Technical University of Munich
Faculty of Civil Engineering and Geodesy
Department of Cartography
Prof. Dr.-Ing. Liqiu Meng

Procedural developing of a reconstruction of archaeological model Applied to the case study: Nymphaeum of the "sanduary di Diana" in Nemi, Italy

Abdullah Alattas

Master Thesis

Bearbeitung: 1.04.2014 – 30.09.2014

Studiengang: Cartography (Master)

Supervisor: **Dipl.-Ing. Stefan Peters**

Cooperation: Archaeologist Dr. Francesca Diosono

2014

Declaration of Authorship

Last name:	First name:

I declare that the work presented here is, to the best of my knowledge and belief, original and the result of my own investigations, except as acknowledged, and has not been submit-ted, either in part or whole, for a degree at this or any other University.

Formulations and ideas taken from other sources are cited as such. This work has not been published.

__________________	_____________________
Location	Date Signature

Abstract

The demand for using representations of places or buildings to provide visual information is increasing nowadays. These representations could be reconstructions of scaled real models or virtual models produced using CAD (Computer-Aided Design) tools. Recently, procedural modeling has been used to create building and cities because it reduces the amount of time and interventional effort of users. It is also considered a useful alternative due to the lowered cost of a model's construction. Especially for the archaeological field, it offers a great benefit for examining archeological hypotheses. Moreover, the shape grammar for the procedural modeling of CG Architecture (CGA shape) produces high quality visualizations of the geometric detail for the building. CGA are context sensitive and enable the user to define interactions between the objects of the hierarchical shape descriptions. CGA shapes are proven to generate massive urban models efficiently with an exceptional level of detail as will be demonstrate in some examples of previous modeling projects. This thesis aims to produce realistic virtual representation of a reconstructed Nymphaeum-part of the sanctuary of Diana in Nemi, Italy by using procedural modeling, a CAD system to increase the level of detail and to represent two different hypotheses. We first introduce an overview of the reconstruction of Nymphaum and provide an overview of modeling approaches that are used in archeological studies. Secondly, we defined the uncertainty principles that are used in archeological data. Next, we describe the elements of the building which were collected from the archeologist team in Nemi. For the implementation of the actual building shape, we determined the CGA rule for different architectural parts. By combining the CGA rules, a reconstruction of the whole site can be generated.

Table of Contents

List of Figures

Abbreviations

CAD	**C**omputer-**A**ided **D**esign
CGA	**C**omputer **G**enerated **A**rchitecture
CE	**C**ity**E**ngine
CVR	**C**ultural **V**irtual **R**eality
GIS	**G**eographic **I**nformation **S**cience
LC	**L**ondon **C**harter
R-VIS	**R**eliability **Vis**ualization
VRML	**V**irtual **R**eality **M**odeling **L**anguage
IT	**I**nformation **T**echnology
H/W	**H**ard**W**are
S/W	**S**oft**W**are
PDM	**P**roduct **D**ata **M**anagement
SAGE	**S**emi- **A**utomatic **G**round **E**nvironment
GMZ	**G**enerative **M**odeling **L**anguage
L-systems	**L**indenmayer **Systems**
OBJ	**O**bject file
Lo**D**	**L**evels of **D**etail
OGC	**O**pen **G**eospatial **C**onsortium
XML	E**x**tensible **M**arkup **L**anguage
KML	Keyhole Markup Language
COLLADA	**Colla**borative **D**esign **A**ctivity

I would like to thank my wife, Rawan Alkhalaf. She was always there, cheering me up, and stood by me through the good times and bad. Also, I would never have been able to finish my dissertation without the guidance of my committee members, help from friends, and support from my family.

1 Introduction

Virtual heritage, also known as 3D cultural heritage, uses information technology from data provided by archaeologists and historians to virtually represent art and architecture. These virtual representation impart valuable knowledge and information about unique locations and buildings that are difficult or costly to visit. Different forms of virtual representations are employed in different business areas such as urban planning, architecture, tourism and archeology. They are often used to study ancient heritage sites, display destinations, or are used in urban management settings to visualize development goals. Virtual heritage is helpful for the documentation and cataloguing of historical studies, experimental architecture, urban history, and also for edutainment of the public. The continued adoption of virtual heritage will thus be an important part of the continuation of communicating cultural ideas by allowing there documentation to be highly detailed and even interactive. This will allow greater access by academics and the public and thus drive further adoption of the techniques.

Different methodological approaches have been applied in the last 20 years that aim to create objects of Cultural Heritage, and to introduce a new field of Archaeology called Virtual Archaeology (Pescarin, 2009). The best approach to achieve a digital model with high detail is to combine different 3D modeling techniques (Guidi et al., 2008). Therefore, the 3D modeling is an extension of the activities of scholars and scientists in the area of cultural heritage. The rapid spread of virtual heritage use among academics and professionals has begun to replace previously used tools like 2D forms of plans and reconstructions. However, the process of generating these virtual representations is time and resource intensive. It requires several different types of equipment, software licenses and technical knowledge of simulating 3D environments. For this reason, automating of the generation process, using procedural modeling techniques, is becoming increasingly attractive because it creates heritage content in a quickly and accurately.

The virtual representation process goes through three steps: collecting the archeological data from different sources, 3D procedural modeling and performing computer simulations to gather different levels of information. In particular, the procedural modeling used for the ancient urban reconstruction of archaeological sites, is a method to create the geometrical object based on a set of predefined rules. In the beginning, procedural modeling was applied to produce complex geometries and textures. But recently, it has been used to generate different typologies using a scripting language of shape grammars. The description of the models is made in a progressive order through an additive process. Defining the parameters' wideness and relation in every step with the existing entities allows the user to control the whole project. After this, the user can program the script that contains an entities description, where the virtual geometry is automatically generated by the software such as CityEngine.

In general, this procedural method involves two steps: analytical step which are done by identifying shapes and rules, and the reproducing step which is to create the architectural and urban structures.

1.1 Motivation

Duo to the lack of representation of archeological studies to the viewers, many kinds of misunderstandings of archeological data has been found between scientists and students from different disciplines. The archeological scientists need a lot of time to collect and represent their data. By using the CAD system, the representation of the data costs a lot of money to create the model and it is really difficult to edit the content. The importantce of their work is to connect the current generation with the past of our cities. By providing a tool that helps to represent an archeologist's concepts and ideas in a short time would reduce this gap. The tools should be able to represent the data in an efficient way and should be allowed to edit the content in order to model different hypotheses.

1.2 Summary of related works

We will show some examples of previous work based on the CityEngine, which is a tool for the procedural modeling of buildings and sites. They are all different in aspects whose parts are not certain. The first example is the Puuc, a subtype of Mayan architecture, which is characterized by a veneer-over-concrete construction technique that results in geometric and repetitive facade structures. The key tool to express uncertainty in Puuc are the control parameters which are simply a set of shape attributes that are used to control the overall result of a building type by storing a range of possible values.

With the help of archaeologists it was possible to create a single set of rules for all the types of Puuc building. Each of the building types was generated in about 5 to 10 minutes by simple modifications of the control parameters of the grammar rule set. Also, there are rules to specify materials and textures, and some rules have been extended to generate other more complex buildings.

The second example is the reconstruction of Ancient Pompeii. It is a large scale reconstruction project and a large part of the site which has not been excavated yet. Only footprints are known so an extensive use of probabilistic rules were needed to extrude the buildings. Pompeii was an ancient Roman town destroyed in 79 AD and it was reconstructed based on footprints and drawings/sketches for selected building types provided by archeologists. 190 design rules were abstracted in order to model the complete city that includes streets and the placement of trees. The highest level of detail of the resulted model contains about 1.4 billion polygons.

The third example is Rome Reborn 2.0 which is an international project led by Prof. B. Frischer, University of Virginia. The purpose of the project is to create an interactive, 3D, digital model illustrating the urban development of ancient Rome. The first stage of the project is Rome Reborn 1.0 which created a model of Rome in the 4th century where all popular major buildings had already been erected. The famous Roman monuments such as the Colosseum and Circus Maximus were represented manually in detailed 3D models, that required several man-years of work by experts in archeology and computer graphics from all over the world.

For that, it would be overwhelming to use similar methods to reconstruct in similar detail the surrounding urban environment and the “fill- in” buildings. Therefore in the Rome Reborn 2.0 version, grammar-based procedural modeling was used. About 7000 domestic buildings have been reconstructed as well as 60% of ancient Rome’s temples which were modeled with the CityEngine. But unlike the Pompeii model, the Rome Reborn 2.0 model did not start with building footprints but rather with crude polyhedral volumes for each building.

The Rome Reborn 2.0 model mixes two types of models: the manually modeled monuments of the (Rome Reborn 1.0) and the procedural models. To keep the visual balance with those manually modeled monuments, the amount of detail in the procedural models actually had to be limited near these monuments.

The east wing of the Louvre has been reconstructed twice using procedural modeling. The aim was to enable examining and exploring how the Louvre palace might have looked like in the past. The facade generation of the proposals was created using procedural rules formulated based on building plans and elevations obtained from the Louvre archives. These rules were written as CGA files in the CityEngine software package.

However more details about the projects and modeling methods for each example are given in the related work section.

1.3 Problem Definition

The reconstruction of virtual heritage may be considered as the key to understanding the past of our cities. The representation of the virtual heritage provides valuable knowledge about unique locations and buildings that have been destroyed. The emergence of digital tools offers the possibility to fully experience 3D virtual reconstructions. Computer graphics tools have various techniques to reconstruct a 3D model of an ancient heritage. Many systems may be used to create the 3D model but the quality of the work depends on the ability of each system to represent the model according to an uncertainty level. An archeologist's ideas have to be implemented by the system in order to represent certain concepts correctly. This supports the visualization of the model. The archeological data is the main source of data used to create the 3D model of any culture heritage. The cultural heritage will then be used to extract the measurement, and the architecture style of the building. The way that the viewer perceives the information about the 3D model of virtual heritage is the most important step. The limitation of the representation of the reconstruction or even of virtual heritage itself, may cause a failure to understand its contents. The procedural modeling system will be tested in reconstruction of Nymphaum of the Sanctuary of Diana at Nemi, Italy.

1.4 Research Goals

The goal of this work is to reconstruct the Nymphaum of the Sanctuary of Diana at Nemi, Italy according to uncertainty concepts and visualization techniques. As mentioned before, the procedural modeling will use CityEngine to implement the reconstruction of the 3D model.

In this context, the objectives of the thesis research are as follows:

- Testing the ability of CE to create a realistic reconstruction of the archeological data.
- Creating a 3D visualization of the current excavation.
- According to the archeological data, the study will reach four levels of detail.
- Modeling two hypotheses.
- Reconstructing the terrain model from the digital elevation model.
- Publishing the result of the 3D model in the Web Scene.

1.5 Thesis Structure

The thesis' structure revolves around the idea of using Procedural modeling to reconstruct the Nymphaum of the Sanctuary of Diana at Nemi, Italy. This will be used to create four Levels of detail (LoDs) and two hypotheses.

The following section contains a detailed explanation for the content of each chapter:

- **Chapter 1:** Provides an overview of the reconstruction of Nymphaum of the Sanctuary of Diana at Nemi, Italy. In addition it will introduce the problem definitions of the study and the goals of the research.
- **Chapter 2:** Introduce the theoretical framework and state of the art methods of the reconstruction process. This chapter defines terms which are used to create the 3D model. In addition, details of scientific fields that use similar approaches will be introduced.
- **Chapter 3:** Describe the method that is used to reconstruct the 3D model using details which start with the overall structure. Also, the archeological data description is presented in full detail. The comparison of procedural modeling (CE) and CityGML will be introduced.
- **Chapter 4:** Presents the result of the methods that were used. Four LoDs will be presented and two hypotheses.
- **Chapter 5: P**resents the discussion of the work.
- **Chapter 6: P**resents the conclusion and the outlook of the research.

2 Theoretical Framework and State of Art

The evolution of computer graphics and imaging provides a lot of visualization tools to create realistic 3D model for archaeological sites. However, this lead to a tensity between veridical and realistic modeling, particulary in the issue of uncertainty. This chapter discusses the veridicality realism tension in more detail. Also, it describes the concept of uncertainty and how it is visualized in Archaeological reconstructions. It gives an overview about the possible approaches of modeling which are: Computer Aided Design-CAD and procedural modeling. Moreover, it demonstrates the advantages of procedural modeling to generate attractive realism models and express uncertainty. In addtion, it gives an outline of some of the most important grammar types and describes CityEngine and CityGML. And finally, the chapter reviews some previous work used CityEngine, which are the Puuc, Pompeii, Roman rebon and the Louver.

2.1 3D Modeling in Archaeology and Culture Heritage

There are many advantages to visualizing historical sites that have mostly disappeared by using virtual 3D models. It allows experts to remotely investigate features of a site that not normally captured using 2D illustrations. 3D models are more than just a modern illustrations, they are visual dynamic research models that allow for accurate dissemination of archeological study and also efficient presentations for public consumption.

The models are often presented to the public in the form of educational entertainment and are used by architectural firms for marketing purposes. 3D modeling is also used by academics and provides a considerable amount of evidence for the publication of reconstruction hypotheses. The need for esthetic, scientific and technical standards in this field is discussed in the overview of the history of Cultural Virtual Reality (CVR) by Frischer et al.

However, the reconstruction of archaeological sites should always take into consideration the level of detail and photo-realism used in when generating a model. This may cause the general public to have a false impression of the accuracy of the reconstruction because some parts of the modeling work is based on estimates or is only one among several different reconstruction hypotheses of how buildings may have looked..Because of this uncertainty, academics must resolve the conflict between realistic and veridical modeling.

2.2 Uncertainty

Uncertainty has had many definitions over the years but there no universally agreed upon definition (Goodchild et al. 1994) , (Klir andWierman 1999). A comprehensive understanding of uncertainty must include an understanding that any data that may contain various concepts including error, accuracy, validity, quality, noise and confidence and reliability is a multi-faceted characterization of uncertainty. In order to move forward in the quantification and visualization of uncertainty it is important to try and identify any common ground among the various definitions of uncertainty.

In 1999, Klir and Wierman declared that uncertainty itself has several arrangements and dimensions which may contain concepts such as fuzziness or vagueness, disagreement and conflict, imprecision and non-specificity. The Departmental Research Initiative (DRI) asked for proposals to capture uncertainty in a workshop at the turn of the century. They defined the focus and possible approaches of the research with results posted at (Navy 2014). They concluded that Uncertainty was:

- Environmental alterations that can be cognoscible and that we can simulate.
- Environmental alterations that can be cognoscible but that we cannot simulate.
- Environmental alterations that are not cognoscible, and that error is essential in.

In addition to the definition proposed above there was a discussion that indicated that uncertainty is not the same as variability however in most applications uncertainty refers to variability. Currently, many works in geographic information science (GIS) have been assigned concepts associated to uncertainty such like error and inaccuracy. Error is described as the variance between a given value and the correct value (Goodchild et al. 1994) and inaccuracy is the variance between a given value and the modeled or simulated value (Goodchild et al. 1994).

Chiles and Delfiner (1999) wrote a complete text on geostatistical models of spatial uncertainty. The text consist of the quantification of the accuracy of interpolated estimates while also employing Monte Carlo simulations to characterize multiple possible maps. These maps discuss a range of reasonable spatial outcomes stated in some observed data. There are also types of uncertainty which were coined by Goovaerts in 1997. These include local and spatial uncertainty.

It is with this in mind that academics should provide a mechanism which allows observers to visualize the level of uncertainty of an entire model or sections of a model. This, in turn, should be accompanied by readily accessible documentation of the modeling process with special emphasis given to assumptions made by the researchers. This is what the *The London Charter* (LC in short) demands in its principles.

2.2.1 London Charter

A group of researchers drafted the London Charter in 2006 in order to improve computer- based visualization projects. The main motivation of this group was to create guidelines for working with computer-based visualization with an emphasis placed on working with virtual cultural heritage for research purposes (London Charter). Before the establishment of these guidelines researchers were not publishing the process by which the models were being generated nor were they referring viewers to the archaeological source data used to generate the model. For hyperrealism 3D interpretations, the issue of intellectual transparency also became a problem.

Forte, a pioneer in 3D modeling of cultural heritage pointed out these issues: “Noticeable gaps are represented by the fact that the models are not “transparent” in respect to the initial information and by the use of peremptory single reconstruction without offering alternatives” (Forte,2000). The transparency issue was needed to be addressed for all visualization work. The document of London Charter states that:

In order to ensure the intellectual integrity of computer-based visualization methods and outcomes, relevant research sources should be identified and evaluated in a structured and documented way (London Charter).

Nowadays research networks, like Anna Bentkowska-Kafel (3DVISA) and Virtual MuseumsTransnationalNetwork(V-MUST), are facillitating theLondon Charterguidelines by making archaeological data readily available to researchers as well as the public (London Charter). Subsequently, more researchers can be sure about the accuracy of computer-based visualizations for research projects and for cultural heritage (London Charter).

The London Charter also establishes guidelines for presenting a transparent model of a historical monument to the public while also maintaining the scientific integrity of the model. These guidelines are reproduced below:

(4.4) It should be made clear to users what a computer-based visualization seeks to represent, for example the existing state, an evidence-based restoration or a hypothetical reconstruction of cultural heritage object or site, and the extent and nature of any factual uncertainty (London Charter).

(4.6). Documentation of the evaluative, analytical, deductive, interpretative and creative decisions made in the course of computer-based visualizations should be disseminated in such a way that the relationship between research sources, implicit knowledge, explicit reasoning, and visualization- based outcomes can be understood (London Charter).

Thus, the work-flow of the visualization and its documentation must be published alongside the actual model because it provides evidence that the virtual model was created using the guidelines of the London Charter, and more importantly to understand the interpretation stages of archaeological data. In the next section we will discuss the different approaches to visualize the uncertainties within the Archaeological reconstruction.

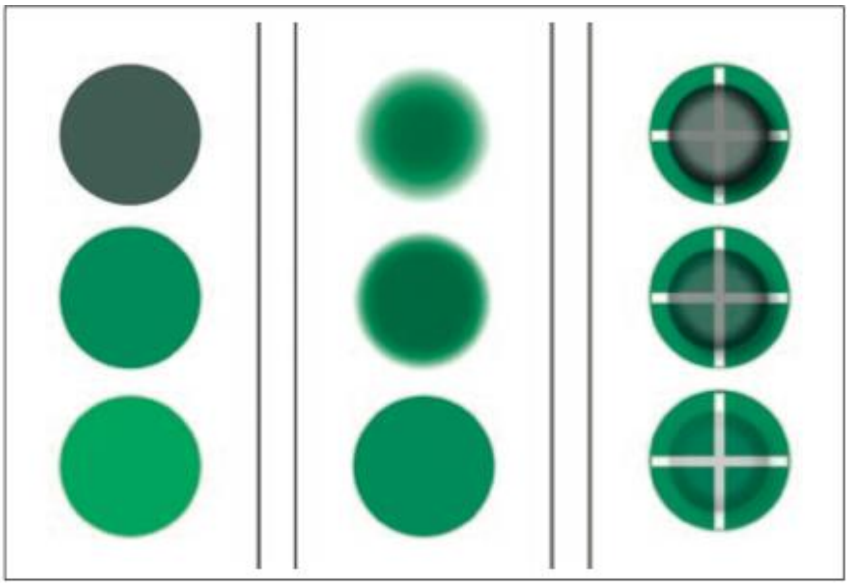
2.2.2 Visualization of Uncertainty of reconstructed antique building structures

The problem of certainty and authenticity within previously created digital visualizations has continually been raised during the last two decades (Bentkowska-Kafel et al. 2012). Knowing of these problems has caused scientists to propose new methods to visualize uncertainty. In spite of new methods of visualization becoming successfully expounded the wide majority of approaches within cultural heritage has continued to produce photorealistic images to their audience.

Culture heritage has to keep up with the impressive photorealism of film, television and gaming. The communicative power of the image has encouraged the public consumption of history in museums and in the media while also allowing for the commercial application of archaeological visualizations. These images are instantly readable and coherent because they camouflage the complexity of source data and expert interpretation. This results in familiar elements of perceived reality and established media that represents reality (Harrison et al. 2012).

The most common approach to redress problems of uncertainty is to use visual cues within a traditionally rendered image to indicate the level of trust of particular elements like manipulating colour or transparency (Pang et al. 1997). Mixing familiar and strange aesthetics will expose the overall coherence of the image. This will cause allow for the production of more reasonable architectural or artefactual models because they demonstrate uncertainty as transparent or in a different color then the rest of the image.

The basic methods for visualizing uncertainty is by following guidelines used in traditional cartography. Where the variables includes location, size, color value, grain, color hue, orientation, and shape. Davis and Keller (1997) confirmed that using color hue, color value, and texture are best for representing uncertain information. Jiang et al. (1995) technique describing hue, lightness, and saturation. Where lightness or darkness is changed to show uncertainty (MacEachren et al. 2005). An extended set of visual variables to represent uncertainty were proposed by MacEachren (1992). These were color saturation, crispness, transparency, and resolution of raster images and of vector line work. In case of using color saturation for example , the elements of the map with pure hues is refer to a high level of certainty, while those with less saturated color are the less certain information, graying out uncertain areas in figure (1) (MacEachren et al. 2005).



.

Figure 1 Point symbol sets depicting uncertainty with variation in (a) saturation, i.e., colors vary from saturated green, bottom, to unsaturated—top; (b) crispness of symbol edge—middle; and (c) transparency of symbol—right. In (c), transparency is applied to th

However, the idea of repeated the methods of visualization uncertainty in literature is the users need to control over the exposition of uncertainty. For that reason, dynamic representations were developed. Howard and MacEachren (1995) described R-VIS for reliability visualization, which is an interactive environment to support exploration of uncertainty. recently, Lucieer and Kraak (2004) implemented a set of linked tools to enable the exploration of uncertainty. Among the first to apply animation to uncertainty representation was Fisher (1993), the result of his approach is that certain parts of the map that indicate certain classification have a stable color, while the uncertain regions change continuously.Complementary uses of animation were proposed by Several other authors. The method of the interaction allowed users to explore the relationships in space and time between uncertainty and particular weather events (MacEachren et al. 2005)

Nevertheless, most research directed to uncertainty visualization has focused on developing representation methods or software applications for the display of uncertainty, or on developing theory about what may work. This allows for a clear delineation of which parts of the model are most surely reconstructed. The questions that still remains however, is how can one expand complex dependencies between uncertain elements as well as display an abundance of possible forms?

In the other hand, another approach is to use non-photorealistic rendering techniques to impact a change in aesthetics commonly anticipated by the public. Replacing photorealistic images with constructed and non-photorealistic images enables a subjective interpretation of the rendering that communicates uncertainty to the audience. Although the use of such an approach in cultural heritage is difficult because photorealism is too embedded and anticipated by the heritage industry and the public.

While using photorealistic models may have its limitations the use of overly simplistic models to reduce false impressions of uncertainty carries its own inherent risks. There are two types of simplification: copy-and-paste strategies and omissions. It is common to produce oversimplified models when one is not sure about the exact form of certain elements in the model. A pillar, for instance, may simply be represented by a similarly shaped which is then copy and pasted throughout the model. This type of repetition results in an unrealistic shape or pattern that occurs with exact regularity within the model (Haegler et al., 2009).

Omission of objects is simply the result of neglecting to explain gaps in the model in order to avoid any uncertainty. This results in inadequately capturing the true level of a structure's decoration or explaining the intentions of its creators. For example, if one color in a Greek building is omitted it will create an extended misconception about the building's original appearance. The modelers would thus rather omit a coloration rather than make a coloration mistake but this in turn oversimplifies the grandeur of the building and creates a form of misrepresentation that some consider worse than making a coloration mistake (Haegler et al., 2009).

These oversimplified models do not alienates non-academics especially children who are accustomed to impressive graphical constructions in movies and games. The alternative to generating a single model with indications of the model's uncertainty is to generate several realistic models. This is a more efficient method of modeling compared to traditional techniques. Rather than using coloration, non-photorealistic rendering (NPR) or levels of transparency to point out that one is not quite sure about a component of a model, one can create several realistic models (Haegler et al., 2009).

Each of these models can be used for public consumption while also displayingthe probability distributions for the uncertainties in each rendering. It is possible to represent the sum of these models by sampling the original and generating a large number of samples that each contain an expert's annotations and clear indications of uncertainty.The advantage of digital visualization over traditional visual aids has always been the capability to manipulate and adjust the model but this still requires a considerable amount of effort. For that reason it is important to balance the number of adjustments so that a reasonable amount of labour is done to communicate the uncertainty (Haegler et al., 2009).

In presentation terms, someone has then to decide which possible elements have to be included in the final images and in which particular combinations. Moreover, each image should be coherent by itself. As such, one will be able to point out and a section of an image while showing another as the more probable scenario. For an example of this type of reproduction see Millett and James's substitute illustrations of timber framed buildings at Cowdery's down (Millett and James 1983; James 1997, 29).

The presentation restrictions of several images are addressed by interactive models which use parametric relationships. User input can be generated automatically in similar system options such as VRML and HTML/java (Roberts and Ryan 1997) and in bespoke virtual environments such as CREATE (Roussou and Drettakis 2003). The user in these systems can examine the uncertainty within the parameter space of a model as outlined by the creator, in order to obtain a perception of the domain of possible parts without being inundated by several parallel images.

The sequence representation of the models allows the public to have a good impression of how it might looked, and at the same time clearly indicates uncertainties for those interested in such knowledge. It is possible to create this representation for multiple models for entire archaeological sites by using the procedural modeling strategy.

Archaeological reconstructions should utilize procedural modelling technologies rather than the previously discussed approaches because it is designed for architectural visualization which was named Shape Grammar, where CGA stand for Computer Generated Architure (Müller et al. 2006) and integrated into the CityEngine modelling package (Procedural, Esri). This technology provides a modeling system that can contain parameterization, and stochastic or probabilistic expression of these parameters. Thus, this system allows for both dynamic adaptations of geometry and quantified conceptualizations of the relative certainty of its elements.

Besides handling uncertainty it clarifies the rigid nature of static geometrical site models without repetition. This reduces the need for intensive manual labor and thus makes the generation of multiple models more feasible for researchers. Moreover, it allows site models to be updated as existing bodies of knowledge about certain sites continue to grow and the remaining uncertainty to be effectively expressed (Haegler et al., 2009).

In the next section we will show the principal differences between manual (traditional) and procedural 3D modeling and after that we will explain in detail what is procedural modeling. When we discusse the possible approaches of modeling.

2.3 Reading an Architect's Mind

Traditional modeling requires a great deal of works at the model's geometrical and visual level. Modifications are done on vertices, surfaces, lines, materials and colors part-by-part or building-by building until the model perfectly matches information sources. The information sources may include: blueprints, drawings or scans of a site provided by archeologists and surveyors. Creating geometry from the template data utilizes some techniques such as image based modeling, sculpting and conventional mesh modeling. This allows the artist to generate the visual appearance of the site step by step, a process much more different from procedural modeling (Haegler et al., 2009).

Something to keep in mind is that while the display or visual effects of a model may be similar to the structure's original appearance it is not necessarily true that the structure logic of the model matches the architectural design of the site. This is especially true due to the fact that there are multiple situations over the course of modeling where obvious or hidden uncertainties are ignored. Procedural modeling takes a parallel approach that uses structural and semantic data, including uncertainty, instead of only utilizing the geometrical aspects of a structure (Haegler et al., 2009).

2.4 From the Geometrical to the Semantic level

The spatial and semantic relationships among architectural elements are more important than appearance attributes or geometrical coordinates when describing sites using procedural modeling. Spatial semantic relationships between the objects of a site are called shapes in grammar based procedural modeling, while architectural relationships are called rule sets. If these rule sets are accurately captured into procedural descriptions then the model will display valid visual attributes and spatial coordinates within a reliable boundary of uncertainty. In other words, the procedural description is a translation of an architect's mind into a computer readable format (Haegler et al., 2009).

There are a number of advantages attributed to utilizing semantic and structural relationships: first, with text-based procedural models it is possible to display uncertainties and annotations while also allowing for changes to be too tracked during the modeling process (cf. LC, principle 4). Secondly, the models are more cohesive and the risk of errors is decreased. Third, the storage size for the model is dramatically reduced while also allowing for the sharing of repeated elements. Fourth, one can still read the structure of a model and its uncertainties even if visualization tools are not available (cf. LC principle 5). And lastly, architectural rules can be easily interpreted into modeling rules if they were present in the literature (Haegler et al., 2009).

2.5 The Source of Archeological Reconstruction Data

In order to generate a model a lot of information from geospatial data collected from a specific site along with non-structured textual descriptions, often given by different sources, that then need to be put into a structured format is required. While these sources cannot be used for direct visualization it is still important generate informational context for the model. Therefore, a lot of analysis and hard work is needed in order to build the model. Procedural modeling provides a method by which this process can be conducted more efficiently through automation (Haegler et al., 2009).

Traditional and procedural modeling of archeological reconstructions utilize the same sources of data which may include: text-based literature, archeological drawings and sketches, old photographs, vector data (geodesic measurements), raster data (digital elevation models) and surveys done by both laser scanning, photogrammetry or computer visual methods.

2.6 Possible Modeling Approaches for Reconstruction 3D Building

2.6.1 Computer Aided Design-CAD

The application of information technology (IT) in the design process is the definition of Computer Aided Design-CAD. There are three compensations for the CAD system: hardware (H/W), software (S/W) and peripherals which are quite specialized for some applications. The CAD system core is the S/W which utilizes graphics for product representation and operates the peripherals for product presentation. The use of CAD does not change the design process but rather aids the product designer from the initial problem identification to the implementation phase.

CAD aids the user by providing:

1. Graphical representations are accurately generated and easily modified
2. Design analysis is done in short time and provides the following tools:

   Analysis methods for finite elements such as, static, dynamic and natural frequency analysis, heat transfer analysis, plastic analysis, fluid flow analysis, motion analysis, tolerance analysis, and design optimization

3. The user can record and recall information with consistency and speed by using Product Data Management (PDM) systems (which allows the user to store a design for future reuse and upgrades)

The first CAD system in use was Sketchpad which was created within the SAGE (Semi- Automatic Ground Environment) research project by Ian Sutherland at MIT. CAD technology was first used by the automotive and aerospace industries. The system was quite costly and required advanced computing hardware alongside customized software components provided by the software supplier. Moreover, the computer systems were mainframes in the first CAD systems while presently the technology is over networks or as standalone workstations such as UNIX or WINDOWS based systems.

The first supplier that offered a PC based CAD system was AUTODESK. Their software AUTOCAD® was first introduced in 1980. Nowadays the main operating system for CAD systems is Microsoft's Windows ® (Bilalis.2000). The first CAD systems were only supported the creation of 2D drafts and modeling. The 2D drafting is still the focus of application. In the mid-1980s, 3D modeling technology became very popular among as IT H/W became more accessible. 3D modeling was primarily a wireframe based in the beginning. The surface modeling systems was used in Aerospace and automotive industries to model the outward body of a product.

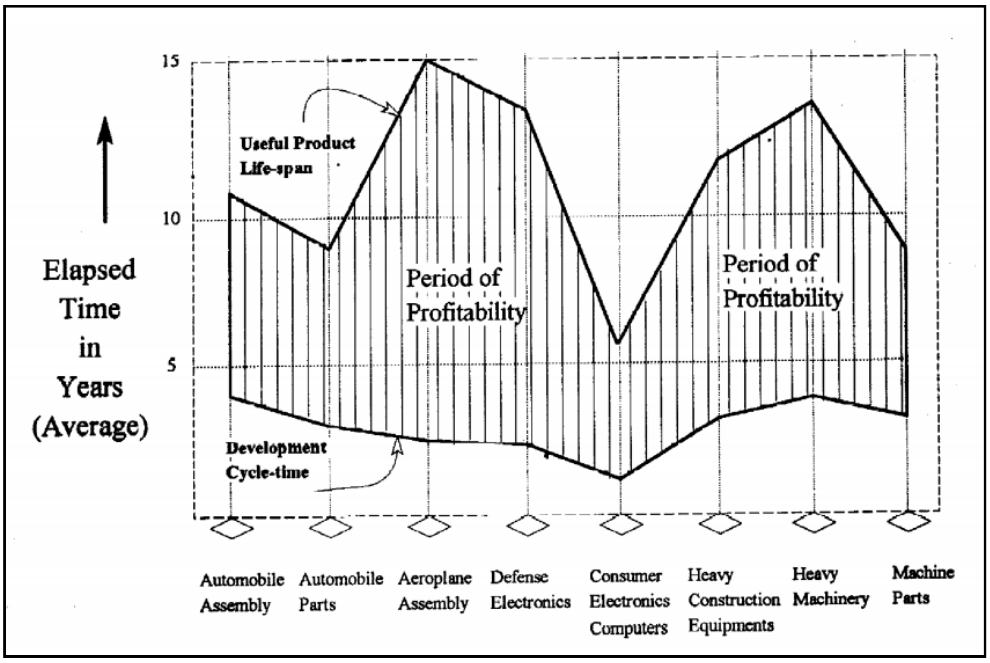


Figure 2 Analysis of the average product development time (indicates the start of the pay-off period) and of the product useful time (period of profitability) for various industrial products.

In addition the solid modeling was realized as the only system that provided understandable representation of the product. However, the main issue with the model was that they lacked adequate support for complex representations. Solid and surface modelling technologies have merged in today's CAD systems because solid modeling is effective for the design of industrial products that have complex mechanical applications. CAD systems have since been adopted by a number of industrial sectors such as, electronics, textiles, and packaging. The CAD system allows reduced the amount of typically used during the design process and allows the introduction of products to reach market much more quickly than ever before. 3D model representations can also be exported to different platforms which can then act as a communication platform between many people from several different departments of an organization.

A digital prototype of a product generated using a CAD system allows for testing and evaluation from various departments and gives them the ability to express their opinion for the product at earlier stages of development. This reduces the number of times that a product must be produced and then refined to finalization over time. In fig.2, there is a representation of the product development time and of the product useful life span. It also allows the researchers to spend more time on the early stages of design rather than on redesigning a product as shown in the figure 3 (Bilalis 2000).

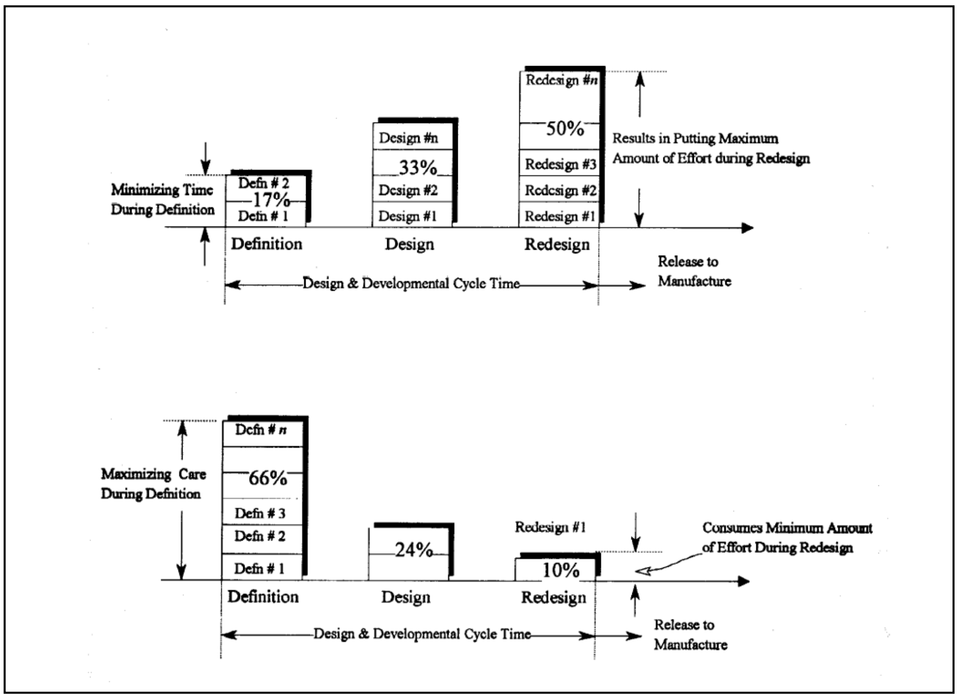

Minimizing Time During Definition
Defn # 2
17%
Defn # 1
Design #n
33%
Design #2
Design #1
Redesign #n
50%
Redesign #3
Redesign #2
Redesign #1
Results in Putting Maximum Amount of Effort during Redesign
Definition
Design
Redesign
Release to Manufacture
Design & Developmental Cycle Time
Maximizing Care During Definition
Defn # n
66%
Defn # 3
Defn # 2
Defn # 1
24%
Redesign #1
10%
Consumes Minimum Amount of Effort During Redesign
Definition
Design
Redesign
Release to Manufacture
Design & Developmental Cycle Time

Figure 3: Distribution of product development time. The earlier a new product definition is Introduced the least redesign is required for the final design.

2.6.2 Procedural Modeling

Procedural Modeling is an approach that uses several techniques to create virtual models by using a set of rules. Generating 3D models of a building or an entire city by using this method has been proposed by many researchers. However, building models typically only consists of outer facades rather than building interiors.

Architectural procedural modeling can draw on a number of production systems such as: Semi-Thue processes (Davis et al. 1994), Chomsky grammars (Sipser 1996), graph grammars (Ehrig et al. 1999), shape grammars (Stiny 1975), attributed grammars (Knuth 1968), L-systems (Prusinkiewicz et al. 1991), or set grammars (Wonka et al. 2003). Some of these system use hard coded modeling rules that have been written in C++ while others use a more constrained grammar-based system. These methods are popular because of how much they allow the modeler to express their desires in the model while also maintaining a high efficiency.

Shape grammars have successfully been used for the construction analysis of architectural designs even though classical shape grammars lack a control mechanism that enables rule selection or transformations in each production step. This limits them to manual use however, the "cityEngine" software package has made the shape grammar concept more useable for architectural modeling. The shape grammar can be complemented by Generative Modeling Language (GML) technology which as introduced by Berndt et al. GMLdescribes the archeological or the architectural asset using a group of low level operations that control entire components.

Moreover, GML and shape grammars both share the potential to change shapes out of the simple translation of parameter settings. Below is an outline of some of the most important grammar types.

2.6.2.1 Shape Grammar

In order to make a successful 3D movies and computer games it is critical to create convincing models. However, modeling cities or any large three-dimensional area requires a tremendous amount of time and money. Yet, through the use of procedural modeling, using shape-Grammars that are capable of creating large areas efficiently can be done at a low cost without sacrificing quality in terms of number of polygons or geometric details. The time needed in this process would is greatly reduced compared to traditional modeling software.

Using shape-Grammars for modeling urban environments was first recognized by Parish and Müller [2001] and Wonka et al. [2003]. Parish and Müller showed that large urban environments, where buildings were made up of simple mass models, could be generated. Wonka et al. [2003] explained how to generate geometrical details on facades for each building individual building. The combination of these two concepts is what allowed the efficient creation of large and detailed urban environments. However, both models pose a number of considerable challenges during the modeling process that need to be addressed.

The Parish and Müller strategy allows for the generation of simple models by adding translated and rotated boxes and then using a shader to add details. Thus, it cannot produce a sufficient quantity of geometrical details. This results in numerous unwanted intersections of architectural elements. On the other hand, the strategy proposed by Wonka et al. (2003), is using split rules that are only sufficient for simple mass models. It is not easy to change these mass models because novel configurations will need additional production rules where it is not easy to handle objects of arbitrary orientation.

For that reason, grammar-based solutions can be proposed in order to generate buildings with detailed shells stemming from complex mass models. Based on this new model can be generated and this approach which allows for context sensitive shape rules is suitable for computer graphics architecture.

Shape Grammar as an Analytical and Design tool

Stiny (1975; 1980) successfully used shape grammars for the construction and analysis of architectural design (Stiny and Mitchell1978; Duarte 2002). The original formulation of the shape grammar operates directly on an order of labeled points and lines. However, the derivation is essentially complex and requires manual work in order to decide which rules to apply. Shape grammar can be made amenable to computer implementation in a process of simplification that converts them into set grammars (Stiny 1982; Wonka et al. 2003). Brick patterns can be computed using cellular textures and it is also possible to generate complex manifold surfaces from simpler ones using generative mesh modeling (Legakis et al. 2001; Havemann 2005). While one important part of the procedural modeling framework may be defined by shape-grammars, it is also important to recapitulate these rules that create architectural shapes. There are some books that emphasize the structure of architecture, such as “The Logic of Architecture” by Mitchell (1990), “SpaceSyntax” (Hillier 1996), Design Patterns (Alexander et al. 1977), a visual dictionary (Ching 1996) and studies of symmetry (March and Steadman 1974); (Shubnikov and Koptsik 1974); (Weyl 1952).

2.6.2.2 Split Grammar:

Split grammars are a particular type of set grammar that operates on shapes. It is convenient to use for the automatic creation of buildings. The objects influenced by the grammar are obvious, attributed, parameterized, labeled shapes that are named basic shapes (Wonka etal. 2003). These basic shapes are made up of cuboids, cylinders and prisms. The basic shapes are indicated in terms of explicit geometric objects, thus, the volume and boundary figures 4 and 5. The vocabulary of a split grammar is

setB={f(b)|bisabasicshape,f∈ F}(notethatf(b)=f(s),f(P),V).

The set of acceptable conversions F will be the affine conversions. A split is the disintegration of a basic shape into shapes from the vocabulary B. For example, a split changes a cuboid with n×m×k cuboids which is organized in a grid with parameterized splitting planes. A split grammar is a set grammar over the vocabulary B which is defined by the basic shapes (Wonka et al. 2003). The disadvantage of the split grammar is that they are mostly fitting for rectangular façade designs structures. The split grammar must be extended with special operations to be able to handle non-rectangular building footprints. In addition, spilt grammars are restricted to a hierarchical subdivision of design. Additional operations are often required for the generation of designs particularly for the creation of building shapes (Mueller et al. 2005).

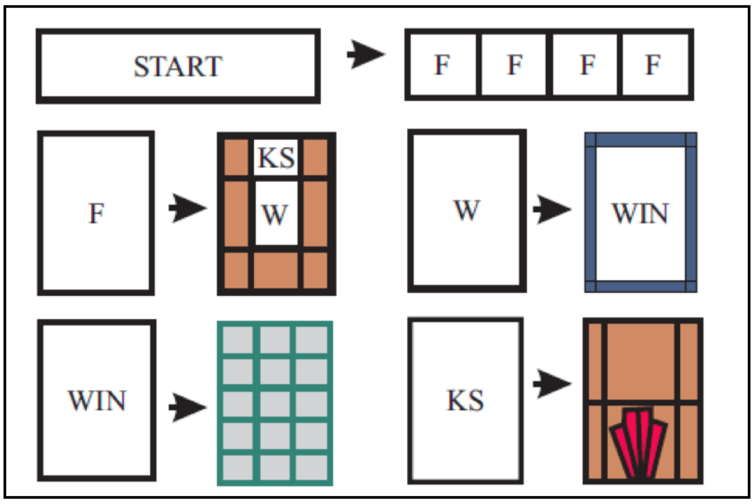


Figure 4: The rules for a simple example split grammar.

The white areas (which contain symbols) represent the non-terminal shapes while the colored elements are the terminal shapes of the split grammar. The start symbol is split into 4 façade elements, which are further split into a window element, a keystone element and some wall elements (Wonka et al. 2003).

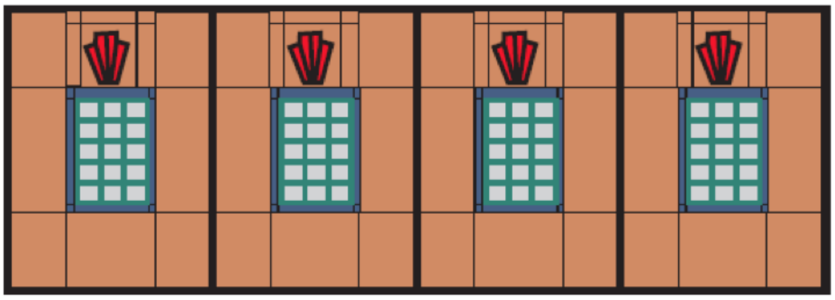


Figure 5 : This figure shows the result of the derivation of the grammar (Wonka et al. 2003).

2.6.2.3 Lindenmayer System

The of modeling of Lindenmayer systems (L-systems) was first introduced in 1968 by Lindenmayer as a theoretical framework for plants but the system did not include enough details to model higher plants. The development of geometric features using L-systems had enough details which allowed computer graphics to be used for actual visualization of plant developmental processes and structures (Lindenmayer 1968). The focused of the L-system was on the topology of plants and connections between cells and large plants modules. Many L- system proposals of geometric interpretations were published which allowed them to be turned into adjustable tools for plant modeling (Prusinkiewicz et al. 1990).

L-systems define complex objects by changing the parts of a simple, major object through the use of a set of rewriting rules or productions. The snowflake curve is an example of a graphical object which is defined in terms of rewriting rules (Figure 6), originally proposed in 1905 by von Koch [155].

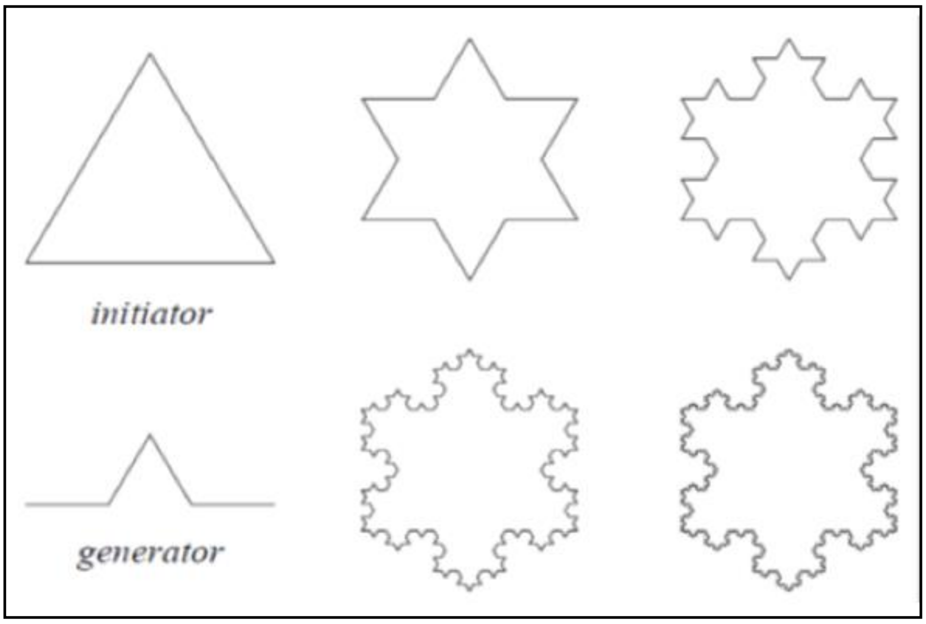


Figure 6: Construction of the snowflake curve

2.6.2.4 CGA Shape Grammar

CGA shape is a new shape grammar use for the procedural modeling of computer graphics architecture (CGA). It is used to create buildings structures with high visual quality and high geometric details. CGA shape works with production rules that repeatedly improve a design. The process follows procedural steps to create buildings efficiently. First the production rules create a mass model which is a simple volumetric model of a building. Then, it constructs the facade and after that it adds final details like doors, windowsand ornaments. The creation of the hierarchic structure and the use of model annotations are specified in the modeling process. This semantic data enables the modeler to reuse the design rules for procedural variations, and to create a large variety of architectures to complete a whole city.

CGA shape is an extension of set grammars that were introduced by (Wonka et al. 2003). The following section will outline the definition of a split, repeat spilt, component split and the modeling basics for one, two and three-dimensional shapes. The notation of the grammar and general rules are used in order to add, translate, scale, and rotate shapes. This was inspired by L- systems (Prusinkiewicz and Lindenmayer 1991) but have been expanded for the modeling of architecture. Although parallel grammars like L-systems are suitable for picking up expansions over time, the applications that have sequential rules enable the modeler to characterize a structure by spatial distributing features and components (Prusinkiewicz et al. 2001). Accordingly, CGA Shape is a sequential grammar (similar to Chomsky grammars).

Shape: The configuration of shapes within the grammar consist of a symbol (string), geometric attributes and numeric attributes. The symbols are used to identify the shapes which can be either a terminal symbol ∊ Σ for terminal shapes, or a non-terminal symbol ∊ V for non-terminal shapes. The geometric attributes define an oriented bounding box in space called a scope. The most important geometric attributes are the position P, the vectors X, Y and Z for describing a coordinate system, and the vector S for size.

Production process: A finite set of basic shapes creates a configuration. With an arbitrary configuration of shapes A, called the axiom, the production process can start. It can then proceed as follows:

First, an active shape with symbol B in the set is selected, then, successor for B is computed using a production rule with B on the left hand side thus creating a new set of shapes BNEW. After that, the shape B is marked as inactive and shapes BNEW are added to the configuration. The first step is then repeated and when the configuration contains no more non-terminals, the production process finishes. The explorations of the derivation tree either depth-first or breadth-first is dependent on the selection algorithm in the first step (Sipser 1996). However, control over the derivation is not enough thus, priority for all rules is specified according to their represented detail based on each shape in order to get a modified breadth-first derivation. In other words, the shape with the highest priority rule is selected in the first step. This strategy ensures that the derivation proceeds in a controlled manner from low detail to high detail. During the process the shapes are not deleted rather they are marked as inactive after replacing them. This allows the modeler query the shape hierarchy beside the active configuration.

Notation: the following form defines the production rules:

id: predecessor: cond Y successor : prob

Where id is a unique identifier for the rule, predecessor ∊ V is a symbol that identifies a shape that will be replaced with successor, and cond is a guard (logical expression) that has to evaluate to true so as to apply the rule. The rule is selected with probability prob. For example:

1: fac(h) : h > 9 Y floor(h/3) floor(h/3) floor(h/3)

Here, the shape f ac is replaced by the rule with three shapes floor, if the parameter h is greater than 9.

Scope rules: We use general rules for shape modification : T (tx , ty , tz) is a translation vector which is added to the scope position P, Rx(angle), Ry(angle) and Rz(angle) rotate the respective axis of the coordinate system, and S(sx , sy , sz) sets the size of the scope. To push and pop the current scope on a stack we use [and]., for any non-terminal symbol ∊ V in the rule it can be created with the current scope. The command I(ob jId) adds a case of a geometric primitive with identifier objId. It is possible to use typical objects like a cube, a quad, and a cylinder, or any.

The example below illustrates the design of the mass model depicted in Figure 7:

1: A Y [T (0,0,6) S(8,10,18) I(”cube”)] T (6,0,0) S(7,13,18) I(”cube”) T (0,0,16) S(8,15,8) I(”cylinder”)

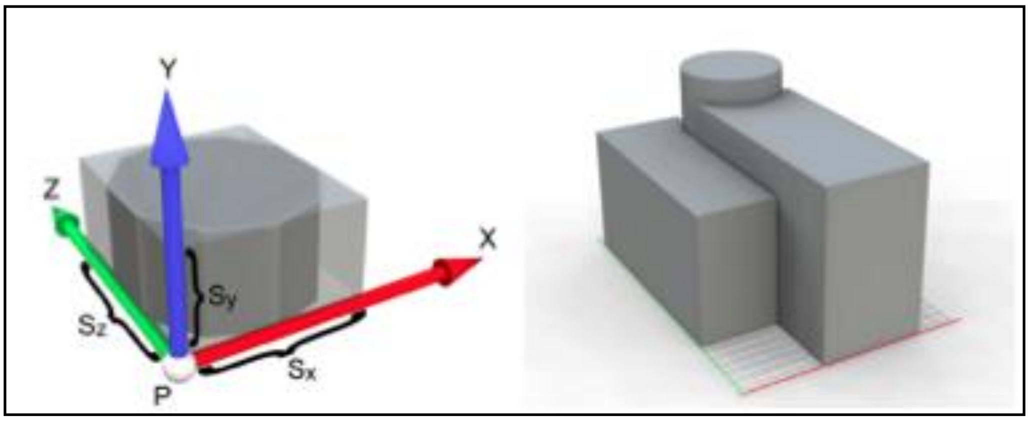


Figure 7: The scope of a shape.

Left: The scope of a shape. The point P, together with the three axes X, Y, and Z and a size S define a box in space that contains the shape. Right: A simple building mass model composed of three shape primitives.

Basic split rule: The current scope is along one axis by the basic split rule. For example, consider the rule to split the facade of figure 8 left into four floors and one ledge:

1: fac Y Subdiv(”Y”,3.5,0.3,3,3,3){ floor | ledge | floor | floor | floor }

The split axis (“X”,”Y”, or “Z”) is described by the first parameter and the split sizes are described by the remaining parameters. Between the de-limiter { and } a list of shapes is given, separated by |. Also, similar split rules can be used to split along multiple axis (“XY”, “XZ”, “YZ”, or “XYZ”), nested splits, or nested combinations of splits and L-system rules.

Scaling of rules: There is a challenge in the previous example because the split is dimensioned to fit with a scope of size y = 12.8 but the rule has to be scaled for other scopes. The architectural parts do not all scale equally and it is essential to have the possibility to differentiate between absolute values and relative values. Values are counted absolute by default and the letter r is used to denote relative values as such:

1: floor Y Subdiv(”X”,2,1r,1r,2){ B | A | A | B }

Where relative values ri are substituted as ri ∗ (Scope.sx − ∑absi)/∑ri , and Scope.sx represents the size of the x-length of the current scope Figure 8 right illustrates the application of the rule above on two different sized floors (with x-length 12 and 10).

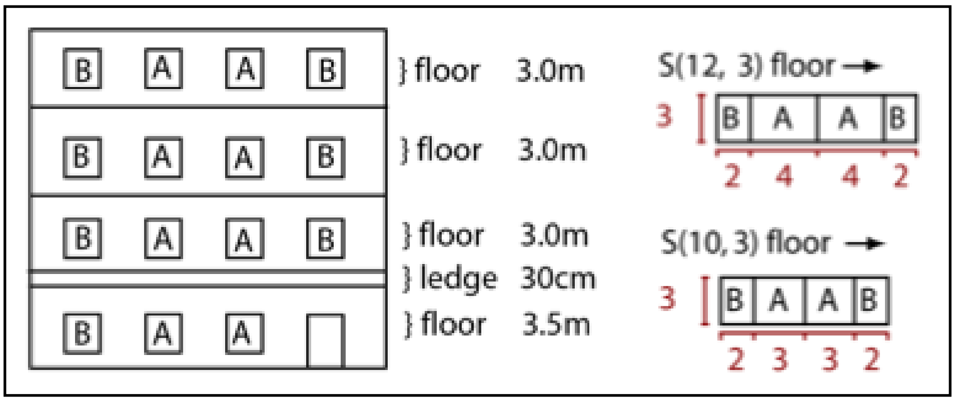


Figure 8: Left: A basic facade design. Right: A simple split that could be used for the top three floors

Repeat: To be able to change larger scales in the split rules a specified element is tiled. For example:

```
1: floor Y Repeat(”X”,2){ B }
```

There is a space along the x-axis of the scope and the floor will be tiled into as many elements of type B as possible. The actual size of the element can then be adjusted accordingly so that the number of repetitions is computed as repetitions = [Sco pe.sx/2].

Component split: All shapes (scopes) have been three-dimensional up until this point. The following command allows the modeler to split objects into shapes of lesser dimensions:

```
1: aYComp(type,param){A|B|...|Z}
```

Where the type of the component split is identified by type with associated parameters param (if any). For example to create a shape with symbol A we write Comp (”faces”){A} for each face of the original three-dimensional shape. Comp (”edges”){B} and Comp(”vertices”){C} to can be used to split into edges and vertices respectively. Commands such as Comp(”edge”,3){A} to can be used to access selected components and to create a shape A aligned with the third edge of the model. Comp(”side f aces”){B} can be used to access the side faces of a cube or polygonal cylinder. Scopes are used when one or multiple axes have a zero size to encode shapes of lesser dimension. Also, the size command S can be used with a non-zero value in the corresponding dimension to go back to higher dimensions for example: to extrude a face shape along its normal and therefore transforming it into a volumetric shape.

2.7 Description of CityEngine:

Parish and Müller presented CityEngine in 2001 which is able to create a complete city by utilizing a relatively small set of statistical and geographical input data that is controlled by the user. According to Parish and Müller, there is no system available that has the same approach. The Generation of Three-Dimensional Geometry for Night Illumination and Urban Visualization project was published by Chen and is a method that is used to generate urban models. It is only possible by the refinement of existing geometry. This approach has to be created manually. CityEngine is able to create the urban environment from scratch by using a hierarchical set of intelligible rules that can be adjusted based on the user’s needs.

The creation of a city in CityEngine can be used to generate a traffic network and surrounding buildings in order to decrease congestion. The streets in cities follow some order of pattern on different scales. Streets are also distributed amongst an urban population where roads are the primary medium of transportation. In 1996, R. Mech and P. Prusinkiewicz reviewed similar applications that used L-systems to support branching and that held the advantage of database amplification. L-systems have since been adopted in CityEngine in order to enable the creation of large cities, based on data collected in four cities around the world, York, Paris, Tokyo and London (Parish et al.2001). Even though the underlying model of the virtual city has been clarified in CityEngine, the system's main goal is to be able to easily modify architectural design. In order to, allow the user to add new subsystems such as, different kinds of transportation networks and land uses, CityEngine has extended the application of L-system mechanisms to function on a higher level which makes the addition of new rules much easier (Parish et al.2001).

The figure 9 identifies several tools found in CityEngine:

- By using the extended L-system input data is integrated to the road generation system
- To define the building distribution the area between roads are divided
- Buildings are generated as a chain of simple solid shapes by using another L-system
- A parser expounds all the results for the visualization software. The visualization software is able to process all texture maps and polygonal geometry. Any 3D modeling has the same process and all rendering software supports procedural texture allowing the integration of any proposed mechanism that generates facades into the pipeline.

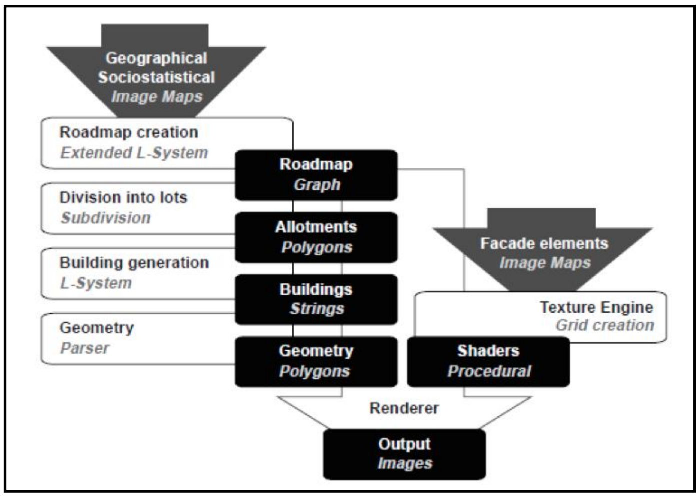


Figure 9: The pipeline of the city creation tool. The dark boxes list the results and data structures of the individual tools in the white rectangles (Parish et al.2001).

Almost all of the input data that is used to build a virtual city is represented by 2D image maps which allow the management of of the system. By using those images the data can be classified into two groups (Parish et al.2001):

- Geographical Maps
    1. Elevation maps
    2. Land/water/vegetation maps
- Sociostatistical maps o Population density
    1. Zone maps (residential, commercial or mixed zones)
    2. Street patterns (control behavior of streets)

    Height maps (maximal house height)

Procedural Modeling Core

CityEngine uses the procedural modeling technique to generate models efficiently. The computer database represents a set of geometric modeling commands that are carried out automatically as using rule file. The commands combine the CGA file in CityEngine and are well known in most 3D applications like extrude, split, or texture. They can also be easily adjusted by the user to create a complex architectural model in a short time. Dynamic City Layouts is a powerful tool that allows the user to create interactive street networks which can be automatically updated in real-time. The user's input can adapt an urban object like streets, sidewalks and whole blocks in order to build the layout of complete cities. When an object is adapted the geometry of all those objects will also be updated.

Customizable UserInterface

The CityEngine user interface is flexible for many tasks and works whether this rule task is used on street networks to edit attributes or simply to study a statistical report. Python can also be used in CityEngine and provides control over repetitive tasks allowing the user to automate specific actions.

Data Interoperability

CityEngine supports many kind of formats for import and export while also allowing the user to access any type of geometry like line or shape data (footprints).

Professionals in the field of Urban planning, Architecture, Entertainment or Simulation are able to transport there work to many formats shown in figure 10.

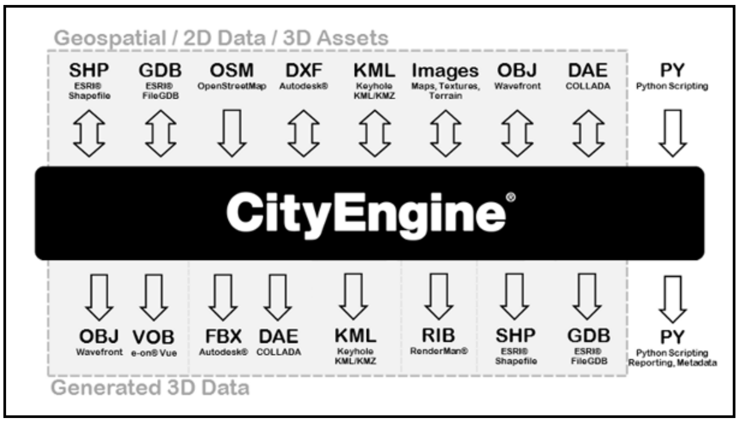


Figure 10: shows the import and export format.

The CityEngine Modeling Pipeline

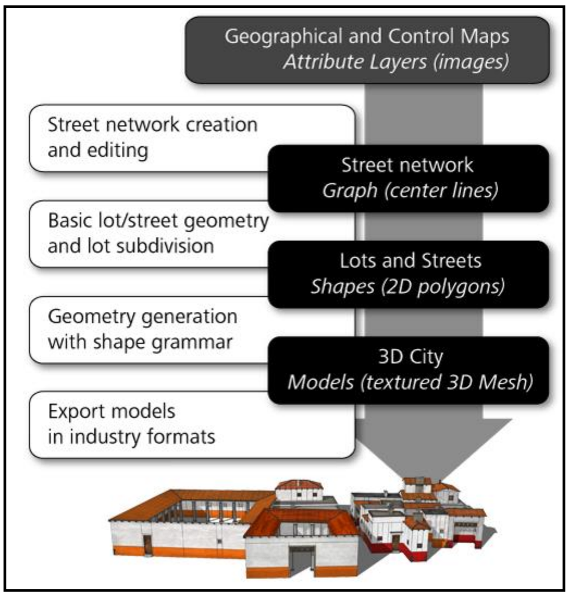

Geographical and Control Maps
Attribute Layers (images)
Street network creation and editing
Street network
Graph (center lines)
Basic lot/street geometry and lot subdivision
Lots and Streets
Shapes (2D polygons)
Geometry generation with shape grammar
3D City
Models (textured 3D Mesh)
Export models in industry formats

Figure 11: shows the individual stages of the pipeline.

When the user models an urban environment using CityEngine the individual stages of the pipeline are shown in Figure 11. The pipeline provides various procedural modeling tools for large scale urban layouts and applies CGA rules to the creation of detailed building models. CityEngine stores scenes as layers which present different stages of the model. Street blocks or building mass geometry are saved in the Obj format and can be imported at different stages of model generation due to the flexibility of the pipeline. Overview of the CityEngine modeling pipeline. Black boxes illustrate data types (layers) and white boxes are the operations to create them. Typically, in the first step, the street network is created, afterwards the resulting blocks are subdivided into lots. Finally, the 3D models of the buildings are generated using the CGA rules. The output of CityEngine is polygonal building models.

Grammar-based Modeling

The used of procedural modeling applications is common mostly when the user needs to build a large number of objects which follow a set of standardized rules. The main goal of the procedural modeling technique is to automate the generation of a model. The quality of the grammar based description is reflected in the number of objects of the generated models quality. This means that unique landmark buildings should not be created using the procedural approach. The initial phases of procedural modeling require a considerable amount of manual labor because rule sets need to be written manually but as the work progress the amount of time and effort put into writing rules reduces dramatically. This is especially true when contrasted against manual modeling which requires that each individual building be generated by hand. Figure 12 below describes both techniques and shows how grammar based modeling is useful.

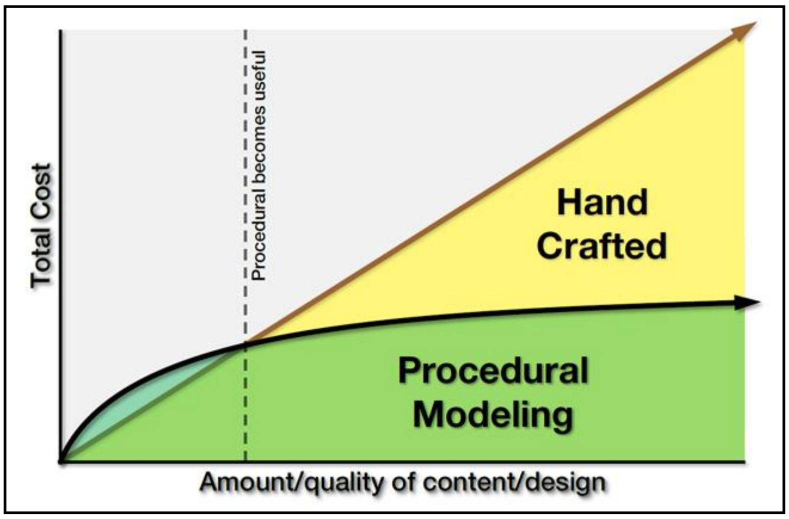
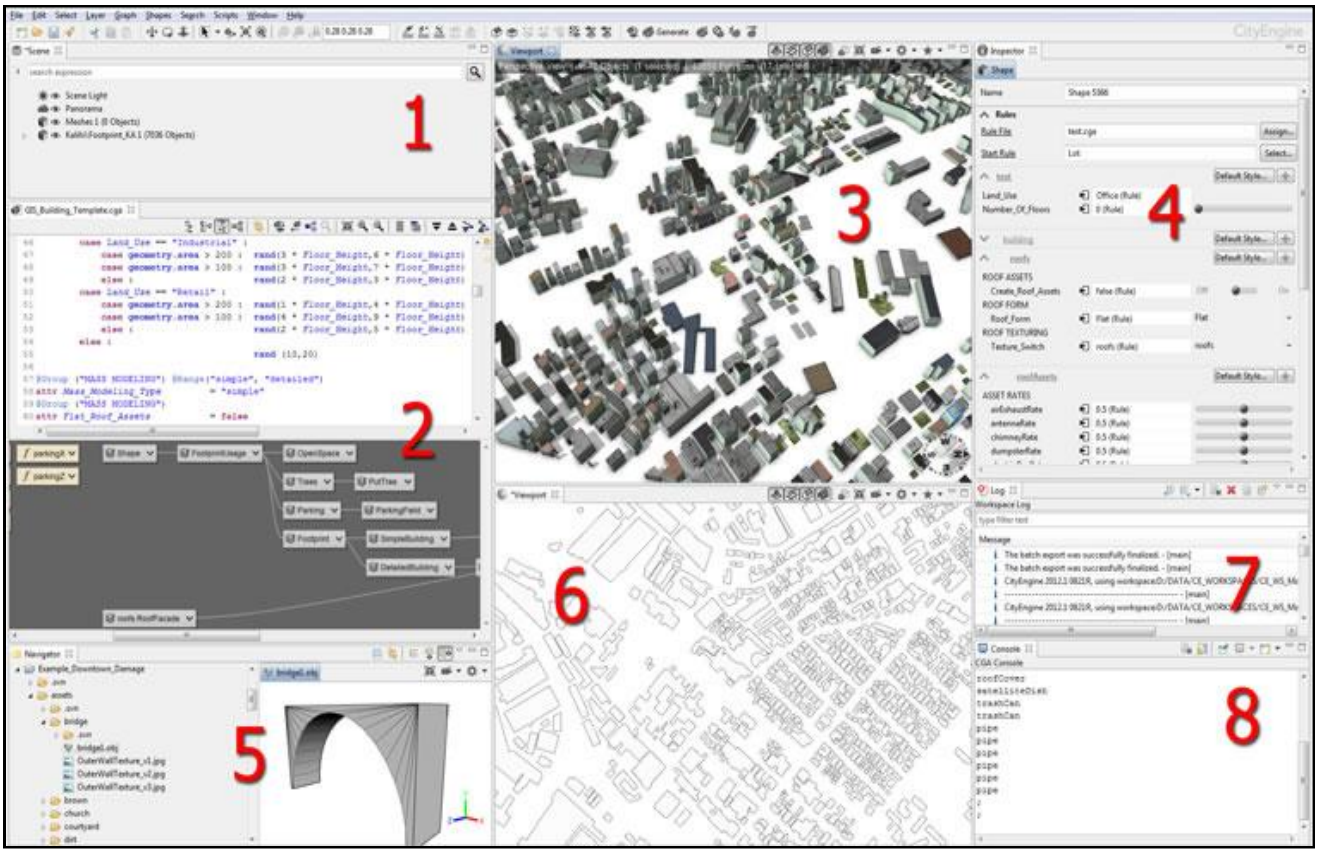


Figure 12: shows the curve of procedural modeling

CityEngine Main Window

The Cityengine user interface's main window contains several sub window parts. The figure 13 is a screenshot of a typical work in progress Cityengine modeling session.



```
        case Land_Use == "Industrial" :
            case geometry.area > 200 :  rand(3 * Floor_Height,6 * Floor_Height)
            case geometry.area > 100 :  rand(3 * Floor_Height,7 * Floor_Height)
            else :                      rand(2 * Floor_Height,5 * Floor_Height)
        case Land_Use == "Retail" :
            case geometry.area > 200 :  rand(1 * Floor_Height,4 * Floor_Height)
            case geometry.area > 100 :  rand(4 * Floor_Height,9 * Floor_Height)
            else :                      rand(2 * Floor_Height,5 * Floor_Height)
        else :
                                        rand (10,20)

@Group ("MASS MODELING") @Range("simple", "detailed")
attr Mass_Modeling_Type         = "simple"
@Group ("MASS MODELING")
attr Flat_Roof_Assets           = false
```

```
The batch export was successfully finalized. - [main]
The batch export was successfully finalized. - [main]
CityEngine 2012.1 0821R, using workspace:D:/DATA/CE_WORKSPACES/CE_WS_Mc
------------------------------------------------ - [main]
CityEngine 2012.1 0821R, using workspace:D:/DATA/CE_WORKSPACES/CE_WS_Mc
------------------------------------------------ - [main]
```

```
roofCover
satelliteDish
trashCan
trashCan
pipe
pipe
pipe
pipe
pipe
pipe
2
2
```

Figure 13: shows CE interface window.

.

The Navigator is the workspace used for project and scene management. The CityEngine project Navigator has the following folders:

- Assets: The assets are used by the shape grammar to control the 3D model and the kind of data that can be used in assets such as the Obj format and the texture of any format
- Data: The data folder holds additional data such as lots or mass models which can be imported into the scene and have certain rules applied
- Images: The images folder consists of additional imagery such as snapshots
- Maps: This folder holds image maps used in map layers such as height maps or water

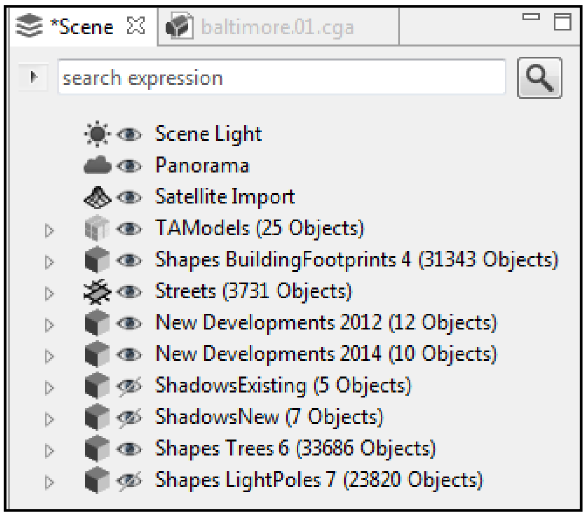
maps

- Models: The models folder holds exported files such as, .fbx, .dae, or .obj.
- Rules: The CGA shape-grammar rules are stored here as a .cga format.
- Scenes: The CityEngine Scenes are stored here

Scene Editor

In the scene editor in figure 14 there are five different layer types:

1. Environment Layers: The common control parameters like a scene's light or panorama are controlled by this layer.
2. Map Layers: The map layers hold all the maps that are controlled as object attributes.
3. Graph Layers: The street network and blocks are stored here.
4. Shape Layers: The shape layers used to generate CGA models by holding static shapes.

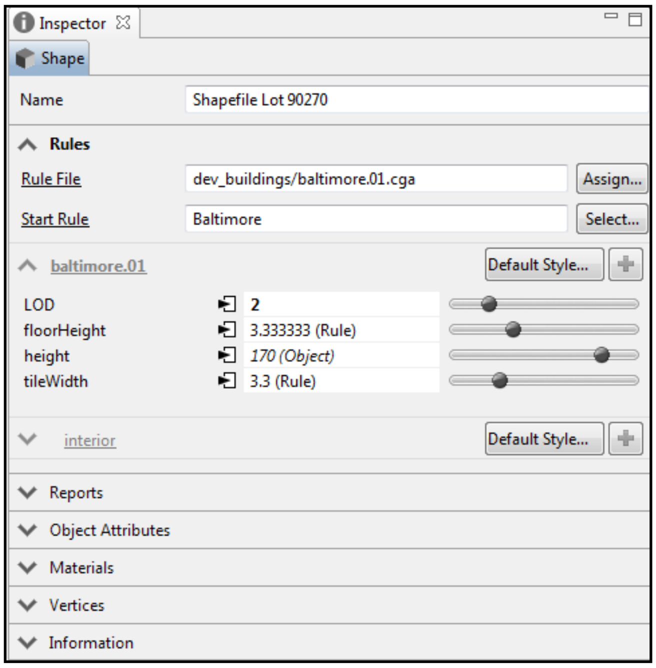


Scene Light
Panorama
Satellite Import
TAModels (25 Objects)
Shapes BuildingFootprints 4 (31343 Objects)
Streets (3731 Objects)
New Developments 2012 (12 Objects)
New Developments 2014 (10 Objects)
ShadowsExisting (5 Objects)
ShadowsNew (7 Objects)
Shapes Trees 6 (33686 Objects)
Shapes LightPoles 7 (23820 Objects)

Figure 14: shows the scene window in CE.

Inspector

The main tool for viewing and modifying objects in CityEngine is the Inspetor tool shown in figure 15. It allows the user full access to the object's attributes depending on the type of object. The inspector provides all attributes and parameters for the CE layer objects such as shapes. When the object is attached with a rule file all the parameters are available for modification. The start rule should is entered in the start rule field. If the start rule does not match any rule in the rule file, no generation will take place. The inspector has the ability to edit one object or a collection of objects. If the attributes are unique for certain objects they will be shown as-is but if there are some attributes with different values in the object collections then the attribute will be marked as a non-unique with the “?” symbol. However, in case of unique attributes, the user can change a value that will affect all the objects in the collection. This batch operation is useful for when the user wants to make an edit to a large collection of objects. The multi edit tool can also be used. It is possible in CE because of the ability of the inspector which automatically groups object collections by kind.



Name: Shapefile Lot 90270

Rule File: dev_buildings/baltimore.01.cga

Start Rule: Baltimore

LOD	2
floorHeight	3.333333 (Rule)
height	*170 (Object)*
tileWidth	3.3 (Rule)

Figure 15: shows the inspector window in CE.

CGA Shape Grammar for CityEngine

As we mentioned before, The CGA shape grammar is an exceptional programming language to generate 3D architectural content. The main idea of grammar-based modeling is to determine CGA rules within CE and build the design by creating more and more details. The rules operate on shapes that consist of geometry in locally oriented bounding boxes that are called the scope. The following figure 16 shows the rule derivation which start from the left with the start shape and on the right with the result of the generated model.

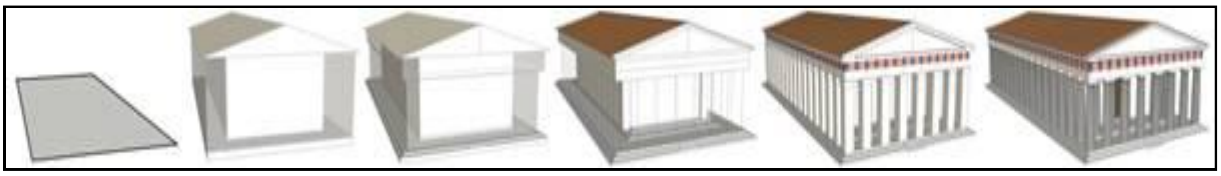


Figure 16: shows the rule derivation which start from the left with the start shape and on the right with the result of the generated model.

The following steps show the generation of building geometries with CGA:

1. The building lots are created in CE or imported. Mass models can be used as starting points.
2. The user considers which rule to apply on the shapes while also assigning one rule file to all shapes or part of the shapes.
3. Then, the user can generate the rule on selected shapes. The start rule must be present in the rule shape or there will not be any generation. The generated model can be explored in the 3D viewer. One of the problems in CE is that if the project is very large it is not recommended to generate all the buildings due to memory constraints.
4. CE provides different possibilities to edit the resulting 3D model:
5. The user can edit the rules.
6. Editing the rule parameters of the rule set.
7. If the user deals with stochastic rules, the random parameters of all single building can be changed.
8. When the model is complete the user can export the project to an external hard disk including the texture. There are no memory restrictions while exporting.

Maplayers

The main function in Map layers are: 1. Import the map object into the scene by using image data. 2. Providing maps of image data to several attributes. 3. The map layer has five types: terrain, texture, obstacle, mapping and function.

Terrain

The Terrain Layer in CE is a unique map layer which visualizes the elevation and topography of the scene using image data. In addition it work as a reference elevation for the align operations. The terrain layer builds a height map mesh elevation for the base of the scene.

Street network

CE provides dynamic ways to create streets and lot shapes from the graph network structures. This eliminates the extra work of creating and subdividing shapes. The Street Growth Wizard gives the user an easy way to create a city layout consisting of streets, blocks and lots. Different growth parameters can be used within the wizard such as, the intentional number of streets or the street pattern figure 17.

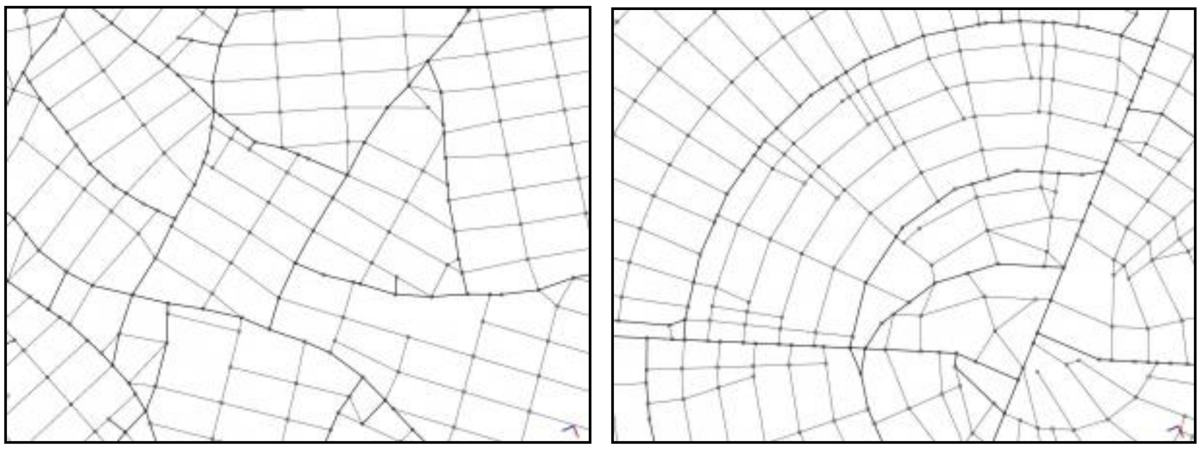


Figure 17: shows street patterns, Left: Organic major street pattern and raster minor street pattern. Right: Radial pattern for both major and minor streets.

Levels of Detail

CE Provides four levels of detail for each model:

1. LoD 0-Mass Model, is used as a proxy for navigations and the preparation of the camera. No texture is used in this level.
2. LoD 1- Low Detail, is also a Simplified model which can be used for walk-throughs and no interior in this level.
3. LoD 2- Standard level, in this level the number of polygon must be Reasonable. The standard level is a perfect model for a high-quality walk-throughs. Also, no Furniture is used here just the interior structure.
4. LoD 3 – High Detail, in this level an extremely huge number of polygons, including the stones projection on the wall and the roof tiles is used. Also, the model can be used for high-quality renderings and are a great test model for academia.

3D Web Scene

The 3D web scene is a web optimized format viewed by the CE Web Viewer and uploaded to ArcGIS online for sharing. The user can create a 3D web scene in CE Web Viewer which is a web application in CE. In addition, Web Viewer is based on WebGL technology that allows the user to view 3D content in web browser. By using WebGL, the user does not need to use any additional plug-ins or CE licenses shown in figure 18.

The functionality of the Web Viewer allows the user to interact with 3D city scenes such as:

- Panning, zooming and changing perspective when navigating the scene.
- Switching between layers.
- Changing the scene to expose different proposals and scenarios.
- Viewing features, attributes, and metadata that can be searched within scene content.
- Galleries for procedural models (from thumbnail Galleries for procedural)

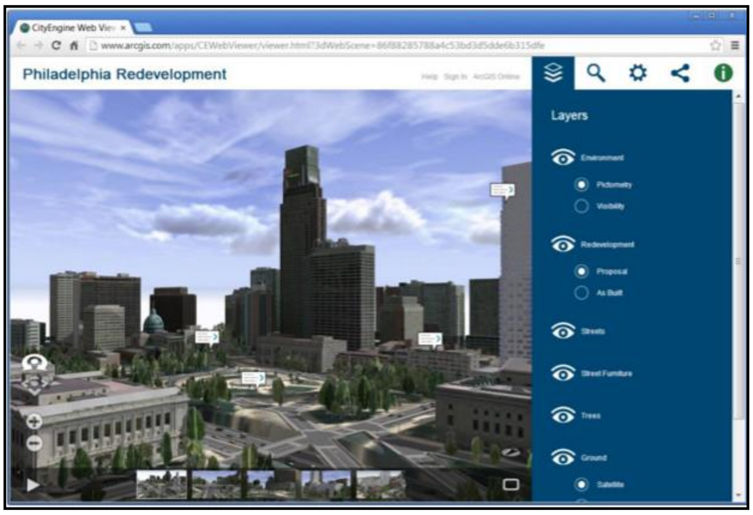

CityEngine Web Viewer
www.arcgis.com/apps/CEWebViewer/viewer.html?3dWebScene=86f88285788a4c53bd3d5dde6b315dfe
Philadelphia Redevelopment
Help Sign In ArcGIS Online
Layers
Environment
Pictometry
Visibility
Redevelopment
Proposal
As Built
Streets
Street Furniture
Trees
Ground
Satellite

Figure 18: shows Exporting ArcSceneTM to 3D Web Scenes.

i. CityEngine Pros and cons

Pros:
CE is program for the automatic generation of urban cities.
CE wizard allows to generated texture city.
CE is able to export the 3D model into many different formats.
CE has the ability to build the 3D model without using additional softwares.
The street network creation in CE can be done automatically.
Height map allows creating the terrain.
Almost all kind of format can be imported or exported in CE.
Computer Generated Architecture is the language that CE used to create the model.
OSM used to import the city layout.
After generating the 3D city model, CE allows to edit all the elements.
The texture in CE is a UV bit maps
There are four LoDs in CE
CGA rules split up, align, move, scale, rotate, extrude, colour and apply textures to shapes
By teaching the software how to build the shapes, the models can have variety.
The user can import any data from ESRI softwares to CE.
The Web scene helps to interact people with the work.

Cons:
CE is commercial software which highly cost.
There is no texture effect in CE such as luminosity or reflection.
The lack of importing the height map into CE, cause many difficulties to create the terrain.
There are no tools to help checking the grammar that the user writes in rule file.
Before start using CE, the user takes some time to get familiar with CGA functions.
CE does not support all forms of shapes such as arches or curves.
The memory limitations in CE cause a lot of crashes.
In the high LoDs, The user need more time to generate the model.
In the high LoDs, the user has to combine different softwares to CE to increase the details on the facades.

2.8 Description of CityGML

CityGMLis an open data, model to storage and exchange, virtual 3D city model based in XML. It is an application for the extendible international standard for spatial data exchange Geography Markup Language 3 (GML3) issued by the Open Geospatial Consortium (OGC) and the ISO TC211. CityGML was developed in order to define basic entities, attributes, and relations of a 3D city model. It is remarkably useful considering the cost effective sustainable maintenance of 3D city models where it is possible to use the same application in different fields. CityGML also handles thematic and semantic properties of representation, classification and accumulation. Geometrical and thematic models are included in CityGML. In 3D city models, the spatial objects have geometrical and topological properties that can be defined consistently by the geometry model. However, all the objects within the model take their properties from the basic objects CityObjects.

The thematic model of CityGML implements the geometry model into different thematic fields like Digital Terrain Models, sites, vegetation, water bodies, city fittings and transportation. Moreover, the additional objects, if not obviously modeled, can be represented by applying generic objects and attributes. Some spatial objects have the same shape and are placed repeatedly in different positions. For example, trees can be modeled as prototypes and used many times in the model of a city. Additionally, single 3D objects can be combined into complex buildings by using the grouping concept. When an object is not geometrically modeled by closed solids then it must be sealed virtually so its volume can be computed. For example, tunnels, pedestrian underpasses or airplane hangars can be sealed by using Closure Surfaces. It is important to integrate 3D objects at their correct position within the Digital Terrain Model. This can be done using Terrain Intersection Curve which prevents buildings from floating over or disappearing inside the terrain.

There are five sequential Levels of Detail (LOD) in CityGML. Increasing the LOD, which relates to both of the geometrical and the thematic description objects, generates more detailed objects. Textures and material can be assigned to fit the surface of the objects. CityGML has the option to save each object as multiple representations (and geometries) at different LODs. The obvious representation of aggregated objects is allowed by generalization relations over different scales. Moreover, there can be external references for the objects to correspond to objects in external datasets. The attributes for Enumerative objects are limited to external code lists and values which are defined in re-definable external dictionaries. In the next section an overview of the relevant work in the field of procedural modeling is given.

CityGML Overview

CityGML is a universal information model and known as an open data model with XML- based storage and exchange for 3D virtual city models. According to the OGC standard, CityGML has played the most significant role in the modularization of urban geospatial information. CityGML is an application that represents both the geographical appearance of 3D city models and semantics properties in version 3.1.1 (GML3). Yet the current version of CityGML is 2.0 and it has improved by gathering a new attributes and content according to OGC 2014. The official 3D models of a number of cities such as Stuttgart, Bonn and Berlin have been made using CityGML in order to demonstrate and support the fields of urban planning, city marketing, and to induct new enterprises (Döllner et al. 2006). The analysis of noise in the North Rhine, Westphalia was conducted by EU Noise Mapping using CityGML for the purposes of environmental protection and sustainability (Czerwinski et al. 2006a; 2006b and 2007). Homeland security in the United States and other government sponsored geo-related analyses has been conducted using CityGML in driving and traffic simulations (Pantzer 2008).

Problems of CityGML

Version 1.0 of CityGML has some problems that should be solved in the future according to (Kolbe, 2007). This includes the size of the file which becomes very large even if the file size is reduced using Gzip compression. The effectiveness of processing XML can be a problem due to memory limitations and Web features and services that have access need to be recognized in an asynchronous way in order to avoid timeouts (OGC 2009). Moreover, the complexity of the city influences any project using CityGML and makes it complicated because it is impossible to implement all the details in city(Mao,2011). For this reason the creators of CityGML have released Version 2.0 in order to solve certain technical problems like backward compatibility as defined by OGC policies and guidelines (http://www.citygml.org/)

CityGML module and LODs

CityGML’s core module makes up the CityGML data model and depends on all of its thematic extensions (OGC 2008). The basic classes such as CityObject, Address, Geometry, and Feature are defined on the core module. They make up eleven thematic extension modules that are implemented in verison 1.0: Appearance, Building, CityFurniture, CityObjectGroup, GenericCityObject, LandUse, Relief, Transportation, Vegetation, WaterBody, TexturedSurface. The issue that arises is that some Applications support only a part of the thematic fields of city objects. Based on the application requirements, the extension modules can be arbitrary. A profile is the combination of more than one module and the union of all modules is CityGML base profile. The concepts of generic city objects is incorporated with the application data.

CityGML has five LoDs that it uses to reflect independent data collection with various application requirements. It simplifies the efficient visualization of objects and data analysis shown in Figure 19. LoD0 represent a 2.5D Digital Terrain Model. LoD1 is the blocks model cover buildings with flat roofs. LoD2 represents distinguished roof structures and surfaces. LoD3 has a detailed roof and wall structure model. LoD4 combines LoD3 and an interior model (OGC 2008.pp.9)

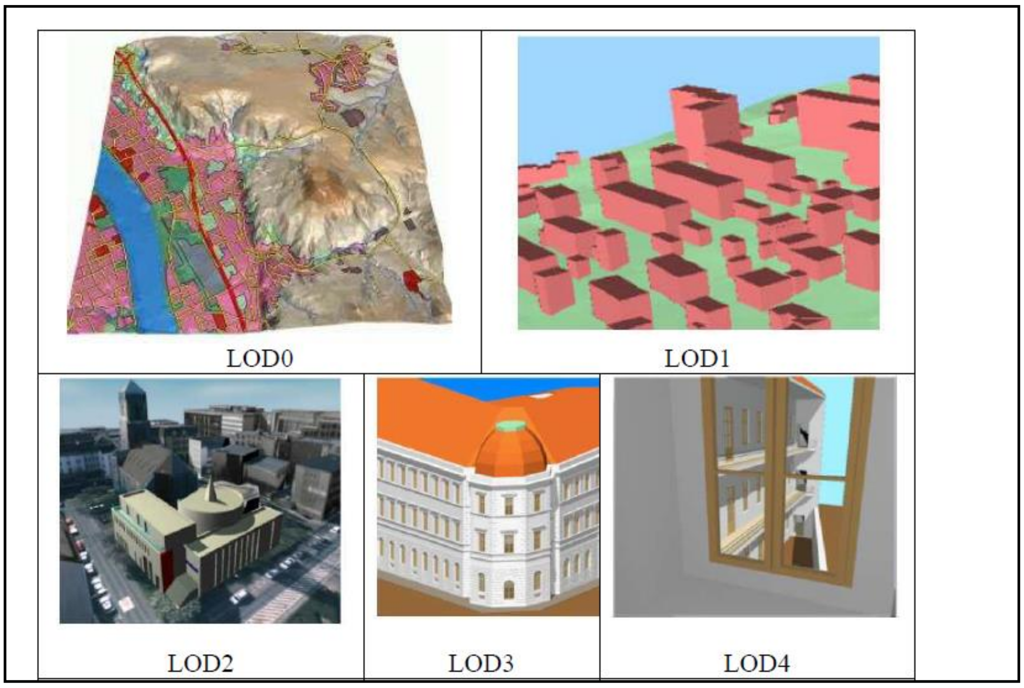


Figure 19: shows the five LoDs defined by CityGML.

CityGML Resources:

CityGML is an open standard that has a wide range of support from both the academic and industry fields. The developers for CityGML visualization, manipulation and management software include both open source and commercial options

Visualization software includes:

- The Aristoteles (2011) is an open source viewer for CityGML developed by the Institute for Cartography and Geoinformation, University of Bonn.

- LandXplorer Xpress Viewer (Autodesk 2011).
- FZKViewer (2011), developed by the Institute for Applied Computer Science, Karlsruhe Institute of Technology.

- CityVu (2011), Ptolemy3D (2011), BS Contact
- Geo 7.2 (2011), FME 2011 Special (2011) is also support CityGML Visualization Software for manipulating CityGML data includes:

- Google Sketchup has a plug in that allows the user to import and export CityGML for editing.
- The Citygml4j is an open source java class library and API. It is developed by
- The Institute for Geodesy and Geoinformation Science of the Berlin University of Technology.
- The HfT Stuttgart (2011) developed QS-City 3D and is a free online service used to check CityGml data. In addition, for management there is 3DCityDB (2011) an open Source 3D geo database used to store, represent and manage virtual 3D city models. It was developed by the Institute for Geodesy and Geoinformation Science at the Berlin University of Technology. The database for 3DCityDB consist of a semantically rich, hierarchically structured, multi-scale urban objects facilitating complex GIS modelling and analysis tasks.

Visualization of 3D city models

CityGML is not able to present or visualize 3D city models directly. As (Kolbe, 2008, pp. 28) points out: “CityGML is complementary to visualization standards like X3D or KML. While these address presentation, behavior, and interaction of 3D models, CityGML is focused on the exchange of underlying urban information behind 3D objects”. Therefore, an efficient 3D city model presentation requires specific 3D visualization technologies. The advent of computer graphics required that visualization occurred on local hardware (Carlson 2003). For example, if the user wanted to draw a line, then they had to insert information about the start point, end point, monitor resolution, frame buffer, and related system calls. Also computer platforms had many differences so visualization program portability was quite low. Standard APIs, like DirectX 2011 and OpenGL 2011, that produce 2D and 3D computer graphics can simplify this type of development process and further increase the portability for writing applications.

These APIs interact with the hardware to provide the function calls that can be used to draw complex 3D objects from simple primitives. The hardware can also be upgraded by manufactures without any detriment to how the applications are supported by the APIs. However, some problems still remain such as, APIs are complicated and sometimes too primitive such that they only support automatic geometric object. This forces the user to calculate the projections, transformations, and render the scene which is not suitable for a complex 3D model. Secondly, there are too many APIs such as OpenGL, DirectX, Mesa 3D (2011), VirturalGL (2011), RISpec (20110), Glide (2011) and all of these have different versions as well.The 3D city models should be viewable by anyone from anywhere therefore these basic graphic APIs must be employed for online 3D city models visualization. The internet is a great tool for reaching broad audiences and so the use of 3D standards has helped spawn all of these APIs. This allows many different types of consumers to interact with the visualization of 3D city models. Currently there are 18 platforms that utilize the various APIs which support these 3D standards they include VRML (Bell et al. 1995), X3D (2011), 3DMLW (2011), COLLADA (2011), and KML (2011).

Extensible 3D is an open source standard for 3D Web delivered graphics. X3D has a specific geometry definition language a run-time engine for architecture and an API that provides an interactive, animated environment for 3D graphics. X3D is used in many fields such as e-learning (Thomas 2008), medication (Jung et al. 2008, Willis 2007, Hamza-Lup et al. 2006) and georelated applications. The 3D web-based visualisation of HLA- compliant (High Level Architecture) Simulations provided by (Araujo et al. 2008) uses X3D. Virtual cultural heritage has used X3D (Eliens et al. 2007), (Cabral et al. 2007) and BIM, GIS, and 3D maps can be implemented in X3D (Nurminen, 2006, 2007), (Alessandro et al. 2007). Consequently, X3D has been improved and has many tools to support it. Bit Management Contact viewer (2011), Octaga (2011), Flux (2007) and InstantReality (2011) are free viewer software that supports X3d. The X3DOM is a free open source framework used to integrate X3D with HTML5 to make 3D visualizations through web browsers.

Geo-Visualisation applications utilize X3D Earth, an open standard, which is a profile for 3D geo visualization over the internet. It has been used in many projects and applications such as, terrain Generator in Rez, JeoSpace globe generator, WorldWind (2008) with an X3D loader, Digital Nautical Chart (2011), Planet 9 Virtual Cities (2011), MBARI Monterey Bay operations (MBARI 2010), and NPS Savage Studio scenario creation (X3D Earth 2011). The OGC and Web3D association have an agreement to combine X3D with OGC standards such as CityGML (Havele 2010).

KML/COLLADA

KML is an XML language which displays a geographic data in online services such as google Earth and Google Maps. The OGC approved that KML is a standard in April 2008. KML used COLLADA (COLLADA borative Design Activity), an open standard for XML to exchange digital assets through different graphics software applications, to be able to visualize 3D models. COLLADA is mainly used in the game industry. The main function of KML/COLLADA is for an earth browser while X3D is preferable to show the 3D city model in online services. X3D is compatible with HTML web and well known browsers like Chrome and Firefox. 3D models of Berlin have been published using Google Earth in KML format (Kada 2009). Several GIS applications anticipate KML for their visualization. Many researchers use KML such as (Calado et al.2008) when he exported hydrographic data into KML and (Chen et al. 2008) who used Google earth to visualize the earthquake impact. KML is an important option for visualization of the city model especially if it merges with CityGML and X3D for better visualization.

Other standards:

X3D and KML/COLLADA are not the only 3D visualization standards. Others such as 3DMLW, O3D and U3D exist and will be briefly outlined in the following secion. 3DMLW is an XML based format filed used for the web and generates 3D and 2D interactive content. The use of 3DMLW, a markup language, requires a particular plug-in and OpenGL for rendering. 3D Technologies R&D developed the 3DMLW plug-in for common browsers such as, Internet Explorer and Mozilla Firefox. O3D is a JavaScript API developed by Google to allow the creation of interactive 3D graphics applications. Universal 3D (U3D) is used as a compressed file format for 3D Computer graphics and its 3D objects can be exported into a PDF and visualized by Acrobat Reader. The main difference between X3D, KML/COLLADA and the previous standards is that these standards are not international used or they have only been created for use by certain applications not approved by OGC or ISO.

Visualization using CityGML

The OGC standard can be used to combine both geometric and semantic information of the city model as done in the Berlin virtual 3D city models by (Döllner etal.2006). The Berlin project is the origin of the CityGML standard. Figure 2.6 shows the architectural structure of the overall Berlin 3D City Model System. The 3D authoring system is accountable for creating, editing, and versioning of the 3D model. It has importing, exporting, grouping and annotating buildings, vegetation plans, and landscape plans components. The 3D Authoring System also provides interactive access to the 3D geo-database. The 3D Geo-Database System is the database that stores and manages virtual 3D city models based on the logical structure of CityGML. This 3D Geo-Database System is the core system of CityGML shown in Figure 20 and utilizes several geo-dataset resources such as Cadastral Data, Digital Terrain Models, Aerial Photography, and Building Models that are converted to CityGML data, and incorporated in a combined database. The exchange of 3D city models in CityGML is easier for different applications and visualization. This allows for a concentration of utilizing CityGML without having to worry about the interoperability of different software packages and suites. Once an authoring system and database have been acquired the city model can be moved in to the presentation phase. A 3D Presentation System should be able to provide real-time visualization and interaction with the virtual 3D city model. In order to reach as broad an audience as possible the

3D city models should be able to switch between KML and X3D formats. The Virtual Berlin project, for example, can be viewed using both the KML (Google Earth) and X3D formats (LandXplore).

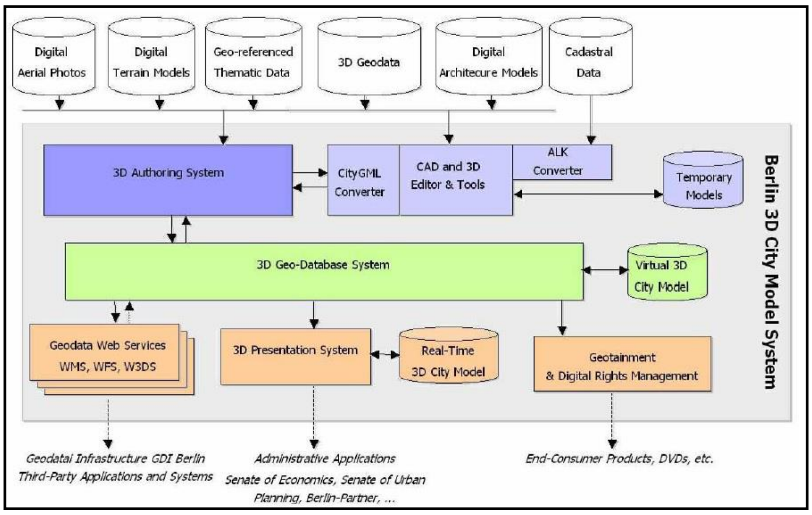
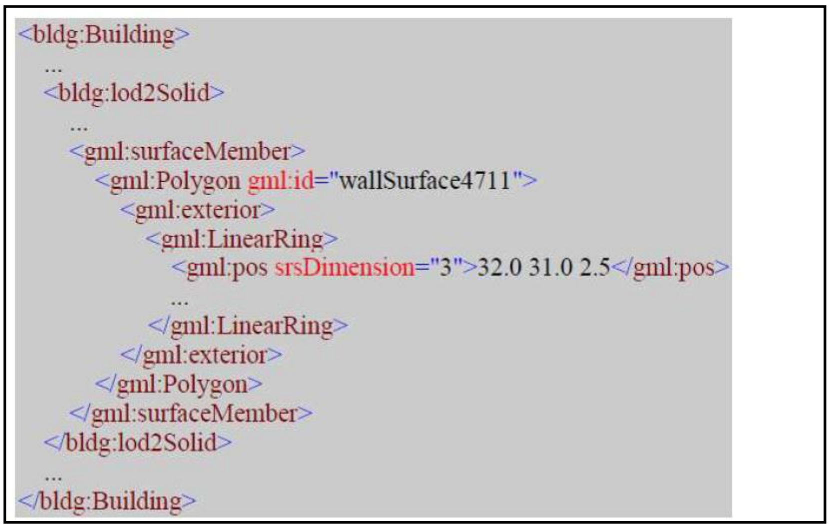

Digital Aerial Photos
Digital Terrain Models
Geo-referenced Thematic Data
3D Geodata
Digital Architecure Models
Cadastral Data
3D Authoring System
CityGML Converter
CAD and 3D Editor & Tools
ALK Converter
Temporary Models
Berlin 3D City Model System
3D Geo-Database System
Virtual 3D City Model
Geodata Web Services WMS, WFS, W3DS
3D Presentation System
Real-Time 3D City Model
Geotainment & Digital Rights Management
Geodatai Infrastructure GDI Berlin
Third-Party Applications and Systems
Administrative Applications
Senate of Economics, Senate of Urban Planning, Berlin-Partner, ...
End-Consumer Products, DVDs, etc.

Figure 20: The system architecture of the virtual 3D city model of Berlin (Source: Döllner et al. 2006, pp. 4).

Geometry Representation

CityGML has adopted the GML3 geometry model to represent spatial properties. The 3D geometry is recorded using the well-known Boundary Representation (B-Rep, cf. Foley et al. 1995). Each dimension contains a geometrical primitive for example, a zero-dimensional object is a Point, a onedimensional is a Curve, a two-dimensional is a Surface, and a three-dimensional is a Solid (OGC 2008). CityGML uses polygons to represent the surface which determines a planar geometry. The (OGC 2008) provides an example of a CityGML file as shown in Figure 21. The Figure shows a polygon for a wall surface with the id 4711. The polygon is a part of the geometry property lod2Solid of a building. Moreover, CityGML supports many kinds of surfaces such as OrientableSurface and TriangulatedSurface.



```
<bldg:Building>
  ...
  <bldg:lod2Solid>
    ...
    <gml:surfaceMember>
      <gml:Polygon gml:id="wallSurface4711">
        <gml:exterior>
          <gml:LinearRing>
            <gml:pos srsDimension="3">32.0 31.0 2.5</gml:pos>
            ...
          </gml:LinearRing>
        </gml:exterior>
      </gml:Polygon>
    </gml:surfaceMember>
  </bldg:lod2Solid>
  ...
</bldg:Building>
```

Figure 21: CityGML surface example (OGC 2008, pp. 25).

Texture:

CityGML also uses raster-based 2D textures which are defined by an image URL that can be any arbitrary image data accessible via the internet. The texture specification comes from COLLADA. The accessing approach for the image texture has five types of wrap mode: none, wrap, mirror, clamp and border shown in Figure 22. The texture images use coordinates in order to be projected onto surfaces. The texture coordinates of the CityGML standard are viable only on polygonal surfaces. An outright mapping of the surface's vertices to points in texture space is defined using the Appearance tag. Figure 23 is an example of the texture mapping where the numbers denote texture coordinates (Source: OGC 2008, pp. 33).

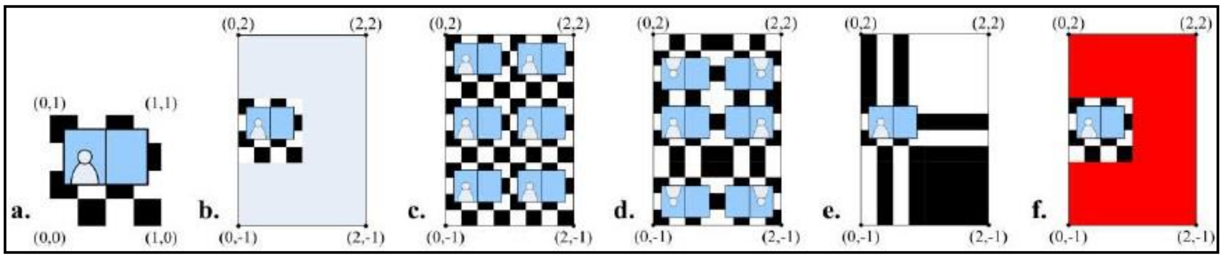
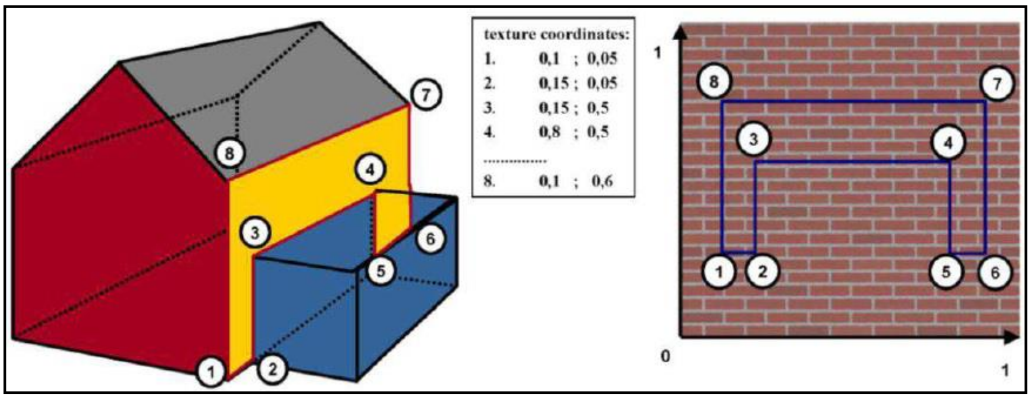


Figure 22: CityGML texture (a) applied to a facade using different wrap modes: (b) none, (c) wrap, (d) mirror, (e) clamp and (f) border. The border color is red.



texture coordinates:

1.	0,1	; 0,05
2.	0,15	; 0,05
3.	0,15	; 0,5
4.	0,8	; 0,5
....		
8.	0,1	; 0,6

Figure 23: Positioning of textures using texture coordinates (Source: OGC 2008, pp. 35).

CityGML Pros and cons

Pros.
Open data model with XML based storage and exchange for 3D virtual model
Represent geographic appearance and semantic properties
Citygml core module makes up the citygml data model (defined all the basic classes such as CityObjects, address) and depend on all of its thematic extension
It hsa five LoDs which use to reflect independent data collection
Citygml include open source and commercial softwares.
Google Sketchup has a plugin that allows to import and export CityGML
CityGML is focusing on the exchange of urban information behind 3D objects.
There are18 platforms that utilize the various APIs which support the 3D standards.
X3D allows to BIM and 3D maps to used in culture heritage.
OGC standard for CityGML allows for integration different type of data.
The user can easily exchange of 3D city model with different applications.
3D presentation system is providing a real time visualization and interaction with the 3D virtual city model.
CityGML uses raster based 2D textures which are defined by an image URL.
The texture uses coordinates in order to be projected into surfaces.

Cons.
Version 1.0 has some problems: the size of the file becomes very large
Effectiveness of processing XML can be a problem due to memory limitation.
The complexity of the city influence any project using citygml
Some applications only support a part of the thematic fields of CityObject.
CityGML is not able to present or visualize 3D city models directly.
An efficient 3D city model presentation requires specific 3D visualization technologies.
The APIs interact with the hardware to provide the function that can be used to draw complex 3D object.
The hardware can be updated without any detriment to how the applications are supported.
Some APIs are complicated and sometimes too primitives.
The user has to calculate the projections, transformations and render the scene which is not suitable for a complex model.

2.9 Related Works

The next section will review several popular examples of using procedural modeling: the modeling of Mayan Xkipché in Mexico, the modeling of ancient Roman Pompeii in Italy, Roman Reborn project and generating alternative proposal for the Louvre.

2.9.1 Puuc Buildings in Xkipché

Xkipché is a small town in the Mexican state of Yucatan about 9 km southwest of Uxmal. It has been well studied over the past 15 years and its above ground remains have been mapped. The well preserved buildings found in Uxmal and Kiuic are great examples of the Pucc style. Puuc is a subtype of Mayan architecture that is distinguished by a veneer-over-concrete construction technique which forms geometric and repetitive façade structures. Researchers from the University of Bonn began archaeological excavations and detailed studies of exposed building remains in 1991. They created non-detailed 3D reconstruction models of the building manually using CAD software.

Architectural shape grammars were used to procedurally generate 3D reconstructions of an archaeological site of the Puuc style buildings found in Xkipché, Mexico (Müller et al. 2006). The modeling system was used to reconstruct the site based on GIS (Geographical Information Systems) inputs such as building footprints, architectural information, and elevation. The results demonstrated that their modeling system, in contrast to traditional 3D modeling, was able to efficiently construct a large number of high quality geometric models at low cost.The procedural modeling approach is more efficient because it does not require the amount of manual work of traditional modeling techniques. Moreover, it is possible to test several hypotheses by adjustingparameters when using procedural modeling approach resulting in the creation of a powerful platform for archaeological discussion and exploration. Müller et al. (2006) also had publichsed the detailed formal descriptions and grammatical encodings (CGA Shape) used to model the Puuc stone buildings in Xkipché. Archaeologists recognized 18 building types among the Puuc style buildings found in Xkipché. The houses of Puuc are built on a platform substructure with an exterior of cut and stucco stone that is filled with densely packed gravel. The walls of the building are made of rubble-filled concrete faced by a thin veneer of polished stone. The dominant characteristics of Puuc-style architecture in Xkipché are a plain lower wall (with openings) above a rather elaborate base molding, and on the upper part of the façade a large medial molding, a frieze (with or without decoration) and a usually high cornice molding (Pollock.1980).

The openings in the walls include doors and small rectangular ventilators below the medial molding. The width of the door openings average about 100-120 cm. Sometimes the doors are framed by columns with simple, rectangular capitals (Carver.1986), complemented by small corbels at the top of each jamb. Wood was used for door lintels and in corbel without structure. If there is a decoration in the frieze it consists of colonettes that are serrated cylinders or mosaic elements of limestone masonry. This creates geometric repetition and symmetry. Long- nosed masks which represent the rain god Chac are other typical elements that decorate Puuc façades and are found over doorways and at the corner of buildings.

The Puuc style predominantly incorporates the use of three-member molding among several different type of molding (Pollock 1980). It consists of an apron member, and an upper reverse apron member, a middle rectangular member, a decorated member and a rectangular member on top. Some of the cornice moldings in Xkipché consist of four members while other moldings have only one or two member. Many of the ancient stone buildings found in Xkipché are partially still standing which is advantageous because the respective elements found in the buildings, like the shapes of stones, are often very specific. In most other scenarios buildings are destroyed and the archaeologists would try to attribute the architectural fragments to their original place in the facade. The decoration details of the buildings in Xkipché remain uncertain and even though a precise reconstruction of the original buildings can conducted based on the number of stones found, we cannot ignore the fact that decorated stones of the old buildings have been replaced by the inhabitants themselves shown in figure 24.

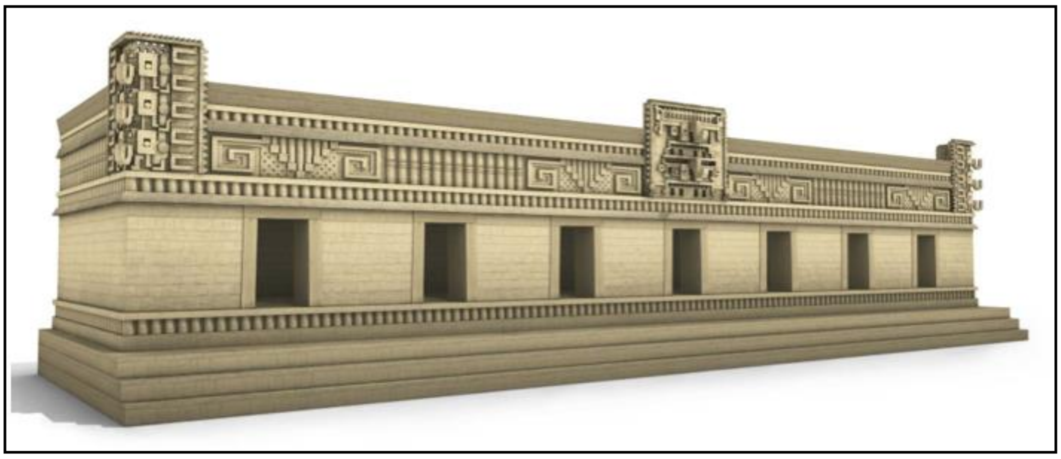


Figure 24: Detailed reconstruction of one of the few highly ornamented buildings in Xkipché.

The whole building has been generated procedurally by using the CityEngine, except the complex mask ornaments which have been created with traditional mesh modeling software. The image has been rendered in Pixar's RenderMan.

2.9.2 Pompeii

Pompeii was a Roman city that was completely entombed by the lava from the volcanic eruption of nearby Mount Vesuvius. The city's past glory has been revived through the use of a 3D model and its people once again walk the streets of this model through use of Virtual Romans. The model consists of an annotated simulation of ancient life in Pompeii based on arachaeological data. The model was generated using procedural modeling and contains semantic data such as land usage, building age, and window and door labels. The semantics were then interpreted automatically to populate the simulations with crowds of ancient Pompeiians. Many European cultural institutions have banded together in order to improve the use of Information and Communication Technology for Cultural Heritage by creating EPOCH (EPO). Reconstructing the ancient city of Pompeii was one of the projects conducted by EPO. Previously, a similar reconstruction, using Virtual Romans, had been produced by LIFE- PLUS cultural heritage project (PAPAGIANNAKIS et al. 2005). This project utilized GIS data such as building footprint 2D maps combined with airborne sensors data to create the model (TAKASE Y et al. 2003) (SCHOLZE S et al. 2002).

While these photogrammetric systems generated impressive results, they were not appropriate for the reconstruction of destroyed archaeological sites because the textures and the missing features needed to be added manually. A related method uses a combination of GIS data and procedural modeling based on shape-Grammar and L-systems (PARISH et al. 2001). Shape grammars are used to analyze and construct the archaeological designs using a set of production rules to operate on shapes directly. By applying these repeatedly, more detail is added to the model. However it should be noted that the derivation process was done manually. After the introduction of Split Grammars by Wonka et al. (WONKA et al. 2003), the operations of Geometric splits were used as a basic approach to subdivide building facades hierarchically. From this approach, Mueller et al. (MÜLLER et al. 2006) introduced the CGA Shape grammar which can be adjusted manually. This approach contains essential features like a rule syntax, context-sensitivity, practical shape definition, occlusion handling and global synchronization of splits. Cellular textures can also be used complementary to shape grammars in order to compute brick patterns.

Extensive studies of Pompeii have been made due to the well-preserved nature of city. Pompeii's architectural evolution is quite apparent with unique styles ranging from the 4th-3rd century B.C.'s Italic model to the 1st century A.D.'s Imperial Rome style. The city is divided into blocks and each block consists of individual homes built contiguously shown in figure 25. The houses differ in size and style. Small homes only have two rooms house to while larger buildings may contain several rooms and a yard. One of the latest Pompeii models was created by Maïm et al. using CityEngine. The grammar rules, a combination of volumes and a repetitive subdivision scheme, were enhanced with a level of detail capable of utilizing detailed textures and geometries in addition to the semantic data used for the crowd engine.

Additionally, the rules included a set of parameters such as, facade proportions, building dimensions, and door widths that can be single values or several values with upper and lower limits determined by archaeological findings (MUELLER et al. 2005).

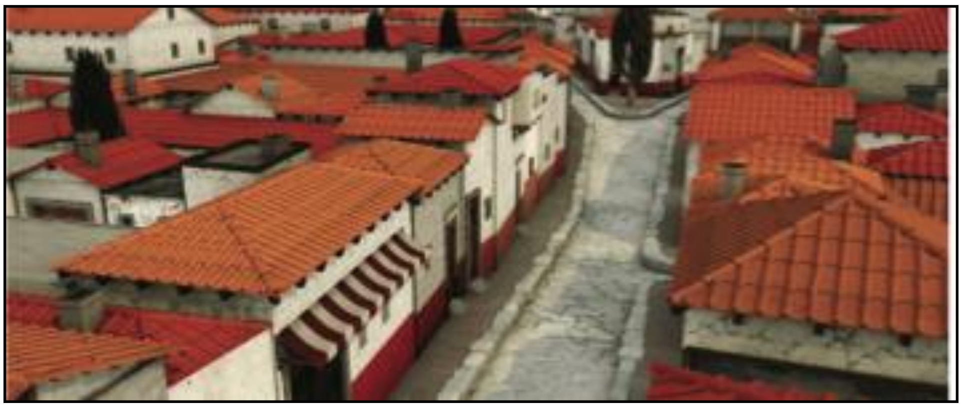


Figure 25: This (hypothetical) street-level view of ancient Pompeii was created by translating archeological drawings, figures and ground-plans into CityEngine rules and rendering the resulting geometry with Pixar's RenderMan.

2.9.3 Roman reborn

Rome Reborn (www.romereborn.virginia.edu) is an international initiative based in the Virtual World Heritage Laboratory at the University of Virginia (http://vwhl.clas.virginia.edu). The project aims to illustrate ancient Rome urban developments (ca. 1,000 BC) from the late BronzeAge to the early MiddleAges (ca. 552 AD). Ancient Rome was at the height of its urban development in 320 AD and so this was the period of time chosen for the beginning of the modeling process. The model consists of an entire city and is one of the largest scholarly virtual reconstructions ever made. It was constructed in two stages: version 1.0 which was first published by exhibition by Walter Veltroni, the Mayor of Rome, and the Rome Reborn Project Director Bernard Frischer on June 11, 2007. Version 1.0 consisted of 9 million polygons. Version 2.0, the current version, underwent a dramatic upgrade of the modeling and has over 400 million polygons along with the geometric detail of many individual elements.

The model contains Class I and Class II elements. Class I elements (~250) are sites that can be readily identified because of the availability of detailed information about location and design. Over 30 sites have been modeled so far using commercial 3D database authoring software such as 3D Studio Max and Multigen Creator. Class II elements lack precise location or design information but they consist of 6,750 buildings and monuments. These elements have been identified from two late-antique catalogues of the building stock of the city. Schematic architecture, and procedural modeling techniques were employed in order to create visually compelling and detailed models of numerous Class II elements.

Version 1.0 of the Rome Reborn model's Class II elements were all made by hand. Positions were determined using laser scan technology of a large scale physical model of ancient Rome. This technique of modeling is great improvement compared to the mesh made model which came directly from the scan data but it carries with it a number of problems. The resultant models were schematic in nature and the architectural detailing of ornamental decorations on doors, windows, and balconies came from textures not geometry. Therefore, an aesthetic discrepancy occurred caused primarily by the stark contrast of the highly detailed Class I models compared to the Class II models which were not properly placed on the DTM (Digital Terrain.Fixing this discrepancy using traditional modeling techniques would have taken an extreme amount of time and resources. The research team chose to use procedural methods instead and so Rome Reborn 2.0 was created to address the discrepancies. Procedural Inc. based in Zurich, Switzerland utilizes "CityEngine" software to create large- scale 3D environments efficiently by employing a shape-grammar-based geometry generation system called "CGA shape". This scripting language was developed in the last decade and is an enhancement of the set and shape grammar syntax that is optimized for architectural content. Rule sets written with CGA shape are adaptive which means it is possible to apply them to 2D or 3D shapes and they will adapt to the spatial dimension of the initial shape automatically.

The archaeological consultants Claudia Angelelli and Bernard Frischer provided an extensive number of images, floor plans, and statistics to design the grammar rules for Rome Reborn 2.0. They started with the reprocessed scan data of the physical model of the city captured by Prof. Guidi's team at the Politecnico di Milano. Then the footprints and mass models of Rome Reborn 1.0 were imported into CityEngine. The mass model, which was represented as polygon data of arbitrary topology, was subdivided into facades, roofs or interiors by Novel split algorithms. Thereafter, and based on archaeological data, the newly developed grammar rules generated the highly detailed 3D building models for the entire city.

The reconstruction of numerous temples in Rome was also completed with CityEngine in addition to the domestic buildings. Doric, Ionic and Corinthian temples were created by one grammar rule set that was written based on the well-described rules of architecture shown in figure 26. The final appearance of the generated model is controlled by the modification of hundreds of attributes within the rule set. The process of collecting geometric information of the ancient temples is limited because most of these temples have been destroyed or damaged. Therefore, the proportions were implemented in a rule set to give the best appearance of the temples. Known parameters were entered by archaeologists and the remaining unknown parameters were calculated as proportions to the known parameters. This generated a complete model of the temple with all the architectural elements aligned automatically.

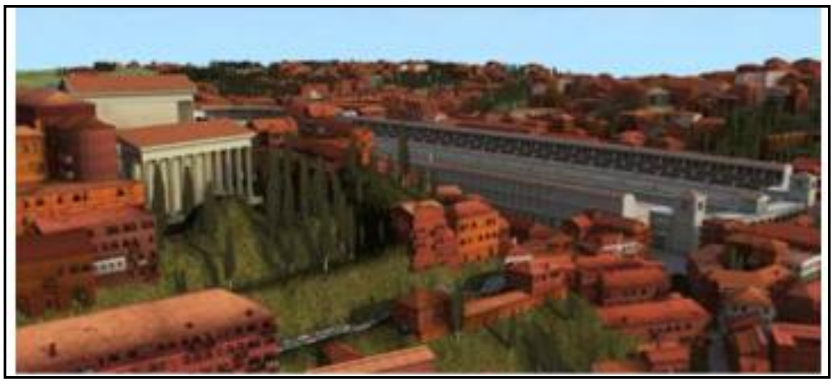


Figure 26: Rome Reborn 2.0 model rendered with Mental Ray, showcasing both procedural and hand modeled content and showing a major Class I monument (the Circus Maximus in the middle ground) integrated with the filler architecture of Class II.

Rome Reborn project also consists of a variety of detail levels used for certain buildings and structures. The highest level of detail used has approximately 50,000 polygons per building, while the mid-level which only has several hundred polygons and the low level consists of only textured massing models. Each level of detail was published in sections, and in order to make the model more dynamic each section was divided up to fit within the 14 regions of the ancient city of Rome. The generated models were then exported by CityEngine to visualization software, using the file formats COLLADA, FBX, OBJ, RIB, and Mental Images' MI. This process runs in batch mode which means the buildings are created and written one by one to a hard disk because the city model can grow too fast and large. The Class II buildings, which are procedurally generated and parametrically modeled temples, were integrated with existing Class I detailed landmarks.

Since they all have the same footprint for model generation, the placement and scaling is proportionate throughout all of the elements of the city model therefore there is no need for transformations. This allows the seamless integration of both the procedural and hand-modeled content in the platform. Thereafter, all the models were exported to the .mi format to be used directly with remote rendering software such as Mental Images' "RealityServer". RealityServer 1 is a remote visualization softwarepackage from MentalImages in Berlin, Germany which allows for the remote navigation of extremely complex models like the Rome Reborn model which has more than 400 million polygons at its highest level of detail. And in order to do that it uses a unique server-side progressive rendering technique.

The visual improvements of the project after applyingCityEngine in the 2.0 version were substantial and also increased the complexity and polygon count of the model.

This made it all the more challenging to access and render the model in real time. The real-time rendering of this large model also used realistic environmental lighting and shaders that made it difficult to explore the model without the use of RealityServer which allowed for remote manipulation of the model over the Internet. If the end user has a browser and a flash plug-in then the model is readily accessible via the Adobe interface. While the RealityServer does all of the extreme processing of the model the image rendering displays at arbitrarily high-resolution while also allowing for various standard camera manipulation techniques, walk navigation, and real-world distance measurements.

2.9.4 The Louvre

The east wing of the Louvre has undergone two 3D digital reconstructions using procedural modeling. The first physical reconstruction is Louis Le Vau's which may have been built before the “Colbert Consultation” between 1664 and 1672 and the second physical reconstruction is the “petit conseil” that still stands today. The purpose of the 3D digital reconstructions is to allow experts and public to examine and explore how the Louvre palace might have looked like in the past. Moreover, it allows for analytic and stylistic comparisons of both versions of the Louvre.

The project is currently still in progress and documents the evolution of designs and stylistic choices of Louvre’s east wing. This allows for the formal analysis of the building using systematic 3D representations. The generation of the facades of the proposals was created using formulated procedural rules that were based on building plans and elevations obtained from the Louvre archives. These rules were written as CGA files in the CityEngine software package (Calogero et al 2011). The interest to use procedural modeling in this work is due to the number of shape grammars set up that include architectural style for the use of the Cultural Heritage representation. From the previously discussed examples of ancient Rome, Pompei and Maya, some models have been defined often with random parameters to allow for reasonable and interesting variations. The shape grammar hypothesis used for the facades of the two reconstructions comprised of tracing an initial outline of the building footprint from available plans and source material. There are more detailed building volumes, in the case of Le Vau, that required the interpretation of a human to produce component polygons. These polygons corresponded with available plans and section information which contributed to the volume mass of the final shape. While complex polygons can be automatically divided into Simpler ones using computer algorithms it is more suitable to solve this problem manually (Calogero et al 2011). Subsequent steps included shape extruding, creating a hierarchy of splits, exporting the footprints as polygons in the .obj format into CityEngine and reflecting the information available about facades through the use of insertions and transformations. The first two splits in bars are along the top and sides of the elevations. Further splits are nested into the first split as shown in figure 27. However, it is important to realize that only absolute dimensions were specified based on the information available from existing plans.

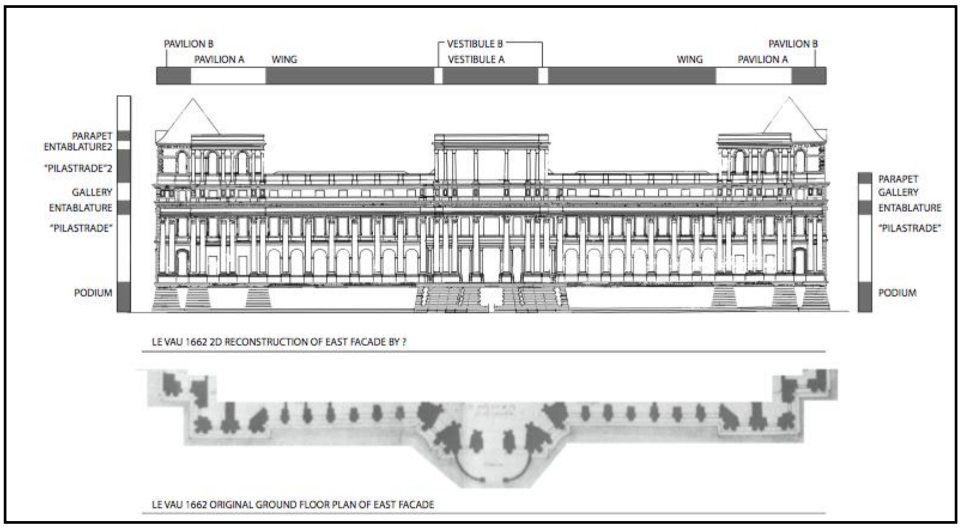


Figure 27: Diagram of Splits for the Le Vau Design of 1662

2.10 Description of the Main Site at Nemi

Archaeological site inquired into this thesis is located in Nemi basin. It is a cultural landscape that contains many archaeological remains located in the Alban Hills about 25 kilometers southeast of Rome, Italy (Figure 18). The name Nemi is derived from the Latin word Nemus which mean "holy wood". They consist of two small volcanic craters surrounded by the medieval towns of Nemi which have the remains of the famous Roman structures and the temple of Diana Nemorensis, and Genzano di Rome.

The area is known for the tunnel built to regulate the lake level the Roman Emmissario. Nemi also contains some Roman remain like villas , stone quarries, basalt paved roads, cisterns, aqueducts, churches and chapels, the rock cut Romitorio S. Michele beside medieval murals and a destroyed large complex (Le Mole) located south of Nemi. (Storemyr 2004). Archaeologists have predominantly focused on the temple of Diana from which a wealth of artifacts have been removed by people throughout the centuries.The most recent eruption of the local volcano the formed the Nemi crater around 45.000 years ago. Seven major earthquakes have been reported in Nemi area between 1806 and 1927 from the "Catalogues dei Forti Terremoti in Italia, 461 a.C.-1990. The 1806, 1892, 1915 and 1927 earthquakes were the most damaging. The effects of the 1927 quake included the damage or collapse of 136 buildings and ancient monuments. It is important to keep in mind that many archaeologic remains have been affected and impaired due natural and human activity, especially Roman walls and cisterns (Storemyr 2004).

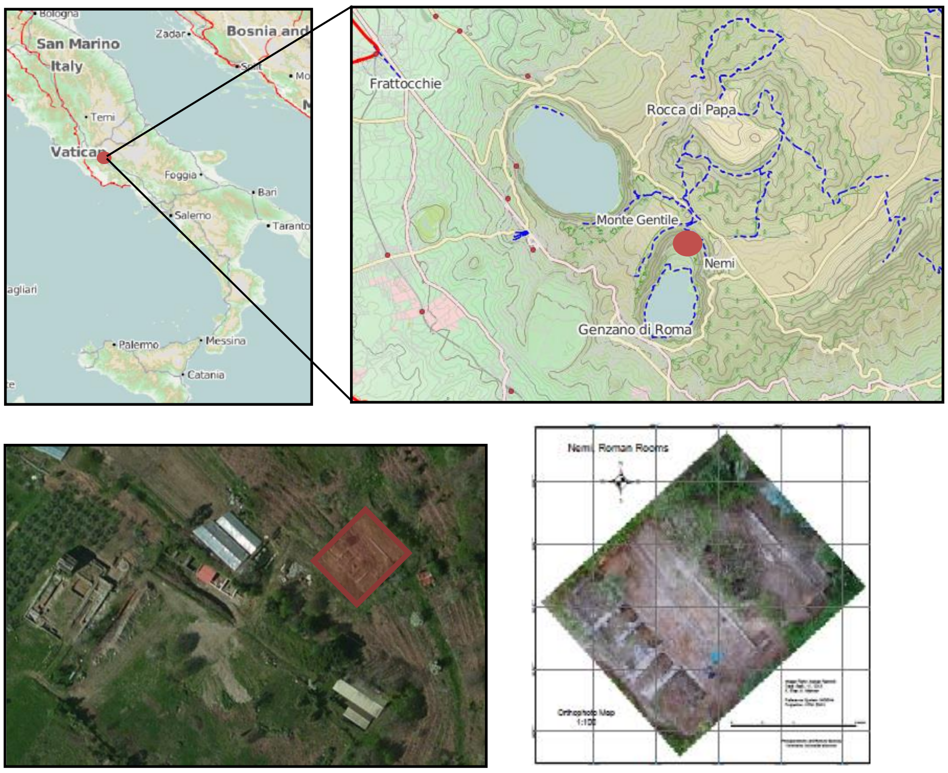


Figure 28: The location of Nymphaeum in Nemi, source: OSM, 2014 and TUM satellite image

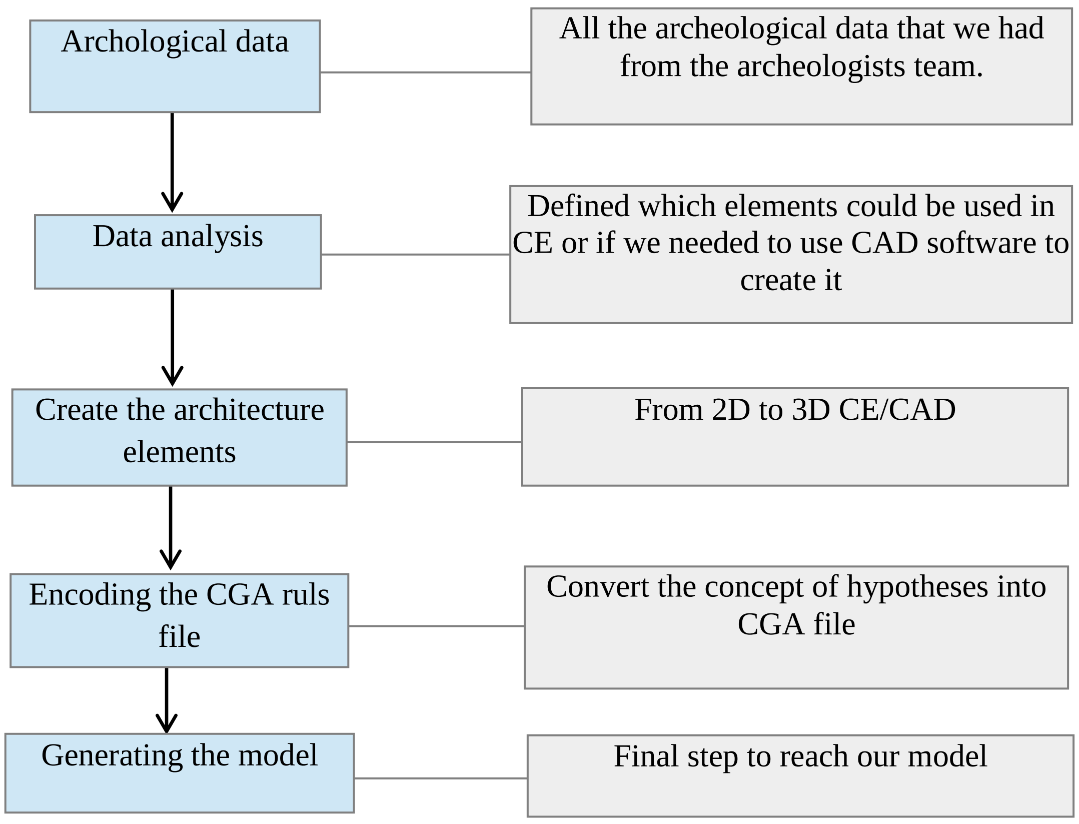
Nevertheless, we are using some of the previously discussed technologies in this chapter to produce high quality representation model for our archaeological reconstruction (the Roman Rooms). By using "CityEngine" the tool for procedural modeling, due to its efficiency to explicit uncertainties while keep the realistic appearance of the model. Further demonstration on the method used in this project in the next chapter.

3 Methodology

In the reconstruction of Nymphaeum project, CityEngine had been used to obtain the final result. Our goal was to test the ability of CE to make a realistic reconstruction of archeological data that we had from the site in Nemi of the Roman Rooms. Also, to create a 3D visualization of the current excavation. The Features of CE were used to build four LoDs and two hypotheses. However, because of the limitation of CE in the creation of complex shapes, the study had to combine another system to reach high LoDs. The CAD system was the right option to use in the study in conjunction with CE because of the wide range of software and techniques. The study faced many problems starting with collecting the data right up until generating the model. The problems and the methods used to solve those problems are described in the follow subchapters.

3.1 Overall structur

The method that was used for reconstruction of the Nymphaeum combined CE with CAD system software to reach four LoDs and two hypotheses. The structure of the workflow in figure 29 shows the sequence of steps that allowed the creation of the 3D model. The study started with archaeological data to build our understanding of each part of the project. According to the workflow, the study had five steps for creating the 3D model.



Archological data
All the archeological data that we had from the archeologists team.
Data analysis
Defined which elements could be used in CE or if we needed to use CAD software to create it
Create the architecture elements
From 2D to 3D CE/CAD
Encoding the CGA ruls file
Convert the concept of hypotheses into CGA file
Generating the model
Final step to reach our model

Figure 29: sho ws the workflow of the reconstruction of Roman Room.

3.2 Archaeological Data Descriptio

The work in our implementation was based on literature and archaeological data provided by the collaboration of TUM, and Dr. Francesca Diosono which was used to create the 3D model of the Nymphaeum reconstruction. Dr. Francesca Diosono is a member of the archaeological team working in the Sanctuary of Diana. The data was categorized into five classes that will be discussed in the following section.

3.2.1 GIS Data:

The GIS data includes a shape file and elevation points file which were created in a previous excursion by TUM. The shape file contains the polygons of Roman Room reconstructions illustrated in figure 30. The elevation point's file contains the height points used to create the DTM shown in figure 31.

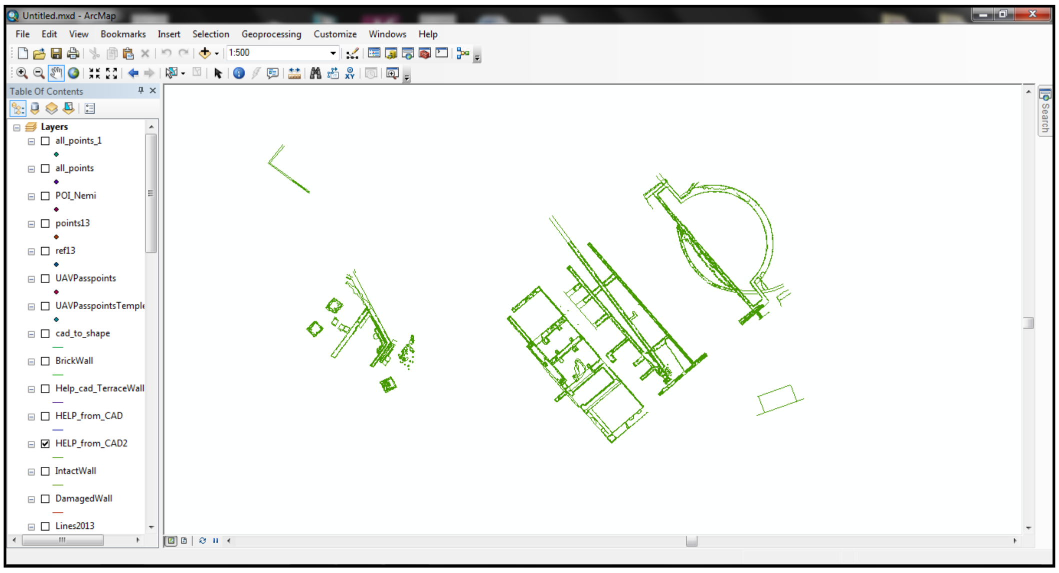


Figure 30: shows the shape file contains the polygons of Roman Rooms.

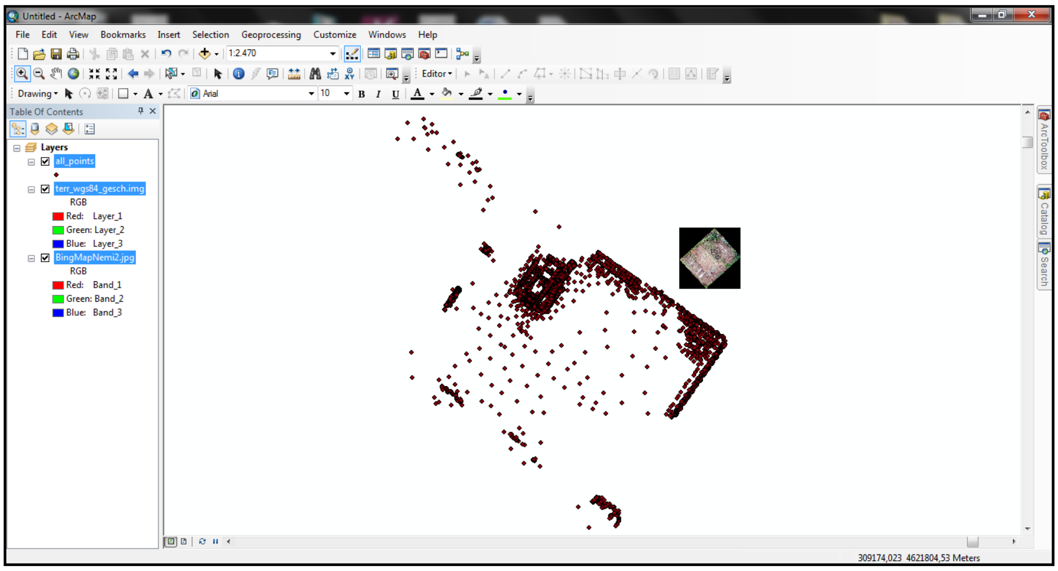


Figure 31: shows the height points in ArcMap.

3.2.2 The First Hypothesis:

The first concept for the Nymphaeum model was created during a previous archaeological study of the site provided by Dr. Francesca Diosono shown in figure 32. The 3D modeling of Nymphaeum was given a general overview of the building and the different levels of the ground surface surrounding the building.

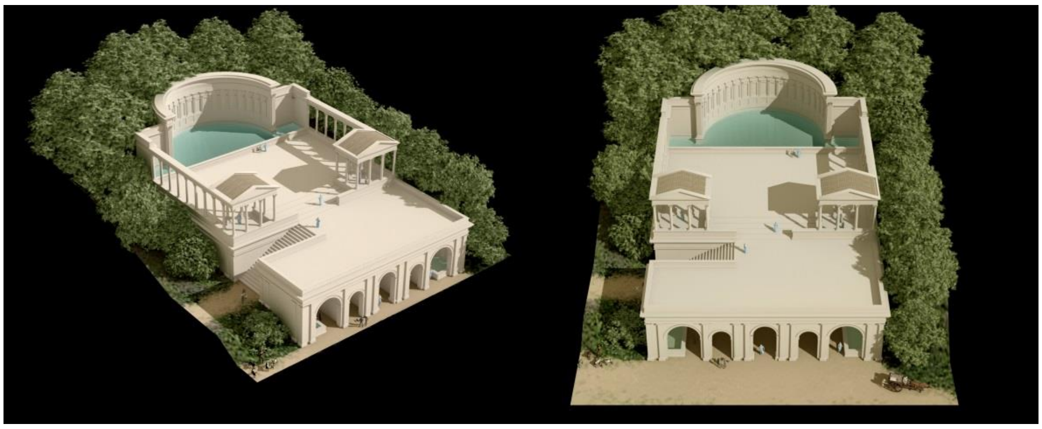


Figure 32: shows the first concept of Nymphaeum Reconstruction.

3.2.3 2D Plan and Elevations:

The 2D plan shows how the rooms were divided and enables the user to extract measurements of the ground floor as shown in figure 33. The Elevations of the building are presented in figure 34 and give the ability to understand different elements of the building. They provided the study with a contextual framework that describes the overall impression of the building including details like floors height, sizes and types of arches, columns, roofs and decorations.

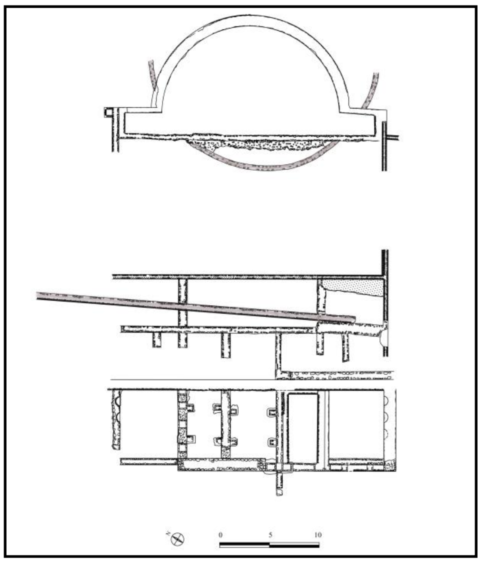


Figure 33: shows the ground plan of Nymphaeum Reconstruction

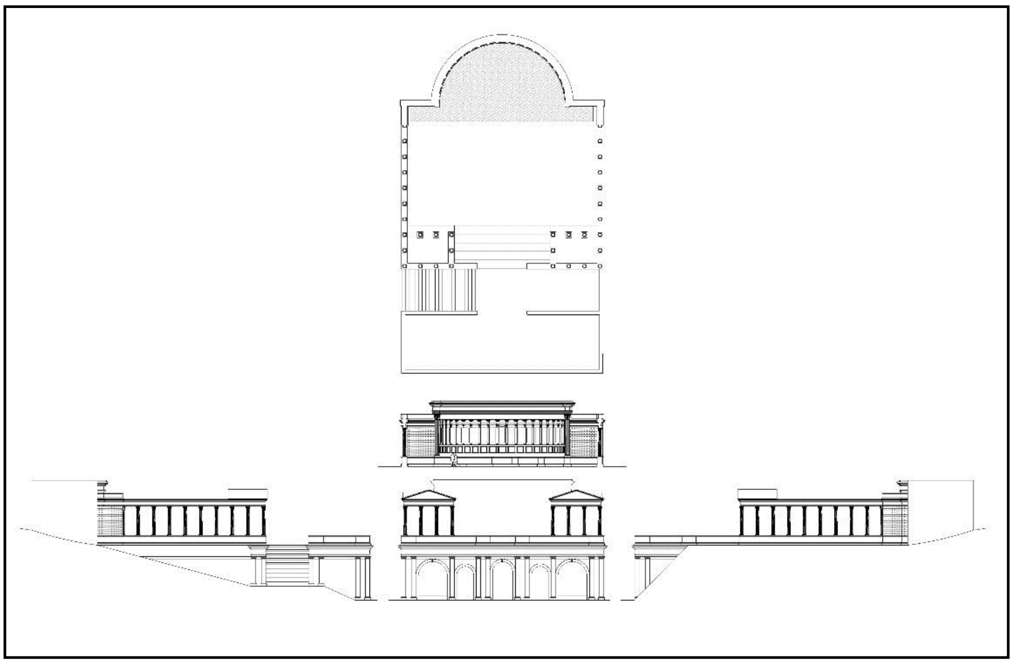


Figure 34: shows the Elevation of the Nymphaeum Reconstruction.

3.2.4 Architecture Elements Reconstruction:

Dr. Francesca Diosono also provided more details about the building elements such as the design and ornamentation of the building’s interior, roofs shown in figure 36, ceilings shown in Figure 38, and rooms located on the ground floor . The structure also contains a number of fountains with unique designs and pipes found in the first and last room of the ground floor shown in Figure 37. The mosaic for the rooms on the ground floor shown in figure 39 also utilize a particular type of decoration provided by Dr. Francesca Diosono shown in Figure35.

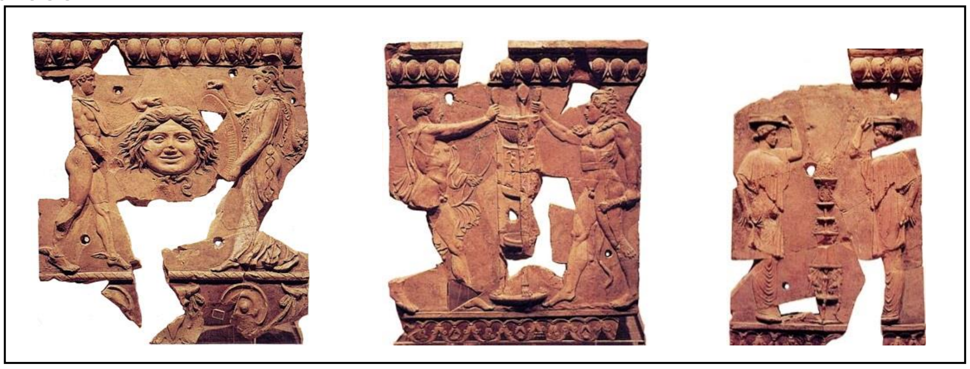
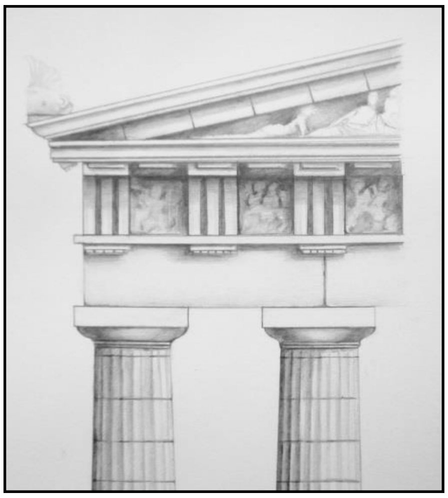


Figure 35: shows the decoration images.



.

Figure 36: shows the roof structure.

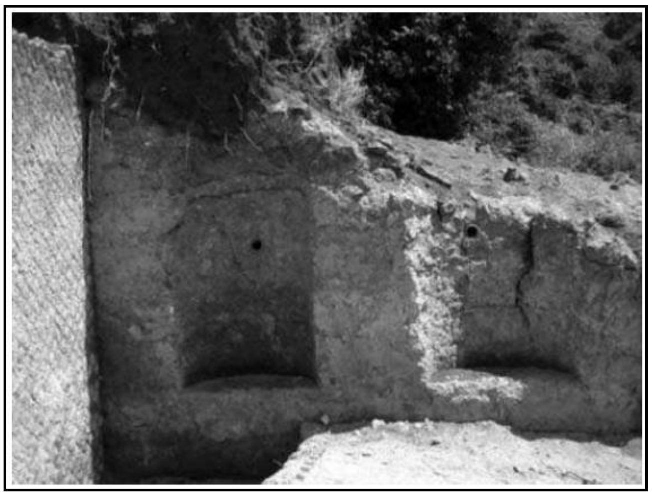


Figure 37: shows the fountains in ground floor.

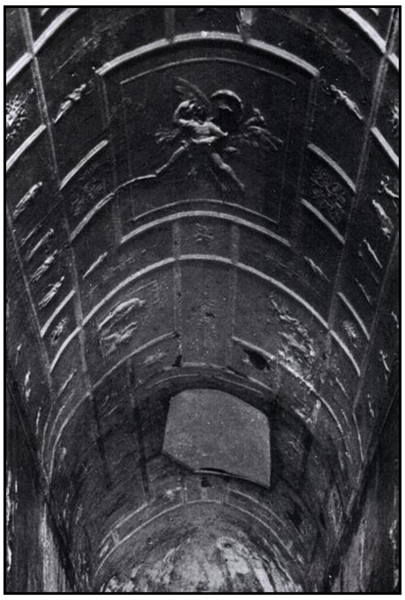


Figure 38: shows the ceiling type in ground rooms.

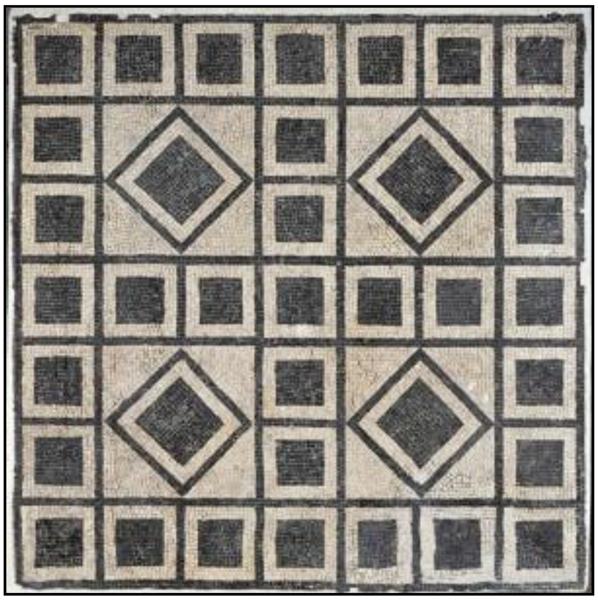


Figure 39: shows the ground floor texture.

3.2.5 Second and Third Hypotheses:

The archaeological team had another hypothesis for Roman Room reconstruction. The second hypothesis predicts a different concept for the rare wall at the back facade which contains columns with statues between each column shown in Figure 40. Also, the third hypothesis predicts that the front facade changes the sizes of the arches and shapes on the ground floor shown in Figure 41.

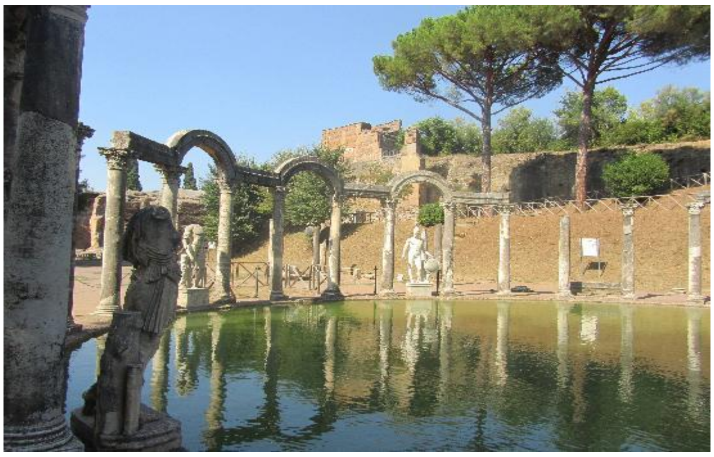


Figure 40: shows the second hypothesis of Nymphaeum Reconstruction.

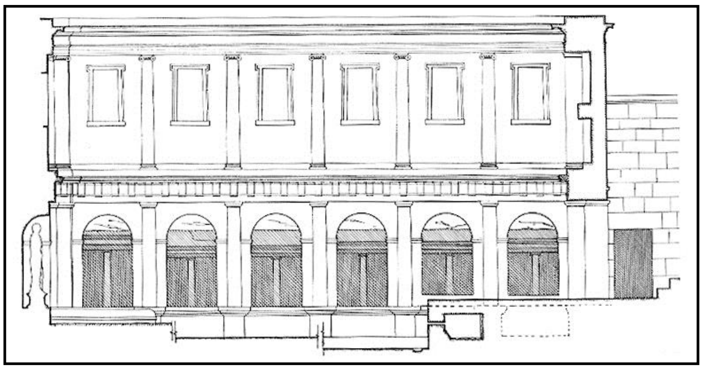


Figure 41: shows the third hypothesis.

3.3 Data analysis

Data analysis was the second step in the workflow. CE is a procedural modeling software, as mentioned before, so the study of the archeological data had to be categorized based on the limitation of the software. The study faced some problems when reconstructing curved shapes and other complex shapes. Moreover, in CE there were difficulties when measuring the necessary information of the reconstruction. The 3D concept of the Nymphaeum model in figure 32 which was created during the previous archaeological study, gave us a full understanding about the position of the construction elements and the level of floors in the entire building. In addition, the study was able to determine the rest of the elements and the techniques that should be used in the study. The 2D elevation image of the first hypothesis was converted to a CAD file using AUTOCAD 2014 in order to extract the measurements that would be used in the next step which was to create the 3D architecture elements figure 42. This included the type of the roof, Arcade, architrave, and the Doric order under the roof which was measured and determined as the distance between columns. The area between the roofs was valuable data that was used to build our knowledge of the building area on the first floor.

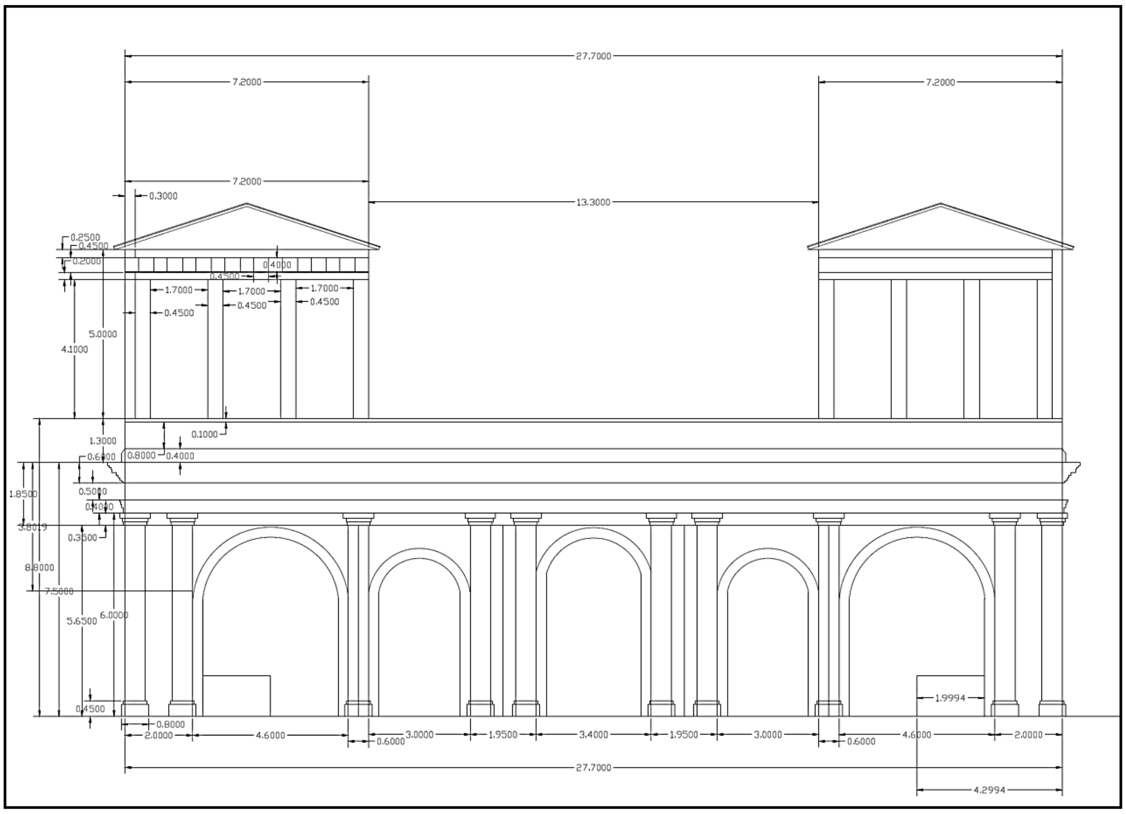


Figure 42: The front facade converted to CAD file source (the author work).

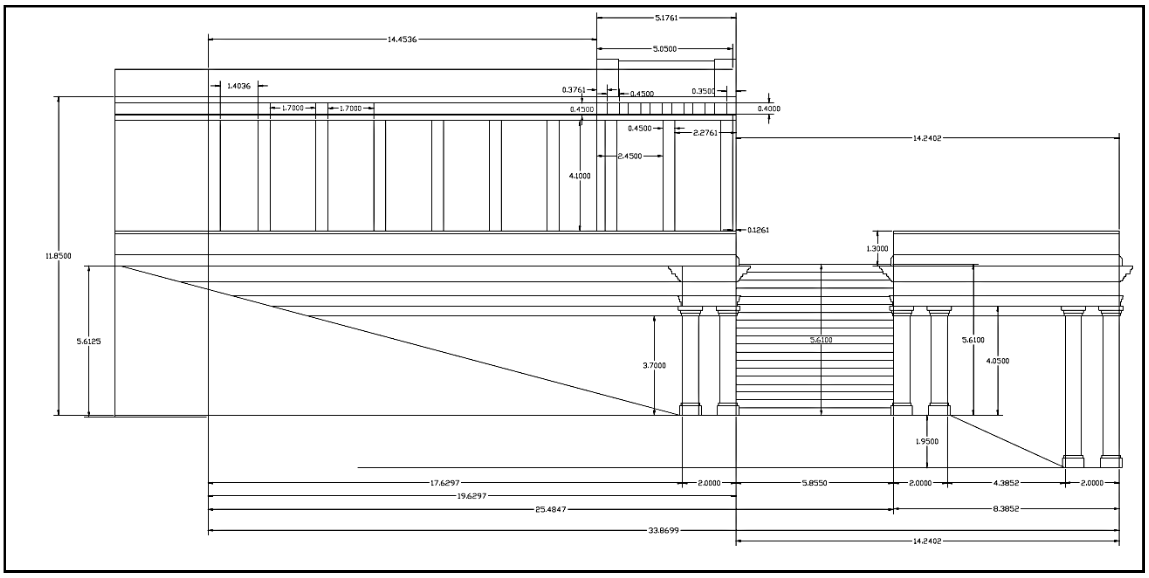


Figure 43: shows the Left facade converted to CAD file source (the author work).

From the CAD file in figure 43 the height of the ground floor was measured and determined using the difference between the arches. The columns' positions and sizes were also measured. The CAD file was then allowed to extract very accurate measurements of the facade which helped create high detail elements for the different LoDs and the two hypotheses. In addition, the end of the ground floor rooms, including stair width and height data was also collected enabling them to be measured from the 2D CAD file. All these analyses allowed the improvement of the final result of the Nymphaeum reconstruction. The analysis of the plan showed the size of the back area of the building which was used as water reservoir according the Archeological information provided by Dr. Francesca Diosono. In particular, this data was collected from the 2D image plane of the base of the Nymphaeum building that we had from the archeological team in Nemi, Italy in figure (). The positions of the arches in the Second and third rooms from the left could be determined from the 2D plan. The GPS points in figure 31 that were collected during the campaign in September 2014 were used to make the DTM (Digital Terrain Model) of the Nymphaeum area. ArcGIS was then used to convert the GPS points into the raster map. Next, the raster map was converted to the height map which was used in CE to build the terrain figure 44.

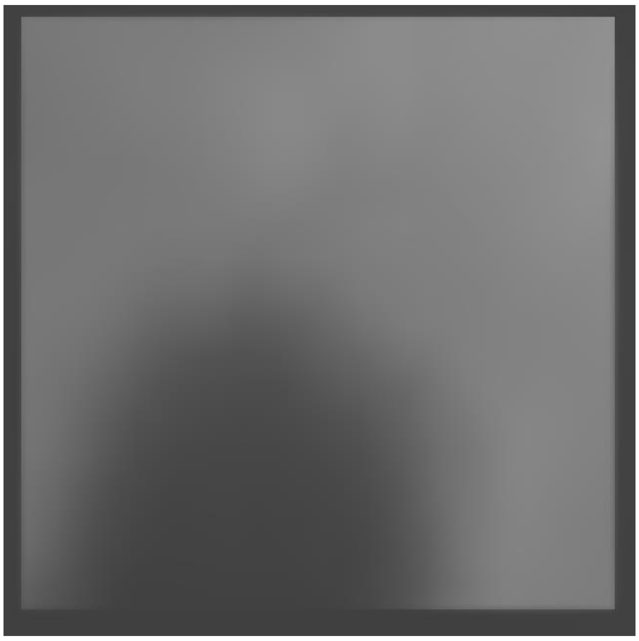


Figure 44: the raster map which converted to height map.

The data analysis of the archeological data gave enough information to decide which elements could be used directly in CE and which should involve CAD system techniques. In addition, the elements that used the CAD system were then imported into CE.

3.4 *Create the Architecture Elements*

The third step was to convert our understanding of the 2D information into 3D modeling. The CAD system was the primary technique used to increase the levels of detail of the reconstruction of the Roman Rooms. Data that was collected from the last steps, including information on arches, columns, roofs, fountains, and decorations were finally created in order to be imported into the next step. The creation of the 3D elements started with arches which contain two categories. First the basic shape of the arches on the ground floor. The measurements of these arches were determined from the AutoCAD file in figure 45. There were three different sizes of arches that needed to be built in different files that were then to be imported to the rule file figure 46. Each one of them was built using the Revit software which is a CAD 3D modeling software used to convert 2D data into 3D elements. The second type was the high LOD which shared the same size as the first type and contained the same basic structure but included more decoration elements. The first arch size was 4.6 M in width and 5.5 M in height, the second was 3 M in width and 4.9 M in height and the third arch was 3.4 M in width and 5.5 M in height.

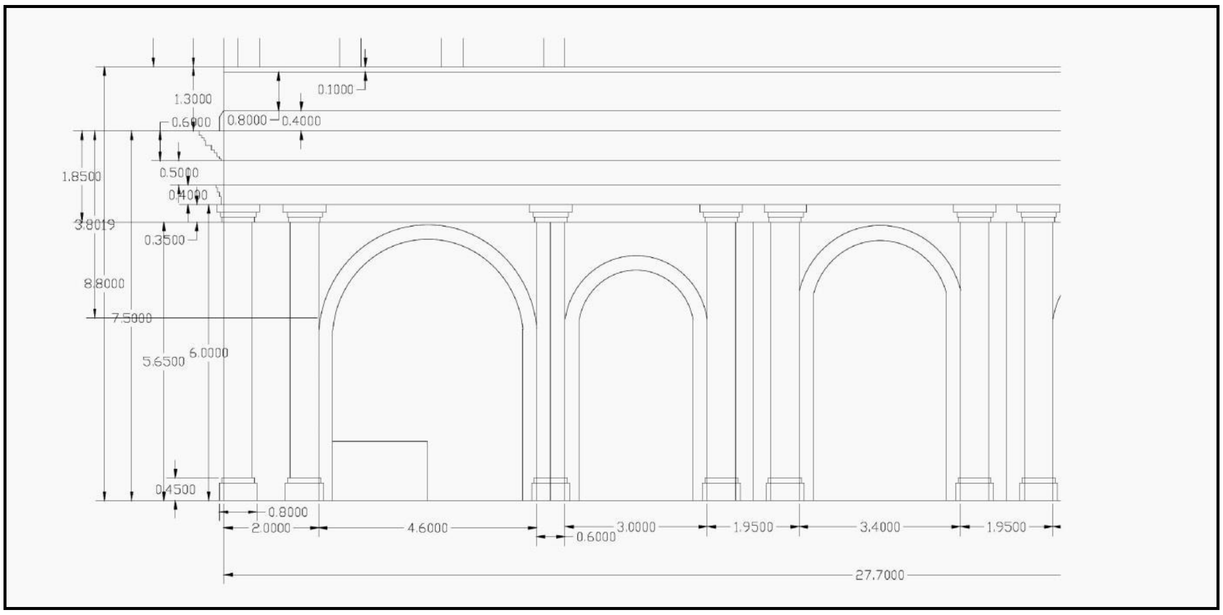


Figure 45: shows the size of the arches in the ground floor façade.

The AUTOCAD file imported to Rivet, allowed modeling exact replicas of each arch. The three sizes of the arches were built in two versions to support the LODs which is shown in figure 46.

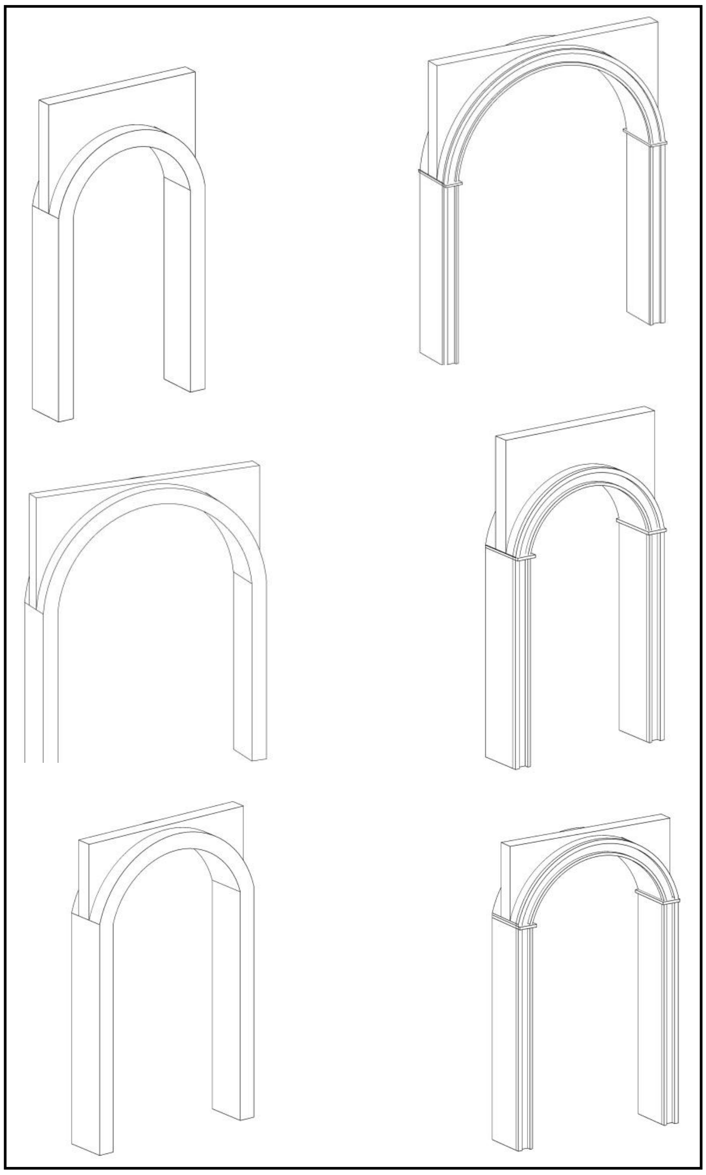


Figure 46: shows the two types of LODs and the sizes of the arches.

The next elements created were the columns which contained three different LoDs. By using the same techniques, the 3D model was created and Imported to the rule file. The Doric style was the column type in the high and very high LoD. The Doric column consists of five parts, the column, astragal, necking, echinus, abacus figure 47. The measurement of columns had to be transferred to the Rivet software from the AUTOCAD file and then rebuilt in 3D. The columns were located on the both ground floor facade and the first floor under the roofs. The figure 48 shows the two LoDs of the columns.

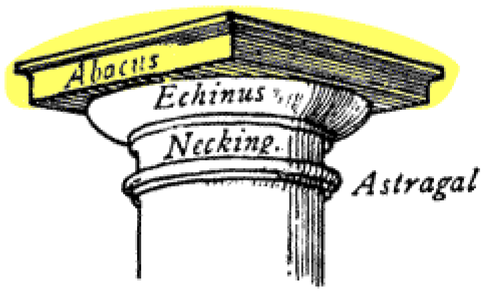


Figure 47: shows which parts that consist the column.

(source http://www.doric-column.com-(2014)).

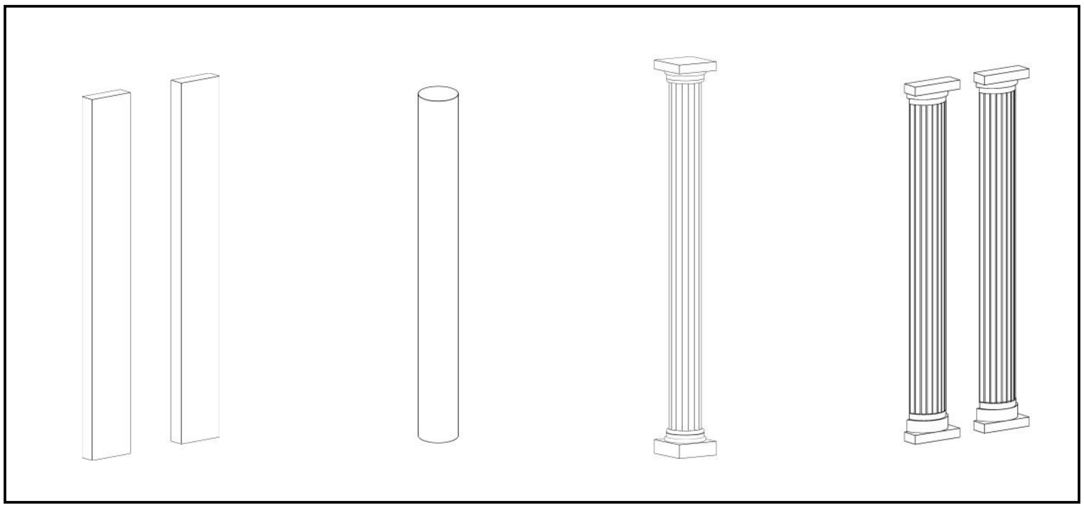


Figure 48: shows the two LoDs of columns.

The creation of the roof in the first floor was the third component to build. According to Dr. Francesca Diosono, the Entablature structure under the roof had three elements, the architrave, the frieze and the cornice figure 36. The roof angle and the height of the roof could be measured by using the CAD file figure 49.

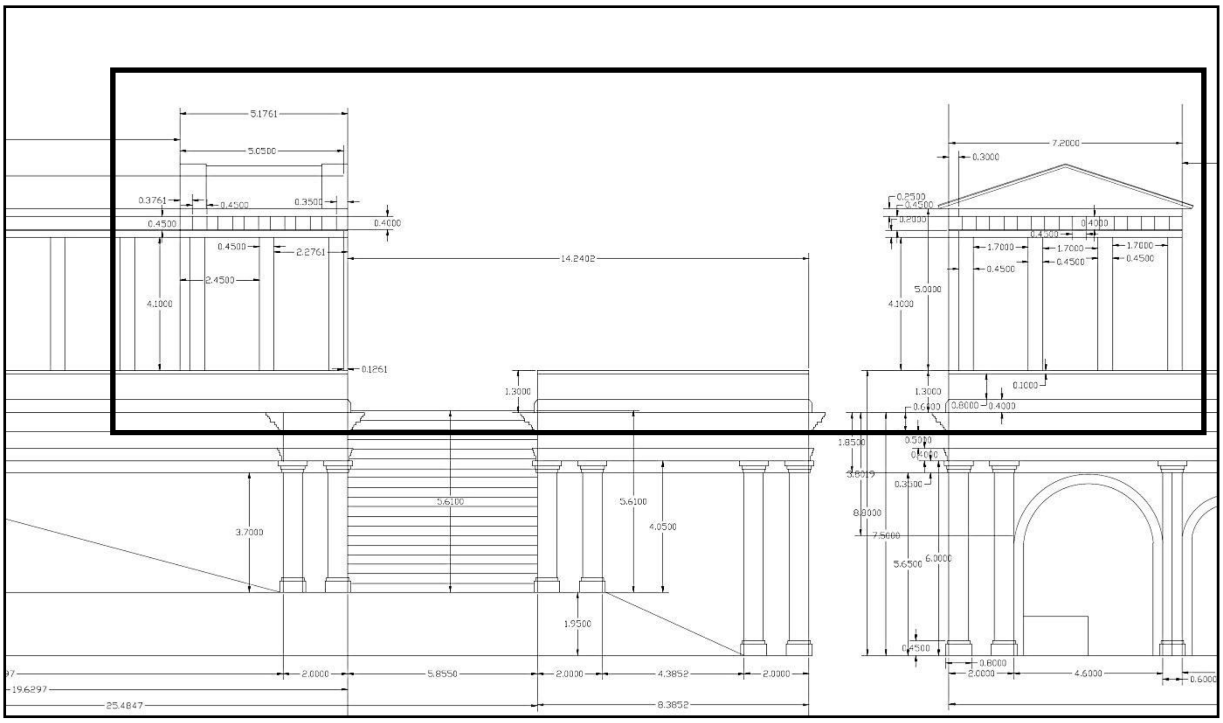


Figure 49: shows the roof structure and measurements.

The creation of the roof using Rivet was done in three steps in order to create all the necessary elements. The first step was to model the primary structure around the roof figure. The second step was to create the structure under the roof, and the third step was to build the roof figure 50. The next step was to combine all parts in CE.

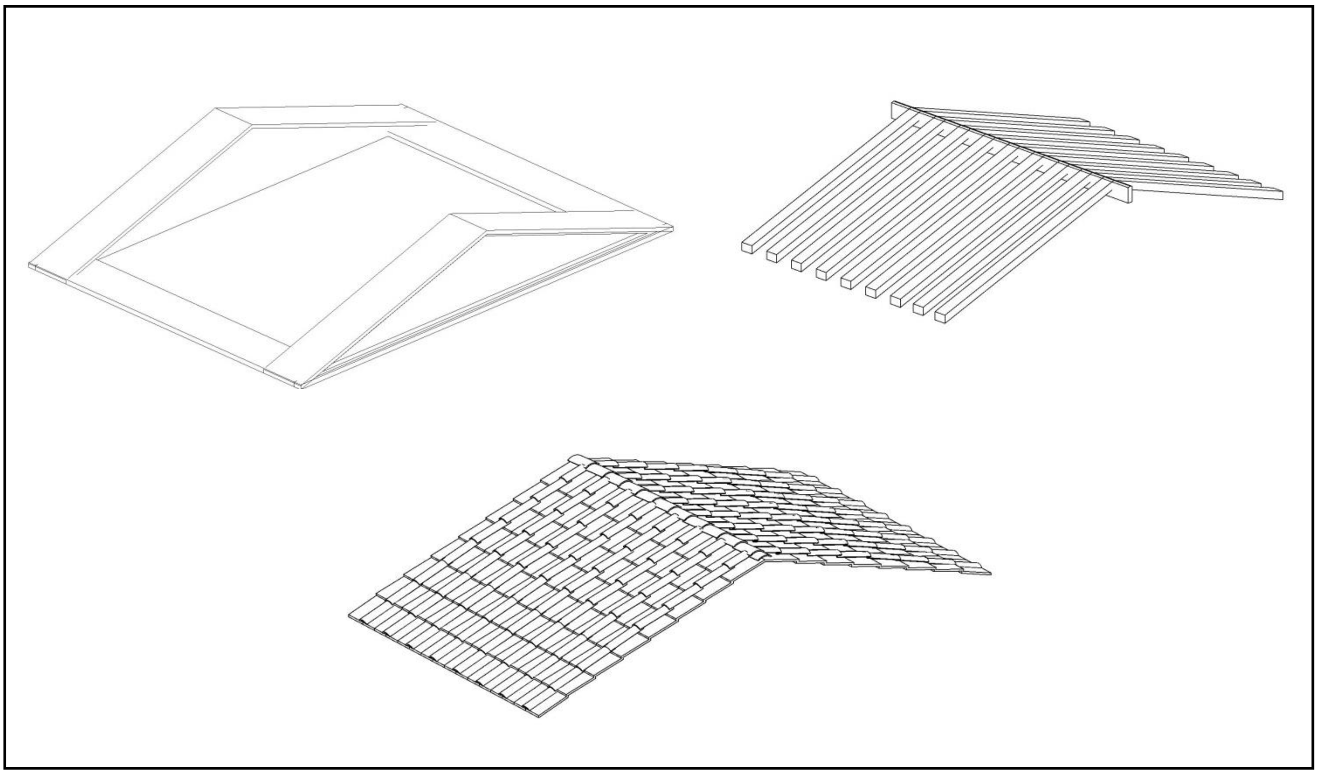


Figure 50: shows the creation steps of the roof elements.

As mentioned before, the frieze is a part of the Entablature, which had the decorations. The decorations consisted of three images that had to be fixed figure 35. The editing of the images required using Adobe Photoshop CC software which then allowed reconstructing the images figure 51.

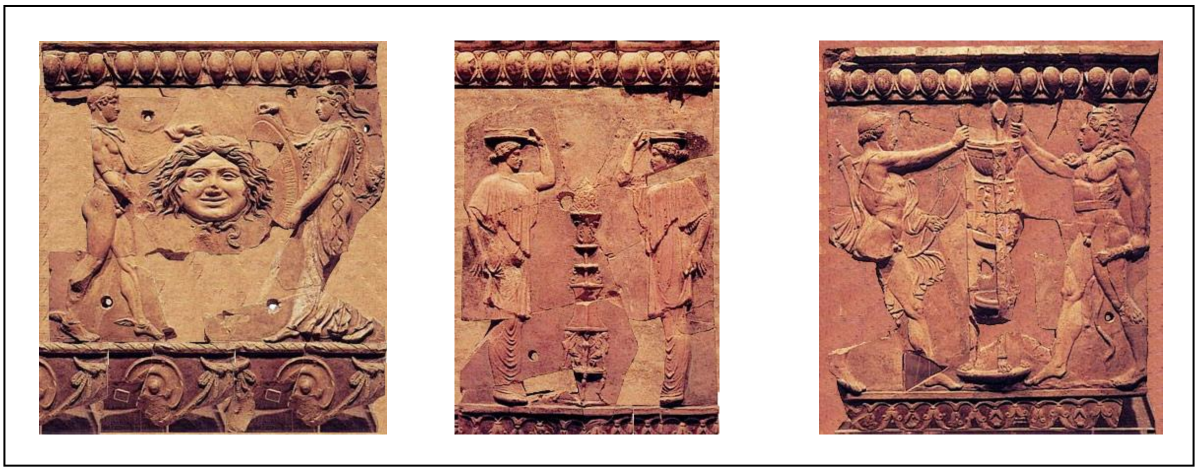


Figure 51: shows the three decoration images after editing.

The back wall of Nymphaeum has a strong curve which was created using Rivet in order to avoid the limitation CE has with complex shapes. According to Dr. Francesca Diosono, the two hypotheses share the same base but there is a difference in the structure on the first floor. The first hypothesis had two LoDs figure 52. The first LoD represented the lower level and the second represented the higher level. The second hypothesis had one LoD in the reconstruction figure 53.

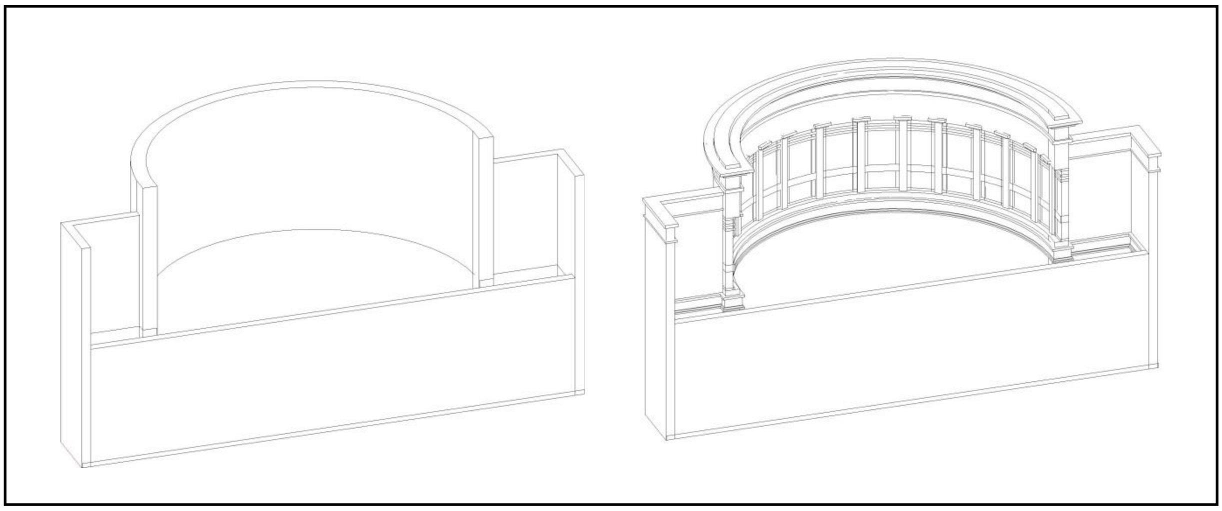


Figure 52: shows the LoDs of first hypothesis. In the left, the low LoD and in the right, the high LoD.

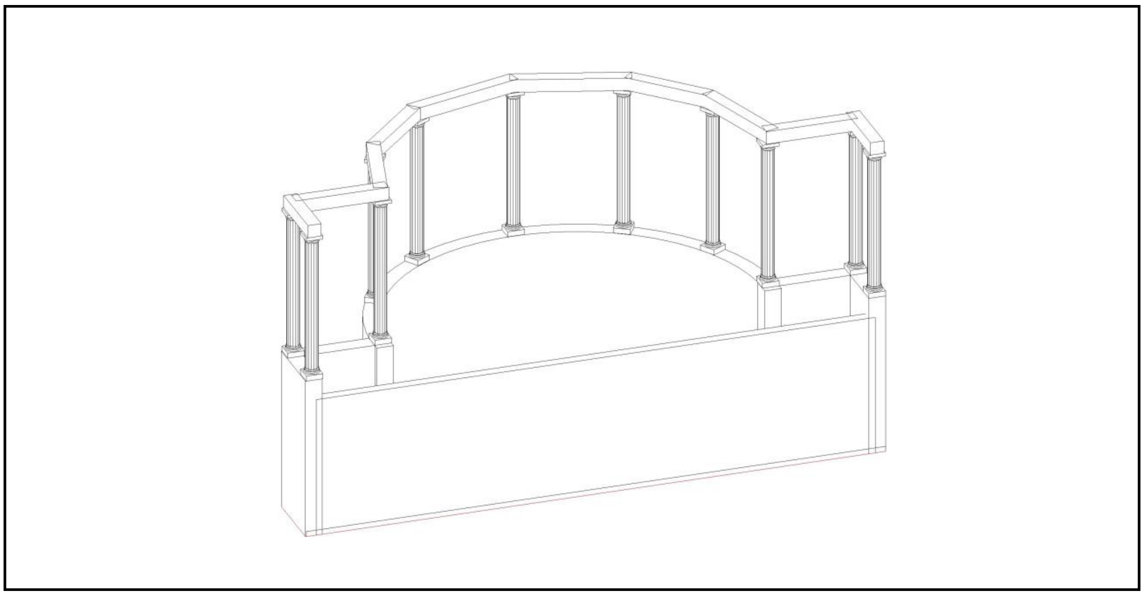


Figure 53: shows the LoD of second hypothesis.

Based on Dr. Francesca Diosono’s opinion in THE SHRINE OF DIANA AT NEMI: RECENT ACQUISITIONS OF NEW EXCAVATIONS 2012 the fountains in the ground floor rooms had a unique shape. The Reconstruction of the fountains included two shapes. The wall with special arches was created first to fill the area of the room figure 37. Next the water shape was created which had to be located inside the tub of the wall. The size of the base was 2.9 m in width and 7.5 in length with a height of 1.2 m. By transferring the measurement to Rivet the final form was created and implemented in the rules file figure 54.

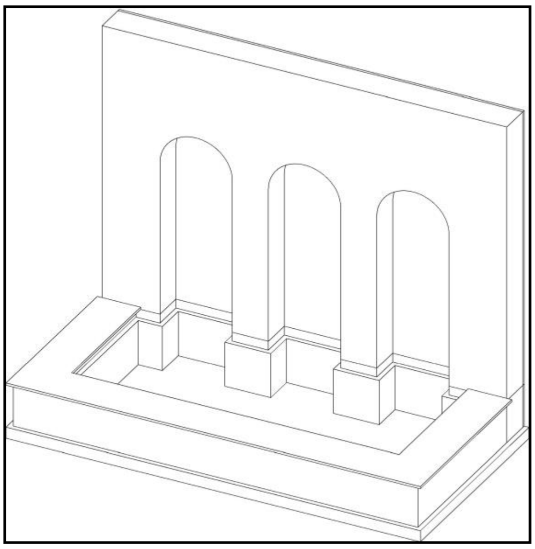


Figure 54: shows the 3D model of the fountains in the ground floor rooms.

The figure 38 shows the vaulted ceiling of the ground floor rooms. The reconstruction of the ceiling started in Rivet by creating a curve. Next, some decorations were added figure 55. The angle and size of the vaulted ceiling is created so that it can be adjusted in CE according to the size of each room.

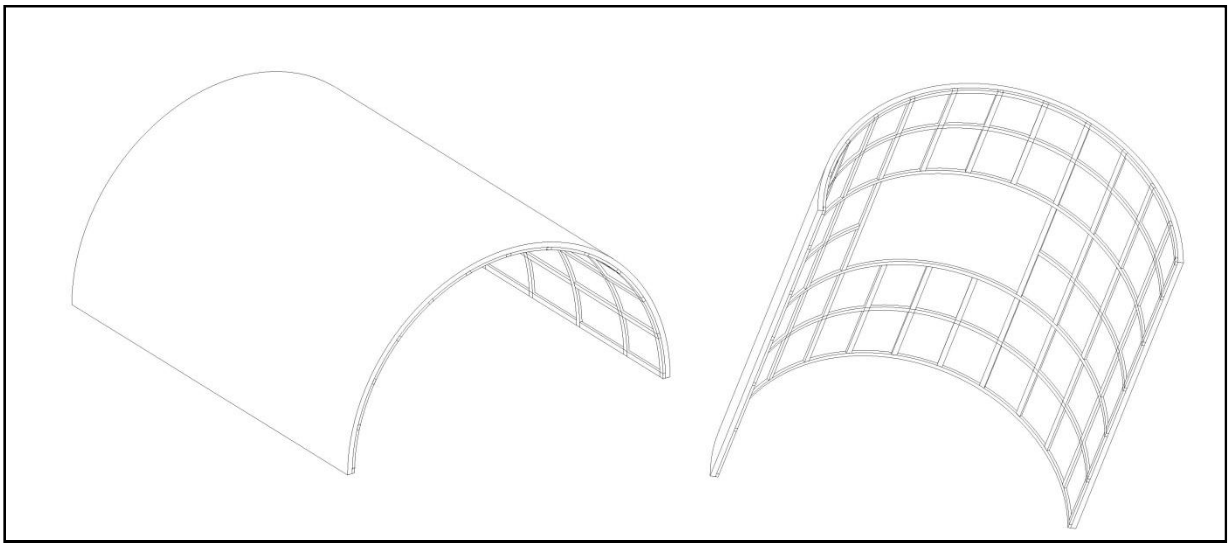


Figure 55: shows the 3D model of vaulted ceiling.

The string course which was located on the ground floor facade was built using two different LoDs. The reconstruction of the string course was created in Rivet according to the information from the archeologist team. The data was then converted to the measurement in the first in order to allow modelling of the 3D shapes figure 56.

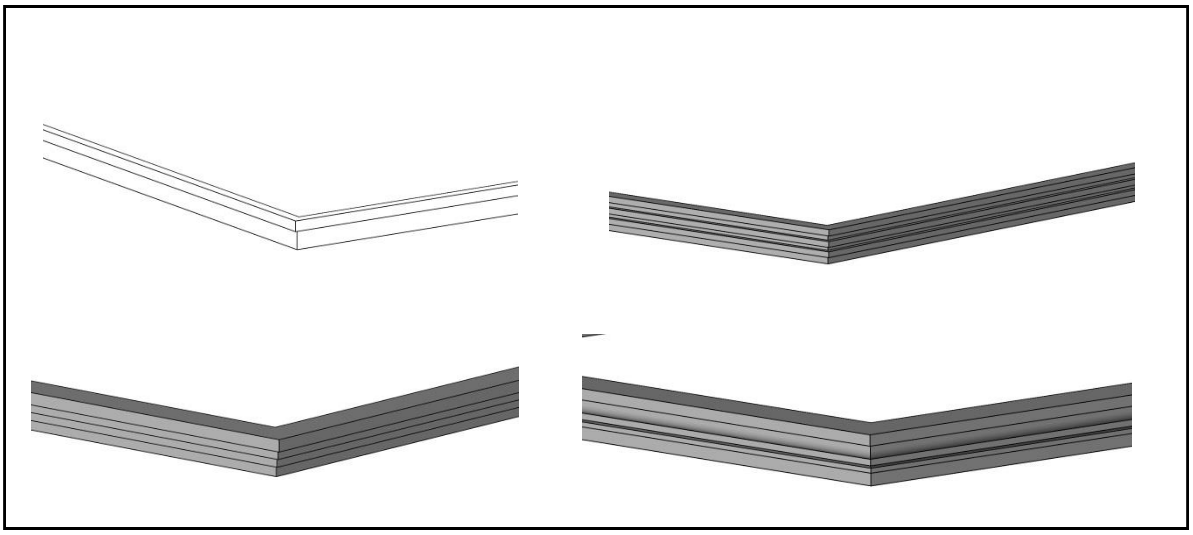


Figure 56: shows the string course that. In the first row the low LoD an in the second row the high LoD.

3.5 *Encoding the CGA Rules File*

The fourth step was to express all the data into a CGA rule which was applied in CE. The footprint was the basic element used to start the rule file. The work started by defending the LoDs that were implemented in the reconstruction of Nymphaeum figure 57. The @Group grouped the attributes in the inspector. @Order sets the category order for an attribute in the inspector. @Range sets the numeric values of an attribute and @Description appends a description to an attribute. These four functions allowed the determination of the four LoDs and the second hypothesis. The LoDs had the range from 0 to 5 with the names Low, Medium, High, Very High and Second Hypothesis.

```
@Group("LoD options",0)  @Order(4)
@Range("Second_Hypothesis","Very_High","High","Medium","Low") @Description("choose level
of detail")
attr LoD                                    = "Medium"
const LoD0                          = LoD == "Low"
const LoD1                          = LoD == "Medium"
const LoD2                          = LoD == "High"
const LoD3                          = LoD == "Very_High"
const LoD4                          = LoD == "Second_Hypothesis"
```

Figure 57: Defining the LoDs in CGA File.

After defining the four LoDs, the attribute parameters and the texture links were written for future use of the code figure 58. The textures were stored in the assets folder. The values of the attributes were then determined according to the measurements in the CAD data.

```
attr height         = 13.5
attr groundfloor_height = 7.5
attr firstfloor_height  =  6
attr door1_width = 4.6
attr door2_width = 3
attr door3_width = 3.4
attr wallthick = .5
attr wallthick1= .7
attr Base_columnthick = .8
attr No_of_Steps = 19
attr column_thick= .45
attr step_height = .2921
attr white = "#ffffff"
```

```
water_material    = "assets/water.jpg"
Picture_1      = "assets/Final_1.jpg"
Picture_2      = "assets/Final_2.jpg"
Picture_3      = "assets/Final_3.jpg"
roof_text      = "assets/roof.jpg"
ground_floor_rooms = "assets/all_lod.jpg"
ground_floor_rooms1 = "assets/re_mosaico4.jpg"
ground_floor_doorstep = "assets/stone.jpg"
Wood = "assets/Wood.jpg"
exterior_color = "assets/White.jpg"
```

Figure 58: Attributes parameters and Textures.

The Lot rule was attached to the footprint which allowed the reconstruction of extruding mass. The height of the mass was taken from the parameters and the word Building was used as a name of the mass. The Split function was used to split the mass into six facades, Front, Left, Right, Back, Top, and Bottom façade. Then each of the four facades had to split again into Ground floor and first floor figure 59. The use of split(y) means that the split function used the y directions according to scope shown in figure 60.

```
Lot -->
    extrude(height) Building

    Building-->
       comp(f){ front : RightFacade   | right :BackFacade   | left :
FrontFacade
    | back :LeftFacade   | top : roof | bottom : Bottom   }
FrontFacade -->
    split(y){ groundfloor_height : GroundfloorFront | firstfloor_height :
Firstfloorfront }

RightFacade-->
 split(y){ groundfloor_height : Groundfloorright | firstfloor_height :
Firstfloorright }

LeftFacade-->
 split(y){ groundfloor_height : Groundfloorleft | firstfloor_height :
Firstfloorleft }

BackFacade-->
    split(y){ groundfloor_height : Groundfloorback | firstfloor_height :
Firstfloorback }
```

Figure 59: shows the extruding of the footprint and the splitting of the façade in CGA.

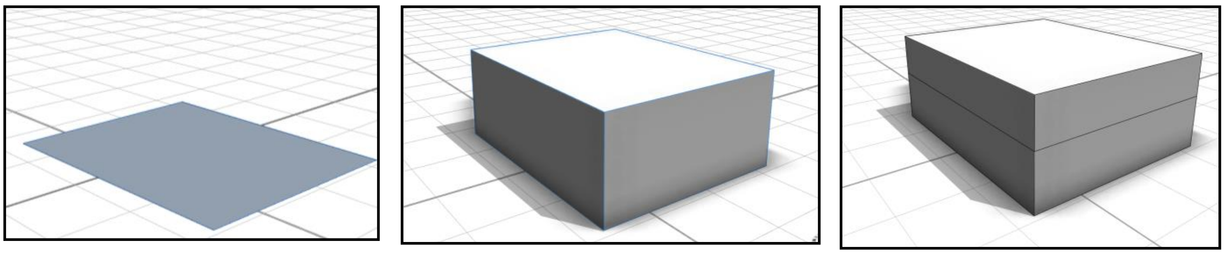


Figure 60: shows in the left, the footprint before the extrude function and in the right, the splitting of the mass.

The next step was defining the rule for the ground floor starting with the front façade. The plan of the ground floor shows a back wall for the ground rooms. By taking the measurement from the CAD file, the front façade wall was copied and moved to the back. The tag [ ] was then used to copy the wall twice. The front wall was split into five parts in the y direction. This rule was repeated to support three LoDs and the low level was kept without any modifications. The code inside the tag [t(0, 0,8.4) Groundfloor_front_back1] indicates that a new layer should be created and the contents should be moved 8.4 m to the back shown in figure 61.

```
GroundfloorFront-->
case LoD4 :
  [split (y){ 5.65 : GroundWall | .35 : Cornisha1 | .4 : Cornisha2 | .5 :
Cornisha3 |.6 : Cornisha4 }]

[t(0, 0,-8.4)Groundfloor_front_back1 ]
```

Figure 61: shows the splitting of the front façade of the ground floor.

Each of the five parts was assigned to a particular measurement. The next step was to define each part and continue splitting the parts until it covered the whole facade figure 62.

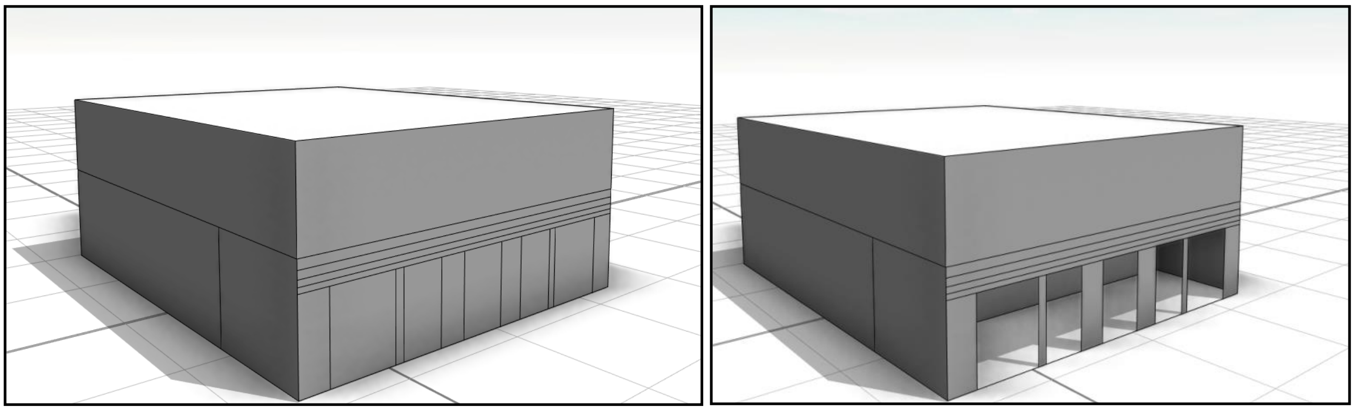


Figure 62: shows the splitting of the ground floor.

In addition, the arches had to be joined using the model by use of the Insert function (I). This functions works by linking the .obj file to the rule. The corresponding arch was then inserted to the model respective of their LoDs figure 63. To ensure that the .obj file would fill the right place without any missing parts the set function was used. Also, the Rotate and Scale functions were utilized to fix the position of each arch shape figure 64.

```
arch_5-->
        case LoD4 :
        set(trim.vertical, false)
        i("data/ground_floorfacadeArch_1_LoD_1.obj")
        r(-90,180,0)
        s(4.55,.5,5.65)
        color(white)
```

Figure 63: shows the CGA code for inserting the arches.

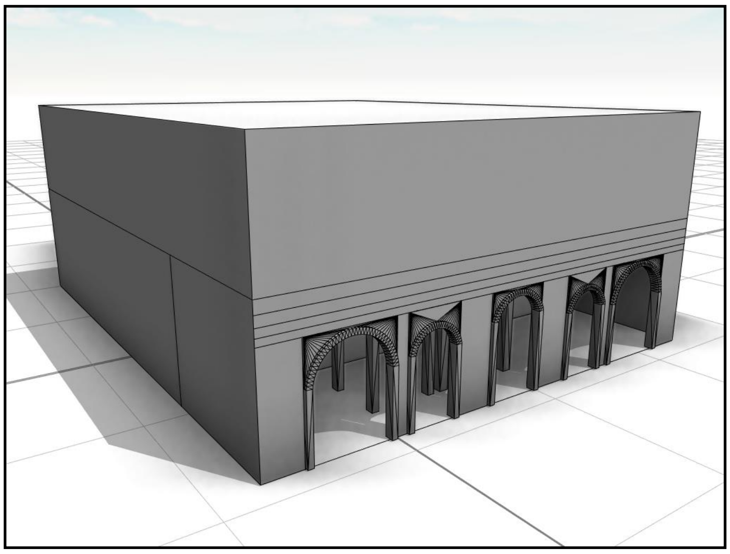


Figure 64: shows the front façade of the ground floor with arches

.

By using the same method, the columns of the ground floor were implemented figure (). To create the wall behind the column, each wall was copied into another layer with a different name to allow insertion of the column figure 65. As shown in the figure 66, each LoD displayed different information.

```
WallC6-->
      case LoD4 :
            [t(2,0,0)WallC_6]
            [Wall_f_g]
      case LoD3 :
            [t(2,0,0)WallC_6]
            [Wall_f_g]
      case LoD2 :
            [t(2,0,0)WallC_6]
            [Wall_f_g]
      case LoD1 :
            donothing
      else :
            donothing

 WallC_6-->
      case LoD4 :
      i("data/colume_ground_wallc_Edit1_LoD_1.obj")
      t(-2.05,0,.3)
      r(-90,0,0)
      s(2.1,.3,6)
      color(white)
      case LoD3 :
      i("data/colume_ground_wallc_Edit1_LoD_1.obj")
      t(-2.05,0,.3)
      r(-90,0,0)
      s(2.1,.3,6)
      color(white)
      case LoD2 :
      i("data/colume_ground_wallc_1.obj")
      r(-90,180,0)
      s(2.1,.3,6)
      color(white)
      else :     donothing
```

Figure 65: shows the inserting method for columns by coping the walls into new layers.

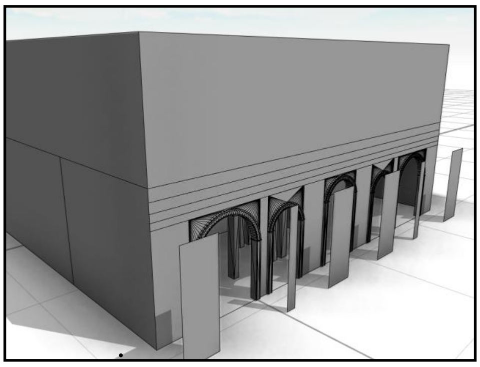
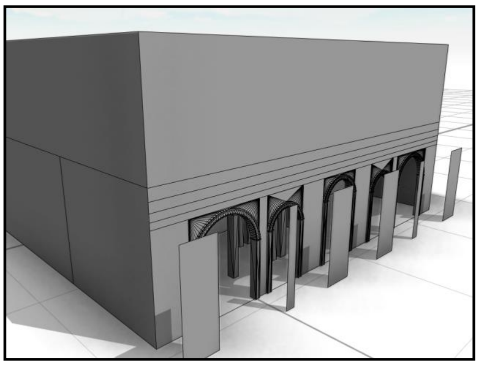



Figure 66: shows columns of the ground floor front facade.

The String Course of the front facade used the same technique as the columns. There were two different designs in the High and very high LoDs figure 67.

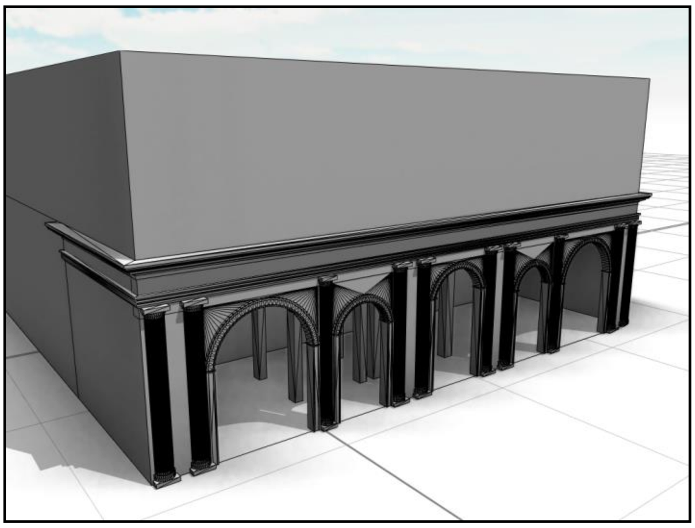


Figure 67: shows the front façade of the ground floor with String Course.

The right and left facades of ground floors were created using the same method as above. The stairs on the left façade were created by first splitting the bottom floor into smaller parts and then extruding each part figure 68. The position of each step had to be modified as in figure 69.

```
Stairs_Base-->
case LoD4 :
split (y) { .5421 : Part1 | .5421 : Part2 |.5421 : Part3 |.5421 : Part4
|.5421 : Part5 |.5421 : Part6 |.5421 : Part7 |.5421 : Part8 |.5421 : Part9
|.5421 : Part10 |.5421 : Part11 |.5421 : Part12 |.5421 : Part13 |.5421:
Part14 |.5421 : Part15 |.5421 : Part16 |.5421 : Part17 |.5421 : Part18 |.5421
: Part19 }
Part1-->
      case LoD4 :
      t(0,0,-5.61)
      extrude (step_height)
      color(white)
```

Figure 68: shows the splitting of the bottom floor to create the stairs steps and the step rule to modify the position

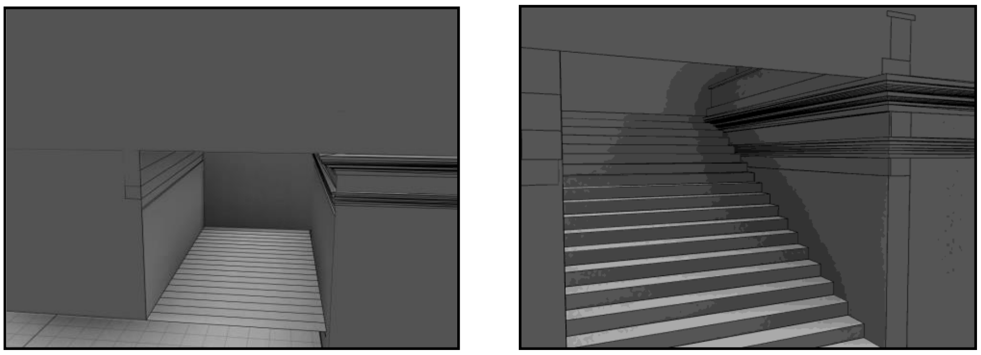


Figure 69: shows the step creation of stairs.

The implementation of the first floor followed the same method. The roof had three parts that were joined together figure 70. The medium level had a flat roof and the rest of LoDs had the three elements of the roof figure 71. RealRoof layer was then reproduced into roof_part2, roof_part3, and roof_part4 to allow creation of the roof.

```
Top_Roof1-->
split (y) {7.2 : RealRoof | ~13.3 : Deletepart | 7.2 : RealRoof }

RealRoof-->
case LoD4 :
[NIL]
[t(0,0,0)roof_part2]

roof_part2-->
[i("data/wood.obj")
s(7.4,4.25,.80)
t(4.65,-.15,0)
r(0,0,90)Wood_Ma]
[t(0,0,0)roof_part3]

roof_part3-->
[i("data/mosic.obj")
s(7.6,4.25,1.55)
t(4.65,-.25,0)
r(0,0,90)Mosic_Ma]
[t(0,0,0)roof_part4]

roof_part4-->
i("data/frame.obj")
s(7.6,5.3,1.6)
t(5.17,-.25,0)
r(0,0,90)
```

Figure 70: shows the creation of the roof in first floor.

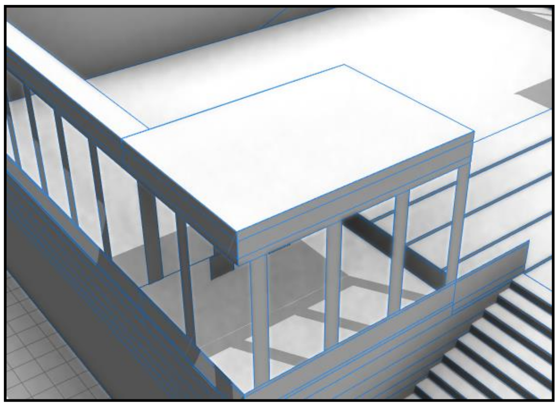
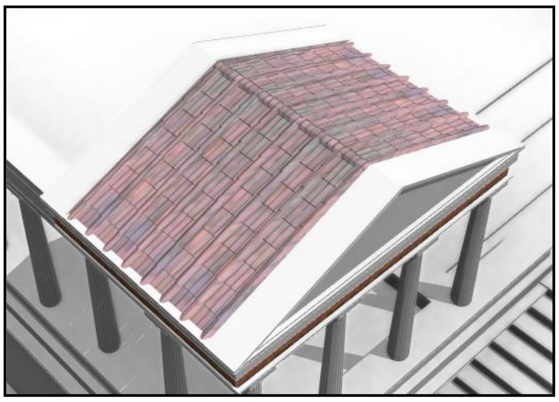



Figure 71: shows the 3D model of the roof.

The back wall of the building was created in Rivet by changing the size of the back wall. The .obj file of the back wall was then inserted and contained two parts: the wall and the water structure figure 72. For the texture, the Projection had to be set up in order to fix the scale of the image that was used according to the scope figure 73.

```
wall_Back-->
      i("data/LoD_5.obj")
      s(28,12.4,14.4777)
      r(-90,180,0)
      t(-27.8,-.2,-6.55)
       color(white)
Back_wall_block_water-->
      i("data/Family1_LOD_1_water.obj")
      s(27.5,10.8,5)
      r(-90,180,0)
      t(-27.5,.4,-4.55)
      setupProjection(0, scope.xy, scope.sx, scope.sy)
      texture(water_material)
      projectUV(0)
```

Figure 72: shows the CGA code of inserting the back wall and the water structure.

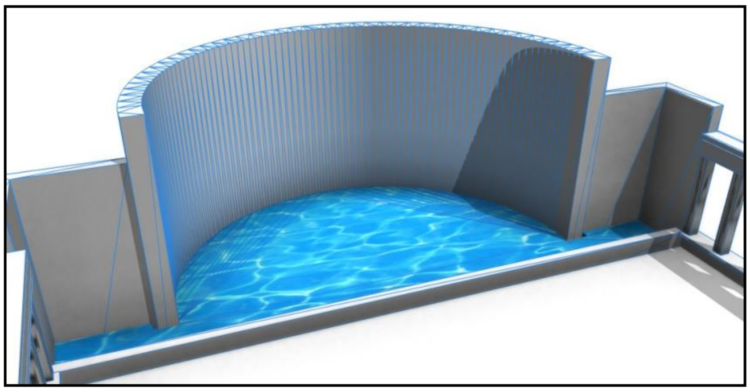


Figure 73: shows the back wall and the water structure.

The rest of the Implementation followed similar or identical methods. Defining the rule for each element under a unique name allowed us to increase the number of rules in the file. The rules file created a structure tree that is shown below figure 74.

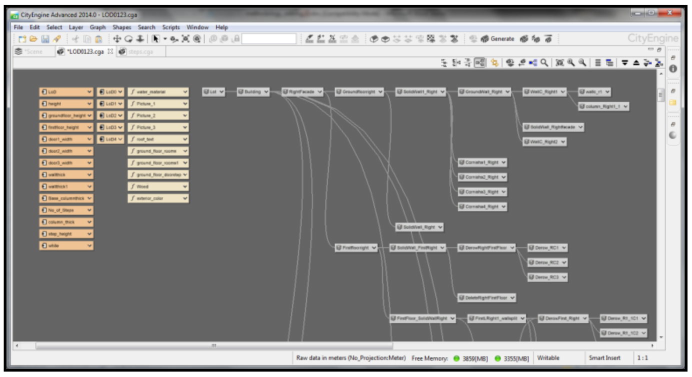
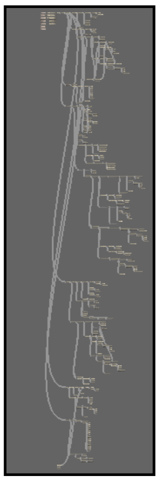



Figure 74: shows the structure tree in CGA file.

Generating the Model

The last step in the workflow was generating the model. In this step, the CGA rules file contained all the functions and parameters used to generate the model. All the data that was extracted from the data analysis step was used in the final implementation and impacted the final result. Also, the ideas and concepts of the archeological team were applied in the implementation. After generating the model, the terrain was created by using the height map shown in figure 75.

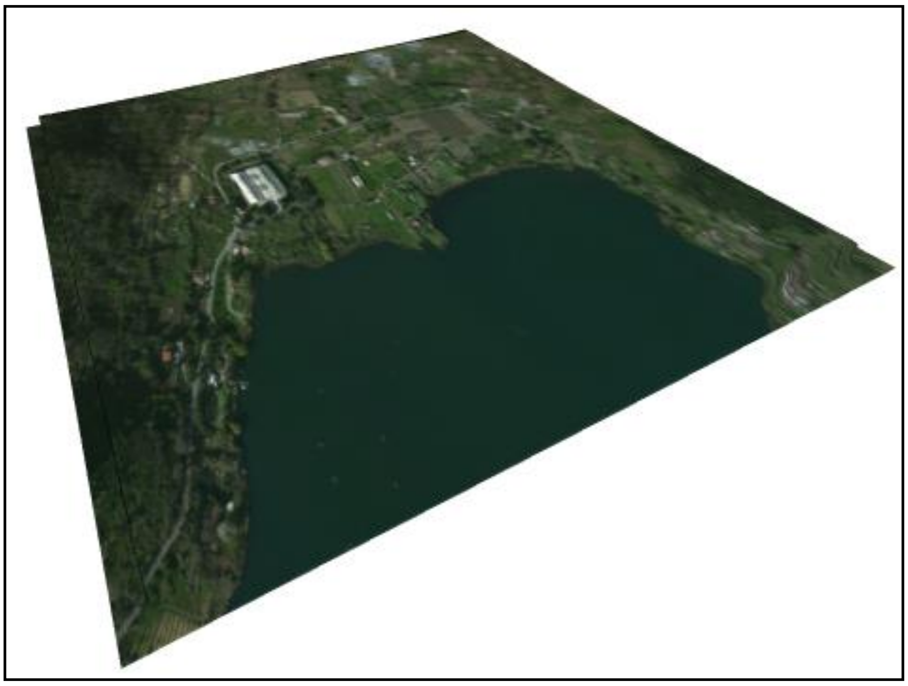


Figure 75: shows the terrain model.

The next step was to transfer the Nymphaeum into the Web Scene so that viewers could navigate between the different LoDs and hypotheses. As a summary of this chapter, the method that was used in the implementation started with the creation of the workflow. The Data Analysis was then conducted to extract information from the basic data that was collected by archaeology team. The creation of the architectural elements was the first stage of converting the 2D data into a 3D model. All the elements were imported to the rules file in the encoding step. The center of the implementation was the encoding of CGA rules file step which was then displayed as the result of the implementation. The last step was generating the model by combining all the rules together to generate four LoDs and two hypotheses.

4 Results

4.1 Creation Four Levels of Detail

As a result of using Procedural Modeling for modeling the Roman Room reconstruction, four LoDs were created. As mentioned in the description of CE, each of the LoDs represents a prespecified definition. This technique allows the user to view all of the LoDs and switch between them seamlessly.

4.1.1 LoD 0 – Low Level:

The first LoD represents the lower level of detail. This LoD shows the creation of the mass of the model. The viewers can navigate around the model, and the mass does not have any textures figure 76.

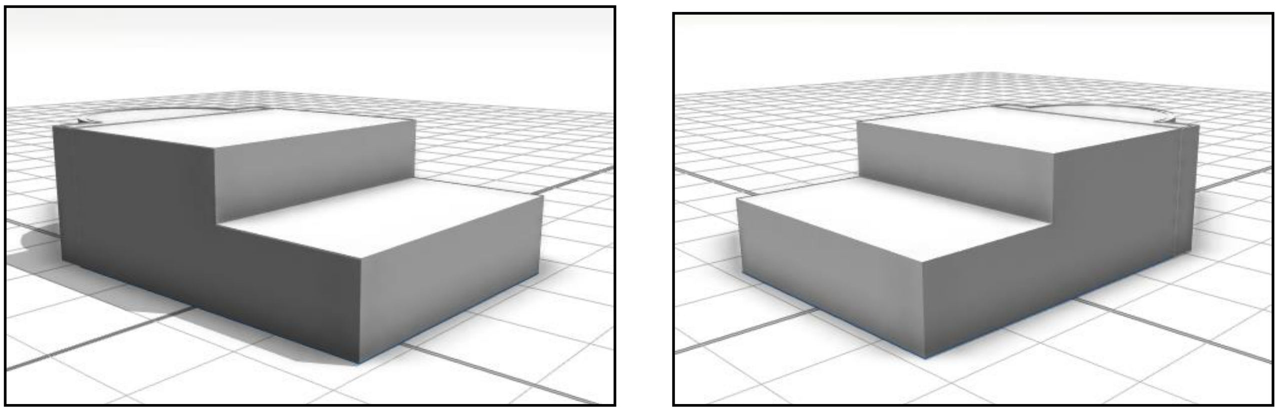


Figure 76: shows LoD 0 – Low Level.

4.1.2 LoD 1 – Medium Level:

The second LoD shows the simpler model which contains the splitting mass of the model. In this level, the viewer can understand the basic structure of the model without any complex shapes. Also, no roofs or interior elements were created figure 77.

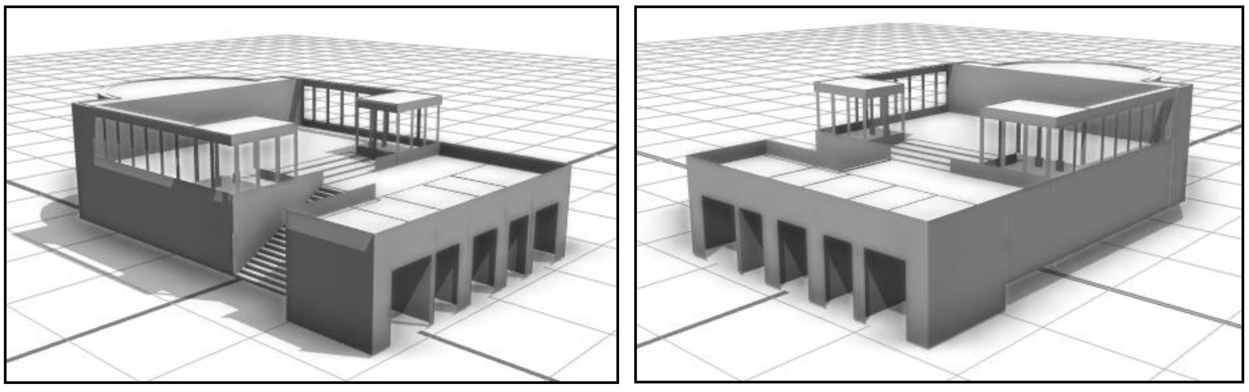


Figure 77: shows LoD 1- Medium Level.

4.1.3 LoD 2- High Level:

The third LoD utilizes a Reasonable polygon number. In this level the model used the essential architecture elements. The arches, columns, string course, roofs and the back wall were implemented in the primary shape level. This allows the viewer to fully comprehend the positions and sizes of all elements. The wall thickness was created in this level. Also, the exterior texture was assigned to the model which was a white color and the water texture was applied to the water form in the back wall. There was no interior structure in this LoD figure 78.

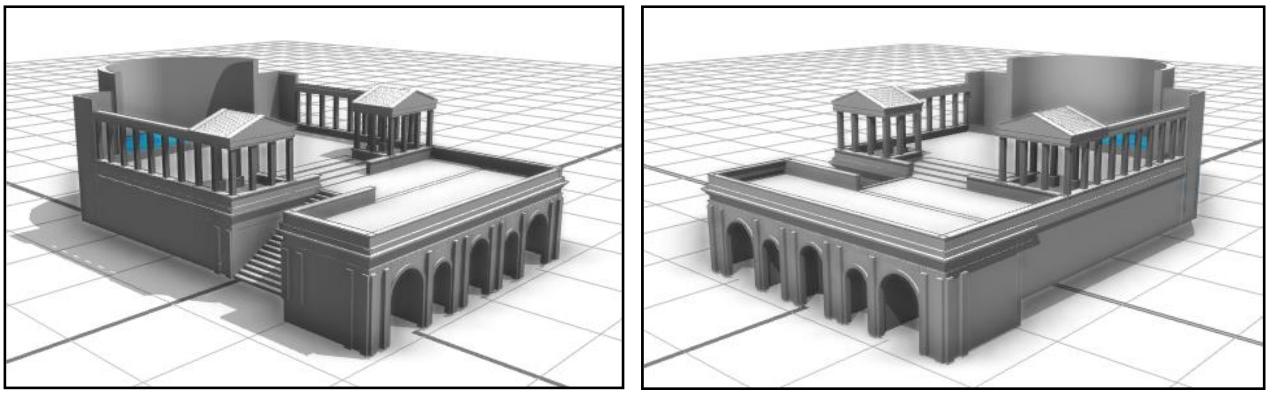
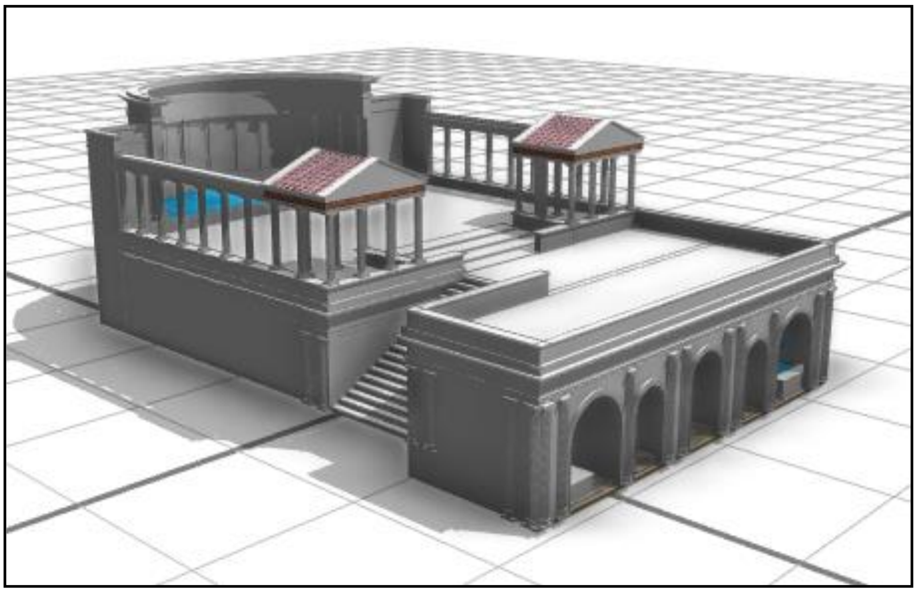


Figure 78: shows LoD 2 – High Level.

4.1.4 LoD 3- Very High Level – First Hypothesis:

The fourth LoD represented a very high Level of detail. In this level, the model was created with full detail elements. The first concept of the Nymphaeum reconstruction was implemented in this level to give enough knowledge to the viewer. The number of polygons was increased in this level in order to support the central idea of LoDs. The vaulted ceiling was designed for the rooms on the ground floor and the fountains were implemented in corresponding rooms shown in figure 79.



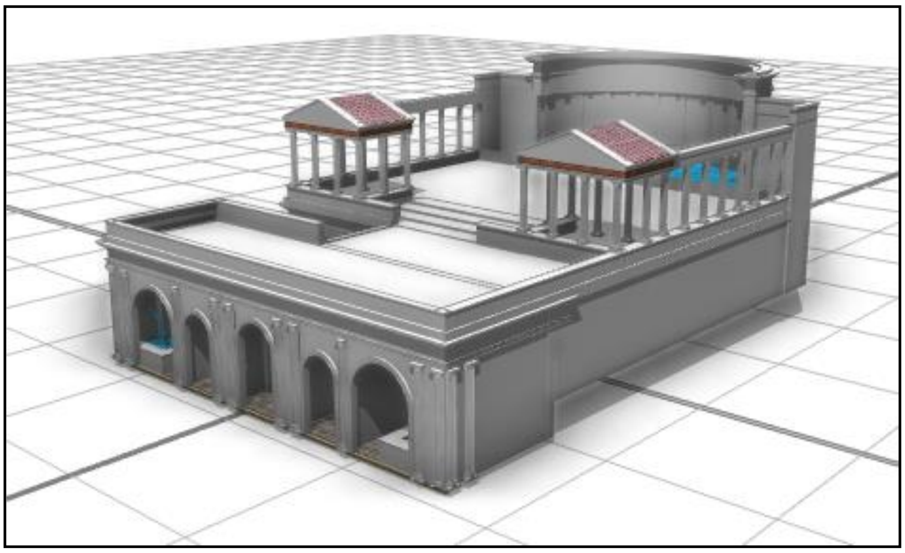
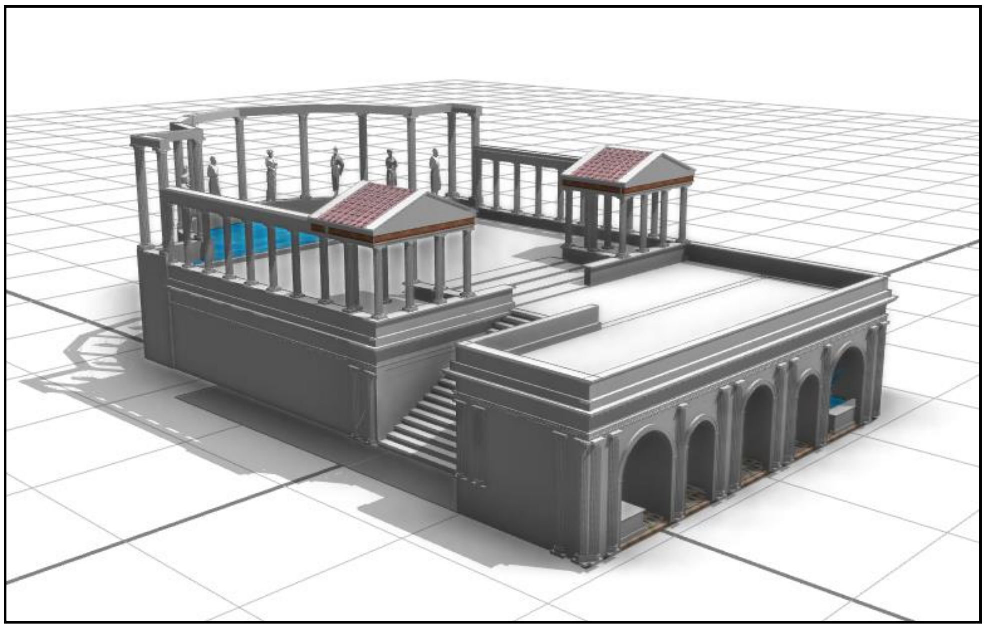


Figure 79: shows the LoD 3- very High Level and first hypothesis.

4.1.5 LOD 4 – The Second Hypothesis:

The second hypothesis had the same elements as the first hypothesis. The only difference was that the back wall of the first floor contains a horizontal structure in the curved wall with statues under the structure. The main idea of this LoD was to give a clear image of the difference between the first and second hypotheses figure 80.



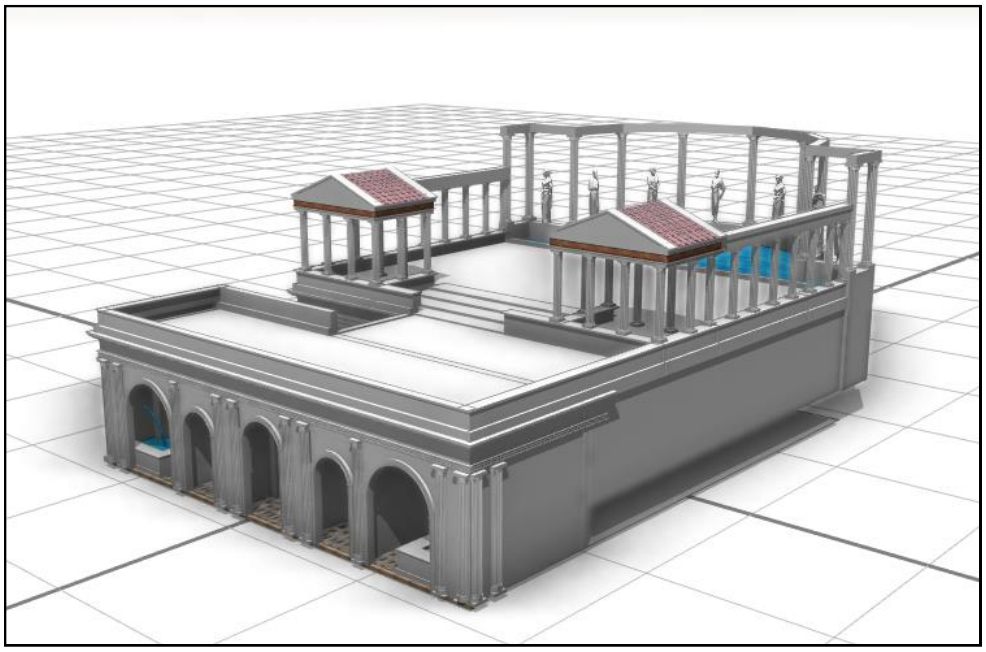


Figure 80: shows the Second hypothesis.

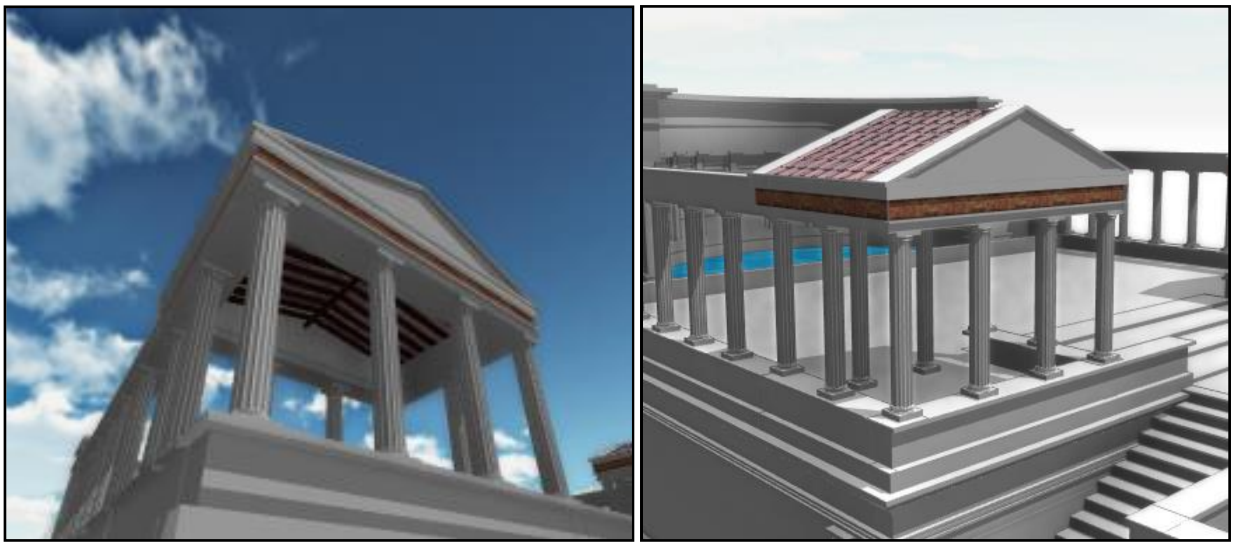


Figure 81: shows the roof structure of the first and second hypothesis.

In addition, the roof structure is presented in figure 81. Also the interior design for the ground floor which contains the ceiling, fountains, and the floor texture is displayed in figure 82. The first floor area which contained the stairs and the two small roofs is shown in figure 83.

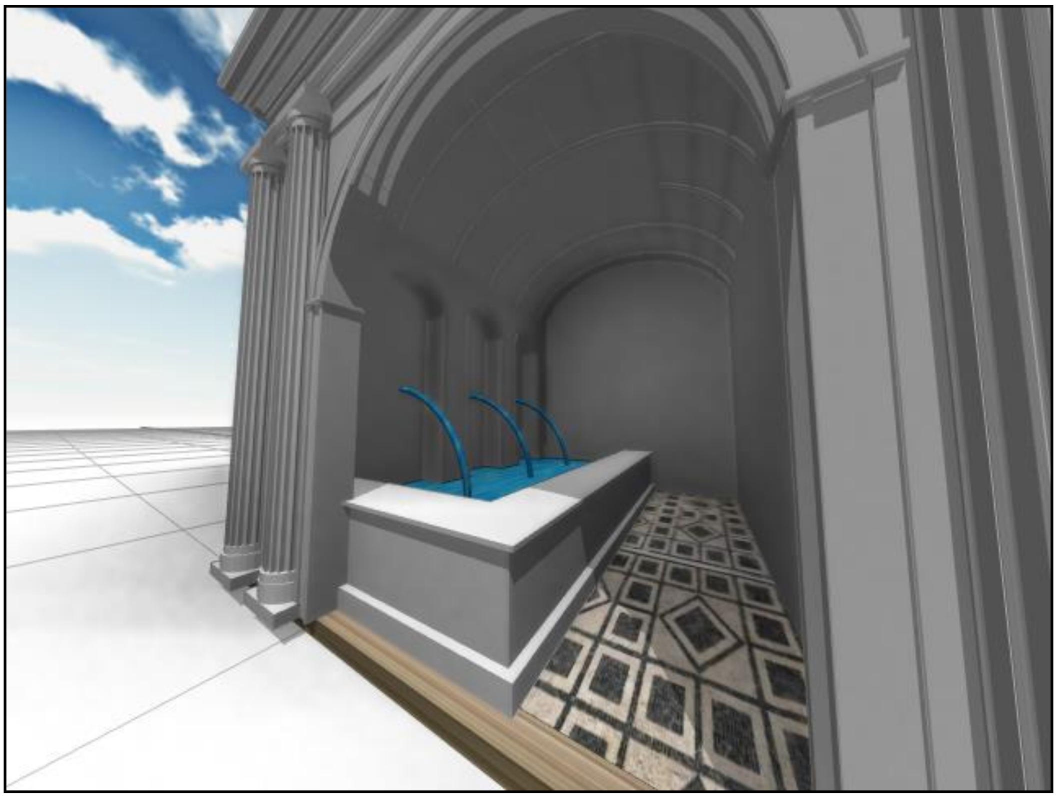


Figure 82: shows the interior design of the ground room.

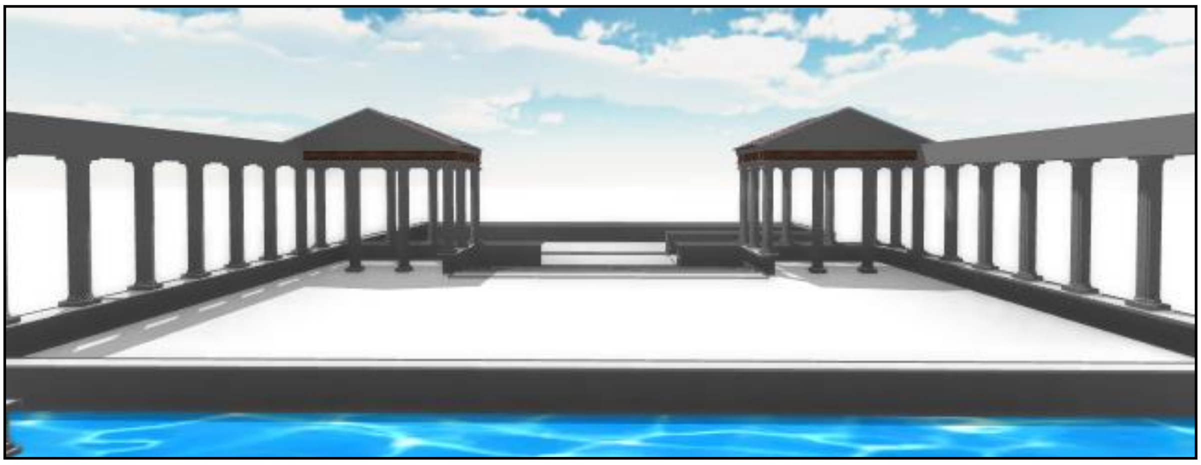


Figure 83: shows the first floor area.

4.2 The terrain creation:

The terrain creation was a result of using the GPS points that collected fron the site in Nemi. The 3D model of the terrain represents the topography of the land shown in figure 84. In all LoDs, the terrain can be used to give a better understanding of the 3D model shown in figure 85.

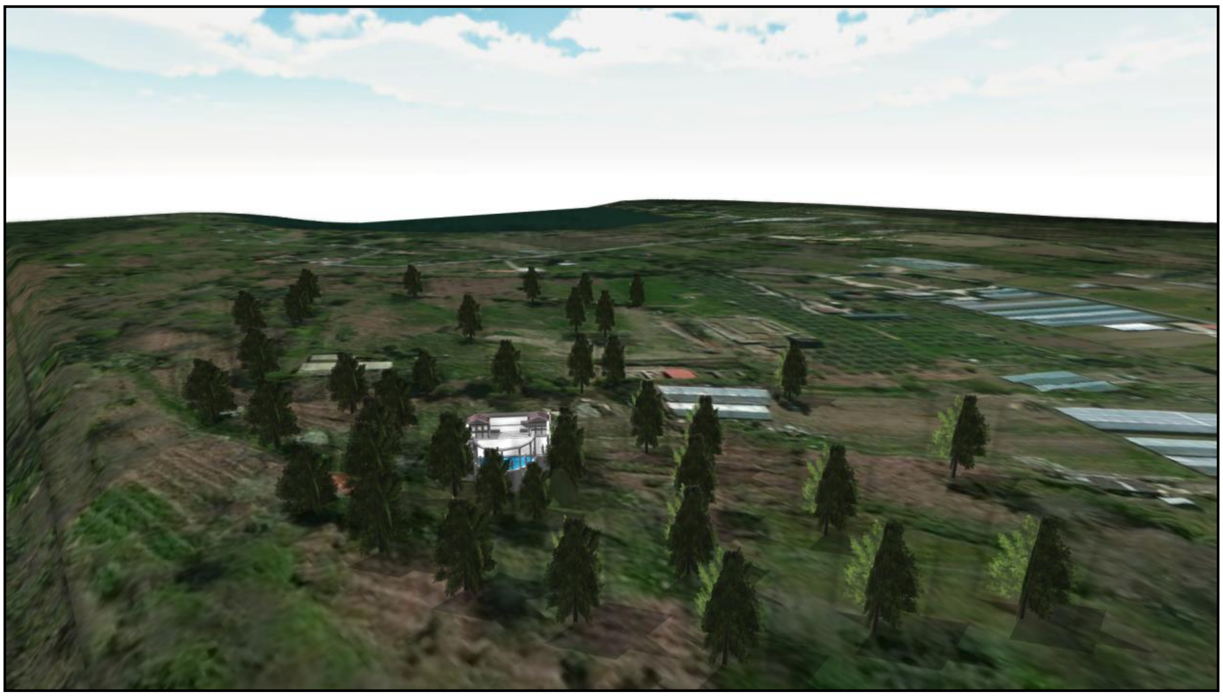


Figure 84: shows the topography of the land with the building model.

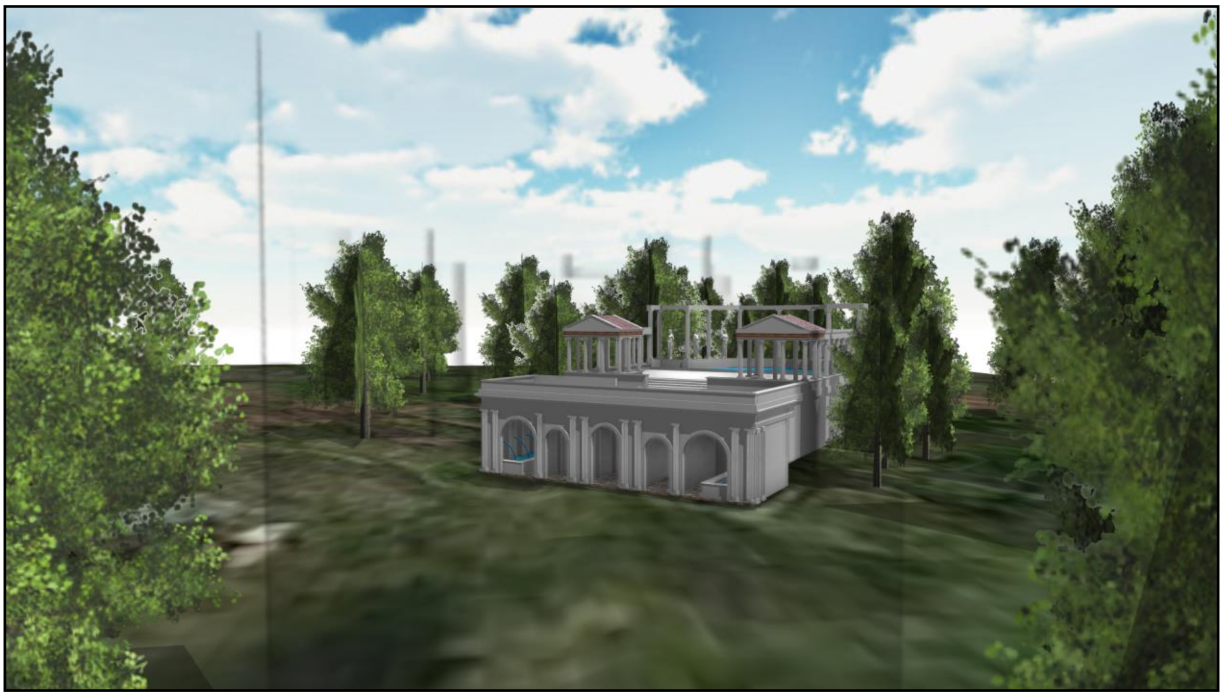


Figure 85: shows the Nymphaeum reconstruction with the terrain.

5 Discussion

The goal of this study was to use procedural modeling techniques to reconstruct the Roman Rooms. All projects presented in the related works section had used CE to generate their models. This allowed the creation of Roman reborn reconstructions and increased efficiency while reducing overall cost. The type of result that was obtained by using CGA alone however, does not match the goal of creating four LoDs. Because of that the idea of utilizing the CAD system in the study was implemented in order to support the final result. The first stage of the study was conducted to determine the functionality and limitations of only using CE for the generation of models. This resulted in a low level of detail without using any additional software. However, to create a high level of detail there was a need to include the CAD system. The primary reason of using CE in all of the related work projects was to avoid the limitations in the CAD software. The file size of CAD generated images is magnitudes larger compared solely using CE during the creation of similar projects.

The archeological data was the starting point of reconstructing the Roman Rooms. The archeologist team needed a lot of time to provide the data but this is typical for most archeological projects. In the remonstration of Xkipche Building, the data collection period took about 15 years in order to study all parts of the building. First, they created non-detailed 3D reconstruction models manually using CAD software. Next a second model was created by using CGA to efficiently construct a large number of high quality geometric models while keeping down costs compared to manual work. They could then test several hypotheses by editing individual parameters. The reconstruction of Nymphaeum utilized a similar method. The type of data that was collected from the archeologist team in Nemi was not useful for immediate work in CE. The limitation of the data was categorized into different types which helped solve the problem. Each type of data was uniquely prepared to allow the extracting of necessary information that was then later used in CE or CAD systems. The elevation of Nymphaeum in figure () was converted to an AutoCAD file in order to extract the measurement of the architecture elements. By using the measurements, all the architecture elements were created to manifest the archeologist's ideas and concepts.

One of the difficulties that we had in the project was accurately understanding and interpreting those ideas. This was very important because any misrepresentation could affect the entire implementation process. Direct communication with the archeology team helped save time because of discrepancies between data and concepts. In many cases the reconstruction of columns and roofs had to be changed more than once but this was a non-issue because by using the measurements from the AUTOCAD file, any element could be changed quickly. The time used to create each element was reduced by half simply because of the use of Rivet software. Another problem that we faced when generating the reconstruction was on how to convert the elements that were created in Rivet software to CE. The Rivet software uses a different measurement system to create 3D elements, which is not directly translatable to CE. This kind of problem had taken a lot of time to be fixed.

CE also could not create the complex and strong curve of the back wall of the Nymphaeum building but once again Rivet was able to do the job. Another problem that we faced during the implementation was on how to name each part of a building with a specific name. If the names were too similar then some conflicts arose during the creation of the rules. This was solved by splitting the footprint into two parts which the shape to be implemented in CE. Furthermore, defining the four LoDs in the rules file allowed the rest of work to be done in sequence, starting from the building's footprint up until the last elements all in high LoD. The generation of the model in the high LoDs took quite some time because of the huge number of polygons. There was a memory limitation problem in all of the related work projects which was solved by using LoD 3 to reduce the number of polygons while still maininting a relatively accurate model. The result of the Nymphaeum model reached the goal of the study which was to create a realistic model. The viewer can easily switch between the four LoDs in and navigate around the model. In the high and very high LoDs, CE has helped represent different elements in a very efficient way that covers all the archeologists' concepts. CE also allows the model to be edited if future data or interpretations are introduced into the study of the Nymphaeum.

Procedural Modeling improves the representation of archeological hypotheses. Instead of using the techniques that were used in the London Charter as mentioned in chapter two, the idea of providing full detail views of different hypotheses was used in this thesis which allows viewers to obtain a true representation of the site. Instead of using colors to represent the uncertain parts of a constructions two full detailed hypotheses of the Numphaum were created to allow better a understanding of different hypotheses. In the reconstruction of the Puuc Buildings in Xkipché as mentioned in chapter two, the same method was used to represent different hypotheses. The viewer can then understand that the archeologist team had different interpretations of the exterior of buildings. In brief, the method that was used in this project allowed us to convert the concepts of the archeologist team in Nemi without giving misleading the viewer.

6 Conclusion and Outlook

In this thesis The Procedural Modeling approach was used to reconstruct the Nymphaeum at Nemi, Italy. By utilizing the CAD system to generate certain textures, the level of detail was increased to cover the concept of LoDs in CE. The main goal of this thesis was to reconstruct a realistic and visually appealing model. The workflow of the work started by understanding the ideas behind each hypothesis which allows the transfer of 2D and text data into 3D modeling. The CAD system was then used to analysis that data and then extract specific measurements from it. By transferring the information into 3D modeling software, the architectural elements were created to implement in CE. The Procedural Modeling technique the allowed us to generate four LoDs and two hypotheses. Each of the LoDs had a unique representation of the data to give a better understanding of the archeological content. This allows the viewer to realize the differences between the two hypotheses and obtain the truth behind both concepts. This will make the work more accessible to a number of different people including seasoned academics and students in the early stages of their career while also preserving the full spectrum of different concepts. The result of the study can be used for future investigations on the site in Nemi. This work can serve as the base work for the creation of any other 3D models and will dramatically reduce the amount of work required. Most of the architectures elements could be edited and converted into any new concepts according to the archeologists' opinions.

As an outlook for further potential steps in enhancing the result of the reconstruction of the Nymphaeum in CE, the following can be addressed:

- Collecting more details about the building from the archeologist team concerning the façade and the interior design of the building.
- Updating the 3D model according the new excavations.
- Increasing the level of detail to even higher levels.
- Creating more hypotheses according to concepts proposed by archeologists.
- Converting the 3D model into CAD software in order to generate realistic results.

References

3DCityDB, (2011): http://opportunity.bv.tu-berlin.de/software/projects/3dcitydb (accessed 2014-08-10).

3DMLW (2011): http://www.3dmlw.com/, (accessed 2014-08-10).

Alessandro M., Nadalutti D., Chittaro L. (2007): Interactive Walkthrough of Large 3D Models of Buildings on Mobile Devices, in Web3D 2007: Perugia, Italy.

Alexander C., Ishikawa S., Silverstein M. (1977): A Pattern Language: Towns, Buildings, Construction. Oxford University Press, New York.

Araujo B., Iwasaki M., Pizzolato B., Boukerche A. (2008): Framework for 3D Web-Based Visualization of HLA-Compliant Simulations, Proc. 13th Int. Symp. on 3D Web Technology (Web3D 08), ACM, Los Angeles, California, 2008.

Aristoteles (2011) Available online at: http://www.ikg.uni- bonn.de/aristoteles/index.php/Aristoteles, (accessed 2014-08-10).

ATI, (2004): ATI Radeon X800 3Dc White Paper, 2004. http://www.ati.com/ products/radeonx800/3DcWhitePaper.pdf (accessed in 2014-08-11).

Autodesk, (2011): http://usa.autodesk.com/adsk/servlet/item?siteID=123112 &id= 13922635&linkID=13111311 (accessed 2014-08-11).

Beacham R., Niccolucci F., Denard H. (2006): An introduction to the London charter, in Papers from the Joint Event CIPA/VAST/EG/EuroMed.

Beacham R., Niccolucci F., Denard H. (2009): The London Charter, Version 2.1, February 2009, http://www .londoncharter.org/.

Bell G., Parisi A., Pesce M. (1995): The Virtual Reality Modeling Language- Version 1.0 Specification, Available online at: http://www.web3d.org/ technicalinfo/specifications /VRML1.0/index.html /, (accessed 2014-08-10).

Bentkowska-Kafel A., Denard H., Baker D. eds (2012): Paradata and Transparency in Virtual Heritage. Farnham: Ashgate

Berndt R., Fellner D.W., Havemann S. (2005): Generative 3D models: a key to more information within less bandwidth at higher quality, in Proceedings of the 10th International Conference on 3D Web Technology (Web3D '05), pp. 111–122, Bangor, UK.

Bilalis N.(2000, January) Computer Aided Design-CAD.Report produced for the EC funded INNOREGIO project . Retrieved from http://www.adi.pt/docs/innoregio_cad-en.pdf.

BIMServer (2011): Available online at: http://www.bimserver.org/, (2011-08-10) Bit Management Contact viewer (2011): http://www.bitmanagement.com/(accessed 2014-08-10).

BS Contact Geo 7.2 (2011): Available online at: http://www.bitmanagement.de/ products/interactive-3d-clients/bs-contact-geo, (accessed 2014-08-10).

Cabral M., Zuffo M., Ghirotti S., Belloc O., Nomura L., Nagamura M., Andrade F., Faria R., Ferraz L. (2007): An experience using X3D for Virtual Cultural Heritage, in Proc. of 12th International Conference on 3D Web Technology (Web3D 07), Perugia, Italy, pp161-164.

Calado A., Abreu P., Chumbinho R., Silva A., Sousa L. and EMEPC Group (2008): Integration of Hydrographic Data Products in a Global Web Based 2D and 3D GIS, Proceedings of the Canadian Hydrographic Conference and National Surveyors Conference 2008.

Calogero E. and Arnold D.(2011): Generating Alternative Proposals For the Louvre Using Procedural Modeling, Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci., XXXVIII-5/W16, 185-189, doi:10.5194/isprsarchives-XXXVIII-5-W16-185-2011.

Carlson W. (2003): A Critical History of Computer Graphics and Animation, A vailable online at: http://design.osu.edu/carlson/history/lessons.html, (accessed 2014-08-21).

Carver N. F. (1986): Silent Cities of Mexico and the Maya. Documan Press Ltd.

Chen A., Leptoukh G., Kempler S., Nadeau D., Zhang Z., (2008): Augmenting the research value of geospatial data using Google Earth. In: De Paor, D. (Ed.), Google Earth Science Journal of the Virtual Explorer, electronic edition, vol. 29.

Chen M. (1999): Generation of Three-Dimensional Geometry for Night Illumination and Urban Visualization. http://graphics.lcs.mit.edu/~maxchen/Boston.html.

Chiles J., and Delfiner P. (1999): Geostatistics: Modeling Spatial Uncertainty. Wiley, New York.

Ching F. D. K. (1996): A Visual Dictionary of Architecture. Wiley.

CityVu (2011): Available online at: http://cityvu.3dgis.it/lang/en/, (accessed 2014-08-10).

COLLADA (2011): https://collada.org/mediawiki/index.php/COLLADA_-_Digital_Asset_and_FX_Exchange_Schema.

Czerwinski A., Kolbe T., Plümer L., Stöcker-Meier E. (2006a): Interoperability and accuracy requirements for EU environmental noise mapping, in International Symposium InterCarto-InterGIS. 2006: Berlin.

Czerwinski A., Kolbe T., Plümer L., Stöcker-Meier E. (2006b): Spatial data infrastructure techniques for flexible noise mapping strategies, in 20th International Conference on Informatics for Environmental Protection EnviroInfo. 2006: Graz.

Czerwinski A., Sandmann S., Stöcker-Meier E., Plümer L. (2007):Sustainable SDI for EU noise mapping in NRW – best practice for INSPIRE, International Journal of Spatial Dana Infrastructures Research, No. 2, 90- 111, 2007.

Davis M., Sigal R., Weyuker E. J., Davis M. D. (1994): Computability, Complexity, and Languages : Fundamentals of Theoretical Computer Science. Academic Press.

Davis T. J., and Keller C. P. (1997): Modelling and visualizing multiple spatial uncertainties. *Computers & Geosciences* 23(4): 397-408.

Digital Nautical Chart, (2011): http://www.nga.mil/portal/site/dnc/, (accessed 2014-08-10).

DirectX, (2011): http://www.microsoft.com/games/en-US/aboutGFW/pages/ directx.aspx, (accessed 2014-08-10).

Döllner J., Kolbe T., Liecke F., Sgouros T., Teichmann K. (2006): The Virtual 3D City Model of Berlin - Managing, Integrating and Communicating Complex Urban Information, in 25th International Symposium on Urban Data Management UDMS. 2006: Aal.

Duarte J . (2002): Malagueira Grammar ; towards a tool for customizing Alvaro Siza’s mass houses at Malagueira. PhD thesis, MIT School of Architecture and Planning.

Dylla K., Frischer B., Mueller P., Ulmer A., Haegler S. (2010) : Rome Reborn 2.0: A Case Study of Virtual City Reconstruction Using Procedural Modeling Techniques . Proceedings of the 37th International Conference, Williamsburg, Virginia, United States of America, March 22-26 (BAR International Series S2079). Archaeopress, Oxford, pp. 62-66.

Ehrig H., Engels G., Kreowski H. J. , Rozenberg G. (1999): Handbook of Graph Grammars and Computing by Graph Transformation: Applications, Languages and Tools. World Scientific Publishing Company.

Eliens A., Wang Y., Riel C., Scholte T. (2007): 3D digital dossiers: a new way of presenting cultural heritage on the web, in Proc. of 12th International Conference on 3D Web Technology (Web3D 07), Perugia, Italy, pp157- 160.

Esri web -http://doc.arcgis.com/en/arcgis-online/reference/about-cityengine-web-viewer.htm (accessed 2014-08-21).

Fisher P. (1993): Visualizing uncertainty in soil maps by animation. *Cartographica* 30(2+3): 20-27.

Flux (2011): http://mediamachines.wordpress.com/, (accessed 2014-08-10).

FME 2011 Special (2011): Available online at: http://evangelism.safe.com/fmeevangelist74/, (accessed 2014-08-10).

Foley J., van Dam A., Feiner S., Hughes, J.(1995): Computer Graphics: Principles and Practice. Addison Wesley, 2nd Ed.

Forte M. (2000): About virtual archaeology: disorders, cognitive interactions and virtuality, Virtual Reality in Archaeology, pp. 247–259.

Frischer B., Niccolucci F., Ryan N., Barcelo J. (2000): From CVR to CVRO: the past, present, and future of cultural virtual reality, in Proceedings of the EuroConference on Virtual Archaeology between Scientific Research and Territorial Marketing (VAST '00), N. Niccolucci, Ed., Arezzo, Italy.

Frischer B. (2008): From digital illustration to digital heuristics. In Beyond Illustration: 2D and 3D Digital Technologies as Tools for Discovery in Archaeology, B. Frischer and A. Dakouri-Hild, Eds. Vol. 1805. BAR International Series, v–xxiv.

FZKViewer (2011): Available online at: http://www.iai.fzk.de/www- extern/index.php?id=1975&L=1, (accessed 2011-08-10).

Glide (2011): http://glide.sourceforge.net/, (accessed 2014-08-10).

Goodchild M., Buttenfield B., Wood J. (1994): On introduction to visualizing data validity. In H. Hearnshaw and D. Unwin, editors, Visualization in Geographical Information Systems, pages141–149.Wiley.

Goovaerts P. (1997): Geostatistics for Natural Resources Evaluation. Oxford University Press, NewYork.

Guidi, Gabriele, Bernard Frischer, and Ignazio Lucenti.(2007): Rome Reborn: Virtualizing Ancient Rome. International Society of Photogrammetry and Remote Sensing, Proceedings of the 2nd ISPRS International Workshop 3D-ARCH 2007, "3D Virtual Reconstruction and Visualization of Complex Architectures," ETH Zurich, Switzerland. www.commission5.isprs.org/ 3darch07/ (accessed 20-8-2014).

Guidi G., Remondino F., Russo M., Rizzi A., Voltolini F., Menna F., Fassi F., Ercoli S., Masci M.E., Benedetti B. (2008): A multi resolution methodology for archeological survey: the Pompeii Forum. Proc. of 14th Int. Conference on Virtual Systems and MultiMedia (VSMM), pp. 51-59, Limassol, Cyprus

Haegler S., Müller P., VanGool L. (2009) :Procedural Modeling for Digital Cultural Heritage. EURASIP Journal on Image and Video Processing Volume 2009, Article ID 852392, 11 pages

Hamza-Lup G., Davis L., and Zeidan O. (2006): Web-Based 3D Planning Tool for Radiation Therapy Treatment, 11th International Symposium on 3D Web Technology (Web3D 06), 18–21 April, Columbia, Maryland.

Harrison M., Keay S., Earl G. (2012): Grammar Modelling and the Visualization of an Uncertain Past : The Case of Building V at Portus. In Proceedings of the 40th Computer Applications and Quantitative Methods in Archaeology Conference (CAA 2012). Southampton, UK.

Havele A. (2010): Web3D and OGC to build cohesive standards for location- based 3D Web visualization services, http://www.web3d.org/press, (accessed 2014-08-10).

Havemann S. (2005): Generative Mesh Modeling. PhD thesis, TU Braun- schweig.

Hiller B. (1996): Space Is The Machine: A Configurational Theory Of Architecture. Cambridge University Press.

InstantReality (2011): http://www.instantreality.org/home/, (accessed 2014-08- 10).

James S. (1997): Drawing Inferences: Visual Reconstructions in Theory and Practice. In The Cultural Life of Images: Visual Representation in Archaeology, edited by B. Molyneaux, 22-48. London: Routledge.

Jiang B., Ormeling F., KainzW. (1995): Visualization support for fuzzy spatial analysis. *Auto Carto 12,* Charlotte, North Carolina. pp. 291-300.

Jung Y., and Behr J. (2008): Extending H-Anim and X3D for Advanced Animation Control, in Web3D 2008: L.A. USA.

Kada M. (2009): The 3D Berlin Project, Photogrammetric Week 2009, Stuttgart, Germany from 7-11 September 2009. 331-340.

Klir G., and Wierman M. (1999): Uncertainty-Based Information: Elements of Generalized Information Theory, 2nd edition. Physica-Verlag. 168pp.

KML (2011): http://www.opengeospatial.org/standards/kml/, (accessed 2014- 08-10).

Knuth D. (1968): Semantics of context-free languages. Mathematical Systems Theory 2, 2, 127–145.

Kolbe T. H. (2007): CityGML–3D Geospatial and Semantic Modelling of Urban Structures, Presentation on the GITA / OGC Emerging Technologies Summit in Washington on 21-Mar-2007, online available at http://www.citygml.org/ fileadmin/citygml/docs/CityGML_ETS4_2007-03- 21.pdf, (accessed 2014-08-10).

Kolbe T. H. (2008): Representing and exchanging 3D city models with CityGML. In: Zlatanova, S., and Lee, J. (Eds.), 3D Geo-Information Sciences, Springer, pp. 15-31.

Legakis J., Dorsey J., Gortler S. J. (2001): Feature-based cellular texturing for architectural models. In Proceedings of ACM SIG- GRAPH 2001, ACM Press, E. Fiume, Ed., 309–316.

Lindenmayer A. (1968): Mathematical models for cellular interaction in development, Parts I and II. Journal of Theoretical Biology,18:280–315.

Lucieer A., and Kraak M.J. (2004): Interactive and visual fuzzy classification of remotely sensed imagery for exploration of uncertainty. International Journal of Geographical Information Science 18(5): 491-512.

MacEachren A. M. (1992): Visualizing uncertain information. *Cartographic Perspectives* (13): 10-19.

MacEachren A. M. (1995): How maps work: Representation, visualization and design. New York, New York: Guilford Press.

MacEachren A.M., Robinson A., Hopper S., Gardner S., Murray R., Gahegan M., Hetzler E. (2005): Visualizing geospatial information uncertainty: What we know and what we need to know. Cartography and Geographic Information Science 32(3):139-160.

Maim J., Haegler S., Yersin B., Mueller P., Thalmann D., Gool L. V. (2007): Populating ancient pompeii with crowds of virtual romans, Proceedings of the 8th International conference on Virtual Reality, Archaeology and Intelligent Cultural Heritage, November 26-30,2007.

Mao B., Harrie L., Ban Y., Fan H. (2011): Real time visualisation of 3D city models in street view based on visual salience, Submitted to International Journal of Geographical Information Science.

March L., and Steadman P. (1974): The Geometry of Environment. MIT Press.

MBARI (2011): http://www.mbari.org/about/, (accessed 2014-08-10).

Mech R., and Prusinkiewicz P. (1996): Visual Models of Plants Interacting with Their Environment. In SIGGRAPH 96 Conference Proceedings, pages 397-410.

Mesa 3D (2011): http://www.mesa3d.org/, (accessed 2014-08-10).

Millett M., and James S. (1983): Excavations at Cowdery's Down, Basingstoke, Hampshire, 1978-81. Archaeological Journal 140: 151-279.

Mitchell W. J. (1990): The Logic of Architecture: Design, Computation, and Cognition. MIT Press.

Mueller P., Vereenooghe T., Ulmer A., Gool L. V. (2005): Automatic reconstruction of roman housing architecture. In Recording, Modeling and Visualization of Cultural Heritage , pp. 287–298.

Müller P., Wonka P., Haegler S., Ulmer A., Gool L. V. (2006): Procedural modeling of buildings. Proceedings of ACM SIGGRAPH 2006 / ACM Transactions on Graphics 25, 3 (2006), 614–623.

Nurminen A. (2006): m-LOMA - a Mobile 3D City Map, in Web3D 2006: Columbia, Maryland.

Nurminen A. (2007): Mobile, hardware-accelerated urban 3D maps in 3G networks, in Web3D 2007: Perugia, Italy.

Octaga (2011): http://www.octaga.com/, (accessed 2014-08-10).

OGC (2008), City Geography Markup Language (CityGML) Encoding Standard, Available online at: http://www.opengeospatial.org/standards/citygml, (accessed 2014-08-10)

OGC (2009): OpenGIS Web Feature Service 2.0 Interface Standard, Available online at: http://www.opengeospatial.org/standards/wfs, (accessed 2014-08- 10).

OpenGL, (2011): http://www.opengl.org/, (accessed 2014-08-10).

Pang A. T., Wittenbrink C. M., Lodha S. K. (1997): Approaches to Uncertainty Visualization. The Visual Computer 13: 370-390.

Pantzer J. (2008): Dusseldorf-traffic simulation with municipal CityGML geodata, 24th Plenary Meeting of the SIG 3D, Available online at: http://www.citygml.org/fileadmin/citygml/docs/2008-06- 03_4_CityGML_D_sseldorf.pdf, (accessed 2014-08-10).

Papagiannakis G., Schertenleib S., Okennedy B., Arevalo-Poizat M., Magnenat-Thalmann N., Stoddart A., Thalmann D. (2005): Mixing virtual and real scenes in the site of ancient pompeii: Research articles. CAVW 16, 1, 11–24.

Parish Y. I. H., Müller P. (2001): Procedural modeling of cities. In Proc. ACM SIGGRAPH, pp. 301–308.

Pescarin S.(2009): Reconstructing Ancient Landscapes, Budapest.

Planet 9 Virtual Cities, (2011): http://www.planet9.com/products_cities/ products_virtcity.html, (accessed 2014-08-10).

Pollock H. E. D. (1980): The Puuc. An Architectural Survey of the Hill Country of Yucatan and Northern Campeche, Mexico. Peabody Museum of Archaeology and Ethnology Harvard University.

Prusinkiewicz P., and Hanan J. (1990): Visualization of botanical structures and processes using parametric L-systems. In D. Thalmann, editor, Scientific Visualization and Graphics Simulation, pages 183–201.

Prusinkiewicz P., and Lindenmayer A. (1991): The Algorithmic Beauty of Plants. Springer Verlag.

RISpec (2011): http://renderman.pixar.com/products/rispec/, (accessed 2014-08- 10).

Roberts J. C., and Ryan N. (1997): Alternative archaeological representations within virtual worlds. In Proceedings of the 4th UK Virtual Reality Specialist Interest Group Conference, edited by R. Bowden, 179–188. Uxbridge: Brunel University.

Scholze S., Moons T., Gool L. V. (2002): A generic 3d model for automated building roof reconstruction. In ISPRS Commission V Symposium 34 , pp. 204–209.

Shubnikov A. V., and Koptsik V. A. (1974): Symmetry in Science and Art. Plenum Press, New York.

Sipser M. (1996): Introduction to the Theory of Computation. Course Technology, Boston.

Stiny G. (1975): Pictorial and Formal Aspects of Shape and Shape Grammars. Birkhauser Verlag, Basel.

Stiny G., and Mitchell W. J. (1978): The palladian grammar. Environment and Planning B 5, 5–18.

Stiny G. (1982): Spatial relations and grammars. Environment and Planning B 9, 313–314.

Storemyr P., Küng A., Bionda D. (2004): EU-Project DEMOTEC-A, Work package 2, Pilot GIS development Nemi: Monitoring and risk assessment of monuments and archaeological sites in the Nemi basin, Colli Albani, Italy. Vol.1: Report (80 p.), vol.2: Catalogue. *Report* no. 2004.039, Expert-Center for Conservation of Monuments and Sites, Zürich.

Takase Y., Sho N., Sone A., Shimiya K. (2003): Auto- matic generation of 3d city models and related applications. In International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences.

Thomas B. J. (2008): WebTOP: An X3D-Based, Web-Delivered, Interactive System for Optics Instruction, in Web3D 2008, L.A. USA.

VirturalGL, (2011): http://www.virtualgl.org/, (accessed 2014-08-10).

VonKoch H. (1905): Une m´ethode g´eom´etrique ´el´ementaire pour l’´etude de certaines questions de la th´eorie des courbes planes. Actamathematica, 30:145–174.

X3D Earth, (2011): http://www.web3d.org/x3d-earth/, (accessed 2014-08-10).

X3D, (2011): http://www.web3d.org/x3d/, (accessed 2014-08-10).

Weyl H. (1952): Symmetry. Princeton University Press.

Willis S. (2007): Protein CorreLogo: an X3D representation of co-evolving pairs, tertiary structure, ligand binding pockets and protein-protein interactions in protein families, in Proc. of 12th International Conference on 3D Web Technology (Web3D 07), Perugia, Italy, pp. 71-80.

Wonka P., Wimmer M., Sillion F., Ribarsky W. (2003): Instant architecture. In Proc. ACM SIGGRAPH, pp. 669–677.

7 Appendix

7.1 CGA rules for reconstruction of the ground floor of Nymphaeum.

```
/**
 * File:    LOD0123.cga
 * Created: 18 Jun 2014 13:38:22 GMT
 * Author:  Abdullah Alattas
 */

version "2013.1"
```

7.2 Defining the four LoDs and Hypotheses

```
@Group("LoD options",0)  @Order(4)
@Range("Second_Hypothesis","Very_High","High","Medium","Low")@Descriptio
n("choose level of detail")
attr LoD                              = "Medium"

const LoD0                            = LoD == "Low"
const LoD1                            = LoD == "Medium"
const LoD2                            = LoD == "High"
const LoD3                            = LoD == "Very_High"
const LoD4                            = LoD == "Second_Hypothesis"
```

7.3 Defining the parameters of the model

```
attr height     = 13.5
attr groundfloor_height = 7.5
attr firstfloor_height  =  6
attr door1_width = 4.6
attr door2_width = 3
attr door3_width = 3.4
attr wallthick = .5
attr wallthick1= .7
attr Base_columnthick = .8
attr No_of_Steps = 19
attr column_thick= .45
attr step_height = .2921
attr white = "#ffffff"
```

7.4 Defining the texture of the model

```
// textures
water_material       = "assets/water.jpg"
Picture_1      = "assets/Final_1.jpg"
Picture_2      = "assets/Final_2.jpg"
Picture_3      = "assets/Final_3.jpg"
roof_text      = "assets/roof.jpg"
ground_floor_rooms = "assets/all_lod.jpg"
ground_floor_rooms1 = "assets/re_mosaico4.jpg"
ground_floor_doorstep = "assets/stone.jpg"
Wood = "assets/Wood.jpg"
exterior_color = "assets/White.jpg"
exterior_terrain = "assets/Untitled.jpg"
```

7.5 Assigning the rule to the footprint

```
Lot -->
    extrude(height) Building
```

7.6 Splitting the mass into facades

```
Building-->
         comp(f){ front : RightFacade  | right :BackFacade  | left :
FrontFacade
    | back :LeftFacade  | top : roof | bottom : Bottom  }


FrontFacade -->
    split(y){ groundfloor_height : GroundfloorFront | firstfloor_height
: Firstfloorfront }

RightFacade-->
   split(y){ groundfloor_height : Groundfloorright | firstfloor_height :
Firstfloorright }

LeftFacade-->
   split(y){ groundfloor_height : Groundfloorleft | firstfloor_height :
Firstfloorleft }

BackFacade-->
    split(y){ groundfloor_height : Groundfloorback | firstfloor_height :
Firstfloorback }
```

7.7 Splitting the front façade into smaller parts

```
GroundfloorFront-->
case LoD4 :
  [split (y){ 5.65 : GroundWall | .35 : Cornisha1 | .4 : Cornisha2 | .5
: Cornisha3 |
            .6 : Cornisha4 }]
     [t(0, 0,-8.4)Groundfloor_front_back1 ]
case LoD3 :
  [split (y){ 5.65 : GroundWall | .35 : Cornisha1 | .4 : Cornisha2 | .5
: Cornisha3 |
            .6 : Cornisha4 }]
     [t(0, 0,-8.4)Groundfloor_front_back1 ]
case LoD2 :
[split (y){ 5.65 : GroundWall | .35 : Cornisha1 | .4 : Cornisha2 | .5 :
Cornisha3 |
            .6 : Cornisha4 }]
     [t(0, 0,-8.4)Groundfloor_front_back1 ]
case LoD1 :
[split (y){ 5.65 : GroundWall | .35 : Cornisha1 | .4 : Cornisha2 | .5 :
Cornisha3 |
            .6 : Cornisha4 }]
     [t(0, 0,-8.4)Groundfloor_front_back1 ]
else:
donothing
```

```
  GroundWall-->
  case LoD4 :
  split (x){ 2 : WallC1 | door1_width : Arch1 | 0.6 : WallC2 |
door2_width : Arch2 |
               1.95 : WallC3 | door3_width : Arch3 | 1.95 : WallC4 |
door2_width : Arch4 |
               0.6 : WallC5 | door1_width : Arch5 | 2 : WallC6 }
  case LoD3 :
  split (x){ 2 : WallC1 | door1_width : Arch1 | 0.6 : WallC2 |
door2_width : Arch2 |
               1.95 : WallC3 | door3_width : Arch3 | 1.95 : WallC4 |
door2_width : Arch4 |
               0.6 : WallC5 | door1_width : Arch5 | 2 : WallC6 }
  case LoD2 :
  split (x){ 2 : WallC1 | door1_width : Arch1 | 0.6 : WallC2 |
door2_width : Arch2 |
               1.95 : WallC3 | door3_width : Arch3 | 1.95 : WallC4 |
door2_width : Arch4 |
               0.6 : WallC5 | door1_width : Arch5 | 2 : WallC6 }
  case LoD1 :
  split (x){ 2 : WallC1 | door1_width : Arch1 | 0.6 : WallC2 |
door2_width : Arch2 |
               1.95 : WallC3 | door3_width : Arch3 | 1.95 : WallC4 |
door2_width : Arch4 |
               0.6 : WallC5 | door1_width : Arch5 | 2 : WallC6 }
  else:
donothing


Cornisha1-->
   case LoD4 :
   extrude (-wallthick)
   color(white)
   case LoD3 :
   extrude (-wallthick)
   color(white)
   //t(0,-.05,0)
   case LoD2 :
   extrude (-wallthick)
   //t(0,-.05,0)
   color(white)
   case LoD1 :
   donothing
   else:
   donothing


Cornisha2-->
case LoD4 :
   [extrude (-wallthick)block]
    [t(0,0,0)Cornisha2_1]
case LoD3 :
   [extrude (-wallthick)block]
    [t(0,0,0)Cornisha2_1]
case LoD2 :
   [extrude (-wallthick)block]
    [t(0,0,0)Cornisha2_1]
case LoD1 :
extrude (-wallthick)
else:
   donothing
```

```
block-->
case LoD4 :
color(white)
case LoD3 :
color(white)
case LoD2 :
color(white)
else:
donothing
```

7.8 Importing the string course obj file to the model

```
Cornisha2_1-->
case LoD4 :
     i("data/Cornisha2_LOD_1.obj")
     r(90,0,180)
     t(-27.85,-9.6,0)
     s(28,9.75,.4)
     color(white)
case LoD3 :
     i("data/Cornisha2_LOD_1.obj")
     r(90,0,180)
     t(-27.85,-9.6,0)
     s(28,9.75,.4)
     color(white)

case LoD2 :
     i("data/Cornisha2_LOD_0.obj")
     r(90,0,180)
     t(-27.85,-9.6,0)
     s(28,9.75,.4)
     color(white)
else :
   donothing

Cornisha3-->
case LoD4 :
extrude (-wallthick)
color(white)
case LoD3 :
extrude (-wallthick)
color(white)
case LoD2 :
extrude (-wallthick)
color(white)
case LoD1 :
extrude (-wallthick)
else :
   donothing

Cornisha4-->
case LoD4 :
   [extrude (-wallthick)block2]
     [t(0,0,0)Cornisha4_1]
case LoD3 :
   [extrude (-wallthick)block2]
     [t(0,0,0)Cornisha4_1]
case LoD2 :
   [extrude (-wallthick)block2]
     [t(0,0,0)Cornisha4_1]
```

```
case LoD1 :
extrude (-wallthick)
else :
   donothing

block2-->
color(white)

Cornisha4_1-->
case LoD4 :
     i("data/Cornisha4_LOD_1.obj")
     r(90,0,180)
     t(-28.2,-9.8,0)
     s(28.7,10.3,.6)
     color(white)
case LoD3 :
     i("data/Cornisha4_LOD_1.obj")
     r(90,0,180)
     t(-28.2,-9.8,0)
     s(28.7,10.3,.6)
     color(white)
case LoD2:
     i("data/Cornisha4_LOD_0.obj")
     r(90,0,180)
     t(-28.2,-9.8,0)
     s(28.7,10.3,.6)
     color(white)
else :
    donothing

Arch1-->
case LoD4:
   [t(4.6,0,-.36)arch_1]
      [NIL]
case LoD3:
   [t(4.6,0,-.36)arch_1]
      [NIL]
case LoD2:
   [t(4.6,0,-.36)arch_1]
      [NIL]
case LoD1:
      NIL
else :
    donothing
```

7.9 Importing the arches obj file to the model

```
arch_1-->
case LoD4 :
      i("data/ground_floorfacadeArch_1_LoD_1.obj")
      r(-90,180,0)
      s(4.55,.5,5.65)
      color(white)
   case LoD3 :
      i("data/ground_floorfacadeArch_1_LoD_1.obj")
      r(-90,180,0)
      s(4.55,.5,5.65)
      color(white)
   case LoD2 :
     i("data/ground_floorfacadeArch_1.obj")
```

```
      r(-90,180,0)
      s(4.55,.5,5.65)
      color(white)
else :
donothing



Arch2-->
case LoD4 :
   [t(3,0,-.36)arch_2]
    [t(0,0,-3)backarch2]
case LoD3 :
   [t(3,0,-.36)arch_2]
    [t(0,0,-3)backarch2]
case LoD2 :
    [t(3,0,-.36)arch_2]
    [t(0,0,-3)backarch2]
case LoD1 :
   NIL
else :
donothing


           backarch2-->
           case LoD4 :
           [t(0,0,0)backarch2_2]
            [t(0,0,-3)backarch2_3]
           case LoD3 :
           [t(0,0,0)backarch2_2]
            [t(0,0,-3)backarch2_3]
            case LoD2 :
           [t(0,0,0)backarch2_2]
            [t(0,0,-3)backarch2_3]
            else :
            donothing

            backarch2_2-->
            case LoD4 :
            i("data/ground_floorfacadeArch_2.obj")
            r(-90,180,0)
            s(3.8,.5,6)
            t(-3.7,0,0)
            color(white)
            case LoD3 :
            i("data/ground_floorfacadeArch_2.obj")
            r(-90,180,0)
            s(3.8,.5,6)
            t(-3.7,0,0)
            color(white)
            case LoD2 :
            i("data/ground_floorfacadeArch_2.obj")
            r(-90,180,0)
            s(3.8,.5,6)
            t(-3.7,0,0)
            color(white)
            else :
            donothing

            backarch2_3-->
            case LoD4 :
             i("data/ground_floorfacadeArch_2.obj")
            r(-90,180,0)
            s(3.8,.5,6)
            t(-3.7,0,0)
            color(white)
            case LoD3 :
             i("data/ground_floorfacadeArch_2.obj")
            r(-90,180,0)
            s(3.8,.5,6)
            t(-3.7,0,0)
            color(white)
            case LoD2 :
            i("data/ground_floorfacadeArch_2.obj")
            r(-90,180,0)
            s(3.8,.5,6)
            t(-3.7,0,0)
            color(white)
            else :
            donothing


arch_2-->
case LoD4 :
      i("data/ground_floorfacadeArch_2_LoD_1.obj")
      r(-90,180,0)
      s(2.95,.5,5.65)
      color(white)
   case LoD3 :
      i("data/ground_floorfacadeArch_2_LoD_1.obj")
      r(-90,180,0)
      s(2.95,.5,5.65)
      color(white)
   case LoD2 :
      i("data/ground_floorfacadeArch_2.obj")
      r(-90,180,0)
      s(2.95,.5,5.65)
      color(white)
  else :
  donothing


Arch3-->
case LoD4 :
   [t(3.45,0,-.36)arch_3]
      [t(0,0,-3)backarch3]
case LoD3 :
   [t(3.45,0,-.36)arch_3]
      [t(0,0,-3)backarch3]
case LoD2 :
   [t(3.45,0,-.36)arch_3]
      [t(0,0,-3)backarch3]
case LoD1 :
   NIL
 else :
  donothing



         backarch3-->
         case LoD4 :
            [t(0,0,0)backarch3_2]
            [t(0,0,-3)backarch3_3]
         case LoD3 :
```

```
        [t(0,0,0)backarch3_2]
        [t(0,0,-3)backarch3_3]
    case LoD2 :
        [t(0,0,0)backarch3_2]
        [t(0,0,-3)backarch3_3]
    else :
          donothing

    backarch3_2-->
    case LoD4 :
     i("data/ground_floorfacadeArch_2.obj")
        r(-90,180,0)
        s(5,.5,7)
        t(-4.15,0,0)
        color(white)
    case LoD3 :
     i("data/ground_floorfacadeArch_2.obj")
        r(-90,180,0)
        s(5,.5,7)
        t(-4.15,0,0)
        color(white)
    case LoD2 :
        i("data/ground_floorfacadeArch_2.obj")
        r(-90,180,0)
        s(5,.5,7)
        t(-4.15,0,0)
        color(white)
   else :
          donothing

      backarch3_3-->
      case LoD4 :
     i("data/ground_floorfacadeArch_2.obj")
        r(-90,180,0)
        s(5,.5,7)
        t(-4.15,0,0)
        color(white)
        case LoD3 :
     i("data/ground_floorfacadeArch_2.obj")
        r(-90,180,0)
        s(5,.5,7)
        t(-4.15,0,0)
        color(white)
    case LoD2 :
        i("data/ground_floorfacadeArch_2.obj")
        r(-90,180,0)
        s(5,.5,7)
        t(-4.15,0,0)
        color(white)
   else :
          donothing


arch_3-->
case LoD4 :
          i("data/ground_floorfacadeArch_3_LoD_1.obj")
          r(-90,180,0)
          s(3.4,.5,5.65)
          color(white)
   case LoD3 :
          i("data/ground_floorfacadeArch_3_LoD_1.obj")
          r(-90,180,0)
```

```
                s(3.4,.5,5.65)
                color(white)
          case LoD2 :
                i("data/ground_floorfacadeArch_3.obj")
                r(-90,180,0)
                s(3.4,.5,5.65)
                color(white)
          else :
          donothing

Arch4-->
case LoD4 :
      [t(3,0,-.36)arch_4]
      [NIL]
  case LoD3 :
      [t(3,0,-.36)arch_4]
      [NIL]
  case LoD2 :
         [t(3,0,-.36)arch_4]
          [NIL]
  case LoD1 :
          NIL
  else :
          donothing

      arch_4-->
      case LoD4 :
                 i("data/ground_floorfacadeArch_2_LoD_1.obj")
                 r(-90,180,0)
                 s(2.95,.5,5.65)
               color(white)
          case LoD3 :
                 i("data/ground_floorfacadeArch_2_LoD_1.obj")
                 r(-90,180,0)
                 s(2.95,.5,5.65)
                 color(white)
          case LoD2 :
                 i("data/ground_floorfacadeArch_2.obj")
                 r(-90,180,0)
                s(2.95,.5,5.65)
                color(white)
           else :
                donothing

Arch5-->
case LoD4 :
          [t(4.6,0,-.36)arch_5]
               [NIL]
   case LoD3 :
          [t(4.6,0,-.36)arch_5]
               [NIL]
   case LoD2 :
         [t(4.6,0,-.36)arch_5]
               [NIL]
   case LoD1 :
         NIL
   else :
                donothing

      arch_5-->
      case LoD4 :
                set(trim.vertical, false)
               i("data/ground_floorfacadeArch_1_LoD_1.obj")
               r(-90,180,0)
                s(4.55,.5,5.65)
                color(white)
         case LoD3 :
               set(trim.vertical, false)
               i("data/ground_floorfacadeArch_1_LoD_1.obj")
               r(-90,180,0)
                s(4.55,.5,5.65)
                color(white)
         case LoD2:
                set(trim.vertical, false)
                i("data/ground_floorfacadeArch_1.obj")
               r(-90,180,0)
               s(4.55,.5,5.65)
               color(white)
         else :
         donthing
WallC1-->
case LoD4:
         [t(2.1,0,0)WallC_1]
         [t(0,0,0)Wall_f_g]
   case LoD3:
         [t(2.1,0,0)WallC_1]
         [t(0,0,0)Wall_f_g]
   case LoD2:
         [t(2.1,0,0)WallC_1]
         [t(0,0,0)Wall_f_g]
   case LoD1:
         donothing
   else :
         donthing

Wall_f_g-->
case LoD4 :
color(white)
case LoD3 :
color(white)
case LoD2 :
color(white)
else:
donothing

WallC_1-->
case LoD4 :
     i("data/colume_ground_wallc_Edit1_LoD_1.obj")
         t(-2.05,0,.3)
         r(-90,0,0)
         s(2.1,.3,6)
         color(white)
   case LoD3 :
         i("data/colume_ground_wallc_Edit1_LoD_1.obj")
         t(-2.05,0,.3)
         r(-90,0,0)
         s(2.1,.3,6)
         color(white)
   case LoD2 :
         i("data/colume_ground_wallc_1.obj")
         r(-90,180,0)
         s(2.1,.3,6)
         color(white)
   else :
         donthing


WallC2-->
case LoD4 :
         [t(.65,0,0)WallC_2]
         [Wall_f_g]
   case LoD3 :
         [t(.65,0,0)WallC_2]
         [Wall_f_g]
   case LoD2 :
         [t(.65,0,0)WallC_2]
         [Wall_f_g]
   case LoD1 :
         donothing
   else :
         donthing

WallC_2-->
case LoD4 :
         i("data/colume_ground_wallc_Edit2_LoD_1.obj")
         t(-.65,0,.3)
         r(-90,0,0)
         s(.65,.3,6)
         color(white)
   case LoD3 :
         i("data/colume_ground_wallc_Edit2_LoD_1.obj")
         t(-.65,0,.3)
         r(-90,0,0)
         s(.65,.3,6)
         color(white)
   case LoD2 :
         i("data/colume_ground_wallc_2.obj")
         r(-90,180,0)
         s(.65,.3,6)
         color(white)
   else :
         donthing

WallC3-->
case LoD4 :
         [t(2,0,0)WallC_3]
         [Wall_f_g]
   case LoD3 :
         [t(2,0,0)WallC_3]
         [Wall_f_g]
   case LoD2 :
         [t(2,0,0)WallC_3]
         [Wall_f_g]
   case LoD1 :
         donothing
   else :
         donthing

WallC_3-->
case LoD4 :
         i("data/colume_ground_wallc_Edit1_LoD_1.obj")
         t(-2.05,0,.3)
         r(-90,0,0)
         s(2.1,.3,6)
         color(white)
   case LoD3 :
         i("data/colume_ground_wallc_Edit1_LoD_1.obj")
         t(-2.05,0,.3)
         r(-90,0,0)
         s(2.1,.3,6)
         color(white)
   case LoD2 :
         i("data/colume_ground_wallc_1.obj")
         r(-90,180,0)
         s(2.1,.3,6)
         color(white)
   else :
         donothing

WallC4-->
case LoD4 :
         [t(2,0,0)WallC_4]
         [Wall_f_g]
   case LoD3 :
         [t(2,0,0)WallC_4]
         [Wall_f_g]
   case LoD2 :
         [t(2,0,0)WallC_4]
         [Wall_f_g]
   case LoD1 :
         donothing
   else :
         donothing


WallC_4-->
case LoD4 :
         i("data/colume_ground_wallc_Edit1_LoD_1.obj")
         t(-2.05,0,.3)
         r(-90,0,0)
         s(2.1,.3,6)
         color(white)
   case LoD3 :
         i("data/colume_ground_wallc_Edit1_LoD_1.obj")
         t(-2.05,0,.3)
         r(-90,0,0)
         s(2.1,.3,6)
         color(white)
   case LoD2 :
         i("data/colume_ground_wallc_1.obj")
         r(-90,180,0)
         s(2.1,.3,6)
         color(white)
   else :
         donothing

WallC5-->
case LoD4 :
         [t(.65,0,0)WallC_5]
         [Wall_f_g]
   case LoD3 :
         [t(.65,0,0)WallC_5]
         [Wall_f_g]
   case LoD2 :
         [t(.65,0,0)WallC_5]
         [Wall_f_g]
   case LoD1 :
         [t(.65,0,0)WallC_5]
         [donothing]
   else :
         donothing



WallC_5-->
case LoD4 :
         i("data/colume_ground_wallc_Edit2_LoD_1.obj")
         t(-.65,0,.3)
         r(-90,0,0)
         s(.65,.3,6)
         color(white)
   case LoD3 :
         i("data/colume_ground_wallc_Edit2_LoD_1.obj")
         t(-.65,0,.3)
         r(-90,0,0)
         s(.65,.3,6)
         color(white)
   case LoD2 :
         i("data/colume_ground_wallc_2.obj")
         r(-90,180,0)
         s(.65,.3,6)
         color(white)
   else :
         donothing

WallC6-->
case LoD4 :
         [t(2,0,0)WallC_6]
         [Wall_f_g]
   case LoD3 :
         [t(2,0,0)WallC_6]
         [Wall_f_g]
   case LoD2 :
         [t(2,0,0)WallC_6]
         [Wall_f_g]
   case LoD1 :
         donothing
   else :
         donothing


WallC_6-->
case LoD4 :
         i("data/colume_ground_wallc_Edit1_LoD_1.obj")
         t(-2.05,0,.3)
         r(-90,0,0)
         s(2.1,.3,6)
         color(white)
   case LoD3 :
         i("data/colume_ground_wallc_Edit1_LoD_1.obj")
         t(-2.05,0,.3)
         r(-90,0,0)
         s(2.1,.3,6)
         color(white)
   case LoD2 :
         i("data/colume_ground_wallc_1.obj")
         r(-90,180,0)
         s(2.1,.3,6)
         color(white)
```

```
              else :
                    donothing
```

7.10 Creating the CGA rule for the ground left façade

7.10.1 Splitting the left façade into two parts.

```
Groundfloorleft-->
case LoD4 :
         split (x){ 25.6 : SolidWall2 |8.4 : SolidWall1 }
   case LoD3 :
         split (x){ 25.6 : SolidWall2 |8.4 : SolidWall1 }
   case LoD2 :
         split (x){ 25.6 : SolidWall2 |8.4 : SolidWall1 }
   case LoD1 :
         split (x){ 25.6 : SolidWall2 |8.4 : SolidWall1 }
   else :
         donothing
```

7.10.2 Splitting the first part into smaller areas.

```
SolidWall1-->
case LoD4 :
         [split (y){ 5.65 : GroundWall_left | .35 : Cornisha1_left | .4
: Cornisha2_left | .5 : Cornisha3_left |.6 : Cornisha4_left }]
         [t(0,0,-7.15)GroundFloor_Copywall]
   case LoD3 :
         [split (y){ 5.65 : GroundWall_left | .35 : Cornisha1_left | .4
: Cornisha2_left |
            .5 : Cornisha3_left |.6 : Cornisha4_left }]
         [t(0,0,-7.15)GroundFloor_Copywall]
   case LoD2 :
         [split (y){ 5.65 : GroundWall_left | .35 : Cornisha1_left | .4
: Cornisha2_left |
            .5 : Cornisha3_left |.6 : Cornisha4_left }]
         [t(0,0,-7.15)GroundFloor_Copywall]
   case LoD1 :
         [split (y){ 5.65 : GroundWall_left | .35 : Cornisha1_left | .4
: Cornisha2_left |
            .5 : Cornisha3_left |.6 : Cornisha4_left }]
         [t(0,0,-7.15)GroundFloor_Copywall]
   else :
         donothing
```

7.10.3 Assigning the wall size

```
Cornisha1_left-->
case LoD4 :
         extrude (-wallthick)
         color(white)
         //t(0,-.05,0)
   case LoD3 :
         extrude (-wallthick)
         color(white)
         //t(0,-.05,0)
   case LoD2 :
         extrude (-wallthick)
         color(white)
         //t(0,-.05,0)
   else :
         donothing

Cornisha2_left-->
case LoD4 :
         extrude (-wallthick)
         color(white)
          //t(0,-.05,0)
   case LoD3 :
         extrude (-wallthick)
         color(white)
          //t(0,-.05,0)
   case LoD2 :
   extrude (-wallthick)
   color(white)
     //t(0,-.05,0)
   else :
         donothing

Cornisha3_left-->
case LoD4 :
         extrude (-wallthick)
         color(white)
          //t(0,-.05,0)
   case LoD3 :
         extrude (-wallthick)
         color(white)
          //t(0,-.05,0)
   case LoD2 :
   extrude (-wallthick)
   color(white)
     //t(0,-.05,0)
   else :
         donothing

Cornisha4_left-->
case LoD4 :
         extrude (-wallthick)
         color(white)
          //t(0,-.05,0)
   case LoD3 :
         extrude (-wallthick)
         color(white)
          //t(0,-.05,0)
   case LoD2 :
   extrude (-wallthick)
   color(white)
     //t(0,-.05,0)
   else :
         donothing
```

7.10.4 Creating the walls of the ground rooms by copying the left wall

```
GroundFloor_Copywall-->
case LoD4 :
          [extrude (wallthick) Wall_g ]
                [t(0,0,-4.25)GroundFloor_Copywall2]
   case LoD3 :
          [extrude (wallthick) Wall_g ]
                [t(0,0,-4.25)GroundFloor_Copywall2]
   case LoD2 :
          [extrude (wallthick) Wall_g ]
                [t(0,0,-4.25)GroundFloor_Copywall2]
   case LoD1 :
          [donothing]
                [t(0,0,-4.25)GroundFloor_Copywall2]
   else :
         donothing

Wall_g-->
case LoD4 :
t(0,0,.3)
s(8.30,.5,7.2)
case LoD3 :
t(0,0,.3)
s(8.30,.5,7.2)
case LoD2 :
t(0,0,.3)
s(8.30,.5,7.2)
else:
donothing

GroundFloor_Copywall2-->
case LoD4 :
          [extrude (wallthick) Wall_g ]
                [t(0,0,-5.35)GroundFloor_Copywall3]
   case LoD3 :
          [extrude (wallthick) Wall_g ]
                [t(0,0,-5.35)GroundFloor_Copywall3]
   case LoD2 :
          [extrude (wallthick) Wall_g ]
                [t(0,0,-5.35)GroundFloor_Copywall3]
   case LoD1 :
          [donothing ]
                [t(0,0,-5.35)GroundFloor_Copywall3]
   else :
         donothing


GroundFloor_Copywall3-->
case LoD4 :
          [extrude (wallthick) Wall_g ]
                [t(0,0,-4.3)GroundFloor_Copywall4]
   case LoD3 :
          [extrude (wallthick) Wall_g ]
                [t(0,0,-4.3)GroundFloor_Copywall4]
   case LoD2 :
          [extrude (wallthick) Wall_g ]
                [t(0,0,-4.3)GroundFloor_Copywall4]
   case LoD1 :
          [donothing ]
                [t(0,0,-4.3)GroundFloor_Copywall4]
   else :
         donothing

GroundFloor_Copywall4-->
case LoD4 :
         [extrude (wallthick) Wall_g ]
   case LoD3 :
         [extrude (wallthick) Wall_g ]

   case LoD2 :
         [extrude (wallthick) Wall_g ]

   case LoD1 :
               donothing
   else :
         donothing


GroundWall_left-->
case LoD4:
         split (x){ 2 : WallC_left1 | 4.4 : SolidWall_leftfacade | 2 :
WallC_left2 }
   case LoD3 :
         split (x){ 2 : WallC_left1 | 4.4 : SolidWall_leftfacade | 2 :
WallC_left2 }
   case LoD2 :
         split (x){ 2 : WallC_left1 | 4.4 : SolidWall_leftfacade | 2 :
WallC_left2 }
   case LoD1 :
         split (x){ 2 : WallC_left1 | 4.4 : SolidWall_leftfacade | 2 :
WallC_left2 }
   else :
         donothing

WallC_left2-->
case LoD4 :
         [t(0,0,0)wall_L]
               [t(0,0,.25)column_left2]
   case LoD3 :
         [t(0,0,0)wall_L]
               [t(0,0,.25)column_left2]
   case LoD2 :
         [t(0,0,0)wall_L]
               [t(0,0,.25)column_left2]
   case LoD1 :
       [donoting]
         [NIL]

   else :
         donothing

SolidWall_leftfacade-->
case LoD4 :
color(white)
case LoD3 :
color(white)
case LoD2 :
color(white)
else :
         donothing

wall_L-->
```

```
case LoD4 :
color(white)
case LoD3 :
color(white)
case LoD2 :
color(white)
else :     donothing
```

7.10.5 Inserting the columns of the left facade

```
column_left2-->
case LoD4 :
               i("data/colume_ground_wallc_Edit1_LoD_1.obj")
         r(-90,0,0)
         s(2.1,.3,6)
         color(white)
   case LoD3 :
               i("data/colume_ground_wallc_Edit1_LoD_1.obj")
         r(-90,0,0)
         s(2.1,.3,6)
         color(white)
   case LoD2 :
         i("data/colume_ground_wallc_1.obj")
         r(-90,0,0)
         s(2.1,.3,6)
         color(white)
   else :
         donothing

WallC_left1-->
case LoD4 :
         [t(0,0,0)wall_L1]
               [t(0,0,.25)column_left1]
   case LoD3 :
         [t(0,0,0)wall_L1]
               [t(0,0,.25)column_left1]
   case LoD2 :
         [t(0,0,0)wall_L1]
               [t(0,0,.25)column_left1]
   case LoD1 :
         [donoting]
               [NIL]
   else :
         donothing
 wall_L1-->
  case LoD4 :
color(white)
case LoD3 :
color(white)
case LoD2 :
color(white)
else :
         donothing

column_left1-->
case LoD4 :
               i("data/colume_ground_wallc_Edit1_LoD_1.obj")
         r(-90,0,0)
         s(2.1,.3,4.05)
         t(0,0,1.95)
```

```
            color(white)
        case LoD3 :
                    i("data/colume_ground_wallc_Edit1_LoD_1.obj")
            r(-90,0,0)
            s(2.1,.3,4.05)
            t(0,0,1.95)
            color(white)
        case LoD2 :
                    i("data/colume_ground_wallc_1.obj")
            r(-90,0,0)
            s(2.1,.3,4.05)
            t(0,0,1.95)
            color(white)
        else :
            donothing

    SolidWall2-->
    case LoD4 :
            t(0,1.95,0)
                    s(25.6,5.55,0)
                    split(x) { ~19.6: SolidWall2_split | 5.85 :
    stairswall_left  }
        case LoD3 :
            t(0,1.95,0)
                    s(25.6,5.55,0)
                    split(x) { ~19.6: SolidWall2_split | 5.85 :
    stairswall_left  }
        case LoD2 :
            t(0,1.95,0)
                    s(25.6,5.55,0)
                    split(x) { ~19.6: SolidWall2_split | 5.85 :
    stairswall_left  }
        case LoD1 :
            t(0,1.95,0)
                    s(25.6,5.55,0)
                    split(x) { ~19.6: SolidWall2_split | 5.85 :
    stairswall_left  }
        else :
            t(0,1.95,0)
                    s(25.6,5.55,0)
                    split(x) { ~19.6: SolidWall2_split | 5.85 :
    stairswall_left  }
```

7.10.6 Splitting the second wall into smaller parts

```
    SolidWall2_split-->
    case LoD4 :
            split (y){ 3.7 : GroundWall_left1 | .35 : Cornisha1_left1 | .4
    : Cornisha2_left1 |
                .5 : Cornisha3_left1 |.6 : Cornisha4_left1 }
        case LoD3 :
            split (y){ 3.7 : GroundWall_left1 | .35 : Cornisha1_left1 | .4
    : Cornisha2_left1 |
                .5 : Cornisha3_left1 |.6 : Cornisha4_left1 }
        case LoD2 :
            split (y){ 3.7 : GroundWall_left1 | .35 : Cornisha1_left1 | .4
    : Cornisha2_left1 |
                .5 : Cornisha3_left1 |.6 : Cornisha4_left1 }
        case LoD1 :
         split (y){ 3.7 : GroundWall_left1 | .35 : Cornisha1_left1 | .4
: Cornisha2_left1 |
            .5 : Cornisha3_left1 |.6 : Cornisha4_left1 }
   else :
         donothing



Cornisha1_left1-->
case LoD4 :
         extrude (-wallthick)
         color(white)
         //t(0,-.05,0)
   case LoD3 :
         extrude (-wallthick)
         color(white)
         //t(0,-.05,0)
   case LoD2 :
         extrude (-wallthick)
         color(white)
         //t(0,-.05,0)
   case LoD1 :
         donothing
   else :
         donothing


Cornisha2_left1-->
case LoD4 :
         [extrude (-wallthick)blockleft1]
         [t(0,0,0)Cornisha2_left1_1]
   case LoD3 :
         [extrude (-wallthick)blockleft1]
         [t(0,0,0)Cornisha2_left1_1]
   case LoD2 :
         [extrude (-wallthick)blockleft1]
         [t(0,0,0)Cornisha2_left1_1]
   case LoD1 :
         donothing
   else :
         donothing
  blockleft1-->
   color(white)

Cornisha2_left1_1-->
case LoD4 :
         i("data/Cornisha2_back_LOD_1.obj")
         r(-90,-90,0)
         t(-10.15,.15,0)
         s(10.3,19.72,.4)
         color(white)
   case LoD3 :
         i("data/Cornisha2_back_LOD_1.obj")
         r(-90,-90,0)
         t(-10.15,.15,0)
         s(10.3,19.72,.4)
         color(white)
case LoD2 :
               i("data/Cornisha2_back_LOD_0.obj")
         r(-90,-90,0)
         t(-10.15,.15,0)
         s(10.3,19.72,.4)
```

```
          color(white)
    else :
          donothing

Cornisha3_left1-->
case LoD4 :
          extrude (-wallthick)
          color(white)
    case LoD3 :
          extrude (-wallthick)
          color(white)
    case LoD2 :
          extrude (-wallthick)
          color(white)
    case LoD1 :
          donothing
    else :
          donothing


Cornisha4_left1-->
case LoD4 :
[extrude (-wallthick)blockleft2]
     [t(0,0,0)Cornisha2_left1_2]
case LoD3 :
[extrude (-wallthick)blockleft2]
     [t(0,0,0)Cornisha2_left1_2]
case LoD2 :
[extrude (-wallthick)blockleft2]
     [t(0,0,0)Cornisha2_left1_2]
else :
          donothing

blockleft2-->
color(white)

Cornisha2_left1_2-->
case LoD4 :
          i("data/Cornisha4_back_LOD_1.obj")
          r(-90,-90,0)
          t(-9.8,.50,0)
          s(10.3,19.72,.6)
          color(white)
    case LoD3 :
          i("data/Cornisha4_back_LOD_1.obj")
          r(-90,-90,0)
          t(-9.8,.50,0)
          s(10.3,19.72,.6)
          color(white)
    case LoD2 :
          i("data/Cornisha4_back_LOD_0.obj")
          r(-90,-90,0)
          t(-9.8,.50,0)
          s(10.3,19.72,.6)
          color(white)
    else :
          donothing

GroundWall_left1-->
case LoD4 :
          split(x) { 17.62: SolidWall2_split1 | 2 : WallC_split1 }
    case LoD3 :
          split(x) { 17.62: SolidWall2_split1 | 2 : WallC_split1 }
    case LoD2 :
          split(x) { 17.62: SolidWall2_split1 | 2 : WallC_split1 }
    case LoD1 :
          split(x) { 17.62: SolidWall2_split1 | 2 : WallC_split1 }
    else :
          donothing

  WallC_split1-->
  case LoD4 :
          [t(0,0,0)wallc_sp1]
                [t(0,0,.25)column_leftrestarea1]
    case LoD3 :
          [wallc_sp1]
                [t(0,0,.25)column_leftrestarea1]
    case LoD2 :
          [wallc_sp1]
                [t(0,0,.25)column_leftrestarea1]
    case LoD1 :
          [donoting]
                [NIL]
    else :
          donothing

 wallc_sp1-->
 case LoD4 :
 color(white)
 case LoD3 :
 color(white)
 case LoD2 :
 color(white)
 else :
          donothing

 column_leftrestarea1-->
 case LoD4 :
          i("data/colume_ground_wallc_Edit1_LoD_1.obj")
          r(-90,0,0)
          s(2.1,.3,4.05)
          color(white)
    case LoD3 :
          i("data/colume_ground_wallc_Edit1_LoD_1.obj")
          r(-90,0,0)
          s(2.1,.3,4.05)
          color(white)
    case LoD2:
          i("data/colume_ground_wallc_1.obj")
          r(-90,0,0)
          s(2.1,.3,4.05)
          color(white)
    else :
          donothing

   stairswall_left-->
   case LoD4 :
   [NIL]
   [t(0, 0,-10.3)stairsGround_Back]
   case LoD3 :
   [NIL]
   [t(0, 0,-10.3)stairsGround_Back]
   case LoD2 :
   [NIL]
```

```
      [t(0, 0,-10.3)stairsGround_Back]
      case LoD1 :
      [NIL]
      [t(0, 0,-10.3)stairsGround_Back]
      else :
             donothing

     stairsGround_Back-->
      case LoD4 :
    color(white)
    case LoD3 :
    color(white)
    case LoD2 :
    color(white)
    else :
             donothing

      SolidWall2_split1-->
      case LoD4 :
    color(white)
    case LoD3 :
    color(white)
    case LoD2 :
    color(white)
    else :
             donothing
```

7.11 *Creating the CGA rule for the ground Right façade*

7.11.1 Splitting the right façade into two parts

```
Groundfloorright-->
case LoD4 :
            split (x){ 8.4 : SolidWall1_Right |25.6 : SolidWall_Right  }
      case LoD3 :
            split (x){ 8.4 : SolidWall1_Right |25.6 : SolidWall_Right  }
      case LoD2 :
            split (x){ 8.4 : SolidWall1_Right |25.6 : SolidWall_Right  }
      case LoD1 :
            split (x){ 8.4 : SolidWall1_Right |25.6 : SolidWall_Right  }
      else :
            donothing
```

7.11.2 Splitting the first wall into smaller parts

```
SolidWall1_Right-->
case LoD4 :
            split (y){ 5.65 : GroundWall_Right | .35 : Cornisha1_Right | .4
: Cornisha2_Right |
            .5 : Cornisha3_Right |.6 : Cornisha4_Right }
      case LoD3 :
            split (y){ 5.65 : GroundWall_Right | .35 : Cornisha1_Right | .4
: Cornisha2_Right |
            .5 : Cornisha3_Right |.6 : Cornisha4_Right }
      case LoD2 :
            split (y){ 5.65 : GroundWall_Right | .35 : Cornisha1_Right | .4
: Cornisha2_Right |
            .5 : Cornisha3_Right |.6 : Cornisha4_Right }
      case LoD1 :
             split (y){ 5.65 : GroundWall_Right | .35 : Cornisha1_Right | .4
: Cornisha2_Right |
             .5 : Cornisha3_Right |.6 : Cornisha4_Right }
      else :
             donothing
```

7.11.3 Extruding the string course of the right wall

```
Cornisha1_Right-->
case LoD4 :
             extrude (-wallthick)
      t(0,-.05,0)
      color(white)
      case LoD3 :
             extrude (-wallthick)
      t(0,-.05,0)
      color(white)
      case LoD2 :
             extrude (-wallthick)
      t(0,-.05,0)
      color(white)
      case LoD1 :
             donothing
      else :
             donothing

Cornisha2_Right-->
case LoD4 :
             extrude (-wallthick)
             color(white)
      case LoD3 :
             extrude (-wallthick)
             color(white)
      case LoD2 :
             extrude (-wallthick)
             color(white)
      case LoD1 :
             donothing
      else :
             donothing

Cornisha3_Right-->
case LoD4 :
             extrude (-wallthick)
             color(white)
     case LoD3 :
             extrude (-wallthick)
             color(white)
      case LoD2 :
             extrude (-wallthick)
             color(white)
      case LoD1 :
             donothing
      else :
             donothing

Cornisha4_Right-->
case LoD4 :
             extrude (-wallthick)
```

```
            color(white)
      case LoD3 :
            extrude (-wallthick)
            color(white)
      case LoD2 :
            extrude (-wallthick)
            color(white)
      case LoD1 :
            donothing
      else :
            donothing
```

7.11.4 Splitting the second wall into smaller parts

```
GroundWall_Right-->
case LoD4 :
            split (x){ 2 : WallC_Right1 | ~4.38 : SolidWall_Rightfacade | 2
: WallC_Right2 }
      case LoD3 :
            split (x){ 2 : WallC_Right1 | ~4.38 : SolidWall_Rightfacade | 2
: WallC_Right2 }
      case LoD2 :
            split (x){ 2 : WallC_Right1 | ~4.38 : SolidWall_Rightfacade | 2
: WallC_Right2 }
      case LoD1 :
            split (x){ 2 : WallC_Right1 | ~4.38 : SolidWall_Rightfacade | 2
: WallC_Right2 }
      else :
            donothing

WallC_Right1-->
case LoD4 :
            [t(0,0,0)wallc_r1]
            [t(0,0,.25)column_Right1_1]
      case LoD3 :
            [t(0,0,0)wallc_r1]
            [t(0,0,.25)column_Right1_1]
      case LoD2 :
            [t(0,0,0)wallc_r1]
            [t(0,0,.25)column_Right1_1]
      case LoD1 :
            [donoting]
            [NIL]
      else :
            donothing
wallc_r1-->
color(white)

column_Right1_1-->
case LoD4 :
            i("data/colume_ground_wallc_Edit1_LoD_1.obj")
      r(-90,0,0)
      s(2.1,.3,6)
      color(white)
      case LoD3 :
            i("data/colume_ground_wallc_Edit1_LoD_1.obj")
      r(-90,0,0)
      s(2.1,.3,6)
      color(white)
      case LoD2 :
            i("data/colume_ground_wallc_1.obj")
      r(-90,0,0)
      s(2.1,.3,6)
      color(white)
      else :
            donothing
```

7.11.5 Changing the scale of the second wall

```
SolidWall_Right-->
case LoD4 :
            t(0,1.95,0)
            s(25.62,5.55,0)
            color(white)
      case LoD3 :
            t(0,1.95,0)
            s(25.62,5.55,0)
            color(white)
      case LoD2 :
            t(0,1.95,0)
            s(25.62,5.55,0)
            color(white)
      case LoD1 :
            t(0,1.95,0)
            s(25.62,5.55,0)
      else :
            donothing

 SolidWall_Rightfacade-->
 color(white)

 WallC_Right2-->
 color(white)
```

7.12 *First Hypothesis with the terrain*:

The figures below show the first hypothesis with the terrain to give better understanding of the environment around the model.

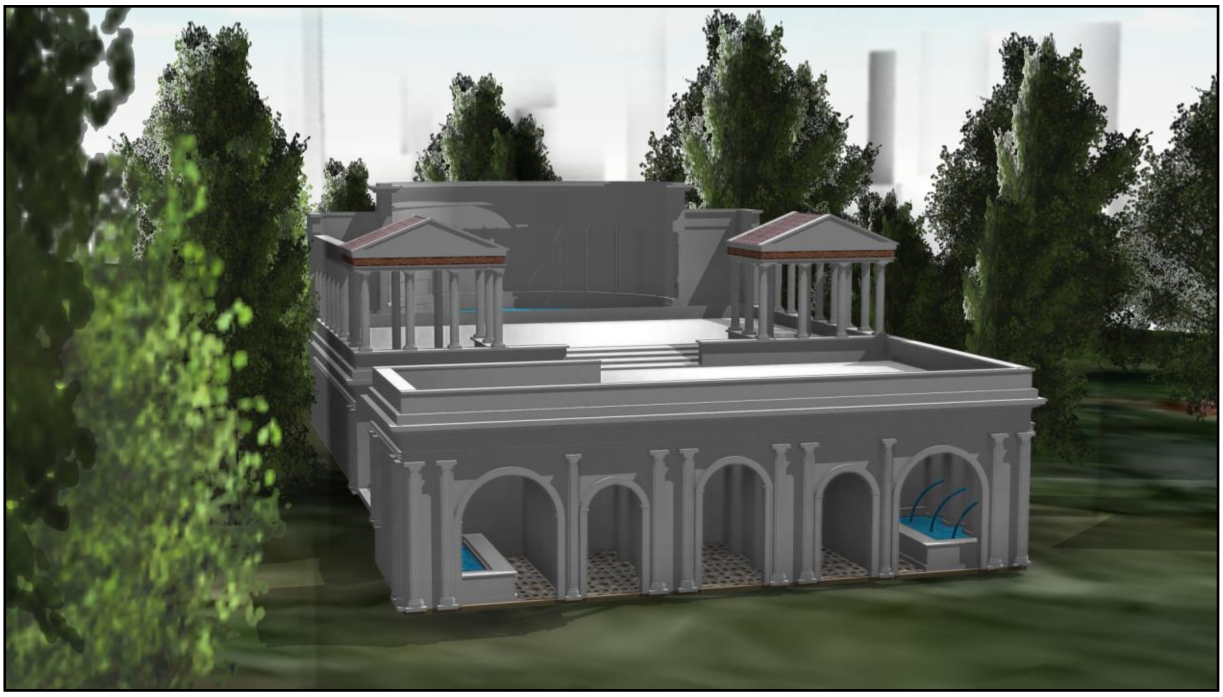


Figure 86: shows the front façade of the first hypothesis.

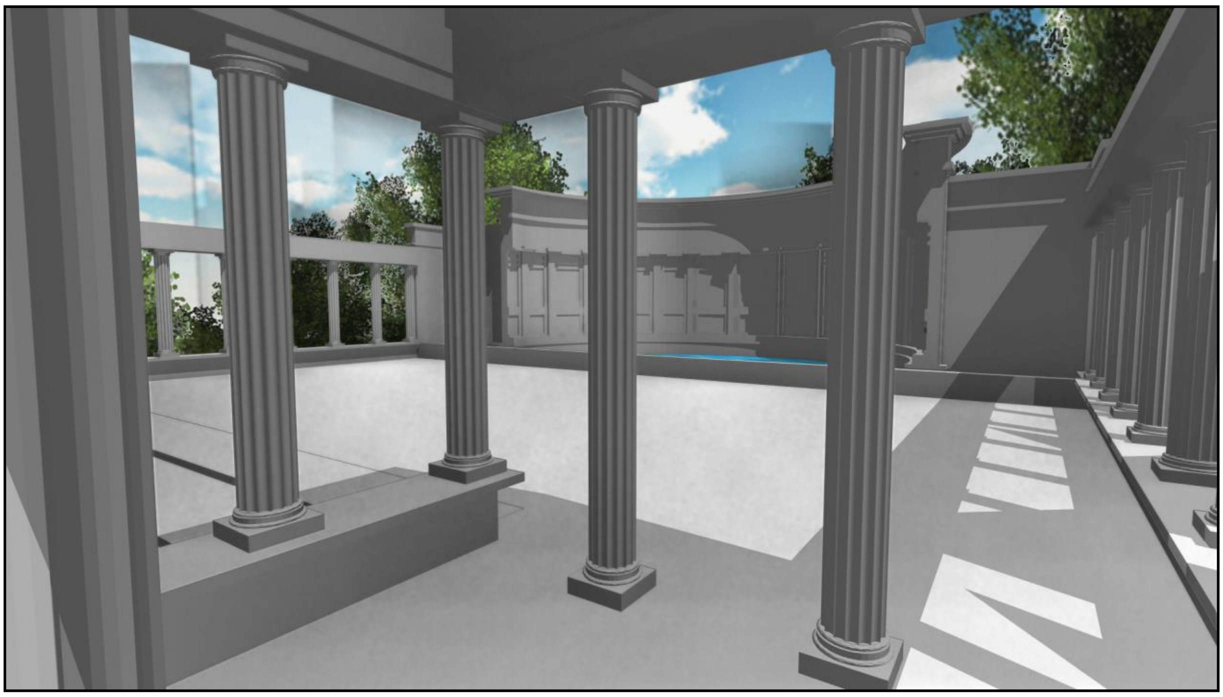


Figure 87: shows the first floor of the first hypothesis.

7.13 *Second Hypothesis with the terrain :*

The figures below show the second hypothesis with the terrain to give better understanding of the environment around the model.

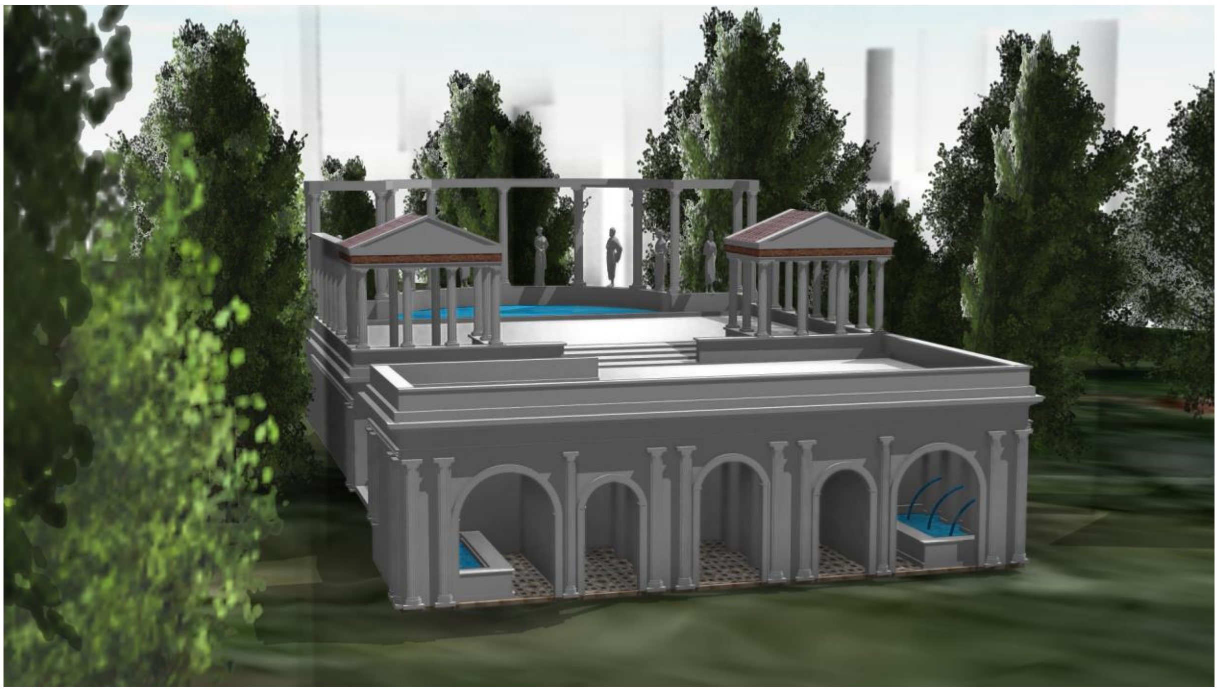


Figure 88: shows the front façade of the second hypothesis.

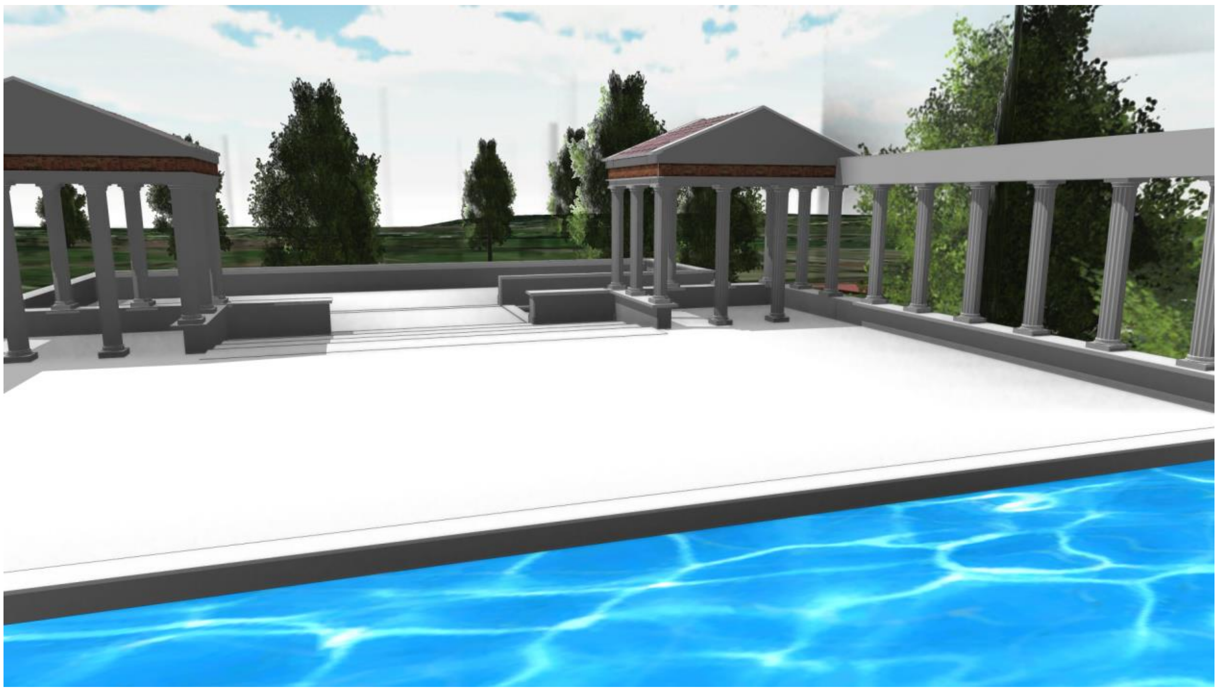


Figure 89: shows the first floor of the second hypothesis.

7.14 *The Interior design of both Hypothesis with the terrain :*

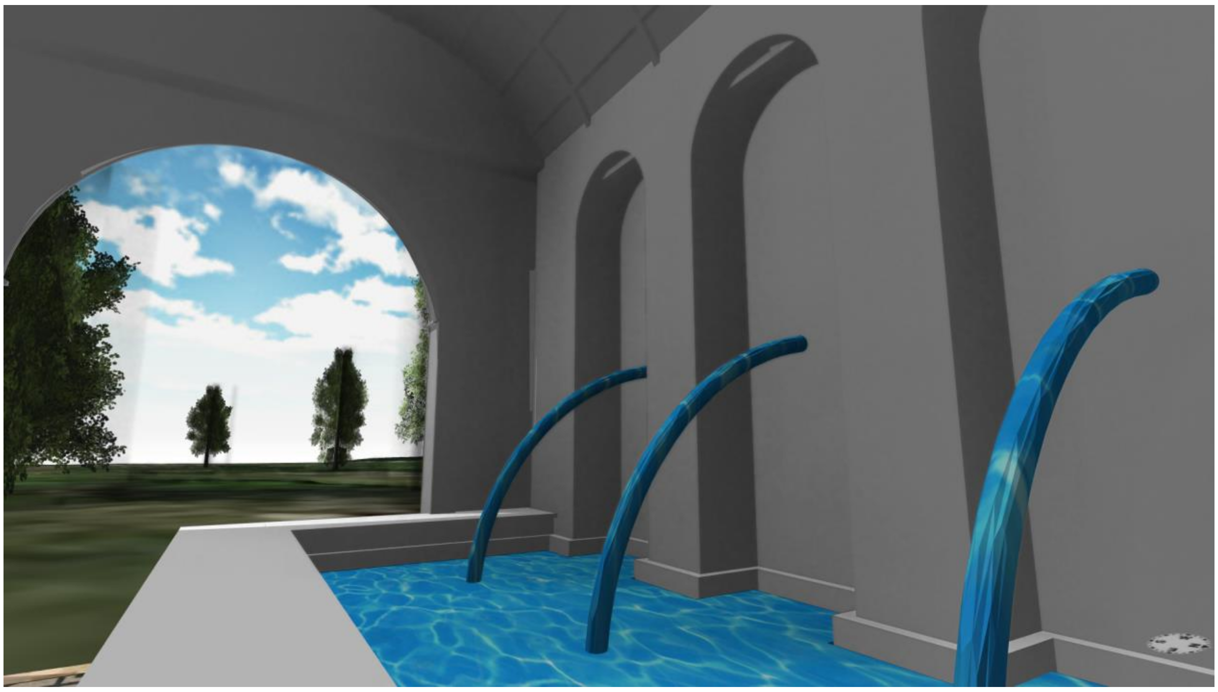


Figure 90: shows the fountain of the ground floor room.

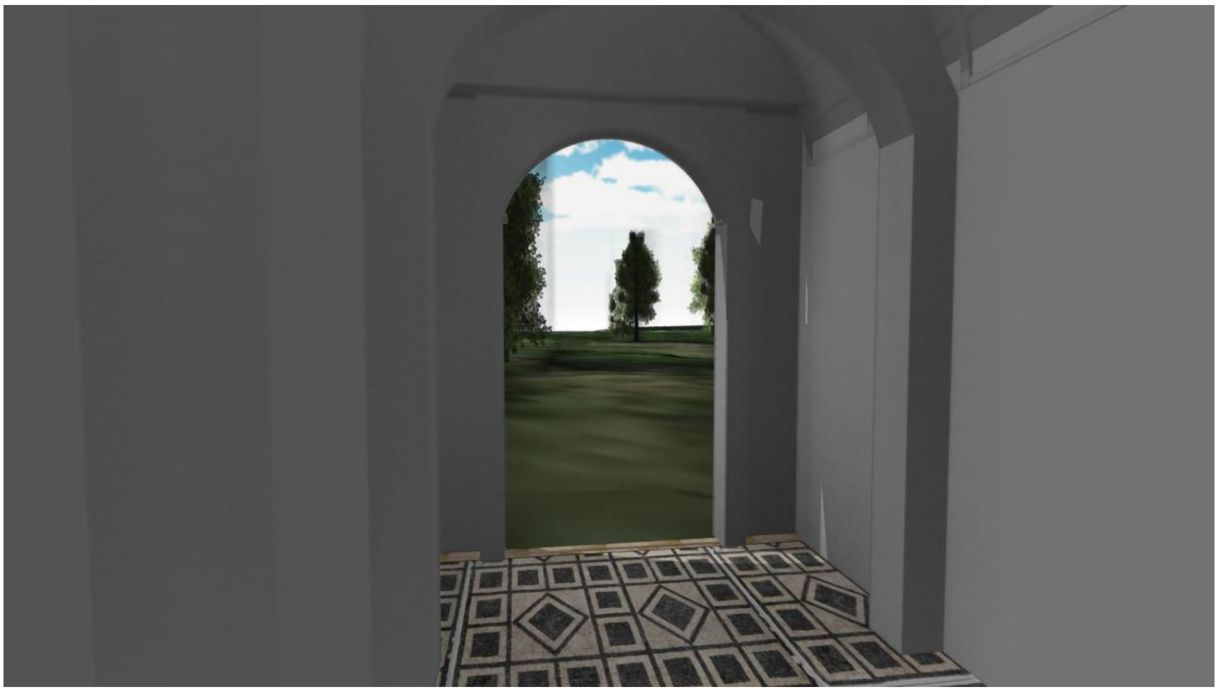


Figure 91: shows the interior design of the midel room of the ground floor.