Procedural City Generation with Combined Architectures for Real-time Visualization

Griffin Poyck
gpoyck@clemson.edu

Procedural City Generation with Combined Architectures for Real-time Visualization

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Fine Arts
Digital Production Arts

by
Griffin Poyck
May 2023

Abstract

The work and research of this paper sought to build upon traditional city generation and simulation in creating a tool that both realistically simulates cities and their prominent features and also creates aesthetic and artistically rich cities using assets that combine several contemporary or near contemporary architectural styles. The major city features simulated are the surrounding terrain, road networks, individual buildings, and building placement. The tools used to both create and integrate these features were created in Houdini with Unreal Engine 5 as the intended final destination. This research was influenced by the city, town, and road networking of *Ghost Recon:Wildlands.* Both games exhibit successful creation and integration of cities in a real-time open world that creates a holistic and visually compelling experience. The software used in the development of this project were Houdini, Maya, Unreal Engine 5, and Zbrush, as well as Adobe Substance Designer, Substance Painter, and Photoshop. The city generation tool was built with the intent that it would be flexible. In this context flexibility refers to the capability to create many different kinds of city regions based on user specifications. Region size, road density and connectivity, and building types are examples of qualities of the city that can be directly controlled. The tool currently uses one set of city assets created with intent for use together and an overall design cohesion but is also built flexibly enough that new building assets could be included, only requiring the addition of building generators for the new set. Alternatively, assets developed with the current generation methods in mind could also be used to change the visual style of the city. Buildings were both generated and placed based on a district classification. Buildings were established as small residential, large residential, religious buildings, and government/commercial before being placed in appropriate locations in the city based on user district specifications.

Table of Contents

List of Figures

Chapter 1

Introduction

The work described in this paper was conducted with three specific goals. The first of these was procedural city generation. Procedural city generation refers to a process by which the elements of a city are placed and represented in a digital 3D environment. This process generally includes rules and procedures by which significant elements of a city, such as buildings and roads, are placed. Procedural cities might also include surrounding terrain and geographic features such as rivers. A significant amount of research work has been conducted in reference to procedural cities, especially their place in virtual worlds for use in movies and video games. A large variety of techniques are used to accomplish various tasks depending on the feature being worked on and the use-case of the work. The visual fidelity and context with which a generation might be used is also dependent on the features and objectives of the respective work. The work of this paper focuses its procedural city generation on terrain, roads, and buildings. Procedural work for the terrain is primarily based on height maps. Roads are largely based on path cost evaluation, and building generation and placement are based on the parameterization of a collection of representative 3D models. The second major goal of the work related to conducting the procedural generation using combined architectural styles in the buildings. A large swath of architectural styles were surveyed with the intent of identifying three styles from similar time periods from largely different parts of the world. Ultimately, the Aztec, Romanesque, and Songnic styles of architecture were chosen for combination in the city model. Because all of the chosen architectural styles were based in old, feudal societies, organic old world style roads were prioritized. Prominent features of these architectures were selected based on a variety of factors including cultural similarities, dissimilarities, and overall visual interest and variety. The final goal of this project is related to real-time visualization. Real-time visualization refers to the ability to view and move around a scene that is lit and rendered in real time, as opposed to a set image or camera path to be rendered and viewed later. Real-time visualization is the standard for interactive applications, especially video games, while more traditional rendering practices are more popular for movie, television, and feature length animation purposes. Real-time visualization, however, could streamline the concept process and visualization for elements, especially in the background of a scene. Real-time assets have the additional requirement of representing both the model and texture in a highly optimized fashion. Unreal Engine 5 was used for the real-time rendering of this work.

This work is related to the field of 3D graphics as a whole, especially as it relates to the representation of virtual worlds. Procedural generation and tools are prominent modern techniques for creating a large variety of assets at a large scale, and is especially relevant to this work both in the design and construction of the models as well as the generation of the materials used in texturing. The generated city layout and creation is related to both video game and movie set design. The combination of different styles of architecture and their relationships with one another is of special interest as well. The real-time nature of the project relates to real-time rendering and lighting. Relevant terms and terminology can be found for review in Appendix C.

The goals of this project in their final state are to create a procedurally generated city focusing on terrain, road, and building generation. This includes creating a convincing representative geographic area and creating roads to effectively traverse the area in the style of old world societies. Buildings should also be procedurally generated from a collection of elements representative of the chosen three architectures. The resulting buildings should be identifiable by their component pieces but should also ultimately create a new and compelling architectural style with aesthetic appeal. Generated buildings and structures should be placed along roads on the terrain in a convincing and realistic way. The buildings and terrain, additionally, should be textured and rendered in real-time lighting. A user should, additionally, be able to move around a scene and view the generated environment. Ultimately, this tool would be able to generate arbitrary terrain and roads, with appropriate building placements based on user designations. The model kit and building generation would be specific to this combined architecture, but elements of the model kit could be used in the process of creating a new style. Any style of building generation would be able to be passed into the rest of the system for final city generation.

The architectural styles chosen for combination were selected in part because of their relative similarity in time period. While not strictly contemporary, focusing on societies with roughly the same technology level and construction capabilities helped maintain cohesive materials and construction methods. Additionally, similarities in culture and views of society, especially in how these views are expressed in the architecture, led to the specific combination of styles. Another facet of the combination is that while Songnic and Romanesque architecture both evolve and influence new styles of architecture in China and Europe respectively, Aztec architecture is largely destroyed or abandoned after the arrival of Spain in central America. Had the Spanish not conquered the area and instead pursued a more diplomatic relationship with the Aztecs, they might have adopted architectural elements of Romanesque architecture found in old churches of Spain. This would additionally allow for the possibility of the Aztec's meeting other civilizations, especially considering the Aztec empire's position in Mexico made it considerably easier for them to reach the Pacific ocean than any European power at the time. Had they eventually come into contact with China, they may have also taken inspiration from ancient Songnic architecture. The similarities in culture, materials, and construction methods would only serve to show these proposed modifications and influences were potentially possible for Aztec capabilities.

Chapter 2

Related Work

The related works chapter focuses on discussing past research that informed many of the decisions made in the research design. This chapter is also meant to better inform the reader of the history of the field as it relates to the project. The following sections are divided by research paper and largely discuss procedural workflows for city or building generation.

2.1 Survey of City Generation Techniques

A Survey of Procedural Techniques for City Generation by George Kelly and Hugh McCabe introduces and describes generalized techniques useful in the field of procedural generation [4]. These techniques may be used together or separately for a wide variety of applications depending on the specific technique. They also identify three key qualities that each of these processes demonstrate as important features. These qualities are abstraction, parametric control, and flexibility. Abstraction refers to the process of removing the need for a user to understand procedures by instead creating an algorithm for the computer to handle when a certain function is needed. Parametric control refers to the ability to manipulate input values in a process or algorithm in order to control its output. Flexibility in this context refers to the ability of a tool to produce a wide range of results without needing to adhere to traditional physical limitations. The primary procedural techniques then discussed are fractals, L-systems, noise functions, tiling, and Voronoi patterning. More information on noise functions is provided in Appendix C. Fractals are nested shapes generated by recursive mathematical algorithms. These algorithms are useful for generating patterned and natural shapes with a large amount of overall cohesion. Kelly and Mccabe define this as 'self-similarity'[4] The recursive nature of these functions also lend them naturally to producing various levels of detail. L-systems refer to Lindenmayer systems[4]. They are a mathematical grammar model originally developed to study growth patterns in algae. In addition, they can be used to create complex structures using a starting condition and a set of productions. L-systems are excellent at modeling and generating complex organic structures with a relatively simple basis. Noise functions are mathematical representations of randomized data. Seeded randomness creates parametric control in the way random values are generated. A noise function first generates randomized data points which are then related to each other using some kind of interpolation function, often linear or cubic. [8] refers to the process of breaking a virtual world into a 2D grid of tiles for referencing and assembly. Tiling is also used in texturing to allow a relatively small texture to cover a larger surface. Texture tiling has become more advanced, especially in reference to landscapes where it has become common practice to combine several textures, layered and interacting in complex ways to produce the final material. This technique allows a level of detail over a large area that is not achievable with a single image. Voronoi texturing is an algorithmic method of creating partitions on a surface based on scattered points and their distance to their closest neighbors[11]. This algorithm creates sections of space divided by intersecting lines. These cell structures can vary widely in shape and size and these characteristics can be driven by the method of point placement for processing. The paper goes on to discuss different applications of each method for different purposes in city generation. It also evaluates each of these techniques against criteria relating to realism, scale, variation, input, efficiency, control, and if it was usable in real-time.

2.2 Citygen

Citygen was an early journey into the power of procedural workflows and their ability to produce a large amount of compelling content without simply adding more labor to a problem[5] This is in part a result of the observation that "additional artist numbers do not generate a proportional yield of content"[5]. This paper also serves as a direct follow up to Kelly and Mccabe's previous paper discussing techniques for procedural city generation[4]. Citygen approaches their method of generation in three stages, producing road networks and buildings. The first stage in this method is the creation of the primary road network. This is generated as a planar graph with an associated adjacency list. Sampling techniques are used to conform roads to the surrounding terrain. The sampling techniques used in creating the primary roads create potential paths forward per sampling technique, and changes in elevation are the most influential factor in determining a chosen path. The closed shapes formed by the primary road networks are referred to as 'city cells', and are the deployment point for secondary roads[5]. Secondary roads follow a growth-based algorithm inspired by, but distinct from, L-systems. The algorithm allows roads to grow from various starting points on primary roads inward within a city cell in parallel to create the secondary road network. As potential roads are proposed to be added, a snapping algorithm determines if the road should be connected to nearby neighbors or removed based on contextual analysis. Enclosed regions in this secondary road network are classified as city blocks. Each city block is divided into lots using a subdivision algorithm. This algorithm attempts to maximize the amount of reasonable, build-able lots in a given block. Buildings are then placed on lots with a bias in generation based hints provided by the user. Suburban buildings retreat from the road while urban buildings attempt to maximize their lot space based on the building's footprint. After buildings are generated and placed, a city with general consistency in nearby buildings and coherent roads has been created.

2.3 Ghost Recon: Wildlands

The development of the world in *Ghost Recon:Wildlands*, a video game made by Ubisoft and released in 2015, used extensive procedural tools in its development relevant to the process of city generation sought in this paper. The technical team, led by Benoit Martinez, developed procedural terrain, roads, railways, bridges, towns, and cities for the game using these procedural workflows[10]. The final terrain map was a 32k x 32k height-field when imported to the engine. Layers on the height-field were used to store information about various parts of the terrain for later use. Roads were created and driven by shortest path nodes in Houdini between designated locations, driven by waypoints in the engine. This organization of the tool allowed artists to adjust the placement of the roads in engine while still maintaining the procedural power of the shortest path operation. Additional support was added to allow tunneling through areas of the height field where appropriate for railways. Another important aspect of the believable terrain comes in the form of the extensive vegetation scattered throughout the world using distribution maps and a variety of rules based on the model's size and orientation. Towns and cities are also created and populated procedurally with given elements using over four hundred thousand building modules and village props combined. Additionally, supplementary tools for placing repeatable objects, such as fences and walls, were developed in Houdini and used in the game engine[10]. The combination of these techniques allowed Ubisoft to create and populate a world with over six hundred and fifty kilometers of roads, ninety kilometers of railways, and fifty-eight villages all seamlessly connected in a realistic and visually compelling fashion. The use of procedural tools without the need to invest in traditional tool development aided tremendously in their production, populating about eighty percent of the world. The work done in this production also demonstrates the progress made in procedural generation for games since the publishing of Citygen in 2007[5].

2.4 Beyond Game Development

Seok Kim, Kavak, and Crooks' paper *Procedural Generation Beyond Game Development* seeks to understand how procedural cities, whose development has been largely tied to video games, can be used to further other fields of research[6] They identify two primary points of research where this technology would be especially helpful. The first of these is social simulation. Social simulation is a paradigm in which 'agents' are individual pieces of a simulation with a particular type that drives their behavior and decision making. Examples of entities used as agents in social simulations are cars, businesses, organizations, and people. Procedural city generation aids in this research in that much of the real world data available is non-standardized and labor intensive to work with. By creating realistic, arbitrary digital versions of urban areas through procedural generation, social scientists would have behavior that closely models observed decision making in a closed environment with easily traceable data. This streamlines the process of trying to ascertain large scale, emergent patterns. The authors identify that this could be specifically useful in research concerning, 'the impact of geography on the robustness of a theory', 'comparing and aligning different theories', and '[s]tandardizing the structure of naming... geographic and population data'[6] The second area of research identified was use as urban test-beds. Urban test beds are simulated regions of an urban environment created with enough detail to conduct simulations for rigorous and repeatable research. The two main components of a test-bed are the urban environment and population. Different levels of abstraction in the representation of a city or given urban sample may vary depending on the test or research being performed. The authors envision this application of cities being useful in the development of self-driving cars and transportation, smart cities, and planning for utility services and local infrastructure. The authors then analyze the effectiveness of procedural city generation in the proposed tasks based on plausibility, diversity vs controllability, interoperability, level of detail, ease of use, and cost[6] They identify the absolute need for a city generation to be plausible, but assert this is less limiting for cities due to both the desired and naturally occurring diversity. Similarly, this diversity is a prominent and desired design feature, but because it is borne of randomness it necessitates a level of control over outputs. In the same fashion, there are a wide variety of methods that can be used a particular generation. In a research oriented tool different solutions for the same piece of an environment with the ability to switch between them should be available. Level of detail identifies the distinction that while most game applications of procedural cities seek a high level of visual fidelity, this will vary largely depending on research purposes and the appropriate amount of resources should be allocated to visuals based on needs. Diversity and control also require significant parameters and their interaction to drive city creation. These controls need to be usable, understandable, and effective in order to facilitate intent driven research. They also identify cost as a major limiting factor in using real world data, and assess that procedural simulations will be similarly undesirable if costly in price or labor requirements. Thus, an advanced and publicly available procedural generator would need to be accessible in order for procedural generators to be used in research as described by the authors.

2.5 Open Street Maps

Open Street Maps(OSM) is a free online tool that allows users to access real world city road and building data for use in digital 3D applications. Open Street Map saves in its own .osm format, which can be imported into 3D programs like Houdini. These files are easily made and downloaded directly from the website. Houdini's import of OSM data imports roads as primitive curves and imports simple block representations for buildings. Uses for this curve representation for roads is readily apparent in the context of the work done on *Ghost Recon: Wildlands.*

2.6 Rules Based Architecture Generation

The generation of historical architecture based on an established rule set discussed by Liu and Ke Wu focuses on the architecture of the Song dynasty[9] Chinese architecture in this period followed a carpentry system known as cai-fen to drive the production of building pieces. They represent this system using a semantic approach, developing a formalized grammar to describe the appropriate construction of various building types. The construction rules are driven by the grammar, and the pieces are a custom module system created based on the description of pieces in cai-fen. This work focuses primarily on the buildings present in a Beijing courtyard, modelling both interior and exterior structures. The buildings of the Beijing courtyard are the gate, gate house, opposite house, festooned gate, veranda, eastern and western side and ear houses, the main house, and the backside house[9] Each building is generated individually before being placed in its appropriate position in the courtyard. The paper makes mention of the importance of hierarchy in Songnic society and its place in architecture. Several aspects of a building's architecture indicate the status of the homeowner. The number of bays in a mansion or palace served as a primary indicator, with distinctions between 7 bay structures made by the 'grade', or height and thickness, of timber beams. Eight grades are described in total, descending in number while ascending in size. Timbers of grades five through one are used in palaces and mansions, determined by the type of building and number of bays. Additionally, roof styles had a strict hierarchy associated with their use. The placements of modules and the types of buildings placed in a given courtyard generation are driven by control parameters determined by the user. Control parameters are distinguished by structure. For example, the main house structure is created using six control parameters.

2.7 Combining Procedural Workflows with Manual Editing

When integrating procedural and manual workflows, natural conflicts arise where designer intentions and model determinations diverge. Bridging these gaps is a necessary part of integrating procedural work into a convincing virtual world. When examining what tools may be required and how integration might be conducted, the paper *Integrating Procedural Generation and Manual Editing of Virtual Worlds* takes a declarative modeling approach to procedural work. In a declarative workflow, a designer describes or places representations of features or locations in a sketch of the virtual world. This sketch is then processed with customized procedural tools to place features where designated and handle interactions between overlapping or nearby features. This contextual processing is referred to as consistency maintenance. This approach helped drive designer efficiency, ease of use, and to establish an integration between manual and procedural work as a baseline. After baseline procedural sketches were produced, manual editing operations to interact with this generation method were considered. Editing operations were divided into coarse, medium, fine, and micro level operations. Coarse level operations focus mainly on the largest features in the world, including cities and mountains. At the medium level, large scale features are edited at a finer level. At the fine level, individual object level elements are edited. At the micro level, data of a given object such as a geometric mesh or texture is added or edited. Coarse and medium level edits are identified as strong candidates for procedural workflows while edits made at the fine and micro level are more oriented toward manual editing. Examples of potential edit operations are then detailed for different elements of the declarative model with associated editing levels. This table also classified edits into six generalized types of modifications, depending on what feature was being modified or what data about a feature was being changed. The three major issues identified in subsequent analysis were the need to preserve manual changes, the similar need to balance control versus overall consistency, and the complications presented in an iterative modeling workflow. Preserving manual changes complicates generation in that once a manual feature has been added to a procedural generation, it must regenerate with this feature in mind. This may also cascade into requiring other features to regenerate. Ensuring the integration of this feature and the correct regeneration of impacted features is necessary both to allow a designer to execute their intent and to avoid frustration and loss of time with lost work. Balancing control against consistency refers to the need for designers to be able to manipulate the world versus the need for the world to make sense and follow its own rules. If no consistency controls are in place, nonsensical designs or potentially large amounts of manual editing are necessary, while too much consistency maintenance can cause the editing process to become obtuse and frustrating. An iterative modeling workflow combines and exacerbates these problems, as after an initial manual edit is made, additional edits may be made as well. Additionally, undo and redo become complicated operations when large scale procedural operations must be computed. Storing previous iterative states of procedural work takes a large amount of memory, so the team addressed redo and undo processes using localized regeneration of features. There is, however, concern that this may not be scalable for medium or coarse level operations. Fully regenerating affected features may feel unresponsive or slow for expectations of the feature based on paradigm.

Proposed solutions included further declarative processing by adding a large amount of low level parameters to force implement desired features. The second solution is a two-phased approach in which procedural and manual operations are performed and integrated separately in stages without interacting directly. Both of these approaches allow editing with a relative level of simplicity, but risk becoming restrictive in what designers are able to create. The last, and most desired, solution is one that would integrate procedural and manual elements effectively across iterations. Needs of such a system and potential mechanisms to allow its function, such as element grouping, feature and area locking, and operation scope.

Chapter 3

Research Design

The ultimate goals of the project were to create compelling cities, and to focus on combining architectural styles to create something new. The tool created was also intended to be able to be used in real-time, meaning that assets created needed to be both visually compelling and resource efficient in order to achieve the best results. The chosen architectural styles that are integrated in the base set of city assets were combined based on both complementary cultural significance as well as complementary visual appeal. The three architectural styles combined were Aztec, Romanesque, and Songnic architecture. All of these architectural styles exist within feudal societies spanning from the second to the sixteenth century and located in largely separated parts of the world. A large motivating factor in choosing architectural styles from a variety of locations was to create new and diverse combinations of features that may not have realistically cross-pollinated over the course of history. By combining features and rules from each style, the assets sought to create a new and visually compelling style that can be partly identified by its component pieces while still looking new and unique.

3.1 Architecture

3.1.1 Aztec Architecture

The most modern of the architectural styles used is the Aztec style, dominating what is now central Mexico in the fifteenth and sixteenth centuries. Aztec architecture prominently includes tiered pyramids known as tecpans [3]. Despite the common purpose of the large structures to imply power, no strict governing rules or features dictated which tecpan should be used for a given purpose. The largest of the tecpans were imperial palaces in the capital, serving as seats of government and religious power. Similarly, tribute city-state tecpans radiated out from seats of power in a hierarchical fashion, keeping outer societies within the fold of the empire. These, however, only account for the administrative uses of the tecpan. The other main functional uses of tecpans were as mansions for the wealthy or as 'pleasure palaces and retreats' [3], used for a variety of recreation including dance halls, sports games, hunting lodges, and schools. These recreational buildings were especially present in larger towns and cities. Tecpans typically held courtyards or community plazas at the peak, as well as several buildings dependent on the function of the tecpan. The further into the interior of the tecpan from the open plaza, the more privileged the access. The homes of nobles in Aztec society were typically made of stone, while peasant homes were made of adobe bricks. Additionally, non palace homes were typically constructed as two single story structures, comprised of a building for living and a steam bath. Homes in general did not have doors. Aztec architecture also prominently features symmetry and geometric patterns in its edifices. These patterns include animal and nature motifs, religious motifs, and simple sweeping patterns 'represent[ing]... religious tenets and the power of the state' [1].

3.1.2 Romanesque Architecture

Romanesque architecture mainly refers to the style used throughout Europe, especially in churches in France, Italy, and Spain. The architecture was largely developed by 'young monastic communities' [2] from the early eleventh to mid twelfth century. Despite the rules of the style, it is widely flexible. Romanesque architecture focuses on using simple principles in combination to create compelling structures. Simple proportions such as 1:2, 2:3, and 3:4 are used alongside the golden ratio and are nested together, becoming complex through their combination across a surface. The Romanesque style also achieves unity through repetition and continuation, extending features and placing them across the structure in multiple scales and styles. These scales are also used in hierarchy, similar to the common proportions, often with the larger version of a feature such as an arch encapsulating a smaller version. Additionally, features are used with variety and contrast in mind in order to help define pieces against their surroundings. This is achieved both by the variance of scale, as well as by using a variety of moldings for arches and columns. Combinations of different shapes are also used, such as alternating between an arch and a pier in a compound pier. The continued repetition of the same shapes, however varied, could become muddying and create very busy structures. Because of this conflict, Romanesque architecture places emphasis on articulation, creating clear boundaries between different features in a building [7]. The most prominently repeated form of the Romanesque style is the semicircular arch, ever present in arcades, piers, arched doors and windows, and the barrel vault roof. Other prominent features of the style include the blind arch, Corinthian and Doric columns, buttresses, and tall bell towers that break proportion rules in order to signify the importance of the building as a church.

3.1.3 Songnic Architecture

Songnic architecture was highly standardized, following the cai-fen module system for its carpentry [9]. This system defined precise timber trades with specific sizes to be used based on the size of the palace being constructed. Songnic beams were created with a 3:2 cross section ratio[9]. Palace size was defined by the number of 'bays', or spaces between timbers along one long face of the palace. These bays also followed a dimension of 2:3, width to height. The structures are defined by these bays and timber grades, three to seven bay mansions, and seven to eleven bay palaces. The bay number is always odd in order to maintain symmetry around the main opening of the structure. A variety of roofs with a strict hierarchy are used based on the status of the building or homeowner. Many of the roof types include distinct versions with and without central main ridges. Roofs of the same type lacking the main ridge are distinguished with a 'Juanpeng' identifier after the main roof name. The nine prominent roof types were Yingshan, Yingshan Juanpeng, Xieshan, Xieshan Juanpeng, Double Eaves Xieshan, Xuanshan, Wudian, cone tented roofs, and pyramidal tented roofs. Structures, especially palaces, were constructed with multiple conjoined buildings surrounding a central courtyard. An additionally important cultural aspect of Songnic architecture is that, much like Aztec architecture, size, and especially height, imply power. The Song Dynasty of China also followed a strict hierarchical structure that was carried into its architecture. In Songnic architecture, however, the emperor and powers of the state could not appear to be less than that of religion, meaning the imperial palace or local governor's home needed to be the largest building within a city. Pagoda towers, therefore, were significantly more prevalent in more rural areas and very rarely seen within city walls. Songnic city walls were built of rammed earth, tapering in thickness from the base to the peak.

3.1.4 Combinations

Initial combinations in the architectural styles were identified first through both cultural and structural similarities. All three styles utilize symmetry, scale as a metric of power, and hierarchical distinctions in structure. Additionally, multi structure dwellings were a common factor between Aztec and Songnic architecture. Timbers and their repetition served to communicate status in Songnic architecture, while the similarly repeated columns in Romanesque architecture sought to create unity in form. These and the hierarchical roofs of the Song Dynasty lend themselves naturally to the repetition and hierarchy of forms used in Romanesque architecture. Different amounts of repetition and different types of features, such as columns and arches of Romanesque architecture, could be used to communicate status in a similar way. These factors created a strong base from which to build other aspects of the combined style. Religious and government buildings are made larger than private residences, and noble private residences are made larger than common ones. Romanesque repetition of features in hierarchy creates unity across a single building while individual choices in which types of features and how much of them might denote some kind of status, no matter what kind of building they are placed on. This, as well as scale, has interesting implications on how to interpret pagoda towers and raises the question of whether height or hierarchy of features should be the primary marker of status.

Several distinctions between the styles must also be addressed in order to create a compelling combination that does not lose its influences. While Songnic architecture largely focuses around the silver ratio, Romanesque makes use of the golden ratio, the silver ratio, and several other simple ratios in combination. Because of the more inclusive and dynamic potential arrangements of the Romanesque style, while still including the fundamental proportions of the other, its rule of proportion served as a driving force. Another conflict comes in the differences between how the Song and Aztec empires distinguished government from religion. Songnic architecture is very strict about the separation of religious and government buildings, with an insistence that government buildings be larger, while Aztec architecture is largely ambivalent about relative size so long as it represented the power and importance of the building. In order to bridge this gap, compromises between both styles must be made to find an even middle ground. This was achieved by removing the distinction that government buildings must be larger than religious ones, but also required the distinction between the two structures. In order to accomplish this, double stair tecpans were used for religious structures while single stair tecpans were used for government, commercial, and private buildings. Roofs are another interesting difference across styles. Because the Aztecs largely used stone or adobe in their construction, roofing was primarily constructed of thatched grass or light volcanic stone as a flat top to the building[1]. Songnic roofs were varied and followed hierarchy, while roman roofs were largely tunnel or barrel vaults. The Songnic roofing style was chosen to create superior visual interest, to distinguish shapes from existing semicircular features from the Romanesque style, and to continue to emphasize the hierarchical emphasis of the styles and societies.

3.2 Terrain

The first aspect of the city generated in the process is the landscape. This was both developed early and exists first in the process because the terrain forms the foundation for other city elements. For a compelling city, features such as roads and buildings should suit and be established based on the surrounding geographic elements. The terrain is generated using a height-field workflow. This allows a very large representative area to be stored as a very small and usable file, lending itself to the real-time nature of the final product. Height-fields are defined in Appendix C. The basic workflow using height-fields to create terrain involved using a variety of noise fields to create initial forms before using erosion simulation to create the effects of water run and sedimentation in the terrain. Even with randomness in the noise, however, using the same noise patterns or erosion processes will create very similar types of environments. In order to create a wide variety of geographic features and an overall more realistic representation, four categories of landscape tiles were created. These categories, or biomes, were created to have tiles with similar large scale features that varied individually while also creating four distinct types of landscape pieces. The categories created were plains, hills, mountains, and valleys or basins. Landscape tiles are of a uniform size, and there are four tile varieties for each category. In order to create organic shapes and variety in the terrain, randomized generation, placement, and orientation of the tiles was necessary. Tiles were generated based on a square grid of user defined size in order to provide a large amount of flexibility. When the grid size is set, randomized biomes, specific tiles, and orientations are assigned to each point. Orientation randomness is locked at ninety degree turns in order to maintain grid alignment. These random values are generated using a nondeterministic function, so each time the size of the grid is changed a new set of tiles is generated. This ensures the same layout is not repeated for the same

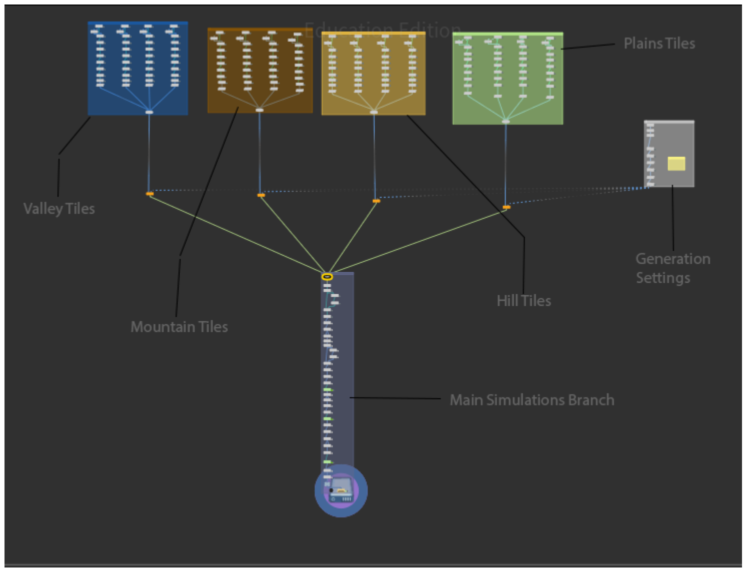


Figure 3.1: Terrain Node Graph breakdown

values. If repeatable randomness was desired, a seeded randomness independent from grid size could be introduced. This would allow the grid size to be expanded or contracted without changing the generation of existing tiles, and would also allow reproduction of tile sets at different sizes. This was not included in the current design in order to streamline the process of quickly generating a variety of layouts before creating a final piece with the desired generation. Instead, finished processing of a given tile set can be saved for further use, and multiple finished states can be saved for comparison. The grid will automatically resize and space points appropriately for tile alignment based on the user specified grid size. Additionally, both more types of tiles in each category and more categories could be added to the terrain generation by adding possible values to the random number generation and adding corresponding height-fields of an appropriate size. Once the tiles are generated, the height-fields are spliced together and processed through a common erosion simulation. The terrain is then exported as a 4033x4033 16 bit raw floating point file containing the height values. Details about this file type are available in Appendix C. This file was then imported to Unreal Engine 5 using the landscape tool. Once imported, a gray diffuse version of the terrain is available to move around and interact with.

3.2.1 Landscape Shader

Once imported to Unreal Engine, a landscape material shader was created for the terrain. This shader combined several physically based rendering(PBR) materials in paint-able layers on the landscape. For more information on PBR materials, see Appendix C. The component materials were developed using material graphs in Substance Designer. The component materials were rock, coarse dirt, and grass. These materials were developed with the intent to layer one on top of the other in the process of representing a realistic landscape. All of these materials were developed in a similar workflow, first creating a base height map before extracting further material maps from this basis. The bottom material in this layering was the rock material. The rock material uses a low scale perlin noise as a basis, creating large areas of raised and lowered features. Rock shapes generated using several nodes in sequence are then scattered across this background in two separate passes with two separate sets of shapes. These shapes are then further modified to break up medium and small level details, including adding scratch and dirt details. A warp is used to break smooth edges on the shapes, while a fractal sum noise is used to create a rock-like surface texture. With the height map now complete, other maps are constructed. A gradient map node is used to construct the base color, while a normal node is used for the normal map. Ambient occlusion and roughness maps are constructed primarily by mixing the height map with a levels node and optionally blending with a grunge map. The middle coarse dirt material was created by first using perlin noise and dirt noise maps within Substance Designer to create a base of irregularly mounded dirt. Rock shapes are then scattered across the surface of this dirt. In order to better separate the rock shapes from the background, a shape splatter color node is used with the data of the original shape splatter used to scatter the rocks. This allows the rock elements to be colored separately from the background gradient map before being placed on top in corresponding locations. The normal, roughness, and ambient occlusion maps are constructed in the same fashion as the rock texture. The grass texture follows a similar workflow, but additionally scatters leaf shapes created in Substance Designer in addition to pebbles. The grass texture also uses the height gradient color map to distinguish between dirt and grass levels in the background height material with fibrous surface texture over the grassy areas. These maps are exported from Substance designer as 4096x4096 bitmaps, using a large texture size because of the large amount of terrain to be covered. Additionally, in order to combat texture tiling in later stages, additional copies of each material are generated at an eight times tile scale with randomized rotation as well.

When developing the landscape shader, textures first needed to be imported to the engine through the process described in Chapter 4 Section 7. Once the textures are imported and prepared for use in engine, both packed images for each material are imported to the shader. A material is then constructed from the two images, utilizing a material function constructed to unpack materials from images in this format. The images are imported to the shader graph as texture objects because they will be sampled multiple times. The eight times scale copies of the materials are also placed in the graph as texture objects. The absolute world position and division nodes are used to create UV tiling settings to be shared by each material in the shader. Additional division nodes for the same purpose at different scales are added later for anti-tiling techniques. All scales are set as editable parameters for real-time adjustment. The constructed materials are then used to create a base layer material automatically for the terrain. This automatic layering first places the grass material over the rock material using a slope mask. This mask is generated using a slope mask node with a vertical slope angle input. The slope blend falloff and slope material blend are created as editable parameters. Masks to drive automatic layering are then generated using images rendered from the height field in Houdini. These maps include erosion simulation information such as the sediment, debris, water, and bedrock maps, as well as the height map itself. These masks are then stored in a single image with one mask per channel before being imported to the engine. Each mask is then used as the alpha for successive standard material layer blend nodes, stacking the materials in order. This finalized auto material is then used as the bottom layer of the paint-able landscape material. Additional layers for each material are added, with an optional top layer for painting foliage megascans on the landscape. These layers are established by the landscape layer blend node. For layer painting to work, however, the landscape tool must first be opened to the paint tab after creating the landscape layer blend node in the material and applying the material to the landscape. At this point, the created landscape layers will be visible in the landscape tool. Layer information must be built and stored for each layer. Layers can be created for weight blending or height blending. Only weight blended layers were used in this landscape material. Once layer information has been built, materials can be painted freely from this window by selecting a layer and using the built-in engine paint tool.

Additional processing is then added to material images before they are unpacked in order to prevent tiling. Close range tiling is combated by first creating an additional copy of each material map by sampling the texture objects a second time. The first of the additional UV coordinates are created, and should be very close to, but different from, the close range UV scale. The x and y coordinates of the texture are then 'swizzled', or swapped, in order to rotate the texture. The image containing the normal map must then have its red and green channels swapped to account for this change. The similarly scaled, rotated textures are then blended using a mask with large blocks of black and white. This process hides the tiling artifacts of the textures close to the players, but does little to combat it in the mid range. The large versions of the textures are then used to improve this area. Texture coordinates of roughly eight times the size are created to be used with the larger version of the textures. These maps are then similarly blended with a linear interpolation, but instead of using a pre-existing mask, one is generated at run-time. This mask is created using the camera depth fade node, which generates a mask based on distance from the camera. The fade offset for this node is converted into a parameter used to control how far from the camera the mask begins. The fade length input is also turned into a parameter, and controls the distance across which the transition between the two materials will occur. This allows more distant areas to use the larger, more randomized materials while using the blended standard and swizzled textures up close. The "For Vertex Shader" boolean option on the camera depth fade node is also set to true in order to ensure its mask is generated in the vertex shader instead of the pixel shader, which helps improve performance.

3.3 Road Networks

When developing the road systems the main priorities were that the roads would function well for navigating the terrain as well as reflect realistic city roads. The road systems were also developed to reflect older roads with more organic shapes in order to maintain characteristics of the feudal societies and time periods the architectural motifs are taken from. This meant ensuring that intersections were limited to four connections, that corners were not too tight, and that roads did not follow unrealistic or impassable paths. Additionally, the roads needed to be able to be generated for an arbitrary terrain input, scaled appropriately, and make meaningful connections in the landscape. To achieve these goals, the road generation is split into several steps. It begins with an initial generation for major highways and large scale layout. This initial generation receives basic processing to combine spline curves and create intersections where curves overlap. The next step is both automatic and manual connections of the pieces generated in the initial step. These connected pieces are then passed through a more complex processing step that makes slope based decisions about which roads are valid and which need to be remapped for navigability. Once each of these pieces are remapped and reintegrated to the initial curves, final processing and manual finalization completes the road system for integration with the terrain.

3.3.1 Initial Generation

The initial road generation creates the basic shapes that the rest of the system will build upon. Two different generation methods were created and can be freely switched between. The two generate different outputs by utilizing similar workflows on slightly different inputs. Both methods used the grid that sets the scale of the terrain as an input, and differed in that one method used a Voronoi fracture of this grid while the other used a scrambled, dense version of the grid. In the fracture based method, a cloud of 1,250 points are scattered across the surface of the grid and used to create a Voronoi cell pattern along the surface. The grid method instead subdivides the grid 3 times to create a similar density of mesh to the fracture method. Both methods then sample the terrain and displace the points on the grid to the corresponding height on the terrain. For both methods, this creates a low resolution copy of the terrain with a connected set of curves across it. In the fracture method, overlapping points are fused and the surface curves are smoothed slightly. In both methods, surface points are then displaced randomly at a uniform scale. In the fracture method, edges with an interior angle lower than ninety degrees or higher than 125 degrees are removed. Start and end points are then designated by the user before paths between these points are traced. These paths primarily consider distance when deciding which curves to use. The initial generation at this stage creates several curve shapes that will serve as the basis of the road network. The number of shapes generated is dependent on the user specified start points, while the complexity of the shapes are controlled both by the user generated end points and subsequent "carve" Houdini nodes that allow the user to control how far paths spread from the start toward their traced end destinations. Through this functionality, users can control how developed each curve shape is before moving to the next steps. The curves are then flattened for processing while height data is stored in a separate variable for use later. A new initial curve in either method can be generated easily by changing the seed for randomized displacement, and the fracture method shapes can be changed further by changing the seed for the point cloud generated on the surface prior to the fracture.

After the basic shapes are created, the points are relaxed, spreading them out from local connections. This helps widen sharp joints and creates space between the curves for later processing. As the relax operation can sometimes introduce overlap and complex shapes, near overlapping points are fused and intersections are created at overlaps. The curves are then smoothed, reversing some of the relax operation's growth and creating smoother and more organic overall shapes. The curve shapes are then passed through an assemble node, which packages the curves and separates each shape as a "piece". These pieces are then connected automatically using a Connect Adjacent Pieces node. The assemble node is critical to packaging and labeling the curve shapes appropriately for connections. The connecting shapes are then merged with the original curve shapes. Additionally, optional user defined lines can be added to ensure shapes are connected appropriately. Two manual connections that can be toggled and manipulated by the user currently exist. Additional connection elements can also be easily added as needed. These manual connection points can also be manipulated at later stages to account for automated processing that may impact them after placement. Near overlapping points are again fused, and intersections are created. The pieces are then merged into one large curve shape and re-sampled to a uniform segment length. The assembled curve shape is then passed along for complex, slope based processing.

3.3.2 Slope Processing

In order to obtain slope data to inform decisions about the roads, a mask was generated on the terrain based on the slope angle. The mask began in areas with a slope angle of forty five degrees and ended at a vertical angle of ninety degrees. This meant that areas with a slope of forty five degrees or less received a zero value in the mask, while other areas received graded values up to one at ninety degrees. These mask values were then used to determine what areas of the map were navigable. A vex script first runs over the primitives in the curve shapes and deletes any where the area it passes through has a mask value of 0.9 or higher. This eliminates very high slope areas from the network and can be tuned to the user's purpose. A python script then runs over the points in the connected curve network. Vex was used for the first process as it automatically runs over every element of a particular type for processing and handles it individually. For the task performed by the python script, however, information about past points and access to those points was necessary. Additionally, iteration variables were necessary. This was potentially solvable using Houdini for loops, but that process would have been clunkier and less efficient. The python script then checks a points mask value, and when it detects that it has entered a high slope area, it begins keeping track of the points in sequence as a "chain". The chain length and an identifying chain number are used to distinguish high slope point chains. As a chain is constructed, its start point is stored. When a slope value that does not qualify is reached, the chain end is recorded. The heights of the start and end of the chain are then compared. If these heights differ significantly, then the high slope was generally in one direction and the area requires a switchback pathing to traverse. If, instead, the two heights are similar with high slope values between then it is likely a bridge is a better solution. The chains are labeled accordingly, with a switchback or bridge start, center, and end recorded for each chain. All points in the chain receive a chain number and the length of the chain for later use. Bridge placement is currently not implemented and remains a flagging and detection feature only. Areas that needed large height change were more common and more commonly impassable than bridge candidates, and were therefore a higher priority to solve.

Areas flagged for switchback handling are then separated from the main curve shape. Switchback start, center, and end points are separated and maintained for processing. Span and center points and their corresponding curve areas are removed from the main shape. Center points on point chains with even chain lengths are slightly closer to the end point than start. To compensate for this, all center points are re-centered based on start and end positions. The normal direction is also calculated during this step. Start and end points are then labeled appropriately in groups for use in the Houdini shortest path node. Simple square grids are then copied to each center point with their orientation and scaled based on the chain length of the area being replaced. Each of these grid locations is then processed separately as a primitive in a Houdini "For Each" loop targeting primitives. Four thousand points are scattered across the surface of the grid being processed. These points are then connected using a Connect Adjacent Pieces node. This node allows each point to search within a designated radius for a designated number of connections. Increasing these values or the amount of points scattered across the grid will cause a significant increase in task time. Once connections are made, near overlapping points are fused and the terrain data is sampled again to set points to the appropriate height. This process essentially creates a triangular mesh representation of the terrain geometry in the local area for path processing. Mask values from the terrain slope mask are also sampled at the same time, and are inverted for the purpose of creating an appropriate relationship in the path tracer. Each connecting piece of this newly created mesh is then passed through another for each loop. This loop manually calculates and stores the slope value of each

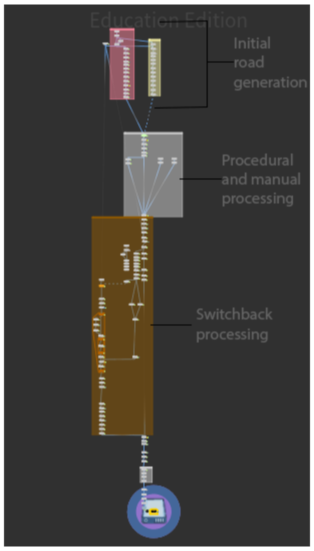


Figure 3.2: Annotated Road Generation Node Graph

curve along the surface and scales the value of slope proportionally. This scaling up of the initial value assists in driving the appropriate behavior in the path tracer.

Once all pieces have been processed, the wire frame of the terrain and the corresponding start and end points, separated from the rest by their identifying chain number, are used to calculate paths through the high slope area. The calculated slope for each piece serves as the primitive cost for using that piece of the path. The inverted mask value is used as the angular turning cost attribute at each point. The processing has thus created a direct relationship between slope and cost while also creating an inverse relationship between slope and turning cost. These relationships create behavior such that in high slope areas it is highly motivated to turn and find a low slope alternative. This is further motivated by omitting distance from the cost. The generated curves are then flattened for final processing and assembly. Once every switchback flagged area has been processed the same way, turns that are too sharp are removed. After this, the amount of connections at each intersection is counted and processed using Vex. Intersections with more than four connections are removed. Near overlapping points are fused, and the process is repeated in order to clean complicated overlapping areas and to ensure that no intersections contain more than four connected curves.

3.3.3 Final Processing with Manual Editing

To begin the final processing steps, the original curve shapes with the switchback flagged sections removed are merged with the newly calculated high slope paths. Near overlapping points are fused and intersections are created. The splines are then merged into a single large curve shape. The shapes are re-sampled to a uniform length of five meters before manual editing begins. Two manual editing steps in which the user can remove primitives, then points for the road network follow. Users are able to add and remove primitives from the selection to be deleted, and are able to go back later and change these parameters. Removing primitives allows the user to remove pieces of curves in whole, removing small disconnected pieces or parts of the road that are too cluttered, superfluous, or otherwise non desirable. The second step that allows deletion of points allows the user to make decisions changing the shapes of existing curves, as well shorten the ends of paths. The curves are then cleaned to remove any remaining floating points or pieces from the editing before being re-sampled to a length of two. It is at this point that the primitive road form can be saved to further develop. Additionally, multiple versions of these could be saved to use with different terrains or as different potential road layouts for a given terrain. Before being integrated to the terrain, an additional layer of contextual editing is made available to the user. This repeats the manual editing steps allowing the removal of both primitives and points, but this time does so with the terrain visible and curves set to appropriate heights. By making the context of the roads visible and obvious, an artist editing the layout will more easily be able to make decisions about which roads are superfluous, necessary, and potentially optional for a given generation or design goal. The masked terrain and final road curves are then both used as inputs in a volume modifying Vex node. This node creates a distance curve of a user specified width along the curves on the surface of the height-field. It uses a smoothing ease value, a minimum width, and a maximum width dictated by the user to smooth the transition from the terrain to the road along the distance curve. A mask is constructed around the curve using the smoothing to control the blending of the roads and the underlying terrain. This masked representation of the road is then combined with the height-field using a linear interpolation.

3.4 Building Generation

3.4.1 Model Kit Design

Building generation was handled by first creating a model kit using architectural features identified in the desired styles. The most prominent features of the Romanesque style were semicircular piers and arches, Corinthian and Doric columns, unity through repetition, and the presence of golden rectangles. The most prominent feature of the Aztec architecture were both administrative and religious tecpans, most notably the double-staired pyramid. Songnic architecture strongly emphasized the silver ratio of 2:3, another simple ratio present in Romanesque architecture, strongly defined and varied roof types, pagoda towers, and palaces distinguished by size and the number of bays along its long face. With this in mind, models were created in Maya for the single and double-staired pyramids, the Songnic roofs, and Romanesque columns. Additionally, modular segments for small and large houses, semicircular arches and piers, and pagoda towers were created. The house component models were designed such that the floor and short sides of these buildings were golden rectangles while the front and back facades were able to house a round number of silver ratio bays. Pagoda segments were developed as octagonal prisms with golden rectangle outer faces. These models were then imported to Houdini for assembly and combination.

3.4.2 Supporting Tools

One of the first tools developed for the building generators was an extendable pier or arch dependent on a line. The tool detected the start and end of the line and placed respective start and end pieces for a pier or arch. Intermediate points were replaced with the modular center piece. The length of the line was altered according to the number of points designated by the user, allowing the user to control the number of completed arches using the line points attribute. A control was added to allow users to easily switch between piers and arches, using the same attribute to control the length of either tool. A simple switch is then used to decide which arcade type should be placed. A similar switch is used to control whether Doric or Corinthian columns are placed where appropriate. The same semicircular arch door is used for each building on a designated face at the center. Buildings were designed with hollow interiors, as well as usable holes in doors to allow for further development and use. The current model kit, however, does lack windows and window parameterization. Windows were ignored in favor of adding a large variety of previously established elements in a variety of configurations instead of breaking these configurations down further based on window placement. This is additionally complicated by symmetry, which was a prominent piece of design philosophy in all three architectural styles.

3.4.3 Building Parameterization

When constructing both large and small houses, the basis for wall placements is established manually using the topology of the floor. Once these placements have been established, side walls, the door wall, and the back wall are placed appropriately. The arch and pier tool was then expanded for each type of house to place the appropriate amount of repeating arches on each wall in the appropriate place according to length. In both houses the side walls receive the base number of arches, while the front wall receives one less than the base number on each side of the door. The back wall receives two additional arches compared to the sides, centered on the back wall. Columns are also procedurally placed along the front and sides of the building in user designated amounts. The pillars' total span is constrained to the size of the respective wall, and nearby placements are fused to prevent overlap. All rectangular Songnic roof styles were available for large houses, with different scales applied to accommodate for differences in size of the initial models. A subset of these roofs are made available to the small houses. Pagodas were generated using a vertical line as a basis. The top end point is designated as the placement point for the roof while all other points along the line receive wall pieces, assembled based on the floor, as well as interior roofs for each segment. Similar to the arch tool, the line lengthens with the user defined point count, allowing variable height towers based on a defined number of stories.

3.4.4 Generation and Saving

After the tools allowing the parameterized generation of each building type were completed, they were used to generate and save a wide variety of each structure. Twenty eight small houses, fifty large houses, and one hundred eight pagodas were generated using the building generation tools. These buildings were saved using a file cache node. As these were individual objects being saved, they were saved as one frame and not as a simulation. This reduces the size of the saved files and keeps them relatively light to both save and load. Naming conventions were established for each building type in order to organize them as they were saved. Naming conventions included all changeable parameters in order to ensure if duplicate buildings were saved, only one of each version of the building would be present when loading later. When buildings are loaded again after saving, they are, however, able to be referenced by simple indices. A more complex system that parses the naming convention to identify particular types of buildings could be developed and used if a project had need of it.

Single stair pyramids are used as commercial and government squares while double stair pyramids are used for religious housing and buildings. Single stair pyramids receive six buildings across the surface. Each placement point is randomly determined to either be a large house or a pagoda tower, and a random selection of previously generated buildings are placed on these slots. Additionally, an extra index value is available in the random generation to populate an empty building in one of the construction spaces. Double pyramids instead place two randomized small houses at the top of the stairs, with two placements generating in the same way as the commercial districts in the back of the pyramid square.

3.5 Building Materials

The feudal era of construction shared by the combined architectural styles influenced material choices. Songnic architecture largely used stone and wood, but also incorporated metals such as cast iron. Some pagoda structures were even built entirely from cast iron, constructed in layers. Aztec architecture featured primarily stone. Floor materials were constructed to represent stone or wood floors for structures. Wooden and stone wall materials were made, with the option of painted wood as well. A simple clay tile material was made for roofing. A thatch roof material was made for small houses. A light and dark stone material was made to represent the stone of the tecpans.

3.5.1 Material Generation

The house stone wall material uses several tile generators to create the base underlying stone shapes. These randomly oriented, randomly placed tile generators with shape variance are then stacked together and blurred to combine the shapes. Surface rock detail and variation are then added on top before final maps are constructed. The stone floor material is similarly built using a tile generator for basic shapes, but only uses one generator which is layered back over itself to reinforce the shapes after surface details have been added to the stones. Dirt and general roughness details are added to the material textures as well before final maps are built. The dark stone material used in the pyramids is constructed similarly to the rock material used in the landscape, using perlin noise as a background for scattered rock shapes. The rock shapes in the dark stone, however, are built to be more square and brick-like, unlike the round stones of the rock material. These shapes are also stacked more closely and with less overlap in order to create the sense of placed, constructed stones. The light stone material was created in Substance Painter using smart materials, combining elements of several concrete materials, fill layers, and generators using smart masks.

The wooden boards material used for both wooden floors and walls was constructed using a brick generated as the basis. The brick shapes were modified to be long horizontally with similar vertical dimensions, and were sized to create an appealing and realistic repeating board pattern. Wood grain detail is then added to the top of this material using a variety of blending operations and noise maps. The ambient occlusion is inverted in order to ensure gaps between boards remain rough and creates highlights on the raised wood details. Other maps are constructed from the height map in the typical fashion. The tile roof material uses a tile generator as its basis, and largely follows the same workflow to introduce scratch detail and rough clay surface variation. A gradient is used to alter the height of the tiles from the top of the texture to the bottom, creating a slope along it. This can later be tiled to create tiers of slopes in the material. The thatch roof material uses a similar workflow, using boards for the basis shapes before using a gradient to create a slope along it. The thatch, however, includes binding straps at the top border and frayed edges at the bottom. Gradient masking is used to create extra feathering detail on the lower edge of the thatched roof. Fibrous details are added throughout.

3.5.2 Manual Surfacing

After materials were generated, substance archive files were exported from Substance Designer and imported to Substance Painter for use in surfacing. Each piece of the model kit was manually surfaced using a combination of the generated materials and existing smart materials available in Substance Painter, namely the beech and walnut wood smart materials. Several simple metallic materials were modified as well. Light dirt and wear were generated for each texture in order to create a more imperfect and realistic final result. Several variations of each piece were created in order to allow for variation in the final surfacing. Roofs with different combinations of colored tiles and materials were generated. Walls and floors of wood and stone were generated for each structure. Columns were surfaced in stone, wood, and metal. Corinthian columns were also given sculpted, high polygon count models in order to create the crown detail. These details are transferred in the surfacing phase using texture baking. This process is more fully described in Appendix C. Several versions of the high poly model were created in order to have several variations of the Corinthian's signature top. Walls and columns were forced to maintain material consistency in the same structure, in order to maintain the overall look of the structure and ensure it remains realistic. Base color, roughness, normal, height, metallic, and ambient occlusion maps are generated for each texture.

3.6 Building Placement

After the road system is integrated with the terrain, generated and saved buildings can be placed along the roads. This process begins by having the user define which primitives in the road map correspond with four district types. These districts include commercial and government, religious, large residential, and small residential classifications. After the user defines the areas for buildings to be placed, points along the primitives in areas with a high slope are deleted. Points are then scattered nearby the roads. These dense point clouds are fused down to potential placement points. The fuse distance in this step dictates building spacing and the scattering parameters dictate how far a building can be placed from the road. Building density is driven by both of these parameters in combination. After potential building locations are identified, each point is processed. An index value is given randomly in a range dependent on the building type. This index is used to indicate which of the previously saved versions of the building will be placed in that position. The terrain is then sampled for height and mask values of the new position. Points that have been displaced into high slope areas are deleted. The remaining points are processed to find the closest road. A new normal for the point is calculated based on the closest road shape, facing the building toward the nearest road. The distance calculated in this step is then also used to remove points deemed too close to the roads. Once processing is complete, buildings are scattered onto appropriate points based on type and index. After buildings are placed, the terrain is processed to integrate the buildings using the same method as was used when integrating the roads.

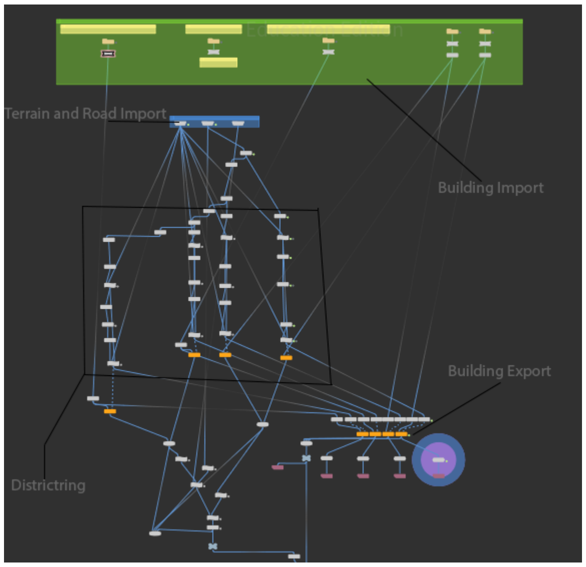


Figure 3.3: City Generation Node Graph with labels

3.7 Unreal Engine

After texture bitmaps are exported from the Substance suite, they must be compressed before being transferred to Unreal Engine, as surfacing a high number of objects with unique materials is not feasible when using six images per material. Images must then be compressed into channels, as many maps are black and white and do not require full color to maintain their information. The base color and normal maps are color, while the rest of the maps are black and white. The blue channel of the normal map can be recalculated based on the red and green, and is also therefore unnecessary. The roughness map of each texture is stored in the alpha channel of the base color map. The normal map's blue channel is used for the metallic map if the material has any metallic components, and for the height map otherwise. The ambient occlusion map is stored in the alpha channel. The alpha channels are used for the roughness and ambient occlusion maps, as maintaining subtlety in value variations is more important in these maps when compared to metallic, height and ambient occlusion. When these two constructed images are imported to the engine, the import settings must also be set correctly. Because various information is being carried on different channels, the images are imported in linear color space with appropriate compression. Color space is defined for this context in Appendix C. This maintains all channels and prevents values from being altered significantly by the compression algorithm. An unpacking material function was created in unreal to quickly create materials from these images. The 'unpacking' process involves establishing 2 vectors with a depth of four as the inputs, representing the two combined texture images with their four channels. Most maps are simply sent out to appropriate outputs using channel masking to extract the original images. The normal map, however, must be recalculated and adjusted because it has been imported without a blue channel and at a value range of zero to one. To correct this, the red and green values are multiplied by two, and then have one subtracted. This appropriately remaps the normal values to a range of negative one to one. The blue channel is then calculated from the corrected values using the DeriveNormalZ node.

The building elements are exported from Houdini using ROP Geometry Output nodes. These nodes in Houdini allow objects to be exported as .obj files. The buildings are exported as four files, one for each district type. This both lowers the overall size of the object to be imported to Unreal and allows some control over material application. While both Houdini and Unreal engine use meters as a base unit, and therefore match in scale, the .obj export file saved units in centimeters. Additionally, Houdini uses a right handed, Y-up coordinate system while Unreal uses a right handed, Z-up coordinate system. Because of these discrepancies, the objects are imported at 100 times scale to adjust for the unit change and rotated to match the new coordinate system. The objects are then placed and aligned with their original position manually, but all objects maintain their relative distances and orientations. Because the scale of the height field was altered slightly by changing its resolution for Unreal import, objects were resized slightly to fit their original placement. The building component objects were imported as instances per import, so all objects of the same type from the same district share a material. Several materials for each piece were created, and a unique material for each piece in each district was applied. The materials were selected and applied manually, allowing the artist to maintain aesthetic color combinations and associations. This also ensures material cohesion across a single building and is simple to achieve. This approach is limiting, however, in the overall granularity of the material application. Other elements of the world space were then adjusted to better create an aesthetically appealing landscape. The default exponential height fog used in Unreal terrain was adjusted for the scale of the terrain. The sky light was then adjusted, moving to a slightly warmer yellow light to mimic sunlight. The sky atmosphere was adjusted to control how light both transports and scatters through the space. These settings have a strong influence on how far and how clearly distant landscape elements can be seen.

Chapter 4

Production

The production chapter focuses on displaying completed work based on the design described in the previous chapter. It presents the work in the order it is processed in the system.

4.1 Terrain

Terrain images detail the terrain generation process, including individual tile generation, tile combination, and final generation. The following four images are example tiles for each of the four biome types used in terrain generation.

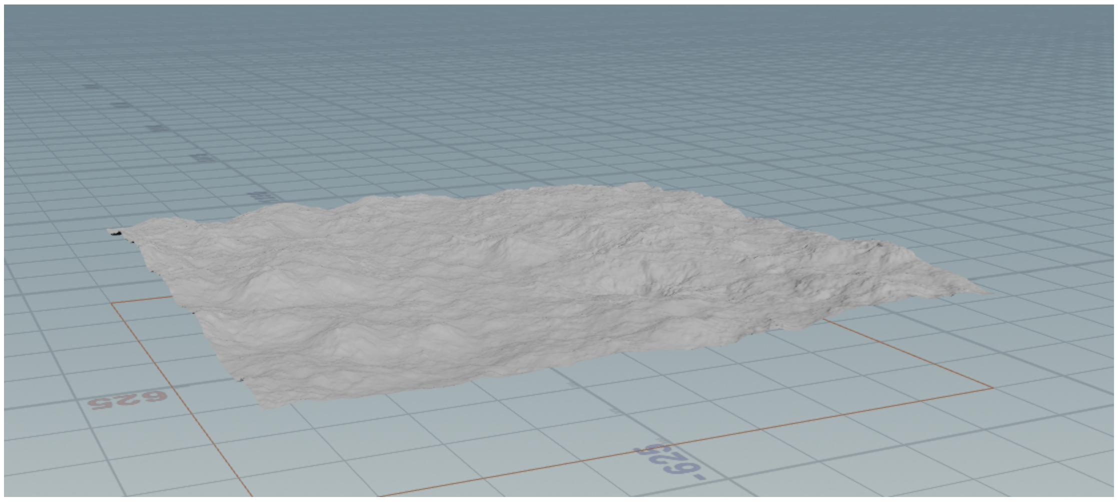


Figure 4.1: Example 'Plains' biome tile

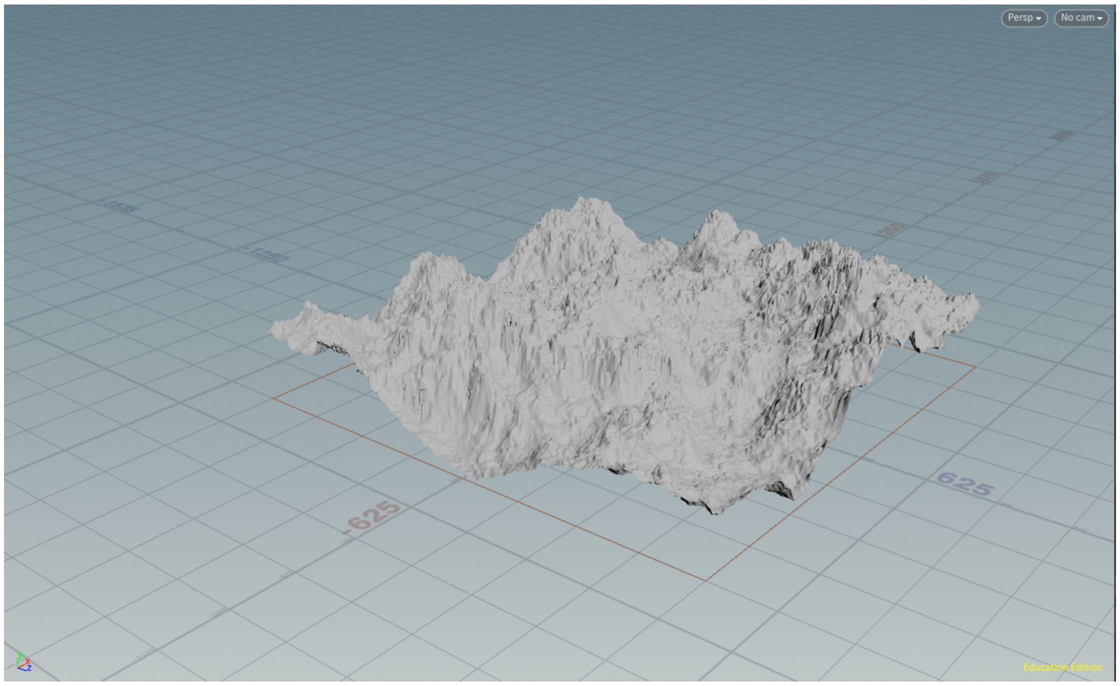


Figure 4.2: Example 'Hills' biome tile

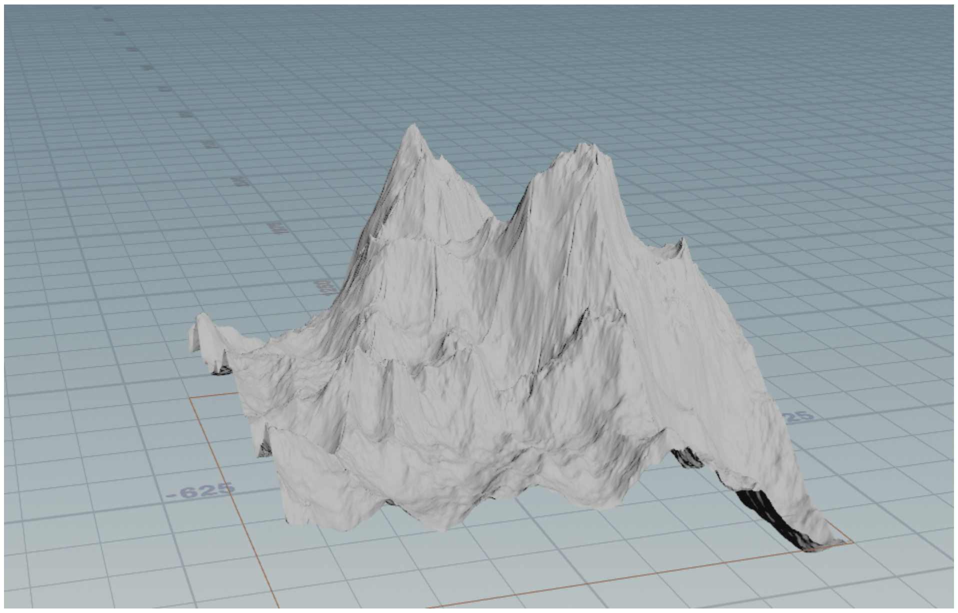


Figure 4.3: Example 'Mountain' biome tile

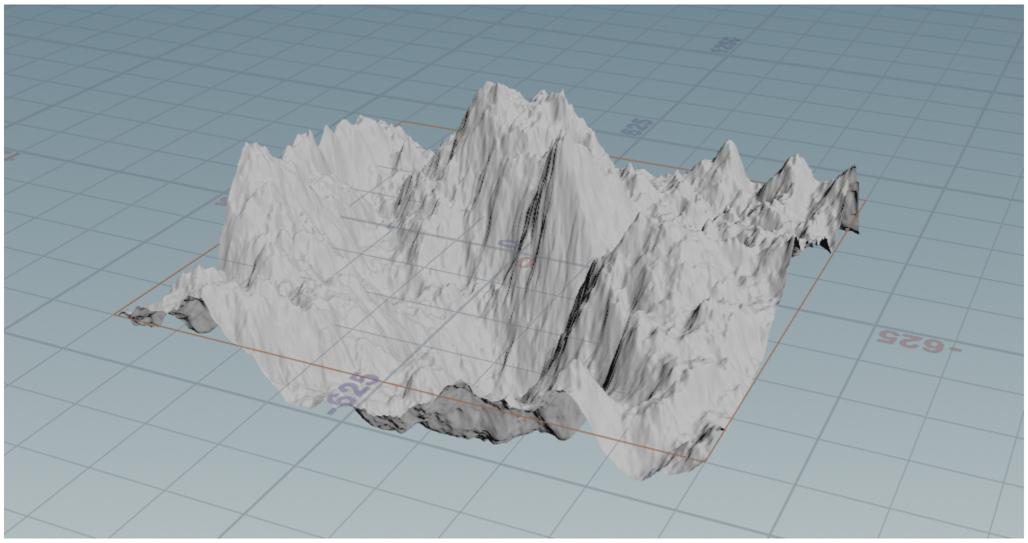


Figure 4.4: Example 'Valley' biome tile

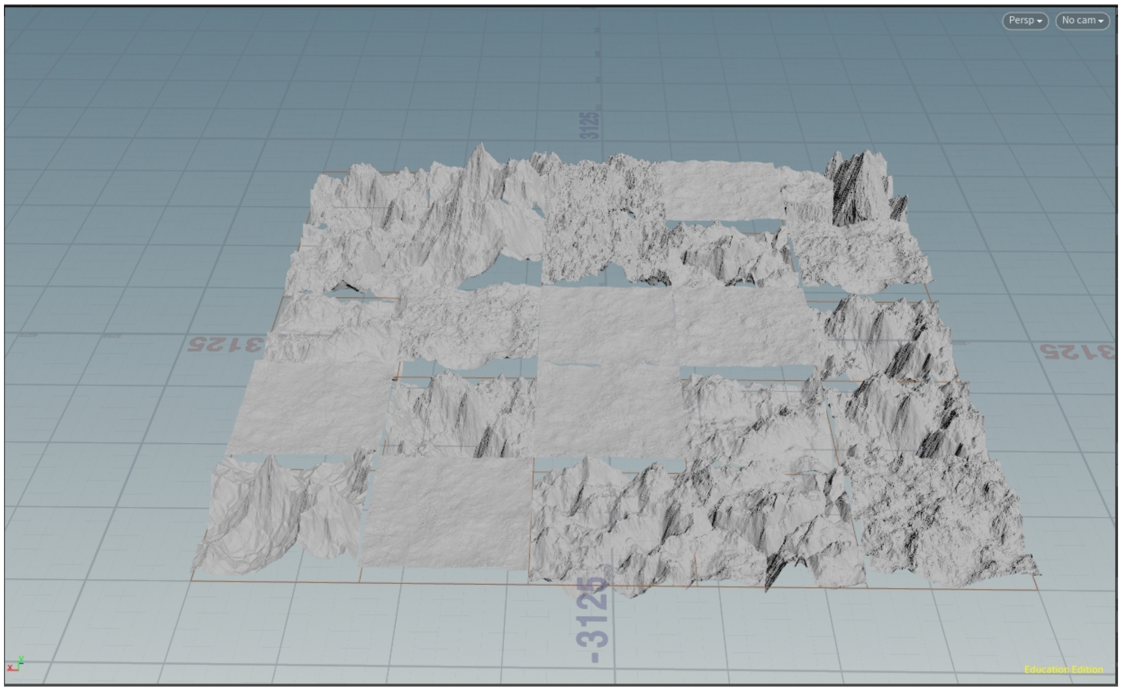


Figure 4.5: Terrain Tile Generation

This image displays assembled tile generations that have not yet been combined for processing.

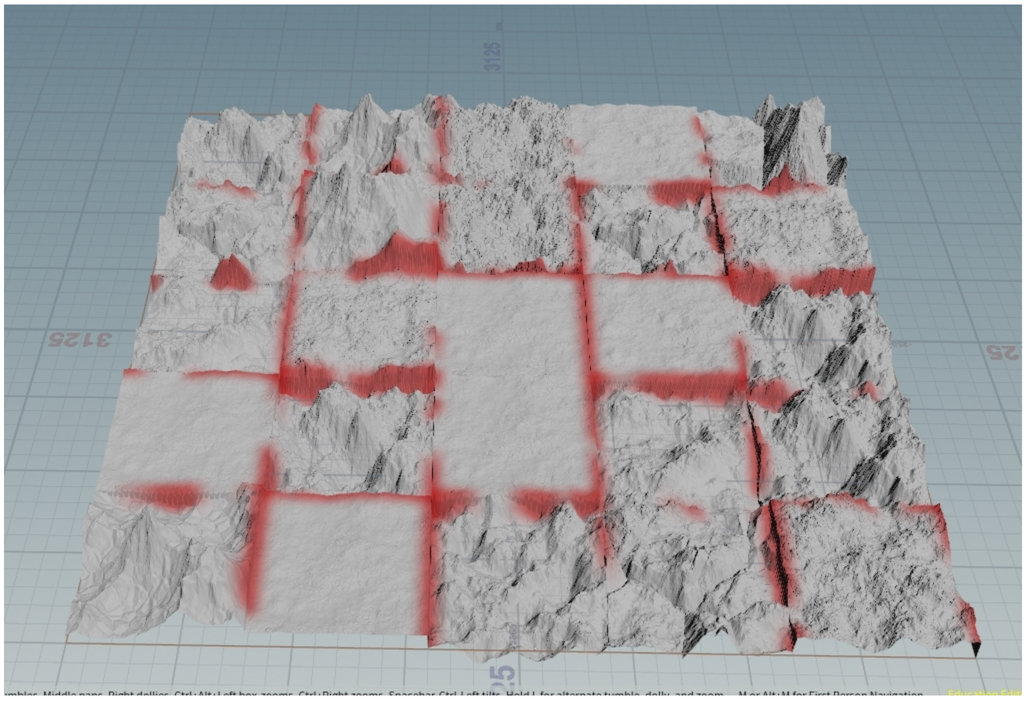


Figure 4.6: Processing to identify tiling artifacts at borders

The final images of this section demonstrate the anti-tiling technique used when creating the final terrain from the initial tile generation.

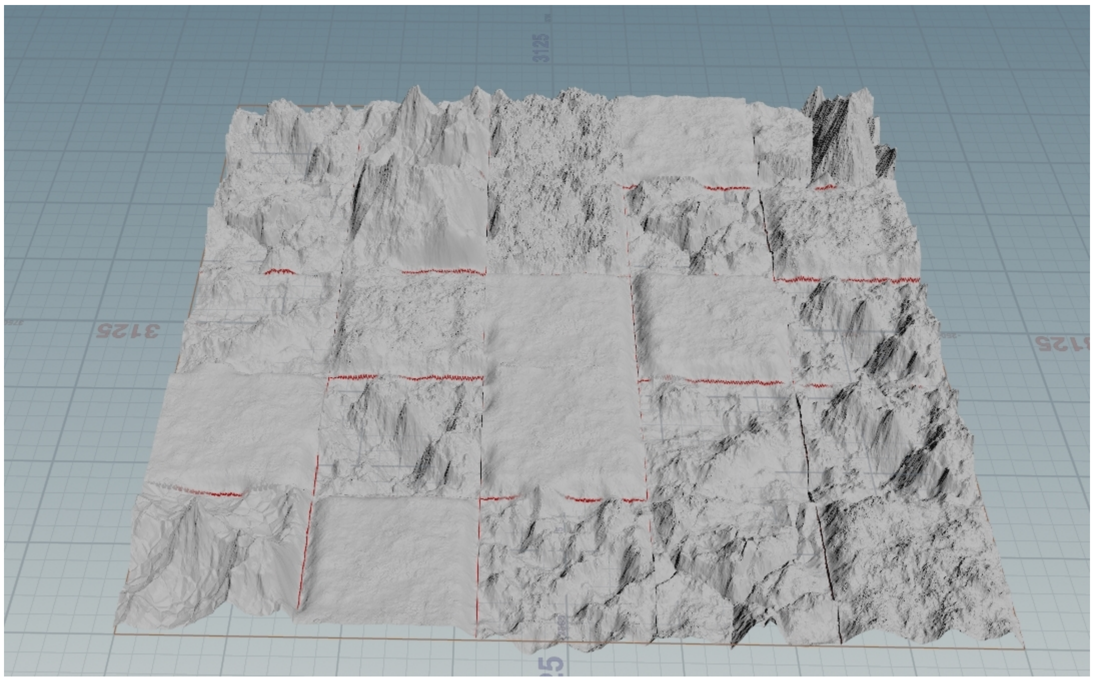


Figure 4.7: Processing correcting tiling artifacts at borders

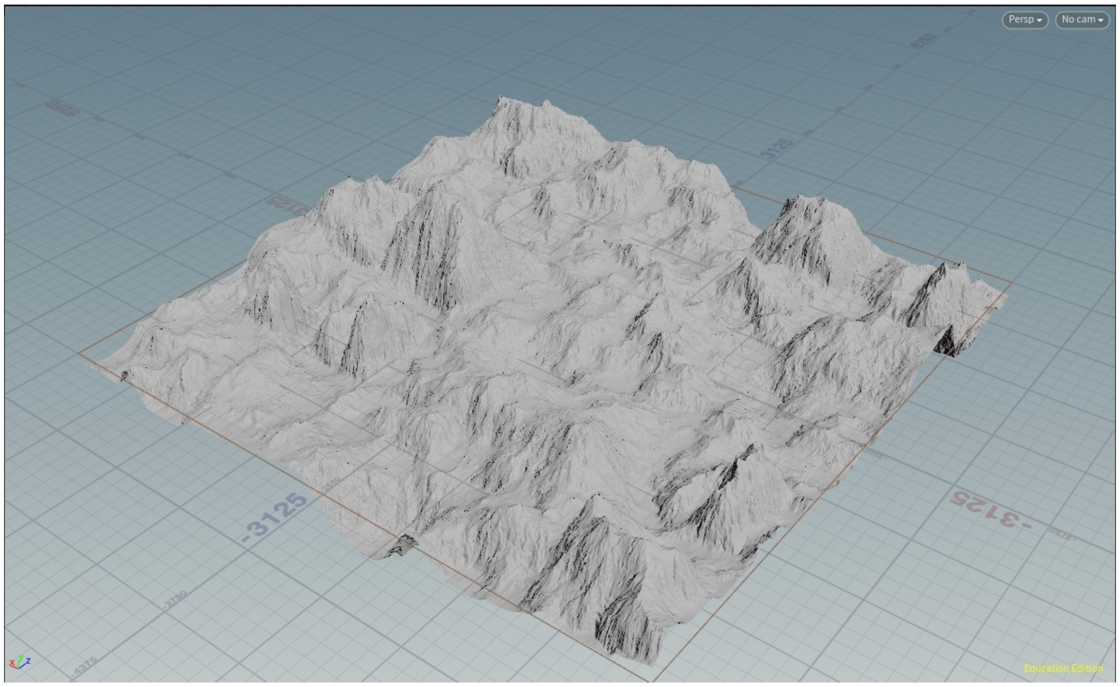


Figure 4.8: Example of a finalized initial terrain generation

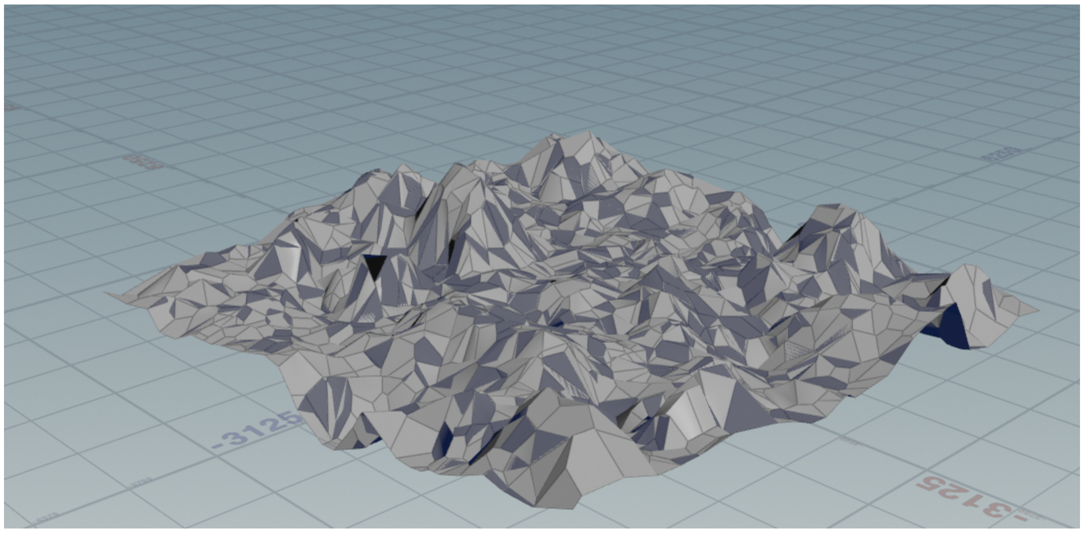


Figure 4.9: Low resolution terrain used as basis for road generation

4.2 Roads

Road images detail initial road generation, procedural and manual editing, and integration with the terrain. The first images of the section demonstrate the low resolution proxy version of the terrain and the initial road forms generated from it.

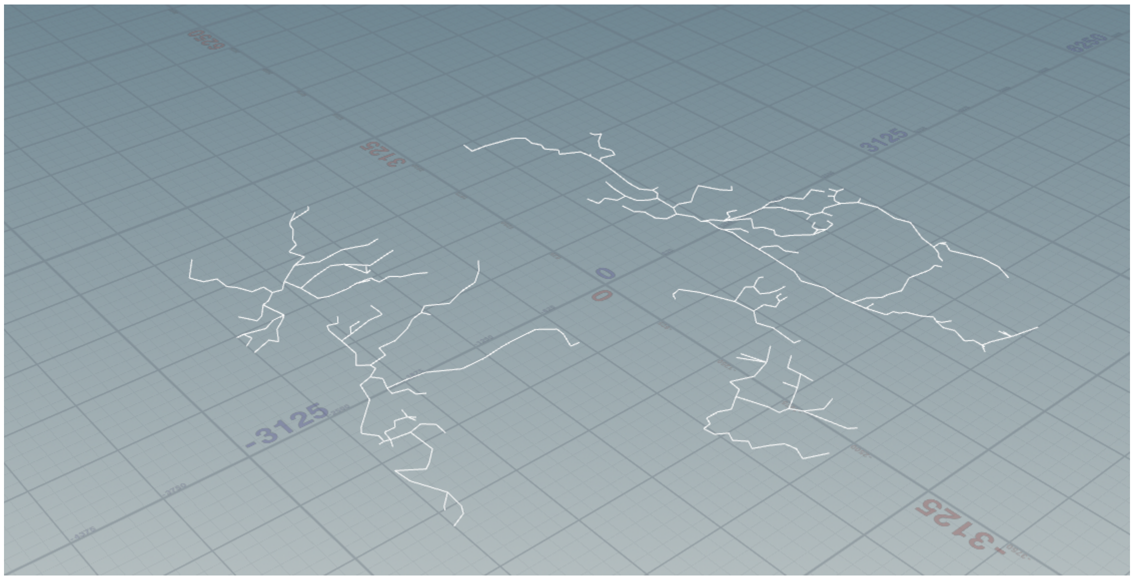


Figure 4.10: Initial road generation using organic shapes

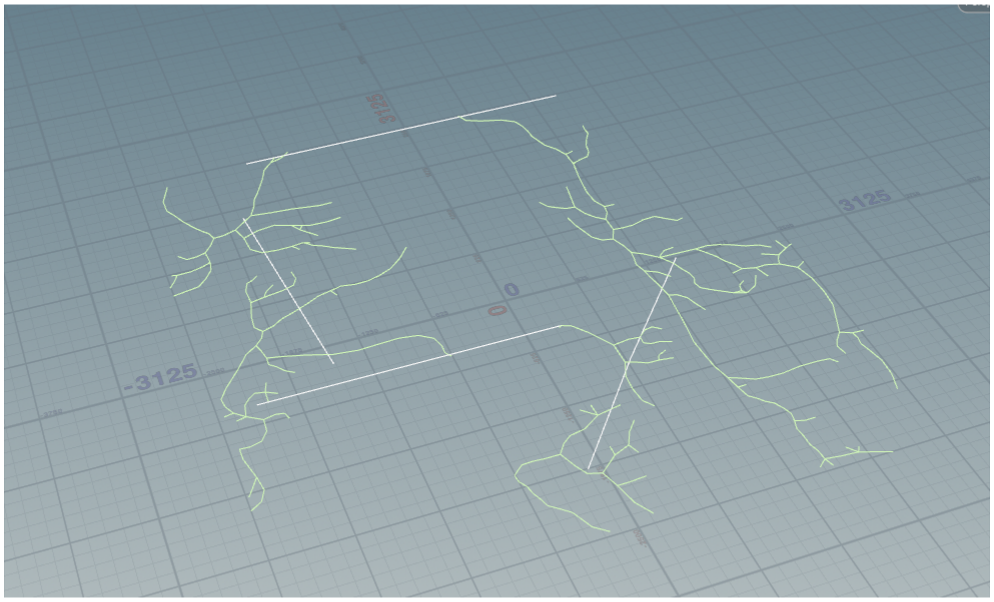


Figure 4.11: Initial road forms with both procedural and automatic connections

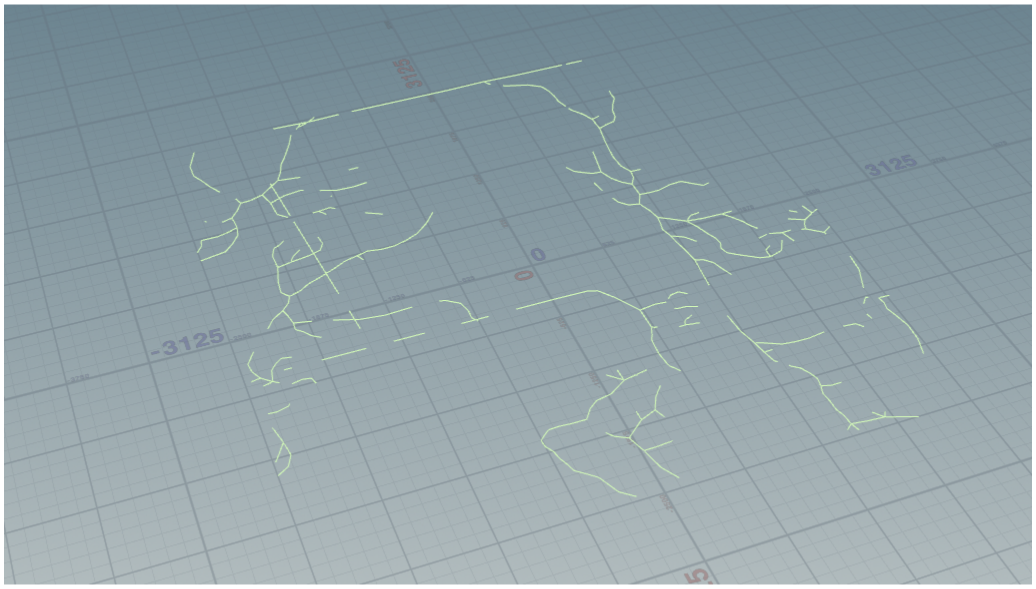


Figure 4.12: Flagging and removal of areas identified for switchback pathing

Figures 4.12-4.15 demonstrate the high slope area processing performed on the roads, beginning with the removal of flagged sections. Grids for localized path tracing are then scattered and processed.

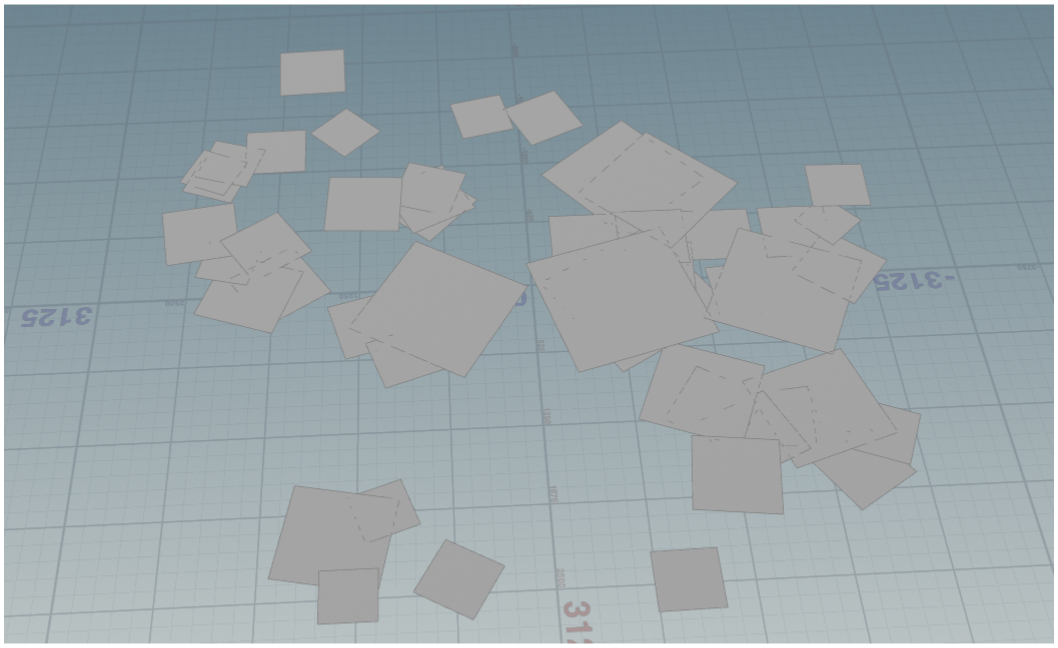


Figure 4.13: Primitives scattered over switchback locations

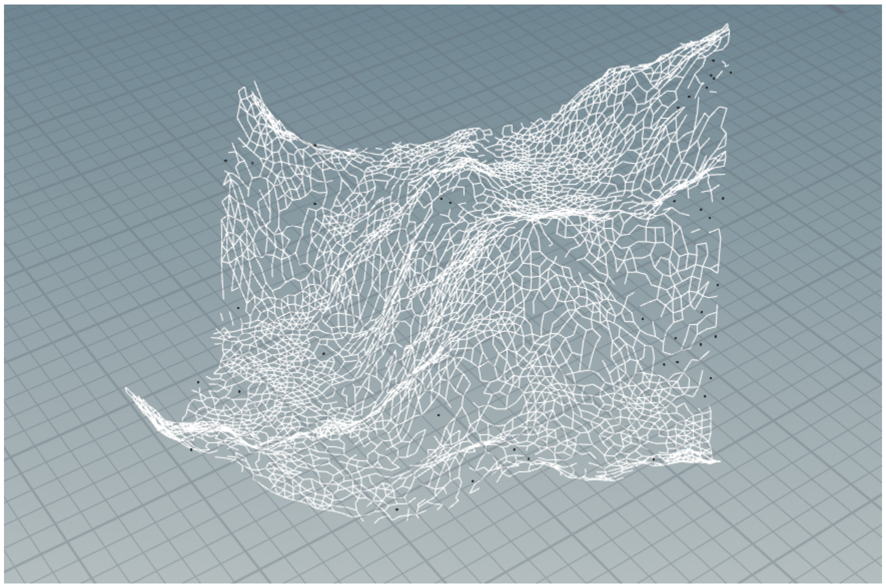


Figure 4.14: Example of localized terrain used for switchback path tracing

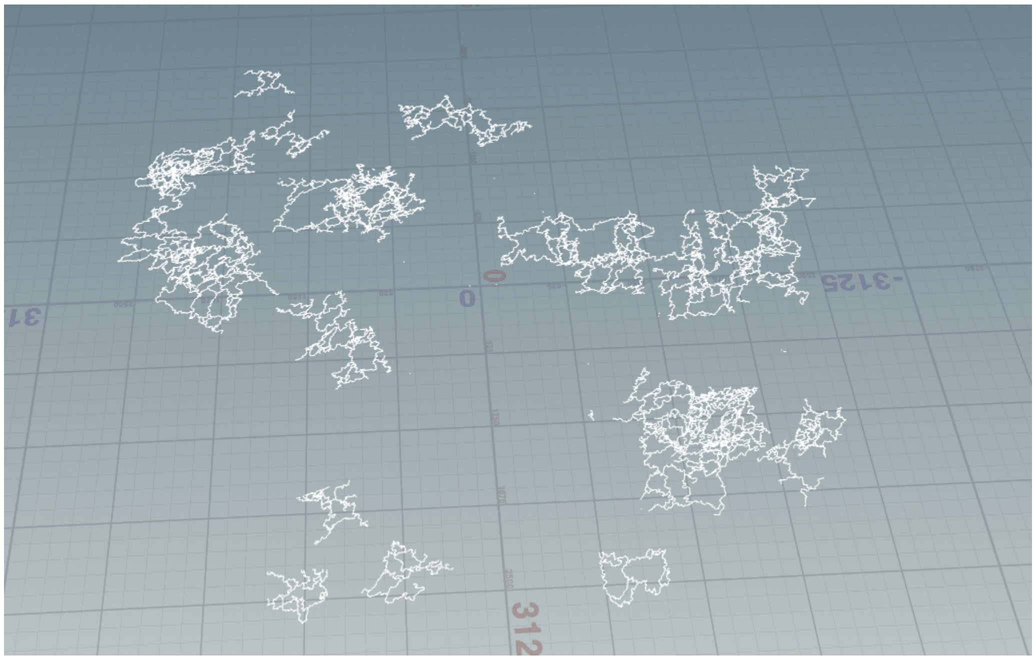


Figure 4.15: Localized switchback processing complete

The naive and contextual editing states are shown below in order to better communicate the need for contextual editing.

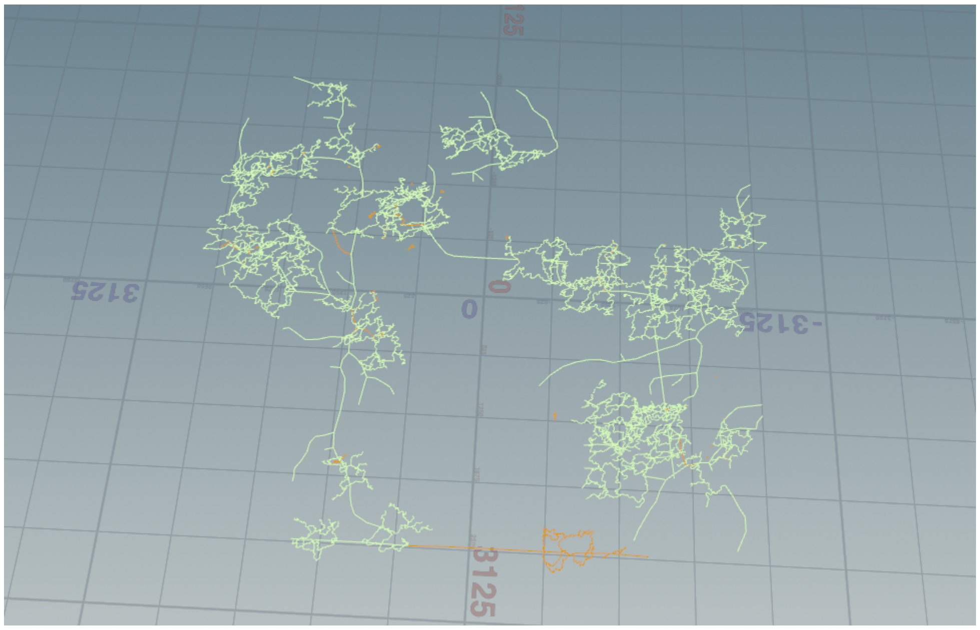


Figure 4.16: Naive editing stage for manual road editing

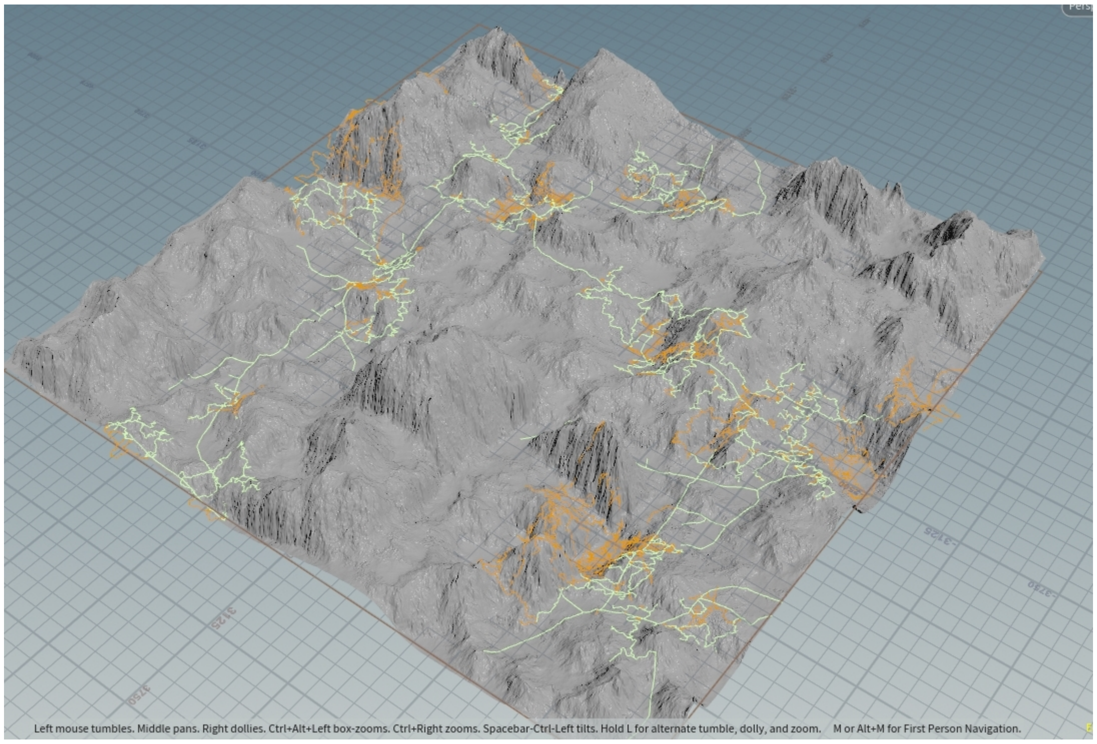

Left mouse tumbles. Middle pans. Right dollies. Ctrl+Alt+Left box-zooms. Ctrl+Right zooms. Spacebar-Ctrl-Left tilts. Hold L for alternate tumble, dolly, and zoom. M or Alt+M for First Person Navigation.

Figure 4.17: Contextual editing stage for manual road editing

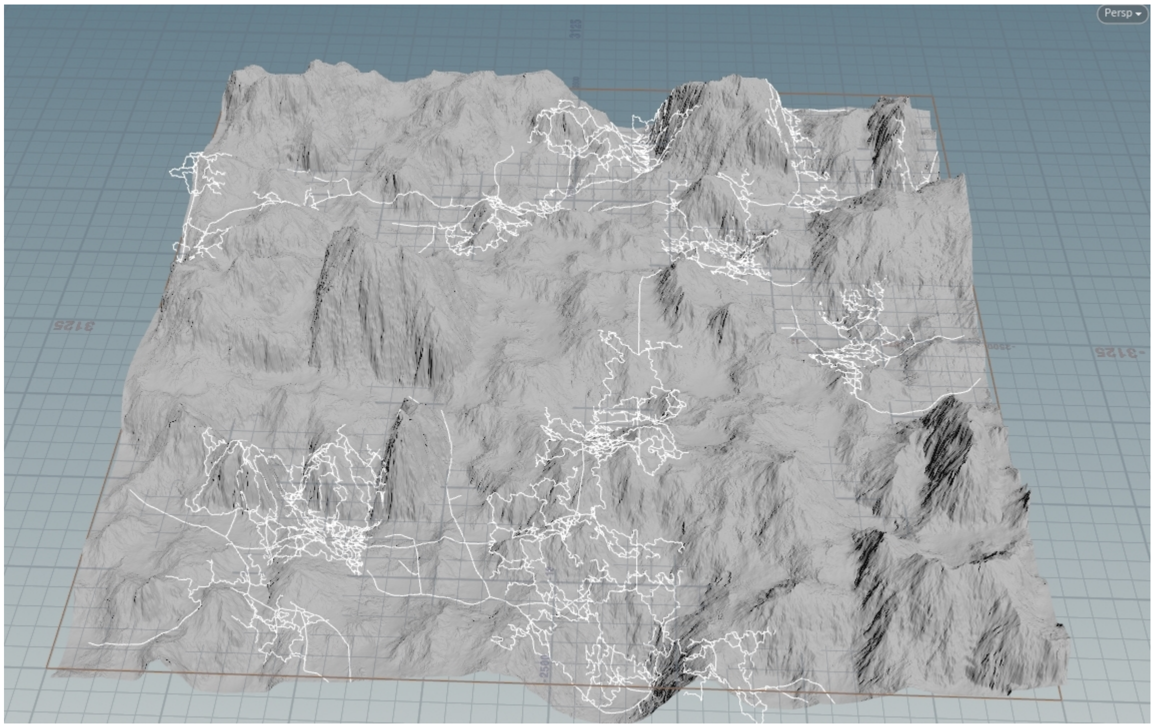


Figure 4.18: Road/terrain integration

The final images of the section demonstrate the process of integrating the road into the terrain, displaying it both with and without supportive displays.

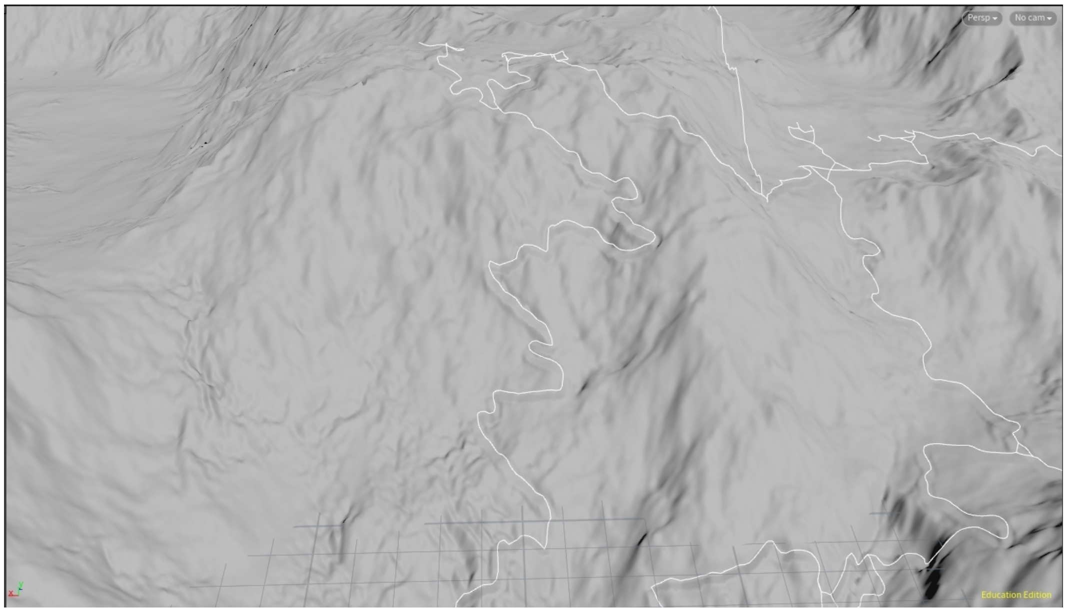


Figure 4.19: A section of integrated road with a road curve display

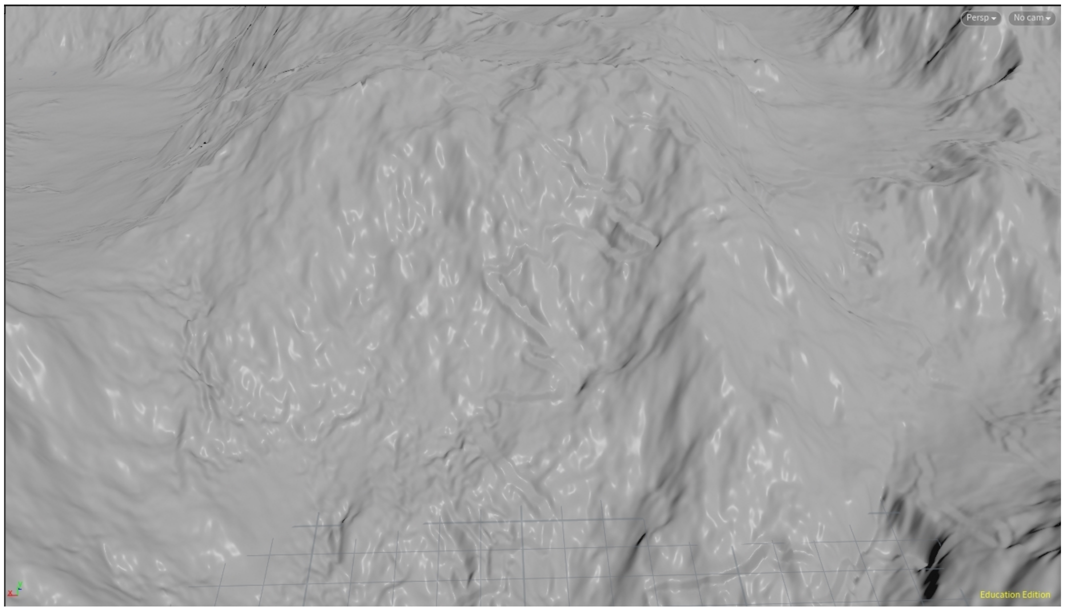


Figure 4.20: The same section of integrated road as Figure 5.18 with no display

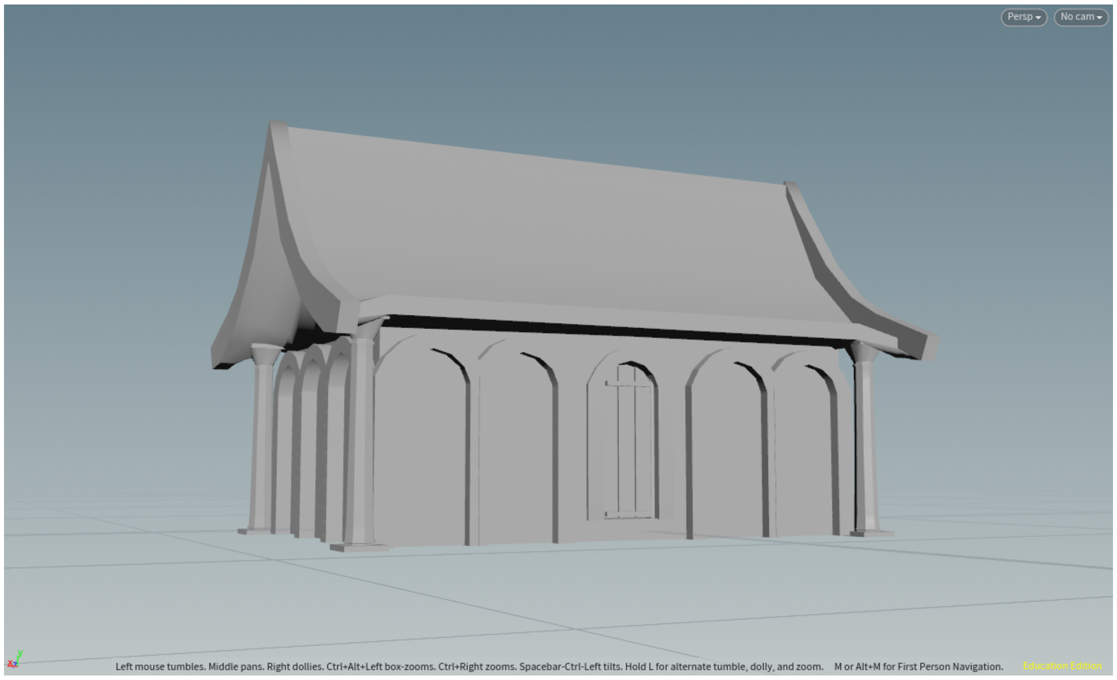
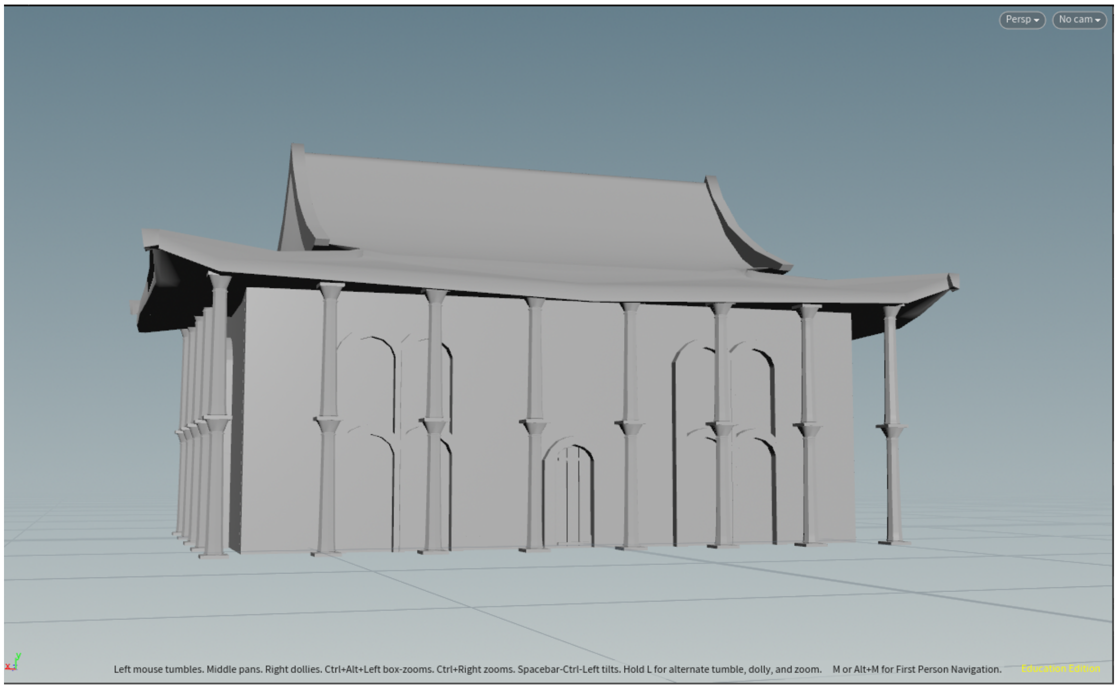

Persp
No cam
Left mouse tumbles. Middle pans. Right dollies. Ctrl+Alt+Left box-zooms. Ctrl+Right zooms. Spacebar-Ctrl-Left tilts. Hold L for alternate tumble, dolly, and zoom. M or Alt+M for First Person Navigation.
Education Edition

Figure 4.21: Example small house generation

4.3 Buildings

4.3.1 Generated Buildings

This section gives example generations for each building generator produced.



Left mouse tumbles. Middle pans. Right dollies. Ctrl+Alt+Left box-zooms. Ctrl+Right zooms. Spacebar-Ctrl-Left tilts. Hold L for alternate tumble, dolly, and zoom. M or Alt+M for First Person Navigation.

Figure 4.22: Example large house generation

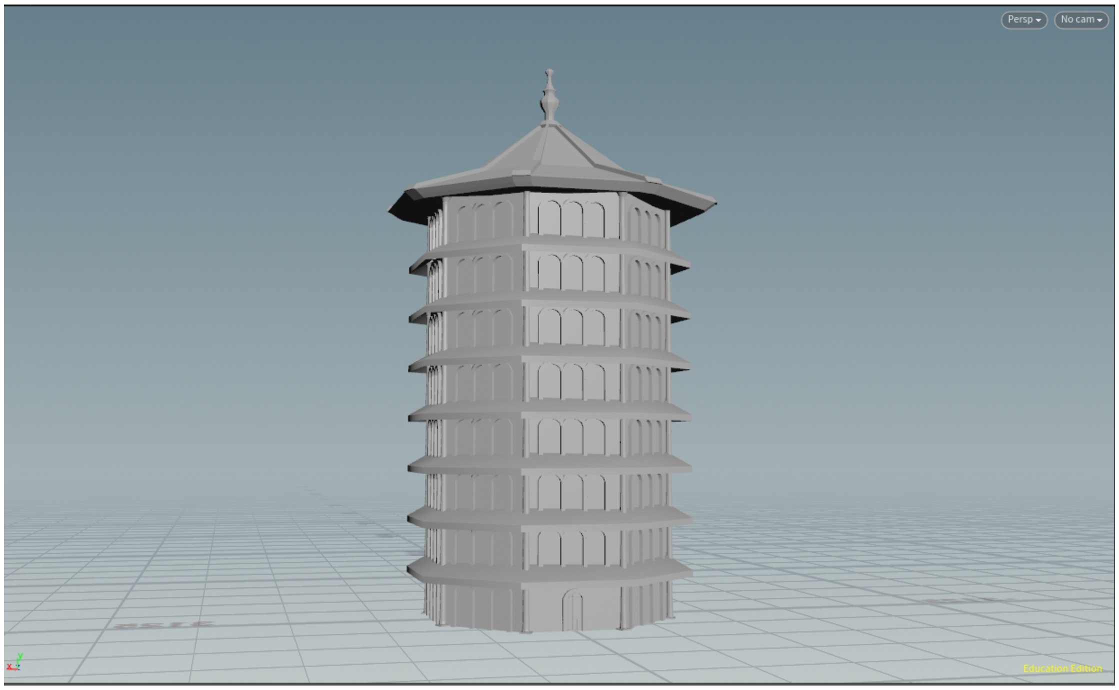


Figure 4.23: Example pagoda generation

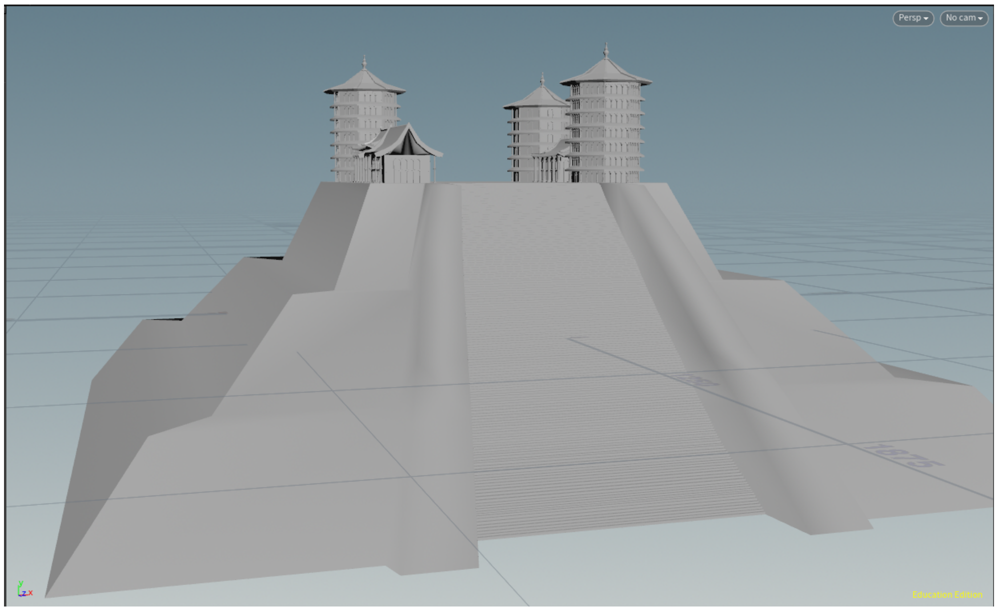


Figure 4.24: Example singe stair tecpan generation

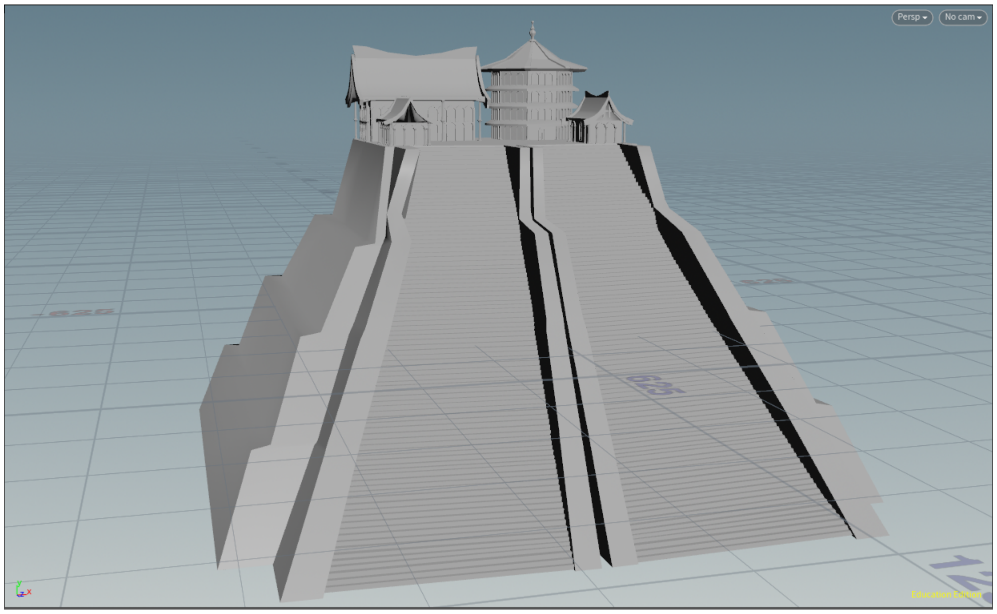


Figure 4.25: Example double stair tecpan generation

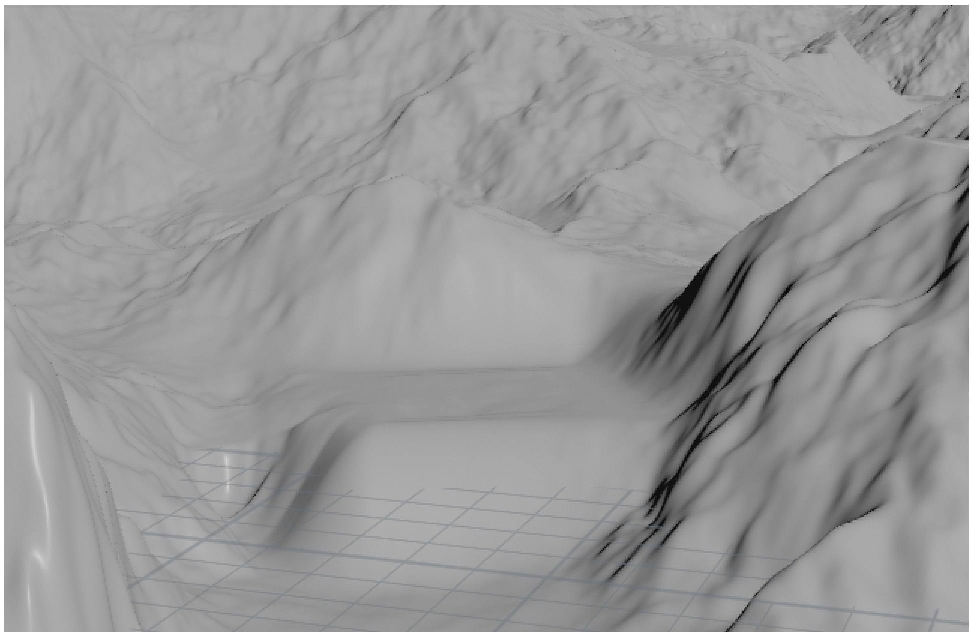


Figure 4.26: Height-field integration for building placement

4.4 Placement

The following images show visual representations of the integration process for buildings.

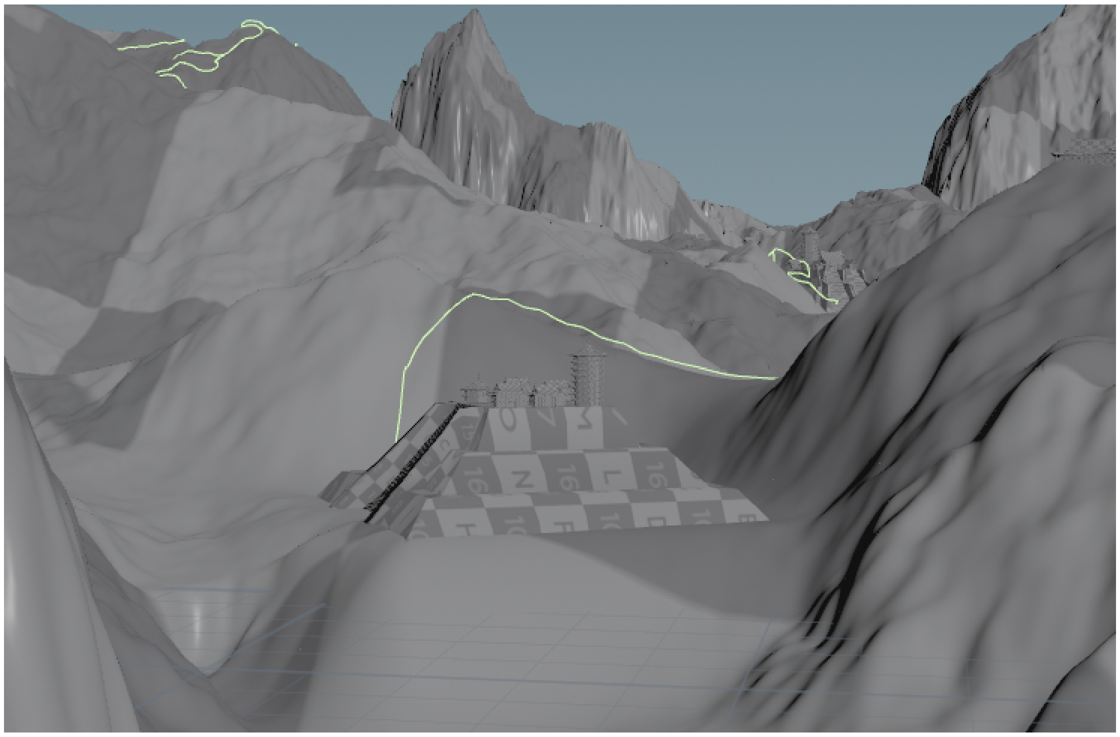


Figure 4.27: Height-field with buildings placed and a curve road display

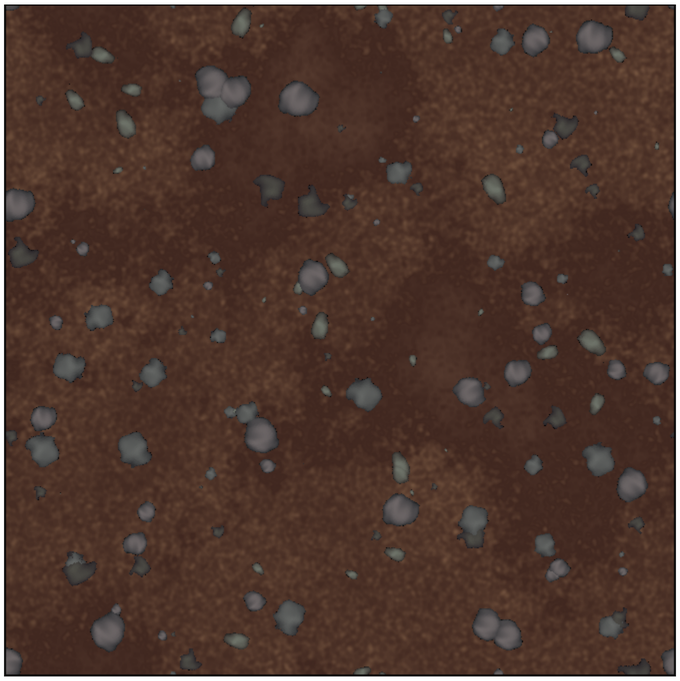


Figure 4.28: Coarse dirt color map

4.5 Materials

The materials section displays materials created in Substance Designer for the project.

4.5.1 Landscape Materials

Materials used in the landscape shader are displayed in their final state from Substance Designer, both as a color map image and as a rendered material applied to a plane.

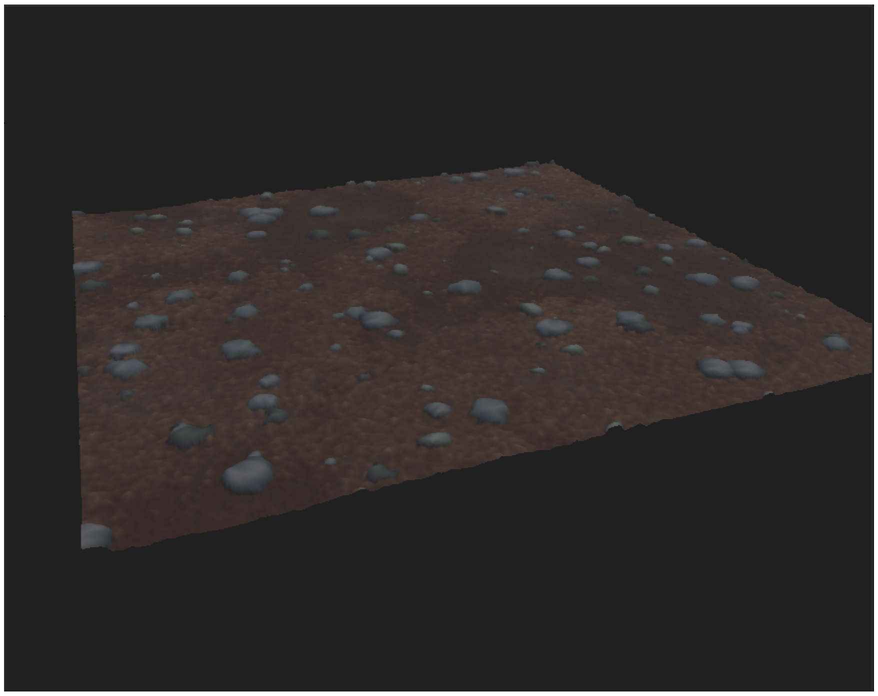


Figure 4.29: Coarse dirt material plane

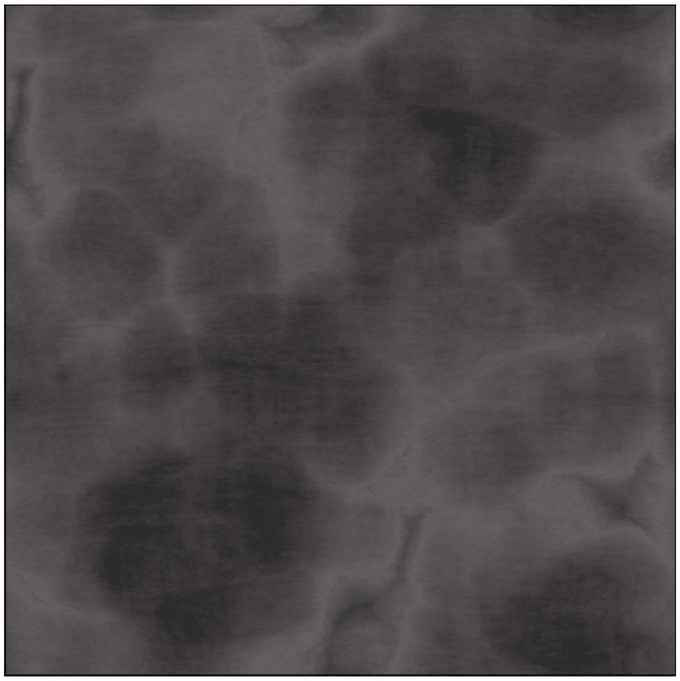


Figure 4.30: Rock color map

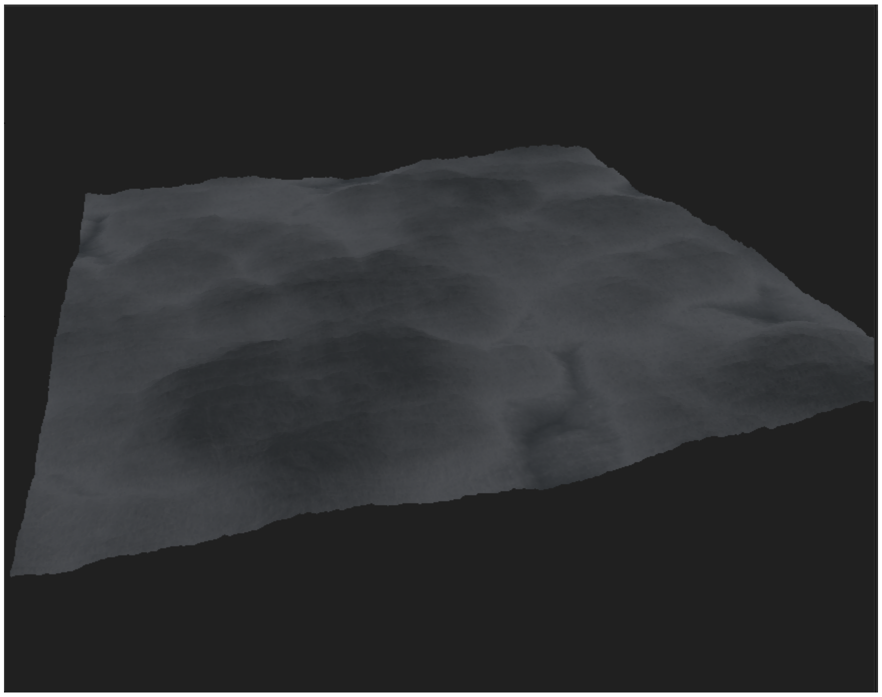


Figure 4.31: Rock material plane

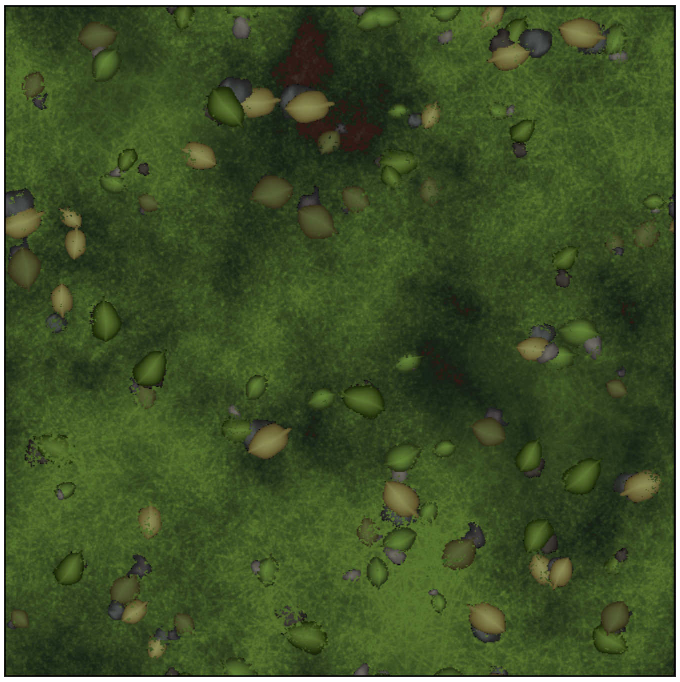


Figure 4.32: Sparse grass color map

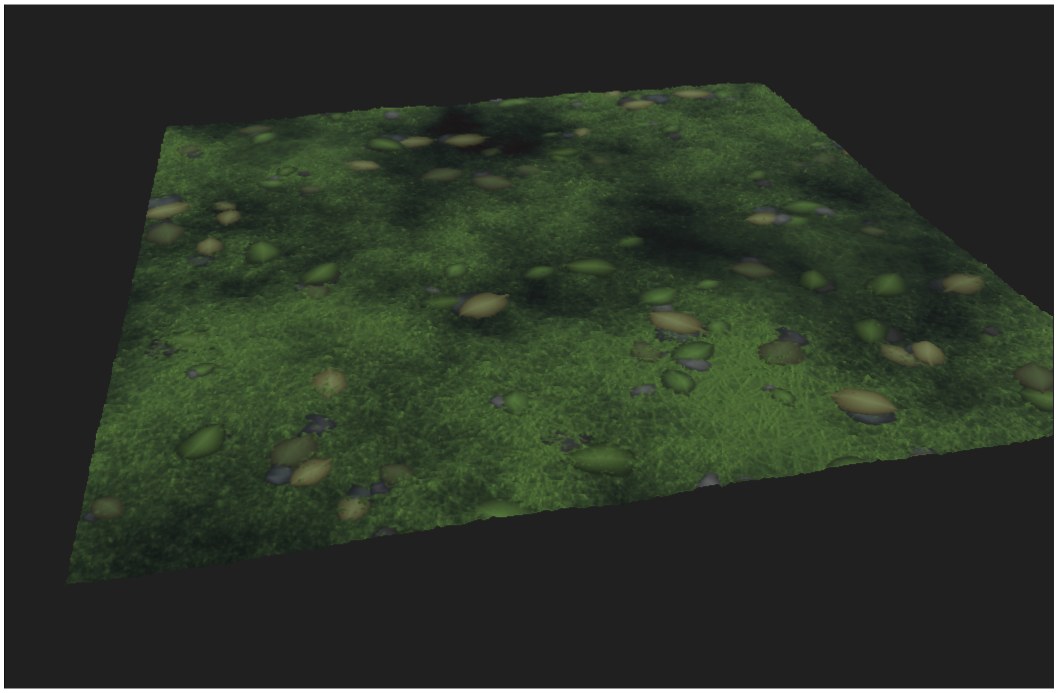


Figure 4.33: Sparse grass material plane

4.5.2 Building Materials

The first several materials are shown with their graphs in order to give a basic visual representation of the creation process. More detailed material graph views are available in appendix B. Subsequent images are represented in the same manner as the landscape materials.

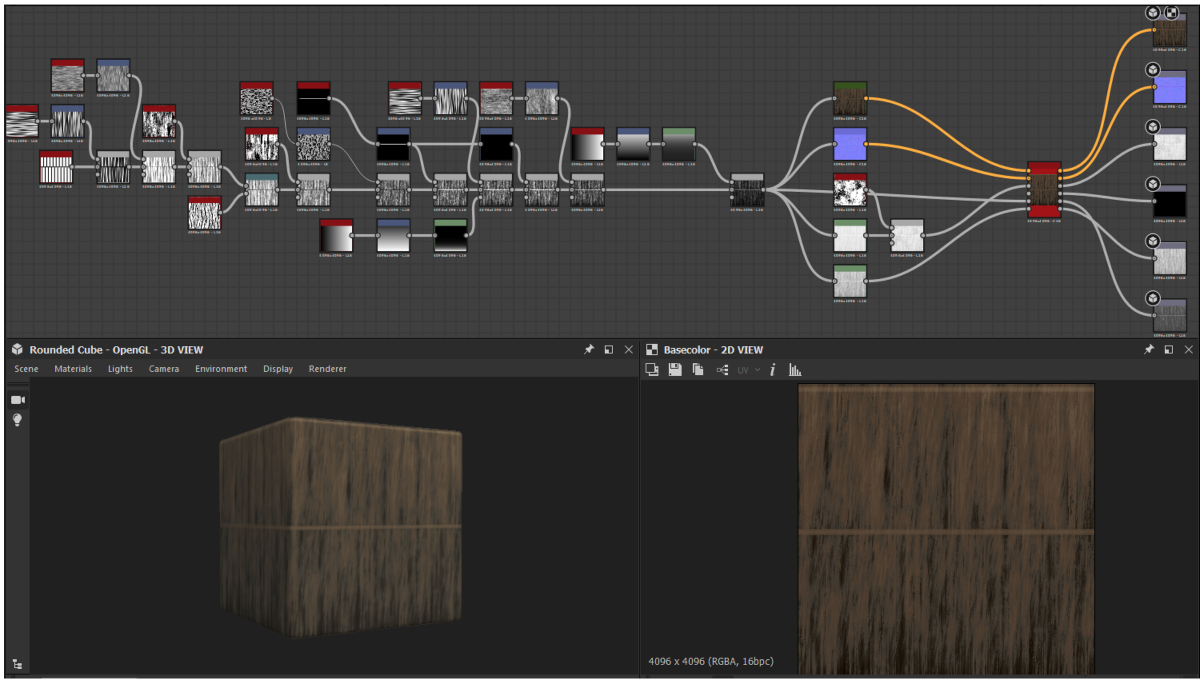


Figure 4.34: Thatch roof material graph

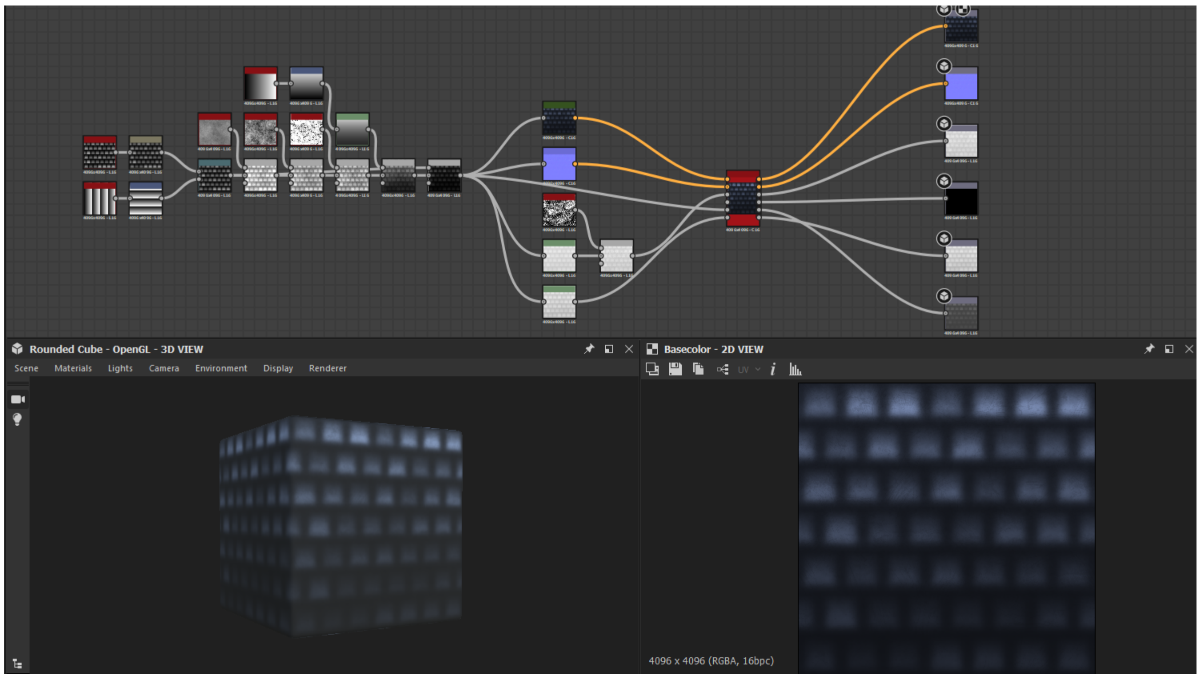


Figure 4.35: Blue tile roof material graph

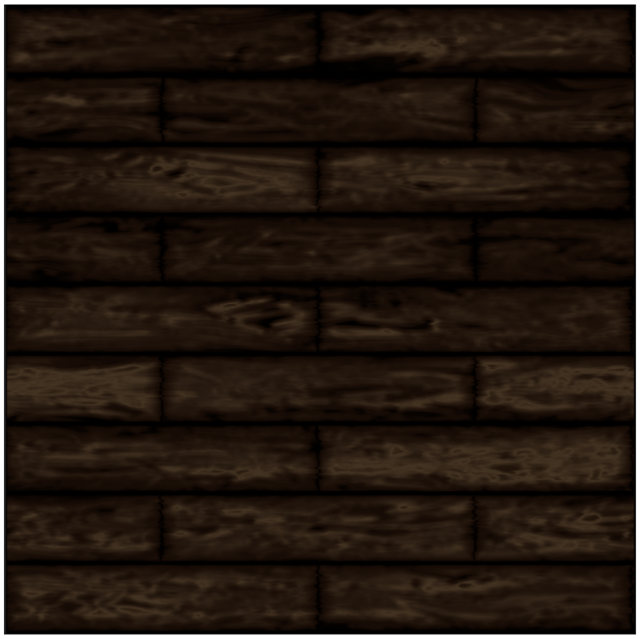


Figure 4.36: Wooden board map

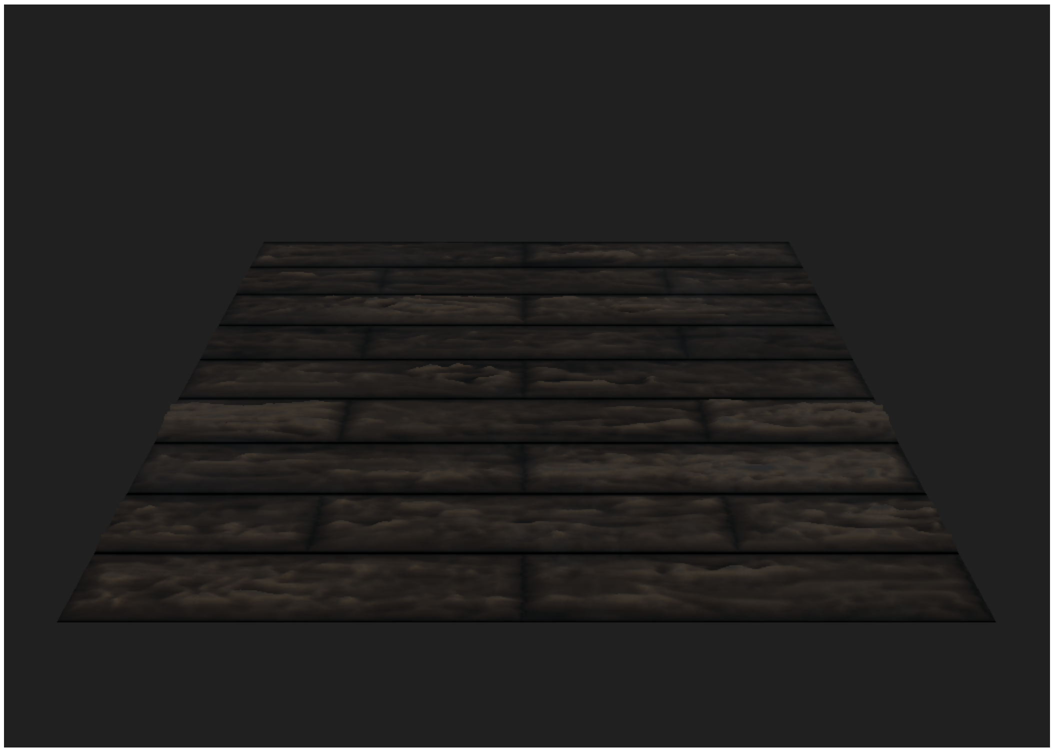


Figure 4.37: Wooden board material plane

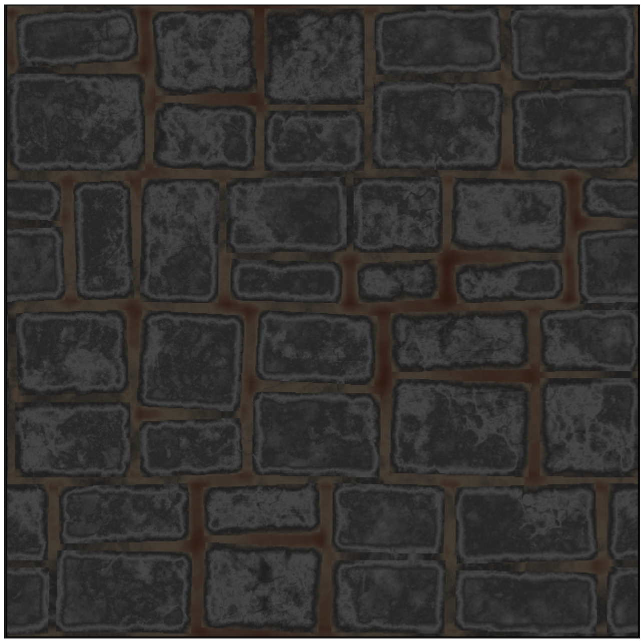


Figure 4.38: Stone floor map

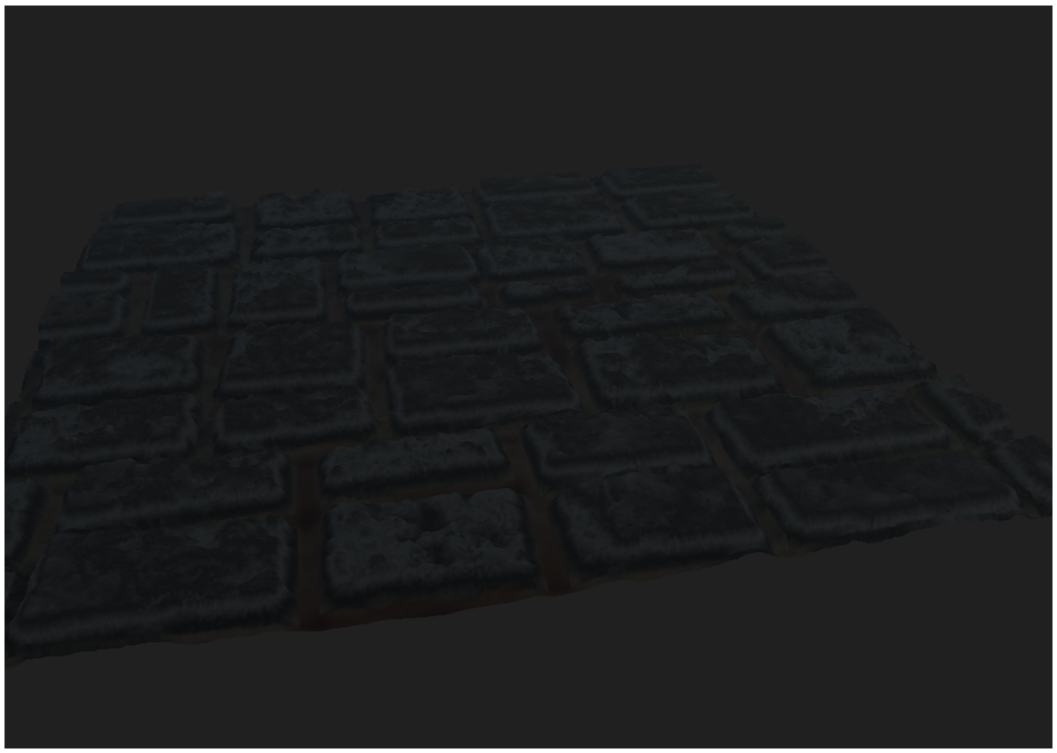


Figure 4.39: Stone floor material plane

Chapter 5

Results

Images in this section depict example city generations rendered in Unreal Engine 5. The images are all created from the same city generation output and depict different regions of the area created. The final image visually displays the transition between different types of textures and anti-tiling techniques used in the landscape shader.

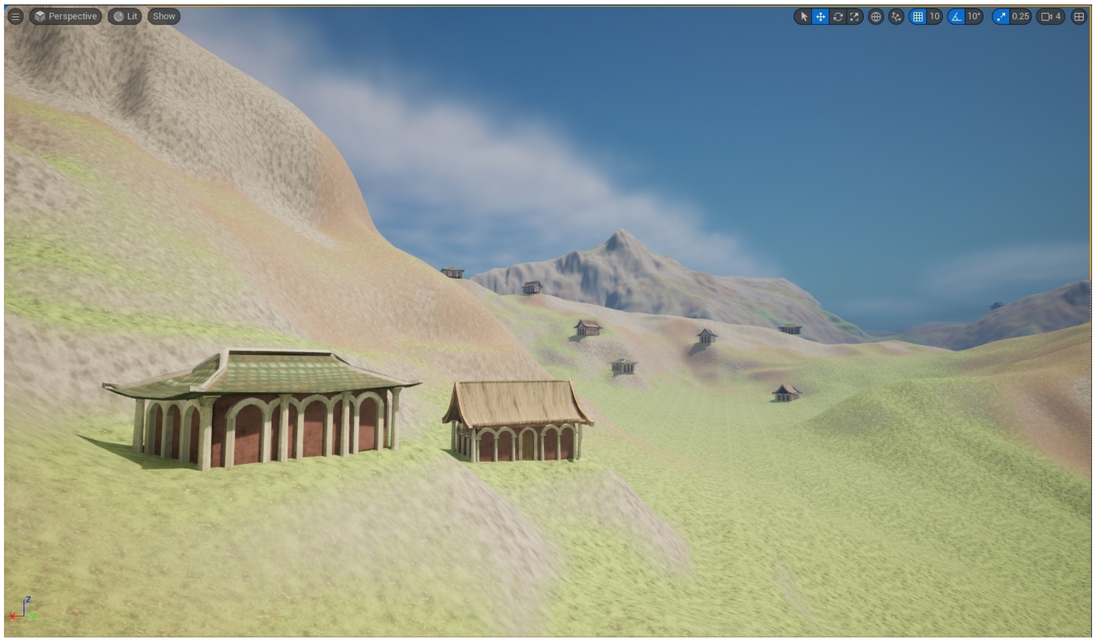


Figure 5.1: Example section of final city generation with materials, rendered in Unreal Engine

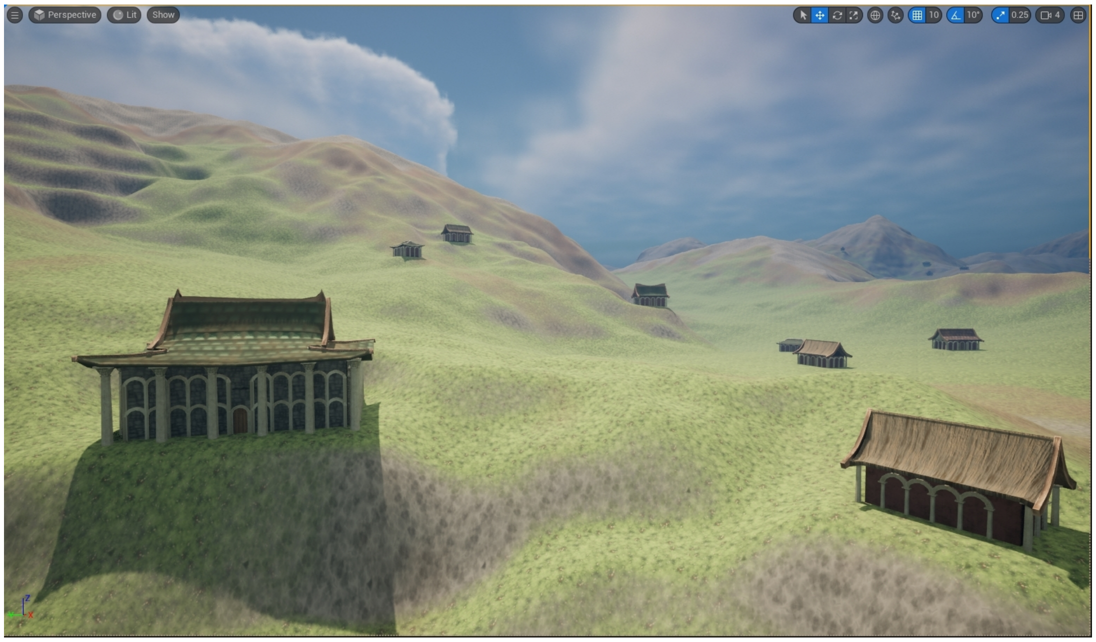


Figure 5.2: Example section of final city generation with materials, rendered in Unreal Engine

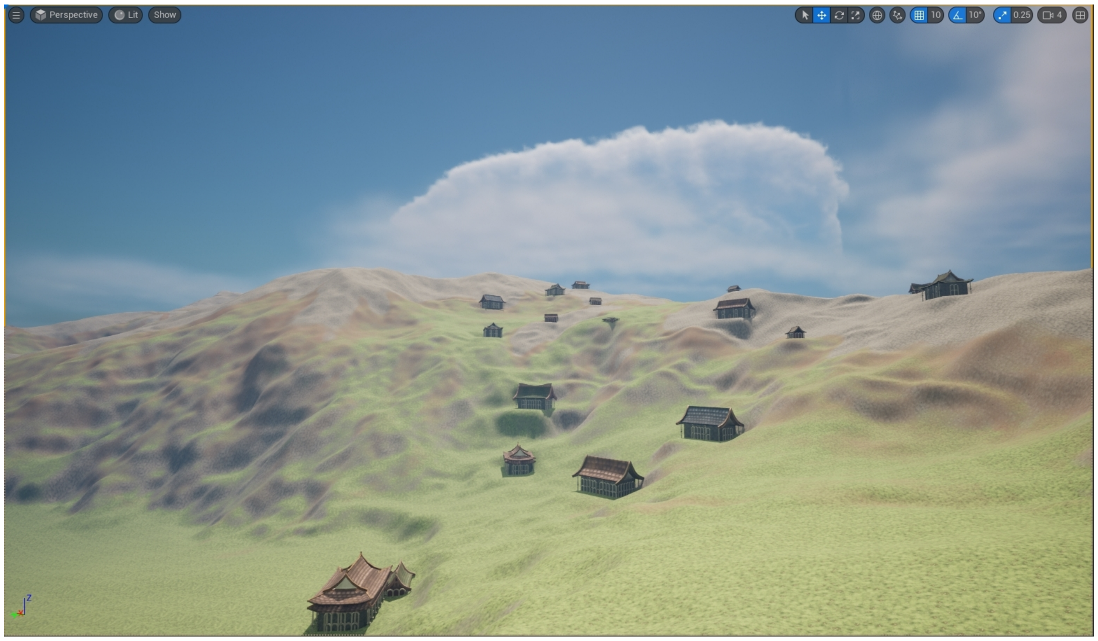


Figure 5.3: Example section of final city generation with materials, rendered in Unreal Engine

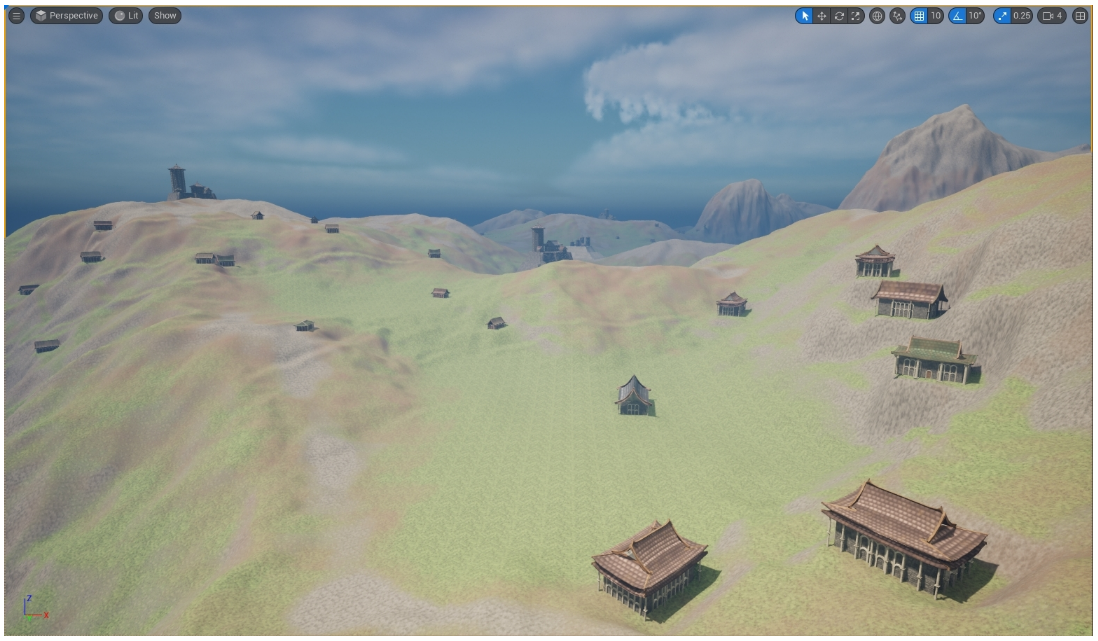


Figure 5.4: Example section of final city generation with materials, rendered in Unreal Engine

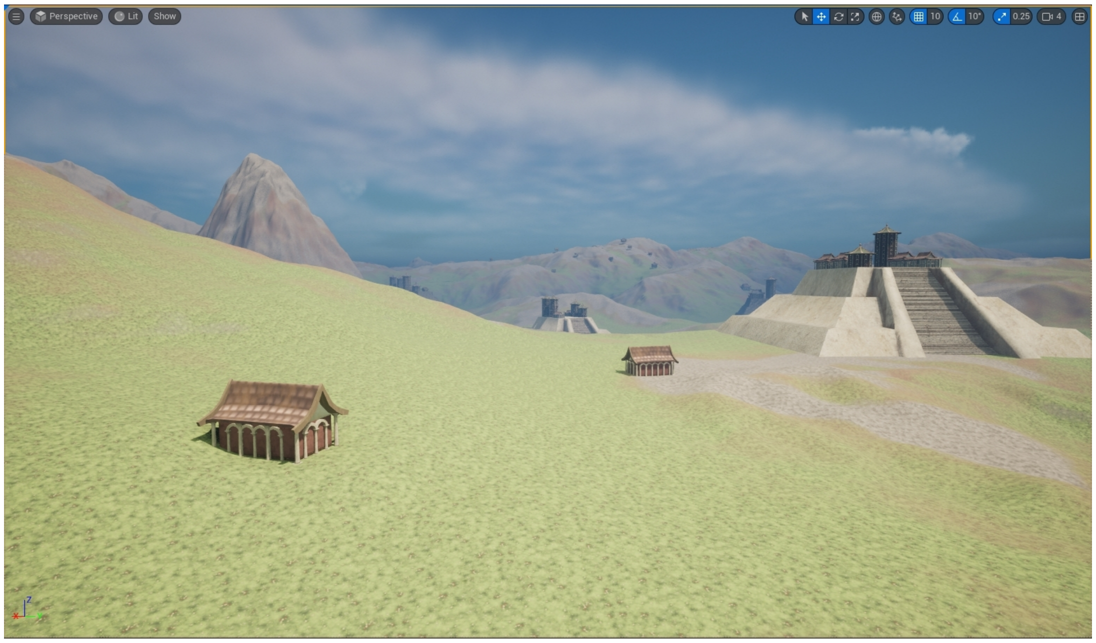


Figure 5.5: Example section of final city generation with materials, rendered in Unreal Engine

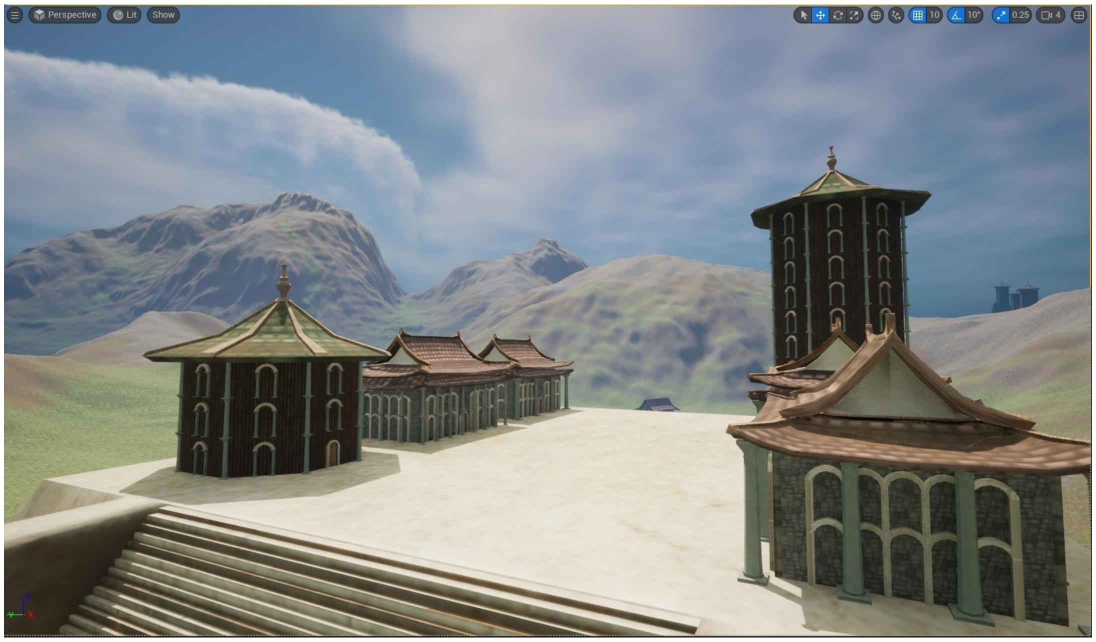


Figure 5.6: A close up of a surfaced single stair tecpan and its procedurally generated buildings, surfaced and rendered in Unreal Engine

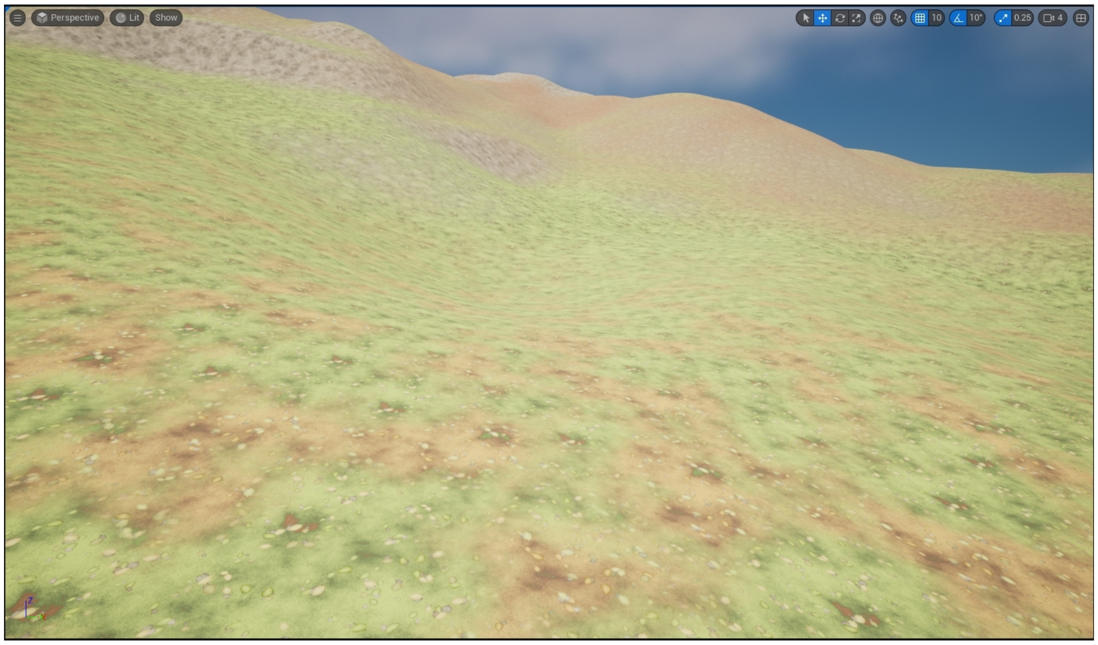


Figure 5.7: Image displaying the anti tiling techniques in the landscape shader. Swizzled and blended textures are shown up close, with a camera based fade to a larger texture in the distance.

Chapter 6

Conclusions and Discussion

6.1 Analysis of Results and Workflow

The project was able to accomplish its baseline goals effectively. A procedural city generation method creating terrain, buildings, and roads using combined architectural styles, presented in a real-time lighting environment was created. The system can be used to create additional versions or assets from the process. The terrain system is flexible, easy to add on to, and effectively hides its tiling artifacts. It is also an efficient representation for game engines, with adjustable export parameters depending on the size of the final output map. Continuing to process the map in Houdini before exporting to Unreal allows the map the receive further feature based adjustments, but also causes some complication in that the map must be resized slightly on import, necessitating slight adjustments to building placement. Road generation effectively models old world roads and navigates difficult terrain, winding up slopes and avoiding impassable areas. The road system also combines procedural and manual workflows, accounting for iterative modeling by allowing the artist to work in stages. The road system does rely on a significant amount of artist input after initial generation to clean up the networks. Additionally, the road system is currently limited in its available types of generation, as this work focused on organically shaped roads.

The building generation tool functions well to combine the model kit into compelling buildings. Each tool could be more advanced individually. Individual structures are somewhat simple rectangles or octagonal towers. Building placement effectively uses user designated districts to drive building types, and buildings are placed on the surface of the terrain with small flattened areas around them. Discrepancies in scale between the height-field before and after transitioning to Unreal make placements somewhat unreliable when processed in Houdini and imported. The combined architecture shows a strong overall design cohesion and effectively uses the elements of its component styles, especially Songnic and Romanesque. Aztec influences can be seen in some stonework patterning and the tecpans, but its representation could be improved overall. A large amount of structures are currently able to be created by the system, but as they are generated manually to be added to a library this process is somewhat slow. The absence of windows does impact how realistic the structures seem, but this is alleviated somewhat by the repetition of piers, arches, and columns across structures. Buttresses could also be added to building structures to further improve the architectural representation.

6.2 Future Work

Much of the future work involves reorganizing tools and how the systems are used together. Future versions of the terrain system could experiment with altering the terrain based on features in Unreal Engine instead of Houdini in order to avoid scale discrepancies. Alternatively, a forced resizing of the terrain to an appropriate export size could be done before processing for the features occurs. If the terrain were altered in Unreal, the road generation tool could be imported as a Houdini Digital Asset, taking the height-field file as an input and generating curves Unreal could use for processing. This would likely involve altering the road system to focus more heavily on procedural processing in Houdini before outputting curves to Unreal. Additionally, more generation options such as grid focused roads or bridge support could be added. Because the road tool is designed to work for an arbitrary height-field, it could be used independently of the other tools. Building generators are very custom and dependent on the model kit, and should therefore be developed primarily to build relationships in how different pieces are placed. The tool should then draw on models from a model kit already imported to Unreal. The objects can then be instanced per building. This would allow far more granular control over material application, and material application could additionally be performed by the tool at generation time. This could then be used to generate single buildings or as a supplementary tool. Similarly, building placement could use the same processes it currently uses to create potential placement points, drawing on the building generator tools to produce buildings for the city. Because it is simply generating points for placement with district designations, any number of arbitrary building generators could be used in conjunction with the city generator as long as they use the same district designations. Thus, several model kits with corresponding sets of building generators could be used to procedurally create a variety of areas with different architectural styles over a realistic terrain, following an artist's intent.

Appendices

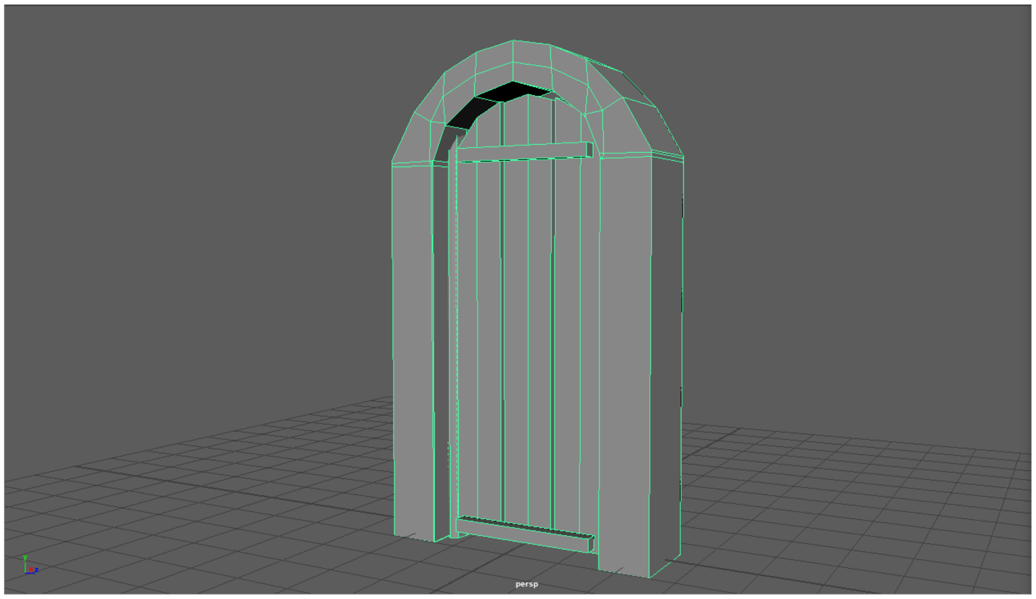


Figure 1: Arch door

Appendix A Model Kit

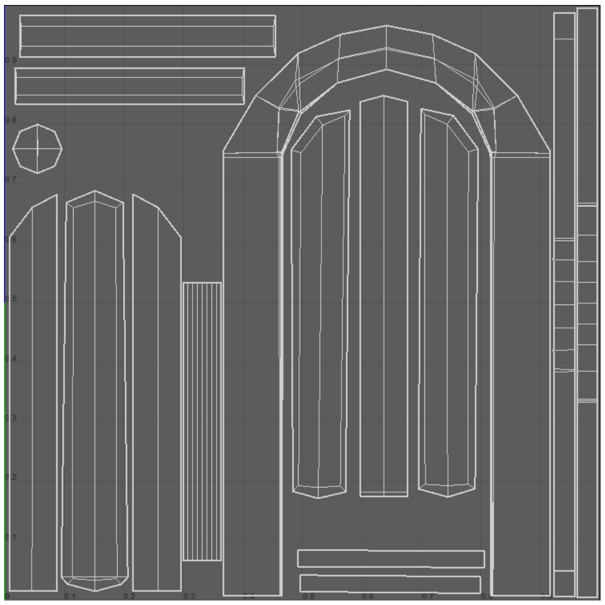


Figure 2: Arch door UV set

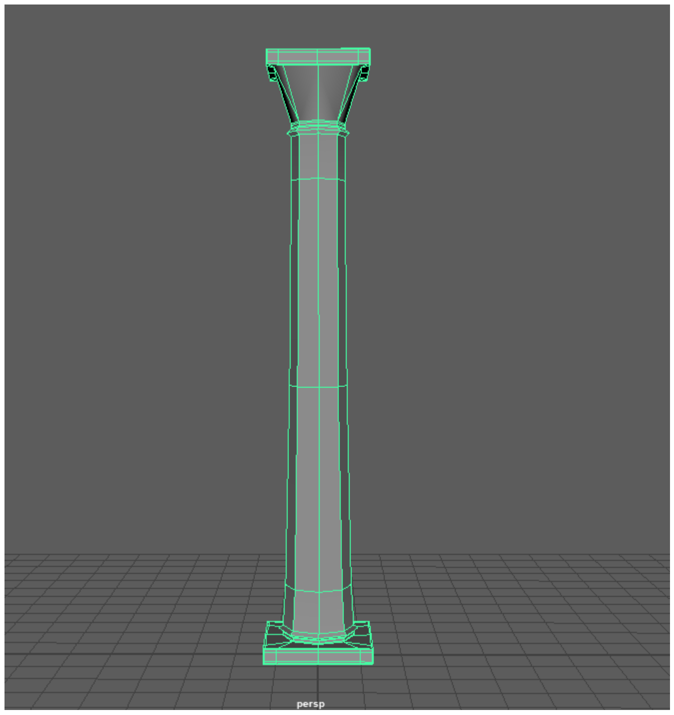


Figure 3: Corinthian column

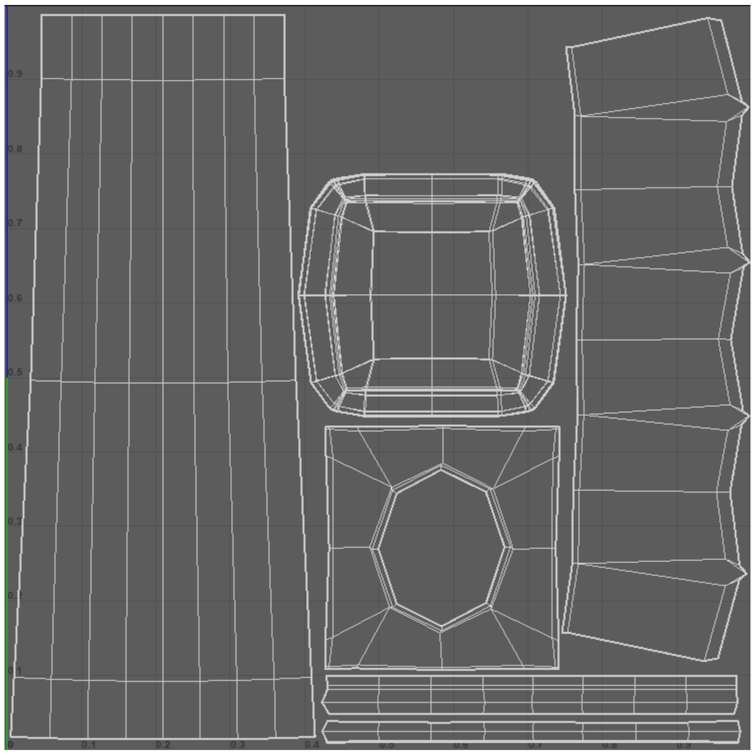


Figure 4: Corinthian column UV set

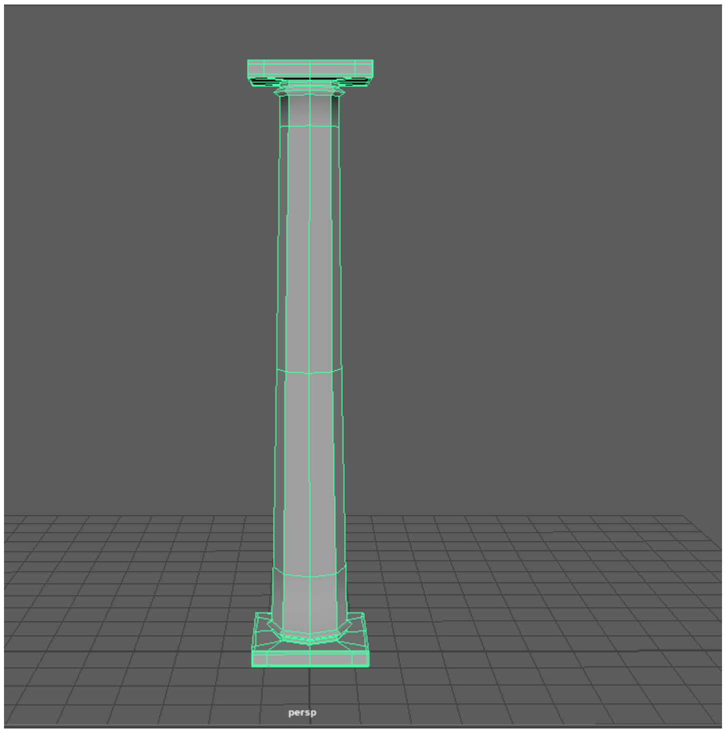


Figure 5: Doric column

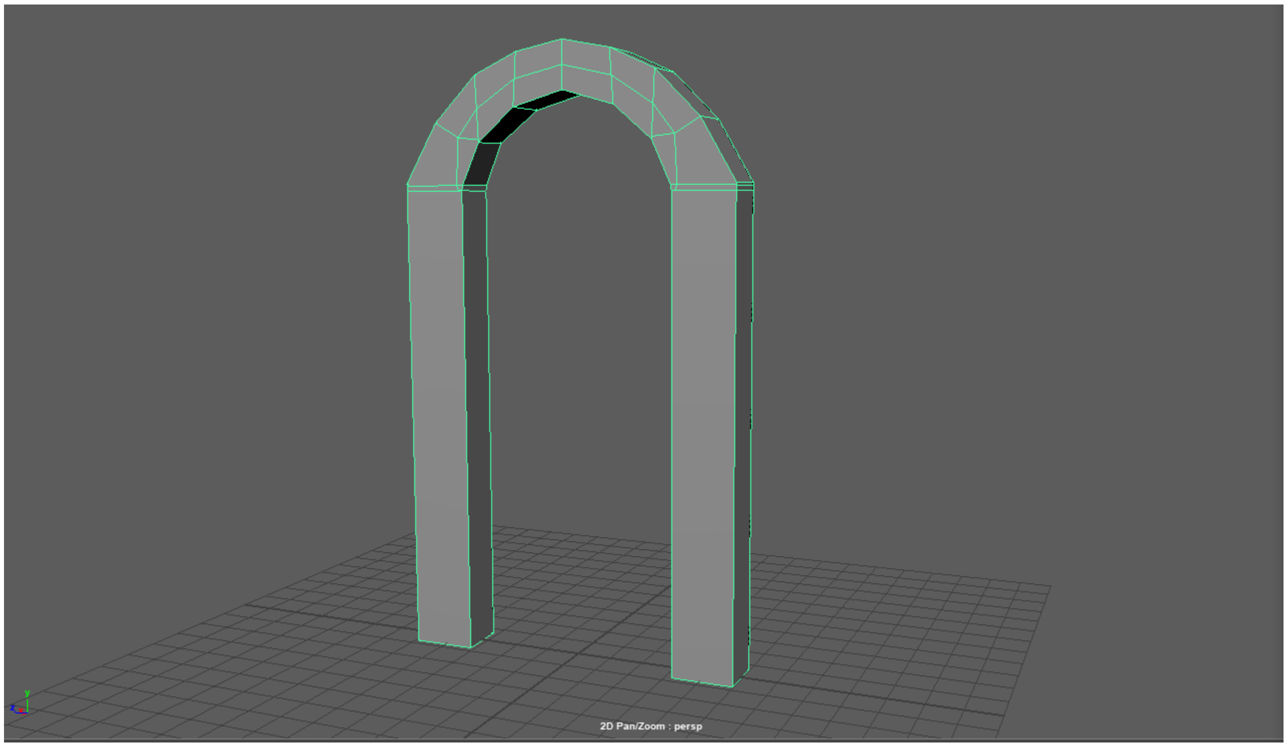


Figure 6: Single semicircular arch

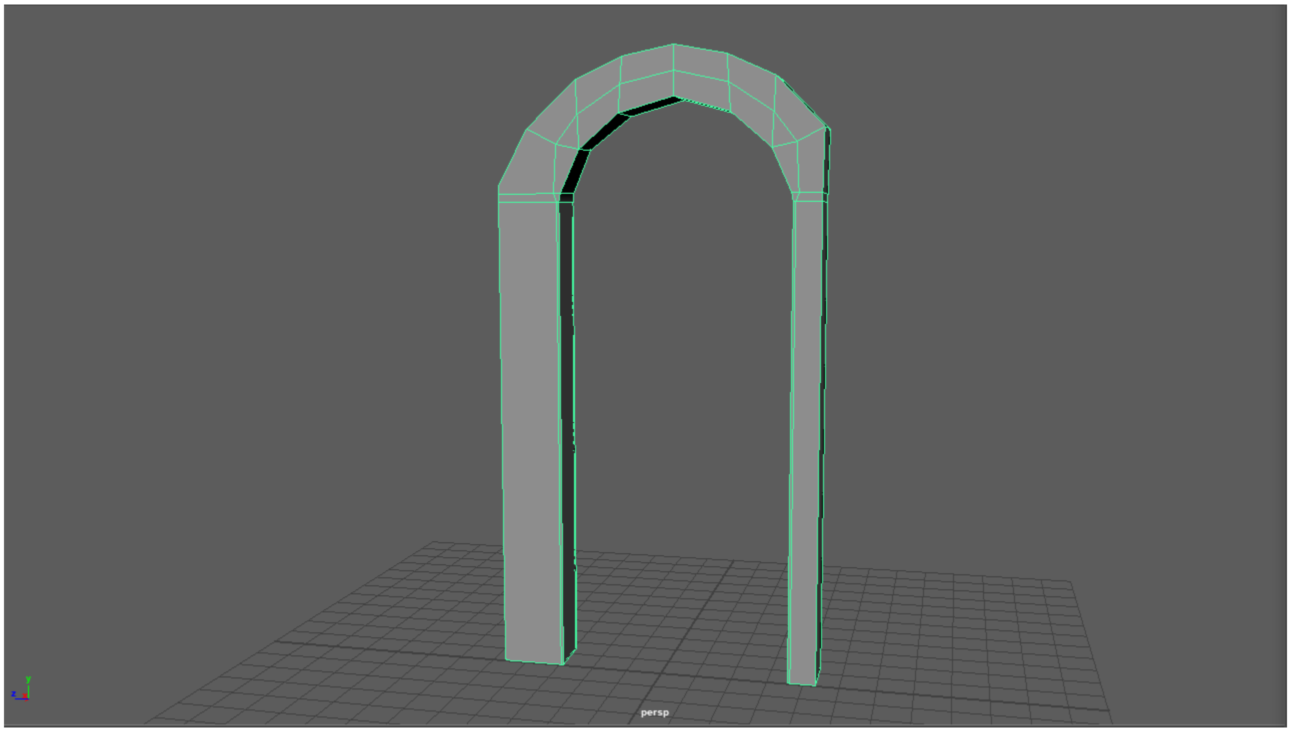


Figure 7: Repeating semicircular arch end piece

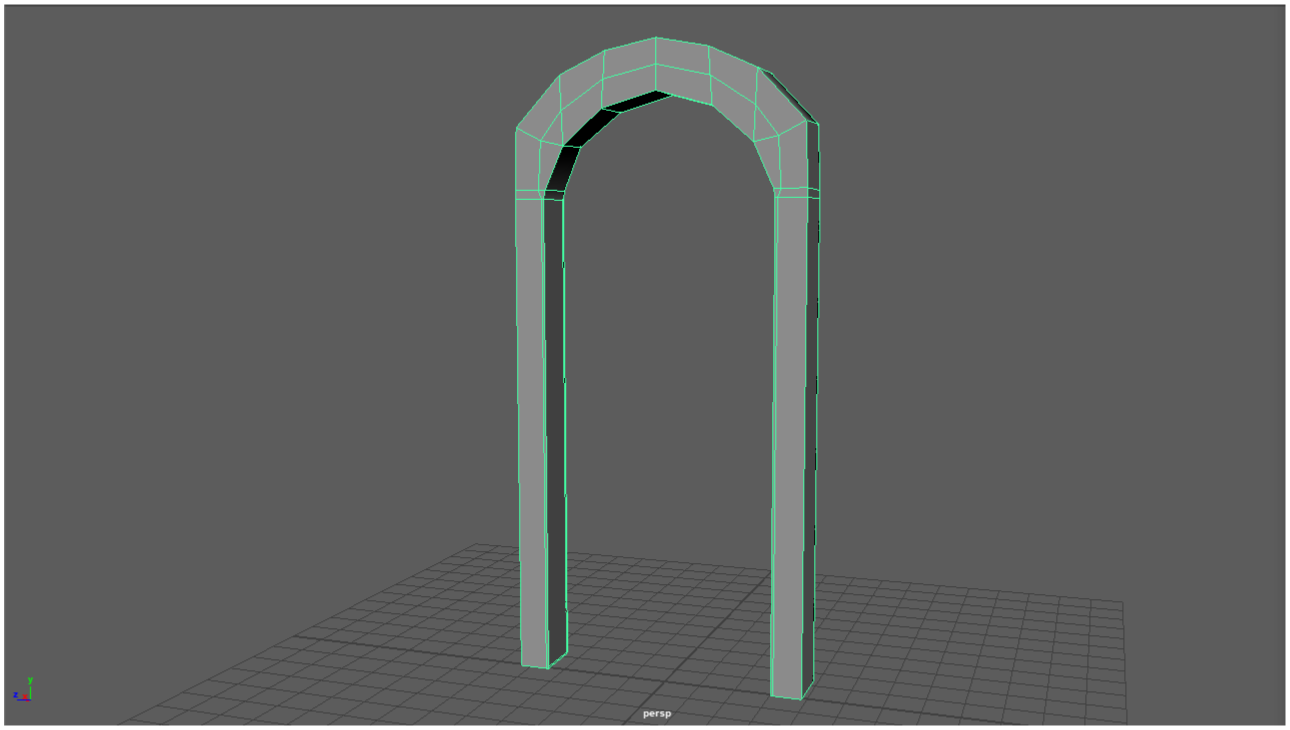


Figure 8: Repeating semicircular arch center piece

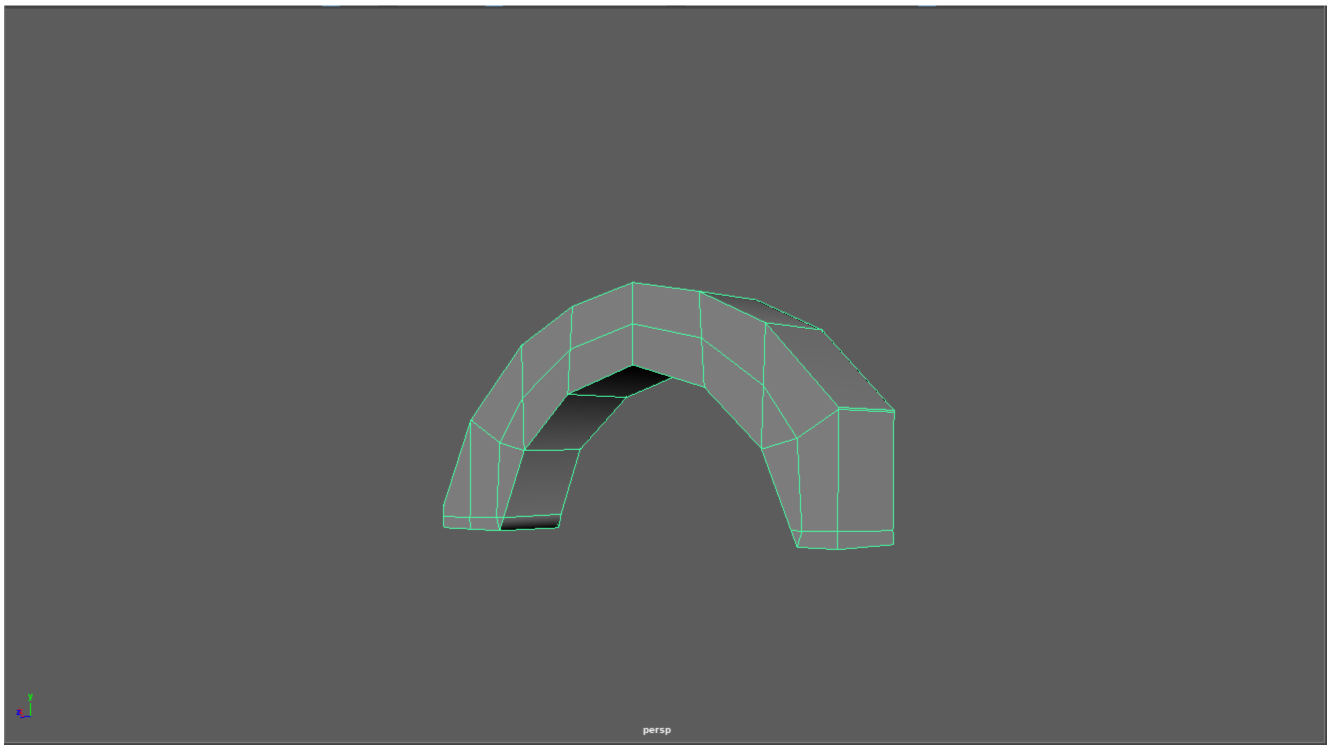


Figure 9: Repeating pier end piece

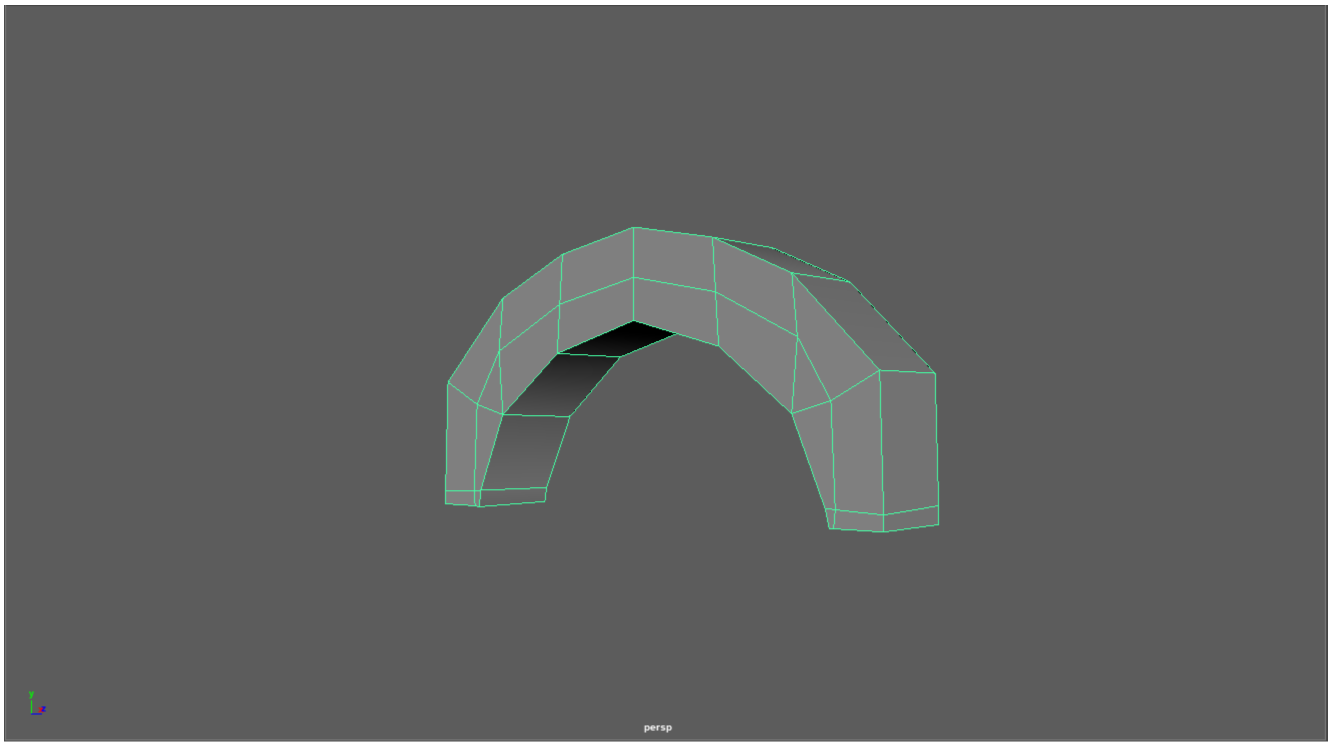


Figure 10: Repeating pier end piece

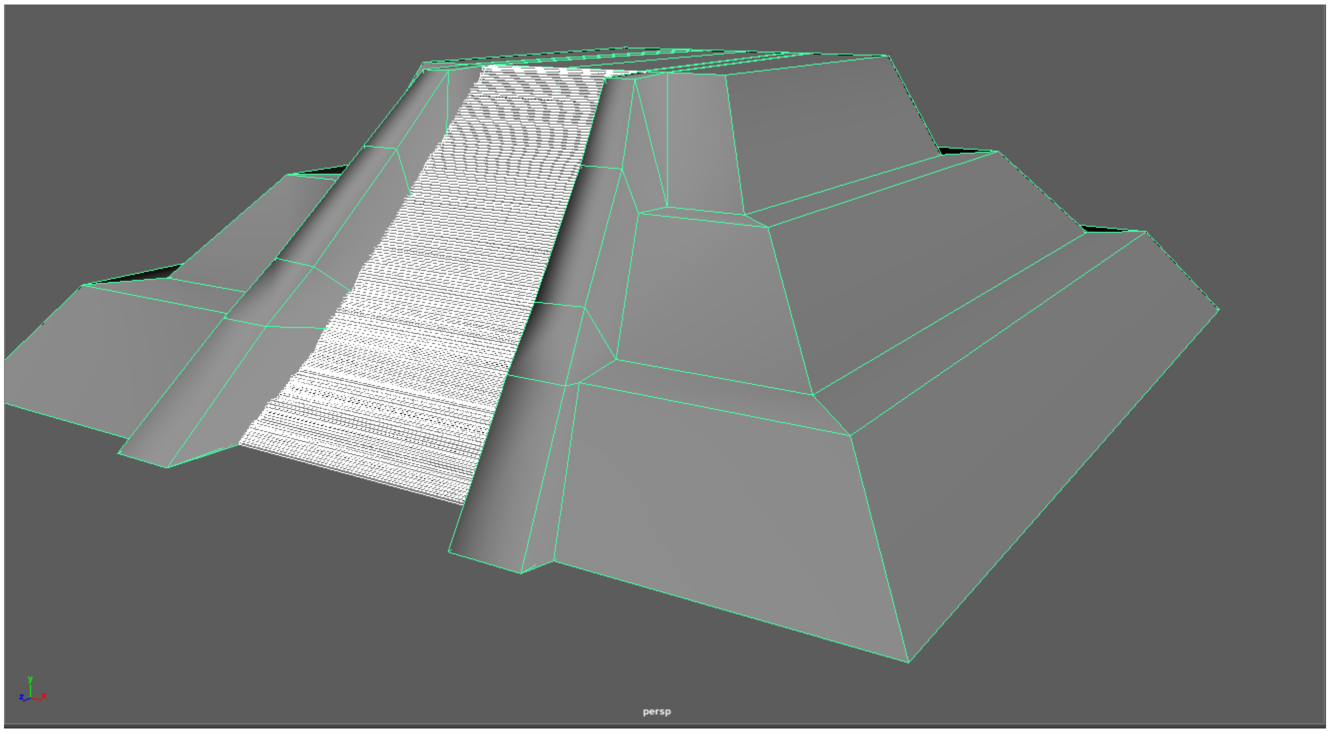


Figure 11: Single stair tecpan

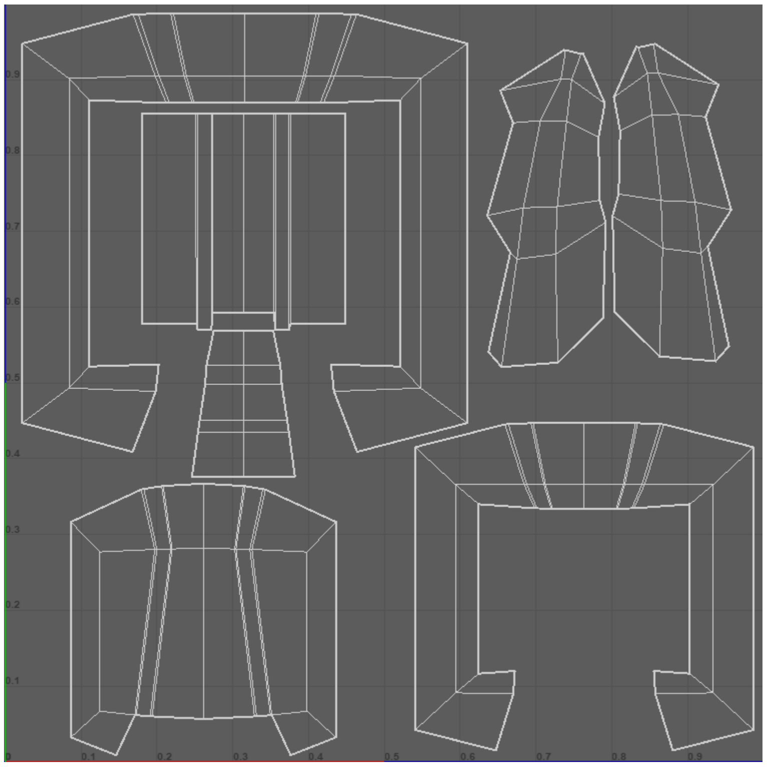


Figure 12: Single stair tecpan UV set

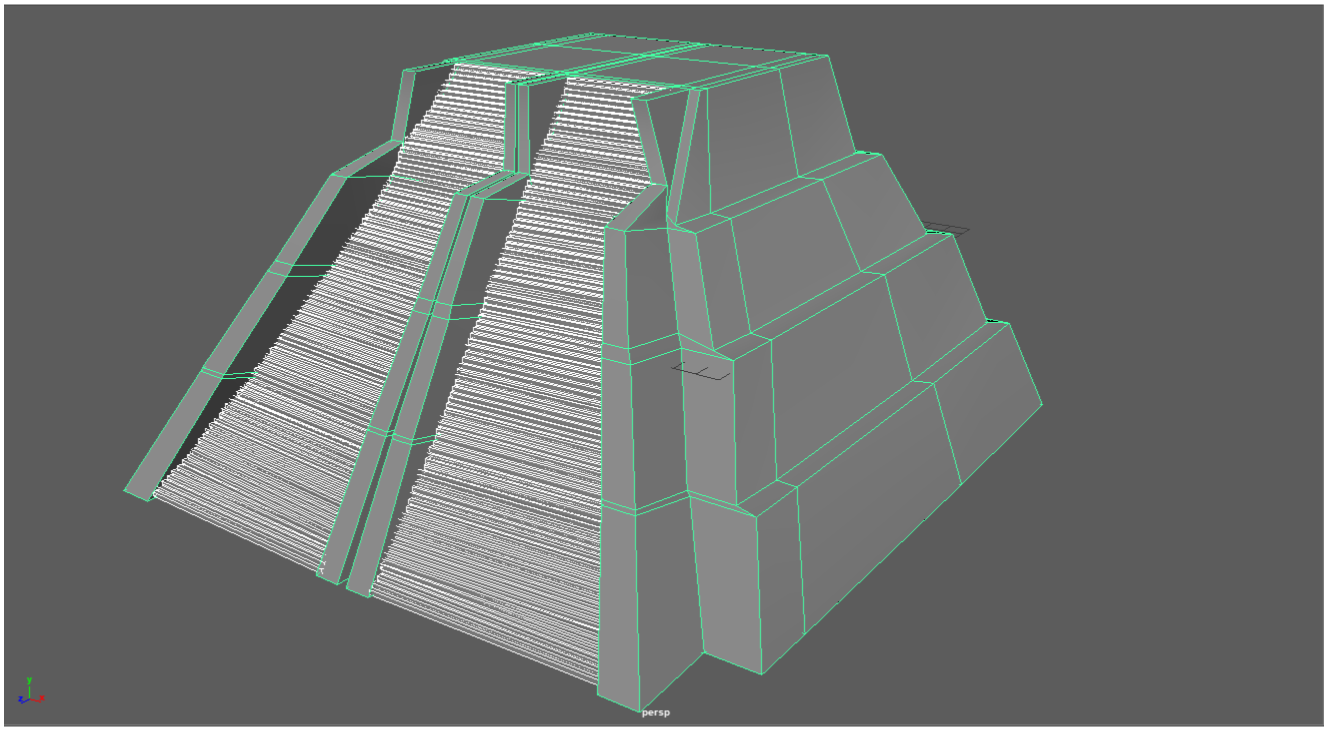


Figure 13: Double stair tecpan

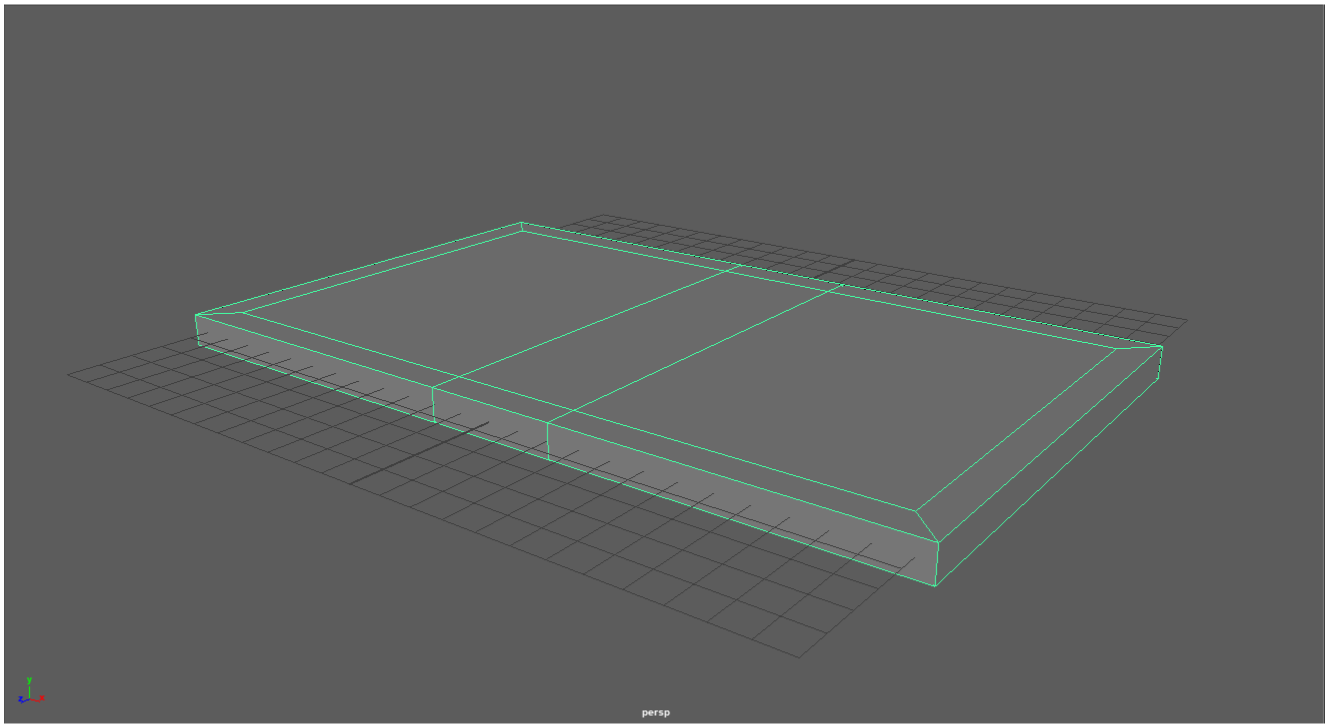


Figure 14: Small house floor

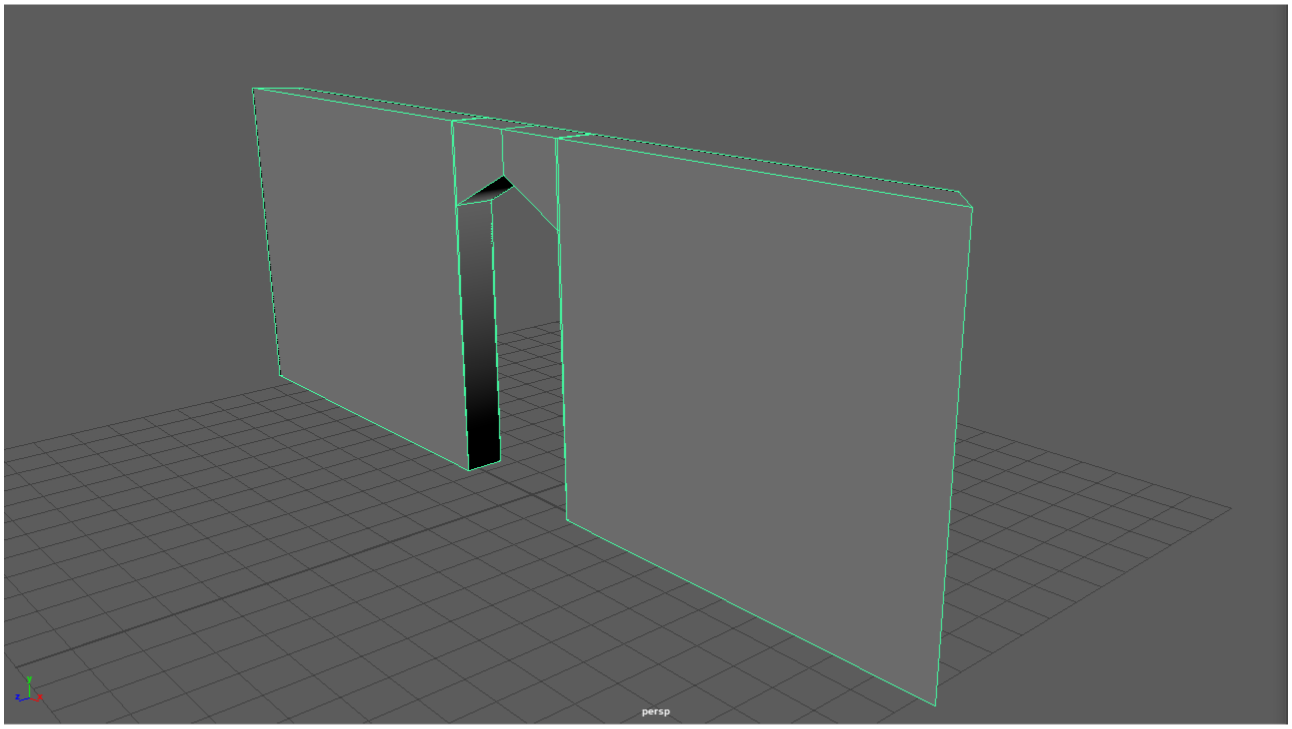


Figure 15: Small house door wall

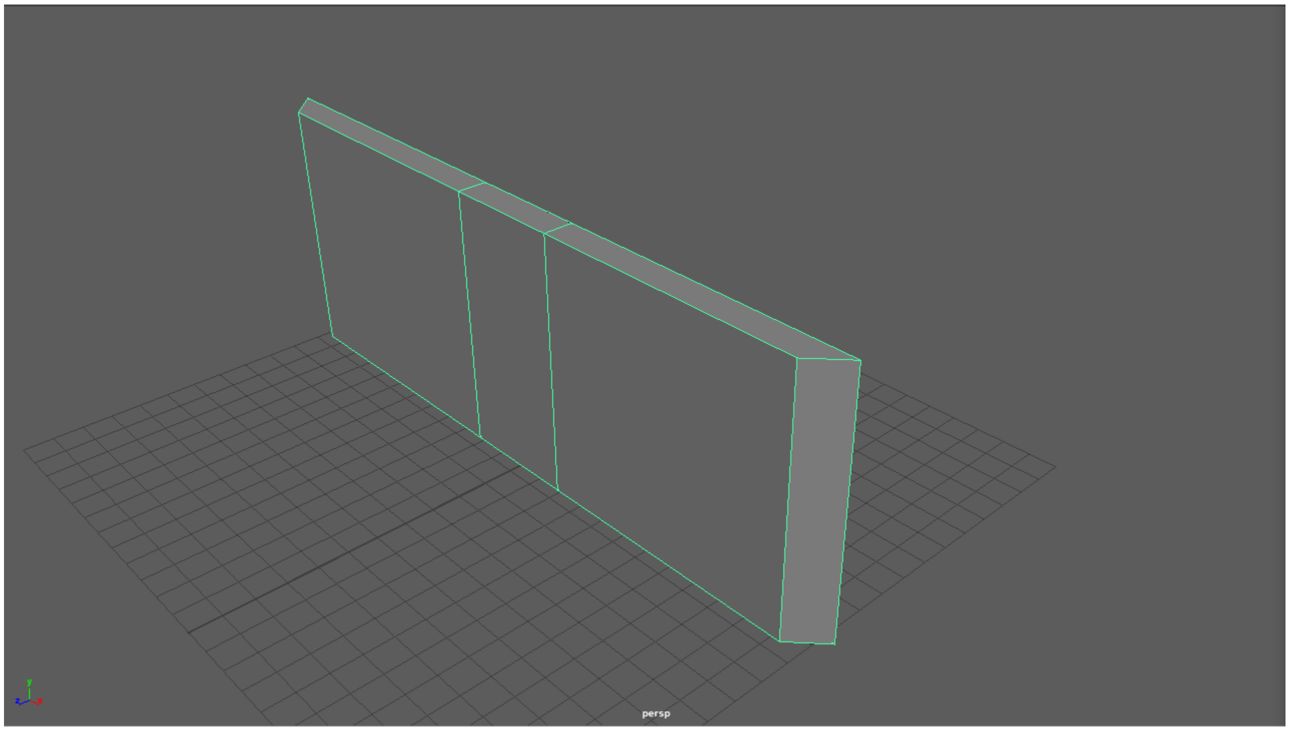


Figure 16: Small house long wall

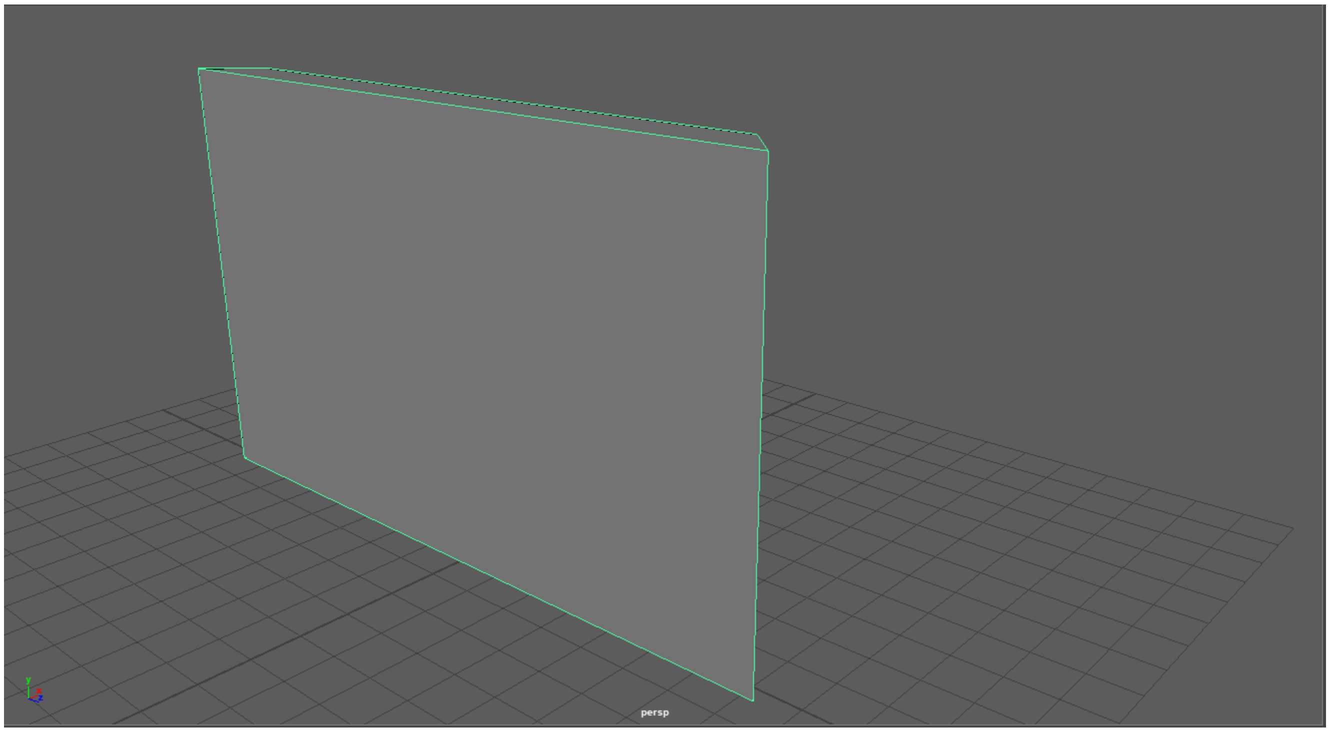


Figure 17: Small house short wall

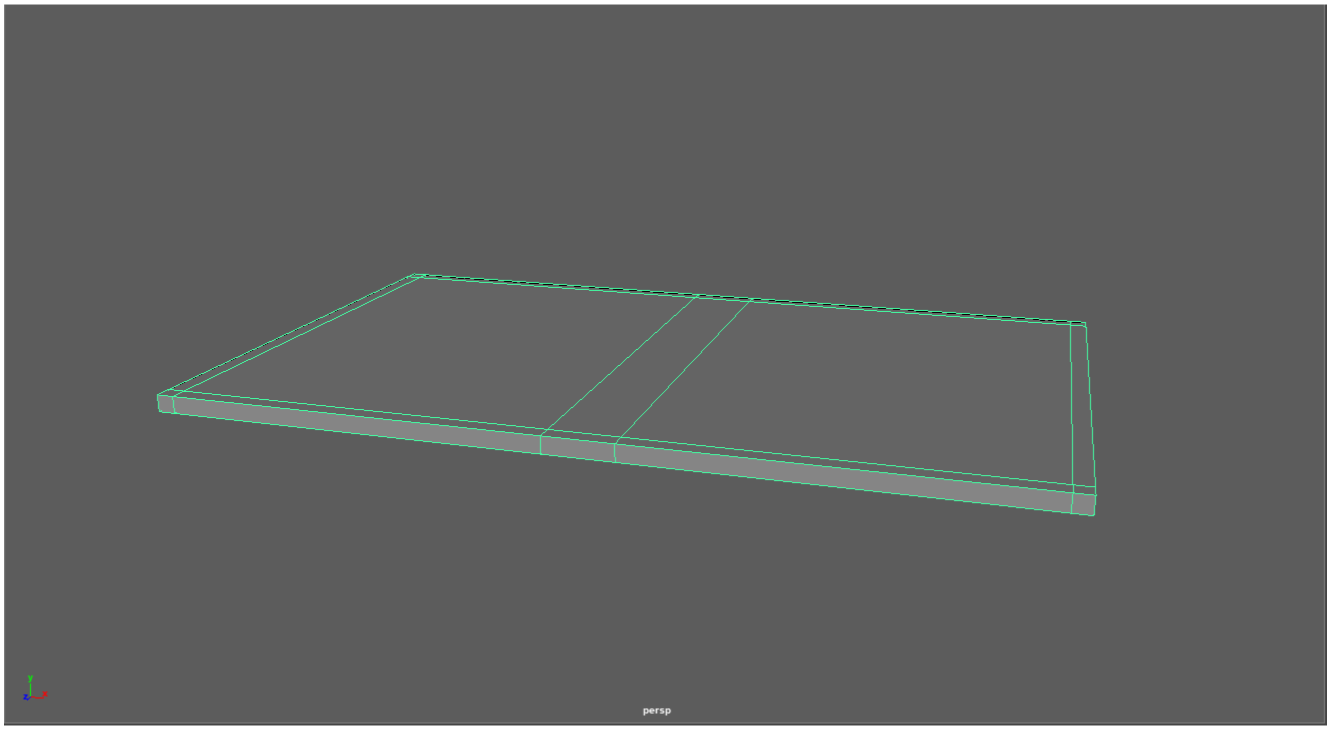


Figure 18: Large house floor

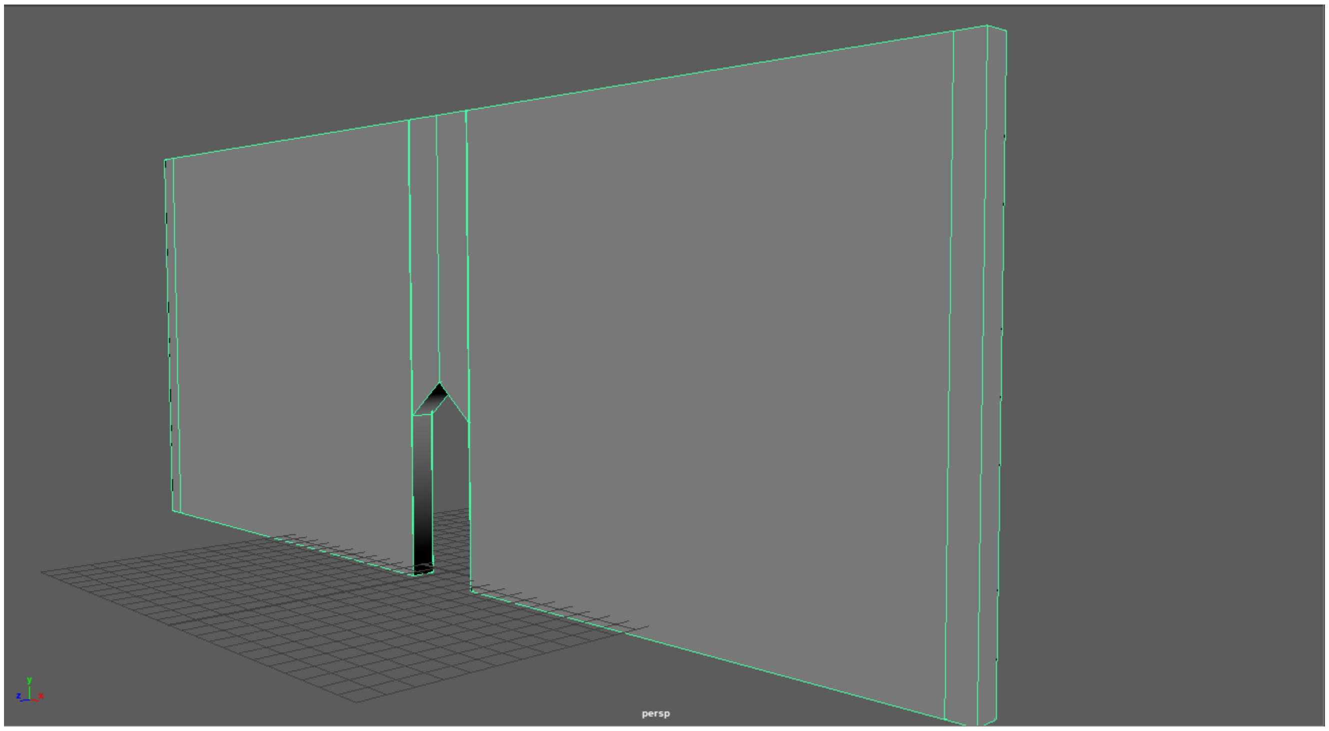


Figure 19: Large house door wall

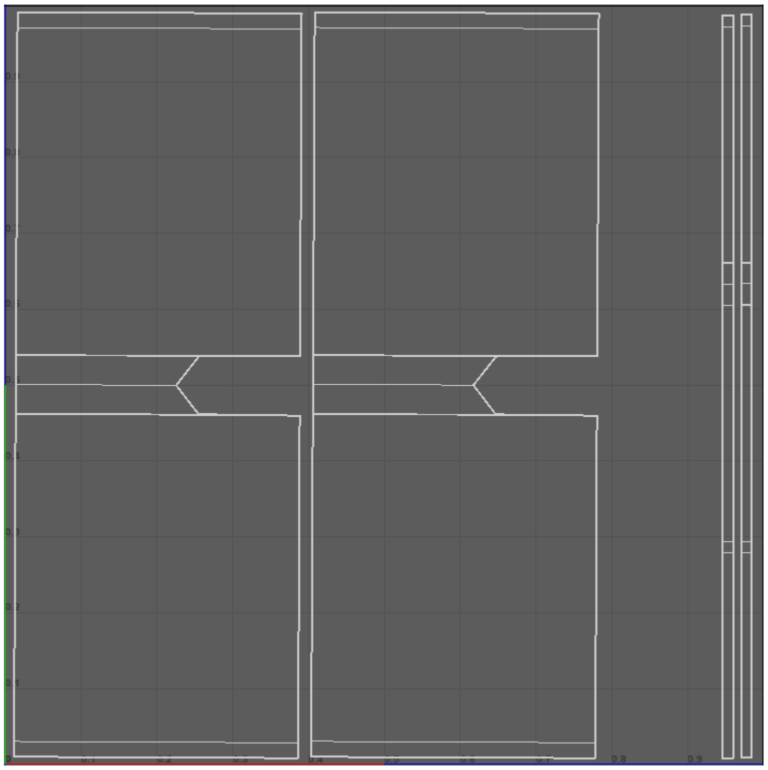


Figure 20: Large house door wall UV set

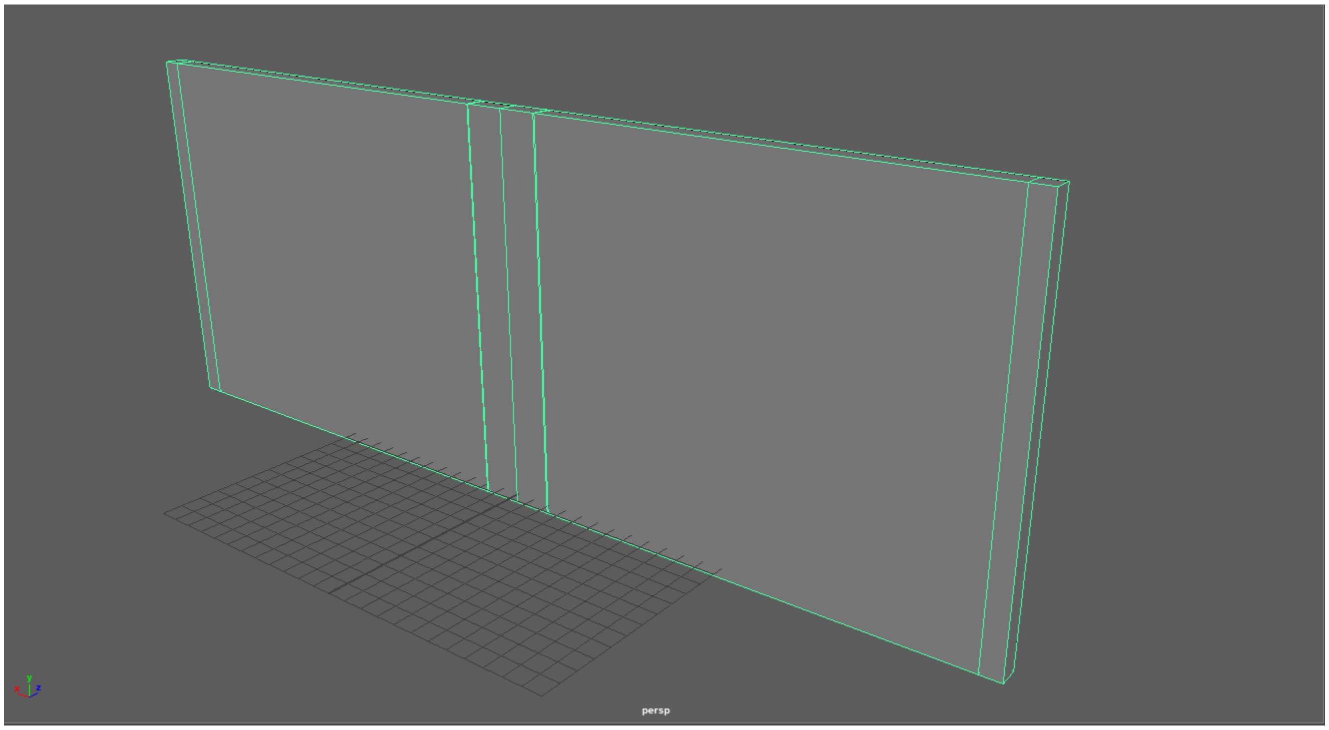


Figure 21: Large house long wall

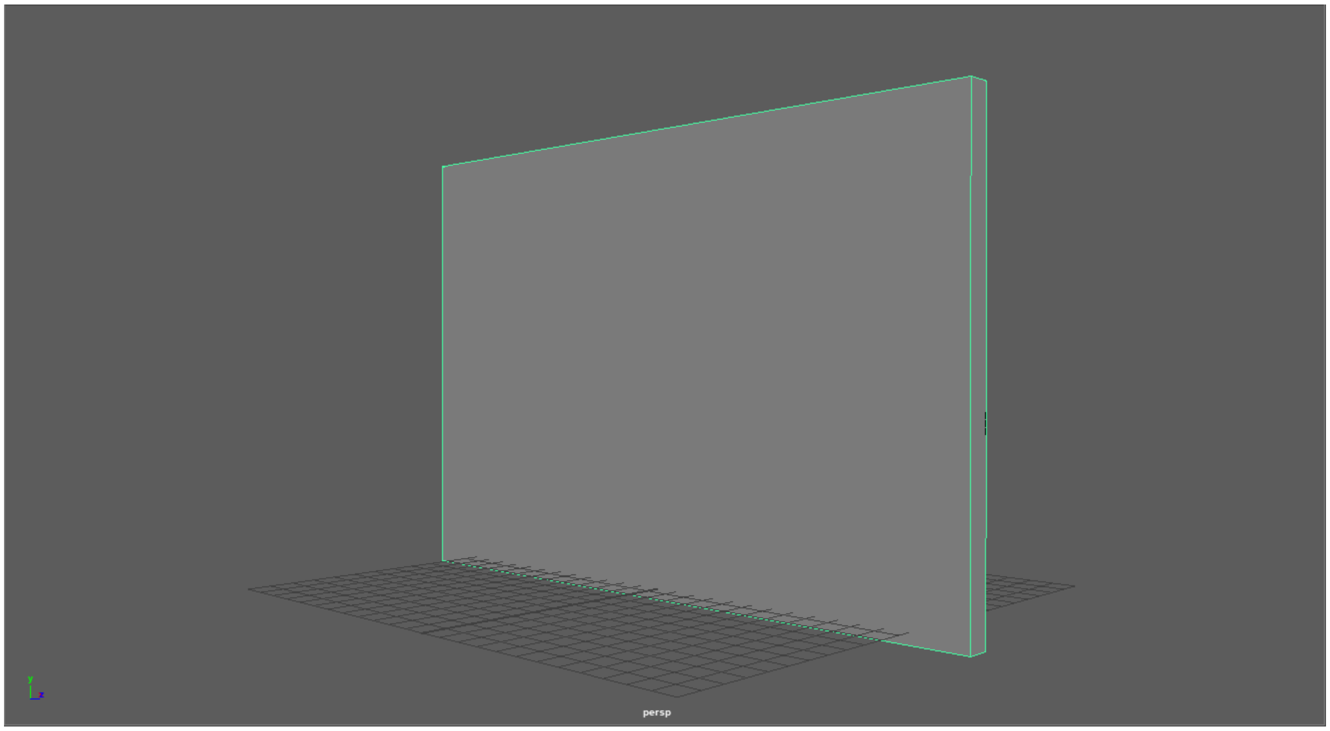


Figure 22: Large house short wall

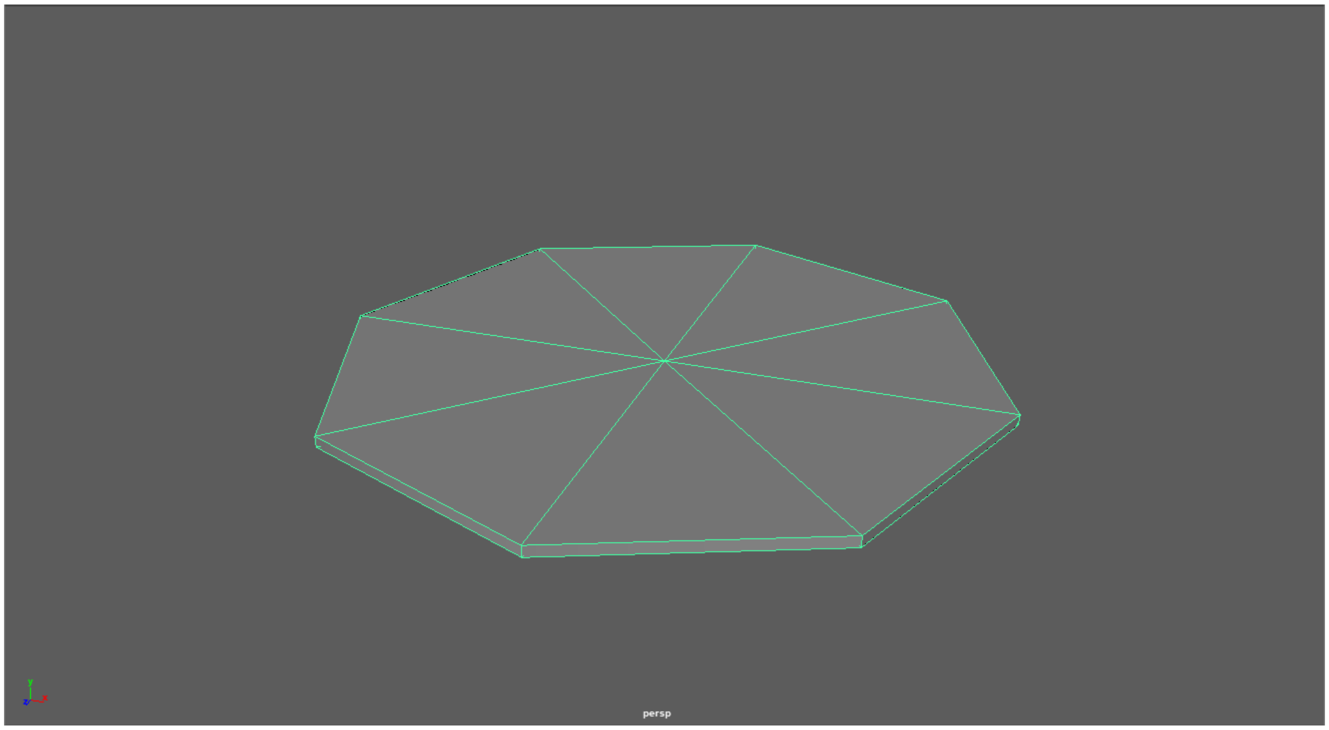


Figure 23: Pagoda floor

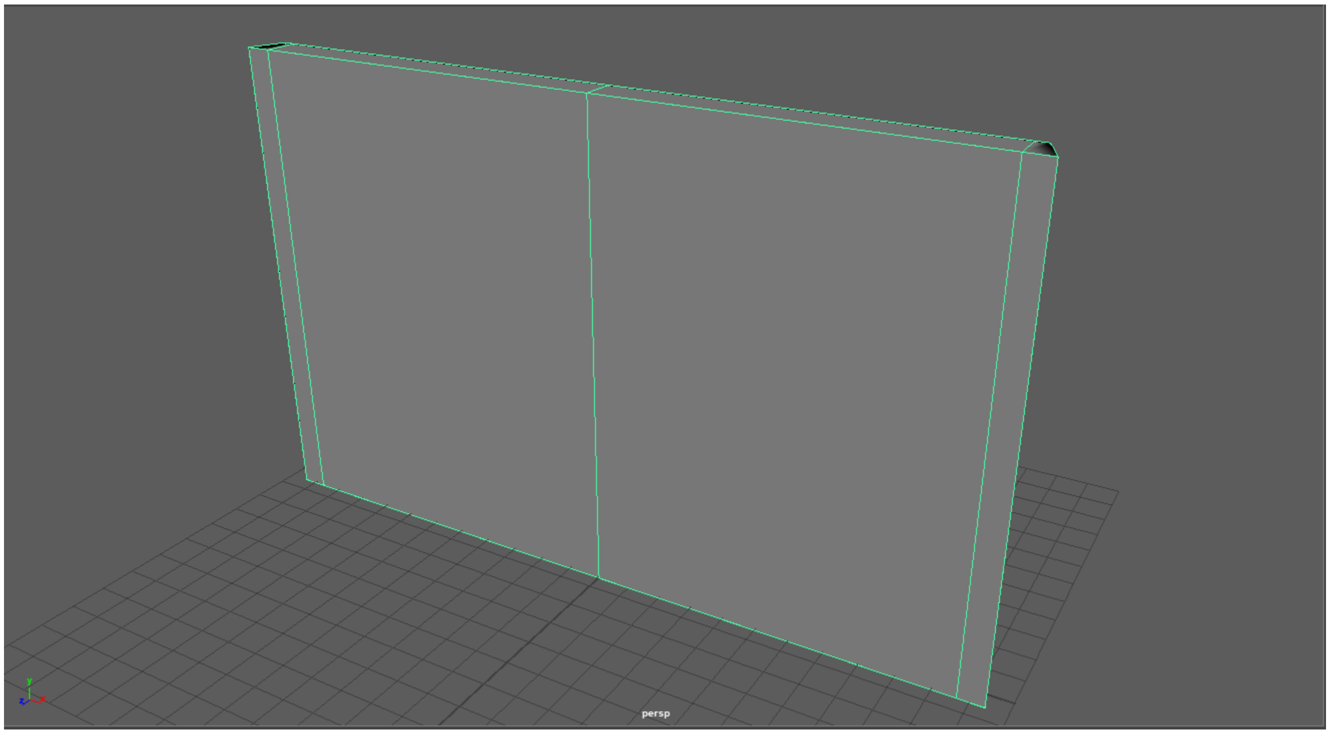


Figure 24: Pagoda wall

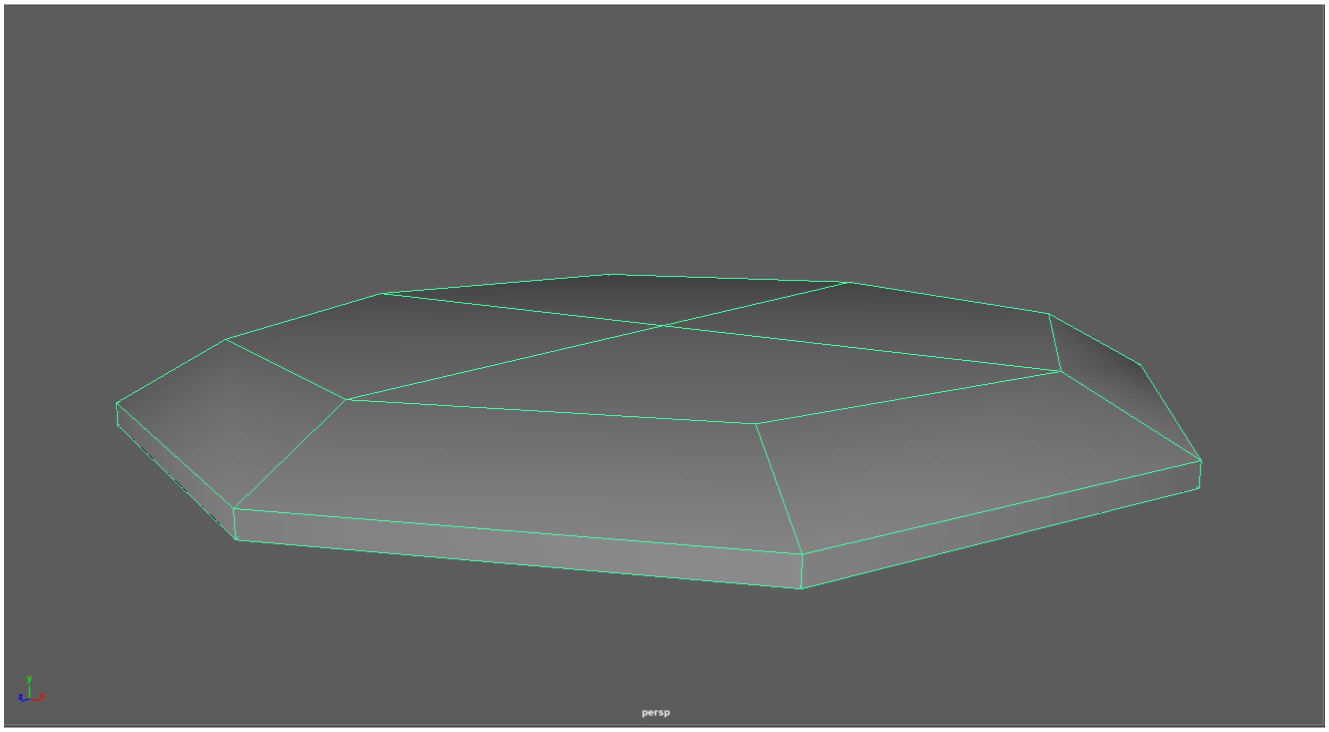


Figure 25: Pagoda segment roof

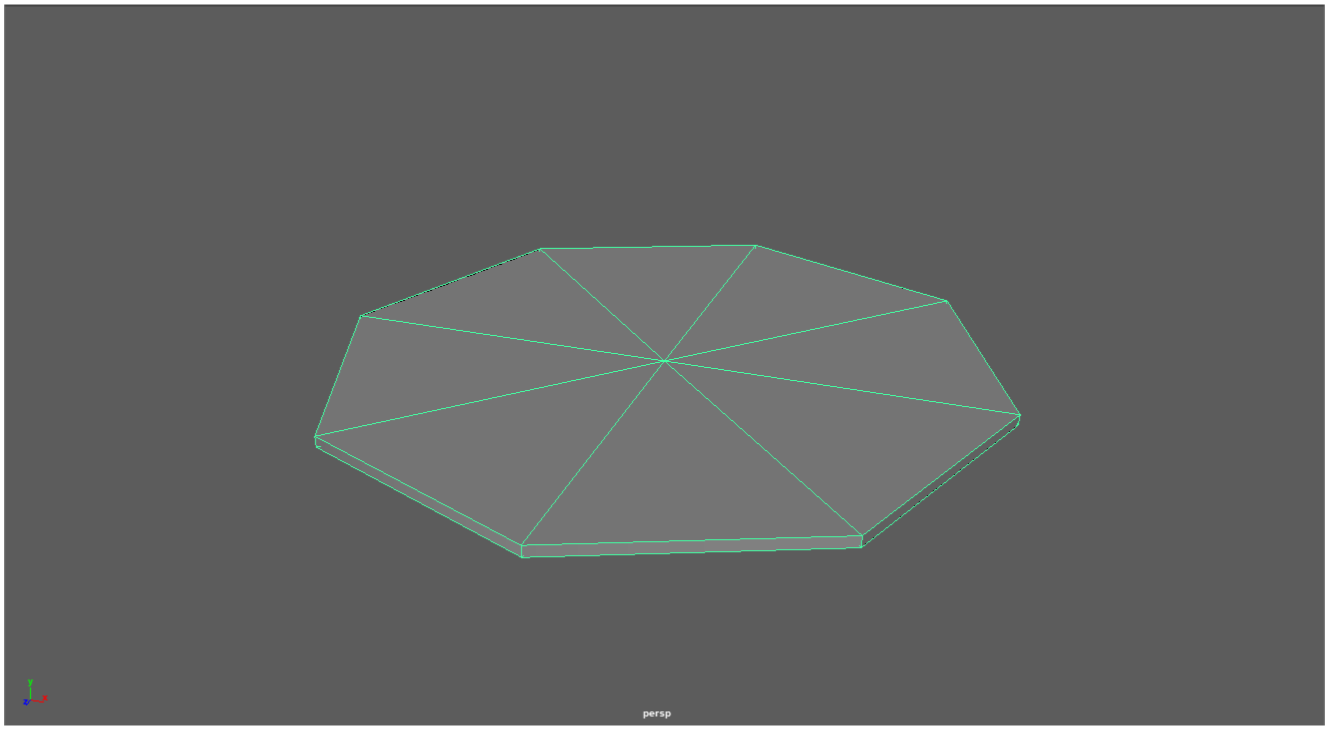


Figure 26: Pagoda roof

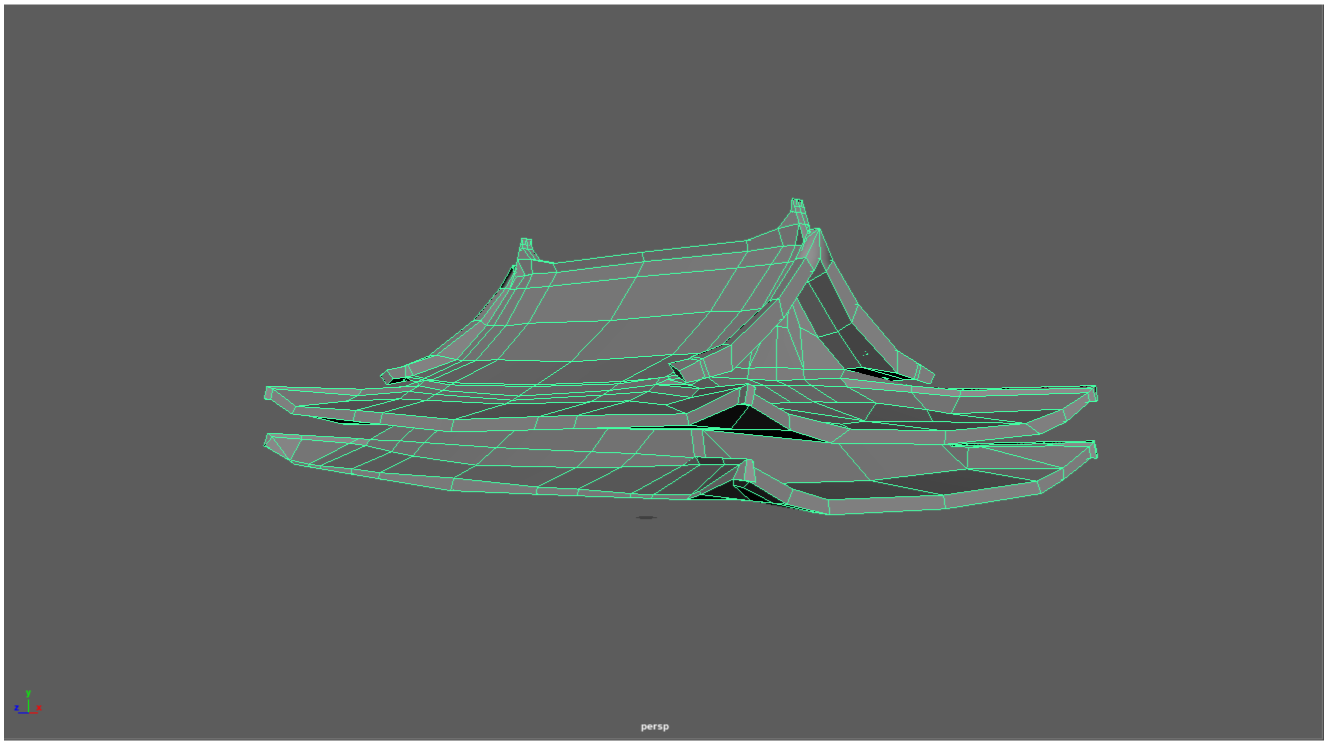


Figure 27: Double Eaves Xieshan roof

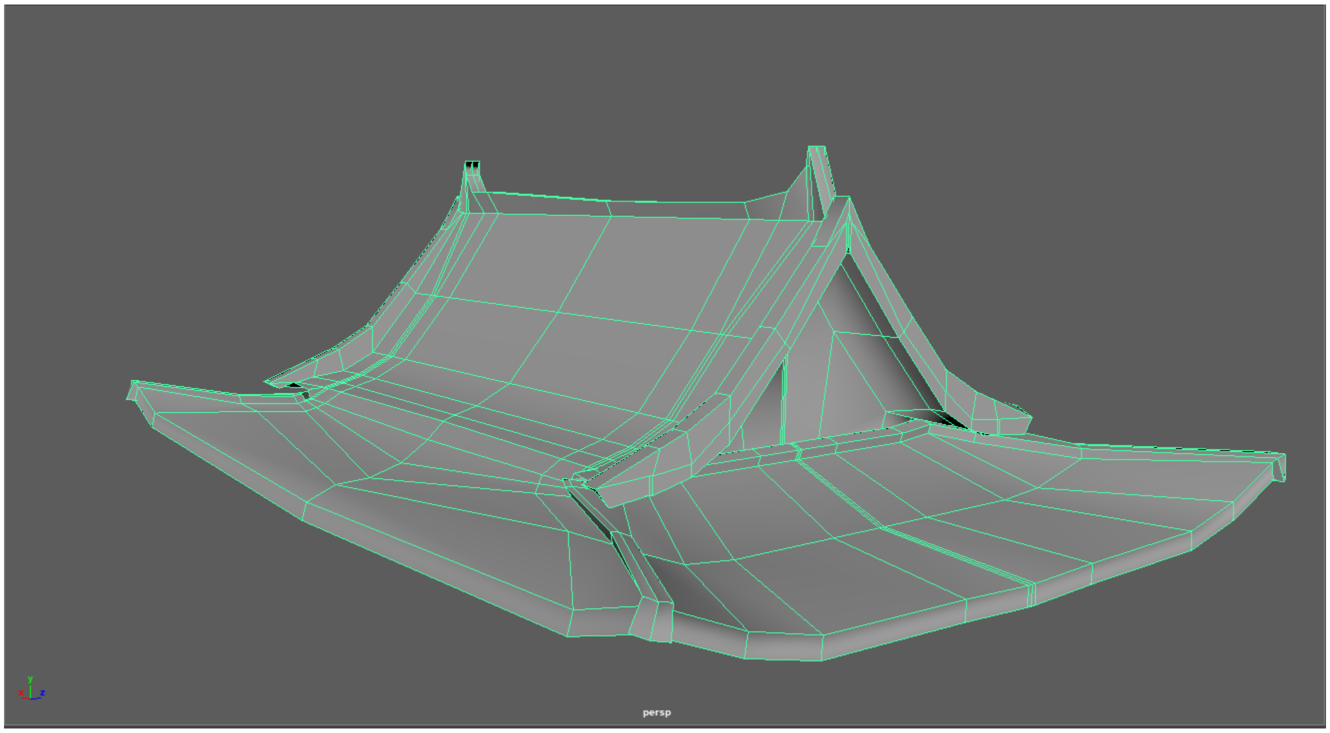


Figure 28: Xieshan roof

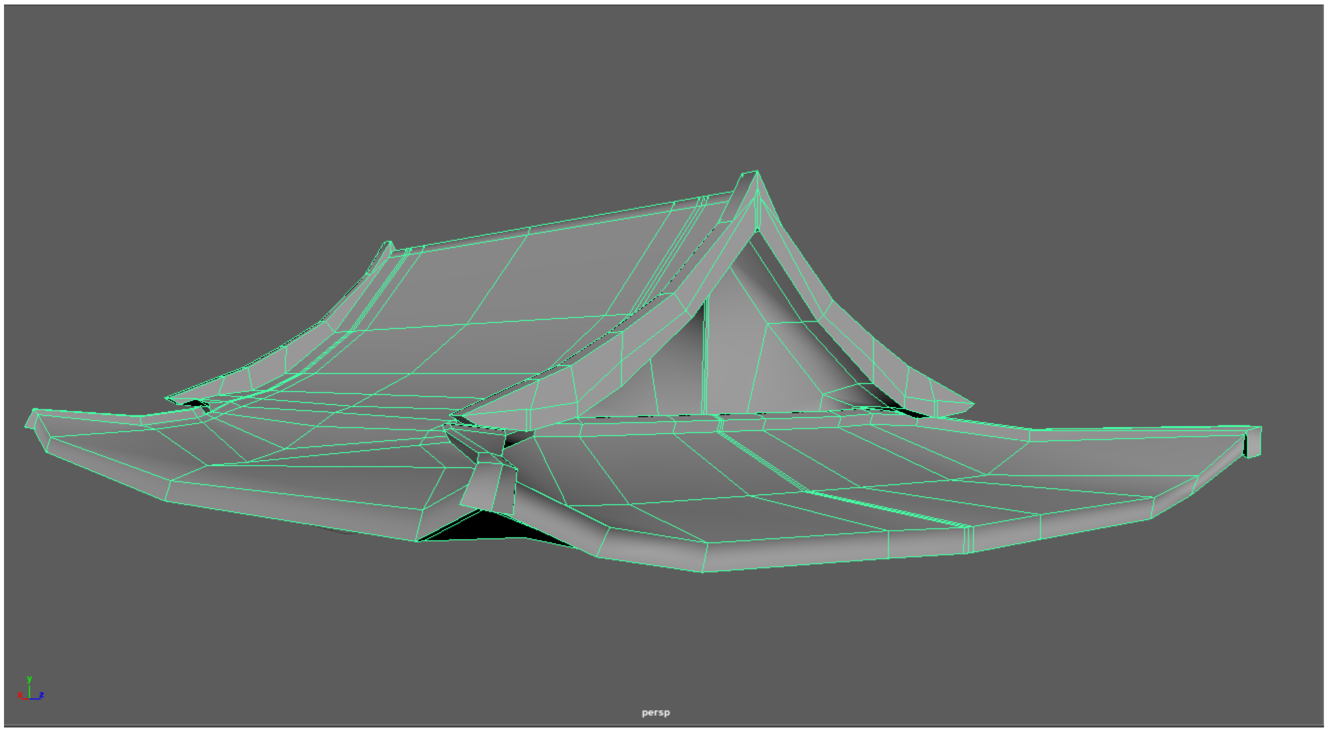


Figure 29: Xieshan Juanpeng roof

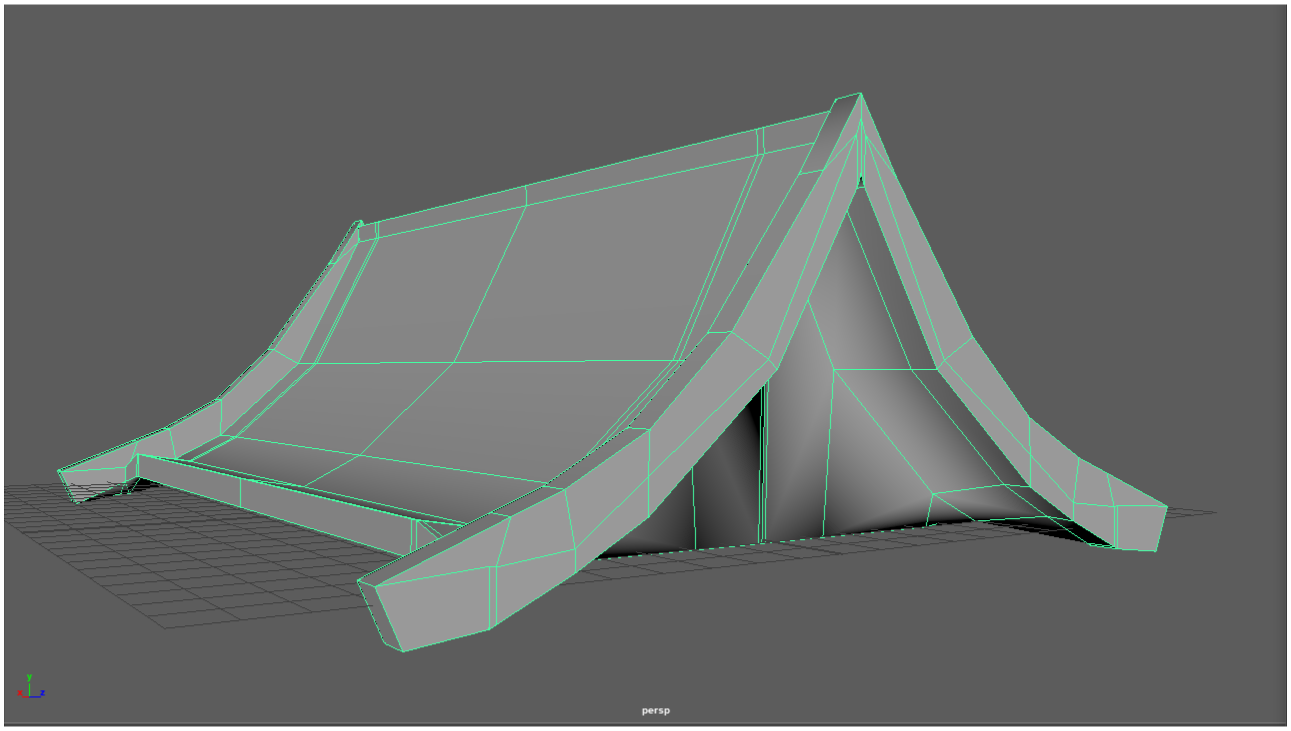


Figure 30: Yingshan roof

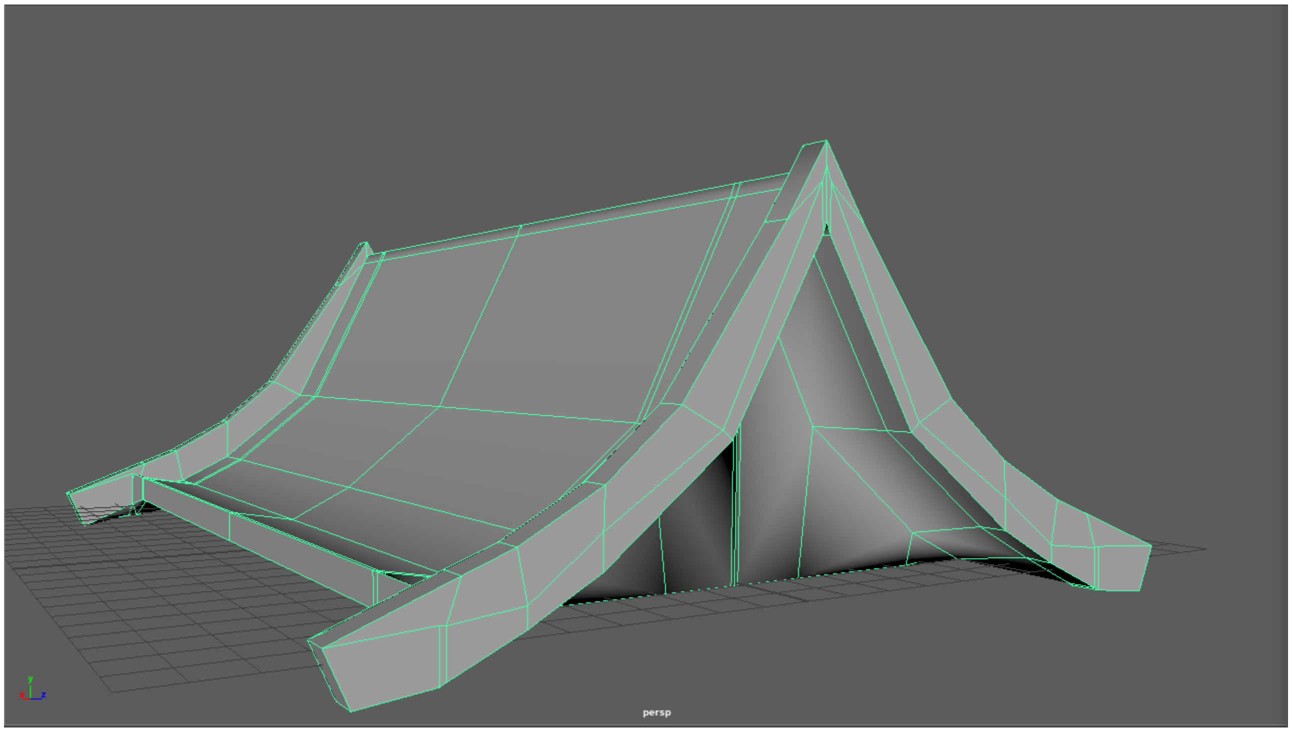


Figure 31: Yingshan Juanpeng roof

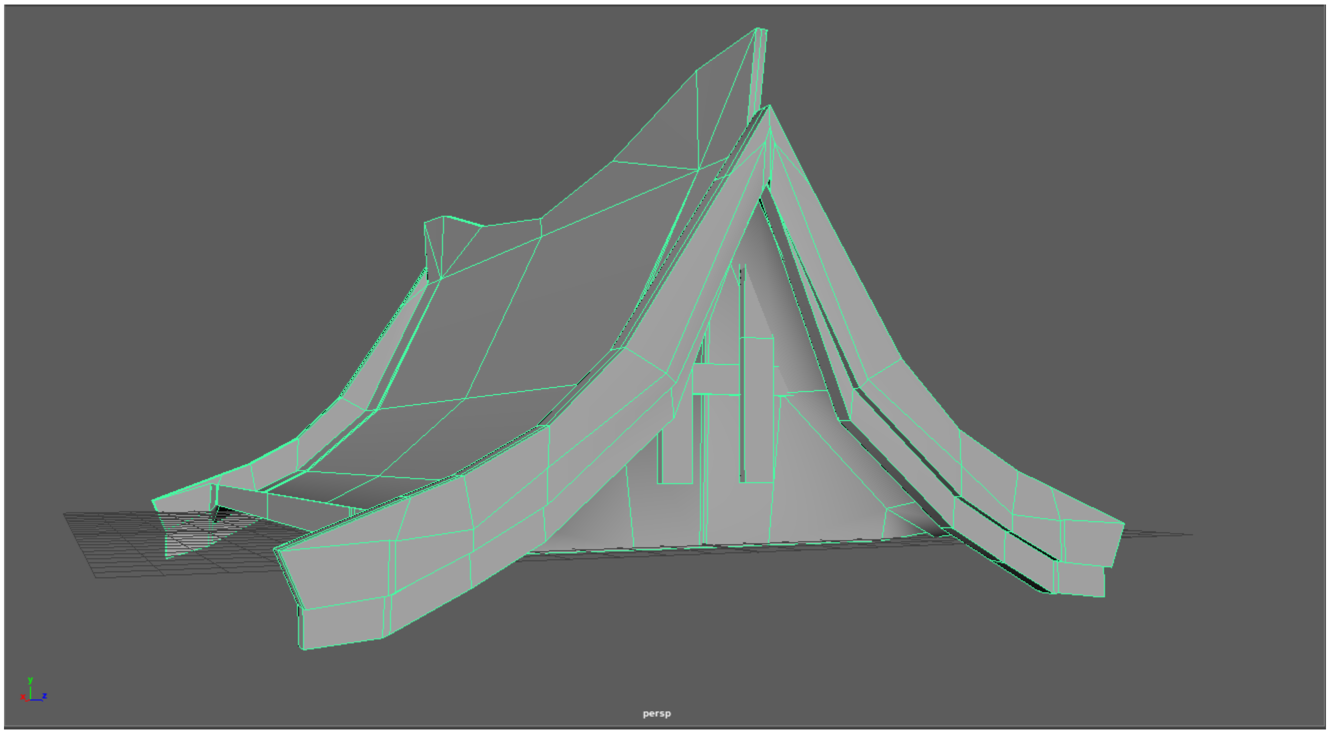


Figure 32: Xuanshan roof

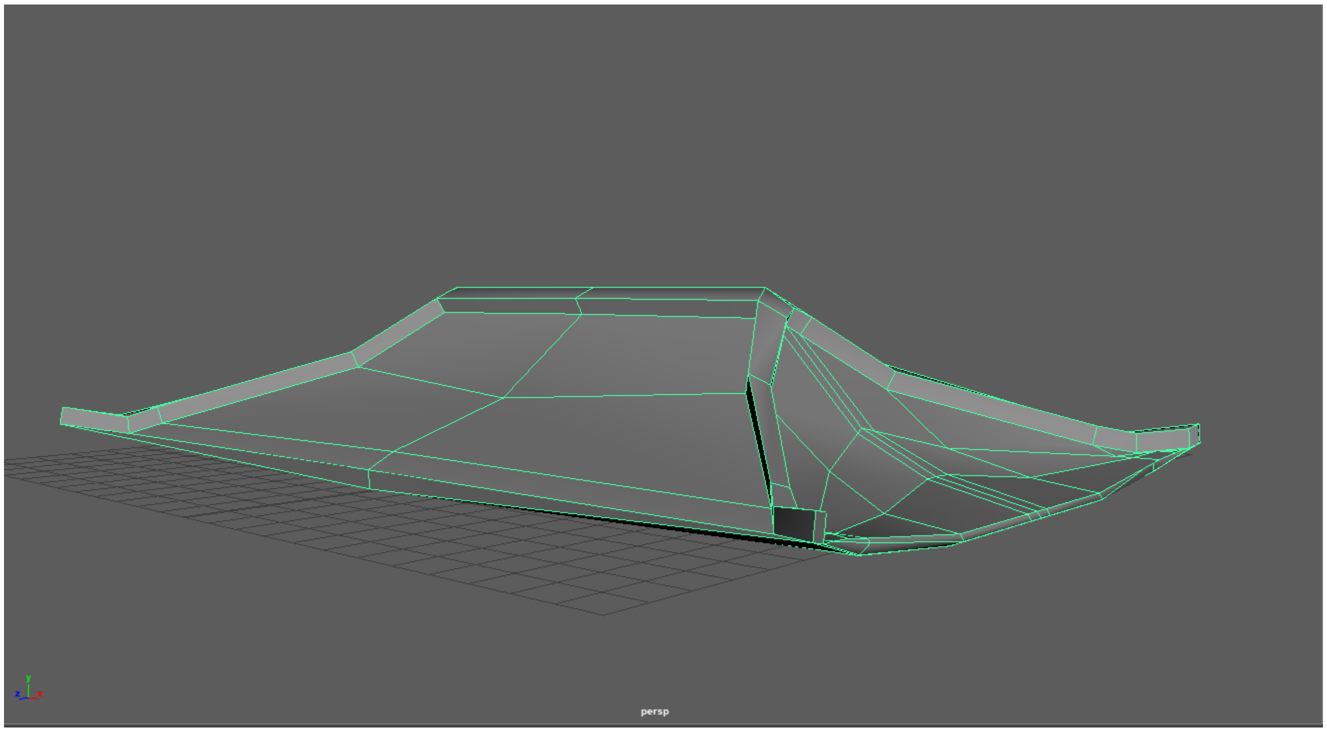


Figure 33: Wudian roof

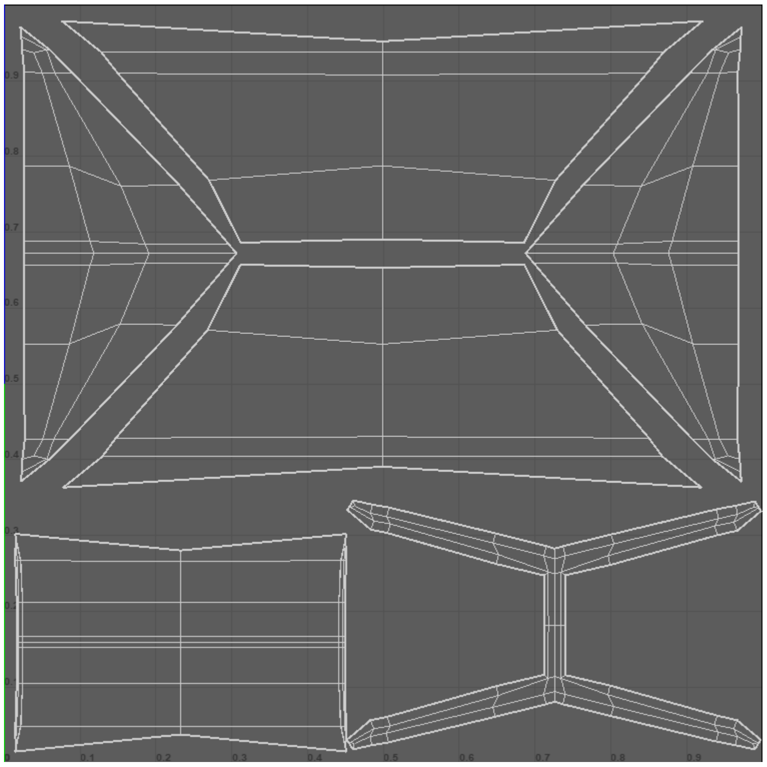


Figure 34: Wudian roof UV set

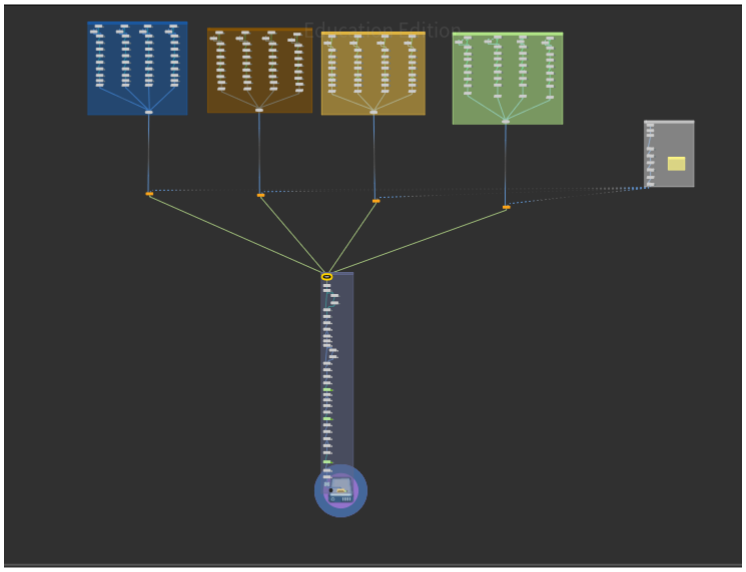


Figure 35: Terrain generation node graph

Appendix B Node Graphs

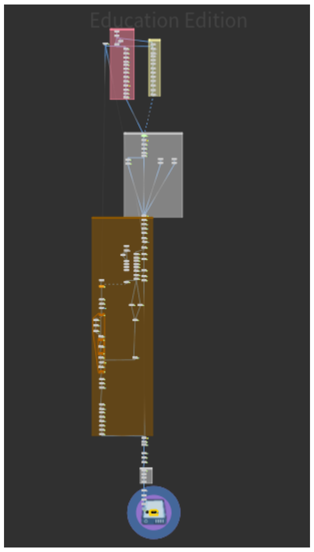


Figure 36: Road generation node graph

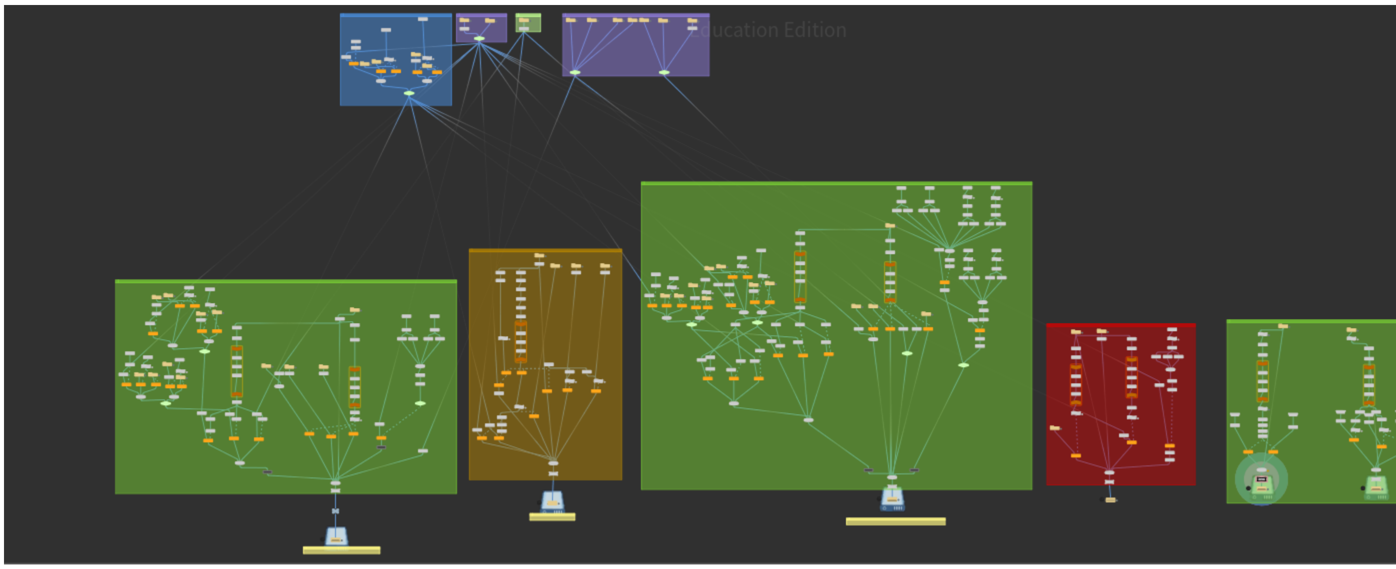


Figure 37: Building generators node graph

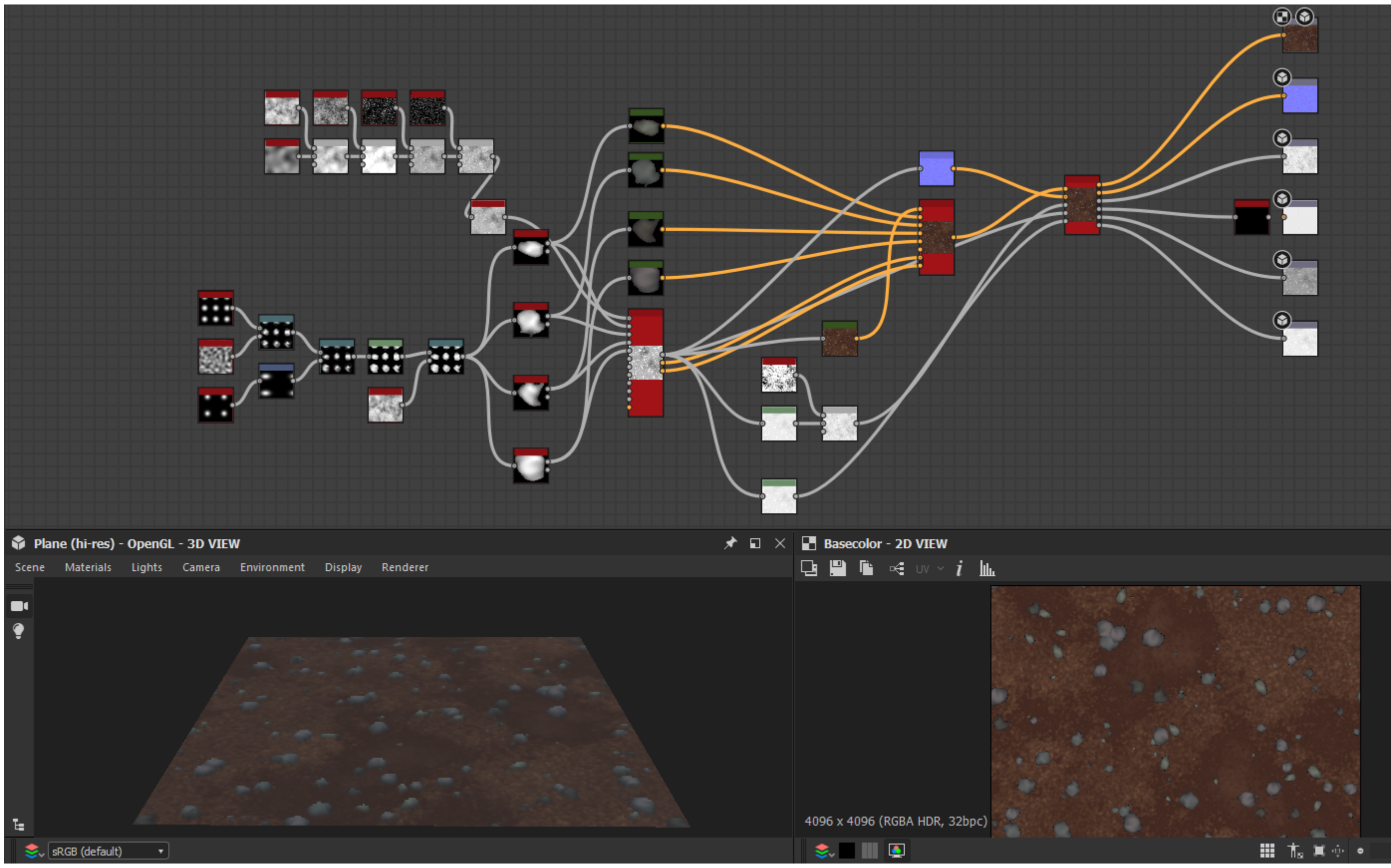


Figure 38: Coarse dirt Substance Designer graph

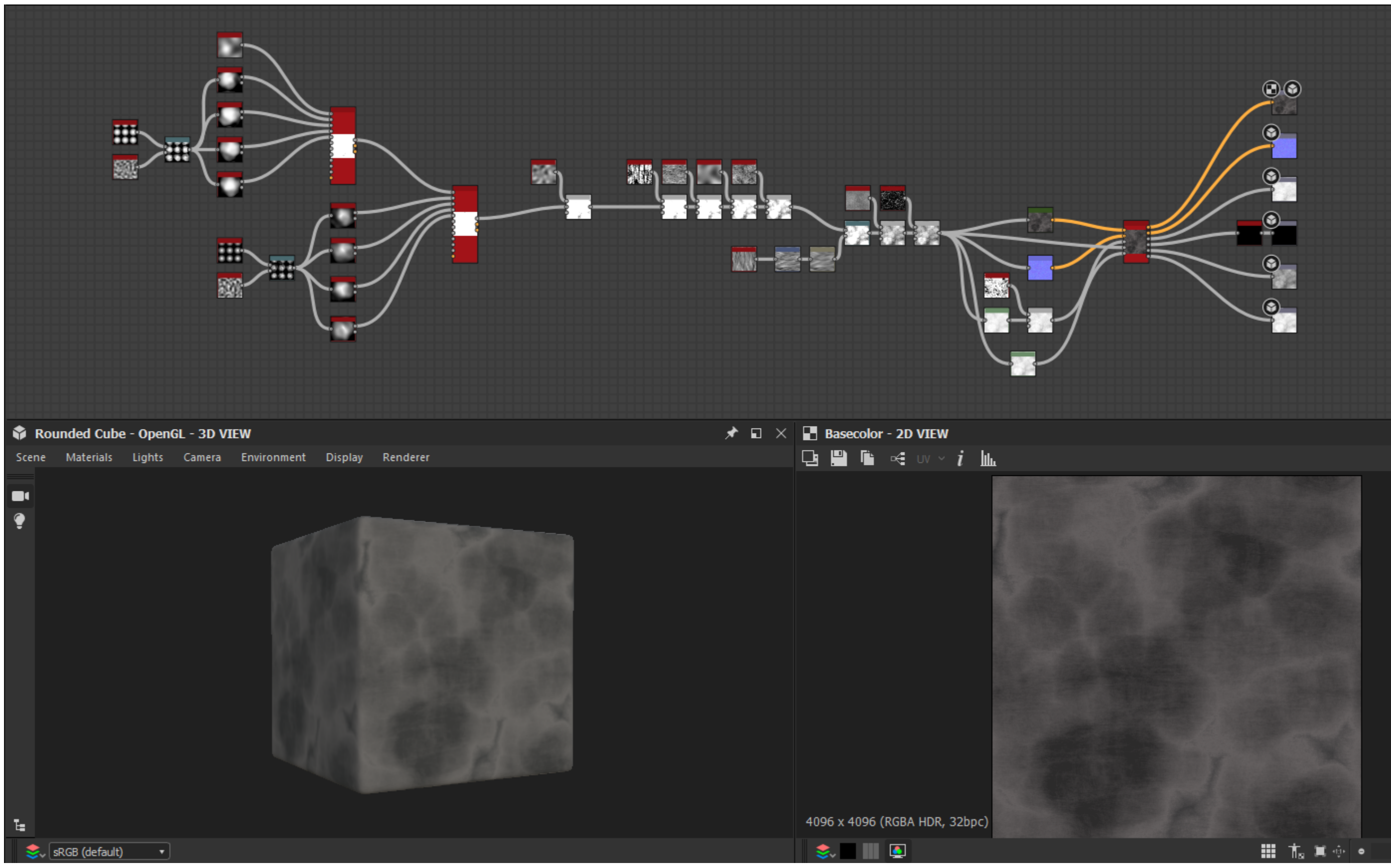


Figure 39: Rock Substance Designer graph

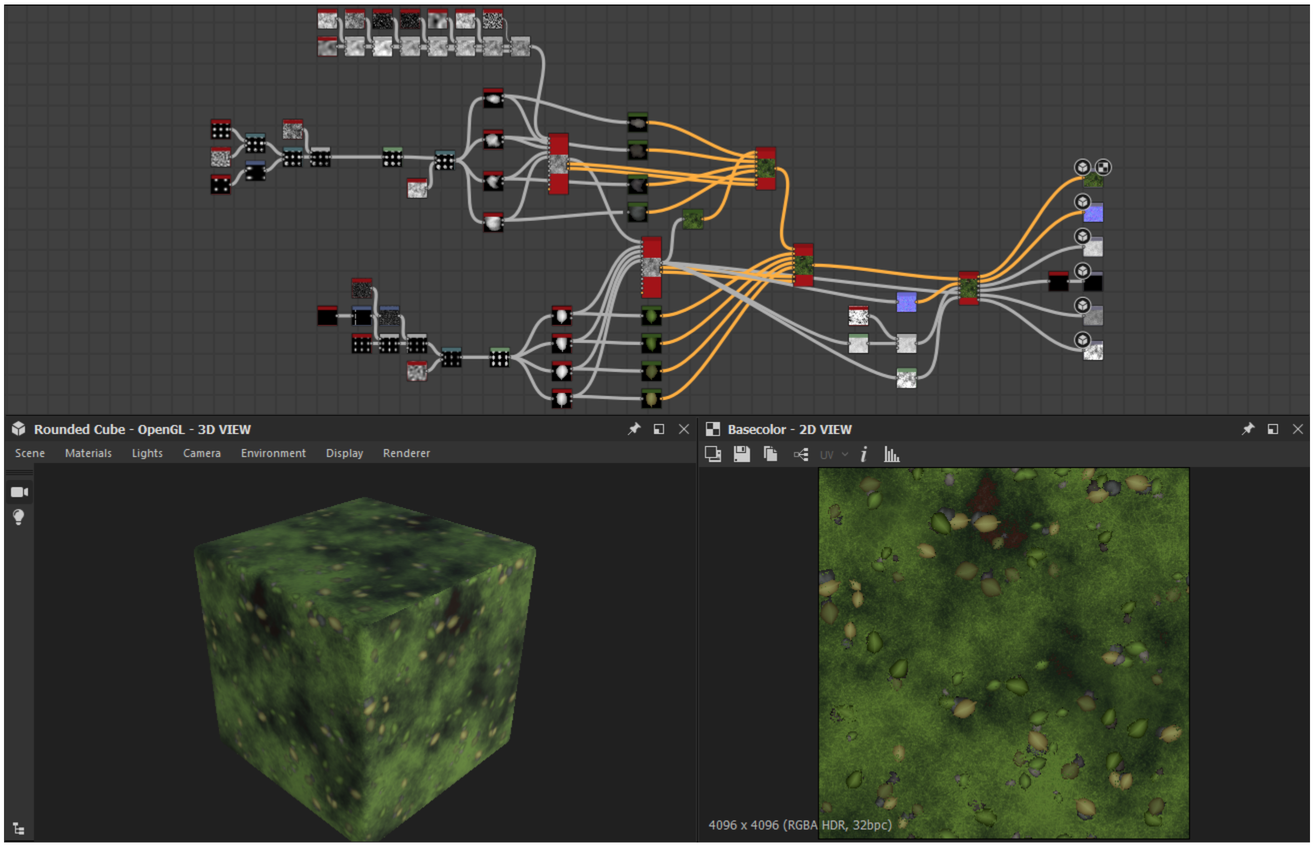


Figure 40: Sparse grass Substance Designer graph

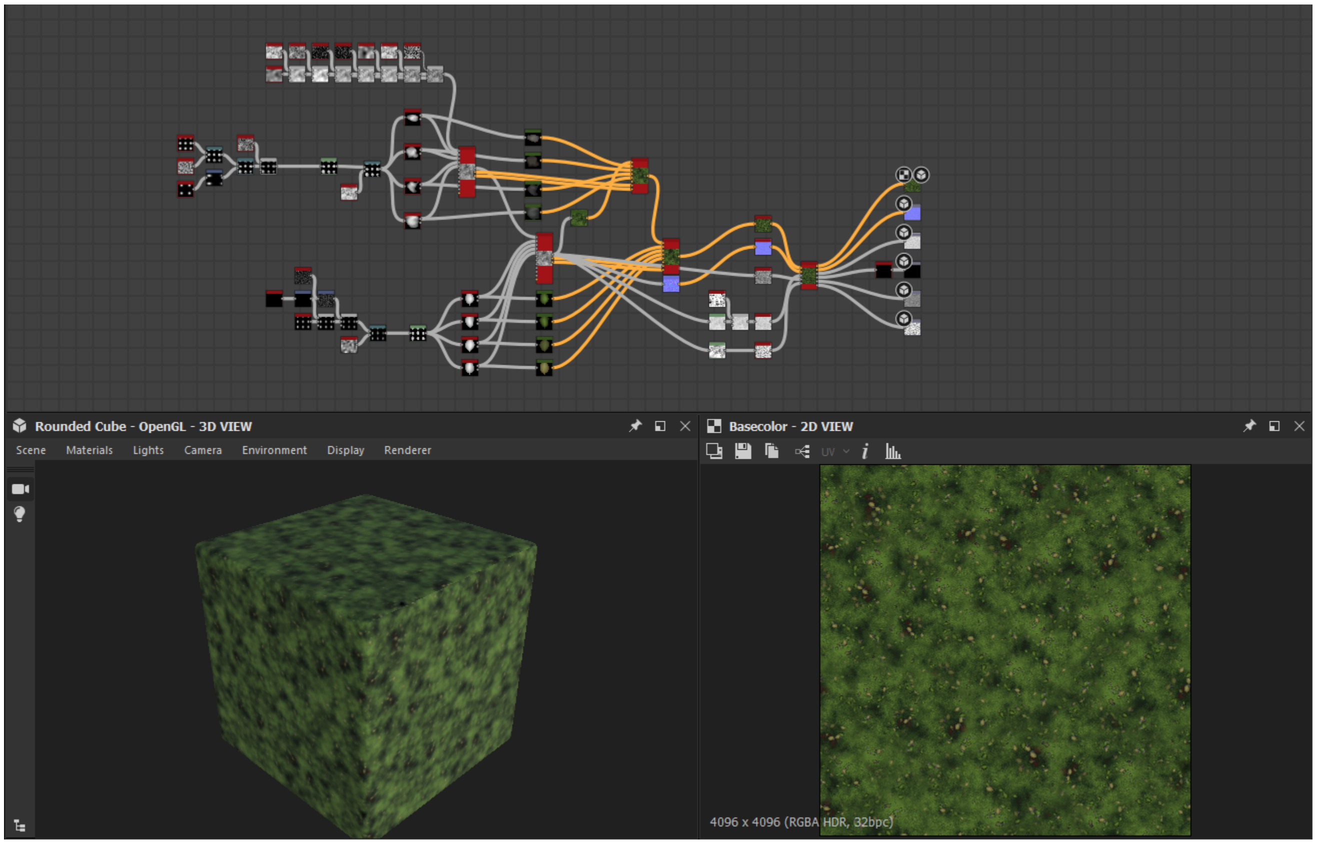


Figure 41: Medium scale tiling Sparse Grass graph

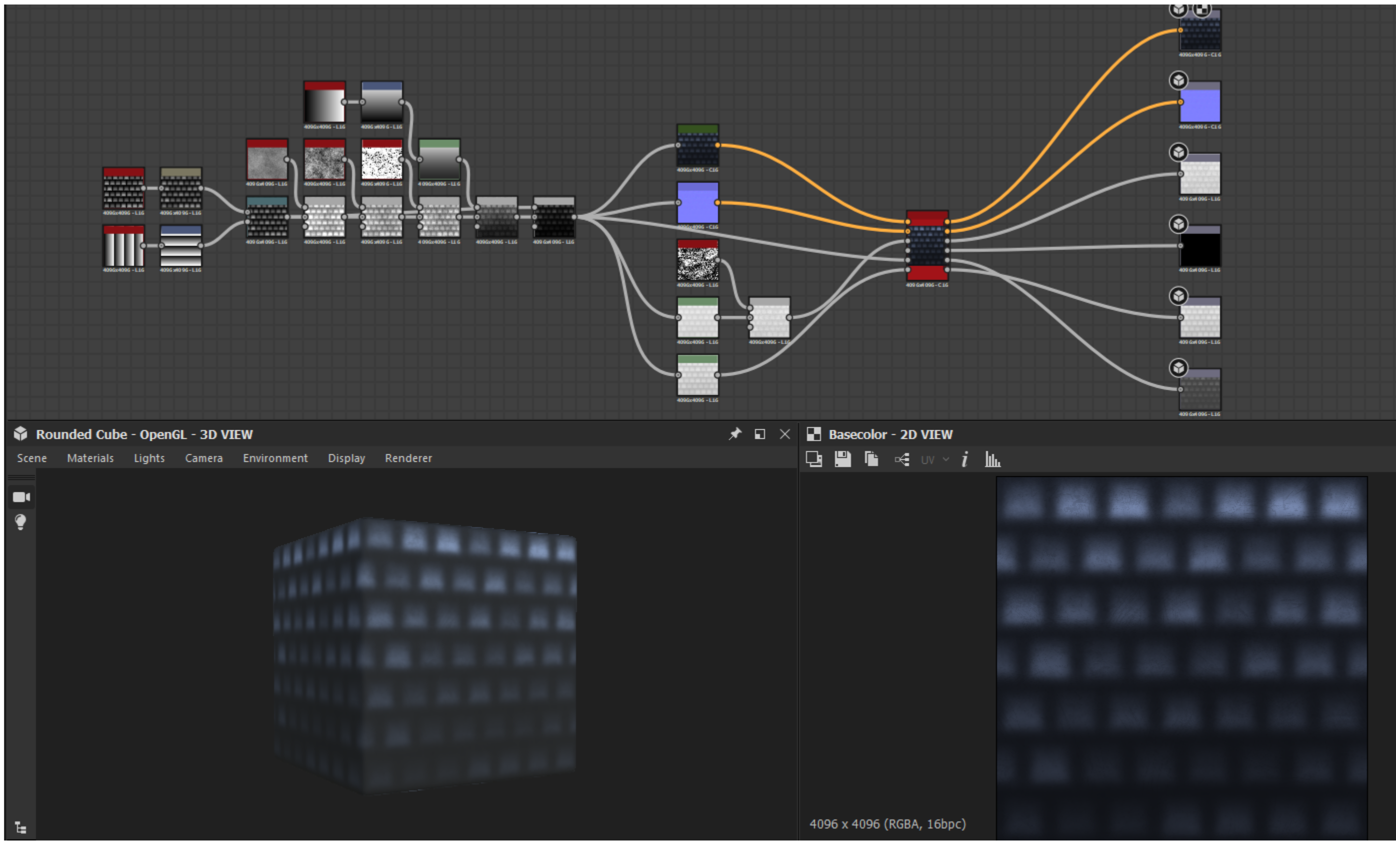


Figure 42: Tile roof Substance Designer graph

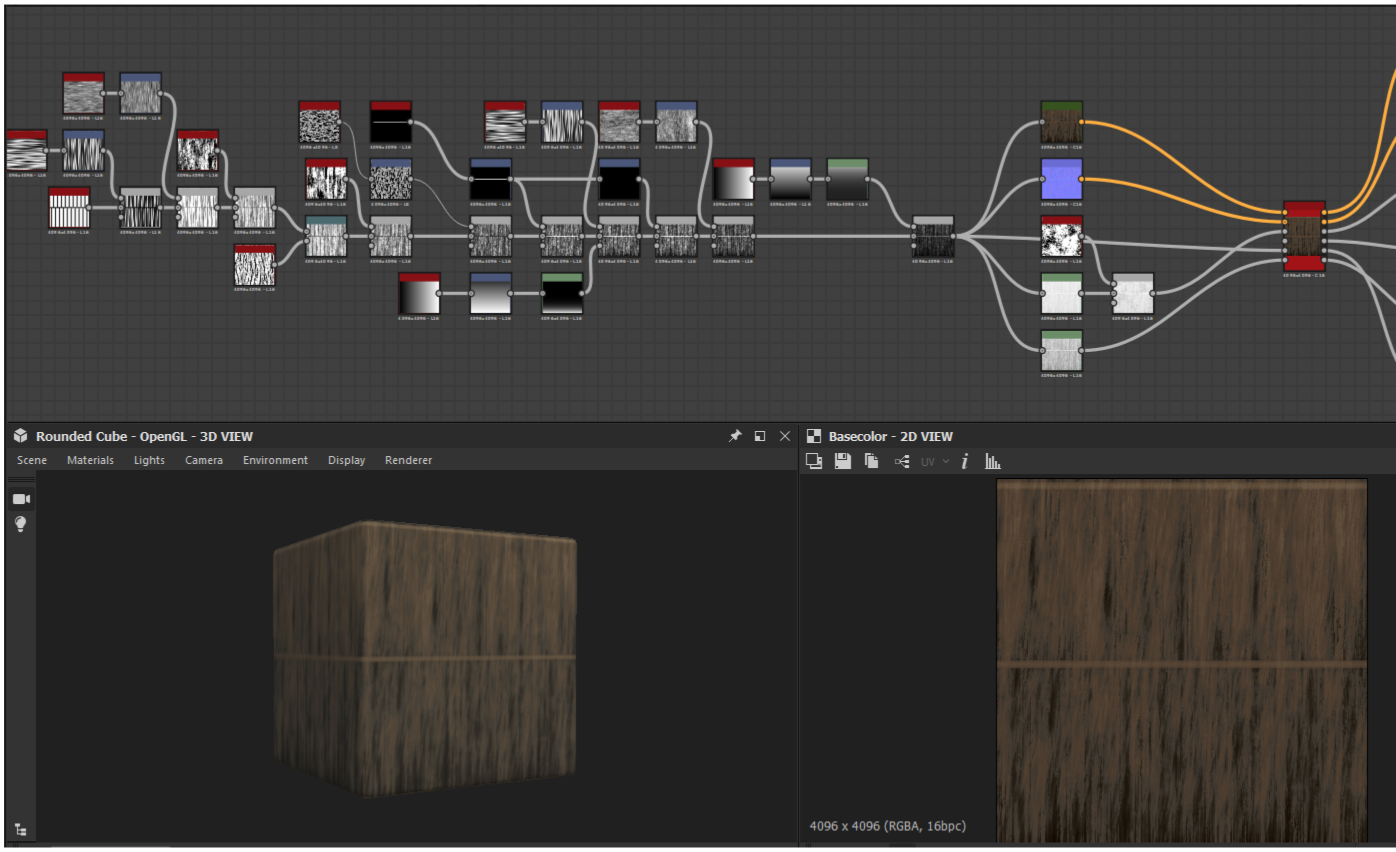


Figure 43: Thatch roof Substance Designer graph

Appendix C Relevant Terminology

A height-field is a data structure in which each point on a grid contains a value representing the distance from the base to that point on the grid. By displacing points on the grid by their values, high density terrain representation can be stored in very small raw floating point image formats.

RAW image files are files that store information in a matrix as indexed floating point numbers. This allows a black and white image to be stored as simple floats instead of 3 or 1 channel color images. 16 bit refers to the floating point precision, or how memory is reserved for maintaining accuracy in floating point numbers after the decimal point.

Noise in 3D art can refer to several things. In rendered images, noise typically refers to graininess in the render where rendering has not fully resolved. In authoring textures and creating terrain, noise refers to patterned randomness created by assigning an area of pixels random values in a given range following some pattern, equation, or other algorithm.

PBR Material workflow references using texture images in a format that functions for Physically Based Rendering. This typically refers to a workflow using either metallic and roughness maps or gloss and specular maps. Materials created in this project were created using a metallic and roughness workflow, and each material generated a base color, normal, roughness, metallic, ambient occlusion, and height map. Example maps and the construction process behind them are available in Appendix B.

Texture baking refers to the process by which new maps representing geometric features are generated for a mesh to assist in the texturing process. The texture baking will often involve processing a higher detail mesh to translate its detail qualities to a more optimized version.

In the context of this project, color space refers to the way colors are stored in an image file for later representation.

Bibliography

[1] M. Aguilar-Moreno. Aztec architecture - part 1. https://citeseerx.ist.psu.edu/document?repid=rep1type=pdfdoi=21
2008.

[2] V. Charles and K. Carl. *Romanesque Art.* Parkstone International, 2016.

[3] S. Toby Evans and J. Pillsbury. *Palaces of the Ancient New World: A Symposium at Dumbarton Oaks, 10th and 11th October 1998.* Dumbarton Oaks Research Library and Collection, 2004.

[4] G. Kelly and H. McCabe. A survey of procedural techniques for city generation. *Institute of Technology Blanchardstown Journal*, 7(5):86–130, 2006.

[5] G. Kelly and H. McCabe. A survey of procedural techniques for city generation. http://www.citygen.net/files/citygen$_{g}dtw07.pdf$, 2007.

[6] J. Seok Kim, H. Kavak, and A. Crooks. Procedural city generation beyond game development. *SIGSPATIAL Special*, 10(2):34–41.

[7] N. Kim. Aesthetics of romanesque architecture. *The Journal of Aesthetic Education*, 55(1):90–108.

[8] K.Perlin. Making noise. noisemachine.com, 2007.

[9] J. Liu and Z. Ke Wu. Rule-based generation of ancient chinese architecture from the song dynasty. *Journal on Computing and Cultural Heritage*, 9(7):1–22.

[10] B. Martinez. Building worlds with houdini / benoit martinez / houdini hive paris. https://www.youtube.com/watch?v=bQ$_{U}1_{M}VKJQ$, 2019.

[11] G. Voronoi. Nouvelles applications des parametres continus a la theorie des formes quadratiques. *Journal für die Reine und Angewandte Mathematik*, (133):97–178.