

A distributed remote presence system for latency critical human-to-human and human-to-computer interaction

Giacomo Tartari

Ph.D. dissertation in Computer Science

“E quindi uscimmo a riveder le stelle.”
–Dante Alighieri, *Inferno* XXXIV, 139

Abstract

In a computer-based distributed stage performance, such as a theater play or opera, the actors and the audiences are spread among different stages in different locations. Actors in different cities can be on the same virtual stage and perform in front of an audience that can enjoy a whole consistent performance.

A distributed stage performance raises a set of challenges both of a principled and of a practical nature. Traditional plays taking place on a single physical stage have a number of characteristics. These include very low delays between actors observing what others are doing, all actors and the audience see all events in the same order, and actors as well as the stage are modified through costumes, props, make-up and light changes. A computer-based system for distributed stage performances has to handle these aspects in principle.

In practice, a system for distributed stage performances has to implement a number of functionalities. The actors need a representation of themselves on each of the stages, referred to as a *remote presence*. The remote presence will represent the actors to a varying degree of accuracy. The virtual remote presence is dressed in a *virtual costume*. It substitutes, in part or entirely, the costumes and make-up of the actor. A distributed performance can need *remote actuation* to interact with remote physical realities. To achieve this is needed a functionality to detect and analyze actor's movements, translate them into gestures and translate the gestures into local and remote actions. Both remote actuation and the remote presence need functionality to detect the state of the stages. The state must be shared among all the stages comprising the show. However, the distance between stages and speed of light result in a non-zero time from when an event happens on a stage until it can be observed on the other stages. This results in the individual stages perceiving an inconsistent state of the virtual stage, potentially hampering the show.

Typically, existing approaches have one or more limitations. The remote presences that they provide targets teleconferencing, not distributed stage shows. There is no masking of the effects of delays, instead traffic-shaping, encoding, and compression are used to reduce latency. Data streams are not separated

on a per-actor basis. Stages may need to do significant processing on the state to extract information about individual actors. Such information can be the gestures performed by the actor. It could also be the position of the actors on stage, allowing to produce remote presences in different layouts from where they are acquired. While some systems provide a state stream per user, they employ many sensors rigged in special cage-like structures. This can limit the user's mobility and make the devices not easily portable. Increased resource usage usually results in acquisition rates around 10 – 15 Hz.

This dissertation presents MultiStage, a system for distributed performances. MultiStage has functionalities to detect the state of the stages including possible gestures from actors, distribute the state among the stages, and to create remote presences. A local stage can customize the appearance and location of each remote presence. Modifying the appearance of remote presences results in what we term *amplified interaction*.

The design partitions MultiStage into a local side and a global side. The local side produces local per-actor state streams. It creates remote presences from local and incoming streams from other stages. The global side receives and distributes state from and to the local side applying a publish subscribe model. The MultiStage prototype is comprised of a number of systems implemented as processes and threads in Go, Python, and C. At each stage at least three computers are used. At each stage four 3D cameras configured into a close to 360 degree in a back to back configuration are used to acquire the state of four actors. The MultiStage sensor suite is a compact and portable device comprised of four 3D cameras and two Mac mini computers, the actors are located around the sensor suite and each camera acquires one actor.

A set of experiments were conducted on MultiStage to document its performance characteristics including CPU utilization, memory and bandwidth usage. Three stages were used, each had two computers for state detections and one for creating remote presences. The global side had one computer for distribution of state. The global side computer was either located at one of the stages or external to the stages across the Internet. All the computers were using less than 50% of the available CPU and 2 GByte of memory. While each stage produced 7 MByte of data per second.

Having separate state streams per actor is advantageous to the local stages because they can individually manipulate the remote presences with regards to amplification, location on the virtual stage and gesture detection. MultiStage uses cheap 3D cameras at each stage to create such state at very low processing cost and low delays. The state streams based on 3D cameras are also used to detect gestures at low processing cost and low delays. The trade-off is that a limited number of gestures can be detected. The quality of the remote pres-

ence created by MultiStage is a compromise between quality of visualization and smoothness. MultiStage visualizes remote presence at 30 fps, from the detection of the state to the rendering on display.

The scalability of MultiStage prototype is limited by the available bandwidth. The outbound bandwidth needed for distribution grows with the size of the data describing the state of each stage, P , and the square of the number of stages, $O(kPn^2)$, where n is the number of stages and k is a factor. A three-stage configuration will use half of a Gbit/sec link. Six stages will saturate a 2 Gbit/s network.

Acknowledgements

I would like to thank all the people without whom this dissertation would not have been possible. Thanks to my advisors Professor Otto Anshus, John Markus Bjørndalen and Dr. Daniel Stødle: your guidance has been indispensable and your humor has been refreshing.

Thanks to Otto who believed in me the whole time and supported me in the moments of dismay.

Thanks to John Markus and Daniel for the many productive discussions and the many encouraging advices (Daniel if you are reading this I owe you a beer).

Thanks to Professors Lars Ailo Bongo for all he help, the job opportunities and the interesting food discussions.

Thanks to my PhD companion Fei Su, who helped me tackling many of the issues encountered in this project.

Thanks to the head of the department Tore Brox-Larsen for keeping the (life-saving) espresso machine in proper condition and for the many funny and interesting discussions.

Thanks to Professor Ha Hoai Phuong, Dr. Bård Fjukstaf and Professor Lars Ailo Bongo again for the feedback on the dissertation draft.

Thanks to the Institute of Computer Science (IFI) staff leader Svein Tore Jensen and student advisor Jan Fuglesteg for their efficiency in getting things done and for their patience in dealing with my clumsy paperwork.

Thanks to the technical staff leader Maria Wulff Haugland, Ken-Arne Jensen, Jon Ivar Kristiansen, and Kai-Evan Nilsen for their prompt support, organization of social events and ingenuity in finding solution to our extravagant problems.

Thanks to my fellow PhD students Lars Tiede, Bård Fjukstad and Joseph Hurley for your company, the good time and the interesting discussions during these hard years.

Thanks to my family for being close in spite of the thousands of kilometers, and to Giorgia Ferrari for being even closer.

Thanks to Nora Nedberg for brightening so many dark days.

Thanks to the Tromsø Klatreklubb and my fellow climbers, because without the chance of blowing off some steam on the climbing walls I would have gone mad years ago.

Thanks to Renard Nilsen and to the Java Team One at HelseNord IKT for *assimilating* me, even if only for six months.

Thank to my ever changing flat mates, your support in this endeavor of mine is not forgotten.

Thanks to all my friends in Italy. Yes now I can come home, but I am not sure I will.

Thanks to everyone else not mentioned here but to whom I still owe a thank you, I am sorry but this is getting way too long.

Thank you very, very much.

This work was funded in part by the Norwegian Research Council, projects 187828, 159936/V30 and 155550/420, and the Tromsø Research Foundation (Tromsø Forskningsstiftelse).

Cathryn Primrose-Mathisen provided professional English language assistance during the preparation of this dissertation. She was not responsible for reviewing the final version.

Contents

Abstract	iii
Acknowledgements	vii
List of Figures	xiii
List of Tables	xv
List of Acronyms	xvii
1 Introduction	1
1.1 Multistage Distributed Performance	2
1.2 Challenges and Solutions	4
1.2.1 Latencies, delays, and their effects	4
1.2.2 Remote Presences	4
1.2.3 User Data Stream	5
1.2.4 Amplified Interaction	5
1.2.5 Gestures	6
1.3 Contributions	6
1.3.1 Lessons Learned	7
1.3.2 Models	7
1.3.3 Artifacts	8
1.3.4 Facts	9
1.4 Publications	10
1.4.1 MultiStage: Acting across Distance	10
1.4.2 Global Interaction Space for User Interaction with a Room of Computers	12
1.4.3 Controlling and Coordinating Computers in a Room with In-Room Gestures	13
1.4.4 Mapping of contribution and publications	14
1.4.5 Mapping of publications and Chapters	15
2 MultiStage Overview	17
2.1 Motivation	17

2.2	Ideas of MultiStage	18
2.3	Concepts	18
2.3.1	Temporal Causal Synchrony	18
2.3.2	Amplified Interactions and Gestures	20
2.4	Architecture	21
2.4.1	Alternative Architectures	25
2.5	Design	28
2.5.1	Discussion	31
2.6	State of The Art	33
2.6.1	Landscape/Broad View	33
2.6.2	Distinctive Systems	34
2.6.3	La Serva Padrona	37
3	User Context State Detection	41
3.1	Idea	41
3.2	Architecture	42
3.3	Design	44
3.4	Implementation	45
3.4.1	Sensor Suite	46
3.4.2	Software	46
3.5	Experiments	49
3.5.1	Design and Configuration	50
3.5.2	Results	50
3.6	Discussion	51
3.7	Lessons Learned	53
3.8	Summary	53
4	Gestures	55
4.1	Need for actor input	55
4.2	Architecture	56
4.3	Design	57
4.4	Implementation	60
4.4.1	Point Motion Analysis	60
4.4.2	Regular Expression Engine	62
4.5	Experiments	63
4.5.1	Latency	63
4.5.2	Resource Utilization	64
4.6	Discussion	64
4.7	Lessons Learned	65
4.8	Summary	66
5	Remote Presence	67
5.1	Idea	68
5.2	Architecture	69

5.3	Design	70
5.4	Implementation	70
5.5	Experiments	72
5.5.1	Configuration	72
5.5.2	Results	75
5.6	Discussion	76
5.6.1	Implementation	76
5.6.2	Amplified Interactions	77
5.7	Lessons learned	77
5.8	Summary	77
6	Global Interaction Space	79
6.1	Idea	79
6.1.1	Usage patterns	80
6.2	Architecture	81
6.3	Design	82
6.4	Implementation	83
6.4.1	Visual Feedback to User	86
6.5	Experiments	87
6.5.1	Configuration	87
6.5.2	Results	87
6.5.3	CPU	88
6.5.4	Memory	88
6.5.5	Network	89
6.6	Discussion	89
6.6.1	Design	89
6.6.2	Implementation	90
6.6.3	Limitations	90
6.6.4	MultiStage Applications	92
6.7	Lessons Learned	93
6.8	Summary	93
7	Contributions	95
7.1	Lessons Learned	95
7.2	Models	97
7.3	Artifacts	98
7.4	Facts	100
8	Discussion	103
8.1	Missing Capabilities	103
8.2	Limitations	104
8.3	Technologies	104
8.4	Architectures	104
8.5	Tools	105

8.6 Feasibility	106
8.7 Scalability	107
9 Conclusions	113
9.1 Future Works	114
Appendices	117
A Amplified Interactions	119
B Code	121
C Unexplored Paths: the Coil Gun Display	123
Bibliography	127
Papers	137

List of Figures

1.1	Vision of a distributed stage.	2
2.1	Legend of icons.	23
2.2	Illustration on the architecture.	24
2.3	Light begin point, fat global side, light endpoint.	26
2.4	Light begin point and global, fat endpoint or L-L-F.	27
2.5	Fat begin point light global and endpoint.	28
2.6	Schematic data flow of the Multistage system.	30
2.7	Design of the Multistage system.	31
2.8	Monitoring and Controllable Temporal Synchronization systems.	32
2.9	Flyer of the play <i>La Serva Padrona</i> , front.	39
2.10	Flyer of the play <i>La Serva Padrona</i> , back.	40
3.1	Architecture of the User Context State Detection.	43
3.2	Design of the User Context State Detection.	45
3.3	The Sensor Suite.	47
3.4	Software implementation of the User Context State Detection.	48
3.5	Memory and CPU usage of the UCSD.	51
3.6	Network traffic generated by the UCSD.	52
4.1	Architecture of the gesture detection system.	57
4.2	Design of the gesture detection system.	58
4.3	User Bounding Box.	59
4.4	The motion dictionary and an example of circular gesture.	59
4.5	Implementation of the gesture detection system.	61
4.6	Discrimination of motions using normalized vectors.	62
4.7	CPU and memory utilization of the gesture detection system.	64
5.1	Remote Presence system Architecture.	69
5.2	Remote Presence system design.	70
5.3	Implementation of the remote presence system.	71
5.4	Output on screen of the Remote Presence system.	72
5.5	Hardware and software configuration for the experiment.	74

5.6	Remote Presence System CPU and memory utilization.	75
5.7	Remote Presence system inbound network traffic.	76
5.8	Output on display wall of three virtual stages.	78
6.1	Architecture of the Global Interaction Space.	81
6.2	Design of the Global Interaction Space.	82
6.3	Implementation of the Global Interaction Space.	84
6.4	Visual Feedback of the Local component.	85
6.5	Example of usage and visual feedback to user.	86
6.6	Hardware and software configuration for the experiment.	88
6.7	Global Interaction Space CPU and memory utilization.	89
6.8	Possible improvement on the current implementation.	91
7.1	Scalability of Multistage.	102
8.1	Scalability Of Multistage in proportion to a Gbit/s link.	107
8.2	Projection on the bandwidth use with four to six stages.	108
8.3	Projection on the bandwidth use with 10k to 80k points.	109
8.4	Scalability overview of MultiStage.	111
8.5	Scalability overview of MultiStage 2.	112
A.1	Output on display wall of three virtual stages.	120
C.1	Coil gun display module.	124

List of Tables

1.1	Map of per paper contributions.	15
1.2	Map of publications and chapters.	15
4.1	<i>Circle</i> gesture latency.	63
4.2	<i>Straight</i> gesture latency.	64
7.1	Collection of performance measurements.	100
7.2	Summary of Gesture detection latency.	102

List of Acronyms

ADR Action Definition from Room

AE Action Executor

API Application Program Interface

CSP Communicating sequential processes

GAD Gesture to Action Dictionary

GAT Gesture to Action Translator

GIS Global Interaction Space

GSA Global State Analysis

GSM Global State Monitoring

HTTP Hypertext Transfer Protocol

JSON JavaScript Object Notation

LSA Local State Analysis

LSM Local State Monitoring

NTP Network Time Protocol

PID Process ID

PMA Point Motion Analysis

PSG Point Stream Generator

- QoS** Quality of Service
- RAM** Random Access Memory
- REGES** Regular Expression Gesture Engine
- REST** Representational State Transfer
- RFID** Radio-frequency identification
- RGB** Red-Green-Blue
- RGSA** Room Global State Analysis
- RGSM** Room Global State Monitoring
- TCP** Transmission Control Protocol
- UCSD** User Context State Detection
- UDP** User Datagram Protocol
- VF** Visual Feedback
- VGA** Video Graphics Array



Introduction

Today's technology has eased communication to such a degree that it is normal to be *always connected* and, at least potentially, constantly in touch with our acquaintances. Different kind of networks, wired and not, have become more pervasive on the planet, and verbally interacting with friends on different continents has been a reality for decades. Meeting people in *virtual places* is not strange anymore and having a distributed conversation with interlocutors from different cities is common.

Another form of communication where people interact is stage performances, such as concerts or plays. Stage performances are usually held in one place even if a performance can be recorded or transmitted and be remotely available. It is usual to think of it as *happening* in one place when audience is present.

In this dissertation, as part of the Verdione project [1], we present our contribution to bringing a stage performance beyond the boundary of a single stage – making it *distributed*. Fig. 1.1 shows the concept of distributed stage discussed in this dissertation. In the picture there are three stages placed in as many different cities. The actors are spread on the three stages interacting with each other through remote presences.

This dissertation describes the architecture, design, and implementation of some parts of the MultiStage system for supporting low latency interaction across distance between users. The usage domain is distributed stage shows. The purpose is to characterize how such a system can be built, and its perfor-

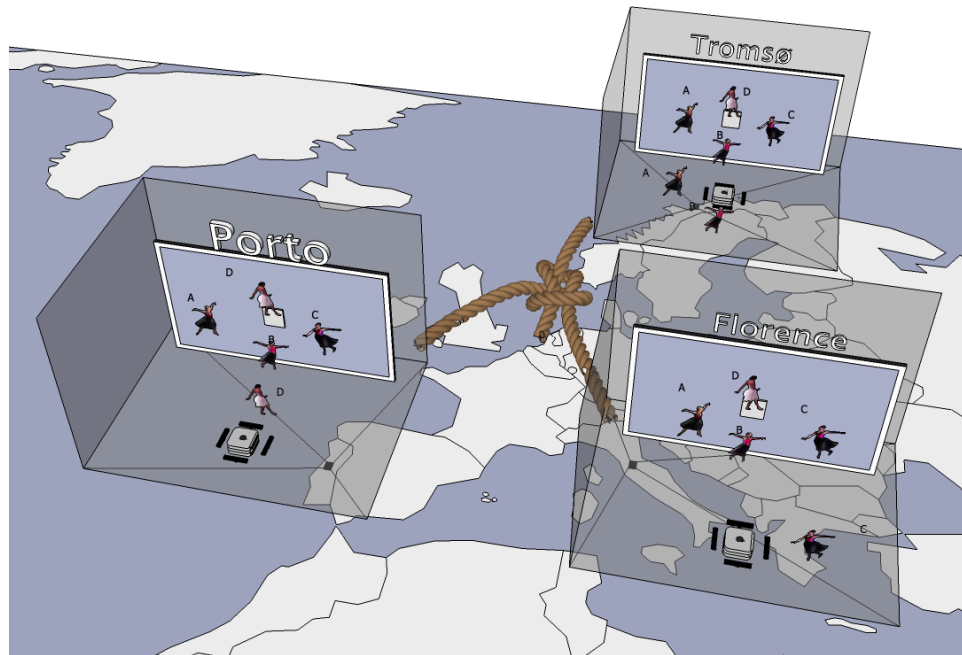


Figure 1.1: Vision of a distributed stage. Actors are performing on the same *virtual stage* but from different locations.

mance characteristics. This dissertation focuses on the systems for detecting the visual state of actors on a stage, detecting gestures done by actors, and the system for creating remote presences of actors.

There are nonetheless technological and physical limits that can render the implementation of such a system problematic. Delays in communications and flawed representations of remote actors can be obstacles to a consistent distributed stage show. There are however means to mitigate these issues or to take advantage of the limitation to expand the level of interaction. For example a digital remote presence can be manipulated in ways not possible for an actor. These manipulations can be activated by the actors on the stage by a gesture giving them a new level of control on the interaction.

1.1 Multistage Distributed Performance

We informally define a multistage distributed performance as *a performance conducted by actors in non-overlapping environments usually representing rooms or stages, potentially at the same time*. This might seem simplistic at first, but we believe this informal definition is generic enough to be used as a starting

point for this dissertation.

Multiple solutions and tools are available on the market to target related issues, mostly videoconferences. As an example, we consider Skype [2], software that is one of the most widespread for videoconferences, but the following reasoning can be applied to most of the conference software available at the time of writing. Skype supports voice and video calls one to one, or many to many with a subscription fee.

It looks like a good starting point but it has several limitations. At the time of writing Skype, with many other commercial systems, is tailored to single users and not to stages; it captures the whole scene without distinguishing the single actors, both in video and in audio. It does not expose an Application Program Interface (API) that allows splitting streams or manipulating them differently, for example different encoding or different destinations. There would be no other way to, for example, detect a user gesture than to capture the output on-screen and process it, locking the logical place where this computation can happen at the end point of the communication. A fee is needed to avail of the conference call with multiple people (more than two) and the layout of the videos on screen is not adjustable by the user.

Maintenance or other issues can cause the service to stop at any moment without notice or the bandwidth can be reduced in the middle of the performance because it is interfering with other functionalities of the third-party infrastructure. Political issues might cause the specific Internet traffic to be filtered or throttled in some countries.

Based on this overview there are multiple areas that can be improved.

- Ability to manipulate streams of data not only at the end point, but potentially at any stage of communication: at the begin point, end point, or distribution. The necessity of further computation on a stream can arise at the same location where the stream is produced; it is not efficient to encode and decode the stream again to access it.
- Ability to split data streams on a per user fashion and to subscribe to, and receive, only a set of the available streams. Not all the stages may need the whole performance from all the stages; it is more flexible to give the opportunity to the single stage to choose the amount of data they can handle and the data streams they need.
- Spatial and temporal data of each stream. Given that the performance is split into different locations but is supposed to be on the same stage, the availability of the spatial/temporal data of each stream makes it possible

to reconstruct the performance as a whole on each physical stage. For example, the remote presence of an actor, reconstructed from a data stream, can be placed on a virtual stage in the exact position where the actor was. Temporal information allows preservation of the interactions between different actors – the timings of their actions – and to reproduce them as close as possible to reality in the case of two actors being on the same stage, or to what should have happened, in the case of two actors being on different stages.

1.2 Challenges and Solutions

There are multiple challenges to be surmounted before distributed performances are able to mimic the real event in a productive way for the artists.

1.2.1 Latencies, delays, and their effects

Avoiding delays in a distributed performance is impossible as the theoretical fastest transmission speed of information is the speed of light. Traveling at the speed of light it takes, roughly $67ms$ to reach the antipode of any point on earth. This time is the worst-case scenario where two peers communicating are at antipodes of the earth. This delay is already significant and does not take into account other prominent factors such as processing times, delays introduced by the medium of transmission (Internet? Other networks?). Even if we do not operate in the worst possible conditions, delays cannot be avoided.

However it is possible to hide the effects, the consequences of the delays. Hiding the effect of the delay can be done in many ways; we do it by manipulating the remote presence of actors from a remote site. Other techniques, such as shared clocks, allow us to detect delays when they become intolerable for the user experience and react by manipulating the remote presence. The various possible manipulations are based on the spatial and temporal data embedded in the streams at the detection site.

1.2.2 Remote Presences

For a multiroom/multistage distributed performance to be believable, the actors/users need to interact in a natural way, as natural as if they were at the same location. Interaction in this context does not only mean one to one, single user to single user, but it can be a more extended and spatially dependent many-to-many interaction. As an example, on a stage, the actors do not only

consider the position of the other performers, but also there are cues, motion and signs coming from the rest of the actors and crew.

In the same way, in a distributed performance, one actor might not need to interact with only one of the remote presences, but potentially needs to consider all the others, because their motions, poses, and positions may convey information essential to the performance result.

1.2.3 User Data Stream

To achieve this goal of consistent interaction between remote stages, the Remote Presence system needs data from the remote stages, and the Detection system takes care of capturing the state of the stages and its occupants. The Detection system also encodes the data in separate streams, each one comprising the data needed to render the remote presence of users. Every stream is annotated with temporal and spatial information, every frame of each stream acquired by the 3D cameras is timestamped, and the raw depth data translated is a 3D point cloud. Assuming a shared clock, shared across the whole distributed system, these streams can be used to reproduce the relative positions and timings of motions in a remote location.

1.2.4 Amplified Interaction

Interactions on a stage can be different from the normal interactions between people. During a stage performance there is a distance between the actors and the audience – both physical distance and cognitive distance. Physical distance is the distance between the spectator and the stage, which can vary, depending on the location, from a few meters (first row in a theater) to potentially hundreds of meters (e.g., last rows in a big theater or at a crowded open air festival). Cognitive distance is all the information the actors give away indirectly by their acting, tone of voice, hairstyle, scene clothes, and other factors contributing to the experience that spectators have about the characters. Costumes, exaggerated gestures, makeup, lights, and sound effects participate to provide the final experience and to help narrow the gap between actors, characters, and spectators.

In a remote distributed performance it is not obvious how, and to an extent whether, these factors can be available or carried over to the remote sites. Different equipment, different stage sizes, and different hour of the day are all examples of differences that can alter the performance.

To mitigate the problem, we explore the possibility of modifying the remote

presence of an actor according to a defined set of conditions and gestures performed by the same. For example, the position of a user on the stage can trigger a different coloring of his remote presence, or a fast waving arm can be made to glow, or a predefined gesture can trigger an animation. This in principle allows the users/actors to overcome the lack of more mundane stage tricks, or the inability to convey them in a remote presence, to enhance the on-stage interaction.

1.2.5 Gestures

If Amplified Interactions are a way for the actors, remote or not, to reach both the audience and other actors, gestures are a way to trigger and integrate the Amplified Interactions on the stage as part of a performance. Gestures are a way to convey information by using motion. A system capable of detecting such motion can augment the natural interaction of actors by the use of Amplified Interactions.

Gestures can also be used to grant the actors another dimension of expressivity, enabling them to decide or command events on the stages, both remote and local. This other dimension can, and often is, planned in traditional stage events, for example with lights and other *effects* synchronized with predefined events in the performance. A gesture-based system can be used to not only replicate this behavior, but also to allow improvisations from the actors to become another tool in their interaction toolbox.

A stream of gestures, annotated with time and space coordinates, can be manipulated at a stream level, filtered, and replicated; for example, sending different gestures to different stages based on the position on the stage. Or at a more fine grained resolution, shifting the single gestures, or set of gestures, in time or space; for example, placing a gesture on a different stage in a different position or delaying it until after another event. This empowers the stages to decide what kind of interactions and/or interference they allow from other stages, as well as allowing them to shape the input from the other stages more directly; for example, deciding which part of a remote stage is allowed to interact with the local stage.

1.3 Contributions

This dissertation makes a set of contributions briefly summarized in the following subsections and more thoroughly exposed in Chapter 7. The contributions are the result of both the effort in designing, implementing and measuring the

systems composing MultiStage and the hindsight knowledge gained in writing this dissertation.

1.3.1 Lessons Learned

During the design, implementation and experiments we distilled few drops of practical wisdom. The following list represent a meaningful subset we believe relevant to highlight among the contributions.

Implicit and Explicit state changes. In principle a stage can modify the state at another stage in two ways: (i) implicitly by modifying the state sent to other stages or (ii) explicitly by asking the other stages to change the state. There are implications and trade offs in both choices.

Single-data stream, single user. The Detection system collects the state of the stage in the context of single users and put this information in a per user data stream. The advantage of having a one-to-one ratio between users and streams is that the remote presence of each user can be individually manipulated with low resource usage.

RGB-D cameras can reduce the CPU usage when detecting an actor's state. RGB-D cameras can be used to record what is in a predetermined volume. If an actor is inside this volume few assumptions can lead to fast detection of the users.

Remote presences as 3D point clouds. Creating a remote presences using 3D point clouds preserves the shape and proportions of a human being, allowing to express, even if in a limited way, the body language of the acquired user.

Observer redefines observed. An observed user on stage can be mapped or redefined into an object being simpler to analyze when looking for gestures. The users are aware of the simplification and adapt their behaviour accordingly. The simplification saves processing and reduces delays.

1.3.2 Models

Among the contributions of this dissertation there are the models used in designing MultiStage. These model are listed here as reference and support to some of the design decisions of MultiStage. To be noted that the following is a short list of the most significant models and not a comprehensive list of all the models that can be found in MultiStage. In Section 1.4 these models

are correlated with the published papers that constitute the platform of this dissertation.

Decoupled producer and consumer with monitored distribution. Producer and consumer are not directly connected, they exchange data through via a distribution system. The distribution system is also monitoring the network performances and can intervene when necessary. Intervention can include replacing lost packets or switching to a pre-recorded stream.

Global Interaction Space. The Global Interaction Space allows a user to run commands on the computers in a room with gestures. A user can steer the computation of many computers in a room without walking, allowing the user to run scripts at the same time on multiple computers.

A gesture recognition model based on simple volumetric detection of users. With 3D sensors is possible to track the volume occupied from an actor on stage. Characteristics of the volume can be changed by the user and tracked to detect gestures. In this way is possible to detect simple gestures with low processing and delays.

1.3.3 Artifacts

Among the products of system research are the artifact constituting the system explored. The following artifacts are an extract of the most relevant. Their relevance to the state of the art is exposed later in this chapter, Section 1.4.

Sensor Suite. The Sensor Suite detects actors on stage, it comprises four 3D cameras and two computers. It has a cumulative horizontal field of view of almost 360 degree.

A User Context State Detection, Analysis, and Sharing System. This system is the software counterpart of the Sensor Suite, it process the data coming from the sensors and encodes it in one data stream per user. After encoding and compressing the streams it delivers them to the distribution system.

Remote Presence system. The Remote Presence system renders colored point clouds on a display. Remote actors can interact visually with the remote presence as if it was another person at the same site.

Bounding box point and point motion system. When users are detected the volume they occupy is approximated by a bounding box. The bounding box is built by identifying the six points that hold the maximum and

minimum value for each of the three coordinates X, Y, and Z. A user with knowledge of how the volume is built and where the control points are situated can move them with his body, and perform gestures.

Gestures through Regular Expressions and User Volume Control Points. This system extends and completes the *Bounding box point and point motion system*. It translates the control points movements in strings of text. The text is searched with regular expression to identify gestures.

The Global Interaction Space. The Global Interaction Space system is the implementation of the Global Interaction Space model.

1.3.4 Facts

During the experiments we collected data on the performance of MultiStage as a whole and on its subsystems. It very useful to use benchmarks in order to compare different version of a software to keep track of its evolution, but it is not trivial to design benchmarks that are serviceable for comparison with other systems. For this reason the performance measurements here exposed are not benchmarks results but the data acquired during experiments in condition as close as possible to real use. The meaning of these data is to give the reader an idea on the resource utilization of MultiStage. In this way is possible to better comprehend the requirements and the capabilities of a MultiStage deployment.

CPU utilization All the CPUs involved in the experiment were Intel Core i7 at 2.7 GHz and Intel Core i5 at 2.5 GHz. The only system to use 50% of the CPU was the Sensor component of the Global Interaction Space (see Chapter 6) on an Intel Core i7 at 2.7 GHz, while the others never used more than 25% of the available CPU.

Memory utilization During the experiments the memory consumption observed on single machines for each system was never above 45%. However, not all the systems were running on machines with 8 GByte of memory, some had only 4 GByte (see Chapter 5). Normalizing the memory utilization of the systems running on 4 GByte of memory to the ones with 8 GBytes we can reconcile the memory utilization of all the systems to be below 25%.

Bandwidth per stage The bandwidth used was a proportional to the number of streams and number of points in the point cloud: streaming four point clouds of 5K points needs 7 MByte/s.

Scalability Combining these facts with the bandwidth used we can deduce the scaling factor of Multistage. If each stage is equipped with four cameras (7 MByte/s with 5K points see Chapter 5) and each stage receives all the streams, assuming an ideal linear scale, at three stages each stage receives 21 MByte/s. At four stages the needed bandwidth is 28 MByte/s, and so on.

Given these numbers, and assuming a central distribution system, the discriminant for the scalability of MultiStage is the bandwidth. In case of a three stages setup it will be 63 MByte/s, with four stages it will be 112 MByte/s. At four stages a Gigabit link will theoretically still be enough (125 MByte/s), but it will need better hardware or a better system architecture or better compression of the transmitted data to scale further.

1.4 Publications

This chapter presents a short description of each paper highlighting the contributions and concepts brought by each of them. Following that, there are tables explicitly connecting the contributions to the papers and the papers to the chapters of this dissertation.

In the description of each paper the dissertation's organization of the contributions is used. This will aid in correlating from which paper each contribution stems. For each contribution the chapter numbers in parenthesis identifies where in the dissertation the topics are treated.

1.4.1 MultiStage: Acting across Distance

The paper reports on a prototype system helping actors on a stage to interact and perform with actors on other stages as if they were on the same stage. The main findings and contributions are listed below.

- Lessons learned
 - Single data stream, single user (Chapter 3).
 - * By using appropriate sensors complex and possibly long time running processing to create a data stream about an actor is not necessary. This will contribute to a lower end to end delay in the system.

- * Flexibility: move around RP, each actor can be treated individually: means the position at the stages and if you lose one data stream the other are not influenced, cut down on bandwidth use by requiring only some of the streams. Analysis can be performed only on some streams.
- Models
 - Decoupled producer and consumer with monitored distribution. The producer and consumer of the data streams are decoupled. Producer and consumer communicate through a third system that takes care of monitoring the delays. When the delays are judged too large the third system takes action by masking the effects of delay as described by Su et al. [3]. This model is mentioned for completeness and not further discussed in this dissertation.
- Artifacts
 - A User Context State Detection, Analysis, and Sharing System (Chapter 1.3.3).
 - * User Context State Detection (Chapter 3)
 - Single data stream, single user: Chapter 1.3.1. For each user a separate data stream is created for further use by other system.
 - Multistage Sensor Suite: Chapter 1.3.3. At each stage four 3D cameras tiled back to back for an almost 360 degree view, continuously record actors.
 - * Analysis: the system processes the recorded data on-the-fly to discover actions by actors that it should react to. A gesture recognition model based on simple volumetric detection of users (Chapter 1.3.2).
 - * Sharing: the system streams data about actors and their actions to remote stages.
 - Remote Presence system (Chapter 5). At each stage each actor is represented by a remote presence. The prototype uses a visualization of the actor.
 - Masking the effects of delays. When the remote presences lag behind

too much because of network and processing delays, the system applies various techniques to hide this, including switching rapidly to a pre-recorded video or animations of individual actors.

- Amplified interactions: the system amplifies actors' actions by adding text and animations to the remote presences to better carry the meaning of actions across distance.
- Facts
 - The system currently scales across the Internet with good performance to three stages, and comprises in total 15 computers, 12 cameras, and several projectors.

1.4.2 Global Interaction Space for User Interaction with a Room of Computers

To interact with a computer, a user can walk up to it and interact with it through its local interaction space defined by its input devices. With multiple computers in a room, the user can walk up to each computer and interact with it. However, this can be logistically impractical and forces the user to learn each computers local interaction space. Interaction involving multiple computers also becomes hard or even impossible to do. This paper presents the following list of contributions.

- Lessons learned
 - We have found the principle of customization to be a simple way of making the local side do exactly what the global side has defined. To customize a computer entering the room, a one-time overhead is taken when downloading action scripts to the computer. This reduces the traffic between the global and local side when low latencies matters the most, which is during actual use of the global interaction space.
- Models
 - Global Interaction Space model: a global interaction space lets users, through in-room gestures, select and issue commands to one or multiple computers in the room (Chapter 6).
 - A gesture recognition model based on simple volumetric detection of users(Chapter 4).

- Artifacts
 - Global Interaction Space system: The architecture of the system defines functionalities to (i) sense and record the state of a room, primarily the state of computers and users. The state of users includes gestures that the user can perform. (ii) translate gestures in to commands. (iii) issue commands to computers and have them execute the commands(Chapter 6).
 - * The prototype was used in a room controlling multiple computers and a display wall.
- Facts
 - CPU and memory and network resources usage is low or insignificant.
 - The latency of detecting a gesture is interactively fast.

This paper won the best paper award at HSI 2013, 6th International Conference on Human System Interaction [5].

1.4.3 Controlling and Coordinating Computers in a Room with In-Room Gestures

This paper is a further development on the precedent. It formalizes concepts that were present in the previous one but not yet well defined or explored. The main contributions are in the list below.

- Lessons learned
 - No lessons were learned solely based on this paper.
- Models
 - Bounding box point and point motion system (Chapter 4).
 - * The volume containing/wrapping the user is detected by sensors and approximated by a bounding box. The bounding box is constructed by determining the maximum and minimum points occupied by the user along each of the XYZ axes (Fig. 4.3 shows a simplified example in two dimensions). By moving their bodies the users are able to move these points, termed

control points.

- Artifacts
 - Gestures through Regular Expressions and User Volume Control Points (Chapter 6).
 - * Given the users the knowledge of the control points they can move their bodies to perform gestures by moving the control points. Translating the motion of the control points into strings of characters allows us to detect the gestures by matching them against regular expressions. This gives the flexibility of detecting a different set of gestures by defining a different set of regular expressions (Fig. 4.4 provides a visual explanation).
- Facts
 - The latency of detecting a gesture is, in average, between 0.9 and 1.3 seconds (see Chapter 4.5.1).

The content of this paper are summarized in a video poster [6] presented at the Verdikt 2013 conference [7] where it won the best poster award.

1.4.4 Mapping of contribution and publications

This section consist of a table that explicitly links the contributions listed previously in this Chapter to the respective publications. In the table header are indicated the publications as the Section in which they are presented.

Contribution \ Publication	Multi Stage	GIS	Gestures
Explicit and implicit state changes			
Single-data stream, single user	✓		
RGB-D cameras can reduce the CPU usage when detecting an actor's state	✓	✓	
Remote presences as 3D point clouds	✓		
Observer redefines observed			✓
Decoupled producer and consumer with monitored distribution	✓		
Global Interaction Space Model		✓	✓
A gesture recognition model based on simple volumetric detection of users		✓	✓
Multistage Sensor Suite	✓		
A User Context State Detection, Analysis, and Sharing System	✓		
Remote Presence system	✓		
Bounding box point and point motion system		✓	
Gestures through Regular Expressions and User Volume Control Points			
Global Interaction Space Artifact		✓	✓

Table 1.1: Map of per paper contributions. For brevity the paper are referenced by a shortened name and not by full title of the publication.

1.4.5 Mapping of publications and Chapters

This section consist of a table that explicitly links the publications listed previously to chapters in this dissertation. The connections are between systems, designed and implemented, and the publication they were part of. The systems are extensively described each in their respective chapter.

Publication \ Chapter	3	4	5	6
MultiStage: Acting across Distance	✓		✓	
Global Interaction Space for User Interaction with a Room of Computers		✓		✓
Controlling and Coordinating Computers in a Room with In-Room Gestures		✓		✓

Table 1.2: Map of publications and chapters of the dissertation. This table connects the systems described later in the dissertation with the publications they were designed and implemented for.

/2

MultiStage Overview

2.1 Motivation

Using the Internet, communication is easy even across continents; we are used to interacting more or less in real-time with people at relatively remote distances. We also have the availability of a diversified and rich set of communication means with different trade-offs in terms of interactivity, availability, and persistence. A phone call, for example, has high interactivity; we speak with our interlocutor as if he was present in the same room. A phone call though has usually no persistence, can be recorded but usually is not, and the availability of the service is dependent on two people making and receiving the call. E-mail, in contrast, has good persistence, can stay on a server for years, and the persistence can be indefinitely extended. But the level of interaction is lower than in a phone call. In other words, sending and receiving e-mail is not like having the other person in the same room. In addition, the availability of the service is not person dependent in the e-mail, but the response time usually is.

Other *flavors* of interactions are available depending on many factors, such as relative position of the users and the object of the communication – gossip, stock options, event data stream, etc. One particular niche is occupied by artistic stage performance, where the interactivity of the communication must be as close as possible to having the performers in the same room. In the era of the Internet and easy communications, artists want to be able to perform together, on the same stage, but from different locations. The challenge is to make this

possible using the same means and technologies used for the other kinds of long distance interactions, and manage to obtain a consistent performance for both the actors and the audience. The motivation for this work is to explore, and possibly mitigate, using a computer science point of view, the problems that keep the artists from performing together remotely.

2.2 Ideas of MultiStage

In the context of a distributed performance, we conceptualize a system to evaluate and investigate a computer-mediated collaborative stage show. The given system tries to give the participants (performers and audience) a consistent view of the stage. More particularly, we are interested in hiding the effects of the unavoidable delays that occur among the performers on different virtual stages during a show, especially if the performance is distributed across different continents. As mentioned before, the information maximum speed is the speed of light, and this already sets a minimum always-present latency, without mentioning the other delays introduced by sampling, processing and transmission of signals, visualization, and so on. With this contextual information, we can assume the delays between stages are unavoidable. The goal of our system is, then, to mask the effects of the delays and give the performers and the audience the impression of a synchronized and distributed performance.

The effects of the delays can be masked (or reduced) in many ways. MultiStage focuses on the remote presence of the participants and its manipulation in an interactive way; for example, using a computer to drive a remote presence, in place of a performer, to follow a script when the detected latency at a remote stage is above a threshold. The manipulation also includes user generated events, such as gestures, to trigger other forms of visual interaction. These interactions can span from special effects and visual enhancements, such as body parts emitting sparks or glows, to a more direct medium such as a text bubble appearing on top of an actor to convey information to the audience.

2.3 Concepts

2.3.1 Temporal Causal Synchrony

In a distributed stage show, causality of the actions of the actors is paramount to the enjoyability of the show for the audience. Even if causality is preserved, delays are present, and based on the amount of delay and the demands of the interaction we define different levels of temporal causal synchrony.

Temporal causal synchrony can be *loose* in the case of low demands on delays, as in a teleconference call where the interaction is unstructured and the parties can tolerate a not well-defined amount of delay. When the interaction is more structured we fall into the case of *interactive* temporal causal synchrony.

In this case, the actors/performers are dealing with a possibly rapid action–reaction situation; for example, dancing or martial arts. For these scenarios where, due to delays, the temporal causal synchrony cannot be achieved, we defined the following approaches to mask the effect of delays.

Actor feedbacks. The actors react to the remote presence video as if it was the real actor. The system does not provide any other means of preserving the temporal causal synchrony. Depending on varying factors, the delays perceived by the audience and actors can be intolerable, preventing any interaction paradigm except loose causal synchrony.

Shared clock, shared performance start time, individual actor scripts. The system synchronizes the clocks of all computers involved in the performance (at each stage/site). We set a start time and begin a countdown on each stage. When the countdown ends, the performance starts at the same time on all the involved stages, and the actors will start the performance in accordance with a script that defines not just the actions, but also the timings of such actions. The scripts must include the delays involved; this implies a priori knowledge about the delays expected during the performance.

Shared clock, individual performance start time, individual or shared actor scripts. The system synchronizes the clocks of all computers involved in the performance. We choose one of the stages to be live and the others are secondary stages. We measure the delay from the secondary stages to the live stage and adjust the start time (and the countdown) of each secondary stage accordingly to the live stage start time plus each secondary stage delay. When the countdowns finish, each actor will start performing following their script. At the live stage all the remote presences will be in interactive causal temporal synchrony with the actors on stage. The actors and audience at the secondary stages will experience the effects of delay.

Act-by-wire. The system synchronizes the clocks of all computers involved in the performance. The computers are constantly monitoring the delays between the stages among other metrics. If any of the delays is above a threshold, the system tries to alleviate the perceived latency by using manipulations of the remote presence. These manipulations can be as simple as substituting the remote presence with a prerecorded version or

blurring the remote presence in order to hide the delay to the audience. More compacted manipulations can include the prediction of the movement of the remote presence, or if possible, animate the remote presence according to the script.

2.3.2 Amplified Interactions and Gestures

On a theater stage, with a significant physical distance between actors and the audience, bold makeup, clothes, and exaggerated movements are used to better project to the audience what the actors are doing. In remote interactive performances there is a distance not only to an audience, but also between the actors.

Consequently, the actors need their appearance, movement and gestures to be amplified such that they become easier to see and understand both for the other users and for the audience. In this way we extend the range of human interaction to remote locations and enrich the communication between them. We term this amplified interaction.

To be able to detect what an actor is doing, we must surround him with an interaction space. An interaction space detects human movements, and analyzes them looking for gestures. A gesture represents a predefined command to the system to execute code to do some functionality. A gesture can be simple, like raising an arm, or complicated like doing two-arm movements. They can also be active like walking in a specific direction or passive as in standing still posturing. A collective (collaborative) gesture is a combination of the above kinds of gestures. Collective gestures can happen at the same stage, or be distributed, comprised of gestures from multiple stages.

For example, when two actors at different stages, within some short timespan, raise their left arms above their head this can be interpreted as a command to the system to animate a lightning between the two raised arms and display it on all the displays. Based on the gestures we can create effects in the remote presence manifesting itself at remote rooms. A user's arm movement can in the remote presence be amplified by having a text bubble appear in the video, and by adding other visual effects to the representation of the user. The users remote presence can even be enhanced by executing a model of the user and using its output as the basis for the remote presence.

2.4 Architecture

The basic functionalities needed to achieve a distributed performance are: (i) detection of actors and their gestures, (ii) remote presence of the actors with possible amplification of the interaction, and (iii) distribution of the remote presence (data).

Detection means capturing each actor on a stage, their pose and position through time, and extracting data from the stage to enable the detection of gestures and reproduction of the actors through a remote presence. Remote presence means representing the remote actors on the local stage in a way that makes it possible to interact; in other words, providing a *replica* of the actors on the local stage.

It is also a goal to preserve space and time relationships between the single actors; in this way we are able to reproduce the performance or part of it. For this reason, we focus on the state of the actors on the stage, monitoring the state of the single users and separating the user state from the background scene. Audio and video are the most commonly used mediums to convey an interaction-worthy presence of a remote user; other means, such as robots, can help. Our take on the matter is to use a 3D point cloud to provide the interaction between stages.

Distribution is needed to connect the multiple stages by transferring the data obtained by detecting the actors. Distribution is from the detection stage to the stage where the actors are replicated via a remote presence.

A functionality that receives all the streams, in a similar way to the distribution, can perform a global analysis of the streams. The result of these computations can be a different number of streams than the ones received; for example, global gestures can be detected and streams of gestures distributed to the stages. This functionality needs to be placed before the distribution; it needs a global view of the state of the stages in order to perform a global analysis.

The way these macro functionalities exchange data is through streams. Given that each actor needs to be detected on a stage and *replicated* on another one (or more than one), at least one stream is associated with each actor from detection to remote presence. Therefore streams of data are the output of the detection, the input of the remote presence, and the input/output of the distribution.

The MultiStage architecture comprises a *begin point* and an *end point*. The begin point is a producer of data and the end point is a consumer; in other words, the begin point handles the detection and the end point handles the remote

presence. The begin and end points are linked by an inter-stage distribution system. Each stage is (potentially) composed of one or more begin and end points; in this way, the stages share a distributed performance. Figs. 2.1 and 2.2 explain the layout of the architecture – how the functionalities are arranged and interconnected.

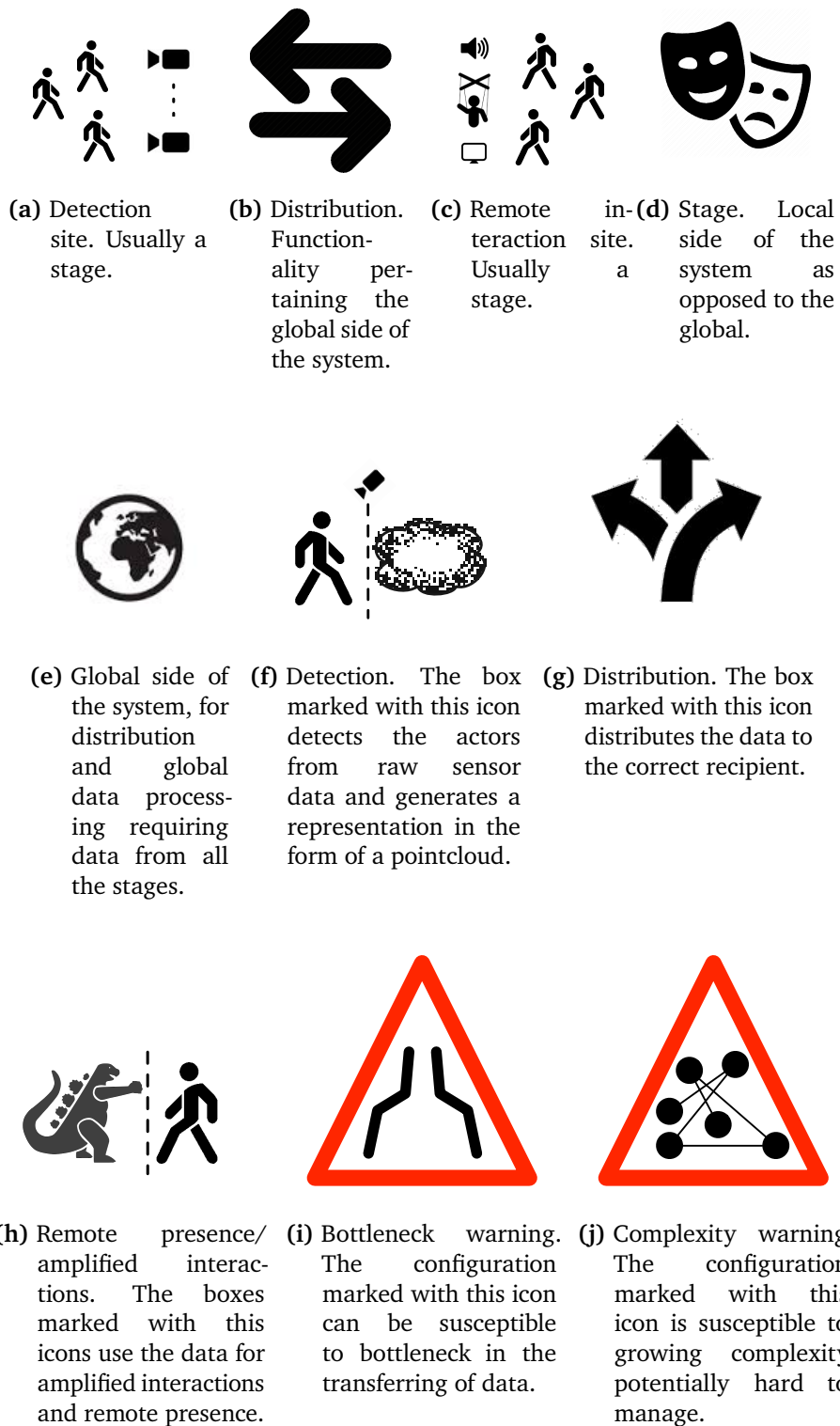


Figure 2.1: Legend of icons.

Fig. 2.2 is organized as a table, with a header on top and a *column header* on the left. The top header indicates the functionalities of the architecture; the column header on the left indicates whether the macro functionality is present on a stage (detection and remote presence) or whether it is global (distribution). Both headers are annotated with icons describing the meaning of rows and columns; Fig. 2.1 provides a legend for the icons. It is possible to see the functionalities present on the stages (top and bottom rows) or globally (middle row).

This logical separation of what pertains on a stage and what is relevant for the whole MultiStage system splits the system into two *sides*, mentioned from now on as the *local side* and the *global side*. It can also be noted that the macro functionality boxes are decorated with an icon showing the additional tasks they perform. For example, the detection (see chapter 3) is performing analysis of the data obtained by the cameras (Fig. 2.1f), while the remote presence (see Chapter 5) is allowing people from different locations to interact, and possibly amplifying this interaction (Fig. 2.1h).

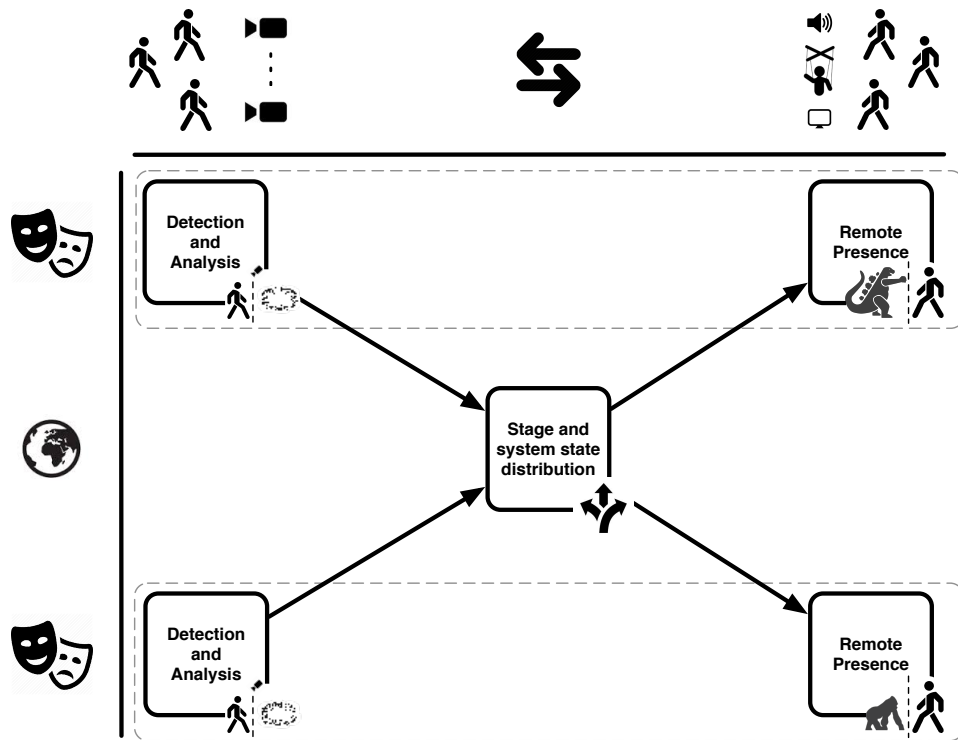


Figure 2.2: Illustration on the architecture. Multiple stages and functionalities, at each stage we have a detection and analysis system and a remote presence system. The data is exchanged, and fed back to the stage, using the global distribution system.

Multiple reasons, including experience from building different systems, led us to this architecture. Extracting a stream of data for each actor at the begin point gives more flexibility for manipulation further along the flow path of the data. Also, new streams can be produced by analyzing the data on the way. Consumers with limited resources and/or limited interest in collecting all the available streams can *subscribe* to only those of interest. This implies that other functionalities can be implemented by manipulating the new streams because they are generated at the begin point. A different combination of functionalities can be chained for different results. We termed this architecture *balanced*.

The main purpose of detection, as will be explained in detail in Chapter 3, is to detect users on stages and provide their state in different streams, one for each user on each stages. This functionality needs to be fixed here at architecture level to allow subsystems leverage on the availability of *personal streams* of data that can be manipulated individually. It is possible to have different implementations of the same functionality as well as alternative architectures. In the following subsection we review a few alternative architectures.

2.4.1 Alternative Architectures

Other architectures could provide the same results with different trade-offs. An example in Fig. 2.3 is where all the computation is done in the global distribution and not much in terms of functionality is left at the stages. Raw data from the detection is processed and the result is augmented, and possibly prerendered. The rendered output, a sequence of images or a video, is streamed to the end point. This solution can look simpler at first – all the computation is kept in the same (at least logical) location, but the bandwidth can be a bottleneck. And scaling problems are likely to arise quickly. We termed this architecture *light begin point, fat global side, light endpoint*, or *L-F-L* to point out that the greatest share of the work is done on the global side. As can be evinced in Fig.2.3.

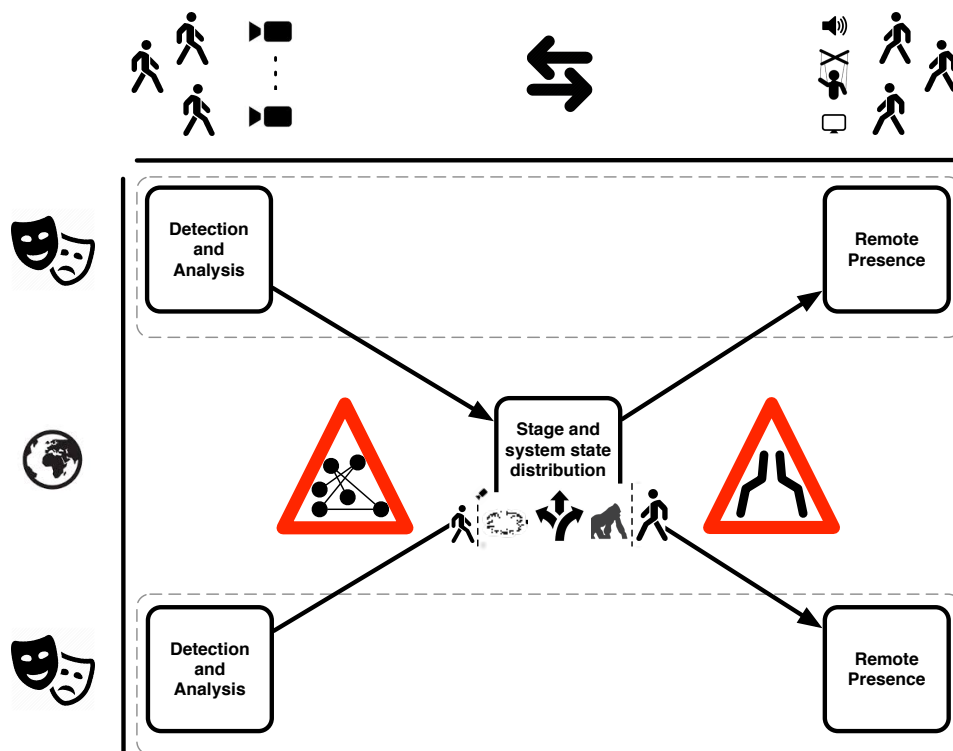


Figure 2.3: Light begin point, fat global side, light endpoint. In this architecture most of the work is done in the global side. The processing of the raw data from the sensors and the remote presence/amplified interactions. The end point receives streams of preprocessed data. Bottlenecks in transmitting raw data from the begin point and already rendered data to the end point are to be expected. As it is expected an increase in complexity to make this architecture scale.

How much bandwidth is needed to send the raw data to the global distribution, and how computationally intensive will it be? How many stages can the global distribution and computation serve before needing more computational power? In other words, will it scale? In our opinion, it would have been hard to make it work properly with commodity hardware without a major redesign and a much steeper cost in terms of complexity. So, to keep the system simple we did not select this particular architecture, and kept the data, logically and spatially, close to the computation.

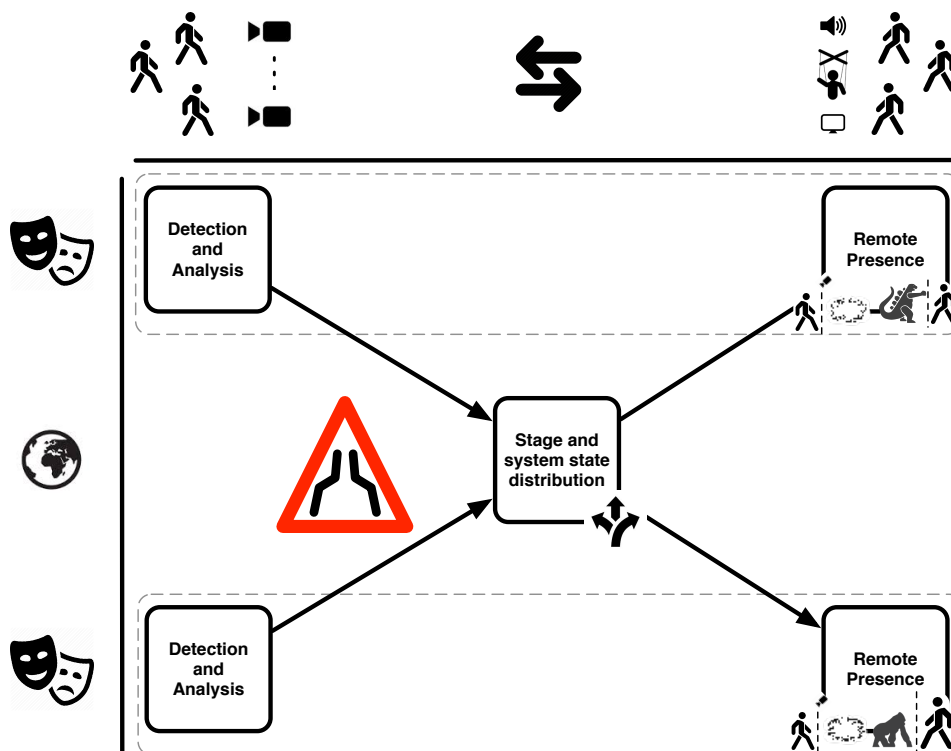


Figure 2.4: Light begin point and global, fat endpoint or L-L-F. Charging the endpoint with all the computational load can be lead to bottle necks due to the raw data from the sensors to be delivered to the endpoint.

Another alternative in Fig. 2.4 allocates all the computations at the endpoint. This is a different solution than the one depicted in Fig. 2.3 but the distribution can still be a bottleneck. This is especially true in cases where the stages are spatially far apart; for example, on different continents, where the bandwidth available may be prone to unpredictable fluctuations [8]. We termed this architecture light begin point, light global side, fat end point, or with the abbreviation pattern used before, L-L-F.

At the other end of the spectrum to the solution in Fig. 2.4 we find the mirrored architecture described in Fig. 2.5, or *F-L-L*. The distribution can also be a bottleneck for this architecture. The begin point detects the actors and processes the streams. The rendering of the streams also happens at the begin point and all the preprocessed data is sent to the end point. In this case, the data represents the final rendering of the remote presence and can be encoded in a single event data stream. But one single stream would reduce the flexibility of the Remote Presence system. The possibility of having global gestures and global events is more complex (and potentially resource demanding) unless the begin points

can bypass the global Distribution system and directly communicate with the end point, complicating the architecture even further.

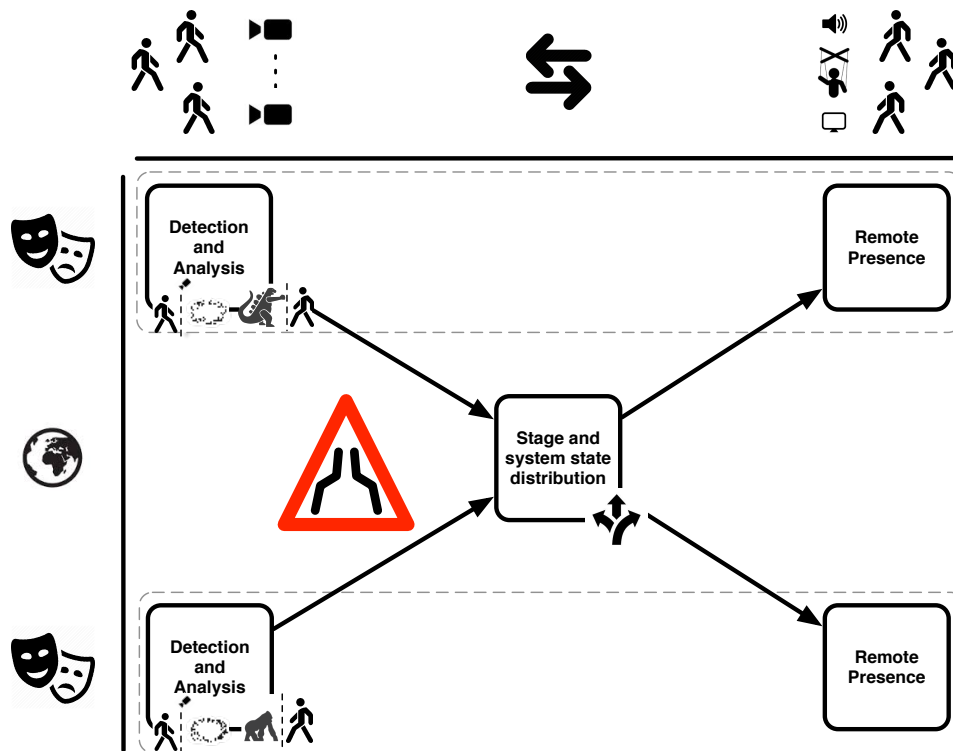


Figure 2.5: F-L-L. At the other end of the spectrum we have all the computational load on the begin point. The same concerns from the L-L-F architecture are pertinent here: delivering the rendered output to the

2.5 Design

The MultiStage system is designed following the *balanced* architecture. Fig. 2.6 shows a simplified version of the data flow. The local sides represent functionalities that are located on the stages and are directly connected to the stages and/or the actors. The global side comprises functionalities that encompass all the stages or the MultiStage system in general. The figure is a symbolic illustration of how the data flows in streams from begin point to end point. It should be noted that the begin and end points can both be on the same stage and in more than one instance. In other words, a stage can be *composed* of many instances of the detection and remote presence functionalities. Fig. 2.7 shows an example with two stages. The data collected is analyzed by the Local Detection/Analysis subsystem, and the results streamed to the global side to

be distributed or processed further by the Global Analysis subsystem.

The Local Detection/Analysis subsystem represents the begin point; it detects the users/actors using 3D (or depth) cameras (or any other device that can perform a 3D detection of the actors on stage), performing detection of the users in the stage volume. The output of the system is a colored point cloud for each camera.

A colored point cloud is a list of 3D points each with an associated color. During detection, all the pixels containing the actor are selected from each image captured, and in the case of 3D cameras each pixel has an associated distance from the camera. During analysis, each tuple depth pixel - color pixel is used to compute the real-world coordinates of the pixel, generating a 3D point with a color. The result is a colored point cloud representing the actor on the stage for each frame of each camera doing the detecting. Each point cloud is packaged with a header containing the stage identifier, the camera identifier, a stream type identifier, a sequence number, and a timestamp. The header minus the sequence number and the timestamp serves as a stream identifier; the systems receiving the resulting packages can discriminate their origin, as well as discriminating a point cloud stream from a gesture stream or a control stream for internal use.

All the streams are sent to the global side which, having them all available, can elaborate the data by using global knowledge. This elaboration can develop in the creation of other streams; for example, a stream comprising a collective gesture performed by actors on more than one stage. After the global analysis, the streams are sent or *published* to the distribution. Other systems can then *subscribe* to any of the streams using the stream header (minus sequence number and timestamp) as identifier.

The Remote Presence system is an example of a subscriber. It aims to recreate part of the reality of one stage on another one, in a manner that allows interactions between the local stage and the remote ones. To achieve this goal it needs to receive the streams published to the distribution. Different kinds of remote presence can use different kinds of streams; for example, a point cloud stream can be directly rendered to a video. A robot impersonating an actor might need a more sophisticated stream containing the position of limbs. A simpler remote presence can be implemented by, for example, projecting a dot on the position of another actor on a remote stage.

Another task performed by the Remote Presence system is Amplified Interactions. This term encompasses all the possible modifications or enhancements that a remote presence system can accomplish to reinforce the virtual presence of remote actors on the local stage. These sorts of *stage tricks* are not unknown

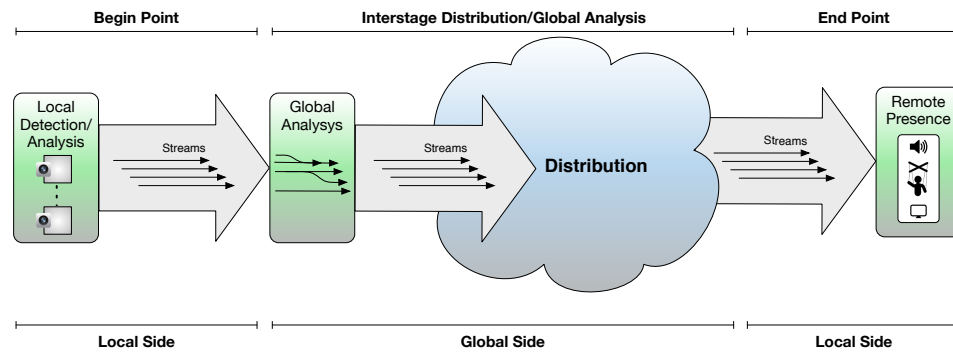


Figure 2.6: Schematic data flow of the Multistage system. The Local Sides represents functionalities that are located on the stages and are directly connected to the stages and/or the actors. The global side comprises functionalities that pertain all the stages or the Multi Stage system in general. The figure is a symbolic illustration of how the data flows, in streams, from stage to stage.

to actors who use makeup, clothing or grandiose body language to emphasize their acting and reach out to the audience. The Amplified Interaction goal is to provide the remote actors with some of the capabilities they have on a real stage and some others that are not possible on a real stage. For example, in the case of a video remote presence, a 3D model could impersonate the actors with a different shape/form (maybe not even human) depending on their position on the stage, or trigger the change with a gesture. Other possible enhancements are actors appearing on a different part of the stage or having a double repeating all their actions.

Two other subsystems are present and vital to the MultiStage system. They are not presented in this work but can be found in Su et al. [9]. Nonetheless, the description of the design would not be complete without a brief introduction to the subsystems and an overview of the functionalities provided. The first functionality is performance monitoring, and the second is masking the effects of delay. The two systems are synergistic; they provide the MultiStage system with performance measurement and tolerance to unexpected delays and the former is needed to achieve the latter. The Performance Monitoring system is designed as a two-part system – a global collection and processing point and a local subsystem monitoring the performance of the computers composing the MultiStage system more details in Su et al. [3]. The metrics collected are memory utilization, CPU utilization, data transmitted and received, and latency between the single machines and the Distribution system. Another important task performed by the monitoring system is to keep the clock of the machines reasonably in sync with Network Time Protocol (NTP).

The other system performs *Controllable Temporal Synchronization* (see Section 2.3) for the Remote Presence systems. Briefly, this means that it controls the flux of information delivered to the remote presence and can react to delays in communication. For example, if the delay is deemed sufficient to disrupt the interactivity of the distributed performance, it can switch the stream from the live (and delayed one) to a prerecorded one masking the effect of the delay. For this control to be possible, the Remote Presence system needs to receive the data from the Controllable Temporal Synchronization system. Fig. 2.8 explains the additional flow of data needed by these two systems.

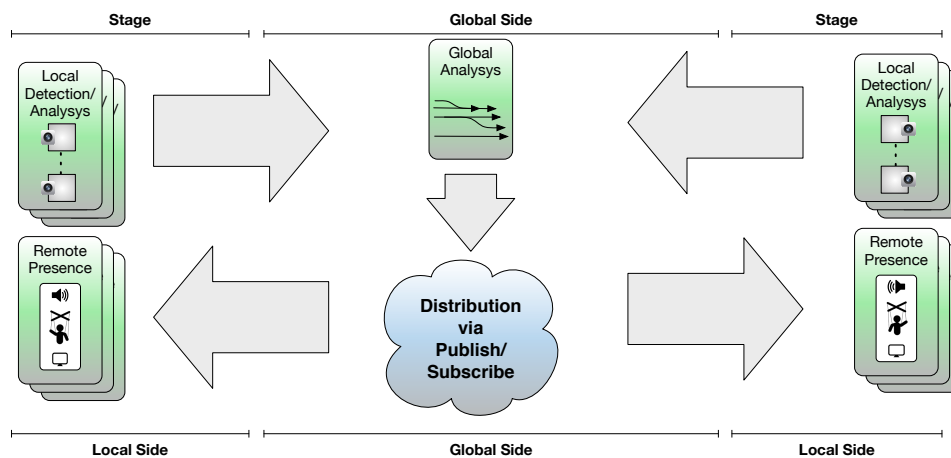


Figure 2.7: Design of the Multistage system.

2.5.1 Discussion

As explained before in Chapter 2.4, the whole system delivers data streams from the begin points to the end points. The data streams are detected by 3D cameras grouped in the center of the stage and pointing away from each other. The camera cluster, we call it the Sensor Suite, detects a predefined volume inside the room covering close to 360 degrees around itself. We chose to use 3D cameras this early in the design because they became available and are inexpensive to buy. In addition, we are familiar with the most common pain point of computer vision, and the use of 3D cameras combined with few assumptions can go a great length to ease the computations needed to detect users and their movements. To give an example, with a 3D camera we can isolate a volume inside a room; this basically isolates the objects in the volume from the background.

The *traditional* way to do this is with computer vision algorithms. A common approach to achieve the same results is background subtraction; this can be seen both as a solution and as another problem, as explained in Piccardi [10].

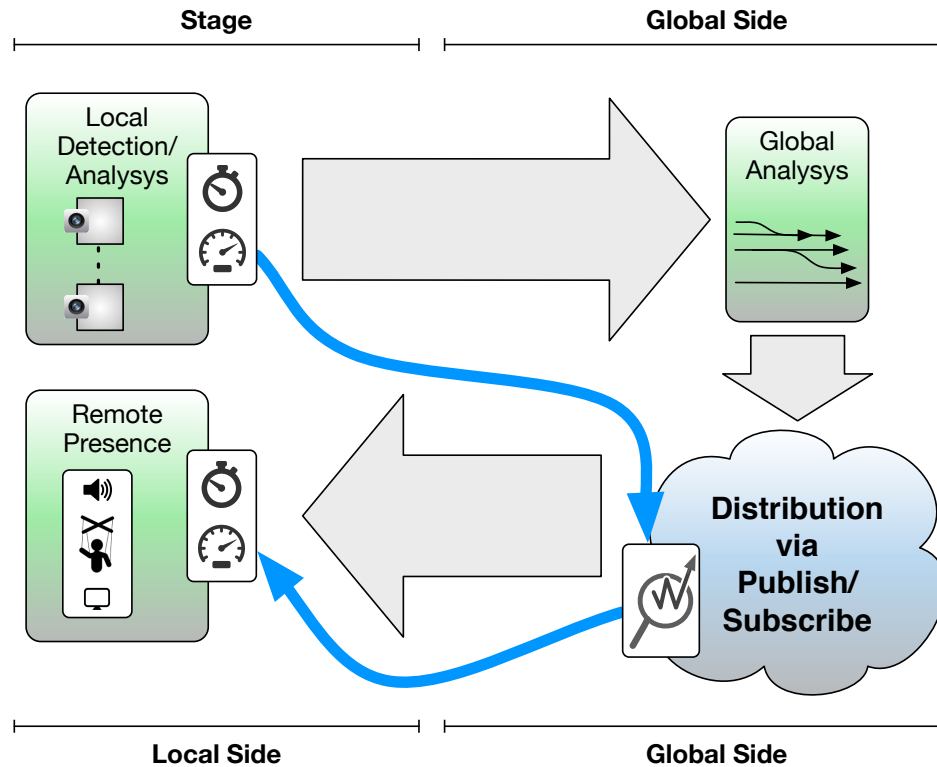


Figure 2.8: The figure shows how the Monitoring and Controllable Temporal Synchronization systems overlays the other component of the Multistage system.

Pertinent to the issue is the illumination of the environment – a slight change in the light conditions can cause a major disruption in different algorithms relying on the intensity value of the pixels, both color and black and white. A sufficient change in the illumination can happen in a few hours in a room with a window, caused by the sun moving in the sky. Of course, there are solutions to this problem that involve more processing of the images, more complexity and potentially more delay (not taking into consideration accelerators or graphical processors). The use of a 3D camera makes this problem more manageable by not being influenced by most of the indoor light, delivering the pixel inside the volume of choice. This brings a reduction in complexity and potentially reduced latency due to less processing.

The global side comprises the distribution and monitoring in addition to the global analysis. The Distribution system protocol follows the *publish–subscribe* paradigm. The motivation for this is that not all the end points may want to receive all the streams. Some might be interested in only a few or do not have the resources to handle them all. This improves the overall flexibility of the system

The monitoring subsystem collects system-wide performance metrics from the local side and delivers them to the global monitoring system. In the case of network disconnections or disruptive delays, the *Controllable Temporal Synchronization* can take action and mask the effects of delays. For this to happen, the price to pay is an increase in complexity – the Remote Presence system needs to receive the data from the *Controllable Temporal Synchronization* system. This has a cost also in terms of delay – the data is received by a process and sent to another one. Even if these processes are running on the same machine, the delay is not zero.

Technological note

At the time of writing, there is a proliferation on the consumer market of different sensors/devices that could have been substituted in part or entirely for the use of the 3D camera. The design of the system tried to keep open as many options as possible by not binding to a specific solution. Any sensor output can be translated into 3D points; therefore, the MultiStage system could be adapted to use other input devices such as wearable motion sensors, floor pad sensors, stereo camera, etc.

2.6 State of The Art

Distributed stage performances have been attempted with various degrees of success and from different *directions*. The previous attempts used various techniques, often tailored to address one specific dimension of the problem. The set of *dimensions* of the problem space include audio and visual remote interactions and in some cases a remote presence.

2.6.1 Landscape/Broad View

There are in literature systems and tools that, even if not built especially for a remote/distributed stage show, present relevant qualities worth mentioning.

Some systems are tailored to a distributed stage, and can focus on achieving the same experience for the audience on each stage involved (Sawchuk et al. [11], Zimmermann et al. [12]).

The most resembling systems are the teleconferencing, they provide a mean to communicate remotely both verbally and visually ([2], [13]). Some tele-

conferencing systems provide the users with some extra functionalities or enhancements over the traditional teleconferencing tools (Tang et al. [14], Nguyen et al. [15]).

These systems can employ more than just one camera or 3D cameras to bring to the users a more immersive experience than just a video stream (Essid et al. [16], Schreer et al. [17], Petit et al. [18], Feldmann et al. [19], Izadi et al. [20]).

An enhanced experience can be obtained not only through immersive 3D environments, but can be the result of a more vivid presence, a remote presence, of the users at one or both ends of the communication channel (Sakamoto et al. [21], Maimone and Fuchs [22], Huang et al. [23], Alexiadis et al. [24]).

Some other systems can allow the remote user to be more present with its *actions* in stead of its *appearance* as in a remote presence, allowing the users to interact remotely (Vasudevan et al. [25]). The approach of interaction can be different but in many cases it involves detecting gestures performed by the users directly, or free handed, or through an input devices (Mistry and Maes [26], Elgendi and Magenat-Thalmann [27], Van den Bergh and Van Gool [28], Pederson et al. [29], Ebert et al. [30], Morishima et al. [31], Farhadi-Niaki et al. [32], Fanello et al. [33], Kellogg et al. [34]).

Many of these systems involve some degree of Human Computer Interaction (HCI), depending on the *dimensions* of the problem and on the purpose of the system. And a large wall sized display can be of great help in aiding the audience and the users to better visualize the performance (Dou et al. [35], Anshus et al. [36], Stødle [4], Wilson and Benko [37], Shoemaker et al. [38], Bragdon et al. [39]), Kim et al. [40], Spindler et al. [41]).

Tele-immersion systems are also relevant. As the name suggests these systems goal is to put the users in an immersive shared environment, where they can collaborate interacting with their respective remote presences. The remote presence created by such systems is usually the result of three dimensional data acquisition and aim to be as close as possible to the reality (Lien et al. [42], Yang et al. [43], Vasudevan et al. [25], Fechteler et al. [44], Mekuria et al. [45], Raghuraman et al. [46], Alexiadis et al. [24]).

2.6.2 Distinctive Systems

Among those systems there are some more close in functionality to a distributed performance and merit more than just a mention.

An example of audio interactions and distributed artistic performance is the Distributed Interactive Performance (DIP) exposed in Sawchuk et al. [11] and Zimmermann et al. [12]. It addresses the music and audio synchronization of two stages.

Two-room interaction systems are described with a focus on achieving audio synchrony. They compensate for the network latency by delaying local actions correspondingly, making both rooms experience the same delay. In Chew et al. [47], a series of experiments based on the DIP system is described with focus on the audio delay, and how the delay affects musicians' cooperation. An artificial delay of 50ms to the remote room's audio stream was tolerable. With the same latency added at both rooms, it became possible to play easily together with a delay of up to 65ms. Adding visual interactions to an audio interaction system we obtain a teleconferencing system. There are many around, some commercial some not, with different capabilities and characteristics. Some are tailored to user cooperation and to a fixed number of locations. Sato et al. [48] illustrate a remote camera system for teleconferencing, supporting user cooperation between a local and a remote room. The system captures 360 degree images as well as supporting pan/tilt/zoom of cameras.

Another approach, with a focus on a three-way collaboration, is in Tang et al. [14], also including some remote presence capabilities. The system allows three people to collaborate in a virtual environment. In each room there is a multi-touch table, camera, speaker, microphone, and two LCD monitors to display the two other rooms. The shadow of remote hand and arm gestures is captured by an infrared camera and displayed on the multi-touch table to show the remote person's behavior. Another relevant system for teleconferencing, focused on informal meetings between rooms, is Dou et al. [35]. The system merges the images from panorama cameras acquiring the background of a room, with a camera acquiring the users when they are close by the display. The key feature of the system is to allow users to maintain eye contact during the conversation by stitching together images from both the panorama and short range cameras.

The collaboration approach is taken a step forward, but in another direction, in Petit et al. [18]. The system presented is a multi-camera real-time 3D modeling system for remote presence and remote collaboration. 3D models of users are computed from two-dimensional (2D) images from multiple cameras, and the 3D models are streamed to remote rooms where users are visualized in a virtual 3D environment. Computing and visualizing collisions and reaction forces to virtual objects in the virtual space strengthen the remote presence.

Another remote presence system, more physical than virtual, is in Sakamoto et al. [21]. The system uses a remote-controlled android, which can have a set

of states to be in: idle, speaking, listening, left-looking, and right-looking. A remote user controls the android's behavior by choosing its state, enhancing the feeling of the presence of the remote user. On the same track of full body interaction, there is in Essid et al. [16], a multi-modal corpus for research into human-to-human interaction through a virtual environment. The virtual environment is defined as a virtual dance studio where a dance teacher can teach students choreographies. Both teacher and students are represented in the virtual studio by 3D avatars. The corpus consists of the recordings of the 3D avatars and outputs from other sensors, such as cameras, depth sensors, audio rigs, and wearable inertial measurement devices.

Continuing on motion classification and interaction, Elgendi and Magenanthalmann [27] present a study on hand gesture speed classification with the goal to improve human-computer interaction. The aim of the study is to train a virtual human to detect hand movements in a noisy environment. The factors of the study are multiple body features like hand, wrist, elbow, and shoulder, evaluated against different gesture speed such as slow, normal, and fast.

Stødle [4] describe a distributed optical sensor system used to implement a device-free interaction space. The system uses an array of commodity cameras as 3D multipoint input to track hands and other objects. The tracked coordinates are used to provide 3D input to applications running on wall-sized displays. We extend this system to suit a multi-display environment; we also use commodity 3D cameras to provide room-wide depth information and gesture detection. Other works on gestures and large displays are Aghajan and Wu [49], Bellucci et al. [50], Bragdon and Ko [51], Blakney [52], Mitra and Acharya [53], Pavlovic et al. [54], Sabir et al. [55], Seyed et al. [56], Shoemaker et al. [38] and Farhadi-Niaki et al. [32].

Another system that uses multiple depth cameras and projectors is LightSpace Wilson and Benko [37]. In LightSpace, the projectors and cameras are calibrated to real-world coordinates so any surface visible by both camera and projectors can be used as a touch screen. Adding the 3D world coordinates of the detected users to this multi-display installation allows for different multi-touch body gestures, such as picking up an object from a display and putting it on another. The body of one or more users can be used to transfer objects to different displays by touching the object on the first display and then touching the other display.

Relevant to the topic is Van den Bergh and Van Gool [28], in which a novel algorithm is presented that detects hand gestures using both Red-Green-Blue (RGB) and depth information. The prototype is used to evaluate the goodness of hand gesture detection techniques using a combination of RGB and depth images. To evaluate the algorithms, a device-free interaction system is developed and

tested. In the same context, Kim et al. [40] present Digits, a personal, mobile interaction space provided by a wrist-worn sensor. The sensor is composed of off-the-shelf hardware components. Digits can detect the pose of the hand without instrumenting it. On the same track of gesture detection, Kellogg et al. [34] proposes to exploit the surrounding electromagnetic waves present in the every modern office/home such as Wi-Fi, TV or cellphone signals. The device, named AllSee, can discriminate some hand motions performed in front of it by how the hand reflects the electro magnetic waves. AllSee implementation is based on energy harvester chips, not unlike the ones used for Radio-frequency identification (RFID), making it a passive device and so reducing the power consumption.

In Bragdon et al. [39], a system designed to support meeting of colocated software developers explores the space of touch and air gestures in a multi-display, multi-device environment. The system is composed of a 42 inches touch-display, two Kinects, a smartphone, and a tablet. Mid-air gestures, like pointing to an object on the bigger display, are supported through the Kinect. In combination with a touch-enabled device, the hand gesture can be augmented to address some of the problems of gesture detection, such as accidental activation or lack of tactile response.

One more special purpose system is Ebert et al. [30]. The system is a touch-free interface to a medical image viewer used in surgery rooms. The system uses a depth camera and a voice recognition system as a substitute for a keyboard and mouse input in an environment where touching those devices can compromise the operation.

Commercial remote conference software includes Skype [2], GoToMeeting [13], and many more. With the emergence of new Web technologies, i.e., Websocket [57] and WebRTC [58], via a browser from any computer available to the users. On the gesture front, many new gadgets are hitting the consumer market, such as Leap Motion Controller and Myo Armband [59], [60].

2.6.3 La Serva Padrona

An example of distributed opera is the play, *La Serva Padrona*, enacted in Tromsø in 2012. The play was *split* into two locations and used displays with back projection to represent the actors on the *other half* stage.

Both stages were in the same building on different floors, and audience and actors were also split in two in accordance to the stages. During the break in the middle of the performance the audience swapped stage to watch live the *other half* of the stage and the actors.

As already stated interacted with the remote presences of the other actors on the displays through displays on each stage. The displays were roughly 2x2 meters placed at regular intervals on the stages in an interleaving fashion, so that gaps between displays on one stage were covered in the other. Between each display there was camera recording the area in front of it and reproducing it on the corresponding display on the other stage. The displays provided a view on the other stage and the choreography put the actors in these views during the play. In this way the actors could (inter)act through these *portals* to the other stage.

Microphones and speakers provided the audio, both for the voices of the actors and for the stage sounds, such as footsteps, or *behind the scene* sounds. The choreography was also tailored to underline and exploit the separation of the two stages: stage jokes and tricks were also adapted to the *double reality*. Figs. 2.9 and 2.10 show the flyer for the event.

Medvirkende

Uberto: Harald Bakkeby Moe, bariton
Bakkeby Moe har tatt Master i utøvende sang ved Norges Musikkhøgskole og gikk ut i 2009. Han har gjort store kor- og kirkeverker, og har rollen som Schaunard i NOSO's oppsetning av "La bohème" i Mai 2012

Serpina: Berit Norbakken Solset, sopran
Etter avsluttet diplomstudium ved Norges Musikkhøgskole i 2005 vant Norbakken Solset i 2006 Rikskonserternes prestisjefylte konkurranse INTRO-klassisk. Hennes spesialitet er kirke musikk, samtidsmusikk og lied, men hun har i de senere år også sunget opera med stor suksess. I 2012 synger hun hovedrollen i den nyskrevne operaen, Heksehimmaren (R. Rasmussen).

Vespone: Jenny Svensson, danser og koreograf
Svensson har Bachelorgrad i dans fra Laban Centre of Contemporary Dance i London.

Musikere:
Bjørn Andor Drage, tangenter
Yuko Kawami, fiolin
Bernt Simen Lund, cello
Nasra Ali Omar, slagverk

Regi-idé og scenografi: Niels Windfeld Lund
Professor i dokumentasjonsvitenskap ved Universitetet i Tromsø, leder av The World Opera and Verdensoperaen i Nord.

Medie-designer og teknisk direktør: Jason Geistweidt
Musiker og elektro-akustisk komponist med doktorgrad fra SARC, Sonic Arts Center, Queens University i Belfast 2006.

Teknikere: Sivert Henriksen, Espen Jakobsen og Jon Marius Aareskjold

Produsent: RakeI Nystabakk

Kostyme: Eva Hodnefjell

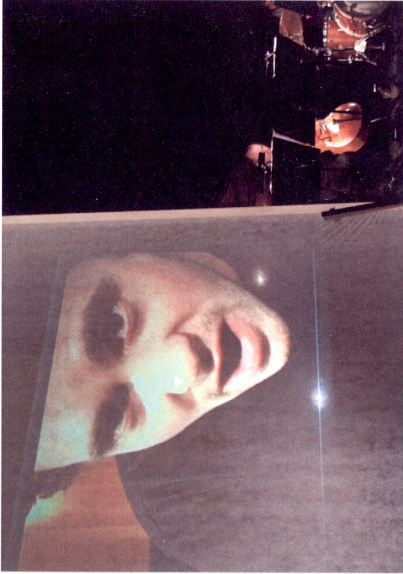
Blomster: Sonja Johnsen blomster AS

Vinduer: Bokstavhuset AS

Arrangør: Verdensoperaen i Nord

Verdensoperaen i Nord presenterer

La Serva Padrona



- en barokkopera i to akter omsatt til et high tech psykodrama i det 21nd århundre med musikk av G. Pergolesi

Tromsø Kulturhus, Verkstedet og Lillescenen
torsdag 08.12 og fredag 09.12

The World Opera

Figure 2.9: Flyer of the play *La Serva Padrona*, an Italian comedy from 1733, authored by Giovanni Battista Pergolesi.

LA SERVA PADRONA : EN OPERA I TO AKTER

"La Serva Padrona" er en komisk opera fra 1733 av den italienske komponisten Giovanni Battista Pergolesi (1710-36). Den er opprinnelig et såkalt *intermezzo*, et pause-innslag mellom akter i en større opera, men er blitt en klassiker i seg selv, og er fortsatt populær med sitt evigaktuelle tema om kærlighet mellom den eldre herren og den unge smarte husholdersken, denne gang i det 21. århundres high tech-samfunn.

Location: 77. etasje på Manhattan (Verkstedet) og i Villaen i Napoli (Lillescene)

Synopsis

I. akt

Uberto, en eldre søkrik italiensk forretningsmann, på forretningsreise i New York, vant til å få alt hva han ber om, er sint og utålmodig på sitt trofé, venninne og tjenerinne Serpina, hjemme i Napoli, fordi hun ikke har brakt ham sjokolade med privatfly i dag. Serpina er blitt så arrogant, at hun tror hun er den som bestemmer. Faktisk, når Uberto kaller for å få stakk og frakk, forbyr Serpina ham å forlate leiligheten, og legger til at fra da av må han adlyde hennes ordre uansett hvor i verden han er. Uberto ber i desperasjon og raseri Vespone om å finne ham en kvinne å gifte seg med, slik at han kan kvitte seg med Serpina.

II. akt

Serpina overbeviser Vespone å lure Uberto til å gifte seg med henne. Hun opplyser Uberto om at hun skal gifte seg med en general ved navn Tempesta. Hun vil forlate sitt hjem i Napoli og ber om unnskyldning for oppførselen sin. Vespone, forkledd som Tempesta, ankommer, og krever 4000 kroner i medgift. Uberto nekter å betale en slik sum. Tempesta truer ham til enten betale å medgift eller gifte seg med jenta selv. Uberto samtykker i å gifte seg med Serpina. Serpina og Vespone avslører sine tricks, men Uberto innser at han har elsket jenta hele tiden. De gifter seg og Serpina får det som hun vil og blir den samme hersker over husholdningen.

Velkommen

*Velkommen til Mixed Reality-verden!
Velkommen til Verdensoperaen i Nord, en operascene i Tromsø for og om vår tids Mixed Reality-tilværelse!
Velkommen til The World Opera, et verdensomspennende kunstnerisk laboratorium!
Velkommen til et opera-eksperiment!*

Niels Windfeld Lund
daglig leder, Verdensoperaen i Nord

Takk til:
Sonja Johnsen blomster AS, Alexander Eichhorn, Carsten Griwodz,
Ken Olsen, IT-klientdrift, Helse-fak – UJT,
Det kunstfaglige fakultet – UJT, Kulturskolen, Kulturhuset,
familie og venner

Sponsorer:
NTR0-fonden, Norges forskningsråd, Sparebank 1 Nord-Norge,
Universitetet i Tromsø










Figure 2.10: Flyer of the play *La Serva Padrona*, an Italian comedy from 1733, authored by Giovanni Battista Pergolesi.

/ 3

User Context State Detection

This chapter describes the *User Context State Detection* system for analysis and sharing, from the idea behind it to the prototype.

3.1 Idea

The idea behind the User Context State Detection (UCSD) system is to be able to detect the state of the remote stage, and share it with another stage. The shared state can be used to enable interaction between stages or to create a distributed stage composed of remote stages. Taking, for example, the position of actors on stage as a shared state, the actors could use this information to interact with each other, and to refer to their relative position as if they were on the same stage. The shared state, or part of it, could also be of interest to other parties not on a stage, such as a remote audience. The audience, however, might be interested not in the whole performance, but just a smaller part of it. Or maybe the interest is in a particular actor and who surrounds them, so the audience would like to follow that particular actor.

In order to reproduce a consistent performance, or part of it, by sharing the state of the stages, it is necessary to preserve the space and time relationships of the

single actors on every stage. For this reason, the focus is on the state of the actors on the stage. Such state can be generalized in at least position and volume occupied at some time, but more data on the actors is of course welcome. That is why the UCSD system filters the state of the whole stage in single streams of state for each actor, embedding spatial and temporal information about each of them. If we assume to have (reasonably) synchronized clocks at the stages, the relationship between actors interacting on the same stage is preserved in the streams and can be replicated or represented at a remote location. Individual streams can be treated separately, giving flexibility to the Remote Presence system – the system in charge of reproducing the user context state at a remote location.

If the previous assumption of a shared clock among the different locations holds, the space and time relationship between any of the streams holds, and the multi-stage performance can be reproduced anywhere. We are not considering here the unavoidable delays of transmitting the state streams to other locations; that is a problem for another system to solve, [3]. The UCSD system is meant to provide the state to share in order to obtain a performance spanning more than one stage.

3.2 Architecture

As summarized in Fig.3.1, the system architecture comprises different components: Local State Monitoring (LSM) records what the sensors/cameras see and passes the raw data to the Local State Analysis (LSA) for further analysis. LSA performs on-the-fly analysis of the data from the LSM and extracts interesting objects and events from the stream. For example, it separates the actors from the background and detects a gesture performed by an actor. In more detail, the LSM interfaces with the sensor to obtain the raw data. The raw data includes a timestamp of acquisition and the spatial configuration of the sensor, most commonly the position on the stage and, in the case of directional sensors (e.g., cameras), also orientation. The data is then passed to the LSA to perform the analysis needed to extract the actors from the background, detect gestures, or other potentially needed analysis. To carry out the analysis, the LSA uses not only the information contained in the raw data, but also the information attached by the LSM, i.e., timestamp and spatial configuration of the sensor.

To clarify, consider the following example: the detection of a gesture by tracking a point in space, which belongs to one of the actors. The LSM delivers to the LSA frames containing raw data timestamp and spatial configuration of the sensor. At this point, the LSA extracts the actor from the background and calculates

the position of the point being tracked. A cache of previous position and time of acquisition of the tracked point can be used to calculate the direction of motion and the speed. These factors, direction of motion and speed, can be used to detect a gesture.

As these two parts (LSM and LSA) belongs to the local side of the architecture (Fig.3.1) they are tasked to deal with a single stage. To make the information on one stage available to other stages the LSA encodes the results of the analysis in streams and delivers them to the global side. Each packet of the stream of data contains the timestamp and spatial configuration of the sensor that generated the stream.

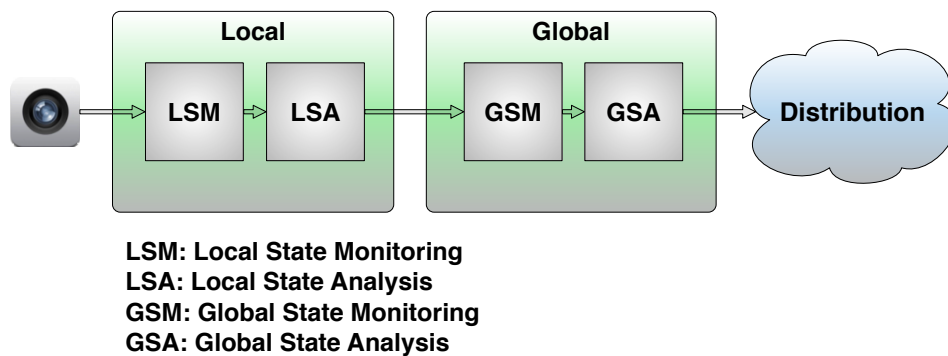


Figure 3.1: Architecture of the User Context State Detection. It analyzes the raw data and streams it to the global side. Here the state of streams is analyzed in a global context and global events are extracted and transmitted in new streams.

On the global side we have Global State Monitoring (GSM) and Global State Analysis (GSA). Similar to their local counterparts they collect the streams (GSM) and analyze them (GSA). They are called global state monitoring and analysis because contrary to the local ones they deal with the streams from all the stages. Having available all the data from the stages, the analysis happens in a global context – all the streams are taken into consideration and can be analyzed to extract global information on the whole performance or to detect global events and gestures. A global gesture is a gesture that can be seen as a set of global states to be reached in sequence, or a single global state to be reached. An example of a global gesture can be as simple as all the actors standing in a predefined position. A slightly more complex one for both detection and the actors could be actors standing in a predefined position *relative* to each other. An even more complex global gesture can be to follow the choreography of a dance.

This kind of event can be used for instance to signal the beginning or the end of a performance, or to automate the lighting effects on the stages, and can only be obtained by funneling all the streams to the GSM/GSA.

This implies that the GSA can create new streams on-the-fly and deliver them to the Distribution system, as well as all the streams received by the local side. The Distribution system works according to a publish–subscribe protocol and each stage can subscribe to any stream, obtaining the whole performance or just a part according to their capabilities. This also allows the stages to receive, for example, the global gesture streams without needing all the streams, reducing the bandwidth needed to receive them and the processing power needed to process them.

To conclude, the flexibility of this solution allows any stage to replicate any subset of the performance comprising any number of actors, from one actor to all the actors. The space and time awareness of the data streams allows the receiving stages to maintain space–time relationships among related streams. For example, a group of two or three people talking or interacting can be virtually repositioned anywhere by the Remote Presence system, without losing the credibility of the interaction, because the relative distances, in space and time, of virtual representations are preserved.

3.3 Design

If we visualize the MultiStage system as a pipeline where data flows from a begin point to an end point, the UCSD system is the first stage of the pipeline. The first stage of the pipeline is where the actors are detected during a show. The data obtained on each actor is encoded into streams and the streams are sent to the distribution. To keep track of the different streams, the UCSD system annotates with a header all the packets that compose the streams. The header contains a stream identifier (ID), a timestamp of the data acquisition, and a sequence number. The ID in the stream header needs to be unique to each stream and must contain information also on its provenance, the stage where it has been generated. This allows the consumer of the data at any stage of the pipeline to know the origin of the stream.

Fig. 3.2 illustrates this concept. The streams of data generated by the UCSD system flow in one direction through the pipeline, making the UCSD system the first pipeline stage where data is produced. Data is delivered to the Distribution system, which accepts subscriptions for data streams from other systems.

Each packet composing the streams is produced by the LSM/LSA from a single

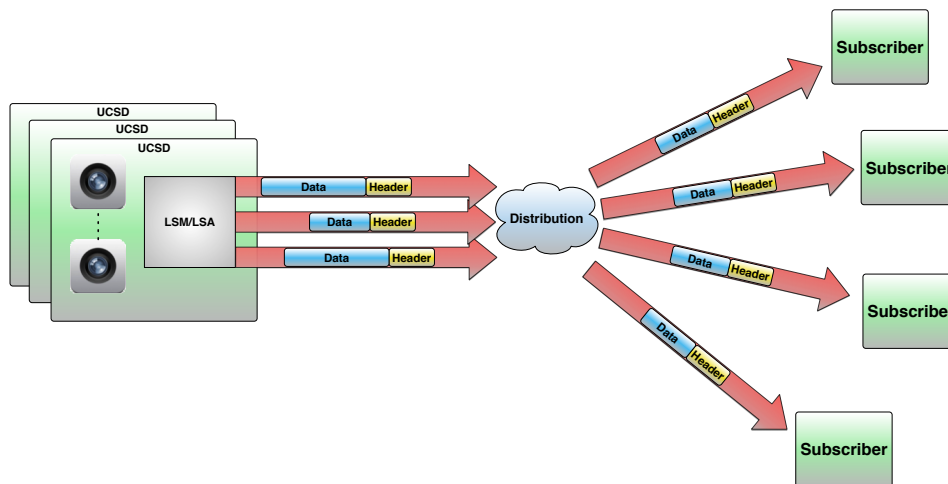


Figure 3.2: Design of the User Context State Detection. The stream packets are annotated with a header that unequivocally identifies the stream. The distribution system uses the stream headers to publish and route the streams.

frame of the sensors. Consecutive packets are independent of each other – there is no encoding of a stream as a whole. This decision trades bandwidth for flexibility – the system does not save bandwidth by making each frame independent from the previous one (e.g., encoding each packet as being function of the previous one). But each packet is self-contained and can be used alone without knowledge of the preceding packets. Another advantage of having the data in each packet independent of each other is better tolerance at packet loss or network delays. When a packet is lost or discarded because it is delivered out of order there is no penalty for the stream consistency.

The system uses User Datagram Protocol (UDP) for network communication. The reasons behind this choice are stateless communication and lack of retransmission. Stateless communication is faster and easier to set up and maintain, with no need to keep track and manage the connection state. Retransmission can be desirable in other scenarios, but in real-time communications the delay of the retransmission can interfere with the performance.

3.4 Implementation

To evaluate the architecture and the design by running experiments, a proof of concept prototype has been built. The prototype consists of both software and hardware. To be more specific, the software has been written almost from scratch and the hardware assembled from *off-the-shelf* components.

3.4.1 Sensor Suite

The sensors of choice for the prototype are 3D cameras, specifically MS Kinects that are cheap, off-the-shelf hardware. 3D cameras are now commodity hardware, and combined with some assumptions their use can save much computational power otherwise dedicated to complex computer vision algorithms. A simplifying assumption we make is that each user is detected by one camera, and the data from each camera is encoded in one stream. Each stream (and user) is bound to exactly one camera and can be identified with it. When the cameras are properly configured and their position and orientation on stage is known, we can extract the volumetric information of the user in front of it.

The combination of the sensors and computers is a self-contained transportable device called the Sensor Suite. The Sensor Suite is a combination of four Kinects, three Mac minis, and one Apple AirPort Extreme (see Fig. 3.3). One of the Mac minis runs the remote presence software (see Chapter 5) to let the actors interact with the other stages. The other two Mac minis control two Kinect cameras each.

Even if the Mac minis have four USB ports, their USB 2.0 bus cannot support more than two Kinects. A different hardware solution might be possible, such as a PC with a USB board to support all the Kinects, but the Mac minis keep the Sensor Suite compact and portable. The resulting system is even more distributed in this way, and it also provides a small degree of redundancy: if one of the Mac minis malfunctions, the other is not influenced and there is still 50% of the monitoring in place. A full working Sensor Suite delivers a close to 360-degree view and can monitor a room or a stage.

The last component of the Sensor Suite is an Apple AirPort Extreme for connectivity, both wired and wireless; more specifically, 1 Gbit/s wired and Wi-Fi 802.11a/b/g/n, even if the wireless connectivity was never used during the experiments. Fig. 3.3 shows the cabled connection of the Sensor Suite. It should be noted that one last Ethernet port is used to connect the Sensor Suite to a local area network (LAN) / wide area network (WAN) for inter-stage communication.

3.4.2 Software

The UCSD system is structured as a pipeline (see Fig. 3.4), with each stage implementing a functionality.

The first stage combines LSA/LSM, gathers the data from the cameras and

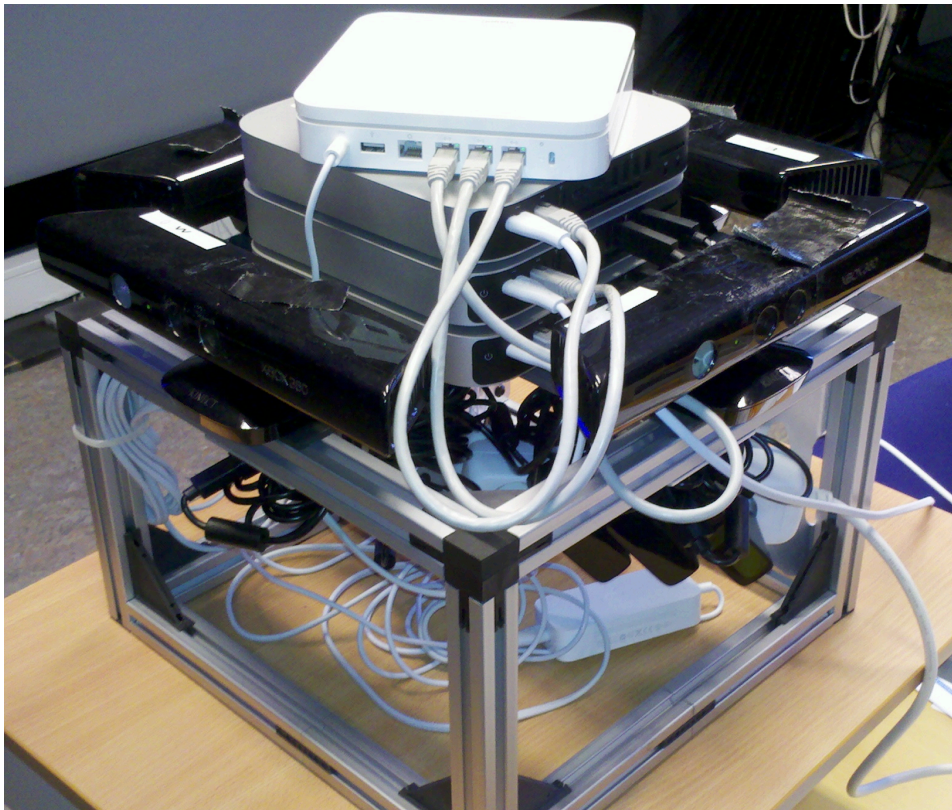


Figure 3.3: The Sensor Suite is composed by four Kinects connected to two Mac minis. An Airport Extreme provides network connection.

performs analysis and monitoring. The output of this stage is a colored point cloud with attached a header comprising a timestamp, a sequence number and a stream ID. The next stage marshals the point cloud and the header in a binary format and then compress it with the Snappy algorithm [61]. The result is packaged into a UDP datagram and sent to the distribution.

To detect the actors, the system uses few assumptions: (i) one user per camera, and (ii) the user occupies a known volume in front of the camera. The first assumption means that the UCSD system detects only one user for each camera. This is not a technological limit, Microsoft and PrimeSense Kinect proprietary drivers can detect more than one user on a single camera. But the proprietary drivers use around 50% of the CPU available on a Mac mini with one Kinect. That would not leave much room for other computations given that the UCSD system uses two Kinects on each Mac mini; for this reason the UCSD system uses the libfreenect [62]. Libfreenect is an open source driver that allows to use the basic functionalities of the Kinect, such as receiving the data acquired with the cameras.

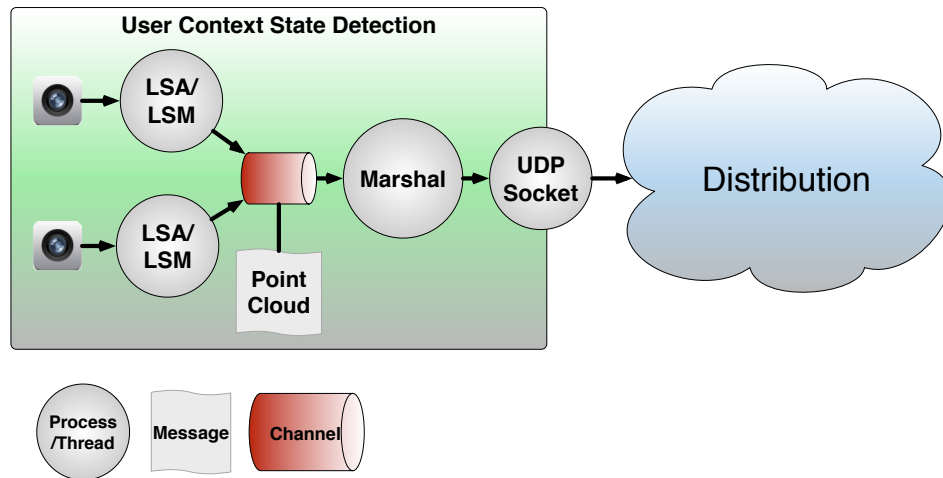


Figure 3.4: Software implementation of the User Context State Detection. The system is structured as a pipeline, the first stage detects the users and creates a point cloud, the second stage marshals it and sends it to the distribution via UDP.

The second assumption is that anything in front of the camera and in the detection volume is considered a user. This again simplifies the implementation allowing a single scan of the depth image to determine whether a user is being detected. To be more specific, a depth image is an image whose pixels represent a depth or a distance from the camera. Having obtained a depth image from a Kinect, the UCSD system proceeds to scan it pixel by pixel and checks whether there are pixels of value inside a predefined range. The pixels whose value is inside the range are kept as part of the detected user, and the ones with pixel value outside the range are discarded as background.

With the same scan it is possible to determine the bounding box of the user: while parsing the depth image, the UCSD system keeps track of the farthest and closest point from the camera, i.e., the pair of points having the maximum and the minimum Z value. The same happens for the other two axes: X leftmost and rightmost, Y highest and lowest. This process identifies six points. The combination of the coordinates of these points is enough to describe a bounding box of the user.

In practice, only two points are necessary to determine a bounding box, the maximum and minimum points – the maximum contains the maximum of all the XYZ coordinates, and the minimum contains the respective minimum. Using only two points, however, limits the flexibility as shown in 4. Therefore the system keeps track of all six points and sends them along the point cloud for later use; for example, graphically debugging by printing the bounding box.

The system generates the colored point cloud by *transforming* the pixels of the depth image in 3D points and matching them with the pixels from the RGB image, assigning a color to each point. The 3D transformation is a projection of the points from camera coordinates to world coordinates; the parameters of the projection matrix are the result of a factory calibration and are stored in the Kinect firmware. The libfreenect driver can access the firmware making the parameters available and in addition provides utility functions to convert depth pixels into to 3D points. The same driver provides RGB and depth image matching pixel by pixel, so to assign a color to a point is necessary to match the pixels with the same index in the two images.

The data so obtained is a colored point cloud of the users and the bounding box containing them. The point cloud is compressed with the Snappy algorithm [61], chosen for its speed not for the compression rate. The compressed data is given a header containing the information to identify the provenance of the data, and it is then serialized and sent to the Distribution system.

The number of points can be too big to fit into a single UDP packet, even when compressed. For this reason a command line argument can be used to specify the number of points, the system down samples the point cloud to the amount requested. This is also useful to save bandwidth. 5000 points compressed with Snappy is the maximum number of points that fits an UDP datagram.

The UCSD system is written in the Go programming language, which was chosen for its simplicity, cleanliness, and above all for its Communicating sequential processes (CSP) [63] inspired concurrency primitives: *channels* and *goroutines* (green threads).

This is visible in Fig. 3.4 – LSM/LSA is implemented as a goroutine calling the libfreenect driver written in C to obtain the images. After the analysis, implemented in the Go code, the LSM/LSA goroutine sends the point cloud and bounding box through a channel. At the receiving end of the channel there is another goroutine which tasks it to marshal the data in a string of bytes, compress it, and send it to the distribution via UDP.

3.5 Experiments

To evaluate the performance of the UCSD system a set of experiments was conducted. The design and setup of the experiments is explained in the following section.

3.5.1 Design and Configuration

The experiments collected three metrics: network traffic, CPU, and memory utilization. The factors were the size of the point cloud and number of stages. The point cloud size is not arbitrary, but the result of our own use of the system and size of UDP datagram. 5000 points is the maximum amount that can fit an up datagram after compression. Using more than one datagram per frame was abandoned during the design to keep the system simple. 1000 points is the least number of points that, in our opinion, makes the interaction still plausible. Below 1000 points it is difficult to distinguish the features of the actors, a fundamental aspect for the interaction.

Two stages were configured for this experiment, both stages were equipped with a Sensor Suite to run the UCSD system. Each Sensor Suite comprises two Mac minis (mid-2011) with an Intel Core i7 at 2.7 GHz and 8 GB of RAM. Two Kinect cameras were connected to each Mac mini, and all the computers were connected to a gigabit Ethernet switch. To collect the data, the UCSD system was left running for little over five minutes. In front of each camera was placed a static object the size of an average human being.

The measurements (CPU memory and inbound traffic) have been collected with the monitoring system discussed in Section 2.5. The monitoring system measures resource consumption of the whole machines running the UCSD system, not at process level. This does not allow for a fine-grained collection of data on each system, but grants an overview on the general behaviour of MultiStage. This overview allows us to understand the trend in the resource utilization and to possibly take action to avoid saturation. In other words it grants the knowledge to quantify the amount of resources needed for a specific MultiStage installation.

3.5.2 Results

As we can see in Fig. 3.5, the resources of the system are not constrained in terms of CPU and memory. Memory usage varies from 13% to 24% among the computers. This variation is attributed to *background noise*, other processes running in the background. Even is the memory utilization varies among the computers it is consistent across point cloud sizes, for all the computers increasing the point cloud size does not produce visible variation in the memory usage.

The CPU utilization grows from 1000 point to 5000 consistently on each computer, as does the standard deviation indicated in the figure by the error bars. However, the CPU utilization is below 5% for any amount of points

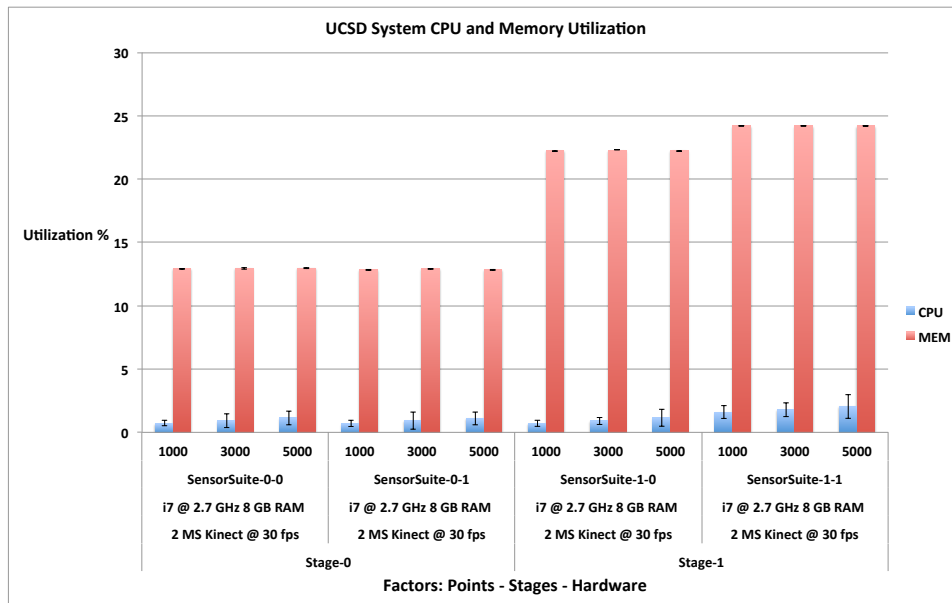


Figure 3.5: Memory and CPU usage of the UCSD compared among different point cloud sizes. The error bars indicate the standard deviation.

used in the experiments and for each computer. The system is not resource restraint.

Fig. 3.6 shows the bandwidth used. The bandwidth used for 5000 points is close to 3.5 MByte/s. According to the Netflix Internet Connection Speed Recommendation [64] the bandwidth recommended for *Ultra HD* quality is 25 Mbit/s. That translates in 3.125 MByte/s, comparable to the bandwidth used by one of the computers of a Sensor Suite to stream two streams at 5000 points.

3.6 Discussion

In the previous section we compared the data-rate of a Sensor Suite to an *Ultra HD* quality movie from Netflix (3125 KByte/s), even if each frame derives from two Video Graphics Array (VGA) images (640X480 pixel). In order to have some perspective we can calculate the bit-rate for the raw data and for the point cloud. The images from a Kinect are a depth image with 16 bits per pixels and an RGB image with 24 bits per pixel. At 30 frames per second the amount of data to move is: $(640 * 480 * (24 + 16)) * 30 = 368.64 \text{ Mbit/s}$ or 46.08 MByte/s of raw data from one camera.

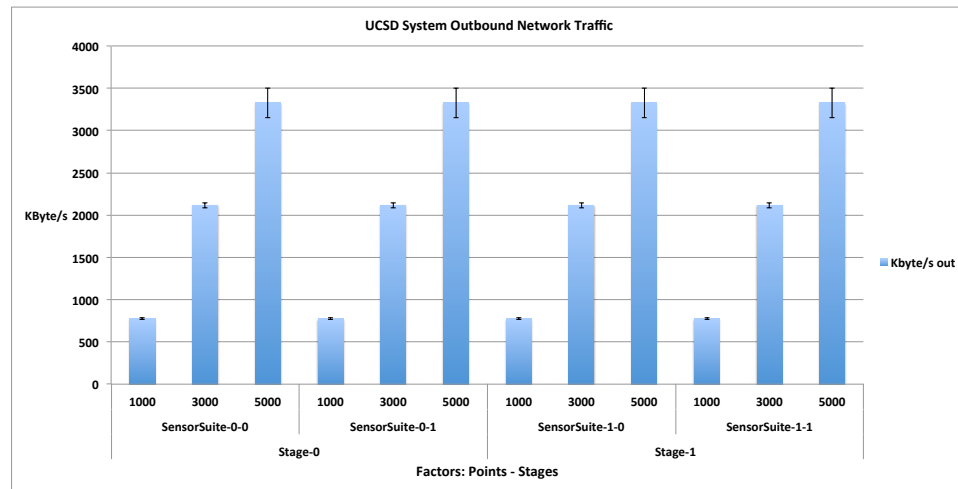


Figure 3.6: Network traffic generated by the UCSD for different point cloud sizes. The error bars indicate the standard deviation.

The UCSD extracts from each frame, both RGB and depth, up to 5000 pixels and converts them into colored 3D points. Each 3D point is represented by three floating-point numbers and a color: $32 * 3 + 24 = 120$ bits. At 30 frames per second the amount of data to move is: $120 * 5000 * 30 = 18$ Mbit/s or 2.25 MByte/s of uncompressed data from one camera.

The UCSD compresses each frame individually with the Snappy algorithm [61]. The result of the compression, visible in Fig. 3.6, is close to 3.5 MByte/s for data from two cameras.

To reduce the amount of bandwidth used the streams could be encoded like a video, making each frame (or consecutive acquisition of point clouds from a camera) a function of its predecessor. This would decrease the bandwidth but limit the flexibility of the system, modifying a frame on the fly would disrupt the successive frames.

Keeping the frames composing a stream independent allows other systems between begin point and end point to interleave, modify and manipulate the data without needing to take care of encoding algorithms. As we will see in the following chapters, other systems have the goal of manipulating the information/data transiting from the begin point to the end point, such as the Remote Presence systems presented in Chapter 5 or systems tasked to *mask the effects of delays* as presented in Su et al. [3].

3.7 Lessons Learned

The Sensor Suite is, to some extent, a distributed system, with the advantage and disadvantage of other distributed systems. It is more complicated to manage, but it provides some degree of fail tolerance: if one of the computers fails the Sensor Suite is still operational at 50%.

Using UDP as the transport protocol simplifies the management of network connections – there is no need to handle the connections to the UCSD system, just *fire and forget*. This leads to a simpler recovery from failures – if a computer comprising the Sensor Suite fails, it can be substituted without renegotiating the connection on the receiving end.

A few thousand points (up to 5000 in our experiments) are enough to represent a user in a way that allows interactions. This limit was set by the size of the UDP datagram; a better compression algorithm, or more compact representation of the data can probably increase this limit. The combination of the information is enough to reproduce the performance of the users in other locations, and allows other remote users to interact with the remote presence.

3.8 Summary

In this chapter the UCSD system has been presented.

The UCSD is a system capable of detecting actors/users on a stage, acquiring their volumetric information and a colored point cloud describing them. It also marshals the point cloud attaching a header and sends it to the Distribution system. The UCSD system is the *begin point* of the MultiStage system, where the streams of data are generated for the other systems.

This chapter also presented the detail of the architecture, design, and implementation of the UCSD system, as well as the evaluation of the experiments and discussion of the results.

/4

Gestures

In this chapter we present a system to detect gestures based on regular expressions. In the MultiStage context, gestures can be used in many ways to gather inputs from the user on stage under the assumption that user input is needed. The user input can be used for the performance itself and be visible to the audience, or can be functional to the development of the show or used for offstage communication.

4.1 Need for actor input

The concept of gesture detection involves the capturing of a subset of movements of the user and giving them a meaning. In other words, we make the system aware of the user and make the user able to interact with the system with their movements.

A way to obtain this is to define a language that both system and human can use, and translate the human motions into that language. If the system can translate user motions into strings of text and these strings can be matched by a regular expression, the system can map user motions to gestures. Different regular expressions can represent different gestures, effectively broadening the gesture dictionary available to the user.

The same principle can be applied to collective gestures from different locations.

For example, in a multistage scenario a collective gesture can be detected when all the users are standing in a defined relative position to each other, or if all the users from all the locations perform a gesture in a short time frame, this can be compared time-wise to a distributed mouse *double click*, where the clicks must happen in a defined time frame and in a defined area. These collective gestures can be used to trigger inter-stage events to enrich the performance. In all these scenarios it is possible to produce a string of text, or phrases of a language, and match them against a regular expression. The only difference in this global scenario is the location where it happens; a global interpreter needed to interpret the global gestures.

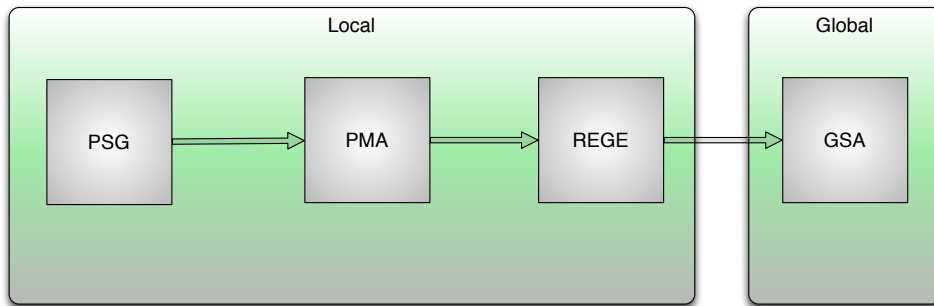
The idea of using languages to identify gestures is not novel as we can see in [65]. What we propose is a system able to recognize gestures in a 3D space, and collective gestures with contributions from different remote locations.

4.2 Architecture

Fig. 4.1 shows the architecture of the Gesture Detection system. The architecture distinguishes a global side and a local side. To detect a global gesture there is a need to analyze the input from different stages, and there is thus a need for a global side for the system. The local side is meant to detect gestures performed by the actors on the local stage, and the gestures are delivered to a Global State Analysis. The GSA performs analysis on received gestures and creates new ones if gestures detected on the stages compose a global gesture. In practice, the GSA generates streams of new discovered events to be delivered to the interested peers. (The same distribution system used in the previous chapters is a good candidate for this task).

We follow the idea previously described by having a stream of points in space as the source of the gesture; the Point Stream Generator (PSG) is the component responsible for that. The PSG is also responsible for tracking the time of detection for each point it generates, so a timestamp accompanies each point. The functionality provided by the Point Motion Analysis (PMA) is the base on which the gesture detection is built. It compares different points at different times and translates them into strings of text to be parsed by the Regular Expression Gesture Engine (REGE). The REGE tries to match each string of characters received against a predefined set of regular expressions; each regular expression is defined to detect a gesture.

If a regular expression matches the text from the PMA, the corresponding gesture is detected. The final output of the local side is a stream of gestures, annotated with position on the stage where they have been performed and



PSG: Point Stream Generator
 PMA: Point Motion Analyser
 REGE: Regular Expression Gesture Engine
 GSA: Global State Analyser

Figure 4.1: Architecture of the gesture detection system.

the time of their execution by the actor. The global side aggregates the various gesture streams from the stages, collecting and analyzing their global state. If the state of the stage, as expressed by the gestures and received by the GSA, determines a collective gesture, the GSA delivers a new event to the Distribution system. The Distribution system is in charge of delivering these events to its subscribers.

4.3 Design

Fig. 4.2 shows the design of the Gesture Detection system. Assuming a technology to obtain a 3D scan of the user, such as a 3D camera, and assuming that the user is facing the camera, we can detect and track six well-defined points of the user: the closest point to the camera, the farthest point still visible by the camera, the topmost and the bottommost point of the user, and the leftmost and rightmost point of the user. This functionality is implemented by the UCSD system, but the design is not tightly coupled to it – any stream of 3D points can be used so other devices can be considered for the purpose.

Given that a plane is defined by a point and a normal, we can combine these six points in combination with the reference axes of the camera coordinates to obtain an axis-aligned bounding box enveloping the user (see Fig. 4.3a). Assuming also that we are tracking the points used to generate the bounding box, we are in effect surrounding the user with six virtual touch screens always in contact with the user (Fig. 4.3b).

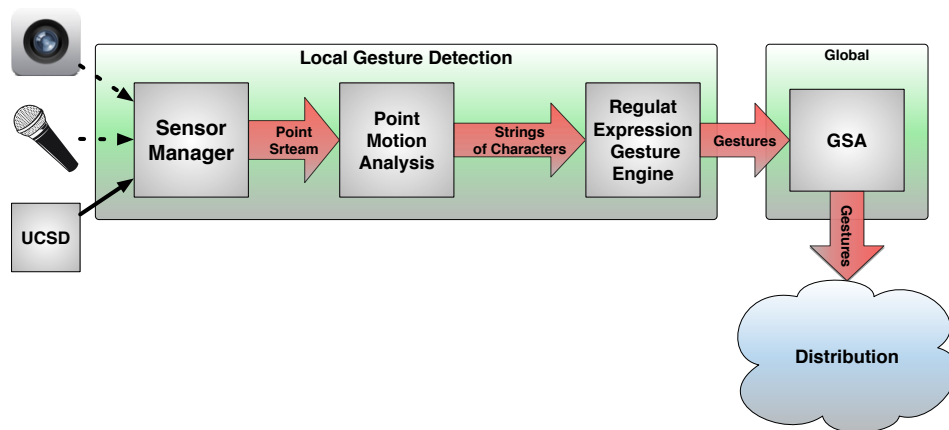
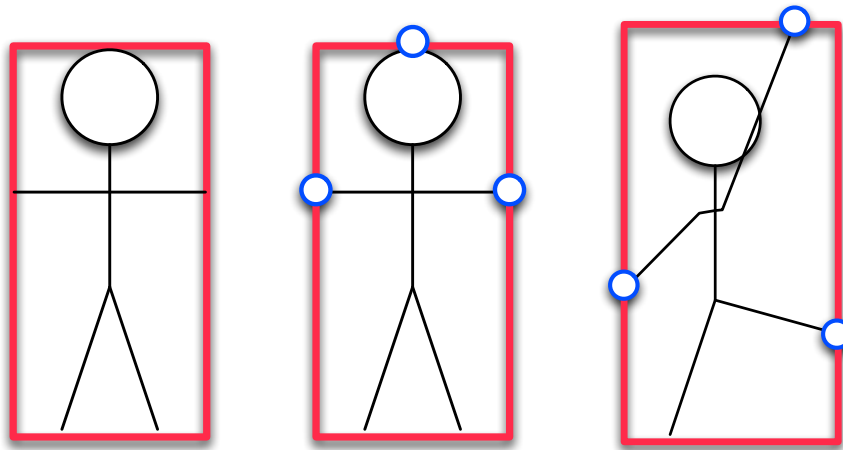


Figure 4.2: Design of the gesture detection system. The design assumes that the UCSD is the source of points to be analyzed, but other sensors producing a stream of point can be used. The PMA performs a motion analysis on the points and outputs strings of characters representing the motion. The Regular Expression Gesture Engine detects gestures by matching the strings produced by the PMA against the regular expressions. The Global State Analysis receives the gestures and perform a global analysis to detect global gestures.

The movements of these points are then used to produce character sequences and are matched with regular expressions to detect gestures. No single body part is tracked, such as hands or head; the user is free to use any part of his body to perform the gestures (Fig. 4.3c).

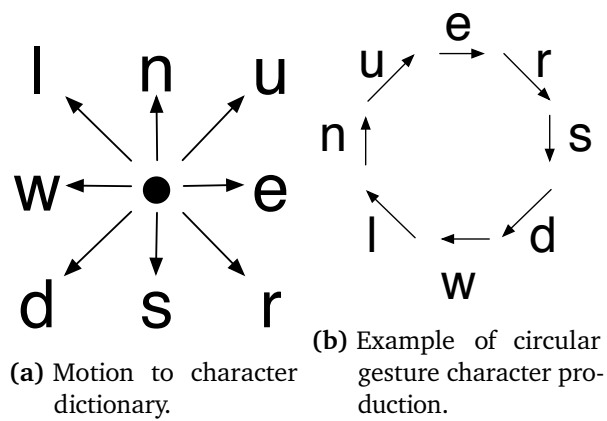
To illustrate the generation of the string of characters, consider Fig. 4.4. First of all we define which set of user movements we want to be translated into characters. We call this set a motion dictionary, Fig. 4.4a illustrates the motion dictionary. The arrows represent the movements the system will look for, and the character each movement will produce.

For simplicity and efficiency we considered only motions along the major axes and the diagonal between them. For example, an up-arrow implies an arm or body part moving straight up and being interpreted as a character, in this case *n*. Fig. 4.4b illustrates a circular motion and the characters produced by doing it. A full circular movement of an arm/hand produces the characters *nuersdwnl*. The system takes into account character repetition and speed of the user movement so that characters are produced only in a predefined speed range of the tracking points.



(a) Bounding box surrounding the user. (b) Bounding box with points used to generate it in evidence. (c) Possible use of a bounding box.

Figure 4.3: User Bounding Box.



(a) Motion to character dictionary.

(b) Example of circular gesture character production.

Figure 4.4: The motion dictionary and an example of circular gesture.

4.4 Implementation

The output from the UCSD system is already space and time annotated and is suitable for the task of detecting user gestures. The implementation embeds the UCSD and in particular uses the bounding box surrounding the user (see Chapter 3), and other components are introduced to fulfill the functionalities needed. In the prototype, the points on the planes behind and below the feet of the user are harder to track, but the four remaining planes have been proven enough for a minimal gesture set.

4.4.1 Point Motion Analysis

The PMA works as follows. The system holds two bounding boxes, *old* and *new*, defined by six points each: the system starts with zeroed bounding boxes. The UCSD system (in this case the LSM/LSA parts of the UCSD system, see Chapter 3) sends a new bounding box from the last detection. The PMA subtracts each of the *new* points from the corresponding *old* points – the *new* top point from the *old* top point, and so on. The result of the subtraction is a vector pointing in the direction of the motion. The vector is projected on the plane corresponding to the point; for example, the top point is projected on the XZ plane, and the closest point to the camera on the XY plane (see Fig. 4.6 for a visual example).

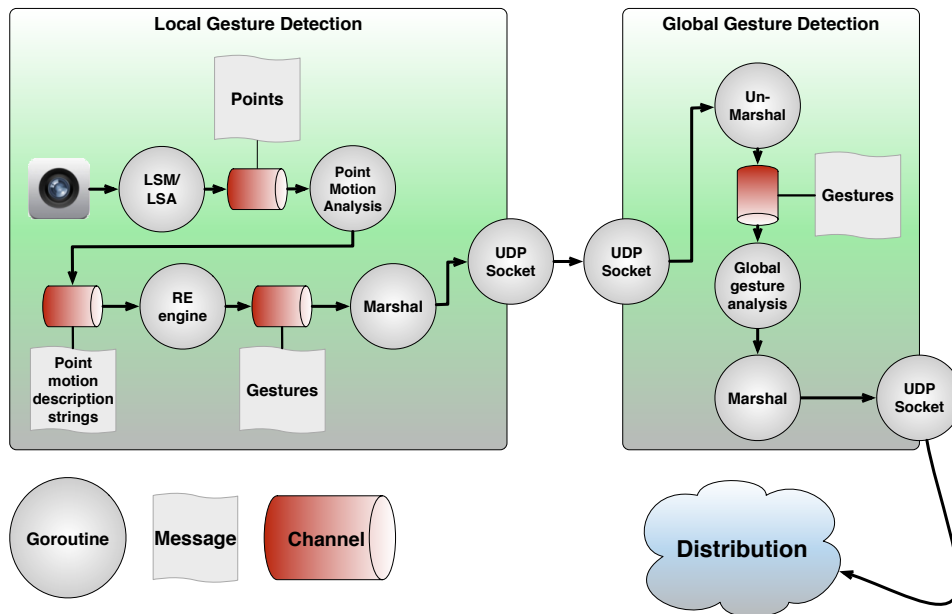
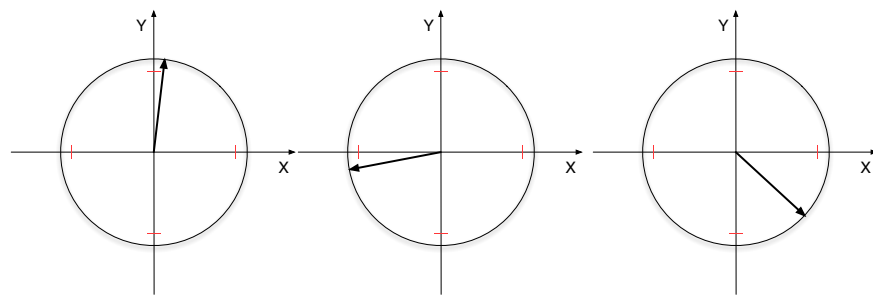


Figure 4.5: Implementation of the gesture detection system. The source of data is the LSM/LSA if the UCSD. The system process the points and analyzes their motions. The result of the analysis is a stream of motion strings that are matched by the regular expression engine in order to detect gestures. The system marshals the local gestures and sends them over the network to the global gesture analyzer. The system is divided in independent functionalities, each of them running in its own goroutine.

The projected vector is normalized and its component analyzed. If one of the elements of the vector is close to one, it means the vector is close to parallel to a main axis of the plane, and the sign of the elements is the discriminant. If none of the elements are close to one, the vector is considered a diagonal, and once again the sign of the elements is the discriminant. For each of the different directions detected, the PMA emits a character in the input string for the Regular Expression Engine (see Fig. 4.4 for a visual example).



(a) The Y element of the vector is above the threshold. The vector produces a vertical motion.
 (b) The X element of the vector is below the threshold. The vector produces an horizontal motion.
 (c) The elements of the vectors are inside the thresholds. The vector produces a diagonal motion.

Figure 4.6: Discrimination of motions using normalized vectors. If one of the elements of the vectors, in this case X and Y, is above a predefined threshold, marked in red on the axes, the vector is considered vertical or horizontal. The sign of the elements decides the direction.

4.4.2 Regular Expression Engine

The Regular Expression Engine is a set of regular expressions against which the motion strings are matched. The prototype limits the motion strings to 100 characters; once the input string reaches this limit it is truncated to a zero length. To alleviate this problem, the system uses two parameters.

(i) The length of the input. Given that the points are extracted from the cameras, the amount of points per second is the same as the frame rate of the camera capturing them, which in this prototype is 30 fps. Assuming a maximum gesture execution time close to three seconds with a maximum string length of 100 characters, we cover the execution time of the gestures if we start the gesture when the input string is of zero length.

(ii) The speed range of the detected points. To maximize the chance that the input string is at zero length when the user starts performing a gesture, the system tracks the speed of the detected points. If a point moves too fast or too slow, the input string is truncated. This filters out glitches of the Detection system, and noise from the cameras.

For example a point can *flicker* between two different positions in consecutive frames, resulting in speed not compatible with a room setting. Also, the user needs an energetic movement to trigger a gesture detection; just standing in

front of the camera is not enough as the system will truncate the input string until the user starts a fast enough motion. This helps the users to initiate a gesture when the input string has just been truncated – giving them three seconds to complete the gesture.

4.5 Experiments

During the use of the system we collected some statistics to verify the resources utilization and latency in detecting the gestures. The configuration was a Mac mini with an Intel i7 at 2.7 GHz and 8 GB of RAM, and two Kinects connected to the computer. The systems also uses a lighter version of the detection algorithm of the UCSD system that does not include the generation of the point clouds.

4.5.1 Latency

Tables 4.1 and 4.2 present the latency of the system in detecting a gesture. For the experiment the system was configured to detect two kind of gestures on four planes. The gestures were chosen based on their robustness to false positives so that users behaving normally in front of the camera would not trigger a gesture.

The first gesture is to describe a circle, both clockwise and counterclockwise. The second gesture is to describe a straight line, both vertical and horizontal. Each of the regular expressions that identify each gesture has a different minimum amount of characters, setting a minimum amount of time for the gesture to be performed/detected. The regular expression for the circular gesture is: $n^*u+e^*r+s^*d+w^*l+n^*$, with a minimum of 9 characters. While the straight line gesture one is: $n\{20,\}$, with a minimum of 20 characters. This difference in the character numbers accounts for the different results shown in the Tables 4.1 and 4.2. The number of characters reported is the actual number of characters in the buffer at the time of detection of the gesture.

	Seconds	Characters	Seconds / Characters
Average	0.957227185	29.71328671	0.0319942844
Median	0.863717786	27	0.032000686
Stdev	0.45646036	13.67923397	0.000711968

Table 4.1: Circle gesture latency.

	Seconds	Characters	Seconds / Characters
Average	1.355079928	41.59756098	0.032467229
Median	1.280198133	39	0.03249347
Stdev	0.47902494	14.34311445	0.000414628

Table 4.2: *Straight* gesture latency.

4.5.2 Resource Utilization

Fig. 4.7 shows the CPU and memory utilization. As we can see, the system is not resources bound – the most exploited resource is the CPU, around 50%, used to process the 3D depth images in order to generate the bounding boxes. The memory usage is quasi irrelevant for the system on which the prototype is deployed. Statistics were collected using the *ps* (the shell script is in Appendix B) during a section of gesture detection by one user. The user kept switching usage between the two cameras.

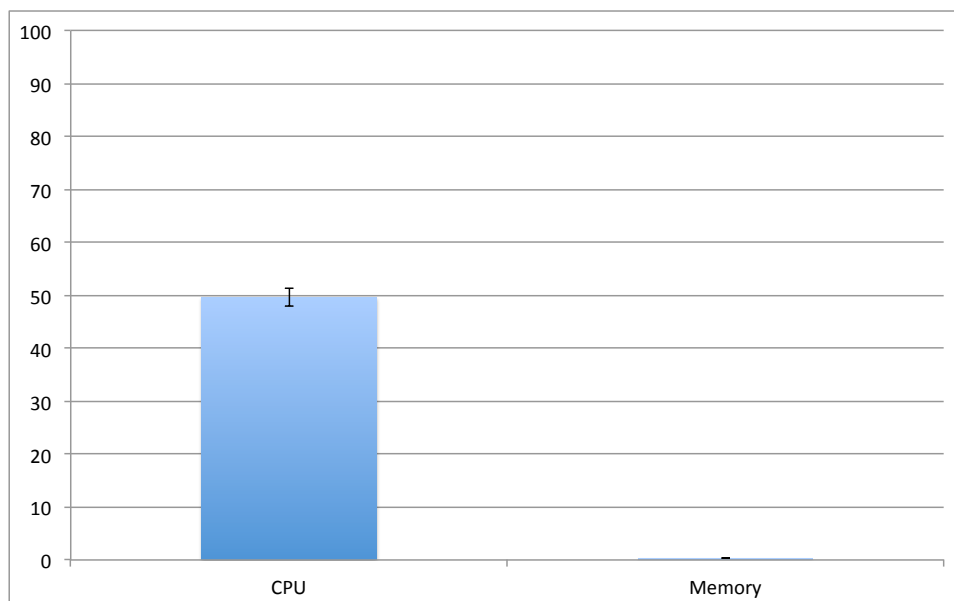


Figure 4.7: Percentage CPU and memory utilization of the gesture detection system with standard deviation.

4.6 Discussion

As we stated earlier, the Regular Expression Engine has an arbitrary limit of 100 characters for the input string. This limitation can cause false negatives if the limit is reached during a gesture, compared to a more complex implementation

such as a circular buffer.

The latency measurements suggest that the time needed by the system to detect a gesture is proportional to the number of characters in the buffer. This is an argument for a circular buffer sized on the maximum number of characters needed to detect a gesture. This number, however, depends on the regular expression chosen, and in some cases cannot be known a priori.

In any case, with the current hardware the rate of the updates to motion strings is 30 updates/s; this translates roughly as a 3 s limit on the gesture duration. We found this value to be sufficient for the gestures we used, and so the need for a more complex solution did not arise.

Another limitation of the current implementation of the system, is the projection of the gesture on predefined orthogonal planes. This implies that even if the planes are positioned in a 3D environment, the gestures exploit only two of the available dimensions. However, this limitation is an artifact of convenience. Assuming a rectangular room, the UCSD system can be oriented so that the walls are perpendicular and parallel to the viewing planes of the cameras. We believe that in this way we facilitate the users in visualizing the planes, and lower the failures in performing a gesture, perceived by the user as false negatives. This limitation can be easily lifted and lead to fully 3D gestures or to a different orientation of the planes for the user's convenience or for a different use.

4.7 Lessons Learned

During the early development of the Gesture Detection system we started by tracking the least amount of points needed to define an axis-aligned bounding box. This was just two points containing the minimums and maximums of the XYZ coordinates. However, the pair of points had scarce expressivity, and although they could be used to detect some user gestures, they were quite limited.

We later found out that we could track other points and that these points could be used for, and are better suited to, performing gestures. We chose these more expressive points lying on the six planes that compose the bounding box. In this way, the users can move the points with their body and limbs.

Due to technological and anatomical limitations, we restricted the number of points from six to four, as described in the previous chapters. The increase in the number of points, from two to six (four), allows a richer gesture vocabulary

as well as multi-limb gestures.

Combining this with the regular expression gave us another level of freedom in the configuration of the system: a string of text is, *potentially*, all that is needed to define a gesture.

4.8 Summary

In this chapter we presented a system to detect gestures based on a bounding box surrounding the users. The system tracks the motion of the user on the bounding box surface and translates it into strings of text. Then the system parses the text matching it against a set of regular expressions to detect user gestures from the user movements.

We also presented details of the implementation of the system. We followed with a discussion of the limits of the system as well as the strengths and weaknesses. We also experimentally measured the resources the system uses when functioning and its latency in detecting gestures.

/5

Remote Presence

Remote presence in the MultiStage context means to recreate the reality of one stage on another one, in a manner that allows interaction between the real stage and the remote one to happen and to be credible. Audio and video are the most commonly used mediums to convey an interaction-worthy presence of a remote user. In the work presented by Zimmermann et al. [12], the authors describe a detailed analysis of the problems of distributed interactive audio performance. The performance is based on music played by professionals, representing one of the most demanding scenarios for a remote interaction. Other forms of interactions do not have such a strict timing requirement and can be addressed with more pragmatic solutions. Here we present a video remote presence system based on a 3D point cloud. This approach does not include audio, the streams of point clouds are rendered to a video.

Even if the most common means of implementing a remote presence are audio and video, specifically on a display, there are other options potentially involving different senses.

- 3D models and computer graphics are well known to the entertainment industry and are the de facto standard for special effects in movies and in high end video games.
- Scratch-and-sniff stickers were popular in the late 1970s: automating the process of releasing smells from similar capsules can add smells to a remote presence.

- Haptic technologies, delivering tactile and force feedback, can be used to enhance an already established remote presence.
- Remote-controlled robots can deliver a remote presence that is not bound to the space limits where remote presences are usually pinned (usually on a display in a room).

All these technologies can be combined to deliver a more credible and complete remote presence, or they can be used to enhance, augment, and *amplify* a remote presence as explained later in this chapter.

5.1 Idea

The basic idea of a remote presence system is to reproduce the captured data to create a *presence* with which the user can interact. The presence could be of a form and quality suitable to enable other remote users of the system to interact with it. The presence is also usually a reproduction or replica of another user, but this constraint can be lifted if the interacting users can interact with another protocol that is not just the body language and speech. It is possible to implement a remote presence system based on teleconferencing system, streaming video and audio from a place to another. However on a stage show it is also important to perceive the relative position of actors and possibly form more than one point of view. Given the potential difference in the stages layout, the possibility of relocating part of the show on a different position is also a requirement. All this can be combined with the necessity of showing only a part of the performance in case the resources available on a stage are not enough to replicate the whole performance.

To achieve this the Remote Presence prototype system renders streams of colored three-dimensional point clouds received from one or more remote stages. The data is acquired on the remote stages by the UCSD system and is annotated with space and time coordinates. This not only preserves the relationship of the actors on different stages allowing to reproduce the show, but also carries the potential for *manipulating* the streams of data. The goal of this manipulation can be as simple as relocating of the actors in another part of the stage. Or it can be a more elaborated control over the visualization to enhance and *amplify* the interaction among both actors and audience. In addition if the resources of the systems on a stage are constrained, the Remote Presence system can subscribe to only a fraction of the streams and allow the actors to interact even if only partially.

5.2 Architecture

The architecture of the system is summarized in Fig. 5.1. The system is divided in three sides: the Remote side, the Global and the Local side. On the Remote side we find the Detection and Analysis functionality, this functionality includes the acquisition of the data on a remote location and the detection of the remote users. The acquired data is transmitted to the Distribution on the Global side. The Distribution in turn deliver the data to the Local side. On the local side the data is received by the Effects of Delay Hiding functionality, this functionality manipulates the received data to hide the effects of the delays. At this point the data is ready to be rendered and delivered to the Rendering functionality, that will provide the user a replica of what has been detected on the Remote side.

The Remote side in this context can be any computer from any location that sends streams to the Distribution system. The source of the data could be a computer in the same room as the Remote Presence system. The Distribution functionality is placed on the Global side because both the Remote and Local side functionalities need to know about it. The Global side is the bridge connecting the other two sides. The Local side is the where the remote presence is provided to the final user by the Rendering functionality. On the same side the Effects of Delay Hiding functionalities takes place, after the data has travelled to the end point of the communication incurring in unavoidable delays.

The architecture does not present a functionality for Amplified Interactions, this functionality, and its absence is explained later in the chapter.

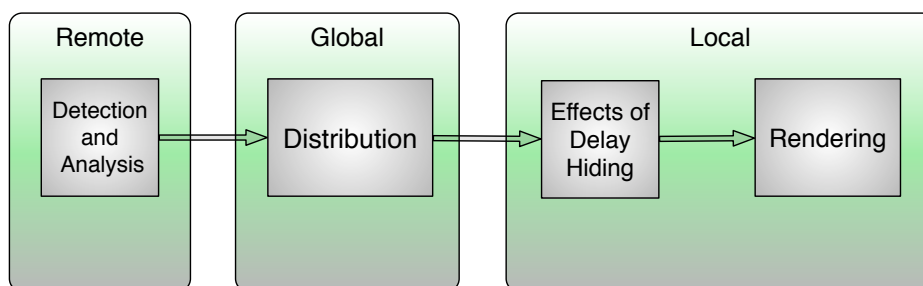


Figure 5.1: Remote Presence system Architecture. The system is divided in three sides comprising the main functionalities. Detection of the user is handled on the Remote side, on the Global side we find the distribution, on the local the Hiding of the Effect of Delay and the Rendering.

5.3 Design

Fig. 5.2 shows how the design of the system. In the design we joined the Remote to the Global side, the Local side is unaware of the distinction between the them and interfaces only with the Distribution. The data is generated by an instance of the UCSD system or other systems with similar capabilities.

As we can see, the Distribution system delivers the data to the Effects of Delay Hiding component, which is the entry point for the data to the system: it feeds the data to the Rendering component after masking the effect of delays if needed.

Given that the streams from the UCSD system are personal, comprising a single user, once inside the Rendering component the streams handled independently.

The separation is made by stream identifier that is unique inside a MultiStage setup. This also enables the integration with the components tasked to mask the effect of delays, Su et al. [3], that can manipulate single streams. Each stream is then rendered on a display and the end user can interact with it.

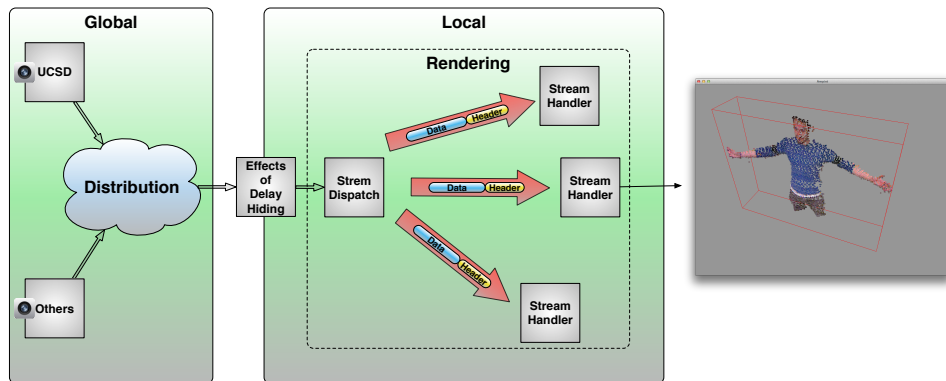


Figure 5.2: Remote Presence system design. Data arrives to the Rendering components from the Global side through the Effects of Delay Hiding component. The Rendering component handles the streams of data independently.

5.4 Implementation

Fig. 5.3 illustrates a schematic view of the implementation of the Rendering component of Remote Presence prototype system. The system is implemented in the Go programming language and uses the Go bindings to the OpenGL [66] API. The system exploits the Go language capabilities using goroutines

for concurrency and parallelism. It receives the streams of data and renders them to a display. The streams are generated by the UCSD system, each stream is divided into frames binary encoded and compressed using the Snappy algorithm [61] (see Chapter 3). Upon arrival in the system, each frame (or packet) is unmarshaled in a stream header and a colored point cloud.

The Remote Presence system holds a data structure mapping stream headers to Go channels, labeled *channel map* in Fig.5.3. The channels are used to supply fresh data to the goroutines handling the streams and are retrieved from the map using the stream header as key. The system uses one goroutine for each stream of data, which translates into one stream per user from another stage.

Each one of these goroutines handles the updates from the remote sites and maintains the state of the stream locally in a scene graph. The leaves of the scene graph are non-overlapping areas of memory, meaning the goroutines can update concurrently with the scene-graph. Another goroutine is used for rendering and wakes up every 16ms (60 fps) to perform the rendering.

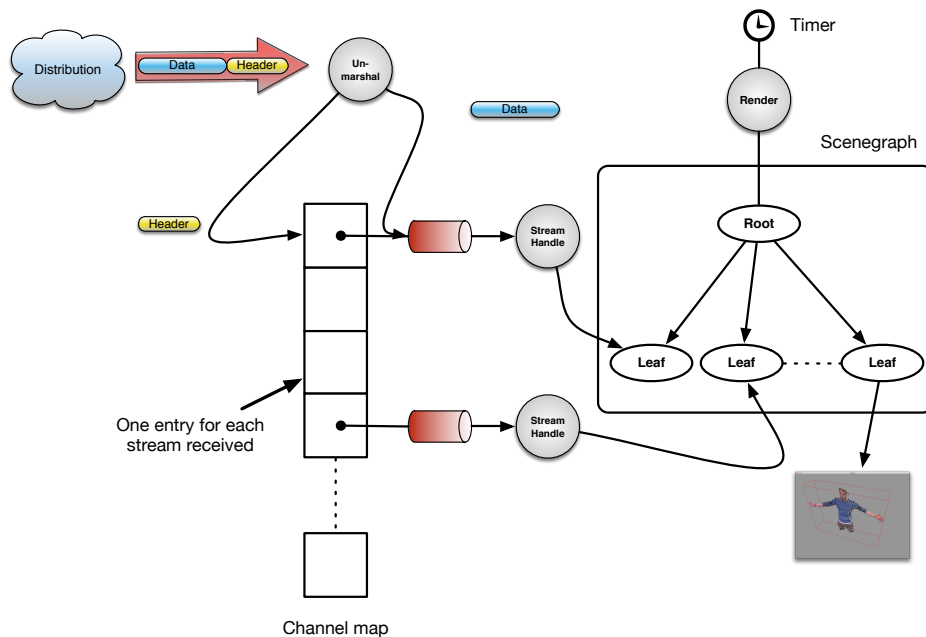


Figure 5.3: Implementation of the remote presence system. Each stream is handled and updated independently by a goroutine. The data is fed to these goroutines through a channel and used to update a scene-graph. Another goroutine renders the scene-graph synchronising with the other ones with a Readers-Writer-Lock.

The visual output of the Remote Presence system is a rendering of the point

cloud, surrounded by its bounding box, visible in Fig. 5.4.

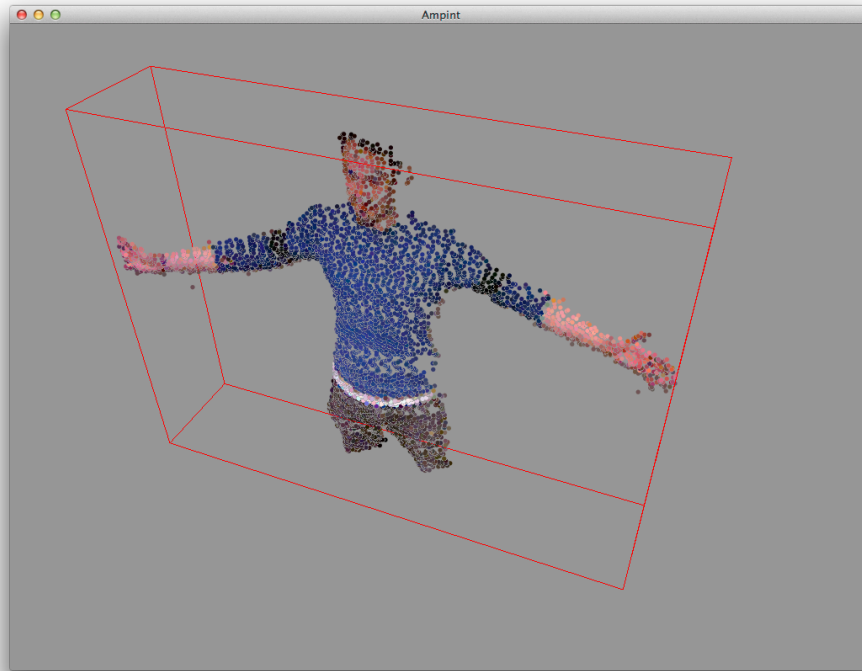


Figure 5.4: Output on screen of the Remote Presence system for one stream.

5.5 Experiments

We conducted an experiment to record the resources utilization of the Remote Presence system as part of a MultiStage installation.

5.5.1 Configuration

Fig. 5.5 shows the configuration of the experiment. The experiment measures the CPU and memory utilization and inbound network traffic of the Remote Presence system. The factors are the number of points from the UCSD system per stream. MultiStage is configured with three stages, each streaming four point cloud at 30 frames per second. Each stage subscribes to all the streams.

Each stage is composed of three computers, one running the Remote Presence system, and two running the UCSD system. Two Kinect cameras are connected to each of the UCSD computers. The Global Side and Distribution is composed by two more computers, one running the Distribution system and one running the Global State Analysis/Monitoring (see Chapter 3). All computers are connected to a 1 Gbit/s switched network.

The computers in the experiment are of two different specifications: the computer comprising Stage-2 are Intel Core i5 @ 2.5 GHz with 4 GB of RAM. All the others are Intel Core i7 @ 2.7 GHz with 8 GB of RAM as indicated in Fig. 5.5 by the dashed boxes.

The measurements (CPU memory and inbound traffic) have been collected with the monitoring system discussed in Section 2.5. The monitoring system measures resource consumption of the whole machines running the Remote Presence system, not at process level. This does not allow for a fine-grained collection of data on each system, but grants an overview on the general behaviour of MultiStage. This overview allows us to understand the trend in the resource utilization and to possibly take action to avoid saturation. In other words it grants the knowledge to quantify the amount of resources needed for a specific MultiStage installation.

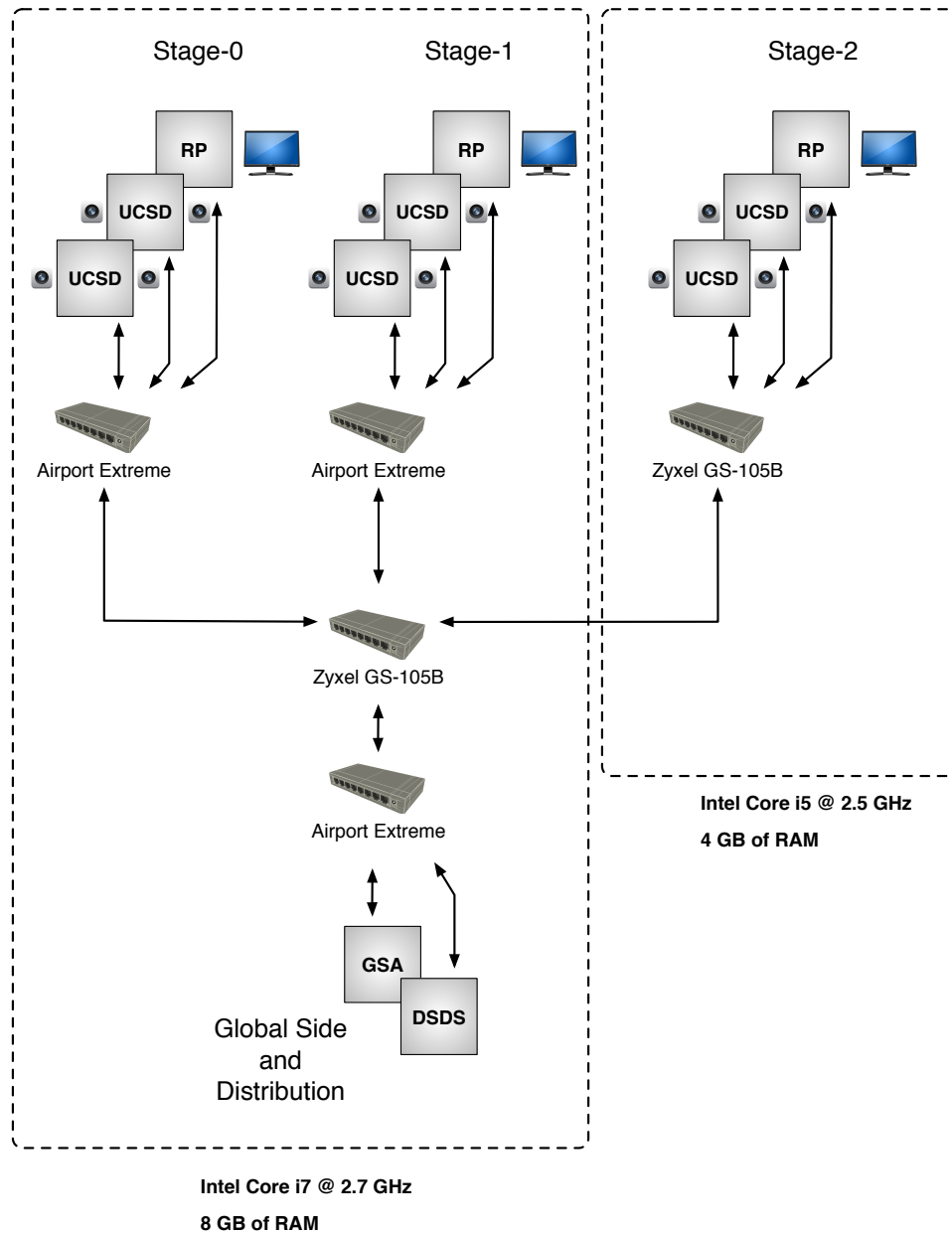


Figure 5.5: Hardware and software configuration for the experiment. Each grey square represents an Apple Mac mini running a system of MultiStage: UCSD and Remote Presence (RP). If the Mac mini has a device connected that is printed aside. This is visible for the Local and Sensor component, having connected to them respectively monitors and cameras. In this configuration the cameras are MS Kinects. All the network connections and switches are Gbit Ethernet.

5.5.2 Results

The results are visible in Fig. 5.6 and Fig. 5.7. The stage-0 machine used a steady 15% of the memory for all the points, from 1K to 5K, while the CPU raised from 10% at 1K points to 22% at 5K points. Stage-1 shows almost the same pattern in the CPU utilization, but a higher memory utilization around 23%. Stage-2 on the other hand shows an even higher memory utilization, roughly double the amount of the memory used by stage-1. However, this higher percent usage is justified by the fact that the total amount of memory available to the stage-2 machine was half of the available to the other two stages. Beside the memory amount stage-2 CPU utilization pattern is comparable to the other two stages. Other discrepancies in the measurements can be attributed to other processes running in the background occupying machine resources.

The inbound network traffic is stable for all the computers, varying from slightly less than 5 Mbyte/s for 1K points to 20 MByte/s for 5K points. This result was expected and corroborate the one in Chapter 3 where the cumulative outbound traffic of a stage was close to 7 MByte/s at 5K points.

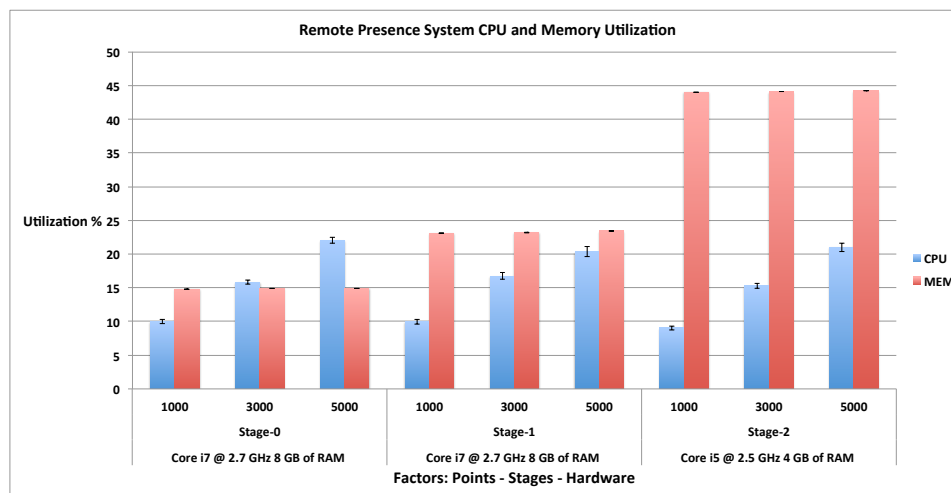


Figure 5.6: Percent CPU and memory utilization measured during the experiments for the Remote presence system. The error bars indicate the standard deviation.

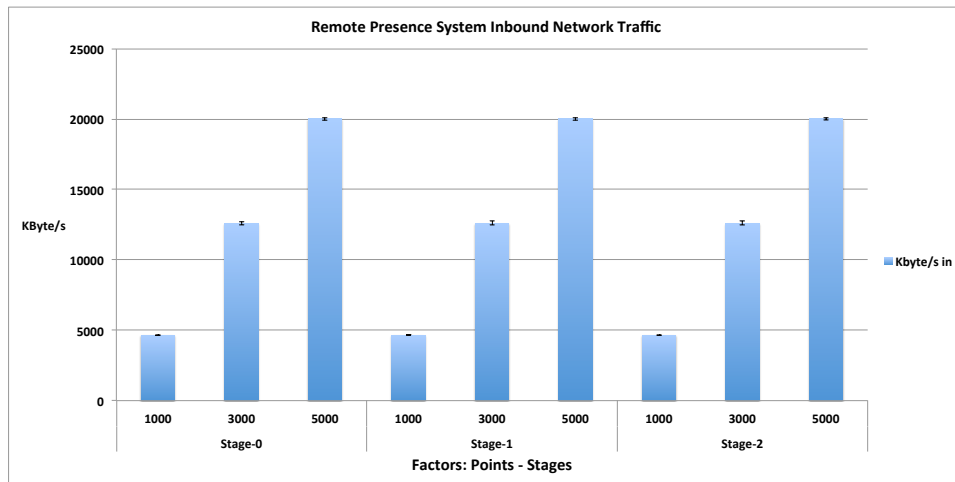


Figure 5.7: Remote Presence system inbound network traffic. The factors are the number of points per stream and the number of stages. The error bars indicate the standard deviation.

5.6 Discussion

5.6.1 Implementation

During the implementation we encountered the issue of how to synchronize the stream-handling goroutines and the rendering goroutine. If the rendering goroutine wakes up and reads the status of a stream to be rendered while another goroutine is updating it, a race occurs with unpredictable outcomes. To solve this problem we employed a *reverse* readers-writer lock (RWLock). It is a traditional RWLock but the readers' lock is held by the stream updating goroutines (writers) and the writer lock is held by the rendering goroutines (reader). The stream goroutines do not share the status of their streams and are allowed to write in memory simultaneously. The rendering goroutine on the contrary needs to read the status of all the streams, and no other goroutine is allowed to write during the rendering. A traditional use of the RWLock (readers holding a reader lock and writers holding the writer lock) would still have led to a correct solution, but at a loss of concurrent update of the stream state.

Another issue of the current implementation of the Rendering component is that it is limited in concurrency by the OpenGL implementation. While the goroutines are multiplexed on system threads, OpenGL calls need to be made from the same thread that created, and holds, the OpenGL context. There is no guarantee that an OpenGL call will be from the same thread that holds the

context, provoking unexpected failures. The Go runtime provides facilities to lock a goroutine to a system thread; in this way all the OpenGL calls can be limited to the goroutine that created the OpenGL context solving this problem. This limitation is at the time of writing being addressed in the next major OpenGL release, Vulkan [67], following a trend established by other platforms, such as Apple Metal [68] and AMD Mantle [69].

5.6.2 Amplified Interactions

We implemented two different prototypes of the Remote Presence system, with different capabilities. The first one was used OpenGL [66] and is described in this Chapter. The second uses the Horde3D [70] graphic engine. Some experimentation of Amplified Interactions has been attempted in the prototype using the Horde3D engine, without resulting in a fully functional proof of concept. An example of the concept is shown in Fig. 5.8. In the figure there are three stages and four actors, the actors are *amplified* by having their remote presence substituted by a 3D model or by having *special effects* applied to the remote presence. More details on this prototype are available in Appendix A.

5.7 Lessons learned

We can conclude that individual, per user streams of three-dimensional information captured at the begin point can provide added functionality and flexibility to the end point. Availability of 3D information on the actors allows cheap manipulation of the remote presence in a virtual environment. An example of flexibility is the opportunity to move the remote presence of the actors to accommodate for differences in stages layout. In addition the space coordinates of the streams allows to keep the displaced remote presence at a relative position consistent with the one of the original actors maintaining the performance coherent. An example of added functionalities is the possibility of masking the effects of delay. Different streams from different stages can incur in different delays or can need different way of masking it. The possibility of doing so is granted by having one stream per user.

5.8 Summary

We designed and implemented two systems for remote presence and Amplified Interaction between remote stages. We discussed and motivated the decisions that led to the global architecture of the distributed system. We measured the

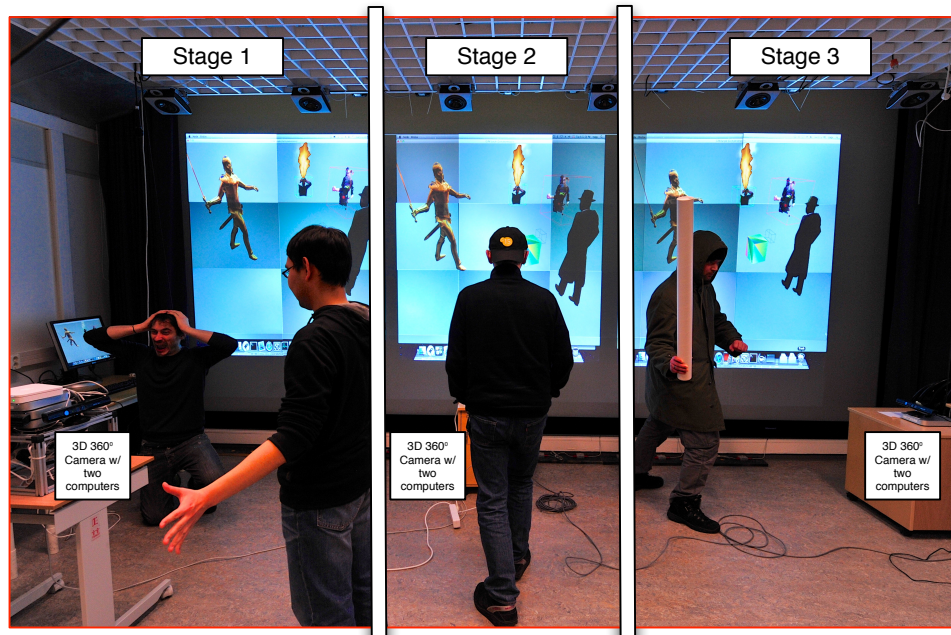


Figure 5.8: Output on display wall of three virtual stages. The output is augmented with 3D models provided by the Horde3D graphic engine. Figure from Su et al. [9]. The prototype in the picture uses the Tromsø Display Wall [36].

performance and the resource consumption of one of the implementations, discussed the results. A discussion of possible alternative architectures and implementation is also present, for a better understanding of the problems surrounding the topic.

/6

Global Interaction Space

6.1 Idea

In a room full of computers, such as an open-plan office, a computer laboratory in a university, or any room rigged with many computers and displays, there is often the need to control the remote computers or run commands on them. If there is the possibility to log in remotely to any computer and run a command, the problem is solved, but this is not often the case because of different issues, not the least of which is to know the IP address or network name of the remote machine. The problem is aggravated if we take into account laptops and other *moving* computers that can have a different dynamically assigned IP address.

A particular scenario in which this issue can disrupt the work-flow is a setup with a big, wall-sized display showing the results of some computations running on any of the computers. The necessity of analyzing the results on the big display, and running different computations on the various machines, implies walking around the room every time the need for fresh data arises, distracting the users and interrupting the work-flow. For this reason we built the Global Interaction Space (GIS), with the intention of controlling a room full of computers by using gestures. The computer configured as part of the GIS makes available to the user a set of actions to be performed, implemented as scripts. The user can execute these actions with gestures, on one or many computers from anywhere in the room.

6.1.1 Usage patterns

We provide here, as an example, a few general patterns of use that we targeted with our prototype.

1. Action execution on one (many) computer(s) with a gesture

The user selects the computer(s) in the room with a gesture. The computer(s) shows the state of selection to the user. (The user repeats the gesture and selects more computers.) The user executes the action on the selected computer(s) with a gesture. The computer(s) shows the state of execution to the user.

2. Action execution on *all* the computers with a gesture

The user selects *all* computers in the room with a single gesture. The computers show the state of selection to the user. The user executes the action on all computers with a gesture. The computers show the state of execution to the user.

3. Action execution on a single computer with a gesture

The user selects a single computer in the room, deselecting all the others, with a gesture. The computers shows the state of selection to the user. The user executes the action on a single computer with a gesture. The computer shows the state of execution to the user.

Examples of use based on the patterns above include:

Interactions at home. A user at home has configured a GIS for their computers: TV, cell phone, tablet, desktop, etc. The configuration comprises the appropriate sensors in the living room and the rest of the house. The user wants to display the content of the screen of all their computers on their TV. The user performs the second pattern in the list above, and the GIS takes care of showing the combined output of each device on the TV.

Interaction in the office. An office or meeting room is configured with a GIS. Each computer in the room and belonging to the person occupying the office is part of the GIS. During a meeting, the occupant of the office can, in turn, run commands on different computers using the first usage pattern in the above list, display different content on a big display, or start a demo with a gesture.

Stage performance. A performance is held on a stage equipped with many computers and configured as a GIS. The actors can perform gestures recognized by the GIS to interact with the computers, manipulating the stage appearance through light and displays, or reproducing sound

effects and sound tracks. Any of the above patterns can be appropriate in this scenario given the diversity of stage performance.

The GIS can be employed in other scenarios, but we believe the ones described above are enough to motivate it.

6.2 Architecture

A GIS comprises a set of functionalities (see Figure 6.1) divided into a global side and a local side. The global side comprises Room Global State Monitoring (RGSM) for monitoring of the room state and Room Global State Analysis (RGSA) to detect gestures from the users. The gestures are then processed and translated into actions by the Gesture to Action Translator (GAT) and delivered to the target computer(s). The correspondence of gesture to action is kept by the Gesture to Action Dictionary (GAD). The local side of the GIS is running on each computer that is part of it and can be the target of a gesture from the user. It is composed of two components: the Action Executor (AE) and the Visual Feedback (VF).

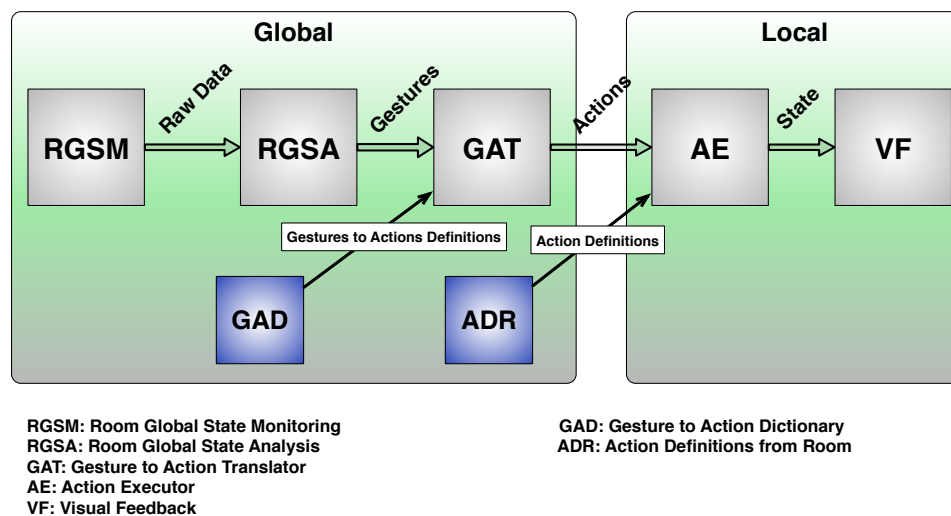


Figure 6.1: Architecture of the Global Interaction Space. The gestures are detected at the Global side by the RGSM and RGSA. The GAT translates the gestures to actions and the AE executes them on the Local side. The VF provides feedback on the state of the execution.

The AE's task is to execute the action received from the global side. The Action Definition from Room (ADR) holds the definition/implementation of the actions; at start-up each computer in the room pulls a fresh list of actions

from the global side. The VF provides the user with information on computer state and the action execution (e.g., ready, running, error etc.), with a simple and brightly colored geometric symbol. In this way, the user can see the state of the room at a glance.

6.3 Design

The design of the GIS is organized as a pipeline (see Figure 6.2). Each stage of the pipeline is in charge of some processing, and outputs data to the next stage, from the raw sensor data to gestures, translating gestures into actions, and then executing the actions on the target computers in the room while providing visual feedback to the user. Three major components emerge in the design: Sensor, Room, and Local components. Each component is a stand-alone part of the distributed system and communicates with the others exchanging messages in an intermediate format. The medium on which the messages are sent and received can be any convenient one; in the case that the components are deployed on different machines, the communications will be through the network.

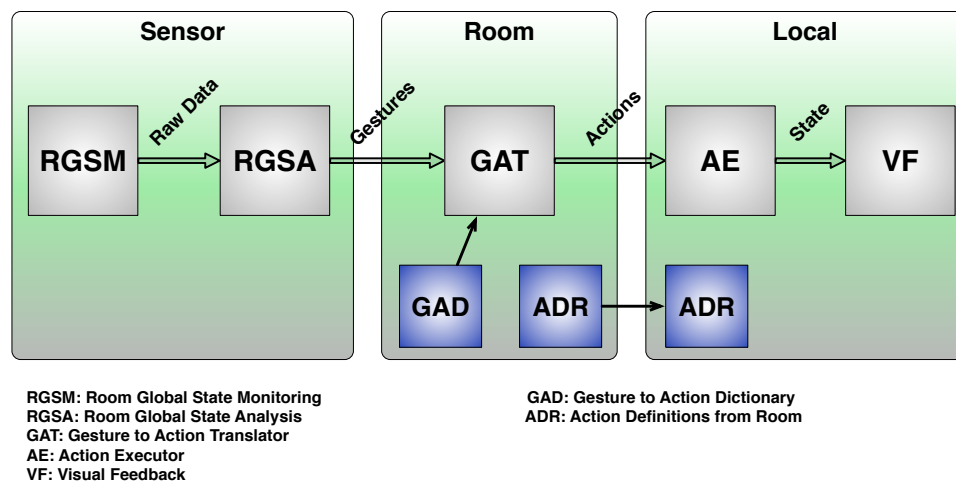


Figure 6.2: Design of the Global Interaction Space. The system is divided in three components. The data flows from the Sensor component, that detect gestures from the data acquired from the sensors, to the Room component. The Room component translates the gestures into actions using the GAD and sends the actions to the Local component targeted by the initial gesture. In order to execute the actions the Local component receives the actions definitions from the Room component. The VF shows the state of the execution.

Two of the main components – Sensor and Room – compose the global side.

This side represents the room as an environment with its configuration, while the local side represents the individual computer running the Local component. The Local component communicates with the global side to receive the action definitions and become part of the GIS. The communication between stages of the pipeline has a blocking receive and non-blocking send semantic. This implies some buffering from the sender, which in turn implies some added complexity, but this added complexity bears its own weight – the pipeline does not stall completely if one of the stages slows down.

6.4 Implementation

Fig. 6.3 shows the implementation of the GIS. As visible in the figure the Sensor component modules RGSM and RGSa are implemented respectively by the UCSD system and the Gesture Detection system (see Chapters 3 and 4). The UCSD monitors the room detecting users and sends their volumetric information to the Gesture Detection system. The Gesture Detection system is aware if the position and orientation of each sensor employed by the UCSD, this information is stored in a configuration file. With this information the Gesture System can associate each detected gesture with its point of execution in the room. In addition the data originated by the UCSD is timestamped, granting the Gesture Detection system the knowledge if the time of execution of each gesture. After detection each gesture is marshaled with its associated point and time of execution and sent to the Room component via a Transmission Control Protocol (TCP) connection.

The Room component function is to translate the gesture to actions and to send these last to the interested computer in the room. The translation is done using a dictionary loaded from a file containing the association between gesture and action. Changing the file changes the system behavior: different gestures will produce different actions. To deliver the actions to the target computer the system uses the Room Manager dictionary. The Room Manager dictionary stores details on the computers comprising the GIS, their location in the room, and their *volume of interest* or bounding box. The Room GAT uses the bounding box to determine whether a gesture is targeting a computer in the GIS. For example, if a gesture is performed inside a bounding box or if a gesture casts a ray that intersects a bounding box, the Room component will send the corresponding action to the computer associated with the bounding box.

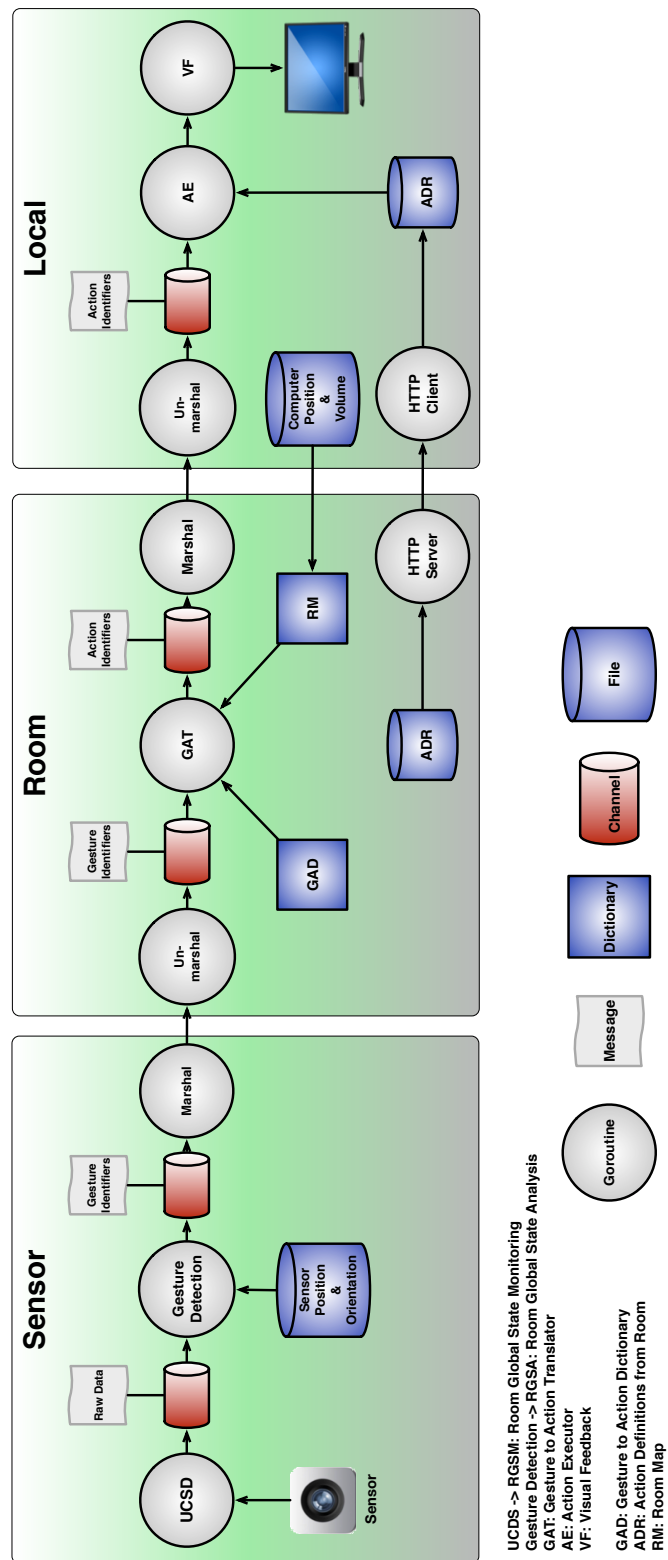


Figure 6.3: Implementation of the Global Interaction Space. The data flow is the same described in Fig. 6.2 with the addition of the implementation detail. The sensor component uses the UCSD and the Gesture Detection systems to detect user gestures. The messages exchanged among the components are marshaled in JSON. The Room Manager holds the position and volume of interest of all the computers in the room. The ADR is implemented with an HTTP server.

To be associated with a bounding box in the GIS a computer must be running the Local component. The Local component on start-up reads a configuration file containing its position and bounding box. This information is promptly communicated to the Room component to fill in the Room map. The Room map also holds the TCP connections that each Local component used to communicate its coordinates and bounding box, binding each Local component to a volume in the room. The Local component, also at start-up, receives the action definitions from the Room component. We implemented this as a Hypertext Transfer Protocol (HTTP) client/server communication where the Room component serves files to the Local component. As a language for the action definitions we chose Python. The Local component executes the actions in a separate process and provides visual feedback to the user. The visual feedback is implemented as simple and colored geometric shapes. Fig 6.4 shows a few samples and some possible future improvements.

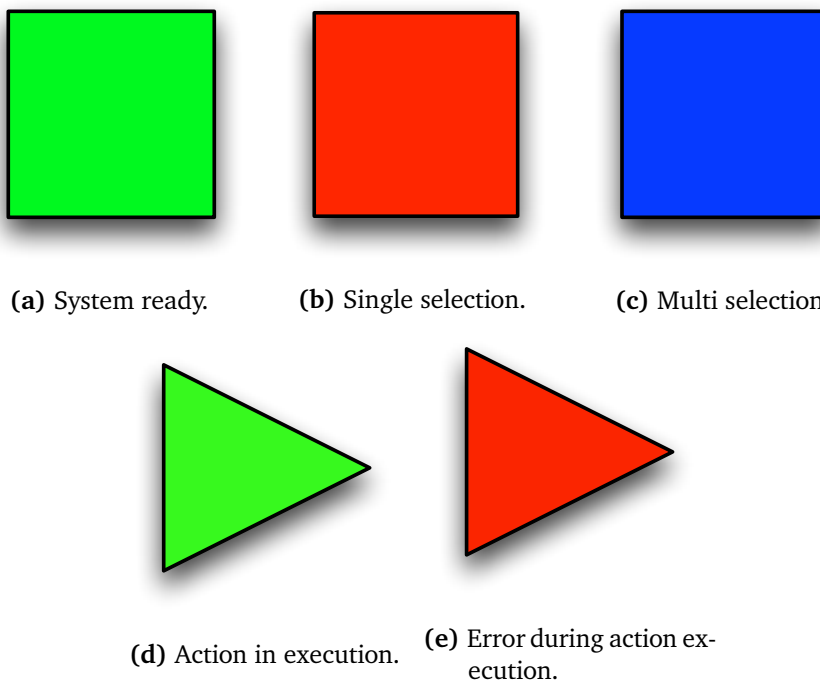


Figure 6.4: Visual Feedback of the Local component. Only one shape is shown at a time, the user can survey the global state of the room with a quick look.

Here we also need to distinguish the different action–gesture relationship hinted at in the use cases. Some gestures are not associated with action definitions from the ADR, but are interpreted by the Local component as a state change, giving us the possibility to discern between select gestures and execute gestures. The execute action is ignored by Local components not in a

selected state.

The marshaling and unmarshaling processes use JavaScript Object Notation (JSON) as intermediate format. Communication between components is done via network connection over TCP.

The system is implemented mostly in the Go programming language. Its CSP-like [63] concurrency features, clear syntax, and rich standard library make it a good choice, in our opinion, to write a prototype that can grow to a fully fledged application in the future.

6.4.1 Visual Feedback to User

As mentioned before and as is visible in Fig. 6.4, the Local component provides the user with visual feedback of its state. The feedback includes the selection state of the computer and the execution state of the action. The selection can be: no selection or ready (Fig. 6.4a), single selection (Fig. 6.4b), or multiple selection (Fig. 6.4c). When the user performs a gesture to execute an action, only the selected computers will execute it. The single selection is a convenience for the user to execute an action only on one computer; when a single selection gesture is performed targeting a computer, all the others selected will be deselected and transition to the ready state. The system does not allow two single selections at the same time, and the user can be certain to run the action only on one.

During the selection, the system does not run any action but just shows its state, when an action is executed, until its termination; the Local component shows the execution symbol (Fig. 6.4d). In the event of a failure detected by the Local component – an internal error or a bad return value from the action execution – the Local component shows the error symbol (Fig. 6.4e). To conclude, the visual feedback provides the user with knowledge of the state of the system and a reaction to the performed gestures. The visual feedback is the only way for



(a) Selection of computer across the room. (b) Action execution on the selected computers. (c) Single selection deselects the other computers.

Figure 6.5: Example of usage and visual feedback to user.

the user to know whether a gesture has been positively detected by the system. Fig. 6.5 shows a working example of some of the different visual feedbacks. As we can see, different Local components are running and changing state in response to the gestures of the user. The user can see immediately the result of their gesture as interpreted by the system. As a final note, the big display behind the user in Fig. 6.5 is a tiled display wall running a modified version of the Local component. This modified version provides a consistent view of the display wall, as it was a single computer.

6.5 Experiments

A set of performance measuring experiments has been conducted on the prototype system. A video of the functioning system is also available online [6].

6.5.1 Configuration

For the experiments, we configured the system with: (i) two computers, each with two cameras, and each running the Sensor component, (ii) one computer running the Room component, and (iii) multiple in-room computers, each running the Local component. All computers were Mac minis at 2.7 GHz and 8 GB RAM connected with a switched 1 Gbit/s link. A schematic of the setup is shown in Fig. 6.6.

We used the command line tool *ps* called in a small shell (see Appendix B.1) to measure each component's (Sensor, Room and Local) CPU and memory utilization. The script accepts the Process ID (PID) of a process and a file name as parameters, it calls *ps* every second with the PID of the process, and it logs the CPU and memory utilization in the file whose name it passes a parameter.

6.5.2 Results

The results of the experiments are shown in Fig. 6.7. The graph reports the CPU and memory percent utilization metrics. The results are obtained by querying the OS from another process using the *ps* tool. The measured metrics are the percentage CPU utilization of the process and the percentage of real memory used by the process.

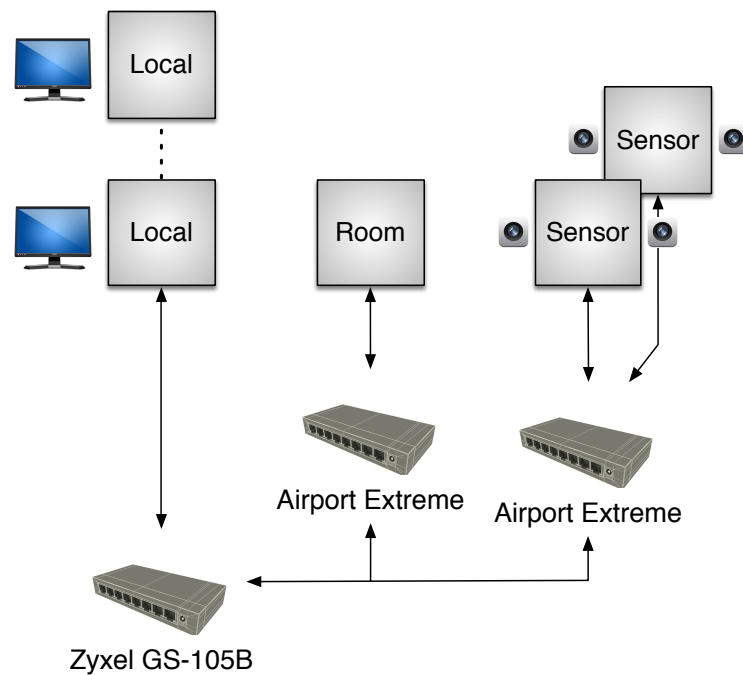


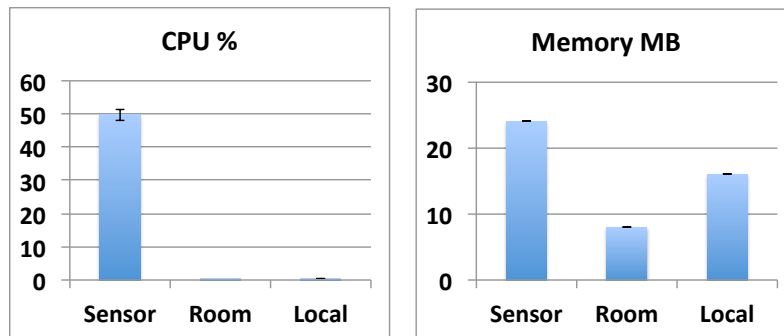
Figure 6.6: Hardware and software configuration for the experiment. Each grey square represents an Apple Mac mini running the component of the GIS written on the square. If the Mac mini has a device connected that is printed aside. This is visible for the Local and Sensor component, having connected to them respectively monitors and cameras. In this configuration the cameras are MS Kinects. All the network connections and switches are Gigabit Ethernet.

6.5.3 CPU

The results show (Fig. 6.7a) that the computers running the Sensor components used close to 50% of the CPU available. This result is in accordance with that in Chapter 4. The Room and Local components have a very low CPU utilization, below 1%. This measurement does not include the action execution, the actions are executed in an external process and the resource utilization can be different with different actions.

6.5.4 Memory

The memory utilization is stable and below 1% for all components. Specifically, we measured 0.3% for the Sensor, 0.1% for the Room, and 0.2% for the Local components. Considering that the physical memory of each of the computers used during the experiments is 8 GB, we can give a more precise enumeration



(a) Percent CPU utilization of the system components.

(b) Memory utilization of the system components in Mega Bytes.

Figure 6.7: Global Interaction Space CPU and memory utilization. The memory utilization measured in percent was 0.3% for the Sensor, 0.1% for the Room, and 0.2% for the Local component on a total of 8 GByte. Here reported in MBytes for clearer comparison. The error bar indicates the standard deviation.

in MB of the memory utilization: 24 MB for the Sensor, 8 MB for the Room, and 16 MB for the Local component (see Fig.6.7b).

6.5.5 Network

We measured less than 100 Bytes transmitted on the network per gesture or action. The low network traffic is motivated by the delay necessary to perform and detect a gesture that sets an upper bound to the amount of network traffic. Combining these information we can deduce that a real situation cannot produce more than few KByte/s of traffic, not a problem for a Gbit/a link.

6.6 Discussion

In this chapter there is a discussion on the design, implementation and limitation of the prototype.

6.6.1 Design

Here are discussed some design decisions. A GIS installation is designed to have only one Room component, while there can be more than one Sensor and Local component. This design decision reflects the fact that multiple sensors

can be part of the room and they can be distributed around the room to better cover its entirety. In addition, the Sensor components are in charge of the gesture detection, a potentially CPU intensive task. In this way, the system can distribute the computational load to the different computers handling the sensors. Being a *singleton* component in a GIS installation makes the Room component a good candidate for hosting the ADR dictionary containing the action definition.

6.6.2 Implementation

Some of the motivations that steered the choice of the technologies in the implementation are discussed here. The action definition are delivered via HTTP, it is a straightforward way to serve files and in case of further developments an HTTP server can be a starting point for more than just serving static files. For example it could implement a Representational State Transfer (REST) API providing a more complex behavior.

Python was chosen for the action definitions because it is a widespread scripting language available by default on many platforms. Its interpreted nature allows us to store the source code, ship it, and run it on most platforms.

Among the different serialization formats, JSON is a well-established format for data exchange, and many languages have libraries to read and write JSON files in case other components of the system, written in other languages, are added. In addition, JSON is text encoded and easy to debug compared to binary formats. Even though binary protocols tend to be more compact, but the analysis of the network traffic in the previous session shows that more compact data encoding is not needed.

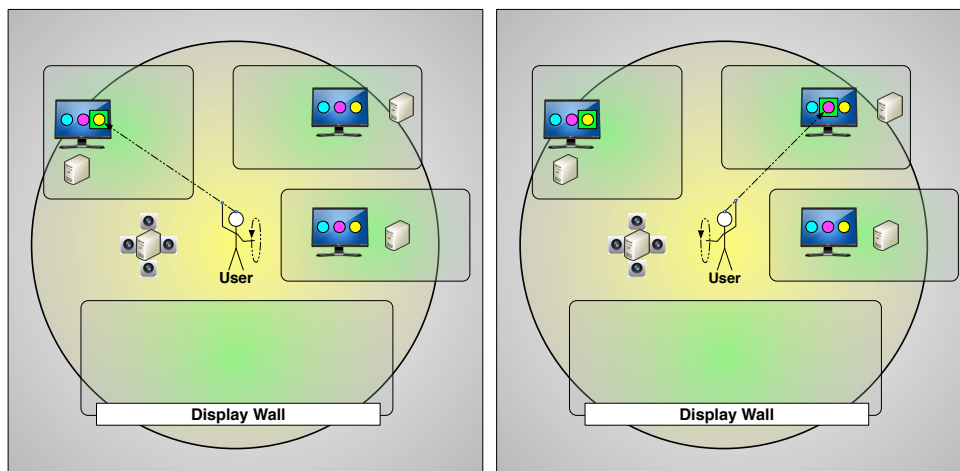
Given the distributed nature of the system and the possibly non-exclusive use of the network, we implemented the communication over a reliable protocol: TCP. A packet lost or delivered out of order can produce unpredictable results.

6.6.3 Limitations

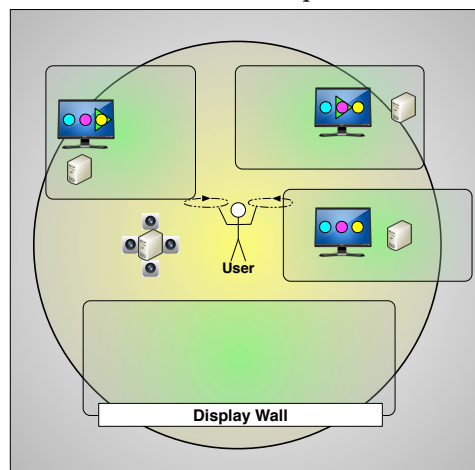
The current implementation has some limitations. First, it assumes that the position of the Sensor and Local components in the room is static, that their position in the room is fixed and read from a configuration file. This simplifies the prototype implementation but does not allow moving computers and sensors to be part of the GIS prototype.

Second is how actions are selected and started. In the present prototype there is

a one-to-one binding between gesture and action execution. This conceptually collapses both selection of an action and execution of an action into a single gesture. This does not scale to more than a few actions with regards to the number of gestures a user can remember and the gesture vocabulary detected by the system. This can be improved by expanding the selection gesture vocabulary with a script selection gesture, and the appropriate visual feedback from the Local component, to allow the user to select both the action to be executed and the computer on which to execute it. Another generic execute gesture can then trigger the execution. Fig 6.8 shows an example.



(a) Selection of an action on a computer (b) Selection of a different action on a computer



(c) Execution of the selected actions

Figure 6.8: Possible improvement on the current implementation. The user can select single actions on any computer and execute them at the same time.

A third limitation derives from the action definitions language, Python. Version

incompatibilities between the interpreters on the Local components can cause the execution of an action to fail. In addition in the case of a complex action, all the needed dependencies must be satisfied by having the necessary Python module installed. An alternative to this could be writing the action definitions in Go and having the Room component cross compile them in a statically linked binary for each architecture and platform of the Local components present in the GIS. The Go compiler produces by default binaries that are self-contained and do not need shared libraries on the host. The Local component would then retrieve the binary file of the action and run it.

A last limitation is the unawareness of the system to different users – it does not track specific users or the provenance of the gestures. There are no user profiles or user settings. This implies that any number of users can be in the room performing gestures and running actions. To not interfere with each other, users need to collaborate.

6.6.4 MultiStage Applications

Outlined its capabilities in the previous sections, we can now fit the GIS in the context of a multistage distributed performance, especially in the context of remote interactions. During a distributed performance the actors cannot communicate directly so they rely upon other mediums to convey information to other stages, such as video streams and remote presence systems. Using a potentially non-optimal medium of communication can leave the experience wanting and diminish the enjoyability of the performance or hamper it altogether. GIS can then potentially assist by delivering additional information and extending the communication on a different axis.

The communication can then happen by defining a set of gestures, and commands, and placing a computer belonging to the GIS at every stage. The actors can in this way influence not only their own environment, but the environment on other stages, possibly compensating for the non-optimal communication. Beyond that we can speculate on the potential of not just using this new means of communication as crutches to sustain a leaking abstraction, that actors on the remote stages can interact as if they were on one stage, but as an instrument of communication as dignified as the ones already in place, as is discussed in Section 5.6.2.

6.7 Lessons Learned

The GIS shows that it is possible to select and use multiple computer at the same time by using only a few gestures. During the use of GIS we found out that circular gestures provide the least amount of false positives during the normal activity of people in the room. The circular motion also provides a sense of completeness to the user, meaning the user knows when he has moved his hands in a complete circle and can expect a result from the system. This mitigates the lack of feedback during the gesture, feedback usually present on visual interfaces or other haptic devices.

The draw back of circular gestures is that they can contain other simpler motions. For example an up-and-down gesture, the gesture of rising and lowering a control point (see Chapter 4), can be detected before the completion of a circle, because rising and lowering the control point is part of describing a circle. The system will then produce the up-and-down gesture, flush the buffer containing the motion string (see Chapter 4) and start waiting for another gesture. In this scenario the circular gesture will never be detected. Other approaches, as seen in [65], use more sophisticated regular expression systems to avoid this issue.

With the knowledge and insight obtained from building the GIS, we designed and implemented Mr. Clean (Tartari et al. [71]). Mr. Clean is a system designed for comparison of scientific visualizations on big displays in the context of data cleaning of biologic dataset.

6.8 Summary

This chapter presented the Global Interaction Space. The GIS provides its users with the ability to execute predefined actions. In the current implementations it means to execute Python scripts, on any computer part of the GIS with gestures, standing anywhere in the room. This capability is useful in scenarios where the user needs to run different computations on different machines and compare the results.

In the context of MultiStage the GIS can be used to enhance the performance by detecting gestures of the actors and running commands on computers on stage. These commands can provide special effects of light or sound to underline moments in the narrative, or used to activate and control Amplified Interactions (see Section 5.6.2).

We presented and, where fitting discussed, the architecture, design and imple-

mentation of the GIS. Presented its limitations, proposed improvements for the future and considered the lessons learned by building the system.

/7

Contributions

Here follows a review of the contributions of this work, similar to what we have already seen in Chapter 1, but with a bias on where these contributions have been made, especially now that we have had a broader view of the systems, their architecture, implementation, and the idea driving their making.

7.1 Lessons Learned

Implicit and Explicit state changes. In principle a stage can modify the state at another stage in two ways: (i) by modifying the state sent to other stages or (ii) asking the other stages to change the state.

The first is an implicit change, the second, an explicit. A receiving stage can refuse an explicit state change but will be oblivious to an implicit one. In the first case the sending stage has the responsibility of changing state before sending it to the remote stages. In the second case the receiving stage receives commands to change the state locally. Which one is used has impact on safety, resource usage, and customization of a stage.

With regards to safety, the implicit approach makes it easier for a malicious stage to forge fake state updates and steer the receiving stages to a harmful state. For example in the case of *mobile robotic telepresence systems* [72]. The explicit approach makes it easier for the stages to avoid

harmful states by masking harmful commands.

With regards to the resource usage, the implicit approach is likely to increase CPU usage at the sending side and saves CPU at the receiving side. Network traffic can also increase in some cases, and local latency at the receiving stages can be reduced.

Customization of a stage is, for example, to define the local layout of a stage including where the remote presences are located. Other customization can involve amplified interactions and how the remote presences are *dressed*. The implicit approach makes it harder for a stage to do local customizations.

Single-data stream, single user. The Detection system collects the state of the stage in the context of single users and put this information in a per user data stream. The advantage of having a one-to-one ratio between users and streams is that the remote presence of each user can be individually manipulated with low resource usage.

For example, two remote presences can be created at different locations on each stage. Each remote presence can be delayed or accelerated and modified to create special effects and mask the effects of delay.

It also allows for clients with limited resources to receive only the streams they can handle or are interested in. A similar concept is presented in other another work, ViewCast [43], where each remote presence is comprise of different streams acquired from different angle all around the user. Each receiving end of the remote presence decides the streams to receive based on the combination of streams that compose the current view of the user on each remote stage from the perspective of the viewer. MultiStage applies a similar concept to the whole performance, allowing to choose the remote presence to receive.

RGB-D cameras can reduce the CPU usage when detecting an actor's state.

Three-dimensional cameras can deliver not just normal images but also *depth images* containing 3D spatial data, for this reason they are often called RGB-D. The depth images can be used to record what is in a pre-determined volume ahead of the camera. An actor to be detected must be inside this volume. This helps in detecting the state of the actor with low delays. In effect, the low delay detection is achieved by limiting what the cameras are recording. The alternative approach would be to let a camera record multiple actors and process the video stream to create separate state streams per actor.

In one scan of the RGB and depth images is possible to acquire the state of the actor in front of the camera. This state can be used to create a remote presence or to detect gestures. The computation needed can be kept low to allow capturing the state at the full frame rate of the camera. The resulting remote presence can therefore have the same fluidity of the full frame rate of the camera. Similar techniques are used in other works (e.g. Fanello et al. [33]) to detect users, where they assume the user stands in front of the camera and no other movements is present in the background.

Remote presences as 3D point clouds. Creating a remote presences using 3D point clouds rendered onto large displays mimics the layout, size and shape, of a human being. The remote presence shows limbs and body posture, allowing to express, even if in a limited way, the body language of the acquired user. Mimicking the human body is also useful for performance measuring experiments of the system when doing human to remote presence interaction, for example a hand shake. Simple everyday interaction can be recorded and analyzed. The representation is scalable with regards of sampling rate and number of points in the cloud.

However this is a trade-off compared to other systems (e.g. [43], [25], [22], [45]) where the remote presence is closer to reality in its look and also has three-dimensional characteristics. This in turn requires more intensive computations, with the effect of slowing the frame rate.

Observer redefines observed. An observed user on stage can be mapped or redefined into an object being simpler to analyze when looking for gestures. The users are aware of the simplification and adapt their behaviour accordingly. The MultiStage gesture system redefines the users' body into four control points being the leftmost, rightmost topmost and nearest part of the body relative the sensor used. With the knowledge of these control points, and how to move them, the users can perform gestures. The simplification saves processing and reduces delays.

This implies that the gesture detection system (see Chapter 4) does not need to detect the exact body part performing the gesture as long as the gesture is simple enough. There is no need to detect a *right* hand or a *hand* at all as long as the user has the means to perform a gesture.

7.2 Models

This section expands on the list of models briefly exposed in Chapter 1.3.2.

Decoupled producer and consumer with monitored distribution. When the producer and consumer of data streams are decoupled they are not directly connected, and potentially unaware of each other. The data is transferred by a distribution system, [3], that is by design distributed on the producer and consumer machines. The distribution system is also monitoring factors such as, for example, end to end latency or lost packets. This opens the possibility to seamlessly substitute the produced data without the consumer noticing when the distribution system detects, for example, an end to end delay that would disrupt the communication. The distribution system can also employ other techniques, such as replacing the leaked packet or adjusting the reproduction times of the streams.

A gesture recognition model based on simple volumetric detection of users.

A gesture can be anything regarding a user: a movement of an arm, a nod of the head, a pose held for a defined amount of time, moving from one position to another one, or even changes in the heart rate. All these potential gestures have in common the tracking of characteristics or features of the person performing the gesture. With 3D sensors is possible to track the volume occupied from an actor on stage. Characteristics of the volume can be changed by the user and tracked to detect gestures. In this way is possible to detect simple gestures with low processing and delays.

Global Interaction Space. The GIS is an extension to the Interaction Space model [4], where each computer in a room can execute commands as response to gestures performed by users in the room. Through the GIS a user can choose one or more computers and run a script on them with a gesture. The GIS is responsible for running the computation on the chosen computers and displaying a result. The model can be useful when a user needs to steer the computation of many computers in a room while standing in the same room, allowing the user to run scripts at the same time on multiple computers.

This model can be applied to stages during shows. A stage can be configured with computers to serve as platform for stage tricks, such as light and sounds, and actors could command these special effect with gestures.

7.3 Artifacts

The following artifacts were created during this project.

Sensor Suite. In order to detect users and actors on a stage sensors must

be employed. The solution proposed in this dissertation is a sensor suite composed of four 3D cameras controlled by two computers. One last element completes the sensor suite, a network switch with wireless capabilities. The network switch provides connectivity between the computers comprising the suite and to the outside world.

The sensor suite has cumulative range of vision close to 360 degree when placed in the middle of a stage or a room. The sensor suite can detect and acquire up to four users. The images acquired by the cameras, can be processed by the computers comprising the suite and shared via network. The system is described in details in Chapter 3.

A User Context State Detection, Analysis, and Sharing System. This system is the software counterpart of the Sensor Suite, it process the data coming from the sensors and encodes it in one data stream per user. After encoding and compressing the streams it delivers them to the distribution system. The system is described in details in Chapter 3.

Remote Presence system. The Remote Presence system creates remote presences of the actors. The remote presences are based on the streams received by the distribution system and normally generated by the UCSD. The remote presences are three-dimensional colored point clouds. The point clouds have enough details to show limbs and body language to the other users. The system is described in details in Chapter 5.

Bounding box and point motion system. An axis-aligned bounding box for a set of points is the box with the smallest volume within which all the points lie, with the constraint that the edges of the box are parallel to the coordinate axes. In this specific case the set of points is the point cloud comprising the detected user. And the coordinate axes are the axes of the camera acquiring the user.

The bounding box is built by identifying the six points that hold the maximum and minimum value for each of the three coordinates X, Y, and Z. A user with knowledge of how the volume is built and where the control points are situated can move them with his body, and perform gestures. The system is described in details in Chapter 4.

Gestures through Regular Expressions and User Volume Control Points. This system extends and completes the *Bounding box point and point motion system*. It translates the control points movements in strings of text, *motion stings*. The motion strings can be parsed and analyzed to determine if the movement of the control point was a predefined gesture. The system uses a regular expressions engine to perform such analysis. The

gestures are predefined as regular expressions and the system tries to match the motion strings against the predefined regular expressions. If a match occur a gesture is detected. The system is described in details in Chapter 4.

The Global Interaction Space. The GIS system is the combination of the systems describes above. The GIS is a distributed system that allows its users to remotely control multiple computers in a room with gestures. The GIS provides a gesture interface to the user; a user can choose a computer and trigger the execution of a computation on it with a gesture. This approach can be extended to many computers at the same time, allowing the user to execute computations at the same time on multiple computers. The system is described in details in Chapter 6.

7.4 Facts

Based on experiments and performance measurements we collected the following facts on the systems and artifact presented in this dissertation. As mentioned in Section 1.3 these measurements are not benchmarks but an overview of the resource utilization of MultiStage. The data collected is meant to be a paragon for future systems built within the same scope. This data are summarized in Table 7.1. A more complete description of the experiment setup is in Chapter 5.5.

System	CPU	Memory	Bandwith
Gesture	50%	< 1%	NA
GIS	50%	< 25 MB	< 1 KByte/s
RP	< 25%	< 45%(< 25%)	20 MByte/s (3 stages, 4 Streams, 5K points, 30 fps)
UCSD	< 5%	25%	< 3.5 MByte/s (2 cameras, 5K points, 30 fps)

Table 7.1: Collection of performance measurements. Gesture Detection system and GIS measurements are at process level, while User Context State Detection (UCSD) and Remote Presence system (RP) rows show machine resource utilization. The Remote Presence results come from two different sets of machines with different hardware configurations, specifically 8 and 4 GByte of memory. The number in parenthesis refers to the normalized utilization for all the machines.

The only systems whose process used 50% of the CPU was the Sensor component of the Global Interaction Space (Chapter 6) and the gesture detection system (Chapter 4). In both cases the memory used was less than 25 MByte,

and sensibly less for the gesture system alone.

The performance of the other systems was not measured at process level but at system level. The tool used for the measurement is described in Su et al. [9]. The User Context State Detection (UCSD) and Remote Presence system (RP) (Chapter 3 and 5) never used more than 25% of the available CPU.

The total machine memory consumption observed for each system during the experiments was never above 45% of the available. However, not all the systems were running on machines with 8 GByte of RAM, some had only 4 GByte (see Chapter 5). Normalizing the memory utilization of the systems running on 4 GByte of memory to the ones with 8 GBytes we can reconcile the memory utilization of all the system to be below 25%.

Combining these facts with the bandwidth used we can deduce the scaling factor of Multistage. If each stage is equipped with four cameras (7 MByte/s with 5000 points see Chapter 5) and each stage receives all the streams, assuming an ideal linear scale, at three stages each stage receives 21 MByte/s. With four stages the needed bandwidth for each stage is 28 MByte/s.

Given these numbers, and assuming a central distribution system, the discriminant for the scalability of MultiStage is the outbound bandwidth of the distribution system. In case of a three stages setup it will be 63 MByte/s, with four stages it will be 112 MByte/s. At four stages a Gigabit link will theoretically still be enough (125 MByte/s), but it will need better hardware or a better system architecture or better compression of the transmitted data to scale further.

This hypothesis has been confirmed by our experiments, Fig. 7.1 shows the results of of an experiments with a three stages configuration. Two of the stages were equipped with a UCSD system while the third was *simulating* one by transmitting a prerecorded point cloud. The rest of the third stage was commensurate with the other two. The bandwidth consumption measured is from the Distribution system point of view, which was running on a single machine. For the experiment each stage was configured with four cameras (for the UCSD system Chapter 3) and a viewer (for remote presence Chapter 5). The number of stages was increased from one to three. In addition each stage subscribed to all the streams from the other stages.

It is possible to configure the stages in a different way to better adapt to different circumstances or to make different trade offs. For example reduce the number of stream subscriptions per stage or the number of points in the point clouds. An expanded discussion on the implications can be found in Chapter 8.

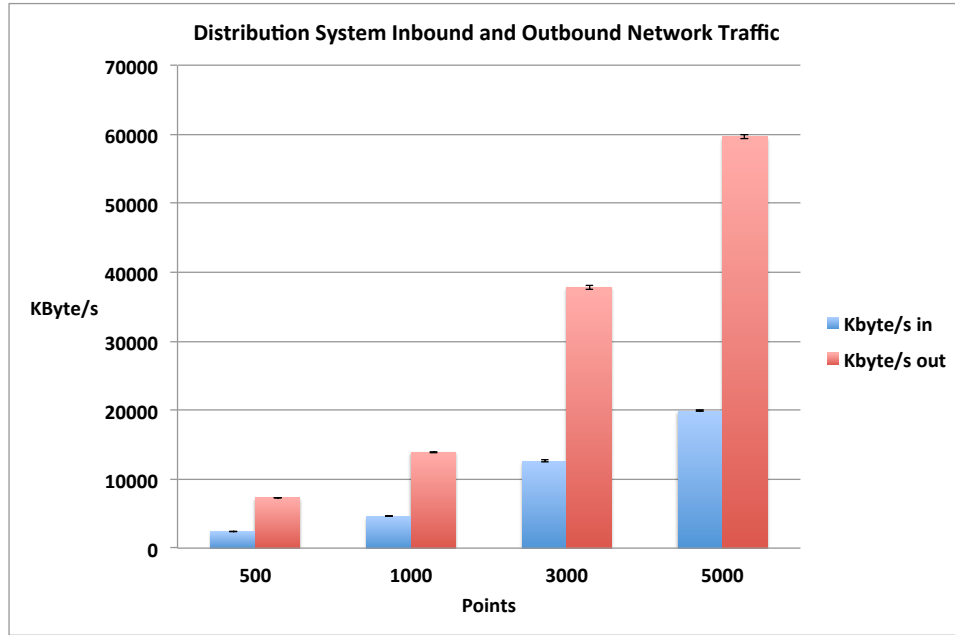


Figure 7.1: Scalability of Multistage. The experiment shows the bandwidth needed by the Distribution system. The figure shows a three stage setup. The error bars indicate the standard deviation.

Another measurement collected during experiments is the delay in detecting a gesture. The data are summarized in Table 7.2 and are detailed in Section 4.5.1. The gestures are detected by matching strings of text produced by the motions of the user. The different gestures have different *length*, meaning different time to complete which explain the different latency. The details on how the gestures are detected are explained in Chapter 4.

Gesture	Average	Median	Stdev
Circle	0.957227185s	0.863717786s	0.45646036s
Straight	1.355079928s	1.280198133s	0.47902494s

Table 7.2: Summary of Gesture detection latency. The different latency for the different gestures are in part caused by the *length* of the gesture, the *Straight* gesture is longer. For details see Chapter 4.

/ 8

Discussion

MultiStage is a distributed system with different capabilities. It is built with different tool and technologies each of them providing different trade offs. The trade offs and the decisions about the architecture of MultiStage are discussed in Chapter 2. Others are exposed in the rest of this chapter.

8.1 Missing Capabilities

During the development, we diverged from the initial plans and did not implement all the subsystems as initially specified in the architecture. For example, we did not implement any global gesture through the GSA (see Chapters 2 and 3). Another divergence is the lack of Amplified Interactions (see Section 5.6.2 and Appendix A). The alternative of developing our own minimal graphic engine was abandoned.

In retrospect it could have been a good idea to start the project using a fully fledged graphic or even a game engine. Among the most recently released and made available to the public we can find Unity [73] and Unreal [74]. However, this approach can present issues in integrating with other systems, such as [3]. Or manifest a non irrelevant overhead from adapting to an API not friendly with the domain. Game engines tend to be monolithic and might be difficult to only use a subset of the functionalities.

8.2 Limitations

Other parts of the system were in line with the original plans, such as the gesture system and the UCSD system. These systems have proven useful, but are not safe from critiques. The bounding box model, described in Chapter 4, is lightweight and device agnostic but is not enough for a complete and generic interaction paradigm. It is quite cumbersome, prone to *gorilla arm*, and the gesture dictionary is quite small. Nonetheless, we believe it was a worthy proof of concept.

8.3 Technologies

Many and different technologies for human–computer interactions are emerging in the consumer market, the Microsoft Kinect used in the Sensor Suite is an example. Other 3D cameras with better characteristics than the Kinect are now on the market, the Kinect 2 or the ZED camera [83] are just an example. Both cameras use a different technology from the first Kinect and exhibit better performances, for example higher frame rate, no interference between two cameras, higher resolution, longer range, and the possibility to be used outdoor. Other upcoming 3D sensors are Tango [75] and Movidius [76], which can outperform the Kinect and be integrated in hand-held devices. Rebuilding the Sensor suite with these technologies can lead to remote presences with higher resolution and resemblance to reality, and potentially more fluid because of the higher frame rate of the sensors.

On the gesture detection side the model employed, the bounding box model (see Chapter 4), is general enough to be used with other technologies. To be more precise, the model is not bound to the Kinect 3D camera, and any device capable of detecting 3D objects is a potential candidate for our model. In addition any of the many smart watches and wrist bands that can track movements in two or even three dimensions can be used almost out the box. By associating a *control point* to wrist worn tracking device a user can perform gestures as it was in front a Sensor Suite. Tools such as the Leap Motion [59] or the Myo Armband [60] could be used to add other levels of interactions not available with the bounding box model.

8.4 Architectures

The architecture we chose was, to our knowledge, the best trade-off given the requirements: scale to a handful of stages and a handful of people. Given

different prerequisites, other architecture could have provided a better trade-off.

We also took for granted the presence of a physical stage or a room where the actors perform. This might not be the case in the future as handheld devices that are wearable, such as the Oculus Rift [78], are growing in power and immersive technologies are becoming mainstream. The implication of this could be a shift in the distributed performance paradigm, and the possibility of having a distributed performance almost anywhere. This paradigm shift will most likely render the current architecture obsolete, and new ones will need to be devised.

8.5 Tools

Another part of the project where we debated choices was the tools used. Tools such as the programming language are quite relevant choices, and do not come without trade-offs of their own. For example the amount and quality of libraries available for that particular language, the productivity and speed of development associated with it or the inherent complexity of the resulting code.

It is not easy to compare all the pros and cons, but we can try to motivate our choice of the Go [79] programming language. We started using Go before version 1.0 was available, so it was not yet completely stable. Nonetheless, it showed a maturity level beyond its young age, in stability, richness of the standard library, and tooling. Google itself was using it in production, and each new release came with a tool, *gofix*, that corrected your entire code base to match the last iteration of the language. *gofix* made the development unhindered by the changes in the language even while the language specification was still in flux. Go also supports concurrency natively at language level, making parallelization of execution simple, in line with the multi/many core era we are facing.

Other tools we debated using were the official driver for Kinect (OpenNI [80]), or the open source ones (OpenKinect [81]). The OpenNI drivers (at the time version 1.x) were proprietary but offered skeletal recognition of the user. The CPU use though was quite high; however, while skeletal recognition is a useful feature, it was not indispensable for our purpose, and the OpenNI API was needlessly complex for our needs. In contrast, the OpenKinect API was a lean and clean C API easily wrappable to be used with Go. Calling the OpenKinect API was possible to obtain the 3D point cloud and the colors associated to each point. Worth mentioning is that all the code was open and publicly available.

Even with the lack of skeletal recognition, the OpenKinect drivers were enough to use the Kinect 3D cameras and without the added complexity of the other drivers. Lastly using the proprietary drivers would make it more difficult to switch to a different camera in the future, while dealing with sets of 3D points can be a more generic API.

8.6 Feasibility

In the most general instance, a multistage system could be deployed on different continents. This possibility brought us to think about the physical feasibility of the project. The speed of light is a hard limit in terms of minimum delay in communications. To help give some context to this statement, here are some approximate numbers. The speed of light in a vacuum is roughly $3 \times 10^8 m/s$, and the earth's circumference is roughly $40000 km$, so in the worst-case scenario in ideal condition, which is to send a signal to earth's antipodes around its circumference, it takes around $67ms$. Given that the results from [47], the worst-case scenario is at least impractical for a distributed musical performance.

In addition the theoretical $67ms$ does not take into account the fragmentation of the path of the signal, a necessity given by the earth's morphology and division into countries. One continuous physical medium of transmission, for example a fiber-optic (or even copper) cable, is practically unattainable. The signal needs to be interrupted and amplified to traverse the globe. Another potential delay is given by the speed of light in a medium different from the void, even if this would probably result in a minor addition to the delay not worth considering.

We are still not considering other sources of delay, such as sensor acquisition, processing, encoding, etc. All these other sources of delay can add a considerable amount to the total. In addition, if we consider more than two stages, we can have different delays related to different relative distances, which makes compensating for delays to obtain a consistent performance even more complicated.

However, if the stages are not too far apart in terms of latency (which translates almost directly to physical proximity), the network delay can be small enough to leave some maneuvering space to deliver acceptable results. For example, following the reports found in [8] about end-to-end performance measurements of Internet links, we find that inside Europe we can expect an end-to-end ping of $40ms$. This leaves some room to deal with the delay.

8.7 Scalability

As mentioned earlier in Section 7.4 we experimented with a three stage setup of MultiStage. The results are summarized in Fig. 8.1. The experiment configuration, as introduced in Section 7.4, was the following: each stage was equipped with four cameras producing four point clouds with a variable number of points at 30 frames per second. Each stage subscribed to all the available streams including the ones it generated. The factors in the experiments are the number of stages and the number of points in the point cloud streams. The data collected is shown from the point of view of the Distribution system. This means that the data *in* is the data collected from the all the stages, while the data *out* is the total data delivered to all the stages.

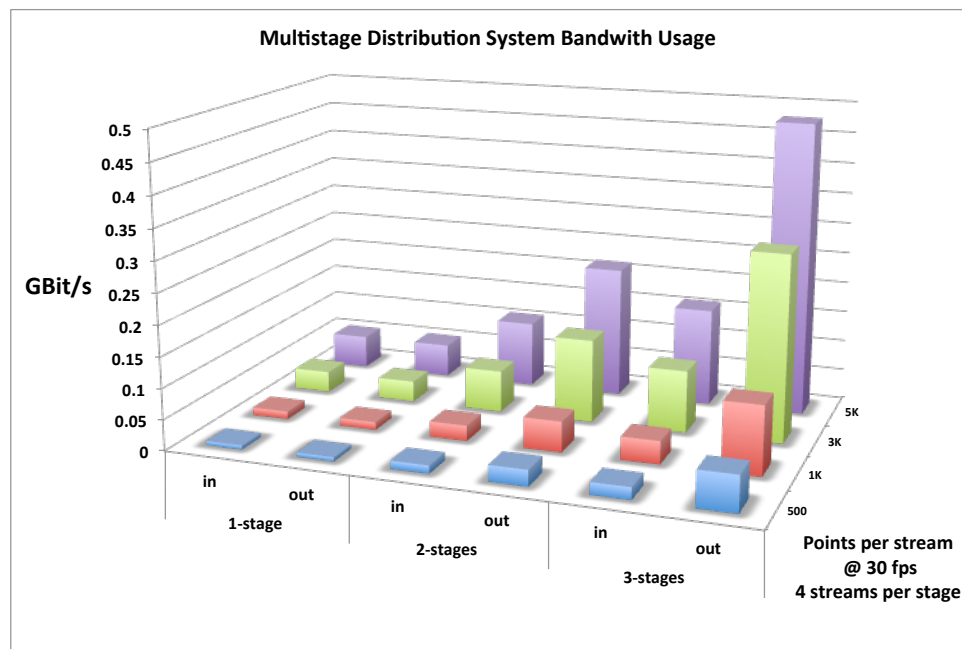


Figure 8.1: Scalability Of Multistage in proportion to a Gbit/s link. Three stages and up to 5K points for each point cloud.

Base on the data from the experiments in Section 7.4 (the data was collected in collaborative effort with Fei Su during the preparation of Su et al. [9]) we deduced the *theoretical* limit of the MultiStage system to be at four stages on a Gbit/s link. However, it is possible to have more bandwidth than one Gbit/s. Therefore, in order to have broader overview on the scalability of the system, we made a projection of the data collected during the experiments.

The projection follows this pattern: doubling the amount of points in the point cloud doubles the *data in* to the Distribution system. The *data in* for an n -stage configuration is n times the *data in* for a one-stage configuration. The

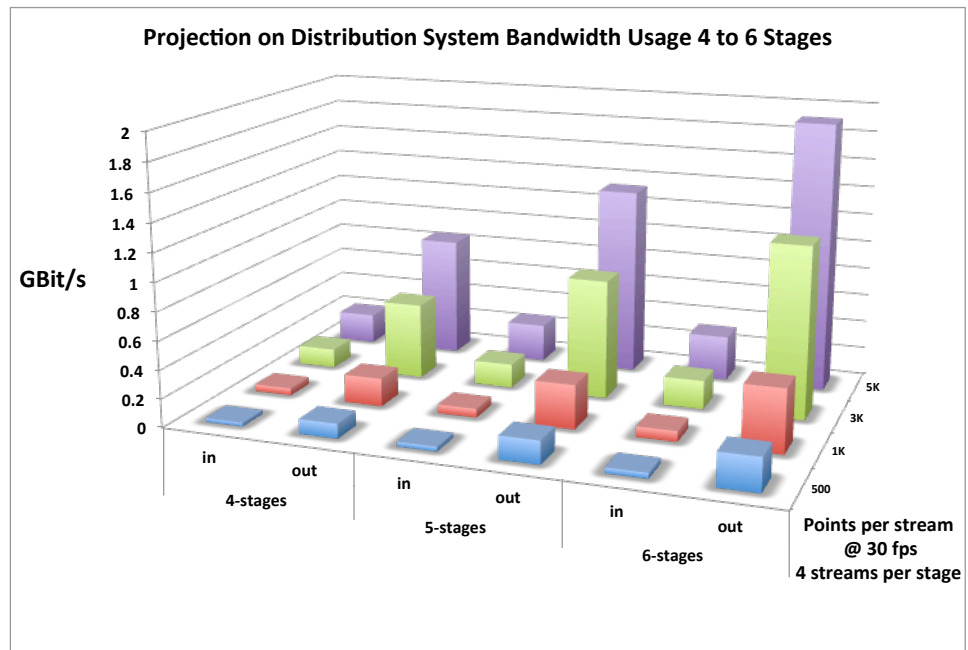


Figure 8.2: Projection on the bandwidth use with four to six stages and 500 to 5K points.

data out from the Distribution system for an n -stage configuration is *data in* times n .

In other words: given *in* the data received by the Distribution system, *out* the data delivered to each stage, k a compression/encoding factor, P the number of points in the point cloud and n the number of stages, the following formulas describe the projection.

$$in = kP * n$$

$$out = in * n$$

Substituting becomes:

$$out = kPn^2$$

Revealing that the outbound bandwidth needed for distribution grows with the square of the number of stages.

Fig. 8.2 shows the projection of bandwidth consumption for four to six stages. While Fig. 8.3 gives a projection on the scale in number of points up to 80K while keeping the number of stages between one and three. As expected a four stage configuration with the highest resolution for the point clouds (5K points) touches the Gbit/s boundary (Fig. 8.2). While a six stages configuration reaches 1.9 Gbit/s of outbound bandwidth from the Distribution system. A higher resolution for the point cloud is also expensive in term of bandwidth. A three-stage configuration reaches and outbound traffic of 7.6 Gbit/s at 80K point, and to stay barely below 1 Gbit/s (0.95 Gbit/a) the system needs to drop the resolution to 10K points.

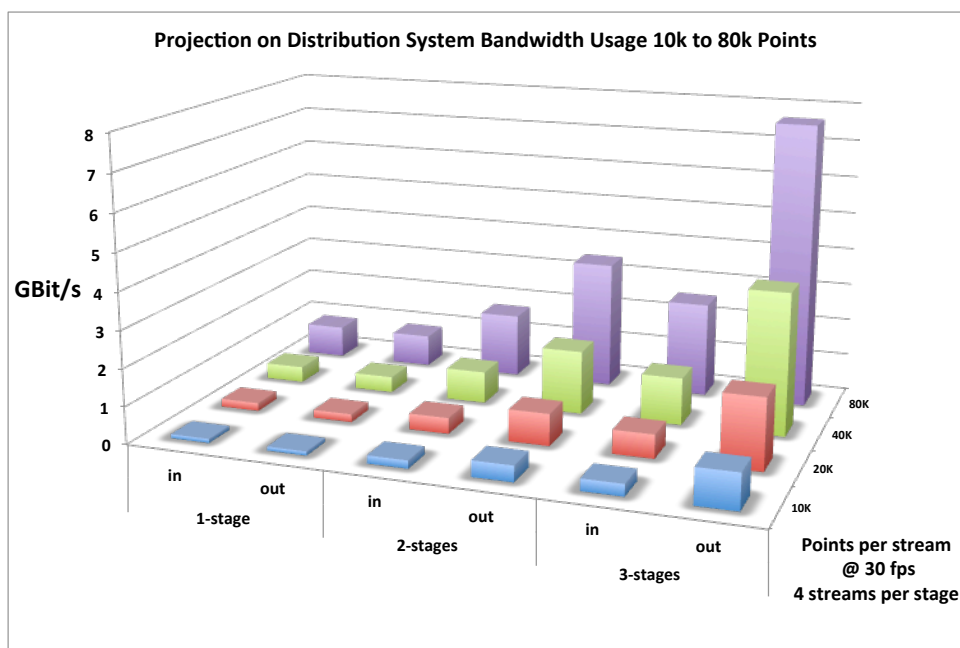


Figure 8.3: Projection on the bandwidth use with 10k to 80k points one to three stages.

To better understand the full scale we also provide Fig. 8.4 that collects the projections and the experimental data to give an overview of the scalability of the system. The overview spans up to six stages and 80K points per point cloud and keeps the inbound and outbound traffic separated. Fig. 8.5 presents the same data but with aggregated inbound and outbound data.

This projection does not take into account other important scaling factors such as the number of streams or the compression level. If we increase the number of streams per stage (above the four we used in the experiments) the result will be more bandwidth occupation, and a rise in CPU and memory consumption for the systems involved.

A better compression will also most likely increase CPU and memory usage but in this case with a reduction of the bandwidth needed. This makes it a good candidate for a speculation on the scalability of the system, but the lack of experimental data in this area does not allow us to make a projection.

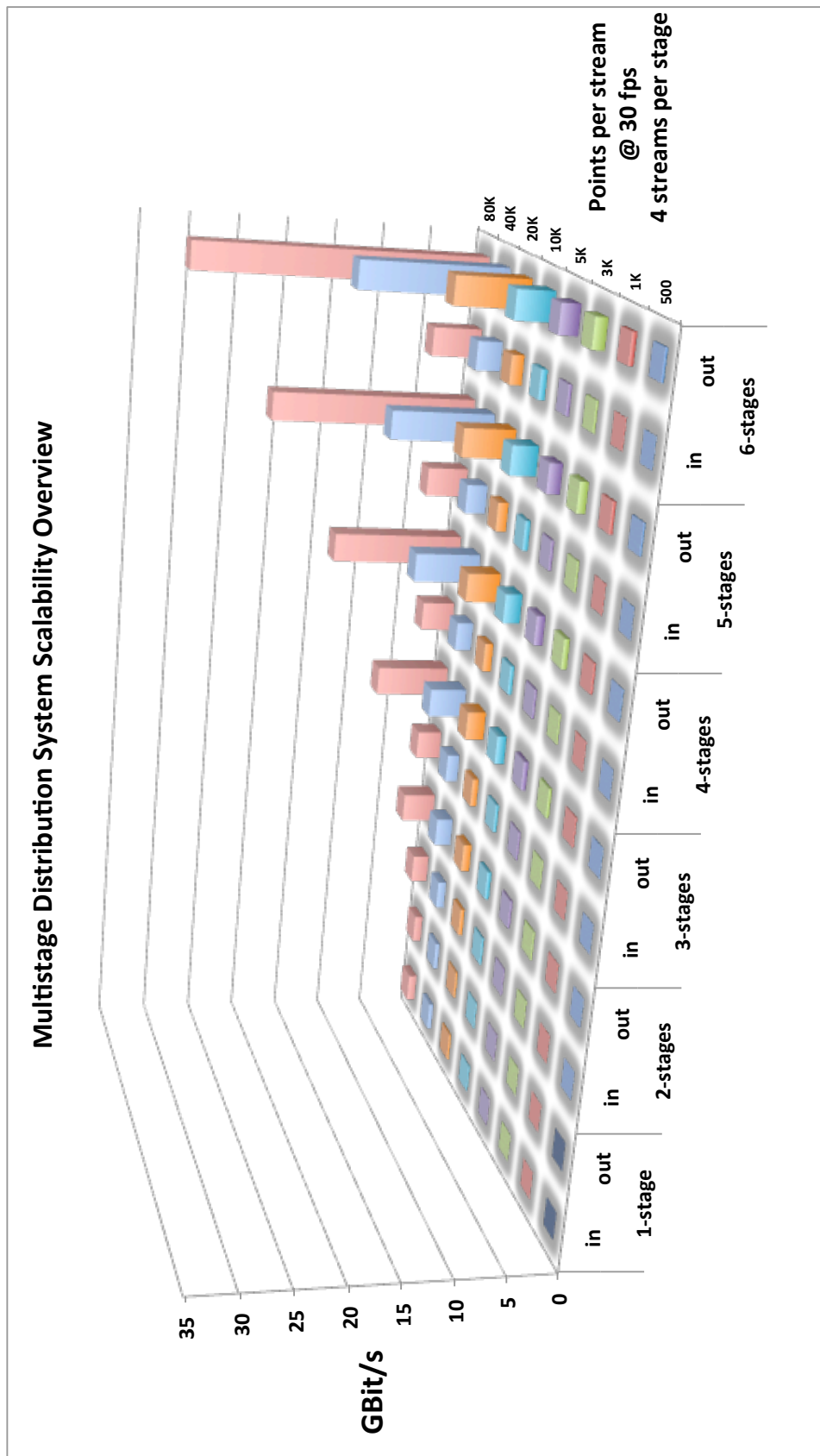


Figure 8.4: Scalability overview of MultiStage. The overview spans up to six stages and 80K points per point cloud.

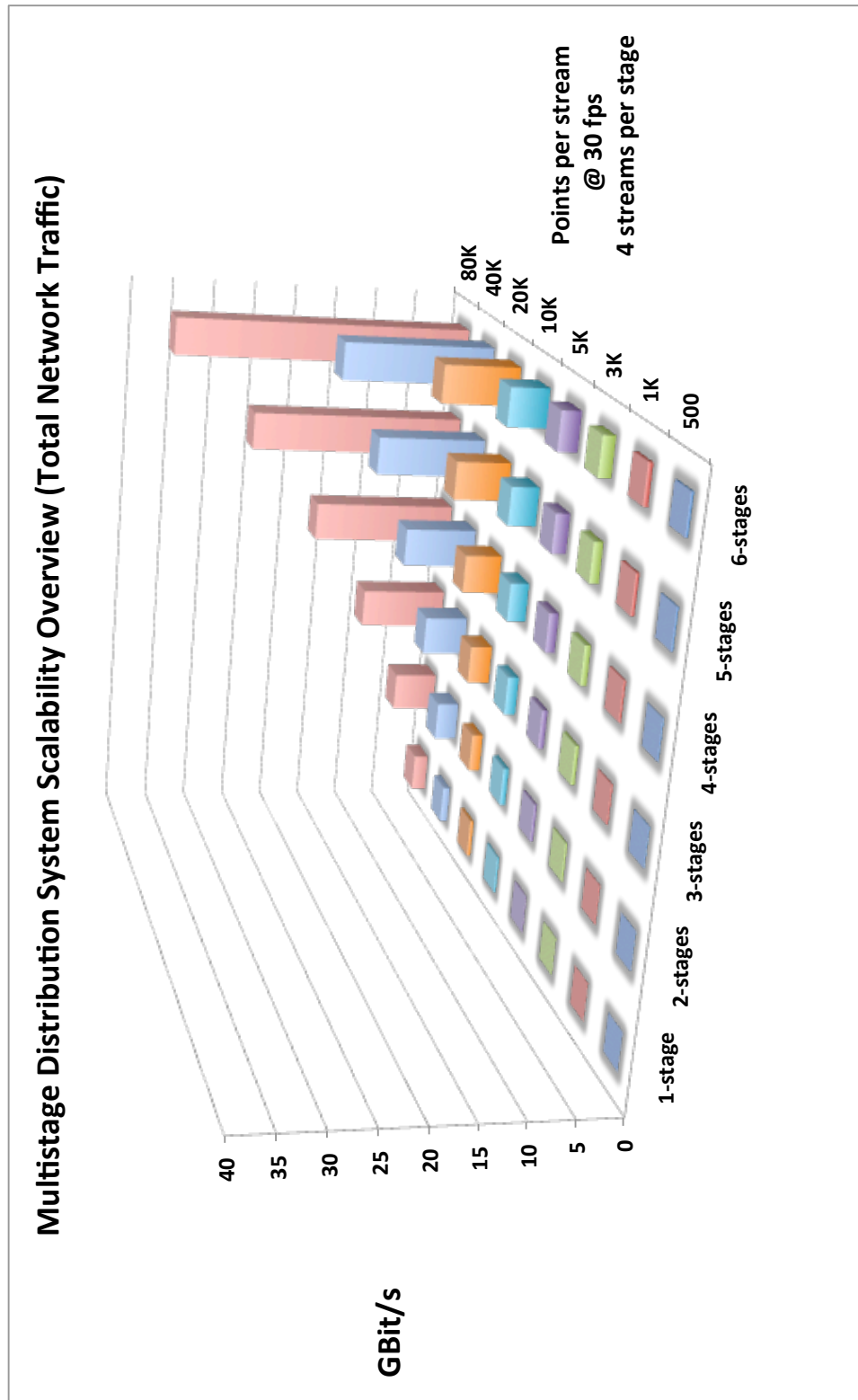


Figure 8.5: Scalability overview of MultiStage with aggregated traffic. The overview spans up to six stages and 80K points per point cloud, the bandwidth is the total traffic, in and out, seen by the Distribution system.

/9

Conclusions

This dissertation presented the MultiStage system and its components, the ideas and principles behind it, and the implemented prototypes (proofs of concept) when available. For each of the components we discussed the architecture, the design, and the implementation.

In Chapter 2 we presented an overview of the concepts and problems as well as a brief look at the state of the art. We also explain the macro architecture of our vision of the MultiStage system and explore some possible alternatives. The chapter concludes with a discussion of the design derived by the chosen architecture.

The following chapter (Chapter 3) presents the UCSD system. The system is meant to collect the state of a remote stage and make it available to other stages. The UCSD system is a working proof of concepts presented in Chapter 1, and through it we learned the lesson about splitting the data stream into single user streams to be manipulated independently. With the UCSD system prototype we also learned that a 3D point cloud is a convenient way to create a remote presence. 3D cameras can be used successfully for remote interactions, and with the more advanced options such as the Myriad chip (Movidius inc.) available on the market, more options suddenly become available. This will bring the possibility of building more powerful, efficient and compact systems similar to the UCSD.

The final recipient of the UCSD system point clouds is a system to display them

in a 3D environment; the Remote Presence system allows remote interactions between stages (see Chapter 5). We applied the *explicit and implicit state change* principle (described in Chapter 7) to the design and implementations in order to fit the model of *decoupled space–time with third-party fixer* (see Section 1.3.2 and Su et al. [3]). With the combination of both, the data can be modified during the transit, improving the flexibility of handling the data.

Chapter 4 presents a gesture system that uses the UCSD system. The gesture system can detect simple gestures performed by users in the room using volumetric information on the users. The information is used to describe a bounding box around the users from a set of control points. Granted the users the knowledge on the control points they can use their body to perform gestures by moving the points. The model is simple and effective, but allows a limited gesture vocabulary.

The gesture system was a fundamental part of the GIS (Chapter 6), a distributed system designed to interact with the computers in a room. The use of gestures to interact with computers is hardly novel, but while implementing such a system we also devised the Global Interaction Space model. The model is useful in a scenario where it is necessary to interact with multiple computers, and even if the system implementing it is a prototype, we believe the model can have practical and useful applications Tartari et al. [71].

Notwithstanding these limitations this dissertation shows that it is possible today to open new channels of interaction between remote stages and allow the actors/users to exploit it. These channels can be gestures performed by the actors/users on the stage to emphasize some actions on stage or to trigger some service routine to be ran on a computer on stage to activate special effects. It could also be *global gestures* performed on all the stages in order to change some global state of the performance. All this can be coupled with Amplified Interactions and give a whole new axis of interaction to the show.

9.1 Future Works

Technologies that can be fruitfully employed for remote presence are growing in number and maturing. There are already immersive systems and wearable devices that can, and probably will, be made smaller and more portable ([82], [76], [83], and [75]). And systems targeting the professional market ([77]). Given these emerging technologies it is not far fetched to think that in near future the capabilities of the UCSD system could be found in a handheld device. The implications are many and promising, like the possibility to have a shared stage in your pocket, enabling the opportunity for *distributed flash mobs*.

Wearable and not intrusive devices could detect gestures more easily while offloading the computation effort from the rest of the *stage*. Remote presence and Amplified Interactions could be experienced through other specialized wearable devices, for example Oculus Rift [78], or directly through a personal device. Global localization could be used to find the local performance with the best latency to either spectate or participate in the show.

Appendices



Amplified Interactions

Amplified Interaction means rendering or representing the user in a different way based on their behavior, based on a script or by giving them the chance to modify their remote presence via a gesture. Examples of amplification include a glowing arm, a different color in rendering the remote presence, or in the case of a 3D model being used, an entirely different 3D model. The user could be rendered with different clothes, be of a different height or at a different position. The amplification could be a text box floating above the remote presence illustrating feelings or inner dialogues that are not directly, or not easily, conveyable with more traditional remote presence systems. The boundaries and the triggers of the amplification can be different in different contexts. For example, in a stage performance reenacting a fantasy battle, the remote presence can be amplified with armors and weapons or be represented on a horse. In a more abstract and creative context, it can be used to directly produce *special effects* based on the movements and/or sounds of the user.

Fig. A.1 shows an example of the idea of Amplified Interactions. The figure shows three stages and four actors, on each stage there is a UCSD detecting the actors and their remote presence is visible on the display-wall. The display wall is divided in three areas, each area represent the Remote Presence system output on the respective stage. Their remote presence differs in ways it is amplified. The actor spreading its hands had no amplifications, its representation is a colored point cloud as the one produced by the system presented in Chapter 5. The actor kneeling with his hands on his head is also represented with a point cloud, but its remote presence is amplified with a flame bursting from his head.

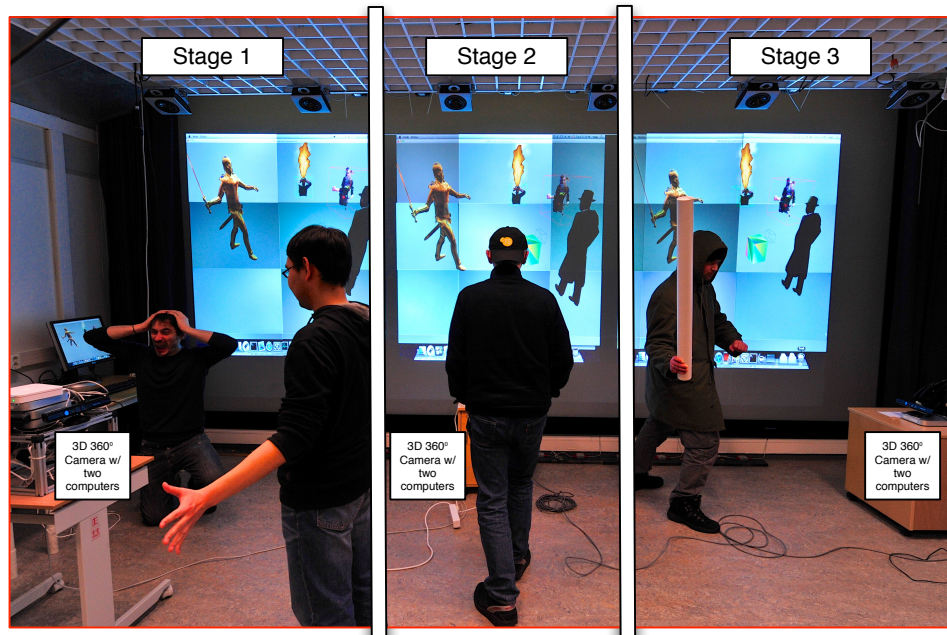


Figure A.1: Output on display wall of three virtual stages. The output is augmented with 3D models provided by the Horde3D graphic engine. Figure from Su et al. [9]. The prototype in the picture uses the Tromsø Display Wall [36].

The remaining two actors are represented with a 3D model that changes their looks but preserves the body language. In alternative the system could change the look of the actors based on their behavior or pose. The prototype gives an idea of the possibilities, but it lacks the necessary features or completeness to achieve the Amplified Interactions we planned and idealized.

The prototype system for Amplified Interactions is based on the Go binding for Horde3D, a graphic engine that supports 3D models and animations. Horde3D is a graphic engine and not a full stack game engine, it does not include a sound and networking API. The engine also allows mixing primitive OpenGL calls with the normal engine functionality; this allowed us to reuse part of the Viewer implementation and merge it with visual effects, such as particles, and 3D models from the Horde3D engine. These features made it a good candidate for prototyping Amplified Interaction.



Code

```
1 #!/bin/bash
2 # Usage: logcpumem [PID] [filename]
3 #         replace [PID] with process ID #
4 #         replace [filename] with file name to use
5 filepath=.                # modify if needed
6 timelimit=6000           # how long to run, in seconds
7 mydate='date +%H:%M:%S'  # the timestamp
8
9 while [ "$SECONDS" -le "$timelimit" ] ; do
10     ps -p$1 -opid -opcpu -opmem -ocomm -c | \
11         grep $1 | sed "s/^/$mydate_/" >> $filepath/$2.log
12     sleep 1
13     mydate='date +%H:%M:%S'
14 done
```

Listing B.1: Shell script used to log CPU and memory usage of a process.



Unexplored Paths: the Coil Gun Display

Following the experience we had with the point cloud, we had the idea of building a real, as in the physical world, 3D point cloud. We did not follow this path so we cannot tell whether it is feasible or not, but the concept is relevant and we are going to explain it here. The core concept is to shoot particles vertically into the air and color them with projected light, or have self-illuminating particles. The chosen means to shoot the particles is coil guns. We briefly examined other techniques to move particles vertically – compressed air jets, springs, or other mechanical devices. The coil gun was the best candidate, with no moving parts and enough force to move a projectile to a human height.

Also, the coil gun is probably the most silent option among the examined ones. The coil gun display is arranged in modules of 1 m^2 (see Fig. C.1), each module containing a hundred coil guns. Each coil gun occupies a space $10\times 10\text{ cm}^2$ and comprises a pipe, around which the coil is wound, ending in a funnel. The funnel is supposed to receive the particle falling after it has been shot vertically, reloading the coil gun. If the particles in flight are lit at proper time with the right color the result is a colored point cloud not bound to a display.

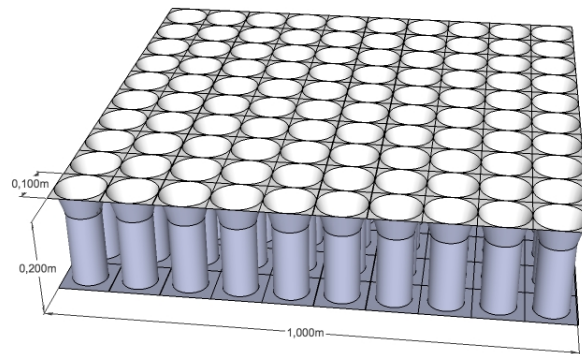


Figure C.1: Coil gun display module.

Bibliography

- [1] Verdione. Verdione Project. URL <http://krikkit.simula.no/>.
- [2] Skype. <http://www.skype.com/>. URL <http://www.skype.com/>.
- [3] Fei Su, J.M. Bjørndalen, Phuong Hoai Ha, and O.J. Anshus. Masking the effects of delays in human-to-human remote interaction. In *2014 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 719–728, September 2014. doi: 10.15439/2014F137.
- [4] Daniel Stødle. *Device-Free Interaction and Cross-Platform Pixel Based Output to Display Walls*. PhD thesis, Ph. d. thesis, Uni. of Tromsø, 2009.
- [5] HSI. HSI'13 6th International Conference on Human System Interaction, 2013. URL <http://www.hsi.wsiz.rzeszow.pl/>.
- [6] Giacomo Tartari. Global Interactin Space for user interaction with a room of computers. URL <https://www.youtube.com/watch?v=71TZ3p1gs14>.
- [7] Verdikt. VERDIKT-konferansen - VERDIKT, 2013. URL <http://www.forskningsradet.no/prognett-verdikt/VERDIKTkonferansen/1226993838234>.
- [8] PingER. PingER. URL <http://www-iepm.slac.stanford.edu/pinger/>.
- [9] Fei Su, Giacomo Tartari, John Markus Bjørndalen, Phuong Hoai Ha, and Otto J. Anshus. MultiStage: Acting across Distance. In Paolo Nesi and Raffaella Santucci, editors, *Information Technologies for Performing Arts, Media Access, and Entertainment*, number 7990 in Lecture Notes in Computer Science, pages 227–239. Springer Berlin Heidelberg, January 2013. ISBN 978-3-642-40049-0 978-3-642-40050-6. URL http://link.springer.com/chapter/10.1007/978-3-642-40050-6_20.
- [10] M. Piccardi. Background subtraction techniques: a review. In *2004 IEEE International Conference on Systems, Man and Cybernetics*, volume 4, pages

3099–3104 vol.4, October 2004. doi: 10.1109/ICSMC.2004.1400815.

- [11] A. A. Sawchuk, E. Chew, R. Zimmermann, C. Papadopoulos, and C. Kyriakakis. From Remote Media Immersion to Distributed Immersive Performance. In *Proceedings of the 2003 ACM SIGMM Workshop on Experiential Telepresence, ETP '03*, pages 110–120, New York, NY, USA, 2003. ACM. ISBN 1-58113-775-3. doi: 10.1145/982484.982506. URL <http://doi.acm.org/10.1145/982484.982506>.
- [12] Roger Zimmermann, Elaine Chew, Sakire Arslan Ay, and Moses Pawar. Distributed Musical Performances: Architecture and Stream Management. *ACM Trans. Multimedia Comput. Commun. Appl.*, 4(2):14:1–14:23, May 2008. ISSN 1551-6857. doi: 10.1145/1352012.1352018. URL <http://doi.acm.org/10.1145/1352012.1352018>.
- [13] Gotomeeting. gotomeeting. URL <http://www.gotomeeting.com/online/>.
- [14] Anthony Tang, Michel Pahud, Kori Inkpen, Hrvoje Benko, John C. Tang, and Bill Buxton. Three's company: understanding communication channels in three-way distributed collaboration. In *Proceedings of the 2010 ACM conference on Computer supported cooperative work, CSCW '10*, pages 271–280, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-795-0. doi: 10.1145/1718918.1718969. URL <http://doi.acm.org/10.1145/1718918.1718969>.
- [15] Viet Anh Nguyen, Tien Dung Vu, Hongsheng Yang, Jiangbo Lu, and Minh N. Do. ITEM: Immersive Telepresence for Entertainment and Meetings with Commodity Setup. In *Proceedings of the 20th ACM International Conference on Multimedia, MM '12*, pages 1021–1024, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1089-5. doi: 10.1145/2393347.2396372. URL <http://doi.acm.org/10.1145/2393347.2396372>.
- [16] Slim Essid, Xinyu Lin, Marc Gowing, Georgios Kordelas, Anil Aksay, Philip Kelly, Thomas Fillon, Qianni Zhang, Alfred Dielmann, Vlado Kitanovski, Robin Tournemenne, Aymeric Masurelle, Ebroul Izquierdo, Noel O'Connor, Petros Daras, and Gaël Richard. A multi-modal dance corpus for research into interaction between humans in virtual environments. *Journal on Multimodal User Interfaces*, pages 1–14. ISSN 1783-7677. doi: 10.1007/s12193-012-0109-5. URL <http://www.springerlink.com/content/p3170r2767630831/abstract/>.
- [17] O. Schreer, I. Feldmann, N. Atzpadin, P. Eisert, P. Kauff, and H. J. W. Belt. 3dpresence -A System Concept for Multi-User and Multi-Party Immersive 3d Videoconferencing. In *5th European Conference on Visual Media Produc-*

- tion (*CVMP 2008*), pages 1–8, November 2008. doi: 10.1049/cp:20081083.
- [18] Benjamin Petit, Jean-Denis Lesage, Clément Menier, Jérémie Allard, Jean-Sébastien Franco, Bruno Raffin, Edmond Boyer, and François Faure. Multicamera Real-Time 3d Modeling for Telepresence and Remote Collaboration. *International Journal of Digital Multimedia Broadcasting*, 2010: 1–12, 2010. ISSN 1687-7578, 1687-7586. doi: 10.1155/2010/247108. URL <http://www.hindawi.com/journals/ijdmb/2010/247108/abs/>.
- [19] I. Feldmann, W. Waizenegger, N. Atzpadin, and O. Schreer. Real-time depth estimation for immersive 3d videoconferencing. In *2010 3DTV-Conference: The True Vision - Capture, Transmission and Display of 3D Video*, pages 1–4, June 2010. doi: 10.1109/3DTV.2010.5506312.
- [20] Shahram Izadi, Richard A. Newcombe, David Kim, Otmar Hilliges, David Molyneaux, Steve Hodges, Pushmeet Kohli, Jamie Shotton, Andrew J. Davison, and Andrew Fitzgibbon. KinectFusion: Real-time Dynamic 3d Surface Reconstruction and Interaction. In *ACM SIGGRAPH 2011 Talks, SIGGRAPH '11*, pages 23:1–23:1, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0974-5. doi: 10.1145/2037826.2037857. URL <http://doi.acm.org/10.1145/2037826.2037857>.
- [21] Daisuke Sakamoto, Takayuki Kanda, Tetsuo Ono, Hiroshi Ishiguro, and Norihiro Hagita. Android as a telecommunication medium with a human-like presence. In *Human-Robot Interaction (HRI), 2007 2nd ACM/IEEE International Conference on*, pages 193–200. IEEE, 2007. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6251688.
- [22] A. Maimone and H. Fuchs. Encumbrance-free telepresence system with real-time 3d capture and display using commodity depth cameras. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 137–146, October 2011. doi: 10.1109/ISMAR.2011.6092379.
- [23] Weidong Huang, Leila Alem, and Franco Tecchia. HandsIn3d: Supporting Remote Guidance with Immersive Virtual Environments. In Paula Kotzé, Gary Marsden, Gitte Lindgaard, Janet Wesson, and Marco Winckler, editors, *Human-Computer Interaction – INTERACT 2013*, number 8117 in Lecture Notes in Computer Science, pages 70–77. Springer Berlin Heidelberg, September 2013. ISBN 978-3-642-40482-5 978-3-642-40483-2. URL http://link.springer.com/chapter/10.1007/978-3-642-40483-2_5. DOI: 10.1007/978-3-642-40483-2_5.
- [24] D.S. Alexiadis, D. Zarpalas, and P. Daras. Real-Time, Full 3-D Reconstruction of Moving Foreground Objects From Multiple Consumer Depth

Cameras. *IEEE Transactions on Multimedia*, 15(2):339–358, 2013. ISSN 1520-9210. doi: 10.1109/TMM.2012.2229264.

- [25] Ramanarayan Vasudevan, Gregorij Kurillo, Edgar Lobaton, Tony Bernardin, Oliver Kreylos, Ruzena Bajcsy, and Klara Nahrstedt. High-Quality Visualization for Geographically Distributed 3-D Teleimmersive Applications. *Multimedia, IEEE Transactions on*, 13(3):573–584, 2011. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5727958.
- [26] Pranav Mistry and Pattie Maes. SixthSense: A Wearable Gestural Interface. In *ACM SIGGRAPH ASIA 2009 Sketches, SIGGRAPH ASIA '09*, pages 11:1–11:1, New York, NY, USA, 2009. ACM. doi: 10.1145/1667146.1667160. URL <http://doi.acm.org/10.1145/1667146.1667160>.
- [27] F. P. M. Elgendi and N. Magenant-Thalman. Real-time speed detection of hand gesture using kinect. In *Workshop on Autonomous Social Robots and Virtual Humans, The 25th Annual Conference on Computer Animation and Social Agents (CASA 2012), Singapore*, 2012. URL <http://www.elgendi.net/papers/casaworkshop12.pdf>.
- [28] M. Van den Bergh and L. Van Gool. Combining RGB and ToF cameras for real-time 3d hand gesture interaction. In *2011 IEEE Workshop on Applications of Computer Vision (WACV)*, pages 66–72, January 2011. doi: 10.1109/WACV.2011.5711485.
- [29] T. Pederson, L. Janlert, and D. Surie. A Situative Space Model for Mobile Mixed-Reality Computing. *IEEE Pervasive Computing*, 10(4):73–83, 2011. ISSN 1536-1268. doi: 10.1109/MPRV.2010.51.
- [30] Lars C. Ebert, Gary Hatch, Garyfalia Ampanozi, Michael J. Thali, and Steffen Ross. You Can't Touch This Touch-free Navigation Through Radiological Images. *Surgical Innovation*, 19(3):301–307, September 2012. ISSN 1553-3506, 1553-3514. doi: 10.1177/1553350611425508. URL <http://sri.sagepub.com/content/19/3/301>.
- [31] Satoru Morishima, Tomohiro Mashita, Kiyoshi Kiyokawa, and Hiroshi Takemura. A waist-mounted ProCam system for remote collaboration. In *Mixed and Augmented Reality (ISMAR), 2012 IEEE International Symposium on*, pages 301–302. IEEE, 2012. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6402584.
- [32] Farzin Farhadi-Niaki, Reza GhasemAghaei, and Ali Arya. Empirical study of a vision-based depth-sensitive human-computer interaction system. In *Proceedings of the 10th asia pacific conference on Computer human*

- interaction*, APCHI '12, pages 101–108, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1496-1. doi: 10.1145/2350046.2350070. URL <http://doi.acm.org/10.1145/2350046.2350070>.
- [33] Sean Ryan Fanello, Ilaria Gori, Giorgio Metta, and Francesca Odone. Keep it simple and sparse: Real-time action recognition. *The Journal of Machine Learning Research*, 14(1):2617–2640, 2013. URL <http://dl.acm.org/citation.cfm?id=2567745>.
- [34] Bryce Kellogg, Vamsi Talla, and Shyamnath Gollakota. Bringing gesture recognition to all devices. In *Usenix NSDI*, volume 14, 2014. URL <https://www.usenix.org/system/files/conference/nsdi14/nsdi14-paper-kellogg.pdf>.
- [35] Mingsong Dou, Ying Shi, J. Frahm, H. Fuchs, B. Mauchly, and M. Marathe. Room-sized informal telepresence system. In *2012 IEEE Virtual Reality Short Papers and Posters (VRW)*, pages 15–18, March 2012. doi: 10.1109/VR.2012.6180869.
- [36] O. Anshus, Daniel Stødle, T. Hagen, Bård Fjukstad, J. Bjørndalen, L. Bongo, Yong Liu, and Lars Tiede. *Nine Years of the Tromsø Display Wall*. 2013.
- [37] Andrew D. Wilson and Hrvoje Benko. Combining multiple depth cameras and projectors for interactions on, above and between surfaces. In *Proceedings of the 23rd annual ACM symposium on User interface software and technology*, UIST '10, pages 273–282, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0271-5. doi: 10.1145/1866029.1866073. URL <http://doi.acm.org/10.1145/1866029.1866073>.
- [38] Garth Shoemaker, Takayuki Tsukitani, Yoshifumi Kitamura, and Kellogg S. Booth. Body-centric interaction techniques for very large wall displays. In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries*, NordiCHI '10, pages 463–472, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-934-3. doi: 10.1145/1868914.1868967. URL <http://doi.acm.org/10.1145/1868914.1868967>.
- [39] Andrew Bragdon, Rob DeLine, Ken Hinckley, and Meredith Ringel Morris. Code space: touch + air gesture hybrid interactions for supporting developer meetings. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, ITS '11, pages 212–221, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0871-7. doi: 10.1145/2076354.2076393. URL <http://doi.acm.org/10.1145/2076354.2076393>.

- [40] David Kim, Otmar Hilliges, Shahram Izadi, Alex D. Butler, Jiawen Chen, Iason Oikonomidis, and Patrick Olivier. Digits: freehand 3d interactions anywhere using a wrist-worn gloveless sensor. In *Proceedings of the 25th annual ACM symposium on User interface software and technology, UIST '12*, pages 167–176, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1580-7. doi: 10.1145/2380116.2380139. URL <http://doi.acm.org/10.1145/2380116.2380139>.
- [41] Martin Spindler, Marcel Martsch, and Raimund Dachsel. Going beyond the surface: studying multi-layer interaction above the tabletop. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems, CHI '12*, pages 1277–1286, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1015-4. doi: 10.1145/2208516.2208583. URL <http://doi.acm.org/10.1145/2208516.2208583>.
- [42] Jyh-Ming Lien, Gregorij Kurillo, and Ruzena Bajcsy. Multi-camera tele-immersion system with real-time model driven data compression. *The Visual Computer*, 26(1):3–15, May 2009. ISSN 0178-2789, 1432-2315. doi: 10.1007/s00371-009-0367-8. URL <http://link.springer.com/article/10.1007/s00371-009-0367-8>.
- [43] Zhenyu Yang, Wanmin Wu, Klara Nahrstedt, Gregorij Kurillo, and Ruzena Bajcsy. Enabling Multi-party 3d Tele-immersive Environments with ViewCast. *ACM Trans. Multimedia Comput. Commun. Appl.*, 6(2):7:1–7:30, March 2010. ISSN 1551-6857. doi: 10.1145/1671962.1671963. URL <http://doi.acm.org/10.1145/1671962.1671963>.
- [44] P. Fechteler, A. Hilsmann, P. Eisert, S. V. Broeck, C. Stevens, J. Wall, M. Sanna, D. A. Mauro, F. Kuijk, R. Mekuria, P. Cesar, D. Monaghan, N. E. O'Connor, P. Daras, D. Alexiadis, and T. Zahariadis. A Framework for Realistic 3d Tele-immersion. In *Proceedings of the 6th International Conference on Computer Vision / Computer Graphics Collaboration Techniques and Applications, MIRAGE '13*, pages 12:1–12:8, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2023-8. doi: 10.1145/2466715.2466718. URL <http://doi.acm.org/10.1145/2466715.2466718>.
- [45] Rufael Mekuria, Michele Sanna, Stefano Asioli, Ebroul Izquierdo, Dick C. A. Bulterman, and Pablo Cesar. A 3d Tele-immersion System Based on Live Captured Mesh Geometry. In *Proceedings of the 4th ACM Multimedia Systems Conference, MMSys '13*, pages 24–35, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1894-5. doi: 10.1145/2483977.2483980. URL <http://doi.acm.org/10.1145/2483977.2483980>.
- [46] Suraj Raghuraman, Karthik Venkatraman, Zhanyu Wang, Balakrishnan

- Prabhakaran, and Xiaohu Guo. A 3d Tele-immersion Streaming Approach Using Skeleton-based Prediction. In *Proceedings of the 21st ACM International Conference on Multimedia*, MM '13, pages 721–724, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2404-5. doi: 10.1145/2502081.2502188. URL <http://doi.acm.org/10.1145/2502081.2502188>.
- [47] Elaine Chew, Alexander Sawchuk, Carley Tanoue, and Roger Zimmermann. Segmental Tempo Analysis of Performances in User-Centered Experiments in the Distributed Immersive Performance Project. In *Proceedings of the Sound and Music Computing Conference, Salerno, Italy*, 2005. URL http://www.smc-conference.org/smc05/papers/ElanieChew/cstz-smc05_final.pdf.
- [48] Y. Sato, K. Hashimoto, and Y. Shibata. A New Remote Camera Work System for Teleconference Using a Combination of Omni-Directional and Network Controlled Cameras. In *22nd International Conference on Advanced Information Networking and Applications, 2008. AINA 2008*, pages 502–508, March 2008. doi: 10.1109/AINA.2008.153.
- [49] H. Aghajan and Chen Wu. Layered and Collaborative Gesture Analysis in Multi-Camera Networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing, 2007. ICASSP 2007*, volume 4, pages IV–1377–IV–1380, April 2007. doi: 10.1109/ICASSP.2007.367335.
- [50] A. Bellucci, A. Malizia, P. Diaz, and I. Aedo. Human-Display Interaction Technology: Emerging Remote Interfaces for Pervasive Display Environments. *IEEE Pervasive Computing*, 9(2):72–76, April 2010. ISSN 1536-1268. doi: 10.1109/MPRV.2010.30.
- [51] Andrew Bragdon and Hsu-Sheng Ko. Gesture select:: acquiring remote targets on large displays without pointing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '11*, pages 187–196, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0228-9. doi: 10.1145/1978942.1978970. URL <http://doi.acm.org/10.1145/1978942.1978970>.
- [52] Andrew Blakney. Fleshing out gestures: augmenting 3d interaction. In *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering, C3S2E '12*, pages 125–126, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1084-0. doi: 10.1145/2347583.2347602. URL <http://doi.acm.org/10.1145/2347583.2347602>.
- [53] S. Mitra and T. Acharya. Gesture Recognition: A Survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 37(3): 311–324, 2007. ISSN 1094-6977. doi: 10.1109/TSMCC.2007.893280.

- [54] V.I. Pavlovic, R. Sharma, and T.S. Huang. Visual interpretation of hand gestures for human-computer interaction: a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):677–695, July 1997. ISSN 0162-8828. doi: 10.1109/34.598226.
- [55] K. Sabir, C. Stolte, B. Tabor, and S.I. O’Donoghue. The Molecular Control Toolkit: Controlling 3d molecular graphics via gesture and voice. In *2013 IEEE Symposium on Biological Data Visualization (BioVis)*, pages 49–56, October 2013. doi: 10.1109/BioVis.2013.6664346.
- [56] Teddy Seyed, Chris Burns, Mario Costa Sousa, Frank Maurer, and Anthony Tang. Eliciting usable gestures for multi-display environments. In *Proceedings of the 2012 ACM international conference on Interactive tabletops and surfaces, ITS ’12*, pages 41–50, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1209-7. doi: 10.1145/2396636.2396643. URL <http://doi.acm.org/10.1145/2396636.2396643>.
- [57] WebSocket. <https://www.websocket.org/>. URL <https://www.websocket.org/>.
- [58] WebRTC. <http://www.webrtc.org/>. URL <http://www.webrtc.org/>.
- [59] LeapMotion. <https://www.leapmotion.com/>. URL <https://www.leapmotion.com/>.
- [60] Myo. <https://www.thalmic.com/en/myo/>. URL <https://www.thalmic.com/en/myo/>.
- [61] Snappy. [google/snappy](https://github.com/google/snappy). URL <https://github.com/google/snappy>.
- [62] Libfreenect. [OpenKinect/libfreenect](https://github.com/OpenKinect/libfreenect). URL <https://github.com/OpenKinect/libfreenect>.
- [63] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, August 1978. ISSN 0001-0782. doi: 10.1145/359576.359585. URL <http://doi.acm.org/10.1145/359576.359585>.
- [64] Netflix. Netflix Internet Connection Speed Recommendations. URL <https://help.netflix.com/en/node/306>.
- [65] Kenrick Kin, Björn Hartmann, Tony DeRose, and Maneesh Agrawala. Proton: multitouch gestures as regular expressions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI ’12*, pages 2885–2894, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1015-

4. doi: 10.1145/2207676.2208694. URL <http://doi.acm.org/10.1145/2207676.2208694>.
- [66] OpenGL. OpenGL. URL <https://www.opengl.org/>.
- [67] Vulkan. Vulkan. URL <https://www.khronos.org/vulkan>.
- [68] Metal. Apple Metal. URL <https://developer.apple.com/metal/>.
- [69] Mantle. AMD Mantle.
- [70] Horde3D. Horde3d next generation graphics engine. URL <http://www.horde3d.org/>.
- [71] G. Tartari, Daniel Stodler, J.M. Bjorndalen, Phuong Hoai Ha, and O.J. Anshus. Global interaction space for user interaction with a room of computers. In *2013 The 6th International Conference on Human System Interaction (HSI)*, pages 84–89, 2013. doi: 10.1109/HSI.2013.6577806.
- [72] Annica Kristoffersson, Silvia Coradeschi, and Amy Loutfi. A Review of Mobile Robotic Telepresence. *Adv. in Hum.-Comp. Int.*, 2013:3:3–3:3, January 2013. ISSN 1687-5893. doi: 10.1155/2013/902316. URL <http://dx.doi.org/10.1155/2013/902316>.
- [73] Unity. <http://unity3d.com>. URL <http://unity3d.com>.
- [74] Unreal. <https://www.unrealengine.com>. URL <https://www.unrealengine.com>.
- [75] Project Tango. <https://www.google.com/atap/projecttango/#project>. URL <https://www.google.com/atap/projecttango/#project>.
- [76] Movidius. <http://www.movidius.com/>. URL <http://www.movidius.com/>.
- [77] ozo. Nokia OZO | Virtual Reality Camera with 360-degree audio and video capture. URL <https://ozo.nokia.com/>.
- [78] Oculus. <http://www.oculus.com/>. URL <http://www.oculus.com/>.
- [79] Go. <http://golang.org/>.
- [80] Openni. <http://structure.io/openni>. URL <http://structure.io/openni>.
- [81] Openkinect. <http://openkinect.org>. URL <http://openkinect.org>.

[82] 36ofly. 36ofly. URL <https://36ofly.com/>.

[83] ZED. URL <https://2013.asiabsdcon.org/papers/abc2013-P7A-paper.pdf>.

Papers