

TECHNICAL UNIVERSITY OF CRETE, GREECE
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

ARCHIMUSIC3D: Real-Time Graphical System for Reciprocal Transformations between Music and Refined Urban Architectural Design



Fotios Giariskanis

Thesis Committee

Associate Professor Katerina Mania (ECE)

Associate Professor Georgios Karistinos (ECE)

Associate Professor Panagiotis Parthenios (ARCH)

Chania, November 2018

Abstract

An urban designer needs advanced analytical tools in order to understand the inner-relations and functions that describe contemporary cities. Many of the recently digitally-crafted architectural design tools for urban environments fail to address the need for a more holistic design approach that captures virtual and physical, immaterial and material including aspects of an urban design which are aesthetic as well as functional. There is a significant potential in combining the fields of composition in music and architecture through the use of information technology. Merging the two fields has the potential to release new, innovative tools for urban designers by uncovering aspects of an urban design not apparent based on the standard, vision-based digital design tools, introducing the additional modality of sound and music. This thesis describes an innovative tool developed at the Technical University of Crete, through which an urban designer can work on the music transcription of a specific urban environment applying music compositional rules and filters in order to identify discordant entities, highlight imbalanced parts and make design corrections.

This thesis presents Archimusic3D, a platform implemented in the Unity development engine that offers sonification of an Urban Virtual Environment (UVE), simulating a real-world cityscape and offering visual interpretation and interactive modification of its soundscape. The system presented in this thesis offers: First of all, the ability to view and convert an urban street to music (ready to play) based on a specific grammar of converting architectural elements to musical elements, secondly, the ability to transform this music in order to harmonize it based on musical rules and finally, the prospect of converting back the aesthetically and harmonically "corrected" musical piece to a newly refined street or urban design.

The presented platform comprises of three scenes, which compile the three main parts of the system's interface; e.g., the 3D scene, the Digital Audio Workstation (DAW) scene and the TouchOSC mobile controller. While interacting with the 3D scene, the user is able to navigate a 3D street in order to observe the 3D graphical depiction of an urban environment. The second scene is the implementation of a Digital Audio Workstation (DAW) program, in which the user is able to see the musical footprint of the street printed as a MIDI (Musical Instrument Digital Interface) file in order to listen to it and process it. Processing is conducted via a third tool representing a TouchOSC mobile controller, which is a virtual mobile console containing faders, potentiometers and buttons. When the users

converts the 3D scene to a musical footprint, they submit the reciprocal transformations which may be applied to the musical part and then, they are able to navigate a newly refined street.

Based on a grammar provided by the Digital Media Lab of the School of Architecture, Technical University of Crete which connects musical with architectural elements, the result is the musical footprint of a street. The basic rules for this translation are: the height of an element is translated to the pitch of a note, the length of the element is translated to the duration of the note, and the depth of the element is translated to the volume of the note. The piano is associated with a window, the flute with a tree, the xylophone with a balcony, the bass with a door, the cello with a building and the violin with a larger scale building. The purpose of this thesis is to develop a tool which utilizes this translation method between music and architecture for compositional experiments on urban design in order to assist designers in 1) identifying urban dissonances, 2) refining their design using musical rules and 3) presenting the output both visually and acoustically.

Acknowledgements

First of all, i would like to thank my Supervisor, Associate Professor Katerina Mania for her advice, support and supervision throughout this whole process but also for trusting me and giving me the opportunity to undertake a project that involving imagination and creativity.

Next I would like to thank, Assistant Professor Panagiotis Parthenios for his invaluable knowledge in the historical context of this project. I would like to thank them for their guidance and support throughout this work and beyond that.

Also, i would like to thank my family and my close friends for their infinite support and patience.

Contents

1	Introduction	1
1.1	Brief Description	1
1.2	Methodology	4
1.3	Purpose of the Thesis	6
1.4	Structure of the Thesis	8
2	Background	9
2.1	Introduction	9
2.2	Historical Context	9
2.3	Composition	12
2.3.1	Counterpoint	13
2.3.2	Digital Audio Workstation (DAW)	14
2.4	Protocols	15
2.4.1	Open Sound Control Protocol (OSC)	15
2.4.2	Musical Instrument Digital Information Protocol (MIDI)	16
2.5	Software	17
2.5.1	Unity	17
2.5.2	TouchOsc	18
2.5.3	UniOsc	18
3	Requirements Analysis	21
3.1	Introduction	21
3.1.1	General Requirements	22
3.2	Use Case Scenarios	23
3.2.1	Urban Street View	24

CONTENTS

3.2.2	Digital Audio Workstation View	24
3.2.3	Traslation	26
3.3	Manual of the System	27
3.3.1	Manual of the DAW interface	28
3.3.2	Manual of the TouchOSC	39
4	Implementation	49
4.1	Introduction	49
4.2	Flow diagram	49
4.3	Building Information Model (BIM)	51
4.3.1	REVIT	52
4.4	Grammar	54
4.5	Components	56
4.5.1	Transform	56
4.5.2	AudioSource	57
4.5.3	Scripts	58
4.6	TouchOsc Controller	74
5	Conclusion	81
5.1	Demo	81
5.2	Summary	85
5.3	Evaluation	86
5.3.1	Advantages	86
5.3.2	Disadvantages	87
5.4	Future Work	87
	References	92

List of Figures

1.1	Archimusic3D	3
1.2	Methodology	5
2.1	Philips Pavilion/Metastaseis B, Iannis Xenakis (1953-1958).	11
2.2	Monastery Sainte Marie de la Tourette (1956-1957)	11
2.3	Composition in Design	12
2.4	Composition in Music	13
2.5	Digital Audio Workstation	14
2.6	OSC protocol	15
2.7	MIDI protocol	16
2.8	Unity platform for 3D interactive applications	17
2.9	TouchOSC	18
2.10	UniOSC	19
3.1	Street View	24
3.2	Daw Program View	24
3.3	Six storey building is converted to A note	26
3.4	Balcony of fourth floor is converted to F note	27
3.5	Part of the Digital Audio Workstation View	28
3.6	Play Button	28
3.7	Pause Button	29
3.8	Stop Button	29
3.9	Mute Button	29
3.10	New Balcony Button	30
3.11	New Window Button	30

LIST OF FIGURES

3.12 New Door Button	32
3.13 New Tree Button	33
3.14 New Larger Scale Building Button	34
3.15 New One-storey Building Button	35
3.16 New Two-storey Building Button	36
3.17 New Three-storey Building Button	36
3.18 New Four-storey Building Button	37
3.19 New Five-storey Building Button	38
3.20 New six-storey Building Button	39
3.21 TouchOsc Archimusic3D Layout	40
3.22 Position / Time TouchOsc Fader	41
3.23 Duration / Length TouchOsc Fader	42
3.24 Note / Height TouchOsc Fader	43
3.25 Volume / Depth TouchOsc Fader	44
3.26 Delete Note Button	45
3.27 Generate Note Button	45
3.28 Bar / Camera Position TouchOsc Fader	46
3.29 Bpm / Bar Speed TouchOsc Potentiometer	47
3.30 Camera's Height TouchOsc Fader	48
4.1 Flow Diagram	50
4.2 Building Information Model	51
4.3 Autodesk Revit	52
4.4 3D Objects from Revit	53
4.5 3D Objects in REVIT	54
4.6 Translation concept	55
4.7 Translation example	56
4.8 Tranform Component	57
4.9 Audiosource Component	58
4.10 Script Component	59
4.11 Position / Time Script	60
4.12 Length / Duration Script	61
4.13 Height / Pitch Script	62

LIST OF FIGURES

4.14	Depth / Volume Script	64
4.15	Delete Script	65
4.16	Generate Script	66
4.17	Mute Left Script	67
4.18	Mute Right Script	68
4.19	Music Script	69
4.20	Mouse Manager Script	70
4.21	Bar Position Script	71
4.22	Bar Speed Script	72
4.23	Camera's height Script	73
4.24	Archimusic3D Template TouchOSC Editor	74
4.25	position x /time fader	75
4.26	duration / length fader	76
4.27	note / height fader	77
4.28	volume / depth fader	77
4.29	delete note button	78
4.30	generate note button	78
4.31	bar / camera position fader	79
4.32	bpm / bar speed potentiometer	79
4.33	camera's height fader	80
5.1	Demo - before the Transformations	83
5.2	Demo - after the Transformations	84

LIST OF FIGURES

Chapter 1

Introduction

1.1 Brief Description

Contemporary cities are live mechanisms, eco-systems, which perform under certain rules often not visible to the human eye at first sight of their architectural design. An urban designer needs advanced analytical tools in order to understand the inner-relations and functions that describe these eco-systems [1]. Many of the recently digitally-crafted architectural design tools for urban environments fail to address the need for a more holistic design approach that captures virtual and physical, immaterial and material including aspects of an urban design which are aesthetic as well as functional. There is a significant potential in combining the fields of composition in music and architecture through the use of information technology. Merging the two fields has the potential to release new, innovative tools for urban designers by uncovering aspects of an urban design not apparent based on the standard, vision-based digital design tools, introducing an additional modality of sound and music [2]. This thesis describes an innovative tool developed at the Technical University of Crete, through which an urban designer can work on the music transcription of a specific urban environment applying music compositional rules and filters in order to identify discordant entities, highlight imbalanced parts and make design corrections.

Acoustic data encoded from the built environment provides a valuable platform on which discordant entities can be more easily identified and also imbalanced parts get highlighted. The cognitive process of analyzing today's chaotic urban eco-system has been augmented with a new dimension of understanding but also intervening through

1. INTRODUCTION

its musical footprint [2]. The purpose of this thesis is a platform, implemented in the Unity development engine that offers sonification of an urban virtual environment (UVE), simulating a real-world cityscape and offering visual interpretation and interactive modification of its soundscape. The system presented in this thesis offers: First of all, the ability to view and convert an urban street to music (ready to play) based on a specific grammar of converting architectural elements to musical elements, secondly, the ability to transform this music in order to 'harmonize' it based on musical rules and finally, the prospect of converting back the aesthetically and harmonically 'corrected' musical piece to a newly refined street or urban design.

The presented platform comprises of three scenes, which compile the three main parts of the system's interface; e.g., the 3D scene, the Digital Audio Workstation (DAW) scene and the TouchOsc mobile controller (Figure 1.1) upper and lower part respectively). While interacting with the 3D scene, the user is able to navigate a 3D street in order to observe the 3D graphical depiction of an urban environment. The second scene is the implementation of a Digital Audio Workstation (DAW) program, in which the user is able to see the musical footprint of the street printed as a MIDI (Musical Instrument Digital Interface) file in order to listen to it and process it. Processing is conducted via a third tool representing a TouchOsc mobile controller, which is a virtual mobile console containing faders, potentiometers and buttons. When the users converts the 3D scene to a musical footprint, they submit the reciprocal transformations which may be applied to the musical part and then, they are able to navigate a newly refined street. At first look, music and architecture seem unrelated, but they share features, which relate to both architectural and musical composition. The first step is to match the parameters and architectural elements of the urban eco-system to those of its musical footprint. Based on a grammar [2] provided by the Digital Media Lab of the School of Architecture, Technical University of Crete which connects musical with architectural elements, the result is the musical footprint of a street.

The presented platform named Archimusic3D (Figure 1.1) developed in the Unity platform features a design tool that provides the visualization of the 3D urban environment of an urban street and the visualization of its musical footprint ready for reciprocal transformations between architectural and musical elements and vice versa, based on the designer's intuition, aesthetics as well as specified musicology rules. For example, in relation to a building's window, the architectural axes of the window are converted into

1.1 Brief Description



Figure 1.1: Archimusic3D

the musical axes of the piano. For instance, the height of the window is converted to the pitch which the piano plays, the length of the window is converted to the duration that the piano is playing this pitch and the depth of the window is converted to the volume that the piano is playing this pitch. The piano is associated with a window, the flute with a tree, the xylophone with a balcony, the bass with a door, the cello with a building and the violin with a larger scale building. While users interact with the second scene representing an implementation of a Digital Audio Workstation (DAW) program,

1. INTRODUCTION

the user can play and stop the sounds of the musical footprint, select and modify the musical parameters such as the time, the volume and the pitch of any instrument's note. At the same time that the user selects and modifies a note, the transformations of the musical parameters of this note are translated, in real-time, to transformations of the architectural parameters of the corresponding 3D object. Furthermore, the user has the ability to add a new note played by any music instrument and by extension a new 3D object.

1.2 Methodology

The methodology analyzed in this thesis expands the hearing experience of the urban environment by marking its basic spatial elements and transforming them to sounds. Using the philosophy behind Xenakis UPIC [3] system as a starting point, we have developed a translation method according to which geometrical data is translated to sounds. Street facades, the fundamental imprint of our urban environment, are first broken down to their main semantic elements. These elements have properties, such as position and size in a 3D (XYZ) system, which are transcribed into sonic data: length in the X-axis is mapped to note appearance in time and note duration (tempo), height in the Y-axis is mapped to note value (pitch) and depth in the Z-axis is mapped to volume (Figure 1.2). Different elements correspond to different timbre and voids to pauses (silence). As mentioned before, the piano is associated with a window, the flute with a tree, the xylophone with a balcony, the bass with a door, the cello with a building and the violin with a larger scale building.

The objects, such as buildings, which make up an urban street, are converted to notes played by musical instruments, such as the piano, which make up a music song and conversely. Any given path of an urban setup can be marked in order to create its soundscape with sounds produced by selected musical instruments. A simulation of an urban environment is created including urban elements fundamental for "reading". Building blocks are provided by the system and external elements can be included.

The application supports five types of architectural elements utilized to build a UVE and consequently for generating sound. These elements are "Buildings", "Larger Scale Buildings", "Windows", "Balconies", "Doors" and "Trees". One can use these elements

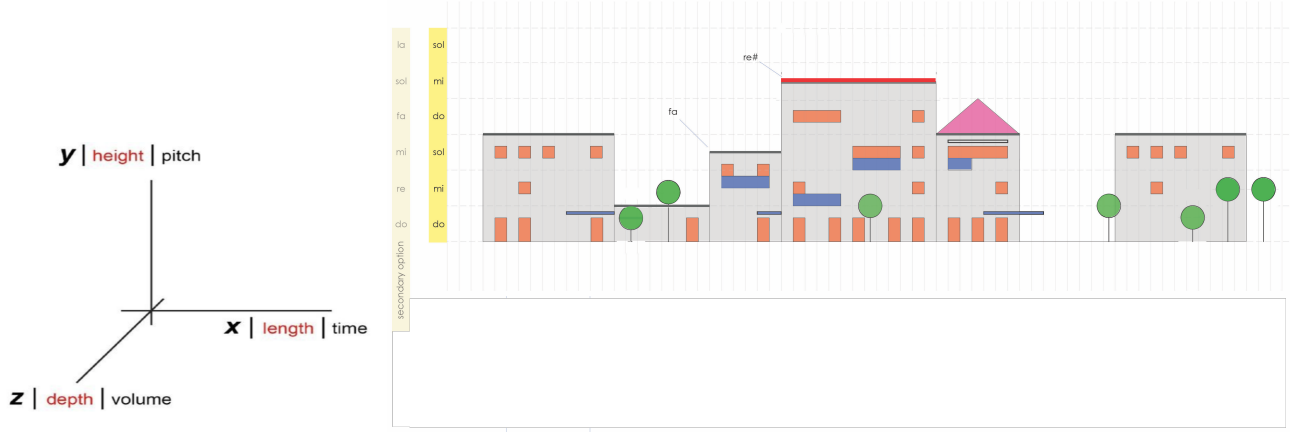


Figure 1.2: Methodology

to build complex urban environments including architectural elements which can be transformed to a sound. In this thesis, we have made the decision to associate the piano with the window, the flute with the tree, the xylophone with the balcony, the bass with the door, the cello with the building and the violin with the larger scale building. These elements are 3D shapes and as such have the following properties: height, length and depth. As a basic unit for mapping, height to note value is the "FLOOR" on which the object is located. The floor is considered to be 3 m high. The first floor is mapped to C, the second to D, the third to E, and so on. As for the length, 1 m is mapped to 1 sec note duration. For example, if we have a building that is 10 m high and 15 m long, it would be translated to the F note with a duration of 15 secs. Respectively, a building or a balcony that is protruding will sound more intense and one that is recessed will have a lower volume, thus mapping the perception that it is further away [2].

The notes take values from the C-major scale which is the most common key signature in the Western music. In order to map architecture to music, the user specifies paths (lines) in the scene that he or she wants to hear. The sound imprint of the selected objects in the Virtual Environment (VE) is created by scanning the path in which they are located. As the scanning progresses, the notes and sound parameters that represent the path's architectural objects are written as MIDI messages in a simulation of a MIDI file. Every type of architectural object is mapped to a channel in the MIDI file and is played by a different music instrument. Then, the file is opened in an MIDI editor for modifying and the architectural impact of the scanned path is visible in real time.

1. INTRODUCTION

The Archimusic3D platform also supports the reverse translation: music to architecture. This is achieved using the channel information of the music score and the time to spatial relationship that exists between notes and 3D objects. Based on the channel or instrument information, buildings, openings, and other architecture elements are created and based on the note value, duration and volume, the height, length, and depth are set. The algorithm can be explained as follows: first, find the type of instrument that the notes belong to in order to find the architectural group and then scan the MIDI file from left to right to match the beginning and the ending of the sound path.

1.3 Purpose of the Thesis

Since in most cases the current image of our cities is not the result of a single, coherent, decision making process, there is an opportunity with the proposed methodology to apply meaning to chaos: in other words, to design. By highlighting the strong inner values and relationships between the prime particles (atoms, bits and notes) of an eco-system and by eliminating the alien interferences, this platform provides valuable tools to assist designers in preserving the eco-systems viability and originality. Furthermore, this eco-systemic methodology has the potential to reveal key patterns, not visible to the human eye, which can then be further analyzed and re-used in attempts to create new eco-systems from scratch.

The research has shown that there is significant potential in trying to apply compositional techniques from the field of music to the field of urban design with the use of information technology. The purpose of this thesis is an innovative design tool which exploits music's compositional rules for the design of refined urban environments. This platform supports a more holistic design approach based on the generation and the processing of the musical footprint of an "chaotic" urban environment, in order to 'harmonize' it based on musicology rules.

The proposed method needs to be tested in specific case studies (take certain streets from different cities and translate them into sounds) in order to calibrate the system and validate the method. These case studies can be followed by a series of surveys in order to get feedback from citizens on issues such as harmony, beauty and tense, as well as on which version of the facade they prefer. Having the ability to edit the acoustic imprint of an urban environment and experiment with different music composition rules and filters

provides urban designers an extended, augmented, cognition level for tuning the result and eliminate discordant elements. The result of this processing is a new elegant song and at the same time a new refined 3D urban street. Furthermore, famous music songs can be converted into 3D neighbourhoods.

1.4 Structure of the Thesis

Chapter 1 introduces a brief description of Archimusic3D platform's purpose and functionality.

Chapter 2 analyzes the historical context in relation to architectural design, theory of music and their combination (Archimusic3D). It focuses on the concept of composition in both music and architecture. It analyzes the technical tools, libraries and protocols employed in this thesis such as the Digital Audio Workstation, the Open Sound Control Protocol (OSC), the Musical Instrument Digital Information Protocol (MIDI), the Unity engine and libraries employed for the programmability of 3D interactive applications, the TuschOSC modular OSC and MIDI control surface for iOS and Android and the UniOSC tool.

Chapter 3 presents the user requirements' gathering and use cases illustrating the possible interactions between the user and the platform. The three main interface scenes of the platform are described, namely the 3D scene, the Digital Audio Workstation View and the TouchOSC mobile controller View. A summary of the grammar for converting architectural elements to sound is offered.

Chapter 4 describes in detail the technical implementation of the Archimusic3D platform presented in this thesis. This chapter presents the final design of the platform's User Interface and the technical methodology which allowed the required functionality to be incorporated in the system employing the protocols and technical tools described in Chapter 2.

Chapter 5 offers a summary of Archimusic3D. It also provides an expert's evaluation of the system as well suggestions for future improvements.

Chapter 2

Background

2.1 Introduction

In order to understand in depth the ways in which architecture and music can be connected and combined, both as sciences and as arts, someone should first study each of these two, regarding terminology, rules, and the technical tools used for architecture and musical composition. It also essential to the study the attempts and the approaches that have been adopted by both artists and scientists, in order to connect music with architecture.

2.2 Historical Context

Spatial mental knowledge processing is a research area quite favourable by many theoreticians, urban planners and architects who are interested in finding ways to effectively represent the "reading" of the city. Whether it is called spatial images (Lynch), spatial schemata (Lee), mental maps (Gould and White), spatial mental models (Tversky) or cognitive collages (Tversky) the hunt for a successful vocabulary for the city's form has been more frequently based on empiricism. With the shift in the 60s and 70s by Kevin Lynch and Christopher Alexander towards a more human articulation of the city, a posteriori studies took the lead and researchers were prompted to base their findings on induction rather than deduction. Kevin Lynch introduced the concept of place legibility, which refers to the coherence of a place through its distinctive characteristic parts. Lynch

2. BACKGROUND

proposed five types of elements which are essential in order to configure a city's imageability an other concept by Lynch: paths (linear elements on which the observer usually moves), edges (used to separate one area from another), districts (main components of the city with common characteristics), nodes (strategically placed points where people meet or use in order to commute) and landmarks (similar to nodes but highly detectable, they help people to navigate and orient themselves in the city) [4].

There are common music patterns of translated neighbourhood facades between significantly different neighbourhoods of different cities, which suggest a deeper inner relation not so much in a quantitative level as in a more conceptual, qualitative, perhaps even emotional level of spatial perception. Compared urban sections share common characteristics in terms of tonality, rhythm or pauses revealing types of urban soundscapes which function independently from their geometrical values or the typical architectural vocabulary. Emerged similarities urge for further research in order to investigate whether the seemingly divergent parts of a city or separate cities share tangible common qualities which are able to affect the values that determine the livability of a contemporary urban environment [1].

Markos Novak [5] invents the term *archimusic*, in order to describe the art and science that results from the conflation of architecture to music, and is to visualization as knowledge is to information. In the context of cyberspace and artificial nature, Novak makes a distinction between the *archimusic* within information and the *archimusic* of information: *Archimusic* within information is architecture and music combined and extended to exist within virtual worlds, but still investigating the preoccupations of our familiar worlds. *Archimusic* of information is the architecture and music that is directly implicit in the artificial nature of cyberspace. Novak's notion that architecture and music are freed from their limitations to matter and sound, respectively, is in reference to Iannis Xenakis' view of music "inside-time" and music "outside-time," as presented in his book *Formalized Music* [6]. The dual proficiency of Xenakis in the fields of music and architecture, as well as his collaboration with Le Corbusier, led to explorations and innovations in composing in space and time. Famous examples include the Philips Pavilion for Expo 1958 in Brussels (Figure 2.1) [7] and the design and construction of the monastery Sainte Marie de la Tourette (Figure 2.2) [8] as explained below (Figure 2.2).

The Philips Pavilion, by Xenakis and Le Corbusier, consisted of nine hyperbolic paraboloids instead of flat surfaces and the music was spatialized. Thus, the concept

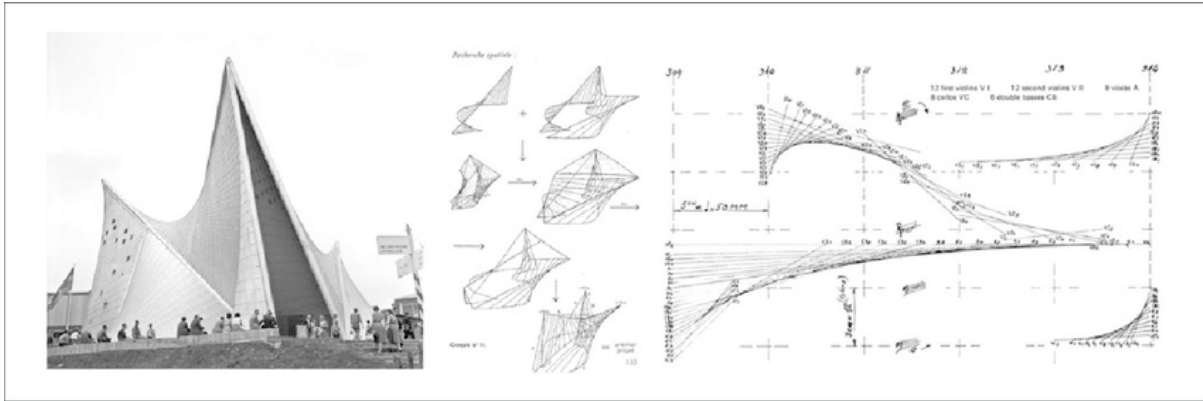


Figure 2.1: Philips Pavilion/Metastaseis B, Iannis Xenakis (1953-1958).



Figure 2.2: Monastery Sainte Marie de la Tourette (1956-1957)

of virtual and real space emerged. As basis for the design of the pavilion, Xenakis used his work *Metastasis*, which was based on the designs and mathematical proportions given by Le Corbusier's *Modulor*, in order to compose the micro- and macrostructure of the

2. BACKGROUND

piece in time. The overall structure and the control of elements, such as the massive glissandi, resulted in the idea of the hyperbolic paraboloids used in the pavilion. In the monastery of La Tourette, Xenakis' detailed work on rhythm proved crucial in designing the undulating panes of the facade. This was achieved by interpreting them in terms of the density of the inner structure of the panes, thus, achieving a contrapuntal texture. Despite the fact that we often think of architecture as material and music as immaterial, we should reconsider the relationship of these two sister arts through a more holistic approach, liberating them from the strict blinders that Western civilization has endowed us. The challenge is to embrace the concept that architecture can lie beyond buildings and music beyond sound.

2.3 Composition

The word composition comes from the Latin *componere*, meaning "put together" and its meaning remains close to this. Writing classes are often called composition classes and

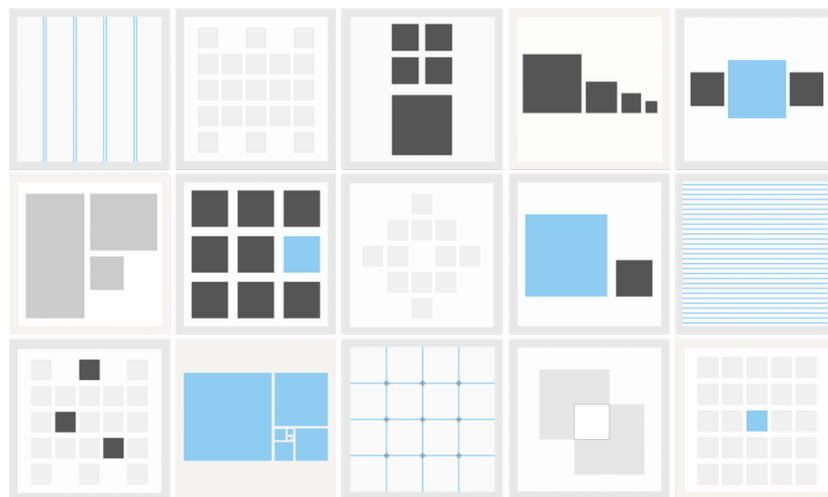


Figure 2.3: Composition in Design

writing music is also called composition. This can also describe things besides writing that are "put together (Figure 2.3)." Someone could say an abstract painting has an interesting composition. Any mixture of ingredients can be called a composition. Geologists study

the composition of the earth: what it's made of and how it formed. In the visual arts, composition is the placement or arrangement of visual elements or 'ingredients' in a work of art, as distinct from the subject. It can also be thought of as the organization of the elements of art. Musical composition can refer to an original piece of music, either a song or an instrumental music piece, the structure of a musical piece, or the process of creating or writing a new song or piece of music. The word "song" is widely misused by people in the popular music industry to describe any musical composition, whether sung or played only by instruments [9].

2.3.1 Counterpoint

In music, counterpoint is the relationship between voices that are harmonically interdependent (polyphony) yet independent in rhythm and contour (Figure 2.4) [10]. It has



Figure 2.4: Composition in Music

been most commonly identified in the European classical tradition, strongly developing during the Renaissance and in much of the common practice period, especially in the Baroque. The term originates from the Latin *punctus contra punctum* meaning "point against point". Counterpoint has been used to designate a voice or even an entire composition. Counterpoint focuses on melodic interaction only secondarily on the harmonies produced by that interaction. In the words of John Rahn: It is hard to write a beautiful

2. BACKGROUND

song. It is harder to write several individually beautiful songs that, when sung simultaneously, sound as a more beautiful polyphonic whole. The internal structures that create each of the voices separately must contribute to the emergent structure of the polyphony, which in turn must reinforce and comment on the structures of the individual voices. The way that is accomplished in detail is 'counterpoint'. Counterpoint theory has been given a mathematical foundation in the work initiated by Guerino Mazzola. In particular, his model gives a structural (and not psychological) foundation of forbidden parallels of fifths and the dissonant fourth. The model has also been extended to microtonal contexts by Octavio Agustin [11].

2.3.2 Digital Audio Workstation (DAW)

The Digital Audio Workstation is the the modern descendant of the counterpoint. A

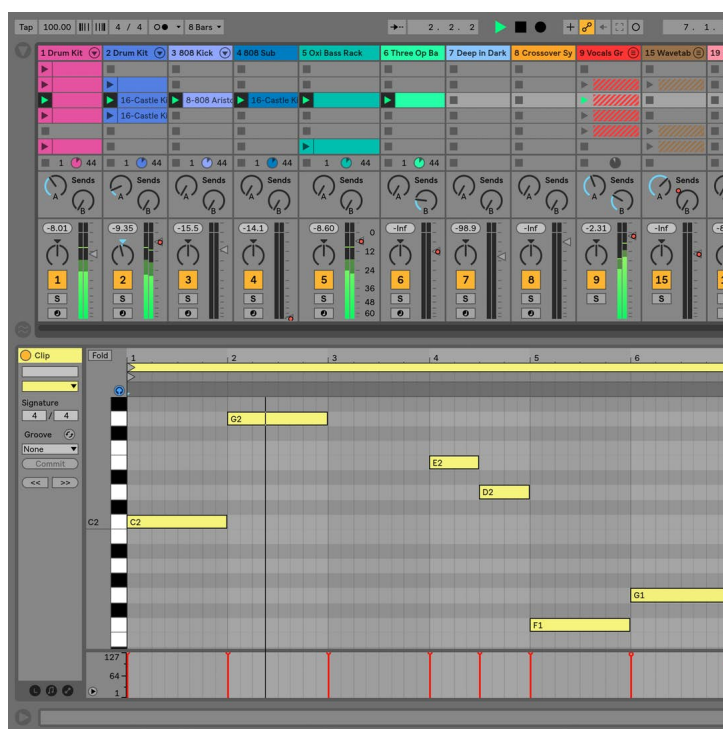


Figure 2.5: Digital Audio Workstation

Digital Audio Workstation (DAW)(Figure 2.5) [12] is an electronic device or application

software used for recording, editing and producing audio files. DAWs come in a wide variety of configurations from a single software program on a laptop, to an integrated stand-alone unit, all the way to a highly complex configuration of numerous components controlled by a central computer. Regardless of configuration, modern DAWs have a central interface that allows the user to alter and mix multiple recordings and tracks into a final produced piece. DAWs are used for the production and recording of music, songs, speech, radio, television, soundtracks, podcasts, sound effects and nearly any other situation where complex recorded audio is needed [13].

2.4 Protocols

In this thesis, the following protocols were implemented to create a platform which is simulating a DAW program.

2.4.1 Open Sound Control Protocol (OSC)

Open Sound Control (OSC) (Figure 2.6) [14] is a protocol for networking sound synthesizers, computers and other multimedia devices for purposes such as musical performance or show control.

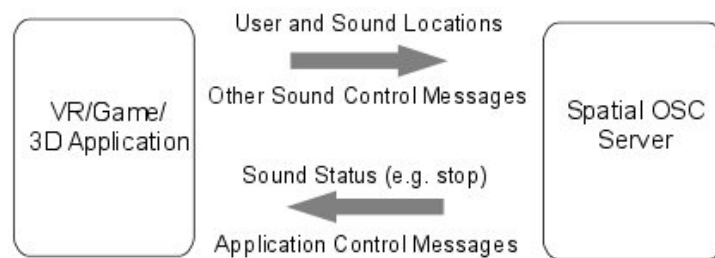


Figure 2.6: OSC protocol

OSC's advantages include interoperability, accuracy, flexibility and enhanced organization and documentation. In this thesis, the OSC protocol is implemented to provide to the user the ability to make transformations on either music's or the architecture's parameters [15]. The platform should provide, to the user, the ability to apply transformations on musical features of the notes (such as the time, duration, pitch, volume) and to apply

2. BACKGROUND

transformations on architectural features of the corresponding objects (such as the position, length, height, depth). In this thesis, for the implementation of the OSC protocol, the TouchOsc mobile application representing the transformations' controller and the UniOSC Unity's asset for connecting the TouchOsc mobile application with Unity, are combined.

2.4.2 Musical Instrument Digital Information Protocol (MIDI)

Musical Instrument Digital Interface(MIDI)(Figure 2.7) [16] is a technical standard that describes a communications protocol, digital interface, and electrical connectors that connect a wide variety of electronic musical instruments, computers, and related audio devices. MIDI carries event messages that specify notation, pitch, velocity, vibrato,

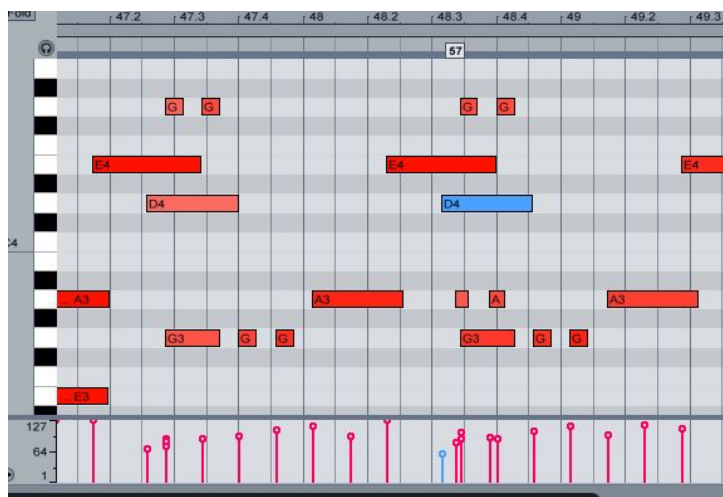


Figure 2.7: MIDI protocol

panning, and clock signals which set tempo. For example, a MIDI keyboard or other controller might trigger a sound module to generate sound produced by a keyboard amplifier. MIDI data can be transferred via MIDI cable, or recorded to a sequencer to be edited or played back. A file format that stores and exchanges the data is also defined. Advantages of MIDI include small file size, ease of modification and manipulation and a wide choice of electronic instruments and synthesizer or digitally-sampled sounds [17]. In this thesis, the MIDI protocol is implemented to provide the visualization of a musical

song. In this thesis, when referring to the visualization of a musical song we mean the actual visualization of the notes which are played by different musical instruments which make up the musical footprint. This musical footprint results from translating an urban street or area to a musical piece based on the translation grammar specified in the system.

2.5 Software

In this thesis, in order to implement the Archimusic3D platform and the protocols mentioned earlier, the below programming platforms and libraries are combined.

2.5.1 Unity

Unity(Figure 2.8) is a cross-platform environment for the development and programmability of interactive 3D graphics applications developed by Unity Technologies. Unity



Figure 2.8: Unity platform for 3D interactive applications

offers users the ability to create interactive applications in both 2D and 3D. The platform offers a primary scripting API in Csharp which is the primary programming language used for the engine. It also offers visual tools for easy tasks such as importing geometry based on drag and drop functionality, mainly meant for non-technical users [18]. In this thesis, the technical implementation of the Archimusic3D platform is based on the advanced libraries offered by the Unity engine, employed through Csharp. When referring to the Archimusic3d implementation, we mean the actual implementation and visualization of the two out of the three distinct parts of the user interface, e.g., the Street View and DAW View and their functionality (scripts, buttons, audiosources, tranform) as analyzed in Section 3.2.

2. BACKGROUND

2.5.2 TouchOsc

TouchOSC(Figure 2.9) is a modular OSC and MIDI control surface for iOS and Android.

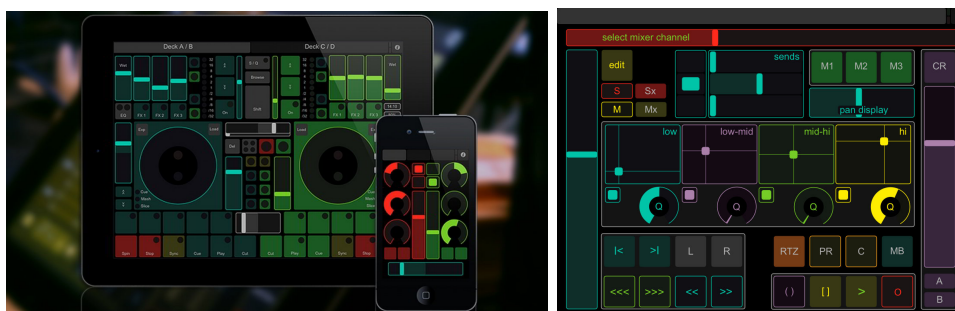


Figure 2.9: TouchOSC

It supports sending and receiving messages according to the Open Sound Control and MIDI protocols [19]. In this thesis, the designer uses the TouchOSC mobile application in order to apply transformations on either the music's or the architecture's parameters. The TouchOSC mobile app is the third distinct part of the Archimusic3D platform. It provides to the user, the ability to apply transformations to the musical features of the notes (such as the time, duration, pitch, volume) and to the architectural features of the corresponding objects (such as the position, length, height, depth). In this thesis, for the implementation of the OSC protocol in Unity, the TouchOsc mobile application representing the transformations' controller and the UniOSC Unity's asset (for connecting the TouchOsc mobile application with Unity) are combined.

2.5.3 UniOsc

UniOSC (Figure 2.10) is a tool to easily create Unity applications which can be controlled by hardware and software that use the OSC protocol for communication. Someone can use UniOSC to either send or receive OSC messages to or from other devices that are connected via Wi-Fi or create own GUI-app for controlling another Unity application via OSC. However, it is strongly recommended using a third party software such as TouchOSC for creating the GUI part [20]. In this thesis, the UniOsc is used to transport data from the TouchOsc mobile application (data from TouchOsc's faders, potentiometers and buttons) to Unity (for using the data from UniOsc tranformations' scripts).

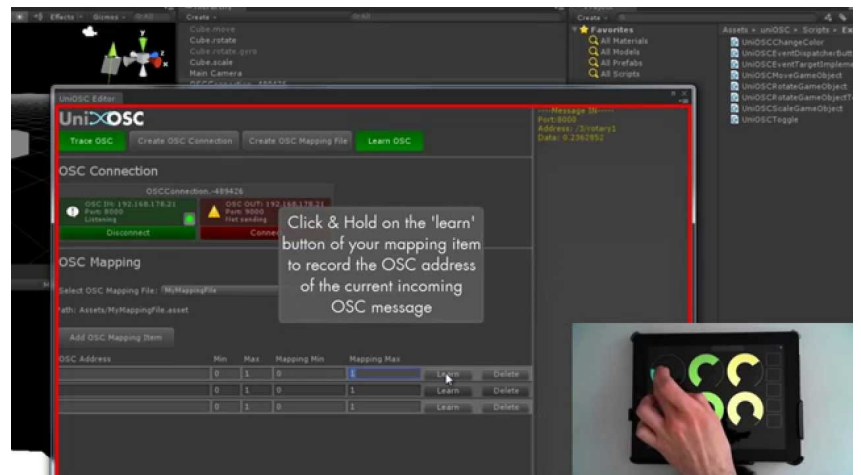


Figure 2.10: UniOSC

2. BACKGROUND

Chapter 3

Requirements Analysis

3.1 Introduction

Contemporary cities are live mechanisms, eco-systems, which perform under certain rules often not visible to the human eye at first sight. An urban designer needs advanced analytical tools in order to understand the inner-relations that describe these eco-systems. Even many of the recently crafted digital design tools for urban environments fail to address the need for a more holistic design approach that captures virtual and physical, immaterial and material including aspects of an urban design which are aesthetic as well as functional. There is a significant potential in combining the fields of composition in music and architecture through the use of information technology. Merging the two fields has the intense potential to release new, innovative tools for urban designers by uncovering aspects of an urban design not apparent based on the standard, vision-based digital design tools, introducing an additional modality of sound and music. This thesis describes an innovative tool developed at the Technical University of Crete, through which an urban designer can work on the music transcription of a specific urban environment applying music compositional rules and filters in order to identify discordant entities, highlight imbalanced parts and make design corrections.

The presented conceptual design tool connects music and architecture, based on the notion of 'composition' common to both fields, in order to result to refined urban environments such as a new refined 3D urban street. The music can play a significant role as a creative modality in order to produce desired and effective architectural designs. The creativity of the musicians can help to produce refined streets and by extension build

3. REQUIREMENTS ANALYSIS

refined cities. The user of the platform should have the concurrent ability to view the visualization of an urban street as well as listen to the sound of the city which comprises of architectural elements converted to music. It is significant to accurately define the functional elements of a complex user interface which connects vision and sound to be interactively manipulated. In addition, it has to be determined how these functions will be included in separate but connected user interfaces, how the user interacts with them and, finally, which musical or architectural form the final scenes or soundscapes will have.

We refer to the completed system as Archimusic3D.

3.1.1 General Requirements

The requested functionality of the proposed system is mentioned below:

1. An urban street should contain specific objects (such as Buildings, Balconies, Doors, Trees), whose features are variable (such as height, length and depth). The user should be available to navigate the visualization of this 3D urban environment in order to observe it, therefore, a user interface window visualizing 3D scenes offering 3D navigation should be included.

2. Specific architectural elements such as the ones mentioned above should convert into notes which are played by specific musical instruments (such as the piano, flute, bass, guitar). These notes have also variable features (such as pitch, duration and volume), whose values are computed by the translation to music of the architectural features of the corresponded objects. The user should be available to view the visualization of the urban environment's musical footprint containing notes played from a variety of musical instruments. The conversion follows grammar rules described in Section 3.2.3.

3. Several different urban streets can be generated including exactly the same architectural elements, depending on the position of these elements on the street. Similarly, several different songs can be generated with exactly the same notes, played by specific musical instruments, depending on the time interval for which these notes are playing. Hence, the user should listen to the musical footprint of each urban environment in order to observe the differentiation of the the acoustic output of each urban street converted to a musical footprint. In order to satisfy this requirement, a window representing a Digital Audio Workstation (DAW) program should be included.

4. The user should have the ability to transform the features of the existing notes (for example to change the pitch, duration, volume, time that a note is playing) and the ability to add new notes played by any selected musical instrument. The transformations of the resulting existing notes should be translated in real-time to transformations of the architectural features (height, length, depth, position of the corresponded objects) signifying transformations of the 3d architectural elements of the urban street. In order to satisfy this requirement, a user interface representing a TouchOsc mobile controller (Figure 3.21), which is a virtual mobile console containing faders, potentiometers and buttons, should be included.

3.2 Use Case Scenarios

In order to satisfy the required specifications, it was decided to employ the Unity programming environment for interactive 3D applications, featuring a split screen design tool. The user interface provides at the half upper screen the visualization of the 3D urban environment of an urban street (Street View) and at the half bottom screen the visualization of its musical footprint, ready for musical transformations (DAW Program View). In order to use the platform, the designer runs on the build of the Archimusic3D Unity's project, having also available a smartphone, on which the TouchOSC application is installed. The user can listen to the sounds that make up the musical footprint of the visualized urban street, while at the same time viewing the 3D urban street in order to observe the refined architectural objects as transformed based on musical conversions.

Summarizing, the presented platform comprises of three scenes, which compile the three main parts of the system's interface; e.g., the 3D scene, the Digital Audio Workstation (DAW) scene and the TouchOsc mobile controller (Figure 1.1, upper and lower part respectively). While interacting with the 3D scene, the user is able to navigate a 3D street in order to observe the 3D graphical depiction of an urban environment. The second scene is the implementation of a Digital Audio Workstation (DAW) program, in which the user is able to see the musical footprint of the street printed as a MIDI file (Musical Instrument Digital Interface) in order to listen to it and process it via a third tool representing a TouchOsc mobile controller, which is a virtual mobile console contains faders, potentiometers and buttons. When users convert the 3D scene to a musical foot-

3. REQUIREMENTS ANALYSIS

print, they submit the reciprocal transformations which may be applied to the musical part and then, they are able to navigate a newly refined street.

3.2.1 Urban Street View

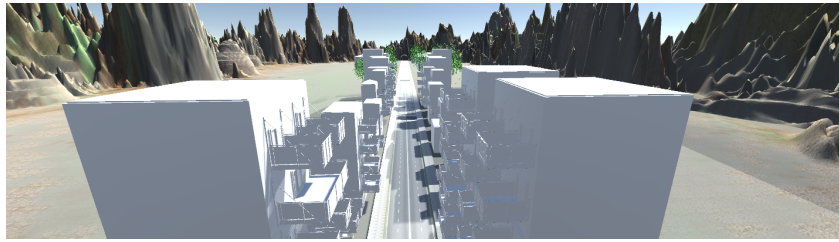


Figure 3.1: Street View

Interacting with the interface shown at Figure 3.1, the user has the ability to navigate the 3D urban street, in order to observe it. The user can change the Camera's Components such as the position, height and rotation via the TouchOsc controller.

3.2.2 Digital Audio Workstation View

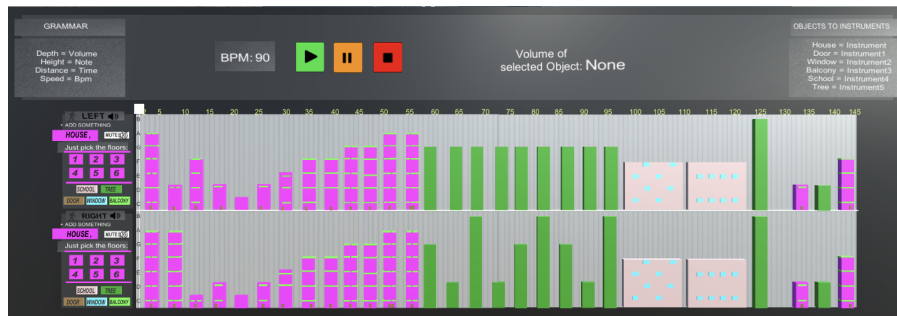


Figure 3.2: Daw Program View

By interacting with the interface shown at Figure 3.2, the user is able to view the street's resulting musical footprint, in order to listen, observe and modify it. The user can play and stop the sounds of the musical footprint, mute a speaker (Left / Right), select and modify the musical parameters (such as the time, duration, the volume and the pitch

that the note is played) of any note played by any instrument via the TouchOsc controller. As the user selects and modifies a note, concurrently, the resulting transformations of the musical parameters of any note are translated, in real-time, to transformations of the architectural parameters of the corresponding 3D object (such as the position, length, depth and the height of that object). Furthermore, the user has the ability to add a new note by any musical instrument and by extension, add a new 3D object representing a specific architectural feature. The buttons as shown in Figure 3.2 are going to be explained in detail below. The magenta bars included in the grey part of the interface represent the buildings differentiated by their height represented by the varied height of the bars. The small green lines appearing on the magenta bars represent the balconies of the buildings. There are also brown spots at a low level of the magental bars representing doors. The green bars signify the trees included in the scene. The pink squares represent the larger scale buildings and the blue dots inside these squares represent the windows of the larger scale buildings.

The user is able to interact with these elements of the grey area of the DAW interface by selecting any bar and at the time of selection, the code connecting this interface with the TouchOSC interface are enabled. If the user selects a magenta bar, then through the TouchOSC interface, the position on the x-axis, the length, the volume of the corresponding note and other changes included in the TouchOSC interface. For these buildings, the user cannot change the height of the building. By changing, though, the position and the scale-x of the building (length), we also change the exact time and the duration the note is being played. The user can also select the green bars signifying trees and then, based on the TouchOSC interface, we can change the position, length and the scale of the tree on the y-axis. Concurrently, we change the music's parameters which correspond to the position (time when the note is being played), the length (duration of music to be played) and the scale y-axis (pitch). The volume signifies how far away the object is moved from the street. In relation to the pink larger-scale buildings, when we select the pink area, we can alter the same parameters as mentioned above for the green bars (trees) including the height of the building. When selecting the blue dots inside the pink squares signifying the windows of larger-scale buildings, then through the TouscOSC controller we can change position x-axis, length and volume as previously. The only difference is when changing the position on the y-axis of the windows this signifies the pitch. Because the trees and buildings lie on the ground, pitch is signified by scale.

3. REQUIREMENTS ANALYSIS

3.2.3 Traslation

At first look, music and architecture seem unrelated, but they share features. The first step is to match the various parameters and variables of the urban eco-system to those of its musical footprint. The Digital Media Lab from the School of Architecture, Technical University of Crete, has developed a translation method [2] between music and architecture for compositional experiments on urban design. According to this method, different urban and architectural elements that define streetscapes are converted into musical notes based on their geometrical properties (x-y-z position, size, etc). The architectural objects (such as buildings) which make up an urban street are converted to notes played by musical instruments (such as the piano), which make up a music song, and conversely. Any given path of an urban setup can be marked in order to create its soundscape with sounds produced by selected musical instruments. A simulation of an urban environment is created including urban elements fundamental for "reading". Building blocks are provided by the system and external elements can be included.

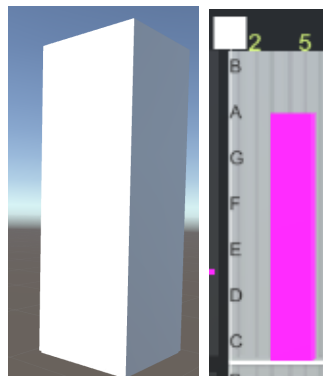


Figure 3.3: Six storey building is converted to A note

The application supports five types of architectural elements utilized to build an urban 3D environment and consequently for generating sound. These elements are "Buildings", "Larger Scale Buildings", "Windows", "Balconies", "Doors" and "Trees". One can use these elements to build complex urban environments which can be transformed to a sound. In this thesis, we have made the decision to associate the piano with the window, the flute with the tree, the xylophone with the balcony, the bass with the door, the cello with the building and the violin with the larger scale building. These elements are 3D shapes and as such have the following properties: height, length and depth. As a basic

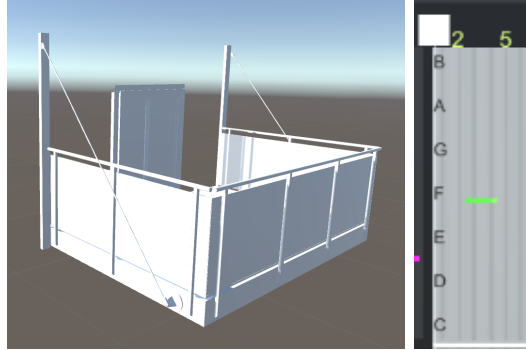


Figure 3.4: Balcony of fourth floor is converted to F note

unit for mapping, height to note value is the "FLOOR" on which the object is located. The basic rules for this translation are: the height of an element is translated to the pitch of a note, the length of the element is translated to the duration of the note, and the depth of the element is translated to the volume of the note. The floor is considered to be 3 m high. The first floor is mapped to C, the second to D, the third to E, and so on. As for the length, 1 m is mapped to 1 s note duration. For example, if we have a building that is 10 m high and 15 m long, it would be translated to the F note with a duration of 15 secs. Respectively, a building or a balcony that is protruding will sound more intense and one that is recessed will have a lower volume, thus, mapping the perception that it is further away. In Figure 3.3 a six-storey building is converted to the A note played by a cello and in figure 3.4 a balcony of the fourth floor is converted to the F note played by a xylophone. When we refer to a 'play-bar' we mean the white cube shown in Figure 3.4 which traverses the notes of the musical footprint of an urban scene to be concurrently played.

3.3 Manual of the System

The way in which the user interacts with the platform and uses it is distinguished in two parts. For the first part the user is able to use the mouse to select an object such as a note or a button. For the second part the user has the ability through the TouchOsc Controller to intervene on the platform's Components as explained in Section 3.3.2.

3. REQUIREMENTS ANALYSIS

3.3.1 Manual of the DAW interface

By interacting with the interface representing the simulation of a DAW program, the user is able to transform a note and press a button using a left mouse click (Figure 3.5).



Figure 3.5: Part of the Digital Audio Workstation View

- Play Button

Figure 3.6 shows the play button. When this button is pressed, the play-bar starts moving and the music begins to play. When referring to 'music' we mean the actual musical footprint which results from translating an urban street to a musical piece based on the translation grammar specified in the system, as explained in 3.2.3.



Figure 3.6: Play Button

- Pause Button

Figure 3.7 shows the pause button. When this button is pressed, the play-bar stops to move and the music stops. When referring to 'music' we mean the actual musical footprint which results from translating an urban street to a musical piece based on the translation grammar specified in the system.

- Stop Button

The Figure 3.8 shows the stop button. When this button is pushed the play-bar moves to the start of the musical footprint and the music stops. When referring to 'music' we mean the actual musical footprint which results from translating an

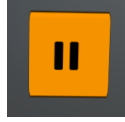


Figure 3.7: Pause Button

urban street to a musical piece based on the translation grammar specified in the system.



Figure 3.8: Stop Button

- Mute Button

The Figure 3.9 shows the mute left / right speaker button. With this button the user can mute the sounds of any speaker. This button can be enabled or disabled.



Figure 3.9: Mute Button

- Balcony Button

The Figure 3.10 shows the "generate a new balcony" button. This button can be enabled or disabled. This button receives information from the TouchOSC controller. When this button is pressed, the "generate note" button (Figure 3.27) of the TouchOsc controller is enabled. After this action, when the "generate note" button of the TouchOSC controller is pressed, a new balcony of the fourth floor is generated and is converted to the F note played by an xylophone as defined by the grammar, as shown in Figure 3.10.

3. REQUIREMENTS ANALYSIS

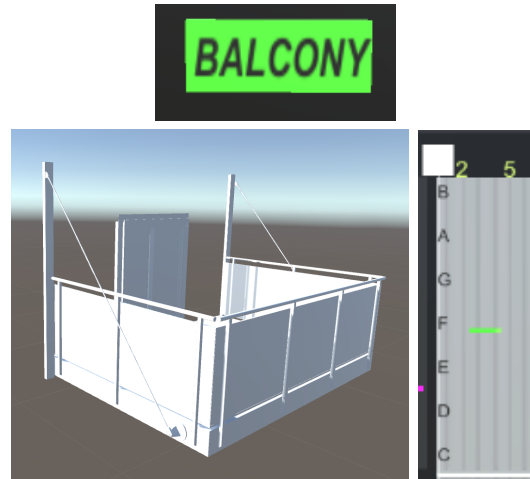


Figure 3.10: New Balcony Button

- Window Button

The Figure 3.11 shows the "generate a new window" button. This button can be



Figure 3.11: New Window Button

enabled or disabled. This button receives information from the TouchOSC controller. When this button is pressed, the "generate note" button(Figure 3.27) of the TouchOsc controller is enabled. After this action, when the "generate note" button of the TouchOSC controller is presseddd, a new window is generated and is converted to the F (note) played by a piano. The fact that the F note is played is

due to the fact that geometry generation occurs at a specific height which translates to the F note. Now, when moving the F note then the window is being moved to the desired location which corresponds to a different note. The architectural elements' movement is being controlled by movement of the notes and not direct movement of the geometry in the 3D scene interface.

3. REQUIREMENTS ANALYSIS

- Door Button

The Figure 3.12 shows the "generate a new door" button. This button can be enabled or disabled. This button receives information from the TouchOSC controller. When this button is pressed, the "generate note" button (Figure 3.27) of the TouchOsc controller is enabled. After this action, when the "generate note" button of the TouchOSC controller is pressed, a new door of the door is generated and is converted to the C note played by a bass. The generation of the door geometry is placing the door below the windows and balconies, this is why the C note is generated. For the door to be moved, the C note is being selected in the DAW interface and by moving the note, the door is simultaneously moved in the desired position.

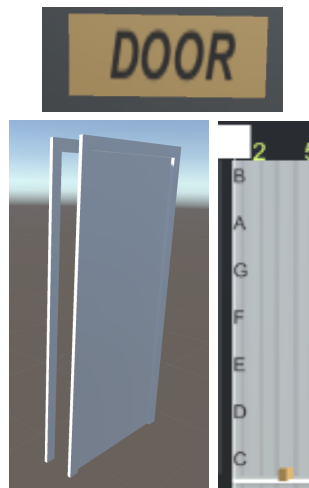


Figure 3.12: New Door Button

- Tree Button

The Figure 3.13 shows the "generate a new tree" button. This button can be enabled or disabled. This button receives information from the TouchOSC controller. Figure 3.13 shows the "generate a new tree" button. When this button is pressed, the "generate note" button (Figure 3.27) of TouchOsc controller is enabled. After this action, when the "generate note" button of the TouchOSC controller is pressed, a new tree is generated and is converted to the G played by a flute as defined by the geometrical placement of the tree. The G note represents the highest point of

the tree. It is shown as a bar in Figure 3.13, because, as defined by the system, trees, buildings and larger-scale buildings have as a base the ground plane and their height is signified by their length (scale on the y-axis). For the other objects (such as doors, balconies, etc), we take as their note equivalent their x-axis position in space.

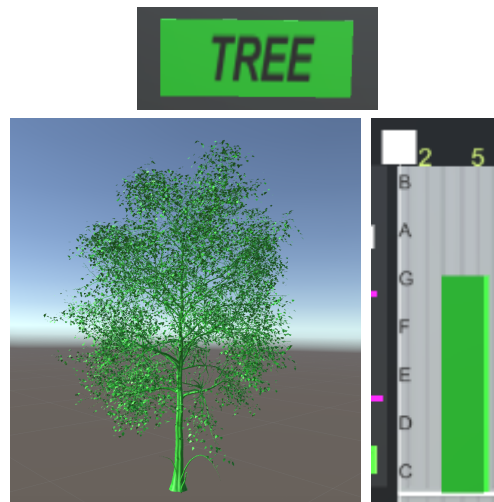


Figure 3.13: New Tree Button

3. REQUIREMENTS ANALYSIS

- Larger Building Button

The Figure 3.14 shows the "generate a new larger scale building" button. This button can be enabled or disabled. This button receives information from the TouchOSC controller. When this button is pressed, the "generate note" button (Figure 3.27) of the TouchOSC controller is enabled. After this action, when the "generate note" button of TouchOSC controller is pressed, a new larger building is generated and is converted to the F note played by a violin. Similarly with trees, as defined by the system, trees, buildings and larger-scale buildings have as a base the ground plane and their height is signified by their length (scale on the y-axis). For the other objects (such as doors, balconies, etc), we take as their note equivalent their x-axis position in space.

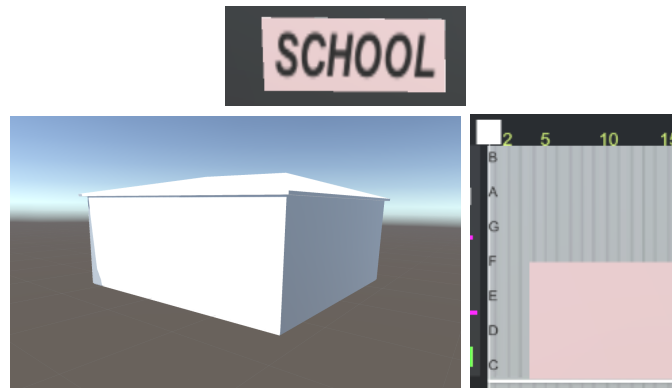


Figure 3.14: New Larger Scale Building Button

- Button 1

The Figure 3.15 shows the "generate a new one-storey building" button.

This button can be enabled or disabled. This button receives information from the TouchOSC controller. When this button is pressed, the "generate note" button of the TouchOSC controller is enabled. After this action, when the "generate note" button (Figure 3.27) of TouchOSC controller is pressed, a new one-storey building is generated and is converted to the C played by a cello. As defined by the system, buildings of any height and larger-scale buildings have as a base the ground plane and their height is signified by their length (scale on the y-axis). For the other

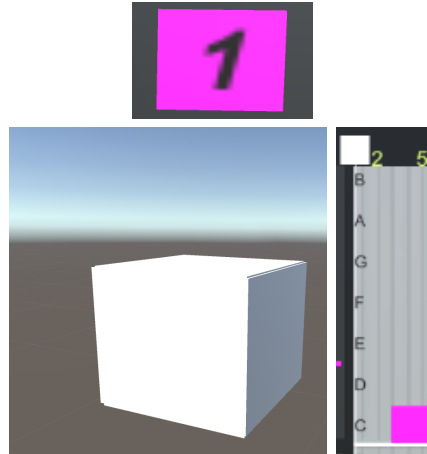


Figure 3.15: New One-storey Building Button

objects (such as doors, balconies, etc), we take as their note equivalent their x-axis position in space.

- Button 2

The Figure 3.16 shows the "generate a new two-storey building" button. This button can be enabled or disabled. This button receives information from the TouchOSC controller. When this button is pressed, the "generate note" button (Figure 3.27) of the TouchOSC controller is enabled. After this action, when the "generate note" button of the TouchOSC controller is pressed, a new two-storey building is generated and is converted to the D note played by a cello. The same applies as previously for buildings of any height.

3. REQUIREMENTS ANALYSIS

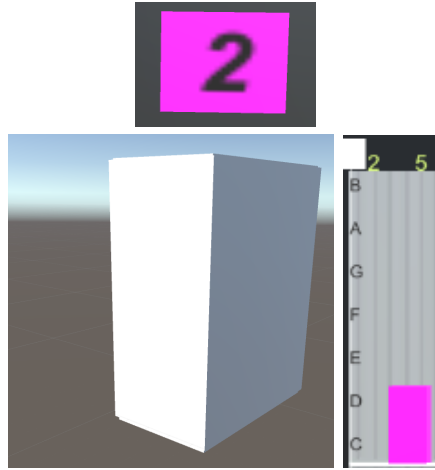


Figure 3.16: New Two-storey Building Button

- Button 3

The Figure 3.17 shows the "generate a new three-storey building" button. This



Figure 3.17: New Three-storey Building Button

button can be enabled or disabled. This button receives information from TouchOSC controller. When this button is pressed, the "generate note" button of the TouchOSC controller is enabled. After this action, when the "generate note" button (Figure 3.27) of TouchOSC controller is pressed, a new three-storey building is generated and is converted to E note played from a cello. The same applies as previously for buildings of any height.

- Button 4

The Figure 3.18 shows the "generate a new four-storey building" button. This button can be enabled or disabled. This button receives information from the TouchOSC controller. When this button is pressed, the "generate note" button (Figure 3.27) of the TouchOSC controller is enabled. After this action, when the "generate note" button of TouchOSC controller is pressed, a new four-storey building is generated and is converted to the F note played by a cello. The same applies as previously for buildings of any height.

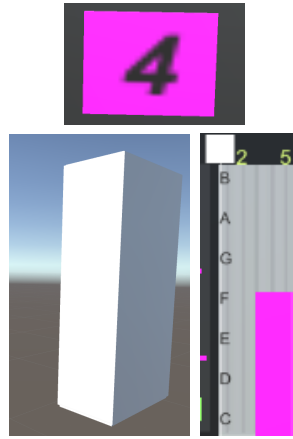


Figure 3.18: New Four-storey Building Button

3. REQUIREMENTS ANALYSIS

- Button 5

The Figure 3.19 shows the "generate a new five-storey building" button. This button can be enabled or disabled. This button receives information from the TouchOSC controller. When this button is pressed, the "generate note" button (Figure 3.27) of the TouchOSC controller is enabled. After this action, when the "generate note" button of TouchOSC controller is pressed, a new five-storey building is generated and is converted to the G note played by a cello. The same applies as previously for buildings of any height.

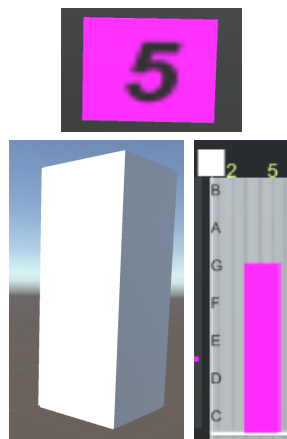


Figure 3.19: New Five-storey Building Button

- Button 6

The Figure 3.20 shows the "generate a new six-storey building" button. This button

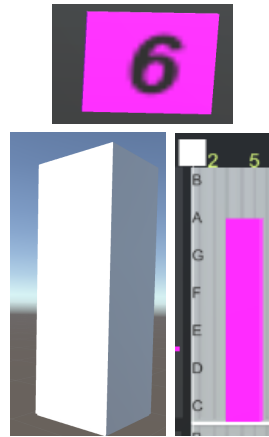


Figure 3.20: New six-storey Building Button

can be enabled or disabled. This button receives information from the TouchOSC controller. When this button is pressed, the "generate note" button(Figure 3.27) of the TouchOsc controller is enabled. After this action, when the "generate note" button of the TouchOSC controller is pushed, a new six-storey building is generated and is converted to the A note played by a cello.

3.3.2 Manual of the TouchOSC

The TouchOSC mobile controller which runs on an Android mobile device and communicates with the main system running on a desktop PC, is used for transformations of musical features of notes and architectural features of objects. The TouchOSC mobile application is connected to the main platform via the UniOSC asset. Archimusic3D's TouchOSC template provides two buttons, six faders and two potentionmeters which are employed to apply transformations on musical and architectural components as analyzed below.

3. REQUIREMENTS ANALYSIS

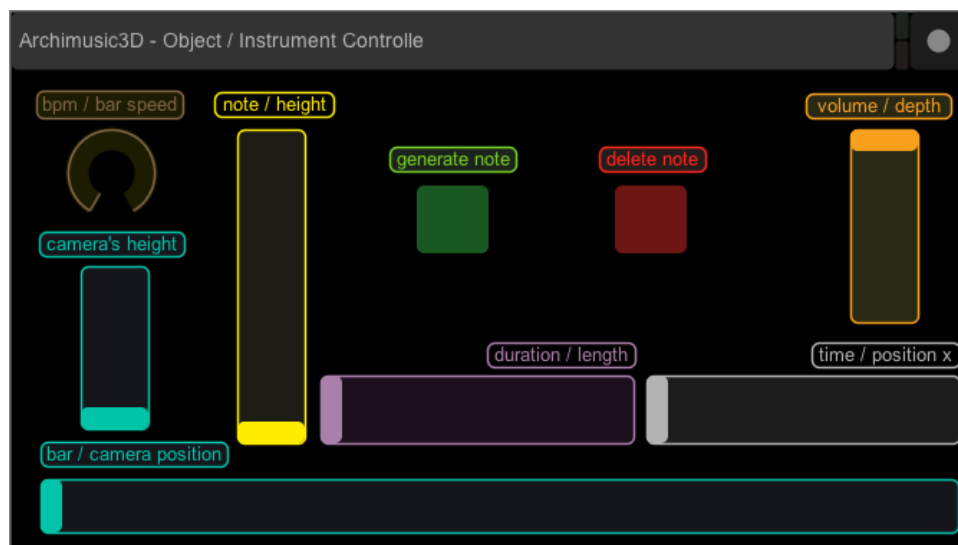


Figure 3.21: TouchOsc Archimusic3D Layout

- time / position x fader

The "time / position x" fader, as shown in Figure 3.21 in grey colour, of the Archimusic3D TouchOSC's mobile template, is the fader which specifies the exact timing on which the selected note is played and the position X of the corresponding selected object receives concurrently. Figure 3.22 shows an example of what happens. Initially, the user selects through a left mouse click a note (shown in green) by interacting with the DAW scene as shown on the right side of Figure 3.22. Subsequently, the user moves the grey slider of the TouchOSC controller on the left side of Figure 3.22. By initially selecting a note, the user is basically also selecting the architectural element of the 3D scene which this particular note represents. In this case, this is a balcony. By moving the slider of the TouchOSC controller, the balcony moves on the X axis as shown in the central scene of Figure 3.22 and simultaneously, the note representing the balcony moves in relation to when it is played. As we can see in Figure 3.22, the note initially (before interacting with the TouchOSC controller) is played at around timing 2 (as shown in green colour in the DAW interface). After interacting with the TouchOSC controller, the note is played around timing 10.

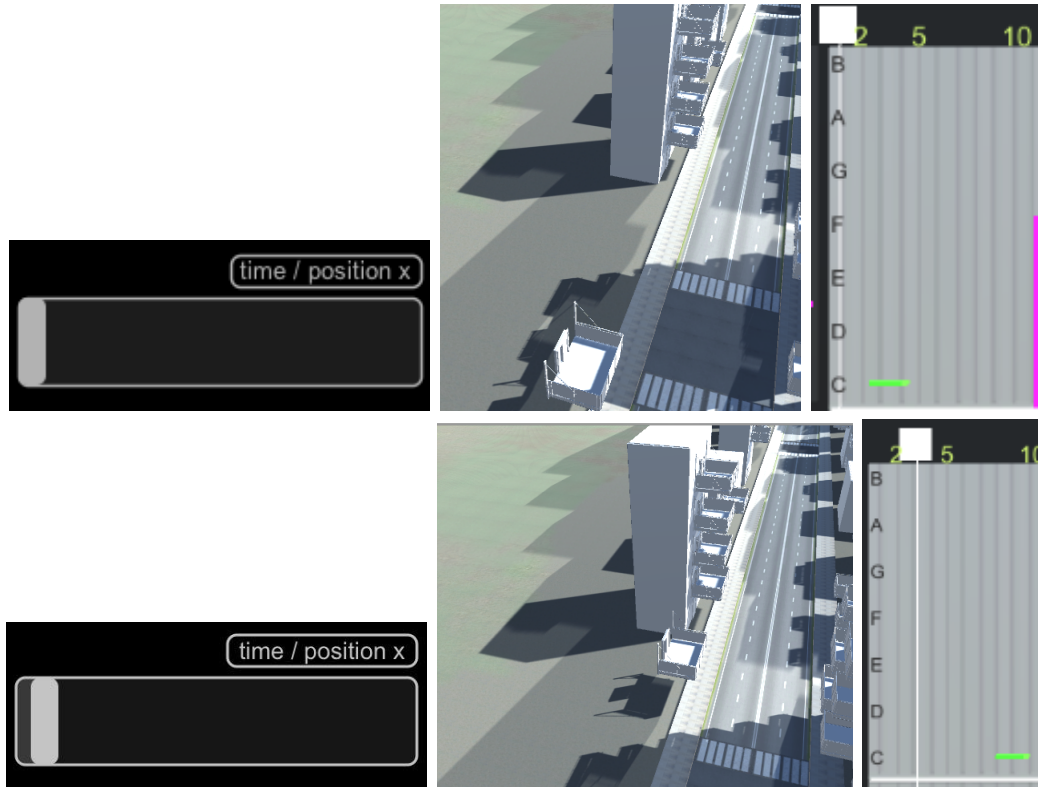


Figure 3.22: Position / Time TouchOsc Fader

- duration / length fader

The "duration / length" fader, as shown in Figure 3.21 in purple colour, of the Archimusic3D TouchOSC's mobile template, is the fader representing the exact duration of the selected note's sound and the length of the corresponding selected object receives concurrently. Figure 3.23 shows an example of what happens. Initially, the user selects through a left mouse click a note (shown in green) by interacting with the DAW scene as shown on the right side of Figure 3.23. Subsequently, the user moves the purple slider of the TouchOSC controller on the left side of Figure 3.23. By initially selecting a note, the user is basically also selecting the architectural element of the 3D scene which this particular note represents. In this case, this is a balcony. By moving the slider of the TouchOSC controller, the balcony's length is enlarged as shown in the central scene of Figure 3.23 and simultaneously, the note representing the balcony's length in relation to the pitch of the

3. REQUIREMENTS ANALYSIS

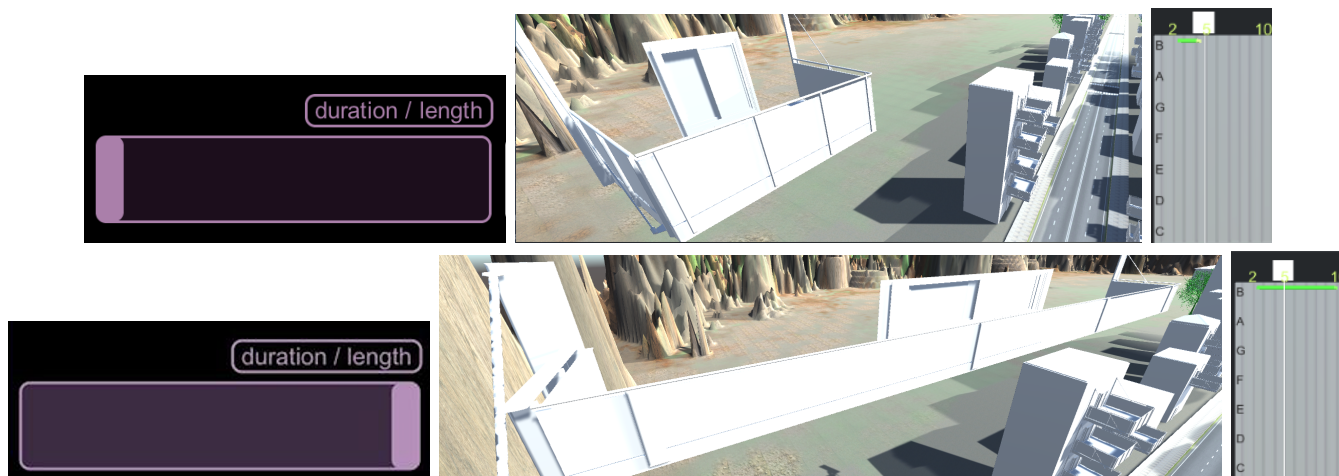


Figure 3.23: Duration / Length TouchOsc Fader

note is played. As we can see in Figure 3.23, the note initially (before interacting with the TouchOSC controller) is started to be played at around timing 2 and it is finished to be played at around timing 4 (as shown in green colour in the DAW interface), of duration 2. After interacting with the TouchOSC controller, the note is started to be played at around timing 2 and it is finished to be played at around timing 10, now of duration 8.

- note / height fader

The "note / height" fader, as shown in Figure 3.21 in yellow colour, of the Archimusic3D TouchOSC's mobile template, is the fader representing the exact pitch of the selected note's sound and the height of the corresponding selected object receives concurrently. The Figure 3.24 shows an example of what happens. Initially, the user selects through a left mouse click a note (shown in green) by interacting with the DAW scene as shown on the right side of Figure 3.24. Subsequently, the user moves the yellow slider of the TouchOSC controller on the left side of Figure 3.24. By initially selecting a note, the user is basically also selecting the architectural element of the 3D scene which this particular note represents. In this case, this is a balcony. By moving the slider of the TouchOSC controller, the balcony's height as shown in the central scene of Figure 3.24 is elevated and simultaneously, the note representing the balcony's height in relation to how much time the note is

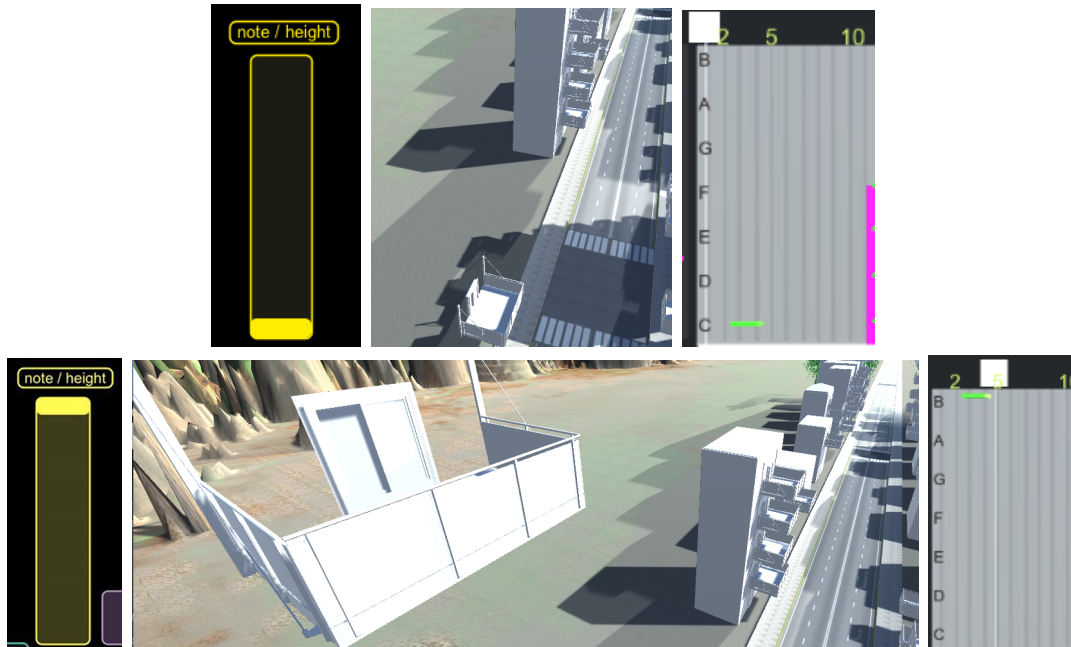


Figure 3.24: Note / Height TouchOsc Fader

played. As we can see in Figure 3.24, the note initially (before interacting with the TouchOSC controller) is a C note (as shown in green colour in the DAW interface) played by an instrument. After interacting with the TouchOSC controller, the note is a B note played by an instrument.

- volume / depth fader

The "volume / depth" fader, as shown in Figure 3.21 in orange colour, of the Archimusic3D TouchOSC's mobile template, is the fader representing the exact volume of the selected note's sound and the depth of the corresponding selected object(how far away from street is placed the object) receives concurrently. The Figure 3.25 shows an example of what happens. Initially, the user selects through a left mouse click a note (shown in green) by interacting with the DAW scene as shown on the right side of Figure 3.25. Subsequently, the user moves the orange slider of the TouchOSC controller on the left side of Figure 3.25. By initially selecting a note, the user is basically also selecting the architectural element of the 3D scene which this particular note represents. In this case, this is a balcony. By moving the slider of the TouchOSC controller, the balcony moves further away from the

3. REQUIREMENTS ANALYSIS

street as shown in the central scene of Figure 3.25 and simultaneously the note's (sound) volume is reduced. As we can see in Figure 3.25, the note initially (before interacting with the TouchOSC controller) is played at volume 1. After interacting with the TouchOSC controller, the note is played at volume 0.

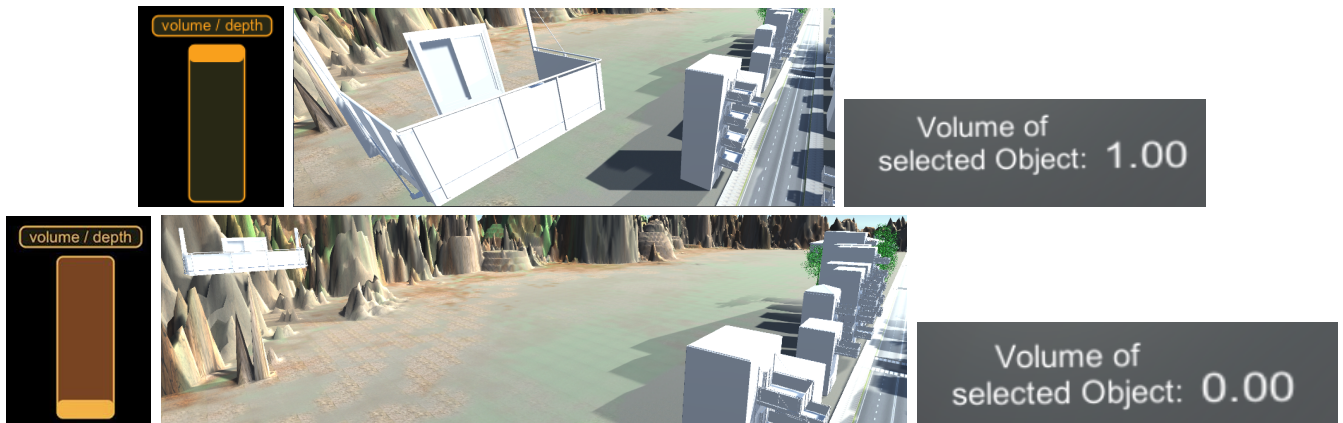


Figure 3.25: Volume / Depth TouchOsc Fader

- delete note button

This is the "delete note" button (Figure 3.26), as shown in Figure 3.21 in red colour, of the Archimusic3D TouchOSC's mobile template. The user selects a note through a left mouse click and after the "delete note" button is pressed. When the "delete note" button is pressed the selected note and the corresponding object are deleted.

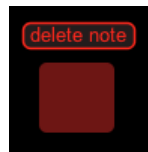


Figure 3.26: Delete Note Button

- generate note button

If a new object button is pressed (such as new "Balcony" button, Figure 3.10), the "generate note" button, of Archimusic3D TouchOSC's mobile template, is enabled. This is the enabled "generate note" button (Figure 3.27), as shown in Figure 3.21 in green colour, of the Archimusic3D TouchOSC's mobile template. The user selects a note through a left mouse click and afterwards, the "generate note" button is pressed. When the "generate note" button is pressed a new note (on the DAW view) and a corresponding object (on street view) are generated whose type are in relation to the type of the new "object" button.



Figure 3.27: Generate Note Button

3. REQUIREMENTS ANALYSIS

- bar / camera position fader

The "bar / camera position" fader, of Archimusic3D TouchOSC's mobile template, is the fader from which the exact position X of the play-bar (on DAW View) and the exact position x of the the camera (on street view) are concurrently received. The Figure 3.28 shows an example of what happens. Initially, the user selects through a left mouse click a note (shown in green) by interacting with the DAW scene as shown on the right side of Figure 3.28. Subsequently, the user moves the blue slider of the TouchOSC controller on the left side of Figure 3.28. By moving the slider of the TouchOSC controller, the play-bar (on DAW view) moves on the X axis as shown in the central scene of Figure 3.28 and simultaneously, the camera (on street view) moves in relation to the position that the slider is placed. As we can see in Figure 3.28, the play-bar initially (before interacting with the TouchOSC controller) is placed at around timing 1 (as shown as a white colour cube in the DAW interface). After interacting with the TouchOSC controller, the play-bar is moved at around timing 60. In addition as we can see in Figure 3.28, the camera initially (before interacting with the TouchOSC controller) is placed at around the start of the street. After interacting with the TouchOSC controller, the camera is moved at around the middle of the street. We note that the play-bar is the white square shown on the DAW view of Figure 3.28.

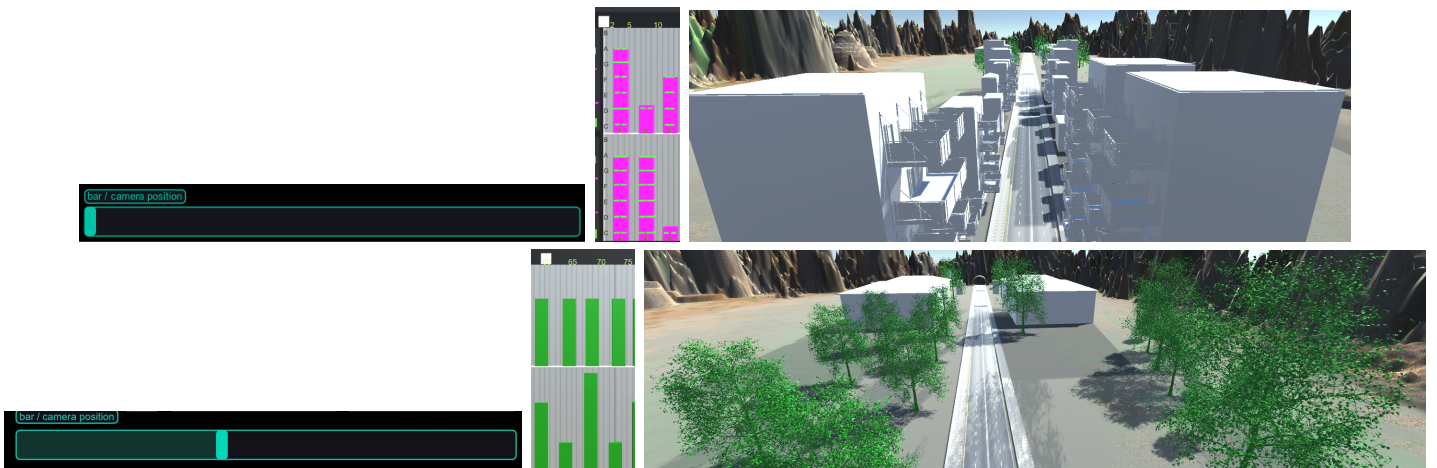


Figure 3.28: Bar / Camera Position TouchOsc Fader

- bpm / bar speed potentiometer

The "bpm / bar speed" potentiometer, of Archimusic3D TouchOSC's mobile template, is the brown potentiometer (as shown the left side of the Figure 3.29) from which the exact speed of the play-bar (on DAW View) and the exact speed of the camera (on street view) are concurrently received. The Figure 3.29 shows an example of what happens. The play-bar's speed, initially, (before interacting with the TouchOSC controller) is 60 (as shown on the right side of Figure 3.29). After interacting with the TouchOSC controller, the play-bar's speed is 120.



Figure 3.29: Bpm / Bar Speed TouchOsc Potentiometer

- camera's height fader

The "camera's height" fader, of Archimusic3D TouchOSC's mobile template, is the blue fader (as shown on the left side of the Figure 3.30) from which the exact height of the camera (on street view) are concurrently received. The Figure 3.30 shows an example of what happens. The camera initially (before interacting with the TouchOSC controller) is placed (as shown the right side of Figure 3.29) over the street. After interacting with the TouchOSC controller, the camera is placed at a higher position.

3. REQUIREMENTS ANALYSIS

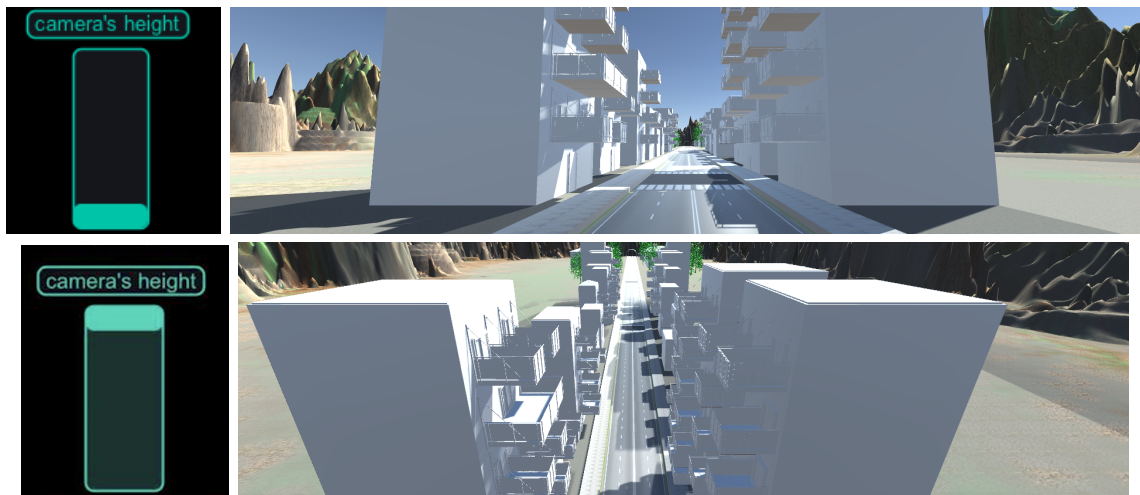


Figure 3.30: Camera's Height TouchOsc Fader

Chapter 4

Implementation

4.1 Introduction

The implementation of this thesis is divided into two stages.

The first stage focuses on the study of the theoretical framework across disciplines and the approaches that have been adopted by artists and scientists in order to connect music and architecture, both as a scientific and artistic endeavor.

The purpose of the first stage was to be familiarized with the terms and the technical tools employed in both architecture and in music. The background work conducted has led to the conception of the idea and the design of Archimusic3D as well as the requirements it should cover. It was also essential to investigate potential issues to be handled so that the most effective implementation of the original concept is produced.

The second phase concerns the solution of technical and user interface issues and the technical processing of individual pieces of the system which implement specific functions and in combination form the final desired platform.

4.2 Flow diagram

Figure [4.1](#) illustrates the flow diagram of the platform signifying data flow in the platform and according to which the user acts. The main scene consists of the projection of a 3D urban street and the projection of its musical footprint.

4. IMPLEMENTATION

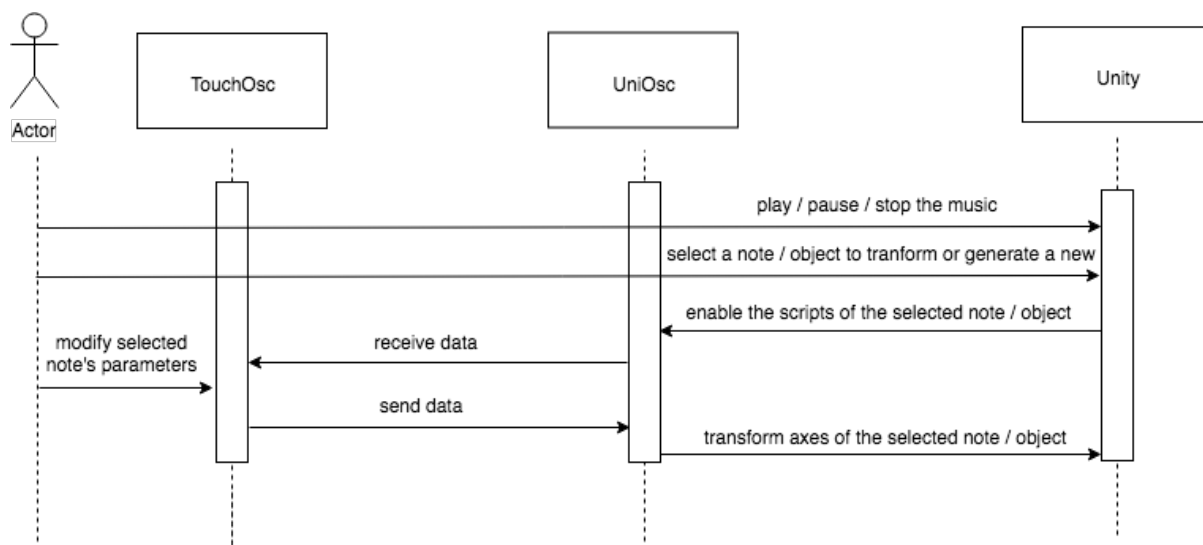


Figure 4.1: Flow Diagram

The user has the ability to start or stop playing the street’s musical footprint or to mute the musical footprint of any of the two sides each directed to different channels (left-right) through the buttons which are provided in the scene and can be selected with the mouse, as explained in Chapter 3.

In addition, the user can select, with the mouse, any note played by any musical instrument and concurrently any architectural element belonging to a specific type of object that the street contains. The user can modify the object’s musical and architectural parameters accordingly, delete the selected note or generate a new note associated with the type of object the user has chosen.

When a note is selected, its UniOSC scripts are enabled (which are the transformations’ scripts of an note / object), which are the scripts that allow the selected note to connect and listen to the ports of the TouchOsc mobile application, which provide to the user the platform’s controller. When the UniOSC scripts of a note are enabled, the user can convert its musical and architectural parameters by modifying the faders’ values which are provided as part of the TouchOsc mobile application.

The transformations that are applied by the user via the TouchOSC are sent as digital data to the UniOSC components. The UniOSC implements the appropriate transforma-

tions of the musical axes of the musical footprint and at the same time of the architectural axes of the 3D urban street, after the checking the type of the musical instrument of the selected note.

4.3 Building Information Model (BIM)

In order to create the musical footprint of an urban street, the urban street's design and geometry should be available in a form that contains information about each of the architectural axes of the objects, such as, for instance, the height and the length, that make up the buildings as well as information about the natural landscape which contains this street. The selected format of the 3D graphical objects, is the Building Information Model (BIM)(Figure 4.2) [21]. Building Information Modelling (BIM) is a

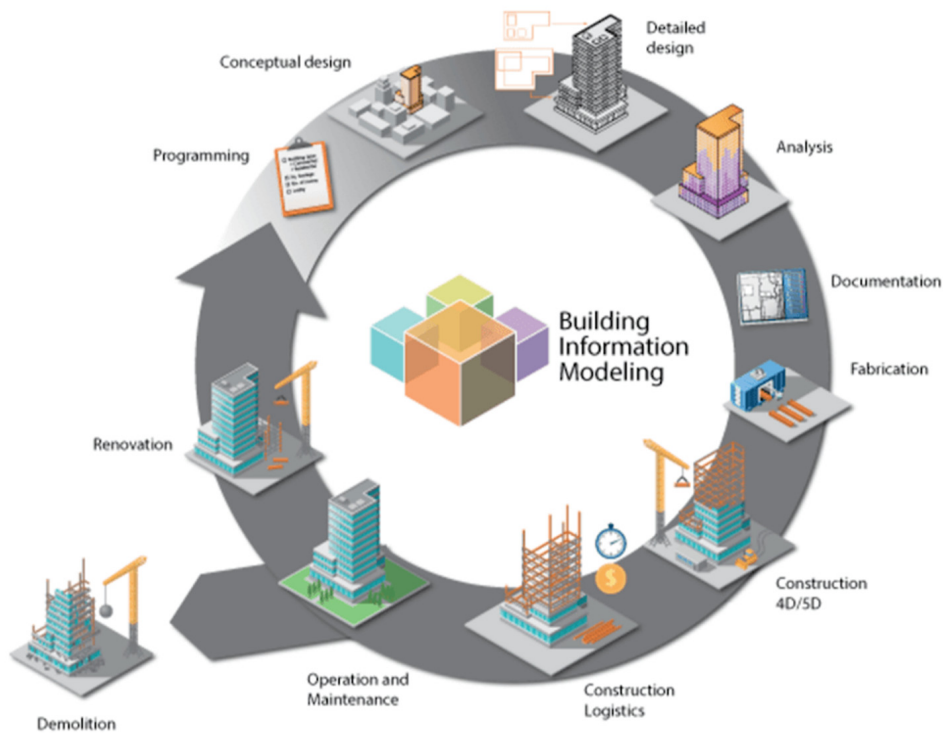


Figure 4.2: Building Information Model

process involving the generation and management of digital representations of physical

4. IMPLEMENTATION

and functional characteristics of places. Building information models (BIMs) are files (often but not always in proprietary formats and containing proprietary data) which can be extracted, exchanged or networked to support decision-making regarding a building or other built asset.

Current BIM software is used by individuals, businesses and government agencies who plan, design, construct, operate and maintain diverse physical infrastructures, such as water, refuse, electricity, gas, communication utilities, roads, railways, bridges, ports and tunnels [22].

In this thesis, the BIM is used to distinguish each of the objects that make up the 3D urban environment (such as a balcony, a building or a tree) and to cover the need for information on the architectural parameters of these objects.

4.3.1 REVIT

In order to adopt the Building information Modeling (BIM), there is a need to employ software that provides the possibility of creating such objects. The software that is chosen for create these objects is REVIT(Figure 4.3) [23].



Figure 4.3: Autodesk Revit

Revit BIM software includes features for architectural design and structural engineering as well as construction. REVIT supports a multidiscipline, collaborative design process. The REVIT software is specifically built for Building Information Modeling (BIM), empowering design and construction professionals to bring ideas from concept to construction with a coordinated and consistent model-based approach. It includes the

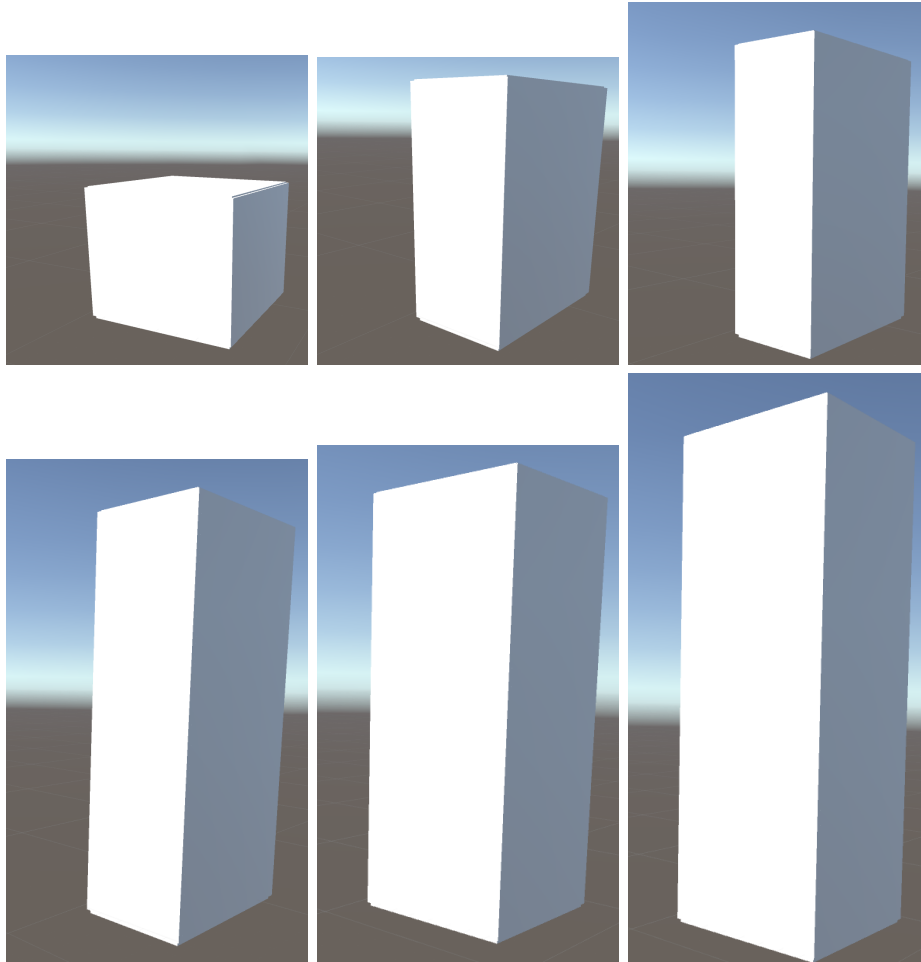


Figure 4.4: 3D Objects from Revit

functionality of all of the REVIT disciplines (architecture and structure) in one unified interface.

There exist several different types of objects in REVIT which were employed in this thesis such as a tree, balcony, window, door, one-storey building, two-storey building, three-storey building, four-storey building, five-storey building, six-storey building, larger scale building.

These objects have been integrated into the Unity platform for use as the types of objects that a user can add to the 3D virtual urban street to be created.

4. IMPLEMENTATION

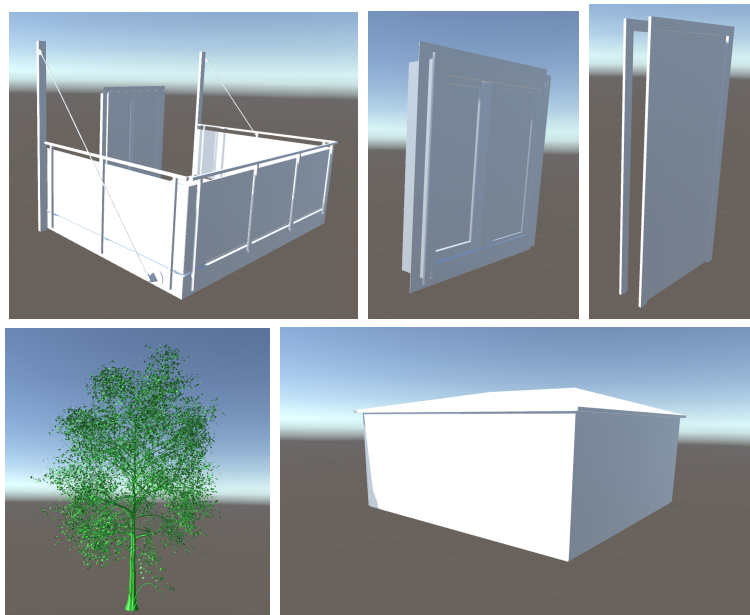


Figure 4.5: 3D Objects in REVIT

4.4 Grammar

Any given urban street can be marked in order to create its soundscape with sounds produced by selected musical instruments. A simulation of an urban environment is created including urban elements fundamental for "reading".

These are buildings, doors, windows and trees. Building blocks are provided by the system and external elements can be included. Once the environment is assembled, the user can choose an urban path to translate to sound. The path is then scanned from the starting to the ending edge and the sound representation of the elements on that path is saved in an Musical Instrument Digital Interface (MIDI) file.

The system utilizes the MIDI protocol to communicate between the graphical representation of the urban environment (a facade of the street is transcribed to its music transcription) and the acoustic output. The acoustic output, modified according to selected music composition rules and filters, follows the reverse process to be translated back to a new, refined urban environment. The result is a more balanced, tuned, built environment without reluctant pieces. Each 3D object is converted to a note played by several musical instruments.

The notes are MIDI messages and are expressed through note number for different octaves. There are 128 note values (11 octaves) mapped to the Western music scale. As a base octave, we chose the third octave because the lower octaves of low frequency prohibit perceptible sound differentiation between distinct elements.

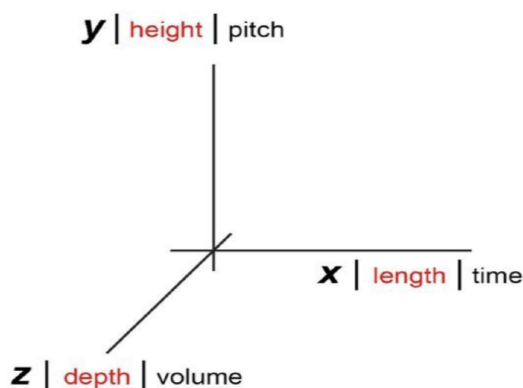


Figure 4.6: Translation concept

The platform supports six types of architectural elements utilized to build a urban 3D graphics environment and consequently for generating sound. These elements are "Buildings", "Larger Scale Buildings", "Windows", "Balconies", "Doors" and "Trees". The user can use these elements to build complex urban environments which can be transformed to a sound. In this thesis, we have the decision to associate the piano with the window, the flute with the tree, the xylophone with the balcony, the bass with the door, the cello with the building and the violin with the larger scale building. These elements are 3D shapes and as such have the following properties: height, length, and depth. The properties take values from the set of real numbers. As shown on Figures 4.7, 4.6, each element's height is translated to note value (pitch), the element's length to note duration (time), and the element's depth (position in Z axis) to volume. As a basic unit for mapping, height on which the element is located) to note value.

For example, the height of a balcony is considered to be 3 meters (m). The note, created by the musical transcription of this balcony, is mapped to C. As for the length, 1 m is mapped to 1 s note duration. For example, if we have a building that is 12 m

4. IMPLEMENTATION

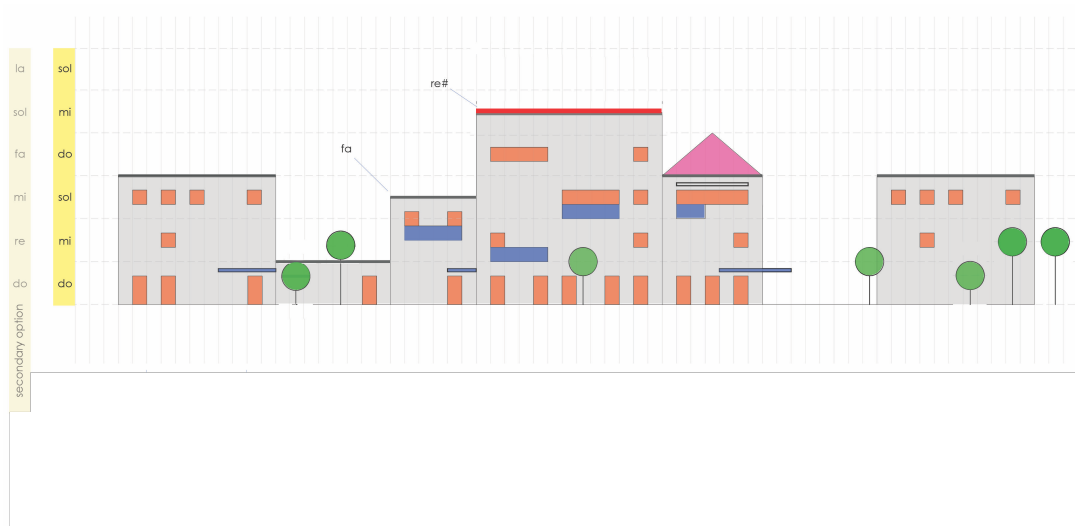


Figure 4.7: Translation example

high and 15 m long, it would be translated to the F note with a duration of 15 secs. Respectively, a tree or a balcony that is protruding will sound more intense and one that is recessed will have a lower volume, thus, mapping the perception that it is further away.

4.5 Components

Each of these objects form an object in Unity which has certain features called components. Components such as 'Transform', 'Mesh Renderer', 'Scripts' and so on may be different for each object. The components of an object run only if they are activated. Depending on transformations applied by the user to the musical axes of a particular note, the corresponding architectural axis of a particular object is simultaneously transformed. In order to be able to implement the requested transformations, access to the values of these components is necessary. We detail below the components incorporated in the objects as well as notes that make up the 3D urban street and its musical imprint.

4.5.1 Transform

Every object in a Scene has a Transform (Unity component which is attached on every object signifying geometrical properties based on the object's position (Figure 4.8). It's

used to store and manipulate the position, rotation and scale of the object. Every Transform can have a parent, which allows the user to apply position, rotation and scale [24].

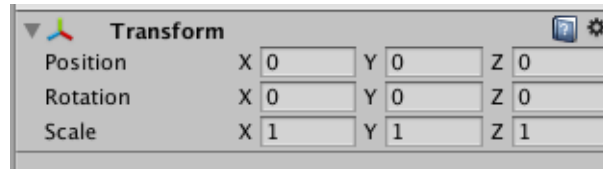


Figure 4.8: Tranform Component

4.5.2 AudioSource

An AudioSource (Figure 4.9) is attached to a gameobject (which is a 3D or 2D object in Unity) for playing back sounds in a 3D environment [25]. In order to hear sounds in the platform, we also need to have an AudioListener (which is a Unity component) to detect any sound. The AudioListener in this project is attached to the "play-bar" object. The AudioListener enables the sounds attached to the AudioSource, so that they can be listened.

Whether sounds are played in 3D or 2D is determined by the AudioImporter settings. The user can play a single audio clip using Play, Pause and Stop. The user can also adjust its volume while playing using the volume property.

4. IMPLEMENTATION

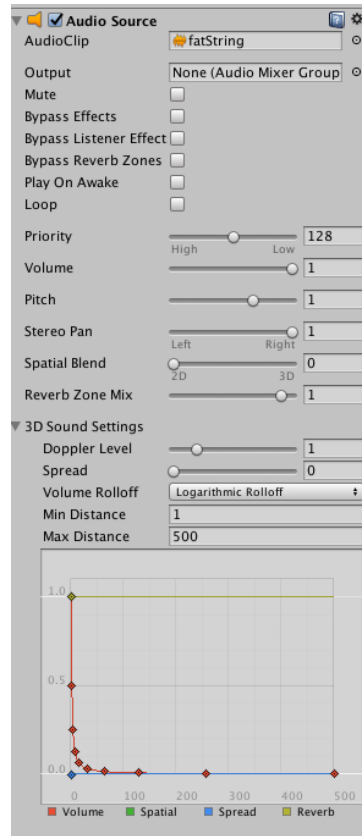


Figure 4.9: Audiosource Component

4.5.3 Scripts

The behaviour of gameobjects (which are 3D or 2D object in Unity) is controlled by the components that are attached to them. Unity allows the user to modify the components using scripts written in Csharp (Figure 4.10). These allow the user to trigger game events, modify component properties over time and respond to user input in any way the user likes [26].

In this thesis, the following scripts are separated into those that are enabled on awake (when the system is started) and remain enabled throughout the use of the platform used and those that are enabled only when the user has selected the object that contains them as components.



Figure 4.10: Script Component

- Position / Time

This script (Figure 4.11) is a UniOSC script, as it receives real-time information from the TouchOsc's "time / position x" fader (Figure 3.22), which is installed on the user's mobile device and it's activated when the user selects one of the objects that contain it as a component.

Included as a component of the selected note (which is a Unity object), checking the type of the selected instrument which plays this particular note (piano, flute, bass, xylophone, violin, cello) and the side of the street this note belongs to (either left or right speaker) signifying an architectural element of the actual 3D urban street (either left or right side of the street respectively) for which a note is being played. In this script the information obtained from a TouchOSC fader (value 0-1) is translated into the determination of the position in the x axis of the 3D object and the determination of the time at which the sound will be heard during the music song. In Figure 4.11 (below) we see the Unity value received by interacting with the TouchOSC component, in this case as shown the /fader2 signifies the time/position x bar on the TouchOSC.

Each architectural element is represented by a different instrument as explained in Chapter 3.

4. IMPLEMENTATION

```
public override void OnOSCMessageReceived(UniOSCEventArgs args)
{
    if (transformToMove == null) return;
    OscMessage msg = (OscMessage)args.Packet;
    if (msg.Data.Count < 1) return;

    float x = transformToMove.transform.position.x;
    float y = transformToMove.transform.position.y;
    float y1 = transformToMove1.transform.position.y;
    float z = transformToMove.transform.position.z;

    switch (movementMode)
    {
        case Mode.Screen:
            // y = position x of the selected note
            // y1 = position x of the selected object
            // transformToMove = selected note
            // transformToMove1 = selected GameObject

            if (tag == "HouseR" | tag == "HouseL" | tag == "TreeL" | tag == "TreeR")
            {
                y = 135 + (2730 * (float)msg.Data[0]);
                y1 = (390 * (float)msg.Data[0]);
            }

            else if (tag == "BalconyR" | tag == "BalconyL")
            {
                y = 119 + (2759 * (float)msg.Data[0]);
                y1 = (390 * (float)msg.Data[0]);
            }

            else if (tag == "SchoolR" | tag == "SchoolL")
            {
                y = 242 + (2568 * (float)msg.Data[0]);
                y1 = (float)23.1 + ((float)366.9 * (float)msg.Data[0]);
            }

            else if (tag == "DoorR" | tag == "DoorL")
            {
                y = 120 + (2770 * (float)msg.Data[0]);
                y1 = (390 * (float)msg.Data[0]);
            }

            else if (tag == "WindowR" | tag == "WindowL")
            {
                y = 112 + (2755 * (float)msg.Data[0]);
                y1 = (390 * (float)msg.Data[0]) - (float)1.5;
            }

            pos = transformToMove.transform.position;
            pos[0] = y; |
            transformToMove.transform.position = pos;
            pos1 = transformToMove1.transform.position;
            pos1[0] = y1;
            transformToMove1.transform.position = pos1;
        }
    }
}
```

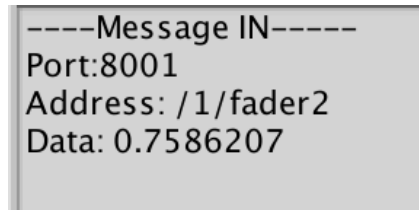


Figure 4.11: Position / Time Script

- Length / Duration Script

This script (Figure 4.12) is a UniOSC script, as it receives real-time information from the TouchOSC's "duration / length" fader(Figure 3.23), which is installed on the user's mobile device and it's activated when the user selects one of the objects that contain it as a component.

This script is included as a component of the selected note (which is a Unity object),

```

public override void OnOSCMessageReceived(UnoOSCEventArgs args){
    // y = lenght / scale x of the selected note
    // y1 = lenght / scale x of the selected object
    // transformToMove = selected note
    // transformToMove1 = selected GameObject

    if (transformToScale == null) return;
    OscMessage msg = (OscMessage)args.Packet;
    if(msg.Data.Count <1)return;

    _data = (float)msg.Data[0];
    // _data*= scaleFactor;
    if (tag == "HouseR" | tag == "HouseL" )
    {
        _scale.Set((float)48.5 + (_data * 100), transformToScale.localScale.y, transformToScale.localScale.z);
        _scale1.Set(1 + (_data * (float)2.6) , transformToScale1.localScale.y, transformToScale1.localScale.z);
    }
    else if ( tag == "TreeR" | tag == "TreeL")
    {
        _scale.Set((float)48.5 + (_data * 100), transformToScale.localScale.y, transformToScale.localScale.z);
        _scale1.Set((float)0.0125 + (_data * (float)0.0675),(float)0.0125 + (_data * (float)0.0675), transformToScale1.localScale.z);
    }
    else if (tag == "BalconyR" | tag == "BalconyL")
    {
        _scale.Set((float)9.7 + (_data * (float)135.8), transformToScale.localScale.y, transformToScale.localScale.z);
        _scale1.Set(1 + (_data * (float)5.4) , transformToScale1.localScale.y, transformToScale1.localScale.z);
    }
    else if (tag == "DoorR" | tag == "DoorL")
    {
        _scale.Set((float)9.7 + (_data * (float)53.35), transformToScale.localScale.y, transformToScale.localScale.z);
        _scale1.Set(1 + (_data * (float)7.4), transformToScale1.localScale.y, transformToScale1.localScale.z);
    }
    else if (tag == "WindowR" | tag == "WindowL")
    {
        _scale.Set((float)23 + (_data * 250), transformToScale.localScale.y, transformToScale.localScale.z);
        _scale1.Set((float)1.5 + (_data * (float)46.7), transformToScale1.localScale.y, transformToScale1.localScale.z);
    }
    else if (tag == "SchoolL" | tag == "SchoolR")
    {
        _scale.Set((float)230 + (_data * (float)115), transformToScale.localScale.y, transformToScale.localScale.z);
        _scale1.Set(1 + (_data * (float)0.4) , transformToScale1.localScale.y, transformToScale1.localScale.z);
    }
    transformToScale.transform.localScale = _scale;
    transformToScale1.localScale = _scale1;
}

```

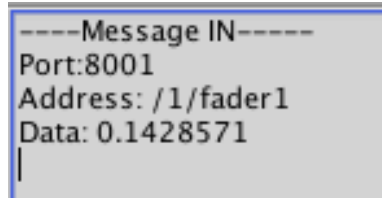


Figure 4.12: Length / Duration Script

checking the type of the selected instrument which plays this particular note (piano, flute, bass, xylophone, violin, cello) and the side of the street this note belongs to (either left or right speaker) signifying an architectural element of the actual 3D urban street (either left or right side of the street respectively) for which a note is being played. In this script, the information obtained from a TouchOSC fader (value 0-1) is translated into the determination of the scale in the X axis of the 3D object and the determination of the duration of the sound that will be heard when this note is playing. In Figure 4.12 (below) we see the Unity value received by interacting with the TouchOSC component, in this case as shown the /fader1 signifies the duration/length bar on the TouchOSC.

4. IMPLEMENTATION

- Height / Pitch

This script(Figure 4.13) is a UniOSC script, as it receives real-time information from the TouchOsc's "note / height" fader(Figure 3.24) , which is installed on the user's mobile device, and it's activated when the user selects one of the objects that contain it as a component. This script is included as a component of the

```
public override void OnOSCMessageReceived(UniOSCEventArgs args)
{
    // y = height of the selected note
    // y1 = height of the selected object
    // transformToMove = selected note
    // transformToMove1 = selected GameObject

    if (transformToMove == null) return;
    OscMessage msg = (OscMessage)args.Packet;
    if(msg.Data.Count < 1) return;

    float x = transformToMove.transform.position.x;
    float y = transformToMove.transform.position.y;
    float z = transformToMove.transform.position.z;
    float x1 = transformToMove1.transform.localScale.x;
    float y1 = transformToMove1.transform.localScale.y;
    float z1 = transformToMove1.transform.localScale.z;
    float Scx = transformToMove.transform.localScale.x;
    float Scy = transformToMove.transform.localScale.y;
    float Scz = transformToMove.transform.localScale.z;
    float Scx1 = transformToMove1.transform.localScale.x;
    float Scy1 = transformToMove1.transform.localScale.y;
    float Scz1 = transformToMove1.transform.localScale.z;

    switch (movementMode) {
        case Mode.Screen:
            _data = (float)msg.Data[0];
            if (tag == "BalconyL")
            {
                y = 140 + (340 * (float)msg.Data[0]);
                y1 = 3 + 18 * (float)msg.Data[0];
            }
            else if (tag == "DoorL")
            {
                y = 120 + (350 * (float)msg.Data[0]);
                y1 = 18 * (float)msg.Data[0];
            }
            else if (tag == "WindowL")
            {
                y = 150 + (340 * (float)msg.Data[0]);
                y1 = 3 + (float)17.7 * (float)msg.Data[0];
            }
            else if (tag == "BalconyR")
            {
                y = (336 * (float)msg.Data[0]) - 240;
                y1 = 3 + 18 * (float)msg.Data[0];
            }
            else if (tag == "DoorR")
            {
                y = (355 * (float)msg.Data[0]) - 263;
                y1 = 18 * (float)msg.Data[0];
            }
        }
    }
```

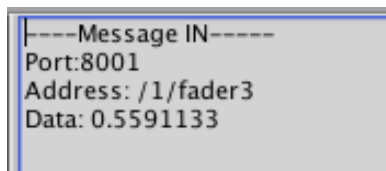


Figure 4.13: Height / Pitch Script

selected note (which is a Unity object), checking the type of the selected instrument

which plays this particular note (piano, flute, bass, xylophone, violin, cello) and the side of the street this note belongs to (either left or right speaker) signifying an architectural element of the actual 3D urban street (either left or right side of the street respectively) for which a note is being played. In this script, the information obtained from a TouchOSC fader (value 0-1) is translated into the determination of the position in the y axis of the 3D object (if the type of the object is either a Window, Door or Balcony) or the scale in the y axis of the 3D object (if the type of the object is either a Tree or Larger Scale Building) and the determination of the pitch which will be heard, when this note is playing. For the buildings signified as magenta bars in the DAW interface, we cannot use this script because they have a default assignment for height. In Figure 4.13 (below) we see the Unity value received by interacting with the TouchOSC component, in this case as shown the /fader3 signifies the note/height bar on the TouchOSC.

- Depth / Volume

This script is included as a component of the selected note (which is a Unity object), checking the type of the selected instrument which plays this particular note (piano, flute, bass, xylophone, violin, cello) and the side of the street this note belongs to (either left or right speaker) signifying an architectural element of the actual 3D urban street (either left or right side of the street respectively) for which a note is being played. This script (Figure 4.14) is a UniOSC script, as it receives real-time information from the TouchOsc's "volume / depth" fader(Figure 3.25), which is installed on the user's mobile device and it's activated when the user selects one of the objects that contain it as a component. In this script, the information obtained from a TouchOSC fader (value 0-1) is translated into the determination of the position in the z axis of the 3D object (depth of the object means how far away the object is from the street) and the determination of the volume that the note will be heard, when this note is playing. In Figure 4.14 (below) we see the Unity value received by interacting with the TouchOSC component, in this case as shown the /rotary2 signifies the volume/depth bar on the TouchOSC.

4. IMPLEMENTATION

```
public override void OnOSCMessageReceived(UniOSCEventArgs args)
{
    if(transformToMove == null) return;
    OscMessage msg = (OscMessage)args.Packet;
    if(msg.Data.Count < 1) return;

    // y1 = depth/position z of the selected object
    // transformToMove1 = selected GameObject

    float x = transformToMove.transform.position.x;
    float y = transformToMove.transform.position.y;
    float y1 = transformToMove2.transform.position.y;
    float z = transformToMove.transform.position.z;

    switch (movementMode) {
        case Mode.Screen:

            if (tag == "TreeL" )
            {
                y1 = 5 + ( 20 * (float)msg.Data[0]);
            }
            else if ( tag == "HouseL")
            {
                y1 = 1 + (24 * (float)msg.Data[0]);
            }
            else if ( tag == "SchoolL")
            {
                y1 = 20 + ((float)24.8 * (float)msg.Data[0]);
            }
            else if ( tag == "BalconyL")
            {
                y1 = (float)1.4 + ((float)24 * (float)msg.Data[0]);
            }
            else if (tag == "DoorL")
            {
                y1 = (float)1.4 + ((float)24 * (float)msg.Data[0]);
            }
            else if (tag == "WindowL")
            {
                y1 = (float)1.63 + ((float)23.95 * (float)msg.Data[0]);
            }
        }
    }
```

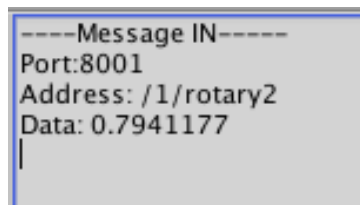


Figure 4.14: Depth / Volume Script

- Delete Script

This script(Figure 4.15) is a UniOSC script, as it receives real-time information from the TouchOSC's "delete note" button (Figure 3.26), which is installed on the user's mobile device and it's activated when the user selects one of the objects that

contain it as a component.

```
public override void OnOSCMessageReceived(UniOSCEventArgs args)
{
    if (transformToMove == null) return;
    OSCMessage msg = (OSCMessage)args.Packet;
    if (msg.Data.Count < 1) return;

    float ms = (float)msg.Data[0];
    if (ms < 0.01)
    {
        Destroy(gameObject);
        Destroy(transformToMoveParent);
    }
}
```

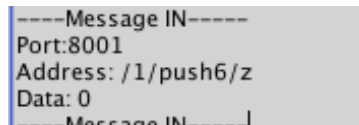


Figure 4.15: Delete Script

In this script, the information obtained from a TouchOSC button (value 0-1) is translated into the decision to delete or not the selected note and the corresponding object. A deleted note and the corresponding object cannot be restored. Figure 4.15 (below) shows the Unity value received by interacting with the TouchOSC component, in this case as shown the /push6/z signifies the delete button on the TouchOSC.

- **Generate Script**

This script (Figure 4.16) is a UniOSC script, as it receives real-time information from the TouchOSC's "generate note" button (Figure 3.27), which is installed on the user's mobile device and it's activated when the user selects one of the objects that contain it as a component. The selected buttons of the DAW interfaces generate the appropriate geometry as explained in Chapter 3. When a button in the DAW interface is pressed generating geometry, then the script is enabled and subsequently, when a TouchOSC button is pressed, the information obtained from a button (value 0-1) belonging on the TouchOSC interface is translated into the decision to generate or not a new note of the type which the selected button is representing. In this thesis, we have the decision to associate the piano with the

4. IMPLEMENTATION

```
public override void OnOSCMessageReceived(UniOSCEventArgs args)
{
    OsdMessage msg = (OsdMessage)args.Packet;
    if (msg.Data.Count < 1) return;

    // if (MouseManager.GetComponent<MouseManager>().selectedObject == gameObject)

    float ms = (float)msg.Data[0];
    if (ms > 0.99)
    {
        if (gameObject.tag == "GenerateTreeL")
        {
            clone = Instantiate(prefab, new Vector3(0, 0, 5), Quaternion.Euler(90, 0, 0)) as GameObject;
            clone1 = Instantiate(prefab1, new Vector3(135, 238.8f, 2540.2f), Quaternion.Euler(0, 0, 0)) as GameObject;
            clone1.GetComponent<UniOSC.UniOSCMoveGameObject>().transformToMove1 = clone.transform;
            clone1.GetComponent<UniOSC.UniOSCMoveGameObject1>().transformToMove1 = clone.transform;
            clone1.GetComponent<UniOSC.UniOSCMoveGameObject2>().transformToMove2 = clone.transform;
            clone1.GetComponent<UniOSC.UniOSCMoveGameObject5>().transformToMoveParent = clone;
            clone1.GetComponent<UniOSC.UniOSCScaleGameObject>().transformToScale1 = clone.transform;
            clone1.GetComponent<Transform>().tag = "TreeL";
            clone1.GetComponent<music>().TRANSFORM1 = clone.transform;
            clone1.GetComponent<AudioSource>().panStereo = -1;
        }
        else if (gameObject.tag == "GenerateSchoolR")
        {
            clone = Instantiate(prefab, new Vector3(23.1f, 0, -30), Quaternion.Euler(0, 0, 0)) as GameObject;
            clone1 = Instantiate(prefab1, new Vector3(242, -174, 2540.2f), Quaternion.identity) as GameObject;
            clone1.GetComponent<UniOSC.UniOSCMoveGameObject>().transformToMove1 = clone.transform;
            clone1.GetComponent<UniOSC.UniOSCMoveGameObject1>().transformToMove1 = clone.transform;
            clone1.GetComponent<UniOSC.UniOSCMoveGameObject2>().transformToMove2 = clone.transform;
            clone1.GetComponent<UniOSC.UniOSCMoveGameObject5>().transformToMoveParent = clone;
            clone1.GetComponent<UniOSC.UniOSCScaleGameObject>().transformToScale1 = clone.transform;
            clone1.GetComponent<Transform>().tag = "SchoolR";
            clone1.GetComponent<music>().TRANSFORM1 = clone.transform;
            clone1.GetComponent<AudioSource>().panStereo = 1;
        }
    }
}
```



-----Message IN-----
Port:8001
Address: /1/push7/z
Data: 0
Message IN

Figure 4.16: Generate Script

window, the flute with the tree, the xylophone with the balcony, the bass with the door, the cello with the building and the violin with the larger scale building. Each specific note is associated with the generation of a specific architectural object. The selected instrument which plays the particular note associated with the geometry generated (piano, flute, bass, xylophone, violin, cello) and the side of the street this note belongs to (either left or right speaker) signifying an architectural element of the actual 3D urban street (either left or right side of the street respectively) for which a note is being played. Figure 4.16 (below) shows the Unity value received by interacting with the TouchOSC component, in this case as shown the /push7/z signifies the generate note button on the TouchOSC.

- Mute Left

This script (Figure 4.17), which is included as a component in the selected Mute button of the DAW interface, mutes the Audiosources of notes which are placed on

the left speaker. This script is enabled or disabled when that "Mute" Left button (Figure 3.9) is pressed by the user.

```
// ----- Mute Left Script -----
void Update()
{
    // if (hitObject.tag == "HouseR" | hitObject.tag == "HouseL" | hitObject.tag == "TreeL" | hitObject.tag == "TreeR"
    if (Input.GetMouseButton(0) && MouseManager.GetComponent<MouseManager>().selectedObject == gameObject )
    {

        count = count + 1;
        if ( count % 2 != 0 )
        {
            Trees = GameObject.FindGameObjectsWithTag("TreeL");
            Houses = GameObject.FindGameObjectsWithTag("HouseL");
            Schools = GameObject.FindGameObjectsWithTag("SchoolL");
            Windows = GameObject.FindGameObjectsWithTag("WindowL");
            Balconies = GameObject.FindGameObjectsWithTag("BalconyL");
            Doors = GameObject.FindGameObjectsWithTag("DoorL");

            for (int i = 0; i < Trees.Length; i++)
            {
                Trees[i].GetComponent<AudioSource>().enabled = true;
            }
            for (int i = 0; i < Houses.Length; i++)
            {
                Houses[i].GetComponent<AudioSource>().enabled = true;
            }
            for (int i = 0; i < Schools.Length; i++)
            {
                Schools[i].GetComponent<AudioSource>().enabled = true;
            }
            for (int i = 0; i < Windows.Length; i++)
            {
                Windows[i].GetComponent<AudioSource>().enabled = true;
            }
            for (int i = 0; i < Doors.Length; i++)
            {
                Doors[i].GetComponent<AudioSource>().enabled = true;
            }
            for (int i = 0; i < Balconies.Length; i++)
            {
                Balconies[i].GetComponent<AudioSource>().enabled = true;
            }
        }
    }
}
```

Figure 4.17: Mute Left Script

4. IMPLEMENTATION

- Mute Right Script

This script (Figure 4.18), which is included as a component in the selected Mute button of the DAW interface, mutes the Audiosources of notes which are placed on the right speaker. This script is enabled or disabled when that "Mute" Right button (Figure 3.9) is pressed by the user.

```
// Speed: 1000000000 per frame
void Update()
{
    Debug.Log(count);
    if (Input.GetMouseButton(0) && MouseManager.GetComponent<MouseManager>().selectedObject == gameObject)
    {
        count = count + 1;
        Debug.Log(count);
        if (count % 2 != 0)
        {
            Trees = GameObject.FindGameObjectsWithTag("TreeR");
            Houses = GameObject.FindGameObjectsWithTag("HouseR");
            Schools = GameObject.FindGameObjectsWithTag("SchoolR");
            Windows = GameObject.FindGameObjectsWithTag("WindowR");
            Balconies = GameObject.FindGameObjectsWithTag("BalconyR");
            Doors = GameObject.FindGameObjectsWithTag("DoorR");

            for (int i = 0; i < Trees.Length; i++)
            {
                Trees[i].GetComponent<AudioSource>().enabled = true;
            }
            for (int i = 0; i < Houses.Length; i++)
            {
                Houses[i].GetComponent<AudioSource>().enabled = true;
            }
            for (int i = 0; i < Schools.Length; i++)
            {
                Schools[i].GetComponent<AudioSource>().enabled = true;
            }
            for (int i = 0; i < Windows.Length; i++)
            {
                Windows[i].GetComponent<AudioSource>().enabled = true;
            }
            for (int i = 0; i < Doors.Length; i++)
            {
                Doors[i].GetComponent<AudioSource>().enabled = true;
            }
            for (int i = 0; i < Balconies.Length; i++)
            {
                Balconies[i].GetComponent<AudioSource>().enabled = true;
            }
        }
    }
}
```

Figure 4.18: Mute Right Script

- Music Script

This script(Figure 4.19), which is included as a component of any existing note, transforms the AudioSource's values of the note (pitch, time, duration, volume, stereo pan left / right) based on the architectural axes (height, position x, length, depth, side of the street left / right) of the corresponding architectural object. The result of this processing is a new sound based on the original Audio that the Audiosource contains.

```
// Update is called once per frame
void Update()
{
    if (TRANSFORM1 != null)
    {
        posz = TRANSFORM1.transform.position.z;
    }
    AUDIO = GetComponent();
    float vol;
    if (tag == "TreeL")
    {
        vol = 1f - ((posz - 5f) / 20f);
        AUDIO.volume = vol;
    }
    else if (tag == "HouseL")
    {
        vol = 1f - ((posz - 1f) / 24f);
        AUDIO.volume = vol;
    }
    else if (tag == "School")
    {
        vol = 1f - ((posz - 20f) / 24.8f);
        AUDIO.volume = vol;
    }
    else if (tag == "BalconyL")
    {
        vol = 1f - ((posz - 1.4f) / 24f);
        AUDIO.volume = vol;
    }
    else if (tag == "DoorL")
    {
        vol = 1f - ((posz - 1.4f) / 24f);
        AUDIO.volume = vol;
    }
    else if (tag == "WindowL")
    {
        vol = 1f - ((posz - 1.63f) / 23.95f);
        AUDIO.volume = vol;
    }
    else if (tag == "TreeR")
    {
        vol = 1f - (((posz * (-1f)) - 20f) / 20f);
        AUDIO.volume = vol;
    }
    else if (tag == "HouseR")
    {
        vol = 1f - (((posz * (-1f)) - 20f) / 24f);
        AUDIO.volume = vol;
    }
}
```

Figure 4.19: Music Script

4. IMPLEMENTATION

• Mouse Manager Script

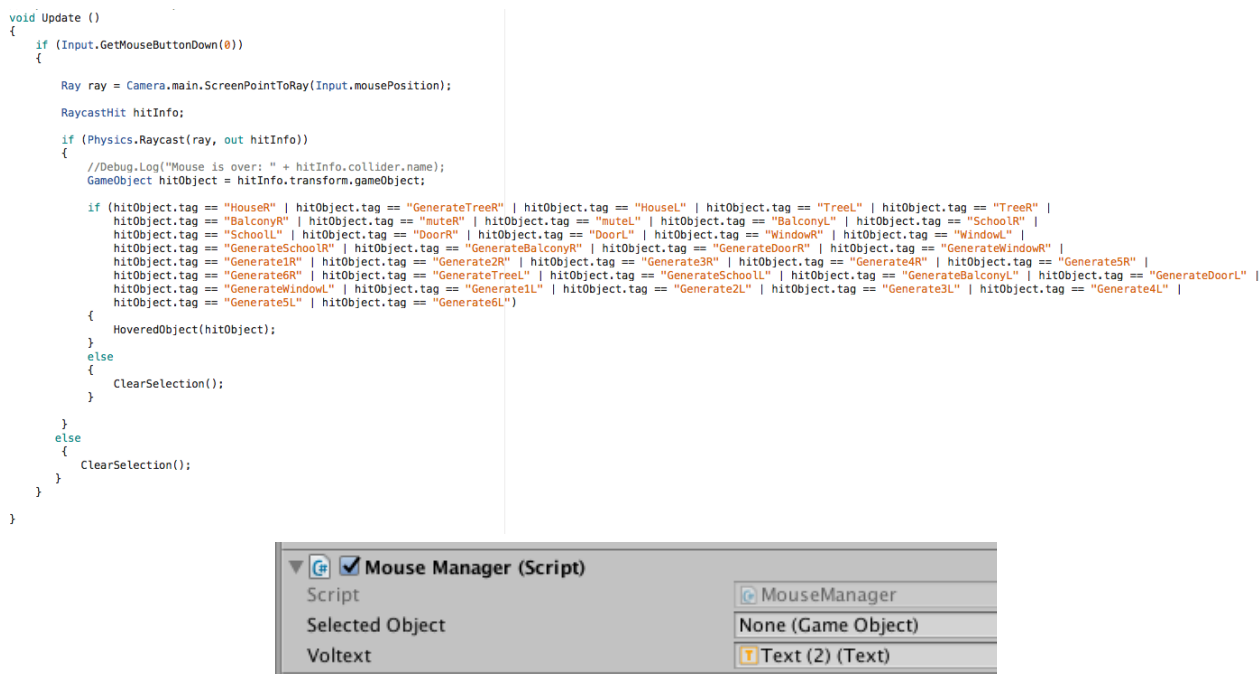


Figure 4.20: Mouse Manager Script

This script(Figure 4.20), which is included as a component in the Mouse Manager Unity object (not apparent in the interface) performs the following action. When the user clicks on on any note or on any button, it enables all UniOSC scripts associated with a note which is connected with an architectural object. If a different element of the interface is pressed, then the scripts associated with the previous selection are disabled. This script is enabled on awake and remain enabled throughout the platform used. The UniOSC scripts receive real-time user input from the TouchOSC interface. In Unity, as shown in Figure 4.20 below, the script changes the selected object from none to a note or a 'generate' button, enabling the appropriate scripts.

- Bar Position Script

```
public override void OnOSCMessageReceived(EventArgs args)
{
    if (transformToMove == null) return;
    OscMessage msg = (OscMessage)args.Packet;
    if (msg.Data.Count < 1) return;

    float y = transformToMove.transform.position.y;

    float y1 = (float)((float)-15.2 + ((float)431.2 * (float)msg.Data[0]));
    pos1 = camera.transform.position;
    pos1[0] = y1;
    camera.GetComponent<Transform>().position = pos1;

    y = 66 + (2868 * (float)msg.Data[0]);

    pos = transformToMove.transform.position; pos[0] = y;
    transformToMove.transform.position = pos;
}
```

```
-----Message IN-----
Port:8001
Address: /1/fader5
Data: 0.4423077
```

Figure 4.21: Bar Position Script

This script (Figure 4.21) is a UniOSC script associated with the play-bar, as it receives real-time information from the TouchOSC, which is installed on the user's mobile device and it is enabled on awake (start of the system) and remain enabled when the platform is used. In this script, the information obtained from a TouchOSC fader (value 0-1) is translated into the position in the x axis of the camera and the time at which the play-bar is placed, based on user's input. Figure 4.21 (below) shows the Unity value received by interacting with the TouchOSC component, in this case as shown the fader5 signifies the bar/camera position on the TouchOSC.

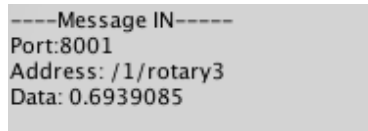
4. IMPLEMENTATION

- Bar Speed Script

```
public override void OnOSCMessageReceived(UniOSCEventArgs args)
{
    if(transformToMove == null) return;
    OscMessage msg = (OscMessage)args.Packet;
    if(msg.Data.Count <1) return;

    float bpm = transformToMove.GetComponent<Metronome>().speed;
    bpm = (float)(6 + (6 * (float)msg.Data[0]));
    transformToMove.GetComponent<Metronome>().speed = bpm;
    float t = 10 * transformToMove.GetComponent<Metronome>().speed;

    bpmtext.text = t.ToString("F0");
}
public void OnUpdate()
{
    float t = 10 * transformToMove.GetComponent<Metronome>().speed;
    bpmtext.text = t.ToString("F0");
}
}
```



```
-----Message IN-----
Port:8001
Address: /1/rotary3
Data: 0.6939085
```

Figure 4.22: Bar Speed Script

This script(Figure 4.22) is a UniOSC script, as it receives real-time information from the TouchOSC, which is installed on the user's mobile device. It is enabled on awake and remain enabled when the platform is used. In this script, the information obtained from a TouchOSC rotary (value 0-1) is translated into the speed that the play-bar moves and concurrently, the camera moves too. As a result, the bpm rate changes (beats per minute) and the camera follows simultaneously the changes occurring at the play-bar. When the bpm rate increases, the camera movement follows simultaneously becoming faster. Figure 4.22 (below) shows the Unity value received by interacting with the TouchOSC component, in this case as shown the rotary3 signifies the bpm/bar speed on the TouchOSC.

- Camera's height Script

```

public override void OnOSCMessageReceived(UniOSCEventArgs args)
{
    if(transformToMove == null) return;
    OSCMessage msg = (OSCMessage)args.Packet;
    if(msg.Data.Count < 1) return;

    switch (movementMode) {
        case Mode.Screen:

            float y = 2 + 21 * (float)msg.Data[0];

            //pos = new Vector3(x,y,Camera.main.nearClipPlane+nearClipPlaneOffset);
            pos = transformToMove.transform.position; pos[1] = y ; //pos[1] = y;
            //pos[2] = Camera.main.nearClipPlane + nearClipPlaneOffset;
            transformToMove.transform.position = pos;

            break;
        case Mode.Relative:
            break;
    }
}

```

```

----Message IN----
Port:8001
Address: /1/fader9
Data: 0.5255102

```

Figure 4.23: Camera's height Script

This script (Figure 4.23) is a UniOSC script, as it receives real-time information from the TouchOsc, which is installed on the user's mobile device. It is enabled on awake and remain enabled when the platform is used. In this script, the information obtained from a TouchOSC fader (value 0-1) is translated into the height that the camera is placed when the walkthrough of the urban street is visualized in real-time. Figure 4.23 (below) shows the Unity value received by interacting with the TouchOSC component, in this case as shown the fader9 signifies the camera's height on the TouchOSC.

4. IMPLEMENTATION

4.6 TouchOsc Controller

TouchOsc (Figure 4.24) is connected to the Archimusic3D platform (Unity's project), via UniOSC Unity's Asset (Figure 2.10). In order to control the UniOSC scripts, the template that appeared in the Figure 4.24 was created on the TouchOSC editor which a tool that enables the creation of the TouchOSC interface. This template consists of an active page and contains two buttons, six faders and two potentiometers.

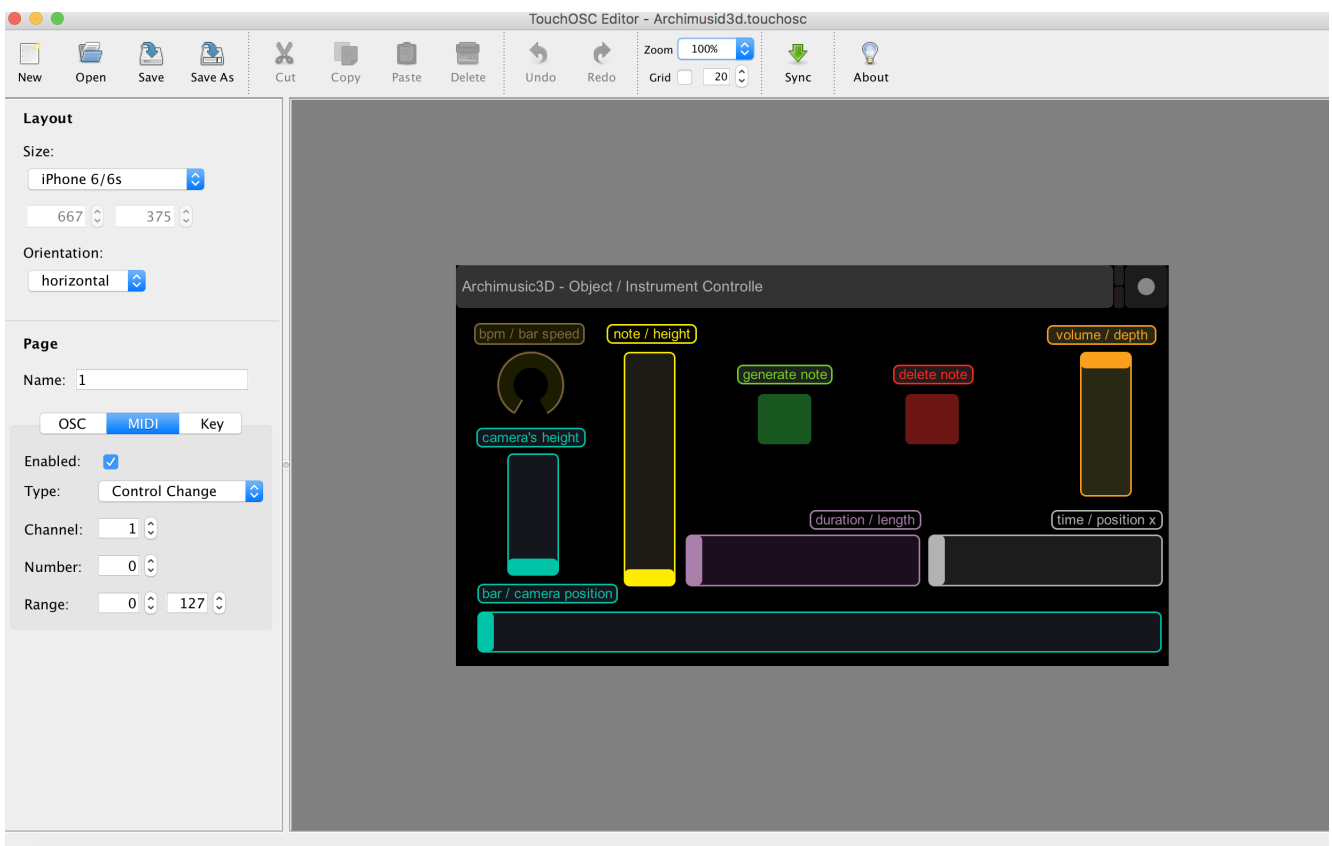


Figure 4.24: Archimusic3D Template TouchOSC Editor

- position x / time fader

Shown on the left side of the Figure 4.25 the grey fader from which the enabled UniOSC Position / Time script of the selected note (as described above) receives values from 0 to 1. Shown on the right side of the Figure 4.25 the Unity's Position / Time script (in Csharp) receives data from the TouchOSC's time/position x fader, via UniOSC Unity's asset. Fader 2 represents the time/position x fader of the TouchOSC.



Figure 4.25: position x /time fader

4. IMPLEMENTATION

- duration / length fader

Shown on the left side of the Figure 4.26, the purple fader from which the enabled UniOSC Length / Duration script of the selected note receives values from 0 to 1. Shown on the right side of the Figure 4.26 the Unity's Length / Duration script (in CSharp) which receives data from the TouchOsc's duration / length fader, via UniOSC Unity's asset. Fader 1 represents the duration / length fader of the TouchOSC.

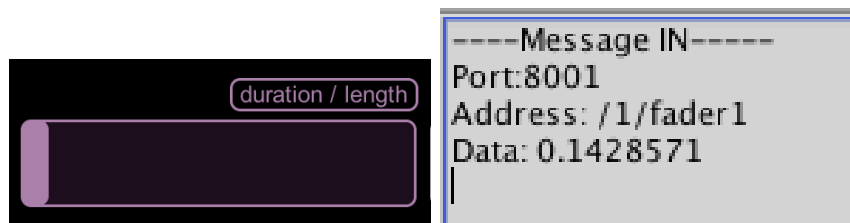


Figure 4.26: duration / length fader

- note / height fader

Shown on the left side of the Figure 4.27, the yellow fader from which the enabled UniOSC Height / Pitch script of the selected note receives values from 0 to 1. Shown on the right side of the Figure 4.27 the Unity's Height / Pitch script (in Csharp) receiving data from a TouchOsc's note / height fader, via UniOsc Unity's asset. Fader 3 represents the note / height fader of the TouchOSC.

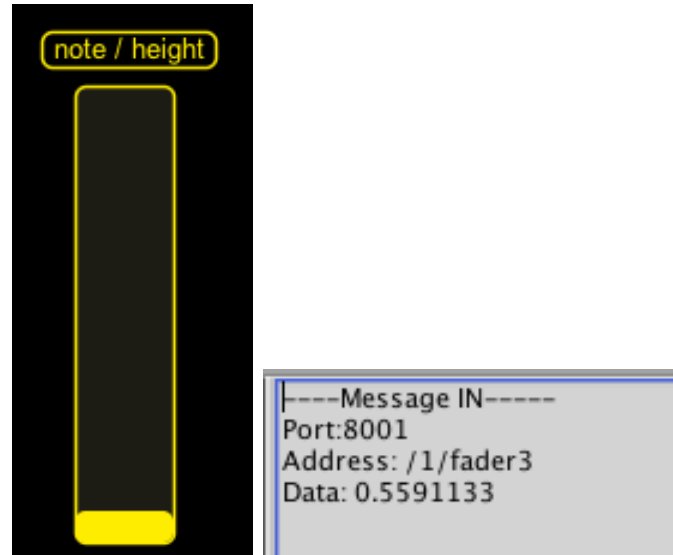


Figure 4.27: note / height fader

- volume / depth fader

Shown the left side of the Figure 4.28 the orange fader from which the enabled UniOsc Depth / Volume script of the selected note receives values from 0 to 1. Shown on the right side of the Figure 4.28 the Unity's Depth / Volume script (in Csharp) receiving data from the TouchOsc's volume / depth fader, via UniOsc Unity's asset. Fader3 represents the volume / depth fader of the TouchOSC.

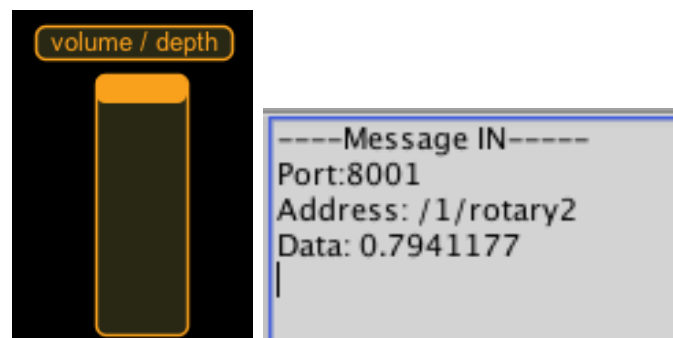


Figure 4.28: volume / depth fader

4. IMPLEMENTATION

- delete note button

Shown on the left side of the Figure 4.29 the red button that receives information from the enabled UniOSC Delete script of the selected note receiving values from 0 to 1. Shown on the right side of Figure 4.29 the Unity's Delete script (in Csharp) receives data from the TouchOsc's delete button, via UniOsc Unity's asset. Push6/z represents the delete fader of the TouchOSC.

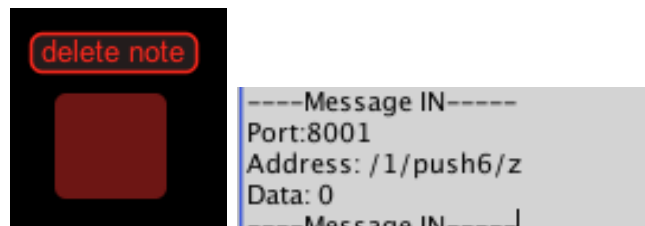


Figure 4.29: delete note button

- generate note button

Shown on the left side of the Figure 4.30 the green button that receives information from the enabled UniOsc Generate script of the selected button values from 0 to 1. Shown on the right side of the Figure 4.30 the Unity's Generate script (in Csharp) receiving data from a TouchOsc's generate button, via UniOsc Unity's asset. Push7/z represents the generate note fader of the TouchOSC.

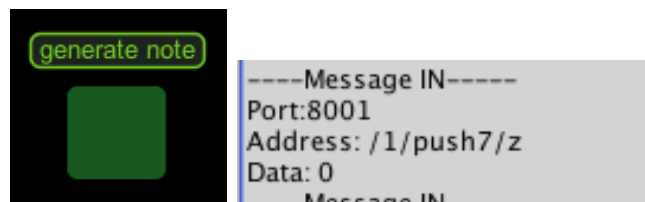


Figure 4.30: generate note button

- bar / camera position fader

Shown on the left side of the Figure 4.32 the blue fader from which the enabled UniOsc Bar Position script receives values from 0 to 1. Shown on the right side of the Figure 4.32 the Unity's Bar Position script (in Csharp) receiving data from

the TouchOsc's bar / camera position fader, via UniOsc Unity's asset. Fader5 represents the bar / camera position fader of the TouchOSC.



Figure 4.31: bar / camera position fader

- bpm / bar speed potentiometer

Shown on the left side of the Figure 4.32 the brown potentiometer from which the enabled UniOsc Bar Speed script receives values from 0 to 1. Shown on the right side of the Figure 4.32 the Unity's Bar Speed script (in Csharp) receiving data from the TouchOsc's bpm / bar speed potentiometer, via UniOsc Unity's asset. Rotary3 represents the bar / camera position fader of the TouchOSC.

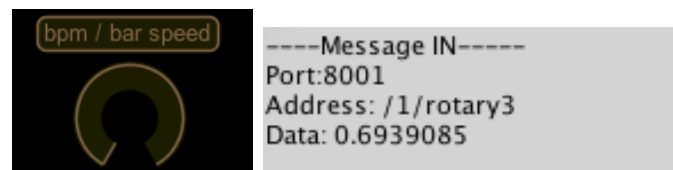


Figure 4.32: bpm / bar speed potentiometer

- camera's height fader

Shown on the left side of the Figure 4.33 the blue fader from which the enabled Camera's height script receives values from 0 to 1. Shown on the right side of the Figure 4.33 the Unity's Camera's height script (in Csharp) receiving data from the TouchOsc's camera's height fader, via UniOsc Unity's asset. Fader9 represents the bar / camera position fader of the TouchOSC.

4. IMPLEMENTATION



Figure 4.33: camera's height fader

Chapter 5

Conclusion

5.1 Demo

In order to test the platform, in terms of functionality and efficiency, the urban street that appeared in the Figure 5.1 was created. The music that emerged from the translation of this street can be characterised as demonstrating disharmony. This is because the musical outcome did not demonstrate any melodic element and did not sound as a musical piece, being unstable and without tempo. In the attempt to harmonize this music, musical rules were used such as Gestalt laws. These laws are musical techniques which indicate which notes need to be played consecutively, often including repetition of notes in specified intervals, so that the musical piece sounds melodically continuous. The result can be characterized as more harmonic music which was then translated to a more refined urban street as shown in the Figure 5.2. We can claim that we have a more refined street because certain elements such as trees were added (added notes of different volume) and the gaps between the buildings were minimized because 'silence' moments were removed in an effort to have a continuous music piece by making the width of the buildings larger. Also, because in the initial scene there were balconies on every floor, there was musical confusion because various notes were played at the same time (balconies signifying notes). By removing balconies, there were individual notes played for each building based on the existent number of balconies and, therefore, not many notes played together, thus, creating a melody.

Another effort, that we made, was to translate a famous music song to an urban street but this failed because to do this it is necessary for the designer to use more than the

5. CONCLUSION

seven notes (semitones and more octaves) played by the musical instruments, that are now available in the platform.

5.1 Demo

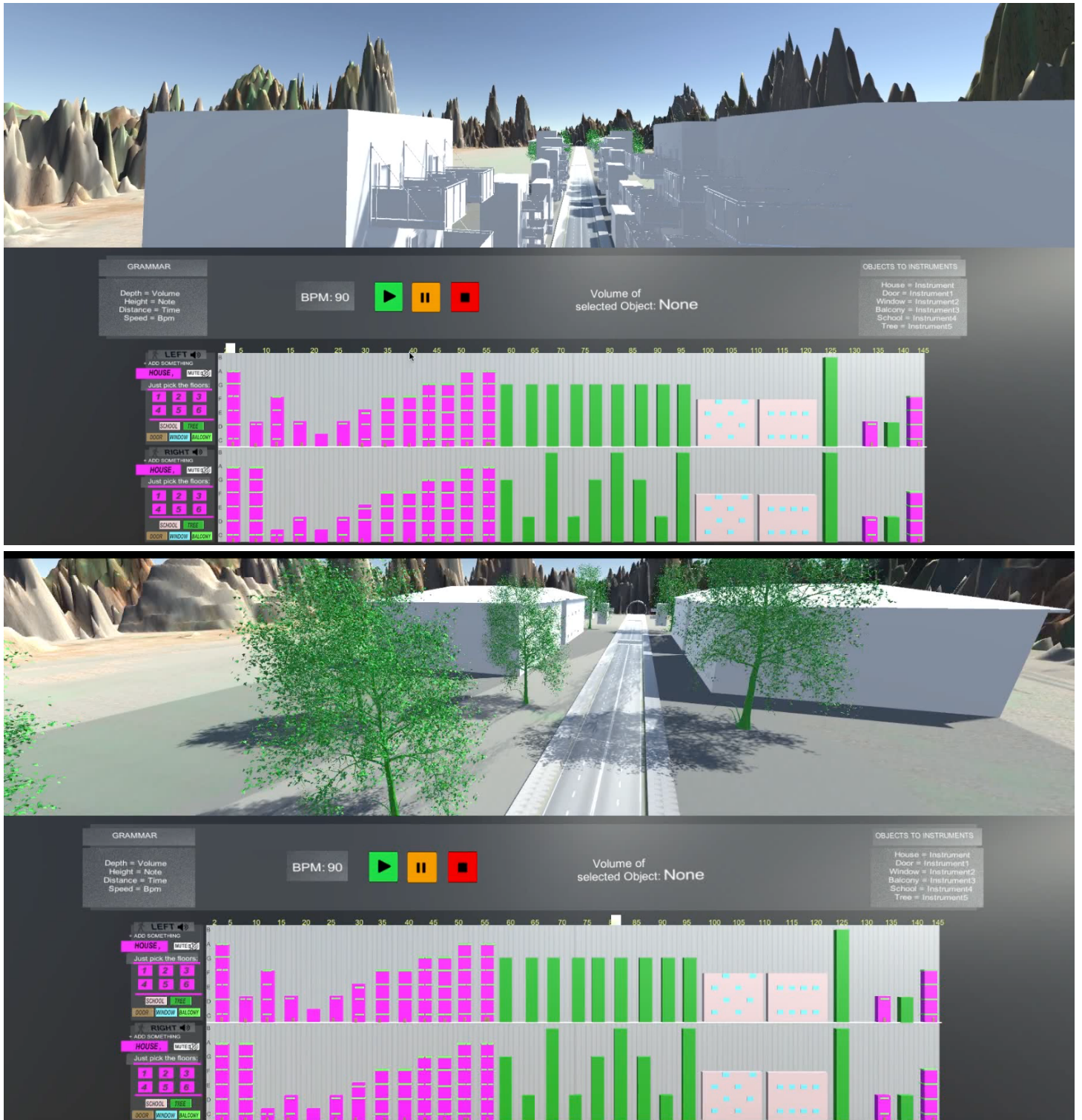


Figure 5.1: Demo - before the Transformations

5. CONCLUSION

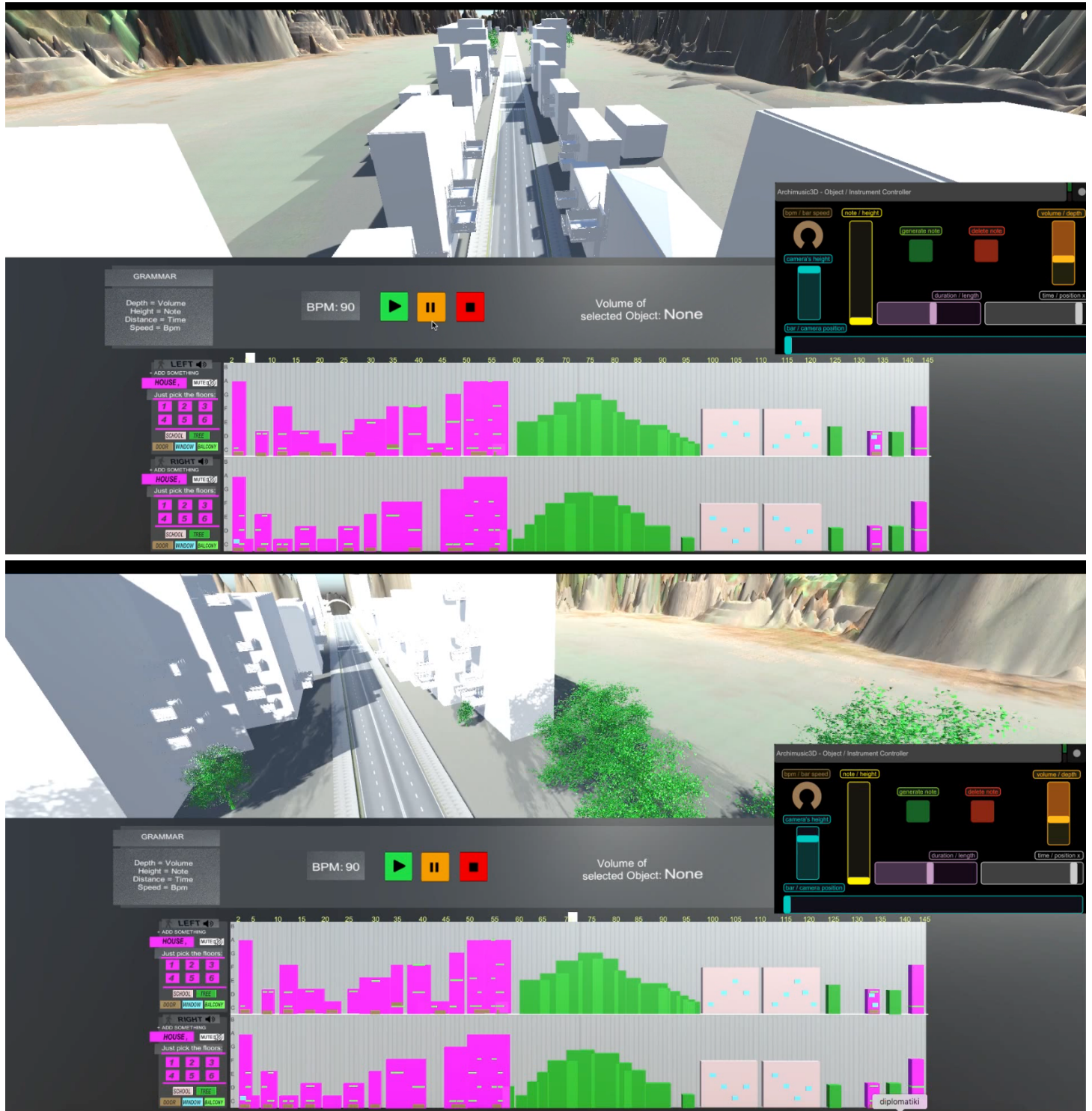


Figure 5.2: Demo - after the Transformations

5.2 Summary

This thesis demonstrated that there is a significant potential in combining the fields of composition in music and architecture through the use of information technology. Merging the two fields has the potential to release new, innovative tools for urban designers by uncovering aspects of an urban design not apparent based on the standard, vision-based digital design tools, introducing an additional modality of sound and music. This thesis describes an innovative tool developed at the Technical University of Crete, through which an urban designer can work on the music transcription of a specific urban environment applying music compositional rules and filters in order to identify discordant entities, highlight imbalanced parts and make design corrections.

The first step was to match the various parameters and variables of the urban ecosystem to those of its musical footprint. The Digital Media Lab from the School of Architecture, Technical University of Crete, provided a translation method between musical and architectural elements for compositional experiments assisting and refining urban design. Street facades, the fundamental imprint of our urban environment, are first broken down to their main semantic elements. These elements have properties, such as position and size in a 3D (XYZ) system, which are transcribed into sonic data: length in the X-axis is mapped to note appearance in time and note duration (tempo), height in the Y-axis is mapped to note value (pitch) and depth in the Z-axis is mapped to volume. Different elements correspond to different timbre and voids to pauses (silence). In this thesis, we have made the decision to associate the piano with the window, the flute with the tree, the xylophone with the balcony, the bass with the door, the cello with the building and the violin with the larger scale building. Acoustic data encoded from the built environment provides a valuable platform on which discordant entities can be more easily identified and also imbalanced parts get highlighted. The cognitive process of analyzing today's chaotic urban eco-system has been augmented with a new dimension of understanding but also intervening through its musical footprint.

In this thesis, we present a complete digital tool named Archimusic3D which utilizes the translation method between music and architecture for compositional experiments in order to assist urban designers in 1) identifying urban dissonances, 2) refining their design using musical rules and 3) presenting the output both visually and acoustically. The presented platform comprises of three scenes, which compile the three main parts of

5. CONCLUSION

the system's interface; e.g., the 3D scene, the Digital Audio Workstation (DAW) scene and the TouchOsc mobile controller. While interacting with the 3D scene, the user is able to navigate a 3D street in order to observe the 3D graphical depiction of an urban environment. The second scene is the implementation of a Digital Audio Workstation (DAW) program, in which the user is able to see the musical footprint of the street printed as a MIDI (Musical Instrument Digital Interface) file in order to listen to it and process it. Processing is conducted via a third tool representing a TouchOsc mobile controller, which is a virtual mobile console containing faders, potentiometers and buttons. When the users converts the 3D scene to a musical footprint, they submit the reciprocal transformations which may be applied to the musical part and then, they are able to navigate a newly refined street.

5.3 Evaluation

In order to understand whether Archimusic3D provides a reliable simulation of a DAW program, an expert sound designer was invited named Mr. Kordolaimis Theodoros, currently an undergraduate student at the Department of Music Technology and Acoustics Engineering of the Technological Educational Institute of Crete, to try the platform and evaluate it.

5.3.1 Advantages

- Based on his evaluation, the Achimusic3D platform is a successful simulation of a DAW program, in the sense that it provides most of the basic functions that a professional DAW program provides such as play, pause, stop , mute left and right speaker, bpm play-bar, MIDI protocol, OSC protocol. Furthermore, the designer felt familiar with the graphical interface of the DAW interface, because it is similar to the graphical interface of standard DAW programs. A new music piece could be created in this platform.
- The selection of the TouchOSC mobile application as a processing console is marked as successful, in the sense that it provides most of the basic functions, that a professional sound processing console provides such as fast real-time response and transformations precision.

- Transformations on the notes function correctly, in the sense that when the musical axes of a note are modified, this new note will sound as the actual note would sound, if played by the musical instrument that produces it. For example, let's assume that a C, played by a piano, with duration 2 secs and volume 1 is converted to a D with duration of 2 secs and volume 1. This new note, although it is the same C note played by a piano with modified pitch, sounds exactly the same with the real D played by the piano.

5.3.2 Disadvantages

- The notes are played even if the play-bar is over them although the system is in stop or pause mode. The correct way is that the notes should be heard only when the play-bar is over them and it is in motion. Furthermore, when a note stops to play, a "click" noise sound is heard.
- In relation to the mute button, is not visible whether this button is switched on or off, which means that there is a need for an indication of the mute button's status (whether it is enabled or not) such as the existence of a Volume Unit (VU) meters which are sound intensity indicators.
- When a note is being played at the same time the user modifies its pitch, the note will remain the same as it was before its pitch was modified. The new note will be heard during the next play of this note. Furthermore, the user cannot select more than one note at a single transformation.

5.4 Future Work

Archimusic3D presented in this thesis is a prototype system and represents the simulation of an urban street containing six different types of 3D objects and the simulation of its musical footprint (which contains six different musical instruments), ready for reciprocal transformations. Obviously, an urban street could contain additional objects such as benches, traffic lights or bridges, adding on to the existing ones in our platform, e.g. trees, buildings, balconies, doors, windows. Therefore, a significant additional capability to be added as future work is the ability to import any architectural element (in .fbx format),

5. CONCLUSION

created in a graphics design software such as REVIT or created by a different designer downloaded from the internet. Respectively, a musical piece can be composed and played employing a variety of musical instruments instead of the six musical instruments included in Archimusic3D to be utilized to translate architectural elements to music. Another important addition capability would be the ability of the user to import any note played by any musical instrument or voice (in .wav, .mp3 or .mp4 format), created by a composer or designer in DAW software such as Ableton or downloaded from the internet. In Archimusic3D presented in this thesis, only one octave without semitones (C, D, E, F, G, A, B) is available for each of the six different musical instruments. In essence, there is a need for a complete translation of all the elements which are included in an urban environment into music and further utilization of the sound's properties in architecture. The term "further utilization of sound's properties" refers to the additions, in the DAW program, of the whole range of musical tools such as, for instance, compressor, as well as musical filters such as reverb. Forming a complete grammar of translation between the whole range of architectural elements and the whole range of sound capabilities would provide a complete framework for processing reciprocal transformations between architecture and music.

An important addition to the platform is the capability of creating and saving new Archimusic3D's projects, as well as the capability of exporting the new refined urban street in .fbx format commonly used by architectural design tools programs and the final corresponding musical pieces in .wav, .mp3, .mp4 format, commonly used by media players. Many of the aforementioned issues could be solved if instead of including the simulation of the DAW program as implemented in Archimusic3D, the platform was connected to an already existing professional DAW program such as Ableton or, even better, if it was possible for the platform to be connected with any professional DAW program.

Furthermore, the platform's functionality can be extended to include different types of scene data. The first would include the ability to import and convert a 2D image in .jpg format to a music footprint automatically translating the architectural content of the image to sounds. The second useful scene capability would be the creation of an entire urban city consisting of a combination of urban streets and open spaces and a corresponding long-duration musical footprint consisting of a combination of the corresponding musical footprints of these urban streets in order to form the musical translation

of an entire city. If this is implemented, the user could select desired paths through the city for translation.

5. CONCLUSION

References

- [1] Parthenios, P.: Adopting a cross disciplinary approach to propose a new design tool for discovering urban design discordances. In: Proceedings of the 8th Study Day of INU - Italian National Institute of Urban Planning - Policies for Italian cities. Naples. (2014) [1](#), [10](#)
- [2] Parthenios, P., Mania, K., Chatzopoulou, N., Petrovski, S.: Reciprocal transformations between music and architecture as a real-time supporting mechanism in urban design. In: International Journal of Architectural Computing. (2016) [1](#), [2](#), [5](#), [26](#)
- [3] Xenakis-Upic:. https://www.youtube.com/watch?v=7_Gu0qDAys0 [4](#)
- [4] Lynch, K.: The image of the city. In: Cambridge, MA: The MIT Press. (1960) [10](#)
- [5] Novak, M.: The music of architecture: computation and composition. (2007) [10](#)
- [6] Xenakis, I.: Formalized music - thought and mathematics in composition. <http://archimusic.info/archimusicrepository/formalized-music/> (1963) [10](#)
- [7] Philips-Pavilion:. https://www.researchgate.net/figure/Philips-Pavilion-Metastaseis-B-Iannis-Xenakis-1953-1958_fig1_308759695 (1958) [10](#)
- [8] Sainte-Marie:. <https://www.flickr.com/photos/29727266@N02/6101167304> (1961) [10](#)
- [9] Composition-Wikipedia:. [https://en.wikipedia.org/wiki/Composition_\(visual_arts\)](https://en.wikipedia.org/wiki/Composition_(visual_arts)) [13](#)

REFERENCES

- [10] Counterpoint:. <https://music.stackexchange.com/questions/63368/what-determines-if-counterpoint-is-good-or-bad> 13
- [11] Counterpoint-Wikipedia:. <https://en.wikipedia.org/wiki/Counterpoint> 14
- [12] Ableton:. <https://www.ableton.com/en/education/> 14
- [13] DAW-Wikipedia:. https://en.wikipedia.org/wiki/Digital_audio_workstation 15
- [14] Osc-protocol:. <http://www.alexshill.com/projects/osc.htm> 15
- [15] OSC-Wikipedia:. https://en.wikipedia.org/wiki/Open_Sound_Control 15
- [16] Midi-protocol:. <https://www.instructables.com/id/What-is-MIDI/> 16
- [17] MIDI-Wikipedia:. <https://en.wikipedia.org/wiki/MIDI> 16
- [18] Unity-Wikipedia:. [https://en.wikipedia.org/wiki/Unity_\(game_engine\)](https://en.wikipedia.org/wiki/Unity_(game_engine)) 17
- [19] TouchOsc:. <https://hexler.net/software/touchosc> 18
- [20] UniOsc:. www.uniosc.monoflow.org/ 18
- [21] BIM:. <http://www.teetkm.gr/wp-content/uploads/2018/01/p1b.jpg> 51
- [22] BIM-Wikipedia:. https://en.wikipedia.org/wiki/Building_information_modeling 52
- [23] Revit:. <https://www.autodesk.com/products/revit/overview> 52
- [24] Unity-Manual-Transform:. <https://docs.unity3d.com/ScriptReference/Transform.html> 57
- [25] Unity-Manual-Audiosource:. <https://docs.unity3d.com/ScriptReference/AudioSource.html> 57
- [26] Unity-Manual-Script:. <https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html> 58