Hands-on: Rapid Interactive Application Prototyping for Media Arts and Performing Arts in Illimitable Space

Serguei A. Mokhov
Concordia University, Montreal, Canada
Montreal, Canada
mokhov@cse.concordia.ca

Miao Song
Concordia University, Montreal, Canada
Montreal, Canada
m_song@cse.concordia.ca

Sudhir P. Mudur
Concordia University, Montreal, Canada
Montreal, Canada
mudur@cse.concordia.ca

Peter Grogono
Concordia University, Montreal, Canada
Montreal, Canada
grogono@cse.concordia.ca

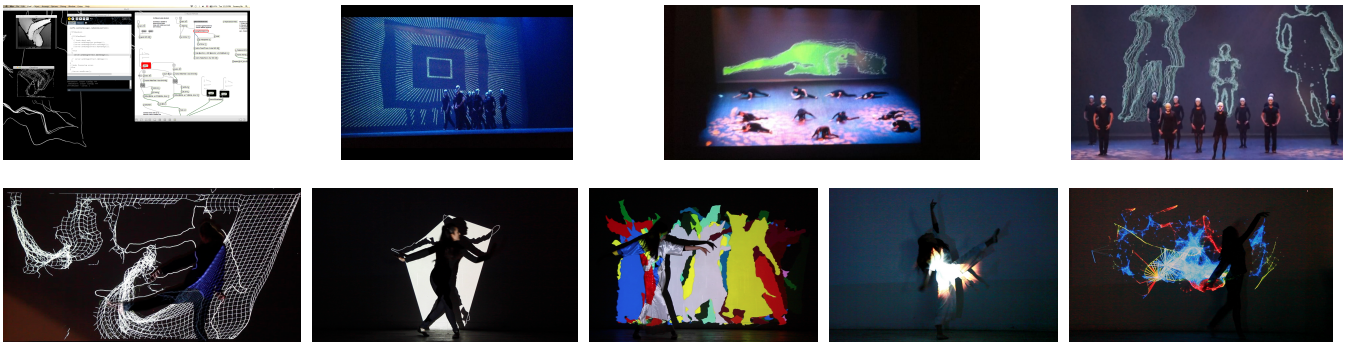


Figure 1: *Gray Zone* and *D3 Demo Day* ISSv2 Productions using Jitter and Processing with Kinect and OpenGL

ABSTRACT

We complement the last three editions of the course at SIGGRAPH Asia (2015, 2016, 2018) and SIGGRAPH (2017) to make it more of a hands-on nature and include OpenISS. We explore a rapid prototyping of interactive graphical applications for stage and beyond using Jitter/Max and Processing with OpenGL, shaders, and featuring connectivity with various devices. Such rapid prototyping environment is ideal for entertainment computing, as well as for artists and live performances using real-time interactive graphics. We share the expertise we developed in connecting the real-time graphics with on-stage performance with the *Illimitable Space System* (ISS) v2 and its OpenISS core framework for creative near-realtime broadcasting, and the use of AI and HCI techniques in art.

CCS CONCEPTS

• **Computing methodologies** → **Computer vision**; • **Human-centered computing** → **Graphics input devices**; • **Information systems** → **Multimedia content creation**; • **Applied computing** → **Sound and music computing**;

KEYWORDS

Jitter/MAX, Processing, Illimitable Space System (ISS), OpenISS, OpenGL, real-time, human-computer interfaces, interaction, computer graphics education, RGBD cameras

ACM Reference format:
Serguei A. Mokhov, Miao Song, Sudhir P. Mudur, and Peter Grogono. 2019. Hands-on: Rapid Interactive Application Prototyping for Media Arts and Performing Arts in Illimitable Space. In *Proceedings of SIGGRAPH'19 Studio, Los Angeles, CA, USA, July 28 - August 01, 2019,* 33 pages.
https://doi.org/10.1145/3306306.3328008

SIGGRAPH'19 Studio, July 28 - August 01, 2019, Los Angeles, CA, USA

ACM ISBN 978-1-4503-6316-7/19/07.
https://doi.org/10.1145/3306306.3328008

COURSES: STUDIO WORKSHOP

Contents

List of Figures

1 SYLLABUS

The tentative outline of the course is below based on the previous editions of the course in 2015 and 2016 held at SIGGRAPH Asia [Song et al. 2015, 2016]. Nearly each item in the syllabus has a hands-on exercise to try things out. The code in the slides and related web resources is provided as a teaching medium. The additional course resources can be found at this URL:
http://mdreams-stage.com/courses/s2019

The course is customizable of various lengths. The described option in this instance is for a half-day format.

The **tentative** schedule is below. We will adjust depending on the audience aptitude and availability of newer equipment and technologies by the time of the course.

- **Introduction (105 minutes)**
  Presenters: Serguei Mokhov
  - (20 min) General Introduction into Interactive Applications and Production Examples with the Illimitable Space System
  - (20 min) Introduction to Max, Sound, OpenGL in Jitter
  - (20 min) Kinect, OpenGL, and Jitter
  - (20 min) Wii and Jitter
  - (25 min) iDevices and Jitter
- Break / hands-on exercises (15 minutes)
- **Advanced Interaction (105 minutes)**
  - (15 min) Tracking
  - (15 min) Shaders in Jitter
  - (25 min) Processing, its VFX, Pipeline/Workflow
  - (25 min) Putting it all Together
  - (25 min) Hands-on exercises

2 BIOS

2.1 Serguei A. Mokhov

Serguei Mokhov had obtained his PhD from Concordia University, Montreal, Quebec, Canada, where he completed his bachelor's and master's degrees in Computer Science and Information Systems Security. Mokhov's diverse research interests include intensional programming, distributed and autonomic computing, digital forensics, information systems security, AI, software engineering, computer graphics and HCI, Linux, and computer networks. His PhD dissertation had to do with *Intensional Cyberforensics*. He also teaches at Concordia and Champlain College of Vermont various

subjects in those disciplines. Mokhov taught CG courses to undergraduate students at Concordia. He originally proposed the OpenGL slides framework for simple hair animation project in 2003.

Song and Mokhov jointly did the stereo 3D software implementation plug-in for Maya in 2008 [Song et al. 2009]. Mokhov later joined Song to work on the ISS as a technical lead at mDreams Stage.

Mokhov is a Co-founder and Executive Vice Director of CCIFF and in particular was involved in co-organizing and chairing its Entertainment Technology (ET) Summit and Exhibition (http://cciff.ca/en/festival/ets/) in 2016 and leading its 2017 edition as a conference with publications.

2.2 Miao Song

Miao Song completed her first Bachelor's degree in Performance Arts and Direction in China. While in China, she worked in the China Central Television Station (CCTV) as a TV director and journalist. She obtained her second Bachelor's degree in Computer Science, Concordia University, Montreal, Canada, and her Master's of Computer Science degree, also at Concordia, focusing on interactive real-time softbody simulation. Song completed the SIP PhD program with research involving a mix of interactive environments, cinema and documentaries, 3D computer graphics and realistic physical based simulation, and haptic responsive environments. Song has been awarded with various scholarships and grants for her research work. Her film project has been screened at national and international film festivals. Song taught HCI Design and Real-time Video courses in Computer Science and Software Engineering as well as Computation Arts departments.

Song produced realtime physical-based softbody simulation system and an interactive documentary prototype remake in OpenGL of her short documentary film followed by the *Illimitable Space System* (ISS) prototype in her doctorate thesis [Song 2012].

Song established mDreams Stage Research Creation group (http://mdreams-stage.com) at Concordia as a part of the Concordia 3D Graphics group and led many related events involving the ISS.

Song is a Founder and Executive Director of the Canada China International Film Festival (CCIFF), http://cciff.ca, held in Montreal, Canada, September 16–19, 2016 and the 2nd edition is upcoming in September 2017.

As Affiliates at Concordia, Song and Mokhov continue to collaborate on interactive graphics and new media projects together with Drs. Sudhir Mudur and Peter Grogono. Our joint work included *Haptic Jellyfish* at the CHI'14 WIP section [Song et al. 2014b] as well as Kinect-based installations featured on stage in a Chinese New Year Gala show on January 2014 [Song and Mokhov 2014] and a theatre production at the Central Academy of Drama in Beijing in April 2014 [Song et al. 2014a]. The corresponding video excerpts can be found at http://vimeo.com/channels/153466 and related publications appeared in [Song et al. 2014a,b]. We also lead ISSv2 in Jitter/Max with Processing with four students that was first deployed on February 14, 2015 for the *Gray Zone* dance at the Chinese New Year Gala 2015 [Song et al. 2015b]. That followed by the District 3 Demo days in Montreal in 2015 and 2016, and a presentation with the live dance at the *CG in Asia* section at SIGGRAPH 2015 [Song et al. 2015b,a] a BoF for CCIFF (http://cciff.ca) at SIGGRAPH 2016 and CCIFF's pre-launches in Montreal and Beijing. Song and Mokhov are leading a parallel development of the ISSv3 using Unity3D, Vuforia, and Kinect SDK for a mobile app and an immersive installation. The app has been featured at SIGGRAPH 2015 *Appy Hour* [Zhang et al. 2015] and its AR/VR extension at the SIGGRAPH Asia 2016 workshop [Bardakjian et al. 2016].

2.3 Sudhir P. Mudur

Sudhir Mudur obtained his Bachelor of Technology (honours) in 1970 from IIT Bombay and his PhD in 1976 from the Tata Institute of Fundamental Research in Mumbai, India. His interest in computer graphics started with his undergraduate thesis project. Since then he has been actively researching the field of computer graphics, particularly the areas 3D modelling, global illumination, virtual environments and applications in CAD/CAM and entertainment. Over this period of more than 3 decades, he has published papers in top computer graphics venues and supervised a large number of doctoral and graduate students, many of whom are well established in the field. His work in the areas of robust geometric computing using interval arithmetic and in global illumination models is well cited. He has extensive experience in working with engineers, artists, animators and designers on projects in areas including aerospace systems, animation, games, immersive environments, arts exhibits, textile arts, font design and Chinese opera.

Mudur is currently a professor and chair of the department of computer science and software engineering at Concordia University. He has been teaching computer graphics and related courses at the undergraduate and graduate level for over 30 years. With his long experience in curriculum development, he has developed a number of new courses in computer graphics and games. He has served as a member of the Editorial Board of a number of computer graphics journals.

Mudur is a senior member of IEEE, member of ACM, member of Eurographics and a member of SIGGRAPH Pioneer Group.

2.4 Peter Grogono

After obtaining B.A. and M.A. degrees in mathematics from Cambridge University, Peter Grogono worked in various fields, including pattern recognition, civil engineering, operating systems and electronic music. In the late 1960s, while at Electronic Music Studios in Putney, England, he designed and implemented MUSYS, a software system for music synthesis that ran on DEC PDP/8 computers.

Grogono moved to Montréal, Québec, in 1973. He acted as a consultant for several years, working on projects as diverse as railroad ticketing and business accounting, before joining Concordia University to advise professors with their computing projects. He obtained a masters degree in 1980 and a Ph.D. in 1984, both in Computer Science, before joining the Department of Computer Science and Software Engineering at Concordia. He contributed to the department's teaching by extensively revising the undergraduate Computer Science program and introducing Software Engineering programs for undergraduate and masters students. Grogono was awarded the first Engineering and Computer Science Faculty Award in 1998 and the first President's Award for excellence in teaching in 2007. In 2001, he helped students to organize the first Canadian University Software Engineering Conference (CUSEC).

The conference has been held annually since then and Grogono is recognized as its "founding father".

Grogono's research areas include programming languages, expert systems, graphics, and software engineering. He has published seven books and more than one hundred research articles. He is now Professor Emeritus, having retired from Concordia University in 2014. He now pursues his hobbies, which include photography and music, while continuing to conduct research into programming languages and graphics.

3 INTRODUCTION

We begin with the introduction to the interactive media and gradually progress into technology specific details.

3.1 Course Rationale

It is founded in a studio-based course in the creation and real-time processing of moving textures and video with OpenGL. This course is particularly applied to installation and performance arts practice. It provides an introduction to interaction approaches for the real-time processing of 2D and higher-dimensional arrays, image and video filters, motion segmentation, and tracking blobs, optical flow, faces, and shapes.

3.2 Tentative Length and Level of Difficulty

- Half Day Course
- Beginner/Intermediate

3.3 Intended Audience

Interaction design artists, OpenGL enthusiasts and developers; computer graphics students (undergraduate and introductory graduate and intermediate-advanced) and student volunteers.

3.4 Course Prerequisites

Some familiarity with graphics concepts, OpenGL, transformations, and programming in general, but no advanced level is required.

Software requirements:

- Clone OpenISS from GitHub
- Download 30-day trial version of Max from http://cycling74.com [Cycling '74 2015]. Relatively modern OS X or Windows laptop for hands-on materials for those who want to try if no computer lab is available. We will bring the interaction devices, such as Kinects and Wii controllers.
- Download *Processing 2.x* [Fry and Reas 2015] (it comes with built-in JDK 7). Its installation is as simple as unpacking it where it was downloaded and running it once.
- Additional libraries will need to installed for both Max and Processing; we will provide a bundle for download.

3.5 Pedagogic Intentions and Methods

This is intended to be a very practical course as in learning by doing, even for the non-hands-on participants. The hands-on approach is the best way to follow this course. The use of Jitter/Max and Processing makes it easier to grasp the concepts due to their artist-friendly setup.

3.6 Special Presentation Requirements

For willing participants to do the hands-on work as a minimal requirement, a computer equipment with OpenGL, Processing 2, Java SDK 7, Max 6 (32-bit) installed to be able to run the examples on OS X or Windows 7 is needed. Additional open source libraries will be required to be installed to work with Kinect and other devices. We prepared a prepackaged set of libraries to download. Unfortunately we cannot supply Kinects and Wiis for all, but many participants may have their smart phones and tablets where some interaction would work directly; if time permits, the participants can try interactivity at the instructor's place after the course and our demo or use their depth cameras.

The participants may also simply listen through, take notes, and do their trials on their own devices. Traditional paper style notes are provided as PDF. The participants are encouraged to install and explore the mentioned software tools prior coming to the course if they prefer to use their own laptops. Notes and the sample code will be made available to the participants where the code would contain the extra comments and practice material.

3.7 Summary

We first use Cycling 74's Max, MSP, and Jitter (commonly known together as simply *Max*) that have documentation that includes fantastic tutorials [Elsea 2013] on all aspects of the software. However, we focus on specifics of integrated multi-device interaction using sensors/controllers such as Kinect, Wii, and iDevices connected to OpenGL for processing in a fast prototyping manner for digital media production.

The attendees will need a 30-day trial version of Max 6.x [Cycling '74 2015]; other libraries and programs such as plug-ins, externals, and *Processing* [Fry and Reas 2015] are free and/or open-source. This setup does not require extensive programming knowledge and the course is structured along Max's visual dataflow programming language. (A FOSS alternative to Jitter/Max, *PureData* [Puckette and PD Community 2014] may be introduced if time permits, but the course currently is centered around Jitter/Max and Processing.)

The second environment is *Processing* [Fry and Reas 2015], a Java-based IDE that is easy to start working with in OpenGL and develop graphical and interactive applications. Instead of *patchers* as in Max, it has a notion of *sketches*, which are also easier to grasp for artistic-minded than traditional programming in imperative languages like pure Java, C++, or C#.

Processing and Max are then made to communicate with each other via *Syphon* (OS X) [Butterworth and Marini 2013; Colubri 2014] or *Spout* (Windows) [Joris and The Resolume Team 2014] to share video frame data as well as UDP-based OSC [Schlegel 2011] protocol to share coordinates and other data structures between the two applications.

The proposed environment is very affordable to most practicing digital media artists, VFX designers, and programmers to compose interactive graphical applications for stage.

All the sample patchers and sketches demonstrated in class will be available from the course instructors. A useful list of externals and resources will also be maintained.

Followed the educational session, we then share our experience in deploying Max/Processing setup as a part of *Illimitable Space*

System v2 (ISSv2) as a case study, on stage, along with the lessons learned.

Suggestions for additional readings will is provided for each course topic. There are additional useful resources on the subject that we may refer to for one concept or another throughout the course. They are listed under the "References" section: [Böttcher 2013; Cycling '74 2015; Elsea 2013; Grogono 2002; Manzo 2011; Microsoft 2012a; Molich and Nielsen 1990; Pelletier 2012; Puckette and PD Community 2014; Rogers et al. 2011; Song 2012; Song et al. 2014a; Stone et al. 2005; Ursu et al. 2009].

Relevant and helpful courses from past SIGGRAPH events on various aspects on color theory in digital media, pipeline design patterns, capturing human body, videography, and storytelling for a variety of applications can be found at [Caldwell 2015; LaViola 2015; Li et al. 2015; Polson 2015; Rhyne 2015; Richardt et al. 2015].

4 PRODUCTION HISTORY WITH ISS AND DEMOS

Illimitable Space System (ISS) is a real-time interactive configurable toolbox used to create visual effects and musical visualizations based on input such as voice or gestures with the corresponding image mapping and multiple input devices [Song et al. 2014a]. We use Illimitable Space System v2 for rapid development of real-time motion-based graphics apps for stage implemented in Processing and Jitter. (Our previous experience using the previous iteration of the ISSv1 system is documented in [Song et al. 2014a,b] where we used a different set of technologies.)

We share our research-creation for a real-time production for an interactive dance show that took place in Montreal, Quebec, Canada as part of the Chinese New Year Gala 2015 on February 14, as well as District 3 Demo Day and the corresponding research work on projection, image mapping, multiple-camera inputs, irregular surface projection, as part of the production and future work.

We show what is possible with the ISSv2 pipeline if time permits:

- Gray Zone quick app demo
- Live dance demo with projection mapping (similar to the District 3 Demo Day and CG in Asia at SIGGRAPH 2015 [Song et al. 2015b])
- Lion face tracking demo
- Physics and falling candies
- An interactive texture distorter patch
- Projection mapping to irregular surface

4.1 ISS Overview

ISS is a flexible and configurable research-creation entertainment software for deployment on stage for real-time motion-based applications, for dance and other shows. ISSv1 supports interactive documentary and voice-based interaction. ISSv1 was exhibited SIGGRAPH Asia'14 and VSMM'14 in 2014 [Song et al. 2014a,b]. ISSv1 was used in Chinese New Year Gala 2014, in the Ascension dance (two days of production in Montreal) [Song and Mokhov 2014] and in three scenes of the Like Shadows theatre production in Beijing, China for about 10 days and 900 people in audience [Song et al. 2014a].

In 2015, the production team created and used a subset of the features of the ISSv2 toolbox for interactive artistic performance Gray Zone [Song et al. 2015b] including enhancements on projection, image mapping, irregular surface projection, multiple-camera tracking and to study and evaluate the user experience before 300 or so audience.

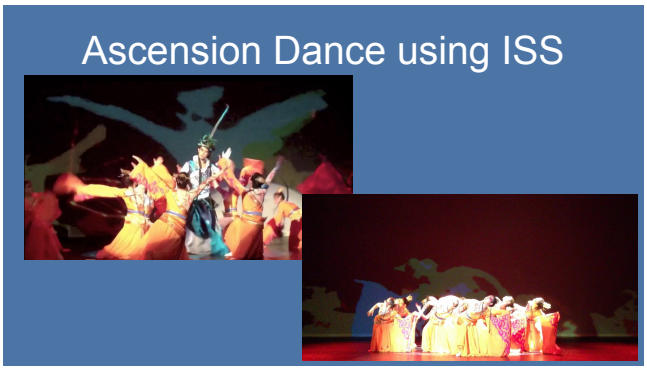


Figure 2: *Ascension* Dance Using ISSv1

ISSv2 switched to other technologies as opposed to ISSv1 (that used XNA and C# on Windows) to allow for rapid turn out with computation artists. Max- and Processing-based rapid prototyping environment is ideal for entertainment computing, as well as for artists and real-time performances using interactive graphics. With this course we share the expertise we developed in connecting the real-time graphics with on-stage performance with the Illimitable Space System (ISS) v2.

Figure 3: *Like Shadows* Theatre Production in Beijing Using ISSv1 in 2014

4.2 ISSv2 Production History

As mentioned, ISSv2 was used for an interactive dance shows at the production for the Chinese New Year Gala 2015 (https://vimeo.com/121177927) and Chinese New Year Gala 2016 and District 3 Demo Day Events (https://vimeo.com/129692753, https://vimeo.com/130122925). The dance was almost five minutes long and used four different visualizations including both music visualizations and visual effects using Jitter/Max and Processing.

The District 3 Demo Day events had 5-minute live performance presentations, based on dance and projection mapping on the body using Processing [Fry and Reas 2015].

In ISS, the requirements change according to each production, so we adopted the agile methodology to confirm the requirements by building different profiles for specific production [Mokhov et al. 2016]. In every profile the visual effects are configured according to the nature and demand of the production. After the configuration is done we design the system according to the visual effects and implement it in the Processing IDE [Fry and Reas 2015] with the help of Java and other libraries like OpenGL. The system is tested using Kinect in the research lab and acceptance testing is done on the day of real production on the stage. With the help of feedback from testing, the system is adjusted accordingly. We reused the components of the system according to the need and optimized the system after every production to make it better. See most recent rendition at TEDxConcordia, see [Mokhov et al. 2017] as well as ChineseCHI (https://youtu.be/8nxxzClnx_I).

Figure 4: *Gray Zone* Dance Using ISSv2

Prior to the productions, a month-long testing and experiments were done at the Hexagram black-box and white-box at Concordia University by the production team [Song et al. 2015b]. The research-creation team mainly focused on testing the musical visualization and motion-based visual effects including testing calibration, image mapping, and floor and wall projections. The blackbox had a similar condition as that of the actual theater used for the production.

There was a back wall projection for the main visual effects and a complementary floor projection. There is a subset consisting of four different modes for the ISSv2 combining both musical and video visualizations that made it into the dance production: line tunnel music visualization, glowing shadows, multi-colored shadows, and outline shadows as shown in the video and in Figure 1 [Song et al. 2015b].

Significantly more Processing-based Kinect and physical-based animations were produced for the District 3 Demo Day event as exemplified in the videos [Song et al. 2015a]. Relying on the prototyping technologies, OSS support, and versatile Kinect v1, we are still able to pull of real-time graphics, physical based modeling, and interaction on stage that is still appealing to the audience today for very small production cost.

As mentioned earlier, in the past we produced similar, ISSv1 Kinect-based installations featured on stage in a Chinese New

Figure 5: Distric 3 Demo Day Using ISSv2 in 2015

Year Gala 2014 show on January 2014 and the Like Shadows theatre production at the Central Academy of Drama in Beijing in April 2014. However, the development aspect and VFX prototyping in this version using Windows, XNA, and Visual Studio was tedious and less extensible. Switching technologies and working with VFX artists on a more open platform using either OS X or Windows with Processing and Jitter/Max allowed us a very rapid design, development, testing and deployment in real productions fast – within weeks or a month from starting the project to the actual production.

For motion tracking we used Kinect v1 in both ISSv1 and ISSv2 as well as regular cameras in ISSv2 expanding its possibilities, but without the voice-recognition component.

We are working on expand the ISSv2 set and already started on the ISSv3 using Unity3D with both Kinects v1 and Kinect v2 and other devices, and AR/VR technologies, for more immersive and gaming experience.

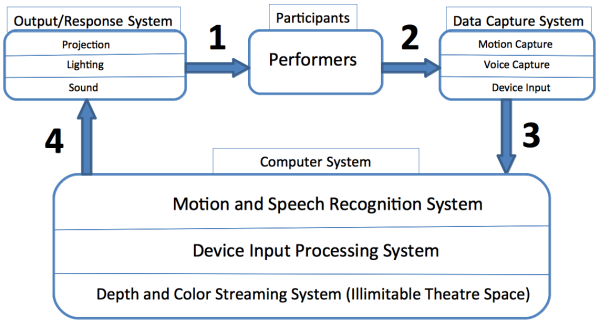


Figure 6: Conceptual Design of an Interactive Performance Installation

Regardless the version, the conceptual design is the same: there are four (4) components in the ISS illustrated in Figure 6: the participants, input devices (data capturing), computer system (for processing) and the output (response) system (e.g., projector) [Song 2012; Song and Mokhov 2014]. The input to the system can be gestures

or the voice commands from the participants, who are mainly the artistic performers or audience and these user interactions can vary with different types of users.

4.3 ISSv2 Components

The second version of the Illimitable Space System is more platform-independent and works with different communicating software systems as well. The research was done on a Mac OS X system with Processing and Max/MSP/Jitter installed. Even though the visual effects can be produced in Processing, for more portability, the ISSv2 passed the input from the Kinect to Max/MSP/Jitter and added the necessary effects there and then output to the projector. A middleware Syphon [Butterworth and Marini 2013; Colubri 2014] or Spout [Joris and The Resolume Team 2014] is used for transmitting the image frames from Processing to Jitter. The oscP5 (Open Sound Control protocol library for Processing) [Schlegel 2011] is used as an additional communication channel to pass short messages, such as skeleton coordinates from Kinect via Processing to Jitter.

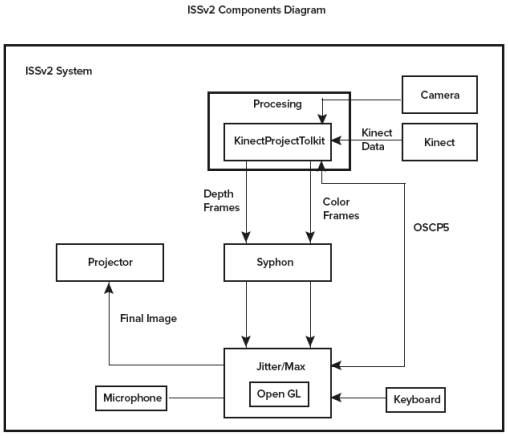


Figure 7: ISSv2 Components Diagram

Syphon acts as a video framesharing middleware between Processing and Max. The depth and color frames captured from input devices such as a camera or Kinect are sent to Jitter/Max by Processing through Syphon on OS X. A Processing library called KinectProjectorToolKit [Kogan 2014a] is used for connectivity with Kinect and calibration mapping with a projector. (An alternative for Syphon on Windows is Spout [Joris and The Resolume Team 2014].) The final image from Jitter/Max will be sent to projector.

4.4 Production Using ISSv2 and Visual Modes

When Gray Zone production happened on the 14th of February 2015 in Montreal in front of around 300 people, the theatre had two seating levels namely upper deck and lower deck. There was a wall projection for the main visual effects and a floor projection. There are four different modes for the ISSv2 combining both musical and video visualizations.

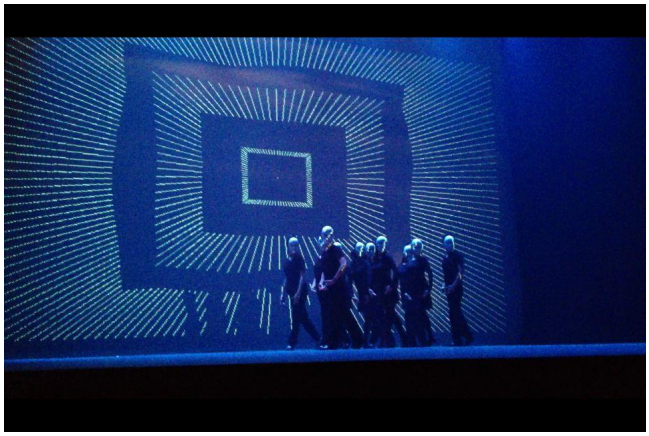


Figure 8: Line tunnel musical visualization

Figure 9: Glowing shadows

Figure 10: Multi-colored shadows

Figures 1(b)–1(d) show different modes made available to the ISSv2 for the production at the Gala. All the images are taken from the venue. The visual effects were manually sequenced with specific time intervals. The Figure 1(b) is a musical visualization while others are gesture-based visualizations. One Kinect was used to capture the dancer motions from the dance. A built-in microphone within the Kinect was used for the sound capture for the musical visualization.

Figure 11: Outline shadows

4.5 Difficulties Encountered During Production

Though enough testing was done prior to the production, the system still had different issues encountered during the production. During the pre-production testing, where side and top lights were more or less fixed before the dance, except the floor-pointing lights. Starting from the lighting conditions of the production environment, it was not controlled by the ISS production crew. This adversely affected the input to the Kinect. There was a duo scene in the dance where all the lights were off but two spot lights. During this performance, the visual effect used was the line shadow mode. Due to the high intensity of the spot light, the Kinect couldn't capture the participants reliably at some time points. The visual effect-generated border lines were not clear compared to the testing results. Also the stage was very wide and all the dancer were sometimes spread across the entire stage. This led to another problem in capturing the dancers.

The roof rails were not high enough for the projector which was used for the floor projection to cover the entire production stage. And due to the very suboptimal lighting conditions, the floor projections were not visible to most of the audience. Since the projector used for the main wall projection was at the very back of the auditorium, an HDMI extender using network cable was used to connect between the ISS operator's system which was at the side of the stage and the system connected to the projector (which was at the back of the auditorium). The short cable length of the USB which was used for connecting the Kinect to the laptop, the ISS operator had to sit beside the stage, close to the Kinect and performers. All the manual queuing was done from here.

Since there were many other performances apart from the interactive dance show, a main projector operator was present at the spot who was not a part of the ISS crew. As a result, right before and after the interactive dance, the projector HDMI cable had to be switched to that of the ISS with the projector system over the default projector input.

Even though there were four different visualization modes, the sequencing of visual effects during the performance was done manually. As stated above, due to the rigorous uncontrollable lighting conditions, capturing the performers was a hard task for the Kinect and ISSv2. The heat due to the lights, time of operation, the Kinect was not capturing the performers very effectively. This likewise adversely affected the visual effects displayed to the audience.

4.6 Image Mapping and Calibration

In real-time video production, especially in the interactive dance performances, image mapping is one of the can be an important features to be considered, such as the District 3 Demo Day. This is the process of projecting the visual effects directly on to the participants in real-time by identifying their position on the stage. In order to perfectly map the visuals, a perfect calibration of the production environment has to be done. The research team has used the aforementioned KinectProjectorToolkit [Kogan 2014b] to perform the calibration with the Kinect and a projector. The calibration process is done at the production environment with the help of a movable white screen. Even though the KinectProjectorToolkit is an integral part of the ISSv2, it was not exploited specifically for the Gray Zone dance's projection or the calibration, but instead as a source of user data, unlike in District 3 Demo Day where stage dimensions were smaller and more testing effort was put prior to the even for this purpose.

4.7 Ongoing Work (ISSv2)

- Scaling out to multiple linked devices and immersive installations with multiple Kinect 2's
- 3D projection

5 COURSE COVERAGE

Sample notes cover example materials and slides for the topics for the half-day course. We begin with the introduction of the use of OpenGL in Jitter/Max, then gradually build up on that by adding Processing, Kinect, Wii, iDevices, tracking, and shaders to with Jitter/Max and OpenGL rendering. We combined all these techniques to illustrate a subset of the presented in a case study we deployed as a part of the ISSv2 (Section 4) [Song et al. 2015b,a,b].

See the slide sets and posted examples linked from this page: http://mdreams-stage.com/courses/s2019

5.1 Introduction to Max/MSP/Jitter

Max [Cycling '74 2015] a dataflow graphical programming language that originated from PureData [Puckette and PD Community 2014] (pd, free, open-source), which in turn was influenced by Lucid [Ashcroft et al. 1995; Ashcroft and Wadge 1977; Wadge and Ashcroft 1985]. Max started off and has become more or less standard tool used in the music technology field for electronic, music composition, music control, and various other tasks for realtime music [Manzo 2011; Winkler 2001] performances in the computation arts community.

Originally, developed by Miller Puckette in mid-1980s at IRCAM, Paris, it was ported to NeXT and ISPW boards in the late 1980s and early 1990s. Its first commercial version for Macintosh computers was released in 1991 under the lead of David Zicarelli, who began distributing MSP viaÂăCycling'74Âăin 1999 and Max since 2000. Max/MSP ([Winkler 2001]) was augmented with Jitter later on to add video processing and graphical capabilities to Max programs; Max/MSP/Jitter has become simply Max.

- Max – the language covering all aspects on:

- MSP – (audio) signal processing
- Jitter – graphics processing

Max allows further export its patchers as a standalone apps.

We briefly go over Max environment, essentials and its visual dataflow programming language.

5.2 OpenGL in Jitter

Jitter is a visual part of Max that extends its dataflow language to manipulate graphics, often used for visual effects by VJs and the like accompanied with a sound. OpenGL in Jitter is a built-in wrapper that exposes an API to OpenGL functions via the `jit.gl` objects. In this module we cover some drawing controls, advanced color manipulation, textures mapping, transformations, and other API of interest. This is very easy to prototype OpenGL-based apps this way.

Some of the API covered here is:

- `jit.gl.render`
- `jit.gl.plato`
- `jit.gl.model`
- `jit.gl.mesh`
- `jit.gl.sketch`
- `jit.gl.videoplane`
- `jit.gl.gridshape`

5.3 Kinect, OpenGL, and Jitter

In this module we work through the connectivity of Max patchers to Kinect (v1). The first way to connect Kinect to a Jitter patch, is directly via a Max external, such as *Synapse*. Other options exist, such as wrappers `jit.freenect.grab` (based on `libfreenect`), `jit.openni` (OpenNI), `dp.kinect` (on Windows, with Kinect SDK [Microsoft 2012b]). *Synapse* was known to work with model 1414 of Kinect, but not 1473 or Kinect v2. It also tracks only a single skeleton. (To track more than one skeleton or Kinect device or version we'll use Processing in Section 5.8 that we can then connect to Max via Syphon/Spout and/or OSC).

5.4 Wii and Jitter

In this short module we show how to connect the Wii controller to a Jitter patch using the OSC protocol and the `OSCulator` program. OSC messaging requires UDP-based networking available.

- http://www.osculator.net

5.5 iDevice Touch, Video, and Jitter

In this module we show how to extend some of the controls into touch devices, such as an iPhone or an iPad that can be mapped to slides and other controls in Max. Likewise, working with a camera is also possible with some limitations.

- We will use *Fantastick* (http://pinktwins.com/fantastick/) (free) app as a control example.
- Other example apps exist for a small fee: *TouchOSC* and *C74*.

For camera access:

- Pocketcam (free)
- iWebcamera (paid)

5.6 Tracking

The tracking module covers applications of computer vision techniques using OpenCV in Jitter to manipulate video frames to extract and track objects or color of interest in them in realtime.

API of interest includes:

- `jit.rgb2luma`
- `cv.jit.track`
- `cv.jit.threshold`
- `cv.jit.label`
- `cv.jit.blobs.centroid`
- `cv.jit.blobs.sort`
- `cv.jit.blobs.binedge`

5.7 Shaders in Jitter

This module goes over the most common shaders concepts in Jitter. Since the use of `jit.matrix` is mostly CPU bound, some speedup can be update via shaders since they use GPU, just like in traditional pipelines.

API of interest:

- `jit.gl.shader`
- `jit.gl.slab`

5.8 Depth cameras and VFX in Processing

- OpenProcessing sketches (e.g., Curtain)
- Projection mapping [Kogan 2014b]
- OpenCV and blobs [Borenstein 2013]
- Syphon [Butterworth and Marini 2013; Colubri 2014] and OSC [Schlegel 2011] to connect Processing and Max.

5.9 Putting it all Together

Here we integrate a sample application that uses some of the presented examples and we discuss performance issues.

- Simultaneous multimodal and multidevice interaction.
- Sound-based input and output.
- iPhone or iPad control to brighten or darken animations, to speedup or slowdown and other forms of control.
- Kinect based or/and Wii-based interactive animation and/or sound controls.

5.10 OpenISS

Open Illimitable Space System is a collection of open-source libraries and toolkits which provides a platform for the artists to augment art, entertainment and technology by leveraging multimodal interaction tools with capabilities including but not limited to motion capture, image processing and visual effects etc. that can be fused with an artist's performance in real time with an immersive visual experience. OpenISS is an open source core for ISSv2 [Song et al. 2015a]–an interactive system for artistic performance controlled by gestures and voice. OpenISS was started by Serguei Mokhov as an education project in Linux/Unix programming primarily in C as a tool to build and link complex projects and systems. Recently, Mokhov's idea to extend these capabilities as a service was proposed to a group of students here at *Concordia University* by providing SOAP and REST API in a scalable ecosystem of the components within `OpenISS`. A partial yet significant work has been achieved in

this process and a subset of the functionalities of these components can now be made available in the form of web services [Psimoulis et al. 2018].

The further idea to expose some of the OpenISS functionality as composable services continues with this work.

OpenISS provides image-processed frames by leveraging various algorithms normally for real-time processing of the images and rendering a final image. It is designed to do so by querying the Java wrappers of the devices or the image generators (simulators), depth cameras (Microsoft's Kinect), or webcams, or phone cameras to get a given image at a given instance. Then it is designed to be able to remix images from multiple sources, including Magenta [Google LLC 2018], OpenCV [Intel Corporation et al. 2018], or user-submitted and then returning the resulting images as a service in a browser or a web application.

The WSDL and REST wrappers around the OpenISS component instance enables the client to not only request the service and query the current frames at a source (service) and display the image to the client itself but also submit a source image from the client to the service along with a desired operation call on the same that "mixes" the submitted image with the service source image and produces the resulted "mixed" image in response as described earlier.

In general, the services we would like to expose are all available via both SOAP and REST APIs. For the SOAP API, the required OpenISSSOAPService class was implemented with initially two API calls doCanny() and contour(). They acquire an image from the devices and apply canny edge detector and contour extraction on the image respectively. For the REST API, **/openiss/opencv/canny PATCH** would set the canny application to true to no matter what image you get from the client POSTed image or the image one requests from the devices. The same idea applies to **/openiss/opencv/contour PATCH** would set the contouring application to true to the target images.

Thus the overall idea is to expose a real-time image capture and processing application with visual effects as a service to both SOAP and REST clients to enable its use at a wider scale, in particular during an ISSv2 live dance or theatre performances where viewers may not be physically present but streaming or want to decorate their image experiences differently, in near real-time. Ideally, regardless of REST or SOAP it is a stateful web service which provides interactive capabilities to the end user via composition of images or even POSTing their own images and composing with other sources. A use case diagram of our API's can be seen in the Figure 13 illustrating various actions a user can perform with our service. However, this illustration must be seen as a subset of such use cases that are a part of either ongoing or future works of this active research.

5.10.1 SOAP API.. General overview of the SOAP architecture can be seen in the Figure 12 OpenISS components are exposed via SOAP API under a specific submodule libfreenect [OpenKinect Contributors 2018] which includes a server backend startup and a wrapper for it using the libfreenect's Java wrapper API. Initially, the server backend is a recording from a real device, which can provide RGB-Depth data, essentially replaying frames using fakenect replay tool and a small pre-recorded RGBD dataset on the loop.

Subsequently, OpenISSSOAPService translates the HTTP SOAP requests to the Java wrapper of libfreenect to basically grab

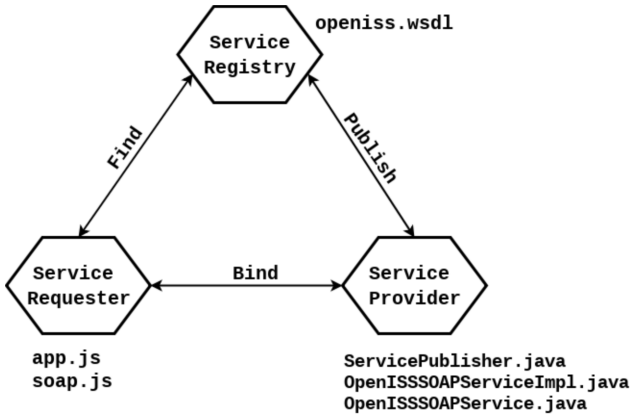


Figure 12: SOAP Architecture

the current frame. *IFF* the backend is not available, it returns a default template image with a text in it simply saying "OpenISS Capture Server Is Not Currently Running." Thus, this results in OpenISSSOAPService exposing two methods: getFrame(type) – returns current frame type (depth or color) from the libfreenect's output running in a loop; and mixFrame(image, type, op), that POSTs the image and "mixes" it with the desired frame and returns the resulting processed image via the desired algorithm.

5.10.2 REST API.. The same components of OpenISS are exposed as RESTful resources as well. **/openiss/depth** GET – which returns the current depth image. **/openiss/rgb** GET – returns current RGB image. A stub API is provided as well for skeleton data extraction. I likewise, returns a "service unavailable" error code if depth is not enabled on the service side. A work in progress is Magenta [Google LLC 2018] to make it a git submodule of OpenISS along with a Magenta client that talks to the OpenISS backend using REST and uses one of the Magenta's sample artistic AI models.

An observer-listener "push-server" and "receive-client", or more rather like peer-to-peer push operation. In this scenario the client can register itself (IP, port) with the server running a depth camera or fakenect instance using SOAP and REST registerPeer API. Then, the server, possibly in multiple threads, posts the current frame to the list of registered "receive-clients", that receive the image and store it and refresh it locally for the display.

5.10.3 Requirements Specification. This current implementation work was set based on a collection of specific requirements, a subset of which was implemented and released on https://github.com/OpenISS/OpenISS; the rest of the requirements are an ongoing work. What follows is a nearly complete list:

(1) Setup and document a local copy of OpenISS in terms of main service components, how to deploy, and how to run.
    In OpenISS/src/api/ create a subdirectory for java/openiss/ws/soap/.
    Create classes OpenISSSOAPService and OpenISSSOAPClient.
    Have OpenISSSOAPClient in js/openiss/ws/soap/ for a NodeJS-based SOAP client.

Define a server backend startup and a wrapper for it using `libfreenect`'s Java wrapper API.
The backend shoud be able to work not as a real Kinect device, but a recording from a real device, replaying frames using `fakenect` replay tool and a small pre-recorded dataset on the loop:
https://bitbucket.org/openiss/public/downloads/test-recording.tar.bz2
`OpenISSSOAPService` should translate HTTP SOAP requests to the Java wrapper of `libfreenect` to basically grab the current frame.
IFF the backend is not available, return a default template image with a text in it simply saying "OpenISS Capture Server Is Not Currently Running."
Thus, this results in `OpenISSSOAPService` exposing two methods: `getFrame(type)` – returns current frame type (depth or color) from the `libfreenect`'s output running in a loop; and `mixFrame(image, type, op)`, that POSTs the image and "mixes" it with the current frame and returns the resulting image ("mixing" for now can be assumed for `op` to be a simple plus (+) of two images, clipped at the image dimensions whichever is smaller.)

(2) Call a Java OpenCV processor and the option to invoke OpenCV optionally, such as the SOAP client can set an OpenCV option and define a service call `OpenISSSOAPService` for it.
Provide a SOAP calls for `doCanny()` in `OpenISSSOAPService` that take either a POST image data from the client and return edge-processed image back, or if depth or color type specified and the depth camera code is there, call it, to get the frame `getFrame()` (color or dept) and `doCanny()` on it and return it to the client.

(3) Do the same with `libfreenec2` using a real device (ask instructor) and the Java wrapper can be used from OpenKinect-for-Processing library.

(4) Expose OpenISS components as RESTful resources. Design and document JSON states transferred between the client and service components.
  (a) Refactor `OpenISSSOAPService`'s calling the imaging functions of `libfreenect` an `opencv` into a util module, `openiss.util.OpenISSImageDriver`.
  (b) Implement `openiss.ws.rest.OpenISSRESTService` to invoke `OpenISSImageDriver` with the REST-like operations below. The design decisions should be thoroughly documented. This is like a filtered pipeline implementation, where both or either depth or color images are returned after a series of filters applied to them either through mixing or OpenCV.
    (i) **/openiss/depth** GET – returns current depth image; POST/PUT/PATCH/DELETE – currently undefined
    (ii) **/openiss/rgb** GET – returns current RGB image; POST/PUT/PATCH/DELETE – currently undefined
    (iii) **/openiss/mix** PATCH – sets the mixing flag on to apply images internally to either "patching" the specified base frame (**/depth**, **/rgb**) with **/depth**, **/canny**, or **/custom** image passed as a request. These can be combined.
    PUT/GET/POST currently undefined; but one of the above GET methods will respect the mixing flag and image.
    DELETE – would unset the flag turning off mixing.
    (iv) **/openiss/opencv/canny** PATCH – would set the canny application to true to the GET'ed images above.
    DELETE – would unset the flag turning off canny application to the current frames.
    PUT/POST/GET – currently undefined
    (v) **/openiss/opencv/contour** PATCH – would set the contouring application to true to the GET'ed images above.
    DELETE – would unset the flag turning off contouring application to the current frames.
    PUT/POST/GET – currently undefined
  (c) Provide an async REST JavaScript Client to refresh the page and get new images, both depth and RGB to test the above API.
  (d) Test with a real Kinect depth camera on the service side. Document the deployment setup and requirements.
  (e) Provide an initial integration with Magenta/p5.js (https://github.com/tensorflow/magenta). Magenta would need to become a git submodule of OpenISS. Will need to define a Magenta client that talks to the OpenISS backend using REST and uses one of the Magenta's sample models.
  (f) Provide a stub API for skeleton data extraction. Return a "service unavailable" error code if depth is not enabled on the service side.
  Recompile `libfreenect` to support skeleton data extractions with the OpenNI2 driver enabled.
  See `SimpleOpenNI`'s mapping of the skeletal joints to Java constants.
  The resource would be **/openiss/skeleton/<id>** and a GET method that would return a JSON object of joint mappings and values. Return service unavailable if **id** is not valid; there may be no **id**'s, if there are, they start at **1**. Can go up to **6** (Kinect 1 up to 2 and Kinect 2 up to 6 skeletons but both can track 6 users). Querying with GET **/openiss/skeleton** would give how many users are currently visible in the frame.
  Write a JavaScript client for now simply displays the values
  Write Java client would parse JSON and get back Java-data structure joints and their values.
  (g) test with a web cam as a image source, and OpenCV processing
  (h) implement **/depth2**, **/rgb2** based on `libfreenect2` with the support for related machinery as above

(5) Implement observer-listener "push-server" and "receive-client", or more rather like peer-to-peer push operation. In this scenario the client registers itself (IP, port) with the server running Kinect or fakenect instance using SOAP and REST registerPeer API.
Then, the server, possibly in multiple threads, posts the current frame to the list of registered "receive-clients", that receive the image and store it and refresh it locally for the

display. The receive-client peer can be written in any language, such as Java, PHP, or in-browser.

(6) Implement the same for OpenCV as per above using a decorator pattern; that is an OpenCV push-server would use the whatever framesource it has locally (Kinect or otherwise), passes its frames through `doCanny()`, and pushes the result to the client.
The two implementations may share register and push logic.

(7) implement a simple sequential BPEL process to access depth data by a client

(8) implement simple BPEL process, image from the client, apply `doCanny()` (SOAP), return processed image.

(9) Write a composite BPEL service:
First, in sequential push freenect, to another service doCanny to mix using the above two services.
Then, use the result in a parallel activity to distribute to four OpenISS push-clients in parallel.
The main BPEL client to this composite service simply makes the initial request, and then is notified in the end that all push-clients were notified (or some failed).

5.10.4 Dependencies. In order to help someone who wants to reproduce the project or deploy our services, it is important to point out all the dependencies the system needs. For the OpenISS and Image Processing pipeline the system will require namely, the following libraries, `freenect`, `fakenect`, `JNA` and `OpenCV`. For the SOAP service, the systems depends on `JAX-WS`. And, for the REST service, `JAX-RS`, `Jersey`, `maven` and `Glassfish` or `Tomcat` are required to reproduce similar results.

See Also

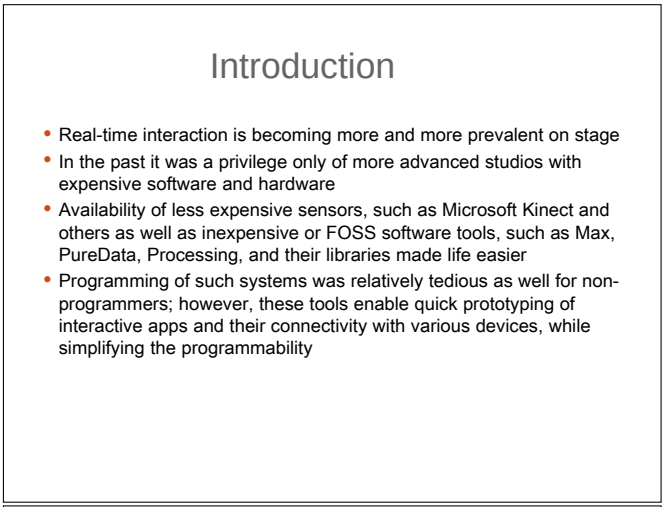
- http://cycling74.com
- http://opensoundcontrol.org
- http://www.osculator.net
- http://pinktwins.com/fantastick/
- http://openprocessing.org
- http://projection-mapping.org

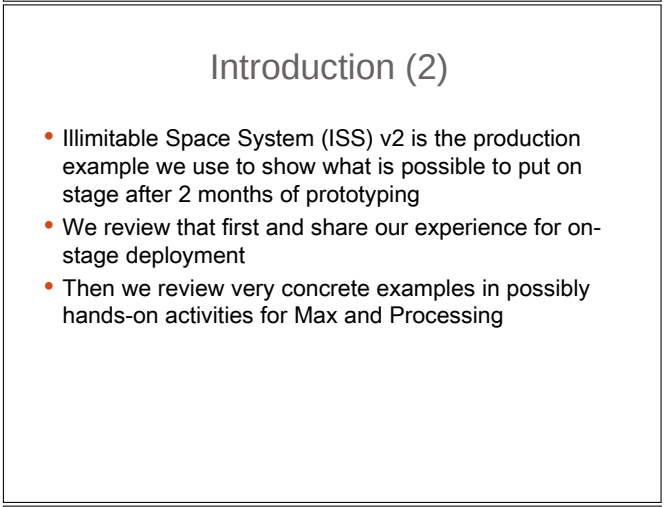
ACKNOWLEDGMENTS

We acknowledge the reviewers of this work and their constructive feedback. This work was sponsored in part by SIGGRAPH Asia 2015–2016, Faculty of Engineering and Computer Science (ENCS), Faculty of Fine Arts (FOFA), Concordia University, Montreal, Quebec, Canada.

Special thanks to:

- mDreams Stage Research Creation Group
- OpenProcessing community
- Projects: OpenCV, SimpleOpenNI, KinectProjectionToolkit
- Open Source community





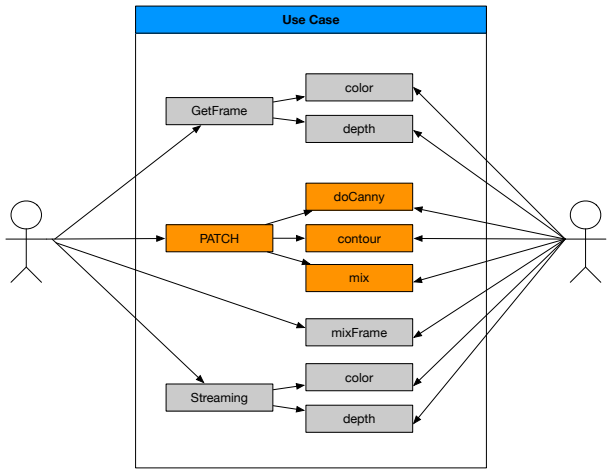


Figure 13: Use Case Diagram of OpenISS OpenCV and Depth Camera APIs

Background

- Miao Song. *Computer-Assisted Interactive Documentary and Performance Arts in Illimitable Space*. PhD thesis, Special Individualized Program/Computer Science and Software Engineering, Concordia University, Montreal, Canada, December 2012. Online at http://spectrum.library.concordia.ca/975072 and https://arxiv.org/abs/1212.6250.

From Traditional to Modern Theatre (1)

- In Laurel's very early work in 1991, Computers as Theatre, ***she applied her knowledge of theatre to Computer-User interface design***.
- She says, *"In many ways, the role of the graphic designer in human-computer interaction is parallel to the role of the theatrical scene designer."*
- Foreword: Computer as Theater (2013, 2nd edition) by Don Norman, *"Shakespeare Said the World Is a Stage: For Us, Computer Applications Are Our Stages"*

Performance Arts and Theatre Production


Beijing Opera
San Cha Kou


Modern Chinese
Drama Teahouse

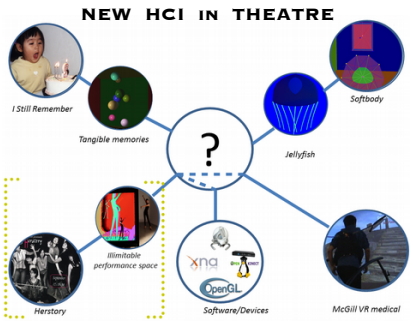

Japanese Noh Stage


Marciel Hallucine

Up Wake


The Silhouettes Dance
Group Performance

OVERALL RESEARCH-CREATION



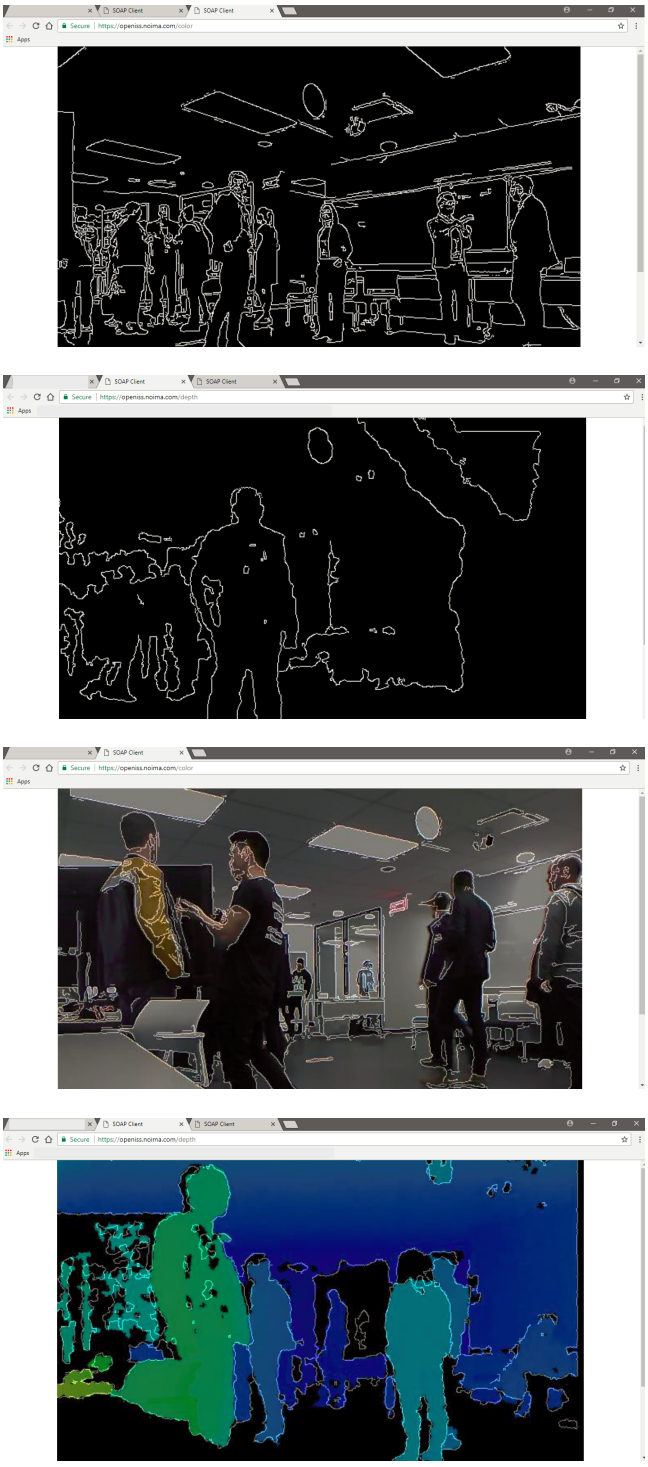
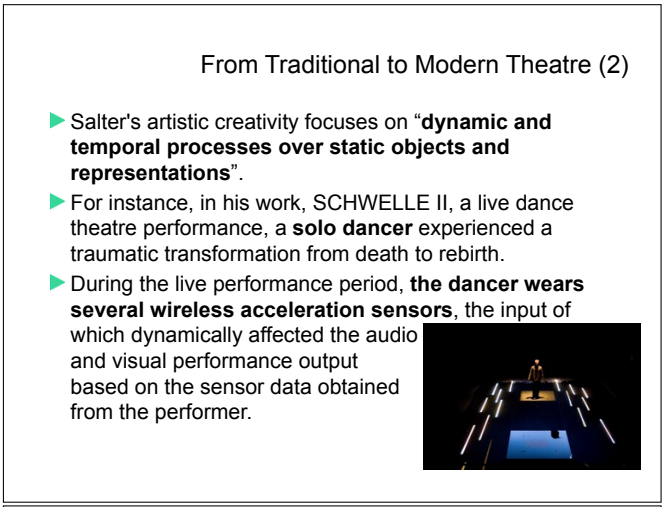
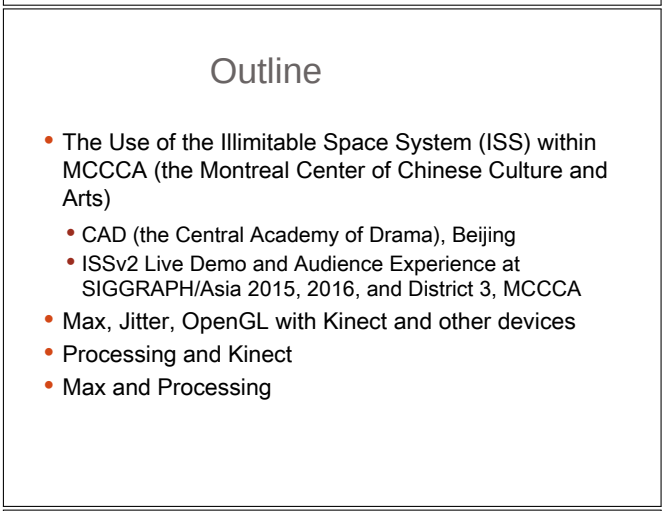
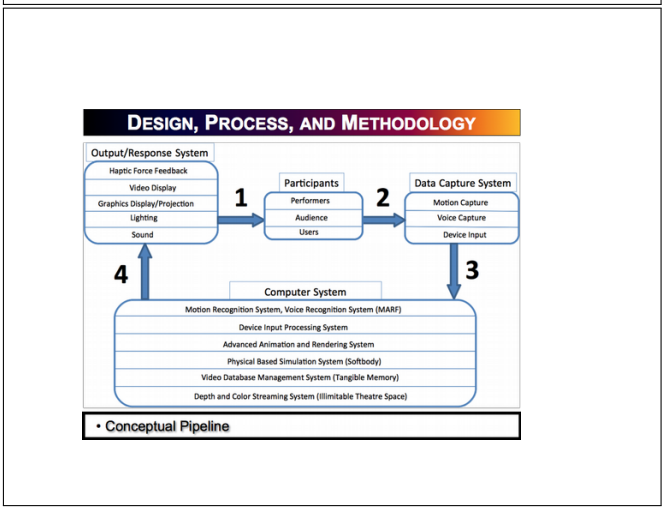


Figure 14: Sample OpenISS Output Rendered in a Browser





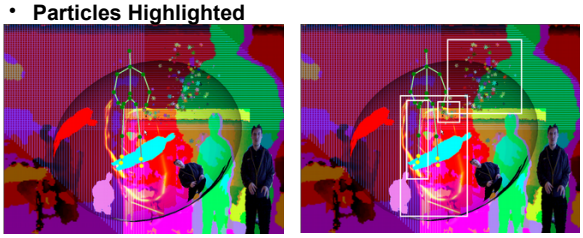


From lab to Production (2011-2013, ISSv1)

ISSv1 with "Like Shadows" at CAD

ISS -- A REALTIME ARTS TOOLBOX: TESTING

- **ISS Skeleton-based Capture and Interaction**
- **Particles Highlighted**



- Real-time motion capture, physically based simulation, music visualization, green-screening, video processing for HCI in performance arts and documentary film.

ISSv1 Technologies and Their Limitations

- Windows
  - XNA 4
  - Kinect 1.x SDK
  - HLSL
- Slow/tedious prototyping and testing
  - ~1.5 years in development before real production
- Not artist-friendly (programmability wise)
- Not portable

Ascension Dance using ISSv1 (2014)

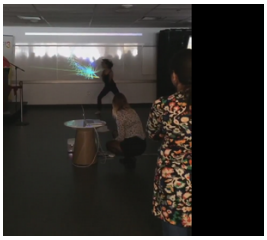
Gray Zone Dance using ISSv2 (February 2015)

- https://vimeo.com/121177927

District 3 Demo Day (June 2015)

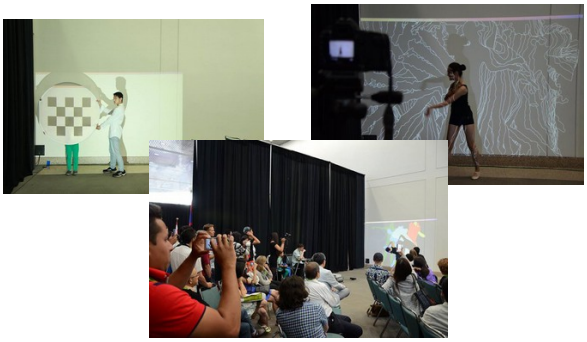
- https://vimeo.com/130122925
- https://vimeo.com/129692753

District 3 Ribbon Cutting (September 2015)



SIGGRAPH International Resources (August 2015)

- https://vimeo.com/141811579



Technologies in ISSv2

- Max 6.x (Gray Zone)
- Processing 2.x (all productions)
- Libraries
  - SimpleOpenNI
  - Syphon / Spout
  - oscP5
  - OpenCV
- ~2 months of development before production
  - Subsequent runs past Gray Zone were smaller 1-3 weeks

Nanjing Week (September 2015)

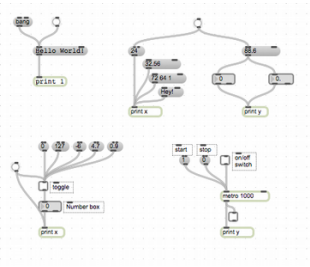
- https://vimeo.com/141081567

A Brief Introduction to Max/MSP/Jitter

Max/MSP Brief History

- Developed by Miller Puckette in the mid-1980s at IRCAM, Paris
- Ported to NeXT and ISPW boards in the late 1980s and early 1990s
- Commercial version for Macintosh computers released by Opcode in 1991 under the lead of David Zicarelli
- David Zicarelli began distributing MSP via Cycling '74 in 1999 and Max since 2000

Max Hello World



`max-hello-world-master.maxpat`

Max/MSP Overview

- Generally hides or takes care of low-level programming
- Designed for "real-time" performance
- Written in C
- Extensible via external objects
- Macintosh support, Windows version released in 2003
- **Graphic and video processing extensions using Jitter**

MSP and Jitter

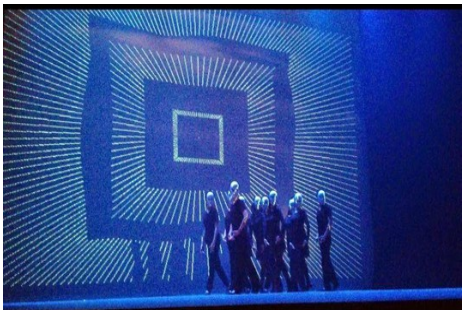
- **Max** is the core of the graphical programming language
- **MSP** is the sound/signal processing part of Max (~)
  - MSP: Max Signal Processing (or Miller S. Puckette)
  - MSP is an extension to Max for audio signal processing
  - The MSP objects were derived from the Pd signal processing infrastructure
- **Jitter** is the graphics processing part to Max

What is Max?

- A graphical data-flow programming language.
- Originated with PureData (**Pd**, free, open-source)
- Max is a standard tool used in the music technology field for composition, music control, and various other tasks.
- It provides a graphical interface and paradigm for modular programming.

Max/MSP & **Jitter**?

Jitter is the image processing part to Max



Purpose

- Good for testing complex solutions fast
  - (unlike C++)
- Excellent for quick prototyping
- Works with many sensors
- And many other reasons…

Introduction to Max Elements

max-elements.maxpat

Help Patches

Obtaining Max

Trial Version

Max: Beginning

- Max is easy to learn
  - However it requires spending some on it and its help files and tutorials
  - Learning by doing: from experimenting and making mistakes

Max

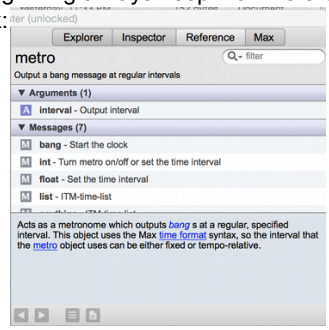
- We use … as of this slideset.
  - Max 6.x and 7.x are available, most of our examples were developed and tested on Max 6 and Max 7.
  - 30 day trial.
- Go to:
  - http://cycling74.com/downloads/
  - Download an appropriate version for your platform.
  - Install.
  - Run.
- Follow documentation:
  - http://cycling74.com/wiki/index.php?title=Max_Documentation_and_Resources

Max Environment: Familiarize First

- When learning on your own:
  - Making Patches from the help window home screen.
  - Max Tutorials up to Keyboard and Mouse input.
  - Referenced tutorials from Peter Elsea and others.
    - Covering all topics
    - Own collection of useful utility objects

Max Environment: Familiarize First

- At the beginning always keep Max Reference open as you work:



Max Environment: Fundamentals (2): Key Programming Elements

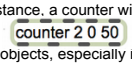
- Many similarities to programming languages and scripting
  - Variables
  - Functions
  - Boolean
    - Expressions
    - Conditionals
  - Arrays
    - 1D (signals)
    - 2D (matrices)
  - ...

Max Environment: Fundamentals

- The Max and Patcher windows
- Box data/flow types:
  - object, message, bang, button, number, floating point number, comment, ...
  - connections use *inlets* and *outlets* via *patch cords*
- Object help files (nearly all object have them):
  - While in edit mode
    - option-click on object
    - select object and use Help menu item
    - inlet/outlet information is available in the "Assistance area" (lower-right display bar) during "mouse-over"
    - select object and use Object->Get Info (command-I) for object specific settings

Objects

- The name of the object is its function.
- Arguments, if present, specify initial values for the object.
- Data come into the object via the inlets, and results are put out the outlets.
- Each inlet or outlet on an object has a specific meaning.
- If an object has several arguments, the order of the arguments determines the meaning.
- For instance, a counter with arguments of 2 0 50 will count from 0 to 50 and back:



- Some objects, especially in Jitter, also have attributes.
  - An attribute argument is an internal variable and is set with two or more parts: the name of the attribute with a @ and one or more values.
  - The order of @attributes is not important.

Max Environment: Fundamentals (2)

- Locking/Unlocking patches:
  - command-click on white space within Patcher window
  - Ctrl+E or command+E to lock
    - Or click anywhere in the patch outside an object, while pressing ctrl or command.
  - click the button in the lower left-hand corner of the Patcher window
- Unlocked mode (for editing, patching, may optionally run)
- Locked mode (for running, can change controls and values, but not edit)

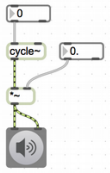
Inlets and Outlets

- Input/output. Need data types.
- E.g., the delay object will send a bang message out the outlet 100 milliseconds after a message is received in the left inlet. The right inlet will change the delay.

- Always right click and press help - or mouse-over for documentation.

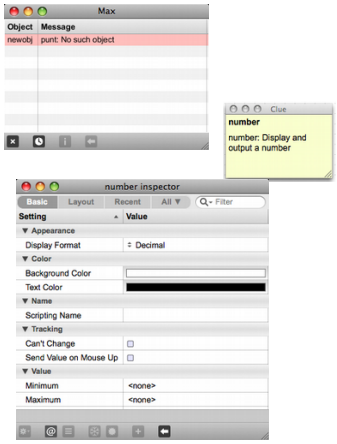
Patch cords aka Wires

- For sending data between objects
- Color coding:
  - a thin black cord is data (numbers, arrays, booleans)
  - a yellow wire is an audio signal
  - a green sending a video matrix (graphics)



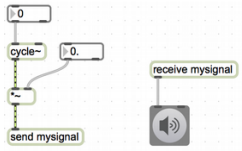
Windows

- The **Max** window contains information sent from Max (like error messages) or things you might like to print.
- The **Clue** window has helpful information about anything the mouse is over. You can add your own clues.
- The **Inspector** window shows current settings of a selected object. The inspector can be a separate window or in the sidebar of a patcher window.



Avoiding Wires

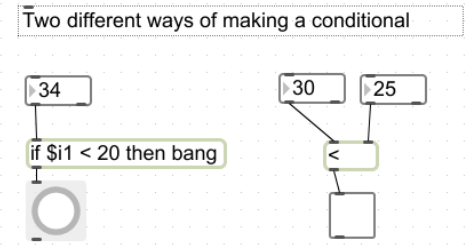
- Wire cords may introduce clutter for large patches
- send and receive objects to the rescue
- send – send the data somewhere "wirelessly" (can be abbreviated as s)
- Need to give the data a name
- receive – receives data "wirelessly" (can be abbreviated as r)
- Need to tell what to receive by the same name as send



Patching with Hot Keys

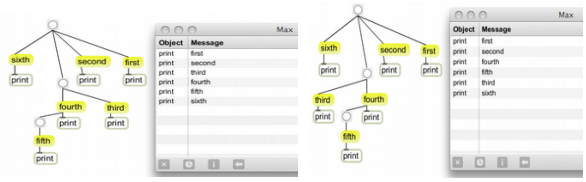
- Power users take advantage of hot keys.
- Pressing a key will drop a new object box at the mouse location.
- Some shortcuts:
  - **n** – arbitrary new object with name entry
  - **b** – button
  - **c** – comment
  - **f** – float number box
  - **i** – number box
  - **j** – jitter object box with jit. prefixed
  - **m** – message box
  - **p** – (lowercase) the object explorer
  - **P** – (uppercase) a new object selected for presentation.
  - **t** – toggle
  - **x** – a window explaining all of these

Conditionals



Multiple Inputs

- Right to Left Action:



OpenGL in Jitter

gl-tut.maxpat
gl-shaders-plato.maxpat
iss.vfx.falling-candies.maxpat
iss.vfx.falling-skittles.maxpat

Jit.gl.render

- The argument in jit.gl.render is the "drawing context".
- All jit.gl objects with this name will be created in this context, and the results displayed in a jit.window or pwindow of the same name (you can name a pwindow with its inspector.)
  - The window must have its depthbuffer attribute set to 1.
- The qmetro clocks screen updates in the usual way.
- The erase message clears the windows, the bang sent to draw (or other destination of your choice) may be useful for mechanisms that affect the drawing, and the bang directly to render draws the image.

Using OpenGL in Jitter

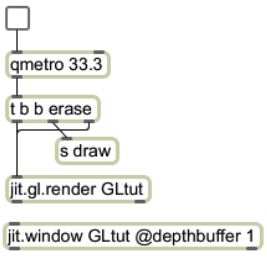
- The Jitter OpenGL objects are prefaced with jit.gl.
- These are mostly designed to make OpenGL available in a painless way.
- As always, making life easy might mean hiding options, but just about everything is available if you dig enough.

Jit.gl.render

- Note you can only draw to one destination.
- That destination can be a jit.matrix if you need the image in more than one place.
- Unfortunately, drawing to a matrix uses CPU rendering instead of the graphics card, so there is a significant loss of efficiency.

Jit.gl.render

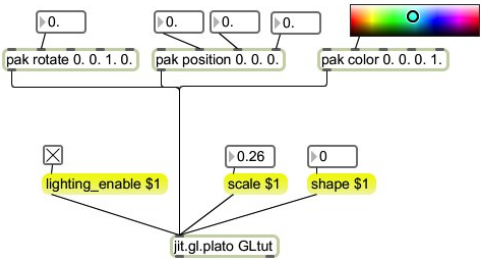
- Jit.gl.render is the key object in the OpenGL world.
- Every GI patch needs something like:



Jit.gl.plato

- jit.gl.plato contains the drawing commands to produce a platonic solid- that is a tetrahedron, hexahedron (cube), octahedron, dodecahedron (12), or icosahedron (20).
- The example shows the object and essential support items.
- The initial view of jit.gl.plato is not impressive.

Jit.gl.plato



`Lab9-ex1.maxpat`

Textures

- There are ways of modifying the texture dimensions via the render object, but it is frankly better to take care of that sort of thing before sending an image to texture.
- One thing is not made clear- textures are assumed to be square.
- Any other aspect ratio is interpolated down to a square shape.

Textures

- Textures apply arbitrary patterns to objects.
- A texture is defined in the render object, then that texture is assigned to other objects.
- There are several ways to define textures, but the easiest is:
  - Send a matrix to a [prepend texture somename] object and on to the render object.
    - The matrix becomes the texture.
    - Any matrix source may be used, including movie or grab.
  - Send the message (texture somename) to the drawing object.
    - The texture image will now be applied to the object.

`gl-tut.maxpat`

Textures

- You will quickly notice that textures from matrices appear upside down.
- This is due to the conflict between QuickTime and OpenGL coordinate systems.
- In QuickTime, the vertical component is positive going down the screen.
  - Important: 0,0 is in the upper left corner.
  - In OpenGL Y is positive going up.

Textures

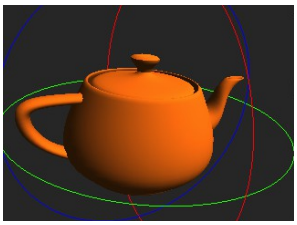
- The appearance of the texture may be modified by the tex_map message to the drawing object.
- There are four modes
  - 0. Default (varies with object).
  - 1. Object linear.
    - The texture is attached to the object.
    - When we look at the jit.gl.sketch object we'll see how this is specified.
  - 2. Sphere map.
    - The texture is reflected from the object.
  - 3. Eye linear.
    - As if you are looking through the object at the texture.

Textures

- There is also a jit.gl.texture object which allows much more detailed control of a texture.
- It's complicated enough that it probably needs its own tutorial.
- The jit.gl.shader object creates textures that interact with objects in a more complex way.
- There is an excellent tutorial in the Jitter tutorial set.

Jit.gl.handle

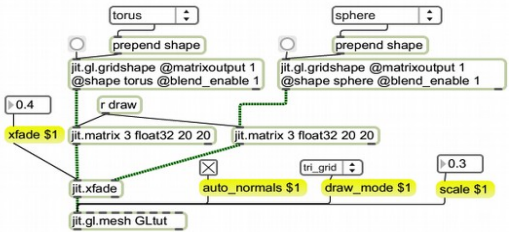
- Jit.gl.handle allows mouse interaction to rotate the image.
- Jit.gl.handle can be attached to a single object or the render object.
- In the latter case, the entire view is rotated. (Set the jit.gl.handle object's inherit_transform to 1 when you do this.)
- When the mouse is clicked on the window, axes orbits appear, and dragging the mouse perform the rotation.
- You can have multiple handles in a scene, but their behavior is likely to be a bit chaotic--reducing the handle radius will help.



Visualization of Audio

Jit.gl.mesh

- That will not be saved, and you will have to set the draw_mode of the jit.gl.mesh by hand.
- Matrices are easy to manipulate to produce mutant images. The drawing above is an xfade between a torus and a sphere.
- The xfade parameter controls a smooth morph between the two.

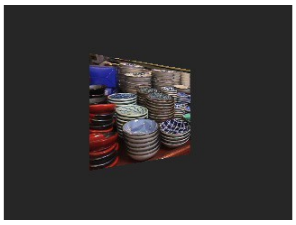


Visualization of Audio

- Visualization is a type of video synthesis that generates images from audio input.
- The effects of this can be tedious or sublime.
- In a typical approach, audio information is mapped to visual attributes of simple shapes.

Jit.gl.videoplane

- Jit.gl.videoplane is a sort of movie screen that you can position in space.
- Any matrix fed to it will be displayed-- this includes movies and grab output.
- The default size of the videoplane is 2.0 x 2.0.
- To precisely fit a 4:3 window, scale jit.videoplanre to 1.11, 0.843.
- When an OpenGL window is resized, objects are resized according to the Y dimension but keep their overall proportions.
- This means that when a window displaying a videoplane is expanded to fullscreen, there will usually be extra blank space at the edges.



Visualization of Audio

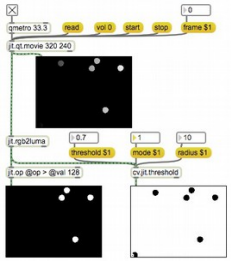
- This is possible with a few simple mechanisms that can be applied to a wide range of images.
- Amplitude.
  - This is related to the loudness of the sound.
  - To match perception of the listener, amplitude must be averaged over a brief period.
  - Slightly differing averaging times can produce quite different effects.
- Frequency.
  - This is related to the pitch of the music.
  - Accurate pitch extraction is only possible with simple sounds, but some degree of error is tolerable in most pitch to image mappings.

Visualization of Audio

- Waveform.
  - This is one way of representing timbre.
  - Waveforms may be drawn directly on the screen, but aside from a general association between the size of the waveform loops and loudness, it is difficult to link sound quality with a particular waveform.
- Spectrum.
  - Another representation of timbre, spectrum produces a set of values that can control many visual parameters or objects.
  - Spectrum displays can convey timbre with some accuracy (after practice), and if detailed enough can suggest pitch.

Black and White

- Most of the cv.jit objects will only work on a greyscale image.
- The easiest way to derive this is with jit.rgb2luma, although is some special cases you may want to split off one color instead.
- That is done with jit.unpack or a one layer jit.matrix with a planemap attribute.

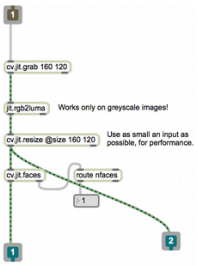


Amplitude Analysis

- Avg~ is a simpler version of average that only produces an output when banged.
- It is limited to absolute response, but may be easier to synchronize with a video generator since its output is a float rather than a signal.

Faces

- cv.jit.faces detects one or more faces in an image



CV.jit

- **cv.jit** is a collection of Max tools for computer vision applications.
- The goals of this project are to provide externals and abstractions to assist users in tasks such as image segmentation, shape and gesture recognition, motion tracking, etc. as well as to provide educational tools that outline the basics of computer vision techniques.
- Original:
  - http://jmpelletier.com/cvjit/

More about Computer Vision

Blobs

- cv.jit.label
  - Identify and track individual shapes.
  - Once images have been through threshold to isolate the brightest or darkest areas, cv.jit.label can attach numbers to each distinct region or "blob".
  - The output of label is a one plane matrix of either long or char of the same size as the input
  - Each cell contain the number of the region it is assigned to.

Kinect

cv.jit.blobs.centroids

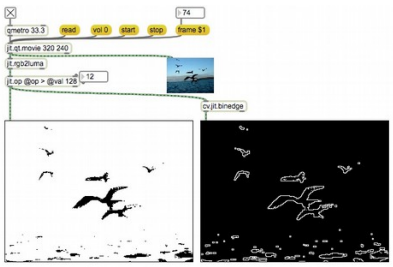
- Once an image is labeled, there are many cv.jit operations available
- cv.jit.blobs.centroid will find the center point of each region
- The output of cv.jit.centroids is a matrix with X Y location and area of each region identified by the threshold and label process
- The XY values are used to draw an X over each region
- The area values could be used to restrict the Xs to the largest objects, eliminating a lot of the noise

Synapse

- http://synapsekinect.tumblr.com/
  - Note: Kinect for Windows doesn't work with Synapse.
  - Synapse only supports "Kinect for Xbox".
  - Model 1414 Kinects work with Synapse, but there are reports that the 1473 models do not work.

```
module9/*
```

cv.jit.binedge



Other Kinect Use Options

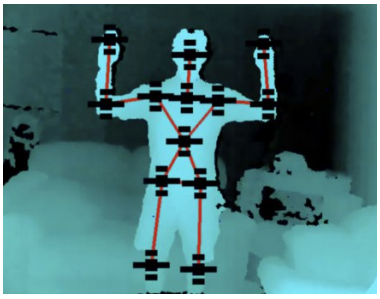
- http://cycling74.com/wiki/index.php?title=Kinect_Page
  - jit.freenect.grab
  - jit.openni
  - dp.kinect
  - **Synapse**
- Examples:
  - http://blairneal.com/blog/jit-freenect-examples/
  - http://synapsekinect.tumblr.com/post/6305020721/download
  - http://synapsekinect.tumblr.com/post/6307752257/maxmsp-jitter
  - http://www.instructables.com/id/Create-Interactive-Electronic-Instruments-with-Max/step5/Xbox-Kinect-and-MaxMSP
  - http://deecerecords.com/projects/#kinectsynapse

Synapse

- We continue with Synapse, since it appears to be the easiest to get working on 64-bit systems.
  - Tracks only one user at the skeleton level
- jit.freenect.grab appears to be more powerful, but no 64-bit version
- Likewise, OpenNI was acquired by Apple in April 2014, so things like jit.openni might not work, but there is hope with OpenNI2
- On Windows, da.kinect seems to offer all Microsoft's SDK for Kinect's functionality.
- You are welcome to experiment.

Synapse

- Point Kinect towards yourself and calibrate.

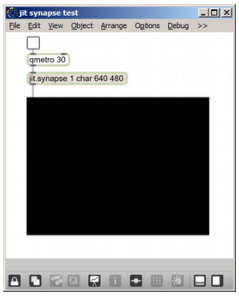


Synapse

- It also seems to be the most popular program today for connecting to Kinect.
- Though this program can only track one user at a time, it's relatively easy to set up and it communicates with Max through a patch called Kinect-Via-Synapse

Synapse

- If all works well you should see same b/w image in pwindow:

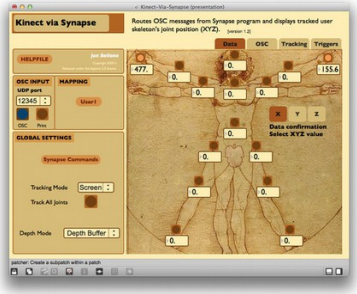


Synapse

- Installation, that worked for us:
  1. Download from:
     http://synapsekinect.tumblr.com/post/6305020721/download
     - http://hihigogo.com/Synapse-Mac.zip
     - http://hihigogo.com/SynapseJit.zip
  2. Download the UI patcher:
     - http://www.deecerecords.com/public/Kinect-Via-Synapse.zip
  3. Unzip all three.
  4. Copy OSC-route.mxo from Kinect-Via-Synapse to the Synapse.app's folder and to Cycling'74's jitter-extensions folders
  5. Copy jit.synapse.mxo from SynapseJit to jitter-extensions and jit synapse test.maxpat to jitter-help.
  6. Connect your Kinect to the USB (worked for me 1473)
  7. Start Synapse.app. You should see a Window similar to the screenshot next slide.
  8. Then open jit synapse test.maxpat and start it. It should see the depth image within jit.pwindow.

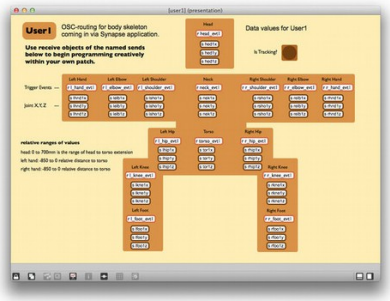
Synapse

- Tracking UI may be helpful to debug the data (Kinect-via-Synapse):



Synapse

- Tracking UI may be helpful to debug the data (Kinect-via-Synapse):



iPhone, iPad, Touch Screen Device

Kinect + Processing + Max + Syphon

- The initial ISSv2 pipeline
- To capture two skeletons

```
ISSv2P/ISSv2P.pde
ISSv2Jit/ISSv2Jit.maxpat
ISSv2Jit/ISSv2MasterVFX.maxpat
```

Wii

Kinect

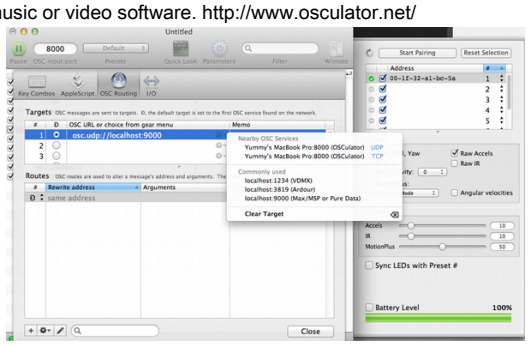
- We will come back to Kinect in Processing
- Recommended reading
  - http://www.deecerecords.com/public/Bellona_LIPAM2012.pdf

OSC (Open Sound Control)

- OSC protocol is common to communicate data (not only sound) between applications and computers
- opensoundcontrol.org

OSCulator for Wii

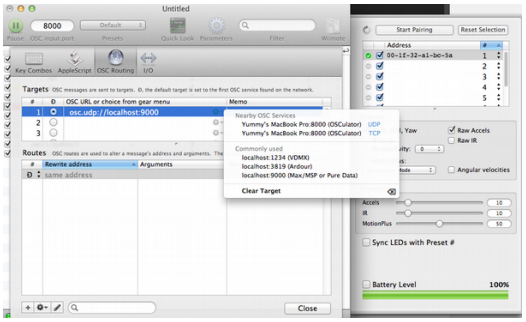
- OSCulator is the missing link between your controllers and your music or video software. http://www.osculator.net/



OSCulator for Wii

OSCulator for Wii

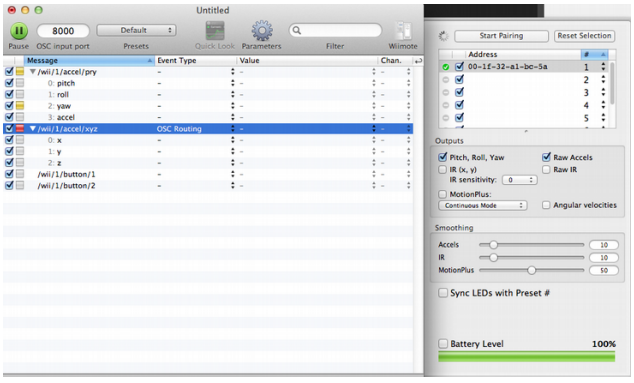
- OSCulator is the missing link between your controllers and your music or video software. http://www.osculator.net/



For iPhone

- Touch function
  - **Fantastick** (free)
    http://pinktwins.com/fantastick/
  - TouchOSC ($4.99)
    http://www.osculator.net/doc/tutorial:1:start
  - C74 ($3.99)
    http://nr74.org/software/c74.html
- Webcam
  - **Pocketcam** (free)
    http://www.senstic.com/iphone/pocketcam/pocketcam.aspx
  - iWebcamera ($4.99)
    http://www.made-apps.com/EN/Products/iPhone/iWebcamera.aspx

OSCulator for Wii



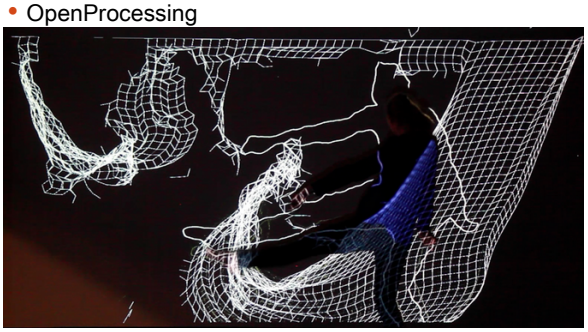
OSCulator

- Also integrates with the iDevices

Shaders in Jitter

Processing

- OpenProcessing



Processing

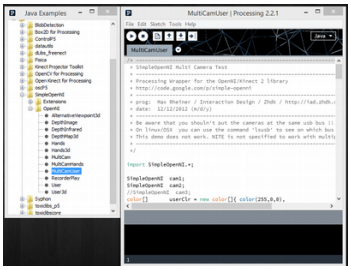
Libraries of Interest

- Very Many!
- In our case
  - fisica
  - KinectProjectorToolkit
  - MSAfluid
  - opencv_processing
  - oscP5
  - SimpleOpenNI
  - Syphon
  - ...
  - OpenKinect for Processing

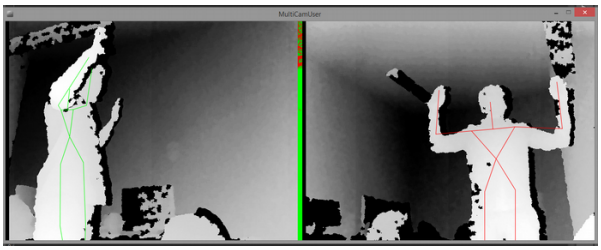
Processing

- FOSS, http://processing.org
- Community sketches: http://openprocessing.org
- Processing 3 has been released after ISSv2 (relies on Java 1.8)
- Simplified Java syntax, easier for artists
- Lower level than Max but higher than pure Java
- Many contributions of sketches at openprocessing.org
- Integration with OpenGL and may devices
  - Kinect
  - Arduino
- Jim Parker's recent book, *Introduction to Game Development Using Processing*, June 2015, 978-1937585402
- More powerful for AI, as well as graphics and sound.

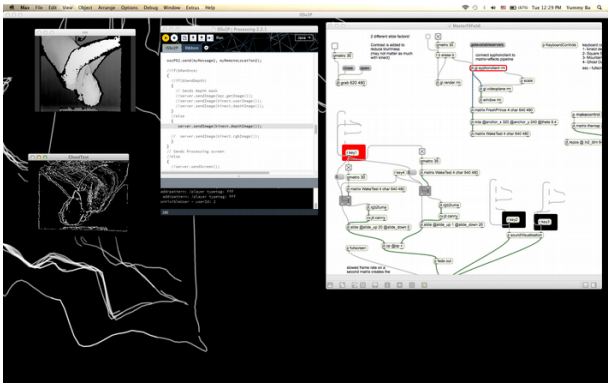
Libraries of Interest



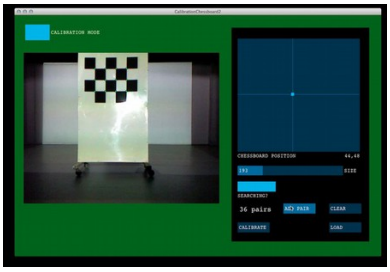
Libraries of Interest



Processing and Max



Libraries of Interest

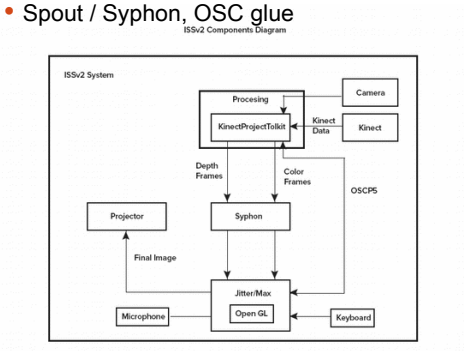


Credits

- Serguei Mokhov
  - (technical lead)
- Miao Song
  - (creative director)
- Julie Chaffarod
  - (VFX R&D lead)
- Milin Li
  - (VFX artist)
- Jilson Thomas
  - (projection mapping)
- Deschanel Li
  - (dancer)
- Holly Ryan
  - (dancer)
- Juetian Xing
  - (production assistant)
- Sebouh Bardakjian
  - (developer)
- Johnathan Llewellyn
  - (developer)
- Satish Chilkaka
  - (software engineer)
- Zinia Das
  - (software engineer)
- Sudhir Mudur
  - (advisor)
- ...

Processing

- Spout / Syphon, OSC glue



Alternatives

- Max → PureData (http://puredata.info/)
  - Jitter → GEM
- Vvvv (http://vvvv.org)

Acknowledgements

- Computer Science and Software Engineering, ENCS, Concordia University
- OVPGSR, Concordia
- Central Academy of Drama, Beijing
- Chinese Academy of Science, RCSC
- Open source projects and their contributors
  - SimpleOpenNI, oscP5, OpenCV, ...
  - Processing and OpenProcessing sketches
  - KinectProjectorToolkit
- Jitter/MaxMSP Cycling'74

Thank you!

- Questions?
- Visit us:
  - mdreams-stage.com

REFERENCES

Edward A. Ashcroft, Anthony A. Faustini, Rangaswamy Jagannathan, and William W. Wadge. 1995. *Multidimensional Programming*. Oxford University Press, London. ISBN: 978-0195075977.

Edward A. Ashcroft and William W. Wadge. 1977. Lucid, a nonprocedural language with iteration. *Commun. ACM* 20, 7 (July 1977), 519–526. https://doi.org/10.1145/359636.359715

Sebouh-Steve Bardakjian, Miao Song, Serguei A. Mokhov, and Sudhir P. Mudur. 2016. ISSv3: From Human Motion in the Real to the Interactive Documentary Film in AR/VR. In *Proceedings of the SIGGRAPH ASIA 2016 Workshop on Virtual Reality Meets Physical Reality (VR Meets PR 2016)*. ACM, New York, NY, USA. https://doi.org/10.1145/2992138.2992139

Greg Borenstein. 2013. OpenCV for Processing. [online]. (July 2013). https://github.com/atduskgreg/opencv-processing.

Niels Böttcher. 2007–2013. An introduction to Max/MSP. [online], Medialogy, Aalborg University Copenhagen. (2007–2013). http://imi.aau.dk/~nib/maxmsp/introduction_to_MaxMsp.ppt.

Tom Butterworth and Anton Marini. 2013. Syphon for Jitter. [online]. (Nov. 2013). https://github.com/Syphon/Jitter/releases/.

Craig Caldwell. 2015. Bringing Story to Life: For Programmers, Animators, VFX Artists, and Interactive Designers. In *ACM SIGGRAPH 2015 Courses (SIGGRAPH'15)*. ACM, New York, NY, USA, 6:1–6:10. https://doi.org/10.1145/2776880.2792697

Andres Colubri. 2014. Syphon for Processing. [online]. (2014). https://github.com/Syphon/Processing/releases.

Cycling '74. 2005–2015. Max/MSP/Jitter. [online]. (2005–2015). http://cycling74.com/products/max/.

Peter Elsea. 2007–2013. Max/MSP/Jitter Tutorials. [online], University of California, Santa Cruz. (2007–2013). ftp://arts.ucsc.edu/pub/ems/MaxTutors/Jit.tutorials/.

Ben Fry and Casey Reas. 2001–2015. Processing – a programming language, development environment, and online community. [online]. (2001–2015). http://www.processing.org/.

Google LLC. 2017–2018. Google Brain Team: Machine Learning Algorithms. [online]. (2017–2018). https://magenta.tensorflow.org/.

Peter Grogono. 2002. Getting Started with OpenGL. [online]. (2002). Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada.

Intel Corporation, Willow Garage, and Itseez. 2000–2018. Itseez: Image Processing Algorithms. [online]. (2000–2018). https://opencv.org/.

Joris and The Resolume Team. 2014. Resolume Arena Blog: Spout – Sharing Video between Applications on Windows. [online]. (May 2014). http://resolume.com/blog/11110/spout-sharing-video-between-applications-on-windows.

Gene Kogan. 2014a. Kinect Projector Toolkit for image mapping and calibration. [online, GitHub]. (July 2014). https://github.com/genekogan/KinectProjectorToolkit.

Gene Kogan. 2014b. Kinect Projector Toolkit for image mapping and calibration. [online]. (July 2014). https://github.com/genekogan/KinectProjectorToolkit.

Joseph J. LaViola, Jr. 2015. Context Aware 3D Gesture Recognition for Games and Virtual Reality. In *ACM SIGGRAPH 2015 Courses (SIGGRAPH'15)*. ACM, New York, NY, USA, 10:1–10:61. https://doi.org/10.1145/2776880.2792711

Hao Li, Anshuman Das, Tristan Swedish, Hyunsung Park, and Ramesh Raskar. 2015. Modeling and Capturing the Human Body: For Rendering, Health and Visualization. In *ACM SIGGRAPH 2015 Courses (SIGGRAPH'15)*. ACM, New York, NY, USA, 16:1–16:160. https://doi.org/10.1145/2776880.2787681

V. J. Manzo. 2011. *Max/MSP/Jitter for Music: A Practical Guide to Developing Interactive Music Systems for Education and More*. Oxford University Press.

Microsoft. 2012a. Human Interface Guidelines: Kinect for Windows v. 1.5. [online]. (2012). http://go.microsoft.com/fwlink/?LinkId=247735.

Microsoft. 2012b. The Kinect for Windows SDK v. 1.5. [online]. (21 May 2012). Online at http://www.microsoft.com/en-us/kinectforwindows/develop/developer-downloads.aspx and http://msdn.microsoft.com/en-us/library/hh855347.

Serguei A. Mokhov, Miao Song, Satish Chilkaka, Zinia Das, Jie Zhang, Jonathan Llewellyn, and Sudhir P. Mudur. 2016. Agile Forward-Reverse Requirements Elicitation as a Creative Design Process: A Case Study of llimitable Space System v2. *Journal of Integrated Design and Process Science* 20, 3 (Sept. 2016), 3–37. https://doi.org/10.3233/jid-2016-0026

Serguei A. Mokhov, Kin-Fung Yiu, Brian Ye, Jie Zhang, Haotao Lai, and Miao Song. 2017. Real-time Motion Capture for Performing Arts and Stage. [online], TEDxConcordia. (Sept. 2017). https://www.youtube.com/watch?v=YgwnEmHFwI8.

R. Molich and Jakob Nielsen. 1990. Improving a human-computer dialogue. *Commun. ACM* 33, 3 (March 1990), 338–348.

OpenKinect Contributors. 2011–2018. OpenKinect: Open Source Drivers for Kinect v1. [online]. (2011–2018). http://openkinect.org.

Jean-Marc Pelletier. 2012. `jit.freenect.grab` – a Max/MSP/Jitter external for Microsoft Kinect. [online]. (7 March 2012). RC5, http://jmpelletier.com/freenect/.

Bill Polson. 2015. Pipeline Design Patterns. In *ACM SIGGRAPH 2015 Courses (SIGGRAPH'15)*. ACM, New York, NY, USA, 21:1–21:59. https://doi.org/10.1145/2776880.2792724

Konstantinos Psimoulis, Paul Palmieri, Inna Taushanova-Atanasova, Yasmine Chiter, Amjrali Shirkhodaei, Navid Golabian, Mohammad-Ali Eghtesadi, Behrooz Hedayati, Piratheeban Annamalai, and Andrew Laramee. 2018. OpenISS Web Services API Implementation for OpenISS-as-a-Service. [online], SOEN487 Team 10 and Team 11, Serguei Mokhov. (April 2018). https://github.com/OpenISS/OpenISS/tree/master/src/api/java.

Miller Puckette and PD Community. 2007–2014. Pure Data. [online]. (2007–2014). http://puredata.org.

Theresa-Marie Rhyne. 2015. Applying Color Theory to Digital Media and Visualization. In *ACM SIGGRAPH 2015 Courses (SIGGRAPH'15)*. ACM, New York, NY, USA, 5:1–5:112. https://doi.org/10.1145/2776880.2792696

Christian Richardt, James Tompkin, Jiamin Bai, and Christian Theobalt. 2015. User-centric Computational Videography. In *ACM SIGGRAPH 2015 Courses (SIGGRAPH'15)*. ACM, New York, NY, USA, 25:1–25:6. https://doi.org/10.1145/2776880.2792705

Yvonne Rogers, Helen Sharp, and Jenny Preece. 2011. *Interaction Design: Beyond Human - Computer Interaction* (3rd ed.). Wiley Publishing. Online resources: id-book.com.

Andreas Schlegel. 2011. oscP5 – A implementation of the OSC protocol for Processing. [online]. (2011). http://www.sojamo.de/libraries/oscP5/.

Miao Song. 2012. *Computer-Assisted Interactive Documentary and Performance Arts in Illimitable Space*. Ph.D. Dissertation. Special Individualized Program/Computer Science and Software Engineering, Concordia University, Montreal, Canada. Online at http://spectrum.library.concordia.ca/975072 and http://arxiv.org/abs/1212.6250.

Miao Song et al. 2014a. Real-Time Motion-Based Shadow and Green Screen Visualization, and Video Feedback for the *Like Shadows* Theatre Performance with the ISS. [theatre production, video, news]. (2–12 April 2014). http://www.concordia.ca/encs/cunews/main/stories/2014/06/04/digital-art-thatillustratesthelandofthelivingandthedead.html and http://www.concordia.ca/content/dam/encs/csse/news/docs/like-shadows-cse-academy.pdf.

Miao Song and Serguei A. Mokhov. 2014. Dynamic Motion-Based Background Visualization for the *Ascension* Dance with the ISS. [dance show, video]. (18–19 Jan. 2014). http://vimeo.com/85049604.

Miao Song, Serguei A. Mokhov, et al. 2015b. Illimitable Space System at CG in Asia International Resources. Talk and Demo. (10 Aug. 2015). http://s2015.siggraph.org/attendees/acm-siggraph-theater-events.

Miao Song, Serguei A. Mokhov, Julie Chaffarod, et al. 2015a. Dynamic Motion-Based Visualization for the *District 3 Demo Day* with the ISSv2 and Processing. [demo, video]. (4 June 2015). https://vimeo.com/130122925 and https://vimeo.com/129692753.

Miao Song, Serguei A. Mokhov, and Peter Grogono. 2014b. A Brief Technical Note on Haptic Jellyfish with Falcon and OpenGL. In *Proceedings of the CHI'14 Extended Abstracts: ACM SIGCHI Conference on Human Factors in Computing Systems*. ACM, New York, NY, USA, 1525–1530. https://doi.org/10.1145/2559206.2581135 Includes video and poster.

Miao Song, Serguei A. Mokhov, Peter Grogono, and Sudhir P. Mudur. 2014a. Illimitable Space System as a Multimodal Interactive Artists' Toolbox for Real-time Performance. In *Proceedings of the SIGGRAPH ASIA 2014 Workshop on Designing Tools for Crafting Interactive Artifacts (SIGGRAPH ASIA'14)*. ACM, New York, NY, USA, 2:1–2:4. https://doi.org/10.1145/2668947.2668953

Miao Song, Serguei A. Mokhov, Peter Grogono, and Sudhir P. Mudur. 2014b. On a Non-Web-Based Multimodal Interactive Documentary Production. In *Proceedings of the 2014 International Conference on Virtual Systems Multimedia (VSMM'2014)*, Harold Thwaites, Sarah Kenderdine, and Jeffrey Shaw (Eds.). IEEE, 329–336. https://doi.org/10.1109/VSMM.2014.7136675

Miao Song, Serguei A. Mokhov, Alison R. Loader, and Maureen J. Simmonds. 2009. A Stereoscopic OpenGL-based Interactive Plug-in Framework for Maya and Beyond. In *Proceedings of VRCAI'09*. ACM, New York, NY, USA, 363–368. https://doi.org/10.1145/1670252.1670333

Miao Song, Serguei A. Mokhov, Sudhir P. Mudur, and Peter Grogono. 2015. Rapid Interactive Real-time Application Prototyping for Media Arts and Stage Performance. In *ACM SIGGRAPH Asia 2015 Courses (SIGGRAPH Asia'15)*. ACM, New York, NY, USA, 14:1–14:11. https://doi.org/10.1145/2818143.2818148

Miao Song, Serguei A. Mokhov, Sudhir P. Mudur, and Peter Grogono. 2016. Hands-on: Rapid Interactive Application Prototyping for Media Arts and Stage Production. In *ACM SIGGRAPH Asia 2016 Courses (SIGGRAPH Asia'16)*. ACM, New York, NY, USA, 19:1–19:29. https://doi.org/10.1145/2988458.2988460

Miao Song, Serguei A. Mokhov, Jilson Thomas, et al. 2015b. Dynamic Motion-Based Background Visualization for the *Gray Zone* Dance with the ISSv2. [dance show, video]. (14 Feb. 2015). https://vimeo.com/121177927.

Miao Song, Serguei A. Mokhov, Jilson Thomas, and Sudhir P. Mudur. 2015a. Applications of the Illimitable Space System in the Context of Media Technology and On-Stage Performance: a Collaborative Interdisciplinary Experience. In *Proceedings of GEM'15*. IEEE. To appear.

Debbie Stone, Caroline Jarrett, Mark Woodroffe, and Shailey Minocha. 2005. *User Interface Design and Evaluation* (1st ed.). Wiley Publishing.

Marian F. Ursu, Vilmos Zsombori, John Wyver, Lucie Conrad, Ian Kegel, and Doug Williams. 2009. Interactive Documentaries: A Golden Age. *Comput. Entertain.* 7, Article 41 (Sept. 2009), 29 pages. Issue 3. https://doi.org/10.1145/1594943.1594953

William W. Wadge and Edward A. Ashcroft. 1985. *Lucid, the Dataflow Programming Language.* Academic Press, London.

Todd Winkler. 2001. *Compositing Interactive Music: Techniques and Ideas Using Max.* MIT Press.

Jie Zhang, Sebouh Bardakjian, Milin Li, Miao Song, Serguei A. Mokhov, Sudhir P. Mudur, and Jean-Claude Bustros. 2015. Towards Historical Exploration of Sites With an Augmented Reality Interactive Documentary Prototype App. In *Proceedings of Appy Hour, SIGGRAPH'2015.* ACM.