3D Δυναμικό Puzzle

Σε X3D-Xj3D

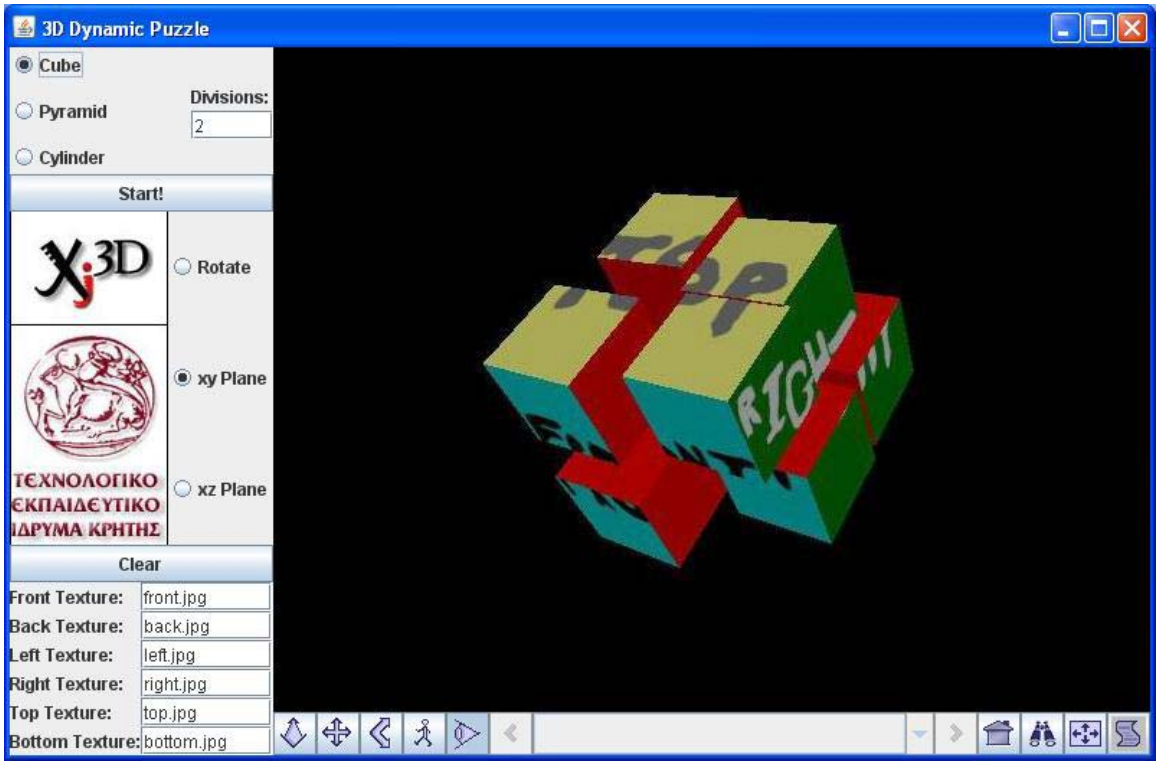
Πτυχιακή Εργασία

ΜαριούΓεώργίου

Υπεύθυνος Καθηγητής:
Μαλάμος Αθανάσιος

ΠΕΡΙΕΧΟΜΕΝΑ

1 Η ΛΕΙΤΟΥΡΓΙΚΟΤΗΤΑ ΤΗΣ ΕΦΑΡΜΟΓΗΣ



1.1 Γενικά

Το 3D Δυναμικό Puzzle είναι μία εφαρμογή-παιχνίδι η οποία επιτρέπει στον χρήστη να επιλέξει ένα από τα τρία διαθέσιμα γεωμετρικά σχήματα: κύβο, πυραμίδα και κύλινδρο, να διακοσμίσει τις πλευρές τους με εικόνες ή φωτογραφίες της προτίμησής του και να επιλέξει τον αριθμό των κομματιών στα οποία θα διασπαστεί το αρχικό σχήμα. Αφού ο χρήστης κάνει τις αρχικές επιλογές και το σχήμα σπάσει, καλείται να το επανασυναρμολογήσει κινόντας και περιστρέφοντας τα κομμάτια και στις τρείς διαστάσεις.

1.2 Διαμόρφωση Αρχικών Επιλογών

Επιλογή Σχήματος: Η επιλογή γεωμετρικού σχήματος γίνεται από την τριάδα radio button στο βόρειοδυτικό μέρος του πάνελ χειρισμού.

Επιλογή εικόνων: Η σχετική διαδρομή των εικόνων που θα χρησιμοποιηθούν εισάγεται στα πεδία κειμένου που βρίσκονται στο νότιο τμήμα του πάνελ χειρισμού. Ο αριθμός και το όνομα των πεδίων κειμένου αλλάζει σύμφωνα με την επιλογή σχήματος έτσι ώστε να συμφωνεί με τις ιδιαιτερότητες του κάθε γεωμετρικού σχήματος.

Επιλογή κατατμήσεων: Ο αριθμός των κατατμήσεων εισάγεται στο πεδίο κειμένου που τιτλοφορείται “Divisions:” στο βορειοανατολικό τμήμα του πάνελ χειρισμού. Ο αριθμός αυτός εφαρμόζεται διαφορετικά στο κάθε σχήμα.

Α) Στον κύβο αφορά τις κατατμήσεις ανα ακμή, δίνοντας συνολικό αριθμό κομματιών την τρίτη δύναμη των επιλεγμένων κατατμήσεων. Π.χ. για τιμές 1, 2, 3 θα εμφανίστουν 1, 8, 27 κομμάτια αντίστοιχα.

Β) Στην πυραμίδα αφορά τις κατατμήσεις στις ακμές που ενώνουν την κορυφή με τη βάση. Δίνει συνολικό αριθμό κομματιών $\Sigma(\lambda^2)$ για λ από 1 μέχρι τον επιλεγμένο αριθμό των κατατμήσεων. Π.χ. για τιμές 1, 2, 3 θα εμφανιστούν 1, 5, 14 κομμάτια αντίστοιχα.

Γ) Στον κύλινδρο αφορά τις κάθετες κατατμήσεις των κυκλικών δίσκων, δίνοντας συνολικό αριθμό κομματιών ίσο με αυτόν των επιλεγμένων κατατμήσεων. Π.χ. για τιμές 1, 2, 3 θα εμφανιστούν 1, 2, 3 κομμάτια αντίστοιχα.

1.3 Εμφάνιση και Καθαρισμός Puzzle

Όταν ο χρήστης καθορίσει τις αρχικές επιλογές μπορεί να εμφανίσει το κατακερματισμένο σχήμα πατώντας το κουμπί Start που βρίσκεται πάνω από το λογότυπο Xj3d στο πάνελ χειρισμού. Ο χρήστης μπορεί επίσης, εάν για οποιονδήποτε λόγο το επιθυμεί, να αποσύρει το εμφανισμένο puzzle πατώντας το κουμπί Clear που βρίσκεται κάτω από το λογότυπο στο πάνελ χειρισμού. Μετά την επιλογή Clear δεν υπάρχει δυνατότητα επαναφοράς του αποσυρμένου puzzle.

Εάν ο χρήστης επιθυμεί να επανεκκινήσει το puzzle πρέπει πρώτα να καθαρίσει το χώρο του πατώντας Clear και στη συνέχεια να πατήσει ξανα Start!. Παρομοίως ο χρήστης μπορεί να εμφανίσει κομμάτια δύο διαφορετικών αρχικών σχημάτων εμφανίζοντας πρώτα του ενός και στη συνέχεια αλλάζοντας τις επιλογές και πατώντας ξανα Start!, παραλείποντας το Clear.

1.4 Χειρισμός Κομματιών

Τα κομμάτια στο 3D Δυναμικό Puzzle μπορούν να κινηθούν και να περιστραφούν προς όλες τις κατευθύνσεις. Ο μεσαίος τομέας δεξιά από το λογότυπο στο πάνελ χειρισμού διαθέτει τρείς επιλογές κίνησης.

i) Rotate: Έχοντας αυτή την επιλογή κίνησης τσεκαρισμένη και κάνοντας click & drag στο κομμάτι που μας ενδιαφέρει προκαλούμε την περιστροφή του σύμφωνα με την κίνηση του ποντικιού.

ii) xy Plane: Έχοντας αυτή την επιλογή κίνησης τσεκαρισμένη και κάνοντας click & drag στο κομματι που μας ενδιαφέρει προκαλούμε την κίνηση του στο επίπεδο μήκου-ύψους, xy στα καρτεσιανά συστήματα συντεταγμένων.

iii) xz Plane: Έχοντας αυτή την επιλογή κίνησης τσεκαρισμένη και κάνοντας click & drag στο κομματι που μας ενδιαφέρει προκαλούμε την κίνηση του στο επίπεδο μήκους-βάθους, xz στα καρτεσιανά συστήματα συντεταγμένων.

1.5 Περιήγηση και Οπτικές Γωνίες

Υπάρχουν πέντε διαφορετικοί τρόποι να προσαρμόσουμε την οπτική γωνία, καθώς και η επιλογή επιστροφής στην αρχική θέση.

Α) Το πρώτο κουμπί από αριστερά στο κάτω μέρος του παραθύρου της εφαρμογής (πάνελ περιήγησης) αντιστοιχεί στην περιήγηση τύπου Fly. Η περιήγηση Fly μας δίνει τη δυνατότητα να κινηθούμε ευθύγραμμα προς οποιαδήποτε κατεύθυνση δείχνει ο κέρσορας κατα τη διάρκεια του click & drag.

Β) Το δεύτερο κουμπί του πάνελ περιήγησης αντιστοιχεί στην περιήγηση τύπου Pan. Η περιήγηση Pan μας δίνει τη δυνατότητα να κινήσουμε την οπτική μας θέση σε άξονα μήκους-ύψους (xy) ενώ διατηρούμαστε στο ίδιο βάθος.

Γ) Το τρίτο κουμπί του πάνελ περιήγησης αντιστοιχεί στην περιήγηση τύπου Tilt. Η περιήγηση Tilt μας δίνει τη δυνατότητα να αλλάξουμε τον προσανατολισμό της οπτικής μας χωρίς να αλλάξουμε τη θέση της.

Δ) Το τέταρτο κουμπί του πάνελ περιήγησης αντιστοιχεί στην περιήγηση τύπου Walk. Η περιήγηση Walk προσομοιώνει το βάδισμα σε ένα νοητό οριζόντιο επίπεδο, δίνοντας μας τη δυνατότητα να κινηθούμε προς οποιαδήποτε κατεύθυνση γύρω, αλλά απαγορεύοντας την κίνηση πάνω ή κάτω από το νοητό αυτό επίπεδο.

Ε) Το πέμπτο, τελευταίο κουμπι το πανελ περιήγησης αντιστοιχεί στην περιήγηση τύπου Examine. Η περιήγηση Examine είναι και η προεπιλεγμένη καθώς είναι η πιο πρακτική για τη λύση του puzzle. Περιστρέφει φαινομενικά τα κομμάτια ως σύνολο, αν και στην πραγματικότητα κινεί την οπτική θέση στην επιφάνεια μιας σφαίρας γύρω από το κέντρο του puzzle.

Το κουμπί με το σπιτάκι στο κάτω δεξί μέρος της οθόνης μας επιστρέφει στην αρχική οπτική μας θέση μπροστά από το puzzle.

2 Η ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΗΣ ΕΦΑΡΜΟΓΗΣ

2.1 Εισαγωγή Στο Πρότυπο X3D:

Η γλώσσα X3D αποτελεί το νέο πρότυπο ISO στον τομέα της κωδικοποιήσης 3D γραφικών στους υπολογιστές, το οποίο δημιουργήθηκε και επεκτείνεται από το μη κερδοσκοπικό, αυτοχρηματοδοτούμενο οργανισμό Web3D consortium. Αποτελεί τον διάδοχο της VRML (Virtual Reality Markup Language) διατηρόντας την ονοματολογία και την κατηγοριοποίηση των αντικειμένων, υποστηρίζει όμως ταυτόχρονα την XML (Extended Markup Language) κωδικοποίηση, καθώς και διαθέτει πιο εξελιγμένα APIs (Application Programming Interface). Το αποτέλεσμα είναι μια απλή και αποτελεσματική περιγραφική γλώσσα, η οποία χρησιμοποιεί τα γνωστά xml εμφωλευμένα tags μορφής

<object>

<property>…</property>

</object>

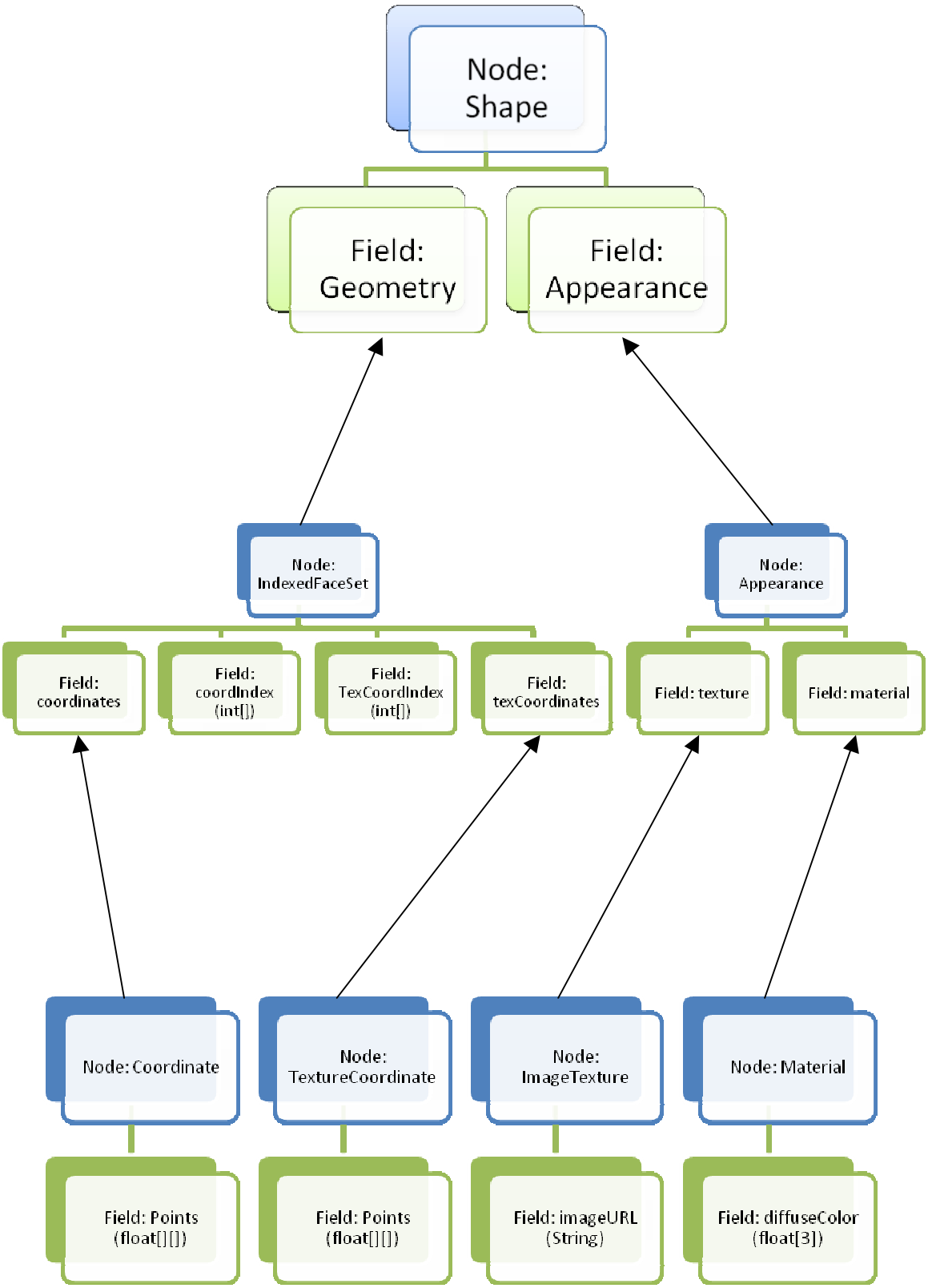
για να ορίσει τη σκηνή, τα αντικείμενα και τις ιδιότητες τους, όπως αυτές είναι γνωστές από την αρκετά διαδεδομένη vrml.

Λόγω της XML φύσης της η X3D καθιστά εύκολη την ενσωμάτωση τρισδιάστατων σκηνών σε δικτυακές εφαρμογές, ακόμα και σε ιστοσελίδες, προάγοντας έτσι την πρωην VRML σε μια γλώσσα η οποία μπορεί να χρησιμοποιηθεί πλέον, εκτός για σκοπούς μοντελοποίησης, και για κοσμητικούς σκοπούς ή για τη δημιουργία τρισδιάστατων διεπαφών. Οποιοσδήποτε browser μπορεί να προβάλει μιά X3D σκηνή απλώς με την εγκατάσταση του κατάλληλου plug-in, το οποίο διατίθεται δωρεάν στο διαδίκτυο.

2.2 Το Xj3D Toolkit και Η Ιεραρχία Των X3DNodes:

Όπως αναφέρθηκε παραπανω το X3D χρησιμοποιείται αποκλειστηκά για την κωδικοποίηση μιας 3D σκηνής, και όχι για την δυναμική επεξεργασία της. Γι' αυτόν το λόγο το Web3D consortium δημιούργησε το Xj3D Toolkit. Το Xj3D Toolkit επιτρέπει την ενσωμάτωση X3D σκηνών σε οποιοδήποτε java πρόγραμμα, καθώς το πακέτο SAI (Scene Access Interface) ειδικότερα περιέχει τις κλάσεις και μεθόδους που θα χρειαστεί κάποιος για να δημιουργήσει ή να επεξεργαστεί μια X3D σκηνή δυναμικά, κατα το χρόνο εκτέλεσης του προγράμματος (runtime). Είναι απαραίτητο να σημειωθεί πως το Xj3D Toolkit είναι ένα υπο ανάπτυξη πακέτο, το οποίο όμως διορθώνεται και επεκτείνεται συνεχώς με μέση περίοδο ενημέρωσης τους 4 μήνες. Αυτό συνεπάγεται την ύπαρξη αρκετών bugs, τα οποία όπως και στο συγκεκριμένο project χρειάζεται να λυθούν προσωρινά με κάποιο work-around.

Το Xj3D αντιμετωπίζει κάθε tag της X3D σκηνής ως ένα αντικείμενο τύπου X3DNode. Για κάθε διαφορετικό tag που θέλουμε να υλοποιήσουμε υπάρχει και ο κατάλληλος απόγονος της κλάσης X3DNode που θα πρέπει να ορίσουμε. Επειδή κάποιες ιδιότητες των nodes αποτελούνται από καθαρά δεδομένα, ενώ άλλες από σειρές νέων X3DNodes, ο προγραμματιστής πρέπει να γνωρίζει την ιεραρχία των nodes. Η ιεραρχία των nodes που χρησιμοποιήθηκαν για την υλοποίηση κάθε σχήματος (shape) σε αυτό το project φαίνεται στη σελίδα που ακολουθεί.


Node: Shape
Field: Geometry
Field: Appearance
Node: IndexedFaceSet
Node: Appearance
Field: coordinates
Field: coordIndex (int[])
Field: TexCoordIndex (int[])
Field: texCoordinates
Field: texture
Field: material
Node: Coordinate
Node: TextureCoordinate
Node: ImageTexture
Node: Material
Field: Points (float[][])
Field: Points (float[][])
Field: imageURL (String)
Field: diffuseColor (float[3])

2.3 Γενική Περιγραφή Των Κλάσεων Του 3D Dynamic Puzzle:

Οι κλάσεις οι οποίες αποτελούν το 3D δυναμικό puzzle, και θα επεξηγηθούν αναλυτικότερα αργότερα, είναι οι παρακάτω έξι:

Α) Η κλάση CreateScene είναι η κεντρική κλάση της εφαρμογής, η οποία περιέχει τις βασικές εντολές για την εμφάνιση και διαχείρηση της διεπαφής, και την υλοποίηση και ομαδοποίηση σχημάτων σύμφωνα με τις προτιμήσεις του χρήστη. Λειτουργεί ως η ραχοκοκκαλιά της εφαρμογής, καλώντας τις υπολοιπες μεθόδους και κλάσεις κατάλληλα ώστε να παρουσιάσει το επιθυμητό αποτέλεσμα.

Β) Η κλάση Point λειτουργεί στοιχειωδώς για να ορίσει ένα αντικείμενο-σημείο τριών συντεταγμένων το οποίο χρησιμοποιείται στη συνέχεια στη δημιουργία των σχημάτων.

Γ) Η CubeFacesPoints είναι η κλάση που υλοποιείται όταν ο χρήστης επιλέξει ως γεωμετρία του puzzle τον κύβο. Δημιουργεί πίνακες σημείων για τις έξι τετράγωνες πλευρές ζητούμενου μήκους που θα αποτελέσουν κομμάτια του «σπασμένου κύβου» καθώς και αντιστοιχεί τα κατάλληλα σημεία του texture όπου αυτό υφίσταται.

Δ) Η PyramidFacesPoints έχει την ίδια λειτουργία με την CubeFacesPoints με μόνη διαφορά ότι δημιουργεί πίνακες σημείων για τις πλευρές μίας πυραμίδας. Καλείται για να εμφανίσουμε είτε το σχήμα πυραμίδα ως ακέραια λύση του puzzle, είτε ως το κορυφαίο κομμάτι της «σπασμένης πυραμίδας».

Ε) Η TrapezeFacesPoints δημιουργεί τους πίνακες σημείων για τις πλευρές των τραπεζίων που προκύπτουν από το σπάσιμο μιάς πυραμίδας. Λειτουργεί σε αντιστοιχία με τις παραπάνω –FacesPoints κλάσεις.

Στ) Η CylinderFacesPoints δημιουργεί επίσης πίνακες σημείων και λειτουργεί σε αντιστοιχία με τις παραπάνω –FacesPoints κλάσεις, με μόνη διαφορά ότι εδώ οι «φέτες» του κυλίνδρου κόβονται βάση της ζητούμενης

γωνίας που αναλογεί σε κάθε κομμάτι, οπότε και ο καταμερισμός γίνεται μετρούμενος σε ακτίνια.

2.4 Δυναμική Δημιουργία Ενός Σχήματος:

Όλα τα σχήματα της εφαρμογής, όπως φαίνεται και στο διάγραμμα ιεραρχίας που προηγείται, υλοποιούνται ως X3DNodes τύπου IndexedFaceSet. Η X3D κωδικοποίηση περιλαμβάνει ορισμούς για απλά σχήματα όπως ο κύβος και ο κύλινδρος, αυτοί οι ορισμοί όμως δεν θα μπορούσαν να χρησιμοποιηθούν σε αυτη την εφαρμογή λόγω του τρόπου διαχείρησης του texture. Το texture κάθε X3DNode που ορίζει σχήμα εμφανίζεται ενιαίο σε κάθε πλευρά του σχήματος, κάτι που δεν είναι καθόλου πρακτικό για μια εφαρμογή puzzle. Προκειμένου να παρουσιάσουμε διαφορετικά textures, και μάλιστα κομμάτια μιας ευρύτερης εικόνας, σε διαφορετικές πλευρές του ίδιου σχήματος πρέπει να υιοθετήσουμε ως τύπο σχήματος το IndexedFaceSet για να ορίσουμε τις πλευρές του σχήματός μας. Ομαδοποιώντας στη συνέχεια τα διαφορετικά IndexedFaceSet έχουμε το ζητούμενο αποτέλεσμα.

Η μέθοδος που ορίζει και δημιουργεί τα νέα IndexedFaceSet σχήματα είναι η addFace η οποία περιέχεται στην κλάση CreateScene. Η addFace κατα την κλήση της δέχεται πέντε ορίσματα:

1) Float[][] points: Το πρώτο όρισμα στην μέθοδο addFace είναι ένας πίνακας από τριάδες float, οι οποίες ορίζουν τα ακραία σημεία της πλευράς που θα υλοποιηθεί. Στο IndexedFaceSet μόνο η μια όψη του δισδιάστατου αντικειμένου είναι ορατή, γι' αυτό και η σειρά με την οποία αναφέρονται τα σημεία αυτά πρέπει να είναι τέτοια ώστε να διαδέχονται το ένα το άλλο με φορά αντίθετη του ρολογιού ως προς την ορατή μεριά τους.

2) Float[][] texturepoints: Το δεύτερο όρισμα είναι ένας πίνακας από δυάδες float ισάριθμες με αυτές του πίνακα points[][]. Αντιπροσωπεύουν τα σημεία επί του αρχικού texture, δύο διαστάσεων μιάς και πρόκειται για εικόνα, που αντιστοιχούν σε κάθε σημείο του χώρου του πίνακα points[][].

3) String texturePath: Το τρίτο όρισμα είναι η διαδρομή και το όνομα αρχείου του texture που θα χρησιμοποιηθεί για το συγκεκριμένο IndexedFaceSet, σε μορφή μεταβλητής τύπου string.

4) X3DNode parent: Το τέταρτο αυτό όρισμα αφορά το group στο οποίο θα προστεθεί το IndexedFaceSet μαζί με τα υπόλοιπα που θα αποτελέσουν το τελικό σχήμα. Πιο συγκεκριμένα ο τύπος του X3DNode που χρησιμοποιείται για να ομαδοποιήσει τις πλευρές είναι το X3DProtoInstance, για λόγους που θα εξηγηθούν αργότερα.

5) Int arreyLength: Το τελευταίο όρισμα είναι ένας ακαίρεος ο οποίος δηλώνει τον αριθμό των σημείων που αποτελούν την πλευρά που υλοποιούμε. Αυτό το όρισμα έχει προστεθεί γιατί στην εφαρμογή υπάρχουν πλευρές τριών σημείων, όπως αυτές της πυραμίδας, αλλά και τεσσάρων σημείων, όπως αυτές του κύβου. Το όρισμα αυτό θα μπορούσε να έχει παραληφθεί μετρώντας απλώς το lengthOf του πίνακα points[][] αλλά η ύπαρξη του εξυπηρετεί σκοπούς χειρισμού εξαιρέσεων και επαλήθευσης.

Τα στιγμιότυπα των nodes που πρόκειται να δημιουργήσουμε προέρχονται από ένα αντικείμενο τύπου X3DScene, το οποίο αντιπροσωπεύει συνολικά ένα X3D σκηνικό που πρόκειται να δημιουργήσουμε ή να επεξεργαστούμε. Για να υλοποιήσει δυναμικά ένα σχήμα η addFace ακολουθεί τα παρακάτω βήματα σύμφωνα με το σχέδιο ιεραρχίας που αναφέρθηκε προηγουμένως. Μέσω του X3DScene αντικειμένου mainScene δημιουργούμε τα nodes που χρειαζόμαστε, τα οποία στο διάγραμμα ιεραρχίας φαίνονται με μπλε χρώμα. Π.χ. για να ξεκινήσουμε πρεπει να έχουμε ένα αντικείμενο shape, οπότε και καλούμε: (1)

```
X3DNode shape = mainScene.createNode("Shape");
```

Ένα shape αντικείμενο έχει δύο χαρακτηριστικά (fields): την γεωμετρία και την εμφάνιση. Προκειμένου να δώσουμε στα πεδία τις τιμές που θέλουμε πρέπει να τις δέσουμε με μεταβλητές. Οι μεταβλητές αυτές μπορούν να είναι

γενικά τύπου X3DField, ή ειδικότερα του τύπου του δεδομένου που πρόκειται να δεχτούν. Το δεύτερο είναι σαφώς σωστότερο. Για να δέσουμε το πεδίο geometry (για αρχή) με αντίστοιχη μεταβλητή καλούμε: (2)

SFNode shape_geometry = (SFNode) (shape.getField("geometry"));

Ο τύπος SFNode σημαίνει πως η μεταβλητή θα δεθεί με ένα node –μόνο ένα γιατί κατ' αντιστοιχία υπάρχει και η MFNode (single-multi). Ο ζητούμενος τύπος γεωμετρίας είναι το IndexedFaceSet. Αφού το node IndexedFaceSet δημιουργηθεί ακριβώς όπως και το shape στο παράδειγμα (1), τότε είναι έτοιμο να ταυτιστεί με τη μεταβλητή και συνεπώς με το πεδίο του σχήματος: (3)

shape_geometry.setValue(indexedFaceSet);

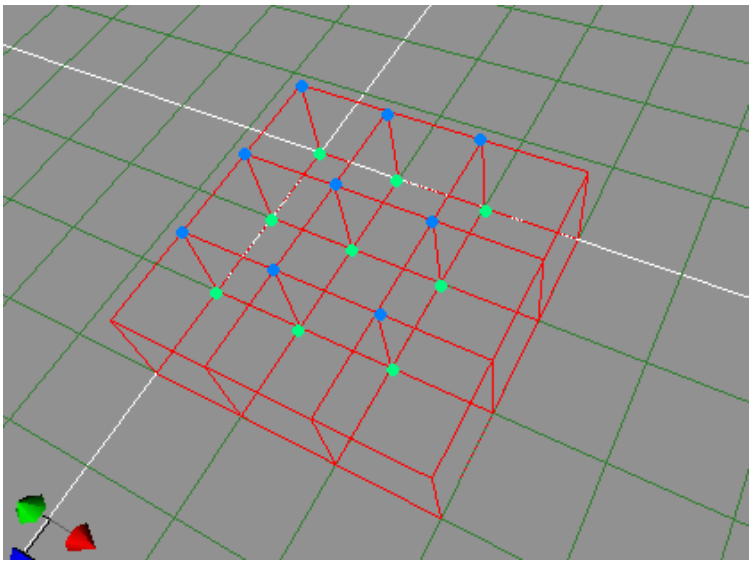
Με τις αντίστοιχες διαδοχές createNode, getField, setValue η addFace ολοκληρώνει το 'χτίσιμο' της ιεραρχίας. Όπως φαίνεται και στο σχήμα τα τελικά πεδία, αυτά δηλαδή που δεν αποτελούνται από άλλα nodes αλλά από καθαρά δεδομένα, είναι έξι. Τα δύο από αυτα, το coordIndex και το texCoordIndex είναι σταθεροί πίνακες οι οποίοι ουσιαστικά δεν χρησιμοποιούνται. Τα coordIndex και texCoordIndex ορίζουν τη σειρά με την οποία θα διαβαστούν τα σημεία των πινάκων coord και texCoord αντίστοιχα. Εφ' όσον εμείς ορίσαμε τους πίνακες αυτούς με τη σειρά που οι ίδιοι θέλαμε, δεν υπάρχει κανένας λόγος να τους διαβάσουμε με διαφορετική σειρά, οπότε και οι Index πίνακες είναι σταθερά της μορφής {0,1,2,3}. Το τρίτο πεδίο δεδομένων είναι το Points του node Coordinate και ταυτίζεται με το πρώτο όρισμα της μεθόδου που περιέχει τα σημεία του σχήματος. Το τέταρτο πεδίο δεδομένων,το Points του node texCoordinate, ταυτίζεται με το δεύτερο όρισμα texturepoints και περιέχει όπως αναφέρθηκε τα σημεία του texture που αντιστοιχούν στο σχήμα. Το πέμπτο πεδίο δεδομένων είναι η διαδρομή του texture όπως αυτή λαμβάνεται από το τρίτο όρισμα της μεθόδου. Τέλος το έκτο πεδίο δεδομένων αφορά το χρώμα πίσω από το texture του σχήματος και είναι σταθερά στην τιμή {1,0,0} το οποίο στο χρωματικό μοντέλο RGB ισοδυναμεί με το καθαρό κόκκινο χρώμα.

2.5 Καθορισμός Πινάκων Σημείων

Αναφέραμε παραπάνω πως η μέθοδος addFace για να υλοποιήσει το σχήμα δέχεται ως όρισμα δύο πίνακες σημείων, έναν πίνακα σημείων στο χώρο για τη γεωμετρία του σχήματος, κι έναν πίνακα σημείων στο επίπεδο για το texture. Οι πίνακες αυτοί δημιουργούνται μετά την επιλογή σχήματος και αριθμού κατατμήσεων βάσει ενός αρχικού σημείου και του αριθμού κατατμήσεων μέσα στις κλάσεις CubeFacesPoints για τον κύβο, PyramidFacesPoints και TrapezeFacesPoints για την πυραμίδα και CylinderFacesPoints για τον κύλινδρο. Ο κλάσεις αυτές δημιουργούν πίνακες για ένα μόνο κομμάτι του puzzle. Επομένως για κάθε κομμάτι του puzzle δημιουργείται και διαφορετικό στιγμιότυπο της αντίστοιχης κλάσης. Για να βρούμε τα αρχικά σημεία που θα δώσουμε ως ορίσματα αρκούν τρεις εμφωλευμένοι for βρόγχοι στο κυρίως σώμα του προγράμματος. Προτού αναφερθούν οι αλγόριθμοι λεπτομερώς πρέπει να σημειωθεί πως το σύστημα μέτρησης των σημείων texture δεν είναι απόλυτο αλλά αναλογικό. Αυτό σημαίνει πως το μήκος και το πλάτος μιας εικόνας δεν εξαρτάται από την ανάλυσή της, αλλά είναι σταθερά μοναδιαία. Πρέπει να σημειωθεί πως οι κλάσεις δημιουργίας πινάκων σημείων έχουν γραφτεί έτσι ώστε να αγνοούν τα εντελώς εσωτερικά κομμάτια του puzzle, αυτά δηλαδή που δεν έχουν κανένα texture. Το puzzle που προκύπτει μετά την κατάτμηση είναι «κούφιο», έτσι ώστε να αποφεύγουμε το πλημμύρισμα της σκηνής με μη-χαρακτηριστικά κομμάτια.

2.5.1 Ο ΚΥΒΟΣ:

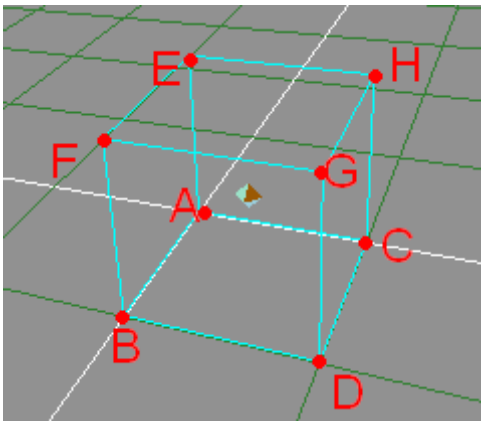
CubeFacesPoints: Ο constructor της κλάσης CubeFacesPoints δέχεται δύο ορίσματα. Το πρώτο όρισμα αφορά το αρχικό σημείο, και το δεύτερο όρισμα τον αριθμό κατατμήσεων. Η CubeFacesPoints αρχικά υπολογίζει το μήκος πλευράς που θα έχουν τα κομμάτια του puzzle. Το συνολικό puzzle, για λόγους πρακτικότητας όσον αφορά την ταύτιση των σημείων texture, έχει πάντα μήκος 1. Η πλευρά των κομματιών του puzzle (step) συνεπώς έχει μήκος step=1/divs, όπου divs ο επιλεγμένος αριθμός κατατμήσεων. Η βασική λειτουργία της CubeFacesPoints είναι να δημιουργεί πλευρές μήκους step προς την θετική φορά των αξόνων με αφετηρία το αρχικό σημείο.



Λέγοντας πλευρές εννοούμε έξι διαφορετικά ζεύγη πινάκων σημείων και πινάκων σημείων texture. Στην παραπάνω εικόνα για το χαμηλότερο επίπεδο ενός κύβου σπασμένου σε τρείς κατατμήσεις ανά ακμή φαίνεται η δημιουργία των υποκύβων βάσει των ανοιχτόχρωμα χρωματισμένων αρχικών σημείων. Τα πιο σκούρα χρωματισμένα σημεία θα αποτελέσουν αρχικά σημεία για το δεύτερο επίπεδο. Για να βρεθούν τα αρχικά σημεία των κατατμημένων κύβων χρησιμοποιούμε τον απλούστερο βρόγχο

```
for(int x=0; x<divsPerSide; x++)
        for(int y=0; y<divsPerSide; y++)
                for(int z=0; z<divsPerSide; z++)
```

Θεωρώντας ως ορίσματα το αρχικό σημείο A(0,0,0) του επόμενου σχήματος, και αριθμό κατατμήσεων 3 (άρα step=0.33) η CubeFacesPoints θα δημιουργούσε τους παρακάτω πίνακες:



Για παράδειγμα η μπροστινή πλευρά BDGF. Στα σημεία της γεωμετρίας χρειάζεται προσοχή στη δεξιόστροφη αναφορά των σημείων, έτσι ώστε η ορατή όψη να είναι από μπροστά:

{ {0, 0, 0.33}, {0.33, 0, 0.33}, {0.33, 0.33, 0.33}, {0, 0.33, 0.33} }

Στα σημεία του texture μας ενδιαφέρει μόνο το επίπεδο της συγκεκριμένης πλευράς. Η διάσταση που μας είναι αδιάφορη στη συγκεκριμένη πλευρά είναι η z οπότε και απλώς αντιστοιχούμε τις x και τις y συντεταγμένες της γεωμετρίας με αυτές του texture:

{ {0, 0}, {0.33, 0}, {0.33, 0.33}, {0, 0.33} }

Ως δεύτερο παράδειγμα για να φανούν ξεκάθαρα οι αντιστοιχίες των σημείων θα πάρουμε την πιο ιδιαίτερη πλευρά ACDB. Προσοχή στην αναφορά των σημείων, η οποία έχει γίνει με σειρά τέτοια ώστε η πλευρά να είναι ορατή από κάτω:

{ {0, 0, 0}, {0.33, 0, 0}, {0.33, 0, 0.33}, {0, 0, 0.33} }

Αυτή τη φορά στα σημεία του texture η αδιάφορη διάσταση είναι η y οπότε και τα ο άξονας x της γεωμετρίας θα αντιστοιχιθεί με τον άξονα x του texture, ενώ ο άξονας z της γεωμετρίας θα αντιστοιχιθεί με τον άξονα y του texture:

{ {0, 0}, {0.33, 0}, {0.33, 0.33}, {0, 0.33} }

Ένα τελευταίο παράδειγμα είναι απαραίτητο για να εξηγηθεί και η τελευταία ιδιαιτερότητα στην αντιστοιχία σημείων γεωμετρίας και σημείων texture. Η πλευρά που μας βολεύει για το παράδειγμα αυτό είναι η πάνω πλευρά EFGH. Τα σημεία της γεωμετρίας της με δεξιόστροφη φορά ορατή από πάνω είναι τα εξής:

{ {0, 0.33, 0}, {0, 0.33, 0.33}, {0.33, 0.33, 0.33}, {0.33, 0.33, 0} }

Η αδιάφορη διάσταση και πάλι είναι η y όμως σε αυτή την περίπτωση υπάρχει κάτι περισσότερο που πρέπει να προσέξουμε. Όσο μεγαλύτερη είναι η τιμή της z συνιστώσας της γεωμετρίας τόσο χαμηλότερο κομμάτι της εικόνας του texture που αντιστοιχεί στο κομμάτι. Με πιο απλά λόγια το κομμάτι του συνολικού puzzle που βρίσκεται πιο «κοντά μας» έχει στην πάνω πλευρά του το χαμηλότερο μέρος της εικόνας. Για να λειτουργήσει αυτή η αντίστροφη αναλογία ο προβληματικός άξονας y του texture αντιστοιχείται με την συμπληρωματική ως προς τη μονάδα τιμή του άξονα z της γεωμετρίας:

{ {0, (1-0)}, {0, (1-0.33)}, {0.33, (1-0.33)}, {0.33, (1-0)} }

Οι πίνακες σημείων που δημιουργούνται είναι public χαρακτηριστικά του στιγμιότυπου της κλάσης έτσι ώστε να μπορούν να διαβαστούν από το κυρίως σώμα του προγράμματος και να εισαχθούν στις addFace ως ορίσματα. Στην πραγματικότητα οι πίνακες σημείων γεωμετρίας δεν ορίζουν

τον κύβο στη θέση της προέκτασης του αρχικού σημείου. Σε προχωρημένη φάση ο κώδικας παραλλάχθηκε ώστε να ορίζει όλους τους υποκύβους στο κέντρο των αξόνων. Αυτό εξυπηρετεί αφ'ενός στην απλότητα των πινάκων (μιας κι η γεωμετρία όλων των υποκύβων είναι η ίδια, και η θέση τους θα πάρει τυχαία τιμή ούτως ή άλλως) και αφ' ετέρου απαιτείται από μια ιδιότητα του σένσορα περιστροφής ώστε τα σχήματα να περιστρέφονται ομοιόμορφα γύρω από το κέντρο τους.

Επιπρόσθετα η κλάση CubeFacesPoints περιέχει τους ορισμούς κάποιων απλών μεθόδων που χρησιμοποιούνται γενικότερα στην εφαρμογή. Οι μέθοδοι αυτοί αναφορικά είναι οι παρακάτω:

i) public static float fiveSignif(float number){

   Η μέθοδος fiveSignif αποτελεί μια απλή μέθοδο στρογγυλοποίησης στο πέμπτο δεκαδικό ψηφίο. Δημιουργήθηκε γιατί κατα την καταμέτρηση των σημείων στη δημιουργία των σχημάτων ορισμένες συντεταγμένες δεν επαλήθευαν τις αναμενόμενες συνθήκες λόγω απόκλισης στο έβδομο ή όγδοο δεκαδικό ψηφίο.

ii) public static float randomFloat(){

   Η μέθοδος randomFloat δημιουργεί έναν τυχαίο float αριθμό ανάμεσα σε 1 και -1. Δημιουργήθηκε για να εξυπηρετήσει τις ανάγκες για τυχαία θέση και περιστροφή των σπασμένων κομματιών.

iii) public static float[] randomPosition(){

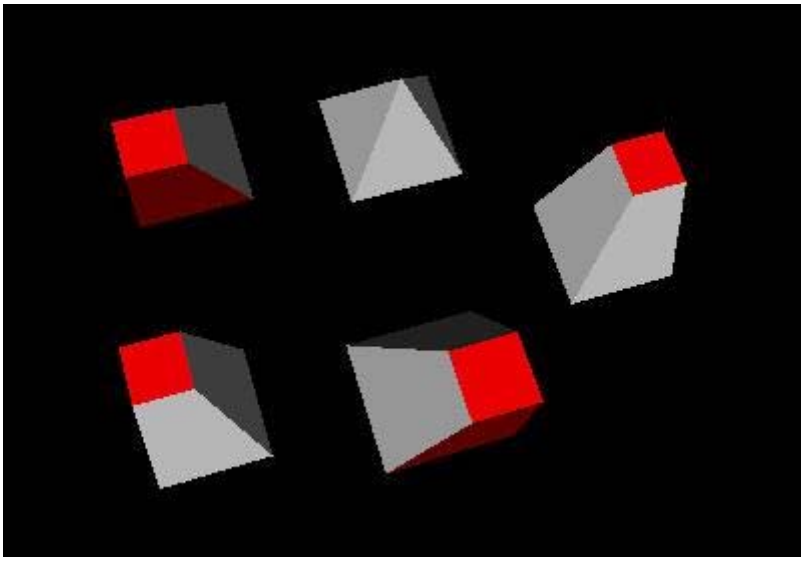
   Η μέθοδος randomPosition δημιουργεί και επιστρέφει έναν πίνακα τριών τυχαίων float αριθμών με τη βοήθεια της μεθόδου randomFloat. Ο πίνακας αυτός μπορεί να χρησιμοποιηθεί ως μία τυχαία θέση στο χώρο ως δεδομένο εισαγωγής στο πεδίο translation των groups.

iv) public static float[] randomRotation(){

Η μέθοδος randomRotation χρησιμοποιεί την randomFloat για να ορίσει έναν πίνακα τεσσάρων τυχαίων float που αντιστοιχούν σε ένα δεδομένο τύπου SFRotation. Τα πρώτα τρία float στοιχεία του πίνακα ορίζουν διανυσματικά τον άξονα περιστροφής. Το τέταρτο float στοιχείο, το οποίο δημιουργείται ως PI*randomFloat, έχει εύρος τιμών 2π (από –π έως +π) και ορίζει τη γωνία περιστροφής πάνω στον προηγουμένως ορισμένο τυχαίο άξονα.

2.5.2 Η ΠΥΡΑΜΙΔΑ:

Υπάρχουν δύο κλάσεις οι οποίες σχετίζονται με τον καθορισμό των σημείων των κομματιών της πυραμίδας. Αυτό συμβαίνει γιατί «σπάζοντας» μια πυραμίδα σε κομμάτια δημιουργούνται τρεις διαφορετικές γεωμετρίες: Μία πυραμίδα, κύβοι και τραπεζοειδή εξάεδρα.



Δυο κλάσεις χειρίζονται αυτά τα σχήματα βάσει του αριθμού των εδρών τους.Η PyramidFacesPoints δημιουργεί τους πίνακες σημείων για την πυραμίδα αποκλειστικά, ενώ η TrapezeFacesPoints αναλαμβάνει τη δημιουργία των πινάκων σημείων για τα εξάεδρα, είτε πρόκειται για κύβο είτε για τραπεζοειδές. Οι κλάσεις αυτές ακολουθούν τη νοοτροπία της CubeFacesPoints. Δέχονται ορίσματα για το αρχικό σημείο και τον αριθμό κατατμήσεων, και δημιουργούν πίνακες σημείων για κάθε έδρα της πυραμίδας ή των εξάεδρων. Αυτό, όπως και στην περίπτωση του κύβου προυποθέτει ότι πριν την κλήση της έχουμε βρει τα αρχικά σημεία των κομματιών. Και πάλι αυτό επιτυγχάνεται με τη χρήση τριών εμφωλευμένων for βρόγχων, λίγο πιο σύνθετων στη μορφή ώστε να περιγράψουν τη γεωμετρία μιας πυραμίδας:

```
for(int y=0; y<(divsPerSide-1); y++)
        for(int x=0; x<(divsPerSide-y); x++)
                        for(int z=0; z<(divsPerSide-y); z++)
```

Παρατηρούμε ότι όσο ανεβαίνουμε στο επίπεδο y της πυραμίδας, τόσο μειώνεται ο αριθμός κατατμήσεων ανα πλευρά του επιπέδου αυτού. Το μισό του μήκους των «χαμένων» κατατμήσεων προστίθεται ως offset στις διαστάσεις x και z για να δημιουργήσει την κλίση της πυραμίδας.

PyramidFacesPoints: Η PyramidFacesPoints θα δημιουργήσει το κορυφαίο κομμάτι του puzzle-πυραμίδα, γι' αυτό και δεν χρειάζεται όρισμα αρχικού σημείου. Διατηρώντας την αποδοχή ότι όλες οι συνολικές διαστάσεις του σχήματος είναι 1 (μήκος, ύψος και βάθος), ξέρουμε αυτομάτως τη σταθερή τιμή του σημείου της κορυφής K(0.5, 1, 0.5). Η PyramidFacesPoints για να βρει τα δύο κατώτερα σημεία των εδρών αρκεί να μετατοπιστεί σε σχέση με το σημείο της κορυφής κατα step στον άξονα των y, και κατα +-step/2 στους άξονες x και z. Για παράδειγμα θεωρόντας ένα puzzle-πυραμίδα με 2 κατατμήσεις, δηλαδή step=0.5 ο πινακας σημείων γεωμετρίας της μπροστινής έδρας για το κορυφαίο κομμάτι θα ήταν ως εξής:

{ {0.5, 1, 0.5}, {0.25, 0.5, 0.75}, {0.75, 0.5, 0.75} }

Για τους πίνακες σημείων των textures η PyramidFacesPoints λειτουργεί πανομοιότυπα με την CubeFacesPoints. Για τη συγκεκριμένη έδρα αδιάφορη διάσταση είναι η z. Επίσης, για τη συγκεκριμένη έδρα, το μήκος του texture αυξάνεται ανάλογα με το x της γεωμετρίας, οπότε και δεν προκύπτει ζήτημα αντιστροφής συντεγαμένων:

{ {0.5, 1}, {0.25, 0.5}, {0.75, 0.5} }

Στιγμιότυπο της κλάσης PyramidFacesPoints χρησιμοποιείται, εκτός για την υλοποίηση του κορυφαίου κομματιού του σπασμένου puzzle, και για την εμφάνιση του ακαίρεου puzzle-πυραμίδα χρησιμοποιόντας ως επιλεγμένο αριθμό κατατμήσεων το 1.

TrapezeFacesPoints: Η κλάση TrapezeFacesPoints είναι η πιο σύνθετη κλάση δημιουργίας πινάκων σημείων της εφαρμογής. Όχι μόνο πρέπει να διαχωρήσει την ανάγκη δημιουργίας κύβου ή τραπεζοειδούς, αλλά πρέπει να επιλέξει τη δημιουργία της κατάλληλης από τα εννέα διαφορετικά είδη μορφής του τραπεζοειδούς. Το ποιά ή ποιές έδρες του τραπεζοειδούς θα είναι κεκκλιμένες εξαρτάται από την τιμή του αρχικού σημείου, κι αυτός είναι ένας έλεγχος που γίνεται μέσα στην TrapezeFacesPoints. Δύο είναι οι τιμές που χρησιμοποιούνται για τον έλεγχο αυτό. Η μία είναι η offset, η οποία δηλώνει το χαμένο μήκος σε σχέση με την αρχή των αξόνων που προκαλεί η κλίση, και αυξάνεται κατα step/2 για κάθε επίπεδο. Η δεύτερη τιμή είναι η endCoord. Η endCoord ισούται με 1-(step+offset). Η offset και η endCoord ουσιαστικά αποτελούν τον υπολογισμό του x ή z για ένα κομμάτι της αριστερής ή δεξιάς πλευράς της πυραμίδας αντίστοιχα, για οποιοδήποτε επίπεδο y. Για να καθοριστεί η κατάλληλη γεωμετρία που πρέπει να υλοποιηθεί για κάθε αρχικό σημείο, γίνεται η παρακάτω σειρά ελέγχων οι οποίοι συνδέονται μεταξύ τους με else για να διατηρούν τη συνοχή των στοιχείων:

i) (Z == endCoord) && (X == offset) : Εαν το αρχικό σημείο ικανοποιεί αύτη τη συνθήκη σημαίνει πως πρόκειτα για σημείο που θα δημιουργήσει τραπεζοειδές εξάεδρο που βρίσκεται στο μπροστινό (z==endCoord) και αριστερό (x==offset) μέρος της πυραμίδας. Αυτό σημαίνει πως οι εξωτερικές πλευρές του, η μπροστινή και η αριστερή, θα πρέπει να δημιουργηθούν με κλίση. Η σταθερή κλίση που χρησιμοποιείται στην εφαρμογή είναι αυτή που προκύπτει με τη μετατόπιση των ανώτερων σημείων κατα step/2. Η TrapezeFacesPoints μετά την εύρεση του τύπου εξαέδρου λειτουργεί ακριβώς όπως και η CubeFacesPoints όσον αφορά τη δημιουργία των πινάκων.

ii) (Z == endCoord) && (X == endCoord) : Το αρχικό σημείο που ικανοποιεί αυτή τη συνθήκη θα δημιουργήσει τραπεζοειδές εξάεδρο που βρίσκεται στο μπροστινό (z==endCoord) και αριστερό (x==endCoord) μέρος της πυραμίδας. Η μπροστινή και η δεξιά πλευρά του δηλαδή θα δημιουργηθεί με κλίση.

iii) (Z == endCoord) : Αφού το υπο εξέταση αρχικό σημείο προσπέρασε τις δύο παραπάνω συνθήκες, αλλά ικανοποιεί την τρέχουσα, σημαίνει πως πρόκειται για τραπεζοειδές εξάεδρο του μπροστινού μέρους της πυραμίδας, αλλά όχι ακραίου όσον αφορά τον άξονα x. Αυτής της κατηγορίας τα εξάεδρα έχουν κεκλιμένη μόνο τη μπροστινή τους έδρα.

iv) (Z == offset) && (X == offset) : Ένα σημείο που ικανοποιεί τη συνθήκη αυτή δημιουργεί τραπεζοειδές εξάεδρο του πίσω (z==offset) και αριστερού (x==offset) μέρους της πυραμίδας. Το εξάεδρο αυτό έχει την πίσω και την αριστερή έδρα του κεκλιμένη

v) (Z == offset) && (X == endCoord) : Η συνθήκη αυτή περιγράφει ένα σημείο που θα δημιουργήσει τραπεζοειδές εξάεδρο του πίσω (z==offset) και δεξιού (x==endCoord) μέρους της πυραμίδας. Το εξάεδρο αυτό θα έχει την πίσω και δεξιά πλευρά του κεκλιμένη.

vi) (Z == offset) : Το υπό εξέταση αρχικό σημείο εφ' όσον προσπέρασε τις δύο προηγούμενες συνθήκες αντιστοιχεί σε τραπεζοειδές εξάδρο του πίσω μέρους της πυραμίδας, αλλά όχι ακραίου όσον αφορά τον άξονα x. Κεκλιμένη θα είναι μόνο η πίσω πλευρά του.

vii) (X == offset) : Δεδομένου ότι το υπό εξέταση αρχικό σημείο προσπέρασε τις προηγούμενες συνθήκες αλλά επαλήθευσε τη συγκεκριμένη, προκύπτει πως αντιστοιχεί σε τραπεζοειδές εξάεδρο που βρίσκεται στην αριστερή πλευρά της πυραμίδας, αλλά όχι ακραίο όσον αφορά τον άξονα z. Τέτοιου τύπου εξάεδρα έχουν κεκλιμένη μόνο την αριστερή πλευρά τους.

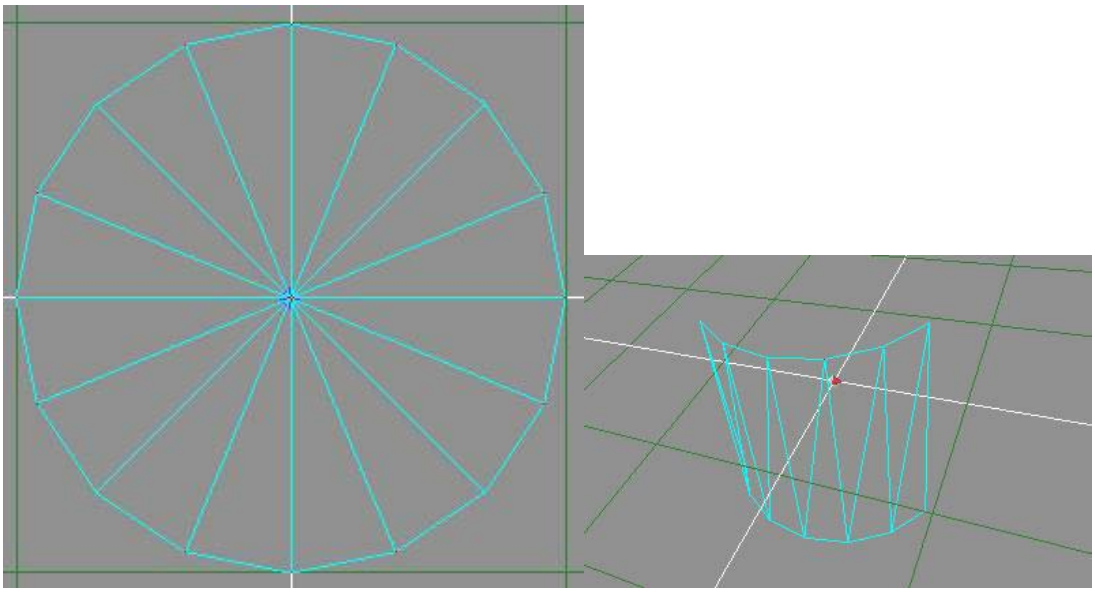
viii) (X == endCoord) : Όπως και στη συνθήκη vii το γεγονός ότι το υπό εξέταση αρχικό σημείο προσπέρασε τις παραπάνω συνθήκες αλλα ικανοποιήσε τη συγκεκριμένη, μας δηλώνει ότι πρόκειται για αρχικό σημείο που δημιουργεί τραπεζοειδές εξάεδρο της δεξιάς πλευράς της πυραμίδας. Συνεπώς το εξάεδρο θα έχει κεκλιμένη μόνο τη δεξιά έδρα του.

ix) (Y == 0) : Προσπερνώντας όλες τις παραπάνω συνθήκες έχουμε εξασφαλίσει πως το αρχικό σημείο που εξετάζουμε δεν δημιουργεί εξωτερικό εξάεδρο όσον αφορά τους άξονες x και z. Η τελευταία αυτή συνθήκη όταν ικανοποιείται, δηλώνει πως το αρχικό σημείο δημιουργεί κύβο της κάτω πλευράς της πυραμίδας. Καμία έδρα του σχήματος αυτού δεν θα είναι κεκλιμένη καθώς πρόκειται για καθαρά εσωτερικό σημείο όσον αφορά τις κεκλιμένες διαστάσεις x και z.

Διαχωρίζοντας τις γεωμετρίες στις παραπάνω κατηγορίες η κλάση TrapezeFacesPoints ορίζει τους κατάλληλους πίνακες σημείων γεωμετρίας λαμβάνοντας υπ' όψιν τις κλίσεις που ταιριάζουν στην περίσταση. Οι πίνακες σημείων texture δεν επιρρεάζονται καθόλου από την κλίση, καθώς αυτή υπολογίζεται πάντοτε στην αδιάφορη διάσταση της γεωμετρίας.

2.5.3 Ο ΚΥΛΙΝΔΡΟΣ:

Ο τρόπος με τον οποίο ένα puzzle-κύλινδρος σπάει στα επιμέρους κομμάτια διαφέρει σε σχέση με τις άλλες δύο γεωμετρίες. Ο κύλινδρος δεν έχει καθόλου οριζόντιες τομές, παρά τεμαχίζεται κάθετα σε – κατα κανόνα – ίσα κομμάτια τύπου πίτας. Παρ' όλο που εκ' πρώτης όψεως η δημιουργία των κομματιών του κυλίνδρου δείχνει υπερβολικά απλή, δεν είναι. Η καμπυλότητα της γεωμετρίας του κυλίνδρου είναι και η πολυπλοκότητα του. Στην X3D μια έδρα που είναι κυρτή ή έχει έστω μία καμπύλη πλευρά, υλοποιείται αναγκαστικά μέσω πολλών στοιχειωδών επίπεδων εδρών ή πλευρών.



Ένας κυκλικός δίσκος αποτελείται από στοιχειώδη τρίγωνα. Το ίδιο ισχύει και για μία κυρτή επιφάνεια.

Ως στοιχειώδες επίπεδο κομμάτι έχει επιλεχθεί αυτό που σχηματίζουν τα ακραία σημεία τόξων μίας μοίρας στην περιφέρεια των κυκλικών δίσκων του κυλίνδρου. Ο περιορισμός που προκύπτει από αυτή την παραδοχή είναι ότι δεν μπορούμε να δημιουργήσουμε κομμάτια μή ακέραιου αριθμού μοιρών, παρά με τη μορφή εξαίρεσης. Παρακάτω στην ανάλυση της λειτουργίας της CylinderFacesPoints θα αναφερθεί και η λύση αυτού του προβλήματος. Για να βρεθούν τα αρχικά σημεία, ή πιο σωστά οι

αρχικές μοίρες από τις οποίες θα δημιουργηθούν τα κομμάτια, χρησιμοποιείται ο παρακάτω απλούστατος βρόγχος

```
for(int j=0;j<divsPerSide;j++)
    float currentStartDeg = j*step;
```

όπου step=360/divs δηλαδή ο αριθμός των μοιρών που αντιστοιχεί σε κάθε κομμάτι. Η σημαντική διαφορά της κλάσης αυτής σε σχέση με τις προηγούμενες είναι πως για την κυρτή πλευρά του κομματιού του κυλίνδρου, καθώς και για τις καμπύλες πάνω και κάτω πλευρές, η CylinderFacesPoints δεν δημιουργεί απλώς από δύο πίνακες σημείων (γεωμετρίας και texture), αλλά από δύο Vector καμπύλων που περιέχουν πίνακες σημείων ισάριθμους με το ακέραιο μέρος του step, όσα δηλαδή είναι και τα στοιχειώδει μέρη.

CylinderFacesPoints: Ο constructor της CylinderFacesPoints δέχεται δύο ορίσματα. Το πρώτο, startDeg, είναι οι μοίρες από τις οποίες θα ξεκινήσει το κομμάτι. Το δεύτερο όρισμα, partsPerSide, είναι ως συνήθως ο επιλεγμένος αριθμός κατατμήσεων. Για να επιλύσει το πρόβλημα που αναφέρθηκε παραπάνω, η πρώτη ενέργεια που κάνει η CylinderFacesPoints είναι να διαχωρίσει το ακέραιο μέρος του step και το δεκαδικό σε διαφορετικές μεταβλητές, intStep και remainingStep. Η CylinderFacesPoints τώρα καλείται να δημιουργήσει πίνακες σημείων για μία κυρτή επιφάνεια, μία επίπεδη αλλά με καμπύλη πλευρά, και δύο κανονικές παραλληλόγραμμες επιφάνειες. Οι δύο πρώτες επιφάνειες θα δημιουργηθούν μέσω ενός βρόγχου for, ο οποίος θα επαναληφθεί για το ακέραιο μέρος του βήματος, όπου και θα δημιουργηθούν τα τόξα μίας μοίρας. Οι συντεταγμένες των σημείων των περιφερειών των κύκλων (οι πάνω και κάτω έδρες του κυλίνδρου δηλαδή) ορίζονται πολύ εύκολα παίροντας ως x το συνημίτονο της τιμής του στοιχειώδους βήματος, και ως z το ημίτονο.Η διάσταση y των κομματιών είναι σταθερά -1 ή 1, καθώς ο κύλινδρος δεν έχει καμία οριζόντια τομή. Μέσα στο βρόγχο αυτό ορίζονται εξ' αρχής τα σημεία που θα αποτελούν τις στοιχειώδεις έδρες και πλευρές:

```
for(int i=0;i<intStep;i++){

    currentX = cos( startDeg+i );
    currentZ = sin( startDeg+i );
    nextX = cos( startDeg+i+1 );
    nextZ = sin( startDeg+i+1 );
```

Έχοντας διασαφηνίσει τις συντεταγμένες των σημείων μπορούμε εύκολα, πάντα εντός του βρόγχου for, να προσθέσουμε στα προαναφερθέντα Vectors τις τιμές όλων των στοιχειωδών εδρών ή πλευρών, για τη γεωμετρία και τα textures τους. Πχ :

```
Float[][] roundFacePart =
{{currentX, -1, currentZ}, {currentX, +1, currentZ},    {nextX, +1, nextZ},
{nextX, -1, nextZ}};
Vector roundFacesPoints.addElement(roundFacePart);

Float[][] roundFaceTexturePart =
{{1-((startDeg+i)/360),0},{1-((startDeg+i)/360),1},{1-((startDeg+i+1)/360),
1}, {1-((startDeg+i+1)/360), 0}};
Vector roundFacesTexturePoints.addElement(roundFaceTexturePart);
```

Ομοίως λειτουργούμε για τα σημεία γεωμετρίας και texture της πάνω και της κάτω πλευράς. Η μόνη εκκρεμότητα που έχει μείνει όσον άφορά τις καμπύλες πλευρές του κομματιού είναι το δεκάδικό μέρος του βήματος, remainingStep. Μαζί με τις μεταβλητές currentX, currentZ, nextX, nextZ ορίζονται στην αρχή του βρόγχου δύο ακόμα μεταβλητές συντεταγμένων:

```
extraX = cos( startDeg+i+1+remainStep ) ;
extraZ = sin( startDeg+i+1+remainStep );
```

Στην τελευταία επανάληψη του βρόγχου for – με τον έλεγχο for(int i=0;i<intStep;i++) – προστίθεται ένας ακόμα πίνακας σημείων γεωμετρίας και texture σε κάθε Vector καμπύλων. Αυτό αντιστοιεί στο, μικρότερο της μίας μοίρας τόξο, που ακολουθεί το τελευταίο ακέραιο τόξο του Vector. Τα σημεία του τόξου αυτού είναι από next έως extra με την ίδια αναλογία που λειτουργούσαν τα current-next στα τόξα μίας μοίρας.

Οι πίνακες γεωμετρίας των δύο επίπεδων πλευρών των σχημάτων δημιουργούνται με τον πλέον γνωστό τρόπο, με αρχικές μοίρες αυτές του ορίσματος startDeg και τελικές μοίρες startDeg+step. Οι συντεταγμένες των

σημείων προκύπτουν έμμεσα από τα ημίτονα και τα συνημίτονα των μοιρών αυτών, ενώ όπως αναφέραμε και πιο πάνω οι τιμές της διάστασης y είναι σταθερά -1 ή 1. Χωρίς να υπάρχει καν ανάγκη δημιουργίας πίνακα σημείων texture, αφού οι έδρες αυτές είναι εξ'ορισμού εσωτερικές, η δημιουργία των πινάκων των πλευρών αυτών είναι αρκετά όμοια με αυτή του κύβου ώστε να μην αξίζει να παρατεθεί παράδειγμα.

Η υλοποίηση των αποτελεσμάτων της CylinderFacesPoints στο κυρίως σώμα της εφαρμογής πρέπει να γίνει με επανειλημένες addFace λόγω των πολλών στοιχειωδών πλευρών που αποτελούν την κυρτή πλευρά και των στοιχειωδών τριγώνων που αποτελούν τον κυκλικό δίσκο. Το παράδειγμα που ακολουθεί είναι υπεραπλουστευμένο όσον αφορά τη σύνταξη και τα ορίσματα των μεθόδων:

```
for (int i=0; i<=intStep; i++){
        addFace(RoundFacePoints.elementAt(i),
                RoundFaceTexturePoints.elementAt(i),
                ……);
        addFace(TopFacePoints.elementAt(i) ,
                TopFaceTexturePoints.elementAt(i),
                ……);
        addFace(BottomFacePoints.elementAt(i)      ,
                BottomFaceTexturePoints.elementAt(i),
                ……);    }
```

Οι δύο επίπεδες πλευρές υλοποιούνται εκτός του βρόγχου αφού είναι μοναδικές για κάθε κομμάτι της πυραμίδας.

Ολοκληρώνοντας και την τεχνική δημιουργίας των πινάκων του κυλίνδρου είμαστε σε θέση να δημιουργήσουμε τα σπασμένα κομμάτια (ή ολόκληρα για τιμή κατατμήσεων 1). Για να συμπληρωθεί η αρχική όψη του puzzle το μόνο που μένει, εφ'όσον τα κομμάτια έχουν δημιουργηθεί, είναι το ανακάτεμά τους. Όπως εξηγήθηκε παραπάνω σε κάθε βρόγχο for στο κυρίως σώμα της εφαρμογής στην κλάση CreateScene, η κάθε έδρα του σχήματος προστίθεται σε ένα parent group το οποίο προσδίδει τη συνοχή στο κομμάτι αυτό. Όλοι οι απόγονοι του node X3DGroupNode διαθέτουν τα πεδία translation και rotation για το χειρισμό της θέσης τους στη σκηνή. Στα

πεδία αυτά, αμέσως μετά την πρόσθεση και της τελευταίας πλευράς, εισάγωνται τυχαίες τιμές θέσης και προσανατολισμού με τη βοήθεια των μεθόδων randomPosition() και randomRotation() που περιέχονται στην κλάση CubeFacesPoints.

2.6 Χειρισμός και Κίνηση Στο Χώρο

Τώρα που στη σκηνή υπάρχουν υλοποιημένα και τεμαχισμένα, σύμφωνα με τις προτιμήσεις του χρήστη, όλα τα κομμάτια του puzzle πρέπει να καταστήσουμε εφικτή την κίνηση τους στο χώρο έτσι ώστε το puzzle να μπορεί να λυθεί. Αυτό το θέμα αποδείχτηκε και το πιο περίπλοκο αλλά και ενδιαφέρον, κυρίως λόγω της πρώιμης μορφής του πακέτου SAI. Η έκδοση του Xj3D με την οποία ολοκληρώθηκε το project, και στην οποία αναφέρεται αυτή η ανάλυση είναι η Xj3D-2-M1-DEV-20071011.

2.6.1 Sensor Nodes

Η αλληλεπίδραση του χρήστη με τη σκηνή γίνεται με τη βοήθεια των sensors. Γενικά ένας sensor που προστίθεται σε ένα group αντικειμένων τα «επικαλύπτει» με τη λειτουργία του. Ο ίδιος ο sensor παραμένει μη ορατός στη σκήνή μας, ο cursor όμως αλλάζει σχήμα όταν τον περνάμε πάνω από ορατά αντικείμενα του ίδιου group, μαρτυρώντας την ύπαρξη και τον τύπο του. Στην X3D ορίζονται αρκετά είδη sensor, αυτά που μας αφορούν όμως στην προκειμένη περίπτωση είναι η κατηγορία των drag sensors. Όπως φανερώνει και το όνομά τους οι sensors αυτοί είναι ικανοί να παρακολουθούν τις μεταβολές της κίνησης του ποντικιού και να τις αποστέλουν όπου εμείς διαλέξουμε. Το ποντίκι κινείται στις δύο διαστάσεις, αντίθετα με την τρισδιάστατη σκήνή μας, έτσι προκύπτει ένας αριθμός διαφορετικών τρόπων ερμηνείας της κίνησης του ποντικιού. Οι διάφοροι απόγονοι της κλάσης dragSensor αποτελούν και τις διαφορετικές αυτές ερμηνείες:

i) PlaneSensor: Ο PlaneSensor ερμηνεύει την κίνηση του ποντικιού ως κίνηση πάνω σε ένα επίπεδο. Ο αισθητήρας αυτός μπορεί να λειτουργήσει σε οποιοδήποτε επίπεδο εμείς διαλέξουμε, κάθετο, οριζόντιο ή κεκλιμένο. Ο PlaneSensor κατά κανόνα χρησιμοποιείται για την κίνηση στο χώρο, αφού τα δεδομένα που επιστρέφει είναι τα σημεία στο χώρο που διέγραψε η τροχιά του ποντικιού.

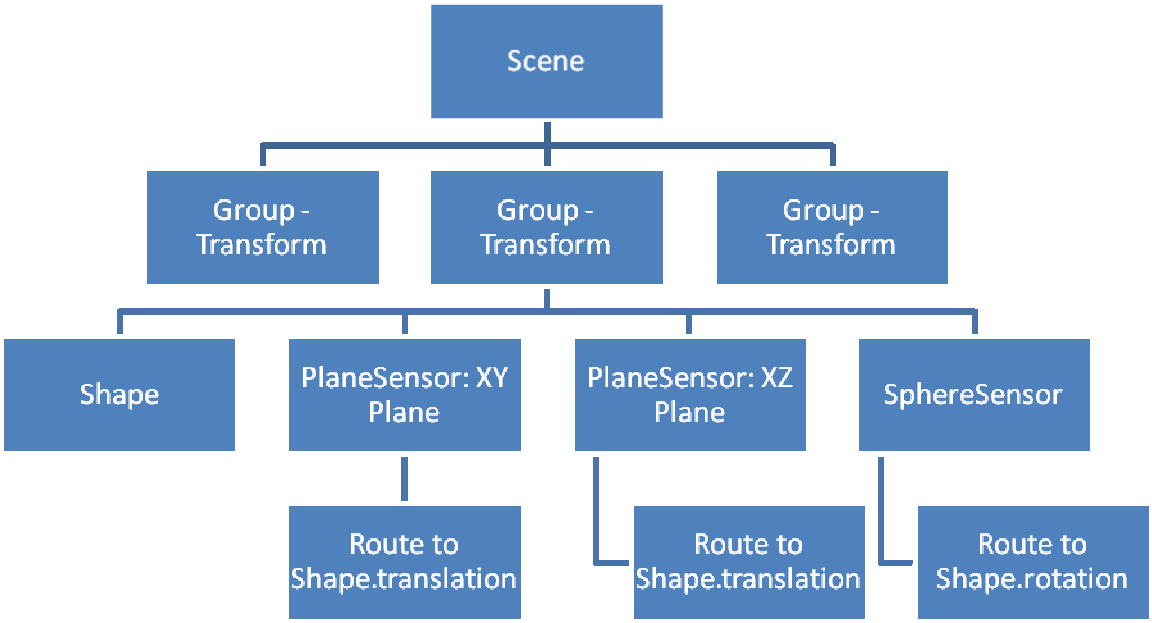
ii) CylinderSensor: Ο CylinderSensor αντίθετα με τον PlaneSensor, χρησιμοποιείται τυπικά για να ελέγξει περιστροφές. Ερμηνεύει την

κίνηση του ποντικιού ώς περιστροφή γύρω από συγκεκριμένο άξονα της επιλογής μας, σε αντιστοιχία με τον τρόπο που περιστρέφεται ένας κύλινδρος γύρω από τον άξονα του.

iii) SphereSensor: Ο SphereSensor είναι επίσης ένας περιστροφικός αισθητήρας, αλλά διαθέτει το μέγιστο βαθμό περιστροφικής ελευθερίας. Ερμηνεύει την κίνηση του ποντικιού ώς περιστροφή γύρω από το κέντρο μιάς νοητής σφαίρας. Το κέντρο αυτό βρίσκεται στη θέση της επιλογής μας, και ταυτίζεται με το κέντρο του group στο οποίο ανήκει ο αισθητήρας.

2.6.2 Η Αρχική Ιδέα

Η αρχική ιδέα για τον μηχανισμό κίνησης των κομματιών ήταν αρκετά απλή, όμως θεωρητικά σωστή και αποτελεσματική. Για να έχουμε πλήρως ελεύθερη κίνηση στον χώρο χρειαζόμαστε δύο PlaneSensors για την κίνηση στα xy και xz, κάθώς κι έναν SphereSensor για την περιστροφή των κομματιών. Ένα τέτοιο σετ αισθητήρων θα προστιθόταν σε κάθε group μαζί με τα κατάλληλα routes (αντιστοίχηση της εξόδου του αισθητήρα με τη θέση ή την περιστροφή του group) όπως φαίνεται στο παρακάτω σχήμα:



Ο χρήστης θα επέλεγε έναν αισθητήρα ως ενεργό κάθε φορά από το γραφικό περιβάλλον και η εφαρμογή θα λειτουργούσε, φαινομενικά, ακριβώς όπως και τώρα. Το πρόβλημα εμφανίστηκε στην υλοποίηση του μηχανισμού αυτού, όταν το SAI παρουσίασε πρόβλημα στη δυναμική δημιουργία οποιουδήποτε drag sensor. Η κονσόλα της java μας ενημέρωνε για το εξής:

“Warning: Unknown sensor type has been added to the sensor manager: SphereSensor”

Και αντίστοιχα:

“Warning: Unknown sensor type has been added to the sensor manager: PlaneSensor”

Όπως φαίνεται παραπάνω δεν εμφανιζόταν κανένα error αλλά ένα warning για κάθε sensor που προστίθονταν στη σκήνη. Η εφαρμογή έμοιαζε να τρέχει κανονικά, οι sensors όμως παρέμεναν ανενεργοί σαν να μην είχαν δημιουργηθεί ποτέ. Κατόπιν, έρευνας αρχικά, και στη συνέχεια συζήτησης με τους υπεύθυνους debuggers στο forum του Web3D Consortium καταλήξαμε πως το αδικαιολόγητο αυτό σφάλμα πρέπει να ελεγχθεί αναλυτικότερα. Κατατέθηκε αναφορά με παράδειγμα στο Xj3D Bugzilla και σύντομα αποδείχτηκε ότι η δυσλειτουργία προκαλούνταν από σφάλμα του εσωτερικού κώδικα των drag sensors. Το σφάλμα αυτό σύμφωνα με τον υπεύθυνο debugger είναι απλούστατο στη διόθωση του σε κάποια επόμενη έκδοση του Xj3D Toolkit, παρ’ όλα αυτά η δυναμική δημιουργία drag sensors παραμένει αδύνατη προς το παρόν. Βλέποντας τη συγκεκριμένη διάταξη με λίγη περισσότερη εμπειρία παρατηρούμε πως δεν θα λειτουργούσε σωστά λόγω περιστροφής των planeSensors μαζί με το group του σχήματος, όμως αυτό δεν ήταν κάτι που είχε ιδιαίτερη σημασία στην παρούσα φάση.

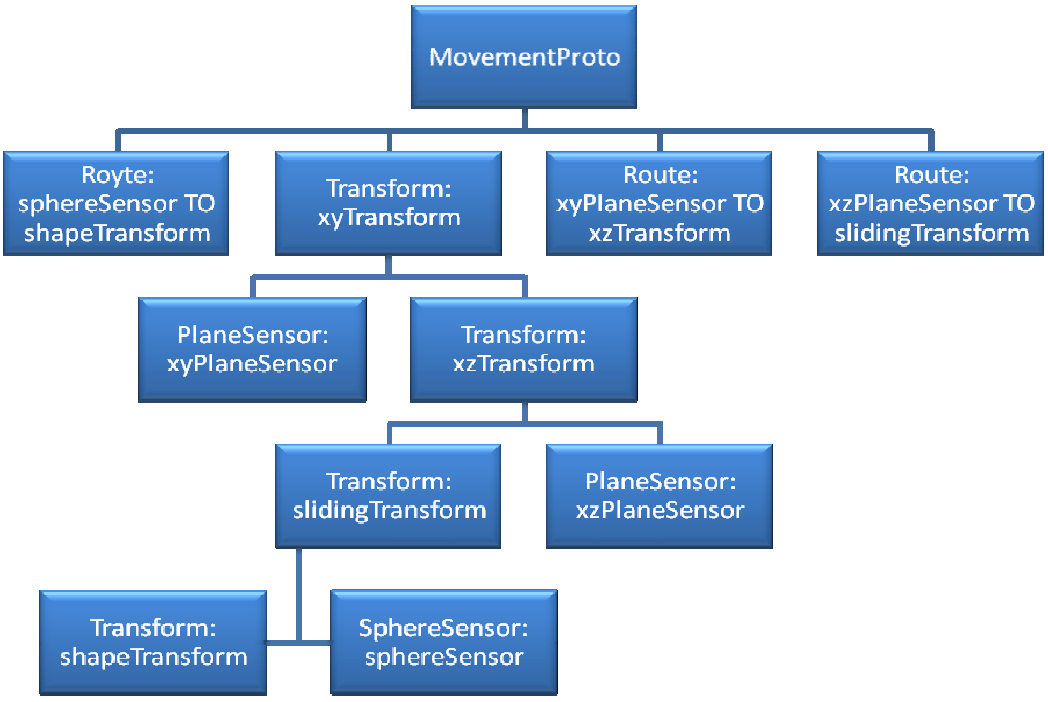
2.6.3 Το X3DPrototype

Αφού η δυναμική δημιουργία οποιουδήποτε απόγονου της κλάσης DragSensor ήταν κατηγορηματικά αδύνατη, έπρεπε να βρεθεί κάποια work-around λύση που να διατηρεί όμως τη δυναμική φύση της εφαρμογής. Μετά από προσεκτική μελέτη του Xj3D 2.0 Code Library, προέκυψε η κλάση X3DPrototype. Στην X3D γενικότερα μπορούμε να ορίσουμε μια ομάδα αντικειμένων ως πρωτότυπο, επιλέγοντας ως ελεύθερα για επεξεργασία οποιαδήποτε πεδία των αντικειμένων επιθυμούμε, και να ανακαλέσουμε αντίγραφα ή παραλλαγές του. Το X3DPrototype λοιπον είναι η αντίστοιχη κλάση που επιτρέπει στο Xj3D να χειρίζεται δυναμικά πρωτότυπα ορισμένα από το χρήστη. Αυτή

ακριβώς η δυνατότητα του Xj3D Toolkit και του SAI έδωσε τη λύση στο, διαφορετικά μη επιλύσιμο, πρόβλημα του bug των drag sensors.

2.6.4 Το MovementProto

Το πρωτότυπο που θα χρησιμοποιηθεί ονομάστηκε movementProto και ορίστηκε σε καθαρή X3D σε ένα .x3d αρχείο, το οποίο και εισάγεται στη σκηνή με το ξεκίνημα της εφαρμογής. Στη συνέχεια μπορούν να υλοποιηθούν οσαδήποτε στιγμιότυπα χρειάζονται για να εξυπηρετήσουν τις ανάγκες του 3D Δυναμικού Puzzle. Η δομή του πρωτοτύπου περιγράφεται στο σχεδιάγραμμα που ακολουθεί:



Τα groups έχουν ονομαστεί σύμφωνα με τον sensor που περιέχουν. Ένας εμπειρικός κανόνας που χρειάζεται προσοχή είναι πως ένας sensor δεν πρέπει να μεταβάλει τη θέση ή την περιστροφή του group που τον περιέχει. Παρ' όλο που μια τέτοια δομή ορισμένες φορές μπορεί να λειτουργήσει κανονικά, δεν συνιστάται γιατί σε πιο σύνθετες

περιπτώσεις, όπως αυτή του παραπάνω σχήματος, μπορεί να προκαλέσει απρόβλεπτη συμπεριφορά. Επίσης οι planeSensors, ή το group που τους περιέχει, δεν πρέπει να περιστραφούν, καθώς περιστρέφοντας τους περιστρέφουμε και το επίπεδο λειτουργίας τους. Έχοντας αυτούς τους κανόνες στο μυαλό μας παρατηρούμε πως κάθε sensor επηρεάζει το διπλανό του group κι όχι το πατρικό του. Ειδικότερα ο xyPlaneSensor μετακινεί τον xzTransform, ο xzPlaneSensor μετακινεί τον slidingTransform, κι ο sphereSensor μετακινεί τον shapeTransform. Ο shapeTransform είναι το group το οποίο περιέχει το σχήμα. Το σχήμα αυτό είναι το μόνο ορατό αντικείμενο όλης της ιεραρχίας, κι έτσι το μόνο που μπορεί να ενεργοποιήσει τον επιλεγμένο sensor. Η κίνηση που θα δει ο χρήστης στο σχήμα εξαρτάται από το ποιός sensor θα είναι enabled και δεν προδίδει την ύπαρξη διαφορετικών επιπέδων κίνησης και ομαδοποίησης.

Μια ίδεα για την απλούστευση του προτοτύπου ήταν να χρησιμοποιηθεί μόνο ένας planeSensor ο οποίος θα περιστρεφόταν κατα 90 μοίρες γύρω από τον άξονα x για να μετακινεί τα σχήματα στα δύο επίπεδα. Η ιδέα αυτή απορρίφθηκε όμως σύντομα γιατί ο κώδικας για την περιστροφή του sensor θα έκανε το προτότυπο ακόμα πιο σύνθετο και υπολογιστικά βαρύτερο από ότι η χρήση ενός δεύτερου planeSensor.

Στον ορισμό του προτοτύπου πρέπει να επιλέξουμε τα πεδία που θα ανήκουν στο prototype interface, τα πεδία δηλαδή που θα είναι επεξεργάσιμα όταν το αντικείμενο prototype υλοποιηθεί. Το interface του movementProto περιλαμβάνει τα παρακάτω πεδία:

i) <field name='movingChildren' type='MFNode'/>: Το πεδίο movingChildren αποτελεί τον τρόπο εισαγωγής των σχημάτων στο shapeTransform. Δέχεται ως όρισμα έναν πίνακα με shapes, τις πλευρές δηλαδή του κάθε κομματιού

ii) <field name='xyPlaneEnable' type='SFBool'/>: Το πεδίο αυτό δέχεται ένα boolean ορισμά που αφορά την ενεργοποίηση του xyPlaneSensor.

iii) <field name='xzPlaneEnable' type='SFBool'/>: Το πεδίο αυτό δέχεται ένα boolean ορισμά που αφορά την ενεργοποίηση του xzPlaneSensor.

iv) <field name='sphereEnable' type='SFBool'/>: Το πεδίο αυτό δέχεται ένα boolean ορισμά που αφορά την ενεργοποίηση του sphereSensor.

Στο interface για λόγους δοκιμών έχουν περιληφθεί επιπρόσθετα οι τιμές περιστροφής του κάθε transform καθώς και η τιμή translation του τελικού shapeTransform, δεν χρησιμοποιούνται όμως πουθενά στην τελική εφαρμογή.

2.6.5 Τελική Μορφή Της Εφαρμογής

Στον κορμό του προγράμματος είμαστε πλέον έτοιμοι να προσθέσουμε τα υλοποιημένα σχήματα στα πλήρως λειτουργικά parent groups που δημιουργήσαμε μέσω του πρωτοτύπου. Σε αυτό το προγραμματιστηκό επίπεδο θεωρούμε το πρωτότυπο ως ένα απλό transform χωρίς να μας απασχολεί η πραγματική σύνθετη δομή του.

Δημιουργούμε ένα στιγμιότυπο του πρωτοτύπου:

transforms[subShapeCounter]= mainScene.createProto("movementProto");

και το προσθέτουμε στη σκηνή μας:

mainScene.addRootNode(transforms[subShapeCounter]);

Το όνομα του στιγμιότυπου του πρωτοτύπου εισάγεται στην addFace ως το πατρικό node, και τα σχήματα προστίθονται στο group όπως περιγράφηκε στο αντίστοιχο κεφάλαιο. Για να δέσουμε τα πεδία του πρωτοτύπου με μεταβλητές χρησιμοποιούμε την πλέον γνωστή μέθοδο getField:

translation = transforms[subShapeCounter].getField("shapeTranslation");

rotation = transforms[subShapeCounter].getField("shapeRotation");

xyPlaneEnable = rootNodes[i].getField("xyPlaneEnable");

xzPlaneEnable = rootNodes[i].getField("xzPlaneEnable");

sphereEnable = rootNodes[i].getField("sphereEnable");

Οι τυχαίες θέσεις και περιστροφές εισάγωνται στα transforms:

translation.setValue(CubeFacesPoints.randomPosition());

rotation.setValue(CubeFacesPoints.randomRotation());

και το 3D Δυναμικό Puzzle είναι έτοιμο για χρηση!

2.7 Αξιολόγηση Μηχανισμού Κίνησης

Αξιολογώντας εκ του αποτελέσματος παρατηρούμε πως το bug της δυναμικής δημιουργίας των drag sensors στην πραγματικότητα ωφέλησε την εφαρμογή. Η αρχική ιδέα για τον μηχανισμό κίνησης των κομματιών προέβλεπε τη δημιουργία τριών sensors, των απαραίτητων groups και των αντίστοιχων routes στον κορμό του προγράμματος για καθε κομμάτι του puzzle. Αυτό σημαίνει δημιουργία 10 περίπου nodes για κάθε κομμάτι. Με τη χρήση των X3DPrototypes πλέον έχουμε τη δημιουργία ενος μόνο node για κάθε κομμάτι του puzzle. Η χρήση αυτής της τεχνικής μας εξοικονόμησε υπολογιστική ισχύ, κάτι που θα φανεί χρήσιμο σε πιο σύνθετες εφαρμογές. Ο τελικός μηχανισμός κίνησης όμως έχει ένα ακόμα bug το οποίο είναι αδύνατον να αντιμετωπιστεί. Οι SphereSensor και οι CylinderSensors διαθέτουν το πεδίο autoOffset. Το πεδίο αυτό καθορίζει εάν το σχήμα θα διατηρεί τις μεταβολές που του προκαλεί ο χρήστης, ή εάν κάθε περιστροφή θα ξεκινά πάντα από το αρχικό σημείο. Το bug του xj3d που μας δημιουργεί πρόβλημα αυτή τη φορά είναι ότι το πεδίο autoOffset, το οποίο εμείς το θέλουμε enabled, παραμένει πάντα απενεργοποιημένο. Αναφορά κατατέθηκε και γι' αυτό το bug στο Xj3D Bugzilla και η απάντηση ήταν και πάλι πως θα διορθωθεί σε κάποια επόμενη έκδοση. Λύση work-around δεν μπορεί να βρεθεί λόγω της πρωταρχικής θέσης του προβλήματος, οπότε και είμαστε αναγκασμένοι να περιμένουμε την διορθωμένη έκδοση του Xj3D. Ο κώδικας έχει γραφεί αγνοώντας το bug αυτό, έτσι ώστε όταν η εφαρμογή τρέξει σε πλατφόρμα με τη διορθωμένη έκδοση του Xj3D, ο SphereSensor να δουλέψει κανονικά χωρίς καμία αλλαγή στον κώδικα. Η μοναδική προσαρμογή που έχει γίνει για τους λόγους επίδειξης, είναι πως η γραμμή του κώδικα που δίνει την τυχαία περιστροφή στα κομμάτια (όπως περιγράφεται στο προηγούμενο κεφάλαιο), έχει μπει σε σχόλια έτσι ώστε το puzzle να μπορεί να λυθεί χωρίς τη χρήση του προβληματικού SphereSensor.

2.8 Συμπεράσματα και παρατηρήσεις για την X3D και το Xj3D Toolkit

Μετά την εκτεταμένη ασχολία με την X3D και το Xj3D Toolkit μπορούμε με ασφάλεια να καταλήξουμε σε συμπεράσματα. Η X3D ως γλώσσα και πρότυπο περιγραφής τρισδιάστατων σκήνων όντως δεν έχει να ζηλέψει τίποτα από την προκάτοχο VRML. Αντιθέτως αποδεικνύεται πολύ πιο ευέλικτη και προσαρμοστική όσον αφορά τα νέα στοιχεία και τις ιδιότητες που προστίθονται στο πρότυπο κατα καιρούς. Επίσης η X3D είναι πολύ φιλική στον μη έμπειρο χρήστη καθώς η μορφή του κώδικά της είναι πολύ οικεία σε οποιονδήποτε έχει ασχοληθεί έστω και λίγο με κάποια mark up language. Όσον αφορά το Xj3D Toolkit, το οποίο ουσιαστικά ήταν και το επίκεντρο του project, πρέπει να πούμε πως πρόκειται να εξελιχθεί σε ένα πανίσχυρο πακέτο διαχείρησης και επεξεργασίας σκηνών. Είναι αναμφισβήτητο πως το Xj3D Toolkit περιέχει ακόμα πολλά bugs, όμως αναπτύσεται από μια πολύ έμπειρη και υπεύθυνη ομάδα προγραμματιστών. Ας μην ξεχνάμε πως όλες οι προβληματικές εκδόσεις του πακέτου που αναφέρθηκαν παραπάνω ήταν developer's snapshots και όχι stable releases. Η X3D και το πακέτο Xj3D είναι ένας πολύ ενδιαφέρον, σχετικά νέος και ανεξερεύνητος τομέας των 3D γραφικών που αξίζει κάποιος να ασχοληθεί συστηματικότερα, αν διατίθεται να ξεπεράσει μόνος το –πιθανότατα προσωρινό– πρόβλημα της έλλειψης tutorials ή αναλυτικών οδηγών χρήσης.

3 ΠΑΡΑΡΤΗΜΑ

3.1 Η Κλάση CreateScene

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.HashMap;
import org.web3d.x3d.sai.*;

public class CreateScene extends JFrame {
        private ExternalBrowser x3dBrowser;
        private X3DScene mainScene;
        public CreateScene() {
                //window initialization
                super("3D Dynamic Puzzle");
                setDefaultCloseOperation(EXIT_ON_CLOSE);
                Container contentPane = getContentPane();
                HashMap requestedParameters = new HashMap();
                X3DComponent x3dComp =
                BrowserFactory.createX3DComponent(requestedParameters);
                JComponent x3dPanel =
        (JComponent)x3dComp.getImplementation();
                contentPane.add(x3dPanel, BorderLayout.CENTER);
                contentPane.add(leftPanel = new JPanel(), BorderLayout.WEST);
                makeGUI();
```

```
            ExternalBrowser x3dBrowser = x3dComp.getBrowser();
            setSize(800,525);
            setVisible(true);

    //scene initialization
            ProfileInfo profile = null;
             try {
                        profile = x3dBrowser.getProfile("Immersive");
             } catch(NotSupportedException nse) {
             System.out.println("Immersive Profile not supported");
             System.exit(-1);
             }

 mainScene = x3dBrowser.createX3DFromURL(new String[] {
"movementProto2.x3d" });
x3dBrowser.replaceWorld(mainScene);
}

private void startScene(String shapeSelection, int divsPerSide ){
        System.out.println("edw 3ekinaei"+shapeSelection);
        int numOfSubShapes = (int) (Math.pow((divsPerSide),3));
        X3DProtoInstance[] transforms = new
X3DProtoInstance[numOfSubShapes];
        int subShapeCounter=0;
```

```
if(shapeSelection=="Cube"){
        String frontTexture = textureText1.getText();
        String backTexture = textureText2.getText();
        String leftTexture = textureText3.getText();
        String rightTexture = textureText4.getText();
        String topTexture = textureText5.getText();
        String bottomTexture = textureText6.getText();
        CubeFacesPoints[] subShapeFacesPoints = new
CubeFacesPoints[numOfSubShapes];
        for(int x=0; x<divsPerSide; x++)
                for(int y=0; y<divsPerSide; y++)
                        for(int z=0; z<divsPerSide; z++){
                        float currentXCoord = CubeFacesPoints.fiveSignif((float)
x/divsPerSide);
                        float currentYCoord = CubeFacesPoints.fiveSignif((float)
y/divsPerSide);
                        float currentZCoord = CubeFacesPoints.fiveSignif((float)
z/divsPerSide);
                        float endCoord = CubeFacesPoints.fiveSignif((float)
                        (divsPerSide-1)/divsPerSide);
                        transforms[subShapeCounter] =
                        mainScene.createProto("movementProto");
                        mainScene.addRootNode(transforms[subShapeCounter]);
                        SFVec3f translation = (SFVec3f)
                        transforms[subShapeCounter].getField("shapeTranslation");
                        //SFRotation rotation = (SFRotation)
                        transforms[subShapeCounter].getField("shapeRotation");
```

if(divsPerSide!=1){

translation.setValue(CubeFacesPoints.randomPosition());

//rotation.setValue(CubeFacesPoints.randomRotation());

}

subShapeFacesPoints[subShapeCounter]=new CubeFacesPoints(new Point(currentXCoord, currentYCoord, currentZCoord), divsPerSide);

if((currentXCoord==0) || (currentYCoord==0) || (currentZCoord==0) ||

(currentXCoord==endCoord) || (currentYCoord==endCoord) || (currentZCoord==endCoord)){

//PROSTHETO FACES MONO GIA PERIFERIAKOUS KYVOUS

addFace(subShapeFacesPoints[subShapeCounter].publicLeftFacePoints, subShapeFacesPoints[subShapeCounter].publicLeftFaceTexturePoints, leftTexture, transforms[subShapeCounter], subShapeFacesPoints[subShapeCounter].arreyLength);

addFace(subShapeFacesPoints[subShapeCounter].publicRightFacePoints, subShapeFacesPoints[subShapeCounter].publicRightFaceTexturePoints, rightTexture, transforms[subShapeCounter], subShapeFacesPoints[subShapeCounter].arreyLength);

addFace(subShapeFacesPoints[subShapeCounter].publicBackFacePoints, subShapeFacesPoints[subShapeCounter].publicBackFaceTexturePoints, backTexture, transforms[subShapeCounter], subShapeFacesPoints[subShapeCounter].arreyLength);

addFace(subShapeFacesPoints[subShapeCounter].publicFrontFacePoints, subShapeFacesPoints[subShapeCounter].publicFrontFaceTexturePoints, frontTexture, transforms[subShapeCounter], subShapeFacesPoints[subShapeCounter].arreyLength);

```
        addFace(subShapeFacesPoints[subShapeCounter].publicBottomFacePoi
nts, subShapeFacesPoints[subShapeCounter].publicBottomFaceTexturePoints,
bottomTexture, transforms[subShapeCounter],
subShapeFacesPoints[subShapeCounter].arreyLength);


        addFace(subShapeFacesPoints[subShapeCounter].publicTopFacePoints,
subShapeFacesPoints[subShapeCounter].publicTopFaceTexturePoints,
topTexture, transforms[subShapeCounter],
subShapeFacesPoints[subShapeCounter].arreyLength);




}subShapeCounter++;

}       }


if(shapeSelection=="Pyramid"){

        float step = CubeFacesPoints.fiveSignif((float) 1/divsPerSide);

        float ls = (float) CubeFacesPoints.fiveSignif((float) step/2);

        String frontTexture = textureText1.getText();

        String backTexture = textureText2.getText();

        String leftTexture = textureText3.getText();

        String rightTexture = textureText4.getText();

        String bottomTexture = textureText5.getText();

        X3DNode pyramidTransform = mainScene.createProto("movementProto");

        SFVec3f translation1 = (SFVec3f)
pyramidTransform.getField("shapeTranslation");

        //SFRotation rotation1 = (SFRotation)
pyramidTransform.getField("shapeRotation");

        if(divsPerSide!=1){
```

```
            translation1.setValue(CubeFacesPoints.randomPosition());

            //rotation1.setValue(CubeFacesPoints.randomRotation());

        }

        mainScene.addRootNode(pyramidTransform);

        PyramidFacesPoints topPyramid = new PyramidFacesPoints(divsPerSide);

        addFace(topPyramid.publicFrontFacePoints,
        topPyramid.publicFrontFaceTexturePoints, frontTexture, pyramidTransform,
        topPyramid.arreyLength);

        addFace(topPyramid.publicLeftFacePoints,
        topPyramid.publicLeftFaceTexturePoints, leftTexture, pyramidTransform,
        topPyramid.arreyLength);

        addFace(topPyramid.publicRightFacePoints,
        topPyramid.publicRightFaceTexturePoints, rightTexture, pyramidTransform,
        topPyramid.arreyLength);

        addFace(topPyramid.publicBackFacePoints,
        topPyramid.publicBackFaceTexturePoints, backTexture,
        pyramidTransform, topPyramid.arreyLength);

        addFace(topPyramid.publicBottomFacePoints,
        topPyramid.publicBottomFaceTexturePoints, bottomTexture,
        pyramidTransform, 4);


        if(divsPerSide >= 2){

            TrapezeFacesPoints[] subShapeFacesPoints = new
            TrapezeFacesPoints[numOfSubShapes];

for(int y=0; y<(divsPerSide-1); y++)

        for(int x=0; x<(divsPerSide-y); x++)

            for(int z=0; z<(divsPerSide-y); z++){

            float endCoord = CubeFacesPoints.fiveSignif((float) 1-step-y*ls);

            float currentYCoord = CubeFacesPoints.fiveSignif((float)
y/divsPerSide);
```

```
                float xzOffset = CubeFacesPoints.fiveSignif( (float) y*ls);

                float partialXCoord = CubeFacesPoints.fiveSignif( (float)
x/divsPerSide);

                float currentXCoord = CubeFacesPoints.fiveSignif( (float)
                partialXCoord + xzOffset);

                float partialZCoord = CubeFacesPoints.fiveSignif( (float)
z/divsPerSide);

                float currentZCoord = CubeFacesPoints.fiveSignif( (float)
                partialZCoord + xzOffset);

                transforms[subShapeCounter] =
mainScene.createProto("movementProto");

                mainScene.addRootNode(transforms[subShapeCounter]);

                SFVec3f translation = (SFVec3f)
                transforms[subShapeCounter].getField("shapeTranslation");

                //SFRotation rotation = (SFRotation)
                transforms[subShapeCounter].getField("shapeRotation");



if(divsPerSide!=1){

        translation.setValue(CubeFacesPoints.randomPosition());

        //rotation.setValue(CubeFacesPoints.randomRotation());

}


if( (currentZCoord == endCoord) || (currentZCoord == xzOffset)

|| (currentXCoord == xzOffset) || (currentXCoord == endCoord) ||
(currentYCoord == 0)){

        subShapeFacesPoints[subShapeCounter]=new TrapezeFacesPoints(new
        Point(currentXCoord, currentYCoord, currentZCoord), divsPerSide);
```

```
	//System.out.println("("+currentXCoord+", "+currentYCoord+",
"+currentZCoord+")");

		addFace(subShapeFacesPoints[subShapeCounter].publicFrontFacePoints
, subShapeFacesPoints[subShapeCounter].publicFrontFaceTexturePoints,
frontTexture, transforms[subShapeCounter],
subShapeFacesPoints[subShapeCounter].arreyLength);


		addFace(subShapeFacesPoints[subShapeCounter].publicLeftFacePoints,
subShapeFacesPoints[subShapeCounter].publicLeftFaceTexturePoints,
leftTexture, transforms[subShapeCounter],
subShapeFacesPoints[subShapeCounter].arreyLength);


		addFace(subShapeFacesPoints[subShapeCounter].publicRightFacePoints
, subShapeFacesPoints[subShapeCounter].publicRightFaceTexturePoints,
rightTexture, transforms[subShapeCounter],
subShapeFacesPoints[subShapeCounter].arreyLength);


		addFace(subShapeFacesPoints[subShapeCounter].publicBackFacePoints
, subShapeFacesPoints[subShapeCounter].publicBackFaceTexturePoints,
backTexture, transforms[subShapeCounter],
subShapeFacesPoints[subShapeCounter].arreyLength);


		addFace(subShapeFacesPoints[subShapeCounter].publicTopFacePoints,
subShapeFacesPoints[subShapeCounter].publicTopFaceTexturePoints, "",
transforms[subShapeCounter],
subShapeFacesPoints[subShapeCounter].arreyLength);


		addFace(subShapeFacesPoints[subShapeCounter].publicBottomFacePoi
nts, subShapeFacesPoints[subShapeCounter].publicBottomFaceTexturePoints,
bottomTexture, transforms[subShapeCounter],
subShapeFacesPoints[subShapeCounter].arreyLength);

}subShapeCounter++;

		}}

}

if(shapeSelection == "Cylinder"){
```

```
String roundTexture = textureText1.getText();

String topTexture = textureText2.getText();

String bottomTexture = textureText3.getText();

CylinderFacesPoints[] subShapeFacesPoints = new
CylinderFacesPoints[divsPerSide];

float step = (float) 360/divsPerSide;

int intStep = (int) Math.floor(step);

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

        float currentStartDeg = j*step;

        subShapeFacesPoints[j] = new CylinderFacesPoints(currentStartDeg,
divsPerSide);

        transforms[j] = mainScene.createProto("movementProto");

        mainScene.addRootNode(transforms[j]);

        SFVec3f translation = (SFVec3f) transforms[j].getField("shapeTranslation");

        //SFRotation rotation = (SFRotation) transforms[j].getField("shapeRotation");

        if(divsPerSide!=1){

                translation.setValue(CubeFacesPoints.randomPosition());

                //rotation.setValue(CubeFacesPoints.randomRotation());

        }
```

//EDW KANW TA ADD FACE MONO

```
for (int i=0; i<=intStep; i++){
        addFace((float[][])subShapeFacesPoints[j].publicRoundFacePoints.eleme
ntAt(i), (float[][])
```

```
subShapeFacesPoints[j].publicRoundFaceTexturePoints.elementAt(i),
roundTexture, transforms[j], 4);

        addFace((float[][])subShapeFacesPoints[j].publicTopFacePoints.elementA
t(i), (float[][]) subShapeFacesPoints[j].publicTopFaceTexturePoints.elementAt(i),
topTexture, transforms[j], 3);

        addFace((float[][])subShapeFacesPoints[j].publicBottomFacePoints.eleme
ntAt(i), (float[][])
subShapeFacesPoints[j].publicBottomFaceTexturePoints.elementAt(i),
bottomTexture, transforms[j], 3);

addFace(subShapeFacesPoints[j].publicEndSideFace, (float[][]) null, "",
transforms[j], 4);

addFace(subShapeFacesPoints[j].publicStartSideFace, (float[][]) null, "",
transforms[j], 4);

        }

}

//edw telos tis dimiourgias

}
```

```
public void addFace(float[][] points, float[][] texturepoints, String texturePath,
X3DNode parent, int arreyLength){

        float[][] mfValues = points, /*gia dokimi
        {{(float)+0.25,(float)+0.25,(float)+0.25},{(float)-
        0.25,(float)+0.25,(float)+0.25},{(float)-0.25,(float)-
        0.25,(float)+0.25},{(float)+0.25,(float)-0.25,(float)+0.25}},*/

        mfTexValues = texturepoints;  /*gia dokimi {{0,0},{1,0},{1,1},{0,1}};*/

        float[] red = {1,0,0};

        int[] indexValues = new int[arreyLength+1];

        for(int i=0; i<arreyLength;i++)

                indexValues[i]=i;

        indexValues[arreyLength] = -1;

        X3DNode myParent = parent;

        X3DNode shape = mainScene.createNode("Shape");

//GEOMETRY SETUP

//denw to geometry me to IFS

        SFNode shape_geometry = (SFNode) (shape.getField("geometry"));

        X3DNode indexedFaceSet = mainScene.createNode("IndexedFaceSet");

//eisagw tis times coordIndex sto IFS

        MFInt32 mCoordIndex = (MFInt32) indexedFaceSet.getField("coordIndex");

        mCoordIndex.setValue(arreyLength, indexValues);

//eisagw tis times texCoordIndex sto IFS

        MFInt32 mTextureCoordinateIndex = (MFInt32)
        indexedFaceSet.getField("texCoordIndex");

        mTextureCoordinateIndex.setValue(arreyLength, indexValues);

//denw to IFS me to coordNode

        SFNode sCoord = (SFNode) indexedFaceSet.getField("coord");
```

```
        X3DNode coordNode = mainScene.createNode("Coordinate");
        sCoord.setValue(coordNode);
//eisagw ton pinaka simeiwn sto coordNode
        MFVec3f mPoint = (MFVec3f) coordNode.getField("point");
        mPoint.setValue(arreyLength, mfValues);
//denw to IFS me to textureCoordNode
        SFNode sTextureCoord = (SFNode) indexedFaceSet.getField("texCoord");
        X3DNode textureCoordNode =
mainScene.createNode("TextureCoordinate");
        sTextureCoord.setValue(textureCoordNode);
        if (texturepoints != null){
//eisagw ton pinaka simeiwn sto textureCoordNode
        MFVec2f mTexturePoint = (MFVec2f) textureCoordNode.getField("point");
        mTexturePoint.setValue(arreyLength, mfTexValues);
                                                }

//APPEARANCE SETUP
        SFNode shape_appearance = (SFNode) shape.getField("appearance");
        X3DNode appearanceNode = mainScene.createNode("Appearance");
//denw to materialNode sto appearanceNode
        SFNode sMaterial = (SFNode) appearanceNode.getField("material");
        X3DNode materialNode = mainScene.createNode("Material");
        if (materialNode == null) {
                System.out.println("Couldn't create material");
        }
        sMaterial.setValue(materialNode);
```

```
//dinw timi sto pedio diffuseColor tou MaterialNode
        SFColor sColor = (SFColor) materialNode.getField("diffuseColor");
        sColor.setValue(red);
//denw to textureNode sto appearanceNode an yparxei texture
        if (texturepoints != null){
                SFNode sTexture = (SFNode) appearanceNode.getField("texture");
                X3DNode imageTextureNode =
mainScene.createNode("ImageTexture");
                sTexture.setValue(imageTextureNode);
                //dinw timi sto textureURL tou imageTextureNode
                MFString sTextureURL = (MFString) imageTextureNode.getField("url");
                sTextureURL.setValue(1,new String[]{texturePath});
        }
//PROSTHETO TA SETARISMENA APPEARANCE KAI GEOMETRY STO SHAPE
        shape_geometry.setValue(indexedFaceSet);
        shape_appearance.setValue(appearanceNode);
//add shape
        MFNode children = (MFNode) myParent.getField("movingChildren");
        children.setValue(1, new X3DNode[] {shape});
}
```

```
private void makeGUI(){
        leftPanel.setLayout(new BorderLayout());
        leftPanel.add(topPanel = new JPanel(), BorderLayout.NORTH);
        topPanel.setLayout(new BorderLayout());
        shapeButtonGroup = new ButtonGroup();
        cubeChoice = new JRadioButton("Cube", true);
        cubeChoice.setActionCommand("Cube");
        pyramidChoice = new JRadioButton("Pyramid");
        pyramidChoice.setActionCommand("Pyramid");
        cylinderChoice = new JRadioButton("Cylinder");
        cylinderChoice.setActionCommand("Cylinder");
        shapeButtonGroup.add(cubeChoice);
        shapeButtonGroup.add(pyramidChoice);
        shapeButtonGroup.add(cylinderChoice);
        topPanel.add(cubeChoice, BorderLayout.NORTH);
        topPanel.add(pyramidChoice, BorderLayout.CENTER);
        topPanel.add(cylinderChoice, BorderLayout.SOUTH);
        topPanel.add(divsPanel = new JPanel(), BorderLayout.EAST);
        divsPanel.setLayout(new GridLayout(2,1));
        divsPanel.add(divsLabel = new JLabel("Divisions: "));
        divsPanel.add(divsText = new JTextField("1"));
        //System.out.println("edw:
        "+shapeButtonGroup.getSelection().getActionCommand());
        leftPanel.add(middlePanel = new JPanel(), BorderLayout.CENTER);
        middlePanel.setLayout(new BorderLayout());
```

```
        middlePanel.add(createButton = new JButton("Start!"),
BorderLayout.NORTH);

        middlePanel.add(clearButton = new JButton("Clear"),
BorderLayout.SOUTH);

        xj3dImage = new ImageIcon("3dpuzzLogo.jpg");

        middlePanel.add(xj3dLabel = new JLabel(xj3dImage),
BorderLayout.WEST);

        middlePanel.add(movementPanel = new JPanel(), BorderLayout.CENTER);

        movementPanel.setLayout(new GridLayout(3,1));

        rotateChoice = new JRadioButton("Rotate");

        xyChoice = new JRadioButton("xy Plane",true);

        xzChoice = new JRadioButton("xz Plane");

        movementButtonGroup = new ButtonGroup();

        movementButtonGroup.add(rotateChoice);

        movementButtonGroup.add(xyChoice);

        movementButtonGroup.add(xzChoice);

        movementPanel.add(rotateChoice);

        movementPanel.add(xyChoice);

        movementPanel.add(xzChoice);

        leftPanel.add(texturePanel = new JPanel(), BorderLayout.SOUTH);

        texturePanel.setLayout(new GridLayout(6,2));

        texturePanel.add(textureLabel1 = new JLabel("Front Texture:"));

        texturePanel.add(textureText1 = new JTextField("front.jpg"));

        texturePanel.add(textureLabel2 = new JLabel("Back Texture:"));

        texturePanel.add(textureText2 = new JTextField("back.jpg"));

        texturePanel.add(textureLabel3 = new JLabel("Left Texture:"));

        texturePanel.add(textureText3 = new JTextField("left.jpg"));
```

```
texturePanel.add(textureLabel4 = new JLabel("Right Texture:"));
texturePanel.add(textureText4 = new JTextField("right.jpg"));
texturePanel.add(textureLabel5 = new JLabel("Top Texture:"));
texturePanel.add(textureText5 = new JTextField("top.jpg"));
texturePanel.add(textureLabel6 = new JLabel("Bottom Texture:"));
texturePanel.add(textureText6 = new JTextField("bottom.jpg"));

createButton.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e){
        String myShapeSelection =
        shapeButtonGroup.getSelection().getActionCommand();
        int myDivsPerSide = Integer.parseInt(divsText.getText());
        System.out.println(myShapeSelection);
        startScene(myShapeSelection, myDivsPerSide);
        }
});

clearButton.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e){
        clearScene();
        }
});

pyramidChoice.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e){
        textureLabel1.setText("Front Texture:");
```

```
                textureText1.setText("front.jpg");
                textureLabel2.setText("Back Texture:");
                textureText2.setText("back.jpg");
                textureLabel3.setText("Left Texture:");
                textureText3.setText("left.jpg");
                textureLabel4.setVisible(true);
                textureText4.setVisible(true);
                textureLabel4.setText("Right Texture:");
                textureText4.setText("right.jpg");
                textureLabel5.setVisible(true);
                textureText5.setVisible(true);
                textureLabel5.setText("Bottom Texture:");
                textureText5.setText("bottom.jpg");
                textureLabel6.setVisible(false);
                textureText6.setVisible(false);
                }
        });
cylinderChoice.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e){
        textureLabel1.setText("Round Texture:");
        textureText1.setText("eikona.jpg");
        textureLabel2.setText("Top Texture:");
        textureText2.setText("top.jpg");
        textureLabel3.setText("Bottom Texture:");
        textureText3.setText("bottom.jpg");
```

```
        textureLabel4.setVisible(false);
        textureText4.setVisible(false);
        textureLabel5.setVisible(false);
        textureText5.setVisible(false);
        textureLabel6.setVisible(false);
        textureText6.setVisible(false);
        }
});

cubeChoice.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e){
        textureLabel1.setText("Front Texture:");
        textureText1.setText("front.jpg");
        textureLabel2.setText("Back Texture:");
        textureText2.setText("back.jpg");
        textureLabel3.setText("Left Texture:");
        textureText3.setText("left.jpg");
        textureLabel4.setVisible(true);
        textureText4.setVisible(true);
        textureLabel4.setText("Right Texture:");
        textureText4.setText("right.jpg");
        textureLabel5.setVisible(true);
        textureText5.setVisible(true);
        textureLabel5.setText("Top Texture:");
        textureText5.setText("top.jpg");
```

```
        textureLabel6.setVisible(true);
        textureText6.setVisible(true);
        textureLabel6.setText("Bottom Texture:");
        textureText6.setText("bottom.jpg");
        }
});

rotateChoice.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e){
        X3DNode[] rootNodes = mainScene.getRootNodes();
        System.out.println(rootNodes.length);
        for(int i=1; i<rootNodes.length; i++){
                SFBool xyPlaneEnable = (SFBool)
rootNodes[i].getField("xyPlaneEnable");
                SFBool xzPlaneEnable = (SFBool)
rootNodes[i].getField("xzPlaneEnable");
                SFBool sphereEnable = (SFBool)
rootNodes[i].getField("sphereEnable");
                xyPlaneEnable.setValue(false);
                xzPlaneEnable.setValue(false);
                sphereEnable.setValue(true);
        }
        System.out.println(lastMovementChoice);
        }
});

xyChoice.addActionListener(new ActionListener(){
```

```
        public void actionPerformed(ActionEvent e){
        X3DNode[] rootNodes = mainScene.getRootNodes();
        System.out.println(rootNodes.length);
        for(int i=1; i<rootNodes.length; i++){
                SFBool xyPlaneEnable = (SFBool)
rootNodes[i].getField("xyPlaneEnable");
                SFBool xzPlaneEnable = (SFBool)
rootNodes[i].getField("xzPlaneEnable");
                SFBool sphereEnable = (SFBool)
rootNodes[i].getField("sphereEnable");
                sphereEnable.setValue(false);
                xzPlaneEnable.setValue(false);
                xyPlaneEnable.setValue(true);
                SFRotation sensorRotation = (SFRotation)
                rootNodes[i].getField("sensorRotation");
                SFRotation slidingRotation = (SFRotation)
                rootNodes[i].getField("slidingRotation");
                }
        System.out.println(lastMovementChoice);
        }
});

xzChoice.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e){
        X3DNode[] rootNodes = mainScene.getRootNodes();
        System.out.println(rootNodes.length);
        for(int i=1; i<rootNodes.length; i++){
```

```
                SFBool xyPlaneEnable = (SFBool)
rootNodes[i].getField("xyPlaneEnable");

                SFBool xzPlaneEnable = (SFBool)
rootNodes[i].getField("xzPlaneEnable");

                SFBool sphereEnable = (SFBool)
rootNodes[i].getField("sphereEnable");

                sphereEnable.setValue(false);

                xyPlaneEnable.setValue(false);

                xzPlaneEnable.setValue(true);

                SFRotation xyRotation = (SFRotation)
rootNodes[i].getField("xyRotation");

                SFRotation xzRotation = (SFRotation)
rootNodes[i].getField("xzRotation");

                //xyRotation.setValue(new float[] {1,0,0,(float)-1.57});

                //System.out.println("xyRotation: "+xyRotation.toString()+",
                xzRotation: "+xzRotation.toString());

                }

        }

});

}


private void clearScene(){

        X3DNode[] nodesToClear = mainScene.getRootNodes();

        int numOfRootNodes = nodesToClear.length;

        for(int i=0;i<numOfRootNodes;i++)

                mainScene.removeRootNode(nodesToClear[i]);

        }
```

```
public static void main(String[] args) {
        new CreateScene();
}
```

```
private JLabel divsLabel,textureLabel1, textureLabel2, textureLabel3,
textureLabel4, textureLabel5, textureLabel6, xj3dLabel ;

private JPanel leftPanel, topPanel, divsPanel, texturePanel, middlePanel,
movementPanel;

private ButtonGroup shapeButtonGroup, movementButtonGroup;

private JRadioButton cubeChoice, pyramidChoice, cylinderChoice,
rotateChoice, xyChoice, xzChoice;

private JButton createButton, clearButton;

private JTextField divsText, textureText1, textureText2, textureText3, textureText4,
textureText5, textureText6;

private ImageIcon xj3dImage;

private JSeparator jSeparator1, jSeparator2;

private String lastMovementChoice="xyPlane", currentPlane="vertical";

}
```

3.2 Η Κλάση CubeFacesPoints

public class CubeFacesPoints {

public CubeFacesPoints(Point initialPoint, int partsPerSide){

float x = initialPoint.x;

float y = initialPoint.y;

float z = initialPoint.z;

float step = (float) 1/partsPerSide;

loat halfLength = (float) step/2;

float endCoord = fiveSignif((float) (1-step));

//System.out.println("Cube gets: "+initialPoint.x+","+initialPoint.y+","+initialPoint.z);

float[][] leftFacePoints = {{-halfLength, -halfLength, -halfLength},{-halfLength, -halfLength, +halfLength},{-halfLength, +halfLength, +halfLength},{-halfLength, +halfLength, -halfLength}};

float[][] rightFacePoints = {{+halfLength, +halfLength, +halfLength}, {+halfLength, -halfLength, +halfLength}, {+halfLength, -halfLength, -halfLength}, {+halfLength, +halfLength, -halfLength}};

float[][] frontFacePoints = {{+halfLength, +halfLength, +halfLength}, {-halfLength, +halfLength, +halfLength}, {-halfLength, -halfLength, +halfLength}, {+halfLength, -halfLength, +halfLength}};

float[][] backFacePoints = {{-halfLength, -halfLength, -halfLength},{-halfLength, +halfLength, -halfLength},{+halfLength, +halfLength, -halfLength},{+halfLength, -halfLength, -halfLength}};

float[][] topFacePoints = {{+halfLength, +halfLength, +halfLength}, {+halfLength, +halfLength, -halfLength},{-halfLength, +halfLength, -halfLength}, {-halfLength, +halfLength, +halfLength}};

float[][] bottomFacePoints = {{-halfLength, -halfLength, -halfLength}, {+halfLength, -halfLength, -halfLength}, {+halfLength, -halfLength, +halfLength}, {-halfLength, -halfLength, +halfLength}};

publicTopFacePoints = topFacePoints;

publicBottomFacePoints = bottomFacePoints;

publicFrontFacePoints = frontFacePoints;

publicBackFacePoints = backFacePoints;

publicLeftFacePoints = leftFacePoints;

publicRightFacePoints = rightFacePoints;

if(partsPerSide==1){

float[][] leftFaceTexturePoints = { {0,0}, {1, 0}, {1, 1}, {0, 1} };

float[][] rightFaceTexturePoints={ {0, 1}, {0, 0}, {1, 0}, {1, 1} };

float[][] backFaceTexturePoints={{1, 0}, {1, 1}, {0, 1}, {0, 0}};

float[][] frontFaceTexturePoints = { {1, 1}, {0, 1}, {0, 0}, {1, 0} };

float[][] bottomFaceTexturePoints = { {0, 0}, {1, 0}, {1, 1}, {0, 1} };

float[][] topFaceTexturePoints = { {1, 0}, {1, 1}, {0, 1}, {0, 0} };

publicLeftFaceTexturePoints = leftFaceTexturePoints;

publicRightFaceTexturePoints = rightFaceTexturePoints;

```
        publicFrontFaceTexturePoints = frontFaceTexturePoints;

        publicBackFaceTexturePoints = backFaceTexturePoints;

        publicTopFaceTexturePoints = topFaceTexturePoints;

        publicBottomFaceTexturePoints = bottomFaceTexturePoints;

}else{

//FTIAXNW LEFT KAI RIGHT. H THESI TOU KYVOU EPIRREAZEI MONO TO TEXTURE OPOTE MONO AFTO ENTOS THS IF

if(x==0){

        float[][] leftFaceTexturePoints = { {z,y}, {z+step, y}, {z+step , y+step}, {z , y+step} };

        float[][] rightFaceTexturePoints = null;

        publicLeftFaceTexturePoints = leftFaceTexturePoints;

        publicRightFaceTexturePoints = rightFaceTexturePoints;

        } else

if(x==endCoord){

        float[][] leftFaceTexturePoints = null;

        float[][] rightFaceTexturePoints={ {1-(z+step), y+step}, {1-(z+step), y}, {1-z, y}, {1-z, y+step}  };

        publicLeftFaceTexturePoints = leftFaceTexturePoints;

        publicRightFaceTexturePoints = rightFaceTexturePoints;

        } else{

float[][] leftFaceTexturePoints = null;

        float[][] rightFaceTexturePoints = null;

        publicLeftFaceTexturePoints = leftFaceTexturePoints;

        publicRightFaceTexturePoints = rightFaceTexturePoints;

        }
```

//FTIAXNW FRONT KAI BACK. H THESI TOU KYVOU EPIRREAZEI MONO TO TEXTURE OPOTE MONO AFTO ENTOS THS IF

```
if(z==0){

        float[][] frontFaceTexturePoints = null;

        float[][] backFaceTexturePoints={{1-x, y}, {1-x, y+step}, {1-(x+step), y+step},
        {1-(x+step), y}};

        publicFrontFaceTexturePoints = frontFaceTexturePoints;

        publicBackFaceTexturePoints = backFaceTexturePoints;

        }else

if(z==endCoord){

        float[][] frontFaceTexturePoints = { {x+step, y+step}, {x, y+step}, {x, y},
{x+step, y} };

        float[][] backFaceTexturePoints = null;

        publicFrontFaceTexturePoints = frontFaceTexturePoints;

        publicBackFaceTexturePoints = backFaceTexturePoints;

        } else{

float[][] frontFaceTexturePoints = null;

float[][] backFaceTexturePoints = null;

publicFrontFaceTexturePoints = frontFaceTexturePoints;

publicBackFaceTexturePoints = backFaceTexturePoints;

}
```

//FTIAXNW TOP KAI BOTTOM. H THESI TOU KYVOU EPIRREAZEI MONO TO TEXTURE OPOTE MONO AFTO ENTOS THS IF

```
if(y==0){

        float[][] topFaceTexturePoints = null;

        float[][] bottomFaceTexturePoints = { {x, z}, {x+step, z}, {x+step, z+step}, {x,
z+step} };
```

```
        publicTopFaceTexturePoints = topFaceTexturePoints;
        publicBottomFaceTexturePoints = bottomFaceTexturePoints;
        } else
if(y==endCoord){
        float[][] topFaceTexturePoints = { {x+step, 1-(z+step)}, {x+step, 1-z}, {x, 1-z},
        {x, 1-(z+step)} };
        float[][] bottomFaceTexturePoints = null;
        publicTopFaceTexturePoints = topFaceTexturePoints;
        publicBottomFaceTexturePoints = bottomFaceTexturePoints;
        } else{
float[][] topFaceTexturePoints = null;
float[][] bottomFaceTexturePoints = null;
publicTopFaceTexturePoints = topFaceTexturePoints;
publicBottomFaceTexturePoints = bottomFaceTexturePoints;
}}              }
public static float fiveSignif(float number){
        float multiedNumber = 100000*number;
        float resultNumber = (float) (Math.round(multiedNumber))/100000;
        return resultNumber;
}

public static float randomFloat(){
        int plusminus = 1;
        if(new java.util.Random().nextBoolean())
                plusminus = -1;
        float randFloat = (float) plusminus * (new java.util.Random().nextFloat());
```

```
        return randFloat;
}

public static float[] randomPosition(){
        float x = randomFloat();
        float y = randomFloat();
        float z = randomFloat();
        //System.out.println("("+x+","+y+","+z+")");
        float[] randPos = {x, y, z};
        return randPos;
}

public static float[] randomRotation(){
        float x = randomFloat();
        float y = randomFloat();
        float z = randomFloat();
        float angle = (float) Math.PI * randomFloat();
        //System.out.println("("+x+","+y+","+z+","+angle+")");
        float[] randRot = {x, y, z, angle};
        return randRot;
}

public float[][] publicLeftFacePoints,publicRightFacePoints,
publicBottomFacePoints, publicTopFacePoints, publicBackFacePoints,
publicFrontFacePoints,publicLeftFaceTexturePoints,
publicRightFaceTexturePoints, publicFrontFaceTexturePoints,
```

```
publicBackFaceTexturePoints, publicTopFaceTexturePoints,
publicBottomFaceTexturePoints;


public int arreyLength = 4;

private int step;


public static void main(String[] argz){

        CubeFacesPoints cube = new CubeFacesPoints(new Point(0,0,0), 2);

        }



}
```

3.3 Η Κλάση PyramidFacesPoints

```
public class PyramidFacesPoints {

public PyramidFacesPoints(int partsPerSide){

        float step = (float) CubeFacesPoints.fiveSignif((float) 1/partsPerSide);

        float ls = (float) CubeFacesPoints.fiveSignif((float) step/2);

        float partialEndCoord = CubeFacesPoints.fiveSignif((float)(partsPerSide-
1)/partsPerSide);

        float pyramidXZ = (float) 0.5/partsPerSide;

        float plus = (float) 0.5+pyramidXZ;
```

```
        float minus = (float) 0.5-pyramidXZ;

        float standarPlus = (float) +pyramidXZ;

        float standarMinus = (float) -pyramidXZ;


float[][] frontFacePoints = {{ 0, +ls, 0}, {standarMinus, -ls, standarPlus}, { standarPlus, -ls, standarPlus} };

float[][] frontFaceTexturePoints = {{(float)0.5, (float)1}, {minus, 1-step}, {plus, 1-step}};

publicFrontFacePoints = frontFacePoints;

publicFrontFaceTexturePoints = frontFaceTexturePoints;


float[][] leftFacePoints = {{ 0, +ls, 0}, {standarMinus, -ls, standarMinus}, { standarMinus, -ls, standarPlus} };

float[][] leftFaceTexturePoints = {{(float)0.5, (float)1}, {minus, 1-step}, {plus, 1-step}};

publicLeftFacePoints = leftFacePoints;

publicLeftFaceTexturePoints = leftFaceTexturePoints;


float[][] rightFacePoints = {{ 0, +ls, 0}, {standarPlus, -ls, standarPlus}, { standarPlus, -ls, standarMinus} };

float[][] rightFaceTexturePoints = {{(float)0.5, (float)1}, {1-plus, 1-step}, {1-minus, 1-step}};

publicRightFacePoints = rightFacePoints;

publicRightFaceTexturePoints = rightFaceTexturePoints;


float[][] backFacePoints = {{ 0, +ls, 0}, {standarPlus, -ls, standarMinus}, { standarMinus, -ls, standarMinus} };

float[][] backFaceTexturePoints = {{(float)0.5, (float)1}, {1-plus, 1-step}, {1-minus, 1-step}};
```

```
publicBackFacePoints = backFacePoints;

publicBackFaceTexturePoints = backFaceTexturePoints;


float[][] bottomFacePoints = {{ standarMinus, -ls, standarMinus}, {standarPlus, -ls,
standarMinus}, { standarPlus, -ls, standarPlus}, {standarMinus, -ls, standarPlus} };

publicBottomFacePoints = bottomFacePoints;


if(partsPerSide == 1){

        float[][] bottomFaceTexturePoints = {{minus, minus}, {plus, minus}, {plus,
        plus}, {minus, plus}};

        publicBottomFaceTexturePoints = bottomFaceTexturePoints;

}else{

        float[][] bottomFaceTexturePoints = null;

        publicBottomFaceTexturePoints = bottomFaceTexturePoints;

        }

}

public float[][] publicLeftFacePoints, publicRightFacePoints,
publicBottomFacePoints,

 publicTopFacePoints, publicBackFacePoints, publicFrontFacePoints,

 publicLeftFaceTexturePoints, publicRightFaceTexturePoints,
publicFrontFaceTexturePoints,

 publicBackFaceTexturePoints,      publicTopFaceTexturePoints,
                publicBottomFaceTexturePoints;


public int arreyLength = 3;

private int step;

}
```

3.4 Η Κλάση TrapezeFacesPoints

```
public class TrapezeFacesPoints {

 public TrapezeFacesPoints(Point initialPoint, int partsPerSide){

        float x = initialPoint.x;
```

```
        float y = initialPoint.y;

        float z = initialPoint.z;

        float step = (float) CubeFacesPoints.fiveSignif((float) 1/partsPerSide);

        float ls = (float) CubeFacesPoints.fiveSignif((float) step/2);

        float offset = CubeFacesPoints.fiveSignif(y*partsPerSide*ls);

        float partialEndCoord = CubeFacesPoints.fiveSignif((float)(partsPerSide-
1)/partsPerSide);

        float endCoord = CubeFacesPoints.fiveSignif(partialEndCoord-offset);

        //System.out.println("Cube gets:
"+initialPoint.x+","+initialPoint.y+","+initialPoint.z);


if(z == endCoord){    //MPROSTINO KOMMATI, GYRTO PROS TA PISW

        if(x == offset){    //ARISTERO KOMMATI, PLAGIO ARISTERI PLEYRA

float[][] frontFacePoints = {{x, y, z+step}, {x+step, y, z+step}, {x+step, y+step, z+ls},
{x+ls, y+step, z+ls}};

float[][] frontFaceTexturePoints = {{x, y}, {x+step, y}, {x+step, y+step}, {x+ls,
y+step}};

publicFrontFacePoints = frontFacePoints;

publicFrontFaceTexturePoints = frontFaceTexturePoints;


float[][] backFacePoints = {{x, y, z}, {x+ls, y+step, z}, {x+step, y+step, z}, {x+step, y,
z}};

float[][] backFaceTexturePoints = null;

publicBackFacePoints = backFacePoints;

publicBackFaceTexturePoints = backFaceTexturePoints;

float[][] leftFacePoints = {{x, y, z}, {x, y, z+step}, {x+ls, y+step, z+ls}, {x+ls, y+step, z}};

float[][] leftFaceTexturePoints = {{z, y}, {z+step, y}, {z+ls, y+step}, {z, y+step}};
```

```
publicLeftFaceTexturePoints = leftFaceTexturePoints;

publicLeftFacePoints = leftFacePoints;


float[][] rightFacePoints = {{x+step, y, z}, {x+step, y+step, z}, {x+step, y+step, z+ls},
{x+step, y, z+step}};

float[][] rightFaceTexturePoints = null;

publicRightFacePoints = rightFacePoints;

publicRightFaceTexturePoints = rightFaceTexturePoints;


float[][] topFacePoints = {{x+ls, y+step, z}, {x+ls, y+step, z+ls}, {x+step, y+step, z+ls},
{x+step, y+step, z}};

float[][] topFaceTexturePoints = null;

publicTopFacePoints = topFacePoints;

publicTopFaceTexturePoints = topFaceTexturePoints;


float[][] bottomFacePoints = {{x, y, z}, {x+step, y, z}, {x+step, y, z+step}, {x, y,
z+step}};

publicBottomFacePoints = bottomFacePoints;

if(y==0){

        float[][] bottomFaceTexturePoints = {{x, z}, {x+step, z}, {x+step, z+step}, {x,
z+step}};

        publicBottomFaceTexturePoints = bottomFaceTexturePoints;

}else{

        float[][] bottomFaceTexturePoints = null;

        publicBottomFaceTexturePoints = bottomFaceTexturePoints;

}

}else
```

```
if(x == endCoord){            //DE3I KOMMATI, PLAGIA DE3IA PLEYRA

        float[][] frontFacePoints = {{x, y, z+step}, {x+step, y, z+step}, {x+ls, y+step,
        z+ls}, {x, y+step, z+ls}};

        float[][] frontFaceTexturePoints = {{x, y}, {x+step, y}, {x+ls, y+step}, {x,
y+step}};

        publicFrontFacePoints = frontFacePoints;

        publicFrontFaceTexturePoints = frontFaceTexturePoints;


        float[][] backFacePoints = {{x, y, z}, {x, y+step, z}, {x+ls, y+step, z}, {x+step, y,
z}};

        float[][] backFaceTexturePoints = null;

        publicBackFacePoints = backFacePoints;

        publicBackFaceTexturePoints = backFaceTexturePoints;


        float[][] leftFacePoints = {{x, y, z}, {x, y, z+step}, {x, y+step, z+ls}, {x, y+step,
z}};

        float[][] leftFaceTexturePoints = null;

        publicLeftFaceTexturePoints = leftFaceTexturePoints;

        publicLeftFacePoints = leftFacePoints;


        float[][] rightFacePoints = {{x+step, y, z}, {x+ls, y+step, z}, {x+ls, y+step, z+ls},
        {x+step, y, z+step}};

        float[][] rightFaceTexturePoints = {{1-z, y}, {1-z, y+step}, {1-(z+ls), y+step}, {1-
        (z+step), y}};

        publicRightFacePoints = rightFacePoints;

        publicRightFaceTexturePoints = rightFaceTexturePoints;
```

```
        float[][] topFacePoints = {{x, y+step, z}, {x, y+step, z+ls}, {x+ls, y+step, z+ls},
        {x+ls, y+step, z}};

        float[][] topFaceTexturePoints = null;

        publicTopFacePoints = topFacePoints;

        publicTopFaceTexturePoints = topFaceTexturePoints;


        float[][] bottomFacePoints = {{x, y, z}, {x+step, y, z}, {x+step, y, z+step}, {x, y,
z+step}};

        publicBottomFacePoints = bottomFacePoints;

        if(y==0){

                float[][] bottomFaceTexturePoints = {{x, z}, {x+step, z}, {x+step,
                z+step}, {x, z+step}};

                publicBottomFaceTexturePoints = bottomFaceTexturePoints;

        }else{

                float[][] bottomFaceTexturePoints = null;

                publicBottomFaceTexturePoints = bottomFaceTexturePoints;

        }


}else{

        float[][] frontFacePoints = {{x, y, z+step}, {x+step, y, z+step}, {x+step, y+step,
        z+ls}, {x, y+step, z+ls}};

        float[][] frontFaceTexturePoints = {{x, y}, {x+step, y}, {x+step, y+step}, {x,
y+step}};

        publicFrontFacePoints = frontFacePoints;

        publicFrontFaceTexturePoints = frontFaceTexturePoints;


        float[][] backFacePoints = {{x, y, z}, {x, y+step, z}, {x+step, y+step, z},
{x+step, y, z}};
```

```
        float[][] backFaceTexturePoints = null;

        publicBackFacePoints = backFacePoints;

        publicBackFaceTexturePoints = backFaceTexturePoints;
float[][] leftFacePoints = {{x, y, z}, {x, y, z+step}, {x, y+step, z+ls}, {x, y+step, z}};

float[][] leftFaceTexturePoints = null;

publicLeftFaceTexturePoints = leftFaceTexturePoints;

publicLeftFacePoints = leftFacePoints;


float[][] rightFacePoints = {{x+step, y, z}, {x+step, y+step, z}, {x+step, y+step, z+ls},
{x+step, y, z+step}};

float[][] rightFaceTexturePoints = null;

publicRightFacePoints = rightFacePoints;

publicRightFaceTexturePoints = rightFaceTexturePoints;


float[][] topFacePoints = {{x, y+step, z}, {x, y+step, z+ls}, {x+step, y+step, z+ls},
{x+step, y+step, z}};

float[][] topFaceTexturePoints = null;

publicTopFacePoints = topFacePoints;

publicTopFaceTexturePoints = topFaceTexturePoints;


float[][] bottomFacePoints = {{x, y, z}, {x+step, y, z}, {x+step, y, z+step}, {x, y,
z+step}};

publicBottomFacePoints = bottomFacePoints;

if(y==0){

        float[][] bottomFaceTexturePoints = {{x, z}, {x+step, z}, {x+step, z+step}, {x,
z+step}};

        publicBottomFaceTexturePoints = bottomFaceTexturePoints;
```

```
}else{
        float[][] bottomFaceTexturePoints = null;
        publicBottomFaceTexturePoints = bottomFaceTexturePoints;
}                                   }
}else
if(z == offset){          //PISO KOMMATI, GYRTO PROS TA MPROS
        if(x == offset){//ARISTERO KOMMATI, PLAGIO ARISTERI PLEYRA


float[][] frontFacePoints = {{x, y, z+step}, {x+step, y, z+step}, {x+step, y+step,
z+step}, {x+ls, y+step, z+step}};
float[][] frontFaceTexturePoints = null;
publicFrontFacePoints = frontFacePoints;
publicFrontFaceTexturePoints = frontFaceTexturePoints;


float[][] backFacePoints = {{x, y, z}, {x+ls, y+step, z+ls}, {x+step, y+step, z+ls},
{x+step, y, z}};
float[][] backFaceTexturePoints = {{1-x, y}, {1-(x+ls), y+step}, {1-(x+step), y+step},
{1-(x+step), y}};
publicBackFacePoints = backFacePoints;
publicBackFaceTexturePoints = backFaceTexturePoints;


float[][] leftFacePoints = {{x, y, z}, {x, y, z+step}, {x+ls, y+step, z+step}, {x+ls, y+step,
z+ls}};
float[][] leftFaceTexturePoints = {{z, y}, {z+step, y}, {z+step, y+step}, {z+ls, y+step}};
publicLeftFaceTexturePoints = leftFaceTexturePoints;
publicLeftFacePoints = leftFacePoints;
```

float[][] **rightFacePoints** = {{x+step, y, z}, {x+step, y+step, z+ls}, {x+step, y+step, z+step}, {x+step, y, z+step}};

float[][] **rightFaceTexturePoints** = null;

publicRightFacePoints = rightFacePoints;

publicRightFaceTexturePoints = rightFaceTexturePoints;

float[][] **topFacePoints** = {{x+ls, y+step, z+ls}, {x+ls, y+step, z+step}, {x+step, y+step, z+step}, {x+step, y+step, z+ls}};

float[][] **topFaceTexturePoints** = null;

publicTopFacePoints = topFacePoints;

publicTopFaceTexturePoints = topFaceTexturePoints;

float[][] **bottomFacePoints** = {{x, y, z}, {x+step, y, z}, {x+step, y, z+step}, {x, y, z+step}};

publicBottomFacePoints = bottomFacePoints;

if(y==0){

float[][] **bottomFaceTexturePoints** = {{x, z}, {x+step, z}, {x+step, z+step}, {x, z+step}};

publicBottomFaceTexturePoints = bottomFaceTexturePoints;

}else{

float[][] **bottomFaceTexturePoints** = null;

publicBottomFaceTexturePoints = bottomFaceTexturePoints;

}}else

if(x == endCoord){ **//DE3I KOMMATI, GYRTI DE3IA PLEVRA**

float[][] **frontFacePoints** = {{x, y, z+step}, {x+step, y, z+step}, {x+ls, y+step, z+step}, {x, y+step, z+step}};

float[][] **frontFaceTexturePoints** = null;

publicFrontFacePoints = frontFacePoints;

publicFrontFaceTexturePoints = frontFaceTexturePoints;

float[][] **backFacePoints** = {{x, y, z}, {x, y+step, z+ls}, {x+ls, y+step, z+ls}, {x+step, y, z}};

float[][] **backFaceTexturePoints** = {{1-x, y}, {1-x, y+step}, {1-(x+ls), y+step}, {1-(x+step), y}};

publicBackFacePoints = backFacePoints;

publicBackFaceTexturePoints = backFaceTexturePoints;

float[][] **leftFacePoints** = {{x, y, z}, {x, y, z+step}, {x, y+step, z+step}, {x, y+step, z+ls}};

float[][] **leftFaceTexturePoints** = null;

publicLeftFaceTexturePoints = leftFaceTexturePoints;

publicLeftFacePoints = leftFacePoints;

float[][] **rightFacePoints** = {{x+step, y, z}, {x+ls, y+step, z+ls}, {x+ls, y+step, z+step}, {x+step, y, z+step}};

float[][] **rightFaceTexturePoints** = {{1-z, y}, {1-(z+ls), y+step}, {1-(z+step), y+step}, {1-(z+step), y}};

publicRightFacePoints = rightFacePoints;

publicRightFaceTexturePoints = rightFaceTexturePoints;

float[][] **topFacePoints** = {{x, y+step, z+ls}, {x, y+step, z+step}, {x+ls, y+step, z+step}, {x+ls, y+step, z+ls}};

float[][] **topFaceTexturePoints** = null;

publicTopFacePoints = topFacePoints;

publicTopFaceTexturePoints = topFaceTexturePoints;

float[][] **bottomFacePoints** = {{x, y, z}, {x+step, y, z}, {x+step, y, z+step}, {x, y, z+step}};

publicBottomFacePoints = bottomFacePoints;

if(y==0){

float[][] **bottomFaceTexturePoints** = {{x, z}, {x+step, z}, {x+step, z+step}, {x, z+step}};

publicBottomFaceTexturePoints = bottomFaceTexturePoints;

}else{

float[][] **bottomFaceTexturePoints** = null;

publicBottomFaceTexturePoints = bottomFaceTexturePoints;

}}else{

float[][] **frontFacePoints** = {{x, y, z+step}, {x+step, y, z+step}, {x+step, y+step, z+step}, {x, y+step, z+step}};

float[][] **frontFaceTexturePoints** = null;

publicFrontFacePoints = frontFacePoints;

publicFrontFaceTexturePoints = frontFaceTexturePoints;

float[][] **backFacePoints** = {{x, y, z}, {x, y+step, z+ls}, {x+step, y+step, z+ls}, {x+step, y, z}};

float[][] **backFaceTexturePoints** = {{1-x, y}, {1-x, y+step}, {1-(x+step), y+step}, {1-(x+step), y}};

publicBackFacePoints = backFacePoints;

publicBackFaceTexturePoints = backFaceTexturePoints;

float[][] **leftFacePoints** = {{x, y, z}, {x, y, z+step}, {x, y+step, z+step}, {x, y+step, z+ls}};

float[][] **leftFaceTexturePoints** = null;

```
        publicLeftFaceTexturePoints = leftFaceTexturePoints;

        publicLeftFacePoints = leftFacePoints;


        float[][] rightFacePoints = {{x+step, y, z}, {x+step, y+step, z+ls}, {x+step,
        y+step, z+step}, {x+step, y, z+step}, {x+step, y, z+step}};

        float[][] rightFaceTexturePoints = null;

        publicRightFacePoints = rightFacePoints;

        publicRightFaceTexturePoints = rightFaceTexturePoints;


        float[][] topFacePoints = {{x, y+step, z+ls}, {x, y+step, z+step}, {x+step,
        y+step, z+step}, {x+step, y+step, z+ls}};

        float[][] topFaceTexturePoints = null;

        publicTopFacePoints = topFacePoints;

        publicTopFaceTexturePoints = topFaceTexturePoints;


        float[][] bottomFacePoints = {{x, y, z}, {x+step, y, z}, {x+step, y, z+step}, {x, y,
z+step}};

        publicBottomFacePoints = bottomFacePoints;

        if(y==0){

                float[][] bottomFaceTexturePoints = {{x, z}, {x+step, z}, {x+step,
                z+step}, {x, z+step}};

                publicBottomFaceTexturePoints = bottomFaceTexturePoints;

        }else{

                float[][] bottomFaceTexturePoints = null;

                publicBottomFaceTexturePoints = bottomFaceTexturePoints;

                }

}}else
```

if(x == offset){//**ARISTERO KOMMATI, MESH OSON AFORA TON Z, PLAGIO MONO ARISTERA**

float[][] **frontFacePoints** = {{x, y, z+step}, {x+step, y, z+step}, {x+step, y+step, z+step}, {x+ls, y+step, z+step}};

float[][] **frontFaceTexturePoints** = null;

publicFrontFacePoints = frontFacePoints;

publicFrontFaceTexturePoints = frontFaceTexturePoints;

float[][] **backFacePoints** = {{x, y, z}, {x+ls, y+step, z}, {x+step, y+step, z}, {x+step, y, z}};

float[][] **backFaceTexturePoints** = null;

publicBackFacePoints = backFacePoints;

publicBackFaceTexturePoints = backFaceTexturePoints;

float[][] **leftFacePoints** = {{x, y, z}, {x, y, z+step}, {x+ls, y+step, z+step}, {x+ls, y+step, z}};

float[][] **leftFaceTexturePoints** = {{z, y}, {z+step, y}, {z+step, y+step}, {z, y+step}};

publicLeftFaceTexturePoints = leftFaceTexturePoints;

publicLeftFacePoints = leftFacePoints;

float[][] **rightFacePoints** = {{x+step, y, z}, {x+step, y+step, z}, {x+step, y+step, z+step}, {x+step, y, z+step}};

float[][] **rightFaceTexturePoints** = null;

publicRightFacePoints = rightFacePoints;

publicRightFaceTexturePoints = rightFaceTexturePoints;

float[][] **topFacePoints** = {{x+ls, y+step, z}, {x+ls, y+step, z+step}, {x+step, y+step, z+step}, {x+step, y+step, z}};

float[][] **topFaceTexturePoints** = null;

publicTopFacePoints = topFacePoints;

publicTopFaceTexturePoints = topFaceTexturePoints;

float[][] **bottomFacePoints** = {{x, y, z}, {x+step, y, z}, {x+step, y, z+step}, {x, y, z+step}};

publicBottomFacePoints = bottomFacePoints;

if(y==0){

float[][] **bottomFaceTexturePoints** = {{x, z}, {x+step, z}, {x+step, z+step}, {x, z+step}};

publicBottomFaceTexturePoints = bottomFaceTexturePoints;

}else{

float[][] **bottomFaceTexturePoints** = null;

publicBottomFaceTexturePoints = bottomFaceTexturePoints;

}}else

if(x == endCoord){

float[][] **frontFacePoints** = {{x, y, z+step}, {x+step, y, z+step}, {x+ls, y+step, z+step}, {x, y+step, z+step}};

float[][] **frontFaceTexturePoints** = null;

publicFrontFacePoints = frontFacePoints;

publicFrontFaceTexturePoints = frontFaceTexturePoints;

float[][] **backFacePoints** = {{x, y, z}, {x, y+step, z}, {x+ls, y+step, z}, {x+step, y, z}};

float[][] **backFaceTexturePoints** = null;

publicBackFacePoints = backFacePoints;

publicBackFaceTexturePoints = backFaceTexturePoints;

float[][] **leftFacePoints** = {{x, y, z}, {x, y, z+step}, {x, y+step, z+step}, {x, y+step, z}};

float[][] **leftFaceTexturePoints** = null;

publicLeftFaceTexturePoints = leftFaceTexturePoints;

publicLeftFacePoints = leftFacePoints;

float[][] **rightFacePoints** = {{x+ls, y+step, z}, {x+ls, y+step, z+step}, {x+step, y, z+step}, {x+step, y, z}};

float[][] **rightFaceTexturePoints** = {{1-z, y+step}, {1-(z+step), y+step}, {1-(z+step), y}, {1-z, y}};

publicRightFacePoints = rightFacePoints;

publicRightFaceTexturePoints = rightFaceTexturePoints;

float[][] **topFacePoints** = {{x, y+step, z}, {x, y+step, z+step}, {x+ls, y+step, z+step}, {x+ls, y+step, z}};

float[][] **topFaceTexturePoints** = null;

publicTopFacePoints = topFacePoints;

publicTopFaceTexturePoints = topFaceTexturePoints;

float[][] **bottomFacePoints** = {{x, y, z}, {x+step, y, z}, {x+step, y, z+step}, {x, y, z+step}};

publicBottomFacePoints = bottomFacePoints;

if(y==0){

float[][] **bottomFaceTexturePoints** = {{x, z}, {x+step, z}, {x+step, z+step}, {x, z+step}};

```
        publicBottomFaceTexturePoints = bottomFaceTexturePoints;
}else{
        float[][] bottomFaceTexturePoints = null;
        publicBottomFaceTexturePoints = bottomFaceTexturePoints;
        }}else
if(y == 0){
        float[][] frontFacePoints = {{x, y, z+step}, {x+step, y, z+step}, {x+step, y+step,
        z+step}, {x, y+step, z+step}};
        float[][] frontFaceTexturePoints = null;
        publicFrontFacePoints = frontFacePoints;
        publicFrontFaceTexturePoints = frontFaceTexturePoints;

        float[][] backFacePoints = {{x, y, z}, {x, y+step, z}, {x+step, y+step, z},
{x+step, y, z+step}};
        float[][] backFaceTexturePoints = null;
        publicBackFacePoints = backFacePoints;
        publicBackFaceTexturePoints = backFaceTexturePoints;

        float[][] leftFacePoints = {{x, y, z}, {x, y, z+step}, {x, y+step, z+step}, {x,
y+step, z}};
        float[][] leftFaceTexturePoints = null;
        publicLeftFaceTexturePoints = leftFaceTexturePoints;
        publicLeftFacePoints = leftFacePoints;

        float[][] rightFacePoints = {{x+step, y, z}, {x+step, y+step, z}, {x+step, y+step,
        z+step}, {x+step, y, z+step}};
        float[][] rightFaceTexturePoints = null;
```

```
        publicRightFacePoints = rightFacePoints;

        publicRightFaceTexturePoints = rightFaceTexturePoints;


        float[][] topFacePoints = {{x, y+step, z}, {x, y+step, z+step}, {x+step, y+step,
        z+step}, {x+step, y+step, z}};

        float[][] topFaceTexturePoints = null;

        publicTopFacePoints = topFacePoints;

        publicTopFaceTexturePoints = topFaceTexturePoints;


        float[][] bottomFacePoints = {{x, y, z}, {x+step, y, z}, {x+step, y, z+step}, {x, y,
z+step}};

        publicBottomFacePoints = bottomFacePoints;

if(y==0){

        float[][] bottomFaceTexturePoints = {{x, z}, {x+step, z}, {x+step, z+step}, {x,
z+step}};

        publicBottomFaceTexturePoints = bottomFaceTexturePoints;

}else{

        float[][] bottomFaceTexturePoints = null;

        publicBottomFaceTexturePoints = bottomFaceTexturePoints;

        }

}}

public float[][] publicLeftFacePoints, publicRightFacePoints,
publicBottomFacePoints,    publicTopFacePoints, publicBackFacePoints,
publicFrontFacePoints,                    publicLeftFaceTexturePoints,
publicRightFaceTexturePoints,
publicFrontFaceTexturePoints, publicBackFaceTexturePoints,
         publicTopFaceTexturePoints, publicBottomFaceTexturePoints;

public int arreyLength = 4;

private int step;
```

```
}
```

3.5 Η Κλάση CylinderFacesPoints

```
import static java.lang.Math.*;
import java.util.*;
public class CylinderFacesPoints {
```

```
public CylinderFacesPoints(float startDeg, int partsPerSide){
        float fullStep = (float) 360/partsPerSide;
        int intStep = (int) floor(fullStep);
        float remainStep = (float)  fullStep - intStep;
        float startRad = (float) toRadians(startDeg);
        float endRad = (float) toRadians(startDeg+fullStep);
        //System.out.println("Cube gets:
"+initialPoint.x+","+initialPoint.y+","+initialPoint.z);
        float plus = (float)1;
        //System.out.println("Int part: "+intStep+", Remaining: "+remainStep);
        Vector topFacesPoints = new Vector(intStep);
        Vector topFacesTexturePoints = new Vector(intStep);
        Vector roundFacesPoints = new Vector(intStep);
        Vector roundFacesTexturePoints = new Vector(intStep);
        Vector bottomFacesPoints = new Vector(intStep);
        Vector bottomFacesTexturePoints = new Vector(intStep);

for(int i=0;i<intStep;i++){
        float currentX = (float) cos( (float) toRadians(startDeg+i) );
        float currentZ = (float) sin( (float) toRadians(startDeg+i) );
        float nextX = (float) cos( (float) toRadians(startDeg+i+1) );
        float nextZ = (float) sin( (float) toRadians(startDeg+i+1) );
        float extraX = (float) cos( (float) toRadians(startDeg+i+1+remainStep) );
        float extraZ = (float) sin( (float) toRadians(startDeg+i+1+remainStep) );
        float[][] roundFacePart = {{currentX, -1, currentZ}, {currentX, +1, currentZ},
        {nextX, +1, nextZ}, {nextX, -1, nextZ}};
```

```
        float[][] roundFaceTexturePart = {{1-((startDeg+i)/360), 0}, {1-
        ((startDeg+i)/360), 1}, {1-((startDeg+i+1)/360), 1}, {1-((startDeg+i+1)/360),
        0}};

        roundFacesTexturePoints.addElement(roundFaceTexturePart);

        roundFacesPoints.addElement(roundFacePart);

        float[][] topFacePart = {{0, +1, 0}, {nextX, +1, nextZ}, {currentX, +1,
currentZ}};

        float[][] topFaceTexturePart = {{plus/2, 1-plus/2}, {(nextX+plus)/2, 1-
        (nextZ+plus)/2}, {(currentX+plus)/2, 1-(currentZ+plus)/2}};

        //System.out.println(topFaceTexturePart[1][0]);

        topFacesPoints.addElement(topFacePart);

        topFacesTexturePoints.addElement(topFaceTexturePart);



        float[][] bottomFacePart = {{0, -1, 0}, {currentX, -1, currentZ}, {nextX, -1,
nextZ}};

        float[][] bottomFaceTexturePart = {{plus/2, plus/2}, {(currentX+plus)/2,
        (currentZ+plus)/2}, {(nextX+plus)/2, (nextZ+plus)/2}};

        bottomFacesPoints.addElement(bottomFacePart);

        bottomFacesTexturePoints.addElement(bottomFaceTexturePart);



if(i==intStep-1){

        float[][] remainingFacePart = {{nextX, -1, nextZ}, {nextX, +1, nextZ}, {extraX,
        +1, extraZ}, {extraX, -1, extraZ}};

        float[][] remainingFaceTexturePart = {{1-((startDeg+i+1)/360), 0}, {1-
        ((startDeg+i+1)/360), 1}, {1-((startDeg+i+1+remainStep)/360), 1}, {1-
        ((startDeg+i+1+remainStep)/360), 0}};

        roundFacesPoints.addElement(remainingFacePart);

        roundFacesTexturePoints.addElement(roundFaceTexturePart);
```

```
        float[][] remainingTopFacePart = {{0, +1, 0}, {extraX, +1, extraZ}, {nextX, +1,
nextZ}};

        float[][] remainingTopFaceTexturePart = {{plus/2, 1-plus/2},
        {(extraX+plus)/2, 1-(extraZ+plus)/2}, {(nextX+plus)/2, 1-(nextZ+plus)/2}};

        topFacesPoints.addElement(remainingTopFacePart);


        topFacesTexturePoints.addElement(remainingTopFaceTexturePart);



        float[][] remainingBottomFacePart = {{0, -1, 0}, {nextX, -1, nextZ}, {extraX, -1,
extraZ}};

        float[][] remainingBottomFaceTexturePart = {{plus/2, plus/2},
        {(nextX+plus)/2, (nextZ+plus)/2}, {(extraX+plus)/2, (extraZ+plus)/2}};

        bottomFacesPoints.addElement(remainingBottomFacePart);

        bottomFacesTexturePoints.addElement(remainingBottomFaceTexturePart)
;

        }

}

float[][] endSideFace = {{0, -1, 0}, {(float) cos(endRad), -1, (float) sin(endRad)},
                {(float) cos(endRad), +1, (float) sin(endRad)}, {0, +1, 0}};

float[][] startSideFace = {{0, -1, 0}, {0, +1, 0}, {(float) cos(startRad), +1, (float)
sin(startRad)},          {(float) cos(startRad), -1, (float) sin(startRad)}};


publicRoundFacePoints = roundFacesPoints;

publicRoundFaceTexturePoints = roundFacesTexturePoints;

publicEndSideFace = endSideFace;

publicStartSideFace = startSideFace;

publicTopFacePoints = topFacesPoints;

publicTopFaceTexturePoints = topFacesTexturePoints;

publicBottomFacePoints = bottomFacesPoints;
```

```
publicBottomFaceTexturePoints = bottomFacesTexturePoints;

}

public Vector publicRoundFacePoints, publicRoundFaceTexturePoints,
publicTopFacePoints, publicTopFaceTexturePoints, publicBottomFacePoints,
publicBottomFaceTexturePoints;

public float[][] publicEndSideFace, publicStartSideFace;

}
```

3.6 Η Κλάση Point

```
public class Point {
      public Point(float a, float b, float c) {
            x = a;
            y = b;
            z = c;
            //System.out.println("Point has:"+x+","+y+","+z);
    }
public float x;
 public float y;
public float z;
}
```

3.7 Το πρωτότυπο movementProto2.x3d

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN"
"http://www.web3d.org/specifications/x3d-3.0.dtd">

<X3D profile='Immersive' >

<head>

 <meta name='Vizthumbnail'
content='Thumb_SpaceSensor_x3d1350461195829340.jpg'/>

 <meta name='ExportTime' content='16:49: 0'/>

 <meta name='ExportDate' content='11/23/2007'/>

</head>

<Scene>

<WorldInfo

 title='Untitled'

 info="'This Web3D World was created with Flux Studio, a Web3D authoring tool"

 "www.mediamachines.com"'/>

<ProtoDeclare

 name='movementProto'>


 <ProtoInterface>

  <field name='movingChildren' accessType='inputOnly' type='MFNode'/>

  <field name='xyPlaneEnable' accessType='inputOutput' type='SFBool'/>

  <field name='xzPlaneEnable' accessType='inputOutput' type='SFBool'/>
```

```
 <field name='sphereEnable' accessType='inputOutput' type='SFBool'/>
 <field name='xzRotation' accessType='inputOutput' type='SFRotation'/>
 <field name='xyRotation' accessType='inputOutput' type='SFRotation'/>
 <field name='sensorRotation' accessType='inputOutput' type='SFRotation'/>
 <field name='slidingRotation' accessType='inputOutput' type='SFRotation'/>
 <field name='shapeTranslation' accessType='inputOutput' type='SFVec3f'/>
</ProtoInterface>

<ProtoBody>

<Transform DEF='xyTransform' rotation='1 0 0 0'>
        <connect nodeField='rotation' protoField='xyRotation'/>
        <PlaneSensor DEF='xyPlaneSensor' enabled='true'>
                <connect nodeField='enabled' protoField='xyPlaneEnable'/>
        </PlaneSensor>
                <Transform DEF='xzTransform' rotation='1 0 0 -1.57'>
                        <connect nodeField='rotation' protoField='xzRotation'/>
                        <PlaneSensor DEF='xzPlaneSensor' enabled='false'>
                                <connect nodeField='enabled'
protoField='xzPlaneEnable'/>
                        </PlaneSensor>
                <Transform DEF='slidingTransform' rotation='1 0 0 1.57'>
                        <connect nodeField='rotation' protoField='slidingRotation'/>
                        <SphereSensor DEF='SphereSensor' enabled='false'>
                                <connect nodeField='enabled'
protoField='sphereEnable'/>
```

```
                    </SphereSensor>

                    <Transform DEF='shapeTransform' rotation='0 1 0 0'>

                    <connect nodeField='translation'
protoField='shapeTranslation'/>

                    <connect nodeField='addChildren'
protoField='movingChildren'/>

                    </Transform>

                </Transform>

        </Transform>

</Transform>


<ROUTE fromNode='xyPlaneSensor' fromField='translation_changed'
toNode='xzTransform' toField='set_translation'/>

<ROUTE fromNode='xzPlaneSensor' fromField='translation_changed'
toNode='slidingTransform' toField='set_translation'/>

<ROUTE fromNode='SphereSensor' fromField='rotation_changed'
toNode='slidingTransform' toField='set_rotation'/>


                    </ProtoBody>

                </ProtoDeclare>

        </Scene>

</X3D>
```