UNIVERSITÀ DI PISA

Corso di Laurea in Informatica Umanistica

RELAZIONE

Realizzazione di due installazioni interattive con interfaccia naturale basata sul sensore Microsoft Kinect

Candidato: *Matteo Lupetti*

Relatore: *Elvira Todaro*

Anno Accademico 2012-2013

Indice Generale

8. Appendice
    8.1. Abbreviazioni
    8.2. Codice sorgente “Il paese degli uomini giusti”
    8.3. Codice sorgente “Melissa”

1. Introduzione

La tesi descrive il processo di realizzazione di due installazioni multimediali interattive. L'interazione dell'utente è meditata dal sensore Microsoft Kinect e sviluppata in linguaggio Java all'interno dell'ambiente di sviluppo Processing.
Del primo progetto è stata realizzata la parte grafica e il codice del programma partendo da un idea di un artista locale, il secondo è stato concepito in maniera originale.
Nella relazione vengono esplorati gli aspetti teorici dell'arte interattiva e generativa, come anche il contesto teorico in cui si sta muovendo l'arte post-moderna e post-tecnologica.
Successivamente è analizzato brevemente il sensore Kinect e gli ambienti di sviluppo disponibili per lo sviluppo di applicazioni che utilizzano questo sensore.
Infine sono descritti i processi di realizzazione delle due installazioni.

2.0 L'arte post tecnologica: interattività e tecnologia al servizio della società

2.1 Arte interattiva

Arte interattiva è una forma di arte le cui opere usano il coinvolgimento attivo dei loro fruitori come mezzo per il raggiungimento dello scopo per cui sono state create. Alcune installazioni d'arte interattive raggiungono questo obiettivo richiedendo all'osservatore o il visitatore di "passeggiare" vicino a loro; altre richiedono che il fruitore agisca in modo attivo dentro di esse.

Opere di questo tipo di arte sono spesso dotate di computer e sensori che rilevano movimento, calore, cambiamenti meteorologici o altri tipi di input a seconda di quanto stabilito dall'artista.

Anche se alcuni dei primi esempi di arte interattiva risalgono al 1920, la maggior parte dell'arte digitale ha fatto il suo ingresso ufficiale nel mondo dell'arte alla fine del 1990. Da quel momento in poi, innumerevoli musei hanno dato spazio all'arte digitale e interattiva.

Secondo il new media artist e teorico Maurice Benayoun, dovremmo considerare come primo pezzo di arte interattiva il lavoro fatto da Parrasio durante la sfida con Zeusi descritta da Plinio, nel V secolo aC, quando Zeusi cercò di chiudere il sipario dipinto da Parrasio. L'opera prende il suo significato dal gesto di Zeusi e non esisterebbe senza di esso. Zeusi, con il suo gesto, divenne parte del lavoro di Parrasio. Ciò dimostra che la specificità dell'arte interattiva risiede spesso non nell'uso di computer o tecnologia ma nella qualità delle situazioni proposte e il coinvolgimento dell'*altro* nel processo di creazione di senso. Tuttavia i computer e calcolo in tempo reale

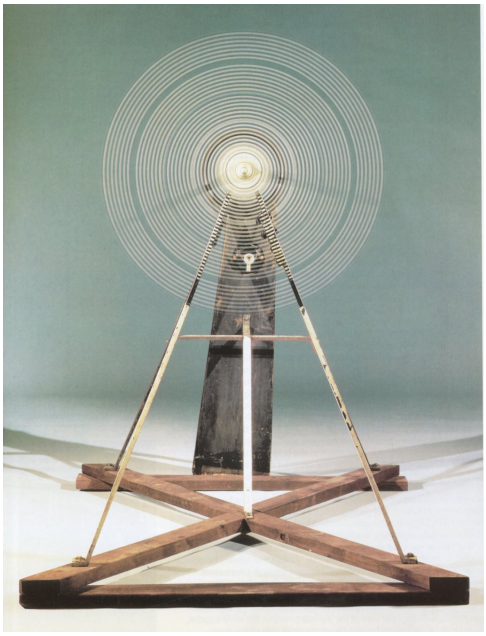


Illustrazione 1: Rotary Glass Plates - Marcel Duchamp

hanno reso più facile il compito e ha aperto il campo della virtualità alle arti contemporanee.

Alcuni dei primi esempi d'arte interattiva risalgono al 1920. Uno di essi è l'opera di Marcel Duchamp chiamato *Lastre di vetro rotanti* (Illustrazione 1). L'opera d'arte richiede allo spettatore di accendere la macchina e stare alla distanza di un metro per vedere un'illusione ottica.

L'attuale idea di arte interattiva ha cominciato a fiorire nel 1960, per motivi in parte politici. A quel tempo, molti artisti trovavano inopportuno che fossero la sola forza creativa all'interno delle loro opere. Gli artisti che sostenevano questa teoria volevano rendere il pubblico partecipe al proprio processo creativo. Uno dei primi esempi sono i "cambio-dipinti" (Illustrazione 2) di Roy Ascott, a proposito dei quali Frank Popper ha scritto:

> "Ascott è stato tra i primi artisti a lanciare un appello per la partecipazione totale dello spettatore".

A parte le motivazioni di carattere “politico”, faceva parte del pensiero corrente che l'interazione e il coinvolgimento svolgessero un ruolo positivo all'interno del processo creativo di un opera.

Nel 1970 gli artisti cominciarono a utilizzare le nuove tecnologie come il video e la comunicazione satellitare, per sperimentare con le performance live e le interazioni attraverso la trasmissione in diretta di video e audio.

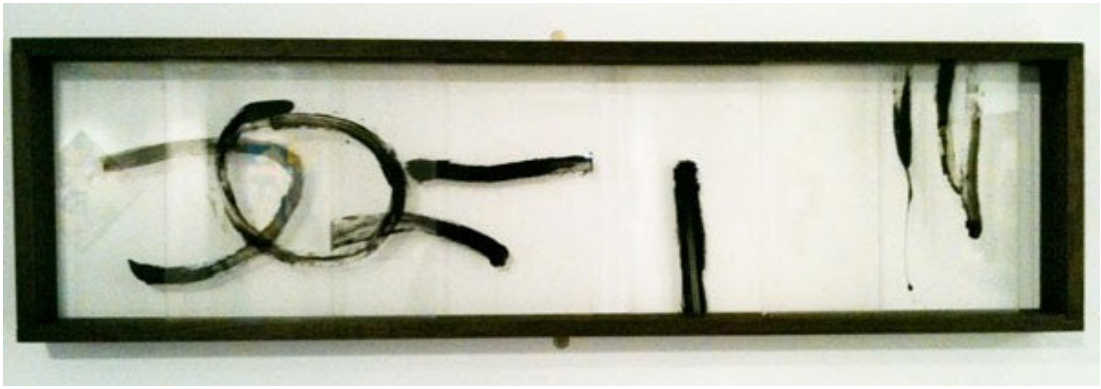


Illustrazione 2: Change Painting - Roy Ascott

L'Arte interattiva ha acquistato sempre più rilevanza dopo il 1990 a causa dell'avvento di interattività basata su computer. Il pubblico e la macchina erano ora in grado di lavorare più facilmente insieme, al fine di produrre un'opera d'arte unica per

ogni fruitore. Dalla fine del 1990, i musei e le gallerie hanno cominciato sempre più ad incorporare questa forma d'arte nei loro spettacoli, alcuni dedicandole intere mostre.

2.2 Arte generativa

Per arte generativa si intende un opera d'arte che in tutto o in parte è stato realizzata con l'impiego di un sistema autonomo. Un sistema autonomo in questo contesto è generalmente definito un agente non umano che può autonomamente determinare caratteristiche di un'opera, che altrimenti richiederebbero decisioni dirette dall'artista. In alcuni casi il creatore umano può sostenere che il sistema generativo rappresenta la propria idea artistica, in altri che il sistema assume il ruolo del creatore.

Più spesso si parla ad arte generativa per riferirsi ad opere d'arte generate su un calcolatore tramite l'utilizzo di un algoritmo.

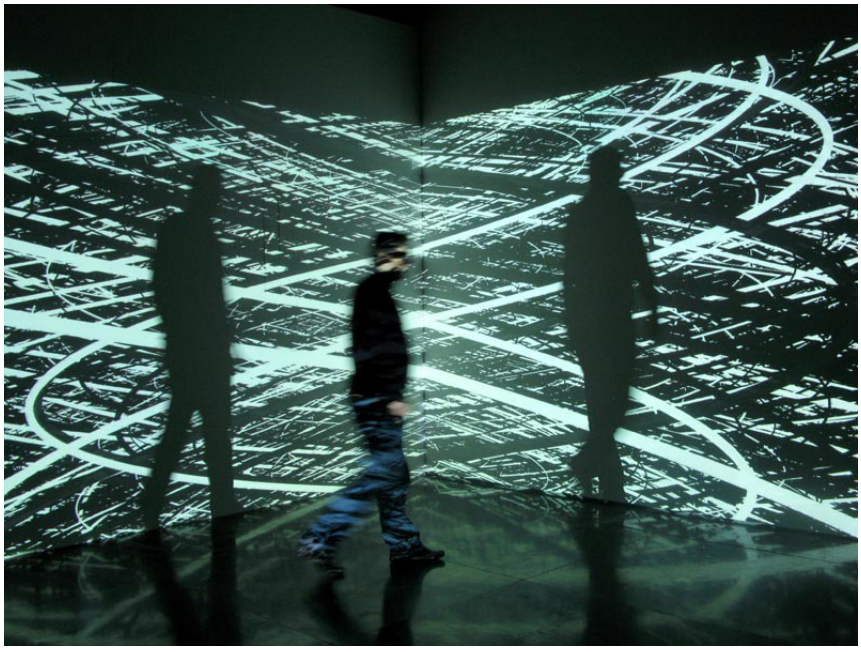


Illustrazione 3: Intallazione "Irrationnal Geometrics", 2008, Pascal Dombis

Sistemi di arte generativa possono essere classificati come essere ordinati, disordinati, o complessi. I sistemi complessi sono quelli che hanno una miscela di entrambi i sistemi ordinati e disordinati e in genere mostrano comportamento emergente.

I sistemi ordinati possono includere arte seriale, mappatura dei dati , l'uso della simmetria e *tiles*, sequenze di numeri e serie, le proporzioni, come il rapporto aureo, e la matematica combinatoria. Sistemi di arte generativa disordinati tipicamente sfruttano una qualche forma di randomizzazione, sistemi stocastici, o aspetti della teoria del caos .

Mentre i sistemi di arte generativa ordinati sono vecchi come l'arte stessa, ed i sistemi di arte generativa disordinati venuto alla ribalta nel 20 ° secolo, la pratica dell'arte generativa contemporanea tende ad inclinarsi in direzione dei sistemi generativi complessi.

Sistemi generativi computazionali che si muovono verso la complessità comprendono l'aggregazione a diffusione limitata, il calcolo evolutivo, L-sistemi, le reti neurali, automi cellulari, sistemi di reazione-diffusione, vita artificiale, e altri metodi biologicamente ispirati, come comportamento sciame .

Mentre parte della produzione dell'arte generativa esiste come artefatti statici prodotti da precedenti processi invisibili, arte generativa può essere anche vista come lo sviluppo di un opera in tempo reale. In genere tali opere non sono visualizzati mai nello stesso modo due volte. Ad esempio, ambienti di programmazione grafica (Max / MSP , Pure Data o vvvv), così come ambienti di programmazione classici e *user-friendly*, come Processing o openFrameworks sono utilizzati per creare in tempo reale generativi audiovisivi.

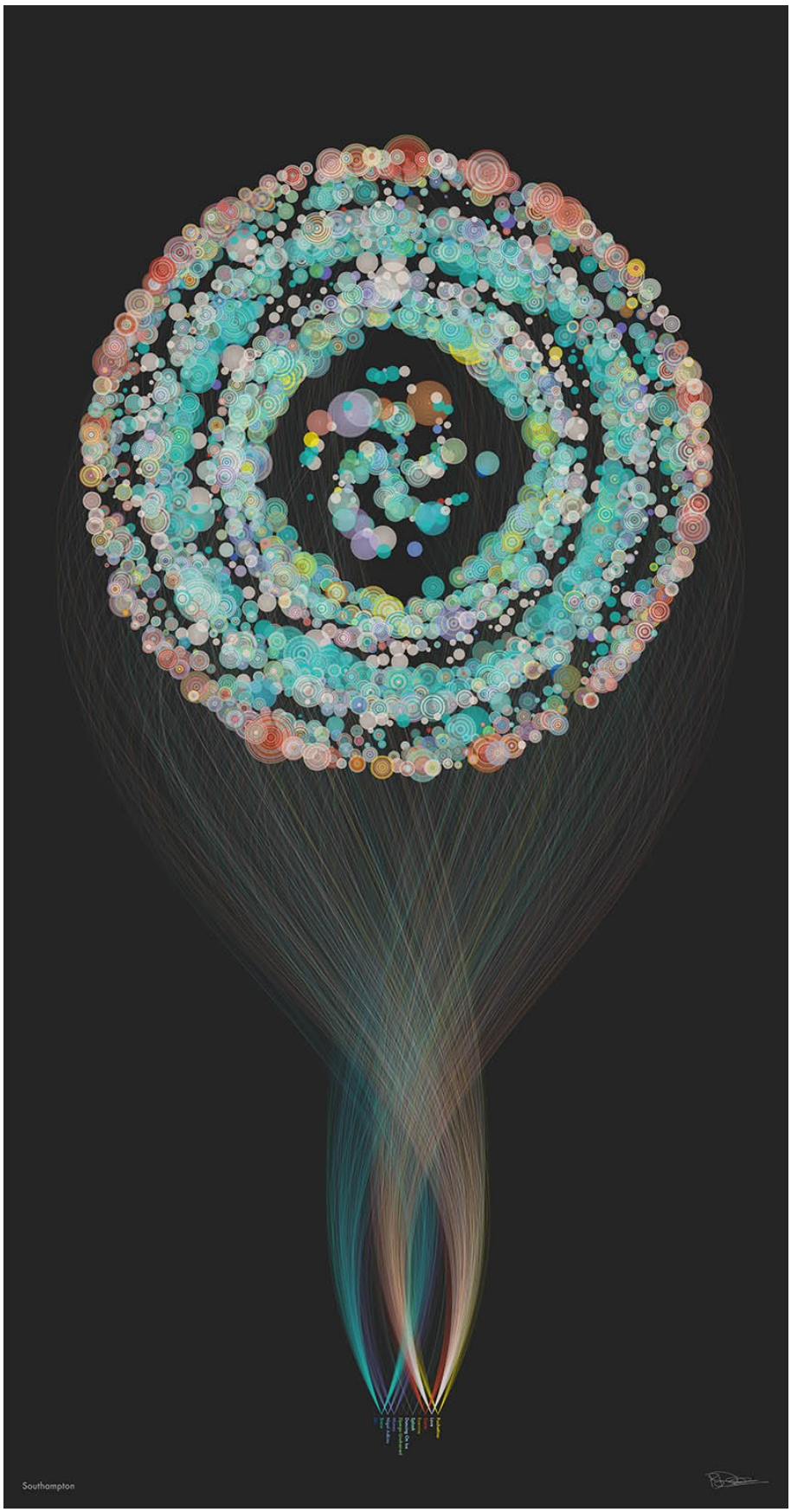


Illustrazione 4: Digital city portraits, Brendan Dawes

2.3 L'arte nell'era post-tecnologica

Molti artisti e collettivi di artisti hanno cominciato negli ultimi anni ad esplorare le possibilità aperte dalla tecnologia a basso costo ed a intendere le proprie installazioni come ponti tra due mondi soventemente considerati distanti: tecnologia e arte/filosofia.

In particolare, nel campo dell'interattività gli artisti si rivolgono alla tecnologia per la necessità di un interazione non mediata ma basata su interfacce naturali.

Oltre a questo è emerso il rifiuto di considerare le proprie opere come mere sperimentazioni estetico/tecnologiche ma al contrario far si che siano portatrici di messaggi di cambiamento culturale.

In particolare si tende a utilizzare l'interazione basata sulla tecnologia per superare l'alienazione dei sensi che permea la società post tecnologica, si utilizza quindi la tecnologia ed il virtuale per promuovere il cambiamento socioculturale e ridare senso ai sensi.

3. La piattaforma Microsoft Kinect

3.1 Cos'è

Kinect (in fase di sviluppo conosciuto con il nome in codice Project Natal) è un dispositivo di *motion sensing* ideato da Microsoft per la console per videogiochi Xbox 360 e PC Windows.
Basato su una piattaforma hardware esterna alla console o il PC, consente agli utenti di controllare e interagire con il sistema attraverso una interfaccia utente naturale, usando gesti e comandi vocali senza la necessità di un controller fisico.
Il sensore Kinect è composto principalmente da un sistema di emettitore/rilevatore infrarosso, un sensore a colori e quattro microfoni direzionali.

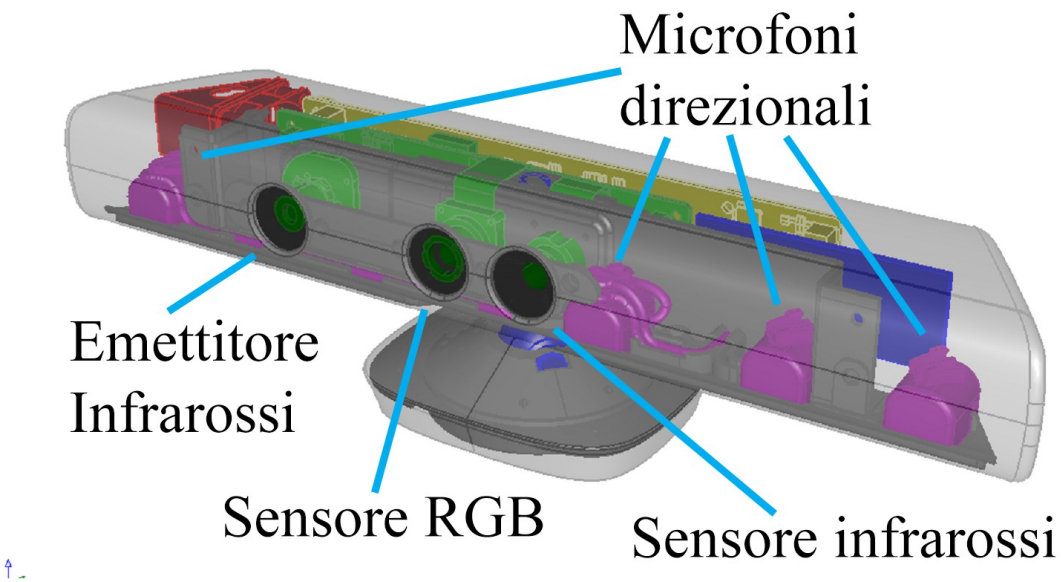


Illustrazione 5: Il sensore Microsoft Kinect - Componenti interni

Il sistema ricava la mappa della distanza nel campo visivo del rilevatore infrarosso tramite la triangolazione dei fasci provenienti dall'emettitore. Una volta ricavata, la mappa della scena viene trasmessa ad un sistema che si occupa di individuare i corpi delle persone, dividerle in parti e calcolare le coordinate più probabili delle articolazioni degli utenti.

Illustrazione 6: La creazione dello scheletro a partire dalla mappa della profondità

Le caratteristiche tecniche del sensore sono le seguenti:

Campo visivo

- Orizzontale: 57 Gradi
- Verticale: 43 Gradi
- Capacità di spostamento della periferica: 27 gradi
- Rilevamento della profondità: 1.2m - 3.5m

Trasferimento dati

- Trasmissione dati del sensore di profondità a risoluzione di 320×240 16-bit con frequenza di aggiornamento di 30 frames/sec
- Trasmissione dati del sensore a colori a risoluzione di 640×480 32-bit colore con frequenza di aggiornamento di 30 frames/sec

Tracciamento scheletrico

- Fino a 6 persone, inclusi 2 giocatori attivi
- Fino a 20 movimenti per ogni giocatore attivo

Audio

- Cancellazione dell'eco
- Riconoscimento vocale di più parlanti

Poiché il sensore Kinect è motorizzato, il meccanismo di inclinazione richiede più energia rispetto a quanto possa fornire una porta USB. Il dispositivo si avvale quindi di un connettore proprietario che unisce la comunicazione USB con alimentazione supplementare. I nuovi modelli di Xbox 360 S hanno incluso una porta AUX speciale per l'alloggiamento del connettore, mentre i vecchi modelli di Xbox e i PC richiedono uno speciale cavo di alimentazione (in dotazione con il sensore), che separa l'alimentazione dal trasferimento dati.

3.2 Storia

La tecnologia di rilevamento di profondità dietro Kinect è stato inventata nel 2005 da Zeev Zalevsky , Alexander Shpunt, Aviad Maizels e Javier Garcia presso la società PrimeSense.
Kinect è stato annunciato il 1 giugno 2009 alle E3 2009 sotto il nome in codice "Project Natal".
Successivamente è stato lanciato sul mercato in Nord America il 4 novembre 2010, in Europa il 10 novembre 2010 in Australia, Nuova Zelanda e Singapore il 18 novembre 2010 e in Giappone il 20 novembre 2010.
Nel dicembre 2010 PrimeSense ha rilasciato i propri driver open source OpenNI e il modulo di per il tracciamento scheletrico NITE, successivamente ha annunciato di aver collaborato con Asus per sviluppare un dispositivo PC-compatibile simile a Kinect, chiamato Wavi Xtion. Il dispositivo è stato commercializzato nel secondo trimestre 2011.
Kinect Microsoft ha rilasciato il suo kit di sviluppo software ufficiale per Windows 7 il 16 giugno 2011. Questo SDK è stato pensato per consentire agli sviluppatori di scrivere applicazioni per Kinect in C + + / CLI , C # o Visual Basic. NET.

3.3 Applicazioni

Il sensore Kinect viene ad oggi utilizzato come interfaccia naturale negli ambiti più disparati, numerosi sviluppatori hanno infatti utilizzato Kinect per applicazioni che vanno oltre la destinazione del sistema di giocare.
Utilizzare il sensore Kinect al posto di una normale videocamera a colori porta i vantaggi e svantaggi sintetizzati nella tabella.

Vantaggi	Svantaggi
Informazioni accurate sulla profondità della scena (massimo 1mm)	Sensibile alle fonti esterne di infrarossi (luce solare diretta). Difficilmente utilizzabile all'aperto.
Informazioni 3D a bassa latenza	Raggio di utilizzo limitato
Aggiornamento in tempo reale: funzionamento normale a 30 Fps	Non rileva componendi della scena altamente riflettenti o di composizione cristallina
Limitazione del raggio di utilizzo non rilevante per la quasi totalità delle applicazioni	Difficoltà di funzionamento in assenza totale di luce
Basso costo	
Portatile	

Philipp Robbel del MIT ha combinato il sensore Kinect con l' iRobot Create per ricreare la mappa una stanza in 3D far rispondere il robot ai gesti umani, mentre il team del MIT Media Lab sta lavorando a una estensione JavaScript per Google Chrome chiamato DepthJS che permette agli utenti di controllare il browser con i gesti delle mani. Un altro gruppo ha mostrato un'applicazione che consente agli utenti di Kinect di suonare un pianoforte virtuale toccando le dita su una scrivania vuota.
Alexandre Alahi alla École polytechnique fédérale de Lausanne ha presentato un sistema di videosorveglianza che combina più dispositivi Kinect per seguire gruppi di persone anche in completa oscurità.
Grazie al sensore in grado di restituire una mappatura 3D della scena Kinect è stato anche modificato per funzionare come scanner 3D, in particolare l'università della California ha sviluppato una versione portatile di Kinect destinata ad essere utilizzata come scanner 3D negli scavi archeologici.

3.4 Ambienti di sviluppo

Come accennato in 3.2 esistono almeno due ambienti di sviluppo software che permettono di accedere ed elaborare i dati del sensore Kinect a livello alto: l'sdk ufficiale di Microsoft e l'sdk open source OpenNI (accompagnato dal *middleware* per l'elaborazione scheletrica NITE).

3.4.1 Microsoft Kinect Sdk

L'ambiente di sviluppo ufficiale di Microsoft basato su windows permette di sviluppare applicazioni che interagiscono con Kinect tramite Visual Studio 2010 e successivi. Questo porta il vantaggio di poter scrivere il codice dell'applicazione sia in C + +, C # o Visual Basic. NET.

Altro vantaggio importante è l'integrazione immediata con le librerie .NET e la possibilità di creare semplicemente applicazioni per Windows in modalità WPF. L'utilizzo di questo ambiente è anche attualmente l'unico modo per sfruttare al massimo i quattro microfoni direzionali per l'interazione vocale.

Essendo delle librerie ufficiali vengono mantenute ed aggiornate frequentemente, permettendo così di utilizzare il sensore kinect al massimo delle sue possibilità.

Il principale svantaggio di utilizzare queste librerie è che l'applicativo risultante è utilizzabile solamente in ambiente Windows.

3.4.2 OpenNi e NITE

L'sdk open source OpenNi ha come vantaggio principale sul concorrente Microsoft il fatto di essere utilizzabile con diversi sistemi operativi: Linux, Mac OS e Windows.

La simulazione scheletrica non è integrata ma è gestita da un middleware chiamato NITE.

3.4.3 Framework Grafici

Per realizzare un installazione interattiva non basta scegliere il mezzo con cui interfacciarsi con i sensori, si deve sopratutto scegliere un ambiente di sviluppo grafico che si occupi di interpretare gli input del sdk e trasformali in esperienza visiva.

Negli ultimi anni sono proliferate librerie e ambienti di sviluppo specificamente pensati per semplificare il lavoro dei creativi, qui andremo ad analizzare le tre prese in considerazione per la realizzazione dei progetti.

3.4.3.1 openFrameworks

openFrameworks è un toolkit open source scritto in C + + e progettato per assistere

il processo creativo, fornendo un ambiente semplice e intuitivo per la sperimentazione. Il toolkit è stato progettato per funzionare come un collante di uso generale, ed integra insieme diverse librerie di uso comune, tra cui:

- OpenGL , GLEW , GLUT , libtess2 e cairo per la grafica
- RTAudio , PortAudio , OpenAL e Bacio FFT o FMOD per input, output e analisi dell'audio
- FreeType per i font
- FreeImage per un'immagine salvataggio e il caricamento
- Quicktime , GStreamer e videoInput per la riproduzione video e afferrando
- Poco per una serie di programmi di utilità
- OpenCV per computer vision
- ASSIMP per caricare i modelli 3D

openFrameworks supporta i sistemi operativi Windows, OSX, Linux, iOS, Android e quattro IDE: XCode, Code :: Blocks, e Visual Studio ed Eclipse. L'API è stata progettata per essere minimalista e facile da imparare.

3.4.3.2 Cinder

Altra libreria scritta in C++ è Cinder, le maggiori differenze con openFramework sono:

- supporta attualmente solo Windows e Mac OSX;
- Cinder utilizza librerie più specifiche di sistema per migliorare le prestazioni mentre openFrameworks permette un migliore controllo sulle sue librerie integrate.

3.4.3.3 Processing

Processing è un linguaggio di programmazione e ambiente di sviluppo integrato (IDE) open source, costruito per le arti elettroniche, new media art e visual design con lo scopo di insegnare i fondamenti della programmazione in un contesto visivo.

Il progetto è stato avviato nel 2001 da Casey Reas e Benjamin Fry , entrambi ex membri dell'Aesthetics and Computation Group all'interno del MIT Media Lab, con lo scopo di insegnare la programmazione attraverso la gratificazione immediata di un riscontro visivo.

La sintassi si basa sul linguaggio Java semplificato per facilitare la programmazione orientata alla grafica.

Processing comprende uno *Sketchbook*, una alternativa minima ad un IDE per l'organizzazione dei progetti.

Ogni sketch di Processing è in realtà una sottoclasse della classe Java PApplet, che implementa la maggior parte delle caratteristiche del linguaggio Processing.

In Processing, ogni sketch deve contenere almeno una classe principale, in cui sarà presente un metodo setup e un metodo draw: la prima viene invocata una sola volta al lancio dell'applicazione, mentre la seconda viene eseguita per ogni frame.

```
// ESEMPIO DI SKETCH PROCESSING
// dichiarazione di variabili globali
float xoffset = 0.0;
float yoffset = 0.0;

// metodo setup - eseguito solo all'avvio
void setup()
{
  size(400, 400);
  PFont font = loadFont("Calibri-24.vlw");
  textFont(font, 24);
  smooth();
  println("Stampo una sola volta su console");
}
```

```
// metodo draw - eseguito per ogni frame
void draw()
{
  background(128);
  println("Stampo ad ogni frame su console");
  text("Stampo ad ogni frame su schermo", xoffset, yoffset);
  xoffset++;
  yoffset++;
}
```

4. Installazione “Il paese degli uomini giusti”

4.1 Contesto

L'occasione di approfondimento della piattaforma Kinect si è presentata con una commissione di un lavoro da parte di un artista locale, Alessandro Bonocore. L'installazione multimediale interattiva è stata ideata per presentare poesie e brani musicali in un contesto narrativo comune. L'interazione con l'utente fa parte del filone comunicativo che l'artista ha immaginato per le sue opere.
Qui di seguito la sintesi dell'opera fatta dall'artista:

> “Inizia una musica di sottofondo, dall'alto scendono lentamente varie lettere e qualche numero in maniera casuale. Nel cadere possono ruotare su se stesse, ondeggiare. Man mano che cadono si depositano sul fondo.
>
> Siamo nel luogo dove vivono gli uomini giusti, un visitatore arriva, si pone davanti alle porte di questa città (lo schermo) ed apre le braccia come per accogliere le lettere che stanno cadendo.
>
> A quel punto le lettere casuali che scendono nella zona delle braccia aperte si fermano, si appoggiano trovando "accoglienza" sulle braccia e diventano messaggi di purezza, versi che scorrono, dando corpo ad una poesia, nello stesso istante dissolve la musica di sottofondo ed inizia la musica legata a quella poesia.

A quel punto tutte le parole in precedenza depositate sul fondo, raggiunta una certa altezza, iniziano a bruciare.

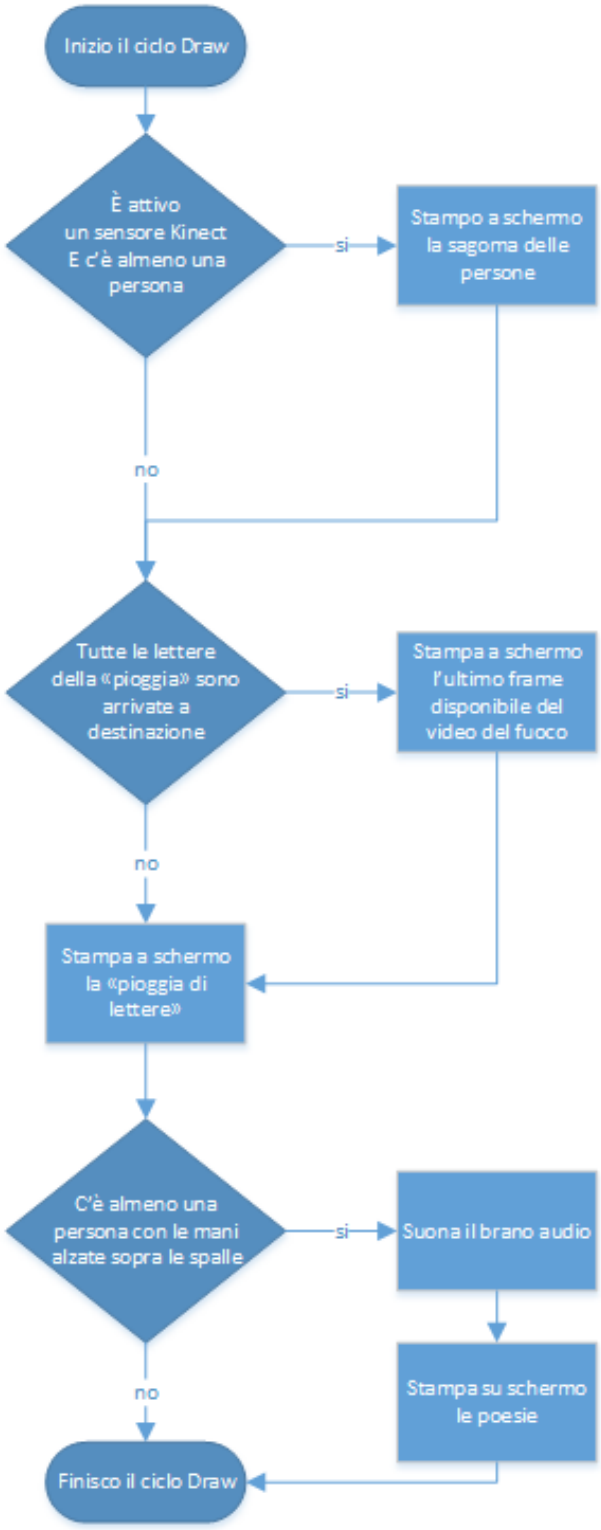
Il fuoco rappresenta nello stesso momento due significati: prima di tutto un inceneritore delle parole superflue e vuote, delle bugie, etc. ma anche il fuoco sacro che è in noi e che avvolge la nostra parte poetica, lasciandola pura e intatta.

Le fiamme si alzano sempre di più, si salva dalle fiamme soltanto la zona occupata dalla persona, dalle braccia aperte in su, dove, in quello "spazio protetto", è presente la poesia.

Tutta la poesia scorre e rimane appoggiata sulle braccia finché rimangono aperte ma, al momento che vengono chiuse, le parole della poesia cadono sul fondo; anche la musica collegata a quella poesia si interrompe e svanisce dissolvendo.

A questo punto la cosa si ripete sempre uguale da capo. Quello che cambia ogni volta è una poesia diversa associata ad una relativa musica diversa.”

L'installazione consiste in un PC collegato ad un sensore Kinect e ad un proiettore, su cui viene eseguito il programma in java. Ad ogni ciclo il programma funziona secondo il seguente diagramma:



4.2 Scelta dell'ambiente di sviluppo

Dopo un iniziale sperimentazione in ambiente di sviluppo Visual C sfruttando l'sdk ufficiale di Microsoft, lo sviluppo è migrato in ambiente di sviluppo Processing e l'sdk OpenNi.
Questo ha non solo permesso di sviluppare un programma multi-piattaforma ma anche di accelerare il processo di creazione in quanto Processing è espressamente pensato per sviluppare applicazioni grafiche in maniera semplificata.
Lo svantaggio principale è stato il non poter utilizzare le funzioni avanzate dell'sdk ufficiale.

4.3 Criteri di programmazione

Il programma si appoggia su librerie esterne per la gestione di Kinect (libreria SimpleOpenNi), la gestione dell'audio (libreria Minim) , dello streaming video (libreria Processing Video / GStreamer), e delle animazioni (libreria Ani).

Nella prima versione del programma si era proceduto ad implementare la simulazione fisica (impatto/reazione) 2D delle lettere che cadono e della sagoma dell'utente tramite la libreria Fisica per Processing. Nonostante questo approccio permettesse un maggior realismo e una maggior interazione dell'utente con l'installazione, il carico di calcolo si è rivelato eccessivo ed il *frame rate* finale scendeva spesso sotto livelli accettabili.

La seconda soluzione rimuove totalmente la simulazione fisica e implementa semplicemente un moto ondulatorio alle lettere che scendono.
Il programma è stato diviso in 8 file per comodità secondo la seguente logica:

- Il file Letters contiene la logica principale del programma: l'inclusione delle librerie utilizzate, la dichiarazione delle variabili globali, del metodo setup e del ciclo draw.
- Il file AniChar contiene la definizione della classe che implementa le lettere che andranno a comporre i versi delle poesie.
- il file Function contiene le funzioni di uso generale: la funzione che stampa le informazioni di debug, la funzione di inizializzazione dello scheletro e la funzione di fast blur.

- Il file Poesia contiene la definizione della classe per la gestione della poesia dal caricamento del file XML alla temporizzazione del render dei singoli versi.
- Il file Scene_Mapper contiene la definizione della classe che estrapola l'immagine della silhouette delle persone rilevate dal sensore Kinect.
- Il file TChar contiene la definizione della classe che implementa le lettere della “pioggia di lettere”.
- Il file Timer contiene la definizione della classe che implementa un rudimentale timer.

Le poesie sono immagazzinate in un file XML e codificate secondo lo standard TEI 5 per rendere più facile un futuro aggiornamento.

4.4 Realizzazione ed implementazione del fuoco

Il “fuoco” che nell'intenzione dell'artista viene a bruciare le lettere e a purificare il *mondo* dell'installazione è un video realizzato con il programma After Effect. La scena è composta da numerosi emettitori di particelle che simulano le fiamme del fuoco (Illustrazione 7). Il video è poi stato renderizzato ed importato in Processing, dove viene riprodotto sfruttando la libreria GStreamer.

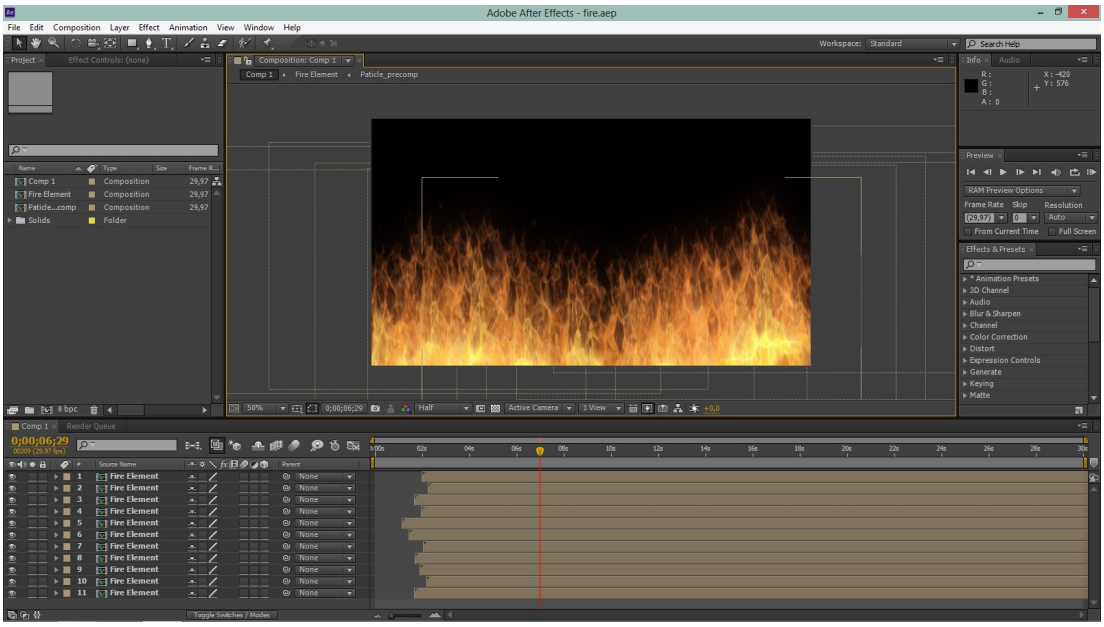
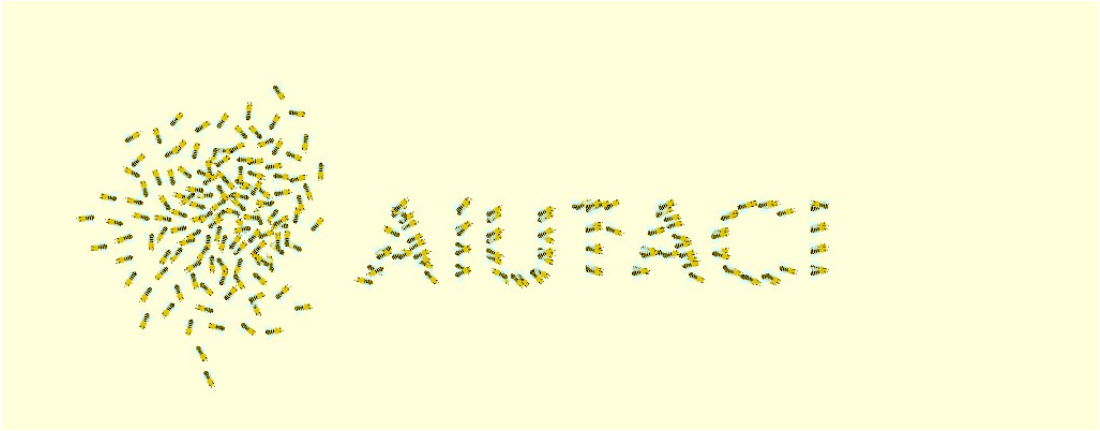


Illustrazione 7: Schermata del video del fuoco in After Effect, prima del rendering finale

5. Installazione “Melissa”



5.1 Contesto

La seconda installazione si sviluppa nel contesto comunicativo della campagna “Salviamo le Api” dell'ONG Greenpeace. Dal rapporto *Bees in Decline: a review of factors that put pollinators and agriculture in Europe at risk:*

> “La prossima volta che vediamo un’ape ronzarci intorno ricordiamoci che la maggior parte del cibo che mangiamo dipende in modo significativo dall’opera delle api e degli altri insetti impollinatori, un servizio chiave per gli ecosistemi.
>
> Senza l’impollinazione effettuata dagli insetti, circa un terzo delle colture a scopo alimentare dovrebbe essere impollinato con altri mezzi, oppure avremmo una produzione di cibo significativamente inferiore.
>
> Senza dubbio le colture più nutrienti e apprezzate della nostra dieta – molta frutta e verdura, insieme ad alcune colture utilizzate come foraggio nella produzione di carne e prodotti lattiero-caseari – sarebbero duramente colpite da un calo numerico degli insetti impollinatori: in particolare, la produzione di mele, fragole, pomodori e mandorle ne soffrirebbe.
>
> Fino al 75 per cento delle nostre colture subirebbe comunque una riduzione di produttività. La stima più recente dei benefici economici a livello globale legati all’impollinazione, ammonta a circa 265 miliardi di euro, questo il valore delle colture che dipendono dall’impollinazione naturale.
>
> Parliamo ovviamente non del valore “reale”, dato che, qualora l’impollinazione naturale venisse gravemente compromessa o dovesse cessare, potrebbe rivelarsi impossibile da sostituire, rendendo il suo vero

valore infinitamente maggiore. Oltre alle coltivazioni, anche le piante selvatiche (si stima dal 60 al 90 per cento) dipendono dall'impollinazione mediata dagli insetti per riprodursi. Di conseguenza anche altri servizi eco-sistemici e gli habitat naturali che li forniscono dipendono – direttamente o indirettamente – dagli insetti impollinatori. Le api – quelle allevate, ma anche molte specie selvatiche – sono il gruppo predominante ed economicamente più importante degli impollinatori nella maggior parte delle regioni geografiche. Le colonie di api domestiche, tuttavia, negli ultimi anni hanno sofferto in misura sempre crescente, nonostante la produzione agricola a livello planetario dipenda dalla loro opera d'impollinazione. Anche il ruolo degli insetti impollinatori selvatici – api e altre specie – è molto rilevante e attrae sempre maggiore attenzione del mondo della ricerca. Le api selvatiche sono a loro volta minacciate da numerosi fattori ambientali, tra cui la mancanza di habitat naturali e semi-naturali, e una crescente esposizione a sostanze chimiche prodotte dall'uomo."

"Le api domestiche e gli altri impollinatori selvatici hanno un ruolo cruciale nella produzione agricola e alimentare. Tuttavia, l'attuale modello agricolo fortemente dipendente dalla chimica, sta minacciando questi insetti, e conseguentemente mettendo a rischio l'approvvigionamento alimentare a livello europeo.

Questo rapporto vuole sottolineare che vi sono forti evidenze scientifiche che individuano responsabilità importanti dei neonicotinoidi e di altri pesticidi nell'attuale declino delle api. Di conseguenza, l'Unione Europea e i governi nazionali dovrebbero:

1 Vietare l'uso di pesticidi nocivi per le api, a partire dalle sette sostanze più pericolose attualmente autorizzate nell'Unione Europea: imidacloprid, thiamethoxam, clothianidin, fipronil, clorpirifos, cipermetrina e deltametrina.

2 Attraverso l'adozione di piani d'azione nazionali per gli insetti impollinatori, sostenere e promuovere pratiche agricole che apportino benefici al servizio di impollinazione all'interno dei sistemi agricoli, come la rotazione delle colture, la promozione di aree di interesse ecologico a livello aziendale e i metodi di agricoltura biologica.

3 migliorare la conservazione di habitat naturali e semi-naturali all'interno e intorno alle aree agricole, nonché incrementare la biodiversità nei campi.

4 Aumentare i finanziamenti per ricerca, sviluppo e applicazione di pratiche agricole ecologiche che si allontanino dalla dipendenza da sostanze chimiche per il controllo dei parassiti, per andare verso l'uso di strumenti basati sulla biodiversità per controllare i parassiti e migliorare la salute degli ecosistemi. A livello europeo bisogna indirizzare maggiori

fondi per la ricerca sull'agricoltura ecologica nell'ambito della PAC (pagamenti diretti) e di Orizzonte 2020 (programma europeo di ricerca)."

L'installazione si propone di sensibilizzare l'opinione pubblica dando il mandato di veicolare i messaggi direttamente alle api stesse (sebbene virtuali).

Uno sciame di api viene proiettato su una parete di un luogo pubblico: quando una persona passa davanti allo schermo lo sciame ne segue i movimenti, e se questa si ferma lo sciame comincia a prendere la forma di messaggi.

In particolare si è voluto dare carattere positivo alla comunicazione: i messaggi iniziano con un cuore o comunque un altro simbolo positivo e amichevole e continuano con l'esposizione di un fatto relativo alla campagna, quasi sempre relativo al nesso api-cibo (esempio "un terzo della frutta che mangi esiste grazie a noi"). Dopo questo arriva una richiesta di aiuto e il rimando ad un contatto della campagna (esempio "sos-bees.org").

5.2 Criteri di programmazione

Sebbene la composizione *hardware* dell'installazione sia esattamente la stessa della precedente (PC, videoproiettore e sensore Kinect), i criteri di programmazione sono sensibilmente variati.

Se il progetto precedente ricadeva solo sotto la categoria di installazione interattiva, questa rientra a invece nel campo dell'arte generativa. Il programma realizzato in Processing fa infatti uso per determinare i movimenti dello sciame d'api di un algoritmo di *flocking* basato sul modello delineato nel 1986 da Craig Reynolds nella pubblicazione "Flocks, Herds, and Schools: A Distributed Behavioral Model."

In particolare ogni veicolo (ape) segue tre regole principali:

- Separazione (o anche "evadere"): orientarsi per evitare la collisione con i tuoi vicini
- Allineamento (o anche "copiare"): orientarsi nella stessa direzione dei tuoi vicini
- Coesione (o anche "centrarsi"): orientarsi nella direzione del centro dei tuoi vicini ("stare nel gruppo")

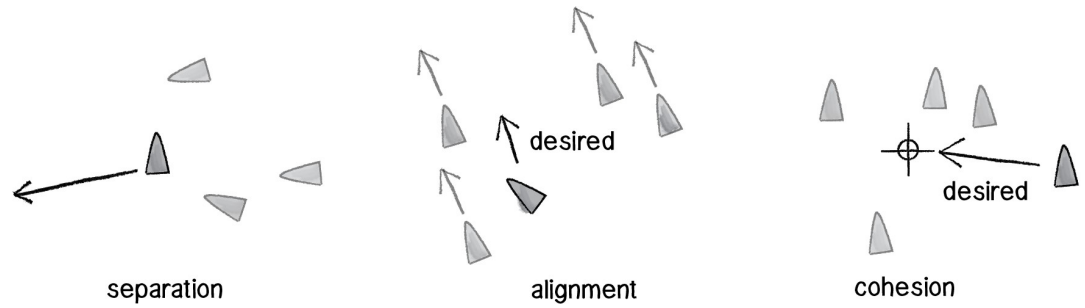


Illustrazione 8: Le regole che definiscono il comportamento delle api (The Nature Of Code)

In base a queste regole ogni ape calcola la sua accelerazione e direzione, creando così l'effetto sciame.

Il programma è suddiviso in quattro file .pde che contengono:

- il file Api la logica fondamentale del programma: le variabili globali, il metodo setup e il metodo draw.
- Il file Boids la definizione della classe che regola il comportamento delle singole api.
- Il file Flock la definizione della classe che gestisce l'array list dei veicoli (api)
- il file Debug la funzione di debug

5.3 Realizzazione della grafica

L'unico elemento grafico presente sulla scena è il corpo delle api. Questo è stato realizzato in grafica vettoriale partendo da un modello di ape reale e riducendolo a forme geometriche semplici (illustrazione 9).

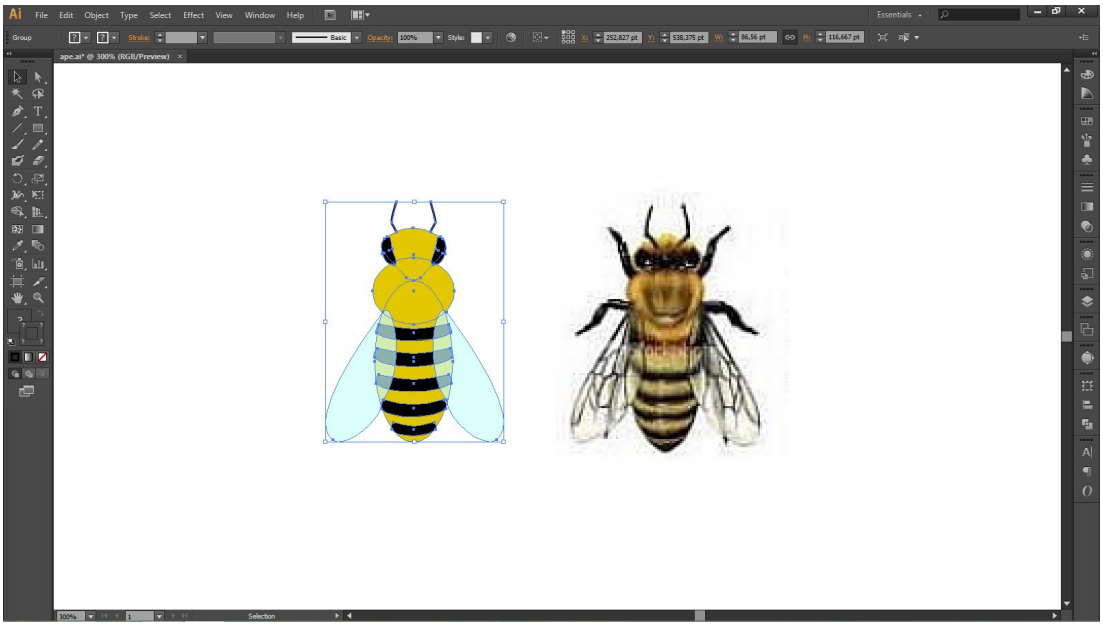


Illustrazione 9: Stilizzazione e riduzione a figure geometriche semplici della figura di un ape

6 Conclusioni

Attualmente le due installazioni sono in fase di preparazione **hardware** per la presentazione al pubblico.

In particolare “Melissa” sarà utilizzata attivamente dall'ONG Greenpeace nel contesto della campagna mondiale “Salviamo le Api”.

7 Bibliografia

Andrea Balzola, Paolo Rosa: *L'arte fuori di sé - Un manifesto per l'età post-tecnologica.* Feltrinelli, 2011.

Daniel Shiffman: *The nature of code: Simulating Natural Systems with Processing.* 2012

Greenpeace Research Laboratories: *Bees in Decline: a review of factors that put pollinators and agriculture in Europe at risk.* 2013

Hsu-Huei Wu, Andrew Bainbridge-Smith*: Advantages of using a Kinect Camera in various applications.* Electrical & computer Engineering, University of Canterbury, New Zealand.

Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, Andrew Blake: *Real-Time Human Pose Recognition in Parts from Single Depth Images.* Microsoft Research Cambridge & Xbox Incubation, 2011.

MobLab Team: *Want more Facebook activity? Surprise them and make 'em happy - Greenpeace conducts research to identify psychological, emotional influencers driving sharing levels.* http://www.mobilisationlab.org/want-more-facebook-activity-surprise-them-and-make-em-happy/ (visitato 25 giugno 2013)

Studio Azzurro: *Ambienti sensibili. Esperienza tra interattività e narrazione.* Cirifino Fabio, Rosa Paolo, Roveda Stefano, Sangiorgi Leonardo. Electa, Milano 1999.

Wikipedia, voce *Interactive Art.* http://en.wikipedia.org/wiki/Interactive_art (visitato il 20 luglio 2013)

Wikipedia, voce *Generative Art.* http://en.wikipedia.org/wiki/Generative_art (visitato il 20 luglio 2013)

8. Appendice

8.1 Abbreviazioni

Software development kit	Sdk
Natural user interface	Nui
Visual Studio	VS
Bidimensionale	2D
Tridimensionale	3D

8.2 Codice sorgente “Il paese degli uomini giusti”

File Letter.pde

```
import ddf.minim.*;
import de.looksgood.ani.*;
import org.apache.batik.svggen.font.table.*;
import org.apache.batik.svggen.font.*;
import processing.video.*;
import SimpleOpenNI.*; // kinect
import java.util.*;

ArrayList<tChar> letters = new ArrayList<tChar>();
int letCont = 700;
ArrayList dPoes = new ArrayList();
Movie myMovie;
SimpleOpenNI context;
boolean autoCalib = true;
boolean state;
boolean gesture = false;
int a = 255;
boolean kinect;
poesia p;
XML uno;
PImage movie, user;
PFont pf;
SceneMapper sceneMap;
Minim minim;
AudioPlayer song;

void setup() {
  size(1280, 720);
  frameRate(30);
  context = new SimpleOpenNI(this,
SimpleOpenNI.RUN_MODE_MULTI_THREADED);
```

```
  if (!context.enableDepth())
  {
    println("Kinect not connected!");
    kinect=false;
  }
  else
  {
    kinect=true;
    context.enableScene();
    context.setMirror(true);
    context.enableUser(SimpleOpenNI.SKEL_PROFILE_ALL);
    sceneMap = new SceneMapper(context);
  }
  // Inizializzo il gestore dell'audio
  minim = new Minim(this);
  song = minim.loadFile("We_re_Saving_Up_For_Sundays.mp3");
  // Inizializzo gestore animazioni
  Ani.init(this);
  int j = 0;
  for (int i = 0; i<letCont; i++)
  {
    if ((i % 50)==0)
    {
      j++;
    }
    tChar t = new tChar(j);
    letters.add(t);
  }
  // carico i file esterni
  pf = loadFont("LiberationSerif-Bold-50.vlw");
  uno = loadXML("poesia.xml");
  // inizializzo l'oggetto poesia
  p = new poesia(uno);
  // inizializzo i filmati
  myMovie = new Movie(this, "fire.mp4");
  myMovie.stop();
  textFont(pf);
}

void draw()
{
  state = true;
  for (int i = 0; i<(letCont-200); i++)
    {
      tChar l = letters.get(i);
      state = (state && l.stp);
    }
  background(0);
  ///_________________Kinect_____________________
  if (kinect)
  {
    context.update();
    stroke(0,0,255);
```

```
    strokeWeight(3);
    pushMatrix();
    translate((width/2)-320, height-480);
    sceneMap.update();
    image(sceneMap.scene,0,0);
    int[] userList = context.getUsers();
    for(int i=0;i<userList.length;i++)
    {
      if(context.isTrackingSkeleton(userList[i]))
      {
        drawSkeleton(userList[i]);
      }
    }
    popMatrix();
  }
  ///_________Gestione stato Programma__________
  if(state)
  {
    //Gestione video
    myMovie.play();
    a=255;
    if (myMovie.available())
    {
    myMovie.read();
    }
    if (!(myMovie.time()<28))
    {
      myMovie.stop();
      letters.clear();
      int j = 0;
      for (int i = 0; i<letCont; i++)
      {
        if ((i % 50)==0)
        {
          j++;
        }
        tChar t = new tChar(j);
        letters.add(t);
      }
    }
    //Gestione immagine Kinect + immagine video
    if (kinect)
    {
      movie = myMovie;
      user = sceneMap.scene;
      movie.blend(user,0,0,height,width,(width/2)-320,height-
480,640,480, LIGHTEST);
      image(movie,0,0);
    }
    else
    {
      image(myMovie,0,0);
    }
```

```
  }
  ///____________________Disegna il resto___________
  fill(0);
  stroke(0);
  pushMatrix();
  textSize(12);
  doDebug();
  popMatrix();
  for (int i = 0; i<letCont; i++)
    {
      tChar l = letters.get(i);
      if ((state)||(gesture && (l.x >300) && (l.x < width-300)
&& (l.y < 100)))
      {
        l.opacity--;
      }
      l.display();
    }
  if((!(dPoes.size()==0))&&(gesture))
      {
      for (int i = 0; i<dPoes.size(); i++)
        {
          AniChar c = (AniChar) dPoes.get(i);
          c.display();
        }
      }
  if(gesture)
  {
    //Gestione Audio
    if (!song.isPlaying())
    {
      song.play();
    }
    p.draw();
  }
  else
  {
    if(song.isPlaying())
    {
    song.pause();
    }
  }
}


void stop()
{
  // always close Minim audio classes when you are finished
with them
  song.close();
  // always stop Minim before exiting
  minim.stop();
```

```
  // this closes the sketch
  super.stop();
}
```

File AniChar.pde

```
class AniChar{
char letter;

  float targety,startx;
  float x;
  float y;
  float alpha;
  Ani yAni;
  Ani xAni;

  AniChar(float x_, float y_, float targety_, char letter_) {
    startx = x = x_;
    targety = targety_;
    y = y_;
    letter = letter_;
    alpha=255;
    xAni = new Ani(this, random(2,4), "x", x+10,
Ani.QUAD_IN_OUT);
    xAni.setPlayMode(Ani.YOYO);
    xAni.repeat(2);
    yAni = new Ani(this, 10.0, random(0,2), "y", targety,
Ani.LINEAR);
    yAni.start();
    Ani.to(this, 5.0, 15.0,"alpha",0);
  }

  void display() {
    pushMatrix();
    fill(255,alpha);
    textSize(50);
    textFont(pf);
    textAlign(LEFT);
    text(letter,x,y);
    popMatrix();
  }

}
```

file function.pde

```
void doDebug(){
  // Debug
  String [] debug = new String[2];
  String [] time = new String[2];
```

```
  debug[0] = "Frame Rate =";
  debug[1] = str(frameRate);
  time[0] = "Time =";
  time[1] = str(millis());
  String txt = join(debug, " ");
  String txt_2 = join(time, " ");
  fill(255);
  textAlign(LEFT);
  text(txt, 10, 30);
  text(txt_2, 10, 70);
  if(!kinect)
  {
    text("Kinect disable", 10, 50);
  }
  else
  {
    text("Kinect enable", 10, 50);
    if (gesture)
    {
      text("Gesture = true", 10, 90);
    }
    else
    {
      text("Gesture = false", 10, 90);
    }
  }
  //fine Debug
}


void drawSkeleton(int userId)
{
  PVector r_hand = new PVector();
  PVector l_hand = new PVector();
  PVector head = new PVector();
  context.getJointPositionSkeleton(userId,
SimpleOpenNI.SKEL_RIGHT_HAND, r_hand);
  context.getJointPositionSkeleton(userId,
SimpleOpenNI.SKEL_LEFT_HAND, l_hand);
  context.getJointPositionSkeleton(userId,
SimpleOpenNI.SKEL_NECK, head);
  if ((r_hand.y > (head.y-60))&&(l_hand.y > (head.y-60)))
  {
    gesture = true;
  }
  else
  {
    gesture = false;
  }
}
```

file kinect_events.pde

```
//
-----------------------------------------------------------------
---
// SimpleOpenNI events

void onNewUser(int userId)
{
  println("onNewUser - userId: " + userId);
  println("  start pose detection");

  if(autoCalib){
    context.requestCalibrationSkeleton(userId,true);}
  else{
    context.startPoseDetection("Psi",userId);}
}

void onLostUser(int userId)
{
  println("onLostUser - userId: " + userId);
}

void onExitUser(int userId)
{
  println("onExitUser - userId: " + userId);
}

void onReEnterUser(int userId)
{
  println("onReEnterUser - userId: " + userId);
  drawSkeleton(userId);
}

void onStartCalibration(int userId)
{
  println("onStartCalibration - userId: " + userId);
}

void onEndCalibration(int userId, boolean successfull)
{
  println("onEndCalibration - userId: " + userId + ",
successfull: " + successfull);

  if (successfull)
  {
    println("  User calibrated !!!");
    context.startTrackingSkeleton(userId);
  }
  else
  {
    println("  Failed to calibrate user !!!");
    println("  Start pose detection");
```

```
    context.startPoseDetection("Psi",userId);
  }
}

void onStartPose(String pose,int userId)
{
  println("onStartPose - userId: " + userId + ", pose: " +
pose);
  println(" stop pose detection");

  context.stopPoseDetection(userId);
  context.requestCalibrationSkeleton(userId, true);

}

void onEndPose(String pose,int userId)
{
  println("onEndPose - userId: " + userId + ", pose: " +
pose);
}
```

Il file Tchar

```
class tChar{


  char cr;
  float x;
  float y=-300;
  int size;
  int opacity;
  int place;
  Ani xAni;
  Ani yAni;
  boolean stp = false;


  tChar(int space)
  {
    cr=(char)int(random(33,126));
    x = random(0, width);
    place = int(random(sizes.length));
    size = int(sizes[place]);
    opacity = int(random(125,255));
    xAni = new Ani(this, random(2,4), "x", x+10,
Ani.QUAD_IN_OUT);
```

```
    xAni.setPlayMode(Ani.YOYO);
    xAni.repeat();
    yAni = new Ani(this, random(25,40),(random(0,1))
+(2*space), "y", (height+50)-(30*space), Ani.LINEAR,
"onEnd:stop");
    yAni.start();
  }

  void display()
  {
    pushMatrix();
    fill(255,opacity);
    textFont(pf[place]);
    textSize(size);
    text(cr,x,y);
    popMatrix();
  }

  void stop()
  {
    xAni.pause();
    stp= true;
  }
}
```

Il file SceneMapper

```
class SceneMapper{
  PImage scene;
  int[] sceneMap;
  int numPixels;
  color bg;//background colour
  color user = color(255);

  SceneMapper(SimpleOpenNI context){
    numPixels = context.userWidth()*context.userHeight();
    sceneMap = new int[numPixels];
    scene = createImage( context.userWidth(),
context.userHeight(), RGB );
```

```
    scene.loadPixels();
    bg = color(0,0);
  }
  void update(){
    context.userMap(sceneMap);
    Arrays.fill(scene.pixels, bg);
    for(int i = 0 ; i < numPixels ; i++){
        if(sceneMap[i] > 0) scene.pixels[i] = user;
    }
    scene.updatePixels();
    //fastblur(scene, 3);
  }
}
```

Il file Poesia

```
class poesia
{
  XML poesia;
  XML[] strofe;
  XML[] versi;
  int numS = 0;
  float pX = 300;
  float pY;
  timer time;
  int rep = 1;
  String v;
  float normalizeY;

  poesia(XML xml)
  {
  poesia = xml;
  strofe = poesia.getChildren("strofa");
  time = new timer(20000.0);
  }

  void draw()
  {
```

```
time.step();
textSize(50);
textAlign(CENTER);
//text(str(time.ripetizioni), 300, 50);
if (numS >= strofe.length)
{
  numS = 0;
}
versi = strofe[numS].getChildren("verso");
pY = -100+(-70*versi.length);
if (rep<time.ripetizioni)
{
  rep= time.ripetizioni;
  numS++;
  pX = 300;
  dPoes.clear();
  for (int i = 0; i < versi.length; i++)
  {
    v = versi[i].getContent();
    normalizeY= 0;
    for (int c = 0; c < v.length(); c++)
    {
      normalizeY += textWidth(v.charAt(c));
    }
    normalizeY= (width/2) - (normalizeY/2);
    pX=normalizeY;
    for (int j = 0; j < v.length(); j++)
    {
      char cr = v.charAt(j);
      AniChar l = new AniChar(pX, pY, (480 + pY), cr);
      dPoes.add(l);
      pX=pX+textWidth(v.charAt(j));
    }
    pY=pY+70;
    pX=0;
}
```

```
    }
  }
}
```

8.3 Codice sorgente “Melissa”

```
import geomerative.*;
import SimpleOpenNI.*; // kinect

Flock flock;
RFont font;
RShape cuore;
RShape qrc;

SimpleOpenNI context;
boolean autoCalib = true;
boolean kinect;
boolean someone = false;

int fontSize = 100;
PShape api;
PVector target = new PVector();

String inp = "", lastText;
String zin = "un terzo:della frutta:che mangi:esiste:grazie a
noi:AIUTACI:SOS-BEES.org";
String[] words = split(zin, ":");
float currTime = 0;
int currWord = 0;

boolean start = false;
boolean msg = false;

void setup() {
  size(1024,400, P2D);
  frameRate(120);
  smooth();
  context = new SimpleOpenNI(this,
SimpleOpenNI.RUN_MODE_MULTI_THREADED);
  if (!context.enableDepth())
  {
    println("Kinect not connected!");
    kinect=false;
  }
  else
  {
    kinect=true;
    context.enableDepth();
    context.setMirror(true);
    context.enableUser();
  }
  api=loadShape("ape.svg");
  RG.init(this);
```

```
  font = new RFont("1CamBam_Stick_2.ttf", fontSize,
RFont.LEFT);
  cuore = RG.loadShape("cuore.svg");
  flock = new Flock();
  for (int i = 0; i < 250; i++) {
    Boid b = new Boid(width/2,height/2);
    flock.addBoid(b);
  }
  smooth();
}

void draw() {
  background(255,255,220);
  if (start && msg && currTime == 0)
  {
    currTime = frameCount;//millis();
  }
  if (start && frameCount > currTime+390)//millis() >
currTime+13000.0)
  {
    currTime= frameCount;//millis();
    currWord++;
  }
  if (!(currWord < words.length)||!msg)
  {
     currTime = 0;
     currWord = 0;
  }
  RGroup grp = font.toGroup(words[currWord]);
  if (kinect)
  {
    context.update();
    int[] userList = context.getUsers();
    if (userList.length >= 1)
    {
      if(context.isTrackingSkeleton(userList[0]))
      {
        someone = true;
        context.getJointPositionSkeleton(userList[0],
SimpleOpenNI.SKEL_NECK, target);
        println(target);
      }
      grp.translate(target.x+100, target.y+60);
      cuore.centerIn(g);
      cuore.scale(0.3);
      cuore.translate(target.x+300, target.y);
    }
    else
    {
      someone = false;
      grp.translate(mouseX+100, mouseY+60);
      cuore.centerIn(g);
      cuore.scale(0.3);
```

```
      cuore.translate(mouseX+300, mouseY);
    }
  }
  else
  {
    grp.translate(mouseX+100, mouseY+60);
    cuore.centerIn(g);
    cuore.scale(0.3);
    cuore.translate(mouseX+300, mouseY);
  }
  RG.setPolygonizer(RG.UNIFORMLENGTH);
  RG.setPolygonizerLength(20);
  RPoint[] rp = grp.getPoints();
  RPoint[] cu = cuore.getPoints();
  if (msg)
  {
    flock.run(rp);
  }
  else
  {
    flock.run(cu);
  }
  //doDebug();
  fill(0);
  //saveFrame("output/frames#####.png");
}

void mouseDragged() {
  flock.addBoid(new Boid(mouseX,mouseY));
}

void keyPressed() {
  if (key == CODED)
  {
    if (keyCode == UP)
    {
      start = true;
    }
    else if (keyCode == DOWN)
    {
      start = false;
    }
      if (keyCode == LEFT)
    {
      msg = true;
    }
    else if (keyCode == RIGHT)
    {
      msg = false;
    }
  }
}
```

```
class Boid {

  PVector location;
  PVector velocity;
  PVector acceleration;
  float r;
  float maxforce;
  float maxspeed;

  Boid(float x, float y) {
    acceleration = new PVector(0,0);
    velocity = new PVector(random(-1,1),random(-1,1));
    location = new PVector(x,y);
    r = 10.0;
    maxspeed = 3.5;
    maxforce = 0.05;
  }

  void run(ArrayList<Boid> boids) {
    flock(boids);
    update();
    borders();
    render();
  }

  void applyForce(PVector force) {
    acceleration.add(force);
  }

  void flock(ArrayList<Boid> boids) {
      PVector sep = separate(boids);   // Separation
      //PVector ali = align(boids);      // Alignment
      PVector coh = cohesion(boids);   // Cohesion
      if (someone)
      {
      PVector sk = seek(/*new
PVector(mouseX,mouseY)*/target,false);
      applyForce(sk);
      sep.mult(2);
      }
      else
      {
        PVector sk = seek(new PVector(mouseX,mouseY),false);
        applyForce(sk);
        sep.mult(2);
//        PVector ali = align(boids);
//        ali.mult(1);
//        applyForce(ali);
//        sep.mult(1.5);
      }
      coh.mult(0.5);
      applyForce(sep);
      applyForce(coh);
```

```
  }


  void update() {
    velocity.add(acceleration);
    velocity.limit(maxspeed);
    location.add(velocity);
    acceleration.mult(0);
  }

  PVector seek(PVector target, boolean slowdown)
  {
    PVector desired = PVector.sub(target,location);
    float d = desired.mag();
    desired.normalize();
    if ((slowdown) && (d < 100.0f))
desired.mult(maxspeed*(d/100.0f));
    else desired.mult(maxspeed);
    PVector steer = PVector.sub(desired,velocity);
    steer.limit(maxforce);
  }

  void render() {
    float theta = velocity.heading2D() + radians(90);
    fill(175);
    stroke(0);
    pushMatrix();
    translate(location.x,location.y);
    rotate(theta);
    scale(0.15);
    shapeMode(CENTER);
    shape(api,0,0);
    popMatrix();
  }

  // Wraparound
  void borders() {
    if (location.x < -r) location.x = width+r;
    if (location.y < -r) location.y = height+r;
    if (location.x > width+r) location.x = -r;
    if (location.y > height+r) location.y = -r;
  }

  PVector separate (ArrayList<Boid> boids) {
    float desiredseparation = 25.0f;
    PVector steer = new PVector(0,0,0);
    int count = 0;
    for (Boid other : boids) {
      float d = PVector.dist(location,other.location);
      if ((d > 0) && (d < desiredseparation)) {
        PVector diff = PVector.sub(location,other.location);
        diff.normalize();
        diff.div(d);
```

```
      steer.add(diff);
      count++;
    }
  }
  if (count > 0) {
    steer.div((float)count);
  if (steer.mag() > 0) {
    steer.normalize();
    steer.mult(maxspeed);
    steer.sub(velocity);
    steer.limit(maxforce);
  }
  return steer;
}

PVector align (ArrayList<Boid> boids) {
  float neighbordist = 50;
  PVector sum = new PVector(0,0);
  int count = 0;
  for (Boid other : boids) {
    float d = PVector.dist(location,other.location);
    if ((d > 0) && (d < neighbordist)) {
      sum.add(other.velocity);
      count++;
    }
  }
  if (count > 0) {
    sum.div((float)count);
    sum.normalize();
    sum.mult(maxspeed);
    PVector steer = PVector.sub(sum,velocity);
    steer.limit(maxforce);
    return steer;
  } else {
    return new PVector(0,0);
  }
}
PVector cohesion (ArrayList<Boid> boids) {
  float neighbordist = 50;
  PVector sum = new PVector(0,0);
  int count = 0;
  for (Boid other : boids) {
    float d = PVector.dist(location,other.location);
    if ((d > 0) && (d < neighbordist)) {
      sum.add(other.location);
      count++;
    }
  }
  if (count > 0) {
    sum.div(count);
    return seek(sum, false);
  } else {
    return new PVector(0,0);
```

```
    }
  }
}
class Flock {
  ArrayList<Boid> boids;
  Flock() {
    boids = new ArrayList<Boid>();
  }

  void run(RPoint[] rpa) {
    for (int j=0; j<boids.size() ; j++)
    {
      Boid b = boids.get(j);
      if (!start)
      {
      b.run(boids);
      }
      else
      {
      if (j<rpa.length)
      {
        PVector sk = b.seek(new
PVector(rpa[j].x,rpa[j].y),true);
        b.applyForce(sk);
        b.update();
        b.render();
      }
      else
      {
        b.run(boids);
      }
      }
    }
  }

  void addBoid(Boid b) {
    boids.add(b);
  }
}
```