{Paweł Janicki}

s [taged] Body

wprowadzenie do pracy
z mediami digitalnymi
w kontekście scenicznym

Spis treści

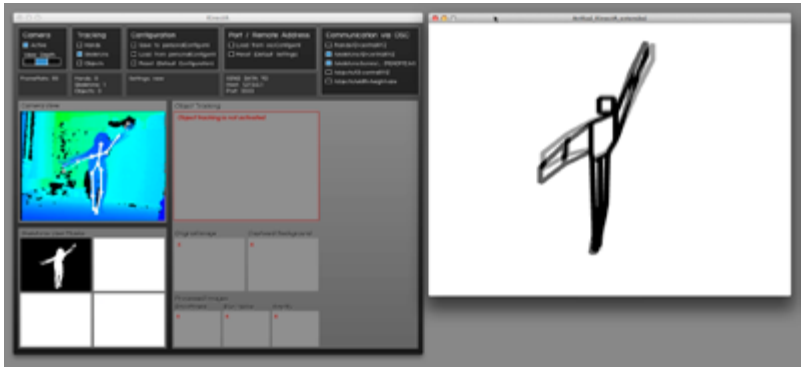
00

wprowadzenie

s[taged]Body czyli ciało – w domyśle performera, tancerza, aktora, uczestnika – na scenie, stające się częścią sceny i jej infrastruktury.

Temat *stricte* medialabowy, bo czynniki takie jak uczestnictwo, technologia i interfejsy pomiędzy ciałem, a urządzeniami digitalnymi od początku znajdowały się w przestrzeni zainteresowań prowadzących warsztaty i ich uczestników podczas kolejnych edycji MediaLabu. Jednak dopiero w 2013 roku (podczas 4 edycji) poruszyliśmy ten temat niejako wprost – o ile kompetencje w posługiwaniu się współczesnym – digitalnym – instrumentarium sztuki stosunkowo dobrze przyjęły się już w muzyce, aktywnościach nowo-medialnych, czy filmie, o tyle bardzo szeroko rozumiana „scena" i środowiska około-teatralne adoptują nowe metody działań nieco wolniej – nic w tym dziwnego, ponieważ w tym konkretnym przypadku metody pracy związane z nowymi technologiami wchodzą w dialog z silnymi tradycjami zupełnie innego rodzaju.

W potocznej opinii osoby pracujące „z ciałem": performerzy, tancerze, aktorzy, stanowią grupę zasadniczo przeciwstawną w stosunku do „nerdów" pracujących z nowymi mediami–niewątpliwie pewne środowiskowe różnice występują, niemniej etosy: medialabowy i sceniczny dają się ze sobą pogodzić, ponadto, gdyby spojrzeć na digitalną rewolucję z perspektywy – również historycznej – nieco szerszej, niż zwykle, to można dostrzec między innymi, że sam termin *digital* pochodzi z łacińskiego *digitus*, czyli „palec", a procesy obliczeniowe były pierwotnie bezpośrednim skanowaniem parametrów i możliwości ciała (np. liczenie na palcach). Tak więc pewien wspólny dla obu aktywności rdzeń istnieje.

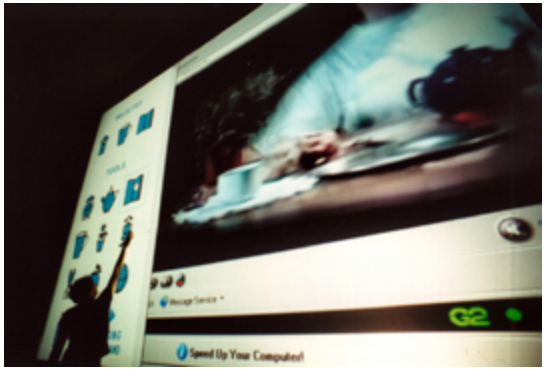


aplikacja KinectA, protokół komunikacyjny OSC i środowisko programistyczne Processing – prosta metoda na wykorzystanie systemu detekcji szkieletu

kurs Artkod: Processing
http://www.nina.gov.pl/kultura-2_0/sam-to-zrób/kurs-processingu

Poniższy mini-podręcznik jest tak napisany, by raczej umożliwiać płynne i bezbolesne wejście w świat medialno-scenicznych hybryd, niż z definicji stawiać przed czytelnikami przesadnie skomplikowane zadania, raczej zniechęcające do dalszej pracy, niż zachęcające do samodzielnego odkrywania nowych rozwiązań na bazie opartych na praktyce wskazówek osób, które „wystartowały wcześniej". Podręcznik nie stawia też czytelnika wobec konieczności zrozumienia całości technologii omawianych w poszczególnych rozdziałach. Celem podręcznika nie jest też eksploracja konkretnego zestawu technologii, ale próba skupienia się na budowie strategii interaktywnych, zachęcie do pracy zbiorowej, protokołach komunikacyjnych i łatwych do zastosowania sensorach. Niemniej skonfrontowanie czytelników z ograniczeniami technologii, którymi potrafią operować skłaniać będzie – przynajmniej potencjalnie – do rozwoju na polu warsztatowym.

Poruszane tematy zostały ponadto tak uszeregowane, by zagadnienia najłatwiejsze znalazły się na samym początku – jeśli np. rozpoczniesz lekturę rozdziału poświęconego wykorzystaniu kamery jako sensora, to przekonasz się, że najprostsze techniki pracy nie tylko nie wymagają samodzielnego programowania, ale nawet użycia komputera. Naturalnie chcąc popracować nad bardziej skomplikowanymi projektami i technologiami pewne doświadczenie w programowaniu – lub chęć zdobycia takiego doświadczenia, choćby w minimalnym zakresie – może się bardzo przydać. Jest to o tyle proste, że w większości przypadków wystarczy nam programowanie wizualne, choćby w Pure-Data [http://puredata.info] lub MaxMSP/Jitter [http://cycling74.com].



„Performance on Demand”
(Media Art Biennale WRO 2001) –
performatywne działanie łączące media
strumieniowe internet

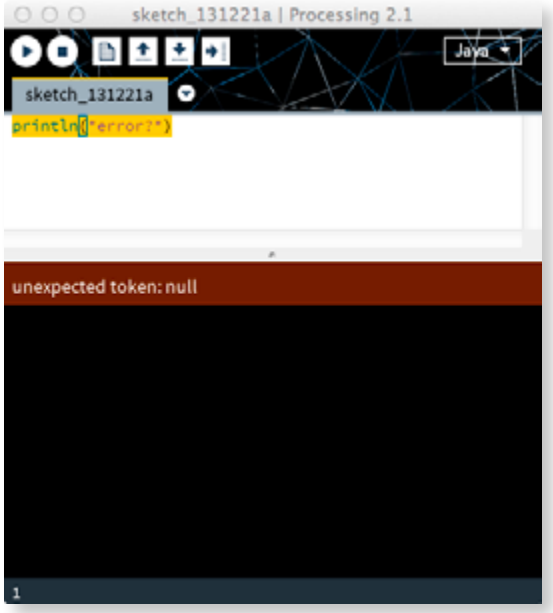
serwis prasowy WRO Art Center

Inspiracją do powstania tego mini-podręcznika były przede wszystkim medialno-ruchowe warsztaty, które przeprowadziłem wspólnie z Ireną Lipińską podczas opolskiej edycji MediaLabu (jesień 2013). Początkowo podręcznik miał przybrać bardziej „techniczną” formę, przypominającą moją wcześniejszą książkę o medialabowym rodowodzie, czyli podręcznik do środowisk MaxMSP/Jitter [http://cycling74.com] i Pure-Data [http://puredata.info], jednak obserwacja tego, w jaki sposób warsztaty przebiegały i jakiego rodzaju wynikające z nich doświadczenia okazały się najważniejsze uznałem, że (po pierwsze) dotkliwy do niedawna brak polskojęzycznej literatury objaśniającej korzystanie ze środowisk programistycznych i aplikacji związanych z programowaniem na potrzeby zastosowań artystycznych powoli przestaje być tak dotkliwy, jak wcześniej oraz (po drugie) ciągle brakuje tekstów, które ulokowane są na pograniczu technologii i koncepcyjnych porad „dla początkujących” – i taki właśnie tekst oddaję w Wasze ręce.

Paweł Janicki

01

błędy w myśleniu



Jakie błędy zwykle popełniamy
i jak ich uniknąć

W kolejnych rozdziałach zamierzam zamieścić wskazówki, które – mam nadzieję – ułatwią w miarę bezbolesne rozpoczęcie samodzielnych eksperymentów w zakresie omawianym w podręczniku, jednak najpierw warto będzie rozprawić się z kilkoma błędnymi poglądami, z jakimi często można się zetknąć pracując nad projektami łączącymi media elektroniczne i scenę (jestem właściwie pewien, że przynajmniej jeden z tych poglądów czytelnicy rozpoznają, jako własny). Opisane poniżej uprzedzenia często blokują nasz kreatywny potencjał, dlatego dobrze będzie – zanim rozpoczniemy praktyczne eksperymenty – przyjrzeć się temu, w jaki sposób myślimy o temacie, którym zamierzamy się zająć. A oto lista typowych przesądów:

1. **Aby tworzyć projekty łączące media i scenę trzeba dysponować wielkim budżetem na sprzęt i specjalistów. Nie można takich projektów realizować w warunkach „niskobudżetowych".**

Nieprawda! Wbrew pozorom, sporo technologicznych komponentów oraz potrzebnej do ich zrozumienia wiedzy jest dostępne przy stosunkowo niewielkich nakładach finansowych lub nawet i bezkładowo (głównie takimi zajmiemy się w tym podręczniku). Komercyjne systemy interaktywne lub kosztowne typy sensorów mogą być zwyczajnie nieprzydatne na początkowym etapie pracy, zanim nie będziemy dokładnie wiedzieć, czego od nich oczekujemy. Podobnie ma się rzecz z – nieraz równie kosztownymi – specjalistami: kiedy dopiero rozpoczynamy eksperymenty we włączaniu mediów elektronicznych w formy sceniczne łatwiej nam będzie podążać za bieżącymi rezultatami naszych działań, niż trzymać się scenariusza powstałego z wyprzedzeniem (a takiego scenariusza będzie z pewnością wymagał specjalista z doświadczeniem, już choćby po to, by wiedzieć czego się od niego oczekuje i jak wiele czasu mu to zajmie). Nie chcę przez to powiedzieć, że budżet i specjalistyczna wiedza się nie liczą – warto jednak nie czekać na wymarzone warunki do pracy (wyposażone studio i zgraję techników czekających na nasze polecenia). Lepiej po prostu zabrać się do pracy w warunkach, jakie jesteśmy w stanie sobie zorganizować – zwłaszcza nie mając żadnego doświadczenia w podobnych aktywnościach przekonamy się szybko, że nasze oczekiwania odnośnie optymalnych warunków pracy mogły być zwyczajnie chybione.

2. **Istnieją specjaliści, którzy wiedzą, jak powinien wyglądać projekt łączący media elektroniczne i scenę – tacy specjaliści będą w stanie zrealizować moją artystyczną wizję. Ja muszę jedynie wyjaśnić im, czego oczekuję.**

Na własny użytek często określam tego typu myślenie jako „poszukiwanie figury mitycznego informatyka". Oczywiście jak najbardziej sensowne może być skorzystanie z umiejętności kogoś, kto np. programuje lepiej od nas, jednak nie oczekujmy od tego kogoś, że zrealizuje za nas naszą wizję (ponadto najprawdopodobniej nie będziemy w stanie przekazać takiej osobie natury problemu, jaki zamierzamy rozwiązać, jeśli nie będziemy dysponować zasobem wspólnych pojęć). Taka osoba może pomóc nam rozwiązać konkretny problem (np. filtrowanie danych z kamery lub napisanie programu konwertującego komunikaty MIDI do OSC – nawiasem mówiąc, protokołami komunikacyjnymi MIDI i OSC zajmiemy się w późniejszych rozdziałach nieco szerzej) – jednak są to zadania stricte techniczne, nie związane z ideą i jakością projektu rozumianego jako

całość. Choreograf nie musi tańczyć (choć na ogół potrafi), ale musi umieć w zrozumiałych dla tancerza pojęciach przekazać mu ideę ruchu, nad którym pracują. Reżyser filmowy, nawet jeśli sam nie potrafi poradzić sobie z obsługą kamery, musi potrafić przekazać operatorowi (lub reżyserowi obrazu) wskazówki dotyczące tego, jak ma wyglądać kręcone ujęcie. Nawet jeśli z jakiegoś powodu uważamy, że taniec lub „obsługa kamery" to niekreatywne zajęcia czysto techniczne (czy ktoś rzeczywiście tak uważa?) musimy mieć o nich na tyle wyrobione pojęcie, by móc się komunikować z ludźmi zajmującymi się takimi aktywnościami – to natomiast wymaga wcześniej nabrania własnego, praktycznego, doświadczenia w odpowiedniej dziedzinie.

3. **Ludzie sceny nie potrafią sobie radzić z projektami medialnymi | Ludzie związani z mediami elektronicznymi nie potrafią zrozumieć sceny. Lub: ludzie sceny to mięso | Ludzie związani z mediami elektronicznymi to technicy pozbawieni wyobraźni.**

Jeśli nie uda nam się sprawić by ludzie zaangażowani w projekt i odpowiedzialni za medialny i sceniczny segment współpracowali i słuchali nawzajem swoich pomysłów, to może się okazać, że zwyczajnie tracimy sporo dobrego koncepcyjnego materiału. Nawet jeśli działamy jednoosobowo, to musimy dogadać się ze sobą samym…

4. **Projekt łączący media elektroniczne i scenę składa się z dwu niezależnych i łączonych dopiero finalnie komponentów związanych odpowiednio ze sceną (choreografią, dramaturgią, itp.) oraz technologią.**

Starajmy się raczej jak najszybciej połączyć oba komponenty. Nawet, jeśli później okaże się, że bieżąca praca nad technologicznymi lub scenicznymi aspektami projektu wymusza osobną pracę nad nimi, to doświadczenie uzyskane na bazie pierwszych wspólnych działań będzie procentowało. Jeśli założymy, że niezależnie przygotowywane komponenty będziemy łączyć dopiero na koniec, może się okazać że finalnie wyjdzie nam twór, przy którym monstrum sprepa-rowane przez hrabiego Frankensteina wyda się raczej nadobne.

5. **Etos DIY (Do It Yourself) jest lepszy niż wykorzystanie gotowych rozwiązań | Należy korzystać z gotowych „profesjonalnych" rozwiązań.**

Jeśli czytasz ten podręcznik, najprawdopodobniej bliższa jest Ci mentalność „medialabowa", która kładzie nacisk na samodzielną pracę nad jak największą ilością elementów projektu. Niemniej czasem warto pokusić się o skorzystanie z czegoś, co w jakimś stopniu przynależy do świata obiektów ready-made. Może nam to nieraz oszczędzić wiele czasu – nie mówię tu o kosztownych systemach interakcji dostępnych na zamówienie (przez cały czas zakładamy, że nasz podręcznik jest przeznaczony dla początkujących), ale o wykorzystaniu przydatnych drobiazgów, które pozwolą nam skierować własną energię na najważniejsze elementy projektu. Przykładowo, przemysł rozrywkowy dostarcza nam sporo gadżetów nadających się do wykorzystania w projektach łączących media elektroniczne i scenę i czasem może się okazać, że w nietypowy sposób zastosowany relatywnie tani i trwały kontroler wiimote dysponujący m.in. możliwością bezprzewodowego przesyłania danych o swoim położeniu i dynamice ruchu pozwoli nam szybko zbudować sensowną

i trwałą pod względem fizycznym bazę sensorów dla performera – „szybko", oznacza w tym wypadku dużo szybciej, niż np. lutując samodzielnie potrzebne komponenty.

6. Rzecz sprowadza się do oprogramowania i sprzętu.

Nie lekceważmy „fizycznej" strony zagadnienia i prób w warunkach scenicznych – zwykle dopiero w warunkach „bojowych" okazuje się, że technologia działa inaczej, niż się spodziewaliśmy. Im wcześniej zmierzymy się z testami w docelowej lokalizacji projektu, tym lepiej. Rozstawiając na scenie np. dopieszczony w domowym zaciszu system śledzenia ruchu oparty o kamerę na podczerwień możemy się przekonać, że w pomieszczeniu, w którym się znajdujemy pojawiają się niespodziewanie źródła podczerwieni (np. bliki światła słonecznego lub aktywne sensory antywłamaniowe…), których się zupełnie nie spodziewaliśmy, wpięcie projektora wideo do komputera powoduje pojawienia się niechcianych „brumów" w systemie dźwiękowym, o za krótkich kablach i braku odpowiednich „przejściówek" nawet nie wspominając. Do tego dochodzą problemy trudniej wytłumaczalne, co nie znaczy, że łatwiejsze do obejścia, np. stabilny jak skała komputer po przeniesieniu w nowe miejsce zawiesza się regularnie, kamera generuje obraz z podejrzanymi szumami, itp.

7. Programowanie to zajęcie wyłącznie techniczne, nie związane z merytoryczną częścią projektu.

Częsty i bardzo poważny błąd. Kod to współczesna lingua franca. Jeśli szukamy odniesień i inspiracji nie wynikających linearnie z któ-

rejś z tradycji scenicznych, to najprawdopodobniej znajdziemy je właśnie w programowaniu. Osoby zajmujące się programowaniem rzadko wykonują swoją pracę w sposób wyłącznie mechaniczny: większość programistów posiada własny styl pisania kodu widoczny choćby w indywidualnych metodach rozwiązywania typowych problemów. Innymi słowy: programista może być kreatywną jednostką, dysponującą potencjałem, który szkoda byłoby zmarnować.

8. Istnieje pewien najlepszy system operacyjny lub program umożliwiający osiągnięcie „profesjonalnych" rezultatów.

Jakkolwiek rzeczywiście istnieją na rynku kosztowne komercyjne systemy interaktywne, to nie należy fetyszyzować ich znaczenia w ogólnym efekcie pracy nad projektem. Niekomercyjne lub niedrogie programy i sensory mogą okazać się aż nadto wystarczające, by sprawnie wykonać zadanie, które chcemy zrealizować. Ponadto, wbrew pozorom, komercyjne systemy wcale nie muszą być bardziej uniwersalne od swych opensource'owych konkurentów – często te pierwsze są zaprojektowane tak, by spełniać bardzo konkretne i wąskie funkcje, nie dając na zbyt wiele możliwości samodzielnego dostosowywania do naszych potrzeb. Zanim wydamy nasz z trudem zdobyty budżet na kosztowny produkt, warto przyjrzeć się temu, co możemy otrzymać za niewielkie pieniądze lub nawet za darmo – zwłaszcza w fazie eksperymentów i uczenia się nie ma sensu zbyt szybkie przechodzenie na specjalistyczne urządzenia i aplikacje.

9. Projekt będzie polegał na przeprowadzeniu serii warsztatów uczących podstaw obsługi systemów interaktywnych i programowania prowadzących do powstanie spektaklu

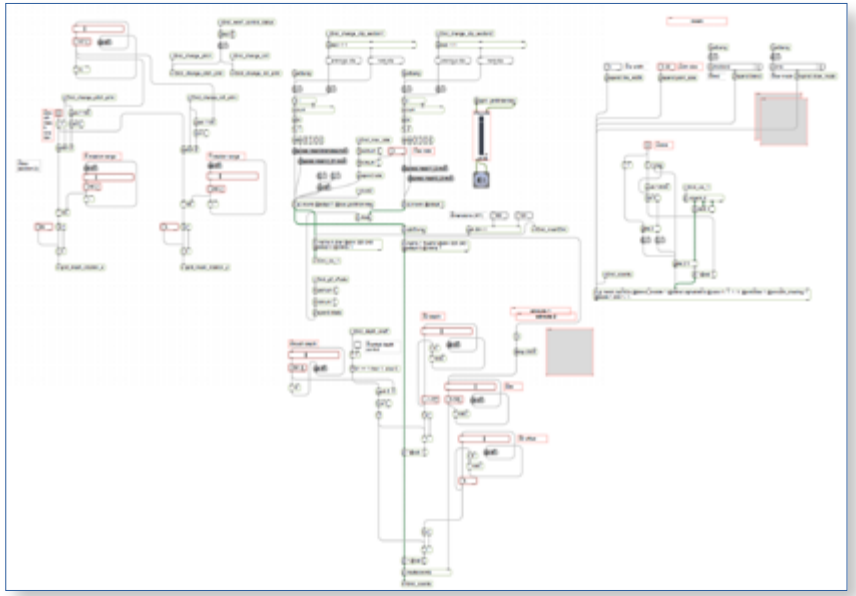
(lub np. performance), który następnie zostanie zaprezentowany publiczności.

W dramacie Nikołaja Erdmana Samobójca główny bohater postanawia nauczyć się gry na tubie. Zanim rozpocznie naukę przelicza dokładnie, ile czasu na nią poświęci oraz ile koncertów w miesiącu będzie dawał, kiedy już zostanie muzykiem. Mało tego, planuje nawet wpływy z koncertów uwzględniwszy ceny biletów, ilość słuchaczy i koszty wynajmu sali. Tytuł dramatu dobitnie świadczy o tym, do jakiego rezultatu prowadzi tego typu myślenie. Nie stawiajmy więc sobie zbyt ambitnych i dalekosiężnych zadań. Jeśli mamy ochotę na eksperymenty łączące media elektroniczne i formy sceniczne, po prostu przeprowadźmy je! Może się oczywiście okazać, że staną się one początkiem większego projektu, choć nie koniecznie w sposób, o jakim początkowo myśleliśmy. Z pewnością duch odkrywcy i radość z zajmowania się czymś interesującym popchną nas dalej, niż indukcyjna seria bazujących na sobie kolejno założeń.

Jakie błędy zwykle popełniamy
i jak ich uniknąć

02

architektura systemu



podobnie jak projekt w MaxMSP, system interaktywny
dobrze jest budować w myśl założonego planu

Typowy system interaktywny na potrzeby sceny daje się opisać jako szeregowe połączenie 3 modułów:

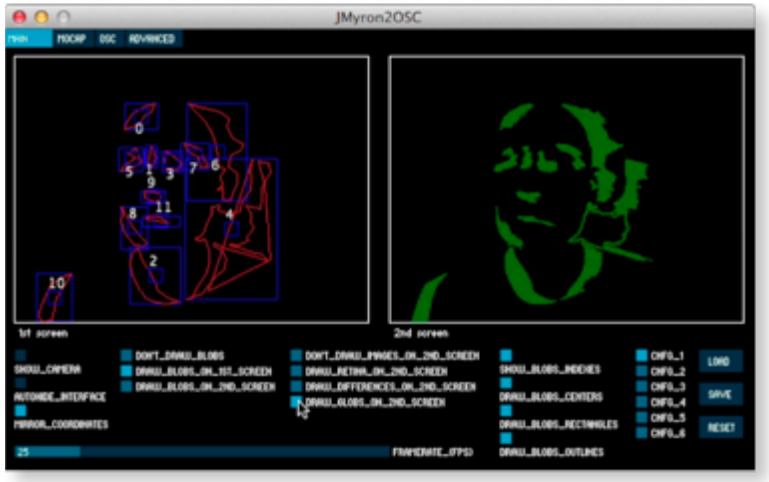
- sensorium i system transmisji danych
- oprogramowanie („mózg" systemu) i protokoły komunikacyjne
- media docelowe (zwykle: projekcja, dźwięk, światło)

Powyższa triada zwykle sprawia kłopoty na etapie pracy nad sensorium i oprogramowaniem. O ile obecność i charakter mediów docelowych, nawet dość intuicyjnie, wydają się oczywiste, o tyle dobór sensorów (osprzętu) i oprogramowania oraz scalenie ich w funkcjonalną całość zwykle nastręcza kłopoty i często zniechęca do dalszych eksperymentów. Dlatego też w niniejszym podręczniku skupimy się właśnie na sensorach i oprogramowaniu, dodając jednak kilka słów dotyczących pewnych kluczowych elementów związanych z mediami docelowymi.

Rozwój i dyfuzja w społeczeństwie technologii informatycznych, zarówno w konsumenckim, jak i eksperymentalnym wydaniu sprawia, że najprawdopodobniej nawet nie zdajemy sobie sprawy, jak wiele otaczających nas przedmiotów może służyć jako sensor (np. światła, czy ruchu) i jak wiele środowisk programistycznych i – często dostępnych na darmowych licencjach – gotowych aplikacji czeka na to by wykorzystać ich możliwości. Ponadto, poza urządzeniami i programami stworzonymi specjalnie po to, by stanowiły technologiczną podstawę do projektów scenicznych mamy do dyspozycji szereg strategii subwersywnych, pozwalających na wykorzystanie do własnych celów gadżetów i oprogramowania przeznaczonych pierwotnie do zupełnie innych celów.

Z wielu różnych powodów (w szczególności ze względu na wyjątkową obszerność tematów) wskazówki dotyczące zastosowania sensorów (i szerzej: sprzętu) i środowisk programistycznych oraz konkretnych aplikacji nie wyczerpują zagadnienia, a wprowadzają w temat, który potem można eksplorować już samodzielnie.

sensorium i system transmisji danych



aplikacja JMyron2OSC traktuje kamerę nie jako elektroniczny rejestrator obrazu, ale jako sensor ruchu

Zwykle myśli się o związkach mediów elektronicznych i sceny w kontekście jakiegoś rodzaju interakcji (bo wykorzystanie linearnego wideo lub np. mappingu jako swego rodzaju medialnej scenografii stało się na tyle powszechne, że – choć uzyskiwanie dzięki nim efekty potrafią być imponujące – nie ma sensu omawiać ich w niniejszym podręczniku).

Interakcja wymaga sensorium – elektronicznych zmysłów umożliwiających systemowi interaktywnemu reakcję na bodźce (ruch aktora/performera, aktywność publiczności, itp.). Większość typów sensorów i metod ich zastosowania została na potrzeby sceny przeszczepiona ze świata sztuki nowych mediów, w szczególności instalacji interaktywnych. Osobne grupy stanowią technologie wzięte z obszaru rozrywki elektronicznej i stworzone specjalnie na potrzeby sceny.

Inny podział sensorów mógłby opierać się przede wszystkim o sposób ich umieszczenia: niektóre typy sensorów w jakiś sposób monitorują przestrzeń we własnym zasięgu, inne wymagają bezpośredniego kontaktu z ciałem (lub choćby trzymania w ręce).
Do pierwszej kategorii należą głównie systemy śledzenia ruchu oparte o kamery, do drugiej m.in. akcelerometry i magnetometry.

Osobne zagadnienie (istotne zwłaszcza dla drugiej grupy sensorów) stanowią systemy przesyłania danych pomiędzy sensorami, a oprogramowaniem sterującym docelowymi mediami (np. dźwiękiem lub projekcją). Systemy przesyłania powinny być szybkie, niezawodne i – na ogół – działające – bez konieczności użycia przewodowych połączeń – trudno przecież sobie wyobrazić sytuację, w której tancerz lub performer jest „uwiązany" na kablu… Dla odmiany systemy śledzenia ruchu oparte o kamery trudno rozpatrywać bez myślenia o odpowiednim oprogramowaniu (można powiedzieć, że kamera staje się sensorem dopiero po uzupełnienie o odpowiedni software) – stąd w dalszej części podręcznika w rozdziale poświęconym pracy z kamerami odwołuję się od razu do aplikacji realizujących konkretne funkcje związane z detekcją i śledzeniem ruchu.

W niniejszym podręczniku przyjrzymy się raczej wyborowi technologii, których wspólnym mianownikiem będzie kombinacja dostępności, niskiej ceny i prostoty obsługi.

04

kamera jako sensor

każda, nawet najtańsza i najprostsza kamera video,
to doskonały i wszechstronny sensor ruch

Nawet prosta kamera sprzężona z odpowiednim oprogramowaniem może stać się wydajnym sensorem ruchu – tego rodzaju technologie często bywają wykorzystywane w instalacjach alarmowych, mogą też przydać się w wielu innych dziedzinach. Warto tu nadmienić, że zastosowania około-artystyczne takich systemów pojawiły się bardzo wcześnie, bo już na początku lat '80 XX wieku. Pionierów tego rodzaju systemów było oczywiście przynajmniej kilku, warto w tym miejscu wspomnieć urodzonego w 1942 r. Myrona Kruegera, znanego m.in. dzięki interaktywnemu środowisku *Videoplace* (1975) – nawiasem mówiąc Krueger rozpoczął eksperymenty z interaktywnymi systemami opartymi o systemy śledzenia ruchu już w późnych latach '60 XX wieku (w 1969 przestawił, wspólnie z Danem Sandinem, Jerrym Erdmanem i Richardem Venezkym, interaktywne środowisko *Glowflow*). Krueger, m.in. dzięki teoretycznej i praktycznej wiedzy o technologii był w stanie sformułować odpowiednie koncepcje i później wcielać je w życie. Krueger zainteresowany jest po dziś dzień interakcją jako samodzielną jakością, nie zaś interakcją „po coś", czy istniejącą jako funkcjonalny element dzieła sztuki – właściwie wszystkie jego realizacje mają charakter interaktywny i w jakiś sposób są związane z kontrolą środowiska medialnego za pomocą ciała (stąd Myron Krueger czasem bywa przywoływany jako pionier *VR* i *AR*).

Systemy śledzenia ruchu, pomimo „artystycznych" zastosowań ciągle nie utraciły związków z systemami nadzoru przemysłowego – np. popularna głównie w Niemczech zaprojektowana przez Friedera Weißa aplikacja EyeCon [http://eyecon.frieder-weiss.de] (wykorzystywana m.in. przez kompanie teatralne i zespoły łączące np. taniec z nowymi mediami) jest zoptymalizowana do pracy z kamerami nadzoru przemysłowego pracującymi w podczerwieni. Nawiasem mówiąc: częste wykorzystanie pasma podczerwieni w tego typu systemach jest spowodowane tym, że dzięki temu możemy uniezależnić się od światła widzialnego – np. scena oświetlona reflektorami podczerwieni i widziana przez kamerę pracującą w tym paśmie promieniowania elektromagnetycznego daje ciągle pełne możliwości kształtowania oświetlenia w zakresie widzialnego spektrum, podobnie nasz system będzie niewrażliwy na np. światło pochodzące z przypadkowych źródeł, choćby lamp błyskowych.

Systemy śledzenia ruchu oparte o kamerę okazały się też „brakującym ogniwem" pomiędzy sztuką technologii cyfrowych, a dyscyplinami związanymi z ciałem, takimi jak taniec, teatr, czy performance – dzięki informacjom z systemów śledzenia ruchu można łatwo powiązać fizyczną aktywność performera, aktora, lub tancerza z tym co dzieje się na planie mediów elektronicznych. Zasadniczo cały zakres sztuki związanej z ciałem zyskał tutaj bardzo wiele nowych możliwości. Powiązanie mediów elektronicznych i ciała możliwe jest oczywiście na wiele sposobów – kamera i zbudowany na niej system śledzenia ruchu ma tę zaletę (czy raczej „cechę", bo w pewnych warunkach może się to okazać wadą), że nie jest inwazyjna i nie wymaga ubierania, nakładania specjalnych sensorów, itp. (na tę bolączkę cierpiały onegdaj systemy VR wymagające zawsze niewygodnego i od pewnego momentu zwyczajnie śmiesznego rynsztunku) – po prostu znajdując się w polu widzenia kamery jesteśmy monitorowani i parametry naszego sposobu poruszania się modulują parametry otaczającego nas medialnego środowiska. Ta gotowość systemu śledzenia ruchu opartego o kamerę na przyjęcie „przypadkowego przechodnia" (znajdując się w polu widzenia kamery jesteśmy

„analizowani", czy tego chcemy czy nie) wydaje się zresztą również bardzo ważna.

Co ciekawe systemy śledzenia ruchu pozwoliły zwrócić uwagę na to, że kamera rejestruje pewne informacje, pozornie nieobecne w wynikowym dwuwymiarowym obrazie: istnieją algorytmy pozwalające zdekomponować z zarejestrowanego lub napływającego w czasie rzeczywistym strumienia kadrów informacji o położeniu obiektów (w tym samej kamery – swoją drogą, tego rodzaju algorytmy należą do nielicznych z grupy, wykorzystywanych przez filmowców: niekanoniczne zastosowania kamery ciągle pozostają w domenie twórców medialnych i... przemysłu rozrywkowego, choć film intensywnie je inkorporuje, zwłaszcza w czasach masowego przechodzenia na technologię 3D) w przestrzeni trójwymiarowej – okazało się więc, że nie jest wcale tak oczywiste, że rozumiemy, czym tak naprawdę kamera jest i jakie daje możliwości. Aby spojrzeć na kamerę pod tym kątem musieliśmy jednak uzupełnić ją o programowalny, cyfrowy symbiont.

Zakres analizy obrazu może obejmować wykrywanie naruszenia zdefiniowanych stref (jak w przypadku ActiveZones2OSC [http://wrocenter.pl/lab]), śledzenie pojedynczego lub wielu obiektów jednocześnie (właśnie tego rodzaju tracking uzupełniony o szereg dodatkowych opcji realizuje JMyron2OSC [http://wrocenter.pl/lab]), analizę kształtu, wykrywanych znaczników (w tym systemy w rodzaju reacTIVision [http://reactivision.sourceforge.net/] lub ARToolKit [http://www.hitl.washington.edu/artoolkit/] oraz niektóre kontrolery gier), itp. Ponadto poza oczywistym zakresem światła widzialnego można budować systemy śledzenia ruchu wykorzystujące kamery podczerwieni lub termiczne (ma to znaczenie, jeśli np. chcemy uniezależnić działanie naszego systemu od przypadkowych zmian oświetlenia).

Zasadniczo każdy system śledzenia ruchu wykorzystujący kamerę działa porównując ze sobą kolejne klatki obrazu: przyjmując, że kamera jest nieruchoma (oraz nie manipulujemy ustawieniami obiektywu i zakładając stałe parametry oświetlenia w paśmie widzianym przez kamerę) różnice w kolejnych klatkach obrazu wynikają z obecności w kadrze ruchomych obiektów. Idąc tym tropem dość łatwo jest budować i implementować (lub przynajmniej wyobrazić sobie taką możliwość) różne algorytmy analizujące – biorąc pod uwagę wydajność współczesnych komputerów może się to odbywać w czasie rzeczywistym – zdigitalizowany obraz pod kątem wykrywania i śledzenia zmian wskazujących na obecność „ruchu" w kadrze (zaawansowane systemy śledzenia ruchu potrafią w tej chwili znaczenie więcej, łącznie z analizą geometrii obrazu i rekonstrukcją przestrzeni 3D).

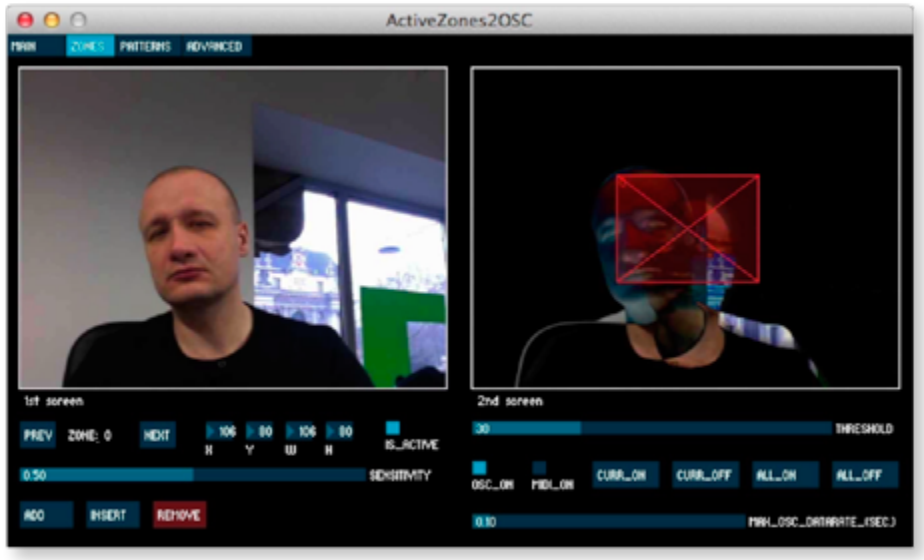
W jaki sposób (na początek) warto pracować z kamerą:

1. **O sile tkwiącej w możliwości potraktowania kamery jako pracującego w czasie rzeczywistym sensora (a nie biernego rejestratora obrazu) łatwo się przekonać sięgając po technikę tak starą jak sama sztuka wideo: wystarczy jakąkolwiek kamerę podpiąć do wyświetlacza (wszystko jedno, jakiego typu), tak by pokazywał w czasie rzeczywistym obraz widziany przez kamerę.**

Teraz wystarczy skierować obiektyw kamery na wyświetlacz – tak by znalazł się w jej polu widzenia. Uzyskamy w ten sposób wizualną pętlę sprzężenia zwrotnego, której parametry bardzo łatwo modyfikować choćby poprzez zmianę położenia kamery względem wyświetlacza. Jeśli mamy taką możliwość (odpowiednio długi kabel lub bezprzewodowe połączenie pomiędzy kamerą a wyświetlaczem) możemy oddać taki „system interaktywny" do dyspozycji performerowi – szybko okaże się, że jego obsługa jest intuicyjna i nie wymaga praktycznie żadnego teoretycznego przygotowania, a efekty połączenia ruchu i pracy tandemu kamera-wyświetlacz mogą się okazać intrygujące. Warto też zauważyć, że do takich eksperymentów wystarczy nam najzwyklejsza kamera USB – nie musimy stosować żadnych kosztownych urządzeń (jeśli nie mamy w tym momencie „pod ręką" kamery USB, a czytamy ten podręcznik za pomocą urządzenia – np. laptopa – wyposażonego w kamerę spróbujmy włączyć obraz z kamery na pełnym ekranie lub w możliwie dużym oknie, a następnie użyć zwykłego lusterka odbijając w nim obraz ekranu i kierując go w stronę kamery).

2. Prosty system detekcji naruszenia stref w monitorowanej przestrzeni.

Następnym logicznym krokiem będzie włączenie komputera – i odpowiedniego oprogramowania – pomiędzy kamerę i wyświetlacz. Będziemy potrzebowali dowolnej, dającej się podłączyć do komputera kamery (może być to nawet kamera wbudowana w laptop, albo kamera USB, lub FireWire). Będziemy też potrzebować oprogramowania (szerzej ten temat zostanie omówiony w jednym z kolejnych rozdziałów) analizującego obraz z kamery i wychwytującego zmiany

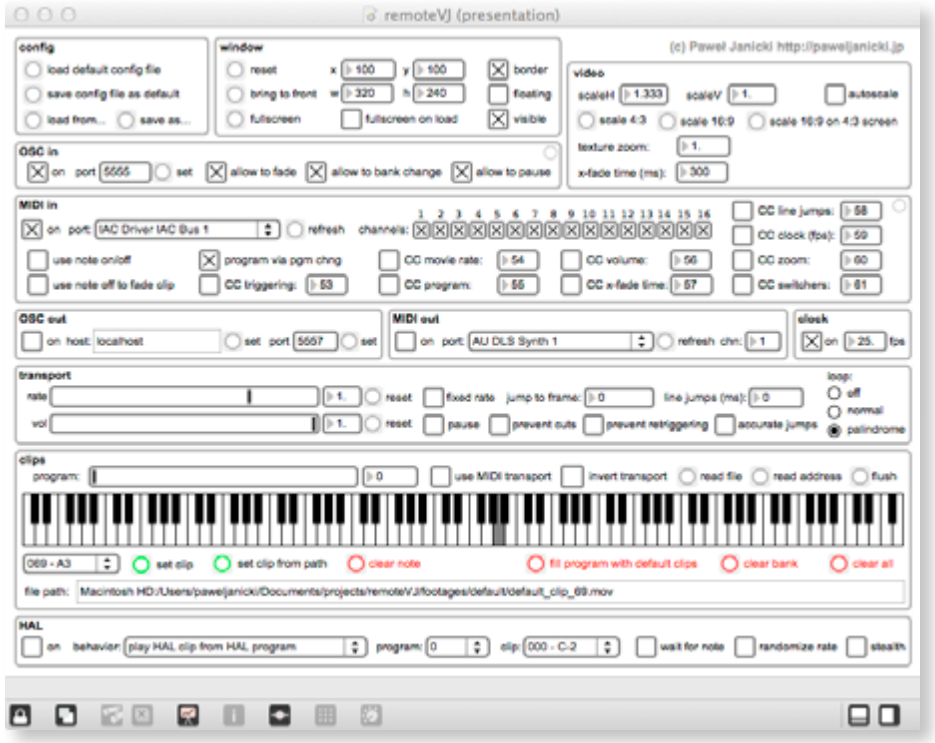


ActiveZones2OSC

w wybranych miejscach kadru. Ponieważ na tym etapie koncentrujemy się na rozwiązaniach najprostszych, proponuję sięgnąć po gotowy i dostępny nieodpłatnie (z polską instrukcją obsługi) program ActiveZones2OSC [dostępny na stronie http://wrocenter.pl/lab] – dzięki niemu będziemy mogli zdefiniować w widzianym przez kamerę obrazie strefy, których naruszenie przez poruszające się obiekty wygeneruje komunikaty MIDI i OSC. Warto poświęcić nieco czasu na naukę pracy z programem (lub sięgnąć po inne narzędzie realizujące wykrywanie ruchu na podobnym poziomie skomplikowania), ponieważ, pomimo prostoty, tworzy on bazę dla bardzo wydajnego systemu interakcji.

Następnym krokiem w pracy z aplikacją ActiveZones2OSC – jeśli już opanowaliśmy tworzenie aktywnych stref – będzie sprzęgnięcie jej z jakimś innym programem lub urządzeniem. Jeśli dysponujemy np. syntezatorem MIDI możemy skierować komunikaty MIDI generowane przez ActiveZones2OSC do portu MIDI, do którego podpinamy również syntezator po to, by wykorzystać aplikację jako rodzaj wirtualnej klawiatury obsługiwanej gesturalnie. Możemy też spróbować jakiejś innej metody na wykorzystanie komunikatów wysyłanych przez ActiveZones2OSC. Jeśli sprzęgniemy go z aplikacją remoteVJ, dostępną również na stronie Centrum Sztuki WRO [http://wrocenter.pl/lab] uzyskamy łatwo możliwość gesturalnej kontroli nad odtwarzanym materiałem wideo.

System złożony z kamery, komputera, wyświetlacza, ActiveZones2OSC i remoteVJ jest już w pełni sprawną technologią możliwą do zastosowania w performance lub innej formie scenicznej. Jeśli uda nam się przetestować w praktyce taki system, to – poza zyskaniem



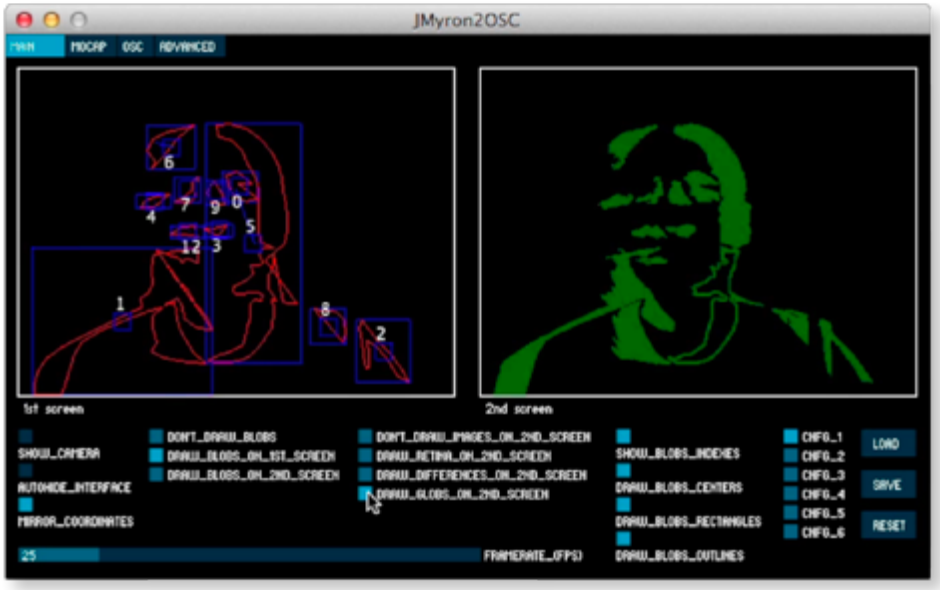
remoteVJ

praktycznego i wartościowego doświadczenia – będziemy w stanie szybko szkicować interaktywne formy sceniczne bazujące na grze pomiędzy performerem a samą przestrzenią. Może się okazać, że tego rodzaju prosta technologia wystarczy na jakiś czas na nasze potrzeby.

3. Detekcja poruszających się w polu widzenia kamery obiektów.

Wykorzystując analogiczny setup (kamera-komputer-wyświetlacz) możemy pokusić się o bardziej wyrafinowane metody śledzenia obiektów, niż przedstawiona w poprzednim przykładzie. Spróbujmy teraz, zamiast wykrywać naruszenie zdefiniowanego wcześniej obszaru, wykryć pozycje i kształty poruszających się w polu widzenia kamery obiektów. Podobnie, jak w poprzednim przykładzie, rozpocznijmy od gotowej aplikacji dostępnej na stronie Centrum Sztuki WRO [http://wrocenter.pl/lab] – będzie to program JMyron2OSC.

Aplikacja ta analizuje obraz pobierany w czasie rzeczywistym przez urządzenie wideo (kamera, digitizer) starając się wyłowić z niego informacje o ewentualnych ruchomych obiektach. Na informacje składają się dane o położeniu obiektu (koordynaty centrum i opisujący obiekt prostokąt) oraz jego kształcie, wektor ruchu, kolor centrum obiektu. Dane te są wizualizowane wewnątrz interfejsu JMyron2OSC, ale też – a w zasadzie: przede wszystkim – mogą być przesyłane do zewnętrznych aplikacji i urządzeń za pomocą protokołu komunikacyjnego OSC (OpenSound Control, więcej o OSC na stronie: http://opensoundcontrol.org).



JMyron2OSC

Tym samym informacje uzyskane z JMyron2OSC można wykorzystać w praktycznie dowolnym celu, sprzęgając program z większością środowisk programistycznych, aplikacji i urządzeń stosowanych podczas konstruowania struktur interaktywnych. Aplikacja dysponuje dwoma głównymi, przełączanymi trybami emisji komunikatów OSC w zależności od potrzeb dzieląc uzyskane dane na niewielką ilość długich komunikatów OSC o zmiennej długości lub dłuższą serię krótkich komunikatów o stałej długości – ma to praktyczne znaczenie w przypadku sprzęgania JMyron2OSC z wizualnymi językami programowania (MaxMSP. Pure-Data, itp.), ponieważ wielu użytkownikom tych ostatnich trudno jest przetwarzać komunikaty OSC o zmiennej długości bez sięgania po skrypty lub inne techniki wiążące się z głębszą znajomością tych środowisk.

Oczywiście JMyron2OSC posiada szereg funkcji umożliwiających dostrojenie działania programu do wymagań użytkownika. Wśród globalnych parametrów i opcji warto wymienić możliwość ustawienia progu zadziałania sensora ruchu (technicznie rzecz biorąc jest to wartość różnicy w poziomie jasności piksela obrazu tła i piksela [o tych samych współrzędnych] bieżącej klatki obrazu), częstotliwości odświeżania obrazu, szeregu parametrów protokołu OSC (w tym możliwość detekcji i transmisji tylko wybranych parametrów systemu śledzenia ruchu oraz przeskalowywania wybranych parametrów), itp.

Jak wspomniałem wyżej, systemy śledzenia ruchu wykorzystujące kamerę jako sensor (pomimo, że działają w oparciu o, w ogólności, jedną i tę samą koncepcję) znaczenie różnią się od siebie sposobami, w jakie w praktyce wcielają w życie ideę leżącą u podstaw ich konstrukcji. Tym samym w zależności od zastosowanego systemu śledzenia ruchu monitorowana przestrzeń i zachodzące w niej wydarzenia są interpretowane na rozmaite sposoby. Niemniej JMyron2OSC został tak opracowany, by generować możliwie standardowy zestaw komunikatów. Tak więc JMyron2OSC analizuje napływające klatki obrazu, starając się wyłowić z nich informacje o poruszających się w kadrze obiektach. Aplikacja potrafi identyfikować wiele ruchomych obiektów jednocześnie, przypisując każdemu z nich unikalny index oraz wyłuskać ze strumienia wideo następujące parametry każdego obiektu:

- pozycję centrum obiektu
- pozycję i wielkość prostokąta, w którym zawiera się obiekt
- ilość i koordynaty punktów obrysu obiektu
- index (unikalny numer reprezentujący ruchomy obiekt – index pozwala śledzić wiele poruszających się jednocześnie obiektów identyfikując każdy)
- kolor centrum
- koordynaty miejsca w którym pojawił się ruchomy obiekt o danym indeksie
- wektor ruchu (wyliczany w stosunku do poprzedniej klatki obrazu)

Poza tym, dla każdej klatki obrazu generowany jest zestaw informacji globalnych i statystycznych obejmujących m.in.:

- ilość ruchomych obiektów na scenie (w aktualnej klatce analizowanego obrazu)
- maksymalna ilość punktów obrysu obiektu
- parametry strumienia wideo (rozdzielczość, fps)

- prędkość procesowania (fps)
- ilość i indeksy nowych ruchomych obiektów
- ilość i indeksy obiektów, które zniknęły z kadru
- ilość i indeksy obiektów, które kontynuują swój ruch

Powyższy opis może się wydać nieco przydługi – ukazuje jednak możliwości tkwiące w bardzo prostej technologii – użyciu najzwyklejszej kamery w połączeniu z odpowiednim oprogramowaniem. W przypadku wykorzystania systemu śledzenia ruchu o możliwościach porównywalnych z JMyron2OSC problemem staje się raczej skonsumowanie dostępnych możliwości. Łatwo zauważyć, że nawet dysponując system śledzenia ruchu należy jeszcze umieć wykorzystać generowane przez taki system komunikaty. Tutaj nie obejdzie się już bez programowania (choćby wizualnego), niemniej – z drugiej strony – jak widać, warto pokusić się o poznanie podstaw pracy z jakim środowiskiem programistycznym lub wizualnym (w rodzaju Pure-Data lub MaxMSP/Jitter).

Oczywiście, jak wspomniałem wyżej, JMyron2OSC nie jest jedyną „gotową" aplikacją spełniającą podobną funkcję i zamiast niej możemy wykorzystać szereg innych narzędzi (np. dostępny również nieodpłatnie i bardzo wydajny OpenTSPS [http://opentsps.com]). Możemy też stworzyć odpowiednią aplikację samodzielnie, jeśli posiadamy podstawowe umiejętności programistyczne.

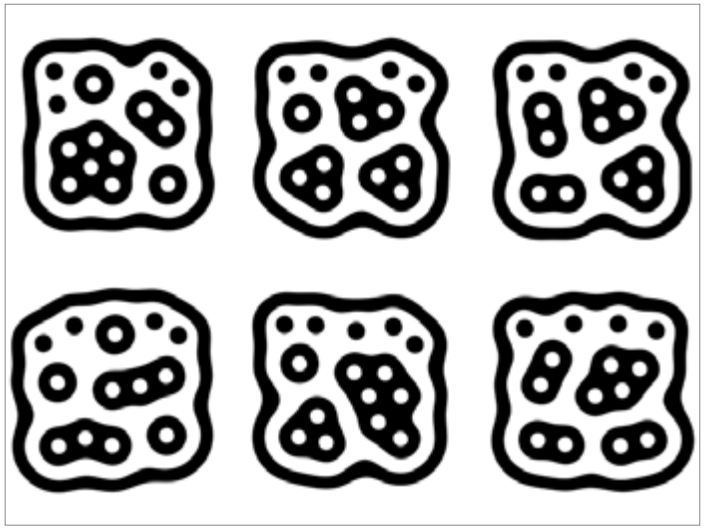
OpenTSPS

4. Detekcja markerów.

Istnieje sporo gotowego oprogramowania umożliwiającego wykorzystanie kamery jako detektora wizualnych markerów (specjalnie

spreparowanych obrazów zawierających kształty rozpoznawane przez oprogramowanie). Jakkolwiek w warunkach scenicznych tego rodzaju koncept znajduje zastosowanie raczej niezbyt często, to warto go poznać już choćby dlatego, że, podobnie jak w przypadku systemów omówionych w punktach [2] i [3], nie wymaga on szczególnych nakładów finansowych i dysponując komputerem z kamerą oraz wydrukiem z markerami możemy śmiało przetestować możliwości takiej technologii.

Najłatwiej rozpocząć eksperymenty z systemami detekcji markerów od przetestowania technologii reacTIVision [http://reactivision.sourceforge.net/]. W tym celu należy pobrać ze strony projektu aplikację rozpoznającą markery (w wersji dla wykorzystywanego przez nas systemu operacyjnego) i wydrukować kilka markerów (standardowo aplikacja reacTIVision jest ustawiona na tryb detekcji zestawu markerów nazwanego amoeba – zestaw markerów w formacie PDF znajdziemy w folderze z programem, w podfolderze „symbols/amoeba/legacy.pdf"). Kiedy tylko uruchomimy aplikację i jakikolwiek program odbierający i monitorujący komunikaty OSC wysyłane przez reacTIVision, a następnie umieścimy jeden lub więcej markerów w polu widzenia kamery program rozpozna marker (numer markera pojawi się na ekranie programu) i wyśle jego parametry (m.in. identyfikator, położenie, kąt, o jaki obrócony jest marker) za pomocą komunikatów OSC do wybranej lokalizacji. Komunikaty te, za pomocą dowolnego środowiska programistycznego zdolnego do odbioru danych poprzez OSC możemy następnie wykorzystać w praktycznie dowolny sposób.



przykładowe markery (zestaw "Amoeba") używane w reacTIVision

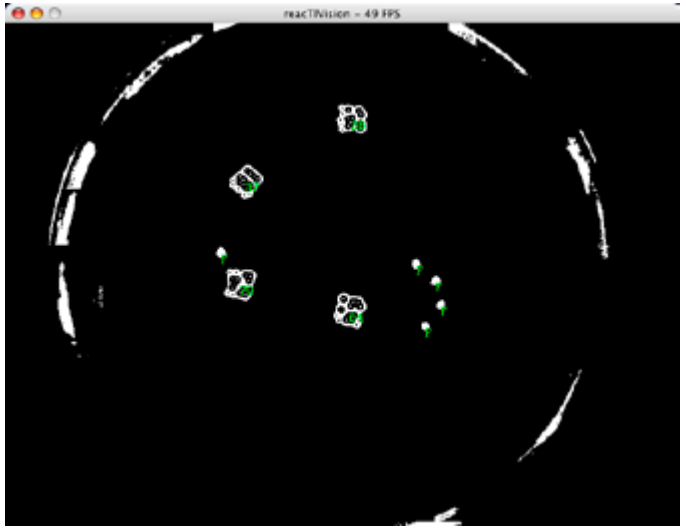
Jakkolwiek reacTIVision zwykle wykorzystywany jest do projektów w mniejszej skali (m.in. słynnego onegdaj instrumentu Reactable), to praca z nim może dać nam nieocenione doświadczenia związane z umiejętnością wykorzystania kamery jako sensora (w tym możliwość przećwiczenia zależności pomiędzy jakością oświetlenia i rozdzielczością kamery a jakością detekcji patternów).

Nieco więcej wiedzy i doświadczenia wymaga zastosowanie któregoś z bardziej zaawansowanych systemów detekcji patternów, przede wszystkim tych, które związane są z Augmented Reality: zespołem technologii zdolnym do obrazu widzianego przez kamerę na bieżąco dodawać generowane za pomocą komputera elementy. Relatywnie łatwo z AR radzą sobie środowiska programistyczne takie jak Processing, MaxMSP/Jitter i OpenFrameworks – pracując w którymś z nich z pewnością poradzimy sobie ze stworzeniem systemu zdolnego rozpoznawać i pozycjonować tagi w przestrzeni trójwymiarowej.

5. Stereoskopia.

Stereoskopia, czyli wykorzystanie kamery o dwu obiektywach pozwalającej rejestrować obraz w trzech wymiarach daje nam szereg ciekawych możliwości.

Przede wszystkim, na rynku dostępne są w tej chwili relatywnie niedrogie kamery stereoskopowe wykorzystywane jako kontrolery dla konsol służących do zabawy w gry wideo (przede wszystkim MS Kinect i jego klony – ponieważ w trakcie pisania tego podręcznika popularność technologii opartych o stereoskopię rośnie można

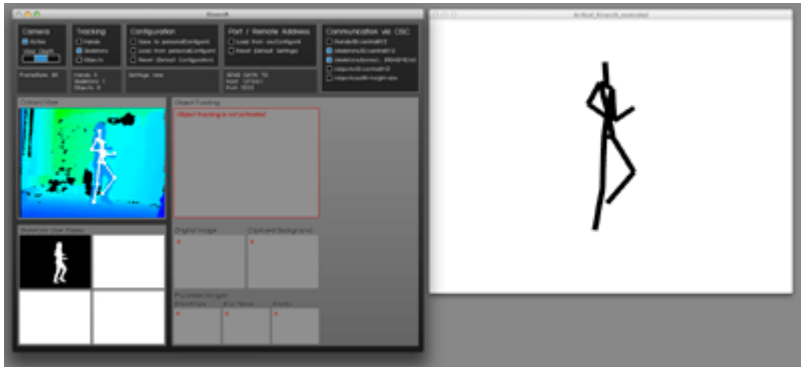


markery widoczne na podglądzie klienta reacTIVision

MS Kinect - stereoskopowa kamera pracująca zarówno w świetle widzialnym jak i w podczerwieni

się spodziewać, że już niedługo będziemy mieli do wyboru więcej podobnych rozwiązań). Tego rodzaju kamery zwykle potrafią rejestrować obraz w paśmie podczerwieni, co z kolei pozwala na uniezależnienie się od oświetlenia w paśmie widzialnym – a to bywa niezwykle ważnym współczynnikiem podczas przygotowywania systemów interaktywnych z myślą o ich zastosowaniu na scenie. Ponadto detekcja obrazu w trzech wymiarach pozwala na jego analizę w dość wymyślny sposób, a przede wszystkim pod kątem wynajdywania w polu widzenia kamery poruszających się obiektów, które daje się zinterpretować jako ludzkie sylwetki. Takie obiekty można następnie za pomocą odpowiedniego oprogramowania rozłożyć na poszczególne elementy (głowa, korpus, stawy rąk i nóg), pozycjonować w przestrzeni 3D i wykorzystać dane o ich położeniu do sterowania wybranych elementów naszego systemu (np. wyświetlanego na ekranie awatara kopiującego ruchy performera) – tego rodzaju systemy określa się na ogół jako „detekcję szkieletu", a w terminologii filmowej motion capture.

Z Kinectem (lub podobną kamerą) stosunkowo łatwo rozpocząć eksperymenty wykorzystując aplikację KinectA [http://www.mihoo.de/kinecta.html] – działa ona bardzo podobnie do omawianych wcześniej programów służących do śledzenia ruchu za pomocą „zwykłych" kamer. KinectA pozwoli nam łatwo wygenerować dane szkieletu, obserwować je i przesłać za pomocą komunikatów OSC do innych aplikacji lub środowisk programistycznych. Do aplikacji i środowisk programistycznych, które możemy wykorzystać w pracy z Kinectem powrócimy jeszcze w rozdziale poświęconym oprogramowaniu.



MS Kinect, KinectA, Processing

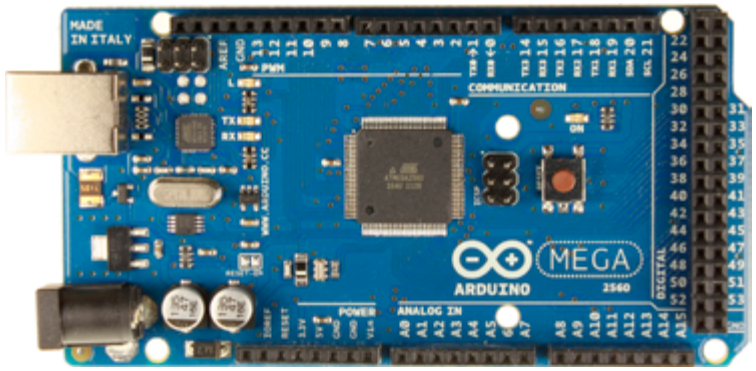
kurs Artkod: Processing
http://www.nina.gov.pl/kultura-2_0/sam-to-zrób/kurs-processingu

Powyższe zestawienie jasno – jak sądzę – ukazuje, że kamera może być tanim, funkcjonalnym i wszechstronnym sensorem. O możliwościach pracy z kamerą specjalnie wspominam na początku sekcji podręcznika omawiającej hardware. Najprawdopodobniej każdy czytelnik posiada przynajmniej jedną kamerę nadającą się do wykorzystania w jakimś systemie śledzenia ruchu. Jeszcze bardziej prawdopodobne jest to, że ma takich kamer kilka. Chcąc rozpocząć pracę z systemami interaktywnymi warto na początek zainteresować się właśnie kamerami i odpowiednim oprogramowaniem – wyrobimy sobie dzięki temu pewne pojęcie o możliwościach pracy z systemami interaktywnymi w ogólności, a w szczególności czymś, co można by określić jako zarządzanie przestrzenią w projektach związanych z interakcją: szybko przekonamy się, że wzajemne relacje przestrzenne pomiędzy kamerą, performerem, ekranem (o ile stosujemy ekran) itp. w dużym stopniu stanowią o jakości efektu finalnego i charakterze naszych działań.

Arduino i RaspberryPI *open source hardware*

Osobną kategorią możliwych do wykorzystania sensorów są takie, które „montujemy” do ciała – mogą być to sensory położenia, przyspieszenia, monitory napięcia mięśni, itp. Chcąc skorzystać z takich sensorów możemy zakupić komercyjny zestaw – zwykle złożony z samych sensorów, systemu transmisji danych i oprogramowania – spełniający nasze wymagania, lub postarać się stworzyć taki system samodzielnie. W tym ostatnim przypadku najprawdopodobniej skorzystamy z jednego z niedrogich urządzeń licencjonowanych na zasadach open source, pozwalających obsługę różnego typu sensorów i przesyłanie uzyskanych z nich danych do pozostałych elementów naszego scenicznego systemu. Typowymi urządzeniami tego rodzaju są nieco różniące się pod względem charakteru i możliwości – ale jednak w obu przypadkach określane jako mikrokontrolery – Arduino i RaspberryPI.

Korzystając z tego typu narzędzi musimy liczyć się z prawdopodobieństwem, że będziemy musieli zaprogramować samodzielnie wybrane przez nas urządzenie oraz zmontować (na ogół bardzo prosty) układ łączący sensory z mikrokontrolerem – czasem ten „układ” to po prostu dwa lub trzy przewody, więc nie ma powodu do niepokoju… Osoby, które nie miały wcześniej do czynienia z lutownicą i programowaniem mogą się w tym momencie poczuć nieswojo, ale z drugiej strony popularność np. Arduino nie bierze się z niczego – jest to bardzo wszechstronne urządzenie umożliwiające kontrolę wielu typów sensorów. Ponadto Arduino w wersji z łączem bluetooth pozwoli nam na bezprzewodowe przesyłanie sygnałów z sensorów, co niewątpliwie stanowi jeden z zasadniczych problemów technicznych, jakie należy rozwiązać konstruując system sensoryczny, który zamierzamy wykorzystać na scenie. Istnieje



Arduino – tani, programowalny mikrokontroler

też sporo gotowego oprogramowania dla Arduino pozwalającego obsłużyć większość typowych sensorów, tak więc przy pewnej dozie szczęścia może się okazać, że programowania – przynajmniej po stronie mikrokontrolera – unikniemy. Więcej informacji o Arduino znajduje się na stronie: [http://arduino.cc]

Z kolei dla RaspberryPI znajdziemy szereg względnie wysokopoziomowych aplikacji i bibliotek pozwalających np. na interaktywne odtwarzanie wideo – w przypadku wykorzystania tego ostatniego urządzenia może się okazać, że tradycyjnie rozumiany komputer jest w naszej infrastrukturze zbędnym elementem. Więcej informacji o RaspberryPI znajduje się na stronie: [http://raspberrypi.org]

Decydując się na pracę z Arduino, RaspberryPI, lub innym urządzeniem tworzonym w duchu DIY i na licencji open source możemy się więc z jednej strony liczyć z faktem, że będziemy musieli poświęcić pewną ilość czasu na naukę pracy z mikrokontrolerem (w zależności od naszych wcześniejszych doświadczeń może to oznaczać również poznanie podstaw pracy z programowaniem i mikroelektroniką), ale – początkowo płaska – krzywa postępu w stosowaniu tego typu technologii z czasem staje się coraz bardziej stroma i po początkowych trudnościach i frustracjach nieoczekiwanie dla samych siebie możemy uświadomić sobie, że jesteśmy w stanie samodzielnie tworzyć elektroniczne komponenty systemu interaktywnego – w tym również takie, których nikt przed nami jeszcze nie zbudował. Niebagatelny jest też fakt, że zarówno Arduino, jak i RaspberryPI są urządzeniami bardzo dobrze opisanymi, nie będziemy więc mieli trudności ze znalezieniem tutoriali i kursów, opisów gotowych projektów lub uzyskaniem pomocy od bardziej zaawansowanych

użytkowników (ta ostatnia zaleta może się okazać przeważająca, ponieważ spora część komercyjnych systemów interaktywnych zarabia na siebie m.in. za pomocą płatnych konsultacji dotyczących wdrożeń i związanych z nimi problemów).

RaspberryPI – pomimo niepozornego wyglądu RaspberryPI to naprawdę wydajny komputer zdolny odtwarzać wideo wysokiej rozdzielczości, obsługiwać sensory i wykonywać wiele innych zadań]

Wii Remote

Wii Remote - ten kontroler zaprojektowany z myślą o grach komputerowych łatwo można wykorzystać jako sensor ruchu/położenia/ przyspieszenia

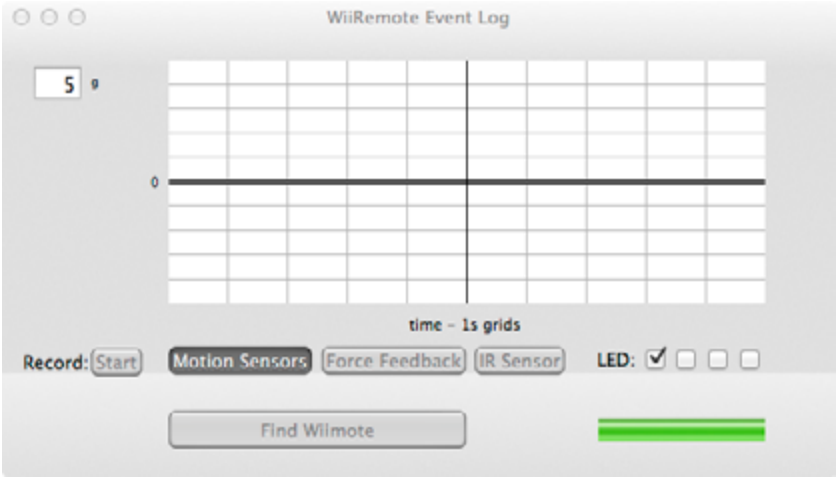
Wii Remote (często, w skrócie, zwany również wiimote) to, podobnie jak MS Kinect kontroler wykorzystywany jako uzupełnienie systemu rozrywki elektronicznej. Wiimote, w przeciwieństwie do MS Kinect nie jest jednak kamerą – jest niewielkim urządzeniem trzymanym zwykle w ręce, przekazującym poprzez bluetooth informacje o (między innymi) dynamice ruchu, orientacji względem pola magnetycznego Ziemi i o stanie przycisków kontrolera (wiimote posiada kilka przycisków, które, naturalnie, mogą być przez trzymającego wciskane lub zwalniane) – dostępnych parametrów jest zresztą więcej, a a dostęp do nich jest częściowo uzależniony od wersji kontrolera i dodatkowego osprzętu. Wiimote, kupiony okazyjnie może kosztować niewiele (nawet poniżej 30 złotych), a stanowi gotowy zestaw sensorów, który bez większego trudu można wykorzystać w interaktywnym systemie kontrolowanym przez aktora, performera lub tancerza. Co więcej, w wiimote wbudowane jest kilka diod oraz system force feedback (układ wytwarzający wibracje) – można dzięki nim performerowi trzymającemu w ręce kontroler przekazywać sygnały. Krótko mówiąc: wiimote jest urządzeniem, na które warto zwrócić uwagę, zwłaszcza, jeśli chcemy szybko rozpocząć pracę związaną z bezpośrednim zaangażowaniem performera i mediów docelowych – kontroler ten nie wymaga długiego procesu przygotowawczego ani budowania własnego hardware – praktyczne eksperymenty z kontrolerem można rozpocząć w kilka chwil od podłączenia go do komputera i dokonania odpowiednich ustawień w oprogramowaniu.

Wykorzystanie wiimote jako sensora przekazującego informacje do pozostałych komponentów systemu interaktywnego jest łatwe, o ile tylko dysponujemy komputerem z łączem bluetooth (możemy

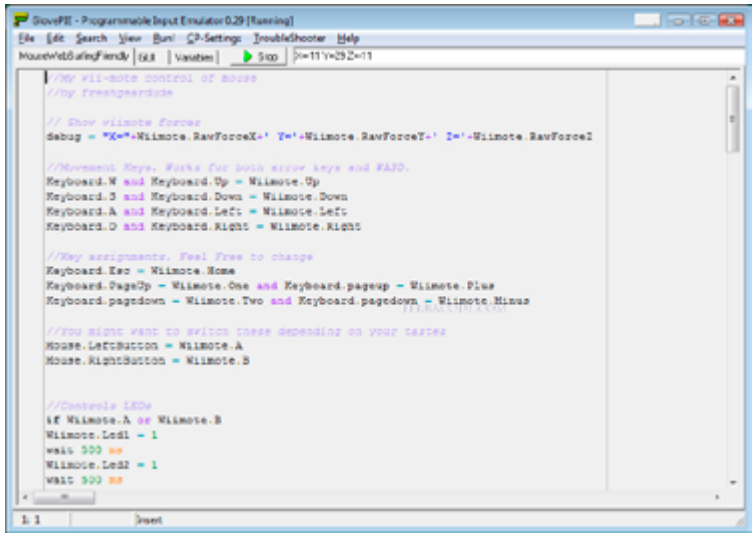
Wii Remote jest lekki, niewielki, wytrzymały i łatwo komunikuje się z komputerem

też nabyć takie łącze w formie urządzenia podpinanego przez USB, a koszt takiego rozwiązania nie powinien przekraczać 20-30 złotych) i systemem operacyjnym Windows lub OSX (w przypadku systemów z rodziny Linux również możemy wykorzystać wiimote, np. za pomocą obiektów wbudowanych w środowisko Pure-Data [http://puredata.info]). Najprościej będzie – podobnie jak robiliśmy to w przypadku danych uzyskiwanych z systemu śledzenia ruchu opartego o kamerę – wykorzystać do tego celu aplikację udostępniającą uzyskane z wiimote dane za pomocą protokołu komunikacyjnego OSC lub MIDI. Typową aplikacją wykorzystywaną do tego celu w systemie OSX jest DarwiinRemoteOSC [https://code.google.com/p/darwiinosc]. Pracując pod systemem Windows można skorzystać z – bardzo zresztą ciekawej jako hub pozwalający transmitować informacje z wielu różnych urządzeń – aplikacji GlovePIE [http://glovepie.org/glovepie.php].

Podobnie, jak w przypadku technologii omawianych powyżej, uzyskane dzięki wiimote komunikaty OSC możemy przesłać do praktycznie dowolnego środowiska programistycznego lub aplikacji. W przypadku danych uzyskanych z wiimote szczególnie łatwa jest bezpośrednia ich konwersja do postaci komunikatów MIDI i kontrola za ich pośrednictwem np. syntezatora lub samplera (w formie fizycznego urządzenia lub oprogramowania).



DarwiinRemoteOSC


GlovePIE - Programmable Input Emulator 0.29 [Running]
File Edit Search View Run! GP-Settings TroubleShooter Help
//My wii-mote control of mouse
//by freshgearduke
// Show wiimote forces
debug = "X="+Wiimote.RawForceX+' Y='+Wiimote.RawForceY+' Z='+Wiimote.RawForceZ
//Movement Keys. Works for both arrow keys and WASD.
Keyboard.W and Keyboard.Up = Wiimote.Up
Keyboard.S and Keyboard.Down = Wiimote.Down
Keyboard.A and Keyboard.Left = Wiimote.Left
Keyboard.D and Keyboard.Right = Wiimote.Right
//Key assignments. Feel free to change
Keyboard.Esc = Wiimote.Home
Keyboard.PageUp = Wiimote.One and Keyboard.pageup = Wiimote.Plus
Keyboard.pagedown = Wiimote.Two and Keyboard.pagedown = Wiimote.Minus
//You might want to switch these depending on your tastes
Mouse.LeftButton = Wiimote.A
Mouse.RightButton = Wiimote.B
//Controls LEDs
if Wiimote.A or Wiimote.B
Wiimote.Led1 = 1
wait 500 ms
Wiimote.Led2 = 1
wait 500 ms
1: 1
Insert

GlovePIE

kilka słów o komercyjnych systemach interaktywnych

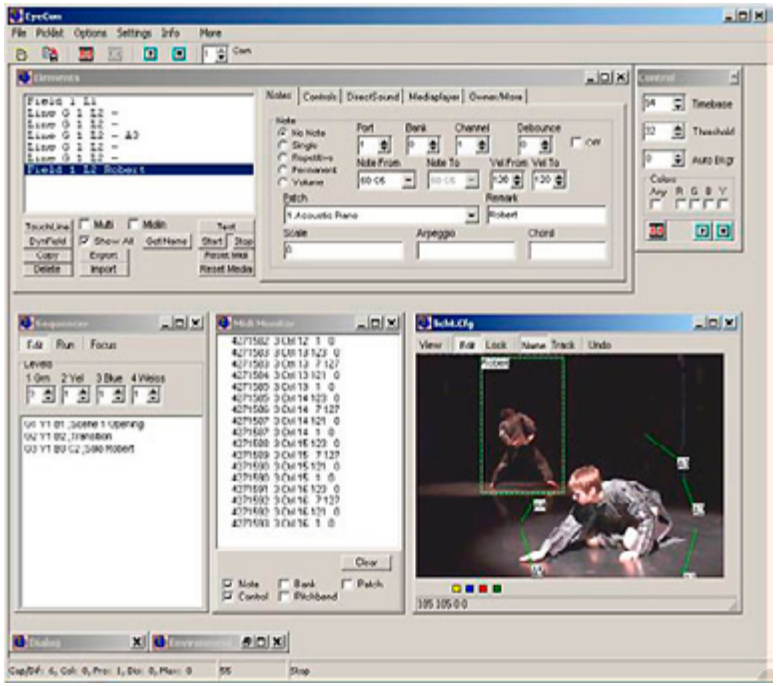
Istnieją, jak już wspomniałem, komercyjne systemy interaktywne budowane z myślą o zastosowaniach scenicznych. Mam na myśli zarówno takie systemy, które składają się wyłącznie z oprogramowania, jak i takie, które zawierają również komponenty sprzętowe. Wiele z tych systemów posiada naprawdę imponujące możliwości i dobry stosunek możliwości do ceny, jakkolwiek w niniejszym podręczniku skupiamy się głównie na tym, co można uzyskać przy minimalnych nakładach finansowych lub całkowicie beznakładowo.

Niemniej warto być może choćby rzucić okiem na produkty takie jak QLab [http://figure53.com/qlab/], EyeCon [http://eyecon.frieder-weiss.de], Isadora [http://troikatronix.com/isadora/about/], czy i-Cubex [https://infusionsystems.com] (dość świadomie nie zaliczam do tej kategorii środowiska MaxMSP/Jitter [http://cycling74.com] – nie jest ono na tyle sprofilowane, by jednoznacznie uznać je za przystosowane do wymagań scenicznych [choć należy zaznaczyć, że świetnie się w takich warunkach sprawdza], jest relatywnie tanie w zestawieniu z pozostałymi wymienionymi produktami, a poza tym stanowi swego rodzaju standard przemysłowy we współczesnej muzyce elektronicznej i sztuce mediów) – nie koniecznie po to, by je od razu kupić, ale po to, by przyjrzeć im się krytycznym (choć zarazem przychylnym) okiem i przez chwilę zastanowić się nad tym, co oferują i czego należy się po nich spodziewać.

Wiele z tych systemów dziedziczy sprzętowe koncepcje obecne w Arduino, czy RasberryPI – tak ma się np. z produktami w rodzaju i-Cubex, które zasadniczo służą sprzęganiu z komputerem zestawu sensorów dość podobnych do tych, które możemy obsłużyć za pomocą wspomnianego Arduino. i-Cubex jest oczywiście w o wiele


LA RUTA SHOW – Main Cue List
GO
80-2 - truck; drive loop
extended highway road ambiance
cab sounds, gear shifting, etc.
5 Cue Lists
6 Active Cues
Number
Q
Target
Pre Wait
Action
Post Wait
Time Remaining
40 vid 1 and countdown
40-1 vid 1
40-2 count 00:12.00
50 door mics
50-1 mic 7 to -3 00:00.00
50-2 mic 8 to -3 00:00.00
60 door mics out
60-1 mic 7 to -inf 50-1 00:05.00
60-2 mic 8 to -inf 50-2 00:05.00
70 close and lock cargo doors
80 truck start and go
80-1 truck; start and go 00:22.00
80-2 truck; drive loop 00:15.00 00:06.00
80-3 truck; cab interior drivi... 00:15.00 00:06.00
80-4 fade up truck; drive loop 80-2 00:15.00 00:06.50
80 - truck start and
80-1 - truck; 00:47.00
80-2 - truck; 01:51.00
80-3 - truck; cab 01:05.00
80-4 - fade up 00:08.50
80-5 - fade up 00:08.50
Basics
Time & Loops
Device & Levels
Trim
Audio Effects
Audio Output Patch
1 - Soundflower (64ch)
Default Levels
Set All Silent
Assign Cargo
Visible Channels
16
master
rear
stern left
stern right
ctr left
ctr right
bow left
bow right
cab
upa
sub
stern L
stern R
center L
center R
bow L
bow R
16
inputs
crosspoints
Edit
Show
406 cues in 5 lists

QLab

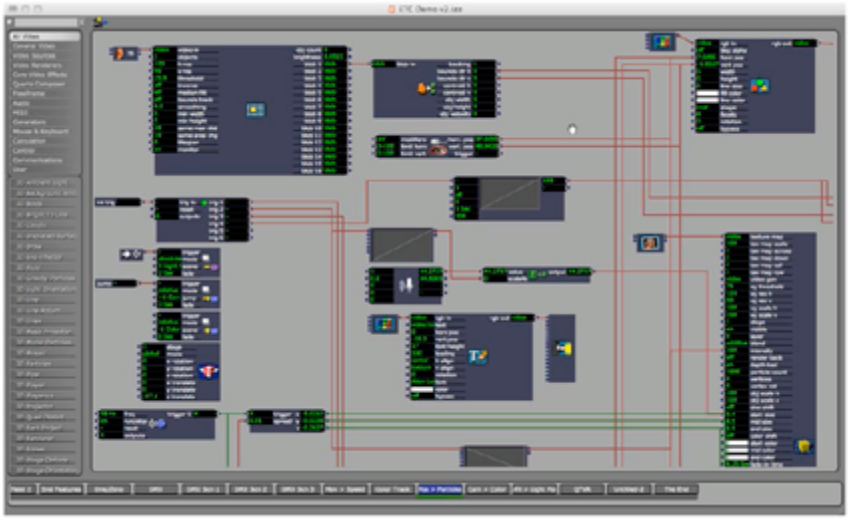

EyeCon
File PicMat Options Settings Info More
Notes
Controls
DirectSound
Mediaplayer
OverMore
Timebase
Threshold
Patch
Remark
Scale
Arpeggio
Chord
Edit Run Focus
Clear

EyeCon

większym stopniu gotowy: dostarczony z oprogramowaniem i okablowanymi sensorami (choć z drugiej strony w dużo mniejszym stopniu programowalny samodzielnie), niemniej jest to w dalszym ciągu znana Arduino koncepcja mikrokontrolera pośredniczącego w wymianie danych pomiędzy komputerem a sensorami.

Programy w rodzaju Isadora w dużej mierze przypominają swoją architekturą open source'owe środowiska w rodzaju Pure-Data [http://puredata.info] (lub jego komercyjnego krewniaka MaxMSP/Jitter [http://cycling74.com]) – posiadają modularną budowę, a poszczególne moduły łączymy wirtualnymi „kablami" informującymi w jaki sposób dane przepływają pomiędzy modułami realizującymi rozmaite funkcje. Jest to oczywiście bardzo elastyczna architektura, ale jeśli jesteśmy na początku drogi i dopiero uczymy się realizować interaktywne projekty sceniczne, to równie dobrze możemy zaznajomić się z programami o architekturze modularnej za pomocą darmowego Pure-Data, nawet jeśli jego interface nie wygląda tak dobrze jak Isadora lub brakuje nam jakiejś wysokopoziomowej funkcji, którą w Pure-Data musimy samodzielnie zbudować. Nie twierdzę przy tym, że Isadora jest niewarta swojej ceny – po prostu uważam, ze warto najpierw samemu zmierzyć się z zagadnieniem budowy systemów interaktywnych w takim stopniu, by móc określić własne potrzeby i sensowność zakupu takiego lub innego komercyjnego systemu.

Natomiast aplikacje takie jak QLab lub EyeCon stanowią bardzo ciekawe przykłady programów, których architektura została bardzo dokładnie dopasowana do charakterystyki typowego spektaklu teatralnego – łącznie z linią czasu podzieloną na sceny. Niewątpliwie



Isadora

pomaga to w zarządzaniu standardowymi projektami scenicznymi wykorzystującymi elementy medialne, jednak po raz kolejny przypomnę, że niniejszy podręcznik jest skierowany do nieco innej grupy odbiorców: osób o zacięciu eksperymentatorski, nie koniecznie zdecydowanych podążać dobrze wytyczonymi ścieżkami, ale raczej gotowych szukać własnych dróg i zwyczajnie lubiących eksperymenty.

i-Cubex

urządzenia mobilne

Z konieczności niniejszy podręcznik nie stanowi pełnego omówienia dostępnych technologii medialnych, a raczej sygnalizuje pewne obiecujące kierunki poszukiwań. Do takich „obiecujących kierunków" bez wątpienia należą coraz popularniejsze urządzenia mobilne w rodzaju smartfonów i tabletów (warto zauważyć, że jakiś czas temu laptop przestał być zaliczany do kategorii urządzeń mobilnych).

Większość współczesnych smartfonów i tabletów i posiada zestaw sensorów umożliwiających pomiar położenia, prędkości przemieszczania się, a często również i innych parametrów. W większości posiadają łącze bluetooth i możliwość korzystania z bezprzewodowej sieci. Tak więc – o ile tylko mamy możliwość pobrania i przesłania danych z sensorów wbudowanych w urządzenie mobilne do komputera – mogą one stan owić interesującą alternatywę nie tylko dla kontrolerów w rodzaju wiimote, ale i np. Arduino.

W tym konkretnym przypadku (tj. kiedy mówimy o wykorzystaniu urządzeń mobilnych jako sensorów) trudno podawać konkretne wskazówki (temat jest na to zbyt szeroki i zbyt szybko konkretne technologie i aplikacje mutują lub zanikają). Zasadniczo jednak niemal na pewno sprawdzi się w takim przypadku architektura systemu, która ciągle przewija się w tym podręczniku (można było o niej przeczytać przy okazji każdego niemal omawianego wyżej typu urządzenia mogącego służyć jako sensor): mianowicie zamiana danych z sensora na komunikaty OSC, transmisja (domyślnie bezprzewodowa) i odebranie ich w docelowym punkcie. W zależności od tego, jakim urządzeniem mobilnym dysponujemy (w szczególności: pod kontrolą jakiego systemu operacyjnego ono pracuje), możemy spróbować odnaleźć aplikację zamieniającą dane z sensorów wbu-

dowanych w urządzenie na wspomniane komunikaty OSC. Możemy też spróbować odpowiednią aplikację zbudować samodzielnie np. za pomocą Processingu [http://processing.org], lub OpenFrameworks [http://openframeworks.cc] – ta ostatnia możliwość jest jeszcze jednym argumentem za tym, że warto nauczyć się podstaw programowania.

oprogramowanie

Temat oprogramowania przewijał się już wcześniej – nie sposób omawiać zagadnień związanych z sensorium bez wspominania o oprogramowaniu. Spróbujemy jednak teraz zebrać rozproszone informacje w całość i uzupełnić o brakujące elementy.

Mówiąc w niniejszym podręczniku o oprogramowaniu na potrzeby interaktywnych systemów wykorzystywanych na scenie wspominałem jak do tej pory o dwóch rodzajach programów:

1. **Gotowe aplikacje realizujące jakąś konkretną funkcję.**

Do tej grupy zaliczają się m.in. wspomniane na początku podręcznika programy realizujące różne wersje systemów śledzenia ruchu opartych o kamerę (np. ActiveZones2OSC [http://wrocenter.pl/lab], JMyron2OSC [http://wrocenter.pl/lab], OpenTSPS [http://opentsps.com], KinectA [http://www.mihoo.de/kinecta.html]), lub programy umożliwiające dostęp do danych przesyłanych przez kontroler wiimote (np. DarwiinRemoteOSC [https://code.google.com/p/darwiinosc] i GlovePIE [http://glovepie.org/glovepie.php] – choć tej ostatniej aplikacji warto oddać sprawiedliwość i nadmienić, że posiada ona dużo większe możliwości, niż jedynie transmisja komunikatów z kontrolera).

Inne potencjalne aplikacje mieszczące się w tej grupie, to np. programy VJskie, które w większości dają się kontrolować w sposób interaktywny i jako takie włączać w aparatus systemu interaktywnego. Podobnie miałaby się rzecz z programami muzycznym, które, poza kontrolą za pomocą MIDI w coraz większym stopniu korzystają z OSC.

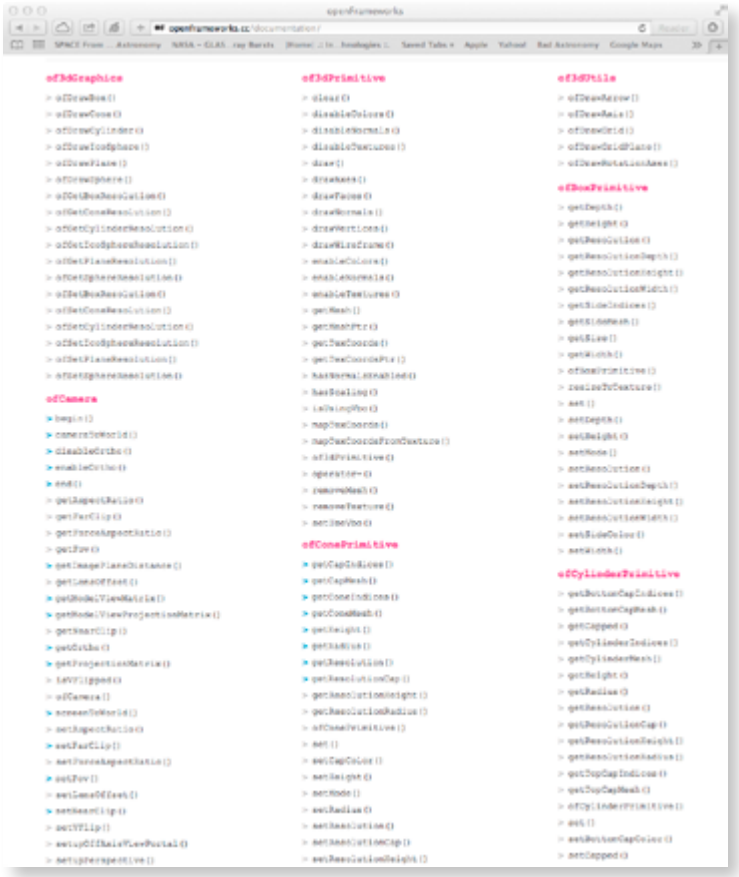
2. Środowiska programistyczne.

A więc narzędzia umożliwiające zarówno zastąpienie „gotowych” aplikacji realizujących pojedyncze zadania, jak i budowanie kompleksowych systemów zarządzających całością wymaganych zadań.

Do tej grupy można zaliczyć „klasyczne” (czyli wymagające pisania kodu) środowiska programistyczne w rodzaju Processing [http://processing.org], czy OpenFrameworks [http://openframeworks.cc], ale także środowiska modularne, nazywane czasem „wizualnymi językami programowania”, takie jak niekomercyjny Pure-Data [http://puredata.info] i – jedyny przewijający się w tym podręczniku komercyjny program – MaxMSP/Jitter [http://cycling74.com] (ten ostatni przypadek czyli MaxMSP/Jitter, wśród wielu nietypowych elementów zawiera mechanizmy pozwalające, poza wizualnym łączeniem modułów, na programowanie w kilku językach, w tym w Javie i pisanie shaderów GLSL). Podobnie do tej samej grupy można zaliczyć języki programowania dla kontrolerów Arduino [http://arduino.cc] i RasberryPI [http://raspberrypi.org].

W tym miejscu nasuwa się oczywiście kilka pytań: Czy lepiej stosować „tekstowe” języki programowania, czy środowiska wizualne? Korzystać z aplikacji realizujących poszczególne zadania i przesyłających wyniki swojej pracy do innych aplikacji, czy lepiej starać się integrować cały budowany przez nas system w ramach jednego programu komputerowego? Pozostać przy DIY i open source, czy zawierzyć komercyjnym rozwiązaniom?

Processing – klarowne i wydajne środowisko programistyczne przydatne zarówno jako narzędzie edukacyjne, jak i pełnoprawny język programowania



część dokumentacji OpenFrameworks – programowanie w C++, obsługa większości popularnych platform i systemow operacyjnych, wysoka wydajność np. w zastosowaniach związanych z 3D

Na te pytania nie ma jednoznacznej odpowiedzi. Dużo zależy od tego, w jaki sposób lubimy pracować i jaki charakter mają projekty, które chcemy realizować. Jednak istnieją dość uniwersalne reguły doboru oprogramowania, o których warto pamiętać:

3. **Unikaj kupowania czegoś, w stosunku do czego nie jesteś pewien, czy spełni Twoje oczekiwania.**

To oczywiście bardzo prosta reguła, sprawdzająca się w wielu dziedzinach życia. W przypadku oprogramowania na potrzeby systemów interaktywnych wykorzystywanych na scenie oznacza, m.in. że większość komercyjnych systemów ma swoje, mniej lub bardziej zubożone ekwiwalenty w postaci oprogramowania *open source* (to oznacza, że możemy np. przetestować za pomocą niekomercyjnego oprogramowania, czy dana architektura jest rzeczywiście tym, czego nam brakuje) – czasem zresztą rzecz ma się nieco inaczej, bo w niektórych przypadkach oprogramowanie *open source* jest na tyle dobre, że przerasta komercyjne odpowiedniki pod względem możliwości. Zaś w innych przypadkach może wcale nie będziemy potrzebować *najlepszego* programu dostępnego na rynku – może wystarczy nam taki, który spełnia nasze wymagania.

4. **Programy, których używasz powinny potrafić komunikować się z innymi programami.**

To bardzo ważna cecha współczesnego oprogramowania – niezależnie od roli, jaką aplikacja ma spełniać warto mieć możliwość sprzęgania jej z innymi aplikacjami lub środowiskami.

W interesującym nas kontekście absolutnym minimum dla większości aplikacji jest możliwość transmisji komunikatów MIDI i OSC (nie ma przy tym znaczenia, czy dany program należy do kategorii aplikacji wypełniających tylko jedno zadanie, czy też jest potężnym, wielozadaniowym środowiskiem). Użytkownicy komputerów z systemem OSX powinni zwrócić uwagę, czy aplikacja – jeśli pracuje z danymi wizualnymi – implementuje protokół Syphon (technologię współdzielenia danych video pomiędzy aplikacjami). Program kontrolujący wizualizacje powinien mieć możliwość pracy z danymi DMX (technologia kontroli świateł scenicznych) – ewentualnie, w razie braku takiej możliwości – powinien móc produkować komunikaty MIDI, które w miarę łatwo dają się skonwertować do formatu DMX.

5. Jeśli używasz zbyt wielu programów realizujących pojedyncze zadania w złożonym systemie, prostota może przerodzić się w chaos.

Pojedyncze aplikacje kontrolujące jakąś wybraną funkcję systemu i komunikujące się z innymi mogą stanowić nieocenioną pomoc w pracy (zwłaszcza, że stosunkowo łatwo znaleźć takie gotowe aplikacje, często dystrybuowane na nieodpłatnych licencjach), jednak, jeśli projekt zaczyna się rozrastać, to zbyt duża liczba takich „krasnoludków" realizujących różnorodne zadania może okazać się trudna do kontrolowania. W takim wypadku – czyli, jeśli czujemy, że techniczna strona projektu wymyka się nam spod kontroli ze względu na mnogość użytych programów – warto pomyśleć nad redukcją ilości komponentów w systemie poprzez wykorzystanie jakiegoś rozbudowanego narzędzia.

6. Warto znać podstawy programowania lub pracy w jakimś wizualnym środowisku.

Niezależnie od tego, czy nas to interesuje, czy nie i w jakim stopniu czujemy się nerdem warto poznać podstawy pracy w jakimś środowisku umożliwiającym pisanie prostych programów lub łączenie modułów w środowisku wizualnym. Komputer to zasadniczo urządzenie wykonujące programy – jeśli więc chcemy zrobić z technologią cyfrową coś kreatywnego, co nie jest jeszcze standardową funkcją gotowych i wykorzystywanych przez miliony innych osób systemów, powinniśmy napisać odpowiedni program…

Nauczenie się podstaw programowania nie jest, wbrew pozorom, aż tak trudne, jak się często wydaje – języki programowania sprofilowane pod kątem zastosowań około-artystycznych dają się w podstawowym zakresie rozgryźć bardzo szybko, jeszcze łatwiej przebiega nauka środowisk wizualnych.

Wybór konkretnego środowiska lub języka programowania jest sprawą indywidualną. W zależności od naszych preferencji może być to klasyczny, „tekstowy" język programowania, lub jakieś środowisko wizualne. Osobom bez żadnego przygotowania technicznego można polecić – jeśli mówimy o „tekstowych" językach programowania – Processing, który jest bardzo dobrze opisany i relatywnie łatwy w początkowym stadium nauki (autor poleca m.in. serię kursów wideo uczących podstaw pracy z Processingiem, w którego produkcję był „zamieszany" – kurs jest, oczywiście nieodpłatnie, dostępny na stronie: http://www.nina.gov.pl/kultura-2_0/sam-to-zrob/kurs-processingu), a – gdyby wybór padł najpopularniejsze

środowiska wizualne – Pure-Data lub MaxMSP/Jitter (w tym przypadku autor podręcznika odwoła się do kursu, który powstał m.in. w ramach jednej z poprzednich edycji MediaLabu: http://labkit.pl/podrecznik-do-puredata-i-maxmsp/)

Natomiast osoby mające pewne – choćby minimalne – doświadczenie w programowaniu mogą zapoznać się z opartymi na C++ frameworkami, takimi jak OpenFrameworks, lub Cinder [http://libcinfer.org].

protokoły komunikacyjne

Termin „protokół komunikacyjny” przewija się w niniejszym podręczniku dość często, najczęściej w odniesieniu do MIDI, OSC lub DMX. Jak nietrudno się domyślić lub wywnioskować z kontekstów, w jakich termin się pojawia ”protokół komunikacyjny” oznacza technologię pozwalającą programom i urządzeniom wymieniać dane w ujednoliconym formacie – innymi słowy komunikować się.

Umiejętność sprzęgania ze sobą aplikacji i środowisk składających się na nasz system interaktywny jest szalenie istotna – jeśli oprogramowanie, którego używamy, nie będzie się ze sobą komunikować, to, *de facto*, nie stworzy systemu. O ile techniczne detale związane z budową poszczególnych protokołów komunikacyjnych nie będą dla nas na początku bardzo istotne, o tyle warto wiedzieć, w jaki sposób dany protokół zastosować i w jakim kontekście.

W przeciwieństwie do gigantycznej ilości dostępnych aplikacji i środowisk programistycznych protokołów komunikacyjnych – przynajmniej tych, które interesują nas w kontekście potencjalnej użyteczności w projektach scenicznych – jest stosunkowo niewiele. Ma to sens, ponieważ to właśnie relatywnie niewielka ilość protokołów komunikacyjnych powinna pośredniczyć w wymianie danych pomiędzy niemal nieskończoną ilością aplikacji i środowisk programistycznych implementujących te protokoły.

Ponieważ, co zapewne jest już oczywiste, interesuje nas przede wszystkim triada MIDI, OSC i DMX przyjrzymy się bliżej właśnie tym trzem protokołom komunikacyjnym:

1. MIDI

MIDI (*Musical Instrument Digital Interface*) to protokół komunikacyjny opracowany pod kątem zastosowań okołomuzycznych (choć daje się też stosować w innych dziedzinach). MIDI ma za sobą relatywnie długą historię (specyfikacja protokołu ujrzała światło dzienne w 1982 r.) i jest mocno zakorzeniony w powszechnym myśleniu o muzycznych technologiach – praktycznie każdy produkowany współcześnie instrument elektroniczny korzysta z MIDI, podobnie rzecz ma się z urządzeniami studyjnymi: mikserami, procesorami efektów, itp. Oprócz samego standardu komunikacyjnego MIDI obejmuje specyfikację gniazd połączeniowych, parametrów elektrycznych urządzeń do transmisji, formatów zapisywanych plików, kolejności ułożenia programów brzmień w niektórych instrumentach, itp. – nie jest więc jedynie formatem danych, jak ma to miejsce w przypadku OpenSound Control.

MIDI nie transmituje sygnałów strumieniowych (nawet w formie zdigitalizowanej), jest więc protokołem „kontrolnym" – komunikaty MIDI mogą np. nieść informację „włącz nutę *C3* z głośnością 100 na kanale 2" – ale interpretacja tych danych, a w szczególności wygenerowanie konkretnego dźwięku, leży po stronie instrumentu (oczywiście może nim być program komputerowy), który komunikaty odbiera.

Komunikaty – i ogólnie: cała struktura protokołu – MIDI bazuje na pewnych ogólnych założeniach:

- komunikaty MIDI adresowane są do 16 odrębnych kanałów (*MIDI channels*) – kanały można potraktować jak wirtualne instrumenty

w obrębie jednego fizycznego urządzenia, ponieważ każdy kanał może być skonfigurowany niezależnie od pozostałych;

- komunikaty MIDI posiadają predefiniowane znaczenia – wśród najczęściej stosowanych warto wymienić *noteon/noteoff* (włączenie/wyłączenie wybranej nuty, z wybraną głośnością, na wybranym kanale) i *control change* (tzw. *CC* – komunikaty sterujące różnymi funkcjami instrumentu, najczęściej związanymi z filtracją i modulacją generowanego dźwięku), program change (wybranie określonego programu brzmieniowego na wskazanym kanale), komunikaty transportu (start, stop, itp.), komunikaty zegara (tzw. *MIDI clock* – pozwalają one synchronizować sprzęt studyjny), *sysex* (komunikaty specyficzne dla konkretnego urządzenia), itd.;

- porty MIDI składają się z trzech gniazd (lub ich wirtualnych odpowiedników): *in*, *out* i *thru* – o ile znaczenie nazw *in* i *out* jest oczywiste, warto zaznaczyć, że *thru* jest gniazdem, na które przekierowywane są informacje z gniazda *in*. Tak więc typowa ścieżka połączeń MIDI wędruje od wyjścia (gniazda *out* lub *thru*) jednego urządzenia do wejścia (*in*) kolejnego;

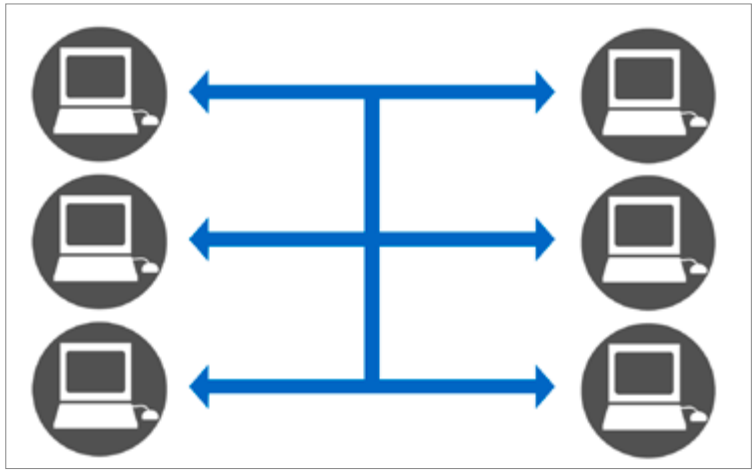
- od strony technicznej komunikaty MIDI składają się z 3 bajtów, pierwszy określa rodzaj komunikatu i kanał, drugi i trzeci bajt zmieniają znaczenie w zależności od pierwszego bajtu, np. dla komunikatu *noteon* będą wskazywały odpowiednio: wysokość i poziom głośności (zwykle nie musimy jednak analizować komunikatów MIDI na tym poziomie, bowiem aplikacje i środowisk implementujące protokół MIDI posiadają specjalizowane funkcje kodujące i dekodujące komunikaty);

- rozdzielczość bitowa i struktura komunikatów MIDI sprawiają, że większość parametrów dostępnych poprzez ten protokół przyjmuje wartości pomiędzy 0 a 127. Najważniejszym wyjątkiem od tej reguły jest parametr określający numer kanału, który przyjmuje wartości pomiędzy 1 a 16 (jednak nie wszystkie urządzenia MIDI obsługują jednocześnie wszystkie 16 kanałów).

2. OSC

OSC (*OpenSound Control*) to technologia, której prototyp opracowano w UC Berkeley Center for New Music and Audio Technology (CNMAT). Jest to protokół o – wbrew nazwie, sugerującej zastosowania wyłącznie na polu muzycznym – uniwersalnym zastosowaniu, korzystający z dowolnego typu łączy sieciowych (w zasadzie jest to protokół nadbudowany nad istniejącymi sieciowymi standardami komunikacyjnymi). W zamierzeniach OSC miał zastąpić MIDI, jednak zakorzenienie tego ostatniego w świecie muzycznym nie pozwoli mu jeszcze przez długi czas oddać pola młodszemu konkurentowi. OSC jednak okazał się na tyle uniwersalny, że znalazł zastosowanie w wielu dziedzinach i obecnie implementowany jest choćby przez NI Reaktor, EyesWeb, Chuck, SuperCollider, Jazzmutant Lemur, MaxMSP/Jitter, Pure-Data, Processing i wiele innych środowisk oraz urządzeń.

OSC nie jest protokołem przystosowanym do przesyłu strumieniowego (innymi słowy: raczej nie prześlemy nim obrazu lub dźwięku w czasie rzeczywistym). OSC zwykle transmituje dane poprzez standardowe protokoły sieciowe TCP/IP lub UDP.



Aby nadać informację musimy ją poprawnie zaadresować: na adres składają dwa parametry: numer IP odbiorcy i port, do którego zostaną przesłane dane. Odbiorca musi jedynie wiedzieć, z którego portu powinien odebrać informacje (jeśli określenia takie jak numer IP lub numer portu nic Ci jeszcze nie mówią nie musisz się niepokoić – są to, co prawda, dość podstawowe informacje dotyczące funkcjonowania sieci komputerowych, niemniej na tym etapie warto przede wszystkim wiedzieć, że numer IP identyfikuje urządzenie podłączone do sieci komunikacyjnej [np. do internetu], zaś numer portu, to elektroniczny odpowiednik skrzynki pocztowej – „miejsca", do którego trafiają przesłane informacje).

Komunikaty OSC formatowane są w następujący sposób:

/nazwakomunikatu argument1 argument2 ... argumentX

Ukośnik przed nazwą komunikatu nie jest wymagany ze względów technicznych – jest to – konwencja nazewnicza przyjęta przez twórców i użytkowników protokołu. W miarę możliwości warto ją respektować – ten nawyk pomoże nam w przyszłości np. tworzyć prace w kooperacji z innymi twórcami i przystosować nasze projekty do standardów, jakie przyjęli twórcy wielu pożytecznych aplikacji (np. systemów śledzenia ruchu i technologii odczytywania danych z zewnętrznych sensorów). Sama nazwa jest dowolna, może składać się z małych i dużych liter (OSC rozróżnia małe i duże litery), cyfr i znaków przystankowych (jakkolwiek przyjęta konwencja nazewnicza zakłada unikanie tego rodzaju elementów w nazwach komunikatów).

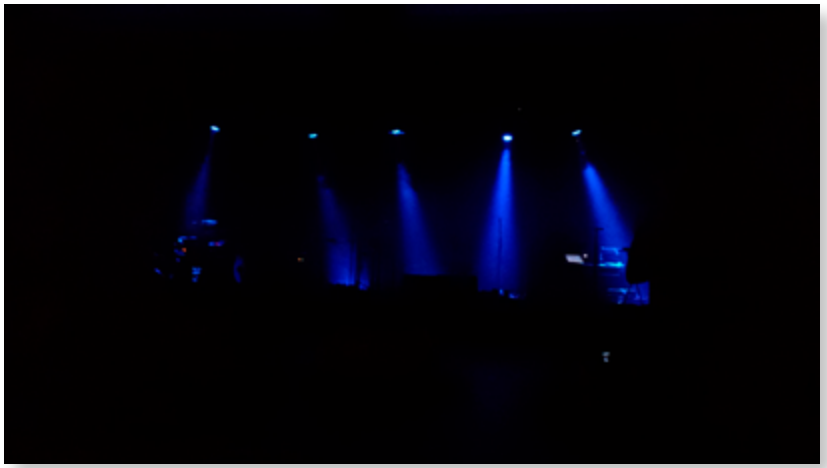
Komunikaty OSC mogą, choć nie muszą, zawierać parametry. Ilość parametrów jest w zasadzie dowolna, podobnie format danych (liczby całkowite i zmiennoprzecinkowe, łańcuchy tekstowe).

Praktyczne zastosowanie OSC wymaga pewnej praktyki. Jakkolwiek protokół jest bardzo sprawny, oferuje niskie opóźnienia (via UDP) i elastyczność w konstruowaniu komunikatów, trzeba też pamiętać o możliwych problemach: OSC jest silnie zależny od infrastruktury sieciowej (jeśli przesyłamy dane pomiędzy komputerami w różnych lokalizacjach, jakość łączy sieciowych ma wpływ na prędkość i stabilność transmisji), ponadto – dotyczy to zwłaszcza systemów operacyjnych z rodziny Windows – transmisje OSC mogą być blokowane przez oprogramowanie antywirusowe i/lub systemowy firewall.

Zaawansowani użytkownicy OSC mogą pokusić się o stosowanie kilku technik niejako zaszytych w protokole, w szczególności dotyczy to manipulowania tzw. *timestamps* – znacznikami identyfikującymi prawidłową pozycję komunikatu w funkcji czasu, co pozwala np. kompensować potencjalne opóźnienia transmisji i precyzyjnie synchronizować aplikacje i urządzenia.

3. DMX

Protokół DMX (oficjalna nazwa brzmi *DMX512*) ma nieco inne zastosowanie, niż MIDI i OSC – najogólniej mówiąc, służy kontroli oświetlenia scenicznego – innymi słowy: komunikaty wysyłane za pomocą tego protokołu określają przede wszystkim poziomy napięcia, jakim zasilane są lampy oświetlające scenę (za pomocą DMX można też



kontrolować inne parametry oświetlenia scenicznego, np. motory ruchomych głowic z lampami lub nieco bardziej egzotyczne urządzenia w rodzaju wytwornic dymu oraz wyrzutni konfetti…). DMX jest też protokołem jednokierunkowym (ma to oczywiście sens: w przeciwieństwie do MIDI i OSC urządzenia w rodzaju lamp, czy wytwornic dymu nie odsyłają komunikatów o swoim stanie). Protokół DMX został opracowany 1986 roku przez USITT (*United States Insitute for Theatre Technology*). Pod kątem *hardware'owym*, przesył danych za pośrednictwem protokołu DMX bazuje na standardzie szeregowego interfejsu RS485.

DMX prawdopodobnie nie jest technologią, którą wykorzystamy we wczesnych projektach, jednak warto poznać jej możliwości, choćby od strony teoretycznej, i zwracać uwagę, czy wykorzystywane przez nas oprogramowanie będzie potrafiło poradzić sobie z wysyłaniem komunikatów DMX – jako zamiennik DMX możemy też potraktować MIDI, ponieważ komunikaty MIDI dają się stosunkowo łatwo przekształcić – za pomocą odpowiednich konwerterów – na DMX.

11

zakończenie

Jeśli – choćby nawet pobieżnie – przebrnąłeś (lub przebrnęłaś) przez lekturę niniejszego mini-podręcznika, z pewnością dojdziesz do wniosku, że poruszany w nim temat, czyli przenoszenie doświadczeń wyniesionych z pracy z nowymi technologiami na pole związane z aktywnościami scenicznymi jest zagadnieniem zarazem prostym, jak i jednocześnie niebywale skomplikowanym.

W przeciwieństwie do potocznych intuicji i poglądów uważam, że samodzielne eksperymenty związane ze zmedializowanym performance rozpocząć jest relatywnie łatwo – współczesne narzędzia (zarówno oprogramowanie, jak i sprzęt) są na tyle powszechnie dostępne, że nie jest obecnie kłopotliwe zestawienie interaktywnego systemu pozwalającego stawiać pierwsze kroki i samodzielnie rozpoznawać możliwości łączenia ruchu, ciała i nowych technologii. Poważniejsze problemy pojawiają się dopiero na etapie przejścia od testów i „raczkowania" do czegoś, co ma być pełnowartościowym scenicznym działaniem.

Takie pełnowartościowe działania powstają przede wszystkim w rezultacie świadomego kreowania nie tylko warstwy scenicznej, ale i technologicznej i tu pojawia się dość typowy problem związany z niechęcią wielu twórców do bezpośredniego operowania technologią. Pośród wielu argumentów, jakie się przywołuje chcąc uzasadnić tego rodzaju fobie pojawia się m.in. taki, który odwołuje się do niemożności nabycia specjalistycznej wiedzy w dającym się przewidzieć (i zarządzać) czasie. Myślę, że zawartość tego podręcznika jednoznacznie ukazuje, że potrzebne informacje stricte techniczne w podstawowym wymiarze można przyswoić stosunkowo bezboleśnie i szybko, a następnie na tej bazie budować coraz bardziej

skomplikowane struktury (o ile potrzebujemy skomplikowania) unikając jednocześnie zbyt długiego okresu przygotowawczego (czyli takiego, w którym nie możemy jeszcze ujrzeć żadnych praktycznych rezultatów naszej aktywności) i związanej z tym frustracji.

Na koniec życzę czytelnikom aby lektura podręcznika okazała się zarówno przyjemna, jak i owocna.

Narodowy Instytut Audiowizualny
był partnerem Medialabu ‘2013

Dofinansowano ze środków Ministra Kultury i Dziedzictwa Narodowego

ORGANIZATOR

FUNDACJA ORTUS

WSPÓŁORGANIZATOR

CENTRUM CYFROWE
projekt:polska

PATRONAT MEDIALNY

ngo.pl

PARTNERZY

Ministerstwo Kultury i Dziedzictwa Narodowego

UNIWERSYTET OPOLSKI

Instytut Sztuki Uniwersytet Opolski

NARODOWY INSTYTUT AUDIOWIZUALNY

Gamfi
www.gamfi.pl