ΑΝΩΤΑΤΟ ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΚΡΗΤΗΣ

Σχολή Τεχνολογικών Εφαρμογών
Τμήμα Εφαρμοσμένης Πληροφορικής και Πολυμέσων

Πτυχιακή Εργασία

«Ανάπτυξη διαδραστικής εφαρμογής: “αόρατα κρουστά” με την χρήση της συσκευής Kinect και της γλώσσας προγραμματισμού Processing»

Κόντος Χρήστος
Α.Μ. 1046

Επιβλέπων καθηγητής : Δρ Βιδάκης Νικόλαος

Επιτροπή Αξιολόγησης :

Ημερομηνία παρουσίασης:

Ηράκλειο Απρίλιος 2014

Περίληψη

Σε αυτήν την πτυχιακή εργασία αναπτύξαμε εφαρμογές που εκμεταλλεύονται την τεχνολογία ανίχνευσης χώρου και κίνησης του Kinect με αποτέλεσμα να δίνεται στον χρήστη η δυνατότητα να αντλεί δεδομένα από το περιβάλλον γύρω του καθώς και από τις κινήσεις του, αλληλεπιδρώντας έτσι με το πρόγραμμα χωρίς να χρειάζονται άλλες συσκευές εισόδου. Έτσι ο χρήστης ενεργοποιώντας για παράδειγμα την κεντρική εφαρμογή της πτυχιακής θα μπορεί να παίξει κρουστά αγγίζοντας με τα χέρια και τα πόδια του συγκεκριμένες ομάδες σημείων στο αέρα γύρω του.

Η άντληση των δεδομένων από τη συσκευή Kinect, η ερμηνεία, η επεξεργασία, η προβολή τους όπως και ο προσδιορισμός της διαδραστικότητας είναι βασικές διαδικασίες που πραγματοποιήθηκαν προγραμματίζοντας με την γλώσσα Processing και τις βιβλιοθήκες OpenNI και SimpleOpenNI. Συγκεκριμένα πήραμε ως είσοδο στο πρόγραμμά μας την σάρωση του πραγματικού χώρου τον οποίο "βλέπει" η υπέρυθρη κάμερα της Kinect και με την κατάλληλη ερμηνεία δημιουργήσαμε ένα μοντέλο αναπαράστασης του χώρου στην οθόνη ικανό να ανιχνεύει την θέση και την κίνηση αντικειμένων και χρηστών στις τρεις διαστάσεις με ακρίβεια εκατοστών. Επιπλέον με την βοήθεια τον εντολών της Processing, κατασκευάσαμε τεχνικές και μεθόδους που επιτρέπουν εικονική πλοήγηση μέσα στο μοντέλο μας. Τέλος εμπλουτίσαμε το μοντέλο αυτό με ομάδες σημείων οι οποίες τροφοδοτούν το πρόγραμμα σε περίπτωση που κάποιο πραγματικό αντικείμενο περάσει μέσα στην οριοθετημένη περιοχή τους. Πάνω σε αυτό το τρισδιάστατο σύστημα έγιναν μαθηματικές πράξεις, δοκιμές και διορθώσεις προκειμένου να βαθμονομηθούν σωστά οι τιμές των μεταβλητών και οι θέσεις των αντικειμένων του προγράμματος αλλά και να ελαχιστοποιήσουμε την καθυστέρηση που δημιουργείται λόγω της αυξημένης υπολογιστικής ισχύς που απαιτούν οι συνεχόμενες καταγραφές και τρισδιάστατες αναπαραστάσεις του χώρου.

Σκοπός της πτυχιακής αυτής ήταν η μελέτη της πρωτότυπης διαδικασίας ανίχνευσης-καταγραφής χώρου, προσώπων και των αντικειμένων που παρέχει η συσκευή Kinect και η εξοικείωση με την ανάπτυξη εφαρμογών Processing που χρησιμοποιούν τις δυνατότητες αυτές και προσφέρουν στους χρήστες νέους τρόπους αλληλεπίδρασης.

Λέξεις – κλειδιά: Kinect για windows, αισθητήρας, υπέρυθρη κάμερα, αντικειμενοστραφής προγραμματισμός, Processing, open-source, SimpleNI, SimpleOpenNI, Minim.

Abstract

In this dissertation we have developed applications that exploit the space and motion detection technology of the Kinect device. By giving the user the ability to pull data from the environment around him and from his own moves and gestures he can interact with the program without the need of other input devices. So the user for example by activating the basic application of the dissertation he can play percussion by tapping his hands and feet at specific point-groups in the air around him.
The extraction of the data from the Kinect device, their interpretation, editing and displaying are essential procedures that were performed with the programming language Processing and the libraries OpenNI and SimpleOpenNI. We took as input to our program the scan of real space which the infrared camera of Kinect captured and with the proper interpretation we created a model representation of space that can detect the position and movement of objects and users in three dimensions with the accuracy of a centimeter. Moreover, with the help of Processing commands, we constructed techniques and methods that allow virtual navigation in our model. Finally we have enriched the model with point groups that give a feedback to the program whenever a real object pass within the demarcated area. Over this three-dimensional system we made math tests and corrections to properly calibrate the variable's values and the positions of the objects in the program and to minimize the delay created by the increased computational power that the consecutive recordings and 3-dimensional representations of space required.
The aim of this dissertation was to study this scanning process of space, persons and objects provided by the Kinect device and to increase the familiarity with the development of Processing applications that offer users new ways of interacting.

Keywords: Kinect for windows, sensor, depth camera, Processing, open-source, SimpleNI, SimpleOpenNI, Minim.

Πίνακας Περιεχομένων

Ενδεικτικοί Πίνακες ή Κώδικας

ΠΙΝΑΚΑΣ 1: ΣΥΓΚΡΙΤΙΚΌΣ ΠΊΝΑΚΑΣ ΣΥΣΤΗΜΆΤΩΝ20 LINK

Ενδεικτικές Εικόνες

1. Εισαγωγή

Στο κεφάλαιο αυτό παρουσιάζουμε μια γενική περιγραφή του αντικειμένου της πτυχιακής αυτής και την οργάνωση των κεφαλαίων του υπόλοιπου τόμου.

1.1 Αντικείμενο της πτυχιακής

Στην παρούσα πτυχιακή ασχοληθήκαμε με την ανάπτυξη διαδραστικών εφαρμογών χρησιμοποιώντας την συσκευή Kinect σε συνδυασμό με την γλώσσα προγραμματισμού Processing. Ειδικότερα αναπτύξαμε εφαρμογές που εκμεταλλεύονται την τεχνολογία ανίχνευσης χώρου και κίνησης του Kinect με αποτέλεσμα να δίνεται στον χρήστη η δυνατότητα να αντλεί δεδομένα από το περιβάλλον γύρω του καθώς και από τις κινήσεις του, αλληλεπιδρώντας έτσι με το πρόγραμμα χωρίς να χρειάζονται άλλες συσκευές εισόδου.

Η άντληση των δεδομένων από τη συσκευή Kinect, η ερμηνεία, η επεξεργασία, η προβολή τους όπως και ο προσδιορισμός της διαδραστικότητας είναι βασικές διαδικασίες που πραγματοποιήθηκαν προγραμματίζοντας με την γλώσσα Processing και τις βιβλιοθήκες της OpenNI και SimpleOpenNI. Συγκεκριμένα πήραμε ως είσοδο στο πρόγραμμά μας την σάρωση του πραγματικού χώρου τον οποίο "βλέπει" η υπέρυθρη κάμερα της Kinect και με την κατάλληλη ερμηνεία δημιουργήσαμε ένα μοντέλο αναπαράστασης του χώρου στην οθόνη ικανό να ανιχνεύει την θέση και την κίνηση αντικειμένων και χρηστών στις τρεις διαστάσεις με ακρίβεια εκατοστών. Επιπλέον με την βοήθεια τον εντολών της Processing, κατασκευάσαμε τεχνικές και μεθόδους που επιτρέπουν εικονική πλοήγηση μέσα στο μοντέλο μας αλλά και αποθήκευση πληροφοριών από το πρόγραμμα με την χρήση του πληκτρολογίου. Τέλος εμπλουτίσαμε το μοντέλο αυτό με ομάδες σημείων οι οποίες τροφοδοτούν το πρόγραμμα σε περίπτωση που κάποιο πραγματικό αντικείμενο περάσει μέσα στην οριοθετημένη περιοχή τους. Πάνω σε αυτό το τρισδιάστατο σύστημα έγιναν μαθηματικές πράξεις, δοκιμές και διορθώσεις προκειμένου να βαθμονομηθούν σωστά οι τιμές των μεταβλητών και οι θέσεις των αντικειμένων του προγράμματος αλλά και να ελαχιστοποιήσουμε την καθυστέρηση που δημιουργείται λόγω της αυξημένης υπολογιστικής ισχύς που απαιτούν οι συνεχόμενες καταγραφές και τρισδιάστατες αναπαραστάσεις του χώρου.

Αντικείμενο της πτυχιακής ήταν η μελέτη της διαδικασίας ανίχνευσης-καταγραφής χώρου, προσώπων και των αντικειμένων που παρέχει η συσκευή Kinect και η εξοικείωση με την ανάπτυξη εφαρμογών Processing που χρησιμοποιούν τις δυνατότητες αυτές.

1.2 Οργάνωση του τόμου

Η συνέχεια της πτυχιακής έχει οργανωθεί στα ακόλουθα κεφάλαια:

- Στο Κεφάλαιο 2 παρουσιάζονται χρήσιμες έννοιες και ορισμοί, σχετικά με ένα Σύστημα Διαχείρισης Δεδομένων
- Στο Κεφάλαιο 3, παρουσιάζεται η τεχνολογία που χρησιμοποιήθηκε για την ανάπτυξη του συστήματος
- Στο Κεφάλαιο 4, παρουσιάζονται γενικές αλλά και ειδικές πληροφορίες για την ανάλυση, σχεδίαση και υλοποίηση του συστήματος
- Στο Κεφάλαιο 5, παρουσιάζονται τα συμπεράσματα της έρευνας και της υλοποίησης
- Στο Κεφάλαιο 6, παρουσιάζονται τα παραρτήματα στα οποία αναφερόμαστε σε

διάφορα σημεία της πτυχιακής

- Στο Κεφάλαιο 7, παρουσιάζεται όλη η σχετική βιβλιογραφία στην οποία βασιστήκαμε για τη μελέτη και εγγραφή της πτυχιακής.

2. Χρήσιμοι ορισμοί

Προτού αναλύσουμε τις τεχνολογίες που χρησιμοποιήθηκαν και τις εφαρμογές που αναπτύχθηκαν, θα ήταν χρήσιμο να αναφέρουμε και να εξετάσουμε μερικούς σχετικούς ορισμούς ώστε να γίνουν πιο κατανοητά τα επόμενα κεφάλαια.

2.1 Τι είναι το Kinect

Kinect ή γνωστό και ως Project Natal είναι μία σειρά συσκευών εισόδου από την Microsoft για τις παχνιδομηχανές Xbox360 και XboxOne αλλά και για Windows Pc. Η συσκευή Kinect καταγράφοντας εικόνα, ήχο αλλά και το βάθος του χώρου, επιτρέπει στους χρήστες να ελέγχουν και αλληλεπιδρούν με την παχνιδομηχανή ή τον υπολογιστή χωρίς την ανάγκη χειριστηρίου/πληκτρολογίου, μέσω μιας φυσικής διεπαφής κάνοντας κινήσεις, χειρονομίες και λέγοντας εντολές. Η πρώτη γενιά Kinect ήταν το Νοέμβριο του 2010 σε μία προσπάθεια να μεγαλώσει το κοινό των χρηστών του Xbox πέρα από την τυπική του βάση. Αργότερα τον Φεβρουάριο του 2012 μία έκδοση για τα Windows βγήκε στην αγορά. Η συσκευή είναι αυτή που χρησιμοποιήσαμε στην πτυχιακή και λέγεται Kinect for Windows. Στην συνέχεια τον Ιούνιο του 2011 η microsoft παρουσίασε το πακέτο ανάπτυξης λογισμικού (SDK) για τα Window 7. Αυτό το πακέτο επέτρεψε στους προγραμματιστές να αναπτύξουν Kinect εφαρμογές σε C++/CLI, C#, ή Visual Basic .NET .

Εικ 1: Το σύστημα ή η συσκευή Kinect

<u>Επισκόπηση</u>

Η συσκευή ή το σύστημα Kinect ανακοινώθηκε για πρώτη φορά τον Ιούνιο του 2009 στο Electronic Entertainment Expo με το κωδικό όνομα "Project Natal". Ακολουθώντας την παράδοσή της να δίνει ονόματα πόλεων η Microsoft έδωσε αυτόν τον τίτλο σαν φόρο τιμής προς τον γεννημένο στην Βραζιλία διευθυντή της Alex Kipman ο οποίος ξεκίνησε αυτό το project. Το όνομα Natal επιλέχτηκε επίσης επειδή σημαίνει "γενέθλιος", αντικατοπτρίζοντας έτσι την άποψη της Microsoft ότι το project σηματοδοτεί "την έναρξη νέας γενιάς συσκευών αναψυχής για το σπίτι".
Παρουσιάστηκαν τρία demos για το Kinect όταν ανακοινώθηκε: Ricochet, Paint Party και Milo & Kate. Η τεχνολογία ανίχνευσης-καταγραφής σκελετού που παρουσιάστηκε έδινε την δυνατότητα ταυτόχρονης καταγραφής τεσσάρων ατόμων, με το χαρακτηριστικό των 48 σκελετικών σημείων για ένα ανθρώπινο σώμα στα 30Hz (ή 30 frames/sec).
Φήμες έλεγαν ότι με την παρουσίαση του Kinect, θα κυκλοφορούσε νέα σειρά κονσολών Xbox 360 για λόγους συμβατότητας. Η Microsoft το διέψευσε δημόσια ανακοινώνοντας ότι το Project Natal θα ήταν συμβατό με όλες της Xbox κονσόλες. Η Microsoft έδειξε ότι θεωρούσε το project πολύ σημαντικό ξεκίνημα, τόσο βασικό όσο ήταν το Xbox Live ή το ξεκίνημα παραγωγής νέας πλατφόρμας κονσόλας. Μάλιστα ο CEO Steve Ballmer ανάφερε το Kinect ως ένα «νέο Xbox» σε μία ομιλία του στο Chicago. Όταν η Microsoft ρωτήθηκε αν η εισαγωγή του Natal θα είχε ως αποτέλεσμα την αύξηση του χρόνου μέχρι την επόμενη κυκλοφορία της νέας γενιάς κονσόλας, ο Shane Kim βεβαίωσε ότι ο κύκλος της ζωής του Xbox 360 θα διαρκέσει μέχρι και το 2015.
Κατά διάρκεια ανάπτυξης λογισμικού για το Kinect, τα μέλη της ομάδας του project πειραματίστηκαν με ένα μεγάλο αριθμό παιχνιδιών που είχα μετατραπεί ώστε να ελέγχονται από το

Kinect, με σκοπό την δοκιμή της λειτουργικότητάς του. Ανάμεσα σε αυτά τα παιχνίδια ήταν το Beautiful Katamari και το Space Invaders Extreme, τα οποία παρουσιάστηκαν τον Σεπτέμβριο του 2009 στο Tokyo Game Show. Σύμφωνα με τον Διευθυντή Παραγωγής Kudo Tsunoda, η διαδικασία δημιουργίας πακέτου για έλεχγο μέσω Kinect σε ήδη κυκλοφωρημένα παιχνίδια απαιτεί σημαντικές αλλαγές στον κώδικα, με αποτέλεσμα να είναι σχεδόν αδύνατο να επιτευχθεί μέσω αναβαθμίσεων λογισμικού.
Παρόλο που το Kinect στα αρχικά πλάνα είχε σχεδιαστεί να περιέχει ένα μικροεπεξεργαστή που θα του επέτρεπε να εκτελεί διεργασίες όπως την ανίχνευση-καταγραφή σκελετού, τον Ιανουάριο του 2010 αποκαλύφθηκε ότι ο δεν θα περιέχει δικό του επεξεργαστή. Αντιθέτως η επεξεργασία θα καλυπτόταν από ένα πυρήνα του επεξεργαστή Xenon του Xbox 360. Σύμφωνα με τον Alex Kipman, το σύστημα του Kinect καταναλώνει περίπου 10-15% από τους υπολογιστικούς πόρους του Xbox 360. Όμως ο ίδιος στη συνέχεια το Νοέμβριο δήλωσε ότι το νέο μοντέλο απαιτούσε πλέον κάτω από 10%.
Τον Ιούνιο στις 13, η Microsoft ανακοίνωσε ότι το σύστημα θα ονομαζόταν επίσημα Kinect από της λέξεις kinetic (κινητικός) και connect (συνδέω), χαρακτηριστικές της φύσεως του project. Επίσης ανακοίνωσε ότι στην Βόρειο Αμερική θα έβγαινε στην κυκλοφορία τον Νοέμβριο του 2010 όπως και ότι θα κυκλοφορούσε νέα έκδοση Xbox 360 με σχεδιασμένη για Kinect θύρα σύνδεσης .
Τον Ιούνιο του 2011, η Microsoft δημοσίευσε το πακέτο ανάπτυξης λογισμικού SDK για μη-εμπορική χρήση. Ενώ τον Ιούλιο κυκλοφόρησε ένα ασπρο Kinect μαζί με το «Xbox 360 Limited Edition Kinect Star Wars Bundle». Μέχρι τότε όλα τα μοντέλα Kinect είχαν κυκλογορήσει σε μαύρο χρώμα.
Στα τέλη του Οκτωμβρίου το 2011, η Microsoft ανακοίνωσε την έναρξη της εμπορικής έκδοσης του προγράμματος Kinect for Windows και την προώθηση του πακέτου SDK σε εταιρείες. Ο David Dennis, διευθυντής τμήματος προϊόντος στην Microsoft, δήλωσε "Υπάρχουν πολλοί οργανισμοί που συνεργαζόμαστε για να βοηθήσουμε να προσδιοριστούν οι δυνατότητες αυτής της τεχνολογίας".
Τον Φεβρουάριο του 2012, η Microsoft ανακοίνωσε την κυκλοφορία της εμπορικής έκδοσης του Kinect for Windows SDK πακέτου και δήλωσε ότι πάνω από 300 εταιρίες σε 25 χώρες εργάζονται πάνω σε Kinect εφαρμογές.

Kinect for Windows

Στις 21 Φεβρουαρίου το 2011, η Microsoft ανακοίνωσε ότι θα κυκλοφορούσε μια μη-εμπορική έκδοση του πακέτου λογισμικού για Windows την ερχόμενη άνοιξη, η οποία προοριζόταν για 12 χώρες. Το πακέτο περιλάμβανε οδηγούς προγραμμάτων (drivers) Windows 7 για την συσκευή Kinect. Έδωσε στους προγραμματιστές την δυνατότητα να δημιουργήσουν και να αναπτύξουν εφαρμογές χρησιμοποιώντας C++, C# ή Visual Basic με το Microsoft Visual Studio 2010, οι οποίες εχουν τα χαρακτηριστικά:

1.Επεξεργασία χαμηλού επιπέδου δεδομένων που προέκυπταν από τον ανίχνευτη βάθους χώρου, κάμερα χρωμάτων και το μικρόφωνο.

2.Την δυνατότητα ανίχνευσης σκελετού ενός ή περισσότερων ατόμων οι οποίοι κινούνται ή κάνουν χειρονομίες μέσα στο οπτικό πεδίο του Kinect.

3. Προηγμένες δυνατότητες ηχητικής επεξεργασίας όπως καταστολή ακουστικού θορύβου και ηχώς, προσδιορισμός - ταυτοποίηση ηχητικής πηγής και ολοκλήρωση με την εφαρμογή Windows speech recognition API για φωνητική αναγνώριση.

4.Παροχή βοηθητικού κώδικα και τεκμηριωμένων πληροφοριών μέσω εγράφων (Documentation).

Τον Μάρτιο του 2012, ο Craig Eisler, γενικός διευθυντής για το Kinect for Windows δήλωσε ότι σχεδόν 350 εταιρείες συνεργάζονται με την Microsoft πάνω σε εφαρμογές του Kinect for Windows.

Έκδοση 1.5
Τον Μάρτιο του 2012, η Microsoft ανακοίνωσε ότι η επόμενη έκδοση του Kinect για Windows SDK είναι διαθέσιμη. Η έκδοση Kinect για Windows 1.5 κυκλοφόρησε στις 21 Μαΐου, 2012. Προσθέτει νέα χαρακτηριστικά, υποστήριξη για πολλές νέες γλώσσες και ντεμπούτο στη αγορά σε 19 ακόμη χώρες.
Το πακέτο λογισμικού Kinect για Windows SDK 1.5 περιλαμβάνει το Kinect Studio, μια νέα εφαρμογή που επιτρέπει στους προγραμματιστές για την καταγραφή, αναπαραγωγή και τον εντοπισμό σφαλμάτων κλιπ με χρήστες που αλληλεπιδρούν με Kinect εφαρμογές.
Ακόμα υποστήριζει νέο σύστημα ανίχνευσης σκελετού μέχρι 10 αρθρώσεων που επιτρέπει σε εφαρμογές να παρακολουθούν το κεφάλι, το λαιμό και τα χέρια του χρήστη, όταν κάθετε ή είναι όρθιος. Το οποίο μπορεί να λειτουργήσει στην κανονική αλλά και στην κοντινή λήψη.
Τέλος αυτή η έκδοση πρόσφερε υποστήριξη σε τέσσερις νέες γλώσσες για την αναγνώριση ομιλίας - Γαλλικά, Ισπανικά, Ιταλικά και Ιαπωνικά. Επιπλέον πρόσθεσε υποστήριξη για διαλέκτους από αυτές τις γλώσσες μαζί με τα Αγγλικά.
Θα γινόταν διαθέσιμο στο Χονγκ Κονγκ, τη Νότια Κορέα και την Ταϊβάν το Μάιο και στην Αυστρία, το Βέλγιο, τη Βραζιλία, τη Δανία, τη Φινλανδία, την Ινδία, την Ολλανδία, Νορβηγία, Πορτογαλία, Ρωσία, Σαουδική Αραβία, Σιγκαπούρη, Νότια Αφρική, Σουηδία, Ελβετία και Ηνωμένα Αραβικά Εμιράτα τον Ιούνιο.

Έκδοση 1.6, 1.7 και 1.8
Το πακέτο λογισμικού Kinect για Windows SDK για τον αισθητήρα πρώτης γενιάς ενημερώθηκε μερικές ακόμα φορές, με την έκδοση 1.6 που κυκλοφόρησε στις 8 Οκτωβρίου 2012, την έκδοση 1.7 που κυκλοφόρησε στις 18 του Μάρτη 2013 και την έκδοση 1.8 κυκλοφόρησε στις 17 Σεπτεμβρίου 2013.

Έκδοση 2
Η δεύτερη γενιά του Kinect για Windows είχε βάση την ίδια βασική τεχνολογία όπως το Kinect για το Xbox One συμπεριλαμβανομένου ενός νέου αισθητήρα. Η κυκλοφορία του είχε προγραμματιστεί για το καλοκαίρι του 2014 στο βόρειο ημισφαίριο. Παραδόθηκαν σχετικά πακέτα λογισμικού στους συμμετέχοντες που πληρούσαν τις προϋποθέσεις αρχίζοντας από τις 22 Νοεμβρίου, 2013.

2.2 Τι είναι η Processing

Εικ 2 : Το σήμα της Processing

Η Processing είναι μία αντικειμενοστραφής γλώσσα προγραμματισμού, ένα περιβάλλον ανάπτυξης εφαρμογών και μία online κοινότητα μαζί. Από το 2001, η Processing προώθησε την εκμάθηση ανάπτυξης λογισμικού μέσα στα πλαίσια των σχεδιαστικών τεχνών και νέων πολυμεσικών εφαρμογών. Αρχικά κατασκευάστηκε για να χρησιμοποιηθεί ως ένα ηλεκτρονικό βιβλίο ψηφιακής ιχνογραφίας και για να βοηθήσει στην διδασκαλία των προγραμματιστικών θεμελίων οπτικού (visual) περιβάλλοντος. Όμως στη συνέχεια εξελίχθηκε σε ένα ισχυρό προγραμματιστικό εργαλείο για επαγγελματίες.
Το project ξεκίνησε από τους Casey Reas και Benjamin Fry, οι οποίοι πιο πριν δραστηριοποιόντουσαν στο Aesthetics and Computation Group του τμήματος πολυμέσων του MIT. Ένας από τους αρχικούς σκοπούς της Processing που είχε τεθεί ήταν να λειτουργήσει ως εργαλείο που θα προέτρεπε άτομα που δεν ήταν προγραμματιστές να ξεκινήσουν τον προγραμματισμό, μέσω της άμεσης σχεδίασης γραφικών. Η γλώσσα έχει δομηθεί πάνω στην JAVA, αλλά χρησιμοποιεί απλοποιημένο συντακτικό και γραφικό

προγραμματιστικό περιβάλλον.

Χαρακτηριστικά

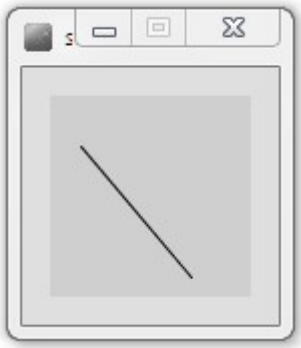
Η Processing συμπεριλαμβάνει ένα βιβλίο ψηφιακής ιχνογραφίας (sketchbook), το οποίο είναι στη ουσία ένα ελαχιστοποιημένο περιβάλλον για την οργάνωση των προγραμμάτων/εφαρμογών, μία απλοποιημένη μορφή της σουίτας λογισμικού τύπου IDE (integrated development environment). Κάθε sketch της Processing είναι μια υποκλάση της Java κλάσης PApplet η οποία παρέχει τα περισσότερα χαρακτηριστικά της γλώσσας Processing. Γενικά όποτε προγραμματίζουμε με την Processing, όλες οι κλάσεις που έχουν δηλωθεί συμπεριφέρονται ως εμφωλιασμένες κλάσεις όταν ο κώδικας μεταφράζεται σε «καθαρή» JAVA πριν περάσει στον compiler. Αυτό σημαίνει ότι η χρήση των (static) στατικών
μεταβλητών και μεθόδων μέσα στις κλάσεις απαγορεύεται εκτός και αν ειπώθει ρητά στην Processing ότι επιθυμούμε να προγραμματίσουμε σε καθαρή JAVA. Η Processing επίσης επιτρέπει στους χρήστες να δημιουργήσουν τις δικές τους κλάσεις μέσα στο PApplet sketch. Αυτό δίνει την δυνατότητα σε πολυπλοκότερες μορφές δεδομένων να παίρνουν κάθε αριθμό σαν παράμετρο και να αποφεύγονται περιορισμοί να χρησιμοποιούνται μόνο κλασικοί τύποι δεδομένων όπως ακέραιοι (int), χαρακτήρες (char), πραγματικοί (float) και τιμές χρωμάτων(color RGB,ARGB, hex).

Εικόνα 3 : Το προγραμματιστικό περιβάλλον της Processing

Παραδείγματα

Παρακάτω θα δωθούν δύο πολύ απλά παραδείγματα της Processing, όπως αναφέραμε θα παρουσιαστούν πιο πολύπλοκα παραδείγματα της γλώσσας καθώς και επεξηγήσεις στο κεφάλαιο Τεχνολογία. Τα παρακάτω Processing sketches είναι ισοδύναμα με ένα προγραμματιστικό παράδειγμα τύπου Hello World και απλά σχεδιάζουν μια γραμμή από το σημείο (15,25) στο (70,90).

```
line(15, 25, 70, 90);
```



Εικόνα 4: Το αποτέλεσμα του προγράμματος 1

Ο παρακάτω κώδικας είναι ένα καλύτερο παράδειγμα που παρουσιάζει την όψη και την λογική της δομής ενός Processing προγράμματος.
Με // γράφουμε σχόλια στον κώδικα.

```
void setup() {            // πρόγραμμα 1
       size(100, 100);
       stroke(0);
       background(188);
}

void draw() {
       line(15, 25, 70, 90);
}
```

Συγγενικά Projects

Design By Numbers
Η Processing βασίστηκε στη δουλειά που είχε γίνει πιο πριν στο project: Design By Numbers του MIT. Δανείστηκε από αυτό πολλές ιδέες και είναι ουσιαστικά το «παιδί» του πειράματος εκείνου.

Wiring, Arduino and Fritzing
Στην συνέχεια, από την Processing δημιουργήθηκε ένα άλλο project που ονομάζεται Wiring, το οποίο χρησιμοποιεί το περιβάλλον ανάπτυξης εφαρμογών της Processing σε συνδυασμό με μια απλοποιημένη μορφή της γλώσσας C++ , ως ένα τρόπο να διδάξει στους καλιτέχνες πώς να προγραμματίζουν microcontrollers. Πλέον υπάρχουν δύο ξεχωριστά hardware projects, το Wiring και το Arduino, τα οποία χρησιμοποιούν το ίδιο περιβάλλον ανάπτυξης εφαρμογών (IDE) και γλώσσα. Το Fritzing είναι ένα άλλο λογισμικό περιβάλλον του ιδίου είδους, το οποίο βοηθάει σχεδιαστές και καλλιτέχνες να αρχειοθετούν τα διαδραστικά τους προτότυπα και να κάνουν ευκολότερα το βήμα από το φυσικό πρωτότυπο στο τελικό προϊόν.

Mobile Processing

Επίσης άλλο ένα project που προέκυψε από την Processing ονομάστηκε Mobile Processing και δημιουργήθηκε από τον Francis Li. Το project επέτρεπε να αναπτυχθεί λογισμικό χρησιμοποιώντας την γλώσσα και το περιβάλλον της Processing για να τρέξει σε κινητές συσκευές που υποστήριζαν JAVA. Σήμερα δεν είναι ενεργό ενώ κάποιες λειτουργίες του παρέχονται άμεσα απο την Processing.

Processing.js

Το 2008, ο John Resig συνέδεσε την Processing με την JavaScript χρησιμοποιώντας το Canvas Element της HTML5 για την αναπαραγωγή των γραφικών (rendering). Επιτρέποντας έτσι να χρησιμοποιείται η Processing στους web-browsers χωρίς να είναι αναγκαία η προσθήκη μιας JAVA επέκτασης (plugin). Έκτοτε η κοινότητα ανοικτού λογισμικού συμπεριλαμβανομένων του κολεγίου Seneca έχουν αναλάβει το project.

iProcessing

Το iProcessing δημιουργήθηκε για να βοηθήσει στην ανάπτυξη iPhone εφαρμογών χρησιμοποιώντας την γλώσσα Processing. Είναι μία ενοποίηση της Processing.js βιβλιοθήκης και ενός Javascript framework εφαρμογών για το iPhone.

Spde
Το Spde (συντομογραφία του Scala Processing Development Environment) αντικαθιστά το απλοποιημένο Java συντακτικό της Processing με την γλώσσα προγραμματισμού Scala. Η οποία είναι επίσης συμβατή με την Java πλατφόρμα και επιβάλλει τις ίδιες απαγορεύσεις όπως το να μην επιτρέπει τις στατικές μεθόδους ενώ παράλληλα δίνει την δυνατότητα να αναπτυχθεί πιο περιεκτικός κώδικας και υποστηρίζει προγραμματισμό με συναρτήσεις (functional programming).

Quil
Το Quil γνωστό προηγουμένως ως clj-processing είναι ένας wrapper για την Processing στην γλώσσα Clojure, μια διάλεκτος της Lisp που τρέχει σε Java πλατφόρμα.

Βραβεία

Το 2005 οι Reas και Fry κέρδισαν το βραβείο Golden Nica από την Ars Electronica στη κατηγορία Net Vision για τη δουλειά τους πάνω στη Processing.

Ο Ben Fry κέρδισε το 2011 το εθνικό βραβείο σχεδιασμού από το μουσείο Smithsonian Coopen-Helwitt στη κατηγορία του διαδραστικού σχεδιασμού. Η δήλωση του βραβείου ήταν η εξής: "Ζωγραφίζοντας σε ένα φόντο χρησιμοποιώντας την επιστήμη των υπολογιστών και των γραφικών, ο Ben Fry επιδιώκει μια μακροχρόνια γοητεία με την οπτικοποίηση των δεδομένων. Ως διευθυντής της Fathom Information Design στη Βοστώνη, ο Fry ανέπτυξε και εγκατέστησε λογισμικό, έγραψε βιβλία και έντυπα που απεικονίζουν και εξηγούν θέματα από το ανθρώπινο γονιδίωμα μέχρι την εξέλιξη των αρχείων κειμένου. Με το Casey Reas, δημιούργησε το project Processing, ένα ανοιχτό (open-source) προγραμματιστικό περιβάλλον που διδάσκει υπολογιστικό σχεδιασμό και δημιουργεί σχεδιαστικό διαδραστικό με πολυμέσα λογισμικό."

Άδεια χρήσης

Οι βιβλιοθήκες πυρήνα της Processing, ο κώδικας που συμπεριλαμβάνεται στις εφαρμογές και τα applets έχουν άδεια χρήσης κάτω από το GNU Lesser General Public License, επιτρέποντας στους χρήστες να δημοσιεύσουν τον δικό τους αυθεντικό κώδικα με δυνατότητα επιλογής άδειας χρήσης. Το προγραμματιστικό περιβάλλον (IDE) έχει άδεια κάτω από το GNU General Public License.

Όνομα

Αρχικά η Processing είχε το URL στο proce55ing.net, επειδή το processing domain ήταν πιασμένο. Τελικά οι Reas και Fry απέκτησαν το domain name. Παρόλο που το όνομα περιείχε αριθμούς και γράμματα εξακολουθούσε να προφέρεται processing. Παρά την αλλαγή του ονόματος η Processing ακόμα χρησιμοποιεί τη συντομογραφία p5 κάποιες φορές .

Όνομα

Αρχικά η Processing είχε το URL στο proce55ing.net, επειδή το processing domain ήταν πιασμένο. Τελικά οι Reas και Fry απέκτησαν το domain name. Παρόλο που το όνομα περιείχε αριθμούς και γράμματα εξακολουθούσε να προφέρεται processing. Παρά την αλλαγή του ονόματος η Processing ακόμα χρησιμοποιεί τη συντομογραφία p5 κάποιες φορές .

2.3 Τι είναι το OpenNI

OpenNI ή *Open Natural Interaction* είναι μια μη κερδοσκοπική οργάνωση πρωτοβουλία της βιομηχανίας, που επικεντρώνεται στην πιστοποίηση και την βελτίωση της λειτουργικότητας φυσικών και οργανικών διεπαφών χρήστη για συσκευές φυσικής αλληλεπίδρασης όπως για παραδειγμα το Kinect, για εφαρμογές που χρησιμοποιούν αυτές τις συσκευές και για λογισμικό τύπου middleware που διευκολύνει την πρόσβαση και τη χρήση των εν λόγω συσκευών.

Επισκόπηση

Ο οργανισμός ιδρύθηκε το Νοέμβριο του 2010, με το δικτυακό τόπο να δημοσιεύεται στις 8 Δεκεμβρίου. Ένα από τα κύρια μέλη είναι η PrimeSense, η εταιρεία πίσω από την τεχνολογία που χρησιμοποιείται στο Kinect.Τον Δεκέμβριο του 2010, η PrimeSense της οποίας η ανίχνευση βάθους είναι σχέδιο αναφοράς στο οποίο βασίζεται το Kinect, κυκλοφόρησε τους δικους της οδηγούς ανοικτού κώδικα μαζί με το middleware ανίχνευσης κίνησης που ονομάζεται NITE. Η PrimeSense αργότερα ανακοίνωσε ότι έχει συνεργαστεί με την Asus για να αναπτύξει μια συσκευή συμβατή με PC παρόμοια με το Kinect, η οποία θα ονομάζεται WAVI Xtion και είχε προγραμματιστεί να κυκλοφορήσει το δεύτερο τρίμηνο του 2012.

Το λογισμικό τους χρησιμοποιείται επί του παρόντος σε μια ποικιλία έργων ανοικτού κώδικα

μεταξύ της ακαδημαϊκής και της χομπιστικής κοινότητας. Πρόσφατα, οι εταιρείες λογισμικού προσπαθήσαν να επεκτείνουν την επιρροή του OpenNI, κάνοντας την εν λόγω ενσωματωμένη τεχνολογία δραματικά απλούστερη.
Μετά την εξαγορά της PrimeSense από την Apple, ανακοινώθηκε ότι η ιστοσελίδα OpenNI.org θα κλείσει στις 23 Απριλίου 2014. Αμέσως μετά την διακοπή λειτουργίας, οι οργανισμοί που χρησιμοποιούσαν το OpenNI θα τους επιτρεπόταν να συνεχίσουν να διατηρούν την τεκμηρίωμένα έγραφα και τα εκτελέσιμα αρχεία για μελλοντική χρήση, όπως στη σελίδα Structure.io

Συσκευές φυσικής αλληλεπίδρασης

Συσκευές φυσικής αλληλεπίδρασης ή φυσικές διεπαφές (Natural Interaction Devices ή Natural Interfaces) είναι συσκευές που ανιχνεύουν και καταγράφουν τις κινήσεις του σώματος και τους ήχους για να επιτρέψουν μια πιο φυσική αλληλεπίδραση των χρηστών με τους υπολογιστές στο πλαίσιο μιας φυσικής διεπαφής χρήστη. Το Kinect και WAVI X-tion είναι παραδείγματα τέτοιων συσκευών.

Το πλαίσιο εργασίας OpenNI

Το πλαίσιο εργασίας ή framework OpenNI παρέχει ένα σύνολο από APIs ανοικτού κώδικα. Αυτά τα APIs προορίζονται να αποτελέσουν πρότυπο για εφαρμογές, δίνοντας πρόσβαση σε συσκευές φυσικής αλληλεπίδρασης. Το πλαίσιο API μερικές φορές αναφέρεται με το όνομα OpenNI SDK.
Τα APIs παρέχουν υποστήριξη για αναγνώριση και καταγράφή φωνής, φωνητικών εντολών, χειρονομιών και άλλων κινήσεων του ανθρώπινου σώματος. Σε αυτή τη πτυχιακή εργασία χρησιμοποιήθηκε το πλαίσιο OpenNI και συγκεκριμένα το SimpleOpenNI ως βιβλιοθήκη της Processing για τις εφαρμογές που αναπτύχθηκαν.

2.4 Τι είναι η Minim

Η Minim είναι μια βιβλιοθήκη ήχου της Processing που χρησιμοποιεί το JavaSound API προσφέροντας μια προγραμματιστικές λύσης σε αυτούς που αναπτύσουν εφαρμογές οι οποίες σχετίζονται με ήχο. Η Minim χρησιμοποιεί το JavaSound API στο παρασκήνιο για το ηχητικό κομμάτι και λειτουργεί με παραδείγματα που παρέχουν χρήσιμο κώδικα εκκίνησης για προγραμματισμό ηχητικών δεδομένων. Η φιλοσοφία πίσω από το API είναι να γίνει η ενσωμάτωση ήχου στις εφαρμογές όσο το δυνατόν απλούστερη, ενώ εξακολουθεί να παρέχει ευελιξία για τους πιο προχωρημένους χρήστες. Δεν υπάρχουν συναρτήσεις επιστροφής και δεν χρειάζεται ποτέ να χειρίζονται απευθείας οι δειγματοληπτικοί πίνακες, δηλαδή όλη η αδιάφορη χρονοβόρα εργασία γίνεται αυτόματα απ'τη βιβλιοθήκη.
Ειδικότερα, στην εξωτερική βιβλιοθήκη Minim του Processing υπάρχουν μέθοδοι για λήψη αντικειμένων για την αναπαραγωγή αρχείων ήχου όπως τα AudioSample, AudioSnippet και AudioPlayer. Ακόμα για λήψη ενός AudioRecorder, της συσκευής δηλαδή που είναι υπεύθυνη για το πώς θα καταγραφεί ο ήχος στο δίσκο. Επίσης λήψη ενός AudioInput, με το οποίο μπορεί να παρακολουθείται το line-in και το μικρόφωνο του υπολογιστή. Τέλος, λήψη ενός AudioOutput, τρόπος με τον οποίο αναπαράγεται ο ήχος που εξάγεται μέσα από μια εφαρμογή του Processing.
Όλες αυτές οι κλάσεις παρέχονται από τα AudioStreams της Minim, τα οποία είναι threads που κάνουν την πραγματική επεξεργασία του ήχου για είσοδο και έξοδο. Για το λόγο αυτό, θα πρέπει να καλείται πάντα η close μέθοδος για κάθε αντικείμενο Audio όταν τελειώσει η χρήση του.

Στοιχεία και χαρακτηριστικά της Minim:

· AudioPlayer: μονοφωνική και στερεοφωνική αναπαραγωγή αρχείων WAV, AIFF, AU, SND, MP3.
· AudioMetaData: αντικείμενο που περιέχει metadata για ένα αρχείο, όπως οι ετικέτες ID3.
· AudioRecorder: μονοφωνική και στερεοφωνική καταγραφή ήχου είτε buffered είτε direct στο δίσκο.
· AudioInput: μονοφωνική και στερεοφωνική παρακολούθηση της εισόδου.

· AudioOutput: μονοφωνική και στερεοφωνική ηχητική σύνθεση.
· AudioSignal: μια απλή διεπαφή για τη δημιουργία καινούριων sound synthesis κλάσεων.
· Όλες οι τυπικές κυματομορφές ακολουθούνται από μια γεννήτρια ροζ θορύβου και μια γεννήτρια λευκού θορύβου. Επιπλέον μπορεί να επεκταθεί η κλάση Oscillator για την εύκολη εκτέλεση της εκάστοτε περιοδικής κυματομορφής.
· AudioEffect: μια απλή διεπαφή για τη δημιουργία καινούριων audio effects.
· Με κάποιο βαθυπερατό, υψιπερατό ή ζωνοπερατό φίλτρο και με τα φίλτρα εγκοπής, μπορεί να επεκταθεί η κλάση IIRFilter για την εύκολη εκτέλεση των εκάστοτε φίλτρων IIR.
· Εύκολο να αποδοθούν τα σήματα και τα αποτελέσματα σε εισόδους και εξόδους ήχου. Όλη η μίξη και επεξεργασία γίνεται αυτόματα.
· Παρέχει μια FFT κλάση που υλοποιεί ανάλυση του φάσματος.
· Παρέχει μία BeatDetect κλάση που υλοποιεί ανίχνευση beat.

2.5 Τι είναι το RGB-Image

Το RGB-Image είναι η εικόνα που παρέχει το Kinect μέσω της RGB κάμερας που διαθέτει. RGB σημαίνει Red Green Blue και συμβολίζει το φάσμα του ορατού φωτός. Έτσι η μια κάμερα του Kinect μπορεί να καταγράψει το ορατό φάσμα φωτός σε 30 frames / second δηλαδή 30 στιγμιότυπα-εικόνες για κάθε δευτερόλεπτο. Παρακάτω δίνεται το πιο απλό πρόγραμμα Processing το οποίο καταγράφει και αναπαράγει στην οθόνη του υπολογιστή αυτό που "βλέπει" το Kinect με την rgb κάμερα.

```
import SimpleOpenNI.*;
SimpleOpenNI kinect;
void setup()
{
size(640, 480);
kinect = new SimpleOpenNI(this);
kinect.enableRGB();
}
void draw()
{
kinect.update();
image(kinect.rgbImage(), 0, 0);
}
```

Εικόνα 5: RGB Image

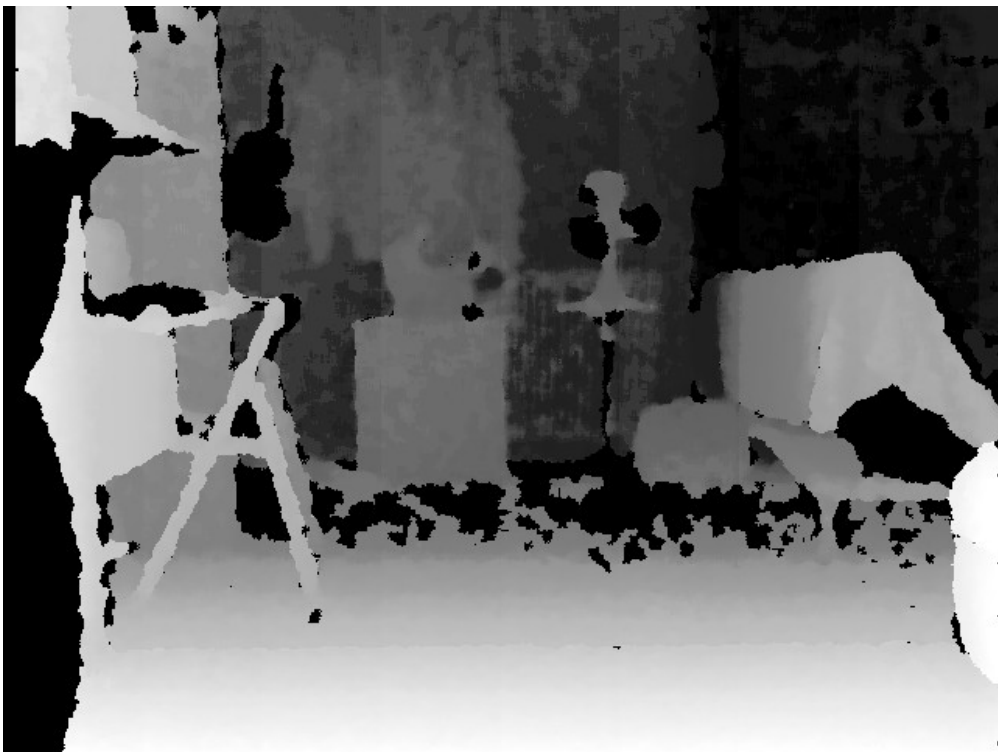
2.6 Τι είναι το Depth-Image

Το depth-Image είναι η εικόνα βάθους που παρέχει το Kinect μέσω της υπέρυθρης depth κάμερας που διαθέτει. Στη ουσία με αυτήν την κάμερα το Kinect μας δείχνει όχι πώς φαίνονται τα αντικείμενα με τα χρώματά τους αλλά που βρίσκονται στο χώρο με συντεταγμένες τριών διαστάσεων. Το depth-Image περιέχει μόνο αποχρώσεις του γκρι. Συγκεκριμένα όσο πιο μακρυά από την κάμερα βρίσκεται ένα σημείο της εικόνας τόσο πιο σκούρο γκρι θα είναι και αντίστροφα. Στην περίπτωση που κάποιο σημείο απεικονίζεται με μαύρο χρώμα τότε ισχύει τουλάχιστον ένα από τα παρακάτω:

1. Το σημείο βρίσκεται πολύ κοντά στην κάμερα σε ακτίνα μικρότερη των 50cm οι οποία είναι η ελάχιστη απόσταση που μπορεί καταγράψει το Kinect for Windows.
2. Το σημείο βρίσκεται πολύ μακρυά από την κάμερα σε ακτίνα μεγαλύτερη των 4,5m οι οποία είναι η μέγιστη απόσταση που μπορεί καταγράψει το Kinect.
3. Το σημείο δεν επιστρέφει πίσω το υπέρυθρο στίγμα που εκπέμπει ο αισθητήρας του Kinect. Αυτό συμβαίνει επειδή λόγω θορύβου, περίπλοκων γωνιών και λεπτομερειών, η αντανάκλαση των υπεριώδων ακτίνων σε εκείνα τα σημεία στρέφεται εκτός της ανιχνευτικής δυνατότητας του Kinect.

Άρα η πληροφορία για την θέση του σημείου χάνεται σε αυτές τις περιπτώσεις με αποτέλεσμα να εμφανίζεται με μαύρο χρώμα στο depth-Image. Παρακάτω δίνεται το πιο απλό πρόγραμμα Processing το οποίο καταγράφει και αναπαράγει στην οθόνη του υπολογιστή αυτό που “βλέπει“ το Kinect με την depth κάμερα.

```
import SimpleOpenNI.*;
SimpleOpenNI kinect;
void setup()
{
size(640, 480);
kinect = new SimpleOpenNI(this);
kinect.enableDepth();
}
void draw()
{
kinect.update();
image(kinect.depthImage(), 0, 0);
}
```



Εικόνα 6: Depth Image

Σύγκριση των RGB και Depth-Image

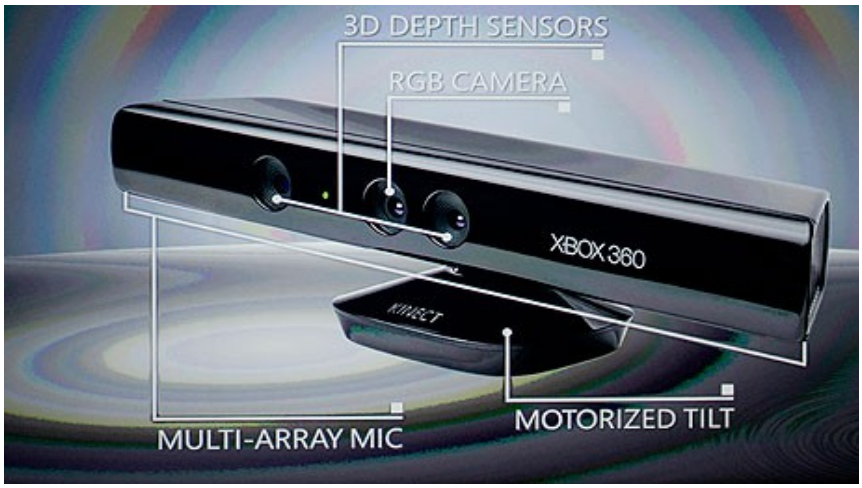
Συγκρίνοντας τις δύο εικόνες παρακάτω βγαίνει το συμπέρασμα ότι έχουν ληφθεί από διαφορετική οπτική γωνία το οποίο είναι απολύτως λογικό αφού οι κάμερες RGB και Depth βρίσκονται σε διαφορετικά σημεία πάνω στην συσκευή Kinect.

Εικόνα 7: Σύγκριση RGB και Depth Image

3. Τεχνολογία

Σε αυτό το κεφάλαιο θα αναλυθεί η τεχνολογία του υλικού και του λογισμικού που χρησιμοποιήθηκαν για την παρούσα πτυχιακή εργασία. Συγκεκριμένα θα παρουσιαστούν τεχνολογικά μέρη του συστήματος Kinect και βασικά στοιχεία της προγραμματιστικής γλώσσας Processing που χρησιμοποιήθηκαν.

3.1 Το σύστημα Kinect



Εικόνα 8: Τα κύρια στοιχεία του Kinect

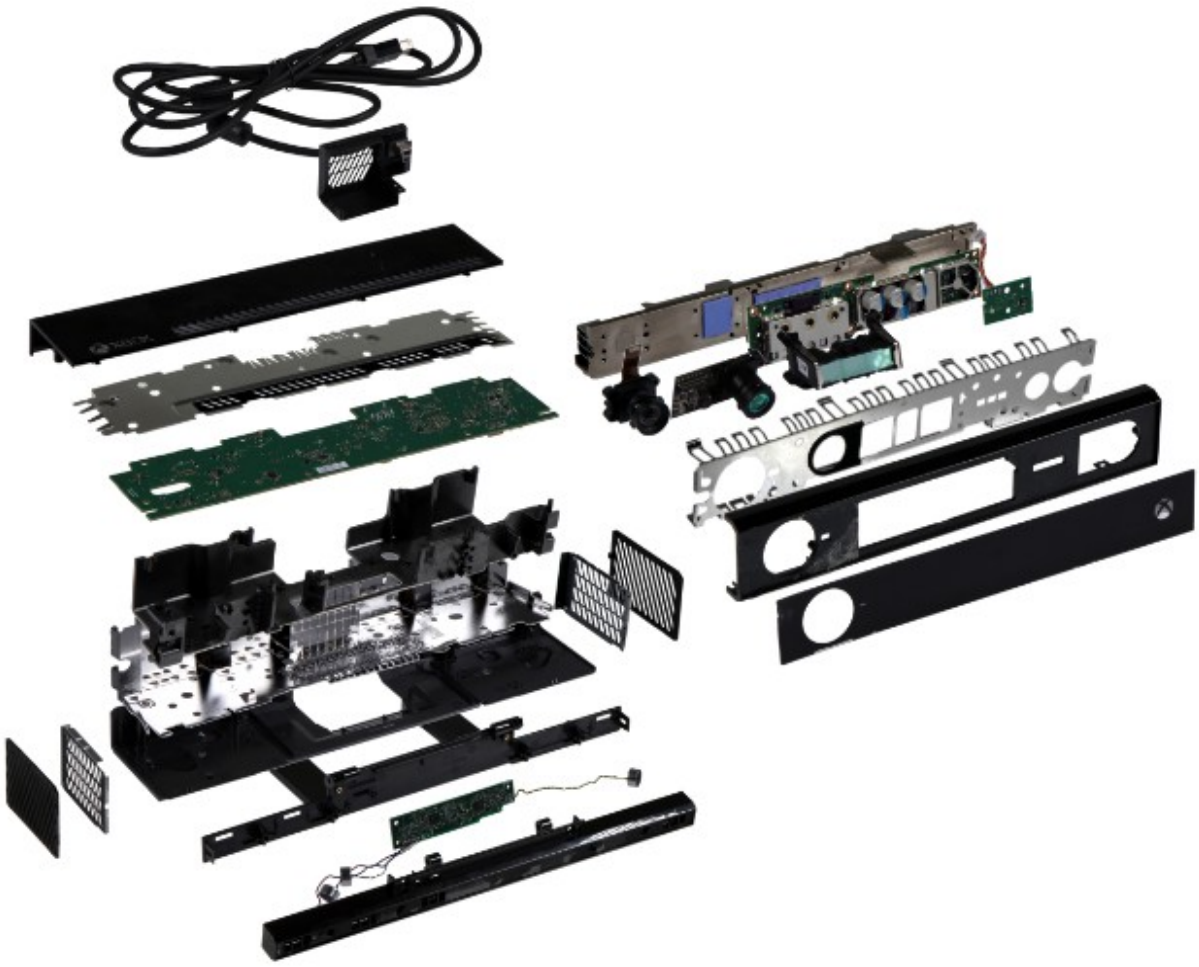
Το Kinect βασίζεται στην τεχνολογία λογισμικού που αναπτύχθηκε εσωτερικά από την Rare, θυγατρική της Microsoft Game Studios που ανήκει στην Microsoft, και την τεχνολογία της depth κάμερας από την ισραηλινή PrimeSense. Η οποία ανέπτυξε ένα σύστημα που μπορεί να ερμηνεύσει συγκεκριμένες χειρονομίες, καθιστώντας δυνατόν τον "hands-free" έλεγχο των ηλεκτρονικών συσκευών χρησιμοποιώντας έναν υπέρυθρο προβολέα, μία κάμερα και ένα ειδικό μικροτσίπ για να παρακολουθείται η κίνηση των αντικειμένων και των ατόμων σε τρεις διαστάσεις. Αυτό το σύστημα 3D scanner που ονομάζεται Light Coding και χρησιμοποιεί μια παραλλαγή της εικόνας με βάση την 3D ανασυγκρότηση.

Ο Αισθητήρας Kinect είναι μια οριζόντια μπάρα που συνδέεται σε μια μικρή βάση με ένα μηχανοκίνητο άξονα περιστροφής και έχει σχεδιαστεί για να τοποθετείται κατά μήκος πάνω ή κάτω από την οθόνη. Η συσκευή διαθέτει "κάμερα RGB, αισθητήρα βάθους και multi-array μικρόφωνο που λειτουργεί με ιδιόκτητο λογισμικό", που παρέχουν πλήρης ανίχνευση-καταγραφή 3D κίνησης, αναγνώριση προσώπου και δυνατότητες αναγνώρισης φωνής. Κατά την έναρξη, η αναγνώριση φωνής ήταν διαθέσιμη μόνο στην Ιαπωνία, Ηνωμένο Βασίλειο, τον Καναδά και Ηνωμένες Πολιτείες. ΗΗπειρωτική Ευρώπη έλαβε το χαρακτηριστικό αργότερα, την άνοιξη του 2011. Επί του παρόντος, αναγνώριση φωνής υποστηρίζεται στην Αυστραλία, τον Καναδά, τη Γαλλία, τη Γερμανία, την Ιρλανδία, την Ιταλία, την Ιαπωνία, το Μεξικό, τη Νέα Ζηλανδία, Ηνωμένο Βασίλειο και Ηνωμένες Πολιτείες.

Εικόνα 9: Μία ματιά στο εσωτερικό του Kinect

Η συστοιχία μικροφώνων του Kinect επιτρέπει τη διεξαγωγή ακουστικού εντοπισμού πηγής και την καταστολή του θορύβου του περιβάλλοντος, επιτρέποντας έτσι ομαδικές συνεδρίες χωρίς επιπλοκές. Ο αισθητήρας βάθους αποτελείται από ένα υπέρυθρο προβολέα λέιζερ σε συνδυασμό με έναν μονόχρωμο αισθητήρα CMOS, ο οποίος καταγράφει τα δεδομένα σε βίντεο 3D κάτω από οποιεσδήποτε συνθήκες φωτισμού του περιβάλλοντος. Το εύρος ανίχνευσης του αισθητήρα βάθους είναι ρυθμιζόμενο, και το λογισμικό του Kinect είναι σε θέση αυτόματης βαθμονόμησης ανάλογα το είδος διαδραστικότητας και το φυσικό περιβάλλον του παίκτη, ενώ παράλληλα ελέγχει την παρουσία των επίπλων ή άλλων εμποδίων.

Περιγράφεται από το προσωπικό της Microsoft ως την κύρια καινοτομία του Kinect, η τεχνολογία του λογισμικού επιτρέπει προηγμένη αναγνώρισης χειρονομιών, αναγνώριση προσώπου και αναγνώριση φωνής. Σύμφωνα με τις πληροφορίες που παρέχονται στους εμπόρους λιανικής πώλησης, το Kinect είναι σε θέση ταυτόχρονης παρακολούθησης έως έξι ατόμων, μεταξύ των οποίων δύο ενεργούς παίκτες για την ανάλυση κίνησης με εξαγωγή χαρακτηριστικών 20 αρθρώσεων ανά παίκτη. Ωστόσο, η PrimeSense έχει δηλώσει ότι ο αριθμός των ανθρώπων που η συσκευή μπορεί να "δει" (αλλά όχι παίκτες) περιορίζεται μόνο από το πόσα άτομα θα χωρέσουν στο οπτικό πεδίο της κάμερας.



Εικόνα 10: Αναλυτικά η δομή του Kinect

Με την χρήση αντίστροφης μηχανικής βγήκε το συμπέρασμα ότι διάφορα μοντέλα αισθητήρων Kinect έχουν έξοδο βίντεο περίπου 9 Hz έως 30 Hz ανάλογα με την ανάλυση. Η προεπιλεγμένη RGB ροή βίντεο χρησιμοποιεί VGA ανάλυση 8-bit (640 × 480 pixel) με ένα φίλτρο χρώματος Bayer, αλλά η συσκευή είναι σε θέση να χρησιμοποιήσει αναλύσεις έως 1280x1024 (με χαμηλότερο ρυθμό καρέ/frame rate) και άλλες μορφές χρώματος, όπως UYVY. Η μονόχρωμη ροή βίντεο του βάθους ανίχνευσης είναι σε ανάλυση VGA (640 × 480 pixel) με βάθος χρώματος 11-bit στην γκρι κλίμακα, το οποίο παρέχει 2.048 διακριτά επίπεδα. Το Kinect μπορεί επίσης να διαθέσει σε ροή video από την κάμερα υπερύθρων άμεσα (δηλαδή πριν μετατραπεί σε χάρτη βάθους), όπως 640x480 βίντεο, ή 1280x1024 με χαμηλότερο ρυθμό καρέ. Ο αισθητήρας Kinect έχει ένα πρακτικό όριο κυμαίνεται 1.2-3.5 m (03.09 - 11.05 ft) απόσταση, όταν χρησιμοποιείται με το λογισμικό Xbox και 0.5m – 4,5m η έκδοση Kinect for Windows. Η περιοχή που απαιτείται για να «χειριστεί» από το χρήστη το Kinect for Windows είναι περίπου 6 τετραγωνικά μέτρα , αν και ο αισθητήρας μπορεί να διατηρήσει την παρακολούθηση μέσω ενός εκτεταμένου εύρους περίπου από 0,5 στα 4,5 m . Ο αισθητήρας έχει ένα γωνιακό οπτικό πεδίο 57 ° οριζοντίως και 43 ° κατακόρυφα, ενώ το μοτέρ περιστροφής προσφέρει κλίση του αισθητήρα έως 27 °, είτε πάνω ή κάτω. Το οριζόντιο πεδίο του αισθητήρα Kinect στην ελάχιστη απόσταση θέασης είναι περίπου 0,8m και το κάθετο πεδίο είναι περίπου 63cm με αποτέλεσμα μια ανάλυση πάνω από 1,3 χιλιοστά ανά pixel. Η συστοιχία μικροφώνων διαθέτει τέσσερα μικρόφωνα και λειτουργεί με κανάλι επεξεργασίας ήχου 16-bit και με ρυθμό δειγματοληψίας 16 kHz. Επειδή ο ηλεκτρονικός μηχανισμός κλίσης του αισθητήρα Kinect απαιτεί περισσότερη ενέργεια από ότι οι θύρες USB του Xbox 360 μπορούν να παρέχουν, η συσκευή χρησιμοποιεί μία ιδιόκτητη υποδοχή USB που συνδυάζει την επικοινωνία με πρόσθετη ισχύ. Επανασχεδιασμένα μοντέλα του Xbox 360 S περιλαμβάνουν μια ειδική θύρα AUX για την υποδοχή βύσματος, ενώ τα μεγαλύτερα μοντέλα απαιτούν ένα ειδικό καλώδιο παροχής ρεύματος (περιλαμβάνεται με τον αισθητήρα), που χωρίζει τη σύνδεση σε ξεχωριστές συνδέσεις USB και τροφοδοσίας. Έτσι τροφοδοτείται με ρεύμα από το ηλεκτρικό δίκτυο μέσω ενός προσαρμογέα AC.

Λογισμικό

Το σύστημα Kinect απαιτεί τουλάχιστον 190 MB διαθέσιμου χώρου αποθήκευσης. Το λογισμικό του συστήματος Kinect επιτρέπει στους χρήστες να χειρίζονται το Xbox 360 Dashboard, μια διεπαφή χρήστη-κονσόλας μέσω φωνητικών εντολών και χειρονομιών. Τεχνικές όπως η αναγνώριση φωνής και η αναγνώριση του προσώπου πραγματοποιούνται αυτόματα για τους χρήστες. Ανάμεσα στις εφαρμογές για το Kinect είναι τοVideo Kinect, το οποίο δίνει τη δυνατότητα voice chat ή video chat με άλλους χρήστες του Xbox 360 ή χρήστες του Windows Live Messenger. Η εφαρμογή μπορεί να χρησιμοποιήσει τη λειτουργικότητα εντοπισμού Kinect και τον μηχανοκίνητο άξονα του αισθητήρα Kinect για να κρατήσει τους χρήστες μέσα στο οπτικό του πεδίο, ακόμη και καθώς κινούνται γύρω του. Άλλες εφαρμογές που υποστηρίζουν το Kinect: ESPN, Zune Marketplace, Netflix, Hulu Plus και Last.fm. Η Microsoft επιβεβαίωσε αργότερα ότι όλες για όλες τις επικείμενες αιτήσεις θα απαιτεί από αυτές να έχουν λειτουργικότητα Kinect για να πιστοποιηθούν. Βιντεοπαιχνίδια που απαιτούν το Kinect έχουν ένα πορφυρό αυτοκόλλητο πάνω τους με ένα λευκό περίγραμμα του αισθητήρα Kinect, είναι τυπωμένο το "Απαιτείται το Kinect Sensor" κάτω από το άσπρο κείμενο, και είναι επίσης σε μωβ συσκευασία. Βιντεοπαιχνίδια που έχουν προαιρετική υποστήριξη Kinect (που σημαίνει ότι δεν είναι απαραίτητο το Kinect για να παίξει το παιχνίδι ή ότι υπάρχουν προαιρετικά Kinect-minigames) διαθέτουν ένα πρότυπη πράσινη θήκη Xbox 360 με μωβ γραμμή κάτω από την κεφαλίδα, ένα περίγραμμα του αισθητήρα Kinect και είναι τυπωμένο το "Better with Kinect Sensor" δίπλα στο λευκό κείμενο.

Εικόνα 11: Μοντέλα που παίζουν Kinect βιντεοπαιχνίδι racing σε επίδειξη

Εικόνα 12: Xbox 360 Βιντεοπαιχνίδια

Το Kinect ξεκίνησε στις 4 Νοεμβρίου 2010, με 17 τίτλους. Τρίτοι εκδότες των διαθέσιμων Kinect βιντεοπαιχνιδιών είναι, μεταξύ άλλων, η Ubisoft, Electronic Arts, LucasArts, THQ, Activision, Konami, Sega, Capcom, Namco Bandai και το MTV-Games.

Εικόνα 13: Στιγμιότυπο χρήστη-Kinect βιντεοπαιχνίδιου πολεμικών τεχνών

Μαζί με τα παιχνίδια του εμπορίου, υπάρχουν επίσης συλλογές στο Xbox Live Arcade που χρησιμοποιούν αποκλειστικά το Kinect.

3.2 Η Γλώσσα προγραμματισμού Processing

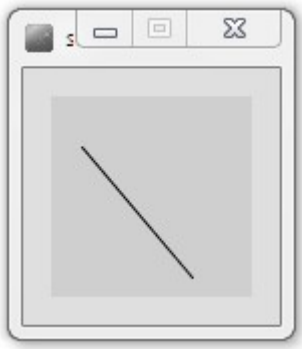
Στο σημείο αυτό θα παρουσιαστούν και θα αναλυθούν βασικές εντολές και απλά προγράμματα της αντικειμενοστραφής γλώσσας προγραμματισμού που χρησιμοποιήθηκε στις εφαρμογές.

Εικόνα 3 : Το προγραμματιστικό περιβάλλον της Processing

Στην εικόνα 3 φαίνεται το προγραμματιστικό περιβάλλον της Processing. Είναι ένα απλό προγραμματιστικό περιβάλλον το οποίο αποτελείται από τα παρακάτω μέρη. Την λευκή περιοχή κειμένου όπου "γράφεται" ο κώδικας. Ακριβώς από πάνω φαίνεται το Sketch_140701 το οποίο είναι το όνομα του προγράμματος, επίσης το όνομα φαίνεται και στον τίτλο του παραθύρου μαζί με τον αριθμό έκδοσης της Processing που είναι εγκατεστημένη. Ακόμα, στο πάνω μέρος υπάρχει μια σειρά από εικονίδια που είναι η μπάρα εργαλειοθήκης (toolbar), ενώ πιο δεξιά που γράφει JAVA βρίσκεται ο Mode Manager, όπου εκεί δίνεται η δυνατότητα επιλογής ανάμεσα στα mode Java, Javascript, CoffeeScript, PDE X, Python και Tweak. Τέλος στα δεξιά κάτω βρίσκεται ένας ακέραιος αριθμός (συγκεκριμένα το 1 στην εικόνα 3) ο οποίος δείχνει ποια γραμμή κώδικα έχουμε επιλέξει ενώ από πάνω στην περιοχή με μαύρο χρώμα παρουσιάζονται τα μηνύματα της εξόδου, του Compiler και λοιπές πληροφορίες που προορίζονται για τον προγραμματιστή.

Το πρόγραμμα 1

Παρακάτω δίνεται σαν πρώτο παράδειγμα το πρόγραμμα 1. Παρουσιάστηκε συνοπτικά και προηγουμένως, όμως τώρα για καλύτερη κατανόηση θα αναλυθεί η λειτουργία και το συντακτικό κάθε γραμμής του.



Εικόνα 4: Το αποτέλεσμα του προγράμματος 1

```
void setup() {
       size(100, 100);  //Σχόλια: Πρόγραμμα 1
       stroke(0);
       background(188);
}

void draw() {
       line(15, 25, 70, 90);
}
```

Ανάλυση

`void setup() {`

Στην πρώτη γραμμή του προγράμματος έχουμε την βασική συνάρτηση setup. Πρόκειται για την συνάρτηση που τρέχει πρώτη στο πρόγραμμα μία φορά και περιέχει τις εντολές που σχεδιάζουν και προσδιορίζουν τα χαρακτηριστικά του παράθυρου που εμφανίζεται με το τρέξιμο της εφαρμογής. Όπως φαίνεται στο πρόγραμμα 1 μέσα στις παρενθέσεις της setup δεν υπάρχει τίποτα άρα δεν δέχεται κάποιο όρισμα. Τέλος η λέξη void δείχνει ότι δεν επιστρέφει κάποια τιμή μετά τη κλήσης της. Αναφέραμε και πιο πριν πως ότι ακολουθεί στην ίδια γραμμή μετά τον συμβολισμό // είναι σχόλια για το πρόγραμμα μας.
Συγκεκριμένα η setup περιέχει τρεις γραμμές κώδικα από το σύμβολο { μέχρι το }

`size(100, 100);`

Η size είναι η εντολή που καθορίζει το μέγεθος του παραθύρου που θα δημιουργηθεί. Σύμφωνα με τις τιμές που περνάνε μέσα στις παρενθέσεις το παράθυρο θα έχει μέγεθος 100 pixels στο μηκός και 100 pixel πλάτος.

`stroke(0);`

Η stroke ορίζει το χρώμα που χρησιμοποιείται για να σχεδιαστούν γραμμές και περιγράμματα γύρω από σχήματα. Στην συγκεκριμένη περίπτωση για ένα όρισμα το εύρος των ακέραιων τιμών που θα μπορούσε να δωθεί είναι 0-255 και καθορίζει τιμές της κλίμακας του γκρι με το 0 να είναι το μαύρο και το 255 το άσπρο. Άρα στο πρόγραμμα 1 δίνοντας 0 επιλέγουμε το μαύρο χρώμα σε περίπτωση σχεδίασης. Ακόμα η εντολή stroke μπορεί να χρησιμοποιηθεί με τρία ορίσματα π.χ. `Stroke(230,9,0);`
Όπου οι τρεις ακέραιες τιμές δηλώνουν το χρώμα στην RGB κλίμακα με εύρος τιμών 0-255. Για κάθε ένα από τα χρώματα κόκκινο πράσινο μπλε έχουμε ένα ακέραιο που δηλώνει την ένταση του. Έτσι στο παράδειγμα με τιμές `(230,9,0)` θα επιλεχτεί ένα κόκκινο περίπου χρώμα σχεδίασης.

`Background(188);`

Με την εντολή αυτή επιλέγουμε το χρώμα του φόντου του παραθύρου. Ισχύει εδώ ότι και με την προηγούμενη εντολή σχετικά με τα ορίσματα και τις τιμές τους όπως σε πολλές άλλες εντολές που σχετίζονται με χρώματα. Άρα το 188 ισούται με ένα ανοικτό γκρι.

`void draw() {`

Σε αυτή την γραμμή του προγράμματος έχουμε την βασική συνάρτηση draw. Πρόκειται για την συνάρτηση που τρέχει συνεχώς στο πρόγραμμα σε αντίθεση με την setup η οποία τρέχει μία φορά

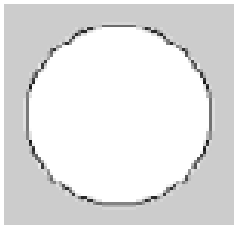
στην αρχή μόνο. Προεπιλεγμένα η Processing προσπαθεί να εκτελέσει την draw με 60 framerate δηλαδή με ρυθμό 60 φορές το δευτερόλεπτο. Αυτό βέβαια αλλάζει ανάλογα με την υπολογιστική βαρύτητα του προγράμματος και την διάθεση των πόρων του συστήματος. Για να προσδιορίσει ο χρήστης το framerate μίας Processing εφαρμογής μπορεί να χρησιμοποιήσει την εντολή framerate(x) όπου x ο ακέραιος αριθμός των εικόνων ανά δευτερόλεπτο μέσα στην setup φυσικά. Σε εφαρμογές που χρησιμοποιείται το Kinect πρέπει να θέτεται το framerate ίσο με 30 που είναι ο προεπιλεγμένος ρυθμός καταγραφής της συσκευής.

```
Line(15, 25, 70, 90);
```

Αυτή είναι η κύρια εντολή του προγράμματος 1 και η μόνη που περιέχεται στην draw.
Στην ουσία η line δέχεται τέσσερις ακέραιους αριθμούς σαν όρισμα. Οι αριθμοί αυτοί συμβολίζουν ανά ζεύγος τις συντεταγμένες ενός σημείου στο δισδιάστατο χώρο. Δηλαδή εδώ συμβολίζονται τα δύο σημεία (15,25) και (70,90). Σημειώνεται ότι το σύστημα δισδιάστατων συντεταγμένων της Processing υπολογίζει ως αρχή των αξόνων (0,0) το πρώτο pixel αριστερά πάνω στο περιεχομένο του παραθύρου της εφαρμογής και συνεχίζει με αύξοντα ρυθμό. Ο πρώτος ακέραιος δηλώνει το μήκος σε αριθμό pixels και ο δεύτερος το πλάτος. Αυτό που κάνει η line είναι να σχηματίσει μία γραμμή στο περιεχόμενο του παραθύρου της εφαρμογής από το ένα σημείο στο άλλο με το επιλεγμένο χρώμα του πιο πρόσφατου stroke στον κώδικα.
Έτσι η εντολή line για κάθε φορά που η draw εκτελείται δηλαδή 60 φορές το δευτερόλεπτο σχεδιάζει μία γραμμή από το (15,25) στο (70,90) με stroke(0) δηλαδή με μαύρο χρώμα.
Τέλος προγράμματος 1.

```
ellipse(50, 50, 80, 80);
```

Εκμεταλλευόμενοι την απλότητα της Processing μπορούμε να χρησιμοποιήσουμε μόνο μία εντολή όπως η ellipse σαν πρόγραμμα. Αυτή η γραμμή κώδικα σημαίνει "σχεδίασε μια έλλειψη, με το κέντρο 50 pixels από την αριστερή και 50 pixels προς τα κάτω από την κορυφή, με πλάτος και ύψος 80 pixels. Χωρίς την επιλογή της εντολής size η Processing δημιουργεί by default παράθυρο διαστάσεων 100x100 pixels.

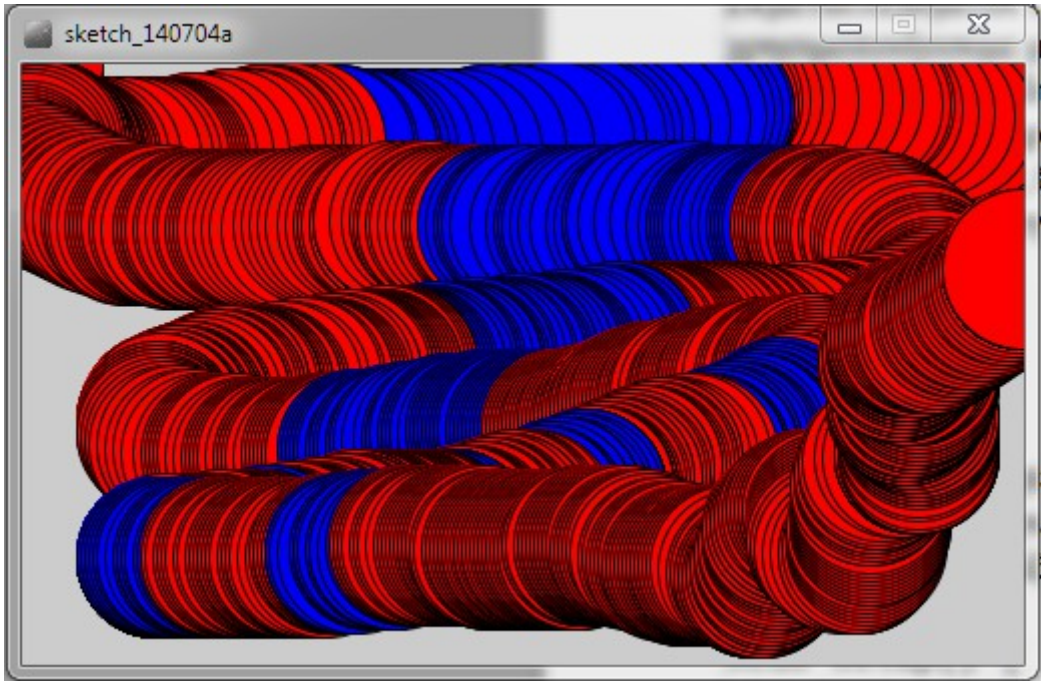


Εικόνα 14: Το αποτέλεσμα της εντολής ellipse(50, 50, 80, 80);

<u>Το πρόγραμμα 2</u>

Το πρόγραμμα 2 σχεδιάζει **με κάθε εκτέλεση της draw (60 φορές/sec)** ένα κόκκινο κύκλο ή έναν μπλε στην περίπτωση που έχουμε πατημένο το κουμπί του ποντικιού, με κέντρο το σημείο που βρίσκεται ο κέρσορας. Τις πληροφορίες για τη θέση βρίσκεται ο κέρσορας τις περνάμε σαν ορίσματα για την τοποθέτηση του κέντρου του κύκλου με τις κωδικοποιημένες λέξεις `mouseX` και `mouseY`.

```
void setup() {
size(500, 300);
smooth();
}
void draw() {
if (mousePressed) {
fill(0, 0, 250);
} else {
fill(255, 0,0);
}
ellipse(mouseX, mouseY, 80, 80);
}
```



Εικόνα 15: Στιγμιότυπο της αναπαραγωγής του Προγράμματος 2

Ανάλυση

Αρχικά όπως αναφέρθηκε τρέχει η setup η οποία με την size δημιουργεί ένα παράθυρο με διαστάσεις 500 pixels οριζόντια και 300 pixels κάθετα.

`Smooth();`
Η smooth σχεδιάζει όλες τις οντότητες-σχήματα με λείες ομαλοποιημένες (anti-aliased) ακμές και γωνίες. Βελτιώνει επίσης την ποιότητα της εικόνας όταν έχει αλλάξει μέγεθος. Σημειώστε ότι η smooth() είναι ενεργή από προεπιλογή. Η noSmooth() μπορεί να χρησιμοποιηθεί για να απενεργοποιήσετε την εξομάλυνση της γεωμετρίας, τις εικόνες και τις γραμματοσειρές.

`mousePressed`
Το περιεχόμενο στη συνθήκη της if είναι η κωδικοποιημένη μεταβλητή τύπου Boolean mousePressed. Επιστρέφει true μόνο στην περίπτωση που πατηθεί το κουμπί του mouse.

`fill();`
Η fill ορίζει το χρώμα που χρησιμοποιείται για να γεμίσουν τα σχήματα. Για παράδειγμα, αν εκτελεστεί το fill(204, 102, 0) όλα τα επόμενα σχήματα θα γεμίσουν με πορτοκαλί χρώμα. Αυτό το χρώμα είτε καθορίζεται από το σύστημα χρώματος RGB ή HSB ανάλογα με την τρέχουσα colorMode ( ). Το προεπιλεγμένο σύστημα χρώματος είναι το RGB, με κάθε τιμή βασικολυ χρώματατος στην περιοχή από 0 έως 255.

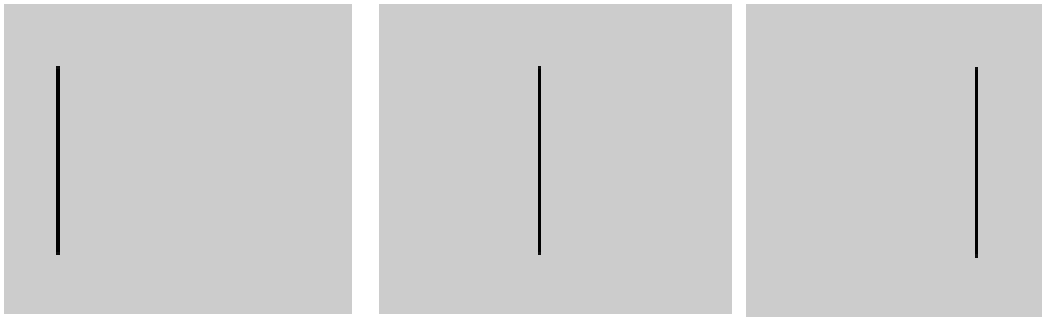
Έτσι σύμφωνα με την if, θα εκτελείται η `fill(0, 0, 250);` σε περίπτωση που πατηθεί το κουμπί του mouse ή αλλιώς η `fill(255, 0,0);` δίνοντας μπλε ή κόκκινο χρώμα για γέμισμα αντίστοιχα.

`mouseX` και `mouseY`
Πρόκειται για κωδικοποιημένες μεταβλητές συστήματος που επιστρέφουν τις οριζόντιες και κάθετες συντεταγμένες της θέσης του κέρσορα.

Το Πρόγραμμα 3

Το πρόγραμμα αυτό σχεδιάζει μία γραμμή στα αριστερά του παραθύρου. Σε κάθε frame δηλαδή κάθε φορά που εκτελείται η draw, η γραμμή επανασχεδιάζεται 1 pixel πιο δεξιά μέχρι να φτάσει το όριο του παράθυρου όπου τότε επιστρέφει στην αρχική της θέση. Αύτο δίνει την αίσθηση ότι η γραμμή κινείται με ταχύτητα ανάλογη του ακέραιου που δίνεται σαν όρισμα στην εντολή frameRate().
Σημειώνεται ότι η width αποτελεί και αυτή μία κωδικοποιημένη λέξη της Processing και δίνει το πλάτος του παράθυρου της εφαρμογής. Όπως επίσης η height αντίστοιχα με το ύψος.

```
void setup() {
  frameRate(50);
}
int pos = 0;
void draw() {
  background(204);
  pos++;
  line(pos, 20, pos, 80);
  if (pos > width) {
    pos = 0;
  }
}
```



Εικόνα 15: Τρία στιγμιότυπα της αναπαραγωγής του Προγράμματος 3

4. Ανάλυση λογισμικού

Σε αυτό το κεφάλαιο θα αναλυθεί το λογισμικό που αναπτύχθηκε για την παρούσα πτυχιακή εργασία. Συγκεκριμένα θα αναλυθούν σε βάθος οι εφαρμογές RGB-Image, Depth-Image και η κύρια εφαρμογή της εργασίας "Αόρατα Κρουστά". Εφαρμογές που απαιτούν την συσκευή Kinect και τις βιβλιοθήκες της Processing SimpleOpenNI και Minim.

4.1 Η εφαρμογή RGB-Image

Η εφαρμογή RGB-Image στην ουσία χρησιμοποιεί την βιβλιοθήκη SimpleOpenNI για να ελέγχει το Kinect και στη συνέχεια να αναπαράγει ότι καταγράφεται με την RGB κάμερα του. Ακόμα με την συνάρτηση mousePressed και την εντολή save δίνεται η δυνατότητα να αποθηκεύονται στιγμιότυπα της ροής βίντεο που παράγεται.

Εικόνα 5: RGB Image

```
import SimpleOpenNI.*;
SimpleOpenNI kinect;
int x=0;
void setup()
{
size(640, 480);
kinect = new SimpleOpenNI(this);
kinect.enableRGB();
}
void draw()
```

```
{
kinect.update();
image(kinect.rgbImage(), 0, 0);
}
void mousePressed(){
save("Image"+x+".png");
x+=1;
}
```

Ανάλυση

`import SimpleOpenNI.*;`
Σε αυτή την σειρά δηλώνεται ότι θα χρησιμοποιηθεί η βιβλιοθήκη SimpleOpenNI και η Processing φορτώνει όλα τα σχετικά δεδομένα.

`SimpleOpenNI kinect;`
Σε αυτή την σειρά δηλώνεται ένα αντικείμενο τύπου SimpleOpenNI το οποίο το ονομάζουμε kinect. Στη συνέχεια στη γραμμή `kinect = new SimpleOpenNI(this);` δημιουργείται ένα στιγμιότυπο του αντικειμένου πάνω στο οποίο γίνονται όλες οι προγραμματιστικές ενέργειες που σχετίζονται με την συσκευή Kinect.

`kinect.enableRGB();`
Αμέσως μετά με την εντολή αυτή ενεργοποιείται η λειτουργία της RGB κάμερας του αντικειμένου kinect.

`kinect.update();`
Στη συνέχεια μέσα στην draw πλέον με την εντολή kinect.update(); πραγματοποιείται η ενημέρωση της ροής πληροφοριών του αντικειμένου kinect. Δηλαδή κάθε φορά που τρέχει η draw και εκτελείται η εντολή αυτή, η Processing ζητάει από το Kinect να στείλει νέο frame και άλλες πληροφορίες που έχει εκ νέου καταγράψει.

`image(kinect.rgbImage(), 0, 0);`
Γενικά η λειτουργία της συνάρτηση image είναι να σχεδιάζει μια εικόνα στο παράθυρο της οθόνης που παράγει η εφαρμογή. Οι εικόνες πρέπει να είναι στο κατάλογο "data" για να φορτωθούν σωστά. Επιλέξτε "Add file ..." από το μενού "Sketch" για να προσθέσετε την εικόνα στον κατάλογο των δεδομένων, ή απλά να σύρετε το αρχείο εικόνας στο παράθυρο του sketch. Η Processing λειτουργεί με GIF, JPEG και PNG αρχεία εικόνων.
Η πρώτη παράμετρος καθορίζει το όνομα της εικόνας ενώ η δεύτερη και τρίτη παράμετρος καθορίζουν τη θέση της επάνω αριστερής γωνία της εικόνας στο παράθυρο. Η εικόνα εμφανίζεται στο αρχικό της μέγεθος, εκτός εάν προστεθούν τέταρτη και πέμπτη παράμετρος οι οποίες καθορίζουν διαφορετικό μέγεθος.
Σε αυτήν την εφαρμογή η image παίρνει σαν παράμετρο για όνομα της εικόνας το `kinect.rgbImage()` το οποίο είναι το αποτέλεσμα μίας συνάρτησης μέσα στο αντικείμενο kinect που επιστρέφει την παρούσα εικόνα-frame η οποία καταγράφεται με την RGB κάμερα. Στην συνέχεια με τις επόμενες 2 παραμέτρους να είναι 0 η εικόνα τοποθετείται με τη αριστερή πάνω γωνία της στο σημείο (0,0)

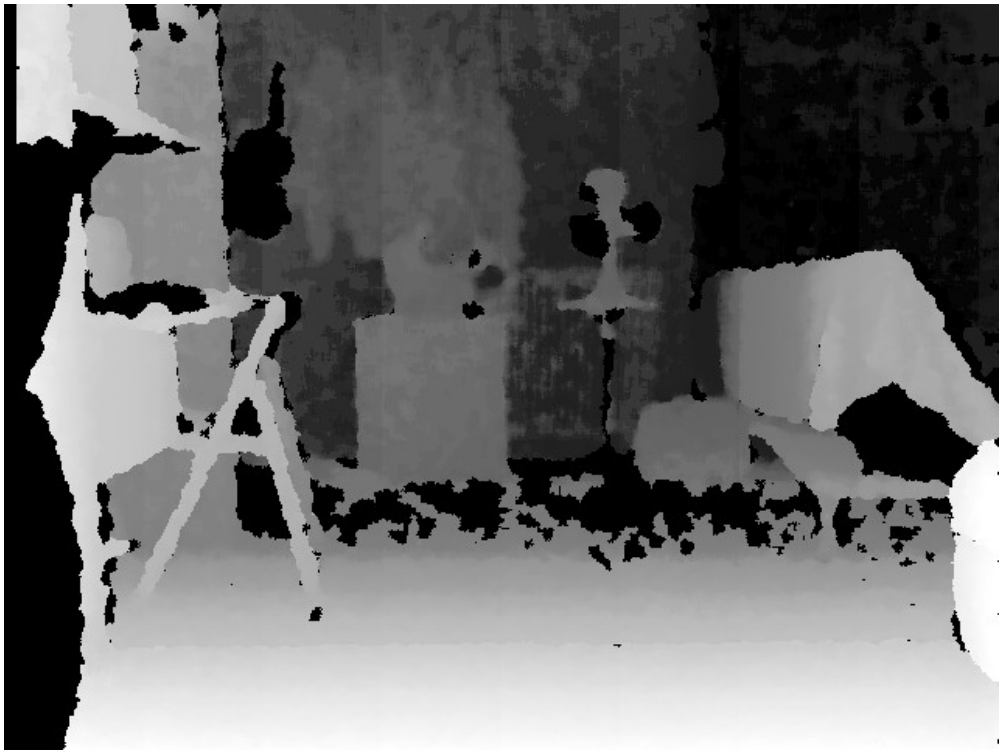
Τέλος με την συνάρτηση mousePressed και την εντολή save δίνεται η δυνατότητα να αποθηκεύονται στιγμιότυπα της ροής βίντεο που παράγεται. Η αποθήκευση εφαρμόζεται σε κάθε click του mouse με το όνομα `Image"x".png` όπου x η εξωτερική μεταβλητή ακέραιου τύπου που αυξάνεται κατά 1 σε κάθε κλήση.

4.2 Η εφαρμογή Depth-Image

Η εφαρμογή Depth-Image χρησιμοποιεί και αυτή την βιβλιοθήκη SimpleOpenNI για να ελέγχει το Kinect και στη συνέχεια να αναπαράγει ότι καταγράφεται από την Depth κάμερα.

```
import SimpleOpenNI.*;
SimpleOpenNI kinect;
int x=0;
void setup()
{
size(640, 480);
kinect = new SimpleOpenNI(this);
kinect.enableDepth();
}
void draw()
{
kinect.update();
image(kinect.depthImage(), 0, 0);
}

void mousePressed(){
save("Image"+x+".png");
x+=1;
}
```



Εικόνα 6: Depth Image

Η μόνη διαφορά στον κώδικα με την προηγούμενη εφαρμογή είναι ότι αντί για

`kinect.enableRGB();` έχουμε `kinect.enableDepth();` και αντί `kinect.RGBImage()` έχουμε `kinect.depthImage()`. Με αυτό το τρόπο αποκτάμε πρόσβαση στην εικόνα βάθους που παρέχει το Kinect μέσω της υπέρυθρης depth κάμερας που διαθέτει. Έτσι όπως αναφέρθηκε στον ορισμό του Depth-image, με αυτήν την κάμερα το Kinect μας δείχνει όχι πώς φαίνονται τα αντικείμενα με τα χρώματά τους αλλά που βρίσκονται στο χώρο με συντεταγμένες τριών διαστάσεων. Το depth-Image περιέχει μόνο αποχρώσεις του γκρι. Συγκεκριμένα όσο πιο μακρυά από την κάμερα βρίσκεται ένα σημείο της εικόνας τόσο πιο σκούρο γκρι θα είναι και αντίστροφα. Στην περίπτωση που κάποιο σημείο απεικονίζεται με μαύρο χρώμα τότε ισχύει τουλάχιστον ένα από τα παρακάτω:
την περίπτωση που κάποιο σημείο απεικονίζεται με μαύρο χρώμα τότε ισχύει τουλάχιστον ένα από τα παρακάτω. Το σημείο βρίσκεται πολύ κοντά στην κάμερα σε ακτίνα μικρότερη των 50cm οι οποία είναι η ελάχιστη απόσταση που μπορεί καταγράψει το Kinect for Windows. Το σημείο βρίσκεται πολύ μακρυά από την κάμερα σε ακτίνα μεγαλύτερη των 4,5m οι οποία είναι η μέγιστη απόσταση που μπορεί καταγράψει το Kinect. Το σημείο δεν επιστρέφει πίσω το υπέρυθρο στίγμα που εκπέμπει ο αισθητήρας του Kinect. Αυτό συμβαίνει επειδή λόγω θορύβου, περίπλοκων γωνιών και λεπτομερειών, η αντανάκλαση των υπεριώδων ακτίνων σε εκείνα τα σημεία στρέφεται εκτός της ανιχνευτικής δυνατότητας του Kinect.
Άρα η πληροφορία για την θέση του σημείου χάνεται σε αυτές τις περιπτώσεις με αποτέλεσμα να εμφανίζεται με μαύρο χρώμα στο depth-Image. Η εφαρμογή depthImage είναι το πιο απλό πρόγραμμα Processing το οποίο καταγράφει και αναπαράγει στην οθόνη του υπολογιστή αυτό που “βλέπει“ το Kinect με την depth κάμερα.

4.3 Η εφαρμογή Αόρατα Κρουστά

Η εφαρμογή αόρατα κρουστά αποτελείται από το αρχείο πηγαίου κώδικα Processing DrumsSimulation και από τρία αρχεία κλάσεων τα οποία ονομάζονται highSigma mediumStigma και lowStigma. Γενικά το βασικό αρχείο δηλώνει τις απαραίτητες βιβλιοθήκες και τα αντικείμενα που σχετίζονται με αυτές, σχεδιάζει και καθορίζει τα χαρακτηριστικά αναπαραγωγής της ροής βίντεο του depth-image, προσφέρει έλεγχο πλοήγησης μέσα στον τρισδιάστατο χώρο τηε εφαρμογής και καλεί τις κλάσεις των αντικειμένων διαφόρων τύπου “Στίγματος” με τις κατάλληλες παραμέτρους. Από την άλλη κάθε αρχείο κλάσης highSigma, mediumStigma και lowStigma περιέχει τις ιδιότητες και μεθόδους που χρειάζονται για να δημιουργηθεί σε κάθε κλήση ένα κομμάτι στον τρισδιάστατο χώρο (το κομμάτι εκείνο το ονομάζουμε στίγμα) το οποίο αντιδρά προγραμματιστικά κάθε φορά που κάποιο υλικό ή κάποιο μέλος του χρήστη περάσει μέσα από τα στοιχειοθετημένα όριά του. Όλο αυτό γίνεται δυνατό χάρη στην τεχνολογία του Kinect. Για κάθε αντίδραση στίγματος χρησιμοποιείται και αναπαράγεται με την Minim ένα αρχείο ήχου τύπου wav. Ανάλογα τη θέση και τα χαρακτηριστικά του στίγματος αναπαράγεται και το σχετικό αρχείο ήχου όπως για παράδειγμα μπότα, κύμβαλο κτλ.
Εν κατακλείδι προγραμματίζοντας με την τεχνολογία Kinect, τη γλώσσα Processing και τις βιβλιοθήκες SimpleOpenNI δημιουργήσαμε αόρατα κρουστά τα οποία μπορεί να ελέγχει ο χρήστης με σχετική ακρίβεια, εύκολα και διασκεδαστικά. Παράλληλα ο προγραμματιστής μελετώντας την εφαρμογή αποκτά βασικές γνώσεις σχετικά με τις νέες διαδραστικές δυνατότητες που είναι σε θέση να προσφέρει το Kinect και η Processing.

To Processing αρχείο DrumsSimulation

```
import processing.opengl.*;
import SimpleOpenNI.*;
import ddf.minim.*;
SimpleOpenNI kinect;
// AudioSnippet objects
Minim minim;
AudioSnippet hihat,ride,crash;
AudioSnippet htom;
AudioSnippet mtom;
AudioSnippet kick,kick2;
AudioSnippet snare;
// declare our Stigmas objects
highStigma hihatArea,rideArea,crashArea;
mediumStigma snareArea,htomArea,mtomArea;
lowStigma kickArea,kickArea2;
float s = 1;
float rotationX=0;
float rotationY=0;
void setup() {
size(1024, 768, OPENGL);
frameRate(30); //kinect's frame rate at defaut is 30 so we choose the
same value for the sketch
kinect = new SimpleOpenNI(this);
kinect.enableDepth();
minim = new Minim(this);
// load all audio files
mtom = minim.loadSnippet("mediumtom.wav");
htom = minim.loadSnippet("hightom.wav");
snare = minim.loadSnippet("snare.wav");
kick = minim.loadSnippet("kick.wav");
kick2 = minim.loadSnippet("kick2.wav");
hihat = minim.loadSnippet("hihat.wav");
crash = minim.loadSnippet("crash.wav");
ride = minim.loadSnippet("ride.wav");
// initialize Stigmas with their origins (x,y,z) and their size
htomArea= new mediumStigma(-150, -100, 1600, 200);
mtomArea= new mediumStigma(150, -100, 1600, 200);
hihatArea = new highStigma(-370, 200, 1800, 150);
rideArea = new highStigma(370, 200, 1800, 150);
crashArea = new highStigma(70, 400, 1600, 170);
snareArea = new mediumStigma(0, -300, 1800, 200);
kickArea = new lowStigma(350, -580, 1500, 100);
kickArea2 = new lowStigma(-350, -580, 1500, 100);
}
void draw() {
background(0);
kinect.update();
```

```
translate(width/2, height/2, -1000);
rotateX(radians(180)); //mirror effect
 // disable mirror maybe but in setup
 //context.setMirror(false);
translate(0, 0, 1400);
rotateX(radians(rotationX));
rotateY(radians(rotationY));
//rotateX(radians(map(mouseY, 0, height, 0, 360))); //mouse navigator
rotateY(radians(map(mouseX, 0, width, 0, 360)));
translate(0, 0, s*-1000);
scale(s);
stroke(255);
PVector[] depthPoints = kinect.depthMapRealWorld();
for (int i = 0; i < depthPoints.length; i+=30) { //10 20 40
PVector currentPoint = depthPoints[i];
// have each Stigma check to see if it includes the currentPoint
mtomArea.check(currentPoint);
htomArea.check(currentPoint);
hihatArea.check(currentPoint);
rideArea.check(currentPoint);
crashArea.check(currentPoint);
snareArea.check(currentPoint);
kickArea.check(currentPoint);
kickArea2.check(currentPoint);
point(currentPoint.x, currentPoint.y, currentPoint.z);
}
//println(kickArea2.pointsIncluded);
if(rideArea.isHit()) {
ride.play();
}
if(!ride.isPlaying()) {
ride.rewind();
}
if(crashArea.isHit()) {
crash.play();
}
if(!crash.isPlaying()) {
crash.rewind();
}
if(htomArea.isHit()) {
htom.play();
}
if(!htom.isPlaying()) {
htom.rewind();
}
if(mtomArea.isHit()) {
mtom.play();
}
if(!mtom.isPlaying()) {
mtom.rewind();
}
```

```
if(snareArea.isHit()) {
snare.play();
}
if(!snare.isPlaying()) {
snare.rewind();
}
if (kickArea.isHit()) {
kick.play();
}
if(!kick.isPlaying()) {
kick.rewind();
}
if (kickArea2.isHit()) {
kick2.play();
}
if(!kick2.isPlaying()) {
kick2.rewind();
}
if (hihatArea.isHit()) {
hihat.play();
}
if(!hihat.isPlaying()) {
hihat.rewind();
}
//if (hihatArea.isHit()) {
//hihat.stop();
//hihat.rewind();
//hihat.play();
//}
// display each mediumStigma and clear its points
hihatArea.draw();
hihatArea.clear();
snareArea.draw();
snareArea.clear();
kickArea.draw();
kickArea.clear();
kickArea2.draw();
kickArea2.clear();
mtomArea.draw();
mtomArea.clear();
htomArea.draw();
htomArea.clear();
crashArea.draw();
crashArea.clear();
rideArea.draw();
rideArea.clear();
}
void stop()
{
//close all AudioPlayer objects
ride.close();
```

```
crash.close();
mtom.close();
htom.close();
hihat.close();
kick.close();
kick2.close();
snare.close();
minim.stop();
super.stop();
}
void keyPressed() {
if(key == 'a' || key == 'A'){
 s = s + 0.02;
}
if(key == 'z' || key == 'Z'){
 s = s - 0.02;
}
if (keyCode == 40) {
rotationX+=5;
}
if (keyCode == 38) {
rotationX-=5;
}
if(keyCode == RIGHT ){
    rotationY+=5;
}
if(keyCode == LEFT ){
    rotationY-=5;
  }
}
```

Ανάλυση

```
import processing.opengl.*;
import SimpleOpenNI.*;
import ddf.minim.*;
```

Αρχικά σε αυτές τις τρεις πρώτες γραμμές δηλώνουμε της βιβλιοθήκες που θα χρησιμοποιηθούν στην εφαρμογή. Η `opengl` είναι η βιβλιοθήκη που είναι υπεύθυνη για την τρισδιάστατη απεικόνιση των γραφικών. Η επόμενες δύο που ακολουθούν όπως ήδη έχουμε αναφέρει είναι η `SimpleOpenNI` η οποία προσφέρει έλεγχο και άντληση δεδομένων από την συσκευή Kinect και η `Minim` η οποία είναι υπεύθυνη για τον ήχο.

```
SimpleOpenNI kinect;
```

Εδώ δηλώνουμε το αντικείμενο kinect το οποίο είναι τύπου `SimpleOpenNI.` Πάνω σε αυτό το αντικείμενο πραγματοποιείται κάθε προγραμματιστική αλληλεπίδραση με την συσκευή Kinect.

```
Minim minim;
```

Στη συνέχεια δηλώνουμε το αντικείμενο με όνομα minim το οποίο είναι τύπου `Minim.` Πάνω σε αυτό το αντικείμενο πραγματοποιείται κάθε προγραμματιστική αλληλεπίδραση με την βιβλιοθήκη ήχου Minim.

```
AudioSnippet hihat,ride,crash;
AudioSnippet htom;
AudioSnippet mtom;
AudioSnippet kick,kick2;
AudioSnippet snare;
```
Σε αυτές τις σειρές δηλώνουμε τα αντικείμενα τύπου AudioSnippet τα οποία θα συνδέσουμε στην συνέχεια με 8 αρχεία ήχου. Έτσι έχουμε 8 διαφορετικά αρχεία που θα απαρτίζουν τα κρουστά. Τα αντικείμενα τύπου AudioSnippet υπάγονται στο αντικείμενο minim.

```
highStigma hihatArea,rideArea,crashArea;
mediumStigma snareArea,htomArea,mtomArea;
lowStigma kickArea,kickArea2;
```
Σε αυτές τις σειρές δηλώνουμε τα αντικείμενα τύπου highStigma, mediumstigma, και lowstigma. Πρόκειται για αντικείμενα που δημιουργούν οι τρεις κλάσεις με τα ίδια ονόματα αρχείων που έχουμε δημιουργήσει και περιέχουν τις ιδιότητες και μεθόδους που χρειάζονται για να δημιουργηθεί σε κάθε κλήση ένα κομμάτι στον τρισδιάστατο χώρο (το κομμάτι εκείνο το ονομάζουμε στίγμα) το οποίο αντιδρά προγραμματιστικά κάθε φορά που κάποιο υλικό ή κάποιο μέλος του χρήστη περάσει μέσα από τα στοιχειοθετημένα όριά του.

```
float s = 1;
float rotationX=0;
float rotationY=0;
```
Δηλώνουμε τρεις εξωτερικές μεταβλητές με ονόματα s, rotationX και rotationY τύπου float και τις αρχικοποιούμε με τις ανάλογες τιμές. Αυτές οι μεταβλητές θα βοηθήσουν στην πλοήγηση μέσα στον τρισδιάστατο χώρο στην συνέχεια.

Μέσα στην συνάρτηση setup() :

```
size(1024, 768, OPENGL);
```
Η size εδώ δέχεται σαν όρισμα τρεις παραμέτρους, οι πρώτες δύο καθορίζουν το μέγεθος του παράθυρου της εφαρμογής. Δηλαδή 1024x768pixels σε πλάτος και ύψος. Η τρίτη παράμετρος είναι μία κωδικοποιημένη λέξη της βιβλιοθήκης OpenGL, υπεύθυνη για τα τρισδιάστατα γραφικά και δηλώνει ότι το παράθυρο που θα δημιουργηθεί στην οθόνη θα απεικονίζει σε τρισδιάστατη μορφή τα περιεχόμενά του.

```
frameRate(30);
```
Εδώ θέτουμε το ρυθμό ή την συχνότητα καρέ σε 30. Δηλαδή η processing θα τρέχει την draw και θα σχεδιάζει 30 φορές κάθε δευτερόλεπτο. Η τιμή 30fps επιλέγεται γιατί είναι ίση με την αντίστοιχη τιμή καταγραφής της συσκευής Kinect.

```
kinect = new SimpleOpenNI(this);
kinect.enableDepth();
```
Στη συνέχεια δημιουργείται ένα στιγμιότυπο του αντικειμένου kinect και καλείται η μέθοδός του enableDepth που ενεργοποιεί το σύστημα ανίχνευσης βάθους του Kinect για να χρησιμοποιηθεί στη συνέχεια από την εφαρμογή.

```
minim = new Minim(this);
```
Παρόμοια εδώ δημιουργείται ένα στιγμιότυπο του αντικειμένου minim το οποίο θα διαχειριστεί τα αντικείμενα audioSnippets δηλαδή τα αρχεία ήχου.

```
mtom = minim.loadSnippet("mediumtom.wav");
htom = minim.loadSnippet("hightom.wav");
```

```
snare = minim.loadSnippet("snare.wav");
kick = minim.loadSnippet("kick.wav");
kick2 = minim.loadSnippet("kick2.wav");
hihat = minim.loadSnippet("hihat.wav");
crash = minim.loadSnippet("crash.wav");
ride = minim.loadSnippet("ride.wav");
```
Με αυτές τις εντολές δημιουργούμε τα αντικείμενα audioSnippets που είχαμε δηλώσει πριν και τα αντιστοιχούμε με τα σχετικά wav αρχεία. Τα ονομάζουμε ακριβώς με τον ίδιο τίτλο που έχουν τα αρχεία κρουστών . Τα αρχεία wav πρέπει να βρίσκονται μέσα στο φάκελο της εφαρμογής.

```
htomArea= new mediumStigma(-150, -100, 1600, 200);
mtomArea= new mediumStigma(150, -100, 1600, 200);
hihatArea = new highStigma(-370, 200, 1800, 150);
rideArea = new highStigma(370, 200, 1800, 150);
crashArea = new highStigma(70, 400, 1600, 170);
snareArea = new mediumStigma(0, -300, 1800, 200);
kickArea = new lowStigma(350, -580, 1500, 100);
kickArea2 = new lowStigma(-350, -580, 1500, 100);
```
Στο σημείο αυτό με τις παραπάνω σειρές κώδικα δημιουργούμε 8 αντικείμενα των κλάσεων highStigma, mediumStigma, και lowStigma. Κάθε αντικείμενο δημιουργείται από τον constructor της κάθε κλάσης παίρνοντας 4 παραμέτρους. Οι πρώτες τρεις είναι τύπου float και συμβολίζουν τις συντεταγμένες του σημείου που θα κατασκευαστεί το εκάστοτε αντικείμενο. Η τέταρτη παράμετρος είναι τύπου integer και συμβολίζει το μέγεθος του αντικειμένου. Περισσότερα για την χρήση και σημασία αυτών των παραμέτρων θα παρουσιαστούν στην ανάλυση των αρχείων των κλάσεων highStigma, mediumStigma, και lowStigma.

Μέσα στην συνάρτηση draw() :

```
background(0);
```
Θέτουμε ως μαύρο το χρώμα του φόντου του παράθυρου.

```
kinect.update();
```
Εδώ πραγματοποιείται η ενημέρωση της ροής πληροφοριών του αντικειμένου kinect. Δηλαδή κάθε φορά που τρέχει η draw και εκτελείται η εντολή αυτή, η Processing ζητάει από το Kinect να στείλει νέο frame και άλλες πληροφορίες που έχει εκ νέου καταγράψει.

```
translate(width/2, height/2, -1000);
```
Με αυτήν την βασική εντολή μεταφέρουμε το σημείο σχεδίασης σχημάτων-αντικειμένων σε όποιο άλλο σημείο στον τρισδιάστατο χώρο της εφαρμογής επιθυμούμε. Συγκεκριμένα σύμφωνα με αυτές τις παραμέτρους το σημείο σχεδίασης θα μεταφερθεί κατά 1024/2=512 pixels πιο δεξιά ως προς το πλάτος, κατά 768/2=384px ως προς το ύψος αφού width=1024 και height=768 και κατά -1000px ως προς το βάθος. Με το βάθος να παίρνει αρνητικές τιμές δημιουργείται η αίσθηση ότι το επόμενο αντικείμενο θα σχεδιαστεί κατά 1000 pixels πιο μακρυά από τον χρήστη που παρατηρεί το παράθυρο στην οθόνη.

```
rotateX(radians(180));
```
Όταν εκτελείτε αυτή τη γραμμή, έχουμε περιστροφή στο σύστημα συντεταγμένων κατά 180 μοίρες γύρω από τον άξονα του x. Το αποτέλεσμα είναι ότι ο άξονας του y περιστρέφεται προς την αντίθετη κατεύθυνση. Οι θετικές τιμές κατά μήκος αυτού του άξονα, ο οποίος προηγουμένως ήταν γυρισμένος προς τα κάτω στον προεπιλεγμένο προσανατολισμό επεξεργασίας, τώρα συνεχίζουν με φορά προς τα 'πάνω, φορά η οποία ταιριάζει με την προβολή του πραγματικού κόσμου που δίνεται σε μας από την βιβλιοθήκη SimpleOpenNI και της συνάρτησης depthMapRealWorld που θα δούμε

παρακάτω.

```
translate(0, 0, 1400);
```
Σημειώνεται ότι όλες οι μετατοπίσεις και οι περιστροφές που πραγματοποιούνται λειτουργούν αθροιστικά. Έτσι εδώ μετατοπίζουμε το σημείο σχεδίασης ή την οπτική απόσταση του χρήστη από την σκηνή της εφαρμογής κατά 1400 μονάδες ως προς το βάθος από το σημείο που είχε μεταφερθεί πριν. Δηλαδή μεταφέρουμε το σύστημα συντεταγμένων κατά 1400 πιο κοντά σε μας ως προς τον άξονα z, σαν να κάνουμε zoom in στην σκηνή της εφαρμογής μας.

```
rotateX(radians(rotationX));
```
Με αυτή την εντολή παίρνουμε την τιμή της μεταβλητής rotationX και την μετατρέπουμε σε rads με την συνάρτηση radians(). Στη συνέχεια πραγματοποιείται περιστροφή στο σύστημα συντεταγμένων κατά την τιμή rads της μεταβλητής rotationX γύρω από τον άξονα του x. Όπως θα δούμε παρακάτω την τιμή της rotationX την αυξομειώνουμε ανάλογα με το πάτημα κουμπιών που ορίζουμε με αποτέλεσμα έτσι να ελέγχουμε την περιστροφή του άξονα X κατά την διάρκεια εκτέλεσης της εφαρμογής.

```
rotateY(radians(rotationY));
```
Εδώ πραγματοποιείται η ίδια διαδικασία της προηγούμενης εντολής αλλά ως προς τον άξονα του Y αυτή τη φορά με την σχετική μεταβλητή `rotationY.`

```
rotateY(radians(map(mouseX, 0, width, 0, 360)));
```
Σε αυτήν την σειρά έχουμε τρεις συναρτήσεις. Η πρώτη που εκτελείτε είναι η map. Δέχεται 5 παραμέτρους και κάνει το εξής: Παίρνει την τιμή της μεταβλητής που περνάει σαν πρώτη παράμετρος. Στην δεύτερη και τρίτη παράμετρος παίρνει το εύρος τιμών της μεταβλητής 0-width. Στη συνέχεια με τις παραμέτρους 4 και 5 δέχεται ένα άλλο εύρος τιμών, 0-360. Σύμφωνα με τις παραμέτρους που δέχτηκε η map επιστρέφει πίσω την ανάλογη τιμή που θα είχε η μεταβλητή `mouseX` στο εύρος τιμών 0-360. Αμέσως μετά η τιμή που επιστρέφει η map μετατρέπεται σε rads και πραγματοποιείται περιστροφή ως προς τον άξονα Y. Το αποτέλεσμα είναι ότι με αυτήν την γραμμή δίνουμε δυνατότητα στον χρήστη να περιστρέφει ως προς τον άξονα Y την σκηνή του προγράμματος μετακινώντας τον ποντίκι δεξιά ή αριστερά.

```
translate(0, 0, s*-1000);
scale(s);
```
Η scale() μας δίνει τον έλεγχο μιας άλλης παραμέτρου: την απόσταση μεταξύ κάθε υποδιαίρεσης σε κάθε άξονα. Αν καλέσουμε την scale με αριθμό μεγαλύτερο από ένα, αυξάνουμε την κλίμακα των αξόνων μας. Για παράδειγμα, αν εκτελέσουμε scale(2); μετά από αυτό, κάθε translate και rotate που κάνουμε θα έχει το διπλάσιο της αποτέλεσμα: καλώντας translate (0,0,50), θα προχωρήσουμε μπροστά μας 100 μονάδες κατά μήκος του άξονα z, όπως μετριόταν στο σύστημα συντεταγμένων πριν από το scale(). Φανταστείτε τις τιμές που πέρασαν στην scale ως ποσοστό. Ένα μέσο για να μην αλλάξει τίποτα είναι να κρατήσει την κλίμακα σε 100%. Για αλλαγή σε 1,5 πρέπει να αυξηθεί η τιμή στο scale κατά 50%, και αντίστροφα με τιμή 0.8 μέσα στο scale μειώνεται η κλίμακα κατά 20%. Δεδομένου ότι με τόσο μικρές τιμές της παράμετρος γίνονται τόσο μεγάλες αλλαγές στην κλίμακα υποδιαιρέσεων των αξόνων, θα πρέπει να δίνεται στο χρήστη λεπτός έλεγχος για αυτή την τιμή. Για να γίνει αυτό θα χρησιμοποιηθεί η keyPressed: Κάθε φορά που ο χρήστης πατά το πλήκτρο A,a , η αξία που περνάμε στην κλίμακα θα αυξάνεται κατά 0,01 αυξάνοντας την ποσότητα του scale από ένα 1%. Και αντίστροφα με το πλήκτρο Z,z. Με αυτό το τρόπο δίνουμε τη δυνατότητα στον χρήστη να κάνει zoom-in και zoom-out στην σκηνή της εφαρμογής.

```
Stroke(255);
```
Η stroke όπως έχουμε ήδη αναφέρει ορίζει το χρώμα που χρησιμοποιείται για να σχεδιαστούν

γραμμές και περιγράμματα γύρω από σχήματα. Στην συγκεκριμένη περίπτωση επιλέγουμε το άσπρο χρώμα στης επόμενες περιπτώσεις σχεδίασης.

```
PVector[] depthPoints = kinect.depthMapRealWorld();
```
Σε αυτό το κομβικό σημείο καλούμε την μέθοδο depthMapRealWorld του αντικειμένου kinect η οποία μας επιστρέφει τις τιμές συντεταγμένων των σημείων που καταγράφει ο αισθητήρας βάθους του Kinect και τις τοποθετούμε σε ένα πίνακα Vectors που ονομάζουμε depthPoints.

```
for (int i = 0; i < depthPoints.length; i+=30) {
PVector currentPoint = depthPoints[i];
mtomArea.check(currentPoint);
htomArea.check(currentPoint);
hihatArea.check(currentPoint);
rideArea.check(currentPoint);
crashArea.check(currentPoint);
snareArea.check(currentPoint);
kickArea.check(currentPoint);
kickArea2.check(currentPoint);
point(currentPoint.x, currentPoint.y, currentPoint.z); }
```
Στη συνέχεια μέσα στην for περνάμε τις τιμές των συντεταγμένων του κάθε σημείου που βλέπει το Kinect στο αντικείμενο currentPoint τύπου vector και τις ελέγχουμε με τις μεθόδους check της κάθε κλάσης για το αν βρίσκονται μέσα στην οριοθετημένη περιοχή του κάθε αντικειμένου των κλάσεων highStigma, mediumStigma, και lowStigma. Τέλος με την εντολή point σχεδιάζεται το κάθε σημείο στο παράθυρο της οθόνης περνόντας στην μέθοδο τις τρεις συντεταγμένες του.
Σημειώνεται ότι η διαδικασία αυτή πραγματοποιείται με συχνότητα 1 ανά 30 σημείων από τη depthMapRealWorld και όχι 1 ανά 1. Αυτό γίνεται για να ελαχιστοποιήσουμε την υπολογιστική ισχύ που απαιτείται σε κάθε εκτέλεση της draw, για να μειωθεί έτσι η καθυστέρηση (latency) δράσης-αντίδρασης κάθε προγραμματιστικής ενέργειας της εφαρμογής με τον χρήστη. Το κόστος αυτής της μειωμένης δειγματοληψίας είναι η μειωμένη σε σχεδιασμένα σημεία “ριγέ” απεικόνιση του Depth-Image.

```
if(rideArea.isHit()) {
ride.play();
}
if(!ride.isPlaying()) {
ride.rewind();
}
```
Ακολουθεί στον κώδικα μια σειρά από if όπως οι 2 παραπάνω, για κάθε αντικείμενο των κλάσεων highStigma, mediumStigma, και lowStigma. Η πρώτη if καλεί την μέθοδο isHit(), η οποία ελέγχει αν “κτυπήθηκε” (βλ στον κώδικα της κλάσης) το στιγμιότυπου του αντικειμένου. Αν επιστρέψει true τότε το αντίστοιχο αντικείμενο audiosnippet θα εκτελέσει την μέθοδο play και θα αναπαράγει το wav αρχείο. Η δεύτερη if με την μέθοδο isPlaying ελέγχει αν το συγκεκριμένο audiosnippet ήδη βρίσκεται σε διαδικασία αναπαραγωγής και στη περίπτωση που δεν συμβαίνει αυτό εκτελείται η μέθοδος rewind η οποία επιστρέφει τον δείκτη αναπαραγωγής του συγκεκριμένου ηχητικού αρχείου στην αφετηρία του ώστε την επόμενη φορά που θα εκτελεστεί η μέθοδος play να αναπαραχθεί από την αρχή.

```
hihatArea.draw();
hihatArea.clear();
...
```
Αμέσως μετά καλούνται οι μέθοδοι draw και clear για κάθε στιγμιότυπο αντικειμένου των κλάσεων highStigma, mediumStigma, και lowStigma. Η μέθοδος draw σχεδιάζει το αντικείμενο στην

απεικόνιση του Depth-Image ενώ η clear ανανεώνει την ανίχνευση σημείων του (βλ στον κώδικα της κλάσης).

```
void stop(){
ride.close();
crash.close();
mtom.close();
htom.close();
hihat.close();
kick.close();
kick2.close();
snare.close();
minim.stop();
super.stop();
}
```

Κλείνοντας την συνάρτηση draw() ακολουθεί η απαραίτητη για τον τερματισμό της εφαρμογής, συνάρτηση της Minim, η stop(). Μέσα στην οποία καλούνται η μέθοδος close για όλα τα audiosnippets και η μέθοδος stop για το αντικείμενο τύπου Minim και της super κλάσης του αρχείου με σκοπό να τερματιστεί η χρήση της βιβλιοθήκης Minim και να αποδεσμευτούν οι πόροι του συστήματος που είχαν σχέση με τον ήχο της εφαρμογής.

```
void keyPressed() {
if(key == 'a' || key == 'A'){
 s = s + 0.02;
}
if(key == 'z' || key == 'Z'){
 s = s - 0.02;
}
if (keyCode == 40) {
rotationX+=5;
}
if (keyCode == 38) {
rotationX-=5;
}
if(keyCode == RIGHT ){
   rotationY+=5;
}
if(keyCode == LEFT ){
   rotationY-=5;
  }
```

Τέλος, η συνάρτηση keyPressed περιέχει τους ελέγχους και τις απαραίτητες αλλαγές με την χρήση πολλαπλών if που αφορούν την πλοήγηση του χρήστη μέσα στην τρισδιάστατη αναπαράσταση depth-image της εφαρμογής.
Συγκεκριμένα πατώντας ο χρήστης το A κάνει zoom-in, το Z zoom-out, τα Up και Down(arrows) περιστρέφει τη σκηνή ως προς τον άξονα X ενώ αντίστοιχα με τα Right και Left ως προς τον άξονα Y.

<u>Το Processing αρχείο της κλάσης highStigma</u>

```
class highStigma {
```

```
PVector center;
color fillColor;
color strokeColor;
int size;
int pointsIncluded;
int maxPoints;
boolean wasJustHit;
int threshold;
highStigma(float centerX, float centerY, float centerZ, int boxSize) {
center = new PVector(centerX, centerY, centerZ);
size = boxSize;
pointsIncluded = 0;
maxPoints = 1000;
threshold = 0;
fillColor = strokeColor = color(random(255), random(255), random(255));
}
void setThreshold( int newThreshold ){
threshold = newThreshold;
}
void setMaxPoints(int newMaxPoints) {
maxPoints = newMaxPoints;
}
void setColor(float red, float blue, float green){
fillColor = strokeColor = color(red, blue, green);
}
boolean check(PVector point) {
boolean result = false;
if (point.x > center.x - size/2 && point.x < center.x + size/2) {
if (point.y > center.y - size/2 && point.y < center.y + size/2) {
if (point.z > center.z - size/2 && point.z < center.z + size/2) {
result = true;
pointsIncluded++;
}
}
}
return result;
}
void draw() {
pushMatrix();
translate(center.x, center.y, center.z);
fill(red(fillColor), blue(fillColor), green(fillColor),
255 * percentIncluded());
stroke(red(strokeColor), blue(strokeColor), green(strokeColor), 255);
box(size,size,size);
popMatrix();
}
float percentIncluded() {
return map(pointsIncluded, 0, maxPoints, 0, 1);
}
boolean currentlyHit() {
return (pointsIncluded > threshold);
```

```
}
boolean isHit() {
return currentlyHit() && !wasJustHit;
}
void clear() {
wasJustHit = currentlyHit();
pointsIncluded = 0;
}
}
```

Ανάλυση

```
highStigma(float centerX, float centerY, float centerZ, int boxSize) {
```
Μετά τις δηλώσεις των απαραίτητων μεταβλητών ακολουθεί ο constructor της κλάσης. Αναθέτουμε τις τιμές των παραμέτρων σε εξωτερικές μεταβλητές της κλάσης. Κάθε αντικείμενο δημιουργείται από τον constructor της κάθε κλάσης παίρνοντας 4 παραμέτρους. Οι πρώτες τρεις είναι τύπου float και συμβολίζουν τις συντεταγμένες του σημείου όπου θα κατασκευαστεί το εκάστοτε στιγμιότυπο αντικειμένου. Αποθηκεύονται και οι τρεις στη μεταβλητή center τύπου PVector. Η τέταρτη παράμετρος είναι τύπου integer και συμβολίζει το μέγεθος του αντικειμένου. Περνάμε την τιμή στην μεταβλητή size. Στη συνέχεια αρχικοποιούμε τις μεταβλητές pointsIncluded=0, maxPoints=1000, threshold=0 και τέλος τις fillcolour και strokeColour τους δίνουμε random τιμή χρώματος με την μέθοδο random.

```
void setThreshold( int newThreshold ){
threshold = newThreshold; }
```
Ακολουθεί η μέθοδος setThreshold, η λειτουργία της οποίας είναι να αλλάζει την τιμή της μεταβλητής threshold με την τιμή της παραμέτρου που θα καλεστεί. Η μεταβλητή ακέραιου threshold (int) συμβολίζει τον ελάχιστο αριθμό σημείων που αν βρεθούν μέσα στη τρισδιάστατη οριοθετημένη περιοχή του αντικειμένου κλάσης highStigma, θα ενεργοποιήσουν τις απαραίτητες μεθόδους για να αναπαραχθεί ο αντίστοιχος ήχος, όπως θα δούμε και στο κώδικα παρακάτω. Σαν προεπιλεγμένη τιμή η μεταβλητή threshold παίρνει την τιμή 0 άρα πρακτικά έστω και ένα σημείο από κάποιο μέλος του χρήστη να περάσει μέσα στο χώρο του αντικειμένου της κλάσης highStigma, θα έχουμε μία αναπαραγωγή του wav αρχείου.

```
void setMaxPoints(int newMaxPoints) {
maxPoints = newMaxPoints;
}
```
Η μέθοδος setMaxPoints κάνει ακριβώς ότι και η setThreshold αλλά για την int μεταβλητή maxPoints. Όπως θα δούμε και μέσα στην draw της κλάσης αυτή η μεταβλητή χρησιμοποιείται για να υπολογιστεί ένα ποσοστό των σημείων της σκηνής μέσα στην οριοθετημένη περιοχή του στιγμιότυπου ώστε να χρωματιστεί με το ανάλογο opacity.

```
void setColor(float red, float blue, float green){
fillColor = strokeColor = color(red, blue, green);
}
```
Στην συνέχεια έχουμε την setColor η οποία δίνει στον προγραμματιστή την δυνατότητα να αλλάξει το χρώμα περιεχομένου και περιγράμματος του στιγμιότυπου.
Οι τρεις παραπάνω μέθοδοι παρόλο που δεν καλούνται στο παράδειγμα της εφαρμογής, είναι χρήσιμες και προσδίδουν ευκολία σε περίπτωση περαιτέρω ανάπτυξης του προγράμματος.

```
boolean check(PVector point) {
```

```
boolean result = false;
if (point.x > center.x - size/2 && point.x < center.x + size/2) {
if (point.y > center.y - size/2 && point.y < center.y + size/2) {
if (point.z > center.z - size/2 && point.z < center.z + size/2) {
result = true;
pointsIncluded++;
}
}
}
return result; }
```

Η check είναι η πιο βασική μέθοδος της κλάσης. Η παράμετρος που δέχεται είναι η μεταβλητή currentPoint τύπου PVector από το κύριο αρχείο, στην κλάση την ονομάζουμε point. Αυτή η μεταβλητή όπως αναφέρθηκε περιέχει τις τιμές των συντεταγμένων του κάθε σημείου που ανιχνεύει το Kinect μέσω της συνάρτησης depthMapRealWorld();
Η μέθοδος check ελέγχει αν το κάθε σημείο βρίσκεται μέσα στη τρισδιάστατη οριοθετημένη περιοχή του αντικειμένου κλάσης highStigma και αν αυτό ισχύει τότε προσθέτει μία μονάδα στην ακέραια μεταβλητή pointsIncluded και επιστρέφει τιμή true.

```
void draw() {
pushMatrix();
translate(center.x, center.y, center.z);
fill(red(fillColor), blue(fillColor), green(fillColor),
255 * percentIncluded());
stroke(red(strokeColor), blue(strokeColor), green(strokeColor), 255);
box(size,size,size);
popMatrix();
}
```

Η draw περιέεχει τις εντολές `pushMatrix();` και `popMatrix();` η σημασία τους είναι πως από την σειρά κώδικα της pushMatrix και κάτω ότι εντολές translate,rotate και άλλες εντολές που αλλάζουν τον σύστημα συντεταγμένων θα αναιρεθούν με την εκτέλεση της επόμενης popMatrix. Αυτό είναι απαραίτητο ώστε να μην αλλάζει το σύστημα συντεταγμένων της εφαρμογής κάθε φορά που εκτελείται ο κώδικας της κλάσης.
Μετά την pushMatrix ακολουθεί η translate που με αυτές τις παραμέτρους μεταφέρει την αρχή του συστήματος συντεταγμένων άρα και του σημείο σχεδίασης στο σημείο που προσδιορίζουν οι τιμές της μεταβλητής (PVector) center.
Στην συνέχεια με την fill θέτουμε το χρώμα περιεχόμενου σχημάτων-αντικειμένων ίσο με τον χρώμα που προσδιορίζει η μεταβλητή fillColor τύπου color. Η τέταρτη παράμετρος της fill προσδιορίζει το opacity σύμφωνα με την μέθοδο `percentIncluded()` που θα δούμε πιο κάτω.
Με την stroke πραγματοποιείται η ίδια διαδικασία για το περίγραμμα, με την τύπου color μεταβλητή strokeColor και με opacity ίσο με 255 που είναι το μέγιστο για να φαίνονται τα αντικείμενα ακόμα και όταν το opacity του fill είναι μηδενικό.
Τέλος με την εντολή box δίνοντας την τιμή της μεταβλητής size στις τρεις παραμέτρους σχεδιάζουμε ένα κύβο στο σημείο σχεδίασης (`center.x, center.y, center.z)` με μήκος, ύψος και βάθος ίσο με την τιμή της size.

```
float percentIncluded() {
return map(pointsIncluded, 0, maxPoints, 0, 1);
}
```

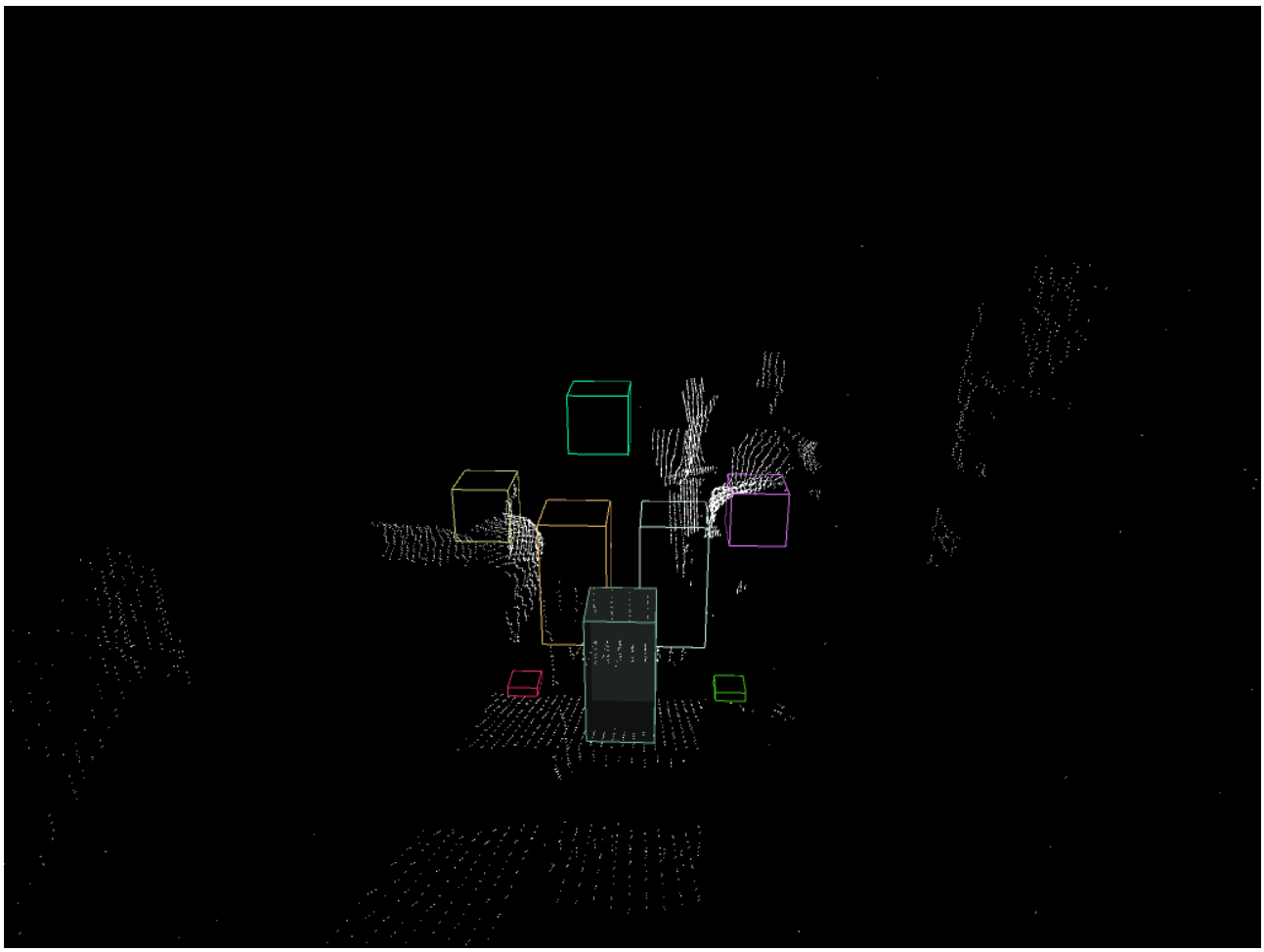
Η μέθοδος percentIncluded() χρησιμοποιεί την τιμή της pointsIncluded, την μεταβάλλει σε ένα ποσοστό σύμφωνα με την maxPoints και επιστρέφει αυτό το ποσοστό συμπεριλαμβανομένων σημείων με μορφή float με σκοπό να προσδιοριστεί το opacity του fill που αναφέραμε στην draw της κλάσης.

```
boolean currentlyHit() {
return (pointsIncluded > threshold);
}
```
Η μέθοδος currentlyHit ελέγχει αν τα σημεία που περιλαμβάνονται στην οριοθετημένη περιοχή του στιγμιότυπου ξεπερνούν το ελάχιστο όριο threshold. Επιστρέφει boolean τιμή. Την μέθοδο αυτή την καλεί η επόμενη μέθοδος, η isHit.

```
boolean isHit() {
return currentlyHit() && !wasJustHit;
}
```
Η μέθοδος isHit() καλείτε από το κύριο αρχείο της εφαρμογής και επιστρέφει το boolean αποτέλεσμα της currentlyHit() λογικό και με το αντίστροφο της τιμής της boolean μεταβλητής wasJustHit. Δηλαδή επιστρέφει true μόνο στην περίπτωση που έχουν εισαχθεί σημεία στο αντικείμενο αλλά δεν βρίσκεται ήδη σε διαδικασία αναπαραγωγής.

```
void clear() {
wasJustHit = currentlyHit();
pointsIncluded = 0;
}
```
Τέλος η μέθοδος clear ανανεώνει την τιμή της wasJustHit και θέτει την τιμή της pointsIncludes ίση με το 0.

Τα άλλα δύο αρχεία της εφαρμογής lowStigma και mediumStigma είναι όμοια σε όλο τον κώδικα με αυτή την κλάση εκτός δύο μικρές διαφορές. Στην check ελέγχεται αν το κάθε σημείο περιλαμβάνεται στο στιγμιότυπο του αντικειμένου για διαφορετικό σχήμα, συγκεκριμένα για διαφορετικό ύψος του σχήματος. Ενώ στη draw η εντολή box που σχεδιάζει το σχήμα του αντικειμένου προσδιορίζει το ύψος ανάλογα την κλάση. Για lowStigma έχουμε για ύψος το 1/3 της size και για mediumStigma το τριπλάσιο της size. Έτσι δεν σχεδιάζονται κύβοι αλλά παραλληλόγραμμα που τηρούν της απαιτήσεις μας. Ο κώδικας των κλάσεων lowStigma και mediumStigma παρατίθεται στο παράρτημα μαζί με τον κώδικα όλων των εφαρμογών που αναπτύχθηκαν για την Πτυχιακή Εργασία.



Εικόνα 17: Ένα στιγμιότυπο του Depth-Image της εφαρμογής "Αόρατα Κρουστά"

5. Συμπεράσματα

Χρησιμοποιώντας την τεχνολογία του Kinect με την γλώσσα Processing δίνεται στον χρήστη η δυνατότητα να ελέγχει εφαρμογές εύκολα με κινήσεις του σώματός του προσφέροντας στον προγραμματιστή ευχέρεια κατά την δημιουργία νέων μορφών διαδραστικότητας. Στην εξελίξιμη κοινότητα της Processing αλλά και στο site της OpenNI υπάρχει όγκος σχετικών εφαρμογών ανοικτού κώδικα που μπορούν να χρησιμοποιηθούν προς αυτήν την κατεύθυνση. Φυσικά υπάρχουν περιορισμοί όπως η υπολογιστική ισχύ που απαιτείται που παράγει καθυστέρηση και τα προσδιορισμένα όρια ανίχνευσης-καταγραφής του αισθητήρα Kinect. Στοιχεία που πρέπει να εξελιχθούν από τους προγραμματιστές και της σχετικές εταιρίες που παράγουν συσκευές όπως το Kinect ώστε να εδραιωθούν στο ευρύ κοινό οι νέες μορφές χειρισμού εφαρμογών και να αντικαταστήσουν με το χρόνο τα συμβατικά μέσα αλληλεπίδρασης.

6. Παραρτήματα

6.1 Ο κώδικας των προγραμμάτων 1,2,3

Το πρόγραμμα 1

```
void setup() {
```

```
        size(100, 100);
        stroke(0);
        background(188);
}

void draw() {
        line(15, 25, 70, 90);
}
```

Το πρόγραμμα 2

```
void setup() {
size(500, 300);
smooth();
}
void draw() {
if (mousePressed) {
fill(0, 0, 250);
} else {
fill(255, 0,0);
}
ellipse(mouseX, mouseY, 80, 80);
 }
```

Το πρόγραμμα 3

```
void setup() {
  frameRate(50);
}
int pos = 0;
void draw() {
  background(204);
  pos++;
  line(pos, 20, pos, 80);
  if (pos > width) {
    pos = 0;
  }
}
```

6.2 Ο κώδικας των εφαρμογών RGB-Image και Depth-Image

Η εφαρμογή RGB-Image

```
import SimpleOpenNI.*;
SimpleOpenNI kinect;
void setup()
{
size(640, 480);
kinect = new SimpleOpenNI(this);
kinect.enableRGB();
```

```
}
void draw()
{
kinect.update();
image(kinect.rgbImage(), 0, 0);
}
```

Η εφαρμογή Depth-Image

```
import SimpleOpenNI.*;
SimpleOpenNI kinect;
int x=0;
void setup()
{
size(640, 480);
kinect = new SimpleOpenNI(this);
kinect.enableDepth();
}
void draw()
{
kinect.update();
image(kinect.depthImage(), 0, 0);
}

void mousePressed(){
save("Image"+x+".png");
x+=1;
}
```

6.3 Ο κώδικας της εφαρμογής Αόρατα Κρουστά

To Processing αρχείο DrumsSimulation

```
import processing.opengl.*;
import SimpleOpenNI.*;
import ddf.minim.*;
SimpleOpenNI kinect;
// AudioSnippet objects
Minim minim;
AudioSnippet hihat,ride,crash;
AudioSnippet htom;
AudioSnippet mtom;
AudioSnippet kick,kick2;
AudioSnippet snare;
// declare our Stigmas objects
highStigma hihatArea,rideArea,crashArea;
mediumStigma snareArea,htomArea,mtomArea;
lowStigma kickArea,kickArea2;
float s = 1;
float rotationX=0;
```

```
float rotationY=0;
void setup() {
size(1024, 768, OPENGL);
frameRate(30); //kinect's frame rate at defaut is 30 so we choose the
same value for the sketch
kinect = new SimpleOpenNI(this);
kinect.enableDepth();
minim = new Minim(this);
// load all audio files
mtom = minim.loadSnippet("mediumtom.wav");
htom = minim.loadSnippet("hightom.wav");
snare = minim.loadSnippet("snare.wav");
kick = minim.loadSnippet("kick.wav");
kick2 = minim.loadSnippet("kick2.wav");
hihat = minim.loadSnippet("hihat.wav");
crash = minim.loadSnippet("crash.wav");
ride = minim.loadSnippet("ride.wav");
// initialize Stigmas with their origins (x,y,z) and their size
htomArea= new mediumStigma(-150, -100, 1600, 200);
mtomArea= new mediumStigma(150, -100, 1600, 200);
hihatArea = new highStigma(-370, 200, 1800, 150);
rideArea = new highStigma(370, 200, 1800, 150);
crashArea = new highStigma(70, 400, 1600, 170);
snareArea = new mediumStigma(0, -300, 1800, 200);
kickArea = new lowStigma(350, -580, 1500, 100);
kickArea2 = new lowStigma(-350, -580, 1500, 100);
}
void draw() {
background(0);
kinect.update();
translate(width/2, height/2, -1000);
rotateX(radians(180)); //mirror effect
 // disable mirror maybe but in setup
 //context.setMirror(false);
translate(0, 0, 1400);
rotateX(radians(rotationX));
rotateY(radians(rotationY));
//rotateX(radians(map(mouseY, 0, height, 0, 360))); //mouse navigator
rotateY(radians(map(mouseX, 0, width, 0, 360)));
translate(0, 0, s*-1000);
scale(s);
stroke(255);
PVector[] depthPoints = kinect.depthMapRealWorld();
for (int i = 0; i < depthPoints.length; i+=30) { //10 20 40
PVector currentPoint = depthPoints[i];
// have each Stigma check to see if it includes the currentPoint
mtomArea.check(currentPoint);
htomArea.check(currentPoint);
hihatArea.check(currentPoint);
rideArea.check(currentPoint);
crashArea.check(currentPoint);
```

```
snareArea.check(currentPoint);
kickArea.check(currentPoint);
kickArea2.check(currentPoint);
point(currentPoint.x, currentPoint.y, currentPoint.z);
}
//println(kickArea2.pointsIncluded);
if(rideArea.isHit()) {
ride.play();
}
if(!ride.isPlaying()) {
ride.rewind();
}
if(crashArea.isHit()) {
crash.play();
}
if(!crash.isPlaying()) {
crash.rewind();
}
if(htomArea.isHit()) {
htom.play();
}
if(!htom.isPlaying()) {
htom.rewind();
}
if(mtomArea.isHit()) {
mtom.play();
}
if(!mtom.isPlaying()) {
mtom.rewind();
}
if(snareArea.isHit()) {
snare.play();
}
if(!snare.isPlaying()) {
snare.rewind();
}
if (kickArea.isHit()) {
kick.play();
}
if(!kick.isPlaying()) {
kick.rewind();
}
if (kickArea2.isHit()) {
kick2.play();
}
if(!kick2.isPlaying()) {
kick2.rewind();
}
if (hihatArea.isHit()) {
hihat.play();
}
```

```
if(!hihat.isPlaying()) {
hihat.rewind();
}
//if (hihatArea.isHit()) {
//hihat.stop();
//hihat.rewind();
//hihat.play();
//}
// display each mediumStigma and clear its points
hihatArea.draw();
hihatArea.clear();
snareArea.draw();
snareArea.clear();
kickArea.draw();
kickArea.clear();
kickArea2.draw();
kickArea2.clear();
mtomArea.draw();
mtomArea.clear();
htomArea.draw();
htomArea.clear();
crashArea.draw();
crashArea.clear();
rideArea.draw();
rideArea.clear();
}
void stop()
{
//close all AudioPlayer objects
ride.close();
crash.close();
mtom.close();
htom.close();
hihat.close();
kick.close();
kick2.close();
snare.close();
minim.stop();
super.stop();
}
void keyPressed() {
if(key == 'a' || key == 'A'){
 s = s + 0.02;
}
if(key == 'z' || key == 'Z'){
 s = s - 0.02;
}
if (keyCode == 40) {
rotationX+=5;
}
if (keyCode == 38) {
```

```
rotationX-=5;
}
if(keyCode == RIGHT ){
    rotationY+=5;
}
if(keyCode == LEFT ){
    rotationY-=5;
  }
}
```

Ο κώδικας της κλάσης highStigma

```
class highStigma {
PVector center;
color fillColor;
color strokeColor;
int size;
int pointsIncluded;
int maxPoints;
boolean wasJustHit;
int threshold;
highStigma(float centerX, float centerY, float centerZ, int boxSize) {
center = new PVector(centerX, centerY, centerZ);
size = boxSize;
pointsIncluded = 0;
maxPoints = 1000;
threshold = 0;
fillColor = strokeColor = color(random(255), random(255), random(255));
}
void setThreshold( int newThreshold ){
threshold = newThreshold;
}
void setMaxPoints(int newMaxPoints) {
maxPoints = newMaxPoints;
}
void setColor(float red, float blue, float green){
fillColor = strokeColor = color(red, blue, green);
}
boolean check(PVector point) {
boolean result = false;
if (point.x > center.x - size/2 && point.x < center.x + size/2) {
if (point.y > center.y - size/2 && point.y < center.y + size/2) {
if (point.z > center.z - size/2 && point.z < center.z + size/2) {
result = true;
pointsIncluded++;
}
}
}
return result;
}
```

```
void draw() {
pushMatrix();
translate(center.x, center.y, center.z);
fill(red(fillColor), blue(fillColor), green(fillColor),
255 * percentIncluded());
stroke(red(strokeColor), blue(strokeColor), green(strokeColor), 255);
box(size,size,size);
popMatrix();
}
float percentIncluded() {
return map(pointsIncluded, 0, maxPoints, 0, 1);
}
boolean currentlyHit() {
return (pointsIncluded > threshold);
}
boolean isHit() {
return currentlyHit() && !wasJustHit;
}
void clear() {
wasJustHit = currentlyHit();
pointsIncluded = 0;
}
}
```

Ο κώδικας της κλάσης mediumStigma

```
class mediumStigma {
PVector center;
color fillColor;
color strokeColor;
int size;
int pointsIncluded;
int maxPoints;
boolean wasJustHit;
int threshold;
mediumStigma(float centerX, float centerY, float centerZ, int boxSize) {
center = new PVector(centerX, centerY, centerZ);
size = boxSize;
pointsIncluded = 0;
maxPoints = 1000;
threshold = 0;
fillColor = strokeColor = color(random(255), random(255), random(255));
}
void setThreshold( int newThreshold ){
threshold = newThreshold;
}
void setMaxPoints(int newMaxPoints) {
maxPoints = newMaxPoints;
}
void setColor(float red, float blue, float green){
fillColor = strokeColor = color(red, blue, green);
```

```
}
boolean check(PVector point) {
boolean result = false;
if (point.x > center.x - size/2 && point.x < center.x + size/2) {
if (point.y > center.y - size && point.y < center.y + size) {
if (point.z > center.z - size/2 && point.z < center.z + size/2) {
result = true;
pointsIncluded++;
}
}
}
return result;
}
void draw() {
pushMatrix();
translate(center.x, center.y, center.z);
fill(red(fillColor), blue(fillColor), green(fillColor),
255 * percentIncluded());
stroke(red(strokeColor), blue(strokeColor), green(strokeColor), 255);
box(size,size*2,size);
popMatrix();
}
float percentIncluded() {
return map(pointsIncluded, 0, maxPoints, 0, 1);
}
boolean currentlyHit() {
return (pointsIncluded > threshold);
}
boolean isHit() {
return currentlyHit() && !wasJustHit;
}
void clear() {
wasJustHit = currentlyHit();
pointsIncluded = 0;
}
}
```

Ο κώδικας της κλάσης lowStigma

```
class lowStigma {
PVector center;
color fillColor;
color strokeColor;
int size;
int pointsIncluded;
int maxPoints;
boolean wasJustHit;
int threshold;
lowStigma(float centerX, float centerY, float centerZ, int boxSize) {
center = new PVector(centerX, centerY, centerZ);
```

```
size = boxSize;
pointsIncluded = 0;
maxPoints = 1000;
threshold = 0;
fillColor = strokeColor = color(random(255), random(255), random(255));
}
void setThreshold( int newThreshold ){
threshold = newThreshold;
}
void setMaxPoints(int newMaxPoints) {
maxPoints = newMaxPoints;
}
void setColor(float red, float blue, float green){
fillColor = strokeColor = color(red, blue, green);
}
boolean check(PVector point) {
boolean result = false;
if (point.x > center.x - size/2 && point.x < center.x + size/2) {
if (point.y > center.y - size/3/2 && point.y < center.y + size/3/2) {
if (point.z > center.z - size/2 && point.z < center.z + size/2) {
result = true;
pointsIncluded++;
}
}
}
return result;
}
void draw() {
pushMatrix();
translate(center.x, center.y, center.z);
fill(red(fillColor), blue(fillColor), green(fillColor),
255 * percentIncluded());
stroke(red(strokeColor), blue(strokeColor), green(strokeColor), 255);
box(size,size/3,size);
popMatrix();
}
float percentIncluded() {
return map(pointsIncluded, 0, maxPoints, 0, 1);
}
boolean currentlyHit() {
return (pointsIncluded > threshold);
}
boolean isHit() {
return currentlyHit() && !wasJustHit;
}
void clear() {
wasJustHit = currentlyHit();
pointsIncluded = 0;
}
}
```

Βιβλιογραφία

web-sites:
en.wikipedia.org
processing.org
code.google.com/p/simple-openni/
processing.org/forum
zigfu.com

e-books:
Getting Started with Processing, Making Things See

software:
Processing 2.0, Notepad++, Eclipse, Cubase 5, Ableton Live 9 suite