# The Kinect distance sensor as human-machine-interface in audio-visual art projects

Matthias Kronlachner

PROJECTREPORT

Advisor:

DI IOhannes m Zmoelnig

Institute of Electronic Music and Acoustics
University of Music and Performing Arts Graz

Electrical Engineering - Audio Engineering

Graz, January 2013

kunst uni graz

institute of electronic music and acoustics

## Abstract

For several years now, the entertainment and gaming industry has been providing multifunctional and cheap human interface devices which can be used for artistic applications.

Since November 2010 a sensor called *Kinect™* for Microsoft's *XBox 360™* is available. This input device is used as color camera, microphone array, and provides as an industry-first, a depth image camera at an affordable price.

As part of the project Pure Data/Gem externals have been developed which allow access to the video and audio streams of the Kinect sensor. By including a framework for *Natural Interaction* (OpenNI), methods can be used for user identification or the extraction of skeletal models from the depth image.

The video streams and motion data produced by the sensor, in combination with the software, is used in multimedia art projects for controlling sounds and video projections. The documentation of these art projects and examples of the usage concerning the developed Pure Data externals make up the practical part of this project.

## Kurzfassung

Die Unterhaltungs- und Spieleindustrie liefert schon seit einigen Jahren multifunktionale und günstige Eingabegeräte welche für künstlerische Anwendungen gebraucht werden können und werden. Seit November 2010 ist der für die Microsoft Spielekonsole *XBox 360™* entwickelte Sensor *Kinect™* erhältlich. Dieser Sensor dient als Farbkamera, Mikrofonarray und liefert zum ersten Mal eine Tiefenbildkamera zu einem erschwinglichen Preis.

Im Rahmen der Projektarbeit werden Pure Data/Gem Externals entwickelt welche den Zugriff auf die Video und Audio Streams des Kinect Sensors ermöglichen. Durch die Einbindung eines Frameworks für *Natural Interaction* (OpenNI) können Verfahren wie Personenerkennung und die echtzeitfähige Extraktion von Skelettmodellen aus dem Tiefenbild verwendet werden.

Die vom Sensor gewonnen Videostreams und Bewegungsdaten werden in künstlerischen Projekten zur Steuerung von Klängen und Videoprojektionen eingesetzt. Die Dokumentation dieser Kunstprojekte sowie die Beispiele zur Anwendung der Pure Data Externals bilden den praktischen Teil dieser Projektarbeit.

# Contents

# 1 Introduction

For several years now, the entertainment and gaming industry has been providing multifunctional and cheap human interface devices which can be used for artistic applications.

Since November 2010 a sensor called Kinect™ for Microsoft's Xbox 360 is available. This input device is used as color camera, microphone array, and provides - as an industry-first - a depth image camera at an affordable price. Soon after the release of the Xbox-only device, programmers from all over the world provided solutions to access the data from a multitude of operating systems running on ordinary computers.

This work outlines the functionality of a low budget sensor device and it's application within Pure Data[1].

Multiple Pure Data/Gem externals[2] and application examples are introduced which allow the full access to the functionality of the Kinect sensor including video (Fig. 1.1) and audio streams. Examples, source code and precompiled binaries can be found at the Authors' GitHub repository [Kro12a].

So far no special colorspace is existent in Gem that describes the distance of every pixel in relation to the camera. Therefore, possible solutions are presented to interpret and work with depth pixel data.

By integrating a framework for "natural interaction" (OpenNI), it is possible to do user identification or extract skeletal models from the depth image.

Other solutions exist to acquire skeleton data, for example OSCeleton[3]. However, the presented externals enable the access to video streams and tracking data *simultaneously inside* Pure Data without the need of additional computer programs. This allows maximum flexibility.

The documentation of the externals include usage examples like distance measurement, outline extraction, background subtraction, hand, user and skeleton tracking, head pose estimation as well as gathering audio streams from the 4 channel microphone array.

For showing the practical application in art projects, the piece *übersetzen - vertimas*[Kro12b] for dancer, sound and video projection (Fig. 1.2) as well as the usage for a concert with the IEM Computer Music Ensemble is presented.

The advantage of using Kinect for projects include independence from visible light due to operation in the infrared spectrum and no need for calibration to gather movement data of people.

[1] Pure Data is a visual programming environment used for computermusic and interactive multimedia applications. Gem stands for *Graphics Environment for Multimedia* and extends Pure Data to do realtime OpenGL based visualizations.

[2] Externals provide additional object classes to extend the functionality of Pure Data.

[3] OSCeleton is a standalone application to gather skeleton data from OpenNI/NITE framework and sends it over Open Sound Control (OSC) to a host.

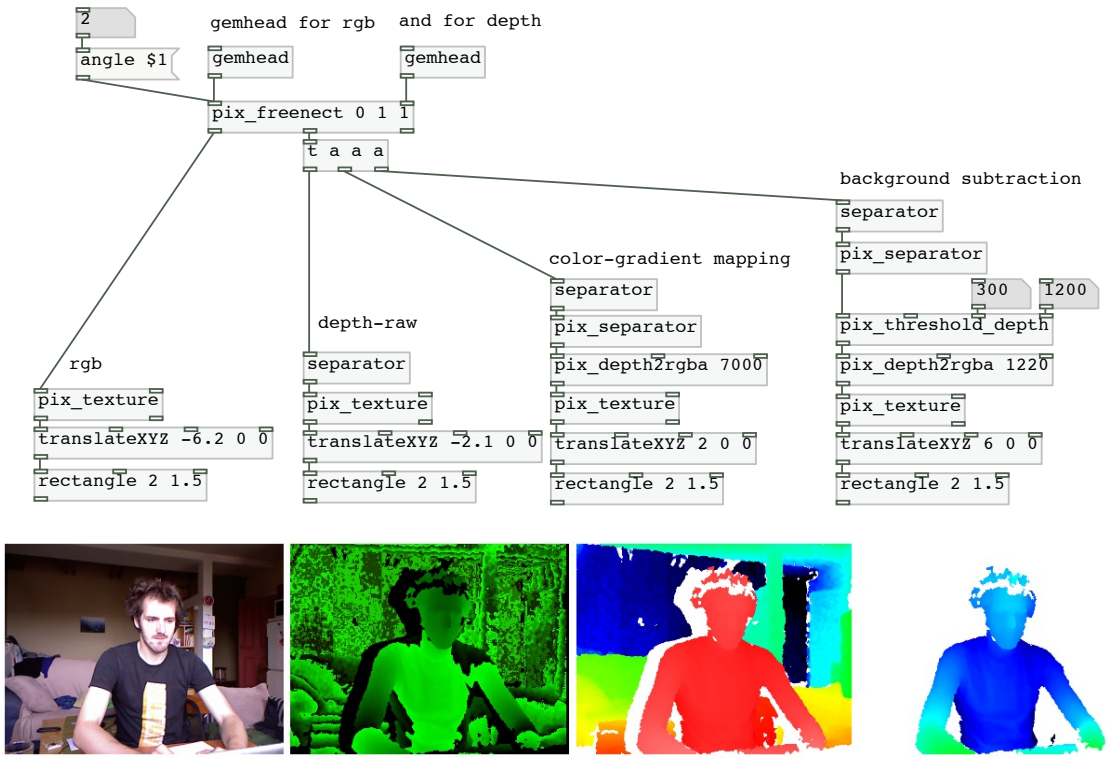



**Figure 1.1:** Kinect videostreams within Pure Data

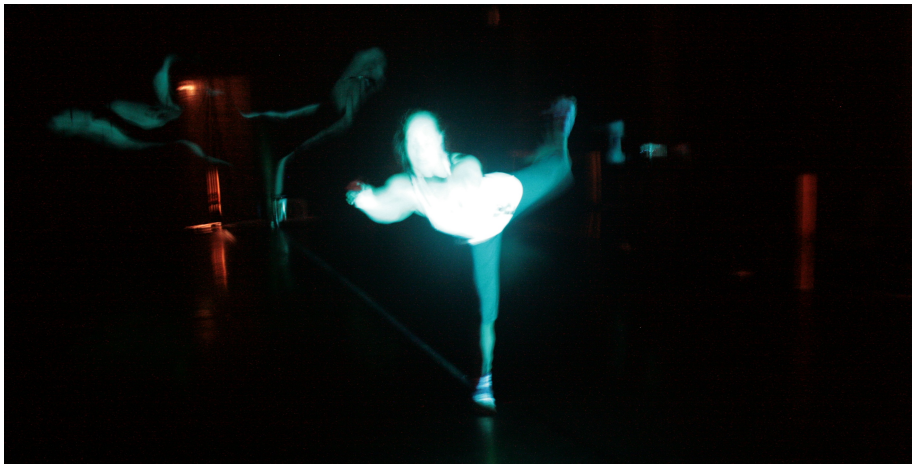


**Figure 1.2:** Bodyprojection with help of the Kinect

# 2 Kinect specifications

The Israeli company Primesense developed the technology for the Kinect device and licensed it to Microsoft. The Kinect sensor is connected by USB[1] 2.0 with additional power supply and features a standard RGB webcam, a depth sensor and a four channel microphone array (Fig. 2.1). It's head can be tilted $\pm 27°$, a three axis accelerometer is measuring the orientation and a three color LED may be used for visual feedback.

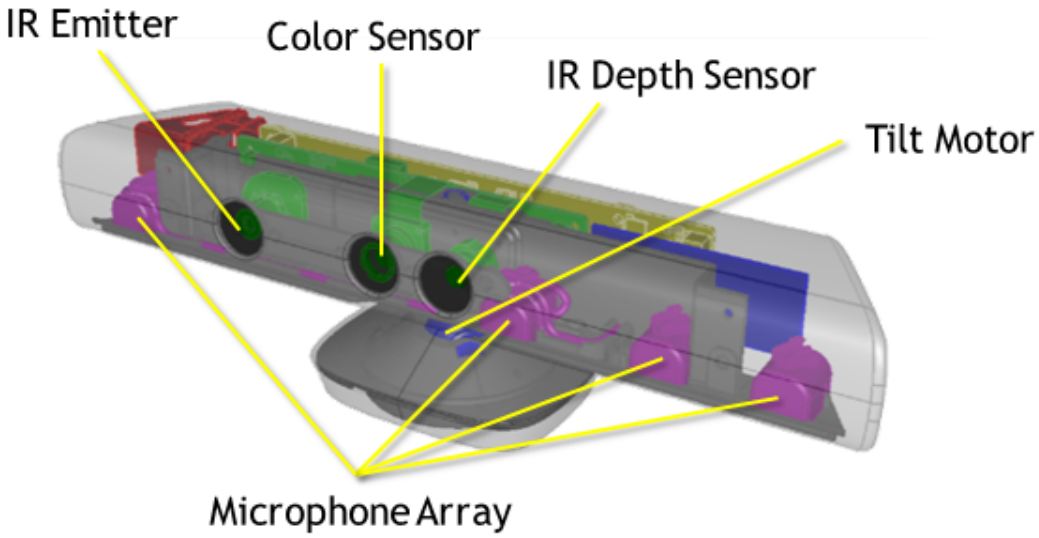



**Figure 2.1:** Position of the sensors [Mic12]

The block diagram of the reference design is shown in Fig. 2.2. The Primesense PS1080 System-on-Chip (SoC) provides a synchronized depth image, color image and audio streams. All depth acquisition algorithms are running on the PS1080 SoC, therefore no computational load is added to the host. [Pri12] Higher level functions like scene analysis and tracking have to be done on the host.

A similar sensor device using the Primesense reference design is available from Asus (Fig. 2.3). The *Xtion* and *Xtion Live* from Asus do not have a motor for controlling the tilt. Therefore they do not need an additional power adapter and can be operated from the USB power supply. The Xtion series includes a two channel microphone array instead of the four channels provided by the Kinect sensor. The *Xtion* is equipted without RGB camera, the *Xtion Live* includes an RGB camera like the Kinect.

[1] Universal Serial Bus

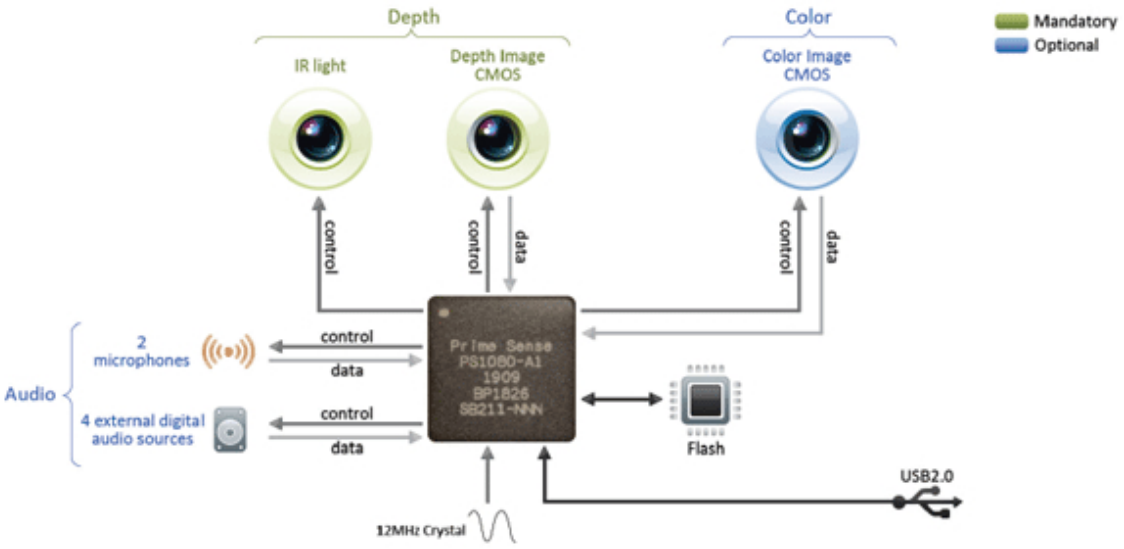



**Figure 2.2:** Primesense reference design [Pri12]


**Figure 2.3:** Asus Xtion Live sensor

The following sections will describe the different sensor elements of the Kinect in detail.

## 2.1 RGB camera

The RGB camera of the Kinect features a standard resolution of 640x480 pixels operating at a framerate of 30 Hz. A high resolution mode offering 1280x1024 pixels can be used. But the framerate drops to about 15 Hz when using the high resolution mode. The native output of the RGB camera is encoded as Bayer pattern[2] image, but the available frameworks (Sec. 3) can convert the raw information to a standard RGB image.

[2] A Bayer filter is a color filter mosaic enabling an array of monochrome image sensors to record color images.

The pipeline of the RGB stream can also be used to output the raw stream of the infrared (IR) camera. Unfortunately the RGB stream and the IR stream can not be used in parallel.

## 2.2 Depth sensor

The depth sensor consists of a 830nm wavelength infrared laser projecting a specific dot pattern onto it's field of view (Fig. 2.4). An infrared camera records these patterns on the objects and an on-board Digital Signal Processor (DSP) computes the distance by correlating the live image with stored reference patterns (Fig. 2.5). More information about the *Light Coding™* technologie can be found in Patent *US 2010/0118123 A1*[FSMA10] and the PrimeSense Homepage[Pri12].

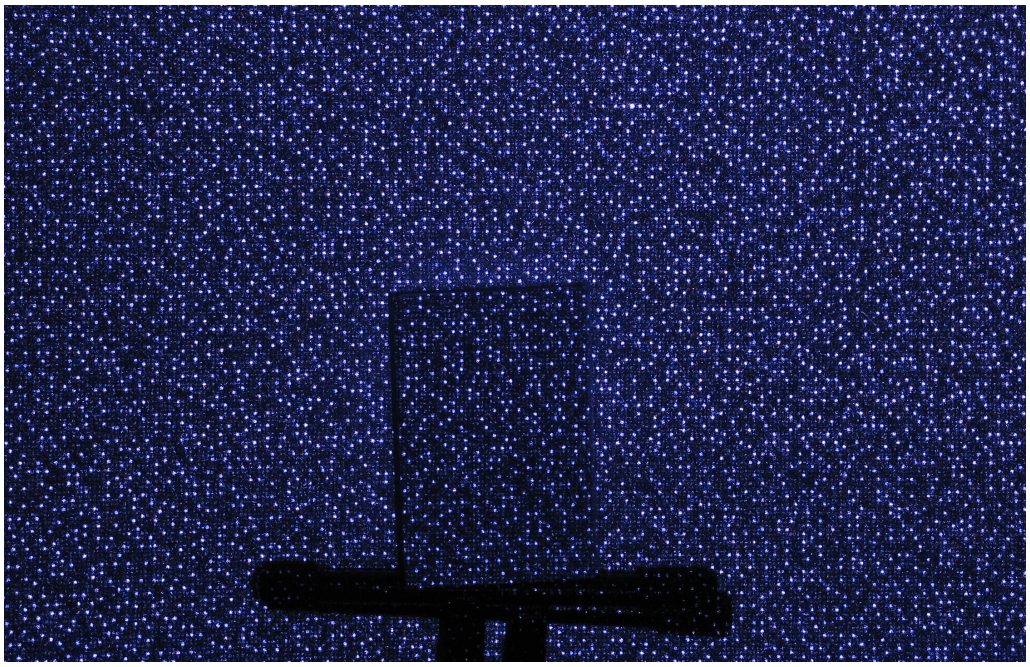


**Figure 2.4:** Infrared dot pattern on wall [fut10]

The output of the depth sensor is a 640x480 pixel video stream. Each pixel is holding 11 bit of depth information, therefore 2048 ($2^{11}$) different values for representing distance are possible. Due to internal computational reasons the rightmost eight pixel columns in the image do not contain data. Therefore the usable resolution is reduced to 632x480.

The libraries for accessing the Kinect depth stream (Sec. 3) support converting the 11 bit raw depth values to real world coordinates in millimeter.

Due to the horizontal displacement of the infrared projector and the infrared camera, the projector and the camera do not share exactly the same field of view. Therefore a shadow especially for objects near to the camera is visible (Fig. 1.1, second and third image from the left).

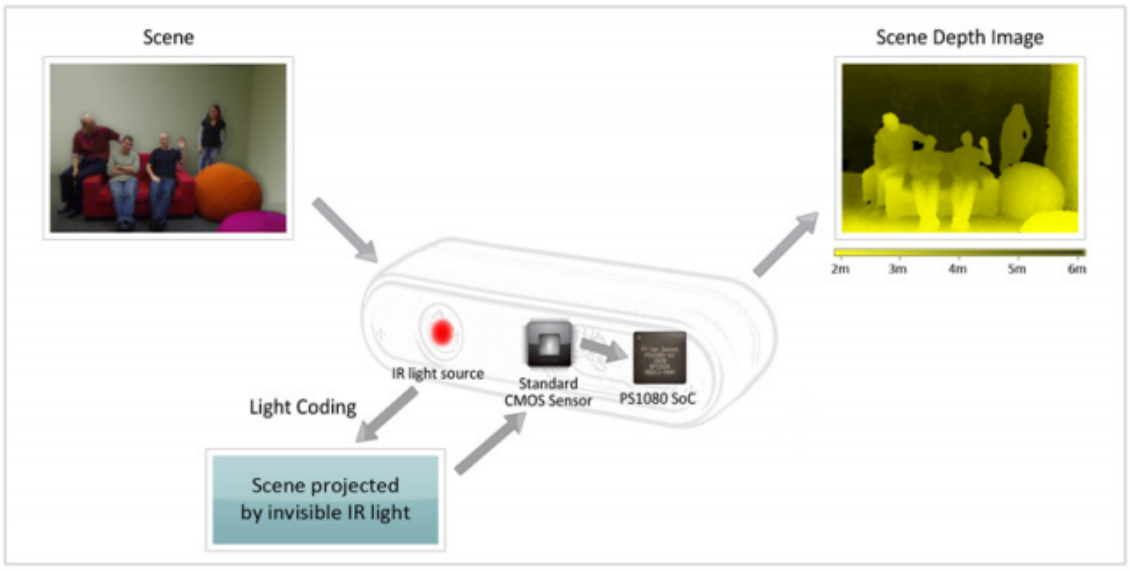



**Figure 2.5:** Depth sensor flow chart [Pri12]

### Range of the depth sensors

The range of the Kinect depth sensor covers approximately 0.7 to 7 meters. The optimum range is given by the manufacturer from 1.2 to 3.5 meters. The field of view covers $58°$ horizontal, $45°$ vertical and $70°$ diagonal (Fig. 2.6).

### Latency of the depth sensor

As A. Bernin [[Ber11]Sec. 6.2] has pointed out in his master thesis, the average latency of the depth sensor is 72.98 ms (Fig. 2.7). This latency indicates about 2 frames delay at a framerate of 30 images per second. ($t = \frac{2}{30[Hz]} = 66.6[ms]$) This value does not include the computation of higher level functions out of the depth video. It was measured with a resolution of 640x480 pixels and a framerate of 30 frames per second.

### Using multiple Kinect devices

Using multiple Kinect devices onto the same scene can cause unrecognized regions in the depth image due to overlapping infrared dot patterns.

A possible solution by adding independent motion to each of the sensors is proposed in [MF12]. Applying vibration to the Kinect senors results in a blurry dot pattern of the interfering Kinect sensors. This can be done by attaching a motor with an eccentric mass to the bottom of the Kinect (Fig. 2.8). The rotation of the motor and its attached mass induces a tiny vibration in the device. As the infrared laser and camera are attached to the same housing, they see their own pattern clearly and undistorted. The depth value can be estimated and recognition is not disturbed by other Kinect sensors. (Fig. 2.9)

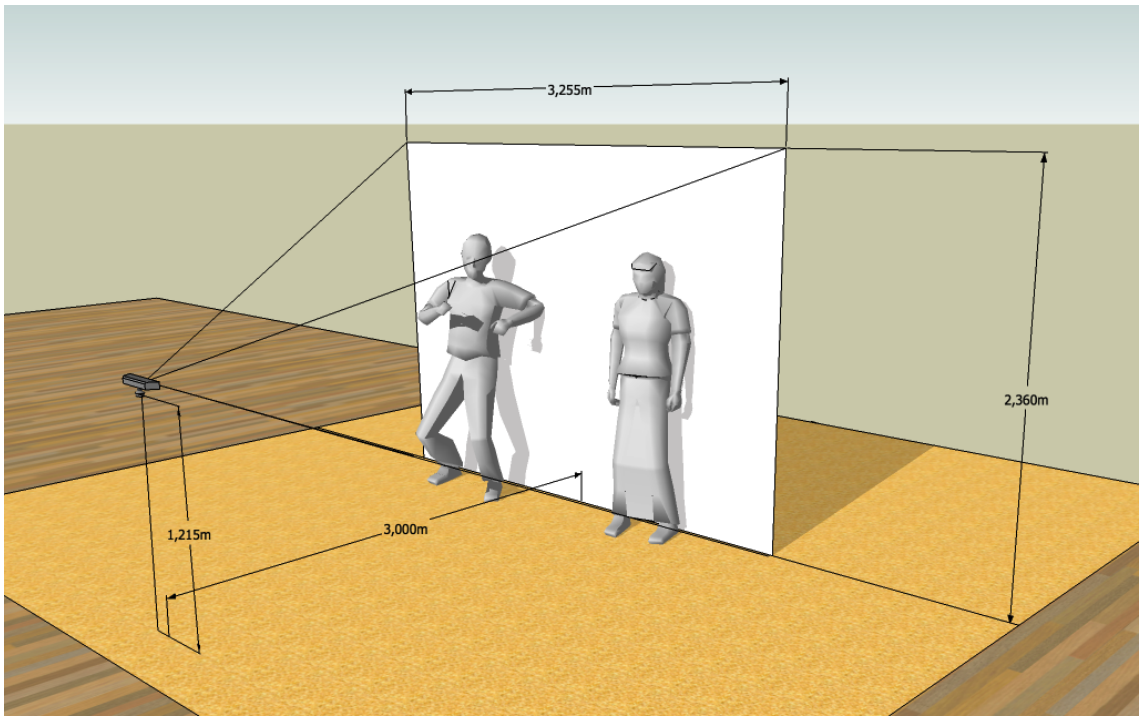



**Figure 2.6:** Depth sensor field of view $58^\circ$ H, $45^\circ$ V, $70^\circ$ D

Apart from overlapping patterns, the USB 2.0 bandwidth has to be considered when using multiple Kinect devices with one computer. Tests showed that two Kinect sensors with activated RGB and depth stream occupy one USB bus. Therefore, additional USB controllers have to be added to the system for operating more than two Kinect sensors.

## 2.3 Microphone array

The Kinect sensor includes a 4 channel microphone array with ADCs[3] running at 16 kHz sampling rate and 24 bit resolution.

Due to its rather low sampling rate and therefore about 8kHz cut off frequency it is mainly intended for recording speech. Basic sound source localization and beam forming algorithms can be used to separate multiple speakers in front of the Kinect sensor. However, those algorithms are not provided by the device itself.

The four omnidirection electret microphones are located behind a grill on the bottom of the sensor bar. (Fig. 2.10)

Following the Primesense reference design (Fig. 2.2), the Kinect supports receiving a four channel audio stream for performing echo cancellation. In practice, the (surround) loudspeaker feeds for the room are sent to the Kinect. The PS1080 SoC subtracts the loudspeaker signals from the microphone signal which results in a better signal-to-noise

[3]ADC - Analog to digital converter

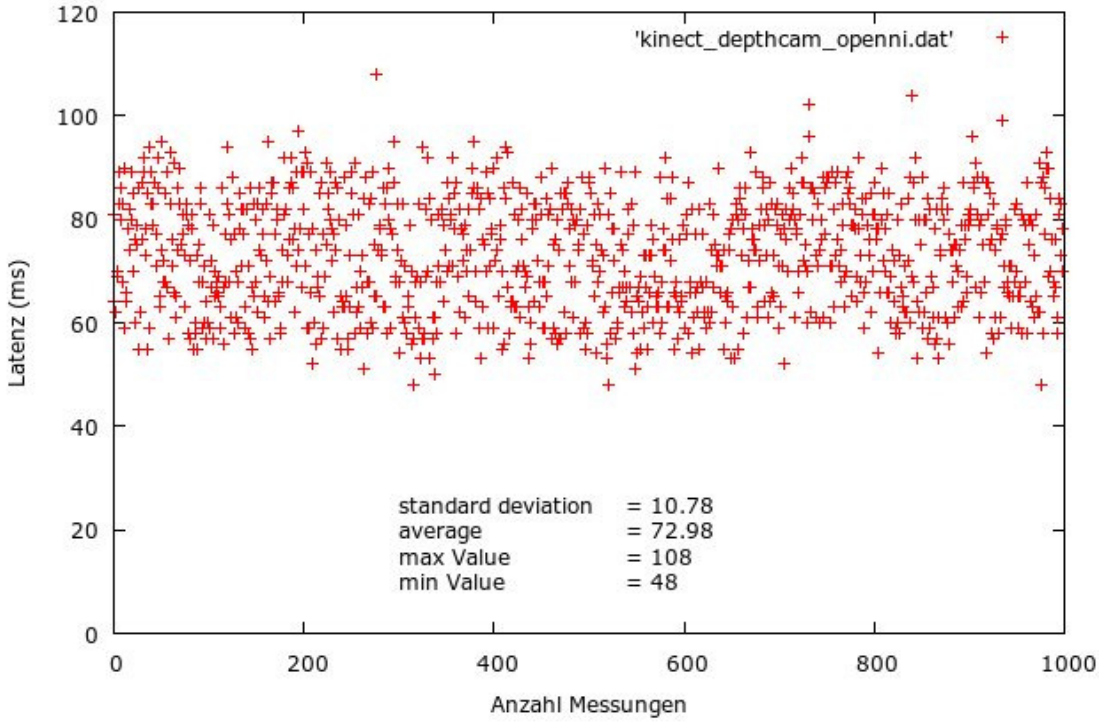



**Figure 2.7:** Latency of Kinect depth sensor using OpenNI [[Ber11]Sec. 6.2]


**Figure 2.8:** DC motor with eccentric mass attached to the Kinect for inducing vibration. [MF12]

ratio.

In the project report *Audio for Computer Gaming* [MM12], Mayer and Meißnitzer took a closer look at the Kinect microphone array including measurements of the impulse responses and beam accuracy using the Microsoft Kinect SDK (Sec. 3.3).


**Figure 2.9:** **A** - Depth map of room using a single Kinect. **B** - Depth map with 5 Kinect sensors overlapping the view, causing interferences and failures in estimation. **C** - Depth map with 5 overlapping Kinects while applying motion to every sensor. [MF12]

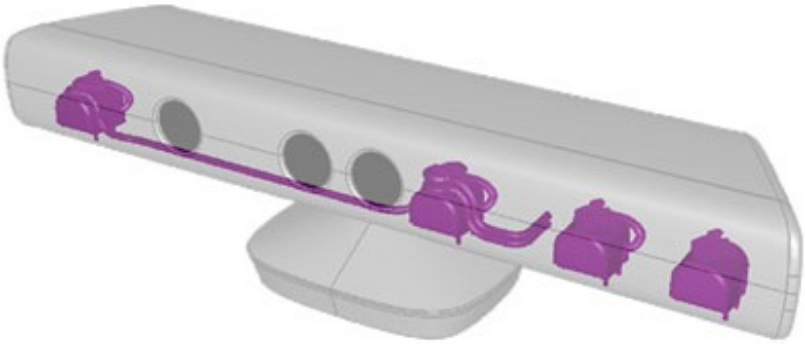


**Figure 2.10:** Four channel microphone array

# 3 Accessing the Kinect

Soon after the release of the Xbox-only device in November 2010, programmers from all over the world provided solutions to access the data from a multitude of operating systems running on ordinary computers. In the beginning, Microsoft did not support the movement of using the Kinect for other purposes. Also the license does not allow using the Xbox Kinect for other purposes than playing with the Microsoft gaming console. After the big success in the Open-Source community, Microsoft changed their policy and released their own Software Development Kit (SDK) for Windows. They released another version of the Kinect sensor called *Kinect for Windows* in January 2012. This version includes minor hardware changes (a shorter USB cable), but has a more open license.

The following chapter should introduce the available libraries and their different functionality for accessing Kinect data streams at application developer level. An overview is given in table 3.1.

## 3.1 libfreenect

The OpenKinect community released the multi-platform open source library libfreenect [Ope11a], allowing access to all data streams of the Kinect. Libfreenect supports multiple

| | libfreenect | OpenNI / NiTE | MS Kinect SDK |
|---|---|---|---|
| rgb | ✓ | ✓ | ✓ |
| depth | ✓ | ✓ | ✓ |
| infrared | ✓ | ✓ | ✓ |
| audio | ✓[a] | | ✓ |
| accelerometer | ✓ | | ✓ |
| led | ✓ | | ✓ |
| motor tilt | ✓ | | ✓ |
| scene analysis | | ✓ | ✓ |
| hand tracking | | ✓ | ✓ |
| skeleton tracking | | ✓ | ✓ |
| face tracking | | | ✓ |
| beam forming | | | ✓ |
| sound source localization | | | ✓ |

[a] libfreenect audio support only for Linux

**Table 3.1:** Comparison of Kinect libraries

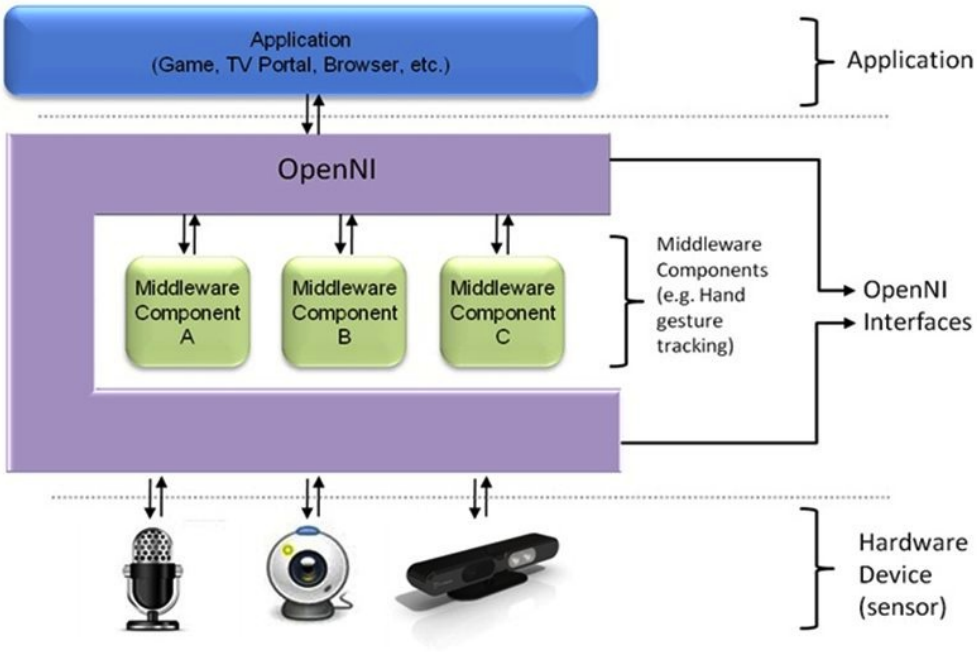



**Figure 3.1:** OpenNI layers [Ope11b]

Kinect devices and allows to identify them by there unique serial number[1].

No higher level functions like Scene Analysis and Tracking are included, but there is the separate open source project *Skeltrack* [Roc12], focusing on skeletal tracking.

Libfreenect is very lightweight, cross-plattform and easy to install. Therefore, it is the library of choice for tasks that do not need built in tracking functionality. The library should also work with the Asus Xtion Series (Fig. 2.3), but this has not been tested by the authors.

## 3.2 OpenNI / NiTE

Primesense, the company that developed the technology for Kinect, founded the non-profit organization OpenNI to support the interoperability of *Natural Interaction* devices. OpenNI is an cross-platform open source framework allowing applications to interface with different sensor devices and accessing higher level functions (called middelware) like skeleton tracking.

The structure of OpenNI covers three layers (Fig. 3.1). The first is the application layer providing the functionality of OpenNI to host applications. The second layer acts as communication layer between the host software, (multiple) hardware sensor input and (multiple) middleware, which analysis data from the sensor. The bottom layer shows different hardware devices attached to the OpenNI framework by specific drivers. [Ope11b]

[1] A Kinect serial number may look like this: A00362807917045A

As OpenNI relies on hardware driver modules for connecting to a specific device, an unofficial driver module *avin2-SensorKinect*[avi10] has to be installed for using the framework with the Kinect sensor. This driver currently does not support audio, accelerometer, led and motor tilt access.

OpenNI features recording and playing back data streams of an attached sensor device. This can be very handy for rehearsal and development situations.

### NiTE

The higher level functionality in OpenNI has to be provided by the so called *middleware*. An example implementation of such a middleware is NiTE by Primesense and provides gathering of the position of numerous standing people, tracking of a detailed skeleton of two people as well as performing hand gesture detection and tracking. NiTE is closed-source and available as precompiled binary for Windows, Mac OS X and Ubuntu Linux from the OpenNI Homepage [Ope11b].

## 3.3 Microsoft Kinect SDK

In June 2011 Microsoft released their Kinect Software Development Kit (SDK) [Mic12]. This Windows only SDK has the most complete feature set for interfacing with the Kinect sensor. It allows position estimation of up to six standing people and extracting the detailed skeleton of two people. It features higher level functions for the microphone array like sound source localization, beam forming and speech recognition.

The Microsoft Kinect SDK has not been used in this project due to its non-cross-plattform compatibility which is a major point when working with Pure Data.

# 4 Pure Data / GEM externals

The need for custom Kinect Pure Data/GEM externals came out of the wish to access all streams and functionality of the Kinect in parallel within one piece of software.

Two different open-source and multi-plattform libraries are used for interconnecting with a Kinect device. Based on libfreenect (Sec. 3.1) `pix_freenect` (Sec. 4.2) offers access to all features provided by the Kinect sensor. The OpenNI (Sec. 3.2) interface for Pure Data is implemented in `pix_openni` (Sec. 4.5). This external does *not* provide access to the motor tilt, accelerometer, LED and the audio streams. But it features higher level functionality like skeleton and hand tracking.

To incorporate the missing functions of the OpenNI based external, specific externals based on libfreenect have been developed to access the accelerometer, motor tilt, LED and audio streams (`freenect`, `freenect_audio`). These externals can be used in parallel to `pix_openni`.

The externals `expr` and `expr~` offer a very flexible way for manipulating float messages or audio samples by entering mathematical formulas. Currently there is no `pix_expr` available, which makes it difficult to operate with color values in a flexible way. Therefore the externals `pix_threshold_depth` (Sec. 4.6) and `pix_depth2rgba` (Sec. 4.1.1) for manipulating depth streams have been developed.

The source code as well as precompiled binaries of all developed externals can be found at the GitHub repository [Kro12a].

## 4.1 Representation of depth data

So far no Gem colorspace exists which describes the distance of a pixel in relation to the camera. Therefore a solution is proposed using RGBA or YUV colorspace for representing 16 bit depth data. For RGBA output the 16 bit depth data is divided into the upper eight most significant bits and the lower eight significant bits. These eight bit values are stored in the red (R) and the green (G) channel respectively. The blue channel (B) is used for additional information about the pixel. For example, if a user is present in that specific pixel, the specific user-id is encoded into the blue (B) channel. The alpha channel (A) is set to 255.

The YUV colorspace uses 4 bytes per 2 pixels and therefore can store the 16 bit per pixel depth information, but additional values like user-ids can not be included. This output mode was implemented for saving main memory and computation time, but currently no additional externals are provided to manipulate YUV depth maps. Therefore it is recommended to stick to the RGBA depth output.

Depending on the numerical representation of color values, depth information can be

| R | G | B | A |
|---|---|---|---|
| 3/8 msb | 8 lsb | 0 or userid (OpenNI) | 255 |

**Table 4.1:** RGBA output of depth data

| YUV422 (2 bytes per pixel) |
|---|
| 11 bit/16bit depth values |

**Table 4.2:** YUV output of depth data

obtained with the following formulas[1].

$$distance = R_{int} * 2^8 + G_{int} \tag{4.1}$$

$$distance = R_{float} * 2^{16} + G_{float} * 2^8 \tag{4.2}$$

RGB data for one specific pixel can be obtained by pix_data. An example how to compute the distance value of one pixel within Gem is shown in Fig. 4.1.

To extract certain parts of an depth image you can use pix_threshold_depth (Sec. 4.6).

### 4.1.1 pix_depth2rgba

For development and visualization purposes it is handy to map distance values onto a color gradient (Fig. 4.2). For this purpose the external pix_depth2rgba (Fig. 4.3) has been developed. The creation argument defines the maximum range of the gradient in *millimeter*. This value can also be changed by the rightmost inlet.

The second inlet allows to change the internal mapping mode from 16 bit depth data in [mm] (default) to 11 bit raw depth data. This is just needed if the depth output of pix_freenect is set to raw mode.

Sending zero or one to the first inlet deactivates or activates the conversation.

pix_depth2rgba changes the pix buffer of the image. If the *unchanged* image is needed for other purposes in parallel, the buffer has to be copied using pix_separator (Fig. 1.1).

## 4.2 pix_freenect

The external pix_freenect is based on *libfreenect* (Sec. 3.1) and offers access to RGB and depth streams (Fig. 1.1). It allows gathering of accelerometer data, controlling the

[1] Gem internally handles color values as 8 bit integers (0-255), but on user level (eg. output of pix_data) normalized floats (0.0-1.0) are used.

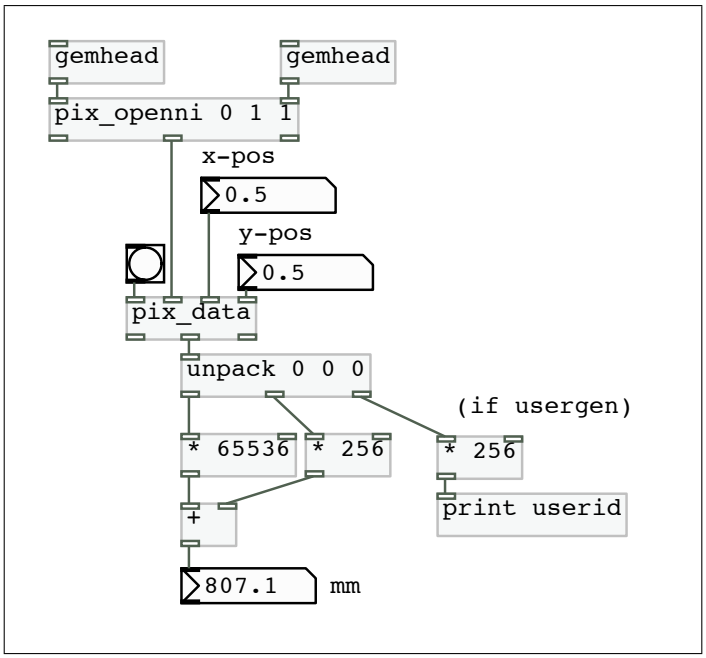



**Figure 4.1:** Computing distance out of depth image using `pix_data`.

**Figure 4.2:** Gradient for displaying depth data - near to far

tilt of the head and changing the LED color. Multiple devices can be accessed by their unique serial number. Due to the limited bandwidth of USB 2.0 a maximum number of two Kinects can be accessed by one USB controller.

### 4.2.1 Creation Arguments

The creation arguments of [pix_freenect] allow to turn on/off streams and select a specific Kinect device, if multiple are used.

`pix_freenect <device id/serial number> <rgb on/off> <depth on/off>.`

Kinect device ids are assigned by the libfreenect library and start with zero. When using multiple Kinect devices its better to address them by their unique serial number (Fig. 4.5). Libfreenect does not guarantee to assign the same id to the same device when being restarted. The Kinect serial numbers can be listed by sending the [info( message to [pix_freenect] (Fig. 4.5). This will print the serial numbers of all Kinect devices to the Pure Data terminal window (List. 4.1).

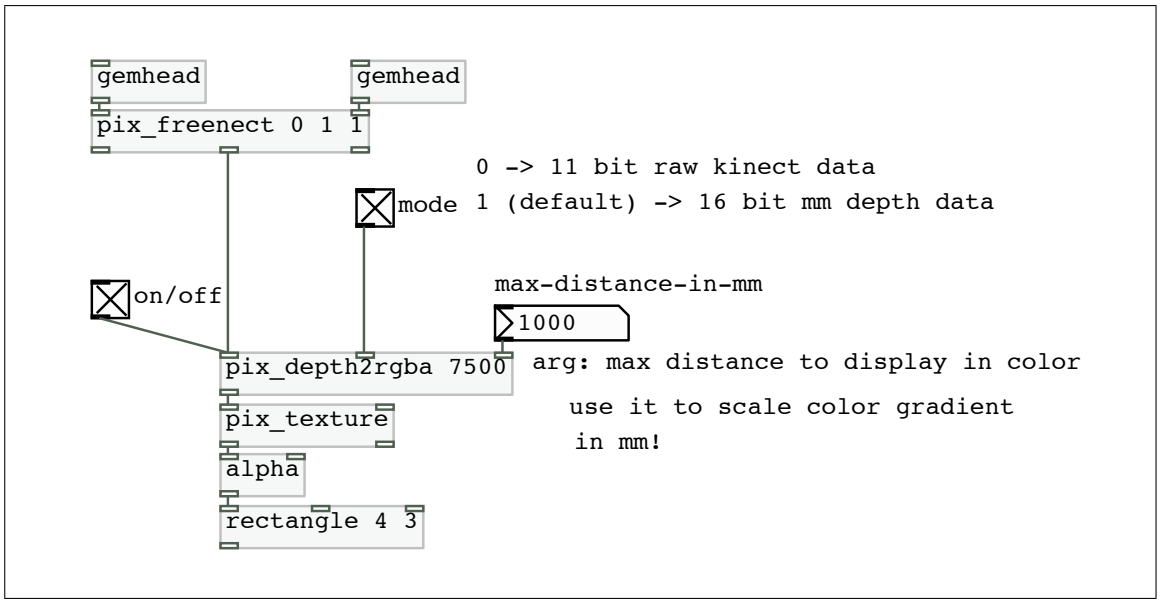
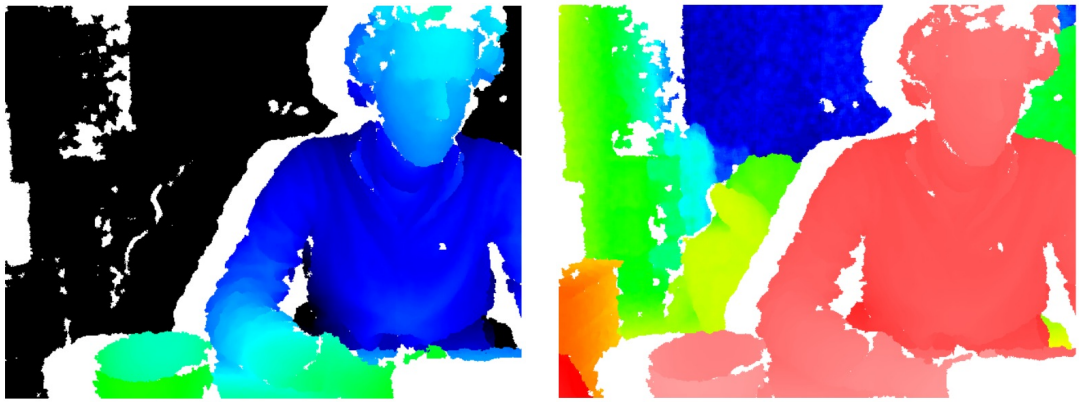



**Figure 4.3:** pix_depth2rgba patch



(a) gradient range $1000mm$ (b) gradient range $7500mm$

**Figure 4.4:** pix_depth2rgba output with different range settings

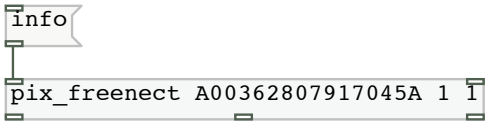



**Figure 4.5:** Open specific Kinect device by serial number and activate RGB and depth stream by creation arguments

```
1 ::freenect status::
2 [pix_freenect]: Number of devices found: 1
3 [pix_freenect]: Device 0 serial: A00362807917045A
4 [pix_freenect]: libfreenect supports FREENECT_DEVICE_MOTOR (3)
5 [pix_freenect]: libfreenect supports FREENECT_DEVICE_CAMERA (3)
```

**Listing 4.1:** Pure Data terminal output of `pix_freenect`

### 4.2.2 Messages/Settings

Internal settings can be changed by sending messages to the first inlet. An overview of available settings can be found in table 4.3.

| | |
|---|---|
| accel | send acceleration data to third outlet |
| info | output kinect serial numbers and libfreenect info to terminal |
| angle <f>[a] | change the tilt of the head (float $-30° \rightarrow 30°$) |
| led <b>[b] | change the color of the LED (int $0 \rightarrow 5$) |
| rgb <b> | activate (1) / deactivate (0) the rgb stream |
| depth <b> | activate (1) / deactivate (0) the depth stream |
| depth_output <i>[c] | output depthmap as RGBA (0) or YUV (1) image |
| depth_mode <i> | depthmap in $[mm]$ (0), rgb-aligned $[mm]$ (1) or 11 bit raw (2) |
| video_mode <f> | output rgb image (0) or infrared image (1) |
| resolution <i> | rgb resolution - 320x240 (0), 640x480 (1), 1280x1024 (2) |

[a] <f> float number
[b] <b> boolean - 0 or 1
[c] <i> integer

**Table 4.3:** Messages/settings for `pix_freenect`

### 4.2.3 Inlets/Outlets

The Kinect externals output two different image streams. Therefore the decision was made to implement two separate `gemhead` inlets and corresponding outlets (Fig. 1.1). The leftmost inlet/outlet pair is used for the RGB stream, the second inlet/outlet is used for the depth stream. The third outlet is used for accelerometer data output. Sending the `accel(` message to `pix_freenect` (Fig 4.7) triggers the output of three messages (`accel`, `tilt_angle` and `tilt_status`) at the third outlet (Tab. 4.4).

| | |
|---|---|
| accel <f> <f> <f> | raw accelerometer data for $x$-, $y$- and $z$-axis |
| tilt_angle <i> | raw tilt motor angle encoder information |
| tilt_status <i> | motor stopped (0), reached limit (1), currently moving (4) |

**Table 4.4:** Output at third outlet for `accel` message

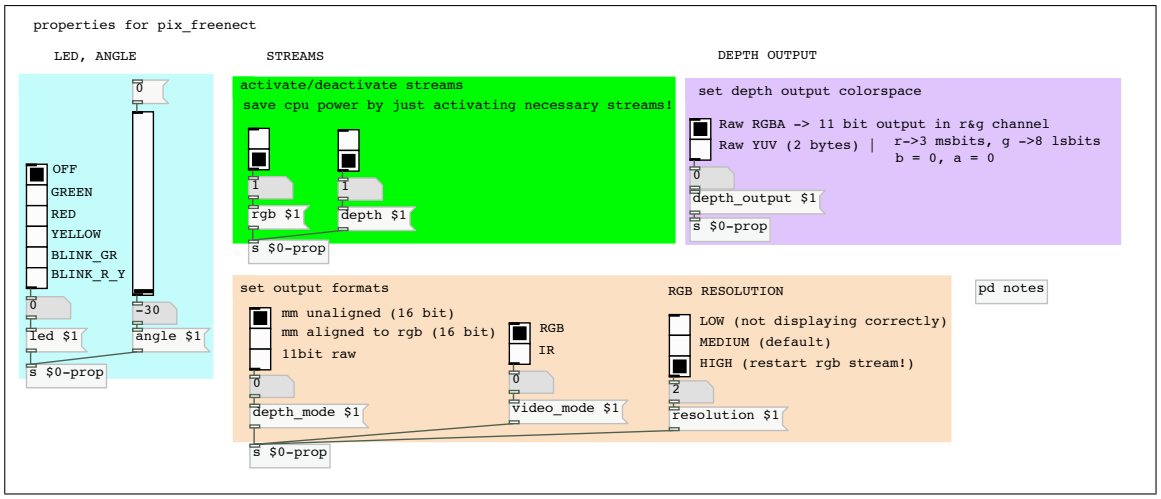



**Figure 4.6:** Properties dialog of `pix_freenect` helpfile

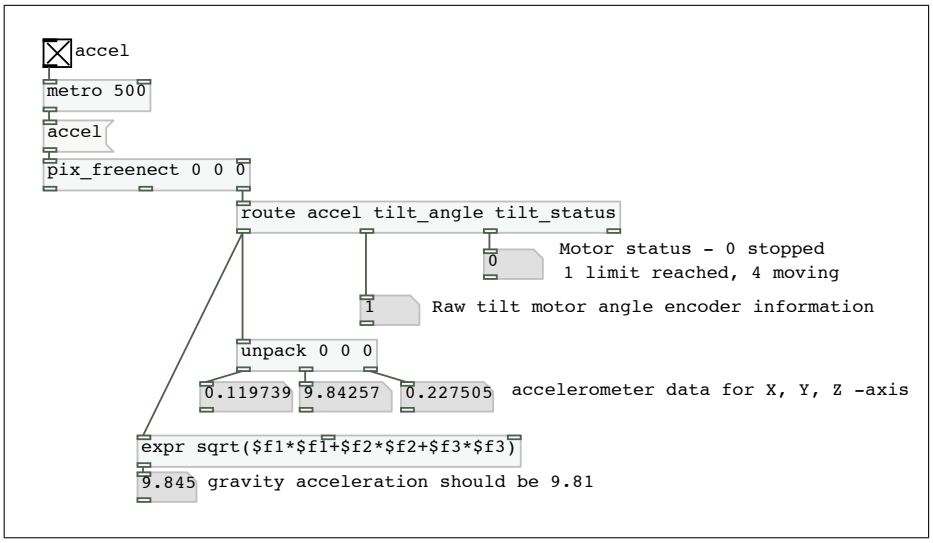



**Figure 4.7:** Sending `accel` message to `pix_freenect`

## 4.3 freenect

The external `freenect` was developed for parallel use with `pix_openni` which lacks access to motor tilt, LED and accelerometer data. These functions are similar implemented as in `pix_freenect` and listed in table 4.5. A screenshot of the helpfile can be seen in figure 4.8.

| | |
|---|---|
| accel | send acceleration data to third outlet |
| bang | output kinect serial numbers and libfreenect info to terminal |
| angle <f> | change the tilt of the head (float $-30° \rightarrow 30°$) |
| led <i> | change the color of the LED (int $0 \rightarrow 5$) |

**Table 4.5:** Messages for `freenect` - arguments: <f> float, <i> int

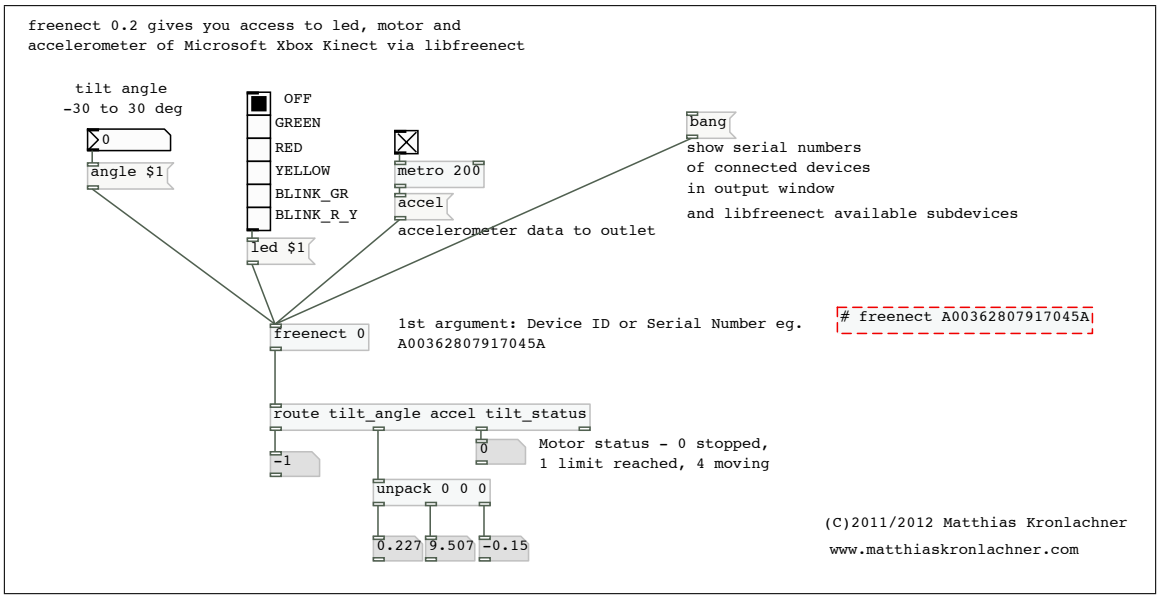



**Figure 4.8:** `freenect` help patch

## 4.4 freenect_audio

Libfreenect audio support is exclusively available for Linux. Therefore the external `freenect_audio` is currently working just with the Linux operating system.

When sending a `bang` to the inlet, each of the four outlets will output a list of floats, representing the samples of the corresponding microphone signal. The list will contain the samples that arrived from the Kinect device since the last `bang`. An internal buffer is responsible for caching the incoming samples. This buffer can overload if the time between two output triggers is too long. Currently resampling has to be done in the Pure Data message domain. A good approach would be to include the adaptive resampling described by Fons Adriaensen [Adr12] into a `freenect~` external.

## 4.5 pix_openni

Based on OpenNI (Sec. 3.2) and NiTE middleware by Primesense, the external `pix_openni` features some higher level functionality. Besides gathering the RGB and depth streams it

is possible to do hand tracking, user tracking and skeleton tracking (Fig. 4.15). Currently it is not possible to get accelerometer data, control the tilt and the color of the LED as well as receiving the audio streams. Therefore the libfreenect based externals freenect (Sec. 4.3) and the currently Linux only freenect_audio (Sec. 4.4) have been developed to be used simultaneously with pix_openni and provide the missing features.

### 4.5.1 Creation Arguments

The creation arguments of pix_openni allow to address multiple Kinect devices, turn on/off streams and tracking functionality.

pix_openni <device id $\geq 1$> <rgb on/off>
<depth on/off> <skeleton tracking on/off> <hand tracking on/off>.

Device IDs are assigned by the OpenNI hardware driver and start with one ($\geq 1$). It can not be guaranteed that the same device will be addressed by the same id after unplugging from the USB port. The Kinect OpenNI driver currently does not allow addressing a specific device by the serial number.

```
1 [pix_openni]: pix_openni 0.12 - 2011/2012 by Matthias Kronlachner
2 [pix_openni]: chosen Kinect Nr: 2
3 [pix_openni]: OPEN NI initialised successfully.
4 [pix_openni]: The following devices were found:
5 [pix_openni]: [1] PrimeSense Sensor (0)
6 [pix_openni]: [2] PrimeSense Sensor (0)
```

**Listing 4.2:** Pure Data terminal output of pix_openni

### 4.5.2 Messages/Settings

Internal settings can be changed by sending messages to the first inlet. An overview of available settings can be found in table 4.6.

### 4.5.3 Inlets/Outlets

The first two inlets/outlets of pix_openni are identical to pix_freenect (Sec. 4.2.3) and output a gem_state corresponding to the RGB and depth video stream. The third outlet is used for the output of messages corresponding to tracking, status and playback data. The output messages look differently depending on the osc_style_output setting (Sec. 4.5.5).

### 4.5.4 Recording/Playback

It is possible to record the RGB and depth stream from the Kinect sensor for later playback. Therefore the OpenNI specific .oni file format is used. Tracking data is not

| | |
|---|---|
| rgb <b>[a] | activate (1) / deactivate (0) the rgb stream |
| depth <b> | activate (1) / deactivate (0) the depth stream |
| usergen <b> | activate (1) / deactivate (0) user tracking |
| skeleton <b> | activate (1) / deactivate (0) skeleton tracking |
| hand <b> | activate (1) / deactivate (0) hand tracking |
| depth_output <b> | output depthmap as RGBA (0) or YUV (1) image |
| registration <b> | activate (1) / deactivate (0) depth to rgb image alignment |
| usercoloring <b> | activate (1) / deactivate (0) usercoloring |
| bang | output available resolutions/framerates of streams |
| video_mode <i>[b] <i> <i> | set rgb resolution/framerate |
| depth_mode <i> <i> <i> | set depth resolution/framerate |
| | |
| real_world_coords <b> | (1) output real world coordinates tracking data (default 0) |
| osc_style_output <b> | (1) output osc style tracking data (default 0) |
| euler_output <b> | (1) output orientation angles of joints |
| | |
| start_user | start skeleton tracking for all users |
| start_user <i> | start skeleton tracking for specific user id |
| stop_user | stop skeleton tracking for all users |
| stop_user <i> | stop skeleton tracking for specific user |
| userinfo | output userinfos at outlet (num_users, CenterOfMass,...) |
| | |
| auto_calibration <b> | (1) auto calibrate and start skeleton if user enters scene |
| skeleton_smoothing <f> | skeleton tracking data smoothing (0.0-> 1.0) default 0.5 |
| hand_smoothing <f>[c] | hand tracking data smoothing (0.0-> 1.0) default 0.5 |
| | |
| | |
| open <s>[d] | open filename (.oni) for playback or recording |
| record <b> | start recording (prior send open message!) |
| play <b> | start playback (prior send open message!) |
| playback_speed <f> | playback speed (1.0 = normal) |
| jump_image_frame <i> | playback from specific image frame |
| jump_depth_frame <i> | playback from specific depth frame |

[a] <b> boolean - 0 or 1
[b] <i> integer number
[c] <f> float number
[d] <s> symbol

**Table 4.6:** Messages/settings for `pix_openni`

recorded in the file. Nevertheless it will be computed in realtime if activated during playback of a `.oni` file. One minute recording will use approximately 180 Megabytes of disc space.

Sending a message in form of `open myfile.oni` to the first inlet (Fig. 4.9) will set the filepath for recording as well as playback. Filenames without absolute path are relative to the location of the Pd-patch (eg. `recordings/my_recording.oni`). An absolute path could look like `/tmp/my_recording.oni`.

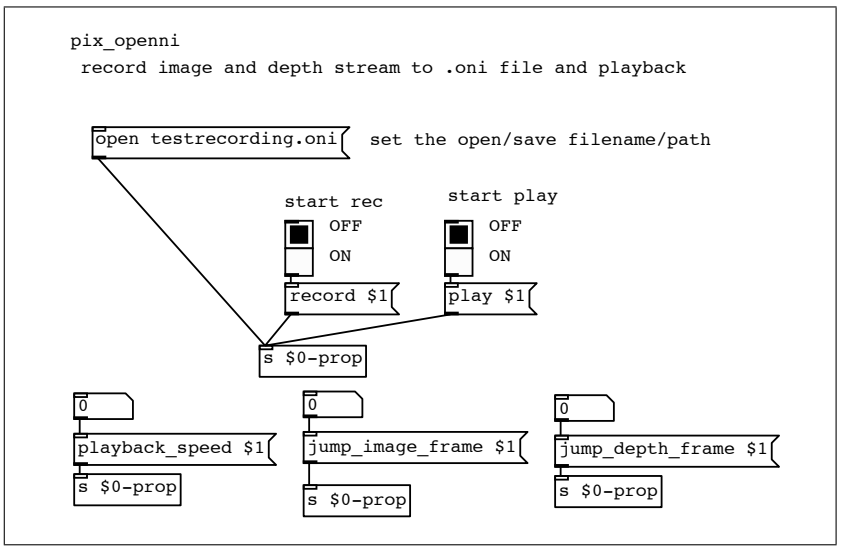



**Figure 4.9:** `pix_openni` record/playback patch

### 4.5.5 pix_openni and tracking

All tracking data provided by the `pix_openni` external will be sent to the rightmost outlet. The messages have common properties.

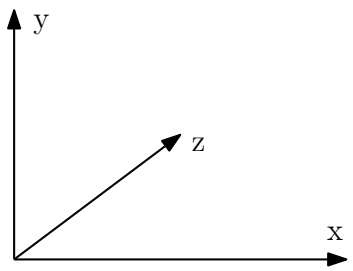



**Figure 4.10:** coordinate system used for tracking data

In the coordinate system used by OpenNI $+x$ points to the right, $+y$ points up and $+z$ points in the direction of increasing distance (Fig. 4.10). The default scaling of all

coordinates is between $0.0$ and $1.0$ for the $x$ and $y$ axis, and between $0.0$ and $7.0$ for the $z$ axis. This can be changed by sending the message `real_world_coords 1` to the first inlet. This setting will change the coordinate output to real world representation in millimetre $[mm]$.

```
hand 1 0.456289 0.420714 0.648926
```

**Listing 4.3:** Normalized coordinate output ($0 \leq xy \leq 1$; $0 \leq z \leq 7$)

```
hand 1 41.9625 50.7432 830.625
```

**Listing 4.4:** Realworld coordinate output in millimetre

The output format of the messages can be changed by sending `osc_style_output 1` to `pix_openni`. The OSC style output can have some advantages if you like to route tracking data directly to other applications or Pd instances through Open Sound Control. Please refer to the Hand tracking section (4.5.6) and Skeleton tracking section (4.5.8) for the specific output message syntax.

```
joint l_foot 1 0.311801 0.392408 2.78771 1
```

**Listing 4.5:** Skeleton default output style

```
/skeleton/joint/l_foot 1 0.311801 0.392408 2.78771 1
```

**Listing 4.6:** Skeleton OSC output style

### 4.5.6 Hand tracking

Hand tracking within `pix_openni` can be activated either by setting the fifth creation argument to 1 or by sending the message `hand 1` to the first inlet.

The hand gets detected by making a waving gesture. After the hand has been detected, the tracking data can be received with Gem frame-rate through the third outlet. Tracking data can be smoothed by adjusting the `hand_smoothing 0.5` parameter between $0.0$ and $1.0$.

The output messages look like:

```
hand <id> <x> <y> <z>
```

or if `osc_style_output 1` (Sec. 4.5.5)

```
/hand/coords <id> <x> <y> <z>
```

Multiple hand tracking has to be activated in the NiTE configuration file. You can find it in `/usr/etc/primesense/Hands_*/Nite.ini` [2]. Add or uncomment following lines:

[2] The asterisk sign `*` notates, that multiples of this directory may exist. Change the configuration file in every folder. For Windows the file can be found in `C:\Program Files\Prime Sense\NITE\Hands\Data`

```
1 [HandTrackerManager]
2 AllowMultipleHands=1
3 TrackAdditionalHands=1
```

**Listing 4.7:** change Nite.ini for multiple hands support

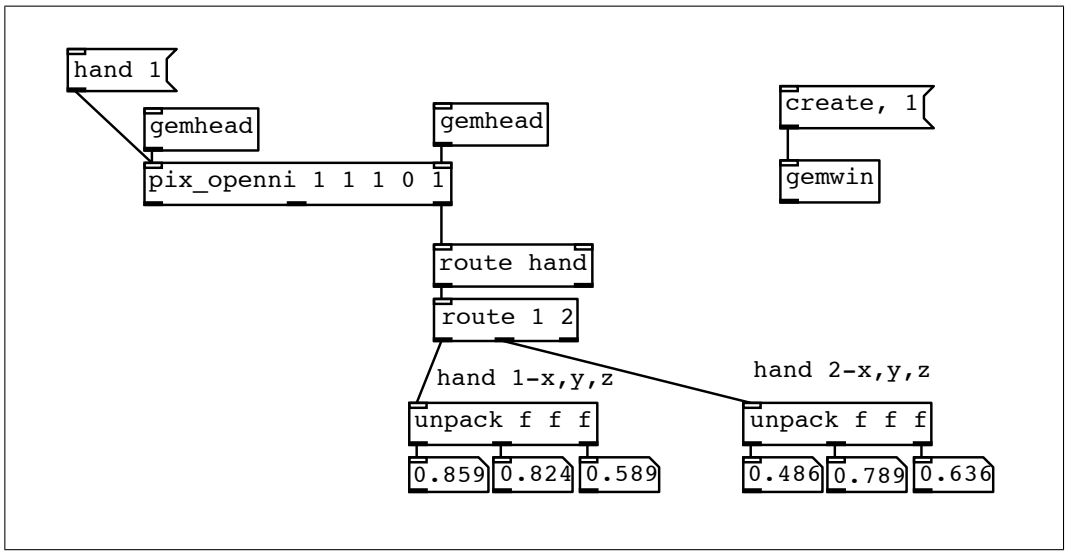



**Figure 4.11:** `pix_openni` multiple hand tracking

### 4.5.7 User generator

The user generator makes it possible to detect persons in the field of view. The number of detected persons in the depth image as well as their centre of mass can be retrieved by sending the message `userinfo` to the left inlet (Fig. 4.12).

```
num_users <number>
user <id> <skeleton-tracking on/off> <x> <y> <z>
```

**Listing 4.8:** userinfo output syntax

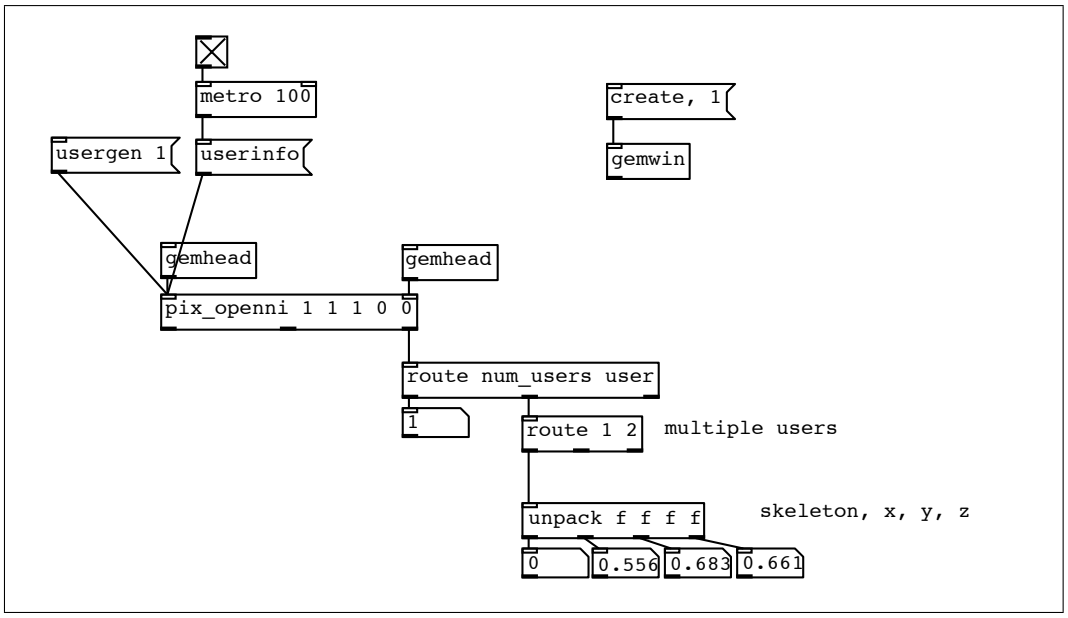



**Figure 4.12:** `pix_openni` user generator

```
num_users 1
user 1 0 0.405938 0.399831 0.912067
```

**Listing 4.9:** userinfo default output style

```
/skeleton/num_users 1
/skeleton/user 1 0 0.444361 0.4358 0.815637
```

**Listing 4.10:** userinfo OSC output style

### User coloring

User coloring allows to write user ids into the blue color channel of the depthmaps' RGBA representation. For using this feature, the user generator (Sec. 4.5.7) has to be turned on. The user generator will detect people in the depthmap. The pixels representing a user will have a blue value $B > 0$ if user coloring is turned on. This functionality can be used to filter recognized users (Fig. 4.13).

In figure 4.14 the blue channel value for the middle pixel and for the bottom left pixel is extracted with `pix_data`, additionally the output messages of `userinfo` is used to overlay the userid. The center pixel belongs to user 1, therefore the Blue channel holds the value 1. Gem uses normalized color values between $0.0$ and $1.0$, therefore a multiplication by `* 256` is necessary to get integer values. For better visual representation `pix_depth2rgba` dyes every detected user in a different color.

| R | G | B | A |
|---|---|---|---|
| depth data | depth data | userid | 255 |

**Table 4.7:** RGBA output of depth data including userid in blue channel

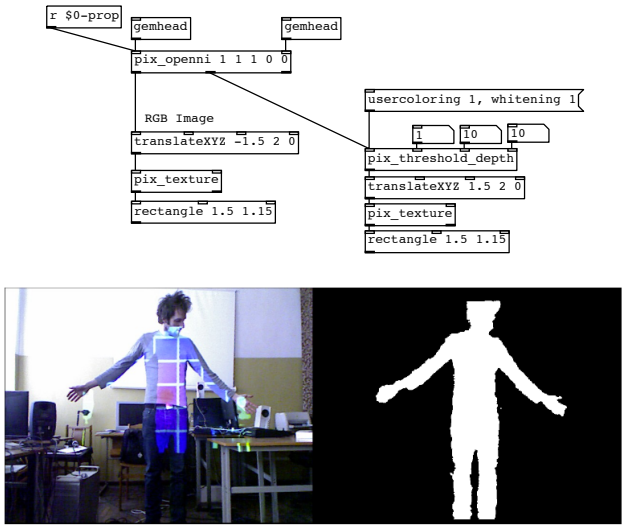



**Figure 4.13:** `pix_openni` user coloring, filtered by `pix_threshold_depth`

### 4.5.8 Skeleton tracking

After a user has been detected by the user generator (Sec. 4.5.7), it can start tracking 15 joints of the body (Fig. 4.15). Therefore skeleton has to be turned on either by setting the fourth creation argument to 1 or by sending the `skeleton 1;` message to `pix_openni`.

Skeleton tracking will start automatically if a user enters the scene. This can be turned off by the `auto_calibration 0;` setting. It is also possible to start and stop skeleton tracking for all or just specific users (Fig. 4.16). Tracking data can be smoothed by adjusting the `skeleton_smoothing 0.5;` parameter between $0.0$ and $1.0$.

OpenNI supports the output of 24 different joints. The NiTE middleware skeleton tracking supports just 15 joints. The skeleton output of `pix_openni` will therefore have additional joints with duplicated coordinates.

The tracking output for one frame looks like

```
joint <jointname> <user-id> <x> <y> <z> <confidence>
```

**Listing 4.11:** skeleton tracking output syntax

or for osc style output

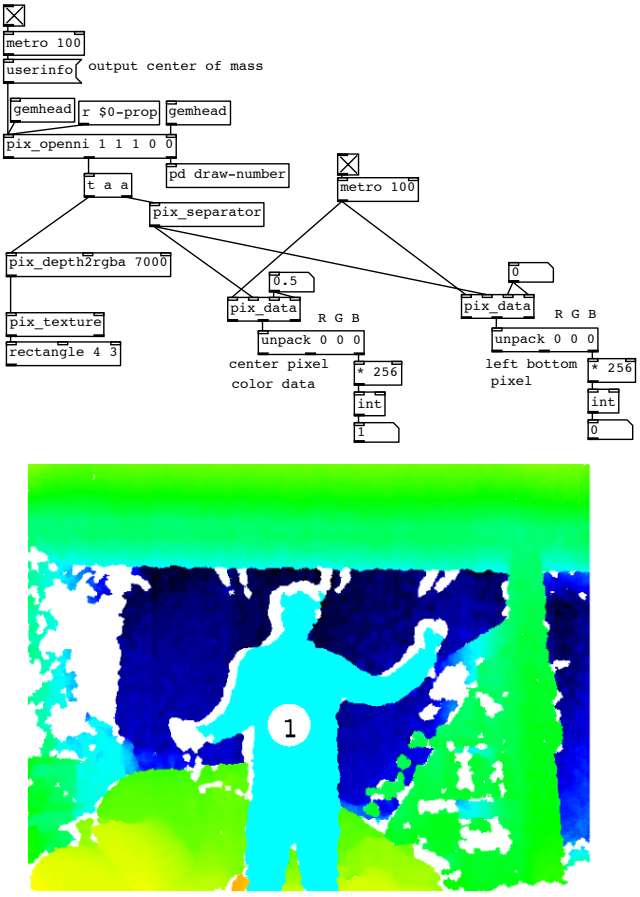



**Figure 4.14:** `pix_openni` user coloring and userinfo

```
/skeleton/joint/<jointname> <user-id> <x> <y> <z> <confidence>
```

**Listing 4.12:** skeleton tracking osc style output syntax

```
joint head 5 0.376254 0.158162 1.31012 1
joint neck 5 0.379469 0.300094 1.35346 1
joint torso 5 0.381939 0.416454 1.3511 1
joint waist 5 0.381939 0.416454 1.3511 0 (duplicate! not valid)
joint l_collar 5 0.381939 0.416454 1.3511 0
joint l_shoulder 5 0.442317 0.298091 1.39435 1
joint l_elbow 5 0.478067 0.420739 1.47322 1
joint l_wrist 5 0.478067 0.420739 1.47322 0 (duplicate! not valid)
joint l_hand 5 0.502907 0.580862 1.37264 1
joint l_fingertip 5 0.502907 0.580862 1.37264 0 (duplicate! not valid)
joint r_collar 5 0.502907 0.580862 1.37264 0 (duplicate! not valid)
joint r_shoulder 5 0.316621 0.302097 1.31258 1
joint r_elbow 5 0.291915 0.431105 1.37859 1
joint r_wrist 5 0.291915 0.431105 1.37859 0 (duplicate! not valid)
joint r_hand 5 0.243468 0.58301 1.26445 1
joint r_fingertip 5 0.243468 0.58301 1.26445 0 (duplicate! not valid)
```

```
17 joint l_hip 5 0.424873 0.531524 1.37506 1
18 joint l_knee 5 0.431999 0.783388 1.37493 1
19 joint l_ankle 5 0.431999 0.783388 1.37493 0 (duplicate! not valid)
20 joint l_foot 5 0.425306 0.991183 1.59826 1
21 joint r_hip 5 0.343947 0.534104 1.32241 1
22 joint r_knee 5 0.3335 0.777346 1.32825 1
23 joint r_ankle 5 0.3335 0.777346 1.32825 0 (duplicate! not valid)
24 joint r_foot 5 0.348461 0.954826 1.55574 1
```

**Listing 4.13:** Skeleton default style output for one frame

```
 1 /skeleton/joint/head 5 0.376254 0.158162 1.31012 1
 2 /skeleton/joint/neck 5 0.379469 0.300094 1.35346 1
 3 /skeleton/joint/torso 5 0.381939 0.416454 1.3511 1
 4 /skeleton/joint/waist 5 0.381939 0.416454 1.3511 0 (duplicate! not valid)
 5 /skeleton/joint/l_collar 5 0.381939 0.416454 1.3511 0
 6 /skeleton/joint/l_shoulder 5 0.442317 0.298091 1.39435 1
 7 /skeleton/joint/l_elbow 5 0.478067 0.420739 1.47322 1
 8 /skeleton/joint/l_wrist 5 0.478067 0.420739 1.47322 0 (duplicate! not valid)
 9 /skeleton/joint/l_hand 5 0.502907 0.580862 1.37264 1
10 /skeleton/joint/l_fingertip 5 0.502907 0.580862 1.37264 0 (duplicate! not
       valid)
11 /skeleton/joint/r_collar 5 0.502907 0.580862 1.37264 0 (duplicate! not valid)
12 /skeleton/joint/r_shoulder 5 0.316621 0.302097 1.31258 1
13 /skeleton/joint/r_elbow 5 0.291915 0.431105 1.37859 1
14 /skeleton/joint/r_wrist 5 0.291915 0.431105 1.37859 0 (duplicate! not valid)
15 /skeleton/joint/r_hand 5 0.243468 0.58301 1.26445 1
16 /skeleton/joint/r_fingertip 5 0.243468 0.58301 1.26445 0 (duplicate! not valid
       )
17 /skeleton/joint/l_hip 5 0.424873 0.531524 1.37506 1
18 /skeleton/joint/l_knee 5 0.431999 0.783388 1.37493 1
19 /skeleton/joint/l_ankle 5 0.431999 0.783388 1.37493 0 (duplicate! not valid)
20 /skeleton/joint/l_foot 5 0.425306 0.991183 1.59826 1
21 /skeleton/joint/r_hip 5 0.343947 0.534104 1.32241 1
22 /skeleton/joint/r_knee 5 0.3335 0.777346 1.32825 1
23 /skeleton/joint/r_ankle 5 0.3335 0.777346 1.32825 0 (duplicate! not valid)
24 /skeleton/joint/r_foot 5 0.348461 0.954826 1.55574 1
```

**Listing 4.14:** Skeleton OSC-style output for one frame

Status messages will be sent to the third outlet in case a user enters or leaves the scene, calibration is starting or a new skeleton is available.

```
new_user <user-id>
/skeleton/new_user <user-id>

lost_user <user-id>
/skeleton/lost_user <user-id>

calib_started <user-id>
/skeleton/calib_started <user-id>

new_skel <user-id>
```

```
/skeleton/new_skel <user-id>

new_skel_failed <user-id>
/skeleton/new_skel_failed <user-id>
```

**Listing 4.15:** Skeleton status messages

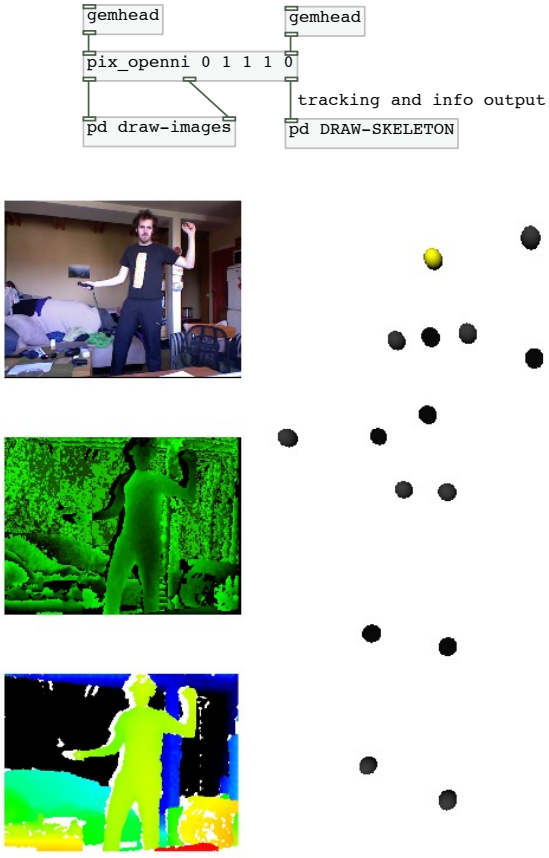



**Figure 4.15:** `pix_openni` skeleton tracking

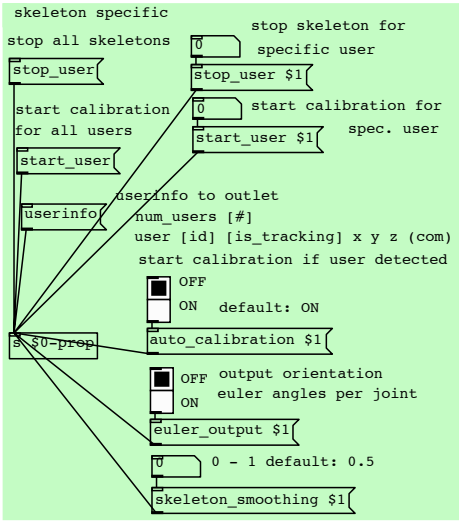



**Figure 4.16:** `pix_openni` skeleton settings

## 4.6 pix_threshold_depth

The external `pix_threshold_depth` can be used to extract parts of a depth image. Therefore minimum and maximum thresholds for the $x$-, $y$- and $z$-axis can be defined. All thresholds must be given in *millimeter*. It can be seen as extraction of a cuboid out of the depth image. This external can be used to do basic tracking using `pix_multiblob` or extracting a projection stencil. The availability of `pix_expr` would make this external redundant, as described in the introduction to section 4.

| | |
|---|---|
| <b>[a] | activate (1) / deactivate (0) computation |
| whitening <b> *(alt. 2nd inlet)* | passed pixels are set to white |
| invert <b> | invert the alpha mask |
| usercoloring <b> | let pass pixels with blue channel $B \neq 0$ |
| lo_thresh <f>[b] *(alt. 3rd inlet)* | set $z$-axis minimum (distance) |
| hi_thresh <f> *(alt. 4th inlet)* | set $z$-axis maximum |
| trim <b> | act. (1) / deact. (0) $x$- and $y$-axis thresholds |
| y_min <f> | set $y$-axis minimum (vertical) |
| y_max <f> | set $y$-axis maximum (vertical) |
| x_min <f> | set $x$-axis minimum (horizontal) |
| x_max <f> | set $x$-axis maximum (horizontal) |

[a] <b> boolean - 1 or 0
[b] <f> float number

**Table 4.8:** Messages/settings for `pix_threshold_depth`

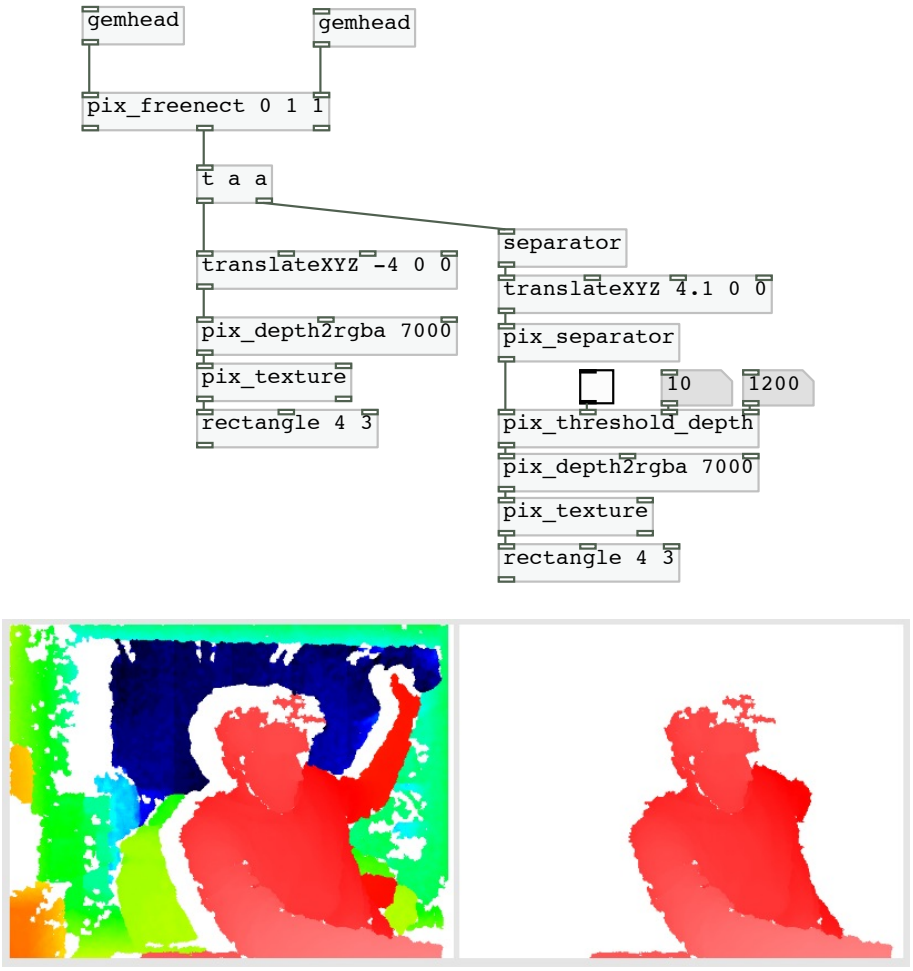



**Figure 4.17:** `pix_threshold_depth` example patch

## 4.7 pix_head_pose_estimation

Based on a paper and software by Gabriele Fanelli [FWGG11] the external `pix_head_pose_estimation` has been developed. It takes the depth map of the Kinect as input and estimates the Euler angles and position of multiple heads detected in the depth map. The estimator works with a reference database and covers a range of about $\pm 75°$ yaw and $\pm 60°$ pitch.

The second outlet of `pix_head_pose_estimation` will output the following message for every detected human head in the depth image.

```
head_pose <id> <x> <y> <z> <pitch> <yaw> <roll>
```

An example patch can be seen in Fig. 4.18. A comparison between real world and estimation is shown in Fig. 4.19. This external is also available as standalone application

sending OSC messages directly to a definable host address.

Estimating the head pose is very useful in virtual reality applications. Chapter 5 of [Kro12c] shows the practical application of the head pose estimation controlling an Ambisonics Binaural Decoder (Fig. 4.20).

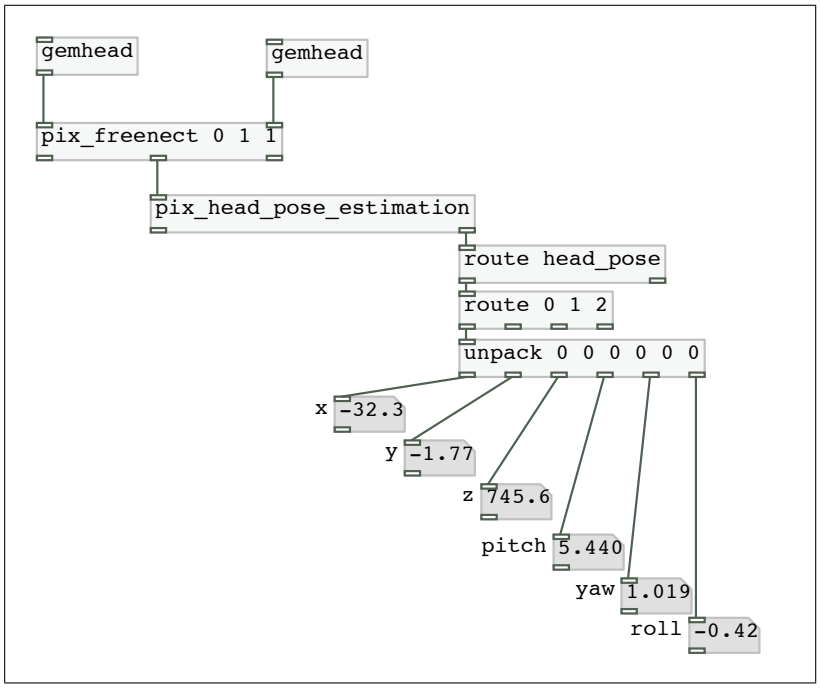



**Figure 4.18:** `pix_head_pose_estimation` example patch

**Figure 4.19:** comparison between real world and head pose estimation

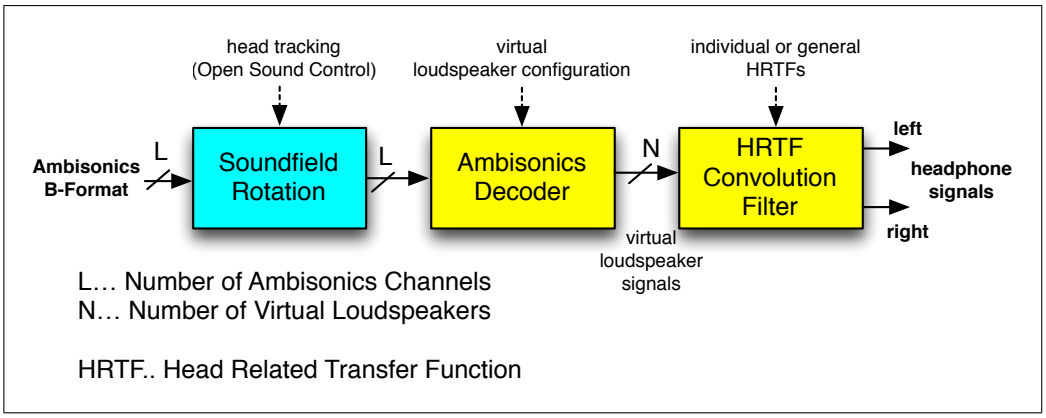



**Figure 4.20:** controlling Ambisonics binaural decoder with head pose estimator [Kro12c]

# 5 Application

## 5.1 ICE - IEM Computermusic Ensemble

ICE[1] is a group of electronic musicians, each playing with a notebook and individual controllers. The target is to play contemporary music, adapted or written for computermusic ensembles.

In March 2012 a network concert between Graz and Vilnius took place. One Kinect sensor was used in Vilnius to track the skeleton of two musician. The tracking data allowed each musician to play his virtual instrument without handheld controllers. Additionally the Kinect video stream showing the stage in Vilnius was sent to Graz and projected on a screen for the remote audience (Fig. 5.2). This application showed the easy and diverse usability of the Kinect sensor. Of course the movement area is quite limited if two people share the field of view (Fig. 2.6, Fig.5.3).

Due to performance reasons the Kinect tracking and video streaming was separated from the audio process and done with an individual computer.

The IEM Computermusic Ensemble is built in a way that audio and control data are streamed from the musician (clients) over network to the conductor (server) (Fig. 5.1). Due to this structure network concerts with remote musician are easy to realise and part of the concept. The use of the Ambisonics spatialisation technique allows different loudspeaker arrangements in every concert venue.

[1] IEM - Institute of Electronic Music and Acoustics, Graz ICE: http://www.iaem.at/projekte/ice

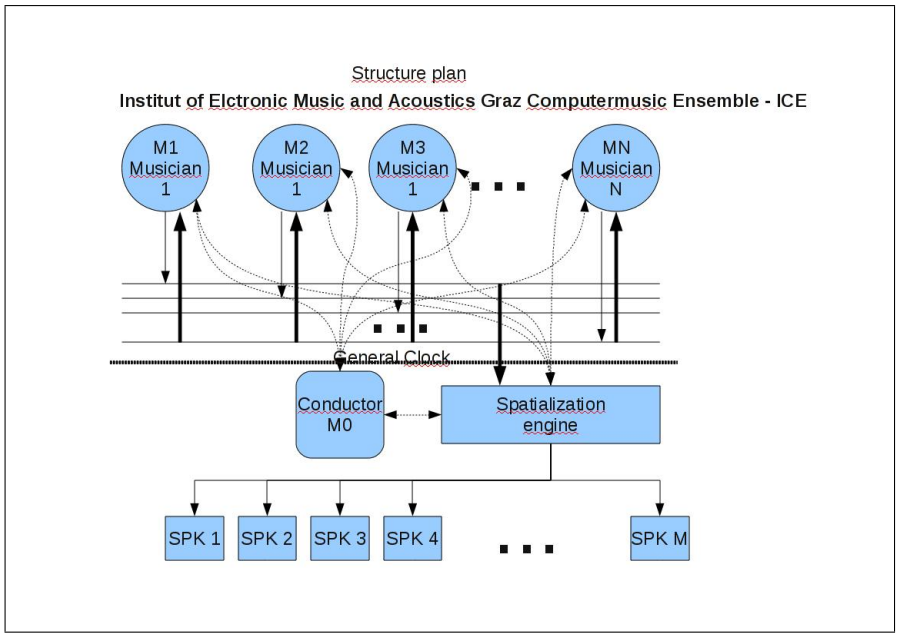



**Figure 5.1:** ICE structure (by Winfried Ritsch)

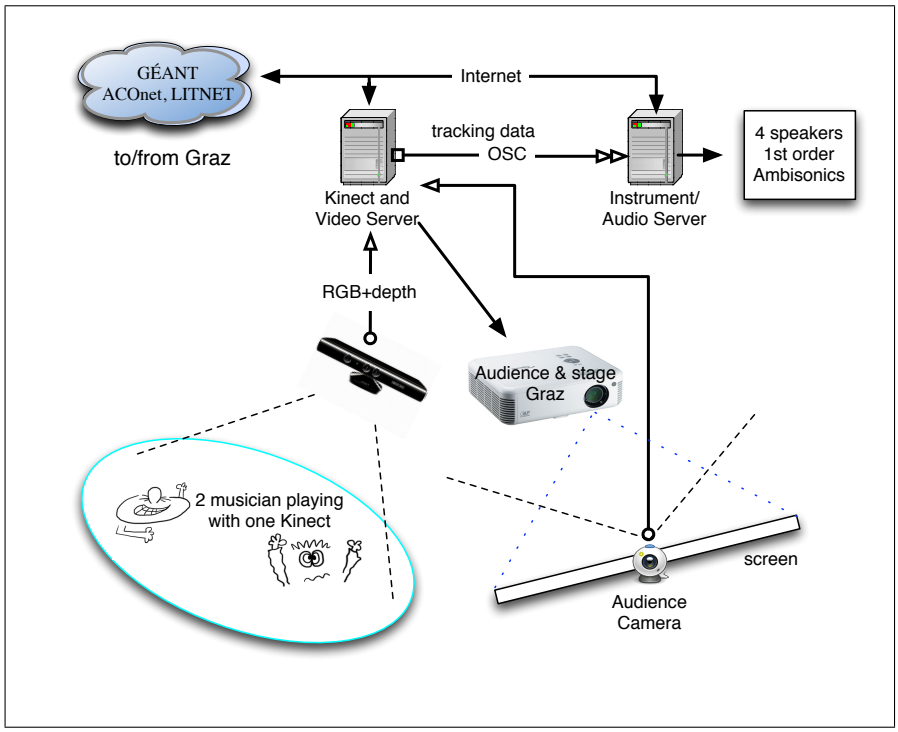



**Figure 5.2:** ICE network concert - stage setup Vilnius

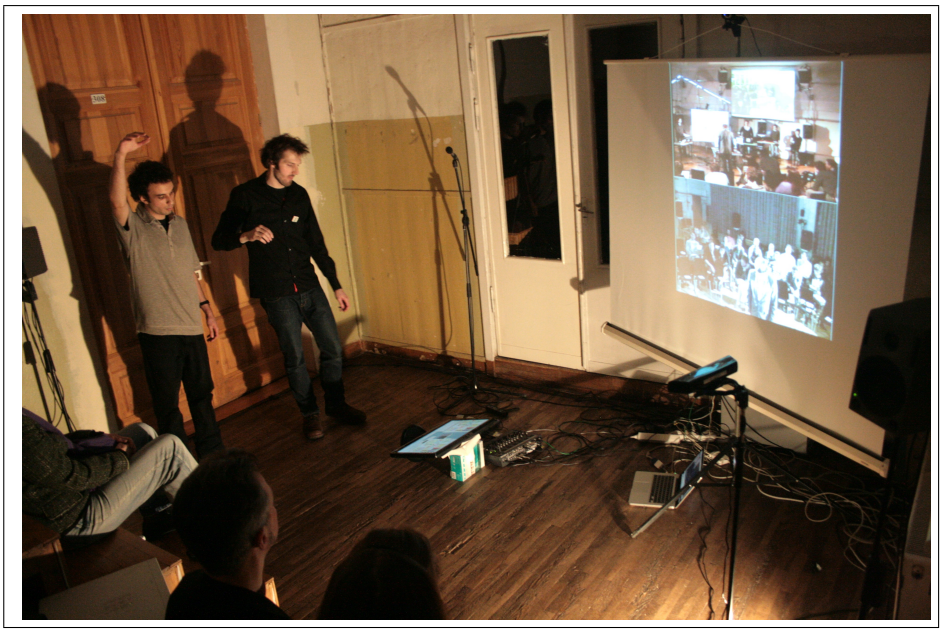


**Figure 5.3:** ICE stage view Vilnius

## 5.2 vertimas - übersetzen - for dancer, sound and projection

The piece *übersetzen - vertimas*[Kro12b] for dancer, sound and projection developed by the author, features the Kinect sensor to translate body movements on stage into sound and turns the dancers body into a hyperinstrument. Additionally, the depth video stream is used to gather the outline of the dancer and project back onto her body in realtime (Fig. 5.4).

Therefore an data-flow filtering and analysis library has been developed to enable quickly adjustable methods to translate tracking data into control data for sound or visual content (Fig. 5.5).

All graphics and sound generation was done within Pure Data. To solve the problem with audible drop outs, two separate instances of Pure Data have been used. One instance was responsible for visual rendering and skeleton extraction. The second instance was responsible for audio and received the tracking data from the visual Pd instance via OSC[2] (Fig. 5.6).

The *Extended View Toolkit* [VW11] was used to align the virtual projection image to the real projection surface.

Choreography for *übersetzen - vertimas* was done by Aira Naginevičiūtė, dancer Greta

[2]Open Sound Control (OSC) is a message based communication protocol for controlling multimedia systems.

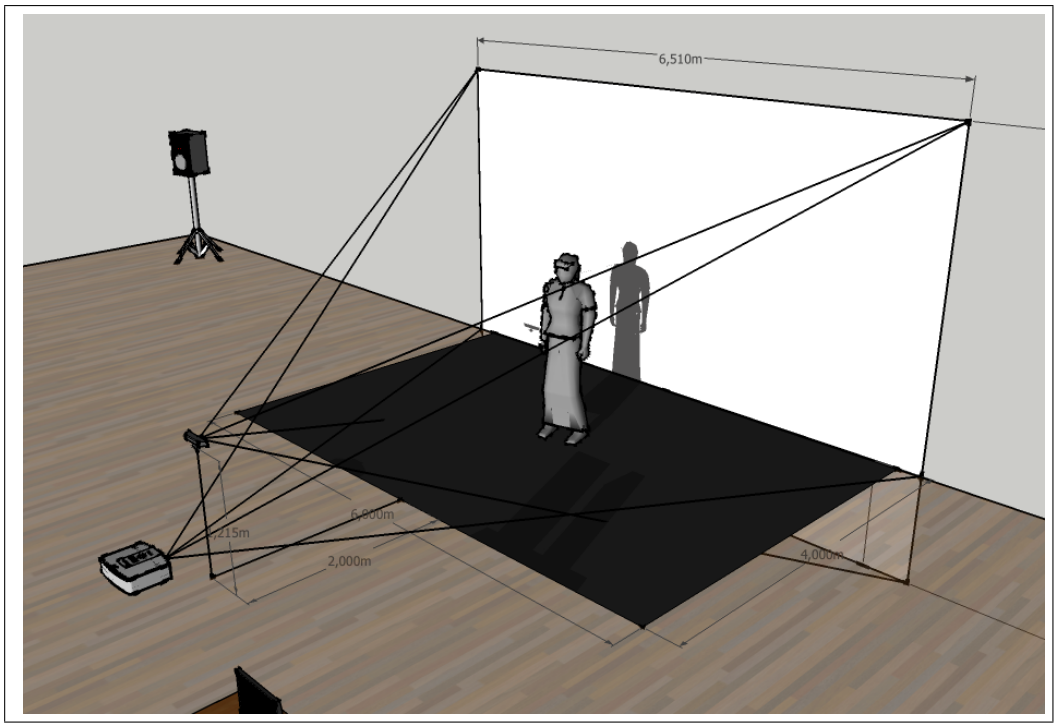



**Figure 5.4:** stage setup *vertimas - übersetzen*

Grinevičiūtė. It's first performance took place at the Jauna muzika festival 2012 in Vilnius, Lithuania and was supported by the Lithuanian Academy of Music and Theatre.

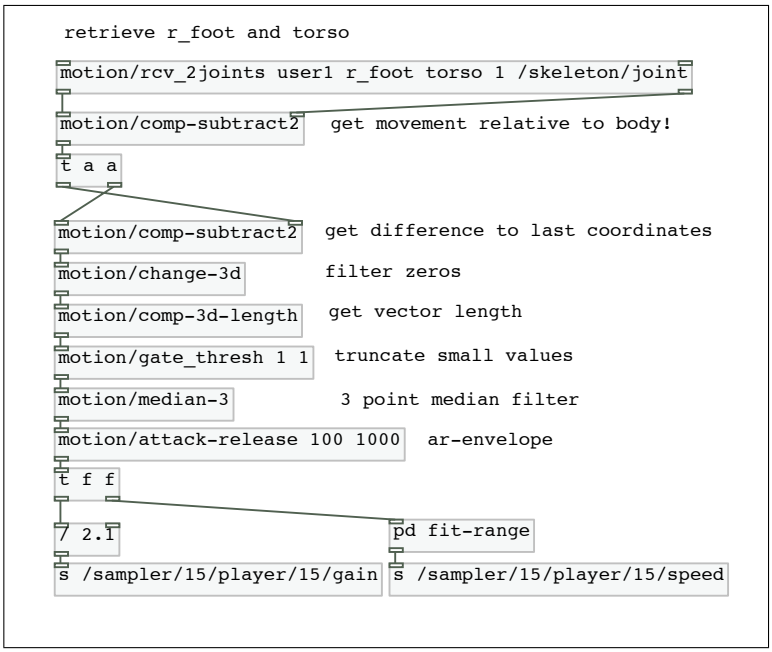



**Figure 5.5:** example patch converting skeleton to audio control data

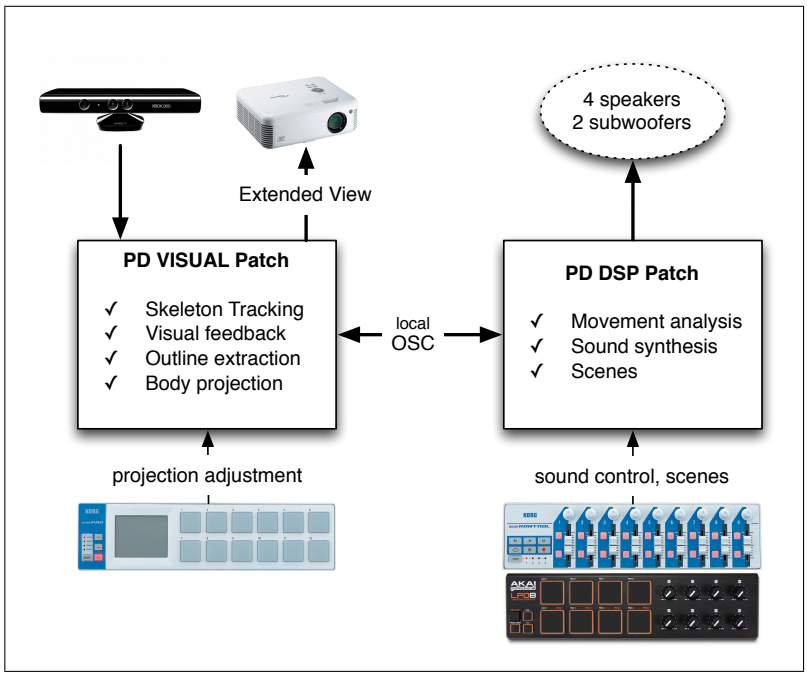



**Figure 5.6:** software structure *vertimas - übersetzen*

# 6 Conclusion

This project report showed the numerous application possibilities of the Kinect sensor. The different data streams of the sensor give many possibilities to create a bridge between art installations and their visitors just by using a single low-cost USB device. The Kinect can be bought in almost every electronic shop around the world. Therefore easy replacement during installations and tours is guaranteed.

Using Kinect allows skeleton tracking without the need of body mounted sensors or reflectors. This makes the usually technoid flavour of an interactive performance invisible and creates some more mysteries about the human-computer-interaction used.

Addressing a multifunctional device like the Kinect sensor rises the need of software, capable of dealing with different kind of data streams. Pure Data emerged to be a stable, easy to use and fast prototyping solution to deal with the video streams, tracking data and audio streams provided by the Kinect. For complicated applications performance problems using Pd may occur. A lower level language like *openFrameworks*[1] or *Cinder*[2] could be a solution for CPU intensive applications.

The cons of using Kinect include the limited range and resolution, the possible interference of other infrared light sources and the momentary dependance on non-open source software for higher level functionality.

[1] openFrameworks is an open source C++ toolkit for creative coding. http://www.openframeworks.cc
[2] Cinder is a community-developed, open source library for creative coding in C++. http://libcinder.org

# Bibliography


[Adr12] F. Adriaensen, "Controlling adaptive resampling," in *Linux Audio Conference 2012, Stanford University, California*, April 2012.

[avi10] avin, "Sensorkinect openni driver module," 12 2010. [Online]. Available: https://github.com/avin2/SensorKinect

[Ber11] A. Bernin, "Einsatz von 3d-kameras zur interpretation von räumlichen gesten im smart home kontext," Master's thesis, Hamburg University of Applied Sciences, Hamburg, Germany, 2011.

[FSMA10] B. Freedman, A. Shpunt, M. Machline, and Y. Arieli, "Depth mapping using projected patterns," Patent US 2010/0 118 123 A1, 05 13, 2010. [Online]. Available: http://www.freepatentsonline.com/20100118123.pdf

[fut10] futurepicture.org, "Looking at kinect ir patterns," 11 2010. [Online]. Available: http://www.futurepicture.org/?p=116

[FWGG11] G. Fanelli, T. Weise, J. Gall, and L. V. Gool, "Real time head pose estimation from consumer depth cameras," in *33rd Annual Symposium of the German Association for Pattern Recognition (DAGM'11)*, September 2011.

[Kro12a] M. Kronlachner, "Source code repository," 08 2012. [Online]. Available: http://github.com/kronihias

[Kro12b] ——, "übersetzen - vertimas - piece for dancer, sound and projection - trailer," Vilnius, Lithuania, 04 2012. [Online]. Available: http://vimeo.com/40919205

[Kro12c] ——, "Ambisonics binaural dekoder implementation als audio plug-in mit headtracking zur schallfeldrotation," Graz, Austria, 2012.

[MF12] A. Maimone and H. Fuchs, "Reducing interference between multiple structured light depth sensors using motion," *IEEE Virtual Reality 2012*, March 4-8, 2012.

[Mic12] Microsoft Developer Network, "Kinect SDK," 09 2012. [Online]. Available: http://msdn.microsoft.com

[MM12] F. Mayer and M. Meißnitzer, "Audio for computer gaming," Institut für Signalverarbeitung und Sprachkommunikation, Graz, Tech. Rep., 2012.

[Ope11a] OpenKinect, "libfreenect," 09 2011. [Online]. Available: http://openkinect.org

[Ope11b] OpenNI, 09 2011. [Online]. Available: http://www.openni.org

[Pri12] PrimeSense, "Primesense homepage," 09 2012. [Online]. Available: http://www.primesense.com

[Roc12] J. Rocha, "Skeltrack," 07 2012. [Online]. Available: http://github.com/joaquimrocha/Skeltrack

[VW11] P. Venus and M. Weger, "Extended view toolkit," 10 2011. [Online]. Available: http://extendedview.mur.at