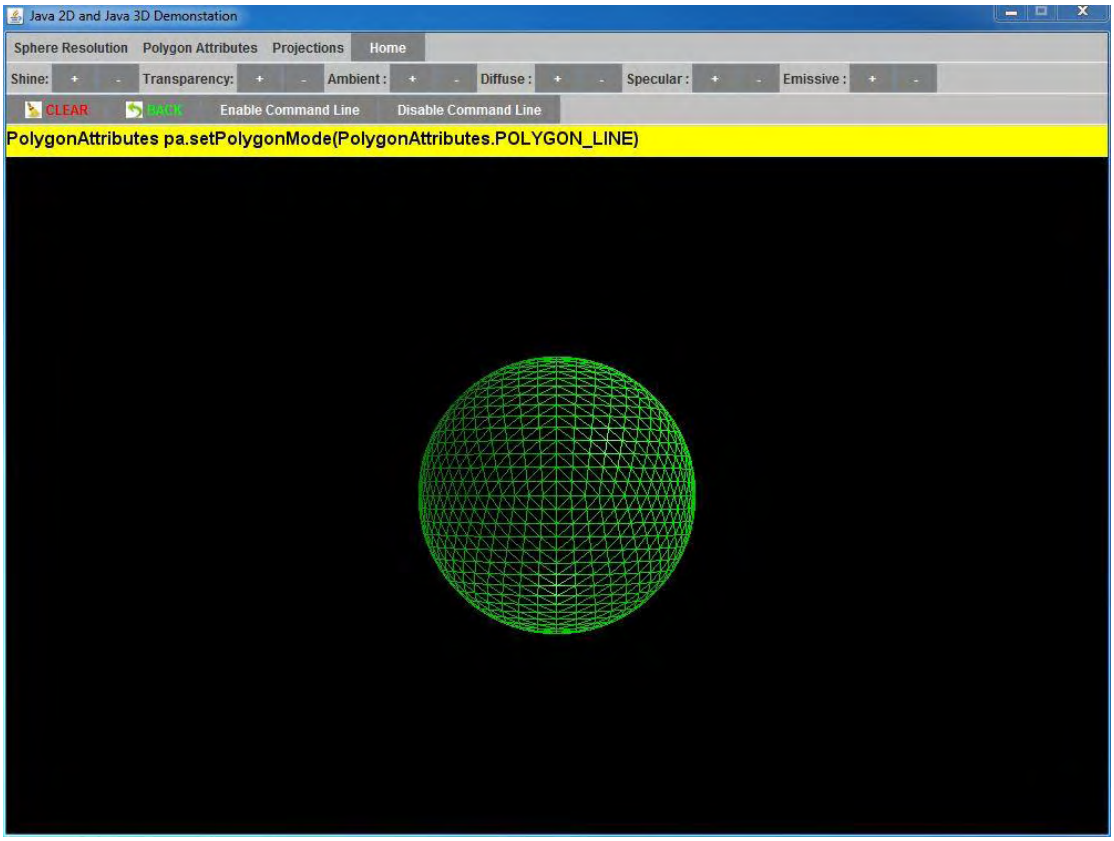
ΕΙΚΟΝΑ 149 RESS 2 - 12 DIVISIONS



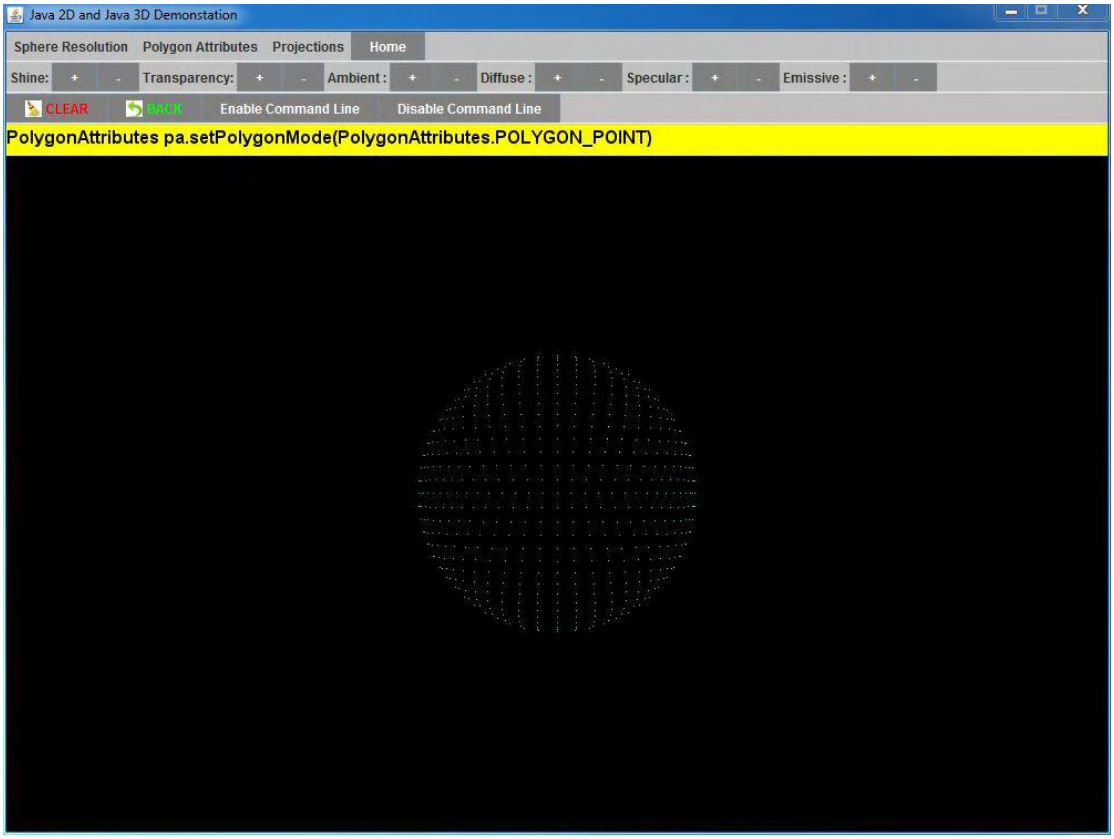
ΕΙΚΟΝΑ 150 RESS 3 - 31 DIVISIONS



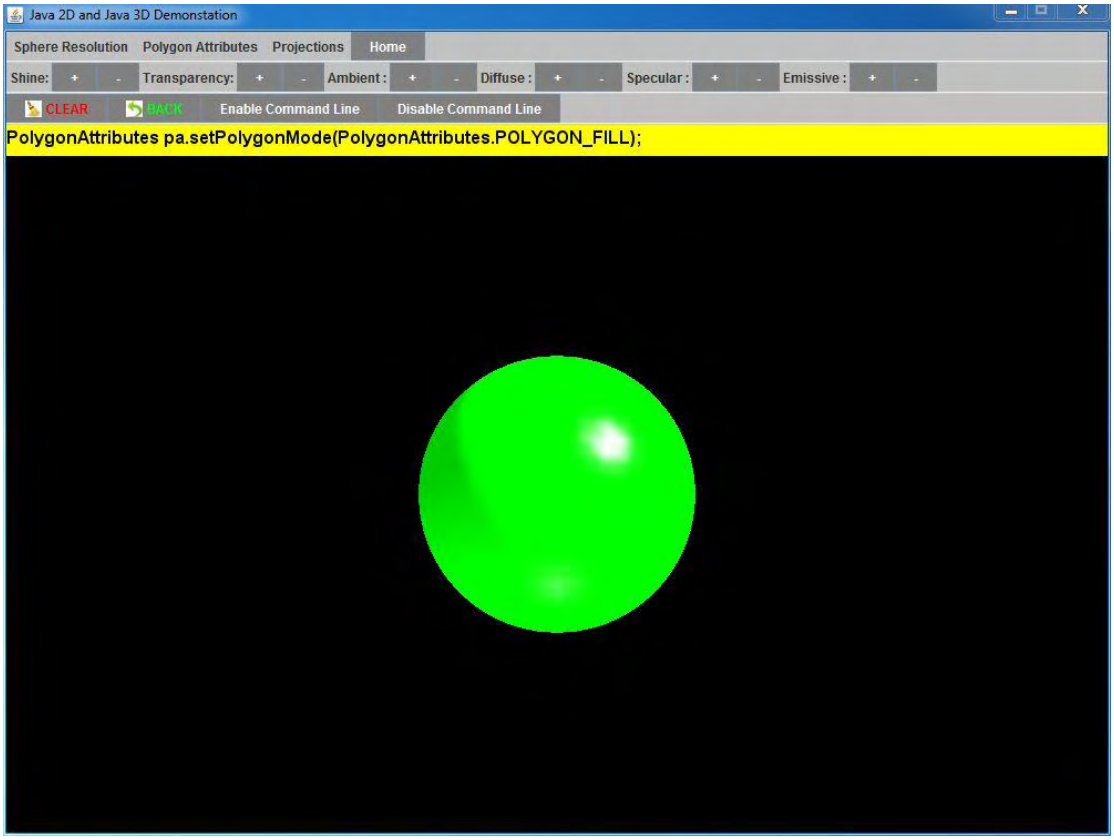
ΕΙΚΟΝΑ 151 RESS 4 – 68 DIVISIONS - Η ΜΕΓΙΣΤΗ ΨΗΦΙΟΠΟΙΗΣΗ ΓΙΑ ΤΗΝ ΣΦΑΙΡΑ (ΠΡΟΕΠΙΛΕΓΜΕΝΗ ΨΗΦΙΟΠΟΙΗΣΗ)



ΕΙΚΟΝΑ 152 POLYGON ATTRIBUTES - LINE ΓΙΑ RESS 4



ΕΙΚΟΝΑ 153 POLYGON ATTRIBUTES – POINT ΓΙΑ RESS 4



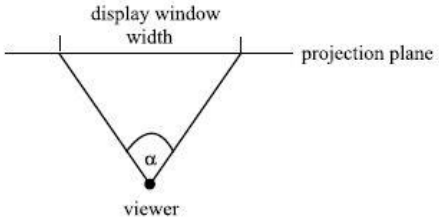
ΕΙΚΟΝΑ 154 POLYGON ATTRIBUTES – FILL ΓΙΑ RESS 4 (ΠΡΟΕΠΙΛΟΓΗ)

12.2.5 Ψαλιδισμα του Όγκου

Για να προβάλουμε μία τρισδιάστατη σκηνή πρέπει να οριστούν ποια είναι τα ορατά αντικείμενα. Αρχικά θα εφαρμοστεί ψαλίδισμα (clipping) στον εικονικό κόσμο δηλαδή θα αφαιρεθούν όλα εκείνα τα αντικείμενα τα οποία δεν μπορεί να δει ο παρατηρητής και έπειτα θα αποδοθούν στην οθόνη (rendering) μόνο τα απαραίτητα αντικείμενα μέσα στον ήδη ψαλιδισμένο όγκο.

Πριν προβάλουμε μία σκηνή πρέπει πρώτα να οριστεί ένας αριθμός από παραμέτρους. Οι συντεταγμένες του σημείου όπου βρίσκεται ο παρατηρητής όπως επίσης και η κατεύθυνση που κοιτά. Πρέπει να οριστεί το επίπεδο προβολής. Το επίπεδο προβολής αντιστοιχεί συνήθως στην οθόνη του η/υ. Η οθόνη του η/υ ή οποιοδήποτε μέσο προβολής μπορεί να προβάλει μόνο έναν τομέα του επιπέδου προβολής που είναι συνήθως ορθογώνιου σχήματος. Αντί να οριστεί αυτός ο ορθογώνιος τομέας ορίζεται η γωνία προβολής. Η γωνία καθορίζει το πεδίο όρασης του παρατηρητή. Η γωνία ορίζει πόσο μακριά το πεδίο όρασης εκτίνεται στα αριστερά και στα δεξιά του παρατηρητή. Έτσι ορίζεται το πλάτος του ψαλιδισμένου ορθογωνίου πάνω στο επίπεδο προβολής. Αντιστοιχεί επίσης στο πλάτος του παραθύρου της οθόνης του η/υ που προβάλλεται η σκηνή. Το ύψος του ορθογωνίου μπορεί να επιλεχθεί αναλογικά με το ύψος του παραθύρου της οθόνης του η/υ.

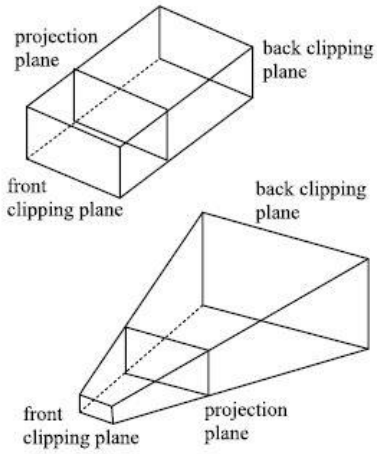
Η ψαλιδισμένη περιοχή (ο ψαλιδισμένος όγκος) αντιστοιχεί σε μία πυραμίδα άπειρου ύψους για την προοπτική προβολή ή σε ένα κουτί με άπειρες προεκτάσεις για την παράλληλη προβολή.



Εικονα 155 η γωνια α καθοριζει το ευρος του επιπεδου προβολης που αντιστοιχει στο πλατος του παραθυρου προβολης

Ένας μεγάλος αριθμός από αντικείμενα μπορούν να βρίσκονται μέσα στον ψαλιδισμένο όγκο. Η απόσταση που μπορεί να δει ένα άνθρωπος είναι σχεδόν χωρίς όριο. Κάποιος μπορεί να δει τα αστέρια στον ουρανό που είναι έτη φωτός μακριά όπως επίσης και ένα δάχτυλο που έχει τοποθετηθεί κοντά και μπροστά από τα μάτια του. Όμως κανείς δεν είναι ικανός να δει τον ουρανό με τα αστέρια και το δάχτυλο την ίδια στιγμή. Τα μάτια προσαρμόζονται σε μία συγκεκριμένη απόσταση. Μόνο αντικείμενα που είναι κοντά σε αυτή την απόσταση μπορούν βρίσκονται μέσα στην περιοχή εστίασης. Αντικείμενα πολύ μακρύτερα ή πολύ πιο κοντά από αυτή την απόσταση δεν είναι μέσα στην περιοχή εστίασης. Το μάτι εστιάζει σε μία καθορισμένη απόσταση και υπάρχει ένα εύρος γύρω από αυτή την απόσταση όπου τα αντικείμενα που βρίσκονται εκεί μπορούν να είναι ορατά γιατί το μάτι έχει την ικανότητα να εστιάσει μέσα σε αυτό το εύρος. Το φαινόμενο αυτό στα γραφικά η/υ μοντελοποιείται με τις clipping planes την μπροστά και την πίσω. Η μπροστά clipping plane προσδιορίζει την μικρότερη απόσταση μέσα στην οποία τα αντικείμενα μπορούν να είναι ορατά. Η πίσω clipping plane προσδιορίζει την μέγιστη απόσταση μέσα στην οποία τα αντικείμενα μπορούν να είναι ορατά. Για την προοπτική προβολή ο ψαλιδισμένος όγκος έχει το σχήμα μίας κόλουρου πυραμίδας. Για την παράλληλη προβολή ο ψαλιδισμένος όγκος έχει το σχήμα ενός κουτιού. Το επίπεδο προβολής βρίσκεται μεταξύ της μπροστά και της πίσω clipping plane και αντιστοιχεί στην

απόσταση με την βέλτιστη εστίαση. Τα αντικείμενα στην μπροστά clipping plane βρίσκονται μπροστά από την οθόνη προβολής και μπορούν να είναι ορατά μόνο με στερεοσκοπική προβολή.



ΕΙΚΟΝΑ 156 CLIPPING PLANES

Έχουμε αναφέρει ότι κάθε προβολή μπορεί να αναλυθεί σε έναν μετασχηματισμό T ακολουθούμενο από μία παράλληλη προβολή στο x/y επίπεδο. Το τρισδιάστατο ψαλίδισμα μπορεί να επιτευχθεί εφαρμόζοντας πρώτα τον μετασχηματισμό T σε όλα τα αντικείμενα και έπειτα υπολογίζοντας την παράλληλη προβολή πάνω στο x/y επίπεδο. Με αυτόν τον τρόπο ακόμα και ο ψαλιδισμένος όγκος για τον προοπτική προβολή από πυραμίδα θα μετασχηματιστεί σε κουτί όπως στην παράλληλη προβολή. Οι πλευρές του κουτιού είναι παράλληλες στους άξονες του συστήματος συντεταγμένων . Το κουτί μπορεί να οριστεί από τις δύο κορυφές του στην διαγώνιό του την (xmin, ymin, zmin) και την (xmax, ymax, zmax). Για να ελέγξουμε αν ένα αντικείμενα βρίσκεται μέσα στον ψαλιδισμένο όγκο πρέπει να βρούμε αν έστω ένα σημείο του (px, py, pz) βρίσκεται μέσα στο κουτί. Αυτό ικανοποιείται αν και μόνο αν :

$$xmin \leq px \leq xmax \ , \qquad ymin \leq py \leq pmax \ , \qquad zmin \leq pz \leq zmax$$

12.2.6 ΚΑΘΟΡΙΣΜΟΣ ΟΡΑΤΩΝ ΕΠΙΦΑΝΕΙΩΝ

Ο ψαλιδισμός είναι υπεύθυνος για το ποια αντικείμενα βρίσκονται έστω και εν μέρει μέσα στον ψαλιδισμένο όγκο. Μερικά από αυτά δεν θα είναι ορατά γιατί αντικείμενα μακριά από τον παρατηρητή μπορεί να βρίσκονται πίσω από αντικείμενα που βρίσκονται πιο κοντά στον παρατηρητή. Εφόσον κάθε προβολή μπορεί να αναλυθεί από έναν μετασχηματισμό T ακολουθούμενο από μία παράλληλη προβολή στο x/y επίπεδο όλη η παράγραφος θα αναφέρεται σε αυτήν την συγκεκριμένη περίπτωση προβολής.

Αλγόριθμος ακρίβειας εικόνας και ακρίβειας αντικειμένου

Ένας απλός αλγόριθμος για τον καθορισμό των ορατών αντικειμένων σε μία σκηνή μπορεί να βασίζεται στην αρχή ότι το ορθογώνιο πάνω στο επίπεδο προβολής που αντιστοιχεί στο παράθυρο προβολής έχει το ίδιο πλέγμα από pixel με το παράθυρο προβολής. Έπειτα μία ακτίνα ρίπτεται σε κάθε pixel στην κατεύθυνση της προβολής για παράδειγμα παράλληλα στον άξονα των z. Το χρώμα του pixel δίνεται από το αντικείμενο το οποίο χτυπά πρώτο η ακτίνα. Η τεχνική αυτή

αναφέρεται ως αλγόριθμος ακρίβειας εικόνας εφόσον βασίζεται στο πλέγμα από pixel της εικόνας που πρέπει να υπολογιστεί. Ο αλγόριθμος αυτός έχει πολυπλοκότητα n·p για μία εικόνα με p pixel και n αντικείμενα. Για μία τυπική ανάλυση οθόνης η/υ το p θα είναι περίπου 1.000.000 pixel. Αντικείμενα θεωρούνται τα πολύγωνα ή τα τρίγωνα που μοντελοποιούν την επιφάνειά τους. Ο αριθμός των αντικειμένων μπορεί να ποικίλει με μεγάλη απόκλιση ανάλογα με τη σκηνή.

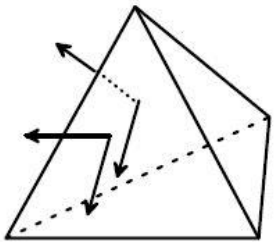
Άλλες στρατηγικές για καθορισμό των ορατών επιφανειών λαμβάνουν υπόψη τις σχετικές θέσεις μεταξύ των αντικειμένων . Τέτοιες τεχνικές καλούνται αλγόριθμοι ακρίβειας αντικειμένου. Αφού πρώτα καθοριστεί ποια αντικείμενα ή μέρη αντικειμένων είναι ορατά μόνο αυτά τα αντικείμενα προβάλλονται στο πλέγμα των pixel. Αυτοί οι αλγόριθμοι πρέπει να συγκρίνουν τα αντικείμενα σε ζευγάρια για να ανακαλύψουν ποια αντικείμενα είναι κρυμμένα για την συγκεκριμένη προβολή. Αυτό οδηγεί σε τετραγωνική πολυπλοκότητα για παράδειγμα n(n-1)/2 στην χείριστη περίπτωση με n τον αριθμό των αντικειμένων στην σκηνή. Συνήθως ο αριθμός των αντικειμένων στην σκηνή είναι πολύ πιο μικρός από τον αριθμό των pixel έτσι ώστε η σχέση $n^2 \ll n \cdot p$ ισχύει. Έτσι οι αλγόριθμοι ακρίβειας αντικειμένου υπερτερούν των αλγορίθμων ακρίβειας εικόνας. Όμως τα βήματα των αλγορίθμων ακρίβειας αντικειμένων είναι πολύ πιο πολύπλοκα από τα βήματα των αλγορίθμων ακρίβειας εικόνας. Ένα πλεονέκτημα των αλγορίθμων ακρίβειας αντικειμένων είναι ότι λειτουργούν ανεξάρτητα από την ανάλυση εφόσον προσδιορίζουν αν τα αντικείμενα είναι ορατά χωρίς να χρησιμοποιούν το πλέγμα των pixel. Μόνο για την τελική προβολή των ορατών αντικειμένων το πλέγμα των pixel χρησιμοποιείται.

Back-face Culling

Ανεξάρτητα από την στρατηγική που ακολουθείται για τον καθορισμό των ορατών αντικειμένων σε μία σκηνή , ο αριθμός των υποψήφιων αντικειμένων (πολύγωνα ή τρίγωνα των επιφανειών) πρέπει να μειωθεί στο ελάχιστο. Τα πολύγωνα τα οποία δεν «κοιτούν» τον παρατηρητή δεν είναι ορατά ούτε μπορούν να κρύψουν άλλα αντικείμενα από τον παρατηρητή. Αυτό σημαίνει ότι εφόσον όλα τα γεωμετρικά σχήματα είναι στερεά, αν η πίσω μεριά ενός αντικειμένου Α κρύβει κάποιο άλλο αντικείμενο Β από τον παρατηρητή τότε το αντικείμενο Β δεν είναι ορατό στον παρατηρητή αλλά επίσης και η πίσω μεριά του αντικειμένου Α δεν είναι ορατή στον παρατηρητή επειδή βρίσκεται πίσω από την μπροστά μεριά του αντικειμένου Α η οποία την κρύβει. Η αφαίρεση όλων των τριγώνων ή πολυγώνων που δείχνουν αντίθετα από εκεί που βλέπει ο παρατηρητής πριν γίνει ο καθορισμός των ορατών αντικειμένων της σκηνής ονομάζεται back-face culling.

Έχει ήδη αναφερθεί ότι τα τρίγωνα και τα πολύγωνα της επιφάνειας προσανατολίζονται βάση της σειράς των κορυφών τους [12.2.4]. Ένα πολύγωνο είναι ορατό μόνο από την πλευρά που οι κορυφές του εμφανίζονται αντίθετα με τη φορά του ρολογιού. Το normal vector του πολυγώνου μπορεί να προσανατολιστεί έτσι ώστε να δείχνει πάντα στην κατεύθυνση όπου το πολύγωνο είναι ορατό. Αν το normal vector δείχνει αντίθετα από εκεί που βλέπει ο παρατηρητής τότε ο παρατηρητής κοιτά το πολύγωνο από την πίσω πλευρά, άρα το πολύγωνο δεν είναι ορατό στον παρατηρητή και μπορεί να αγνοηθεί. Για μία παράλληλη προβολή στο x/y επίπεδο η κατεύθυνση της προβολής είναι παράλληλη στον άξονα των z. Ο z άξονας δείχνει προς τον παρατηρητή. Ένα πολύγωνο μπορεί να είναι ορατό από την μπροστινή πλευρά του μόνο σε περίπτωση σχηματισμού οξείας γωνίας μεταξύ του normal vector του πολυγώνου και της κατεύθυνσης της προβολής δηλαδή τον z άξονα. Στο τετράεδρο της εικόνας 157 μπορούμε να δούμε τα δύο παράλληλα διανύσματα που υποδεικνύουν την κατεύθυνση της προβολής (δείχνουν στην ίδια κατεύθυνση με τον άξονα των z) και τα άλλα δύο διανύσματα που είναι τα normal vectors των δύο πλευρών του τετράεδρου. Το normal vector της μπροστινής πλευράς σχηματίζει οξεία γωνία με το διάνυσμα της κατεύθυνσης της προβολής ενώ το normal vector της πίσω πλευράς σχηματίζει αμβλεία γωνία με το διάνυσμα της κατεύθυνσης της

προβολής. Η μπροστινή πλευρά θα είναι ορατή στον παρατηρητή εκτός αν υπάρχει κάποιο άλλο αντικείμενο μπροστά της ενώ η πίσω πλευρά δεν θα είναι ορατή στον παρατηρητή. Το back-face culling θα αφαιρέσει το τρίγωνο της πίσω πλευράς το οποίο θα αγνοηθεί στην διαδικασία καθορισμού των ορατών αντικειμένων στην σκηνή.



ΕΙΚΟΝΑ 157 BACK-FACE CULLING

Μία πλευρά μπορεί να αφαιρεθεί από το back-face culling αν και μόνο αν το normal vector σχηματίζει αμβλεία γωνία (> 90) με την κατεύθυνση της προβολής. Το εσωτερικό γινόμενο του normal vector $\mathrm{n} = (nx, ny, nz)^T$ με το μοναδιαίο διάνυσμα στην κατεύθυνση της προβολής $ez = (0, 0, 1)^T$ είναι : $ez^T \cdot n = \cos(\varphi) \cdot \|ez\| \cdot \|n\|$, όπου ϕ η γωνία μεταξύ των δύο διανυσμάτων και $\|n\|$ το μήκος του διανύσματος n. Το μήκος ενός διανύσματος είναι πάντα θετικός αριθμός και επίσης και τα δύο διανύσματα δεν είναι μηδενικά διανύσματα, έτσι το δεξί μέρος της : $ez^T \cdot n = \cos(\varphi) \cdot \|ez\| \cdot \|n\|$ είναι αρνητικό αν και μόνο αν το cos(ϕ)<0 δηλαδή αν ϕ>90. Το πρόσημο του εσωτερικού γινομένου δείχνει αν η επιφάνεια στην οποία αντιστοιχεί το n πρέπει να συμπεριληφθεί στην διαδικασία καθορισμού των ορατών αντικειμένων στην σκηνή. Όλες οι επιφάνειες με αρνητικό εσωτερικό γινόμενο μπορούν να μην συμμετάσχουν στην διαδικασία καθορισμού των ορατών αντικειμένων. Εφόσον ένα από τα διανύσματα του εσωτερικού γινομένου είναι το μοναδιαίο διάνυσμα το εσωτερικό γινόμενο μπορεί να γραφεί πιο απλά ως : $ez^T \cdot n = (0 \quad 0 \quad 1) \cdot \begin{pmatrix} nx \\ ny \\ nz \end{pmatrix} = nz$.

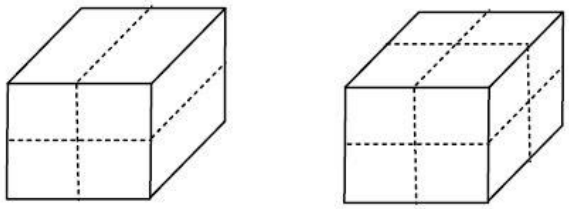
Αρκεί να ελέγξουμε το πρόσημο της z συνιστώσας του normal vector.

Χωρική Κατάτμηση – Spatial Partitioning

Η spatial partitioning προσπαθεί να μειώσει περισσότερο το υπολογιστικό κόστος. Ο ψαλιδισμένος όγκος υποδιαιρείται σε περιοχές για παράδειγμα σε 8 κουτιά ίσου μεγέθους. Τα αντικείμενα ανατίθενται στο κουτί ή στα κουτιά με τα οποία δεν έχουν κενή τομή (object ∩ subdivides box ≠ ∅). Τα αντικείμενα που βρίσκονται στα όρια που διαχωρίζονται τα κουτιά ανατίθενται και στα δύο κουτιά. Στην περίπτωση των αλγορίθμων ακρίβειας αντικειμένου πρέπει να ελεγχτούν μόνο τα αντικείμενα μέσα στο ίδιο κουτί. Αν ένα κουτί βρίσκεται πίσω από ένα άλλο τα αντικείμενα του μπροστινού κουτιού θα προβάλλονται πριν από τα αντικείμενα του πίσω κουτιού. Με αυτό τον τρόπο τα αντικείμενα από το μπροστινό κουτί θα αντικαταστούν αυτόματα τα αντικείμενα από το πίσω κουτί τα οποία δεν είναι ορατά από τον παρατηρητή. Για μία κατάτμηση του ψαλιδισμένου όγκου σε k κουτιά , τα n αντικείμενα στην σκηνή, στην ιδανική περίπτωση ,θα διαμοιραστούν ίσα στα κουτιά. Η υπολογιστική πολυπλοκότητα μειώνεται από n^2 σε $k \cdot \left(\frac{n}{k}\right)^2 = \frac{n^2}{k}$. Αυτό είναι αληθές αν κανένα από τα αντικείμενα δεν βρίσκεται πάνω στα όρια διαχωρισμού των κουτιών και επίσης αν τα αντικείμενα είναι ίσα διαμοιρασμένα στα κουτιά δηλαδή κάθε κουτί να περιέχει $\frac{n}{k}$ κουτιά. Το παραπάνω συμπέρασμα δεν θα ισχύει αν τα κουτιά γίνουν πολύ μικρά.

Για τους αλγορίθμους ακρίβειας εικόνας είναι καλύτερο να γίνεται η διάτμηση του

ψαλιδισμένου όγκου όπως φαίνεται στην εικόνα 158. Το ορθογώνιο του επιπέδου προβολής χωρίζεται σε μικρότερα ορθογώνια και κάθε μικρότερο ορθογώνιο περιέχει ένα κουτί του ψαλιδισμένου όγκου. Για κάθε pixel μόνο αυτά τα αντικείμενα πρέπει να εξεταστούν τα ποία βρίσκονται μέσα στο κουτί το οποίο σχετίζεται με το μικρότερο ορθογώνιο στο οποίο βρίσκεται το pixel.

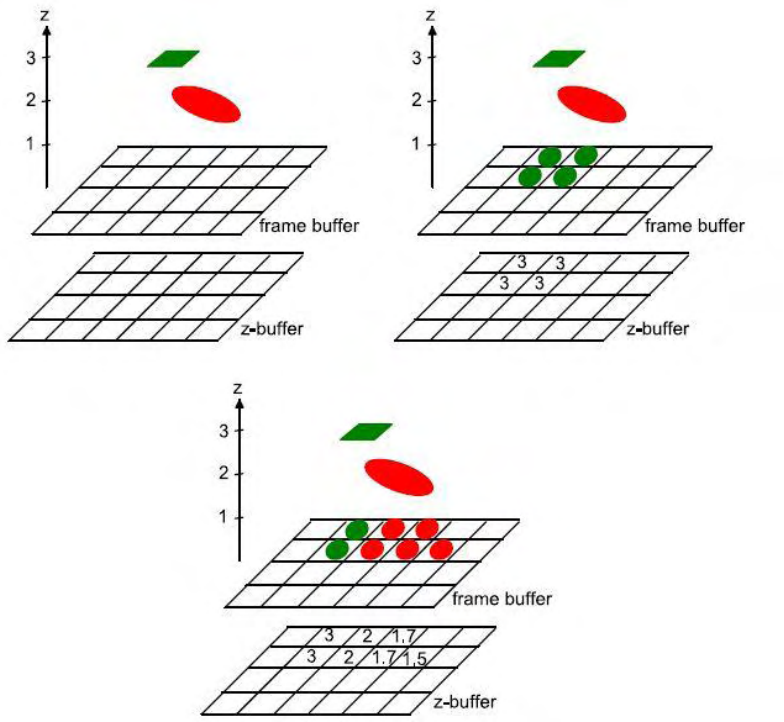


ΕΙΚΟΝΑ 158 ΔΙΑΤΜΗΣΗ ΤΟΥ ΨΑΛΙΔΙΣΜΕΝΟΥ ΟΓΚΟΥ ΓΙΑ ΑΛΓΟΡΙΘΜΟΥΣ ΑΚΡΙΒΕΙΑΣ ΕΙΚΟΝΑΣ ΣΤΑ ΑΡΙΣΤΕΡΑ ΚΑΙ ΑΛΓΟΡΙΘΜΟΥΣ ΑΚΡΙΒΕΙΑΣ ΑΝΤΙΚΕΙΜΕΝΟΥ ΣΤΑ ΔΕΞΙΑ

Αναδρομικοί αλγόριθμοι υποδιαίρεσης χωρίζουν την ψαλιδισμένη περιοχή όλο και περισσότερο μέχρις ότου μία περιοχή να είναι αρκετά μικρή για να αποφασιστεί αν το αντικείμενο είναι ορατό μέσα σε αυτή. Ένα άνω όριο για το μέγιστο βάθος της υποδιαίρεσης δίνεται από την ανάλυση της εικόνας.

Τεχνικές Ακρίβειας Εικόνας – Αλγόριθμος Z-Buffer

Ο αλγόριθμος z-buffer είναι η πιο ευρέως διαδεδομένη τεχνική για τον καθορισμό των ορατών επιφανειών των αντικειμένων. Ο αλγόριθμος χρησιμοποιεί έναν frame buffer (buffer πλαισίων) για τα χρώματα των pixel της εικόνας και έναν z-buffer στον οποίο μία z-τιμή εκχωρείται για κάθε pixel. Ο z-buffer αρχικοποιείται με την απόσταση ή την z συνιστώσα της πίσω clipping plane. Ο frame buffer αρχικοποιείται με την εικόνα του φόντου ή με το χρώμα του φόντου. Τα αντικείμενα προβάλλονται με αυθαίρετη σειρά και οι προβολές τους αποθηκεύονται στον frame buffer. Αν όλα τα αντικείμενα έμπαιναν με αυτόν τον τρόπο στον frame buffer τότε όλα τα αντικείμενα που θα προβάλλονταν αργότερα θα αντικαθιστούσαν τα αντικείμενα που προβλήθηκαν ήδη όταν οι προβολές τους επικαλύπτονται. Εφόσον η σειρά των προβολών δεν είναι ορισμένη , αυτό θα οδηγούσε σε λάθος αποτελέσματα για παράδειγμα αντικείμενα μακρινά στον παρατηρητή θα έκρυβαν αντικείμενα πιο κοντινά στον παρατηρητή. Έτσι πριν ένα pixel του προβαλλόμενου αντικειμένου μπει στον frame buffer η z-τιμή του (δηλαδή η απόστασή του από το επίπεδο προβολής) συγκρίνεται με την z-τιμή που υπάρχει ήδη για αυτό το pixel στον z-buffer. Αν η τιμή στον z-buffer είναι μεγαλύτερη τότε το νέο χρώμα του pixel του νέου προβαλλόμενου αντικειμένου μπαίνει στον frame buffer και η τιμή του pixel στον z-buffer ανανεώνεται. Σε αντίθετη περίπτωση ούτε ο z-buffer ούτε ο frame buffer ανανεώνονται.



ΕΙΚΟΝΑ 159 Z-BUFFER ΑΛΓΟΡΙΘΜΟΣ

Σύμφωνα με την εικόνα 159 υπάρχουν δύο αντικείμενα προς προβολή ένα ορθογώνιο και μία έλλειψη. Ο παρατηρητής βλέπει την σκηνή κάτω από το επίπεδο προβολής ή τον frame buffer. Ο frame buffer αρχικοποιείται με το χρώμα του φόντου στην περίπτωση αυτή με το χρώμα άσπρο και ο z-buffer με την τιμή z της πίσω clipping plane. Οι αρχικές τιμές του z-buffer δεν φαίνονται στην εικόνα 159. Το ορθογώνιο προβάλλεται πρώτο και εφόσον το ορθογώνιο βρίσκεται μέσα στον ψαλιδισμένο όγκο και η z τιμή του είναι μικρότερη από τν z τιμή την πίσω clipping plane,το ορθογώνιο προβάλλεται στον frame buffer και η z τιμή του εισέρχεται στον z-buffer. Μετά προβάλλεται η έλλειψη όπου για κάποια pixel όπου τα δύο σχήματα επικαλύπτονται η z τιμές των αντίστοιχων pixel της έλλειψης είναι μικρότερες από τις z τιμές των pixel του ορθογωνίου. Έτσι η έλλειψη υπερισχύει του ορθογωνίου εν μέρει στον frame buffer και οι z τιμές της μπαίνουν στον z-buffer. Αν προβάλλονταν πρώτα η έλλειψη τότε η έλλειψη θα έμπαινε στον frame buffer και οι z τιμές της στον z-buffer εφόσον οι z τιμές της είναι μικρότερες από εκείνες της πίσω clipping plane. Ύστερα θα προβάλλονταν το ορθογώνιο και θα έμπαιναν στον frame buffer μόνο τα pixel στα οποία αντιστοιχούσαν οι μικρότερες z τιμές από εκείνες που ήδη είχε ο z-buffer στα αντίστοιχα pixel (σε εκείνα που αφορούσαν μόνο την πίσω clipping plane). Έτσι το ορθογώνιο θα προβάλλονταν εν μέρει και στα pixel που επικαλύπτεται με την έλλειψη πάλι δεν θα πραγματοποιούνταν η προβολή του γιατί οι z τιμές της έλλειψης είναι μικρότερες από εκείνες του ορθογωνίου. Και στις δύο περιπτώσεις οδηγούμαστε στο ίδιο αποτέλεσμα.

Οι z τιμές ενός αντικειμένου δεν είναι συνήθως σταθερές. Πρέπει να αποφασιστεί για κάθε pixel ξεχωριστά αν θα μπει ή όχι στον frame buffer και στον z-buffer. Για να εισάγουμε πολύγωνα στον frame buffer και στον z-buffer η τεχνική scan-line (τεχνική γραμμών σάρωσης) εφαρμόζεται σε κάθε σειρά από pixel στο επίπεδο προβολής. Έστω το επίπεδο του πολυγώνου δίνεται από την εξίσωση : $A \cdot x + B \cdot y + C \cdot z + D = 0$. Η z τιμή κατά μήκος μίας scan-line μπορεί να υπολογιστεί από τον τύπο : $z_{new} = z_{old} + \Delta_z$, εφόσον οι z τιμές του επιπέδου αλλάζουν με γραμμικό τρόπο όταν δειγματοληπτούμε κατά μήκος της γραμμής. Η τιμή z_{old} είναι η z συντεταγμένη του προβαλλόμενου πολυγώνου στο pixel (x,y). Η νέα z συντεταγμένη z_{new} για το επόμενο pixel (x+1,y) πρέπει να ικανοποιεί την $A \cdot x + B \cdot y + C \cdot z + D = 0$ όπως επίσης και για το προηγούμενο σημείο (x,y, z_{old})

πρέπει να την ικανοποιεί εφόσον και τα δύο σημεία βρίσκονται πάνω στο επίπεδο. Έτσι έχουμε : $0 = A \cdot (x+1) + B \cdot y + C \cdot z_{new} + D = A \cdot (x+1) + B \cdot y + C \cdot (z_{old} + \Delta_z) + D = A \cdot x + B \cdot y + C \cdot z_{old} + D + A + C \cdot \Delta_z = A + C \cdot \Delta_z$

Έτσι η αλλαγή της z συντεταγμένης κατά μήκος της scan-line είναι $\Delta_z = -\frac{A}{C}$.

Τεχνικές Ακρίβειας Εικόνας – Scan-Line

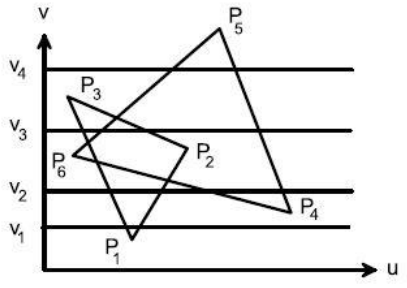
Για τον z-buffer αλγόριθμο η προβολή ενός πολυγώνου μπορεί να βασιστεί στην τεχνική scan-line. Σαν εναλλακτική λύση μπορούμε επίσης να προβάλλουμε τις ακμές όλων των πολυγώνων και να εφαρμόσουμε την τεχνική scan-line έτσι ώστε να καθορίσουμε ποια πολύγωνα πρέπει να σχεδιαστούν και ποια όχι. Οι άξονες συντεταγμένων της ορθογώνιας ψαλιδισμένης περιοχής πάνω στο επίπεδο προβολής ορίζονται ως u και v. Η τεχνική αυτή βασίζεται σε τρεις πίνακες. Ο πίνακας ακμών περιέχει όλες τις μη οριζόντιες ακμές και έχει την ακόλουθη δομή :

v_{min}	u(v_{min})	u_{max}	Δ_u	Polygon numbers

v_{min} είναι η μικρότερη v τιμή της προβαλλόμενης ακμής , u(v_{min}) είναι η u τιμή που αντιστοιχεί στο v_{min} · v_{max} που υποδηλώνει την μεγαλύτερη v τιμή της ακμής. Δ_u είναι η κλίση της προβαλλόμενης ακμής. Η στήλη Polygon numbers περιέχει την λίστα όλων των πολυγώνων στα οποία ανήκουν οι ακμές. Οι ακμές είναι ταξινομημένες με αυξάνουσα σειρά ως προς τις τιμές v_{min} . Για ίδιες τιμές v_{min} οι ακμή με την μικρότερη u(v_{min}) τιμή γράφεται πρώτη. Ο δεύτερος πίνακας είναι ο πίνακας των πολυγώνων ο οποίος περιέχει πληροφορίες για τα πολύγωνα και έχει την ακόλουθη δομή :

Polygon no.	A	B	C	D	Color	In-flag

Το Polygon No. Είναι ένα αναγνωριστικό των πολυγώνων. Οι συντελεστές A, B, C, D ορίζουν το επίπεδο που αντιστοιχεί στο πολύγωνο βάση της εξίσωσης $A \cdot x + B \cdot y + C \cdot z + D = 0$. Η στήλη Color περιέχει το χρώμα ή πληροφορία για την σκίαση των πολυγώνων. Η στήλη In-flag δείχνει αν η θέση που βρίσκεται η scan-line είναι εντός ή εκτός του πολυγώνου. Ο τελευταίος πίνακας περιέχει μία λίστα με όλες τις ενεργές ακμές. Ενεργές ακμές λέγονται αυτές οι ακμές που τέμνουν την τρέχουσα scan-line. Αυτές οι ακμές ταξινομούνται την u τιμή των σημείων τομής τους με αυξάνουσα σειρά. Ο αριθμός των γραμμών του πίνακα με τις ενεργές ακμές μπορεί να είναι διαφορετικός για κάθε scan-line. Ο αριθμός των γραμμών και των εγγραφών των άλλων δύο πινάκων παραμένει σταθερός εκτός από την στήλη In-flag.



ΕΙΚΟΝΑ 160 ΚΑΘΟΡΙΣΜΟΣ ΕΝΕΡΓΩΝ ΑΚΜΩΝ ΓΙΑ ΤΙΣ SCAN-LINES V1, V2, V3, V4

Για το σχήμα της εικόνας 156 οι ενεργές ακμές για τις scan-lines v1, v2, v3, v4 είναι :

V1 : P3P1, P1P2

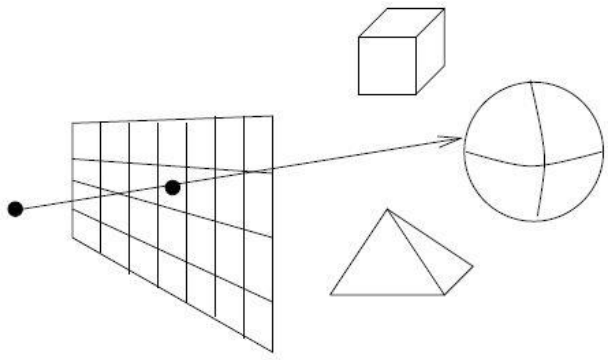
V2 : P3P1, P1P2, P6P4, P5P4

V3 : P3P1, P6P5, P3P2, P5P4
V4 : P6P5, P5P4

Για κάθε scan-line καθορίζονται πρώτα οι ενεργές ακμές και όλα τα In-flags αρχικοποιούνται σε 0. Όταν η γραμμή σαρώνεται , κάθε φορά που συναντάται μία ακμή τα In-flags του πολυγώνου που ανήκει η ακμή πρέπει να αλλάξουν από 0 σε 1 ή από 1 σε 0 εφόσον κάθε φορά που συναντάμε μία ακμή σημαίνει ότι εισερχόμαστε ή εξερχόμαστε από το πολύγωνο. Για κάθε pixel πρέπει να οριστούν αν είναι ορατά αυτά τα πολύγωνα με την τιμή In-flag = 1. Για αυτό το λόγο η z τιμή για κάθε πολύγωνο με In-flag=1 πρέπει να υπολογιστεί βάση της εξίσωσης του επιπέδου. Το πολύγωνο με την μικρότερη τιμή z είναι αυτό που είναι ορατό στο συγκεκριμένο pixel.

Τεχνικές Ακρίβειας Εικόνας – Ray Casting

Η τεχνική ray casting για κάθε pixel στο ψαλιδισμένο ορθογώνιο του επιπέδου προβολής ρίπτει μία ακτίνα (ray) παράλληλα στην κατεύθυνση της προβολής. Η ακτίνα πρέπει να ξεκινά από την μπροστινή clipping plane και να τελειώνει στην πίσω clipping plane. Το πρώτο αντικείμενο που συναντά η ακτίνα καθορίζει το χρώμα του pixel . Η τεχνική ray casting είναι κατάλληλη για παράλληλη όσο και για προοπτική προβολή χωρίς κάποιον μετασχηματισμό. Οι ακτίνες είναι παράλληλες στην κατεύθυνση της προβολής για την παράλληλη προβολή και για την προοπτική προβολή οι ακτίνες ρίπτονται κατά μήκος των ενώσεων του κέντρου της προβολής με τα pixel. Τα pixel αντιστοιχούν στο κέντρο κάθε τετραγώνου στο πλέγμα του επιπέδου προβολής της εικόνας 161.



ΕΙΚΟΝΑ 161 ΤΕΧΝΙΚΗ RAY – CASTING

Για προοπτική προβολή με κέντρο προβολής το (x0, y0, z0) η ακτίνα στο pixel με συντεταγμένες (x1, y1, z1) μπορεί να παραμετροποιηθεί βάση των εξισώσεων :

$$x = x0 + t \cdot \Delta x \ , \qquad y = y0 + t \cdot \Delta y \ , \qquad z = z0 + t \cdot \Delta z$$

Όπου:

$$\Delta x = x1 - x0 \ , \qquad \Delta y = y1 - y0 \ , \qquad \Delta z = z1 - z0$$

Για τιμές με t<0 η ακτίνα βρίσκεται πίσω από το κέντρο της προβολής, για t∈ $[0,1]$ η ακτίνα βρίσκεται μεταξύ της προβολής και του επιπέδου προβολής ,για t>1 η ακτίνα βρίσκεται πίσω από το επίπεδο προβολής.

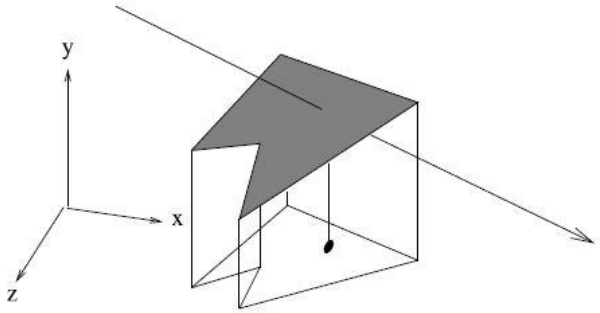
Για να καθορίσουμε αν η ακτίνα τέμνει ένα πολύγωνο και αν ναι, να καθορίσουμε σε ποιο σημείο το τέμνει πρέπει να υπολογιστεί το σημείο τομής της ακτίνας με το επίπεδο του πολυγώνου

$A \cdot x + B \cdot y + C \cdot z + D = 0$. Έπειτα γίνεται ένας έλεγχος αν το σημείο τομής βρίσκεται μέσα στο πολύγωνο. Λόγω των παραμετροποιημένων εξισώσεων που ορίστηκαν για την ακτίνα παραπάνω η εξίσωση του πολυγώνου μας δίνει την τιμή t :

$$t = -\frac{Ax0 + Byo + Cz0 + D}{A\Delta x + B\Delta y + C\Delta z}$$

Εφόσον η $A \cdot x + B \cdot y + C \cdot z + D = 0$ περιγράφει ένα επίπεδο αυτό σημαίνει ότι τουλάχιστον έναν από τους συντελεστές A, B, C είναι μη μηδενικός , επίσης ο παρονομαστής μπορεί να γίνει 0 αν και μόνο αν η ακτίνα είναι παράλληλη στο επίπεδο. Σε αυτή την περίπτωση το επίπεδο δεν λαμβάνεται υπόψη για την προβολή στο συγκεκριμένο pixel.

Για να καθορίσουμε αν το σημείο τομής βρίσκεται μέσα στο πολύγωνο, το πολύγωνο προβάλλεται μαζί με το σημείο τομή σε ένα από τα επίπεδα που ορίζονται από τους άξονες του συστήματος συντεταγμένων. Αυτό σημαίνει ότι μία από τις συντεταγμένες γίνεται 0. Για να αποφύγουμε λάθη στρογγυλοποίησης, το επίπεδο που επιλέγεται για την προβολή πρέπει να είναι εκείνο που είναι πιο παράλληλο στο επίπεδο του πολυγώνου. Αυτό σημαίνει ότι το normal vector του πολυγώνου με το normal vector στο επίπεδο προβολής πρέπει να είναι όσο πιο παράλληλα γίνεται. Η μεταξύ τους γωνία πρέπει να είναι κοντά στις 0 ή στις 180 μοίρες. Άρα το εσωτερικό γινόμενο των normal vector τους πρέπει να είναι κοντά στο 1 ή στο -1 αν θεωρήσουμε ότι τα normal vectors είναι κανονικοποιημένα. Το εσωτερικό γινόμενο του normal vector (A, B, C) με το normal vector του επιπέδου που ορίζεται από τους άξονες είναι ο παράγοντας (A, B, C) που θα γίνει 0 για την προβολή. Έτσι το επίπεδο προβολής επιλέγεται κάθετο στον παράγοντα που έχει την μεγαλύτερη απόλυτη τιμή (A, B, C). Μετά την προβολή ο κανόνας odd-parity [6.2.3] εφαρμόζεται για να αποφασιστεί αν το σημείο τομής βρίσκεται μέσα στο προβαλλόμενο πολύγωνο όπως φαίνεται στην εικόνα 162.



Εικονα 162 προβολη πολυγωνου για να αποφασιστει αν το σημειο βρισκεται εντος του πολυγωνου

Πρέπει να λάβουμε υπόψη την συνοχή (Coherence) για να μειωθεί η υπολογιστική πολυπλοκότητα της τεχνικής ray casting. Με τον όρο Coherence αναφερόμαστε στην εκμετάλλευση θεωρήσεων όπως οι παρακάτω :

- Γειτονικά pixel συνήθως παίρνουν το χρώμα τους από το ίδιο πολύγωνο
- Μόλις η ακτίνα τέμνει ένα πολύγωνο, δεν είναι απαραίτητο να υπολογίσουμε τις τομές με πολύγωνα που βρίσκονται μακρύτερα.

Χωρίς την εκμετάλλευση της Coherence (1000x1000x100) 100 εκατομμύρια έλεγχοι τομών θα πρέπει να γίνουν για ανάλυση 1000x1000 pixel και με 100 αντικείμενα στην σκηνή. Η Coherence μπορεί για παράδειγμα να ελαττώσει τον χρόνο υπολογισμού με τον ακόλουθο τρόπο. Όταν ένα πολύγωνο έχει υπολογιστεί καθορίζοντας το χρώμα του pixel , ο έλεγχος τομής για τα γειτονικά

pixel θα πρέπει να εφαρμοστεί πρώτα σε αυτό το πολύγωνο. Όταν η νέα ακτίνα τμήσει πάλι το ίδιο πολύγωνο δεν θα χρειάζονται περαιτέρω έλεγχοι τομής.

12.2.7 Πηγες Φωτος και Αποσβεση Φωτος

Σε κάθε scene graph υπάρχει η διακλάδωση του φωτισμού. Η διακλάδωση αυτή υλοποιείται μέσα σε μία μέθοδο την add Light() καλούμενη για το αντικείμενο της κλάσης Simple Universe που ορίζει τον εικονικό κόσμο κάθε επιλογής. Στην μέθοδο αυτή ορίζονται οι πηγές φωτός που υπάρχουν μέσα στον εικονικό κόσμο έτσι ώστε να κάνουν ορατά τα αντικείμενα που έχουμε δημιουργήσει. Αν δεν προσθέσουμε φως στον εικονικό κόσμο τότε όλα τα αντικείμενα δεν είναι ορατά, εμφανίζεται απλά ένα μαύρο παράθυρο.

Στις περισσότερες περιπτώσεις οι πηγές φωτός παρέχουν άσπρο ή γρι φως (άσπρο φως που δεν έχει την πλήρη ένταση). Επίσης υπάρχει η δυνατότητα να έχουμε χρωματιστά φώτα. Το χρώμα αλλά και η ένταση του φωτός καθορίζονται από RGB τιμές.

Η απλούστερη μορφή φωτός είναι το φως περιβάλλοντος (ambient light). Το ambient light δεν έρχεται από κάποια συγκεκριμένη πηγή φωτός και δεν έχει κατεύθυνση. Αντιπροσωπεύει το φως το οποίο βρίσκεται παντού στην σκηνή προερχόμενο από πολλές αντανακλάσεις του φωτός σε διάφορες επιφάνειες. Σε ένα δωμάτιο με μία λάμπα πάνω στο τραπέζι , κάτω από το τραπέζι δεν θα είναι εντελώς σκοτεινά μολονότι η λάμπα δεν μπορεί να διασκορπίσει το φως κάτω από το τραπέζι. Το φως περιβάλλοντος είναι η απλοποίηση των υπολογισμών για τον φωτισμό. Ο σωστός τρόπος να υπολογίσεις το φως περιβάλλοντος είναι να ανιχνεύσεις όλες τις αντανακλάσεις του φωτός από τα αντικείμενα της σκηνής. Αυτό βέβαια θα αύξανε υπερβολικά το υπολογιστικό κόστος οπότε αυτή η προσέγγιση δεν είναι επιτρεπτή για γραφικά πραγματικού χρόνου. Για το ambient light είναι αρκετό να ορίσουμε το χρώμα του.

Μία κατευθυνόμενη πηγή φωτός για τον ορισμό της εκτός από το χρώμα χρειάζεται και κατεύθυνση. Οι ακτίνες του φωτός από μία κατευθυνόμενη πηγή φωτός (directional light) είναι παράλληλες. Αυτή η πηγή φωτός χρησιμοποιείται για την μοντελοποίηση φωτός ερχόμενο από μία πηγή σχεδόν σε άπειρη απόσταση, για παράδειγμα από τον ήλιο.

Μία λάμπα μοντελοποιείται σαν σημειακή πηγή φωτός (point light). Ένα point light έχει θέση και οι ακτίνες του φωτός διαχέονται σε όλες τις κατευθύνσεις από την θέση αυτή. Η ένταση του φωτός μειώνεται καθώς μεγαλώνει η απόσταση. Το φαινόμενο αυτό ονομάζεται απόσβεση (attenuation). Το επόμενο επιχείρημα δείχνει ότι η ένταση του φωτός μειώνεται τετραγωνικά με την απόσταση από την πηγή φωτός. Αν ένα point light είναι στο κέντρο μίας σφαίρας ακτίνας r τότε όλη η ενέργεια του φωτός θα διανεμηθεί ίσα στο εσωτερικό κομμάτι της σφαίρας. Αν η σφαίρα αντικατασταθεί από μία μεγαλύτερη σφαίρα με ακτίνα R τότε όλη η ενέργεια του φωτός δεν θα αλλάξει αλλά θα διανεμηθεί σε μία μεγαλύτερη επιφάνεια. Η αναλογία των επιφανειών των 2 σφαιρών είναι :

$$\frac{4\pi r^2}{4\pi R^2} = (\frac{r}{R})^{\,2}$$

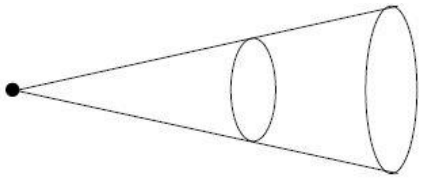
Για αναλογία $\frac{r}{R} = \frac{1}{2}$ κάθε σημείο της εσωτερικής επιφάνειας της μεγάλης σφαίρας λαμβάνει μόνο το ένα τέταρτο της ενέργειας που λαμβάνουν τα σημεία της εσωτερικής επιφάνειας της μικρότερης σφαίρας. Το θεωρητικό μοντέλο της απόσβεσης αποτελεί ο πολλαπλασιασμός της έντασης του φωτός ενός point light με τον παράγοντα $1/d^2$, όπου d η απόσταση του αντικειμένου από την πηγή φωτός. Η ένταση του φωτός θα μειώνεται τόσο γρήγορα με την απόσταση έτσι ώστε οι

διαφορές στην ένταση για μεγαλύτερες αποστάσεις θα είναι σχεδόν απαρατήρητες. Αντιθέτως για αντικείμενα κοντά στην πηγή φωτός θα παρατηρούνται δραστικές αλλαγές. Η ένταση μπορεί να είναι πολύ μεγάλη και μπορεί να τείνει στο άπειρο όταν μία επιφάνεια είναι ακριβώς μπροστά από την πηγή φωτός. Για να αποφύγουμε αυτά τα φαινόμενα η μείωση της έντασης που δημιουργείται λόγω της απόσβεσης μοντελοποιείται από έναν γενικό τετραγωνικό πολυώνυμο στον παρονομαστή :

$$f_{att} = min\left\{\frac{1}{c1 + c2d + c3d^2}, 1\right\}$$

Όπου οι σταθερές c1, c2, c3 μπορούν να επιλεχθούν ατομικά για κάθε point light. d είναι η απόσταση του αντικειμένου από την πηγή φωτός. Ο τύπος αυτός εγγυάται ότι η ένταση δεν θα περάσει ποτέ την τιμή 1. Οι σταθερές μπορούν επίσης να προσαρμοστούν έτσι ώστε να δημιουργηθεί μία πιο ήπια απόσβεση από την απόσβεση με $1/d^2$. Η σταθερά c2 του γραμμικού όρου μπορεί να χρησιμοποιηθεί για να μοντελοποιήσει ατμοσφαιρική απόσβεση. Η τετραγωνική μείωση της έντασης του φωτός προέρχεται από την διανομή της ενέργειας του φωτός σε μεγαλύτερη επιφάνεια για μεγαλύτερη απόσταση. Επιπροσθέτως, μέρος του φωτός απορροφάται από σωματίδια σκόνης που βρίσκονται στον αέρα προκαλώντας ατμοσφαιρική αδιαφάνεια (θολότητα – opacity). Αυτό οδηγεί σε γραμμική μείωση της έντασης όσο η απόσταση αυξάνεται. Ο αριθμός των σωματιδίων σκόνης τα οποία πετυχαίνει μία ακτίνα φωτός αυξάνει ανάλογα με την απόσταση που καλύπτει η ακτίνα.

Μία άλλη πηγή φωτός είναι τα spotlights. Τα spotlights έχουν κατεύθυνση στην οποία διαχέουν το φως τους σε μορφή κώνου. Το spotlight χαρακτηρίζεται από το χρώμα του φωτός, από την θέση, την κατεύθυνση αλλά και ένα όριο για την γωνία του κώνου που σχηματίζει το φως. Η απόσβεση υπολογίζεται κατά βάση όπως και για τα point lights. Η τετραγωνική μείωση της έντασης σε σχέση με την αύξηση της απόστασης μπορεί να φανεί από την εικόνα 163.

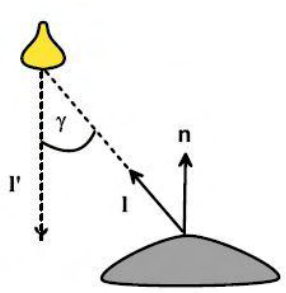


ΕΙΚΟΝΑ 163 ΚΩΝΟΣ ΦΩΤΟΣ ΑΠΟ SPOTLIGHT

Όλη η ενέργεια από το spotlight διανέμεται σε έναν κύκλο του οποίου η ακτίνας αυξάνει γραμμικά με την απόσταση. Άρα η επιφάνεια αυξάνει τετραγωνικά με την απόσταση. Για ένα πιο ρεαλιστικό μοντέλο του spotlight θα πρέπει να λάβουμε υπόψη ότι η ένταση του φωτός είναι μικρότερη κοντά στα όρια του κώνου από ότι στο κέντρο του. Σύμφωνα με το μοντέλο του Warn μία παράμετρος p χρησιμοποιείται για να ελέγξει πόσο γρήγορα μειώνεται η ένταση του φωτός από το κέντρο του κώνου πηγαίνοντας προς τα όριά του. Έστω ένα σημείο στην επιφάνεια που φωτίζεται από ένα spotlight, I ένα διάνυσμα που δείχνει από το σημείο που βρίσκεται το spotlight προς ένα σημείο πάνω στην επιφάνεια του αντικειμένου που φωτίζεται. I_s είναι ο άξονας του κώνου που δείχνει προς την κατεύθυνση του φωτός. Τότε η ένταση του φωτός στο σημείο της επιφάνειας υπολογίζεται από το μοντέλο του Warn σύμφωνα με τον τύπο :

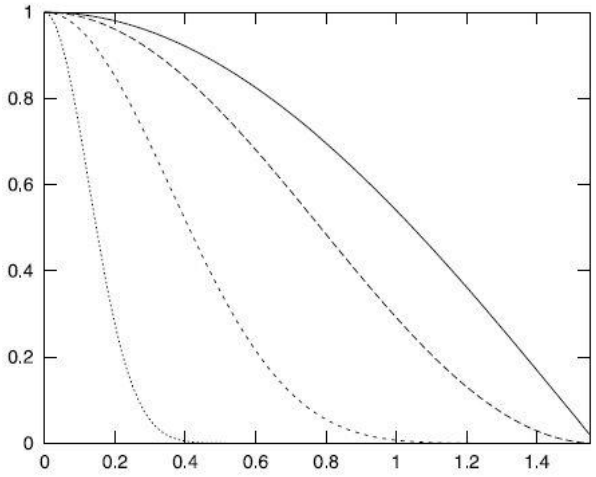
$$I = I_s \cdot f_{att} \cdot (\cos\gamma)^p = I_s \cdot f_{att} \cdot (-I_s^T \cdot I)^p$$

I_s η ένταση του spotlight, f_{att}ο παράγοντας απόσβεσης και γ η γωνία μεταξύ του I και I_s. Η τιμή p ελέγχει κατά πόσο το spotlight είναι εστιασμένο (focus). Για p=0 το spotlight συμπεριφέρεται σαν point light. Για μεγαλύτερο p το φως συγκεντρώνεται περισσότερο στον άξονα του κώνου και λιγότερο στα όρια. Το cos στο τύπο μπορεί να υπολογιστεί ως το εσωτερικό γινόμενο των διανυσμάτων I και I_s αν και τα δύο είναι κανονικοποιημένα (έχουν μήκος 1).



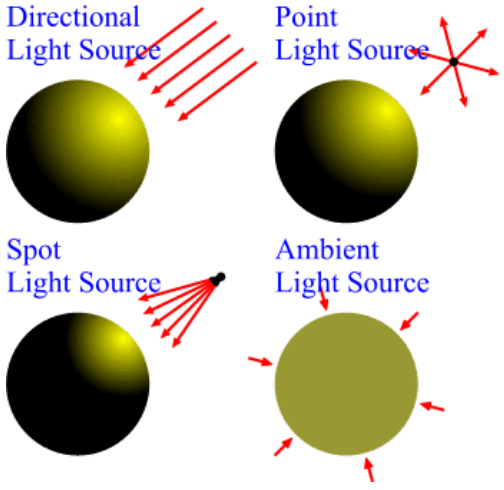
Εικονα 164 μοντελο του warn για spotlight

Η εικόνα 165 δείχνει την επίδραση της παραμέτρου p. Η συνάρτηση $(\cos\gamma)^p$ σχεδιάζεται για τις τιμές του p = 1, 2, 8, 64. Στο p=1 αντιστοιχεί η πιο δεξιά καμπύλη, η ένταση μειώνεται αργά προς το 0 με αύξηση της γωνίας γ. Οι μεγαλύτερες τιμές του p οδηγούν σε πολύ μικρές εντάσεις εφόσον το $(\cos\gamma)^p$ είναι σχεδόν 0. Πρέπει να σημειωθεί ότι η ένταση I_s στο μοντέλο του Warn μπορεί να πάρει διαφορετικές τιμές για κάθε μία από τις τρεις συνιστώσες του χρώματος red, green, blue.



Εικονα 165 εξισωση $(\cos\gamma)^p$

Για την μέθοδο add Light() του εικονικού κόσμου της επιλογής αυτής έχει χρησιμοποιηθεί ambient light, ένα directional light και ένα point light.



Εικονα 166 πηγες φωτος

12.2.8 Ανταναклαση Φωτος

Για να προστεθεί φωτισμός και σκίαση στην σκηνή είναι απαραίτητο να ορίσουμε για όλες τις επιφάνειες των αντικειμένων τον τρόπο που αντανακλούν το φως. Το μοντέλο φωτισμού που ακολουθείται εδώ δεν είναι σωστό από την σκοπιά της φυσικής. Το φως περιβάλλοντος έτσι όπως μοντελοποιείται στον εικονικό κόσμο δεν υφίσταται με την ίδια εφαρμογή και στην πραγματικότητα, ούτε είναι σταθερό παντού. Το φως περιβάλλοντος είναι μία απλούστευση του πραγματικού φωτός για απλούστευση των υπολογισμών και της απόδοσης. Για τον ίδιο λόγο, το φως το οποίο αντανακλάται από αντικείμενα δεν συμπεριλαμβάνεται στους υπολογισμούς για τον φωτισμό. Τα αντικείμενα μέσα σε μία σκηνή μόνο αντανακλούν το φως από ορισμένες πηγές φωτός και δεν αντανακλούν το φως που διαχέεται πάνω τους από αντανακλάσεις φωτός άλλων αντικειμένων. Το φως περιβάλλοντος αντικαθιστά αυτές τις πολύπλοκες αντανακλάσεις.

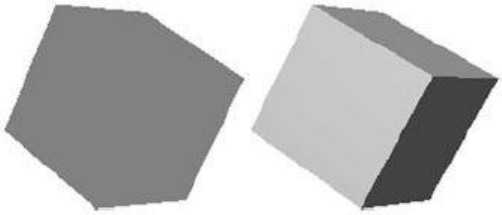
Θεωρούμε ένα σημείο στην επιφάνεια ενός αντικειμένου για το οποίο υπολογίζεται το χρώμα που πρέπει να του ανατεθεί συμπεριλαμβάνοντας τις πηγές φωτός που υπάρχουν στην σκηνή όπως επίσης και τις ιδιότητες αντανάκλασης της επιφάνειας. Για κάθε ένα από τα εφέ που θα μελετηθούν στην παράγραφο αυτή καθορίζεται μία RGB τιμή η οποία μοντελοποιεί την επίδραση του συγκεκριμένου εφέ στην επιφάνεια όταν εκείνο επιδρά μόνο του. Για να καθορίσουμε το τελικό χρώμα της επιφάνειας που αντιστοιχεί το συγκεκριμένο σημείο πρέπει να προστεθούν όλα τα εφέ. Οι μέγιστη τιμή για τα βασικά χρώματα red, green, blue είναι 1.

Τα αντικείμενα μπορεί να εκπέμπουν φως από μόνα τους. Το εκπεμπόμενο φως των αντικειμένων συμπεριλαμβάνεται στο τελικό χρώμα μόνο όταν το αντικείμενο φαίνεται πιο φωτεινό για τον παρατηρητή. Το εκπεμπόμενο χρώμα δεν φωτίζει άλλα αντικείμενα σε αυτό το μοντέλο. Αν ένα αντικείμενο εκπέμπει φως τότε η εκπομπή αυτή συνεισφέρει ένταση σε κάθε ένα από τα βασικά χρώματα : κόκκινο, πράσινο και μπλε. Αν θεωρήσουμε ότι η εκπομπή φωτός από το αντικείμενο είναι το μοναδικό εφέ που υφίσταται το αντικείμενο ένα pixel στην επιφάνεια του αντικειμένου i θα έχει ένταση : I = $k_ι$. Αυτή η ένταση πρέπει να προσδιοριστεί ξεχωριστά για το κόκκινο, το πράσινο και το μπλε. Το αντικείμενο μπορεί να μην εκπέμπει λευκό φως και για αυτό να έχει διαφορετικές εντάσεις για τα τρία βασικά χρώματα. Ο σωστός ορισμός είναι :

$$I^{red} = k_i^{red}, I^{green} = k_i^{green}, I^{blue} = k_i^{blue}$$

Εφόσον η εξίσωση του φωτισμού για όλα τα εφέ θα παραμείνει ίδια για όλα τα βασικά χρώματα για κάθε εφέ θα παρουσιάζεται μόνο μία εξίσωση για τον υπολογισμό της έντασης του χρώματος. Προφανώς οι υπολογισμοί θα πρέπει να γίνουν και για τα τρία βασικά χρώματα γιατί μπορεί να οδηγήσουν σε διαφορετικές εντάσεις όταν το αντικείμενο δεν εκπέμπει λευκό φως αλλά φως διαφορετικού χρώματος.

Ένα αντικείμενο που εκπέμπει φως δεν θεωρείται πηγή φωτός, εκπέμπει φως ορατό μόνο από τον παρατηρητή. Τα αντικείμενα που εκπέμπουν φως πρέπει να συνδυάζονται με τις πηγές φωτός για παράδειγμα μία λάμπα μοντελοποιείται από ένα point light και από ένα αντικείμενο το οποίο εκπέμπει φως. Το point light φωτίζει την σκηνή αλλά το ίδιο παραμένει αόρατο, ενώ το αντικείμενο δεν φωτίζει άλλα αντικείμενα αλλά εκπέμπει φως για τον παρατηρητή και κάνει την λάμπα (το point light) ορατή. Η ένταση του εκπεμπόμενου φωτός στην επιφάνεια είναι σταθερή και δεν οδηγεί σε τρισδιάστατα εφέ. Αν δεν υπάρχουν πηγές φωτός στην σκηνή αλλά μόνο αντικείμενα που εκπέμπουν φως αυτό θα οδηγήσει σε επίπεδες προβολές που φαίνονται στην εικόνα 167.



Εικονα 167 κυβος χωρις και με εφε φωτισμου και σκιασης

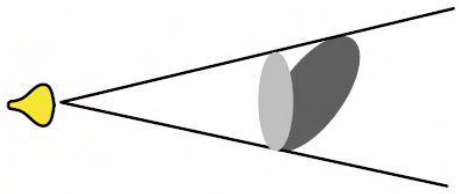
Όλα τα επόμενα εφέ φωτισμού είναι αποτελέσματα της αντανάκλασης φωτός που προέρχεται από πηγές φωτός της σκηνής. Η εξίσωση φωτισμού είναι πάντα της μορφής :

$$I = I_{light\ source} \cdot f_{pixel}$$

$I_{light\ source}$ είναι η ένταση του φωτός η οποία προέρχεται από πηγή φωτός. f_{pixel} είναι ένας παράγοντας που εξαρτάται από πολλές παραμέτρους για παράδειγμα το χρώμα της επιφάνειας, την λάμψη της, την απόσταση από την πηγή φωτός σε περίπτωση που πρέπει να συνυπολογιστεί η απόσβεση και η γωνία υπό την οποία το φως χτυπά την επιφάνεια στο συγκεκριμένο pixel. Για το φως περιβάλλοντος η εξίσωση φωτισμού είναι:

$$I = k_a \cdot I_s$$

Όπου I_s είναι η ένταση του φωτός περιβάλλοντος και k_a είναι ο παράγοντας αντανάκλασης φωτός περιβάλλοντος της επιφάνειας. Όπως και το φως που εκπέμπουν τα αντικείμενα έτσι και το φως περιβάλλοντος δεν οδηγεί σε τρισδιάστατα εφέ. Οι προβολές των αντικειμένων εμφανίζονται επίπεδες και έχουν ομογενές χρώμα. Τρισδιάστατα εφέ δημιουργούνται μόνο από πηγές φωτός που έχουν κατεύθυνση αντιθέτως με το φως περιβάλλοντος που προέρχεται από όλες τις κατευθύνσεις ή από καμία συγκεκριμένη κατεύθυνση. Μόνο αν το φως έχει κατεύθυνση μπορεί να δημιουργηθεί ένα μη ομογενές εφέ σκίασης στην επιφάνεια του αντικειμένου. Σε θαμπές επιφάνειες μία ακτίνα φωτός αντανακλάται ίσα προς όλες τις κατευθύνσεις. Το ποσό του φωτός που αντανακλάται εξαρτάται από την ένταση του φωτός, τον παράγοντα αντανάκλασης φωτός της επιφανείας και την γωνία υπό την οποία το φως χτυπά την επιφάνεια.



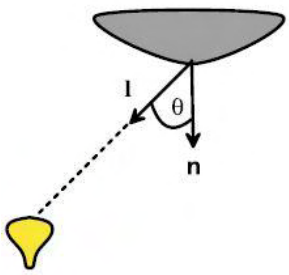
Εικονα 168 η ενταση του φωτος εξαρταται απο την γωνια που πεφτει το φως στο αντικειμενο

Ολόκληρη η ενέργεια της πηγής φωτός φτάνει τον κύκλο άσχετα αν ο κύκλος είναι κάθετος στον άξονα του κώνου του φωτός ή αν παρουσιάζει κάποια κλίση. Αλλά η περιοχή με κλίση είναι μεγαλύτερη από την κάθετη περιοχή. Αυτό σημαίνει ότι η περιοχή με την κλίση λαμβάνει λιγότερη ενέργεια ανά σημείο (ή ανά pixel) από την κάθετη. Όσο μεγαλύτερη η κλίση της περιοχής τόσο λιγότερη ενέργεια λαμβάνει ανά σημείο.

Το εφέ της αντανάκλασης του φωτός για εντελώς θαμπές επιφάνειες μπορεί να υπολογιστεί σύμφωνα με τον νόμο συνημίτονου του Lambert για την εξίσωση φωτισμού :

$$I = I_L \cdot k_d \cdot cos\theta$$

Όπου I_L είναι η ένταση του φωτός που χτυπά την επιφάνεια, $0 \leq k_d \leq 1$ είναι η παράμετρος αντανάκλασης της επιφάνειας ή του υλικού (material), θ η γωνία μεταξύ του normal vector n της επιφάνειας στο συγκεκριμένο σημείο και το διάνυσμα I που δείχνει στην κατεύθυνση από όπου έρχεται το φως. Αυτού του είδους η αντανάκλαση σε θαμπές επιφάνειες καλείται διάχυτη ανάκλαση (diffuse reflection).



Εικονα 169 diffuse reflection

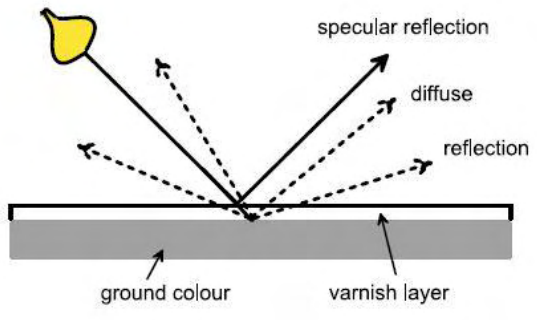
Η εξίσωση φωτισμού για την διάχυτη ανάκλαση είναι έγκυρη μόνο για γωνίες θ μεταξύ 0 και 90 μοιρών. Σε αντίθετη περίπτωση η ακτίνα φωτός χτυπάει την επιφάνεια από την πίσω μεριά έτσι δεν συμβαίνει αντανάκλαση. Στην περίπτωση κατευθυνόμενου φωτός που προέρχεται από μία πηγή φωτός που βρίσκεται σε άπειρη απόσταση η μεταβλητή I_L στην παραπάνω εξίσωση φωτός έχει την ίδια τιμή παντού. Στην περίπτωση ενός point light I_L είναι η ένταση της πηγής φωτός πολλαπλασιασμένη με τον παράγοντα απόσβεσης f_{att} ο οποίος εξαρτάται από την απόσταση του σημείου της επιφανείας από την πηγή φωτός. Για ένα spotlight εκτός από την απόσβεση πρέπει να λάβουμε υπόψη και τον παράγοντα που καθορίζει την εστίαση.

Η εξίσωση φωτισμού πρέπει να υπολογιστεί για κάθε pixel βέβαια υπάρχουν τεχνικές που χρησιμοποιούν διάφορες τεχνικές προσέγγισης υπολογίζοντας μόνο τις τιμές για μερικά pixel , συνήθως τα pixel που αντιστοιχούν στις κορυφές των πολυγώνων των επιφανειών. Για τα υπόλοιπα pixel χρησιμοποιείται η μέθοδος της παρεμβολής. Ακόμα όμως και για τον υπολογισμό των τιμών των pixel στις κορυφές απαιτείται μεγάλο υπολογιστικό κόστος. Έτσι το cosθ στην εξίσωση φωτισμού Lambert αντικαθίσταται από το εσωτερικό γινόμενο του normal vector στην επιφάνεια με το διάνυσμα I που δείχνει στην κατεύθυνση από την οποία έρχεται το φως. Και τα δύο διανύσματα πρέπει να είναι κανονικοποιημένα. Η εξίσωση φωτισμού του Lambert γίνεται :

$$I = I_L \cdot k_d \cdot (n^T \cdot I)$$

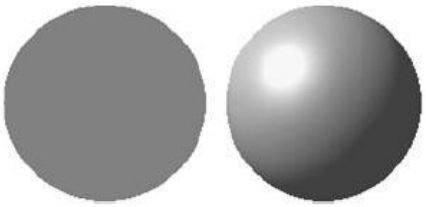
Στην περίπτωση της κατευθυνόμενης πηγής φωτός και μίας επίπεδης επιφάνειας τα διανύσματα I και n παραμένουν σταθερά στην επιφάνεια. Η επιφάνεια θα σκιαστεί ομοιογενώς με το ίδιο χρώμα. Στην εικόνα 167 μπορούμε να παρατηρήσουμε ότι ο δεξιά κύβος που φωτίζεται από κατευθυνόμενη πηγή φωτός έχει σκιαστεί διαφορετικά στις πλευρές του εφόσον το φως τις χτυπάει με διαφορετικές γωνίες. Αλλά η σκίαση σε κάθε πλευρά ξεχωριστά είναι σταθερή.

Η διάχυτη ανάκλαση σε θαμπές επιφάνειες αντανακλά το φως προς όλες τις κατευθύνσεις. Η κατοπτρική ανάκλαση (specular reflection) συμβαίνει μόνο σε γυαλιστερές (στιλπνές) επιφάνειες. Τέτοιες γυαλιστερές επιφάνειες αντανακλούν τουλάχιστον ένα μέρος του φωτός με παρόμοιο τρόπο με τους καθρέφτες. Σε αντίθεση με την διάχυτη ανάκλαση η ιδανική κατοπτρική ανάκλαση γίνεται μόνο προς μία κατεύθυνση. Το διάνυσμα που δείχνει την πηγή του φωτός αντανακλάται στο normal vector της επιφάνειας. Το διάνυσμα που δείχνει την πηγή του φωτός και το διάνυσμα που δείχνει την κατεύθυνση της αντανάκλασης έχουν την ίδια γωνία με το normal vector της επιφάνειας.



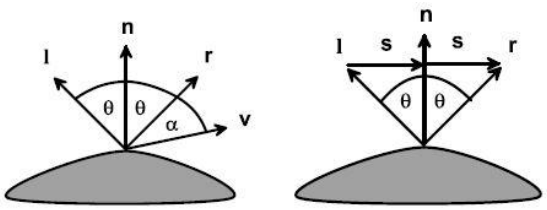
ΕΙΚΟΝΑ 170 ΔΙΑΧΥΤΗ ΚΑΙ ΚΑΤΟΠΤΡΙΚΗ ΑΝΑΚΛΑΣΗ

Οι γυαλιστερές επιφάνειες έχουν συχνά ένα πολύ λεπτό διαφανές στρώμα για παράδειγμα βερνίκι. Όταν το φως χτυπά την επιφάνεια μέρος του φωτός εισχωρεί μέσα στο στρώμα του βερνικιού και αντανακλάται πάνω στην θαμπή επιφάνεια του αντικειμένου. Αυτό το μέρος του φωτός υπόκειται διάχυτη ανάκλαση (diffuse reflection) και το χρώμα του ανακλώμενου φωτός εξαρτάται κατά κύριο λόγο από το χρώμα του δαπέδου της θαμπής επιφάνειας όπως φαίνεται στην εικόνα 170. Ένα άλλο μέρος του φωτός αντανακλάται κατευθείαν από το διαφανές στρώμα με κατοπτρική ανάκλαση. Έτσι η κατοπτρική ανάκλαση δεν αλλάζει συνήθως το χρώμα του φωτός. Αυτό μπορεί να φανεί και στην εικόνα 171 όπου η σφαίρα στα δεξιά έχει ένα άσπρο γυαλιστερό σημείο πάνω στην επιφάνειά της ενώ το χρώμα του αντικειμένου είναι γκρι. Το γυαλιστερό λευκό σημείο δημιουργείται λόγω της κατοπτρικής ανάκλασης. Η σκίαση που οφείλεται στην διάχυτη ανάκλαση εξαρτάται μόνο από την γωνία που το φως χτυπά την επιφάνεια στον παράγοντα αντανάκλασης.



ΕΙΚΟΝΑ 171 ΣΦΑΙΡΑ ΧΩΡΙΣ ΚΑΙ ΜΕ ΕΦΕ ΦΩΤΙΣΜΟΥ ΚΑΙ ΣΚΙΑΣΗΣ

Η θέση του παρατηρητή δεν έχει σημασία όταν υπολογίζεις την διάχυτη ανάκλαση. Αν ή που μπορεί να δει ο παρατηρητής την κατοπτρική ανάκλαση εξαρτάται από την θέση του. Στην περίπτωση μίας επίπεδης επιφάνειας που φωτίζεται από μία μόνο πηγή φωτός θα υπάρχει ακριβώς ένα σημείο στην επιφάνεια όπου ο παρατηρητής θα μπορεί να δει το αποτέλεσμα της ιδανικής κατοπτρικής ανάκλασης. Αυτό βέβαια είναι αληθές μόνο για τέλειους καθρέφτες. Για επιφάνειες που είναι λίγο ή πολύ γυαλιστερές , το διάνυσμα του φωτός αντανακλάται ανώμαλα στην κατεύθυνση γύρω από την κατεύθυνση της ιδανικής κατοπτρικής ανάκλασης. Με αυτόν τον τρόπο μία κυκλική πιο φωτεινή περιοχή δημιουργείται πάνω στην επιφάνεια εκτός από ένα φωτεινό σημείο στο οποίο γίνεται ιδανική κατοπτρική ανάκλαση.



ΕΙΚΟΝΑ 172 ΥΠΟΛΟΓΙΣΜΟΣ ΙΔΑΝΙΚΗΣ ΚΑΤΟΠΤΡΙΚΗΣ ΑΝΑΚΛΑΣΗΣ

Για τον υπολογισμό του μοντέλου για την ιδανική κατοπτρική ανάκλαση όπως φαίνεται στην εικόνα 172 έχουμε : το διάνυσμα I είναι το διάνυσμα που δείχνει την κατεύθυνση στην οποία το φως χτυπά την επιφάνεια, n είναι το normal vector της επιφάνειας σε αυτό το σημείο, v είναι το διάνυσμα που δείχνει την κατεύθυνση του παρατηρητή δηλαδή στην κατεύθυνση της προβολής, r είναι το διάνυσμα που δείχνει την κατεύθυνση της ιδανικής κατοπτρικής ανάκλασης. Το normal vector n σχηματίζει την ίδια γωνία θ με τα διανύσματα I και r. Για το λόγο αυτό χρησιμοποιούμε τα βοηθητικά διανύσματα s που φαίνονται στο δεξιά σχήμα της εικόνας 172. Το διάνυσμα s είναι η προβολή του I στο n. Εφόσον τα n και I είναι κανονικοποιημένα το διάνυσμα της προβολής είναι s=n·cosθ . Το διάνυσμα r πρέπει να ικανοποιεί την εξίσωση :

$$r = n \cdot cos\theta + s$$

Τα βοηθητικά διανύσματα s μπορούν να προσδιοριστούν από το διάνυσμα I και από την προβολή του I στο n:

$$s = n \cdot cos\theta - I$$

Αντικαθιστώντας το s στην εξίσωση $r = n \cdot cos\theta + s$ έχουμε :

$$r = 2 \cdot n \cdot cos\theta - 1$$

Όπως και στην περίπτωση της διάχυτης ανάκλασης το εσωτερικό γινόμενο μπορεί να υπολογιστεί από την γωνία που σχηματίζουν τα διανύσματα n και I : $n^T \cdot I = cos\theta$.Έτσι το κανονικοποιημένο διάνυσμα r στην κατεύθυνση της ιδανικής κατοπτρικής ανάκλασης είναι :

$$r = 2 \cdot n \cdot (n^T \cdot I) - 1$$

Υποθέτουμε ότι η επιφάνεια δεν φωτίζεται από την πίσω πλευρά , δηλαδή πρέπει να ισχύει ότι 0≤θ≤90. Αυτό είναι αληθές μόνο όταν το εσωτερικό γινόμενο $n^T \cdot I$ είναι θετικό.

Μόνο για ιδανικό καθρέφτη η κατοπτρική ανάκλαση θα έχει την ίδια ακριβώς κατεύθυνση με αυτή του διανύσματος r. Για τις περισσότερες γυαλιστερές επιφάνειες, η κατοπτρική ανάκλαση μπορεί να είναι ορατή γύρω από την κατεύθυνση της ιδανικής κατοπτρικής ανάκλασης. Όσο πιο πολύ ο παρατηρητής αποκλίνει από την κατεύθυνση της ιδανικής κατοπτρικής ανάκλασης τόσο λιγότερο ορατό είναι το εφέ της μη ιδανικής κατοπτρικής ανάκλασης. Το μοντέλο φωτισμού του Phong λαμβάνει αυτό το γεγονός υπόψη μειώνοντας την ένταση της κατοπτρικής ανάκλασης βάση της γωνίας α όπως φαίνεται στην εικόνα 172 στο αριστερά σχήμα. Η ένταση της κατοπτρικής ανάκλασης μειώνεται όσο αυξάνεται η γωνία α.

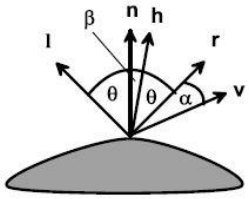
$$I = I_L \cdot W(\theta) \cdot (cosa)^n$$

"Οπου I_L η ένταση του φωτός που θα μπορούσε ήδη να έχει μειωθεί λόγω του παράγοντα της απόσβεσης για ένα point light και για ένα spotlight. Για το spotlight στον προηγούμενο παράγοντα πρέπει να προστεθεί ο παράγοντας απόκλισης από τον άξονα του κώνου του φωτός. Η τιμή

0≤W(θ)≤1 είναι το ποσοστό του φωτός το οποίο αντανακλάται κατευθείαν στις γυαλιστερές επιφάνειες για παράδειγμα το ποσοστό του φωτός στο οποίο γίνεται κατοπτρική ανάκλαση. Στις περισσότερες περιπτώσεις θα είναι W(θ)=k_{sr} όπου k_{sr} ο σταθερός παράγοντας κατοπτρικής ανάκλασης της επιφάνειας που είναι ανεξάρτητος από την γωνία θ. n είναι ο εκθέτης κατοπτρικής ανάκλασης της επιφάνειας. Για έναν τέλειο καθρέφτη $n = \infty$. Ένα μικρότερο n οδηγεί σε μία λιγότερο εστιασμένη κατοπτρική ανάκλαση. Για ένα n = 64 η κατοπτρική ανάκλαση μπορεί να είναι ορατή μόνο πολύ κοντά στην κατεύθυνση της ιδανικής κατοπτρικής ανάκλασης. Για ένα n = 1 η περιοχή που θα είναι ορατή η κατοπτρική ανάκλαση θα είναι πολύ μεγαλύτερη. Όσο πιο γυαλιστερή είναι η επιφάνεια τόσο μεγαλύτερος εκθέτης κατοπτρικής ανάκλασης πρέπει να επιλεγεί.

Η τιμή του cosα στην εξίσωση του μοντέλου του Phong μπορεί να υπολογιστεί από το εσωτερικό γινόμενο των κανονικοποιημένων διανυσμάτων v και r cosα = $r^T \cdot v$. Το διάνυσμα r θα είναι κανονικοποιημένο όταν τα διανύσματα I και n είναι κανονικοποιημένα το οποίο φαίνεται από το δεξιό σχήμα της εικόνας 172.

Το μοντέλο φωτισμού του Phong δεν βασίζεται στις αρχές της φυσικής. Είναι μία ευρετική μέθοδος για να βελτιώσει τα ρεαλιστικά εφέ της κατοπτρικής ανάκλασης. Μία τροποποιημένη εκδοχή του μοντέλου φωτισμού του Phong αντικαθιστά την απόκλιση της κατεύθυνσης της προβολής από την κατεύθυνση της ιδανικής κατοπτρικής ανάκλασης , δηλαδή την γωνία α, με μία άλλη γωνία που βασίζεται στο διάνυσμα h που βρίσκεται μεταξύ της κατεύθυνσης της πηγής του φωτός και της κατεύθυνσης του παρατηρητή. Ο παρατηρητής μπορεί να δει την ιδανική κατοπτρική ανάκλαση όταν το διάνυσμα h που βρίσκεται μεταξύ των διανυσμάτων I και v συμπίπτει με το normal vector n της επιφάνειας. Το μέτρο για την απόκλιση πλέον θεωρείται η γωνία β όπως φαίνεται στην εικόνα 173.



ΕΙΚΟΝΑ 173 ΤΟ ΔΙΑΝΥΣΜΑ Η ΚΑΙ Η ΓΩΝΙΑ Β ΣΤΟ ΤΡΟΠΟΠΟΙΗΜΕΝΟ ΜΟΝΤΕΛΟ ΤΟΥ PHONG

Ο όρος cosα αντικαθίσταται από τον όρο cosβ στην εξίσωση φωτισμού του μοντέλου του Phong. Το cosβ μπορεί και αυτό να γραφεί σαν εσωτερικό γινόμενο : $cos\beta = n^T \cdot h$. Το διάνυσμα h δίνεται από τον τύπο : $h = \frac{I+v}{\|I+v\|}$. Στην περίπτωση κατευθυνόμενης πηγής φωτός και παράλληλης προβολής το διάνυσμα h δεν αλλάζει σε αντίθεση με το διάνυσμα r στο αρχικό μοντέλο του Phong .

Όλα τα παραπάνω αναφέρονται σε μία πηγή φωτός, σε ένα σημείο και για τα τρία βασικά χρώματα το κόκκινο, το πράσινο και το μπλε. Όταν υπάρχουν περισσότερες από μία πηγές φωτός και επίσης φως περιβάλλοντος $I_{ambient\ light}$ στην σκηνή, οι εντάσεις που έχουν υπολογισθεί πρέπει να προστεθούν. Σε περίπτωση που η επιφάνεια εκπέμπει φως πρέπει να προστεθεί και η ένταση $I_{self\ emission}$. Έτσι οδηγούμαστε στην συνολική εξίσωση φωτισμού για ένα σημείο της επιφάνειας :

$$I = I_{self\ emission} + I_{ambient\ light} \cdot k_a + \sum_j I_J \cdot f_{att} \cdot g_{cone} \cdot \left(k_d \cdot (n^T \cdot I_j) + k_{sr} \cdot (r_j^T \cdot v)^n\right)$$

Όπου k_a είναι ο παράγοντας αντανάκλασης για το φως περιβάλλοντος που είναι συνήθως ίδιος με τον παράγοντα k_d για την διάχυτη ανάκλαση. I_Jείναι η ένταση της j-οστής πηγής φωτός. Για μία κατευθυνόμενη πηγή φωτός οι δύο παράγοντες f_{att} και g_{cone} είναι ίσοι με 1. Μόνο για ένα point light ο f_{att} μοντελοποιεί την απόσβεση που εξαρτάται από την απόσβεση και για ένα spotlight ο

g_{cone} αντιστοιχεί στο πόσο γρήγορα η ένταση του φωτός μειώνεται στο όριο του κώνου που σχηματίζει το φως. Η συνολική εξίσωση του φωτός βασίζεται στο αρχικό μοντέλο του Phong. Συνήθως ο παράγοντας k_{sr} της κατοπτρικής ανάκλασης δεν συμπίπτει με τους παράγοντες k_a και k_d.

Η ένταση I έχει όριο την μονάδα. Αν το άθροισμα υπερβεί την μονάδα τότε η ένταση περικόπτεται στην μονάδα. Ο υπολογισμός της συνολικής εξίσωσης του φωτός για τα χρώματα κόκκινο, πράσινο και μπλε δεν απαιτεί επιπρόσθετο υπολογιστικό κόστος εφόσον οι συντελεστές $I_{self\ emission}, I_{ambient\ light}, k_a, I_J, k_d, k_{sr}$ μπορεί να είναι διαφορετικοί για κάθε χρώμα αλλά δίνονται και δεν χρειάζεται να υπολογιστούν από κάποιον αλγόριθμο. Οι άλλοι συντελεστές χρειάζονται πιο πολύπλοκους υπολογισμούς. Είναι ίδιοι για κάθε χρώμα αλλά εξαρτώνται από το επιλεγμένο σημείο της επιφάνειας και τις ιδιότητες της πηγής του φωτός.

12.2.9 Υλικο – Material

Στην Java 3D μία εμφάνιση (Appearance) αντιστοιχίζεται σε κάθε αντικείμενο για να καθορίσει τις ιδιότητες της επιφάνειας του αντικείμενου. Ένα πολύ σημαντικό χαρακτηριστικό της κλάσης Appearance είναι το υλικό (Material) το ποίο καθορίζει την εμφάνιση ενός αντικειμένου όταν βρίσκεται υπό φωτισμό. Αν δεν έχει οριστεί ένα Material για την Appearance τότε δεν μπορεί να φωτιστεί το αντικείμενο με την συγκεκριμένη Appearance παρόλο την ύπαρξη πηγών φωτός στον εικονικό κόσμο. Οι ιδιότητες του Material όπως επίσης και της επιφάνειας που εφαρμόζεται ορίζονται στην εντολή :

Material mat = new Material(ambientColor, emissiveColor, diffuseColor, specularColor, shininessValue)

Και οι 4 παράμετροι είναι χρώματα που ορίζονται από αντικείμενα της κλάσης Color3f. Η παράμετρος ambientColor ορίζει το πόσο φως περιβάλλοντος αντανακλάται από το υλικό. Η τιμή της κυμαίνεται από 0.0 μέχρι 1.0. Η προεπιλεγμένη τιμή του είναι (0.2, 0.2, 0.2). Η παράμετρος emissiveColor αναφέρεται στο RGB χρώμα που εκπέμπεται από το υλικό αν εκπέμπεται κάποιο χρώμα. Η τιμή του κυμαίνεται από 0.0 μέχρι 1.0 και η προεπιλεγμένη τιμή του είναι (0.0, 0.0, 0.0). Η παράμετρος diffuseColor αναφέρεται στο RGB χρώμα του υλικού όταν η επιφάνεια που φέρει το υλικό είναι φωτισμένη. Η τιμή του κυμαίνεται από 0.0 μέχρι 1.0 και η προεπιλεγμένη τιμή του είναι (1.0, 1.0, 1.0). Η παράμετρος specularColor αναφέρεται στο RGB χρώμα του υλικού που κάνει την επιφάνεια γυαλιστερή. Η τιμή του κυμαίνεται από 0.0 μέχρι 1.0 και η προεπιλεγμένη τιμή του είναι (1.0, 1.0, 1.0). Η παράμετρος shininessValue αναφέρεται στην λάμψη του υλικού και παίρνει τιμές από 1.0 μέχρι 128.0 με την ελάχιστη λάμψη να αντιστοιχεί στην τιμή 1.0 και την μέγιστη στην τιμή 128.0.

Στην εφαρμογή η σφαίρα φέρει μία εμφάνιση Appearance που έχει δημιουργηθεί βάση της κλάση Material με τις παρακάτω παραμέτρους :

Material mat = new Material()

mat.setAmbientColor(0.0f, 0.4f,0.0f)

mat.setEmissiveColor(0.0f, 0.4f,0.0f)

mat.setDiffuseColor(0.0f,0.4f,0.0f,1.0f)

mat.setSpecularColor(0.4f, 0.4f, 0.4f)

mat.setShininess(104.0f)

Η μόνη μεταβλητή που είναι διαφοροποιημένη είναι του χρώματος πράσινου για να δημιουργήσουμε το πράσινο χρώμα που έχει η σφαίρα. Επίσης αυτή η μεταβλητή είναι η μεταβλητή που αλλάζει για να δούμε τις διαφορετικές αντανακλάσεις του φωτός. Για να χαρακτηρίσουμε την εμφάνιση με το συγκεκριμένη υλικό γράφουμε :

Appearance greenApp = new Appearance()

greenApp.setMaterial(mat)

Για να μπορέσουμε να διαβάσουμε αλλά και να αλλάξουμε τις τιμές των παραμέτρων του υλικού πρέπει να αναθέσουμε στο υλικό κάποιες «ικανότητες» (capabilities).

mat.setCapability(Material.ALLOW_COMPONENT_WRITE)

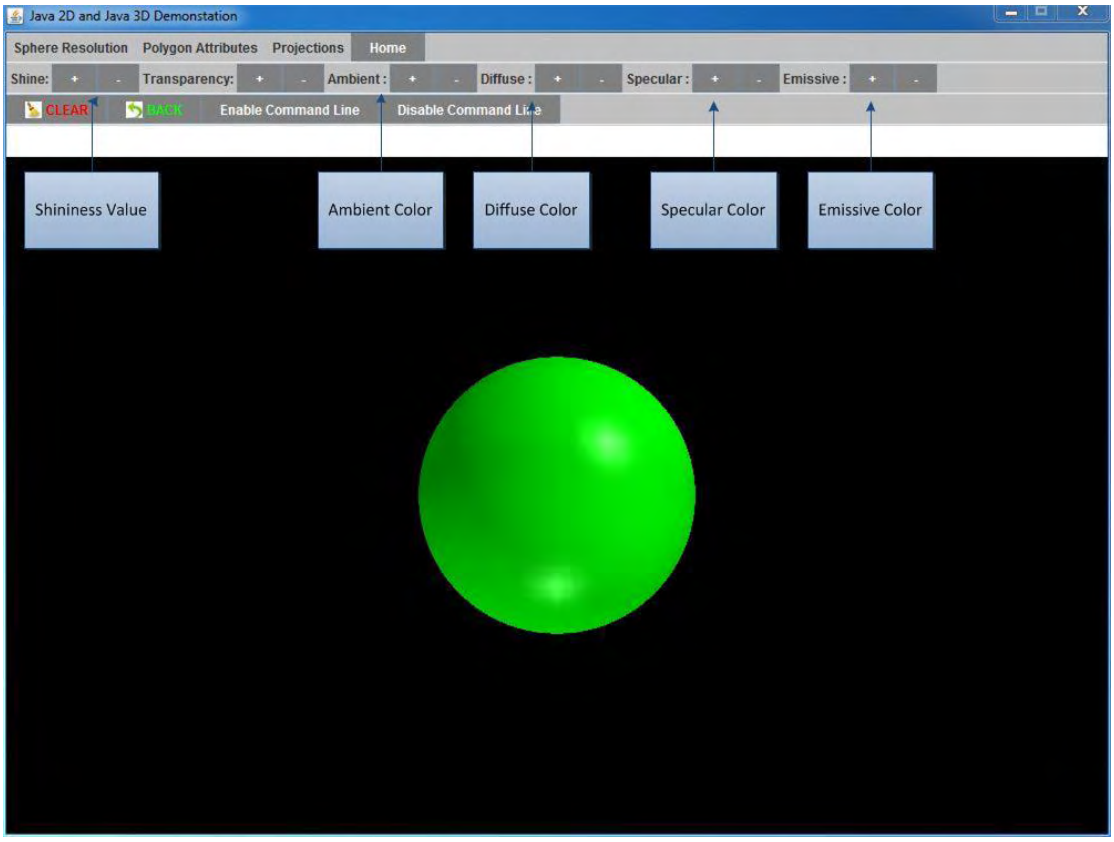
mat.setCapability(Material.AMBIENT)

mat.setCapability(Material.EMISSIVE)

mat.setCapability(Material.DIFFUSE)

mat.setCapability(Material.SPECULAR)

Στην εφαρμογή ο χρήστης μπορεί να αλλάξει τις παραμέτρους του υλικού και να δει τις διαφορές που επιφέρουν αυτές οι αλλαγές στον φωτισμό της σφαίρας. Για να αυξομειώσουμε τις παραμέτρους : emissiveColor, diffuseColor, specularColor, shininessValue χρησιμοποιούμε τα κουμπιά «+», «-» που βρίσκονται δίπλα από την αντίστοιχη ετικέτα.



ΕΙΚΟΝΑ 174 ΠΑΡΑΜΕΤΡΟΙ ΓΙΑ ΤΟ ΥΛΙΚΟ (MATERIAL)

12.2.10 ΔΙΑΦΑΝΕΙΑ – TRANSPARENCY

Οι διαφανείς επιφάνειες αντανακλούν ένα μέρος του φωτός αλλά τα αντικείμενα που βρίσκονται από πίσω τους μπορούν να είναι πλήρως ορατά. Ένα τυπικό διαφανές αντικείμενο είναι ένα καθαρό γυαλί. Με τον όρο διαφάνεια εννοούμε ότι μόνο ένα μέρος του φωτός των αντικειμένων πίσω από την διάφανη επιφάνεια μπορεί να περάσει μέσα από την διάφανη επιφάνεια, χωρίς να συμβαίνει καμία στρέβλωση όπως συμβαίνει με το παγωμένο γυαλί. Δεν θα μιλήσουμε για τα ημιδιαφανή αντικείμενα (ημιδιαφανή αντικείμενα ονομάζονται αυτά τα αντικείμενα που επιτρέπουν μέρος του φωτός να περνάει από μέσα τους αλλά με μεγάλη στρέβλωση και διάθλαση π.χ. ένα φιμέ τζάμι) και επίσης το φαινόμενο της διάθλασης δεν θα συμπεριληφθεί στους υπολογισμούς.

Έστω μία επιφάνεια F2 τοποθετημένη πίσω από μία διαφανής επιφάνεια F1. Για την τεχνική της παρεμβάλλουσας ή φιλτραρισμένης διαφάνειας χρειάζεται ένας συντελεστής μετάδοσης ο k_{trans} ∈ [0,1]. Ο συντελεστής αυτός προσδιορίζει το μέρος του φωτός που μπορεί να περάσει μέσα από την διάφανη επιφάνεια F1. Η επιφάνεια είναι εντελώς διαφανής δηλαδή αόρατη όταν $k_{trans} = 1$. Για $k_{trans} = 0$ η επιφάνεια είναι αδιαφανής και μπορούμε να την αντιμετωπίσουμε όπως όλες τις υπόλοιπες επιφάνειες για τις οποίες έχουμε μιλήσει. Η ένταση του χρώματος I_p ενός σημείου P της διαφανούς επιφάνειας F1 καθορίζεται από την σχέση :

$$I_p = (1 - k_{trans}) \cdot I_1 + k_{trans} \cdot I_2$$

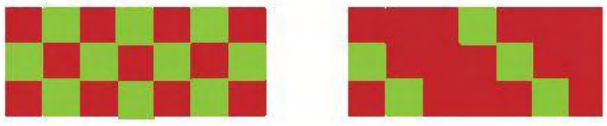
Όπου I_1είναι η ένταση που θα αντιστοιχούσε στο σημείο P αν η επιφάνεια F1 ήταν αδιαφανής. I_2 είναι η ένταση του αντίστοιχου σημείου της επιφάνειας F2 όταν η επιφάνεια F1 είναι εντελώς αόρατη ή έχει αφαιρεθεί από τη σκηνή. Οι τιμές για το κόκκινο, το πράσινο και το μπλε χρώμα προκύπτουν από το χρώμα που ανατίθεται στην διάφανη επιφάνεια. Με αυτόν τον τρόπο είναι επίσης εφικτό να μοντελοποιηθούν χρωματιστές διαφανείς επιφάνειες.

Γενικά οι διαφανείς επιφάνειες περιπλέκουν την διαδικασία επιλογής των ορατών επιφανειών. Ειδικότερα όταν ο z-buffer αλγόριθμος χρησιμοποιείται μπορούν να συμβούν τα ακόλουθα προβλήματα :

- **Ποια z τιμή θα πρέπει να αποθηκευτεί στον z-buffer όταν μία διάφανη επιφάνεια προβάλλεται.** Αν είναι αποθηκευμένη στον z-buffer μία τιμή z ενός αντικειμένου Ο που βρίσκεται πίσω από τη διάφανη επιφάνεια τότε ένα αντικείμενο μεταξύ του αντικειμένου Ο και της διάφανης επιφάνειας θα υπερισχύσει του Ο αντικειμένου και θα υπερισχύσει στον frame buffer άσχετα αν βρίσκεται πίσω από την διάφανη επιφάνεια. Αν αντιθέτως η z τιμή της διάφανης επιφάνειας είναι αποθηκευμένη στον z-buffer τότε το αντικείμενο Ο δεν μπορεί να εμφανιστεί στον frame buffer αν και θα έπρεπε να είναι ορατό πίσω από την διάφανη επιφάνεια.
- **Ποια τιμή πρέπει να μπει στον frame buffer.** Αν υπολογίζεται η παρεμβάλλουσα διαφάνεια από την σχέση $I_p = (1 - k_{trans}) \cdot I_1 + k_{trans} \cdot I_2$, η πληροφορία για την τιμή I_1 χάνεται για τα αντικείμενα που μπορεί να βρίσκονται πίσω από την διάφανη επιφάνεια. Ακόμα η τιμή I_1 μπορεί να μην είναι αρκετή. Είναι πιθανό να εφαρμοστεί η τεχνική alpha – blending. Εφόσον η κωδικοποίηση των τιμών RGB απαιτούν τρία bytes και τα blocks των τεσσάρων bytes χειρίζονται πιο εύκολα από τους η/υ, είναι σύνηθες να χρησιμοποιούνται 4 bytes για μία τιμή alpha. Αυτή η τιμή alpha αντιστοιχεί στον συντελεστή μετάδοσης k_{trans} για την διαφάνεια. Αλλά ακόμα και με την τιμή alpha δεν είναι ξεκάθαρο σε ποιο αντικείμενο πίσω από την διάφανη επιφάνεια πρέπει να εφαρμοστεί η τεχνική alpha – blending, δηλαδή πώς να εφαρμοστεί η εξίσωση $I_p = (1 - k_{trans}) \cdot I_1 + k_{trans} \cdot I_2$ εφόσον η επιλογή για το αντικείμενο με την τεχνική alpha – blending εξαρτάται από την τιμή z.

Αδιαφανή αντικείμενα πρέπει να εισέρχονται πρώτα στον z-buffer και μετά οι διαφανείς επιφάνειες. Όταν εισαχθούν οι διαφανείς επιφάνειες πρέπει να εφαρμοστεί alpha – blending στον frame buffer. Θα υπάρξει πρόβλημα όταν διαφανείς επιφάνειες καλύπτουν άλλες διαφανείς επιφάνειες, σε αυτή την περίπτωση η σειρά με την οποία εισέρχονται ο επιφάνειες στον z-buffer πρέπει να είναι σωστή , δηλαδή πρώτα οι πίσω επιφάνειες και μετά οι μπροστά. Για αυτό το λόγο είναι σύνηθες να ταξινομούμε τις επιφάνειες ως προς την z συνιστώσα τους.

Μία άλλη τεχνική για την διαφάνεια είναι η Screen – door . Η μείξη ή η παρεμβολή των χρωμάτων μίας διάφανης επιφάνειας με το αντικείμενο που βρίσκεται πίσω από την επιφάνεια ορίζεται από την σχέση $I_p = (1 - k_{trans}) \cdot I_1 + k_{trans} \cdot I_2$ και εφαρμόζεται σε ομάδες από pixel και όχι σε κάθε pixel ξεχωριστά. Ο συντελεστής μετάδοσης k_{trans} = 0.25 σημαίνει ότι κάθε τέταρτο pixel παίρνει το χρώμα του από το αντικείμενο πίσω από την διάφανη επιφάνεια ενώ όλα τα υπόλοιπα pixel παίρνουν το χρώμα τους από την διαφανή επιφάνεια. Η εικόνα 175 δείχνει την τεχνική Screen – door για μερικά pixel. Το πράσινο χρώμα αντιστοιχεί στο αντικείμενο πίσω από την διάφανη επιφάνεια ενώ το κόκκινο χρώμα στην διάφανη επιφάνεια. Για το αριστερό σχήμα ο k_{trans} =0.5 ενώ για το δεξί σχήμα o k_{trans} =0.25.



ΕΙΚΟΝΑ 175 SCREEN - DOOR ΤΕΧΝΙΚΗ

Η τεχνική screen – door είναι κατάλληλη για τον αλγόριθμο z-buffer. Οι z τιμές επιλέγονται ανάλογα με την επιφάνεια που έρχονται, είτε από την διάφανη επιφάνεια είτε από το αντικείμενο πίσω από την διάφανη επιφάνεια. Για k_{trans} =0.25 το 75% των pixel θα έχουν την z τιμή της διάφανης επιφάνειας ενώ το 25% των pixel θα έχουν την z τιμή του αντικειμένου. Ένα αντικείμενο που θα προβληθεί αργότερα θα αντιμετωπιστεί σωστά. Αν είναι μπροστά από την διάφανη επιφάνεια θα υπερισχύσει όλων. Αν είναι πίσω από άλλο αντικείμενο στο οποίο έχει εφαρμοστεί η τεχνική screen – door δεν θα εισαχθεί στον frame buffer. Αν το αντικείμενο είναι πίσω από την διάφανη επιφάνεια και πιο κοντά από όλα τα υπόλοιπα αντικείμενα στην διάφανη επιφάνεια τα pixel που αντιστοιχούν σε κομμάτι της επιφάνειας που επικαλύπτεται θα πάρουν το χρώμα του αντικειμένου βάση του αλγορίθμου screen - door. Αν και η τεχνική screen – door λειτουργεί καλά με τον z-buffer αλγόριθμο τα αποτελέσματα είναι αποδεκτά μόνο για υψηλή ανάλυση. Για συντελεστή μετάδοσης k_{trans} = 50% τα αποτελέσματα της τεχνικής screen – door με την τεχνική της παρεμβάλλουσας διαφάνειας είναι σχεδόν ίδια. Αλλά για k_{trans} κοντά στο 1 ή στο 0 η τεχνική screen- door τείνει να εμφανίζει μοτίβα κουκκίδων αντί για ρεαλιστικά εφέ διαφάνειας.

Η Java 3D παρέχει την κλάση Transparency Attributes για την μοντελοποίηση της διαφάνειας. Ορίζουμε ένα αντικείμενο της κλάσης αυτής γράφοντας :

TransparencyAttributes ta = new TransparencyAttributes()

Για να επιλέξουμε μία από τις διαθέσιμες τεχνικές διαφάνειας : παρεμβάλλουσα ή screen - door χρησιμοποιούμε την μέθοδο set Transparency Mode() με παράμετρο TransparencyAttributes.BLENDED για την τεχνική παρεμβάλλουσας διαφάνειας ή TransparencyAttributes.SCREEN_DOOR για την τεχνική διαφάνειας screen – door. Στην εφαρμογή χρησιμοποιούμε την παρεμβάλλουσα διαφάνεια :

ta.setTransparencyMode(TransparencyAttributes.BLENDED)

Ο συντελεστής μετάδοσης k_{trans} προσδιορίζεται από την μέθοδο set Transparency() ως float τιμή μεταξύ του 0 και του 1.

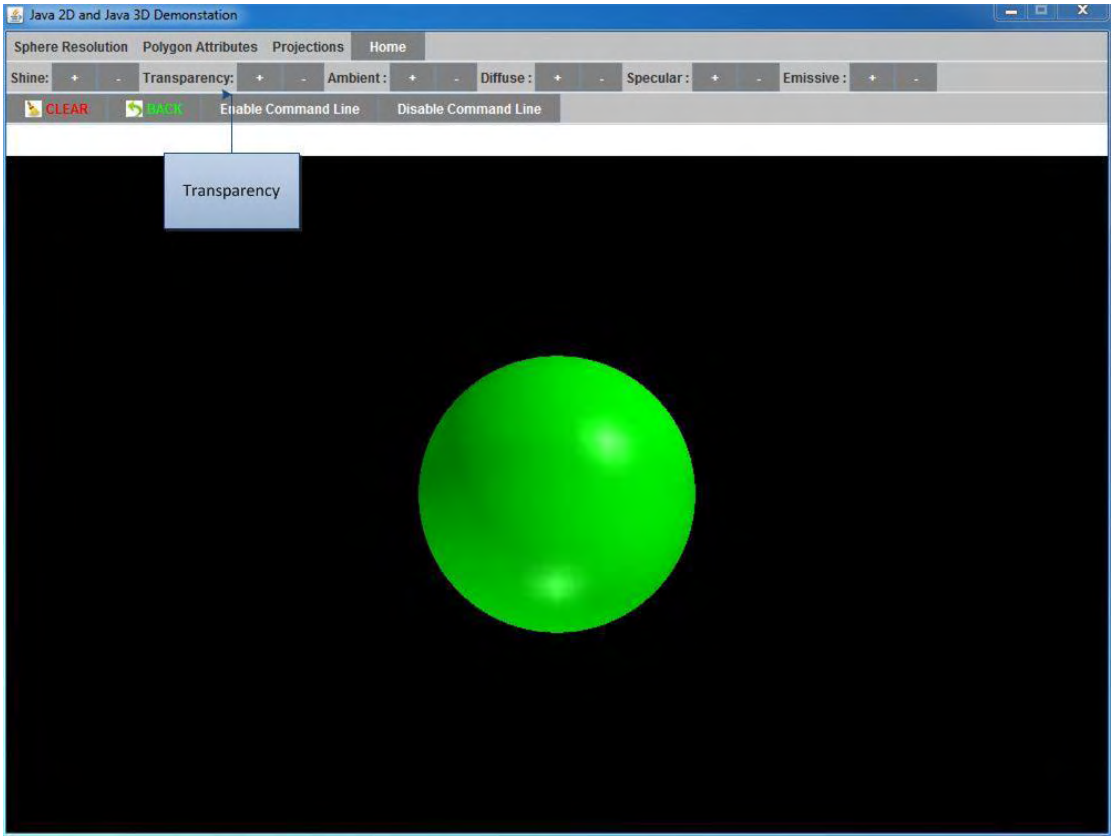
ta.setTransparency(in_trans)

Το αντικείμενο της κλάσης Transparency Attributes ανατίθεται στην εμφάνιση Appearance με την μέθοδο set Transparency Attributes().

greenApp.setTransparencyAttributes(ta) // greenApp το στιγμιότυπο της κλάσης Appearance για αυτή την επιλογή.
Για να μπορέσουμε να αλλάξουμε την τιμή του συντελεστή μετάδοσης πρέπει να αναθέσουμε στο στιγμιότυπο της κλάσης Transparency Attributes την παρακάτω «ικανότητα» (capability).

ta.setCapability(TransparencyAttributes.ALLOW_VALUE_WRITE)

Στην εφαρμογή ο χρήστης μπορεί από τα κουμπιά «+» και «-» δίπλα από την ετικέτα «Transparency» να αλλάξει τον συντελεστή μετάδοσης και να κάνει την σφαίρα αδιαφανή ή ακόμα και εντελώς διάφανη.



ΕΙΚΟΝΑ 176 ΔΙΑΦΑΝΕΙΑ

12.2.11 ΤΟ SCENE GRAPH ΤΗΣ ΕΠΙΛΟΓΗΣ

Για να παρουσιάσουμε το scene graph της επιλογής πρέπει πρώτα να μιλήσουμε για ένα νέο είδος group node το Switch node. Μέχρι τώρα έχουμε αναφερθεί σε branch groups που περιέχουν

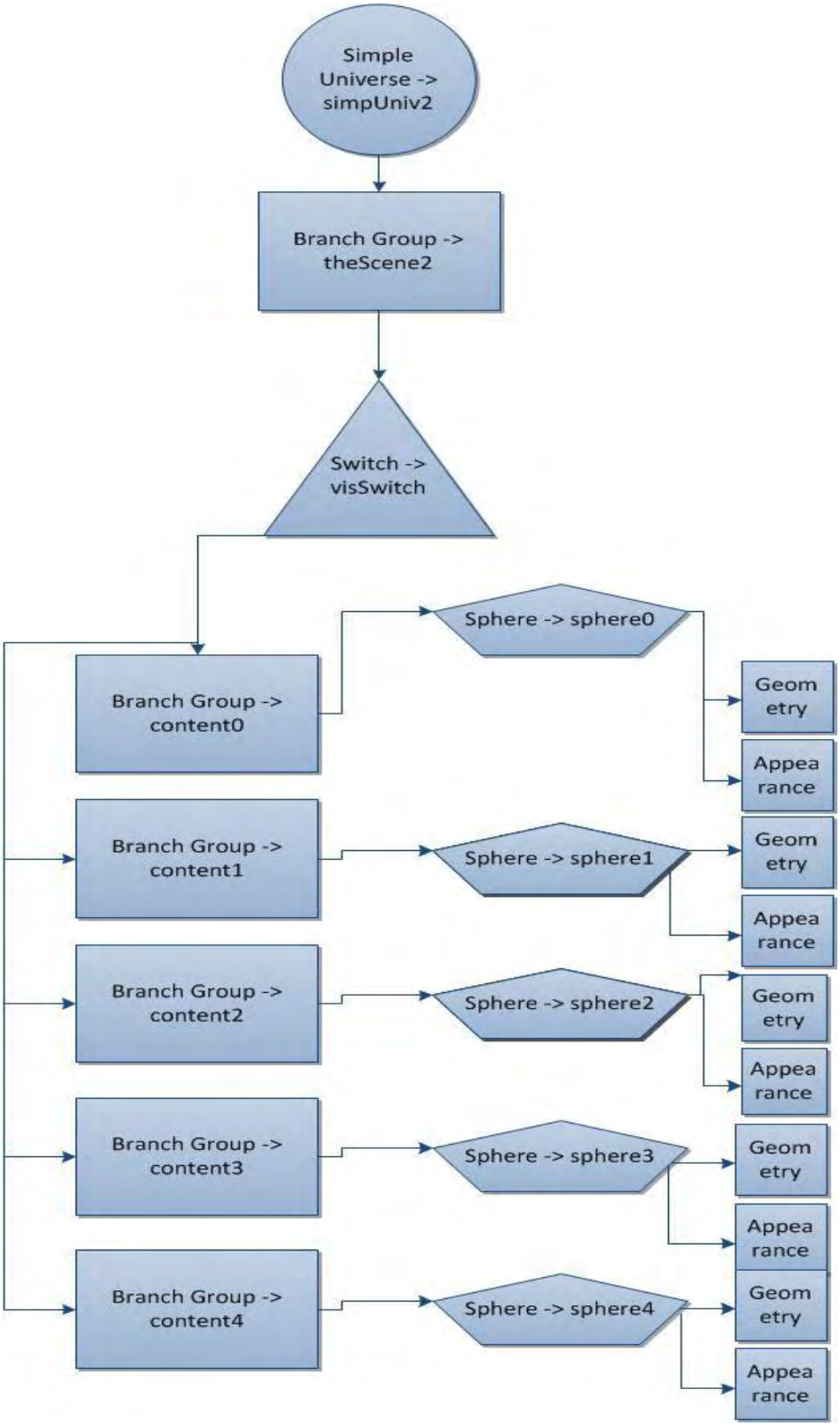
διακλαδώσεις content ή view και σε transform groups που εφαρμόζουν στα αντικείμενα που τους έχουν ανατεθεί έναν μετασχηματισμό τύπου Transform3D. Οι switch nodes επιτρέπουν στην Java 3D να επιλέγει δυναμικά μεταξύ των sub graphs. Ο κόμβος switch περιέχει μία διατεταγμένη λίστα από παιδιά και από switch τιμές. Οι switch τιμές καθορίζουν ποιο παιδί ή ποια παιδιά θα αποδοθούν από την Java 3D. Για να μπορέσει η Java 3D να αλλάξει τις τιμές που ελέγχουν ποια παιδιά θα αποδοθούν πρέπει να ορίσουμε την αντίστοιχη ικανότητα (capability) στο switch node.

Switch visSwitch = new Switch() // δημιουργία switch κόμβου

visSwitch.setCapability(Switch.ALLOW_SWITCH_WRITE)

Με την μέθοδο add Child() εισάγουμε τους κόμβους που θέλουμε ως παιδιά στο switch node. Με την μέθοδο set Which Child() ορίζουμε ποιο παιδί θα αποδοθεί αν εισάγουμε ως παράμετρο έναν μη αρνητικό ακέραιο που αντιστοιχεί στο παιδί που προστέθηκε στον switch node με εκείνον τον αριθμό σειράς στη λίστα (το πρώτο παιδί έχει την τιμή 0, το δεύτερο την τιμή 1 κτλ), αν εισάγουμε την τιμή CHILD_NONE δεν θα αποδοθεί κάποιο παιδί και αν εισάγουμε την τιμή CHILD_ALL θα αποδοθούν όλα τα παιδιά. Ο visSwitch κόμβος έχει παιδιά τις σφαίρες που έχουν δημιουργηθεί με τις 5 διαφορετικές αναλύσεις και ο λόγος που προτιμήθηκε είναι να ότι μπορούμε να εναλλάσσουμε τις σφαίρες που είναι ήδη μέρος ενός live scene graph πολύ εύκολα.

Με κύκλο έχει σχεδιαστεί το αντικείμενο Simple Universe ενώ με ορθογώνια έχουν σχεδιαστεί οι κόμβοι τύπου Branch Group. Με τρίγωνο έχει σχεδιαστεί ο κόμβος Switch ενώ με πεντάγωνα τα φύλλα που είναι αντικείμενα Sphere. Με τετράγωνα έχουν σχεδιαστεί τα node components που αναφέρονται στα φύλλα του δέντρου .



Εικονα 177 scene graph app3dPanel

2.2.12 Clear and Back– Επαναφορα και Επιστροφη

CLEAR

Με τον όρο «clear» στην επιλογή αυτή εννοούμε να μεταφέρουμε την σφαίρα στην αρχική της θέση (μεταφέροντας την προβολή στην αρχική της θέση) και ψηφιοποίηση να επαναφέρουμε την προβολή σε προοπτική και τις παραμέτρους του υλικού στις αρχικές τους τιμές. Για να μεταφέρουμε την προβολή στην αρχική της θέση χρησιμοποιούμε τις ίδιες εντολές όπως και στο κουμπί «Home» που αναλύθηκε στην παράγραφο 12.2.2 [12.2.2]. Για την αρχική ψηφιοποίηση επιλέγουμε το παιδί του κόμβου switch (παιδί 0) που αντιστοιχεί στην προεπιλεγμένη ψηφιοποίηση (ress 4) γράφοντας :

visSwitch.setWhichChild(0)

Για να επαναφέρουμε την προβολή σε προοπτική καλούμε την μέθοδο set Projection Policy() με παράμετρο την τιμή View.PERSPECTIVE_PROJECTION όπως μελετήθηκε στην παράγραφο 12.2.1 [12.2.1] . Για να επαναφέρουμε τις παραμέτρους του υλικού στις αρχικές τους τιμές γράφουμε τις εντολές :

app3dPanel.amb = 0.4f // η παράμετρος που αλλάζει για το ambient color

app3dPanel.dif = 0.4f // η παράμετρος που αλλάζει για το diffuse color

app3dPanel.spec = 0.4f // η παράμετρος που αλλάζει για το specular color

app3dPanel.emis = 0.4f // η παράμετρος που αλλάζει για το emissive color

app3dPanel.in_trans = 0.0f // ο συντελεστής μετάβασης για την διαφάνεια

app3dPanel.in_shine = 104.0f // η παράμετρος που αλλάζει για την shininess

για να εφαρμόσουμε τις αλλαγές στις τιμές των παραμέτρων καλούμε τις αντίστοιχες μεθόδους γράφοντας :

ta.setTransparency(app3dPanel.in_trans) // για την διαφάνεια

mat.setDiffuseColor(0.0f,app3dPanel.dif,0.0f,1.0f) // για το diffuse color

mat.setSpecularColor(app3dPanel.spec,app3dPanel.spec,app3dPanel.spec) // για το specular color

mat.setAmbientColor(0.0f,app3dPanel.amb,0.0f) // για το ambient color

mat.setEmissiveColor(0.0f,app3dPanel.emis,0.0f) // για το emissive color

mat.setShininess(app3dPanel.in_shine) // για την shininess

Για να μην είναι ορατό το πλευρικό panel καλούμε την μέθοδο set Visible() με παράμετρο false για τα αντικείμενα που διαμορφώνουν το πλευρικό panel.

SMMenu.setVisible(false) // JPanel που περιέχει τα κουμπιά για το scaling

RMMenu.setVisible(false) // JPanel που περιέχει τα κουμπιά για το rotate

TMMenu.setVisible(false) // JPanel που περιέχει τα κουμπιά για το translate

Στην εφαρμογή για να επαναφέρουμε τον εικονικό κόσμο στην αρχική του κατάσταση πατάμε το κουμπί «Clear».

BACK

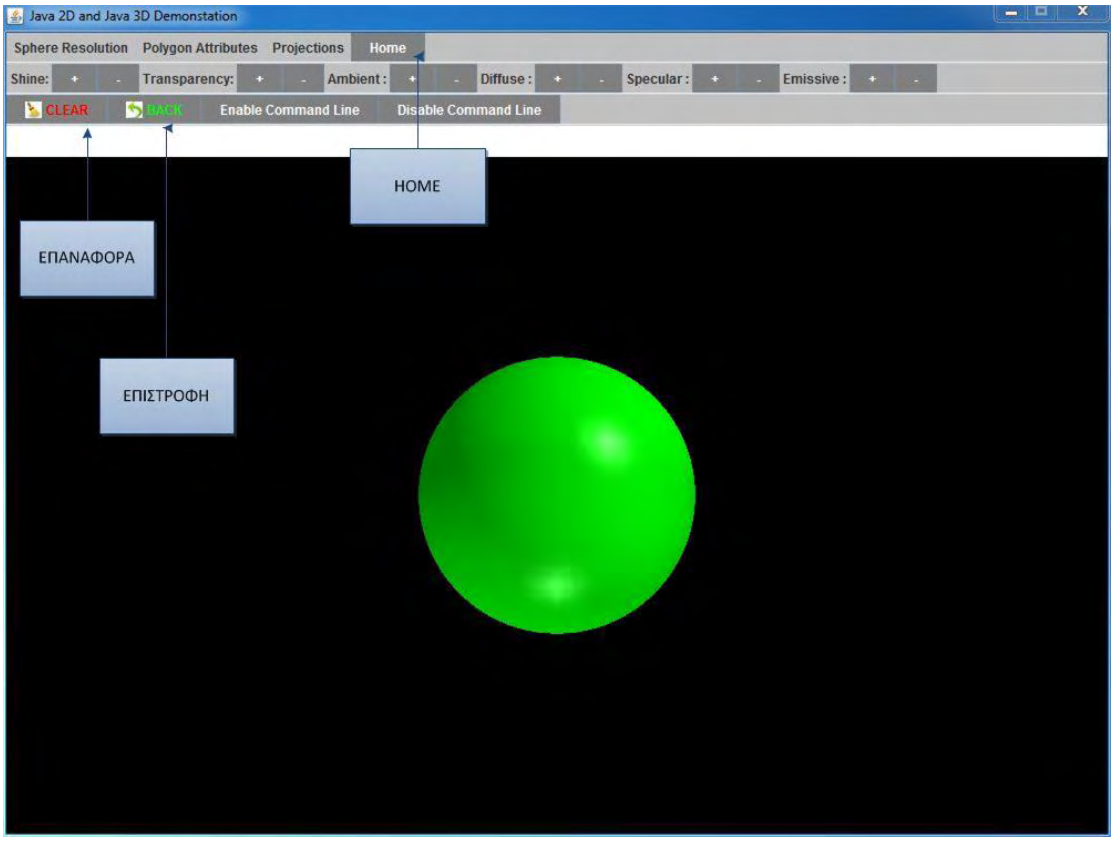
Η εφαρμογή είναι χωρισμένη σε επιλογές που επικοινωνούν μέσω του κοινού εξωφύλλου (CoverPanel). Είναι δυνατή η επιστροφή μέσα από μία εφαρμογή στο εξώφυλλο για να μας δοθεί η δυνατότητα να επιλέξουμε μία άλλη επιλογή. Για να πραγματοποιηθεί αυτό θα πρέπει όπως και στην διαδικασία της επαναφοράς (clear) να επαναφέρουμε τον εικονικό κόσμο στην αρχική του κατάσταση. Έτσι επαναφέρουμε τον εικονικό κόσμο της επιλογής στην αρχική του κατάσταση με τον κώδικα που αναφέρθηκε στο «Clear». Επίσης πρέπει να γίνει ορατό το CoverPanel της εφαρμογής και να μην είναι πλέον ορατό το panel στο οποίο βρισκόμαστε. Για την επιλογή αυτή το panel είναι το App3dPanel.

app3dPanel.setVisible(false)

setContentPane(coverPanel)

coverPanel.setVisible(true)

Στην εφαρμογή για να επιστρέψουμε στο εξώφυλλο πατάμε το κουμπί «BACK».



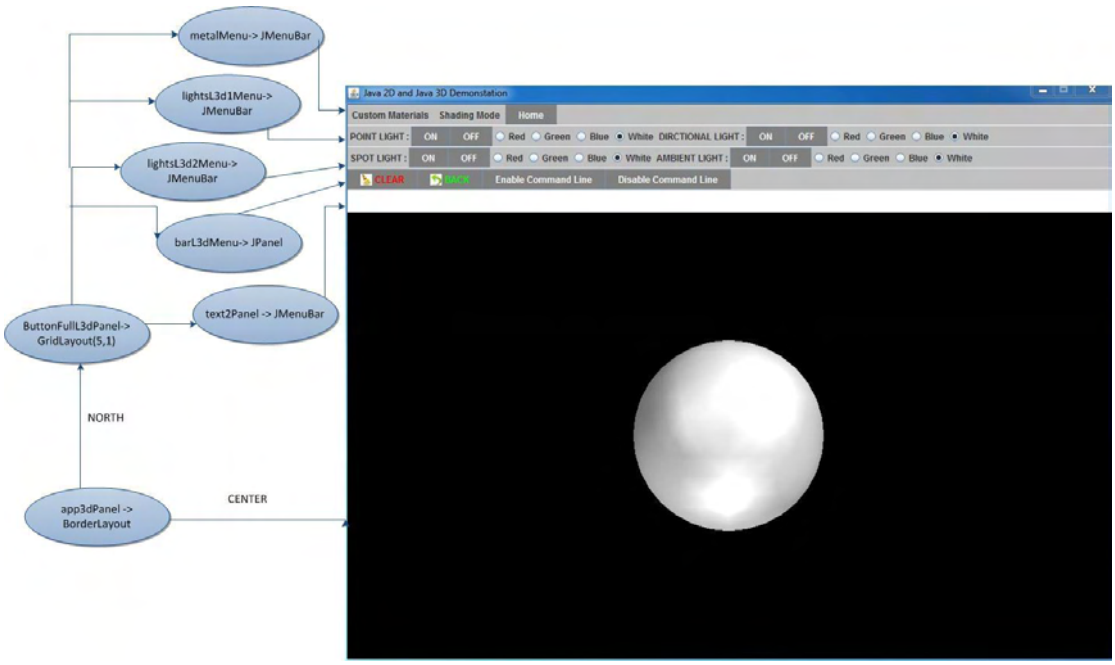
ΕΙΚΟΝΑ 178 ΚΟΥΜΠΙΑ ΓΙΑ ΕΠΑΝΑΦΟΡΑ, ΕΠΙΣΤΡΟΦΗ ΚΑΙ ΜΕΤΑΦΟΡΑ ΣΤΗΝ ΑΡΧΙΚΗ ΘΕΣΗ

13. Java 3D – 9η επιλογη (Lighting and Shading)

13.1 Παρουσιαση

Η ένατη επιλογή της εφαρμογής είναι το αντικείμενο της κλάσης Lights3dPanel. Είναι ένα panel που περιέχει Swing components όπως κουμπιά και ετικέτες. Περιέχει επίσης ένα αντικείμενο της κλάσης text2Panel που είναι ένα JPanel που αποτελεί την γραμμή εντολών. Ακόμα κύριο συστατικό αυτής της επιλογής είναι ο εικονικός κόσμος που έχει δημιουργηθεί ο οποίος περιέχει μία σφαίρα. Σκοπός της επιλογής αυτής είναι ο χρήστης να μπορεί να εναλλάξει τις πηγές φωτός που έχουν προστεθεί στον εικονικό κόσμο, σβήνοντας ή ανάβοντας τες. Μπορεί ακόμη να αλλάξει το χρώμα του φωτισμού και το υλικό της επιφάνειας της σφαίρας για να μπορέσει να διακρίνει διαφορές μεταξύ των διαφορετικών πηγών φωτός που έχουν χρησιμοποιηθεί. Ο χρήστης έχει την δυνατότητα να παρατηρήσει τις διαφορές στο εφέ που δημιουργούν δύο αλγόριθμοι σκίασης. Όπως και στην προηγούμενη επιλογή υπάρχει η δυνατότητα περιήγησης μέσα στον εικονικό κόσμο. Τέλος παρέχονται οι λειτουργίες επιστροφή στα εξώφυλλο και επαναφοράς του εικονικού κόσμου στην αρχική του κατάσταση.

Το Lights3dPanel έχει custom δομή που δημιουργείται με border Layout και grid Layout.



Εικονα 179 lights3dPanel Layout

13.2 Λειτουργικοτητα

13.2.1 Πηγες Φωτος στην Java 3D

Η παράγραφος αυτή αποτελεί συνέχεια του προηγούμενου κεφαλαίου μιας που παρουσιάζει την υλοποίηση των πηγών φωτός που μελετήθηκαν στην παράγραφο 12.2.7 [12.2.7] . Η Java 3D παρέχει κλάσεις για τις πηγές φωτός που υλοποιούνται μέσα στην μέθοδο add Light(). Η μέθοδος αυτή δημιουργεί ένα αντικείμενο της κλάσης branch group στο οποίο ανατίθενται ως

παιδιά οι πηγές φωτός που προστίθενται στην σκηνή. Το ίδιο το branch group προστίθεται στο αντικείμενο της κλάσης Simple Universe με την μέθοδο add Branch Graph().

BranchGroup bgLight = new BranchGroup()

Για κάθε πηγή φωτός ορίζεται ένα χρώμα από 3 float τιμές μεταξύ του 0 και του 1 για τις τιμές red, green, blue του RGB μοντέλου. Κάθε πηγή φωτός μπορεί να έχει διαφορετικό χρώμα. Για κάθε πηγή φωτός πρέπει να ορίσουμε τα όρια μέσα στα οποία επιδρά. Για αυτό τον λόγο δημιουργούμε ένα αντικείμενο της κλάσης Bounding Sphere.

BoundingSphere bounds = new BoundingSphere(new Point3d(0.0,0.0,0.0), Double.MAX_VALUE)

Σε αυτή την επιλογή της εφαρμογής έχουμε χρησιμοποιήσει μία πηγή φωτός από κάθε είδος. Όλες οι πηγές φωτός χρησιμοποιούν τα ίδια όρια μέσα στα οποία έχει ισχύ η επίδρασή τους τα όρια της σφαίρας bounds.

Για να δημιουργήσουμε φως περιβάλλοντος χρησιμοποιούμε την κλάση AmbientLight.

Color3f ambientColour = new Color3f(amb_col_r,amb_col_g,amb_col_b)

ambientLight = new AmbientLight(ambientColour)

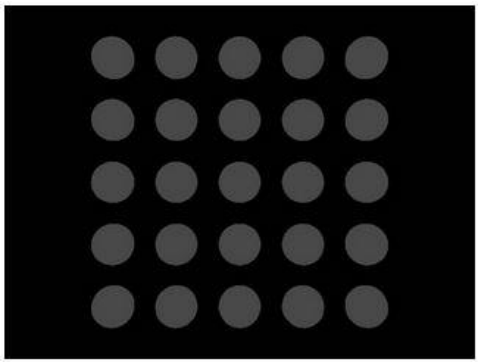
ambientLight.setInfluencingBounds(bounds)

ambientLight.setCapability(AmbientLight.ALLOW_STATE_WRITE)

ambientLight.setCapability(AmbientLight.ALLOW_COLOR_WRITE)

bgLight.addChild(ambientLight)

Στην πρώτη εντολή ορίζεται το χρώμα για το φως περιβάλλοντος το οποίο παίρνει τις τιμές : (0.8f,0.8f,0.8f) αντίστοιχα για τις red, green , blue τιμές. Συνήθως το χρώμα του φωτός περιβάλλοντος είναι το γκρι. ((0.0f,0.0f,0.0f) -> μαύρο, (1.0f,1.0f,1.0f)-> άσπρο)). Στην δεύτερη εντολή δημιουργείται το αντικείμενο της κλάσης AmbientLight παίρνοντας ως όρισμα το χρώμα ambientColour. Στην τρίτη εντολή θέτονται τα όρια για το φως περιβάλλοντος. Στην εφαρμογή οι πηγές φωτός μπορούν να αλλάξουν χρώμα αλλά και κατάσταση (να ανοίξουν ή να κλείσουν) για αυτό σε όλες τις πηγές φωτός έχουμε ορίζει τις αντίστοιχες ικανότητες (capabilities). Εδώ οι capabilities για το ambient light ορίζονται στις εντολές 4 και 5. Στην τελευταία εντολή η πηγή φωτός που δημιουργήσαμε ανατίθεται ως παιδί στο branch group bgLight.



ΕΙΚΟΝΑ 180 ΦΩΣ ΠΕΡΙΒΑΛΛΟΝΤΟΣ

Για να δημιουργήσουμε μία κατευθυνόμενη πηγή φωτός χρησιμοποιούμε την κλάση Directional Light.

Color3f dirColour = new Color3f(dir_col_r, dir_col_g, dir_col_b)

Vector3f lightDir = new Vector3f(0.0f, 4.0f, -1.0f)

dirLight = new DirectionalLight(dirColour, lightDir)

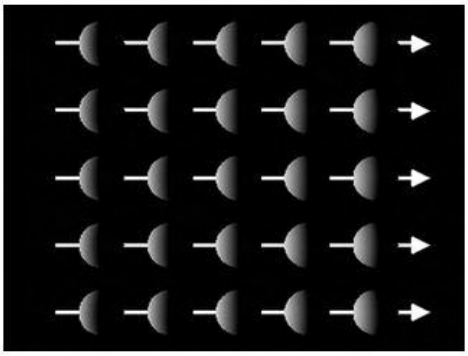
dirLight.setInfluencingBounds(bounds)

dirLight.setCapability(PointLight.ALLOW_STATE_WRITE)

dirLight.setCapability(PointLight.ALLOW_COLOR_WRITE)

bgLight.addChild(dirLight)

Στην πρώτη εντολή ορίζεται το χρώμα για την κατευθυνόμενη πηγή φωτός το οποίο παίρνει τις τιμές : (1.0f,1.0f,1.0f) αντίστοιχα για τις red, green , blue τιμές. Το χρώμα που έχουμε επιλέξει είναι το άσπρο. Εκτός από το χρώμα για την κατευθυνόμενη πηγή φωτός πρέπει να οριστεί η κατεύθυνση για τις παράλληλες ακτίνες στην μορφή ενός διανύσματος $(x, y, z)^T$ από float τιμές. Στην δεύτερη εντολή ορίζεται το παραπάνω διάνυσμα. Στην τρίτη εντολή δημιουργείται ένα αντικείμενο της κλάσης Directional Light παίρνοντας ως ορίσματα το χρώμα dirColour και το διάνυσμα της κατεύθυνσης lightDir. Στην τέταρτη εντολή θέτονται τα όρια επίδρασης αυτής της πηγής φωτός ενώ στην πέμπτη και στην έκτη εντολή ορίζονται οι capabilities για να επιτρέπεται η αλλαγή του χρώματος αλλά και της κατάστασης της κατευθυνόμενης πηγής φωτός. Στην έβδομη εντολή η πηγή φωτός ανατίθεται ως παιδί στο branch group bgLight.



ΕΙΚΟΝΑ 181 ΚΑΤΕΥΘΥΝΟΜΕΝΗ ΠΗΓΗ ΦΩΤΟΣ

Για να δημιουργήσουμε ένα point light χρησιμοποιούμε την κλάση Point Light.

pointColour = new Color3f(point_col_r, point_col_g, point_col_b)

attenuation = new Point3f(0.2f,0.01f,0.01f)

Point3f position1 = new Point3f(1.0f,1.0f,1.0f)

pointLight1 = new PointLight(pointColour,position1,attenuation)

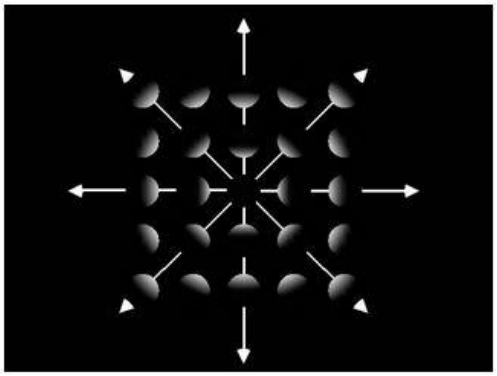
pointLight1.setInfluencingBounds(bounds)

pointLight1.setCapability(PointLight.ALLOW_STATE_WRITE)

pointLight1.setCapability(PointLight.ALLOW_COLOR_WRITE)

bgLight.addChild(pointLight1)

Στην πρώτη εντολή ορίζεται το χρώμα για το point light το οποίο παίρνει τις τιμές : (0.3f,0.3f,0.3f) αντίστοιχα για τις red, green , blue τιμές. Το χρώμα που έχουμε επιλέξει είναι γκρι αλλά πιο σκούρο από εκείνο του ambient light. Εκτός από το χρώμα πρέπει να οριστεί η θέση αλλά και η απόσβεση του φωτός με αντικείμενο της κλάσης Point3f που αποτελούνται από float τιμές. Στην δεύτερη εντολή ορίζεται η απόσβεση και στην τρίτη η θέση. Στην τέταρτη εντολή δημιουργείται ένα αντικείμενο της κλάσης Point Light παίρνοντας ως ορίσματα το χρώμα pointColour και τα αντικείμενα της κλάσης Point3f attenuation και position1. Στην πέμπτη εντολή θέτονται τα όρια επίδρασης αυτής της πηγής φωτός ενώ στην έκτη και στην έβδομη εντολή ορίζονται οι capabilities για να επιτρέπεται η αλλαγή του χρώματος αλλά και της κατάστασης του point light. Στην τελευταία εντολή η πηγή φωτός ανατίθεται ως παιδί στο branch group bgLight.



ΕΙΚΟΝΑ 182 POINT LIGHT

Τέλος για να δημιουργήσουμε ένα spotlight χρησιμοποιούμε την κλάση Spot Light.

Color3f spotColour = new Color3f(spot_col_r, spot_col_g, spot_col_b)

spotLight = new SpotLight(spotColour,new Point3f(-2.0f,1.0f,2.0f),new Point3f(0.15f,0.15f,0.0f),new Vector3f(1.0f,0.0f,-1.0f),(float) (Math.PI/2.0),0.0f)

spotLight.setInfluencingBounds(bounds)

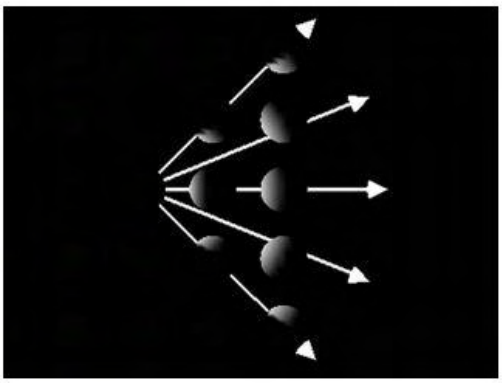
spotLight.setCapability(SpotLight.ALLOW_STATE_WRITE)

spotLight.setCapability(PointLight.ALLOW_COLOR_WRITE)

bgLight.addChild(spotLight)

Στην πρώτη εντολή ορίζεται το χρώμα για το spotlight το οποίο παίρνει τις τιμές : (0.3f,0.3f,0.3f) αντίστοιχα για τις red, green , blue τιμές όπως και το point light. Εκτός από το χρώμα, τη θέση αλλά και την απόσβεση του φωτός όπως στο point light πρέπει να οριστεί επίσης η κατεύθυνση στην οποία λάμπει το spotlight, το πόσο το spotlight έχει εστιάσει αλλά και η γωνία του κώνου φωτός. Η πρώτη παράμετρος στο αντικείμενο που έχει δημιουργηθεί στην δεύτερη εντολή για το spotlight αποτελεί το χρώμα, η δεύτερη την θέση (point3f), η τρίτη την απόσβεση (point3f), η τέταρτη παράμετρος την κατεύθυνση (vector3f), η πέμπτη την γωνία και η έκτη την εστίαση. Η γωνία (float) ορίζει τη γωνία στην οποίο αντιστοιχεί το μισό από το άνοιγμα του κώνου του φωτός. Η εστίαση (float) παίρνει τιμές από 0 μέχρι 120 και καθορίζει πόσο το spotlight έχει εστιάσει στον κεντρικό άξονα. Για την τιμή 0 το φως θα έχει την ίδια ένταση στον κεντρικό άξονα αλλά και στα όρια του κώνου, για την τιμή 120 το φως θα είναι εστιασμένο στον κεντρικό άξονα του κώνου και σχεδόν καθόλου στα όρια. Στην τρίτη εντολή θέτονται τα όρια επίδρασης αυτής της πηγής φωτός ενώ στην τέταρτη και στην πέμπτη εντολή ορίζονται οι capabilities για να επιτρέπεται η αλλαγή του χρώματος

αλλά και της κατάστασης του spotlight. Στην τελευταία εντολή η πηγή φωτός ανατίθεται ως παιδί στο branch group bgLight.



ΕΙΚΟΝΑ 183 SPOTLIGHT

Στην εφαρμογή έχουν χρησιμοποιηθεί οι 4 παραπάνω πηγές φωτός με τις αντίστοιχες παραμέτρους. Τα χρώματα των πηγών μπορούνε να αλλάξουν από το προεπιλεγμένο (άσπρο) σε κόκκινο, πράσινο και μπλε. Αυτό γίνεται αλλάζοντας τις τιμές των αντικειμένων των χρωμάτων που παίρνουν ως παραμέτρους οι πηγές φωτός. Πιο συγκεκριμένα αλλάζουν οι τιμές amb_col_r, amb_col_g, amb_col_b του ambientColour για ο φως περιβάλλοντος, οι τιμές dir_col_r, dir_col_g, dir_col_b του dirColour για το κατευθυνόμενο φως, οι τιμές point_col_r, point_col_g, point_col_b του pointColour για το point light και οι τιμές spot_col_r, spot_col_g, spot_col_b του spotColour για το spotlight. Οι αλλαγές στα χρώματα γίνονται με την βοήθεια της μεθόδου set Color() καλούμενη κάθε φορά για τα αντικείμενα των πηγών φωτός με παράμετρο το νέο χρώμα που θέλουμε να δώσουμε στην πηγή φωτός τύπου Color3f. Για παράδειγμα για να δώσουμε μπλε χρώμα στο spotlight γράφουμε :

spot_col_r=0.0f

spot_col_g=0.0f

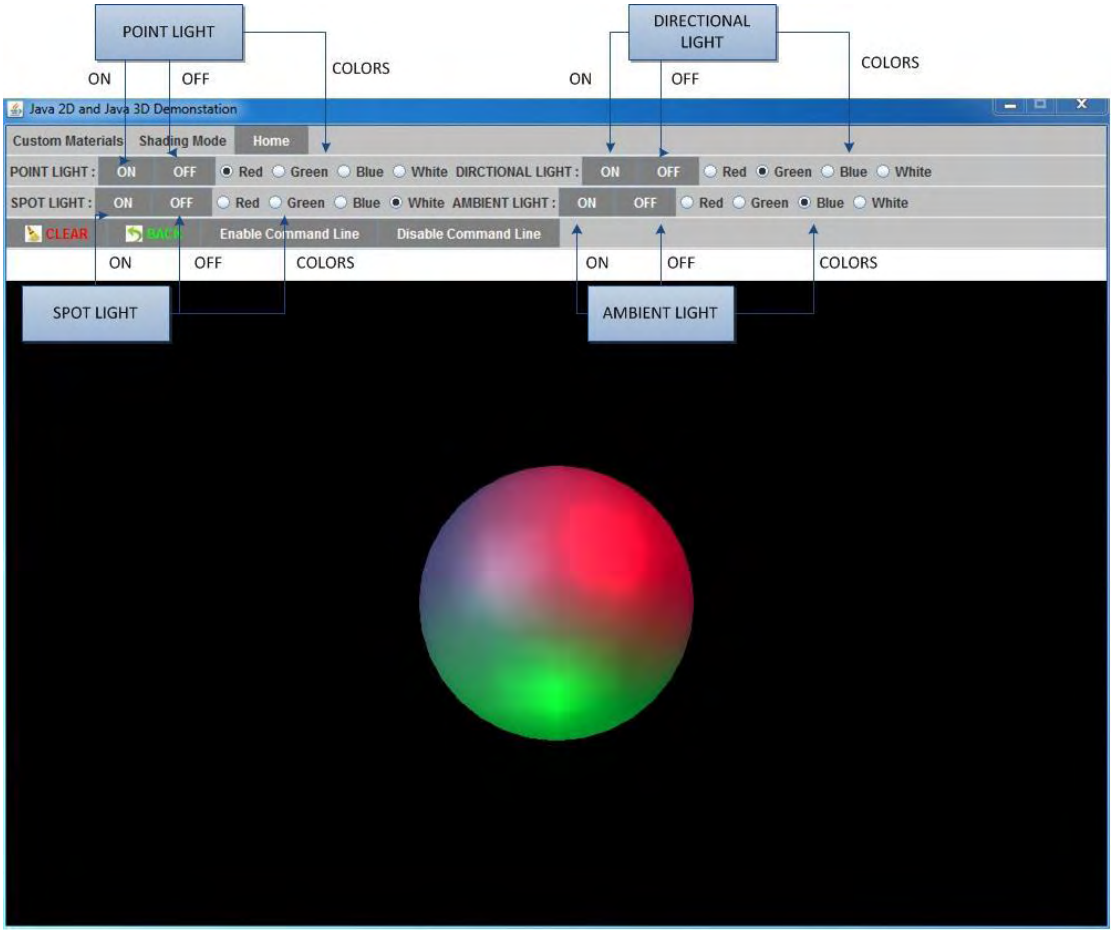
spot_col_b=0.3f

spotLight.setColor(new Color3f(spot_col_r,spot_col_g,spot_col_b))

Επίσης δίνεται η δυνατότητα στον χρήστη να ενεργοποιήσει ή να απενεργοποιήσει τις πηγές φωτός. Αυτό πραγματοποιείται με την βοήθεια της μεθόδου set Enable() καλούμενη για τα αντικείμενα των πηγών φωτός με παράμετρο true αν θέλουμε να ενεργοποιήσουμε τις πηγές φωτός ή false αν θέλουμε να τις απενεργοποιήσουμε. Για παράδειγμα αν θέλουμε να ενεργοποιήσουμε το spotlight γράφουμε :

spotLight.setEnable(true)

Στην εφαρμογή ο χρήστης μπορεί να πραγματοποιήσει τις αλλαγές στα χρώματα των πηγών φωτός επιλέγοντας δίπλα από την αντίστοιχη ετικέτα το χρώμα της αρεσκείας του (κόκκινο, μπλε, πράσινο, άσπρο - default). Για να απενεργοποιήσει ή να ενεργοποιήσει τις πηγές φωτός επιλέγει αντίστοιχα το «ON» ή το «OFF» δίπλα από την ετικέτα της πηγής φωτός που τον ενδιαφέρει.



ΕΙΚΟΝΑ 184 ΚΟΥΜΠΙΑ ΓΙΑ ΕΝΕΡΓΟΠΟΙΗΣΗ ΑΠΕΝΕΡΓΟΠΟΙΗΣΗ ΠΗΓΩΝ ΕΝΕΡΓΕΙΑΣ ΚΑΙ ΕΠΙΛΟΓΗ ΧΡΩΜΑΤΩΝ

13.2.2 Προσαρμοσμενα Υλικα – Custom Materials

Στην παράγραφο 12.2.9 [12.2.9] παρουσιάστηκε η κλάση Material η οποία χαρακτηρίζει την κλάση Appearance και καθορίζει την εμφάνιση ενός αντικειμένου όταν βρίσκεται υπό φωτισμό. Επειδή σε αυτή την επιλογή της εφαρμογής χρησιμοποιούνται όλες οι πηγές φωτισμού και με διαφορετικά χρώματα έχουν επιλεχθεί να παρουσιαστούν κάποια custom materials για να μπορούν να φανούν επάνω τους τα εφέ που δημιουργούνται από τις πηγές φωτός. Τα υλικά που είναι διαθέσιμα στην επιλογή είναι : οψιδιανός, τουρκουάζ, χαλκός, μεταλλικό μωβ και μεταλλικό κόκκινο. Τα υλικά αυτά είναι αντικείμενα της κλάσης Material.

Material material = new Material()

Κάθε ένα από αυτά χαρακτηρίζεται από ένα Ambient Color, ένα Diffuse Color, ένα Specular Color και από τη μεταβλητή Shininess.

- Οψιδιανός (Obsidian)

  material.setAmbientColor (new Color3f(0.05375f, 0.05f, 0.06625f))

  material.setDiffuseColor (new Color3f(0.18275f, 0.17f, 0.22525f))

  material.setSpecularColor(new Color3f(0.332741f, 0.328634f, 0.346435f))

material.setShininess(0.3f)

- Τουρκουάζ (Turquoise)

  material.setAmbientColor (new Color3f(0.1f, 0.18725f, 0.1745f))

  material.setDiffuseColor (new Color3f(0.396f, 0.74151f, 0.69102f))

  material.setSpecularColor(new Color3f(0.297254f, 0.30829f, 0.306678f))

  material.setShininess(0.1f)

- Χαλκός (Copper)

  material.setAmbientColor (new Color3f(0.19125f, 0.0735f, 0.0225f))

  material.setDiffuseColor (new Color3f(0.7038f, 0.27048f, 0.0828f))

  material.setSpecularColor(new Color3f(0.256777f, 0.137622f, 0.086014f))

  material.setShininess(0.1f)

- Μεταλλικό Μωβ (Metallic Purple)

  material.setAmbientColor (new Color3f(0.25f, 0.17f, 0.19f))

  material.setDiffuseColor (new Color3f(0.10f, 0.03f, 0.22f))

  material.setSpecularColor(new Color3f(0.64f, 0.00f, 0.98f))

  material.setShininess(0.08f)

- Μεταλλικό Κόκκινο (Metallic Red)

  material.setAmbientColor (new Color3f(0.25f, 0.15f, 0.15f))

  material.setDiffuseColor (new Color3f(0.27f, 0.00f, 0.00f))

  material.setSpecularColor(new Color3f(0.61f, 0.13f, 0.18f))

  material.setShininess(0.12f)

Για να μπορέσουμε να διαβάσουμε αλλά και να αλλάξουμε τις τιμές των παραμέτρων του υλικού αναθέσαμε στο υλικό τις «ικανότητες» (capabilities):

material.setCapability(Material.ALLOW_COMPONENT_WRITE)

material.setCapability(Material.AMBIENT)

material.setCapability(Material.DIFFUSE)

material.setCapability(Material.SPECULAR)

material.setCapability(Material.EMISSIVE)

Το αντικείμενο της κλάσης Appearance που έχει χρησιμοποιηθεί είναι το metalApp :

Appearance metalApp = new Appearance()

Για να χαρακτηρίσουμε την metalApp με το υλικό material γράφουμε :

metalApp.setMaterial(material)

Η προεπιλογή για το υλικό είναι το αλουμίνιο το οποίο εκτός των άλλων χαρακτηρίζεται και από το Emissive Color.

- Αλουμίνιο (Aluminum)

  material.setAmbientColor (new Color3f(0.2f,0.2f,0.2f))

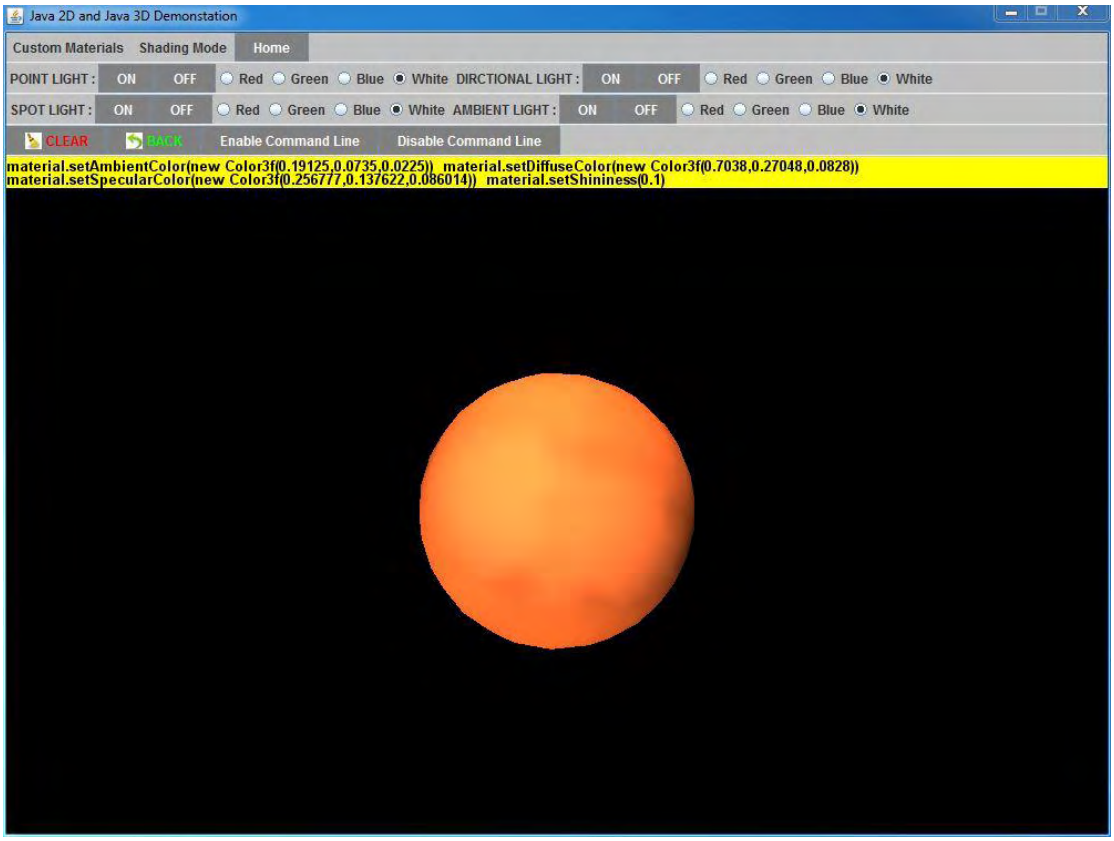
  material.setDiffuseColor (new Color3f(0.6f,0.6f,0.6f))

  material.setSpecularColor(new Color3f(0.5f,0.5f,0.5f))

  material.setEmissiveColor(new Color3f(0.0f,0.0f,0.0f))

  material.setShininess(20.0f)

  Στην εφαρμογή ο χρήστης μπορεί να επιλέξει κάποιο από τα διαθέσιμα υλικά από το μενού «Custom Materials».



ΕΙΚΟΝΑ 185 ΠΑΡΑΔΕΙΓΜΑ ΥΛΙΚΟΥ – ΧΑΛΚΟΣ

13.2.3 ΣΚΙΑΣΗ

Για τον υπολογισμό της αντανάκλασης του φωτός στην παράγραφο 12.2.8 [12.2.8] θεωρήθηκε ότι το normal vector της επιφάνειας είναι γνωστό σε κάθε σημείο. Για να υπολογιστεί το σωστό χρώμα ενός pixel πάνω στην επιφάνεια προβολής δεν είναι αρκετό να καθοριστεί μόνο ποια

επιφάνεια του αντικειμένου είναι ορατή στο συγκεκριμένο pixel αλλά και σε ποιο σημείο οι γραμμές που ενώνουν τα σημεία που βρίσκεται το αντικείμενο με το κέντρο της προβολής συναντούν την επιφάνεια. Για τις κυβικές επιφάνειες ελευθέρου σχήματος αυτό θα σήμαινε ότι ένα σύστημα εξισώσεων πρέπει να επιλυθεί του οποίου οι μεταβλητές θα ήταν υψωμένες στην δύναμη του 3, αυτό θα οδηγούσε σε τεράστιο υπολογιστικό κόστος ανά pixel. Για αυτό οι αντανακλάσεις του φωτός δεν υπολογίζονται απευθείας για τις επιφάνειες ελεύθερου σχήματος, αλλά με προσεγγίσεις με πολύγωνα. Μία απλή προσέγγιση αγνοεί τα αρχικά normal vectors της επιφάνειας ελεύθερου σχήματος και χρησιμοποιεί τα normal vectors της επιφάνειας των πολυγώνων. Η επίπεδη (flat) σκίαση απλοποιεί περισσότερο αυτή την ιδέα. Για ένα πολύγωνο, το χρώμα καθορίζεται μόνο από ένα pixel το οποίο βασίζεται μόνο σε ένα normal vector. Όλα τα άλλα pixel που προβάλλονται ως μέρη αυτού του πολυγώνου παίρνουν το ίδιο χρώμα με αποτέλεσμα το προβαλλόμενο πολύγωνο να αποκτά ομοιογενές χρώμα. Η προσέγγιση αυτή είναι σωστή υπό τις ακόλουθες υποθέσεις :

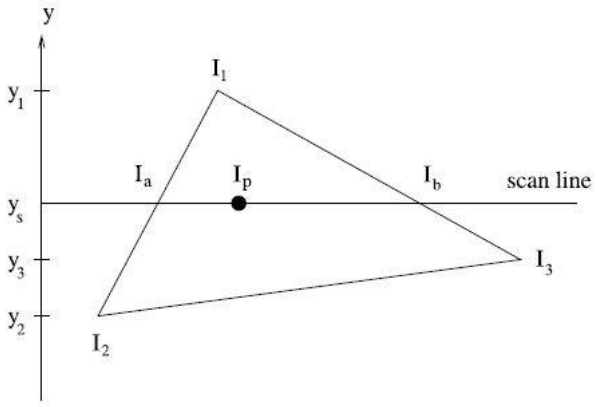
- Η πηγή του φωτός είναι σε άπειρη απόσταση έτσι ώστε το $n^T \cdot I$ να είναι σταθερό. Αυτό εφαρμόζεται μόνο στις κατευθυνόμενες πηγές φωτός.
- Ο παρατηρητής βρίσκεται σε άπειρη απόσταση έτσι ώστε το $n^T \cdot v$ να είναι σταθερό. Αυτό είναι αληθές για την παράλληλη προβολή.
- Το πολύγωνο αναπαριστά την πραγματική επιφάνεια του αντικειμένου και δεν είναι απλά μία προσέγγιση μίας καμπύλης επιφάνειας.
- Δεν υπάρχει κατοπτρική ανάκλαση.

Με αυτές τις υποθέσεις, η σκίαση μπορεί να υπολογιστεί με γρήγορο και εύκολο τρόπο αλλά δεν οδηγεί σε ρεαλιστικές εικόνες.

ΕΙΚΟΝΑ 186 FLAT ΣΚΙΑΣΗ ΣΕ ΣΦΑΙΡΑ ΜΕ ΔΙΑΦΟΡΕΤΙΚΗ ΨΗΦΙΟΠΟΙΗΣΗ

Στην εικόνα 186 είναι φανερό ότι ακόμη και στην ψηφιοποίηση με τον μεγαλύτερο αριθμό τριγώνων φαίνονται καθαρά τα τρίγωνα. Για την flat σκίαση απαιτείται ψηφιοποίηση με πάρα πολύ μεγάλο αριθμό τριγώνων για να μην γίνεται ορατό αυτό το ανεπιθύμητο εφέ. Έτσι αντί για την flat σκίαση χρησιμοποιείται η παρεμβάλλουσα σκίαση (interpolated). Η παρεμβάλλουσα σκίαση απαιτεί τον ορισμό των normal vectors στις κορυφές των πολυγώνων ή των τριγώνων. Τα normal vectors στις τρεις κορυφές ενός τριγώνου μπορεί να διαφέρουν για την παρεμβάλλουσα σκίαση όταν το τρίγωνο προσεγγίζει ένα μέρος μίας καμπύλης επιφάνειας. Σε κάθε κορυφή τα normal vectors των τριγώνων που μοιράζονται την κορυφή αυτή παρεμβάλλονται. Όταν μία καμπύλη επιφάνεια προσεγγίζεται από τρίγωνα και τα αντίστοιχα normal vectors ορίζονται για τις κορυφές των τριγώνων η Gouraud σκίαση υπολογίζει το χρώμα της κάθε κορυφής βασισμένη στα αντίστοιχα normal vectors. Η σκίαση των άλλων σημείων εσωτερικά του τριγώνου βασίζεται στην παρεμβολή των χρωμάτων που προέρχονται από τις τρεις κορυφές. Αυτό οδηγεί σε γραμμική διαβάθμιση χρώματος σε όλη την επιφάνεια του τριγώνου.

Μία γρήγορη τεχνική για τον υπολογισμό των εντάσεων στο τρίγωνο χρησιμοποιεί την τεχνική scan line. Για την scan line ys οι εντάσεις Ia, Ib στις ακμές του τριγώνου υπολογίζονται όπου η scan line τέμνει το τρίγωνο. Αυτές οι τιμές βρίσκονται από σταθμισμένη παρεμβολή μεταξύ των κορυφών των αντίστοιχων ακμών του τριγώνου. Η ένταση αλλάζει γραμμικά κατά μήκος της scan line με αρχική τιμή Ia και τελική Ib. Η αρχή αυτή φαίνεται στην εικόνα 187.



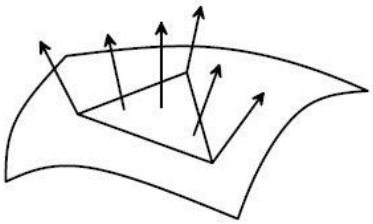
ΕΙΚΟΝΑ 187 ΤΕΧΝΙΚΗ SCAN LINE ΓΙΑ ΤΟΝ ΥΠΟΛΟΓΙΣΜΟ ΤΗΣ ΣΚΙΑΣΗΣ GOURAUD

Οι εντάσεις υπολογίζονται βάση των εξισώσεων :

$$Ia = I1 - (I1 - I2)\frac{y1 - ys}{y1 - y2}\ , Ib = I1 - (I1 - I3)\frac{y1 - ys}{y1 - y3}\ \ , Ip = Ib - (Ib - Ia)\frac{xb - xp}{xb - xa}$$

Οι εντάσεις των χρωμάτων που υπολογίζονται παίρνουν ακέραιες τιμές μεταξύ του 0 και του 255. Συνήθως οι εντάσεις σε ένα τρίγωνο δεν θα διαφέρουν σημαντικά έτσι ώστε η κλίση της γραμμικής καμπύλης της έντασης κατά μήκος της scan line θα είναι μικρή. Σε αυτή την περίπτωση ο αλγόριθμος midpoint μπορεί να εφαρμοστεί για να καθορίσει τις διακριτές τιμές της έντασης.

Το ανεπιθύμητο εφέ που κάνει ορατά τα τρίγωνα (ακμές) με την flat σκίαση διορθώνεται με την σκίαση Gouraud. Πάραυτα, λόγω της γραμμικής παρεμβολής που χρησιμοποιεί η σκίαση Gouraud το ελάχιστον και το μέγιστο της έντασης των χρωμάτων στο τρίγωνο θα βρίσκεται πάντα στις κορυφές. Το γεγονός αυτό μπορεί να οδηγεί πάλι σε ορατές ακμές ή κορυφές. Η σκίαση Phong βασίζεται στην παρεμβολή όπως και η σκίαση Gouraud. Όμως αντί να παρεμβάλλουμε τις εντάσεις των χρωμάτων από τις κορυφές του τριγώνου για τον υπολογισμό των εντάσεων των χρωμάτων των άλλων σημείων, παρεμβάλλουμε τα normal vectors στις κορυφές. Με αυτόν τον τρόπο είναι πιθανόν η ελάχιστη και η μέγιστη ένταση χρώματος να βρίσκεται μέσα στο τρίγωνο ανάλογα με την διαμόρφωση των normal vectors στις κορυφές και ανάλογα με την κατεύθυνση από την οποία έρχεται το φως. Η εικόνα 188 δείχνει μία καμπύλη επιφάνεια και ένα τρίγωνο το οποίο προσεγγίζει ένα μέρος της επιφάνειας.



ΕΙΚΟΝΑ 188 NORMAL VECTORS ΓΙΑ ΤΗΝ ΣΚΙΑΣΗ PHONG

Τα normal vectors στις κορυφές του τριγώνου είναι τα normal vectors στην επιφάνεια σε αυτά τα σημεία. Εσωτερικά του τριγώνου τα normal vectors είναι κυρτοί συνδυασμοί των normal vectors των κορυφών.

Η σκίαση Phong απαιτεί πολύ περισσότερο χρόνο υπολογισμού από την σκίαση Gouraud. Για την σκίαση Gouraud οι πολύπλοκοι υπολογισμοί για τον φωτισμό (πηγές φωτός, αντανακλάσεις) πρέπει να γίνουν μόνο για τις τρεις κορυφές του τριγώνου. Το υπόλοιπο τρίγωνο σκιάζεται με την

απλή τεχνική scan line και κάνοντας απλούς υπολογισμούς. Για την σκίαση Phong μετά την παρεμβολή των normal vectors όλοι οι υπολογισμοί για τον φωτισμό πρέπει να γίνουν για κάθε ένα από τα pixel.

Στην Java 3D η προεπιλεγμένη τεχνική σκίασης είναι η Gouraud. Για να αλλάξουμε την σκίαση σε flat πρέπει να αλλάξουμε την εμφάνιση (Appearance) που χρησιμοποιούμε. Αρχικά πρέπει να ορίσουμε ένα αντικείμενο της κλάσης Coloring Attributes η οποία χαρακτηρίζει την Appearance ως προς την επιλογή χρώματος αλλά και το μοντέλο σκίασης.

ColoringAttributes ca = new ColoringAttributes()

Για να αλλάξουμε την τεχνική σκίασης χρησιμοποιούμε την μέθοδο set Shade Model() με την ανάλογη παράμετρο. Για flat σκίαση εισάγουμε την παράμετρο ColoringAttributes.SHADE_FLAT ενώ για την σκίαση Gouraud ColoringAttributes.SHADE_GOURAUD.

ca.setShadeModel(ColoringAttributes.SHADE_GOURAUD)

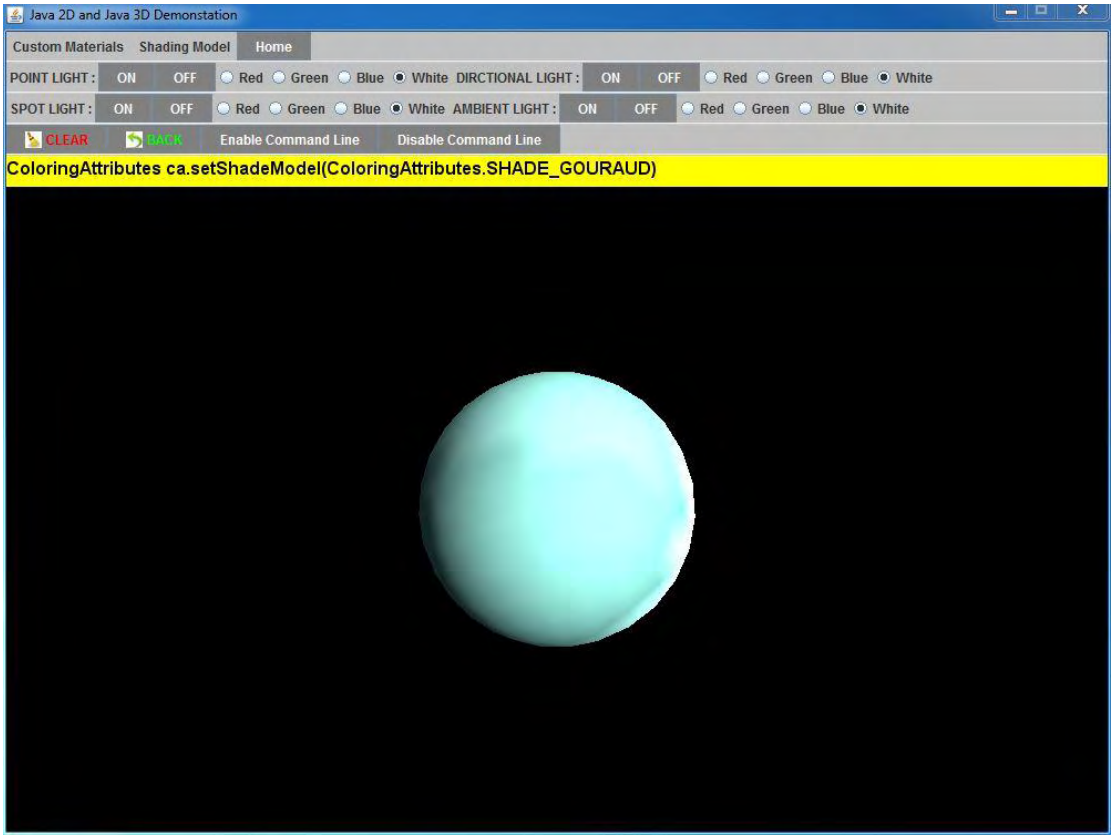
Το αντικείμενο της κλάσης Appearance που έχει χρησιμοποιηθεί είναι το metalApp. Για να χαρακτηρίσουμε την metalApp με τα Coloring Attributes γράφουμε :

metalApp.setColoringAttributes(ca)

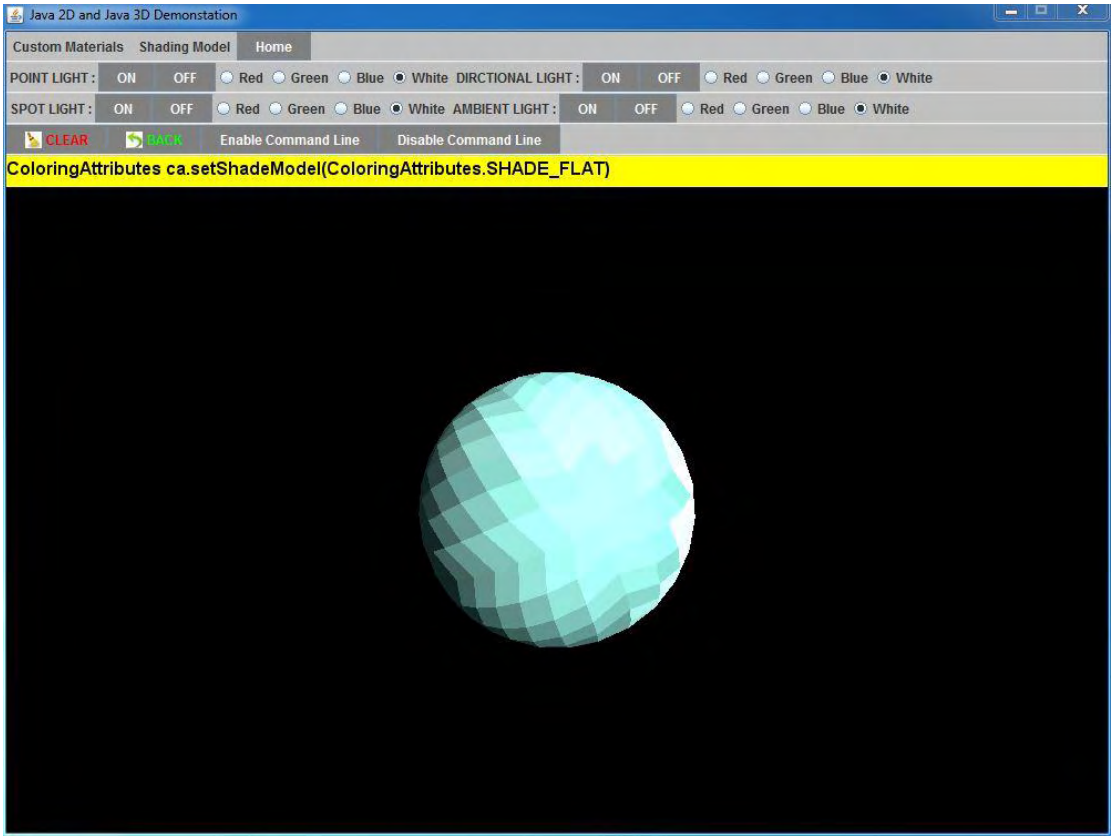
Για να μπορέσουμε να αλλάξουμε το μοντέλο σκίασης αναθέτουμε στο αντικείμενο της κλάσης Coloring Attributes την παρακάτω «ικανότητα» (capability):

ca.setCapability(ColoringAttributes.ALLOW_SHADE_MODEL_WRITE)

Στην εφαρμογή ο χρήστης μπορεί να αλλάξει την τεχνική σκίασης από το μενού «Shading Model» σε flat ή Gouraud. Το προεπιλεγμένο μοντέλο σκίασης είναι το Gouraud. Σκίαση εφαρμόζεται στη σφαίρα που είναι και το μοναδικό αντικείμενο του εικονικού κόσμου της επιλογής αυτής.



ΕΙΚΟΝΑ 189 ΜΟΝΤΕΛΟ GOURAUD



ΕΙΚΟΝΑ 190 ΜΟΝΤΕΛΟ FLAT

13.2.4 Σκιες

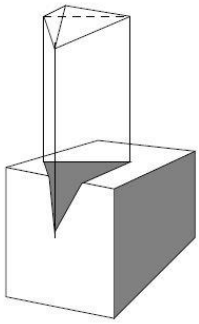
Η σκιά ορίζεται ως η απουσία φωτός από μία πηγή φωτός η οποία δεν φτάνει την επιφάνεια του αντικειμένου στο οποίο δημιουργείται η σκιά. Η εξίσωση φωτισμού περιέχοντας τις σκιές γίνεται :

$$I = I_{self\ emission} + I_{ambient\ light} \cdot k_a + \sum_j S_j \cdot I_J \cdot f_{att} \cdot g_{cone} \cdot \left(k_d \cdot (n^T \cdot I_j) + k_{sr} \cdot (r_j^T \cdot v)^n\right)$$

Η εξίσωση παραμένει ίδια εκτός από την προσθήκη του παράγοντα S_j.

$$S_j = \begin{cases} 1, αν\ το\ φως\ από\ την\ πηγή\ φωτός\ j\ φτάνει\ την\ επιφάνεια \\ 0, αλλιώς\ (\ δημιουργείται\ σκιά\ ) \end{cases}$$

Έχουν μελετηθεί στην παράγραφο 12.2.6 [12.2.6] μέθοδοι που καθορίζουν αν ένα αντικείμενο μέσα στην σκηνή είναι ορατό από τον παρατηρητή ή δεν είναι ορατό λόγω άλλων αντικειμένων που βρίσκονται μπροστά του. Ο καθορισμός της σκιάς βασίζεται στην ίδια λογική μόνο που αντί για τον παρατηρητή εφαρμόζουμε τις μεθόδους αυτές για τις πηγές φωτός. Όταν μία επιφάνεια είναι ορατή από μία πηγή φωτός τότε το S_j = 1 και δεν δημιουργείται σκιά από αυτή την πηγή φωτός στην επιφάνεια. Όταν η επιφάνεια δεν είναι ορατή από την πηγή φωτός τότε $S_j = 0$ και δημιουργείται σκιά στο αντικείμενο. Η εικόνα 191 δείχνει την σκιά που δημιουργείται πάνω στον κύβο η οποία προκαλείται από ένα τετράεδρο που μπλοκάρει το φως που προέρχεται από μία πηγή φωτός που βρίσκεται πάνω και από τα δύο σχήματα. Όταν υπάρχει σκιά σε μία επιφάνεια δεν σημαίνει ότι η επιφάνεια είναι μαύρη. Το φως περιβάλλοντος θα αντανακλάται. Και αν υπάρχουν παραπάνω από μία πηγές φωτός στην σκηνή η επιφάνεια μπορεί να μπλοκάρεται από μία πηγή φωτός αλλά μπορεί να φωτίζεται κανονικά από τις άλλες.



Εικονα 191 δημιουργια σκιας

Την σχέση μεταξύ σκιών και καθορισμού ορατότητας των αντικειμένων εκμεταλλεύεται ο two-pass z ή two-pass depth buffer αλγόριθμος (αλγόριθμος z δύο περασμάτων). Στο πρώτο πέρασμα εφαρμόζεται ο αλγόριθμος z – buffer με τους ακόλουθες αλλαγές. Ο παρατηρητής αντικαθίσταται από μία πηγή φωτός. Για μία κατευθυνόμενη πηγή φωτός , εφαρμόζεται παράλληλη προβολή στην αντίθετη κατεύθυνση από αυτή του φωτός. Για κάποιο point light ή spotlight ,εφαρμόζεται προοπτική προβολή με κέντρο προβολής την θέση της πηγής φωτός. Σε όλες τις περιπτώσεις η προβολή μπορεί εντέλει να αναπαρασταθεί από έναν μετασχηματισμό T_L ακολουθούμενο από μία παράλληλη προβολή στο x/y επίπεδο. Στο πρώτο πέρασμα λαμβάνονται υπόψη μόνο οι τιμές για τον z – buffer, τον Z_L. Ο frame buffer και οι υπολογισμοί του δεν χρειάζονται. Το δεύτερο πέρασμα του αλγορίθμου είναι ίδιο με τον z – buffer αλγόριθμο για τον παρατηρητή με την ακόλουθη μετατροπή.

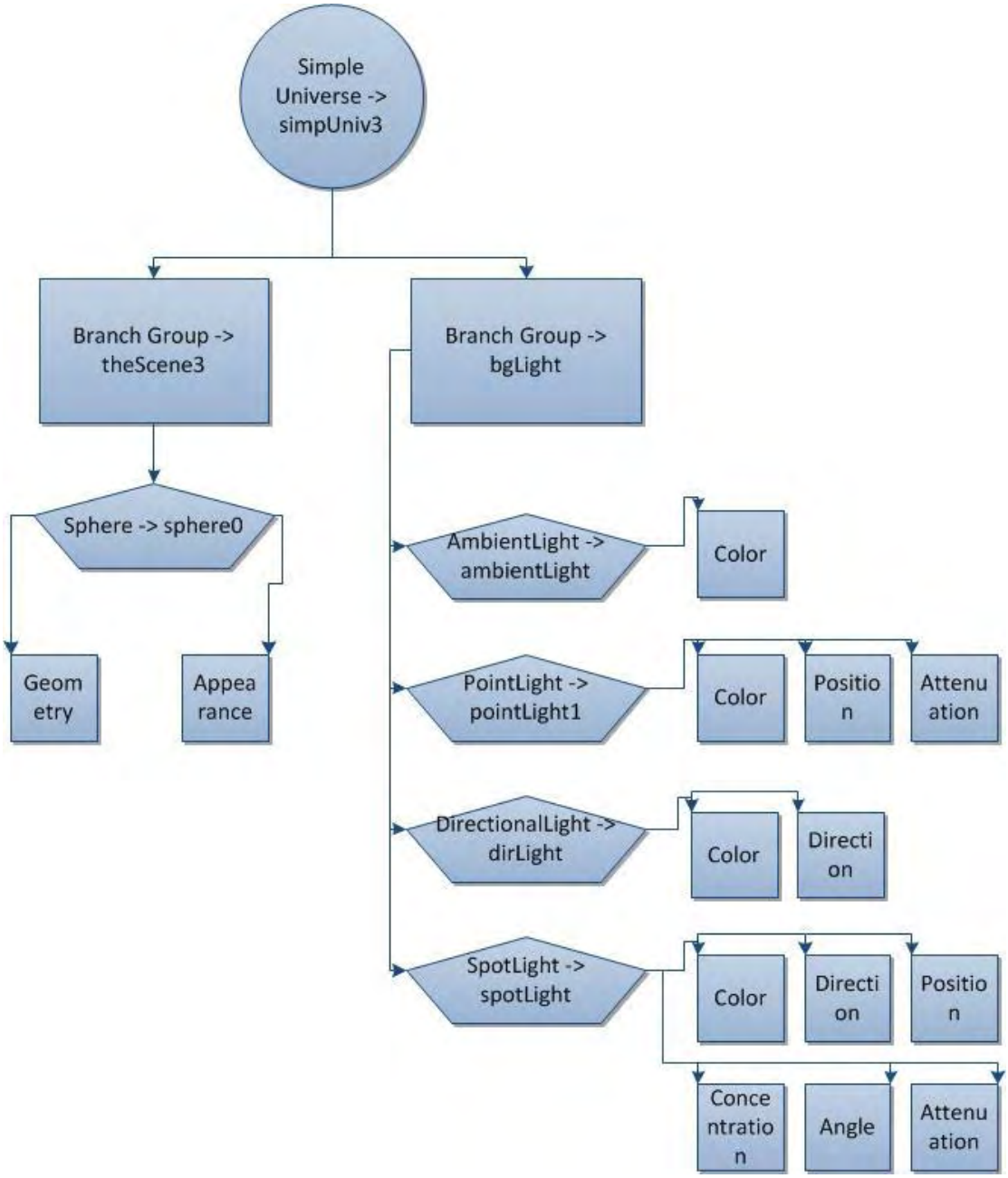
Χρειάζεται ως συνήθως ένας μετασχηματισμός T_V για να μετατρέψει την προοπτική προβολή με κέντρο προβολής τον παρατηρητή σε παράλληλη προβολή στο x/y επίπεδο. Ο Z_V δηλαδή ο z – buffer για τον παρατηρητή γεμίζει με τιμές. Πριν γίνει η προβολή της επιφάνειας στον frame buffer F_V για τον παρατηρητή, γίνεται ένας έλεγχος φωτισμού για να ελεγχθεί αν η επιφάνεια φωτίζεται από την συγκεκριμένη πηγή φωτός. Αν οι συντεταγμένες ενός σημείου της επιφανείας που θα προβληθεί είναι (xv, yv, zv) ο μετασχηματισμός:

$$\begin{pmatrix} xL \\ yL \\ zL \end{pmatrix} = T_L \cdot T_V^{-1} \cdot \begin{pmatrix} xv \\ yv \\ zv \end{pmatrix}$$

Δίνει τις συντεταγμένες από το ίδιο σημείο όπως το «βλέπει» η πηγή φωτός. Ο T_V^{-1} είναι ο αντίστροφος μετασχηματισμός του T_V , δηλαδή ο αντίστροφος πίνακας του T_V . Η τιμή zL συγκρίνεται με την αντίστοιχη τιμή στον z – buffer Z_L για την πηγή φωτός στην θέση (xL, yL). Αν υπάρχει στον Z_L μικρότερη τιμή από την zL στην θέση αυτή , τότε θα υπάρχει αντικείμενο μεταξύ της πηγής φωτός και της επιφανείας έτσι η επιφάνεια αυτή δεν φωτίζεται από την αυτή την πηγή φωτός. Στην επιφάνεια δημιουργείται σκιά και ο παράγοντας $S_j = 0$ για αυτή την πηγή φωτός. Όταν υπάρχουν περισσότερες από μία πηγές φωτός στην σκηνή το πρώτο πέρασμα του αλγορίθμου εφαρμόζεται για κάθε μία από τις πηγές φωτός. Στο δεύτερο πέρασμα καθορίζεται για κάθε πηγή φωτός αν φωτίζει την επιφάνεια ή αν δημιουργεί σκιά και επιλέγονται ανάλογα οι παράγοντες S_j.

13.2.5 Το Scene Graph της Επιλογης

Στο scene graph της επιλογής αυτής εκτός από τα αντικείμενα της σκηνής θα συμπεριλάβουμε και τις πηγές φωτός εφόσον μελετήσαμε πως ορίζονται και δημιουργούνται στα πλαίσια της Java 3D.



Εικονα 192 scene graph lights3dPanel

Με κύκλο έχει σχεδιαστεί το αντικείμενο Simple Universe με ορθογώνια έχουν σχεδιαστεί οι κόμβοι τύπου Branch Group, ενώ με πεντάγωνα τα φύλλα που είναι αντικείμενα Sphere, Ambient Light, Point Light, Directional Light, Spot Light. Με τετράγωνα έχουν σχεδιαστεί τα node components που αναφέρονται στα φύλλα του δέντρου.

13.2.6 Clear and Back – Επαναφορα και Επιστροφη

CLEAR

Με τον όρο «clear» στην επιλογή αυτή εννοούμε να μεταφέρουμε την σφαίρα στην αρχική της θέση (μεταφέροντας την προβολή στην αρχική της θέση) να ενεργοποιήσουμε όλες τις πηγές φωτός, να επαναφέρουμε το άσπρο χρώμα των πηγών φωτός, να θέσουμε το μοντέλο σκίασης σε Gouraud και να επαναφέρουμε τις παραμέτρους του υλικού στις σχετικές τιμές για το υλικό Αλουμίνιο που έχει καθοριστεί ως προεπιλογή. Για να μεταφέρουμε την προβολή στην αρχική της θέση χρησιμοποιούμε τις ίδιες εντολές όπως και στο κουμπί «Home» που μελετήθηκε στην παράγραφο 12.2.2 [12.2.2]. Για να ενεργοποιήσουμε όλες τις πηγές ενέργειας χρησιμοποιούμε την μέθοδο set Enable() με παράμετρο true.

pointLight1.setEnable(true)

dirLight.setEnable(true)

spotLight.setEnable(true)

ambientLight.setEnable(true)

Τα χρώματα κάθε πηγής φωτός είναι Radio Buttons τα οποία δίνουν την δυνατότητα της προεπιλογής κάπου χρώματος. Στην εφαρμογή το προεπιλεγμένο χρώμα είναι το άσπρο. Αν έχει αλλάξει το χρώμα κάποια πηγής φωτός για να επαναφέρουμε το προεπιλεγμένο χρώμα γράφουμε :

ambWhite.setSelected(true) // αντιστοιχεί στο άσπρο χρώμα για το ambient light

spotWhite.setSelected(true) // αντιστοιχεί στο άσπρο χρώμα για το spotlight

pointWhite.setSelected(true) // αντιστοιχεί στο άσπρο χρώμα για το point light

dirWhite.setSelected(true) // αντιστοιχεί στο άσπρο χρώμα για το directional light

Για να θέσουμε το προεπιλεγμένο μοντέλο σκίασης (Gouraud) καλούμε για το αντικείμενο της κλάσης Coloring Attributes την μέθοδο set Shade Model() με την παράμετρο ColoringAttributes.SHADE_GOURAUD.

ca.setShadeModel(ColoringAttributes.SHADE_GOURAUD) // το ca είναι το αντικείμενο της κλάσης Coloring Attributes

Για να επαναφέρουμε τις παραμέτρους του υλικού στις τιμές του υλικού Αλουμίνιο γράφουμε τις εντολές :

material.setAmbientColor (new Color3f(0.2f,0.2f,0.2f))

material.setDiffuseColor (new Color3f(0.6f,0.6f,0.6f))

material.setSpecularColor(new Color3f(0.5f,0.5f,0.5f))

material.setEmissiveColor(new Color3f(0.0f,0.0f,0.0f))

material.setShininess(20.0f)

Στην εφαρμογή για να επαναφέρουμε τον εικονικό κόσμο στην αρχική του κατάσταση πατάμε το κουμπί «Clear».

BACK

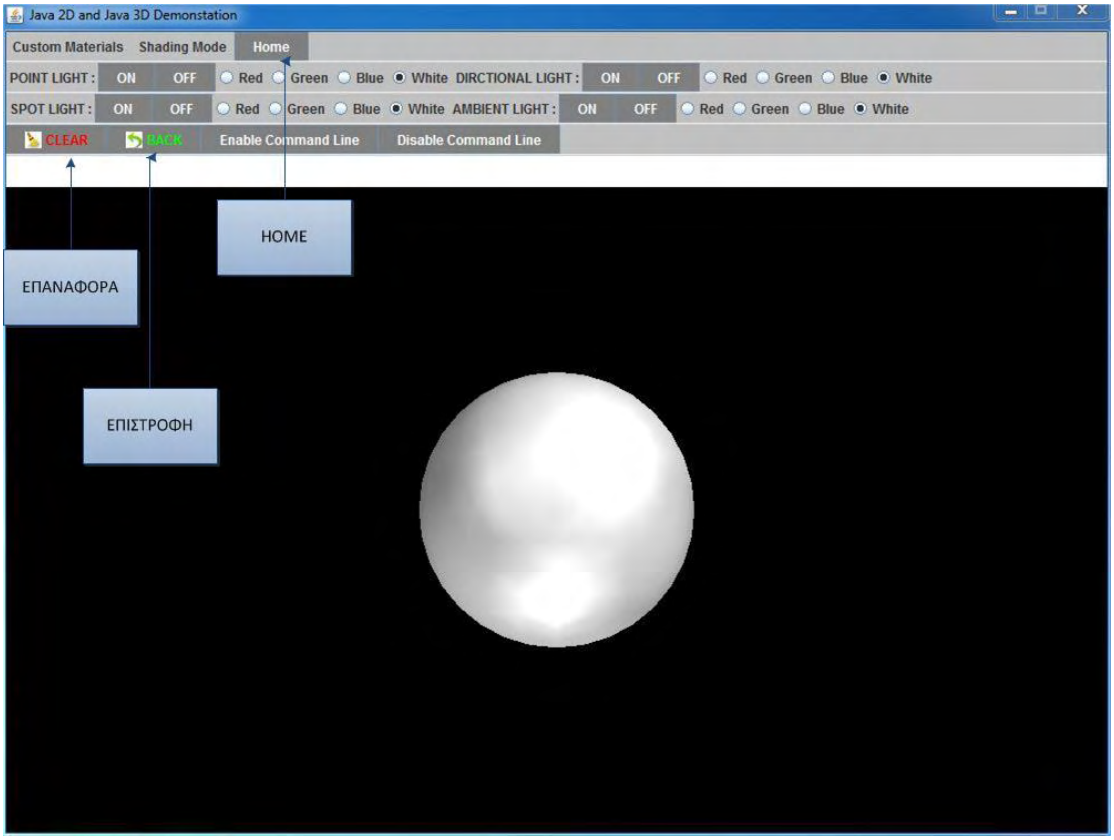
Η εφαρμογή είναι χωρισμένη σε επιλογές που επικοινωνούν μέσω του κοινού εξωφύλλου (CoverPanel). Είναι δυνατή η επιστροφή μέσα από μία εφαρμογή στο εξώφυλλο για να μας δοθεί η δυνατότητα να επιλέξουμε μία άλλη επιλογή. Για να πραγματοποιηθεί αυτό θα πρέπει όπως και στην διαδικασία της επαναφοράς (clear) να επαναφέρουμε τον εικονικό κόσμο στην αρχική του κατάσταση. Έτσι επαναφέρουμε τον εικονικό κόσμο της επιλογής στην αρχική του κατάσταση με τον κώδικα που αναφέρθηκε στο «Clear». Επίσης πρέπει να γίνει ορατό το CoverPanel της εφαρμογής και να μην είναι πλέον ορατό το panel στο οποίο βρισκόμαστε. Για την επιλογή αυτή το panel είναι το Lights3dPanel.

lights3dPanel.setVisible(false)

setContentPane(coverPanel)

coverPanel.setVisible(true)

Στην εφαρμογή για να επιστρέψουμε στο εξώφυλλο πατάμε το κουμπί «BACK».



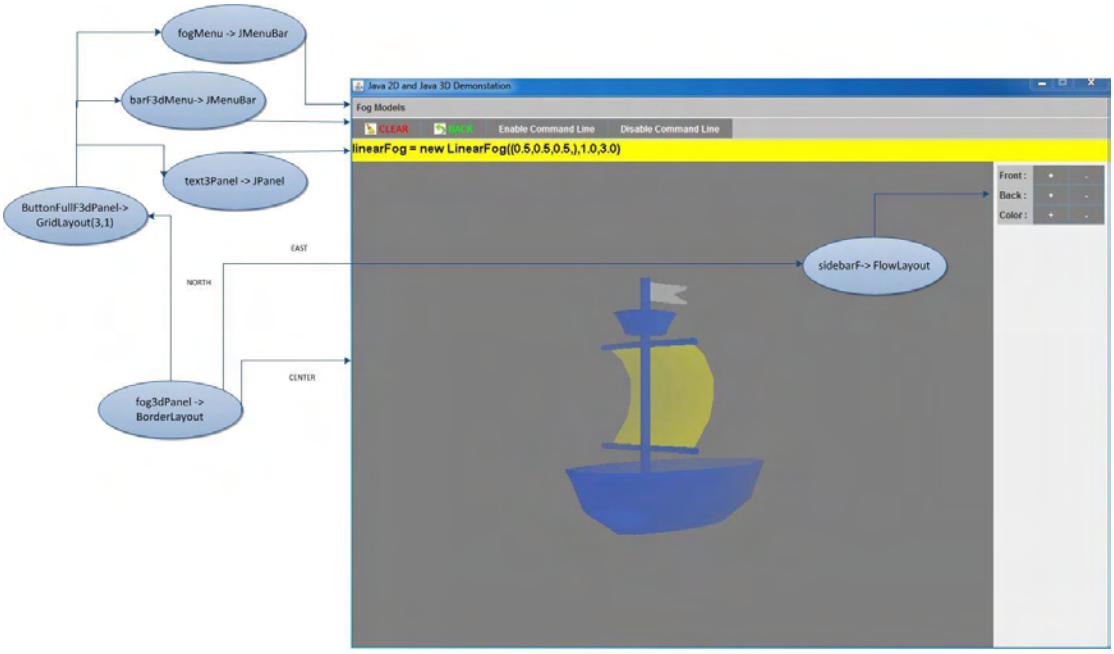
ΕΙΚΟΝΑ 193 ΚΟΥΜΠΙΑ ΓΙΑ ΕΠΑΝΑΦΟΡΑ, ΕΠΙΣΤΡΟΦΗ ΚΑΙ ΜΕΤΑΦΟΡΑ ΣΤΗΝ ΑΡΧΙΚΗ ΘΕΣΗ

14. Java 3D – 10η Επιλογη (Fog)

14.1 Παρουσιαση

Η δέκατη και τελευταία επιλογή της εφαρμογής είναι το αντικείμενο της κλάσης Fog3dPanel. Είναι ένα panel που περιέχει Swing components όπως κουμπιά και ετικέτες. Περιέχει επίσης ένα αντικείμενο της κλάσης text3Panel που είναι ένα JPanel που αποτελεί την γραμμή εντολών. Ακόμα κύριο συστατικό αυτής της επιλογής είναι ο εικονικός κόσμος που έχει δημιουργηθεί ο οποίος περιέχει ένα αντικείμενο το οποίο έχει εισαχθεί και είναι αρχείο μορφής Wavefront. Σκοπός της επιλογής αυτής είναι ο χρήστης να μπορεί να εφαρμόσει στον εικονικό κόσμο το εφέ της ομίχλης και με τα δύο διαθέσιμα μοντέλα και να διαφοροποιήσει τις παραμέτρους τους για να δει τις αλλαγές που επιφέρουν στον εικονικό κόσμο. Τέλος παρέχονται οι λειτουργίες επιστροφή στα εξώφυλλο και επαναφοράς του εικονικού κόσμου στην αρχική του κατάσταση.

Το Fog3dPanel έχει custom δομή που δημιουργείται με border Layout, Flow Layout και grid Layout.



Εικονα 194 fog3dPanel layout

14.2 Λειτουργικοτητα

14.2.1Εισαγωγη Έτοιμων Γεωμετρικων αντικειμενων

Η Java 3D δίνει την δυνατότητα στον προγραμματιστή να εισάγει στα προγράμματα που δημιουργεί διάφορους τύπους αρχείων για τρισδιάστατα αντικείμενα για γραφικά η/υ. Με αυτόν τον τρόπο μπορούν να χρησιμοποιηθούν εργαλεία μοντελοποίησης και σχεδίασης για την δημιουργία πολύπλοκων γεωμετρικών αντικειμένων που μπορούν έπειτα να ενσωματωθούν στον εικονικό κόσμο. Θα μιλήσουμε για την εισαγωγή αρχείων Wavefront αντικειμένων. Στο παράρτημα Β [Παράρτημα Β] υπάρχει ο σχετικός σύνδεσμος από όπου μπορείτε να κατεβάσετε τρισδιάστατα αντικείμενα αυτού του τύπου αρχείων.

Τα αρχεία τύπου Wavefront είναι αρχεία ASCII που περιέχουν τις ακόλουθες πληροφορίες. Οι γραμμές που περιέχουν σχόλια ξεκινούν με το σύμβολο # . Οι κορυφές που χρειάζονται για την μοντελοποίηση του τρισδιάστατου αντικειμένου ξεκινούν με το γράμμα v ακολουθούμενο από τρεις τιμές που καθορίζουν τις x, y, z συντεταγμένες της κάθε κορυφής. Με τον ίδιο τρόπο τα normal vectors προσδιορίζονται από την συντομογραφία vn (vertex normal) αντί από το γράμμα v. Η περιγραφή των πολυγώνων ξεκινά με το γράμμα f (face). Ένα πολύγωνο ορίζεται από τους δείκτες των κορυφών του. Δείκτες των αντίστοιχων normal vector ανήκουν επίσης στα πολύγωνα. Το γράμμα g (group) χρησιμοποιείται για τις ομάδες. Με αυτόν τον τρόπο τα πολύγωνα μπορούν να συνδυαστούν σε ομάδες και μπορούμε να αναφερόμαστε σε αυτά ως επιμέρους αντικείμενα (sub object). Για παράδειγμα ένα ελικόπτερο μπορεί να έχει ξεχωριστές ομάδες για το πιλοτήριο, για τον έλικα και για την ουρά. Κάθε ομάδα προσδιορίζεται από το όνομά της, έτσι η Java 3D μπορεί να έχει άμεση πρόσβαση σε αυτές τις ομάδες και για παράδειγμα να τους αναθέσει ξεχωριστά χρώματα. Για να φορτώσουμε το αρχείο Boat.obj που είναι αρχείο τύπου Wavefront σε μία Java 3D σκηνή γράφουμε :

```
ObjectFile f0 = new ObjectFile(ObjectFile.RESIZE)

Scene s0 = null;

Try

{

s0 = f0.load("Boat.obj")

}

catch (Exception e)

{

System.out.println("File loading failed:" + e)

}
```

Έπειτα το φορτωμένο αντικείμενο μπορεί να ανατεθεί ως παιδί στο transformation group loadT με την μέθοδο add Child() και χρησιμοποιώντας την μέθοδο get Scene Group().

```
loadT.addChild(s0.getSceneGroup())
```

Η μέθοδος get Name Objects() δημιουργεί έναν Hash table με ονόματα όλων των ομάδων που είναι ορισμένες στο αρχείο του φορτωμένου αντικειμένου έτσι ώστε να έχουμε πρόσβαση στα επιμέρους αντικείμενα. Για να τυπώσουμε τα ονόματα των επιμέρους αντικειμένων γράφουμε :

```
Hashtable namedObjects = s0.getNamedObjects()

Enumeration enumer = namedObjects.keys()

String name

while (enumer.hasMoreElements())

{

name = (String) enumer.nextElement()
```

System.out.println("Name: "+name)

}

Το αρχείο Boat.obj περιέχει τα επιμέρους αντικείμενα : samme01 , stamme02 , buttum , body , torn , mast , in , flagg , seil. Για να αναθέσουμε διαφορετικά χρώματα στα επιμέρους αντικείμενα για παράδειγμα στο flagg γράφουμε :

Appearance redApp = new Appearance()

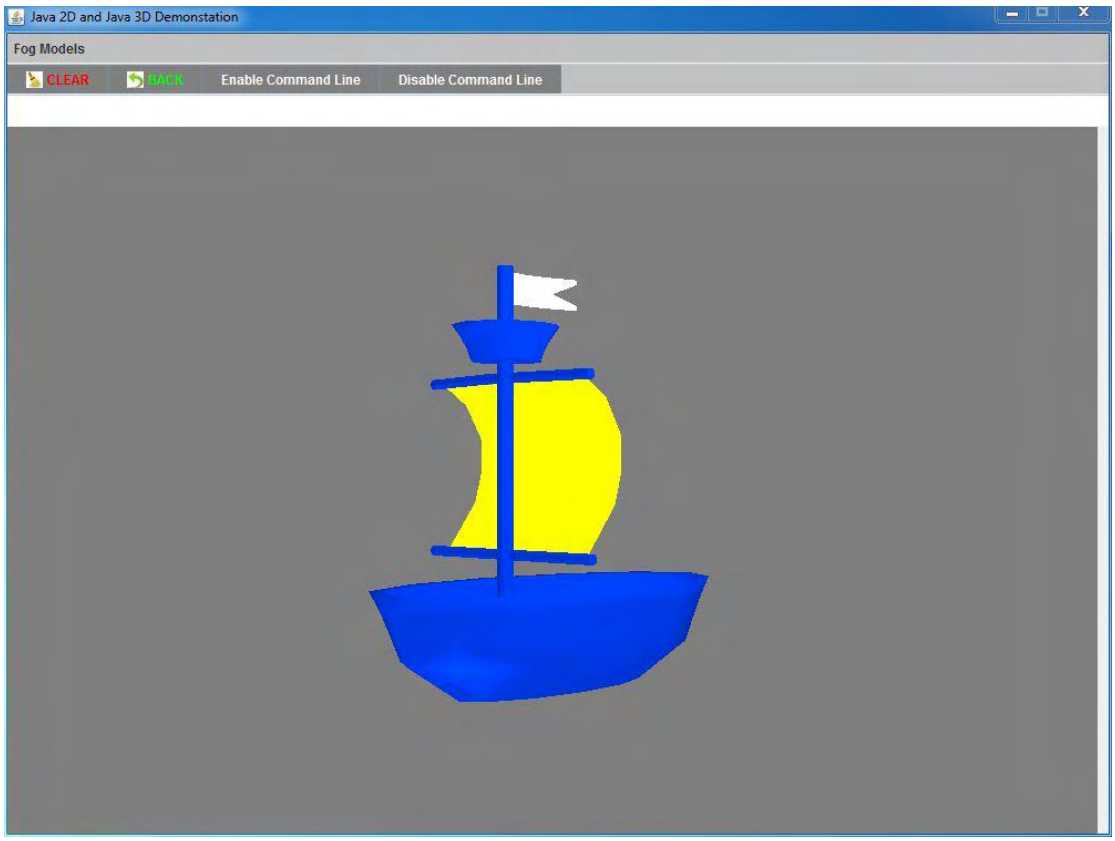
setToMyDefaultAppearance(redApp, new Color3f(1.0f,1.0f,1.0f))

Shape3D fl = (Shape3D) namedObjects.get("flagg")

fl.setAppearance(redApp)

Δημιουργούμε το αντικείμενο της εμφάνισης που θέλουμε να αναθέσουμε στο flagg και έπειτα δημιουργούμε έναν Shape3D αντικείμενο για το επιμέρους αντικείμενο flagg του Boat.obj αρχείου. Καλούμε την μέθοδο set Appearance() για το Shape3D αντικείμενο με όρισμα το αντικείμενο της επιθυμητής εμφάνισης.

Με τον παραπάνω τρόπο έχουμε εισάγει το καράβι που αντιστοιχεί στο Boat.obj αρχείο στον εικονικό κόσμο της δέκατης επιλογής της εφαρμογής έχοντας ζωγραφίσει όλα τα επιμέρους αντικείμενα. Το αρχείο Boat.obj μπορείτε να το κατεβάσετε από τον σύνδεσμο που δίνεται στο Παράρτημα Β [Παράρτημα Β].



ΕΙΚΟΝΑ 195 ΕΙΚΟΝΙΚΟΣ ΚΟΣΜΟΣ ΤΗΣ ΔΕΚΑΤΗΣ ΕΠΙΛΟΓΗΣ ΤΗΣ ΕΦΑΡΜΟΓΗΣ

14.2.2 Φόντο – Background

Για να αλλάξουμε το χρώμα του φόντου του εικονικού κόσμου απαιτείται ένα αντικείμενο της κλάσης Background. Ορίζουμε το επιθυμητό χρώμα το οποίο είναι αντικείμενο της κλάσης Color3f όπου στην συγκεκριμένη επιλογή αποτελείται από τρεις μεταβλητές γιατί πρέπει να αλλάζει ανάλογα με την επιλογή του χρήστη. Το χρώμα ανατίθεται στο Background ως παράμετρος.

Color3f fogColour = new Color3f(fog_col,fog_col,fog_col) // προεπιλεγμένη τιμή fog_col = 0.5f

Background bg = new Background(fogColour)

Πρέπει να ορίσουμε έναν όγκο μέσα στον οποίο θα έχει ισχύ το φόντο που έχουμε δημιουργήσει. Αυτό γίνεται δημιουργώντας ένα αντικείμενο της κλάσης Bounding Sphere και χρησιμοποιώντας την μέθοδο set Application Bounds().

BoundingSphere bounds = new BoundingSphere(new Point3d(0.0,0.0,0.0),Double.MAX_VALUE)

bg.setApplicationBounds(bounds)

Για να μπορέσουμε να αλλάξουμε το χρώμα του φόντου αναθέτουμε στο αντικείμενο της κλάσης Background την αντίστοιχη «ικανότητα» (capability):

bg.setCapability(Background.ALLOW_COLOR_WRITE)

14.2.3 Ομιχλη - Fog

Η ομίχλη αποτελείται από εξαιρετικά μικρές σταγόνες νερού που δεν απορροφούν το φως αλλά το διασκορπίζον προς όλες τις κατευθύνσεις. Λόγω αυτής της διασκόρπισης ή της διάχυτης ανάκλασης η ομίχλη τείνει να έχει σχεδόν λευκό χρώμα. Όταν υπάρχει ομίχλη η ορατότητα των αντικειμένων μειώνεται με την αύξηση της απόστασης. Η ομίχλη δημιουργεί ένα λευκό ή γκρι χρώμα φόντου. Το χρώμα του αντικειμένου αναμειγνύεται με το χρώμα της ομίχλης.

Η ομίχλη βασίζεται σε μία αύξουσα συνάρτηση ανάμειξης (blending function) $b : \mathbb{R}_0^+ \rightarrow [0,1]$ όπου b(0)=0 και $\lim_{d\rightarrow\infty} b(d) = 1$. Δεδομένη της απόστασης d ενός αντικειμένου από τον παρατηρητή, της έντασης του χρώματος I_{object} του αντικειμένου και της έντασης του χρώματος I_{fog} της ομίχλης , η ομίχλη υπολογίζεται με παρόμοιο τρόπο όπως η διαφάνεια με την τεχνική της παρεμβάλλουσας διαφάνειας.

$$b(d) \cdot I_{fog} + (1 - b(d)) \cdot I_{object}$$

Όταν το b(d) προσεγγίζει την τιμή 1 με την απόσταση d να αυξάνεται, το χρώμα της ομίχλης επικρατεί του χρώματος του αντικειμένου σε μεγαλύτερες αποστάσεις.

Η blending function έχει συνήθως γραμμική ή εκθετική κλίση. Η γραμμική ομίχλη δεν μειώνει την ορατότητα μέχρι την απόσταση d = d0, δηλαδή δεν γίνεται ανάμειξη του χρώματος του αντικειμένου με το χρώμα της ομίχλης. Για αποστάσεις μεγαλύτερες του d = d1, η ομίχλη επικρατεί ολοκληρωτικά τόσο ώστε αντικείμενα σε μεγαλύτερες αποστάσεις από την απόσταση d1 δεν είναι ορατά. Το εφέ της ομίχλης αυξάνεται γραμμικά μεταξύ των αποστάσεων d0 και d1. Η blending function για την γραμμική ομίχλη είναι η:

$$b(d) = \begin{cases} 0, \alpha\nu\ d \leq d0 \\ \dfrac{d - d0}{d1 - d0}, \alpha\nu\ d0 < d < d1 \\ 1, \alpha\nu\ d1 \leq d \end{cases}$$

Η πιο ρεαλιστική εκθετική ομίχλη βασίζεται σε εκθετική αύξηση του εφέ της ομίχλης ελεγχόμενη από έναν παράγοντα α>0. Η blending function είναι η :

$$b(d) = 1 - e^{-\alpha \cdot d}$$

Όσο μεγαλώνει ο παράγοντας α τόσο πιο πυκνή είναι η ομίχλη. Προτείνεται να προσαρμόζεται το χρώμα του φόντου στο χρώμα της ομίχλης. Η γραμμική και η εκθετική ομίχλη μοντελοποιούν ομίχλη με σταθερή πυκνότητα.
Για να δημιουργήσουμε γραμμική ομίχλη η Java 3D παρέχει την κλάση Linear Fog.

LinearFog linearFog = new LinearFog(fogColour,lin_front,lin_back)

Με το παραπάνω αντικείμενο της κλάσης Linear Fog δημιουργείται γραμμική ομίχλη χρώματος fogColour και αποστάσεων d0 = lin_front και d1 = lin_back της blending function. Το χρώμα fogColour είναι το χρώμα που έχει χρησιμοποιηθεί και για το φόντο της επιλογής αυτής. Οι προεπιλεγμένες τιμές για τις αποστάσεις είναι lin_front = 1.0f και lin_back = 3.0f. Πάλι πρέπει να ορίσουμε έναν όγκο μέσα στον οποίο θα έχει ισχύ το εφέ της ομίχλης. Χρησιμοποιούμε το ίδιο αντικείμενο bounds της κλάσης Bounding Sphere που χρησιμοποιήσαμε και για το φόντο στην προηγούμενη παράγραφο [14.2.2].

linearFog.setInfluencingBounds(bounds)

Για να μπορέσουμε να αλλάξουμε το χρώμα της ομίχλης όπως επίσης και τις αποστάσεις d0, d1 αναθέτουμε στο αντικείμενο της κλάσης Linear Fog τις αντίστοιχες «ικανότητες» (capabilities):

linearFog.setCapability(LinearFog.ALLOW_DISTANCE_WRITE)

linearFog.setCapability(LinearFog.ALLOW_COLOR_WRITE)

Για να δημιουργήσουμε εκθετική ομίχλη η Java 3D παρέχει την κλάση Exponential Fog.

ExponentialFog expFog = new ExponentialFog(fogColour,exp_density)

Με το παραπάνω αντικείμενο της κλάσης Exponential Fog δημιουργείται εκθετική ομίχλη χρώματος fogColour και πυκνότητας exp_density που είναι ο παράγοντας α της blending function. Η προεπιλεγμένη τιμή για τον παράγοντα α είναι 1.0f. Χρησιμοποιούμε το χρώμα και τον όγκο του φόντου που χρησιμοποιήθηκαν και στην γραμμική ομίχλη.

expFog.setInfluencingBounds(bounds)

Για να μπορέσουμε να αλλάξουμε το χρώμα της ομίχλης όπως επίσης και τον παράγοντα πυκνότητας α αναθέτουμε στο αντικείμενο της κλάσης Exponential Fog τις παρακάτω «ικανότητες» (capabilities):

expFog.setCapability(ExponentialFog.ALLOW_DENSITY_WRITE)

expFog.setCapability(ExponentialFog.ALLOW_COLOR_WRITE)

Για να μπορούμε να εναλλάσσουμε τα μοντέλα ομίχλης εύκολα αναθέτουμε σε ένα κόμβο τύπου Switch ως παιδιά την γραμμική και την εκθετική ομίχλη που δημιουργήσαμε.

Switch objSwitch = new Switch()

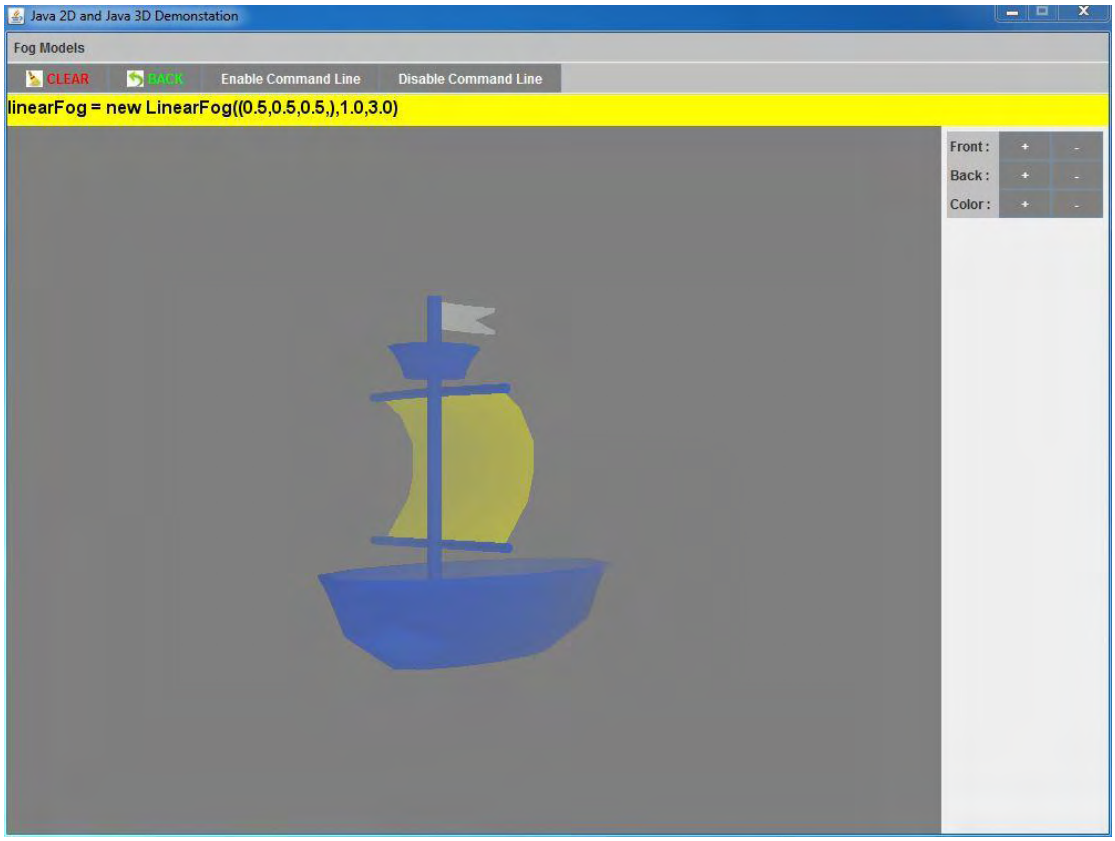
objSwitch.addChild(linearFog)

objSwitch.addChild(expFog)

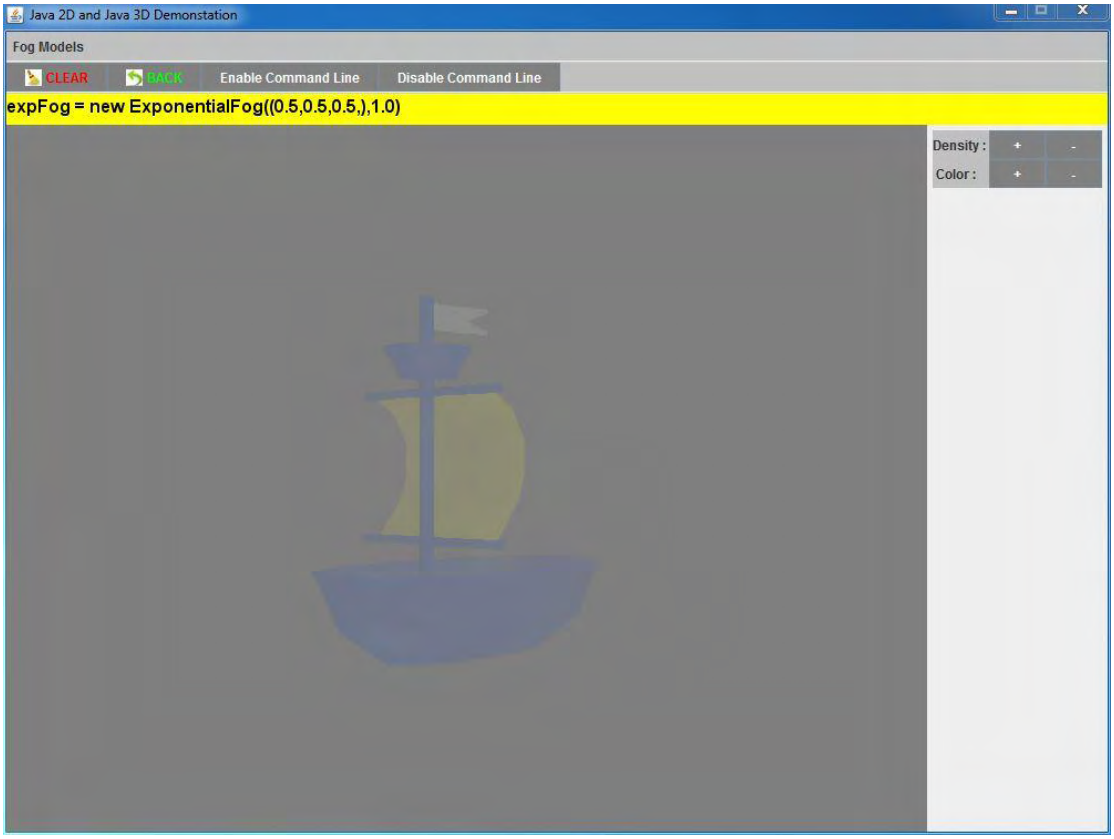
Για να μπορέσει η Java 3D να αλλάξει τις τιμές που ελέγχουν ποια παιδιά θα αποδοθούν πρέπει να ορίσουμε την παρακάτω ικανότητα (capability) στο switch node.

objSwitch.setCapability(Switch.ALLOW_SWITCH_WRITE)

Στην εφαρμογή ο χρήστης από το μενού «Fog Models» μπορεί να επιλέξει να εφαρμόσει στον εικονικό κόσμο γραμμική ή εκθετική ομίχλη επιλέγοντας «Linear» ή «Exponential». Για κάθε μοντέλο εμφανίζεται ένα νέο panel στα δεξιά του παραθύρου το οποίο περιέχει κουμπιά. Για την γραμμική ομίχλη με τα κουμπιά «+» και «-» δίπλα από τις αντίστοιχες ετικέτες ο χρήστης έχει την δυνατότητα να αλλάξει τις αποστάσεις d0 και d1 της blending function όπως επίσης και το χρώμα της ομίχλης. Για την εκθετική ομίχλη με τα κουμπιά «+» και «-» δίπλα από τις αντίστοιχες ετικέτες ο χρήστης μπορεί να αλλάξει τον παράγοντα α της blending function και το χρώμα της ομίχλης όπως και στην γραμμική ομίχλη.



ΕΙΚΟΝΑ 196 ΓΡΑΜΜΙΚΗ ΟΜΙΧΛΗ

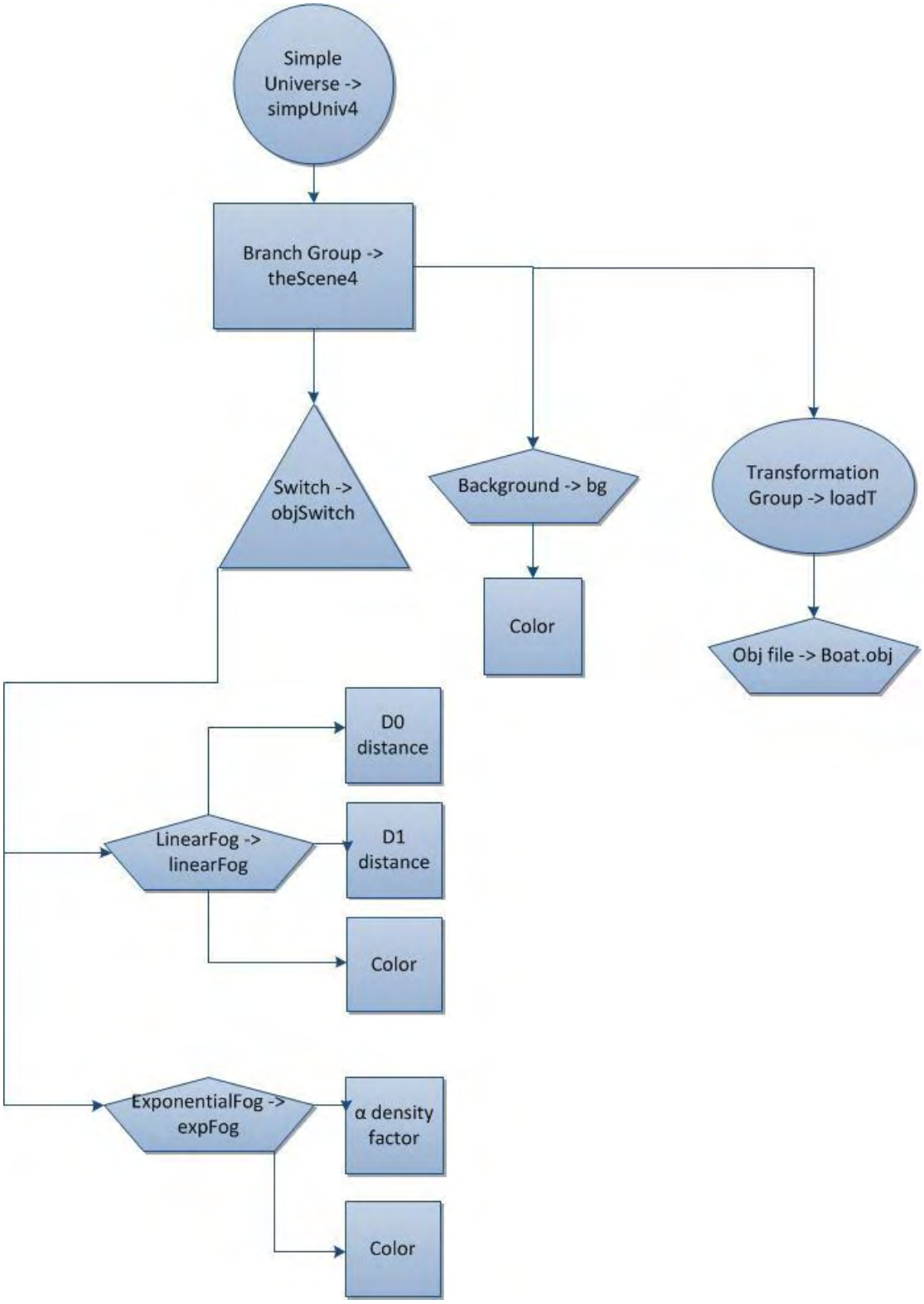


ΕΙΚΟΝΑ 197 ΕΚΘΕΤΙΚΗ ΟΜΙΧΛΗ

14.2.4 ΤΟ SCENE GRAPH ΤΗΣ ΕΠΙΛΟΓΗΣ

Στο scene graph της επιλογής αυτής περιέχονται τα αντικείμενα της σκηνής και οι πηγές φωτός.

Με κύκλο έχει σχεδιαστεί το αντικείμενο Simple Universe ενώ με ορθογώνιο έχει σχεδιαστεί ο κόμβος τύπου Branch Group. Με τρίγωνο έχει σχεδιαστεί ο κόμβος Switch ενώ με έλλειψη ο κόμβος τύπου Transformation Group. Με πεντάγωνα έχουν σχεδιαστεί τα φύλλα που είναι αντικείμενα Linear Fog, Exponential Fog και Background. Με τετράγωνα έχουν σχεδιαστεί τα node components που αναφέρονται στα φύλλα του δέντρου .



EIKONA 198 SCENE GRAPH FOG3DPANEL

14.2.5 CLEAR AND BACK – ΕΠΑΝΑΦΟΡΑ ΚΑΙ ΕΠΙΣΤΡΟΦΗ

CLEAR

Με τον όρο «clear» στην επιλογή αυτή εννοούμε να επαναφέρουμε το χρώμα του φόντου , να απενεργοποιήσουμε το εφέ της ομίχλης και να επαναφέρουμε τις παραμέτρους για τις αποστάσεις d0 , d1 της γραμμικής ομίχλης και την παράμετρο α της εκθετικής ομίχλης στις αρχικές τους τιμές. Για να επαναφέρουμε το χρώμα του φόντου χρησιμοποιούμε το αντικείμενο της κλάσης Background με την μέθοδο set Color() με παράμετρο το επιθυμητό χρώμα (γκρι). Η τιμή της μεταβλητής fog_col είναι 0.5f.

bg.setColor(fog_col, fog_col, fog_col)

Για να απενεργοποιήσουμε το εφέ της ομίχλης χρησιμοποιούμε το switch node και επιλέγουμε με την παράμετρο CHILD_NONE να μην αποδίδεται κάποιο από τα παιδιά του κόμβου με αποτέλεσμα να μην υπάρχει εφέ ομίχλης στον εικονικό κόσμο.

objSwitch.setWhichChild(Switch.CHILD_NONE)

Επαναφέρουμε τις παραμέτρους της γραμμικής και εκθετικής ομίχλης στις αρχικές τους τιμές γράφοντας :

fog3dPanel.lin_front = 1.0f

linearFog.setFrontDistance(fog3dPanel.lin_front) // απόσταση d0

fog3dPanel.lin_back = 3.0f

linearFog.setBackDistance(fog3dPanel.lin_back) // απόσταση d1

fog3dPanel.exp_density = 1.0f

expFog.setDensity(fog3dPanel.exp_density) // παράμετρος α

linearFog.setColor(fog_col, fog_col, fog_col) // θέτει το χρώμα της γραμμικής ομίχλης στο χρώμα του φόντου που έχει την αρχική του τιμή 0.5f λόγω της επαναφοράς του χρώματος του φόντου

expFog.setColor(fog_col, fog_col, fog_col) // θέτει το χρώμα της εκθετικής ομίχλης στο χρώμα του φόντου

Στην εφαρμογή για να επαναφέρουμε τον εικονικό κόσμο στην αρχική του κατάσταση πατάμε το κουμπί «Clear».

BACK

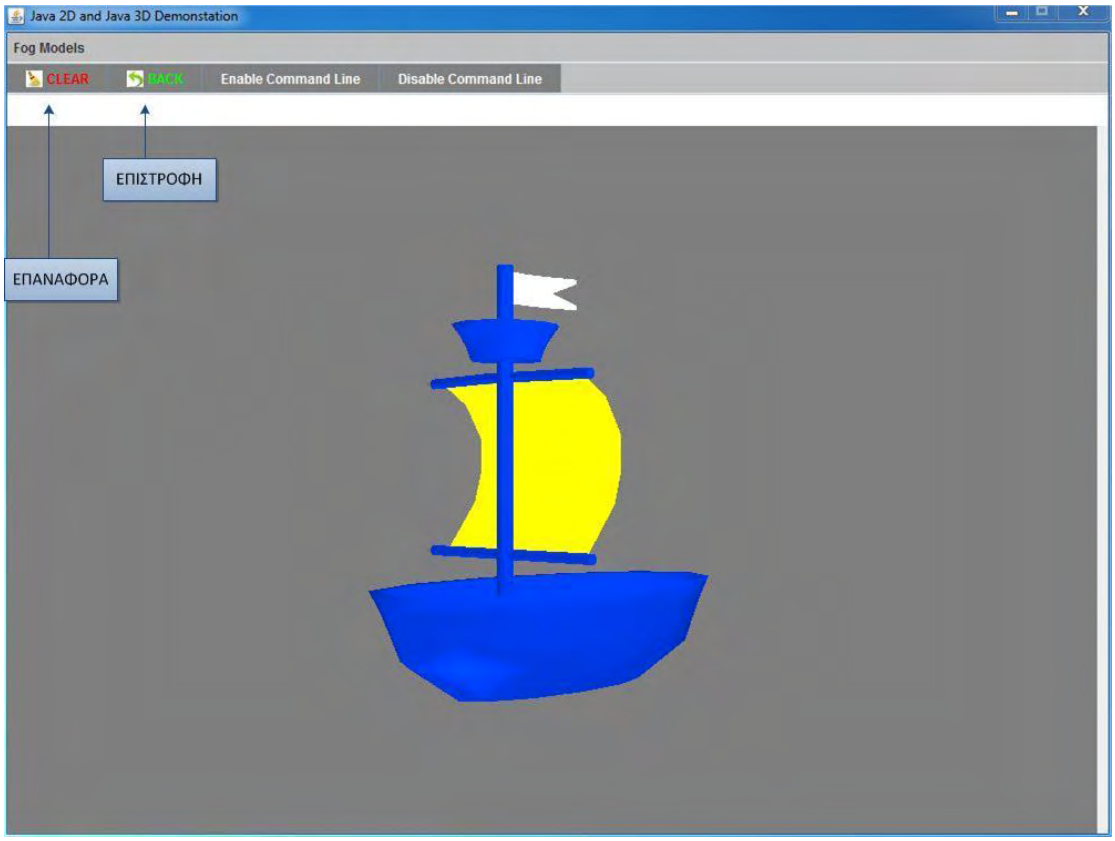
Η εφαρμογή είναι χωρισμένη σε επιλογές που επικοινωνούν μέσω του κοινού εξωφύλλου (CoverPanel). Είναι δυνατή η επιστροφή μέσα από μία εφαρμογή στο εξώφυλλο για να μας δοθεί η δυνατότητα να επιλέξουμε μία άλλη επιλογή. Για να πραγματοποιηθεί αυτό θα πρέπει όπως και στην διαδικασία της επαναφοράς (clear) να επαναφέρουμε τον εικονικό κόσμο στην αρχική του κατάσταση. Έτσι επαναφέρουμε τον εικονικό κόσμο της επιλογής στην αρχική του κατάσταση με τον κώδικα που αναφέρθηκε στο «Clear». Επίσης πρέπει να γίνει ορατό το CoverPanel της εφαρμογής και να μην είναι πλέον ορατό το panel στο οποίο βρισκόμαστε. Για την επιλογή αυτή το panel είναι το Fog3dPanel.

Fog3dPanel.setVisible(false)

setContentPane(coverPanel)

coverPanel.setVisible(true)

Στην εφαρμογή για να επιστρέψουμε στο εξώφυλλο πατάμε το κουμπί «BACK».



ΕΙΚΟΝΑ 199 ΚΟΥΜΠΙΑ ΓΙΑ ΕΠΑΝΑΦΟΡΑ ΚΑΙ ΕΠΙΣΤΡΟΦΗ

15. Επιλογος

15.1 Συμπερασματα

Στην παρούσα εργασία έγινε μία εισαγωγή στον κόσμο των γραφικών η/υ. Εξηγώντας το απαραίτητο μαθηματικό υπόβαθρο, τις αρχές, την λογική όπως επίσης και θεμελιώδεις αλγορίθμους των δισδιάστατων αλλά και τρισδιάστατων γραφικών ο αναγνώστης μπορεί να αποκτήσει βασικές γνώσεις για τα πρώτα του βήματα στην δημιουργία γραφικών η/υ. Μέσω της παρουσίασης κώδικά της εφαρμογής σε Java 2D και Java 3D κάτω από την αντίστοιχη θεωρία καθίσταται δυνατή πρακτική άσκηση πάνω στα έννοιες που μελετήθηκαν. Η εφαρμογή δίνει την δυνατότητα της οπτικής εξοικείωσης με το θεωρητικό υπόβαθρο που παρουσιάζεται στα κεφάλαια, όπως επίσης και με τις μεθόδους που χρησιμοποιούνται αλλά και με την παρατήρηση των αποτελεσμάτων της εναλλαγής των παραμέτρων των μεθόδων. Πρέπει να σημειωθεί ότι η δημιουργία γραφικών με Java 2D και Java 3D είναι μία τάση της εποχής μιας που εφαρμόζεται σε applications για κινητά τηλέφωνα με λειτουργικό Android όπως επίσης και σε αρκετές εφαρμογές για η/υ.

15.2 Προοπτικες Επεκτασης / Εξελιξης

Οι πιθανές προοπτικές εξέλιξης της παρούσας διπλωματικής εργασίας θα ήταν η ενσωμάτωση στην ήδη υπάρχουσα εφαρμογή επιλογές για κίνηση για τα δισδιάστατα αλλά και για τα τρισδιάστατα γραφικά χρησιμοποιώντας και αλληλεπίδραση με τον χρήστη όχι μόνο μέσω του ποντικιού αλλά και μέσω του πληκτρολογίου. Επιπλέον πιο συγκεκριμένα για τα τρισδιάστατα γραφικά θα μπορούσαν να υλοποιηθούν επιλογές που θα περιείχαν ήχο αλλά και αλληλεπίδραση του χρήστη με άλλα αντικείμενα του εικονικού κόσμου. Τέλος ενδιαφέρον παρουσιάζει η τεχνική του συστήματος σωματιδίων που ανήκει στο κομμάτι της εικονικής πραγματικότητας και παρουσιάζει την μοντελοποίηση εφέ όπως ο καπνός, η φωτιά, οι σπίθες, τα σύννεφα, το χιόνι και άλλα.

16. ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Introduction to Computer Graphics Using Java 2D and Java 3D, Second Edition, Frank Klawonn

[2] Διαλέξεις μαθήματος «Γραφικά Η/Υ» της διδάσκουσας Επίκουρης Καθηγήτριας κ. Παναγιώτας Τσομπανοπούλου, Πανεπιστήμιο Θεσσαλίας
http://inf-server.inf.uth.gr/courses/CE416/

[3] Programmer's Guide to the Java^{TM} 2D API
http://docs.oracle.com/javase/6/docs/technotes/guides/2d/spec/j2d-bookTOC.html

[4] 2D Graphics – The Java^{TM} Tutorials , Sun Microsystems
http://docs.oracle.com/javase/tutorial/2d/TOC.html

[5] Java 2D API Specification
http://docs.oracle.com/javase/6/docs/technotes/guides/2d/spec.html

[6] Java 2D Graphics, Jonathan Knudsen

[7] Oracle Article – Learning Java 2D, Part 1
http://www.oracle.com/technetwork/articles/javase/java2dpart1-137217.html

[8] Oracle Article – Learning Java 2D, Part 2
http://www.oracle.com/technetwork/articles/javase/java2dpart2-139741.html

[9] The Java 3D API Specification , Sun Microsystems
http://www-evasion.imag.fr/~Francois.Faure/enseignement/ressources/java/j3dguide/j3d-webTOC.html

[10] Developer.com – Richard G. Baldwin's article :
Processing Image Pixels, Applying Image Convolution in Java
http://www.developer.com/java/ent/article.php/3590351/Processing-Image-Pixels-Applying-Image-Convolution-in-Java.htm

[11] Developer.com – Richard G. Baldwin's article :
Processing Image Pixels, Applying Image Convolution in Java, Part 2
http://www.developer.com/java/other/article.php/3596351/Processing-Image-Pixels-Applying-Image-Convolution-in-Java-Part-2.htm

[12] Developer.com – Richard G. Baldwin's article :
Using the Java 2D Lookup Op Filter Class to Process Images
http://www.developer.com/java/other/article.php/3654171/Using-the-Java-2D-LookupOp-Filter-Class-to-Process-Images.htm

[13] Developer.com – Richard G. Baldwin's article :
Using the Java 2D Lookup Op Filter Class to Scramble and Unscramble Images
http://www.developer.com/java/other/article.php/3681466/Using-the-Java-2D-LookupOp-Filter-Class-to-Scramble-and-Unscramble-Images.htm

[14] Developer.com – Richard G. Baldwin's article :
Using the Java 2D Color Convert Op and Rescale Op Filter Classes to Process Images
http://www.developer.com/java/other/article.php/3698981/Using-the-Java-2D-ColorConvertOp-and-RescaleOp-Filter-Classes-to-Process-Images.htm

[15] Wikipedia Article :
Computer Graphics
http://en.wikipedia.org/wiki/Computer_graphics#Applications

[16] Styling digital images with CONVOLVE OP
http://www.java-tips.org/java-se-tips/java.awt.image/styling-digital-images-with-convolveop.html
[17] Wikipedia Articles :
Abstract Window Toolkit
http://en.wikipedia.org/wiki/Abstract_Window_Toolkit#cite_note-2
[18] Wikipedia Article :
Swing (Java)
http://en.wikipedia.org/wiki/Swing_(Java)
[19] Wikipedia Article :
Java 2D
http://en.wikipedia.org/wiki/Java_2D
[20] Wikipedia Article :
Java 3D
http://en.wikipedia.org/wiki/Java_3D
[21] Curves
http://www.e-cartouche.ch/content_reg/cartouche/graphics/en/html/unit_Curves.html
[22] Wolfram Mathworld – Bezier Curve
http://mathworld.wolfram.com/BezierCurve.html
[23] Wolfram Mathworld – Bernstein Polynomial
http://mathworld.wolfram.com/BernsteinPolynomial.html
[24] Java Notes
http://www.leepoint.net/notes-java/index.html
[25] JH Labs – Blurring for Beginners
http://www.jhlabs.com/ip/blurring.html
[26] Διαλέξεις μαθήματος «Computer Game Technologies», Πανεπιστήμιο Stirling
http://www.cs.stir.ac.uk/courses/CSC9N6/lectures/
[27] Java 3D Tutorial
http://www.java3d.org/tutorial.html
[28] Java 3D Tutorials, Jonathan Miyamoto
http://www.vrupl.evl.uic.edu/LabAccidents/java3d/
[29] Java 3D Programming, Daniel Selman
http://www.tecgraf.puc-rio.br/~ismael/Cursos/Cidade_CG/labs/Java3D/Java3D_onlinebook_selman/onlinebook.html
[30] Διαλέξεις μαθήματος «Graphics Programming : Java »,Gary Hill, Πανεπιστήμιο Northampton
Java 2D : http://www.computing.northampton.ac.uk/~gary/csy3019/CSY3019SectionC.html
Java 3D : http://www.computing.northampton.ac.uk/~gary/csy3019/CSY3019SectionD.html
[31] http://www.java3d.nl/

17. Παραρτημα Α

17.1 Συναρτηση draw Simple Coordinate System()

```
public void drawSimpleCoordinateSystem(int xmax, int ymax,Graphics2D g2){
        int xOffset = 0;
        int yOffset = 0;
        int step = 20;
        String s;
        Font fo = g2.getFont();
        int fontSize = 11;
        Font fontCoordSys = new Font("Arial", Font.BOLD, fontSize);
        AffineTransform flip = new AffineTransform();
        flip.setToScale(1,-1);
        AffineTransform lift = new AffineTransform();
        lift.setToTranslation(0,fontSize);
        flip.preConcatenate(lift);
        Font fontUpsideDown = fontCoordSys.deriveFont(flip);
        g2.setFont(fontUpsideDown);
        g2.drawLine(xOffset,yOffset,xmax,yOffset);
        for (int i=xOffset+step; i<=xmax; i=i+step)
        {
          g2.drawLine(i,yOffset-2,i,yOffset+2);
          g2.drawString(String.valueOf(i),i-7,yOffset-30);
        }
        g2.drawLine(xOffset,yOffset,xOffset,ymax);
        s="  ";
        for (int i=yOffset+step; i<=ymax; i=i+step)
        {
          g2.drawLine(xOffset-2,i,xOffset+2,i);
          if (i>99){s="";}
          g2.drawString(s+String.valueOf(i),xOffset-25,i-20);
```

```
            }
            g2.setFont(fo);
          }
          public void setTrans(Trans trans){
            in_trans=trans;
          }
      }
```

17.2 Συναρτηση set To My Default Appearance()

```
public  void setToMyDefaultAppearance(Appearance app, Color3f col){
            app.setMaterial(new Material(col,col,col,col,150.0f));
}
```

18. Παραρτημα Β

- Java 3D download
  http://java3d.java.net/binary-builds.html
  Οδηγίες για την εγκατάσταση της Java 3D ανά λειτουργικό
  http://download.java.net/media/java3d/builds/release/1.5.2/README-download.html
  Εισαγωγή των jar αρχείων στον IDE Netbeans
  http://link-voyager.blogspot.gr/2010/11/installing-java-3d-linking-with.html
  Εισαγωγή των jar αρχείων στον IDE Eclipse
  http://www.cs.tufts.edu/~jacob/86/java3d.html
- .obj files download
  http://people.sc.fsu.edu/~jburkardt/data/obj/obj.html
- Boat.obj download
  https://www.dropbox.com/s/tx4lixl2fs7bslr/Boat.obj