Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αιγαίου**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Προηγμένες Τεχνολογίες Αλληλεπίδρασης και Εφαρμογές

Τμήμα Μηχανικών Σχεδίασης Προϊόντων και Συστημάτων,
ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ
Παναγιώτης Κουτσαμπάσης, επίκουρος καθηγητής

(α) Έισαγωγή στη γλώσσα προγραμματισμού Processing
(β) Χρήση της Processing με τον αισθητήρα MS Kinect

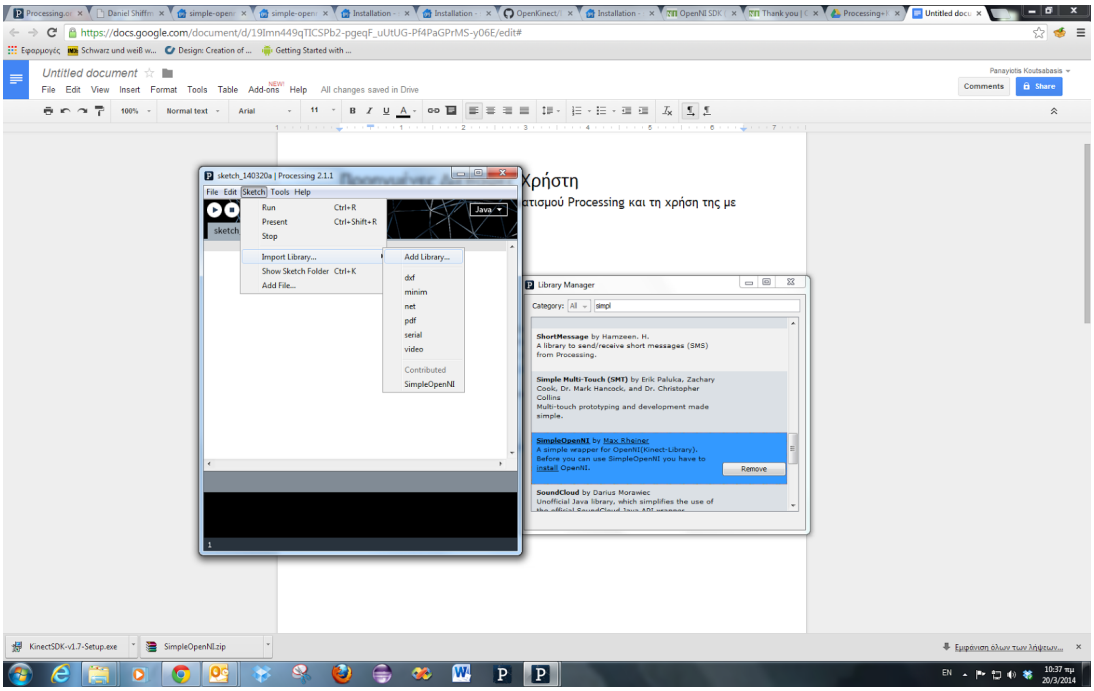
Στόχοι

- Εγκατάσταση: Processing, Βιβλιοθήκες OpenNI, SimpleOpenNI, Kinect Drivers, Microsoft Kinect SDK.
- Εισαγωγή στη Processing
    - Ιστορικά στοιχεία και σκοπός
    - Γενική δομή ενός προγράμματος processing
    - Απλή ζωγραφική: σχήματα (shapes), χρώματα (colours), κείμενα (text), εικόνες (images)
    - Αλληλεπίδραση με το χρήστη, μέσω του ποντικιού
    - Συνθήκη if, Επανάληψη for
    - Δημιουργία και κλήση μεθόδων
    - *Δημιουργία και κλήση κλάσεων*
- Εισαγωγή στο Microsoft Kinect
    - Ιστορικά στοιχεία και σκοπός
    - Η δομή της συσκευής και οι δυνατότητες ανίχνευσης και καταγραφής της
    - Προγραμματισμός σε Processing για αξιοποίηση των δεδομένων της κάμερας βάθους (depth camera)
        - Διάβασμα των ροών βίντεο από τις 2 κάμερες του kinect και απεικόνιση τους στην οθόνη του η/υ
        - Προσβαση στη κάμερα χρώματος: επίδειξη του χρώματος ενός pixel
        - Πρόσβαση στη κάμερα βάθους: επίδειξη της απόστασης ενός pixel από το kinect
        - Η πλέον στοιχειώδης δυνατότητα αλληλεπίδρασης με το χρήστη: επίδειξη του κοντινότερου σημείου στο kinect
    - Προγραμματισμός σε Processing για αξιοποίηση των δεδομένων των κινήσεων του σώματος των χρηστών (skeleton data)
        - Τι είναι τα δεδομένα σκελετού;
        - Ανιχνεύοντας το δεξί χέρι του χρήστη

        - Ζωγραφίζοντας το σκελετό του χρήστη
        - Διαχωρίζοντας υπόβαθρο από το χρήστη με χρήση της κάμερας βάθους
        - Αλλάζοντας το υπόβαθρο, με χρήση και των 2 καμερών.
    - Χρήση του kinect + simpleOpenNI για ανίχνευση των χειρονομιών που υποστηρίζονται (WAVE, CLICK, HAND_RAISE)
        - Τι είναι η χειρονομία;
        - Είδη χειρονομιών: καθαρές χειρονομίες, χειρισμοί και πόζες.
        - Πρόγραμμα σε kinect + simpleOpenNI ανίχνευσης των χειρονομιών WAVE, HAND_RAISE, CLICK
        - Πρόγραμμα σε kinect + simpleOpenNI: Υπολογισμός της απόστασης μεταξύ των αρθρώσεων.
        - Κατασκευή νέων χειρονομιών στη βιβλιοθήκη simpleOpenNI;

Εγκατάσταση

Θα χρειαστεί να εγκαταστήσετε τα παρακάτω λογισμικά:
1. Processing - http://processing.org/ -- Απλά αποσυμπιέστε το αρχείο RAR σε κάποιο folder κάτω από το οποίο έχετε τα περιβάλλοντα ανάπτυξης κώδικα.
2. Zigfu Browser Plugin http://zigfu.com/en/downloads/browserplugin/ -- Απλά τρέξτε το αρχείο για να εγκαταστήσετε τα προγράμματα OpenNI drivers (for Kinect), OpenNI software framework και PrimeSense middleware (for skeleton tracking).
3. Kinect for Windows SDK v1.8.
http://www.microsoft.com/en-us/download/details.aspx?id=40278 -- Κατεβάστε και εγκαταστήστε.
4. SimpleOpenNI.
https://simple-openni.googlecode.com/svn/trunk/SimpleOpenNI-2.0/dist/all/SimpleOpenNI.zip
    3.1. Αποσυμπιέστε στο folder Processing\libraries.
    3.2. Τρέξτε τη Processing
    3.3. Sketch | Import Library… | Add Library
    3.4. Επανακινείστε τη Processing.



Έπειτα θα χρειαστεί να συνδέσετε το kinect με τον υπολογιστή σας σε κάποια θύρα USB. Εφόσον τα παραπάνω προγράμματα έχουν εγκατασταθεί σωστά, θα δείτε ειδοποίηση κάτω δεξιά ότι η νέα σας συσκευή εντοπίστηκε και εγκαταστάθηκε επιτυχώς.

Σημείωση: Το πρώτο μέρος του εργαστηρίου αφορά την εκμάθηση βασικών δεξιοτήτων προγραμματισμού σε Processing και δεν απαιτεί τη χρήση του kinect. Το kinect είναι απαραίτητο για να δοκιμάσετε τα παραδείγματα του δεύτερου μέρους.

Έισαγωγή στη γλώσσα προγραμματισμού Processing

Ιστορικά στοιχεία και σκοπός

Η γλώσσα προγραμματισμού Processing έχει δημιουργηθεί για τη **διδασκαλία βασικών εννοιών προγραμματισμού** με έμφαση στη χρήση οπτικών στοιχείων που ζωγραφίζονται επί της οθόνης του υπολογιστή. Πρόκειται για μια πολύ συνοπτική και ευέλικτη γλώσσα, στην οποία είναι δυνατόν να γραφτούν προγράμματα που έχουν κάποια λειτουργικότητα με ελάχιστες γραμμές κώδικα. Ο στόχος των δημιουργών της γλώσσας ήταν να χρησιμοποιηθεί από καλλιτέχνες και σχεδιαστές διαδραστικών συστημάτων χωρίς να απαιτείται προηγούμενη γνώση προγραμματισμού. Βεβαίως, καθώς ανακαλύπτει κάποιος τις δυνατότητες, χρειάζεται να μελετήσει στο επίπεδο του προγραμματισμού για να κάνει πιο προχωρημένες εφαρμογές.

Η Processing είναι γλώσσα ανοικτού κώδικα και βασίζεται στη Java (επίσης ανοικτού κώδικα), ενώ κάθε πρόγραμμα της Processing μετατρέπεται σε Java και εκτελείται ως applet. Οι δημιουργοί της είναι οι **Casey Reas και Ben Fry** οι οποίοι ήταν διδακτορικοί φοιτητές στο

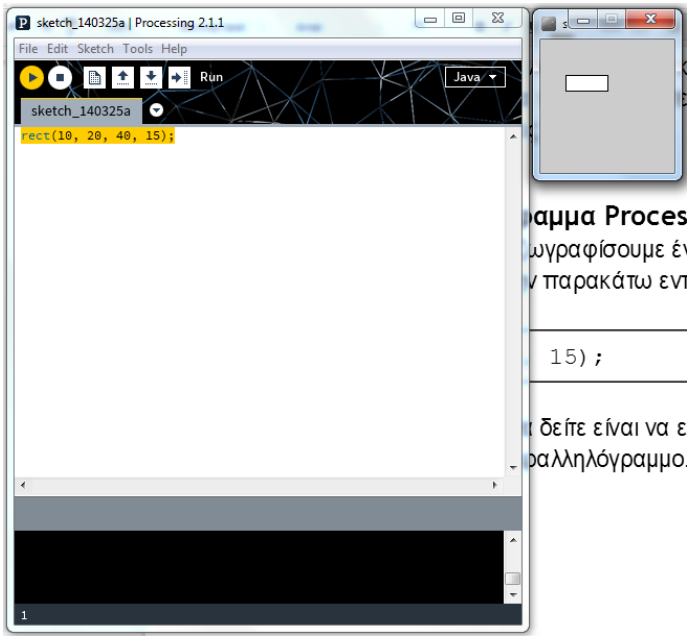
Aesthetics and Computation Group, MIT Media Lab. Η πρώτη έκδοση της γλώσσας ανακοινώθηκε το 2001, και έκτοτε έχει εξελιχθεί αρκετά από μια σημαντική κοινότητα ερευνητών, δημιουργών και προγραμματιστών κυρίως στα πανεπιστήμια UCLA, Carnegie Mellon. Η Processing μπορεί να χρησιμοποιηθεί αυτόνομα, αλλά και (αυτό είναι το πιο συναρπαστικό) σε συνεργασία με συσκευές όπως το Kinect, Arduino, πολυαπτικά τραπέζια, κινητά τηλέφωνα (π.χ. Android) αλλά και εφαρμογές προσωπικού υπολογιστή, ιδιαίτερα αν είναι γραμμένες σε Java. Για να συνεργαστεί με άλλες συσκευές και προγραμματιστικά περιβάλλοντα απαιτούνται κατάλληλες βιβλιοθήκες.

Ένα απλό πρόγραμμα Processing

Έστω ότι θέλουμε να ζωγραφίσουμε ένα παραλληλόγραμμο στην Processing. Ανοίξτε το SDK και πληκτρολογήστε την παρακάτω εντολή, και εκτελέστε το πρόγραμμα.

```
rect(10, 20, 40, 15);
```

Το αποτέλεσμα που θα δείτε είναι να εμφανίζεται ένα μικρό παράθυρο όπου είναι ζωγραφισμένο ένα παραλληλόγραμμο.



Μόλις γράψατε το πρώτο σας πρόγραμμα στη Processing! Αλλά τι έχει συμβεί; Μόλις πατήσατε το κουμπί της εκτέλεσης του προγράμματος...

1. Δημιουργήθηκε ένα νέο παράθυρο όπου φαίνεται το αποτέλεσμα. Ουσιαστικά πρόκειται για το **καμβά** ζωγραφικής. Επειδή δεν έχουμε προσδιορίσει το μέγεθος ή το υπόβαθρο του καμβά, ο καμβάς έχει αρχικοποιηθεί σε μέγεθος 100x100 pixels, γκρίζο υπόβαθρο.

2. Εντός του καμβά ζωγραφίστηκε το **παραλληλόγραμμο**. Αυτό συνέβη επειδή δώσατε την εντολή rect(10, 20, 40, 15);
3. Πως ήξερε η Processing που θα ζωγραφίσει το παραλληλόγραμμο; Από τους αριθμούς 10, 20 οι οποίοι είναι οι **συντεταγμένες (x, y) της άνω αριστερής γωνίας του παραλληλογράμμου**. Σημειώστε ότι το **σημείο (0,0) βρίσκεται στην πάνω αριστερή γωνία** του καμβά.
4. Πως ήξερε η Processing το μέγεθος του παραλληλογράμμου; Από τους αριθμούς 40, 15, όπου το 40 είναι το **μήκος** και το 15 το **ύψος**.

Το μέγεθος του καμβά είναι πολύ μικρό εξ' ορισμού. Μπορείτε να το ορίσετε εσείς με την εντολή `size(640, 480);` η οποία θα ορίσει το πλάτος (άξονας x) σε 640 και το ύψος (y) σε 480.

Επίσης, το χρώμα του υποβάθρου ίσως να μην σας αρέσει. Μπορείτε να χρησιμοποιήσετε την εντολή `background(r, g, b);` όπου θα πρέπει να δώσετε συγκεκριμένες τιμες στα r, g, b (red, green, blue) εντός του διαστήματος [0,255].

Προσθέστε τις παραπάνω εντολές και τρέξτε το πρόγραμμα σας. (προσοχή: η σειρά με την οποία προσθέτετε στον καμβά έχει σημασία!)

Οι εντολές background(), size() αποτελούν μεθόδους της Processing. Το σύνολο των μεθόδων που μπορεί κάποιος να χρησιμοποιήσει αποτελεί την **αναφορά της γλώσσας (language reference)** και μπορείτε να το δείτε από το μενού Help | Reference. Θα χρειαστεί να ανατρέχετε στην αναφορά συχνά ώστε να εντοπίζετε χρήσιμες εντολές.

Η γενική δομή ενός απλού προγράμματος Processing

Κάθε πρόγραμμα της Processing δεν απαρτίζεται απλά από κάποια σειρά εντολών, αλλά αποτελείται από μέρη που ονομάζονται μέθοδοι. Οι μέθοδοι είναι πολύ ισχυρά εργαλεία στον προγραμματισμό, επειδή μας επιτρέπουν να ορίσουμε εντός του σώματος τους εντολές που εκτελούνται κάθε φορά που τις καλούμε. Η Processing περιέχει δύο (2) ειδικές μεθόδους που καλούνται αυτόματα από το μηχανισμό εκτέλεσης του προγράμματος και έπιτελούν διαφορετικό έργο, τις setup(), draw().

Η μέθοδος setup()

Η μέθοδος **setup()** αρχικοποιεί το πρόγραμμα Processing**.** Η αρχικοποίηση κατά κανόνα περιλαμβάνει το μέγεθος του καμβά και το υπόβαθρο του (χρώμα ή εικόνα). Ουσιαστικά πρόκειται για μια **μέθοδο που εκτελείται μία μόνο φορά στην αρχή**. Η γενική μορφή της setup φαίνεται εδώ:

```
//μια κενή setup():
void setup(){
  //αυτό είναι σχόλιο, το block του προγράμματος είναι κενό
}
```

Ενώ η παρακάτω setup δίνει κάποιες αρχικές τιμές.

```
//μια setup() που θέτει τιμές στο υπόβαθρο και το μέγεθος του καμβά
void setup(){
  background(200, 150, 50); //rgb values, κάθε μία εντός του[0,255]
  size(640, 480); //640 = πλάτος, 480 = ύψος
}
```

Στα παραπάνω παραδείγματα παρατηρείστε:

- τα **σχόλια** (σημειώσεις για εμάς ή άλλους προγραμματιστές που δεν λαμβάνονται υπόψη από το διερμηνευτή του προγράμματος),
- κάθε μέθοδος περιλαμβάνει **εντολές** (που εκτελούνται όποτε καλείται η μέθοδος) εντός **αγκυλών** {},
- κάθε μέθοδος ξεκινάει με το εάν επιστρέφει κάποια τιμή (int, string, κ.α.) ή όχι (void)
- **Κάθε εντολή τελειώνει με ;**
- Η **κάθε μέθοδος παίρνει παραμέτρους** οι οποίες προσδιορίζουν τον επακριβώς τι θέλουμε να γίνει όταν την καλούμε. Έτσι:
    - Οι παράμετροι της μεθόδου `background(200, 150, 50)` προσδιορίζουν τις τιμές RGB (Red, Green, Blue) του χρώματος στο οποίο θέλουμε να ζωγραφιστεί το υπόβαθρο. Κάθε χρωματική τιμή πρέπει να είναι στο διάστημα [0,255].
    - Οι παράμετροι της μεθόδου `size(640, 480)` προσδιορίζουν το πλάτος (640) και το ύψος (480) του καμβά που θα δημιουργηθεί.

Η μέθοδος draw()

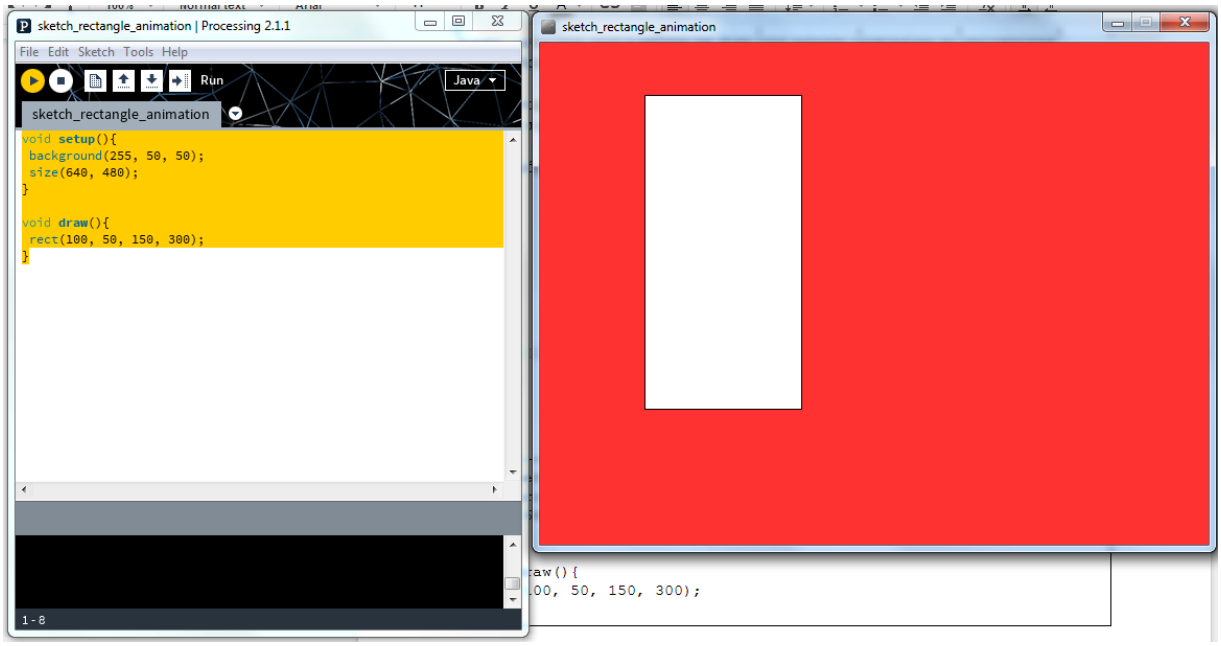
Η μέθοδος draw() ζωγραφίζει στο καμβά, σύμφωνα με τις εντολές που της δίνουμε εντός του σώματος της {}. Η μέθοδος εκτελείται συνεχώς από τη Processing ξαναζωγραφίζοντας τον καμβά. Προφανώς αυτή η ιδιότητα της draw είναι χρήσιμη προκειμένου να υλοποιήσουμε αλληλεπιδράσεις με το χρήστη ή/και animations.

Προς το παρόν όμως απλά μπορούμε να μεταφέρουμε την εντολή `rect(10, 20, 40, 15);` εντός του σώματος της `draw();`

Άρα το πρόγραμμα μας πλέον, μαζί με το αποτέλεσμα φαίνονται παρακάτω:

```
void setup(){
 background(255, 50, 50);
 size(640, 480);
}

void draw(){
 rect(100, 50, 150, 300);
```

```
}
```



Ζωγραφίζοντας σχήματα

Για να ζωγραφίζουμε ένα σχήμα στη Processing χρειάζεται να δώσουμε μια εντολή ανάλογα με το σχήμα. Τα **σχήματα (2Δ)** που υποστηρίζονται είναι τα (**βλ. reference**):

- Τόξο (arc),
- Έλλειψη (ellipse),
- Γραμμή (line),
- Σημείο (point),
- Τετράεδρο (quadrilateral),
- Παραλληλόγραμμο (rectangle),
- Τρίγωνο (triangle),

Και οι αντίστοιχες εντολές τους είναι (2D Primitives): arc(); ellipse(); line(); point(); quad(); rect(); triangle();

Σε ποιο σημείο της οθόνης μπορούμε να ζωγραφίσουμε κάθε ένα από τα παραπάνω σχήματα; Η απάντηση είναι “σε όποιο σημείο θέλουμε, αρκεί να το προσδιορίσουμε από τις **συντεταγμένες** του της μορφής (x,y) όπου το (0,0) είναι το πρώτο σημείο πάνω αριστερά. Για την ακρίβεια δεν ζωγραφίζουμε στην οθόνη του Η/Υ όπως τη βλέπουμε, αλλά σε κάποιο **καμβά** που δημιουργείται αυτόματα από την processing. Και με τον όρο σημείο εννοούμε το **εικονοκύτταρο (pixel)**, δηλαδή το μικρότερο στοιχείο της οθόνης το οποίο χαρακτηρίζεται από τη θέση και το χρώμα του. Για όλα τα παραπάνω σχήματα (εκτός της γραμμής και του σημείου) οι πρώτες δύο παράμετροι προσδιορίζουν το κέντρο ή τη πρώτη ακμή τους (βλ. reference).

Ποιες άλλες λεπτομέρειες του σχήματος μπορούμε να προσδιορίσουμε;
Για κάθε σχήμα, τις εντελώς απαραίτητες λεπτομέρειες προκειμένου να ζωγραφιστεί. Έτσι, π.χ. για την έλλειψη ellipse(a, b, c, d), προσδιορίζουμε όπου: a: x-συντεταγμένη, b: y-συντεταγμένη (τα x, y προσδιορίζουν το κέντρο της έλλειψης), c: πλάτος (στον άξονα των x), d: ύψος (άξονας y). Για να δώσουμε περισσότερες λεπτομέρειες πρέπει να χρησιμοποιήσουμε και άλλες εντολές.

Άσκηση
Επεκτείνετε το παραπάνω πρόγραμμα ώστε να δημιουργήστε κάθε ένα σχήμα από τα παραπάνω. Κοιτάξτε το language reference, για την ακριβή σύνταξη. Τι συμβαίνει όταν το ένα σχήμα πέφτει πάνω στο άλλο;

Άσκηση
Σχεδιάστε κάτι απλό με χρήση βασικών σχημάτων, π.χ. ένα σπίτι ή ένα πρόσωπο.

Χρησιμοποιώντας Χρώμα

Κάθε χρώμα αναπαρίσταται στη Processing από τρεις αριθμούς σύμφωνα με το πρότυπο RGB: Red-Green-Blue. Η τιμή του κάθε αριθμού κυμαίνεται στο [0, 255]. Άρα το χρώμα (255, 0, 0) είναι το πιο κόκκινο που μπορεί να παραχθεί. (Βεβαίως η αντίληψη των χρωμάτων είναι σε κάποιο βαθμό υποκειμενική, θα το διαπιστώσετε καθώς δημιουργείτε παραλλαγές των χρωμάτων).

Κάθε σχήμα που φτιάχνουμε έχει **περίγραμμα (stroke)** και **γέμισμα (fill)** τα οποία μπορούν να χρωματιστούν. Για να δώσουμε χρώμα στο περίγραμμα κάποιου σχήματος χρησιμοποιούμε την εντολή stroke(r, g, b) ενώ για γέμισμα την εντολή fill(r, g, b). Οι εντολές πρέπει να δωθούν **πριν η Processing ζωγραφίσει** το σχήμα. Επίσης, ισχύουν για όλα τα επόμενα σχήματα. Δοκιμάστε π.χ. τον παρακάτω κώδικα.

*Σημείωση: Αν θέλουμε να χρησιμοποιήσουμε μόνο αποχρώσεις στο **ασπρόμαυρο** φάσμα, τότε καλούμε τις εντολές (μεθόδους) περνώντας ως παράμετρο μόνο έναν αριθμό στο [0,255], όπου 0 = μαύρο και 255 = άσπρο. Π.χ. η εντολή* `background(0)` *θα μας χρωματίσει το υπόβαθρο σε μαύρο.*

```
void setup(){
 size(640, 480);
}

void draw(){
 background(0);
```

```
 fill(0, 255, 0);
 stroke(0, 0, 255);

 arc(50, 50, 100, 60, HALF_PI, PI);
 ellipse(100, 50, 70, 100);
 line(150, 50, 250, 70);
 point(260, 50);
 quad(300, 50, 340, 80, 330, 130, 260, 130);
 rect(100, 250, 50, 180);

 fill(0, 0, 255);
 stroke(0, 255, 0);
 triangle(250, 300, 500, 420, 250, 420);
}
```

Χρήση εικόνων (images)

Για να χρησιμοποιήσουμε μια εικόνα σε ένα πρόγραμμα της Processing, θα πρέπει: (α) να τη τοποθετήσουμε στο φάκελο data της Processing (από εκεί διαβάζονται οι εικόνες), (β) να δηλώσουμε στο πρόγραμμα μας ότι θα χρησιμοποιήσουμε κάποια εικόνα, (γ) να τη φορτώσουμε στο πρόγραμμα μας εντός της μεθόδου `setup()`, και (δ) να την δείξουμε στο χρήστη. Για να κάνουμε το:

(α): Πηγαίνετε στο φάκελο όπου έχετε τις εικόνες σας, και κάνετε απλά drag 'n' drop του εικονιδίου της εικόνας στο παράθυρο του SDK, για κάθε εικόνα που θα χρησιμοποιήσει το πρόγραμμα σας.

(β): Η δήλωση μιας εικόνας γίνεται ως καθολική μεταβλητή πριν τη δήλωση της setup() με την εντολή: `PImage img; //όπου img είναι το αναγνωριστικό για την εικόνα`

(γ): Η φόρτωση στο πρόγραμμα γίνεται εντός της `setup()` με την εντολή `loadImage`, π.χ.

```
img = loadImage("myImage.png");
```

(δ) Η τοποθέτηση της εικόνας γίνεται με την εντολή:

```
image(img, x, y);
image(img, x, y, width, height);
```

Άσκηση

Χρησιμοποιείστε τις παραπάνω εντολές για να δείξετε κάποια εικόνα στο καμβά.

Γράφοντας (σύντομο) κείμενο

Για να γράψουμε κάποιο κείμενο στο καμβά χρησιμοποιούμε τη μέθοδο `text` η οποία έχει δύο μορφές:

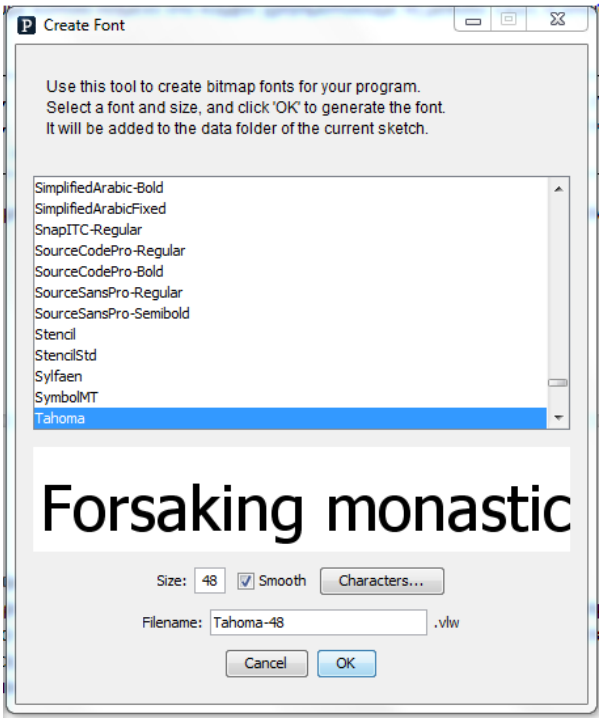
```
text(data, x, y); //τυπώνει το περιεχόμενο του data στο (x,y)
text(data, x, y, width, height); //τυπώνει το περιεχόμενο του data στο (x,y) με πλάτος width και ύψος height
```

Το περιεχόμενο του `data` παραπάνω μπορεί να είναι `String, int, char, float`.
ΠΡΟΣΟΧΗ: Τα x, y, αναφέρονται στο **κάτω αριστερά σημείο** από το οποίο θα ξεκινήσει να γράφεται το κείμενο και όχι στο πάνω αριστερά, όπως όταν ζωγραφίζουμε σχήματα.

Η Processing μπορεί να κάνει διάφορες επεξεργασίες τυπογραφίας. Οι πιο απλές είναι οι εξής:

Ορισμός γραμματοσειράς

Αν θέλετε να γράψετε σε κάποια γραμματοσειρά εκτός αυτής που παρέχεται εξ' ορισμού από τη Processing, θα πρέπει:
(α) Να τη δημιουργήσετε από το SDK. Επιλέγετε από το μενού Tools | Create Font και επιλέγετε τη γραμματοσειρά που επιθυμείτε από το παράθυρο που εμφανίζεται.



(β) Να ορίσετε τη γραμματοσειρά και να τη φορτώσετε στο πρόγραμμα σας, ως εξής (π.χ. για τη γραμματοσειρά που φαίνεται παραπάνω):

```
PFont font = loadFont("Tahoma-48.vlw");
```

Χρώμα

Χρησιμοποιείτε την εντολή `fill(r, g, b);`

Μέγεθος γραμματοσειράς

Χρησιμοποιείτε την εντολή `textSize(size); // size is in pixels (not points!)`

Εύρεση του πλάτους ενός κειμένου

Συχνά χρειάζεται να γνωρίζουμε στο πρόγραμμα μας το πλάτος ενός κειμένου. Για να μην το υπολογίζουμε κάθε φορά, η Processing μας προσφέρει τη μέθοδο `textWidth(data)`.

Άσκηση

Δοκιμάστε τις παραπάνω μεθόδους για να εμφανίσετε ένα απλό κείμενο.

Υλοποιώντας απλές αλληλεπιδράσεις

Η θέση του ποντικιού

Ό,τι είδαμε μέχρι τώρα αφορά στη ζωγραφική στατικών σχημάτων και χρωμάτων. Πως μπορούμε όμως να αλληλεπιδράσουμε με τη Processing? Ας δούμε από το πιο βασικό στοιχείο της αλληλεπίδρασης που είναι το σημείο στο οποίο δείχνει ο **κέρσορας του ποντικιού**. Πολλές από τις αλληλεπιδράσεις που μπορούν να υποστηριχθούν χρειάζεται να γνωρίζουν που βρίσκεται ο κέρσορας και ανάλογα να αντιδράσουν.

Ο κέρσορας του ποντικιού βρίσκεται πάντοτε σε ένα σημείο (pixel) της μορφής (mouseX, mouseY). Χρησιμοποιώντας τις μεταβλητές **mouseX, mouseY** μπορούμε να αναφερόμαστε στη τρέχουσα θέση του ποντικιού! Έτσι, αν δοκιμάσουμε να δημιουργήσουμε μια έλλειψη που θα ακολουθεί το ποντίκι μας γράφουμε τον παρακάτω κώδικα.

```
void setup(){
 size(640, 480);
}

void draw(){
 ellipse(mouseX, mouseY, 70, 100);
}
```

Ασκήσεις

1. Θα παρατηρήσετε ότι η έλλειψη ζωγραφίζεται συνεχώς, χωρίς να σβήνεται η προηγούμενη… πως μπορείτε να αλλάξετε το πρόγραμμα σας ώστε να σβήνεται η προηγούμενη έλλειψη;

2. Δοκιμάστε να αντιστρέψετε τα mouseX, mouseY.
3. Αλλάξτε λίγο το πρόγραμμα σας ώστε να εμφανίζεται, αντί της παραπάνω έλλειψης, ένα μικρό κυκλάκι πάντα λίγο πάνω από το κέρσορα.

Πάτημα του ποντικιού (και συνθήκη if)

Η πιο συνηθισμένη περίπτωση όπου θέλουμε το πρόγραμμα μας να αλληλεπιδρά με το χρήστη είναι όταν ο χρήστης πατάει το πλήκτρο του ποντικιού.

Έστω ότι στο παραπάνω πρόγραμμα που ο χρήστης ζωγραφίζει ένα μικρό κυκλάκι, θέλουμε αυτό να γίνεται μόνο όταν έχει πατημένο το ποντίκι. Πως το κάνουμε; Η απάντηση είναι ότι θα πρέπει να ελέγχουμε αν ο χρήστης πατάει το πλήκτρο του ποντικιού και μόνο τότε να ζωγραφίζουμε!

Για να υλοποιήσουμε τον παρακάτω κώδικα θα χρειαστεί να θυμηθούμε πως συντάσσεται η **συνθήκη if**. Οι εντολές που βρίσκονται εντός του σώματος της συνθήκης if εκτελούνται μόνο όταν η συνθήκη είναι αληθής. Η σύνταξη της είναι απλή και φαίνεται παρακάτω:

```
if (condition) {
  //code
}
ή
if (condition) {
  //code
} else {
  //code
}
ή
if (condition2) {
  //code
} else if (condition2) {
  //code
} else if (... conditionN){
  //code
}
```

Επίσης, πρέπει να μάθουμε ότι με η Processing μας προσφέρει τη μεταβλητή boolean **mousePressed** με την οποία μπορούμε να ελέγξουμε ανά πάσα στιγμή αν ο χρήστης έχει πατήσει το ποντίκι. Επομένως, για να λειτουργήσει το πρόγραμμα μας αρκεί να κάνουμε δώσουμε την εξής εντολή:

```
if (mousePressed)
  ellipse(mouseX, mouseY-10, 10, 10);
```

Άλλη χρήσιμη μεταβλητή σχετικά είναι η **mouseButton** που παίρνει τιμή **LEFT** όταν έχει πατηθεί το αριστερό πλήκτρο και **RIGHT** όταν έχει πατηθεί το δεξί πλήκτρο.

Άσκηση
Πως θα μπορούσαμε να επεκτείνουμε το πρόγραμμα μας ώστε να σβήνει το καμβά όταν ο χρήστης πατήσει το δεξί πλήκτρο του ποντικιού;

Λύση

```
void setup(){
 size(640, 480);
 background(0);
}

void draw(){
 if (mousePressed)
  ellipse(mouseX, mouseY-10, 10, 10);

 if (mouseButton == RIGHT)
   background(0);
}
```

Η μέθοδος mousePressed()

Όταν θέλουμε να εκτελεστεί κάποιος κώδικας σε απάντηση του πατήματος του πλήκτρου του ποντικιού μπορούμε - εναλλακτικά (και πιο κομψά) της παραπάνω συνθήκης if - να χρησιμοποιήσουμε τη μέθοδο `mousePressed()`. Πρόκειται για μια ακόμα ειδική μέθοδο της Processing η οποία εκτελείται αυτόματα μόλις ο χρήστης πατήσει (και αφήσει) το πλήκτρο του ποντικιού.

Η μέθοδος redraw()

Η μέθοδος `redraw()` είναι μια ακόμα ειδική μέθοδος που καλείται αυτόματα από την Processing για να ξαναζωγραφίσει (καλώντας (εκτάκτως) την draw()). Είναι πολύ χρήσιμη μέθοδος, ιδιαίτερα όταν την καλούμε από κάποια άλλη μέθοδο που έχουμε φτιάξει εμείς. Η `redraw()` πρέπει να καλείται εκτός της draw().

Άσκηση
Συνδυάστε αυτά που είδαμε στα δύο τελευταία κεφάλαια (“Γράφοντας (σύντομο) κείμενο” και “Υλοποιώντας απλές αλληλεπιδράσεις”) για να φτιάξετε πρόγραμμα που:
1. Γράφει ένα σύντομο κείμενο στον καμβά με μέγεθος γραμματοσειράς 30, καθώς και το μέγεθος του (αρχικά 30).
2. Όταν ο χρήστης πατάει το πλήκτρο του ποντικιού να μικραίνει το μέγεθος της γραμματοσειράς κατά 1 και να ξανατυπώνει το κείμενο και το πλάτος του. Όμως, αν η

γραμματοσειρά είναι μικρότερη από 10, να ξαναγίνεται 30.

Λύση

```
String textToPrint = "Hello";
int textSize = 30;

void setup(){
  size(640, 480);
}

void draw(){
  background(0);
  PFont font = loadFont("Tahoma-48.vlw");
  fill(255, 0, 0);
  textSize(textSize);
  text(textToPrint, 100, 100);
  float width = textWidth(textToPrint);
  text(width, 100, 200);
}

void mousePressed(){
  textSize = textSize -1;
  if (textSize < 10)
    textSize = 30;
}
```

Άσκηση

Συνδυάστε αυτά που είδαμε στα δύο τελευταία κεφάλαια (“Γράφοντας (σύντομο) κείμενο” και “Υλοποιώντας απλές αλληλεπιδράσεις”) για να φτιάξετε πρόγραμμα που γράφει ένα σύντομο κείμενο στον καμβά, και όταν ο χρήστης πατάει το πλήκτρο του ποντικιού να τυπώνει ένα παραλληλόγραμμο γύρω από το κείμενο.

Λύση

```
String textToPrint = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
int tSize = 30;
int textX, textY;

void setup(){
  size(640, 480);
  background(0);
}

void draw(){
  PFont font = loadFont("Tahoma-48.vlw");
  fill(255, 0, 0);
```

```
  tSize = 40;
  textSize(tSize);
  textX = 0;
  textY = 150;
  text(textToPrint, textX, textY);
}

void mousePressed(){
  float width = textWidth(textToPrint);
  fill(0);
  stroke(255,0,0);
  rect(textX, textY-tSize, width, tSize);
}
```

Επαναλήψεις (for loop)

Πολύ συχνά χρειάζεται να επαναλάβουμε εντολές σε ένα πρόγραμμα. Π.χ. Έστω ότι θέλουμε να ζωγραφίζουμε 100 τετράγωνα σε διαφορετικά σημεία του καμβά - έστω διαγωνίως, όπου το επόμενο θα είναι στη θέση (x+20, y+20) από το προηγούμενο. Τι θα κάνουμε στο πρόγραμμα μας; Με τα όσα έχουμε δει ως τώρα, θα πρέπει να επαναλάβουμε 100 εντολές εντός του draw()… Προφανώς δεν πρέπει να κάνουμε κάτι τέτοιο, αλλά να χρησιμοποιήσουμε την επανάληψη for, ώστε να αρκεί μια εντολή.

Η επανάληψη for επαναλαμβάνει τις εντολές που εσωκλείονται στο σώμα της όσο δεν παύει να ισχύει η συνθήκη τερματισμού (termination). Η γενική δομή της for είναι:

```
for (initialization; termination; increment){
  //code
}
```

Συνήθως η for συντάσσεται με κάποιο μετρητή int i ως εξής (π.χ.):

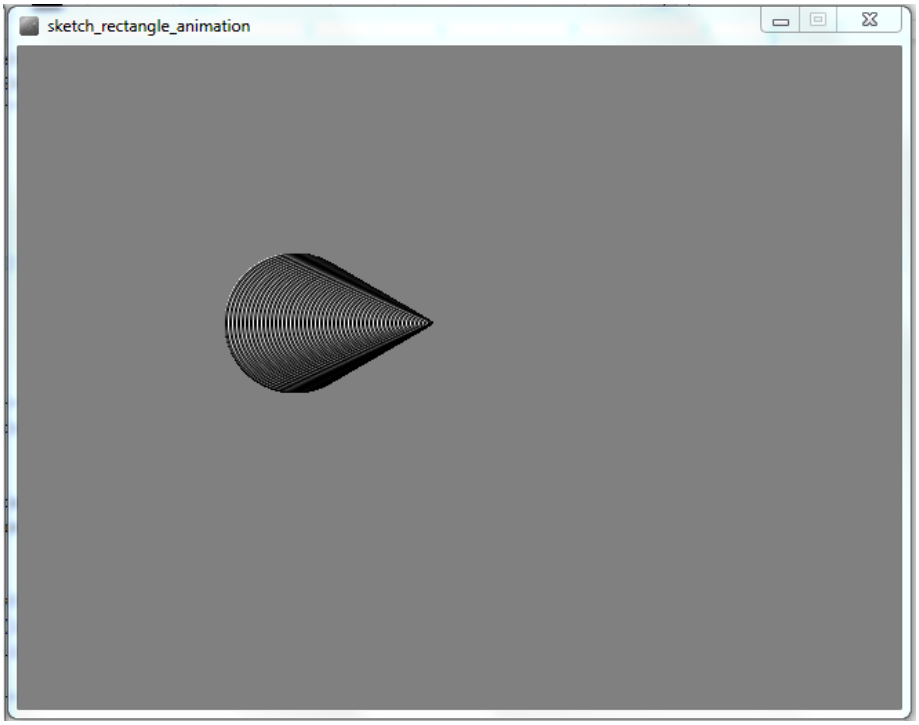
```
for (int i=0; i<100; i++){
  //code
}
```

Έτσι, για να λύσουμε το παραπάνω πρόβλημα απλά γράφουμε την εντολή

```
for (int i=0; i<100; i++){
  rect(50+i*20, 50+i*20, 100, 100);
}
```

Με τις επαναλήψεις μπορούμε να δημιουργήσουμε ενδιαφέροντα σχήματα… π.χ. ένας κώνος (με τη ψευδαίσθηση των 3Δ) που ζωγραφίζεται με επαναληπτική ζωγραφική κύκλων που μικραίνουν βηματικά:

```
void setup(){
 size(640, 480);
 background(128);
}

void draw(){
 for (int i=0; i<100; i++)
  ellipse(200+i, 200, 100-i, 100-i);
}
```



Μέθοδοι

Οι μέθοδοι είναι πολύ ισχυρές έννοιες στο προγραμματισμό μιας και μας επιτρέπουν να ορίσουμε εντός του σώματος τους εντολές που εκτελούνται κάθε φορά που τις καλούμε.

Μέχρι στιγμής έχουμε δει ότι δηλώνουμε μόνο 2 μεθόδους σε ένα πρόγραμμα της Processing, τις setup() και draw() - και ότι αυτές οι μέθοδοι καλούνται αυτόματα από την Processing.

Όμως, μπορούμε να δηλώσουμε και δικές μας μεθόδους. Και με τη δήλωση και την κλήση

τους μπορούμε να αναπτύξουμε με κομψό τρόπο το πρόγραμμα μας.

Κοιτάξτε τον παραπάνω κώδικα που ζωγραφίζει σταδιακά έναν κώνο. Γιατί όχι να ορίσουμε μια μέθοδο που το κάνει. Ας την ονομάσουμε myCone(). Για να την ορίσουμε θα πρέπει:

1. Να βγούμε έξω από το σώμα της draw() και να τη δηλώσουμε αντίστοιχα void myCone()
2. Να αντιγράψουμε το κώδικα που φτιάχνει το κώνο εντός της myCone().
3. Να καλέσουμε την myCone() από το σώμα της draw();

```
void setup(){
 size(640, 480);
 background(128);
}

void draw(){
  myCone();
}

void myCone(){
  for (int i=0; i<100; i++)
    ellipse(200+i, 200, 100-i, 100-i);
}
```

Αν τρέξετε το πρόγραμμα, το αποτέλεσμα είναι ακριβώς το ίδιο… Είναι ίσως πιο κομψό (αν και αυτό είναι και θέμα γούστου), αλλά αυτός είναι ο μόνος λόγος για τον οποίο έχουμε μεθόδους; Ασφαλώς και όχι. Ο βασικός λόγος είναι ότι μπορούμε να βάλουμε παραμέτρους στη μέθοδο μας και να την καλέσουμε πολλές φορές με διαφορετικές τιμές για κάθε παράμετρο. Για να το κάνουμε αυτό θα πρέπει:

1. Να αλλάξουμε τη δήλωση της επικεφαλίδας της μεθόδου ώστε να δέχεται ορίσματα
2. Να χρησιμοποιήσουμε τα ορίσματα στο σώμα της μεθόδου.
3. Να καλέσουμε τη μέθοδο με ορίσματα - εδώ πλέον μπορούμε με διαφορετικά ορίσματα κάθε φορά…

Για να φτιάξουμε τη μέθοδο λοιπόν σε μια μορφή που μπορούμε να την καλέσουμε πολλές φορές αλλάζουμε το κώδικα ώς εξής:

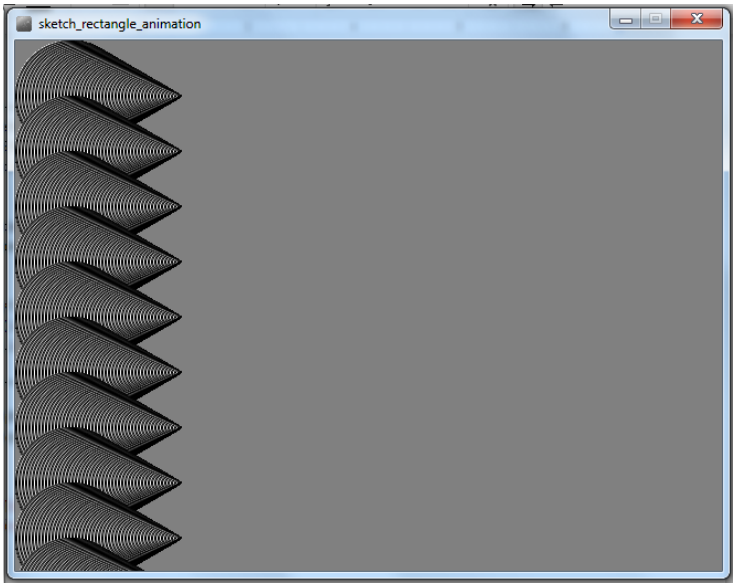
```
void setup(){
 size(640, 480);
 background(128);
}

void draw(){
  myCone(200, 200);
}
```

```
void myCone(int x, int y){
  for (int i=0; i<100; i++)
    ellipse(x+i, y, 100-i, 100-i);
}
```

Τώρα μπορούμε να την καλέσουμε πολλές φορές εντός της draw(), ή ακόμα να την βάλουμε σε μια νέα επανάληψη for.

Άσκηση

Καλέστε τη μέθοδο myCone() με χρήση της for για 10 φορές αλλάζοντας την ακμή του κώνου κάθε φορά κατά +50 στον άξονα των y ώστε να έχετε το παρακάτω μοτίβο στο αποτέλεσμα. Πόσες επαναλήψεις έχει κάνει συνολικά το πρόγραμμα σας;



Ασκήσεις (προαιρετικές, για εξάσκηση)

1. Φτιάξτε μια απλή εφαρμογή σε Processing μετάφρασης/απόδοσης ορολογίας ως εξής: Αρχικά φτιάξτε ένα tag cloud από όρους του πεδίου της Επικοινωνίας Ανθρώπου-Υπολογιστή επιλέγοντας μερικούς όρους από τη σελίδα: http://hci-dpsd.wikispaces.com/hci-translation (επιλέξτε καλύτερα κάποιους σχετικά με τη συγκεκριμένη διεπαφή) και δείξτε το στο μισό της οθόνης. Ο χρήστης θα πρέπει να επιλέγει τον όρο με το ποντίκι, ώστε στο άλλο μισό της οθόνης να εμφανιστεί η μετάφραση/απόδοση του όρου. Παίξτε με διαφορετικά χρώματα, animations, σχήματα, τυπογραφία, ώστε η εφαρμογή να γίνει ελκυστική.

2. Φτιάξτε μια απλή εφαρμογή παροχής οδηγιών για τα γραφεία των καθηγητών στο Α' Γυμνάσιο σε Processing. Αρχικά διατάξτε τις φωτογραφίες και τα ονόματα μερικών

καθηγητών του τμήματος στο πρώτο μισό της οθόνης (θα τις βρείτε από τα προσωπικά web sites τους). Καθώς ο χρήστης επιλέγει κάποιο καθηγητή με το ποντίκι, θα πρέπει να φαίνονται οδηγίες (αριθμός γραφείου και κάποιο σχεδιάγραμμα ή χάρτης προς αυτό). Υποθέστε ότι η εφαρμογή θα δείχνει οδηγίες από τον 1ο όροφο του Α' Γυμνασίου, απέναντι ακριβώς από τον ανελκυστήρα (για να φτιάξετε τις οδηγίες/σχεδιάγραμμα από το σημείο αυτό προς τα γραφεία).

3. Φτιάξτε μια απλή εφαρμογή παροχής οδηγιών για τις διαφορετικές αίθουσες και κτίρια του Τμήματος σε Processing. Αρχικά διατάξτε στο πρώτο μισό της οθόνης τα ονόματα και φωτογραφίες των εξής κτιρίων: Α' Γυμνάσιο, Γραμματεία, Πνευματικό Κέντρο, Κτήριο Πρώην Καζίνο, Βιβλιοθήκη, Εστιατόριο. Καθώς ο χρήστης επιλέγει κάποιο κτίριο με το ποντίκι, θα πρέπει να φαίνεται το σημείο όπου βρίσκεται το κτίριο σε κάποιο χάρτη στο άλλο μισό της οθόνης.

Γενική οδηγία: φτιάξτε μια αρχική έκδοση με ό,τι πιο απλό χρειάζεται. Αφού κάνετε το πρόγραμμα σας να λειτουργεί σε μια πολύ απλή έκδοση του, να φτιάξετε σταδιακά πιο εξελιγμένες εκδόσεις του.

Χρήση της Processing με τον αισθητήρα MS Kinect

Τι είναι το kinect? - Η δομή της συσκευής και οι δυνατότητες ανίχνευσης και καταγραφής

Ποιος έφτιαξε το kinect? - Ιστορικά στοιχεία και σκοπός

Το kinect είναι ένας "**πολυαισθητήρας**" εικόνας, βίντεο, κίνησης και ήχου που κυκλοφόρησε στην αγορά το **Νοέμβριο 2010** ως το πλέον βασικό στοιχείο της πλατφόρμας παιχνιδιών βίντεο Microsoft Xbox 360.

Η τεχνολογία δεν ήταν αμέσως ανοικτή, αλλά λόγω των επαναστατικών τεχνολογικών δυνατοτήτων του, ερευνητές από πανεπιστήμια και άλλες εταιρίες κινητοποιήθηκαν ραγδαία προκειμένου να "σπάσουν" την ασφάλεια της συσκευής και να τη χρησιμοποιήσουν και για άλλους σκοπούς. Λίγες μόνο μέρες μετά τη κυκλοφορία του Kinect κυκλοφόρησαν προγράμματα οδηγοί (drivers) για την έγκατάσταση του στο περιβάλλον του προσωπικού υπολογιστή, και εντός του 2011 έγιναν διαθέσιμες κάποιες πρώτες βιβλιοθήκες πρόσβασης στη λειτουργικότητα (μέρος) του kinect.

Η Microsoft διέθεσε τελικά το **Νοέμβριο του 2012 το kinect for Windows μαζί με το kinect SDK** για ανάπτυξη εφαρμογών σε γλώσσες προγραμματισμού που υποστηρίζει το **MS Visual Studio** όπως η C++, C#. Αυτή είναι και η κύρια πλατφόρμα ανάπτυξης εφαρμογών σε kinect.

Η μη-κερδοσκοπική κοινότητα **OpenNI (Open Natural Interface)** (http://www.openni.org) έχει διαθέσει τις περισσότερες βιβλιοθήκες πρόσβασης στο kinect. Η κοινότητα αποτελείται από προγραμματιστές που διαθέτουν τα προγράμματα τους σε ανοικτό κώδικα και έχει στενή συνεργασία και υποστήριξη από την εταιρία PrimeSense (Ισραήλ) η οποία ήταν ο σημαντικότερος συνεργάτης της Microsoft για τη δημιουργία του kinect.

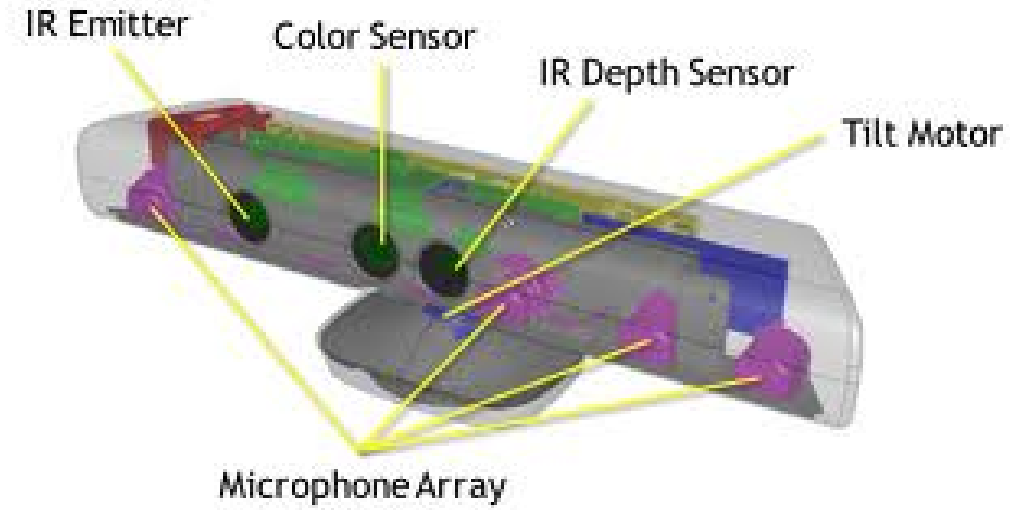
Η βιβλιοθήκη **SimpleOpenNI** που θα χρησιμοποιήσουμε στο εργαστήριο χρησιμοποιεί το **OpenNI SDK** για να δώσει πρόσβαση στο kinect από τη γλώσσα προγραμματισμού Processing. 'Εχει γραφτεί από τον Max Reiner (interaction designer, προγραμματιστής της Processing, συγγραφέας του βιβλίου Generative Design) και έγινε διαθέσιμη το Σεπτέμβρη του 2013 (https://code.google.com/p/simple-openni/).

Το Νοέμβριο του 2013 η Apple ανακοίνωσε την αγορά της εταιρίας PrimeSense για 350 εκ. $. Ο δικτυακός τόπος του OpenNI θα κλείσει τον Απρίλιο του 2014, ενώ δεν θα παρέχεται πλέον υποστήριξη από την Apple. Το πως θα επηρεάσει αυτή η εξέλιξη τη σχετική κοινότητα ανοικτού κώδικα είναι κάτι που μένει να δούμε στο άμεσο μέλλον.

Δυνατότητες της συσκευής kinect και γκάμα εφαρμογών

Η συσκευή kinect είναι ένας “πολυαισθητήρας” με πολλές ενδιαφέρουσες δύνατότητες:

1. Έγχρωμη κάμερα 2Δ (**RGB camera**), 1280x960 pixels, 30fps.
2. Προβολέα υπέρυθρου φωτός (**IR Infrared projector - emitter**). Εκπέμπει για να φωτίσει καλύτερα το χώρο.
3. Υπέρυθρη κάμερα (**IR camera**). Συλλαμβάνει τις αντανακλάσεις του υπέρυθρου φωτός και (μέσω κατάλληλου λογισμικού) υπολογίζει τις αποστάσεις κάθε σημείου (pixel) της εικόνας από τη συσκευή. Επίσης, 1280x960 pixels, 30fps.
4. **Συστοιχία 4 μικροφώνων**. Το λογισμικό που τα ελέγχει μπορεί να καταγράψει την κατεύθυνση από την οποία έρχεται ο ήχος.
5. **Μοτέρ κίνησης**, το οποίο επιτρέπει στη συσκευή να κινηθεί στον οριζόντιο άξονα κατά +/- 27 μοίρες.



Τεχνικές προδιαγραφές του kinect (http://msdn.microsoft.com/en-us/library/jj131033.aspx):

Kinect	Array Specifications
Viewing angle	43° vertical by 57° horizontal field of view
Vertical tilt range	±27°
Frame rate (depth and color stream)	30 frames per second (FPS)
Audio format	16-kHz, 24-bit mono pulse code modulation (PCM)
Audio input characteristics	A four-microphone array with 24-bit analog-to-digital converter (ADC) and Kinect-resident signal processing including acoustic echo cancellation and noise suppression
Accelerometer characteristics	A 2G/4G/8G accelerometer configured for the 2G range, with a 1° accuracy upper limit.

Γιατί είναι τόσο σημαντική η υπολογιστική όραση σε 3Δ;

Η πιο σημαντική καινοτομία που φέρνει το kinect είναι ότι επί της ουσίας πρόκειται για μια **κάμερα βάθους (depth camera)**. Με άλλα λόγια το kinect **μπορεί να εκτιμά με μεγάλη ακρίβεια την απόσταση κάθε σημείου από αυτό, κατ' αντιστοιχία με το ανθρώπινο μάτι**! Αν και υπάρχουν και άλλες κάμερες βάθους για στρατιωτικές εφαρμογές, το kinect είναι η πρώτη συσκευή που διατίθεται ευρέως με σκοπό να αναπτυχθούν εφαρμογές και υπηρεσίες για όλους.

Κάποιοι ερευνητές θεωρούν το kinect ως μια τεχνολογική ανακάλυψη αντίστοιχη του προσωπικού υπολογιστή και του παγκόσμιου ιστού! Ο λόγος είναι ότι με το kinect ο η/υ μπορεί να έχει στη διάθεση του μια συνεχή ροή δεδομένων βίντεο (και αναπαραστάσεων αυτών, π.χ. ανθρώπινες φιγούρες και σκελετοί, αν και αυτό ακόμα αναπτύσσεται - ραγδαία πάντως) του περιβάλλοντος σε 3Δ αντίστοιχη αυτή που προσφέρει το ανθρώπινο μάτι στον εγκέφαλο. Το kinect μπορεί να χρησιμοποιηθεί (και ήδη χρησιμοποιείται, ερευνητικά) για ευρεία γκάμα εφαρμογών όπως: **αναγνώριση χειρονομιών και στάσεων σώματος για κιναισθητική αλληλεπίδραση, βοηθητική τεχνολογία για ΑμΕΑ, αναγνώριση προσώπου (εκφράσεων, συναισθημάτων, εκφοράς λόγου, κ.α.), αναγνώριση**

αντικειμένων (και διαχείριση σκηνών, π.χ. για κινηματογραφία), ανάλυση κίνησης (για κινηματογραφία, παιχνίδια, κ.α.) και βαδίσματος, 3Δ ανίχνευση (scanning) αντικειμένων (για διαχείριση νεφών σημείων, ή/και μετάπειτα αναπαραγωγή και 3Δ εκτύπωση), 3Δ όραση σε ρομπότ (για αποφυγή εμποδίων), τηλεπαρουσία, κ.ο.κ. Πριν την έλευση του kinect όλες οι παραπάνω εφαρμογές αντιμετωπίζαν τρομερούς περιορισμούς επειδή χρησιμοποιούσαν 2Δ υπολογιστική όραση. Η σχετική έρευνα και τεχνολογική ανάπτυξη είναι ραγδαία σε όλες τις παραπάνω κατευθύνσεις με τη χρήση του kinect.

Προγραμματισμός σε Processing για αξιοποίηση των δεδομένων των καμερών βάθους (depth camera) και χρώματος (rgb camera)

Ανασκόπιση βασικών εννοιών: εικόνα, εικονοκύτταρο, ανάλυση εικόνας, βίντεο, ρυθμός ανανέωσης, χρώμα

Τι είναι μια εικόνα (image) για τον η/υ; Η σύντομη απάντηση είναι ότι είναι **ένα σύνολο από εικονοστοιχεία ή εικονοκύτταρα (pixels)** τα οποία διατάσσονται σε ένα **πίνακα 2Δ**. Η ανάλυση της εικόνας μας δίνει το σύνολο των pixels για μια εικόνα.

Έτσι, η μέγιστη ανάλυση 1920x960 των καμερών του kinect μας πληροφορεί ότι κάθε εικόνα που συλλαμβάνει η κάμερα απαρτίζεται από 1.843.200 pixels. Η αλήθεια είναι όμως ότι για να δείξουμε βίντεο σε ικανοποιητικό ρυθμό ανανέωσης (frame rate, frames per second, fps) δηλαδή σε 30 fps, θα πρέπει η κάμερα να ρυθμιστεί σε ανάλυση **640x480** (σε αυτήν την ανάλυση θα εργαστούμε παρακάτω) και ο συνολικός αριθμός των pixels/εικόνα είναι 307.200, για ένα δευτερόλεπτο βίντεο περίπου 9 εκ. pixels, κ.ο.κ.

Το pixel είναι το μικρότερο συστατικό μιας εικόνας και χαρακτηρίζεται από **τη θέση του** στη διάταξη της εικόνας και το **χρώμα** του.

Η θέση του είναι της μορφής **(x, y)**, αν χρησιμοποιούμε την 2Δ κάμερα και της μορφής **(x, y, z)** για την 3Δ κάμερα.

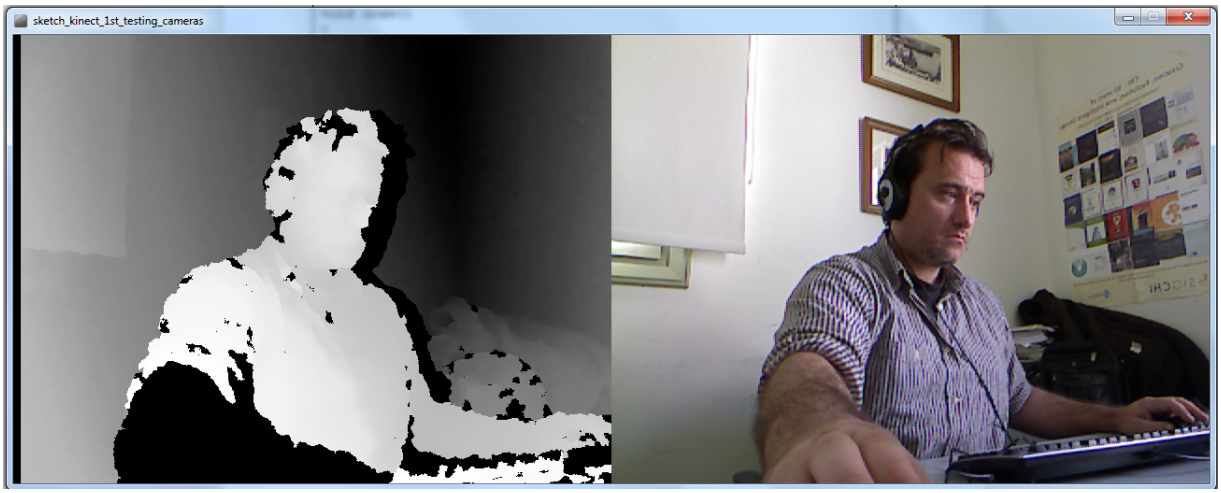
Το κυρίαρχο πρότυπο μοντελοποίησης του χρώματος στους η/υ είναι το **RGB (Red, Green, Blue)**, σύμφωνα με το οποίο κάθε χρώμα αποτελεί συνδυασμό των παραπάνω τριών χρωμάτων. Κάθε χρώμα αποθηκεύεται ως μια αριθμητική τιμή που κυμαίνεται στο [0, 255]. Έτσι, π.χ. το πλέον έντονο κόκκινο χρώμα που μπορεί να παραχθεί είναι το (255, 0, 0). Οι διαθέσιμες τιμές του παραπάνω διαστήματος είναι 256 (= 2 εις την 8η), και άρα απαιτούνται 8 bits (1 byte) για την αποθήκευση κάθε χρώματος.

Όταν μια εικόνα είναι ασπρόμαυρη (όπως αυτή της κάμερας βάθους του kinect), τότε μόνο ένας αριθμός στο [0,255] χρησιμοποιείται για να αναπαραστήσει μια **απόχρωση μεταξύ του άσπρου - μαύρου**. Το 0 είναι το μαύρο.

Διάβασμα των ροών βίντεο από τις 2 κάμερες του kinect και απεικόνιση τους στην οθόνη του η/υ

Το πρώτο πρόγραμμα που θα φτιάξουμε θα διαβάζει το βίντεο από τις δύο κάμερες του kinect και θα το δείχνει στην οθόνη του υπολογιστή μας. Ο κώδικας φαίνεται και επεξηγείται αμέσως παρακάτω:

```
import SimpleOpenNI.*; //eisagvgi paketoy (bibliothikis) kodika
SimpleOpenNI kinect; //dilosi metavlitis kinect

//h methodos setup arxikopoiei to programma (sketch)
void setup()
{
  size(640*2, 480); //to megethos tou kamva
  kinect = new SimpleOpenNI(this); //dimiourgia toy kinect sti
mnimi
  kinect.enableDepth(); //energopoihsi tis kameras vathous
  kinect.enableRGB(); //energopoihsi tis kameras xromatos
}

//h methodos trexei epanaliptika, sinexos
void draw()
{
  kinect.update(); //enimerone to kinect synexos me ta dedomena poy
katagrafoun oi kameres
  PImage depthImage = kinect.depthImage(); //pare tin eikona tis
kameras vathous
  PImage rgbImage = kinect.rgbImage(); //pare tin eikona tis
kameras xromatos
  image(depthImage, 0, 0); //deixe tin eikona tis kameras vathous
sto (0,0)
  image(rgbImage, 640, 0); //deixe tin eikona tis kameras xromatos
sto (640,0)
}
```



Παρατηρήσεις επί του αποτελέσματος (των 2 ροών βίντεο) του προγράμματος:
1. Περιορισμοί και δυνατότητες της **κάμερας βάθους** (η εικόνα αριστερά): η κάμερα βάθους δείχνει τα σημεία (pixels) σε κάποια **ασπρόμαυρη απόχρωση**. Για να καταλάβουμε το βάθος σε σχέση με την απόχρωση θα πρέπει να θυμόμαστε τα εξής:

(α) Όπως είπαμε και πριν, η εικόνα έχει ανάλυση **640x480** pixels (σημεία) και συνολικό αριθμό 307.200 pixels / frame. Επίσης ο ρυθμός ανανέωσης είναι 30 fps.

(β) Όποια **σημεία είναι σε απόσταση μικρότερη των 20 ιντσών** ή περίπου μισό μέτρο (1 inch = 2,54 cm) η κάμερα βάθους **δεν τα διακρίνει** (γι αυτό είναι μαύρα).

(γ) Από την απόσταση του μισού περίπου μέτρου και έπειτα, **η κάμερα βάθους βλέπει σε απόσταση μέχρι περίπου 25 πόδια ή περίπου 7,62 μέτρα** (1 πόδι = 0,3 μέτρα). Όσα σημεία είναι μακρύτερα από αυτήν την απόσταση επίσης φαίνονται μαύρα.

(δ) Για τα υπόλοιπα σημεία, **όσο πιο λευκό είναι το σημείο, τόσο κοντύτερα βρίσκεται στη κάμερα βάθους** του kinect.

(ε) Θόρυβος στο περίγραμμα. Παρατηρείστε την εικόνα της κάμερας βάθους: στο **περίγραμμα των αντικειμένων υπάρχουν μαύρα σημεία**. Η κάμερα βάθους συλλαμβάνει στα σημεία επειδή φωτίζονται από τον προβολέα υπερύθρων (IR emmiter) και αντανακλάται το φώς πίσω στη κάμερα, αλλά **στις άκρες των αντικειμένων το φως δεν αντανακλάται προς τη κάμερα, αλλά προς άλλες κατευθύνσεις του χώρου.** Όταν δεν υπάρχει αντανάκλαση σημείων του χώρου στη κάμερα, τα σημεία απλά δεν καταγράφονται και σημειώνονται ως μαύρα. Για να αντιμετωπιστεί το πρόβλημα, θα πρέπει κάποιος να έχει 2 ή περισσότερα kinect και τα αντικείμενα να είναι σταθερά (χρήσιμο για εφαρμογές 3D scanning).

(στ) **Διαφορά ευθυγράμμισης των εικόνων των 2 καμερών** του kinect: Αν παρατηρήσετε προσεκτικά τις 2 φωτογραφίες θα διαπιστώσετε ότι υπάρχει μια μικρή διαφορά στην ευθυγράμμιση των εικόνων που οφείλεται στο ότι η κάθε εικόνα συλλαμβάνεται από διαφορετική κάμερα. Αυτό σημαίνει ότι κάθε pixel της εικόνας που δείχνει η κάμερα βάθους αντιστοιχεί σε διαφορετικό pixel της κάμερας χρώματος. Οι εφαρμογές που θέλουν να χρησιμοποιήσουν και τις 2 κάμερες, θα πρέπει να λαμβάνουν υπόψη τους αυτή τη διαφοροποίηση. Θα το δούμε αυτό παρακάτω σε εφαρμογή που αφαιρεί το υπόβαθρο και το αντικαθιστά με φωτογραφία.

Κάποιες παρατηρήσεις επί του κώδικα:
1. **Χρειάζονται ελάχιστες γραμμές κώδικα** στην Processing (με τη χρήση της βιβλιοθήκης SimpleOpenNI) για να έχουμε πρόσβαση στη ροή βίντεο του kinect - αν χρησιμοποιούσαμε οποιαδήποτε άλλη γλώσσα προγραμματισμού θα χρειαζόμασταν 10δες γραμμές εντολών για να αρχικοποιήσουμε τη συσκευή, να ανοίξουμε κάθε μία ροή βίντεο, να την δείξουμε στην οθόνη, κλπ.
2. Τις εντολές που είναι σημειωμένες με κίτρινο θα τις χρησιμοποιούμε σε κάθε πρόγραμμα μας με αυτή τη μορφή.
3. Τις υπόλοιπες θα τις χρησιμοποιούμε πολύ συχνά, αλλά ίσως όχι σε αυτή τη μορφή ακριβώς...

Επίδειξη του χρώματος ενός pixel

Για να δούμε το χρώμα ενός Pixel της εικόνας που διαβάζει το kinect, θα πρέπει να υλοποιήσουμε **τη μέθοδο mousePressed()** της Processing. Μόλις ο χρήστης πατήσει σε κάποιο σημείο της εικόνας θα πρέπει:
(α) Να διαβάσουμε το χρώμα (στο σημείο mouseX, mouseY).
(β) Να τυπώσουμε τα r, g, b.

Ο παρακάτω κώδικας τυπώνει το χρώμα στην κονσόλα της Processing.

```
import SimpleOpenNI.*;
SimpleOpenNI kinect;

void setup(){
  size(640*2, 480);
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();
  kinect.enableRGB();
}

void draw(){
  kinect.update();
```

```
  PImage depthImage = kinect.depthImage();
  PImage rgbImage = kinect.rgbImage();
  image(depthImage, 0, 0);
  image(rgbImage, 640, 0);
}

void mousePressed(){
  color c = get(mouseX, mouseY); //apothieyse to xroma sto c
  println("red: " + red(c) + ", green: " + green(c) + ", blue: " +
blue(c));
  //grapse ta r, g, b, sti konsola
}
```

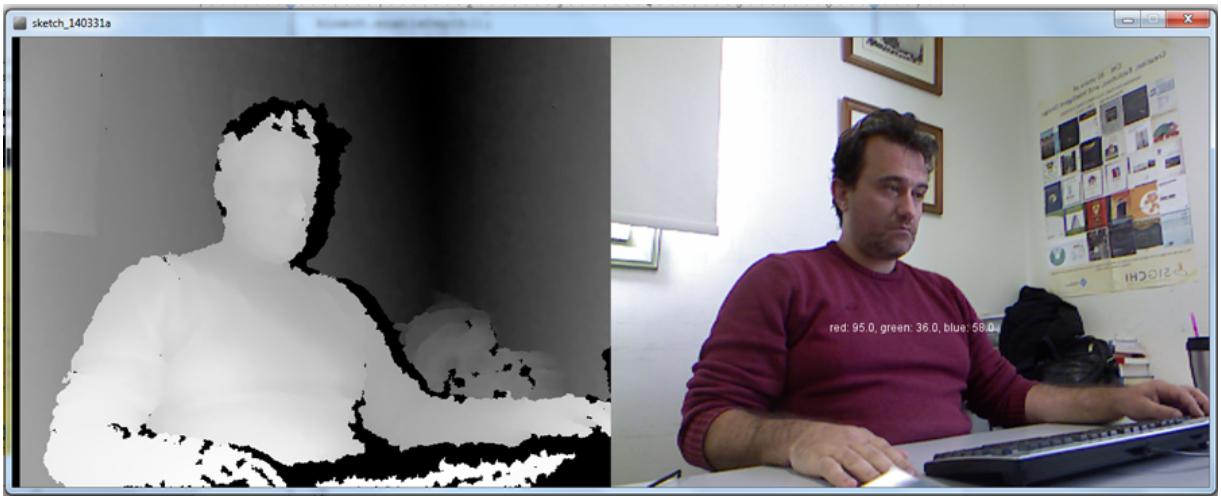
Για να τυπώσουμε το χρώμα στο σημείο που πάτησε ο χρήστης πρέπει να κάνουμε επιπλέον τα εξής:
(γ) να δείχνουμε το μήνυμα επί της εικόνας με τη χρήστη της εντολής text(), αντί της println().

Άσκηση
Δοκιμάστε το, αντικαθιστώντας την println(...), με την text(...);

Τι παρατηρείτε; Το μήνυμα εμφανίζεται στιγμιαία και έπειτα εξαφανίζεται. Αυτό συμβαίνει επειδή η draw() καλείται αυτόματα αμέσως για να ανανεώσει το βίντεο και στην επόμενη εικόνα το μήνυμα έχει χαθεί. Ένας γρήγορος τρόπος για να εμφανιστεί το μήνυμα σωστά, είναι να “παγώσετε” το βίντεο όταν ο χρήστης πατήσει το ποντίκι, και να το “ξεπαγώνετε” μόλις το αφήσει. Για να το κάνετε:
(δ) προσθέστε την εντολή noLoop() στο τέλος της mousePressed(). Η εντολή έχει ως αποτέλεσμα να μην κληθεί ξανά η draw().
(e) Για να κληθεί ξανά θα πρέπει να χρησιμοποιήσετε την εντολή loop(). Καλέστε την, εντός του σώματος της μεθόδου mouseReleased(), υλοποιώντας τη.

```
import SimpleOpenNI.*;
SimpleOpenNI kinect;

void setup(){
  size(640*2, 480);
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();
  kinect.enableRGB();
}

void draw(){
  kinect.update();
  PImage depthImage = kinect.depthImage();
  PImage rgbImage = kinect.rgbImage();
```

```
  image(depthImage, 0, 0);
  image(rgbImage, 640, 0);
}

void mousePressed(){
  color c = get(mouseX, mouseY); //apothieyse to xroma sto c
  String message = "red: " + red(c) + ", green: " + green(c) + ",
blue: " + blue(c);
  //apothikeyse to minima sto message
  text(message, mouseX, mouseY); //typose to minima sto simeio poy
patise o xristis
  noLoop(); //"pagose" to video
}

void mouseReleased(){
  loop(); //"xepagose" to video
}
```



Δοκιμάστε το πρόγραμμα σας και παρατηρείστε τις τιμές που σας επιστρέφονται για κάθε pixel κάθε μιας από τις 2 εικόνες. Μερικές παρατηρήσεις είναι οι εξής:
1. Κάθε τιμή RGB της ασπρόμαυρης εικόνας έχει ίδιες τιμές για τα r, g, b. Π.χ. αν χτυπήσετε στο μαύρο θα δείτε r=0, g=0, b=0, σε κάποιο άλλο γκρίζο r=50, g=50, b=50, κ.ο.κ.. Όπως έχουμε πει και στα προηγούμενα, αυτό είναι φυσιολογικό μιας και οι τιμές εδώ κυμαίνονται στο ασπρόμαυρο φάσμα.
2. Κάθε τιμή της έγχρωμης εικόνας μας δίνει τα rgb. Σε κάποιες περιπτώσεις ίσως είναι διαφορετικό το αποτέλεσμα από αυτά που περιμέναμε. Αυτό συμβαίνει επειδή η αντίληψη του χρώματος είναι υποκειμενική, και εξαρτάται από τα γύρω χρώματα σε μεγάλο βαθμό.

Επίδειξη της απόστασης ενός pixel από το kinect

Όπως έχουμε αναφέρει στα προηγούμενα, η κάμερα βάθους του kinect αντιστοιχεί κάποιο χρώμα στο ασπρόμαυρο φάσμα σε κάθε pixel κατ΄αναλογία της απόστασης του από την κάμερα. Επίσης, αναφέραμε ότι το kinect “βλέπει” 3Δ εικόνες κατ΄αντιστοιχία με το ανθρώπινο μάτι, και ότι αυτό αποτελεί μια εξαιρετική τεχνολογική δυνατότητα που ανοίγει το δρόμο για πλήθος νέων εφαρμογών. Βεβαίως το kinect δεν πλησιάζει ούτε κατά διάνοια την αντιληπτική ικανότητα του ανθρώπινου ματιού, αφού π.χ. ως προς την απόσταση όρασης μπορεί να “δει” στο διάστημα 0,5m - 7,6m, και όχι κοντύτερα ή μακρύτερα.

Στο επόμενο παράδειγμα θα γράψουμε πρόγραμμα που να υπολογίζει την απόσταση κάθε pixel από το kinect, ώστε για κάθε pixel που επιλέγει ο χρήστης με το ποντίκι να εκτυπώνεται η απόσταση του από το kinect. Τα σημαντικά στοιχεία που πρέπει να ξέρουμε για να το πετύχουμε αυτό είναι:

1. **Για κάθε frame, το kinect αποθηκεύει τις αποστάσεις των σημείων από τη κάμερα βάθους σε ένα μονοδιάστατο πίνακα** που μπορεί να αποκτηθεί από το πρόγραμμα μας καλώντας τη μέθοδο **`kinect.depthMap()`**.
2. Αφού η εικόνα έχει 2Δ, πως αποθηκεύονται οι τιμές για κάθε pixel σε μονοδιάστατο πίνακα; Ως εξής: **κατά σειρά** ξεκινώντας από το pixel στη θέση (0,0), έπειτα (0,1), … (0, 639), (1, 0), (1, 1), … κ.ο.κ.

Θέση πίνακα	Εικονοκύτταρο - θέση (x,y)	Σχόλιο
0	0,0	πάνω αριστερά pixel
1	1,0	δεύτερο (προς x) πάνω αριστερά pixel
2	2, 0	τρίτο (προς x) πάνω αριστερά pixel
...	...	...
639	639, 0	τελευταίο (προς x) πάνω pixel
640	1, 0	δεύτερο (προς y), πρώτο (προς χ) pixel
641	1, 1	δεύτερο (προς y), δεύτερο (προς χ) pixel
...	...	...

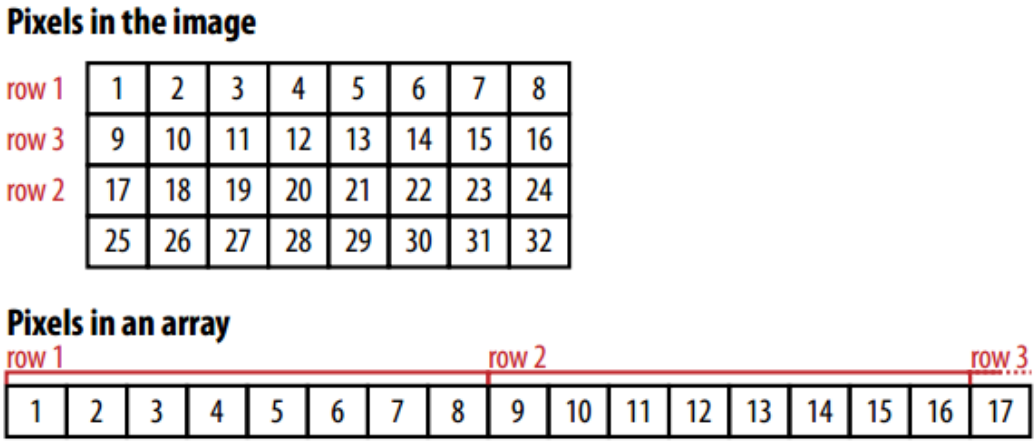


Figure 2-12. *Pixels in a two-dimensional image get stored as a flat array. Understanding how to split this array back into rows is key to processing images.*

Έστω ότι θέλουμε να αναφερθούμε στο 5ο pixel στον άξονα των x, και το 120ο του άξονα y. Θα πρέπει να πολλαπλασιάσουμε τα y με το 640, και να προσθέσουμε τα x, η θέση είναι δηλαδή η (x + y*640)**.** Άρα είναι το pixel στη θέση [4+120*640] (η πρώτη θέση του πίνακα έχει τιμή 0, και η 5η τιμή 4).

Για να βρούμε τη σωστή τιμή στο μονοδιάστατο πίνακα αποστάσεων που μας επιτρέφεται από τη μέθοδο `kinect.depthMap()`, θα πρέπει να εργαστούμε στο πρόγραμμα μας με αντίστοιχο τρόπο. Δηλαδή **να ορίσουμε μια ακέραια μεταβλητή (έστω clickPosition) η οποία να μετατρέπει τα (mouseX, mouseY) στο σωστό ακέραιο αριθμό.** Αυτό θα το κάνουμε με την παρακάτω εντολή:

```
int clickPosition = mouseX + mouseY*640;
```

Χρησιμοποιώντας τα παραπάνω στοιχεία, το παρακάτω πρόγραμμα τυπώνει στο σημείο που πατάει ο χρήστης με το ποντίκι την απόσταση του σημείου από τη κάμερα βάθους του kinect. Δοκιμάστε το πατώντας σε διάφορα σημεία της εικόνας.

```
import SimpleOpenNI.*; //eisagvgi paketoy kodika
SimpleOpenNI kinect; //dilosi metavlitis

//h methodos setup arxikopoiei to programma moy (sketch)
void setup()
{
  size(640, 480); //to megethos tou kamva
  kinect = new SimpleOpenNI(this); //dimiourgia toy kinect sti
mnimi
  kinect.enableDepth(); //energopoihsi tis kameras vathous
}
```

```
//h methodos trexei epanaliptika, sinexos
void draw()
{
  kinect.update(); //enimerone to kinect synexos me ta dedomena poy
katagrafoun oi kameres
  PImage depthImage = kinect.depthImage(); //get the depth image
into a PImage
  image(depthImage, 0, 0); //deixe tin eikona tis kameras vathous
sto (0,0)
}

//h methodos kaleitai opote o xristis pataei me to pontiki
void mousePressed(){
  //to depthMap epistrefei tis apostaseis olon ton simeion tou
image apo to kinect!
  int [] depthValues = kinect.depthMap();
  //epeidh einai monodiastatos pinakas, prepei na anafertho se
auton pol/ntas to y me 640
  int clickPosition = mouseX + (mouseY * 640);
  //h apostasi einai se milimeters
  int clickDepth = depthValues[clickPosition];
  //ektyposi se kokkino, gia na fainetai
  fill(255, 0, 0);
  text("milimeters: " + clickDepth, mouseX, mouseY);
  noLoop(); //freeze the video
}

void mouseReleased(){
  loop(); //continue with video grabbing
}
```

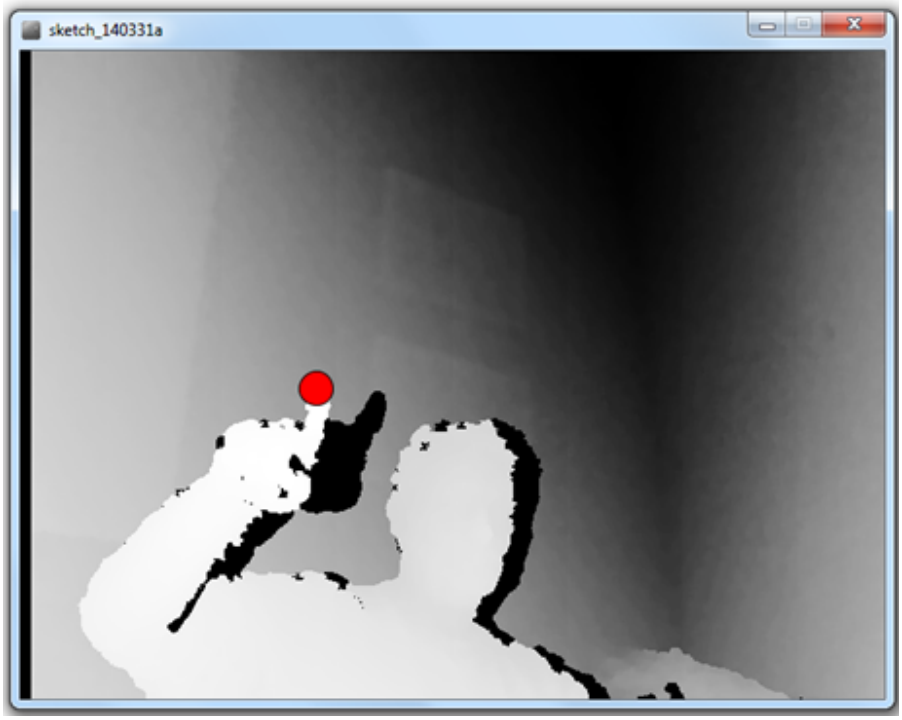
Επειδή οι παραπάνω εντολές μπο ρεί να χρησιμοποιούνται συχνά στα προγράμματα σας, μπορείτε να φτιάξετε μια μέθοδο int pixelDepth(int x, int y); για να τις καλείτε ως εξής.

```
int pixelDepth(int x, int y){
  int[] depthValues = kinect.depthMap();
  int clickPosition = mouseX + (mouseY * 640);
  int clickDepth = depthValues[clickPosition];
  return clickDepth;
}
```

Επίδειξη του κοντινότερου σημείου στο kinect

Στο προηγούμενο παράδειγμα είδαμε πως μπορούμε να έχουμε πρόσβαση στην απόσταση κάθε pixel από την κάμερα βάθους του kinect. Τώρα θα δούμε πως μπορούμε να παρακολουθούμε το χρήστη ώστε να εντοπίζουμε το κοντινότερο σημείο του στο kinect,

υλοποιώντας με τον πιο απλό τρόπο κάποια χειρονομία δειξίματος (pointing).



Εδώ θα χρειαστεί να φτιάξουμε έναν μικρό αλγόριθμο, σχεδιάζοντας το πρόγραμμα μας σε ψευδοκώδικα:

```
Πάρε το πίνακα βάθους του kinect (depth array);

//οι παρακάτω εντολές διατρέχουν όλα τα pixels της εικόνας!
Για κάθε γραμμή του πίνακα βάθους
      Για κάθε pixel στη γραμμή του πίνακα βάθους
            Πάρε την απόσταση του από το kinect;
            Αν η απόσταση είναι η μικρότερη ως τώρα
                  Αποθήκευσε την απόσταση;
                  Αποθήκευσε το σημείο (mouseX, mouseY);

//πλέον βρέθηκε το κοντινότερο σημείο
Ζωγράφισε το depth image;
Ζωγράφισε ένα κόκκινο κύκλο στο κοντινότερο pixel (mouseX, mouseY);
```

```
import SimpleOpenNI.*;
SimpleOpenNI kinect;

//se autes tis metavlites tha apothikeuoume kathe fora to
kontinotero pixel
```

```
int closestValue, closestX, closestY;

void setup()
{
  size(640, 480);
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();
}

void draw()
{
  closestValue = 8000; //~8m einai h megisti apostasi orasis tou
kinect!
  kinect.update(); //ananeosi eikonas

  //pare to pinaka vathous
  int[] depthValues = kinect.depthMap();

  //gia kathe grammi toy pinaka vathous...
  for (int x = 0; x < 640; x++) {
    //gia kathe pixel tis grammis...
    for (int y = 0; y < 480; y++) {
      //Pare tin apostasi tou apo to kinect
      int position = x + y * 640; //thesi toy pixel sto
monodiastato pinaka
      int currentDepthValue = depthValues[position]; //apostasi
      //An auti i apostasi einai i kontinoteri mexri stigmis...
      if (currentDepthValue > 0 && currentDepthValue <=
closestValue) {
        //Apothikeuse tin apostasi
        closestValue = currentDepthValue;
        //Apothikeuse to simeio
        closestX = x;
        closestY = y;
      }
    } //synexise gia oles ta ta pixels tis grammis
  } //synexise gia oles tis grammes

  //Zografise to depth image stin othoni
  image(kinect.depthImage(),0,0);
  // Zografise ena mikro kokkino kyklo sto kontinotero simeio
  fill(255,0,0);
  ellipse(closestX, closestY, 25, 25);
}
```

Ερώτηση (επανάληψης): Γιατί ελέγχουμε στο if αν η currentValue είναι μεγαλύτερη του μηδέν;

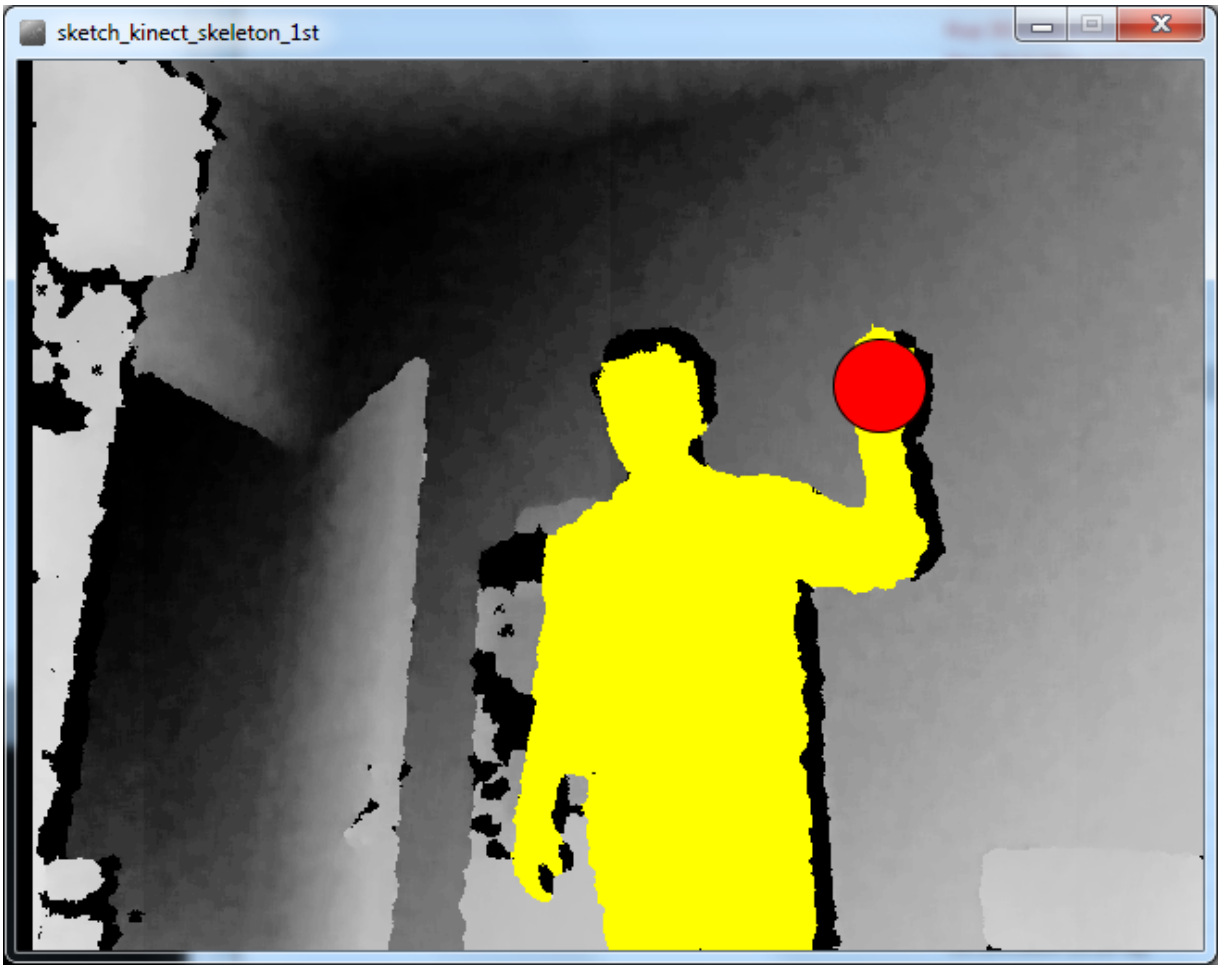
Ποια είναι αμέσως μεγαλύτερη τιμή που μπορεί να πάρει η currentValue μετά το 0;

Προγραμματισμός σε Processing για αξιοποίηση των δεδομένων των κινήσεων του σώματος των χρηστών (skeleton data, centre of mass, userImage(), userPixels[], ...)

Στο προηγούμενο κεφάλαιο εργαστήκαμε σε προγράμματα που επεξεργάζονται τα σημεία (pixels) που μας παρέχει η κάμερα βάθους του kinect. **Για να κάνουμε τις εφαρμογές μας διαδραστικές έπρεπε να "σαρώσουμε" όλα τα pixels της εικόνας, μέχρι να βρούμε αυτό που μας ενδιαφέρει (π.χ. το κοντινότερο στην κάμερα).** Σε αυτό το κεφάλαιο θα ακολουθήσουμε ένα πιο άμεσο δρόμο για να κάνουμε τις εφαρμογές μας πιο διαδραστικές: θα δούμε πως μπορούμε να έχουμε πρόσβαση, μέσω της βιβλιοθήκης SimpleOpenNI, σε πληροφορίες σχετικά με φιγούρες χρηστών και σημεία του σώματος τους που μπορεί να μας ενδιαφέρουν για τις εφαρμογές μας.

Ανίχνευση του χεριού του χρήστη

Ο σκοπός είναι να γράψουμε ένα πρόγραμμα που θα ανιχνεύει το δεξί χέρι του χρήστη και θα εμφανίζει ένα κόκκινο κύκλο σε αυτό που θα είναι "κολλημένος" στο χέρι του συνεχώς. Αν και εκ πρώτης όψεως, το πρόγραμμα μας μπορεί να μοιάζει με το αμέσως προηγούμενο, όταν εμφανίζαμε το κοντινότερο pixel στη κάμερα που ήταν το χέρι του χρήστη μόνο όταν αυτός έδειχνε προς τα εμπρός, αλλιώς θα μπορούσε να είναι οποιοδήποτε άλλο σημείο της σκηνής.



Πριν προχωρήσουμε να δούμε τον κώδικα του προγράμματος που λύνει το παραπάνω πρόβλημα θα πρέπει να έχουμε υπόψη μας, ότι θα πρέπει να εκτελέσουμε κάποια βήματα, ως εξής:

1. **Ενεργοποίηση της ανίχνευσης χρηστών**. Αρχικά, στη μέθοδο `setup()`, χρησιμοποιούμε τη μέθοδο `kinect.enableUser();` για να ενεργοποιήσουμε τη δυνατότητα του kinect να ανιχνεύει χρήστες - κατ' αντιστοιχία με τη μέθοδο `kinect.enableDepth();` που ενεργοποιεί τη κάμερα βάθους.
2. **Δείχνοντας τις σιλουέτες των χρηστών.** Εντός της μεθόδου `draw()`, παίρνουμε από το kinect τη σιλουέτα του χρήστη (για την ακρίβεια τις σιλουέτες όλων των χρηστών, μέχρι 6, της σκηνής) με την εντολή `PImage userImage = kinect.userImage();` και τη δείχνουμε με την εντολή `image(userImage, 0, 0);` Πάλι, κάνουμε ακριβώς ότι θα κάναμε για να δείξουμε το `depthImage`, όπως κάναμε μέχρι τώρα σε όλα τα προηγούμενα.
3. **Διαβάζοντας από το kinect τον αριθμό των χρηστών.** Το κάνουμε με την εντολή `int[] userList = kinect.getUsers();` Σε κάθε μια θέση του πίνακα `userList[i]` αποθηκεύεται ένας ακέραιος αριθμός που είναι η "ταυτότητα" του χρήστη.

Ουσιαστικά πρόκειται για έναν αύξοντα από σιλουέτες που μπαίνουν στη σκηνή. Το kinect προσπαθεί να εκτιμήσει αν ένας χρήστης που εισέρχεται στη σκηνή ήταν και προηγουμένως, αν όχι τότε αυξάνει τον αριθμό των χρηστών κατά 1.

4. **Ελέγχοντας αν το kinect ανιχνεύει το σκελετό των χρηστών της σκηνής.** Πριν κάνουμε οτιδήποτε σχετικό με το σκελετό (όπως αυτό που θέλουμε τώρα, δηλαδή να εντοπίσουμε το δεξί χέρι και να του "κολλήσουμε" ένα κόκκινο κύκλο), πρέπει να ελέγξουμε αν όντως το kinect ανιχνεύει το σκελετό! Σημειώστε ότι είναι πιθανό το kinect να έχει ανιχνεύσει χρήστη, χωρίς να καταφέρει το ίδιο για το σκελετό του: π.χ. όταν ο χρήστης εισέρχεται στη σκηνή (γενικά δεν είναι δύσκολο να υποθέσει το kinect ότι: όταν ένα (μεγάλο σχετικά) κινούμενο αντικείμενο εισέρχεται στη σκηνή, αυτό είναι άνθρωπος - βλ. εικόνα). Γι αυτό πρέπει να εσωκλείσουμε το κώδικα σχετικά με το σκελετό σε μια συνθήκη της μορφής: `if (kinect.isTrackingSkeleton(userList[i])) { ... }`

5. **Παίρνοντας τη θέση της άρθρωσης του δεξιού χεριού.** Η βιβλιοθήκη simpleOpenNI μας τοποθετεί στη μεταβλητή `PVector rightHand` τη θέση της άρθρωσης (joint) του δεξιού χεριού. Ο τύπος δεδομένων PVector μας δίνει το 3Δ σημείο της μορφής (x, y, z). Με την (μακρυά...) εντολή `float confidence = kinect.getJointPositionSkeleton(userList[i], SimpleOpenNI.SKEL_RIGHT_HAND, rightHand);` παίρνει τιμή το `rightHand`. Αν παρατηρήσετε προσεκτικά την εντολή θα διαπιστώσετε ότι προσδιορίζουμε τον χρήστη και το είδος της άρθρωσης, και ότι επιστρέφεται μια δεκαδική τιμή εμπιστοσύνης με την οποία το kinect απαντάει στο ερώτημα του αν έχει τη συγκεκριμένη άρθρωση.
6. **Μετατρέποντας τη 3Δ θέση της άρθρωσης σε 2Δ προβολή** (ώστε να δείξουμε τη προβολή στην οθόνη του η/υ). Με την εντολή `kinect.convertRealWorldToProjective(rightHand, convertedRightHand);` τοποθετείται στη μεταβλητή `PVector convertedRightHand` η προβολή της rightHand ώστε στη συνέχεια να τη ζωγραφίσουμε στην οθόνη (βλ. σχήμα).

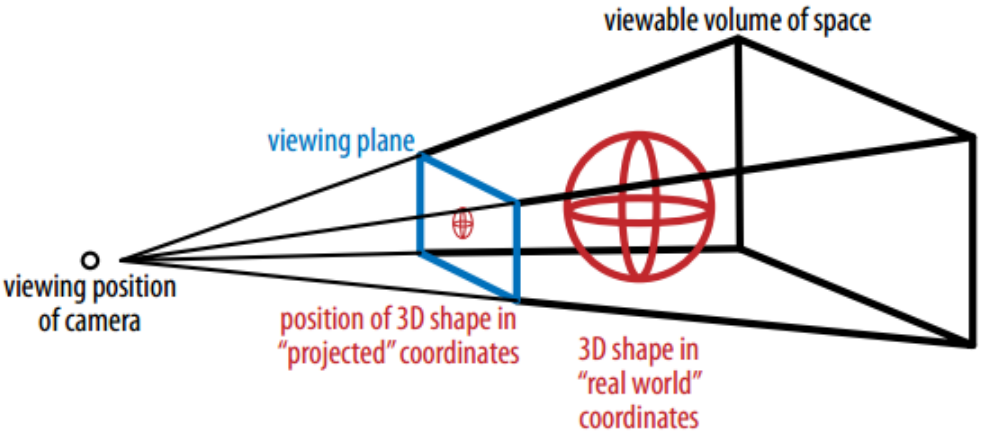


Figure 4-5. *The relationship between projective and real-world coordinates. Real-world coordinates locate the object in 3D space. Projective coordinates describe where you'd see it on the viewing plane.*

7. **User tracking callback methods**. Όταν θέλουμε να ανιχνεύσουμε τη σιλουέτα (και εν συνεχεία) το σκελετό του χρήστη, τότε πρέπει να δηλώσουμε 3 μεθόδους που καλούνται αυτόματα (callback methods) από την βιβλιοθήκη simpleOpenNI. Αυτές είναι οι:
   a. `void onNewUser(SimpleOpenNI curContext, int userId) //καλείται αυτόματα, μόλις το kinect εντοπίσει ένα νέο χρήστη`
   b. `void onLostUser(SimpleOpenNI curContext, int userId) //καλείται αυτόματα, μόλις το kinect χάσει το χρήστη από το οπτικό του πεδίο`
   c. `void onVisibleUser(SimpleOpenNI curContext, int userId) // καλείται αυτόματα, κάθε φορά που ο χρήστης είναι ορατός από το kinect`

Για τους σκοπούς του προγράμματος μας, μόνο στην πρώτη χρειάζεται να καλέσουμε τη μέθοδο `curContext.startTrackingSkeleton(userId)` ώστε να ξεκινήσει η ανίχνευση του σκελετού. Στις άλλες μεθόδους προς το παρόν δεν χρειάζεται να κάνουμε κάτι.

```
import SimpleOpenNI.*;
SimpleOpenNI kinect;

void setup() {
  size(640, 480);
  //start the kinect
  kinect = new SimpleOpenNI(this);
  //enable the depth camera
  kinect.enableDepth();
  //enable user tracking
  kinect.enableUser(); // (1)
}
```

```
void draw() {
  //update the image from the kinect
  kinect.update();
  //get the user image (not the depth image)
  PImage userImage = kinect.userImage(); // (2)
  image(userImage, 0, 0); //show the user image
  //get the list of users from the kinect
  int[] userList = kinect.getUsers(); // (3)
  //for each user in the user list
  for(int i=0;i<userList.length;i++) {
    //if the kinect is tracking the user skeleton
    if (kinect.isTrackingSkeleton(userList[i])) { // (4)
        //make a vector to store the user's left hand (hand on the
right of the screen)
        PVector rightHand = new PVector();
        //put the position of the user's left hand into that vector
        float confidence =
kinect.getJointPositionSkeleton(userList[i],
        SimpleOpenNI.SKEL_RIGHT_HAND, rightHand); // (5)
        //convert the detected hand position to "projective"
coordinates that will match the depth image
        PVector convertedRightHand = new PVector();
        kinect.convertRealWorldToProjective(rightHand,
convertedRightHand); // (6)
        // and display it
        fill(255,0,0);
        ellipse(convertedRightHand.x, convertedRightHand.y, 50,
50);
      }
   }
}

// user-tracking callbacks!
void onNewUser(SimpleOpenNI curContext, int userId) { // (7)
  //just a notification message
  println("onNewUser - userId: " + userId  + ". Start tracking
skeleton");
  //this is the only method needed in these callbacks for now
  //when the kinect spots a new user, it has to start tracking of
his skeleton
  curContext.startTrackingSkeleton(userId); // (7)
}

void onLostUser(SimpleOpenNI curContext, int userId) // (7)
{
  //just a notification message
```

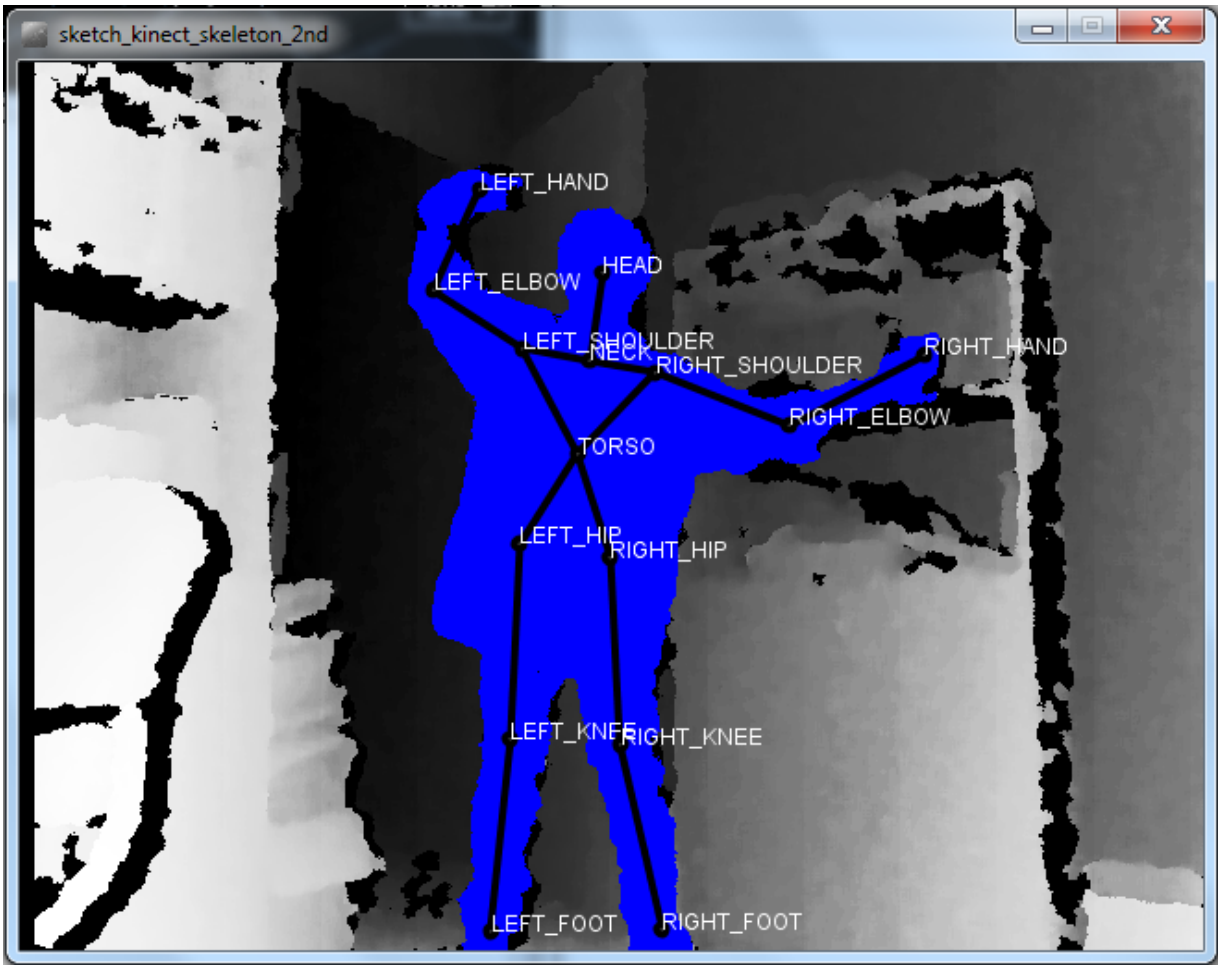
```
  println("onLostUser - userId: " + userId);
}

void onVisibleUser(SimpleOpenNI curContext, int userId) // (7)
{
  //just a notification message
  println("onVisibleUser - userId: " + userId);
}
```

Πρόσβαση στα δεδομένα σκελετού (skeleton data) των χρηστών της σκηνής και ζωγραφική τους στη ροή βίντεο

Τα δεδομένα σχετικά με τις ανθρώπινες σιλουέτες ονομάζονται στη γλώσσα του kinect, δεδομένα σκελετού (skeleton data). Τα δεδομένα σκελετού είναι απαραίτητα για πάρα πολλές διαδραστικές εφαρμογές, όπως και για ανίχνευση χειρονομιών και στάσεων του σώματος. Γιατί δεν ξεκινήσαμε να εργαζόμαστε με αυτά δεδομένα σκελετού απευθείας; Επειδή, παρότι ίσως είναι πιο απλό εννοιολογικά, είναι πιο δύσκολο σε επίπεδο προγραμματισμού. Αρχικά θα δούμε ποια είναι τα δεδομένα του σκελετού και στη συνέχεια θα φτιάξουμε προγράμματα για να χρησιμοποιήσουμε τα δεδομένα αυτά.

Το kinect μπορεί να αναγνωρίσει μέχρι το πολύ 6 σιλουέτες στη σκηνή. Η βιβλιοθήκη SimpleOpenNI μπορεί να αναγνωρίσει **15 αρθρώσεις (joints)** του σκελετού των **2 από τους 6 ανθρώπους της σκηνής**. Σημειωτέον ότι το Kinect SDK 1.8 αναγνωρίζει 20 αρθρώσεις: επιπλέον αυτών που φαίνονται στο σχήμα, αναγνωρίζει καρπούς χεριών (left/right wrist), αστραγάλους (left/right) και κεντρικό ισχύο (hip centre. Το Kinect 2.0 αναγνωρίζει 25: επιπλέον 2 αντίχειρες, 2 μεγάλα δάκτυλα χεριού, 1 λαιμό).

Οι αρθρώσεις των δεδομένων σκελετού του kinect δεν αντιστοιχούν στις πραγματικές αρθρώσεις του ανθρώπινου σκελετού: αυτό έχει συμβεί για να διευκολυνθεί η ανίχνευση του σκελετού από την κάμερα βάθους. Π.χ. ο κορμός (torso) δεν είναι άρθρωση, είναι όμως ένα από τα πλέον συνηθισμένα σημεία που ανιχνεύει το kinect καθώς ο χρήστης κινείται και αλλάζει στάση του σώματος - σκεφτείτε ότι είναι πολύ πιθανό ο χρήστης να καθίσει, να σκύψει, να φαίνεται από τη μέση και πάνω, να βάλει τα χέρια του πίσω από το κορμό του, κ.α. Με βάση τις αρθρώσεις που προσφέρονται από τη βιβλιοθήκη simpleOpenNI μπορούμε να ζωγραφίσουμε τα **μέρη του σώματος (limbs)** με γραμμές που συνδέουν κοντινές αρθρώσεις. Οι αρθρώσεις συνδεδεμένες φαίνονται στο παρακάτω σχήμα:



Ο στόχος του προγράμματος μας είναι να ζωγραφίσουμε το σκελετό του χρήστη. Ο κώδικας του προγράμματος φαίνεται αμέσως παρακάτω, και έπειτα εξηγείται.

```
import SimpleOpenNI.*;
SimpleOpenNI  kinect;
void setup()
{
  size(640,480);
  kinect = new SimpleOpenNI(this);
  // enable depthMap generation
  kinect.enableDepth();
  // enable user tracking
  kinect.enableUser();
}
void draw()
{
  // update the cam
```

```
  kinect.update();
  // draw userImage
  image(kinect.userImage(),0,0);
  // get the user list
  int[] userList = kinect.getUsers();
  // for each user in the scene
  for(int i=0;i<userList.length;i++)
  {  // if the skeleton is being tracked
    if(kinect.isTrackingSkeleton(userList[i]))
    {
      stroke(0);
      strokeWeight(4);
      drawSkeleton(userList[i]); //draw the skeleton
    }
  }
}
// draw the skeleton with the joints provided by the kinect
void drawSkeleton(int userId)
{
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_HEAD,
SimpleOpenNI.SKEL_NECK);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_NECK,
SimpleOpenNI.SKEL_LEFT_SHOULDER);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_SHOULDER,
SimpleOpenNI.SKEL_LEFT_ELBOW);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_ELBOW,
SimpleOpenNI.SKEL_LEFT_HAND);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_NECK,
SimpleOpenNI.SKEL_RIGHT_SHOULDER);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_SHOULDER,
SimpleOpenNI.SKEL_RIGHT_ELBOW);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_ELBOW,
SimpleOpenNI.SKEL_RIGHT_HAND);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_SHOULDER,
SimpleOpenNI.SKEL_TORSO);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_SHOULDER,
SimpleOpenNI.SKEL_TORSO);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_TORSO,
SimpleOpenNI.SKEL_LEFT_HIP);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_HIP,
SimpleOpenNI.SKEL_LEFT_KNEE);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_KNEE,
SimpleOpenNI.SKEL_LEFT_FOOT);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_TORSO,
SimpleOpenNI.SKEL_RIGHT_HIP);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_HIP,
SimpleOpenNI.SKEL_RIGHT_KNEE);
```

```
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_KNEE,
SimpleOpenNI.SKEL_RIGHT_FOOT);
  drawJoint(userId, SimpleOpenNI.SKEL_HEAD);
  drawJoint(userId, SimpleOpenNI.SKEL_NECK);
  drawJoint(userId, SimpleOpenNI.SKEL_LEFT_SHOULDER);
  drawJoint(userId, SimpleOpenNI.SKEL_RIGHT_SHOULDER);
  drawJoint(userId, SimpleOpenNI.SKEL_LEFT_ELBOW);
  drawJoint(userId, SimpleOpenNI.SKEL_RIGHT_ELBOW);
  drawJoint(userId, SimpleOpenNI.SKEL_TORSO);
  drawJoint(userId, SimpleOpenNI.SKEL_LEFT_HIP);
  drawJoint(userId, SimpleOpenNI.SKEL_RIGHT_HIP);
  drawJoint(userId, SimpleOpenNI.SKEL_LEFT_KNEE);
  drawJoint(userId, SimpleOpenNI.SKEL_RIGHT_KNEE);
  drawJoint(userId, SimpleOpenNI.SKEL_LEFT_FOOT);
  drawJoint(userId, SimpleOpenNI.SKEL_RIGHT_FOOT);
  drawJoint(userId, SimpleOpenNI.SKEL_RIGHT_HAND);
  drawJoint(userId, SimpleOpenNI.SKEL_LEFT_HAND);
}
// draws a joint of the skeleton (for a user and a joint id)
void drawJoint(int userId, int jointID) {
  PVector joint = new PVector();
  // get the joint position
  float confidence = kinect.getJointPositionSkeleton(userId,
jointID, joint);
  // if there isn't much confidence, do nothing...
  if(confidence < 0.5){
    return;
  }
  PVector convertedJoint = new PVector();
  // convert to projected coordinates
  kinect.convertRealWorldToProjective(joint, convertedJoint);
  //draw the joint
  ellipse(convertedJoint.x, convertedJoint.y, 5, 5);
  //write the text for this joint
  writeTextForJoint(jointID, convertedJoint.x, convertedJoint.y);
}
// writes the name of a joint provided by the kinect
void writeTextForJoint(int jointID, float x, float y){
  if (jointID == SimpleOpenNI.SKEL_HEAD)
    text("HEAD", x, y);
  else
  if (jointID == SimpleOpenNI.SKEL_NECK)
    text("NECK", x, y);
  else
  if (jointID == SimpleOpenNI.SKEL_LEFT_SHOULDER)
    text("LEFT_SHOULDER", x, y);
  else
```

```
  if (jointID == SimpleOpenNI.SKEL_RIGHT_SHOULDER)
    text("RIGHT_SHOULDER", x, y);
  else
  if (jointID == SimpleOpenNI.SKEL_LEFT_ELBOW)
    text("LEFT_ELBOW", x, y);
  else
  if (jointID == SimpleOpenNI.SKEL_RIGHT_ELBOW)
    text("RIGHT_ELBOW", x, y);
  else
  if (jointID == SimpleOpenNI.SKEL_TORSO)
    text("TORSO", x, y);
  else
  if (jointID == SimpleOpenNI.SKEL_LEFT_HIP)
    text("LEFT_HIP", x, y);
  else
  if (jointID == SimpleOpenNI.SKEL_RIGHT_HIP)
    text("RIGHT_HIP", x, y);
  else
  if (jointID == SimpleOpenNI.SKEL_LEFT_KNEE)
    text("LEFT_KNEE", x, y);
  else
  if (jointID == SimpleOpenNI.SKEL_RIGHT_KNEE)
    text("RIGHT_KNEE", x, y);
  else
  if (jointID == SimpleOpenNI.SKEL_LEFT_FOOT)
    text("LEFT_FOOT", x, y);
  else
  if (jointID == SimpleOpenNI.SKEL_RIGHT_FOOT)
    text("RIGHT_FOOT", x, y);
  else
  if (jointID == SimpleOpenNI.SKEL_LEFT_HAND)
    text("LEFT_HAND", x, y);
  else
  if (jointID == SimpleOpenNI.SKEL_RIGHT_HAND)
    text("RIGHT_HAND", x, y);
}
//
--------------------------------------------------------------
// SimpleOpenNI callbacks
void onNewUser(SimpleOpenNI curkinect, int userId)
{
  println("onNewUser - userId: " + userId + ", start tracking
skeleton");
  curkinect.startTrackingSkeleton(userId);
}
void onLostUser(SimpleOpenNI curkinect, int userId)
{
```

```
  println("onLostUser - userId: " + userId);
}
void onVisibleUser(SimpleOpenNI curkinect, int userId)
{
  println("onVisibleUser - userId: " + userId);
}
```

Μολονότι ο κώδικας είναι μακρύς, αρκετά από τα βασικά του στοιχεία τα έχουμε δει από το προηγούμενο παράδειγμα. Δηλαδή, και εδώ χρειάζεται να ενεργοποιήσουμε την ανίχνευση χρηστών, να ζωγραφίσουμε σιλουέτες, να διαβάσουμε τους χρήστες που βλέπει το kinect, να υλοποιήσουμε τις αυτόματες μεθόδους (user tracking callback methods). Επίσης, αρκετά από τα νέα στοιχεία του προγράμματος μας επαναλαμβάνονται. Άρα οι νέες έννοιες δεν είναι τόσο πολλές!

Σε σύγκριση με τα προηγούμενα, το νέο στοιχείο είναι ότι κάθε έναν χρήστη ζωγραφίσουμε το σκελετό του. Το πρόγραμμα μας ζωγραφίζει 3 ειδών στοιχεία: **αρθρώσεις** του σώματος (joints, σύμφωνα με την ανατομία του kinect), **μέρη** του σώματος (limps, γραμμές που συνδέουν αρθρώσεις) και **ονομασίες** (αρθρώσεων). Τα παραπάνω επιτυγχάνονται με τις μεθόδους `drawSkeleton, drawJoint, writeTextForJoint.`

1. `drawSkeleton`. Η μέθοδος έχει δύο μέρη.
   a. Αρχικά χρησιμοποιεί την μέθοδο `drawLimb` η οποία μας παρέχεται από τη βιβλιοθήκη simpleOpenNI για να ζωγραφίσουμε μια σύνδεση μεταξύ δύο αρθρώσεων.
   b. Στη συνέχεια ζωγραφίζει την άρθρωση με τη μέθοδο `drawJoint` η οποία είναι μια μέθοδος που έχουμε γράψει στην ίδια λογική του προηγούμενου παραδείγματος (ζωγραφική του σημείου στο δεξί χέρι του χρήστη).
      i. Η μέθοδος `writeTextForJoint` καλείται εντός της `drawJoint` για να τυπώσει το κείμενο που αναφέρεται στην άρθρωση.

Τις παραπάνω μεθόδους `drawSkeleton, drawJoint, writeTextForJoint,` είναι πιθανό να τις χρησιμοποιήσετε για λόγους αποσφαλμάτωσης (debugging) κάποιας πιο εξελιγμένης εφαρμογής. Π.χ. αν αναπτύσετε εφαρμογή στην οποία ο χρήστης χρησιμοποιεί κάποιες πόζες (postures) για να αλληλεπιδράσει (π.χ. μια εφαρμογή γυμναστικής), τότε μπορείτε να χρησιμοποιήσετε αυτές τις μεθόδους, για να δείτε αν το πρόγραμμα σας εργάζεται σωστά για κάθε πόζα.

Υπολογισμός του κέντρου της μάζας (centre of mass) των χρηστών της σκηνής

Συχνά χρειάζεται να εκτιμήσουμε αν είναι χρήστες στη σκηνή. Δεν είναι απαραίτητο να χρησιμοποιήσουμε τα δεδομένα σκελετού γι αυτό το σκοπό, εξάλλου ξέρουμε ήδη ότι το kinect μπορεί να κάνει εκτίμηση των χρηστών χωρίς να έχει δεδομένα σκελετού.

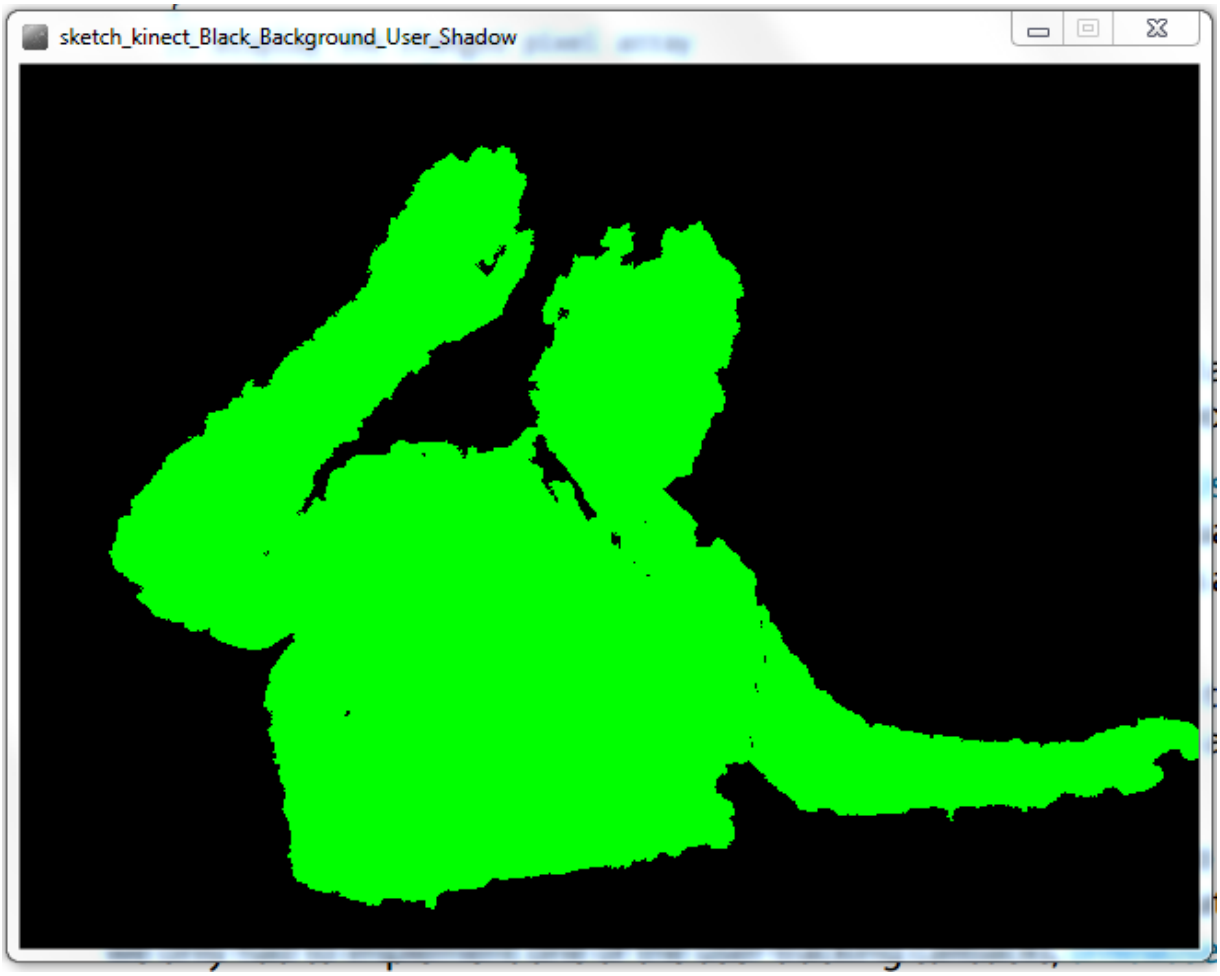
Η simpleOpenNI μας παρέχει τη μέθοδο CoM (Centre of Mass) η οποία μας επιστέφει το κέντρο του σώματος του χρήστη. Ο κώδικας του προγράμματος μας είναι ο εξής::

```
import SimpleOpenNI.*;
SimpleOpenNI  kinect;
void setup() {
  size(640,480);
  kinect = new SimpleOpenNI(this);
  // enable depthMap generation
  kinect.enableDepth();
  // enable user tracking
  kinect.enableUser();
}
void draw() {
  kinect.update();
  image(kinect.depthImage(), 0, 0);
  int[] userList = kinect.getUsers();

  for(int i=0;i<userList.length;i++) {
    PVector position = new PVector();
    kinect.getCoM(userList[i], position); //for this user, get the
position of the CoM
    kinect.convertRealWorldToProjective(position, position);
    fill(255, 0, 0);
    textSize(40);
    text(userList[i], position.x, position.y);
  }
}

// -----------------------------------------------------------------
// SimpleOpenNI callbacks
void onNewUser(SimpleOpenNI curkinect, int userId) {
  println("onNewUser - userId: " + userId + ", start tracking
skeleton");
  curkinect.startTrackingSkeleton(userId);
}
void onLostUser(SimpleOpenNI curkinect, int userId) {
  println("onLostUser - userId: " + userId);
}
void onVisibleUser(SimpleOpenNI curkinect, int userId) {
  println("onVisibleUser - userId: " + userId);
}
```

Το αποτέλεσμα θα είναι κάτι τέτοιο.

Διαχωρισμός υποβάθρου από το χρήστη, με χρήση της κάμερας βάθους

Στα παραπάνω παραδείγματα χρησιμοποιήσαμε τα δεδομένα του σκελετού. Όμως, μπορούμε να χρησιμοποιήσουμε το kinect για να αναγνωρίζουμε ανθρώπινες σιλουέτες χωρίς να χρησιμοποιούμε αυτά τα δεδομένα. Μια ενδιαφέρουσα εφαρμογή της αναγνώρισης χρηστών στη σκηνή από το kinect είναι όταν θέλουμε να διαχωρίσουμε τους χρήστες από το υπόβαθρο.



Για να καταλάβουμε το παράδειγμα θα πρέπει να έχουμε υπόψη μας τα εξής:

1. Το kinect μπορεί να ανιχνεύσει τη παρουσία χρηστών στη σκηνή (χωρίς απαραίτητα να έχει δεδομένα του σκελετού τους, αρχικά δεν τα έχει). Όσο το kinect ανιχνεύει έναν τουλάχιστον χρήστη, η βιβλιοθήκη simpleOpenNI μπορεί να μας τους επιστρέφει στο πρόγραμμα μας μέσα από τη μέθοδο `int[] userMap();` Το **userMap είναι ένας πίνακας ακεραίων μήκους 640x480 που κάθε στοιχείο (αναπαριστά pixel) έχει τιμή 1 αν ανήκει στη σιλουέτα του χρήστη και 0 αν όχι**.
    a. Σημείωση: απαραίτητη προϋπόθεση για να ανιχνευτούν χρήστες στη σκηνή είναι η προηγούμενη ενεργοποίηση της κάμερας βάθους.
2. Η Processing μας παρέχει τη μέθοδο **`loadPixels();` η οποία φορτώνει όλα τα Pixels της εικόνας που βλέπει ο χρήστης στον πίνακα `int[] pixels;`** Αυτή η δυνατότητα είναι πολύ χρήσιμη για να τα επεξεργαστούμε, και να τα αλλάξουμε δυναμικά! Μετά την οποιαδήποτε αλλαγή κάνουμε, για να ανανεώσουμε την εικόνα πρέπει να καλέσουμε την **`updatePixels();`**
3. Επομένως, για να πετύχουμε το αποτέλεσμα που φαίνεται στην παραπάνω εικόνα χρειάζεται:

a. να θέσουμε το υπόβαθρο μαύρο, και
b. να φτιάξουμε μια επανάληψη for, στην οποία θα βάφουμε πράσινο κάθε pixel που είναι διάφορο του 0 στο userMap.

Ο παρακάτω κώδικας λύνει αυτό το πρόβλημα.

```
import SimpleOpenNI.*;
SimpleOpenNI kinect;

int[] userMap; //an array of pixels of the kinect image that show
the user

void setup() {
  size(640, 480);
  kinect = new SimpleOpenNI(this);
  // enable the depth camera
  kinect.enableDepth();
  // enable user tracking
  kinect.enableUser();
}

void draw() {
  // display a black background
  background(0);
  kinect.update();
  // if there is a user in the scene ...
  if (kinect.getNumberOfUsers() >0){
    /* Prepare the output image pixels array, for further
processing.
    This array (int[] pixels) contains all pixels that correspond
to the
    output image of this sketch (to be displayed on the screen).
    Currently, it consists black pixels! */
    loadPixels();
    /* Prepare the user pixels. In userMap[] all pixels that
correspond to the
    user are located. */
    userMap = kinect.userMap();
    //For all pixels that correspond to the user(s) image of the
scene
    for (int j =0; j < userMap.length; j++) {
      // if the pixel is part of the user
      if (userMap[j] != 0) {
        // set the sketch pixel (pixels[j]) to color green (to
create a live user shadow)
        pixels[j] = color(0, 255, 0);
      }
```

```
    }
    updatePixels(); //Updates the display window with the data in
the pixels[] array
  }
}

//
-----------------------------------------------------------------
// SimpleOpenNI callbacks
void onNewUser(SimpleOpenNI curkinect, int userId)
{
  println("onNewUser - userId: " + userId + ", start tracking
skeleton");
  curkinect.startTrackingSkeleton(userId);
}
void onLostUser(SimpleOpenNI curkinect, int userId)
{
  println("onLostUser - userId: " + userId);
}
void onVisibleUser(SimpleOpenNI curkinect, int userId)
{
  println("onVisibleUser - userId: " + userId);
}
```

Αλλαγή υποβάθρου με χρήση των 2 καμερών

Το να διαχωρίσουμε απλά το χρήστη από το υπόβαθρο είναι χρήσιμο, αλλά γιατί να μην δείχνουμε ένα πιο ενδιαφέρον υπόβαθρο; Και γιατί να μην δείξουμε και το χρήστη κανονικά με χρώμα, και όχι απλά τη “σκιά” του; Για να λύσουμε το παραπάνω πρόβλημα, θα χρειαστεί να κάνουμε δύο επιπλέον πράγματα: (α) να βάλουμε μια εικόνα υποβάθρου (εύκολο), και (β) να χρησιμοποιήσουμε τη κάμερα χρώματος ανανεώνοντας σωστά την εικόνα που δείχνουμε στο χρήστη (το νέο στοιχείο της άσκησης, σε σχέση με τη προηγούμενη).

Πιο συγκεκριμένα, τα βήματα που πρέπει να ακολουθήσουμε είναι τα εξής:
Στη setup():

1. **Ενεργοποιούμε** και τις δύο κάμερες
2. **Ευθυγραμμίζουμε την εικόνα από τις δύο κάμερες**. Αυτό το κάνουμε με την εντολή `kinect.alternativeViewPointDepthToImage();` Πλέον οι εικόνες από τις 2 κάμερες θα είναι ευθυγραμμισμένες για την εφαρμογή μας.
3. **Θέτουμε στο υπόβαθρο** την εικόνα της επιλογής μας . Εδώ, είναι απαραίτητο η εικόνα του υποβάθρου να έχει ανάλυση 640x480. Αν η εικόνα που έχουμε δεν έχει αυτήν την ανάλυση μπορούμε να καλέσουμε την εντολή `backgroundImage.resize(640, 480);` (πιθανώς όμως να μας “τεντώσει” την εικόνα αν οι αναλογίες δεν είναι σωστές)

Στη draw(), αρχικά θα κάνουμε ό,τι και προηγουμένως, δηλαδή θα φορτώσουμε τα pixels της εικόνας (που περιέχουν το υπόβαθρο) και το userMap. Επιπλέον όμως, θα χρειαστεί να φορτώσουμε και τα pixels της rgb κάμερας. Και εντός της επανάληψης for, να αντικαταστήσουμε το κάθε user pixel της εικόνας που βλέπει ο χρήστης (όχι με πράσινο χρώμα όπως πριν αλλά), με τα pixels της rgb κάμερας! Πιο συγκεκριμένα:

1. Για να **φορτώσουμε τα pixels της κάμερας χρώματος**, χρησιμοποιούμε την εντολή

`rgbImage.loadPixels();` η οποία φορτώνει τα pixels στον πίνακα `int[] rgbImage.pixels` (ο οποίος περιέχει για κάθε θέση του ένα χρώμα).

2. **Εντός της επανάληψης for, ελέγχουμε για κάθε pixel της εικόνας που βλέπει ο χρήστης αν είναι user pixel**. Αν ναι, τότε αντικαθιστούμε το pixel με το χρώμα του από τον πίνακα `rgbImage.pixels`.

```
import SimpleOpenNI.*;
SimpleOpenNI kinect;

int[] userMap; //the pixels of the image that show the user
PImage backgroundImage;

void setup() {
  size(640, 480);
  kinect = new SimpleOpenNI(this);
  // enable both depth and RBG cameras
  kinect.enableDepth();
  kinect.enableRGB();
  // enable user tracking
  kinect.enableUser();
  // turn on depth-color image alignment
  kinect.alternativeViewPointDepthToImage();
  // load the background image
  backgroundImage = loadImage("earth.jpg");
  /* In case you have an image that is not 640x480,
  you can resize it inside Processing with:
  backgroundImage.resize(640, 480);*/
}

void draw() {
  // display the background image
  image(backgroundImage, 0, 0);
  kinect.update();
  // if there is a user in the scene ...
  if (kinect.getNumberOfUsers() >0){
    // get the Kinect (live) color image
    PImage rgbImage = kinect.rgbImage();
    /* Get the rgb image into a pixel array in memory, for further
processing… This array (int[] rgbImage.pixels) contains all pixels
that correspond to the rgbImage (i.e. the 'live' kinect rgb image).
*/
    rgbImage.loadPixels();

    /* Prepare the output image pixels array, for further
processing. This array (int[] pixels) contains all pixels that
correspond to the output image of this sketch (to be displayed on
```

```
the screen). Currently, it consists of the background image */
    loadPixels();

    /* Prepare the user pixels. In userMap[] all pixels that
correspond to the user are located. */
    userMap = kinect.userMap();

    //For all pixels that correspond to the user(s) image of the
scene
    for (int j =0; j < userMap.length; j++) {
      // if the pixel is part of the user
      if (userMap[j] != 0) {
        // set the sketch pixel (pixels[j]) to the user pixel
        //(taken from the rgb image: rgbImage.pixels[j])
        pixels[j] = rgbImage.pixels[j];
      }
    }
    updatePixels(); //Updates the display window with the data in
the pixels[] array
  }
}

//
-----------------------------------------------------------------
// SimpleOpenNI callbacks
void onNewUser(SimpleOpenNI curkinect, int userId)
{
  println("onNewUser - userId: " + userId + ", start tracking
skeleton");
  curkinect.startTrackingSkeleton(userId);
}
void onLostUser(SimpleOpenNI curkinect, int userId)
{
  println("onLostUser - userId: " + userId);
}
void onVisibleUser(SimpleOpenNI curkinect, int userId)
{
  println("onVisibleUser - userId: " + userId);
}
```

Χρήση του kinect + simpleOpenNI για ανίχνευση των χειρονομιών που υποστηρίζονται (WAVE, CLICK, HAND_RAISE)

Τι είναι η χειρονομία;

Οι χειρονομίες για την κιναισθητική αλληλεπίδραση είναι κατ' αντιστοιχία ό,τι μας προσφέρει το ποντίκι στη περίπτωση της αλληλεπίδρασης με διεπαφές γραφικών (WIMP, Hypertext, multimedia) στο προσωπικό υπολογιστή.

Μια πρώτη παρατήρηση είναι ασφαλώς ότι οι χειρονομίες είναι **έμφυτο στοιχείο της ανθρώπινης επικοινωνίας**, σε αντίθεση με το ποντίκι το οποίο αποτελεί εφεύρεση.

Επιπρόσθετα, πολλές χειρονομίες είναι συνδεδεμένες με το **πολιτισμικό υπόβαθρο και τις συνήθειες μας** και κατά συνέπεια κάποιες χειρονομίες να μας είναι 'γνωστές' ή 'άγνωστες', άλλες κοινωνικά 'κομψές/σεμνές' ή 'άκομψες/άσεμνες'.

Επίσης, **περισσότερες από μια χειρονομίες είναι δυνατό να εκφράζουν το ίδιο μήνυμα και το αντίστροφο (πολλαπλή σημασία / αμφισημία)**. Άρα, ο αριθμός των πιθανών χειρονομιών προς υιοθέτηση/χρήση σε μια κιναισθητική διεπαφή μπορεί να είναι πάρα πολύ μεγάλος (σίγουρα πολλές 100δες), σε αντίθεση με το πεπερασμένο σύνολο χειρισμών που μπορεί να κάνει κάποιος με τη χρήση του ποντικιού (λίγες 10άδες). Βεβαίως στα πλαίσια κάποιας συγκεκριμένης εφαρμογής το σύνολο των χειρονομιών ορίζεται από την ομάδα σχεδίασης/ανάπτυξης. Επομένως χρειάζεται **διερεύνηση και έλεγχος** προς την επιλογή και ακριβή προσδιορισμό των χειρονομιών σε μια εφαρμογή κιναισθητικού ελέγχου.

Μια επιπλέον παρατήρηση είναι η εξής: με δεδομένο ότι είμαστε στην αρχή της ανάδυσης κιναισθητικών διεπαφών, **δεν έχουν αναπτυχθεί ακόμα επαρκώς** συμβάσεις (conventions), σχεδιαστικά υποδείγματα (design patterns) και ευρέως αποδεκτές τεχνικές αλληλεπίδρασης (interaction techniques) για κιναισθητικό έλεγχο.

Το πεδίο είναι απολύτως ανοικτό σε νέες ιδέες και εφαρμογές. Οι σχεδιαστές εφαρμογών κιναισθητικού ελέγχου καλούνται να εντοπίσουν κατάλληλες χειρονομίες και να τις αντιστοιχίσουν με ενέργειες του συστήματος, να προσδιορίσουν τις χειρονομίες και τις αποκρίσεις επακριβώς, λαμβάνοντας υπόψη τις ικανότητες και υπόβαθρο των χρηστών καθώς και το πλαίσιο χρήσης.

Σε αυτή τη διαδικασία, αν και δεν είναι πάντοτε εφικτό, οι σχεδιαστές θα πρέπει να αποβάλλουν τη σκέψη της μεταφοράς αλληλεπιδράσεων από το περιβάλλον του προσωπικού υπολογιστή. Είναι προτιμότερο κάποιος να κάνει κάποια αναδρομή σε παλαιότερες προσπάθειες κιναισθητικού ελέγχου αναζητώντας ενδιαφέροντα παραδείγματα και τεχνικές αλληλεπίδρασης (σε πειραματικό - ερευνητικό επίπεδο κατά κανόνα, με λιγοστές εξαιρέσεις εμπορικών προϊόντων όπως το Wii).

Είδη χειρονομιών: "Καθαρές χειρονομίες" (pure gestures), χειρισμοί (manipulations) και πόζες (poses)

Η έννοια της χειρονομίας έχει οριστεί με διαφορετικούς τρόπους σε διάφορα πεδία, όπως η γλωσσολογία (linguistics), σημειολογία (semiotics) και τις τέχνες… στην Αλληλεπίδραση Ανθρώπου-Η/Υ ένας σημαντικός ορισμός είναι ο εξής:

> *"A gesture is a motion of the body that contains information. Waving goodbye is a gesture. Pressing a key on a keyboard is not a gesture because the motion of a finger on its way to hitting a key is neither observed nor significant. All that matters is which key was pressed." (Kurtenbach, G., & Hulteen, E. A. (1990). Gestures in human-computer communication. The art of human-computer interface design, 309-317.)*

Ο παραπάνω ορισμός μας εξηγεί τι θεωρείται χειρονομία και τι όχι, στη βάση της διάκρισης του εάν η κίνηση του σώματος υπονοεί κάποια πληροφορία ή μήνυμα. Επίσης, προσδιορίζεται ότι εάν μια κίνηση του σώματος γίνεται αποκλειστικά για χειρισμό του Η/Υ τότε αυτό δεν θεωρείται χειρονομία!

Η χρησιμότητα του παραπάνω ορισμού έγκειται στο ότι διακρίνει τις **χειρονομίες (gestures)** από τους **χειρισμούς (manipulations)**, κυρίως επειδή υποννοείται ότι οι χειρονομίες προϋπάρχουν στην ανθρώπινη επικοινωνία, ενώ οι χειρισμοί είναι εφευρέσεις-ανακαλύψεις με σκοπό τη χρήση του Η/Υ. Σημειώστε όμως ότι όταν διατυπώθηκε δεν είχαμε τις τεχνολογικές δυνατότητες που έχουμε σήμερα να σχεδιάσουμε κιναισθητικές διεπαφές - σήμερα έχουμε τη δυνατότητα να μην έχουμε διεπαφές υλικού όπως το ποντίκι/πληκτρολόγιο όταν μιλάμε για κιναισθητικό έλεγχο, επομένως οι χειρισμοί για κιναισθητικό έλεγχο απαιτούν τελικά κάποιας μορφής κίνηση του σώματος των χρηστών. Κατά μια ευρεία έννοια λοιπόν και οι χειρισμοί υπονοούν πληροφορία που αφορά συγκεκριμένες εντολές προς τον Η/Υ, και γι αυτό θα τις θεωρήσουμε ως μια κατηγορία χειρονομιών. Τόσο οι χειρονομίες όσο και οι χειρισμοί **εκτυλίσσονται στο χρόνο**.

Μια επιπλέον έννοια που δεν υποννοείται από τα παραπάνω αλλά είναι απαραίτητη για εφαρμογές κιναισθητικού ελέγχου είναι η **πόζα (posture)**, δηλαδή η στάση του ανθρωπίνου σώματος μέσω της οποίας επίσης εκφράζεται κάποια πληροφορία ή μήνυμα. Η διαφορά της πόζας από τις χειρονομίες και χειρισμούς είναι ότι είναι στατική: **ο χρήστης απαιτείται να ακινητοποιηθεί** ενώ ο Η/Υ μπορεί να αναγνωρίσει τη πόζα **σε μια στιγμή στο χρόνο**. Μια πόζα μπορεί να επικοινωνεί κάποιο ευρέως αναγνωρίσιμο μήνυμα (π.χ. "βαριέμαι", "κάνε ησυχία", κ.α.) κατ' αντιστοιχία με μια χειρονομία. Μπορεί όμως και να είναι μια επινόηση του σχεδιαστή για να γίνει κάποιος χειρισμός! Μια πόζα περιγράφεται από κάποια σχέση συγκεκριμένων μερών του ανθρωπίνου σώματος, π.χ. η πόζα "ψηλά τα χέρια" απαιτεί από το χρήστη να έχει τα μπράτσα του σε οριζόντια θέση (90 μοίρες από το κορμό) και τα χέρια του κατακόρυφα (90 μοιρές από τους αγκώνες). Η πόζα universal pause του klinect απαιτεί από το χρήστη να είναι σε όρθια θέση με το δεξί χέρι κολλημένο στο σώμα και το αριστερό χέρι

τεντωμένο σε γωνία 45 μοιρών από το σώμα, και όταν ασκείται γίνεται αμέσως παύση του (οποιουδήποτε) παιχνιδιού.

Η παραπάνω διάκριση είναι πολύ χρήσιμη εννοιολογικά, επειδή όπως θα δούμε στη συνέχεια μας βοηθάει να αντιληφθούμε και τι χρειάζεται για να υλοποιήσουμε για κάθε έναν από τους παραπάνω τύπους χειρονομιών. Όμως για διευκόλυνση μας, στη συνέχεια θα αναφερόμαστε στις χειρονομίες όπως τις διακρίναμε παραπάνω με τον όρο ‘καθαρές χειρονομίες (pure gestures) και εν γένει στους 3 παραπάνω τύπους χειρονομιών απλά ως χειρονομίες. Άρα:

- Κάθε κίνηση του χρήστη που δίνει κάποια πληροφορία ή εντολή στο σύστημα θεωρείται ως **χειρονομία**.
- Οι χειρονομίες διακρίνονται σε:
    - **Καθαρές χειρονομίες (pure gestures)**, όταν παρέχουν στο σύστημα κάποια πληροφορία ή εντολή - χωρίς όμως να κάνουν χειρισμό κάποιου αντικειμένου. Η πιο συνιθισμένη χειρονομία είναι ο χαιρετισμός (wave) για να ξεκινήσει το σύστημα.
    - **Χειρισμοί (manipulations)**, όταν επηρεάζουν τη συμπεριφορά κάποιου αντικειμένου της διεπαφής. Συνηθισμένοι χειρισμοί είναι η επιλογή, η μεγέθυνση, η κύλιση, κ.α.
    - **Πόζες (pozes)**, όταν ο χρήστης παίρνει μια προκαθορισμένη στατική θέση για να επικοινωνήσει με το σύστημα. Η πιο συνιθισμένη πόζα είναι η universal pause.

Μερικές συχνές χειρονομίες για το kinect XBox και άλλα εργαλεία κιναισθητικού ελέγχου …

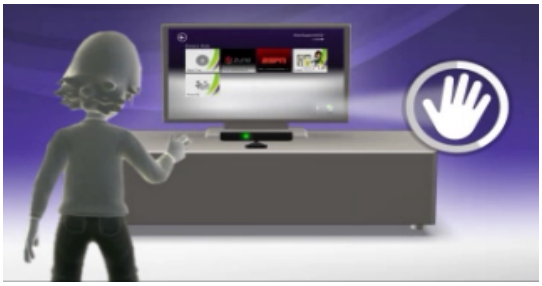
Δεν υπάρχουν ευρέως καθιερωμένες χειρονομίες, σχεδιαστικά υποδείγματα ή τεχνικές αλληλεπίδρασης για εφαρμογές κιναισθητικού ελέγχου. Αυτό είναι θετικό από την άποψη ότι το πεδίο είναι ανοικτό για πειραματισμό και καινοτομία, η οποία όμως θα πρέπει να έρθει μέσα από τη στενή επαφή με τη τεχνολογία μιας και δεν υπάρχουν (ακόμα;) σχεδιαστικά περιβάλλοντα για κιναισθητικό έλεγχο.

Έχουν δοκιμαστεί 10δες χειρονομίες διαφορετικές ανά παιχνίδι ή εφαρμογή. Πολλές από αυτές είναι κατάλληλες για τη συγκεκριμένη εφαρμογή αλλά δεν μπορούν εύκολα να γενικευτούν. Στα επόμενα θα δούμε μια συλλογή αυτών που είναι κάπως γενικεύσιμες. Σε καμία περίπτωση όμως δεν είναι καθιερωμένες.

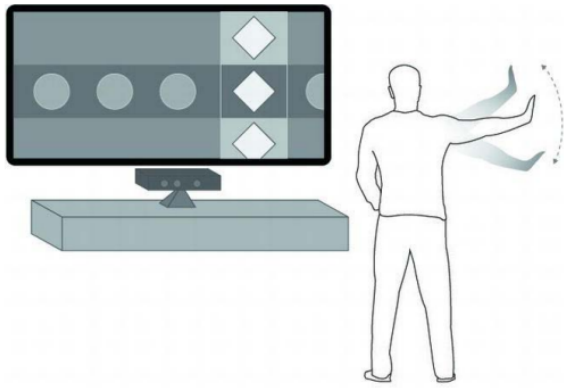
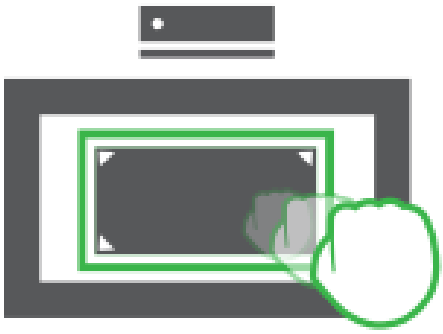
> *Σημείωση: το τι είναι καθιερωμένο ή όχι από τα παρακάτω αλλάζει πολύ γρήγορα. Το πιθανότερο είναι ότι αν διαβάσετε αυτές τις σημειώσεις σε 2 χρόνια από τώρα να έχει αλλάξει δραματικά το τοπίο! Το ίδιο ισχύει και για νέα εργαλεία/εφαρμογές πέρα αυτών που καταγράφονται εδώ.*

Το kinect μέχρι και την έκδοση 1.8 υποστηρίζει ευρέως (για τα περισσότερα παιχνίδια) τις εξής χειρονομίες:

- **wave hand (arm)** για έναρξη (από παύση),
- **hover hand (button)** για επιλογή,
- **press (push) hand** για επιλογή - πλέον προτιμάται του hover button, από την έκδοση 1.7 και μετά.
- **universal pause (poze)** για παύση,
- **horizontal right arm up/down** για κάθετη κύλιση (vertical scrolling) - πλέον εγκαταλείπεται από την έκδοση 1.7 η κάθετη κύλιση χάρην της οριζόντιας.
- **hand grip and horizontal hand move** για οριζόντια κύλιση
- **2-hand grip and move hands to-clap** για επιστροφή στη κεντρική οθόνη
- **grip forward/backward** για μεγέθυνση
- **grip/move hand/release** για πιάσιμο/μετακίνηση/άφεση αντικειμένου

<table>
<tr>
<td> 
Wave (pure gesture)</td>
<td> 
Hover Button (manipulation)</td>
</tr>
<tr>
<td> 
Universal Pause (poze)</td>
<td> 

Hand press - ο χρήστης τεντώνει το χέρι του μόλις είναι επί του στόχου και αυτός επιλέγεται…</td>
</tr>
</table>

 Horizontal right arm up/down for vertical scrolling.	 Hand grip and horizontal hand move for horizontal scrolling
 2-hand grip and move hands to-clap για επιστροφή στη κεντρική οθόνη	 grip forward/backward για μεγέθυνση

Περισσότερα:

- http://hwcdn.libsyn.com/p/4/4/c/44c89c7f273167b4/Xbox_One_Kinect_Voice_Gesture.pdf?c_id=6458139&expiration=1397651666&hwt=3de4fbfc3a0f1ca2ce039da296354007
- http://support.xbox.com/en-US/xbox-one/kinect/common-gestures
- http://support.xbox.com/el-GR/xbox-360/kinect/body-controller

Σημειωτέον ότι άλλες τεχνολογίες ανίχνευσης όπως π.χ. leap motion (https://www.leapmotion.com), Myo (https://www.thalmic.com/en/myo), δίνουν έμφαση στην ανίχνευση δακτύλων και παλάμης και χρησιμοποιούν άλλα ρεπερτόρια χειρονομιών.

Χειρονομίες που υποστηρίζονται από τη βιβλιοθήκη simpleOpenNI για την Processing (WAVE, CLICK, HAND_RAISE)

Η βιβλιοθήκη simpleIOpenNI for Processing υλοποιεί (μόνο) 3 χειρονομίες: WAVE, CLICK, HAND_RAISE. Πιο συγκεκριμένα:

- Η χειρονομία WAVE είναι ο χαιρετισμός με το ένα μας χέρι. για την ασκήσουμε σηκώνουμε το μπράτσο μας σε σχεδόν παράλληλο προσανατολισμό με το έδαφος και κρατάμε το βραχίονα σε γωνία περίπου 90% με το μπράτσο μας. Στη συνέχεια κουνάμε το βραχίονα δεξιά-αριστερά. Έπειτα από 2 επαναλήψεις, η simpleOpenNI θεωρεί ότι ασκούμε τη χειρονομία WAVE.
- Η χειρονομία CLICK ασκείται όταν τεντώνουμε το χέρι μας μπροστά από το σώμα μας σε σχεδόν παράλληλη θέση με το έδαφος. Είτε δείχνουμε (με το δείκτη του χεριού μας) ή όχι, η simpleOpenNI θεωρεί ότι ασκούμε τη χειρονομία CLICK (εξάλλου, θυμηθείτε ότι τα δεδομένα σκελετού δεν μας δίνουν άρθρωσεις (joints) στα δάκτυλα του χρήστη, άρα δεν μπορούμε να ξέρουμε αν ο χρήστης δείχνει ή όχι!).
- Η χειρονομία HAND_RAISE ασκείται απλά όταν σηκώσουμε και το χέρι μας και το κρατήσουμε σχεδόν τεντωμένο σε περίπου κάθετη θέση από το έδαφος.

Γιατί η simpleOpenNI δεν υλοποιεί περισσότερες χειρονομίες, ώστε να μας παρέχει ένα ευρύ ρεπερτόριο χειρονομιών;

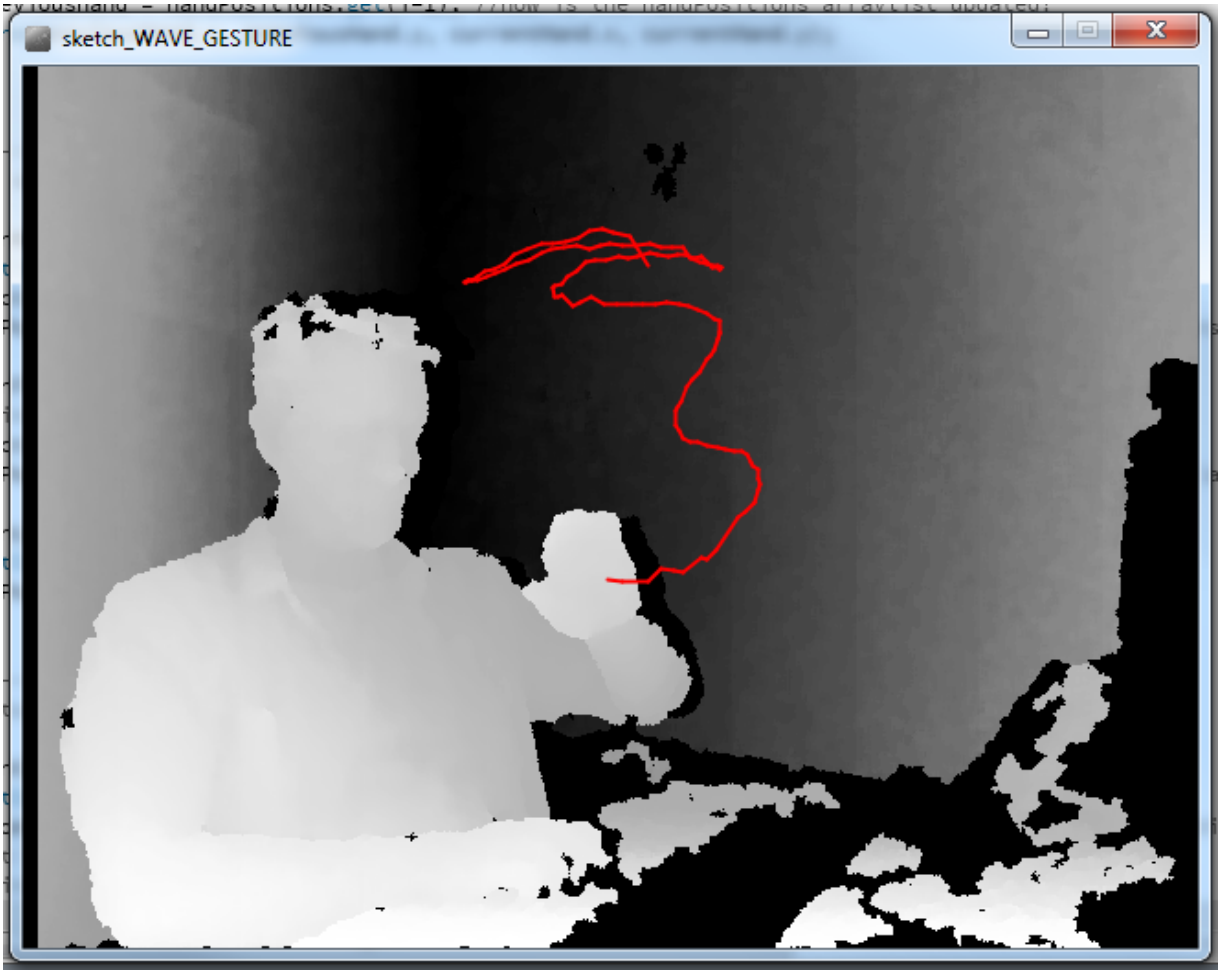
Η απάντηση είναι σύνθετη, οι βασικοί λόγοι είναι επιχειρηματικοί, βιωσιμότητας της δουλειάς, τεχνολογικοί, χρονικοί και σχεδιαστικοί:

1. Το kinect είναι τεχνολογία της Microsoft. Η κοινότητα προγραμματιστών ανοικτού κώδικα της openNI δεν μπορεί να γνωρίζει πάντοτε τα σχέδια της Microsoft, παρά μόνο ακολουθεί την εξέλιξη του kinect και προσαρμόζεται, στο βαθμό που προλαβαίνει να το κάνει.
2. Η εταιρία που υποστήριζε έμπρακτα την κοινότητα openNI (διαθέτοντας χρόνο στους προγραμματιστές της να εργάζονται για ανοικτό κώδικα, φιλοξενούσε το site, κ.α.) ήταν η Prime Sense - η οποία είχε συνεργαστεί στενά με τη Microsoft για να κατασκευαστεί το kinect. Η εταιρία εξαγοράστηκε από την Apple (βλ. http://www.forbes.com/sites/shelisrael/2013/11/25/why-would-apple-buy-primesense) η οποία δεν υποστηρίζει, ως γνωστόν, ούτε κατ' ελάχιστον τη φιλοσοφία του ανοικτού λογισμικού, γι αυτό και "κατέβασε" το δικτυακό τόπο της openNI (πλέον δεν λειτουργεί το http://openni.org/ - αντ' αυτού το λογισμικό φιλοξενείται μέχρι στιγμής εδώ: https://github.com/OpenNI/OpenNI).
3. Πολύ γρήγορη τεχνολογική εξέλιξη του αισθητήρα kinect με προσθήκη νέων δυνατοτήτων. Π.χ. βλ. http://www.microsoft.com/en-us/kinectforwindows/meetkinect/faq.aspx για σύγκριση των Kinect SDK 1.8 (Σεπτέμβριος 2013) και 2.0 (Ιούλιος 2014). - Μέχρι να γραφτεί η νέα έκδοση της βιβλιοθήκης, έχει βγει καινούρια έκδοση του αισθητήρα.
4. Ποιες χειρονομίες; - Δεν υπάρχει κάποιο καθιερωμένο σύνολο χειρονομιών, εξαρτώνται πολύ από τις εφαρμογές.

Πρόγραμμα σε kinect + simpleOpenNI ανίχνευσης των χειρονομιών WAVE, HAND_RAISE, CLICK

Ας επιστρέψουμε όμως στο kinect + processing + simpleOpenNI και ας δούμε ένα παράδειγμα ανίχνευσης και απλής χρήσης των χειρονομιών WAVE, HAND_RAISE, CLICK. Αυτά που θέλουμε να κάνει το πρόγραμμα μας είναι:

1. Να παρακολουθεί τη σκηνή μέχρις ότου αναγνωρίσει κάποια χειρονομία (μία κάθε φορά, έστω για αρχή την) WAVE
2. Μόλις ασκηθεί (ολοκληρωθεί) η χειρονομία, να αρχίσει να ζωγραφίζει την πορεία του χεριού του χρήστη στην οθόνη.



Τα σημαντικά σχετικά θέματα που χρειάζεται να γνωρίζουμε είναι τα εξής:

1. **Ανίχνευση του χεριού του χρήστη** - Προκειμένου να ασκηθούν χειρονομίες είναι κρίσιμο να μπορούμε εύκολα να εντοπίσουμε το χέρι του χρήστη. Είδαμε σε προηγούμενο παράδειγμα πως μπορούμε να φτιάξουμε κώδικα που να εντοπίζει το χέρ του χρήστη ενεργοποιώντας τα δεδομένα σκελετού. Επειδή ο εντοπισμός του

χεριού είναι πολύ συνηθισμένη απαίτηση, η simpleOpenNI μας παρέχει ένα σύνολο μεθόδων που μας βοηθάει να ανιχνεύουμε το χέρι του χρήστη. Αυτές είναι:

a. Η μέθοδος `simpleOpenNI.enableHand()` η οποία **ενεργοποιεί το σύστημα ανίχνευσης του χεριού του χρήστη**. Πρέπει να κληθεί στη `setup();` Ανιχνεύει ένα χέρι, όποιο "δει" πρώτο το kinect.
b. Οι callback methods που **καλούνται αυτόματα από την simpleOpenNI όταν ανιχνεύεται κατά το χρόνο εκτέλεσης του προγράμματος κάποια κίνηση του χεριού**. Οι μέθοδοι αυτοί πρέπει να υλοποιηθούν, και είναι οι:
   i. `void onNewHand(SimpleOpenNI curContext,int handId,PVector pos)`
   ii. `void onTrackedHand(SimpleOpenNI curContext,int handId,PVector pos)`
   iii. `void onLostHand(SimpleOpenNI curContext,int handId)`
c. Για να κληθούν οι παραπάνω callback methods, πρέπει προηγουμένως να κληθεί η μέθοδος `simpleOpenNI.startTrackingHand(PVector position);`

2. **Ανίχνευση των χειρονομιών** - Αντίστοιχα απαιτείται να χρησιμοποιηθούν μέθοδοι ενεργοποίησης της ανίχνευσης της χειρονομίας και καταγραφής της εξέλιξης της στο χρόνο (callback methods).
   a. Η μέθοδος `simpleOpenNI.startGesture(int gesture)` η οποία **ενεργοποιεί την ανίχνευση συγκεκριμένης χειρονομίας**. Πρέπει να κληθεί στη `setup();` Οι τιμές του `int gesture` μπορούν να είναι (μόνο):
   `SimpleOpenNI.GESTURE_WAVE;`
   `SimpleOpenNI.GESTURE_HAND_RAISE;`
   `SimpleOpenNI.GESTURE_CLICK;`
   b. Οι callback methods που **καλούνται αυτόματα από την simpleOpenNI όταν ανιχνεύεται κατά το χρόνο εκτέλεσης του προγράμματοςκάποια χειρονομία**. Στο συγκεκριμένο παράδειγμα χρειαζόμαστε τη μέθοδο:
      i. `void onCompletedGesture(SimpleOpenNI curContext,int gestureType, PVector pos);` - Η μέθοδος καλείται μόλις ολοκληρωθεί η χειρονομία από το χρήστη. Εδώ θέλουμε να καλέσουμε τη μέθοδο `simpleOpenNI.startTrackingHand(PVector position);` για να ξεκινήσει η καταγραφή των κινήσεων του χεριού.

3. Στο συγκεκριμένο παράδειγμα θέλουμε να ανιχνεύσουμε μια εκ των 3 χειρονομιών που υποστηρίζονται από τη simpleOpenNI και έπειτα απλά να ακολουθήσουμε το χέρι ζωγραφίζοντας ένα ίχνος που το "ακολουθεί". Γι αυτό το λόγο θα χρησιμοποιήσουμε μια λίστα (array list) που θα ονομάσουμε handPositions, από σημεία (θέσεις) του χεριού για τις οποίες θα ζωγραφίζουμε γραμμές που τα ενώνουν.
   a. Η λίστα θα ενημερώνεται αφού τελειώσει ο χρήστης τη χειρονομία, γι αυτό στην callback method onCompletedGesture καλούμε τη μέθοδο startTrackingHand - μετά την κλήση αυτής της μεθόδου, καλούνται οι hand callbacks όπου και ενημερώνεται η λίστα των σημείων (handPositions) όπου βρίσκεται κάθε φορά το χέρι του χρήστη.

b. Αν χαθεί το χέρι από τη σκηνή (callback method onLostHand), η λίστα σημείων αδειάζει (handPositions.clear())
c. Για να ζωγραφιστεί η κίνηση του χεριού, πρέπει να γραφεί μια επανάληψη στη draw(), η οποία (σε κάθε frame) θα διατρέχει τη λίστα handPositions και ζωγραφίζει γραμμές από το προηγούμενο σημείο προς το επόμενο.

Ακολουθεί ο κώδικας για την ανίχνευση των 3 χειρονομιών (απλά βγάλτε από το σχόλιο όποια χειρονομία θέλετε να ασκήσετε).

```
import SimpleOpenNI.*;
SimpleOpenNI kinect;
ArrayList<PVector> handPositions; //this array list stores all hand
positions (3D points), after a gesture has been applied
PVector currentHand; //current position of the hand (3D point)
PVector previousHand; //previous position of the hand (3D point)

void setup() {
  size(640, 480);
  kinect = new SimpleOpenNI(this);
  //kinect.setMirror(true);
  //enable depth camera
  kinect.enableDepth();
  // enable hand + gesture generation
  kinect.enableHand();
  kinect.startGesture(SimpleOpenNI.GESTURE_WAVE);
  //kinect.startGesture(SimpleOpenNI.GESTURE_HAND_RAISE);
  //kinect.startGesture(SimpleOpenNI.GESTURE_CLICK);
  handPositions = new ArrayList();   //create an empty array list
for hand positions...
  stroke(255, 0, 0);
  strokeWeight(2);
}

void draw() {
  kinect.update();
  image(kinect.depthImage(), 0, 0);
  for (int i = 1; i < handPositions.size(); i++) { //i = 1, i.e.
skip the first point because you also need the previous one to draw
the hand trace!
    currentHand = handPositions.get(i); //how is the handPositions
arraylist updated?
    previousHand = handPositions.get(i-1); //how is the
handPositions arraylist updated?
    line(previousHand.x, previousHand.y, currentHand.x,
currentHand.y);
  }
```

```
}

//
------------------------------------------------------------//
hand events
//hand events

void onNewHand(SimpleOpenNI curContext,int handId,PVector pos) {
  println("onNewHand - handId: " + handId + ", pos: " + pos);
  kinect.convertRealWorldToProjective(pos, pos);
  handPositions.add(pos); //it is updated here (a new position is
added), when the hand first appears
}
void onTrackedHand(SimpleOpenNI curContext,int handId,PVector pos)
{
  //println("onTrackedHand - handId: " + handId + ", pos: " + pos
);
  kinect.convertRealWorldToProjective(pos, pos);
  handPositions.add(pos); //it is updated here (a new position is
added), when the hand is tracked (again and again)
}
void onLostHand(SimpleOpenNI curContext,int handId) {
  println("onLostHand - handId: " + handId);
  handPositions.clear(); ////it is updated here (cleared), when the
hand disappears
}

//
------------------------------------------------------------
// gesture events

void onCompletedGesture(SimpleOpenNI curContext,int gestureType,
PVector pos) {
  println("onCompletedGesture - gestureType: " + gestureType + ",
pos: " + pos);
  curContext.startTrackingHand(pos); //after the gesture has been
applied, start tracking hand positions (through hand tracking
callbacks
  //int handId = curContext.startTrackingHand(pos);
  //println("hand tracked: " + handId);
}
```

Το παραπάνω είναι ένα απλό παράδειγμα ενεργοποίησης του μηχανισμού ανίχνευσης χειρονομιών στην simpleOpenNI, για τις χειρονομίες που υποστηρίζονται.

Υπολογισμός της απόστασης μεταξύ των αρθρώσεων, χρησιμοποιώντας τα δεδομένα σκελετού

Η σημασία του υπολογισμού της απόστασης μεταξύ των αρθρώσεων των δεδομένων σκελετού

Ο υπολογισμός της απόστασης μεταξύ αρθρώσεων του ανθρώπινου σκελετού είναι ένα από τα θεμέλια στα οποία στηρίζονται οι κιναισθητικές διεπαφές.

Έστω ότι θέλουμε να κάνουμε μια χειρονομία μεγένθυνσης/σμίκρυνσης της οθόνης μας. Το πρόγραμμα μας θα πρέπει να υπολογίζει την απόσταση των (αρθρώσεων των) δύο χεριών για να εκτελέσει τη χειρονομία. - Έστω ότι θέλουμε να δείξουμε ένα αντικείμενο στην οθόνη. Μπορούμε να υπολογίζουμε την απόσταση του χεριού από τον ώμο, μαζί με το βάθος των δύο αρθρώσεων από την κάμερα (ώστε να βεβαιωθούμε ότι ο χρήστης δείχνει μπροστά). - Έστω ότι ο χρήστης πρέπει να χτυπήσει παλαμάκια, ή να ενώσει τα χέρια του πάνω από το κεφάλι του για να κάνει μια αλληλεπίδραση με το σύστημα… κ.ο.κ. Σε κάθε περίπτωση, η απόσταση 2 (συνήθως) ή περισσότερων αρθρώσεων (που μας ενδιαφέρουν κάθε φορά) είναι πολύ βασική πληροφορία για να καταλάβουμε τη χειρονομία. Όταν η απόσταση έχει συγκεκριμένες τιμές, τότε το σύστημα πρέπει να αντιδράσει ανάλογα.

> *Σημείωση: Όπως έχουμε δει, η βιβλιοθήκη simpleOpenNI μας δίνει πληροφορίες για 15 αρθρώσεις του ανθρώπινου σκελετού. Δεν μπορεί να μας δώσει πληροφορίες για ανθρώπινα δάκτυλα, επομένως δεν έχουμε τη δυνατότητα να εκτελέσουμε χειρονομίες με αυτά χρησιμοποιώντας τα δεδομένα σκελετού. (θα μπορούσαμε βεβαίως να πειραματισμούμε με άλλες λύσεις όπως το να φορέσουμε γάντια με χρώματα στα δάκτυλα, κ.α. αλλά εδώ δεν θα δουλεύαμε με δεδομένα σκελετού (παρά μόνο συνδυαστικά, π.χ. απόσταση άρθρωσης σκελετού από δάκτυλο) αλλά βασικά απευθείας στα δεδομένα των καμερών σε επίπεδο pixel).*
> *2η Σημείωση: Ούτε το kinect SDK 1.8 μας δίνει αρθρώσεις της παλάμης του χρήστη. Η έκδοση 2.0 του αισθητήρα και SDK έγινε τον Ιούλιο του 2014 και υποστηρίζει επιπλέον 2 αρθρώσεις στη παλάμη: του αντίχειρα και του μεσαίου δακτύλου.*

Σύντομη αναφορά στην αφαίρεση διανυσμάτων

Πως υπολογίζουμε την απόσταση μεταξύ δύο αρθρώσεων; Η σύντομη απάντηση είναι η **αφαίρεση διανυσμάτων**. Κάθε άρθρωση του σκελετού είναι ένα σημείο στο 3Δ χώρο. Αλλά είναι και ένα διάνυσμα από το σημείο (0,0,0) προς το σημείο (x,y,z) όπου βρίσκεται η κάθε άρθρωση!

Πιο συγκεκριμένα, μπορούμε να θεωρήσουμε κάθε σημείο της άρθρωσης ως διάνυσμα που:

1. **Περνάει από το σημείο (0,0,0)**. Ο χώρος μας είναι 3Δ, επομένως για κάθε διάνυσμα

χρειαζόμαστε 3 συντεταγμένες για κάθε σημείο.

2. Καταλήγει στο σημείο (x, y, z) όπου βρίσκεται η άρθρωση που μας ενδιαφέρει.
3. Έχει **μήκος** (μέτρο) την απόσταση από το σημείο (0,0,0) μέχρι το σημείο (x,y,z). Η Processing μας παρέχει την μέθοδο `mag();` από το (magnitude = μέγεθος) οποία μας επιστρέφει το μήκος του διανύσματος. Γενικότερα, το μήκος ενός διανύσματος υπολογίζεται ως η τετραγωνική ρίζα του αθροίσματος των τετραγώνων των σημείων του στο χώρο. Δηλαδή, για διάνυσμα AB n-σημείων, το μήκος του συμβολίζεται με |AB| και υπολογίζεται ως εξής:

$$| \overrightarrow{AB} | = \sqrt{\alpha_1^2 + \alpha_2^2 + ... + \alpha_n^2}$$

4. Έχει **κατεύθυνση (προς το σημείο (x,y,z))** που ορίζεται από τις τιμές (xμ, yμ, zμ) του **μοναδιαίου διανύσματος** για το διάνυσμα μας. Το μοναδιαίο διάνυσμα α ενός διανύσματος AB είναι το διάστημα μήκους 1 που ξεκινάει από την αρχή των αξόνων και έχει την ίδια διεύθυνση με το AB. Η Processing μας παρέχει την εντολή `normalize();` η οποία μετατρέπει ένα διάνυσμα στο μοναδιαίο του.

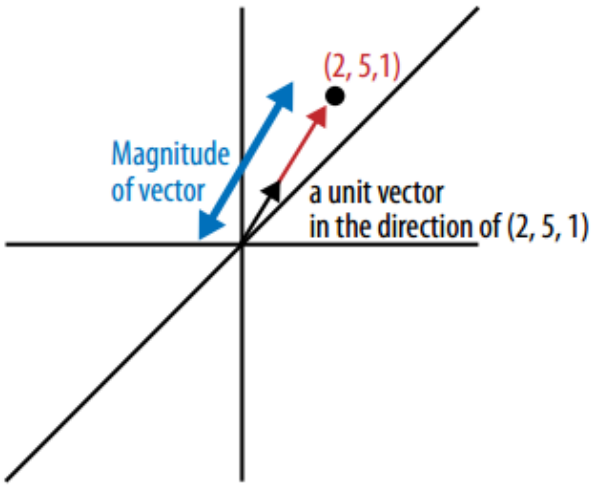


Figure 4-14. *A normalized vector. The vector is broken down into its unit vector, which describes its direction, and its magnitude, which describes the distance it covers.*

Έτσι, π.χ. οι παρακάτω εντολές: ορίζουν ένα διάνυσμα, υπολογίζουν το μέγεθος του και το τυπώνουν στη κονσόλα:

```
PVector myVector = new PVector(2,5,1);
float magnitude = myVector.mag();
myVector.normalize(); //converts myVector into its unit vector!
println("magnitude: " + magnitude);
println(myVector.x, myVector.y, myVector.z);

/*Αν θέλουμε να κρατήσουμε το διάνυσμα μας από τη μετατροπή σε
```

```
μοναδιαίο, θα πρέπει να ορίσουμε ένα αντίγραφο του σε νέα μεταβλητή
και να μετατρέχουμε αυτή σε μοναδιαίο διάνυσμα:*/
PVector myVectorNormalized = new PVector(myVector.x, myVector.y,
myVector.z);
myVectorNormalized.normalize();
```

Σύμφωνα και με το παρακάτω σχήμα, έστω ότι έχουμε 2 σημεία στο 3Δ χώρο που αναπαριστούν τις αρθρώσεις των δύο χεριών του χρήστη. Το αριστερό χέρι είναι στο σημείο (2, 5,1) και το δεξί στο (4, 2, 0). Τα σημεία αυτά μπορούν να αναπαρασταθούν με 2 διανύσματα ΑΒ (αριστερό χέρι), ΑΓ (δεξί χέρι) που ξεκινούν από το σημείο (0, 0, 0). Το διάνυσμα που προκύπτει από την αφαίρεση ΑΒ-ΑΓ είναι ένα τρίτο διάνυσμα ΓΒ που έχει αφετηρία το σημείο Γ και κατάληξη το Β (η κατεύθυνση του είναι προς το σημείο του διανύσματος από το οποίο αφαιρούμε).

Η Processing μας δίνει την εντολή `PVector.sub(vector1, vector2);` η οποία μας επιστρέφει το διάνυσμα που προκύπτει από την αφαίρεση.

Το μήκος του διανύσματος ΑΓ μας δίνει την απόσταση των δύο αρθρώσεων.

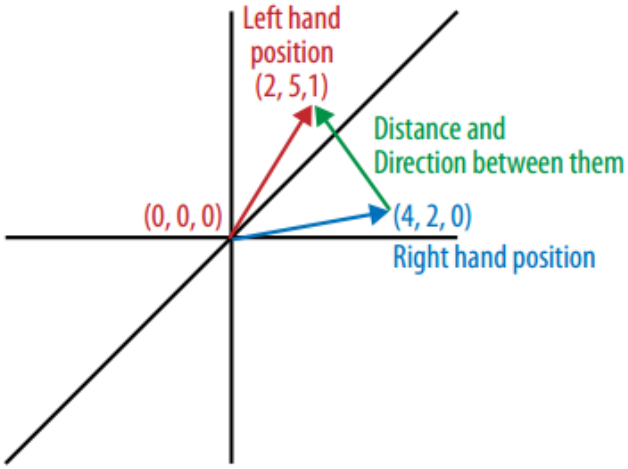


Figure 4-15. *The result of subtracting two vectors is a third vector that takes you from the end point of one to the end point of the other. In this case, it captures the distance and direction between two hands detected by OpenNI.*

Πρόγραμμα υπολογισμού και οπτικοποίησης της απόστασης των αρθρώσεων των 2 χεριών

Με βάση τα παραπάνω, ο υπολογισμός της απόστασης των (αρθρώσεων) των χεριών είναι απλός. Τα βασικά βήματα είναι τα εξής:

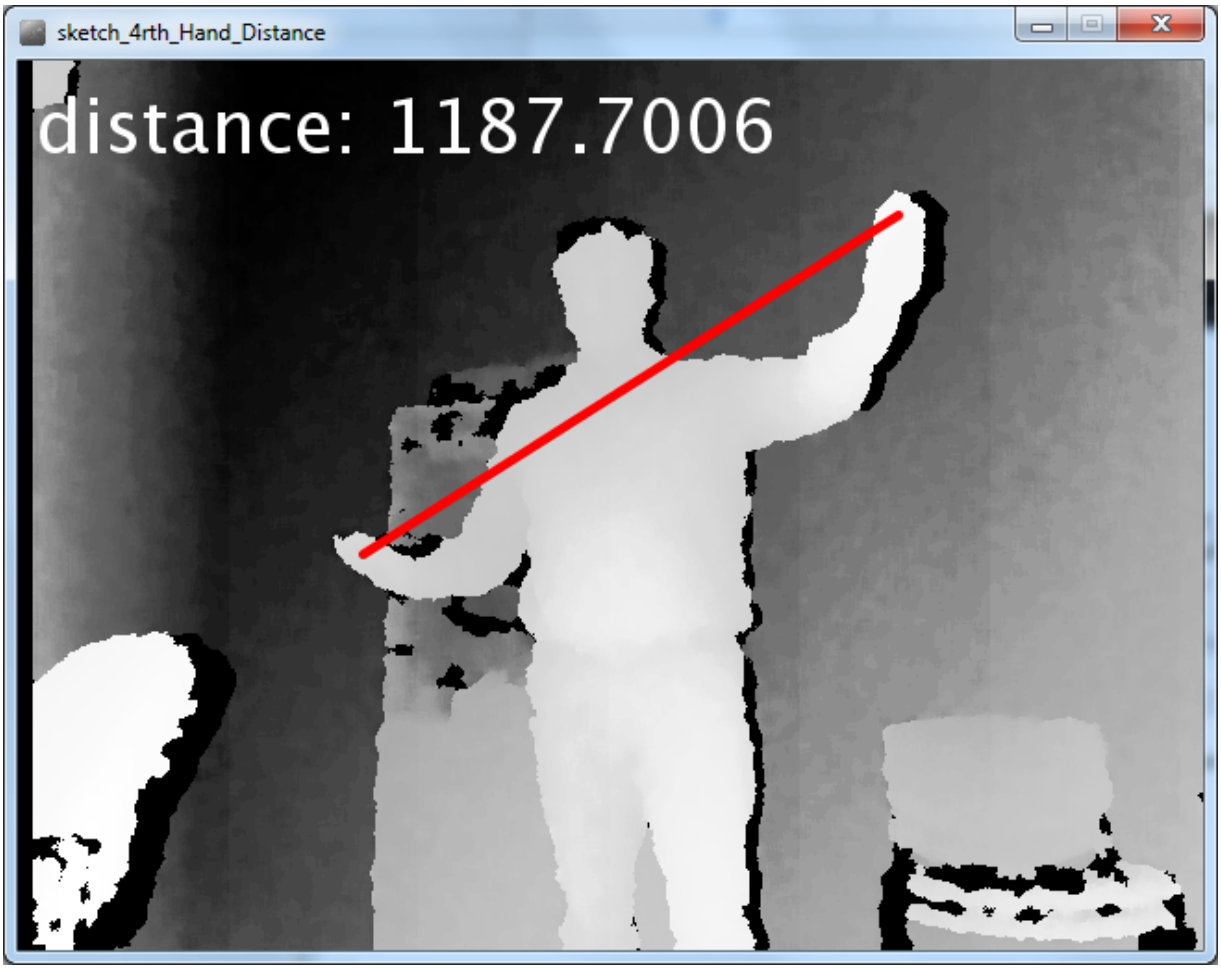
1. Κατά τα γνωστά, εντός της setup() ενεργοποιούμε τη κάμερα βάθους και την ανίχνευση χρηστών πριν ξεκινήσουμε.
2. Κατά τα γνωστά, εντός της draw() ξεκινάμε να ανιχνεύουμε το σκελετό ενός χρήστη.

3. Αποθηκεύουμε τις 2 αρθρώσεις σε δύο PVector.
4. Αφαιρούμε το ένα PVector από το άλλο φιάχοντας ένα νέο διάστημα differenceVector.
5. Ζωγραφίζουμε μια γραμμή μεταξύ των δύο αρθρώσεων.
6. Τυπώνουμε την απόσταση.

```
import SimpleOpenNI.*;
SimpleOpenNI kinect;

void setup() {
  kinect = new SimpleOpenNI(this);
  kinect.enableDepth();
  kinect.enableUser();
  size(640, 480);
  stroke(255, 0, 0);
  strokeWeight(5);
  textSize(40);
}
void draw() {
  kinect.update();
  image(kinect.depthImage(), 0, 0);
  IntVector userList = new IntVector();
  kinect.getUsers(userList);
  if (userList.size() > 0) {
    int userId = userList.get(0);
    if (kinect.isTrackingSkeleton(userId)) {
      PVector leftHand = new PVector();
      PVector rightHand = new PVector();
      kinect.getJointPositionSkeleton(userId,
SimpleOpenNI.SKEL_LEFT_HAND, leftHand);
      kinect.getJointPositionSkeleton(userId,
SimpleOpenNI.SKEL_RIGHT_HAND,rightHand);
      // calculate difference by subtracting one vector from
another
      PVector differenceVector = PVector.sub(leftHand, rightHand);
      // calculate the distance and direction of the difference
vector
      float magnitude = differenceVector.mag();
      differenceVector.normalize();
      // draw a line between the two hands
      kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_HAND,
SimpleOpenNI.SKEL_RIGHT_HAND);
      // display the distance
      text("distance: " + magnitude, 10, 50);
    }
  }
```

```
}
//
--------------------------------------------------------------
// SimpleOpenNI callbacks
void onNewUser(SimpleOpenNI curkinect, int userId) {
  println("onNewUser - userId: " + userId + ", start tracking
skeleton");
  curkinect.startTrackingSkeleton(userId);
}
void onLostUser(SimpleOpenNI curkinect, int userId) {
  println("onLostUser - userId: " + userId);
}
void onVisibleUser(SimpleOpenNI curkinect, int userId) {
  println("onVisibleUser - userId: " + userId);
}
```

Το αποτέλεσμα του προγράμματος δείχνει κάτι τέτοιο. Το παραπάνω πρόγραμμα μπορεί να τροποποιηθεί εύκολα για να μετράει την απόσταση μεταξύ δύο οποιονδήποτε αρθρώσεων του σκελετού.



Επίσης, στην κλάση PVector υπάρχει η μέθοδος dist(PVector secondPVector) η οποία μας επιστρέφει την απόσταση μεταξύ 2 σημείων! Στο συγκεκριμένο παράδειγμα, θα την καλούσαμε ως `leftHand.dist(rightHand);`

Κατασκευή νέων χειρονομιών στη βιβλιοθήκη simpleOpenNI;

Η κατασκευή μιας νέας χειρονομίας στην βιβλιοθήκη simpleOpenNI απαιτεί επέκταση της βιβλιοθήκης - η simpleOpenNI είναι γραμμένη σε Java.

Επιπλέον, η simpleOpenNI χρησιμοποιεί το NiTE software framework της εταιρίας Prime Sense η οποία είναι γραμμένη σε C++. Το NiTE software framework διανεμόταν από την Prime Sense μόνο σε εκτελέσιμο κώδικα (binary). Άρα, η νέα χειρονομία θα πρέπει να υποστηρίζεται από το NiTE, εκ των προτέρων. Το NITE υποστηρίζει τις παρακάτω χειρονομίες: Click, Wave, Raise hand, (αυτές υποστηρίζονται από το simpleOpenNI), Swipe left, Swipe right, Cycle, Hand moved.

Η κατασκευή χειρονομιών εκτός αυτών που χρησιμοποιεί το NiTE framework είναι εφικτή, βλ. http://andrebaltazar.files.wordpress.com/2011/02/nite-controls-1-3-programmers-guide.pdf

Έχουν φτιαχτεί κάποιες χειρονομίες εκτός της simpleOpenNI, βλ. https://github.com/dbu/processing-kinect/tree/master/java-processing και https://code.google.com/p/kineticspace/

Η προσπάθεια να αναπτυχθεί το openSimpleNI έχει ατονήσει τους τελευταίους μήνες, λόγω της εξαγοράς της Prime Sense από την Apple (βλ. και http://www.onformative.com/lab/kinect). Με δεδομένο ότι η Apple δεν φαίνεται ότι θα υποστηρίξει την παραπέρα ανάπτυξη της openSimpleNI, το ερώτημα είναι αν η κοινότητα της Processing θα βασιστεί σε άλλα πλαίσια λογισμικού πέραν του OpenNI, όπως π.χ. http://shiffman.net/p5/kinect/ και http://www.magicandlove.com/blog/research/

==========

Ασκήσεις (30% του τελικού βαθμού, 3 ή 4 άτομα η κάθε ομάδα)

1. Κατασκευάστε πρόγραμμα σε Processing+simpleOpenNI και χρήση του αισθητήρα Kinect, που να χρησιμοποιείται σε μια απλή διαδραστική εγκατάσταση μετάφρασης/απόδοσης ορολογίας ως εξής:
   1.1. Φτιάξτε ένα tag cloud από όρους του πεδίου της Επικοινωνίας Ανθρώπου-Υπολογιστή επιλέγοντας μερικούς όρους από τη σελίδα: http://hci-dpsd.wikispaces.com/hci-translation και δείξτε το στο μισό της οθόνης.
   1.2. Ο χρήστης θα πρέπει να επιλέγει τον όρο δείχνοντας τον, ώστε στο άλλο μισό της οθόνης να εμφανιστεί η μετάφραση/απόδοση του όρου.
   1.3. Παίξτε με διαφορετικά χρώματα, animations, σχήματα ώστε η εφαρμογή να γίνει ελκυστική.
2. Κατασκευάστε πρόγραμμα σε Processing+simpleOpenNI και χρήση του αισθητήρα Kinect, που να χρησιμοποιείται σε μια απλή διαδραστική εγκατάσταση παροχής οδηγιών για τα γραφεία των καθηγητών στο Α' Γυμνάσιο.
   2.1. Αρχικά διατάξτε τις φωτογραφίες και τα ονόματα μερικών καθηγητών του τμήματος στο πρώτο μισό της οθόνης.
   2.2. Καθώς ο χρήστης επιλέγει κάποιο καθηγητή, θα πρέπει να φαίνονται οδηγίες (αριθμός γραφείου και κάποιο σχεδιάγραμμα ή χάρτης προς αυτό).
   2.3. Υποθέστε ότι η εγκατάσταση θα βρίσκεται στον 1ο όροφο του Α' Γυμνασίου, απέναντι ακριβώς από τον ανελκυστήρα (για να φτιάξετε τις οδηγίες/σχεδιάγραμμα από το σημείο αυτό προς τα γραφεία).
3. Κατασκευάστε πρόγραμμα σε Processing+simpleOpenNI και χρήση του αισθητήρα Kinect, που να χρησιμοποιείται σε μια απλή διαδραστική εγκατάσταση παροχής οδηγιών για τις διαφορετικές αίθουσες και κτίρια του Τμήματος.
   3.1. Αρχικά δείξτε σε ένα χάρτη της Ερμούπολης σημάδια ("pins") με τα εξής κτίρια:

Α’ Γυμνάσιο, Γραμματεία, Πνευματικό Κέντρο, Κτίριο Πρώην Καζίνο, Βιβλιοθήκη, Εστιατόριο.

3.2. Καθώς ο χρήστης επιλέγει κάποιο σημάδι (pin) στο χάρτη, θα πρέπει να εμφανίζεται φωτογραφία με την είσοδο του κτιρίου και συνοπτική περιγραφή του (όνομα, δραστηριότητες που φιλοξενεί) στο άλλο μισό της οθόνης.

4. Κατασκευάστε πρόγραμμα σε Processing+simpleOpenNI και χρήση του αισθητήρα Kinect, που να χρησιμοποιείται σε μια απλή διαδραστική εγκατάσταση επίδειξης της ιστορίας του Α’ Γυμνασίου Ερμούπολης ως εξής:
   4.1. Όταν στη σκηνή δεν ανιχνεύονται χρήστες, η εφαρμογή θα δείχνει σε ένα μαύρο φόντο κάποιο βοηθητικό μήνυμα (ή το όνομα της).
   4.2. Όταν ο χρήστης πλησιάσει μεταξύ 6-4 μέτρων, η εφαρμογή θα δείχνει μια παλιά εικόνα του Α’ Γυμνασίου (π.χ. 1960 ή παλαιότερη) μαζί με την αντίστοιχη χρονολογία και σχετικές με την εποχή πληροφορίες (πολύ γενικές, 1-2 προτάσεις, να μη χρειάζεται κύλιση).
   4.3. Όταν ο χρήστης πλησιάσει μεταξύ 4-2,5 μέτρων, η εφαρμογή θα δείχνει μια νεότερη εικόνα του Α’ Γυμνασίου (π.χ. 1990) μαζί με την αντίστοιχη χρονολογία και σχετικές με την εποχή πληροφορίες (πολύ γενικές, 1-2 προτάσεις, να μη χρειάζεται κύλιση).
   4.4. Όταν ο χρήστης πλησιάσει σε απόσταση < 2,5 μέτρων, η εφαρμογή θα δείχνει μια πρόσφατη εικόνα του Α’ Γυμνασίου μαζί με την χρονολογία και σχετικές με την σημερινή χρήση πληροφορίες (πολύ γενικές, 1-2 προτάσεις, να μη χρειάζεται κύλιση).
   4.5. Για ιστορικά στοιχεία και φωτογραφίες σχετικά με τη Σύρο, πέρα από το να ψάξετε στο διαδίκτυο, βλ. τα σχετικά βιβλία (υπάρχουν στη βιβλιοθήκη):
      4.5.1. Αμπελάς, Τ.Δ. (1976) Ιστορία της νήσου Σύρου : από των αρχαιοτατων χρόνων μέχρι των καθ' ημας, Επιμέλεια επανέκδοσης: Χρήστος Α. Καλημέρης.
      4.5.2. Μανος Ελευθερίου (επιμέλεια, 2001) Σύρος : ένα νησί, μια ιστορία, καρτ ποσταλ και φωτογραφίες του 19ου και 20ού αιώνα. Ελληνικά Γράμματα.

5. Ακολουθώντας αντίστοιχα βήματα με τα παραπάνω, κατασκευάστε πρόγραμμα σε Processing+simpleOpenNI και χρήση του αισθητήρα Kinect, που να χρησιμοποιείται σε μια απλή διαδραστική εγκατάσταση επίδειξης την ιστορία της πλατείας Μιαούλη.
6. Ακολουθώντας αντίστοιχα βήματα με τα παραπάνω, κατασκευάστε πρόγραμμα σε Processing+simpleOpenNI και χρήση του αισθητήρα Kinect, που να χρησιμοποιείται σε μια απλή διαδραστική εγκατάσταση επίδειξης την ιστορία του θεάτρου Απόλλων.
7. Ακολουθώντας αντίστοιχα βήματα με τα παραπάνω, κατασκευάστε πρόγραμμα σε Processing+simpleOpenNI και χρήση του αισθητήρα Kinect, που να χρησιμοποιείται σε μια απλή διαδραστική εγκατάσταση επίδειξης την ιστορίας του νησίου της Σύρου (π.χ. με χάρτες απεικόνισης του).
8. Ακολουθώντας αντίστοιχα βήματα με τα παραπάνω, κατασκευάστε πρόγραμμα σε Processing+simpleOpenNI και χρήση του αισθητήρα Kinect, που να χρησιμοποιείται σε μια απλή διαδραστική εγκατάσταση επίδειξης την ιστορίας του οικισμού της Άνω Σύρου.

ΑΝΑΦΟΡΕΣ

Οι σημειώσεις του κεφαλαίου “Εισαγωγή στη γλώσσα προγραμματισμού Processing” βασίζονται στο βιβλίο:

1. Reas, C., & Fry, B. (2007). Processing: a programming handbook for visual designers and artists (Vol. 6812). MIT Press.

Οι σημειώσεις του κεφαλαίου “Χρήση της Processing με τον αισθητήρα MS Kinect ” βασίζονται στο βιβλίο:

2. Borenstein, G. (2012). Making Things See: 3D vision with Kinect, Processing, Arduino, and MakerBot. " O'Reilly Media, Inc.". Κεφάλαια 1, 2, 4.

Τα παραδείγματα των δεδομένων σκελετού (κεφ. 4) του βιβλίου Making things See εργάζονται για παλαιότερη έκδοση της βιβλιοθήκης SimpleOpenNI από την 1.96. Αν χρειαστεί να τρέξετε κάποιο άλλο παράδειγμα θα πρέπει να κάνετε διορθώσεις (πράγμα που έγινε για τις σημειώσεις).

ΑΛΛΕΣ ΠΗΓΕΣ

Processing 2: http://www.processing.org/
Kinect Experiments: overview of available kinect libraries and drivers, (for Processing) http://www.onformative.com/lab/kinect/