A Kinect-Based Framework For Better User Experience in Real-Time Audiovisual Content Manipulation

Emmanouil Potetsianakis
UMR CNRS LTCI
Telecom ParisTech
75013 PARIS, France
potetsianakis@telecom-paristech.fr

Emmanouil Ksylakis
Applied Informatics and Multimedia
Technological Educational Institute of Crete
71410 HERAKLION, Greece
tp2241@edu.teicrete.gr

Georgios Triantafyllidis
Medialogy Section, AD:MT
Aalborg Univeristy
2450 COPENHAGEN, Denmark
gt@create.aau.dk

***Abstract*—Applications for real-time multimedia content production, because of their delay-sensitive nature, require fast and precise control by the user. This is commonly achieved by specialized physical controllers that are application-specific with steep learning curves. In our work, we propose using the inputs from Microsoft Kinect as a controller interface.**

Originally introduced as a peripheral of XBox, Kinect is a multimodal device equipped with RGB Camera, Depth Sensor and Microphone Array. We use those inputs in order to provide a non-tactile controller abstraction to the user, targeting multimedia content creation. Current Kinect-based solutions, try to recognize natural gestures of the user, and classify them as controller actions. The novelty of our implementation is that instead of extracting gesture features, we directly map the inputs from the Kinect to a suitable set of values for the multimedia application. By removing the gesture recognition concept, we are able to create a generic and lightweight framework, with a clear interface to the user. We examine the usability of the framework through the development and evaluation, of a Kinect-controlled real-time multimedia application.

***Keywords*—Human Computer Interaction, MS-Kinect, Multimedia, Mixed Media.**

I. Introduction

Nowadays there is a vast range of applications, that either present or create/modify audiovisual content. From a timing perspective, real-time multimedia applications, present the content as they produce it; in order to provide a seamless experience, they require precise control over the inputs. To achieve this precision, expensive specialized hardware is used. Our research is focused on using Microsoft Kinect as an input device, to control real-time multimedia applications.

Kinect is a multimodal input device, originally marketed as a peripheral of Xbox game console, that can provide Video, Audio and Depth information of the scene. It uses an infrared projector/camera pair - and with *Time-of-Flight* technology, calculates the distance of points within its range. The output consists of a matrix filled with those distance values, called *Depth Image*. In order for an application to extract data from the device, a middleware library is used, such as Microsoft SDK [1] or OpenNI [2]. The middleware, besides parsing the raw Kinect data, can track the position of the user and provide a set of depth values (*Coordinates*), for predefined points of interests, on his body. Those points are called *Joints* and their naming is consistent to the body part they represent (i.e. RightHand, Head, LeftShoulder etc.).

Most of the research efforts on Kinect interfaces, concentrate on utilizing Joint Coordinate data in order to identify user *Gestures* [3]. The data classification required for gesture recognition, can be done either by checking if a Set of Rules applies, or by running Machine Learning algorithms [4]. The downside in the first case is that in order for the classification to be efficient, every user must have his own Rules Set. If there is only one Rules Set, the user must be trained to it and the Gestures are not considered to be *Natural*. As for the Machine Learning scenario, a large data set must be provided for the Gesture Recognition engine to be efficient. In both cases, the classification process is computational expensive and can downgrade the user experience.

The fundamental difference between our framework, and the standard use of Kinect as a controller, lays on the principle that instead of performing complex Gesture Recognition techniques, then extracting gesture parameters and parse them as input values to the application; we select features of interest (a set of Joint Coordinates) and directly map their values to a range suitable for the multimedia application. This way, the time consumed to perform a Gesture is abstracted and we achieve performance closer to real-time, since the delay binds to the rate data is being extracted from Kinect. On terms of computational cost, the expensive Gesture Recognition algorithms, are replaced by simple mapping functions. By removing the Gesture concept we are also able to reduce the time required for the user to familiarize with the application. Specifically in the case of mapping the Joint Coordinates of the hands, the behavior of the interface is similar to a multi-touch screen (or touchpad). This provides a user interface experience that is already familiar, hence minimizing the learning curve – meaning that once the user is aware of which characteristics are mapped to what values, is able to accurately control the desired application.

In the following Section, we present the current state of the art on Gesture Recognition interfaces. Then, in Section III, we detail our framework design, to present it implemented on a sample application in Section IV. We show the future prospects in Section V, and conclude the paper in Section VI.

II. State of The Art

Gesture Recognition techniques have been utilized in applications varying from controlling multimedia playback [5], to web browsing [6], to medical imaging interaction [7] and others [8]. On the aforementioned examples, the interaction with the user does not produce multimedia content, instead it affects the way the content is being represented. As a result, there are not strict timing constraints and Gesture Recognition can be used to provide inputs in similar scenarios.

In the field of real-time multimedia content creation, Odowichuk et. al. [9] have created a platform that simulates music instrument (xylophone) playing. This is achieved by, initially creating a virtual representation of the instrument, then extracting motion characteristics from the gestures, spatially mapping them and parse the resulting values as parameters to a sound generator. In this case, the main principle is closer to our work, than the other examples, since it includes direct mapping of values, even though it is coupled with Gesture Recognition techniques. However, this work targets a very specific application, and due to their attempt for spatial mapping on the virtual xylophone, latency and temporal jitter are observed, rendering their method unsuitable for real-time multimedia. In order to minimize the effect of those issues, the authors are planning on implementing Machine Learning for movement prediction.

Churnside et. al. [10] designed a gestured-based audio interface system that uses Joint Coordinates to adjust the speed and volume of audio and video playback. There are similarities with our work, in the way Hand Coordinates are mapped to several parameters (different parameter per movement axis), the multimedia manipulation approach (real-time), and the separated interface-multimedia engines design. The system design however, is completely rigid and cannot be applied to other applications.

Several low-latency techniques are used for human-robot interaction, but they address to trained operators and assume specific underlying hardware setup [11] [12] [13].

On more generic frameworks, Deshayes et.al. have developed a framework for gesture-based applications, with Statechart modeling [14]. Even though their approach provides solutions on the application development and verification side, they are using traditional Gesture Recognition techniques. Finally, Vidakis et. al. [15] have examined the capabilities of Kinect as a multimodal interface, using Voice and Gestures, but there is no specific example on the real-time multimedia domain.

III. The Framework

The generic implementation of our framework can be split in two distinct elements; the *System Engine*, responsible for gathering and mapping the data from Kinect, providing the Parameter Values for application input, and the *Interface*, that displays information on the current state of the system to the user.

1) System Engine: The System Engine of our framework lays between the OpenNI middleware [2] and the multimedia application, as shown in Fig. 1. To setup the System Engine, a *Parameter Set* must be provided. The first part of a Parameter Set consists of a List with the multimedia application parameters and their possible values. Table I shows an example Parameter List for a Band-Pass Filter audio application. The second part is a Map between the aforementioned list and the framework parameters (Joint Coordinates). The shared key for the Map and the List, in order to build the Parameter Set, is the Parameter Name.

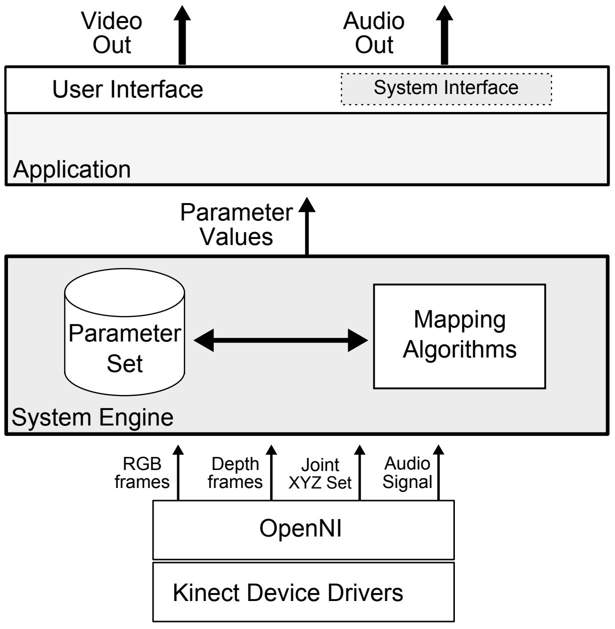


Fig. 1: System Architecture

The Map can be *Physical*, *Visual* or *Mixed*. The difference between them, is on the Joint Coordinate system used by the framework. The *Physical* Map uses the spatial coordinates of the Joints (distance from the device), as provided by the underlying middleware, in millimetres (mm). *Visual* maps to their projection coordinates (i.e. the Joints position on the screen), therefore the values are in pixels (px). In *Mixed*, the mapping method may vary between parameters. An example of a *Mixed* map for the values in Table I is shown in Table II.

To start, the user must initialize the framework and select a feature of the multimedia application. Afterward, the System Engine gathers the input data from Kinect and using the Parameter Set, maps the required Joint Coordinates to application-specific values, in order to parametrize the selected feature. Then, the parameter values are parsed to the multimedia application, where the content manipulation takes place. Any possible feedback, is delivered through the framework interface.

TABLE I: Application Parameter List

Parameter	Min. Value	Max. Value
Band Width	0 (Hz)	40000 (Hz)
Pass Band	20 (Hz)	20000 (Hz)
Active	OFF	ON
Volume	0 %	100 %

TABLE II: Parameter Map

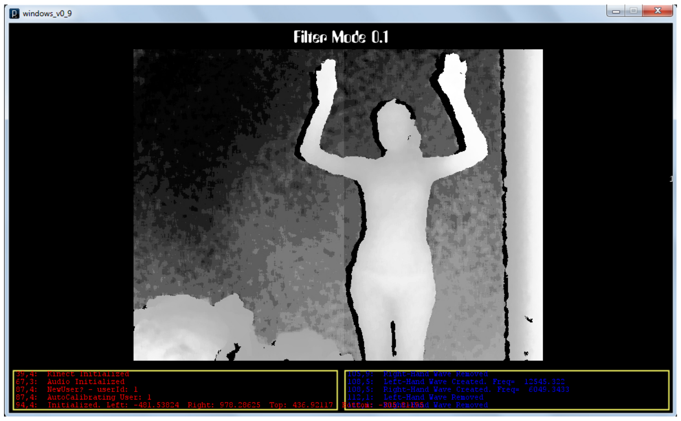
Parameter	Joint	Min.	Max.
Band Width	RightHand Y	120 (px)	330 (px)
Pass Band	RightHand X	300 (px)	550 (px)
Active	Users	0	1
Volume	RightHand Y	Torso Y	Head Y

2) The Interface: In order to easily operate the framework, an user interface (UI) implementation is needed. The framework interface can be embedded to the application interface, or implemented separately. Even though no specific UI rules apply, a typical interface paradigm consists of four elements:

1) *Area of Effect*: The virtual frame, in which the Joint Coordinates are mapped
2) *Virtual Buttons*: Pre-defined Areas of Effect, that implement switch behaviors
3) *Output Screen*: Real-time video output (if available)
4) *Monitor Screen*: Indications on the screen regarding the state of the framework

A Graphical User Interface (GUI) paradigm, can vary in complexity and information feedback; from a full implementation in which all four elements appear on the screen, to a very minimal, consisting of a *Monitor* or *Output* Screen only. The more elements are implemented, the easier it is for the user to operate the application, since there is more information available.

An example of a minimal interface implementation, for a real-time audio editing application, that implements only the *Monitor* Screen can bee seen in Fig.2. In this case, the Monitor Screen, consists of a *Depth Image* grayscale representation, and two message panels. The user is able to see a 2D projection of his position on screen, while the distance is represented in shades of gray; white being the closest to the device, and black out of range. At the bottom of the Monitor Screen, there are two message panels. The one on the left is used for informative messages regarding the current state of the application, and the other shows information on the Joint Coordinates and their mapped values.



```
39,4:  Kinect Initialized
67,3:  Audio Initialized
87,4:  NewUser? - userId: 1
87,4:  AutoCalibrating User: 1
94,4:  Initialized. Left: -481.53824  Right: 978.28625  Top: 436.92117  Bottom: -305.81195
```

```
105,9:  Right-Hand Wave Removed
108,5:  Left-Hand Wave Created. Freq=  12545.322
108,5:  Right-Hand Wave Created. Freq=  6049.3433
112,1:  Left-Hand Wave Removed
[illegible]  Right-Hand Wave Removed
```

Fig. 2: Sample framework interface implementing the Monitor Screen element

IV. Application Example

We created a demo multimedia application using our framework. The application was developed in Java, via the Processing wrapper - using OpenGL [16] for rendering the visual output, and minim [17] library for audio playback-/editing. In this application, the user can on-the-fly select an audio track, apply and parametrize a Band-Pass filter, or chose a visualization. In order to achieve these functions, the Joint Coordinates for hands are extracted for controlling the application.

The Interface of the application can be seen in Fig.3. Since the application is controlled by the RightHand and LeftHand coordinates, there are two dots on the screen, used as pointers for their current position. On the top part of the Interface, visible Virtual Buttons responsible for the application functions are located. A Volume button/indicator is on the bottom-left side of the screen, and a Monitor button/indicator on the bottom-right. The user selects the desired function by moving one of the pointers to the respective button. The message panel, on the bottom of the Interface, is reserved for displaying the title for the selected playback track.

The interface shown in Fig.3 is for the Band-Pass Filter mode. The current visualization selected is a real-time representation of the audio wave (for right and left channel) in the center of the screen. This can be used from the user as a visual aide. An alternative visualization that the user may select is, the bar spectrogram of the audio. That visualization is achieved by performing Discrete Fourrier Transformation (DFT) to the frequency domain.

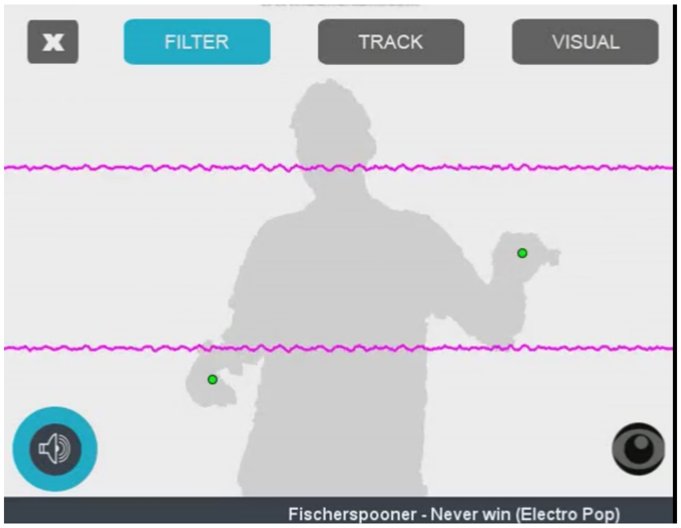


Fig. 3: Application Interface (in Band-Pass Filter mode)

To setup the System Engine of the framework, the application Parameter List must be parsed. In order to achieve this without editing the code, the framework has an integrated Extensible Markup Language (XML) document reader. This way, the Parameter List can be easily modified/updated, even after the application is deployed. For this application we are using the same Band-Pass Filter Parameters, as shown in Table I – expressed in XML format in Listing 1.

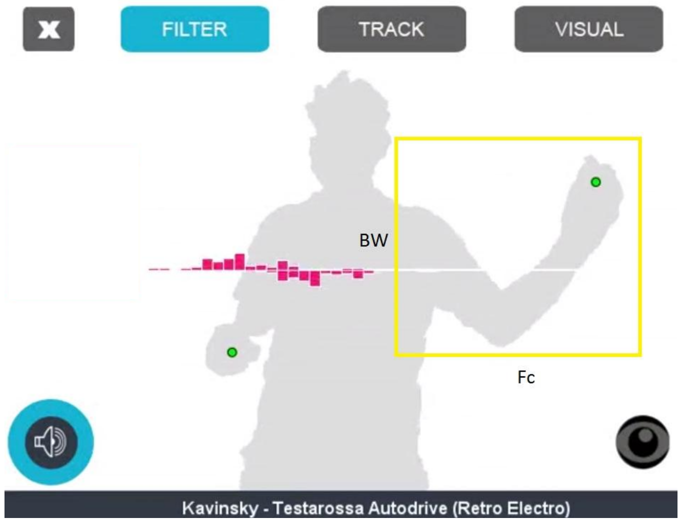
The *initial* field, specifies a parameter value, in case the function is activated, before the user sends any relevant input. It can have any values between the given *min*, and *max*. The *default* field, defines the framework behavior, when the user does not provide any input. This can be done intentionally, if the user decides to use some other feature of the application, while keeping the currently active on (in our case the Band-Pass Filter). It accepts the *min* and *max* keywords, any value in between, and the *HOLD* keyword, which keeps the last given value, until the user resumes control. Those two fields, even though optional, they are essential in real-time multimedia applications, to avoid contextual discontinuities, while handling different tasks simultaneously.

```
<Parameters>
    <Parameter Name="Band Width">
        <min> 0 </min>
        <max> 20000 </max>
        <initial> 40000 </initial>
        <default> HOLD </default>
    </Parameter>
    <Parameter Name="Pass Band">
        <min> 20 </min>
        <max> 20000 </max>
        <initial> 20000 </initial>
        <default> HOLD </default>
    </Parameter>
    <Parameter Name="Active">
        <min> FALSE </min>
        <max> TRUE </max>
        <initial> FALSE </initial>
        <default> FALSE </default>
    </Parameter>
    <Parameter Name="Volume">
        <min> 0 </min>
        <max> 100 </max>
        <initial> 50 </initial>
        <default> HOLD </default>
    </Parameter>
</Parameters>
```

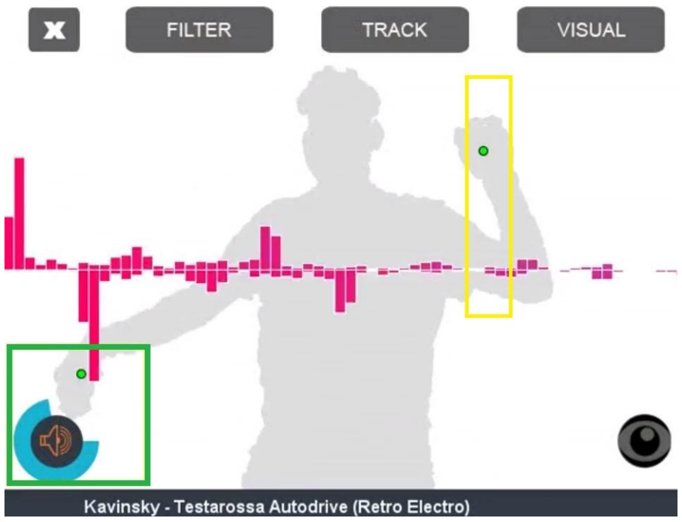
Listing 1: sample XML file

For the Parameter Map, we are using the same values as shown in Table II. The *Band Width* and *Pass Band* values are coupled – this causes the creation of a smaller rectangular Area of Effect, defined by the respective parameter range (values from the Parameter Map). In Fig. 4(a) the Band-Pass Filter is activated and its Area of Effect is highlighted. The Band Width parameter of the filter is set by the RightHand position on the Y axis, within the Area of Effect, while the Pass Band on the X axis.

The Volume function is coupled to its Button. As a result, the Volume parameter can be changed by the RightHand Y position, only if the user has his LeftHand over the Volume Virtual Button, as Fig. 4(b) shows. Otherwise, the value specified in the *default* field applies, and since it set to *HOLD*, the most recent value remains the same, until it is updated.



(a) Band-Pass Filter activated, with highlighted Area of Effect



(b) Volume Control activated, with highlighted Area of Effect

Fig. 4: Application Interface (in Band-Pass Filter mode)

V. Future Work

We are currently working on removing our framework from the multimedia application stack, to increase its interoperability. This way it can be used in a completely independent way, implemented with minimal effort, and even run the framework on a different local machine than the application. In order to separate the framework from the stack, we are introducing communication with the main application by using Open Sound Control [18]. OSC is a content format, that is already being used for communication in state of the art multimedia applications [19] [20]. The framework will independently perform the input gathering and mapping process, and the resulting values will be parsed to the application in a OSC-compliant format.

On top of that, we are extending the framework in order to support fully distributed processing. In this version, we will support scenarios where the user data gathering takes places in a remote machine – then, it is distributed over HTTP with a streaming service, and the data mapping and application control takes place locally. In order to achieve this, we add data compression and transportation techniques. Transporting RGB and Audio data captured by Kinect, will be done by implementing the MPEG Dynamic Adaptive Streaming over HTTP (DASH) standard [21], using the GPAC framework

[22]. With DASH a series of video streams are broadcasted and the client is able to chose according to its requirements and connection speed. Using the same DASH stream, other extra parameters can be parsed [23], in our case the Joint Coordinates and framework specifications.

For compressing the Depth stream, we will integrate the Point Clouds Library (PCL) [24]. PCL offers algorithms for lossy and lossless compression [25] [26]. The resulting Depth maps will be delivered in a DASH-like manner, and the client application will be able to chose the one it requires.

VI. Conclusion

Our work proposes a new approach to enhance the user experience in real-time multimedia applications controlled by Kinect. By implementing Parameter Mapping, and completely abstracting the Gesture Recognition concept, we created a low-latency, lightweight framework where various actions can be performed, in an application-agnostic environment. Any multimedia application can either be built on top of our framework, or easily integrate it, with a few modifications to its code. By implementing low-overhead networking capabilities, a decentralized approach can be realized. Finally, with our current work, we will be able to create completely scalable and heavily distributed, Kinect-controlled multimedia systems.

References

[1] Microsoft, *Microsoft Kinect SDK*. [Online]. Available: http://www.microsoft.com/en-us/kinectforwindows/

[2] PrimeSense, *OpenNI: Open-source SDK for 3D sensors*. [Online]. Available: http://www.openni.org

[3] O. Patsadu, C. Nukoolkit, and B. Watanapa, "Human gesture recognition using Kinect camera," in *Computer Science and Software Engineering (JCSSE), 2012 International Joint Conference on*, May 2012, pp. 28–32.

[4] G. Murthy and R. Jadon, "A Review of Vision Based Hand Gestures Recognition," *International Journal of Information Technology and Knowledge Management*, vol. 2, no. 2, pp. 405–410, 2009.

[5] S.-Y. Lin, Y.-C. Lai, L.-W. Chan, and Y.-P. Hung, "Real-Time 3D Model-Based Gesture Tracking for Multimedia Control," in *Pattern Recognition (ICPR), 2010 20th International Conference on*, Aug 2010, pp. 3822–3825.

[6] D. Liebling and M. R. Morris, "Kinected browser: depth camera interaction for the web," in *Proceedings of the 2012 ACM international conference on Interactive tabletops and surfaces*. ACM, 2012, pp. 105–108.

[7] R. Johnson, K. O'Hara, A. Sellen, C. Cousins, and A. Criminisi, "Exploring the potential for touchless interaction in image-guided interventional radiology," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2011, pp. 3323–3332.

[8] L. Cruz, D. Lucio, and L. Velho, "Kinect and RGBD images: Challenges and applications," in *Graphics, Patterns and Images Tutorials (SIBGRAPI-T), 2012 25th SIBGRAPI Conference on*. IEEE, 2012, pp. 36–49.

[9] G. Odowichuk, S. Trail, P. Driessen, W. Nie, and W. Page, "Sensor fusion: Towards a fully expressive 3D music control interface," in *Communications, Computers and Signal Processing (PacRim), 2011 IEEE Pacific Rim Conference on*. IEEE, 2011, pp. 836–841.

[10] A. Churnside, C. Pike, and M. Leonard, "Musical MovementsGesture Based Audio Interfaces," in *Audio Engineering Society Convention 131*, Oct 2011. [Online]. Available: http://www.aes.org/e-lib/browse.cfm?elib=16022

[11] M. Van den Bergh, D. Carton, R. de Nijs, N. Mitsou, C. Landsiedel, K. Kuehnlenz, D. Wollherr, L. Van Gool, and M. Buss, "Real-time 3D hand gesture interaction with a robot for understanding directions from humans," in *RO-MAN, 2011 IEEE*, July 2011, pp. 357–362.

[12] P. Barattini, C. Morand, and N. M. Robertson, "A proposed gesture set for the control of industrial collaborative robots," in *RO-MAN, 2012 IEEE*, Sept 2012, pp. 132–137.

[13] D. Xu, Y.-L. Chen, C. Lin, X. Kong, and X. Wu, "Real-time dynamic gesture recognition system based on depth perception for robot navigation," in *Robotics and Biomimetics (ROBIO), 2012 IEEE International Conference on*, Dec 2012, pp. 689–694.

[14] R. Deshayes and T. Mens, "Statechart Modelling of Interactive Gesture-Based Applications," in *Proc. First International Workshop on Combining Design and Engineering of Interactive Systems through Models and Tools (ComDeis-Moto),. Lisbon, Portugal (September 2011), iNTERACT*, 2011.

[15] N. Vidakis, A. Vlasopoulos, T. Kounalakis, P. Varchalamas, M. Dimitriou, G. Kalliatakis, E. Syntychakis, J. Christofakis, and G. Triantafyllidis, "Multimodal desktop interaction: The face-object-gesture-voice example," in *Digital Signal Processing (DSP), 2013 18th International Conference on*. IEEE, 2013, pp. 1–8.

[16] R. J. Rost, *OpenGL Shading Language*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 2004.

[17] J. A. Mills III, D. Di Fede, and N. Brix, "Music Programming in Minim," in *Proceedings of the 2010 Conference on New Interfaces for Musical Expression (NIME 2010). Sydney, Australia: New Interfaces fr Musical Expression*, vol. 2010, 2010, pp. 37–42.

[18] M. Wright, "Open sound control 1.0 specification," *Published by the Center For New Music and Audio Technology (CNMAT), UC Berkeley*, 2002.

[19] A. Schmeder, A. Freed, and D. Wessel, "Best Practices for Open Sound Control," in *Linux Audio Conference*, Utrecht, NL, 01/05/2010 2010.

[20] M. Wright, A. Freed, and A. Momeni, "Opensound control: State of the art 2003," in *Proceedings of the 2003 Conference on New Interfaces for Musical Expression*, ser. NIME '03. Singapore, Singapore: National University of Singapore, 2003, pp. 153–160. [Online]. Available: http://dl.acm.org/citation.cfm?id=1085714.1085751

[21] I. Sodagar, "The MPEG-DASH Standard for Multimedia Streaming Over the Internet," *IEEE Multimedia*, vol. 18, no. 4, pp. 62–67, 2011.

[22] Le Feuvre, Jean and Concolato, Cyril and Moissinac, Jean-Claude, "GPAC: Open Source Multimedia Framework," in *Proceedings of the 15th International Conference on Multimedia*, ser. MULTIMEDIA '07. New York, NY, USA: ACM, 2007, pp. 1009–1012. [Online]. Available: http://doi.acm.org/10.1145/1291233.1291452

[23] C. Concolato, J. Le Feuvre, and R. Bouqueau, "Usages of DASH for Rich Media Services," in *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, ser. MMSys '11. New York, NY, USA: ACM, 2011, pp. 265–270. [Online]. Available: http://doi.acm.org/10.1145/1943552.1943587

[24] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[25] F. Bannò, P. S. Gasparello, F. Tecchia, and M. Bergamasco, "Real-time Compression of Depth Streams Through Meshification and Valence-based Encoding," in *Proceedings of the 11th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry*. ACM, 2012, pp. 263–270.

[26] J. Kammerl, N. Blodow, R. Rusu, S. Gedikli, M. Beetz, and E. Steinbach, "Real-Time Compression of Point Cloud Streams," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, May 2012, pp. 778–785.