Procedural Tibetan Monastery Generator

Eddie Huang

Advisors: Norm Badler and Adam Mally

University of Pennsylvania

ABSTRACT

Being able to quickly generate variations of complex assets sharing a similar style is a problem that many modeling artists face when tasked with filling in a scene. Specifically, if the artist was tasked to create a scene with many buildings, individually modeling these buildings and manually adding variation and ornamentation to them would be very time consuming and inefficient. Instead, procedural generation can be utilized to help streamline and optimize this process.

The goal of this project is to develop a Houdini Digital Asset (HDA) capable of generating variations of Tibetan monasteries. Since these buildings are already quite complex in structure and have many different stylistic features, having a generator that can quickly create variations of these buildings could be very useful. The user will also be able to go in and adjust some of the building parameters to meet their aesthetic desires. Overall, this project aims to create a robust generator that generates monasteries similar to that of Tibetan monasteries in real life.

1. INTRODUCTION

Problem Statement. There are a wide variety of architectural styles in the world and if an artist wanted to model many different buildings with the same overarching style it would be inconvenient to have to go in and remodel certain parts or create new models entirely. The problem I plan to tackle with this project is how to properly parametrize and constrain the stylistic features of Tibetan Buddhist monasteries such that procedural generation is possible. The unique structure and layout of these types of buildings will be a challenging task to capture within a generator.

Motivation. From what I have seen so far, although there are various buildings and temple generators that have already been done, there are no exact procedural generators for this specific building type that I have been able to find. Procedural generation is also a challenging problem to tackle in general, as determining and properly constraining the unique features of the subject of the generator is quite challenging in of itself.

Proposed Solution. The way I plan to tackle the procedural generation for this problem is to utilize Shape Grammars and L-Systems as the basis of the generator. By using a shape grammar or l-system to generate a basic foundation of the monastery, it will be very easy to create variations of the overall structure by adjusting the number of iterations of the grammar, or making adjustments to the rules of the grammar itself. Then, I can create adjustable assets to be mapped into the overall base structure with certain constraints in order to add in the details of the monastery while also allowing the user to fine tune the appearance.

Contributions. This project makes the following contributions to the field of computer graphics:

- Allows artists to quickly generate complex building structures
- Allows artists to adjust certain stylized parameters for more variation and artistic freedom
- Allows production teams to potentially export these assets out to other computer graphics software for usage in scenes for games or films

1.1 Design Goals

The target audience of my project is anyone who has visited these monasteries or is interested in this style of buildings. In a more practical sense the project could potentially be used in scenes for games or films if they match the aesthetic and setting of the scene. For example, if the given scene takes place in a plain, forested, or mountainous landscape, this procedural generator could be very useful as a way to quickly generate various monasteries to add to the scene.

1.2 Projects Proposed Features and Functionality

There are several features and functionalities that will be implemented throughout this project to allow for the procedural generation of the monasteries. The overall features of the project will be the following:

- Houdini Digital Asset for the Monastery Generator
- Adjustability of Stylized Parameters

Some algorithms and computer graphics techniques that will be utilized to implement these features include the following:

- Shape Grammars
- L-Systems
- Procedural Noise Functions

2. RELATED WORK

The core topics and techniques of this project have already been researched and used in many different settings. L-systems have been used in a variety of computer graphics settings when creating recursive, organic forms like plants, rivers, and trees. Shape grammars have also been used in building generation in the past. Many procedural generators have already been created in Houdini as well, so doing some research about these different generators will help me get started as well as have a reference point for evaluation of my generator at the end of my project.

2.1 L-Systems and Shape Grammars

There are many papers available discussing the main concepts of L-Systems and shape grammars as well as their uses and implementations. One paper I looked into focuses on the procedural modeling of cities through a variety of methods, including L-Systems [MP01]. In this paper, L-Systems are used as a way to generate road networks for a city and also divide up lots for where buildings will be placed. The buildings are also created using a parametric, stochastic L-System, which iterates some number of times and then modifies the base shape of the building that has been created from the roads based on the output of the L-System. The L-System contains various rules and modules for scaling, translating, extruding, and other geometric operations to transform the base shape into a building.

Another paper I found provides an introduction to shapes and shape grammars that has helped me get a better conceptual understanding of the main concepts [Sti80]. The paper starts out by discussing shapes, boolean shape operations and shape transformations to build up some basic knowledge needed to discuss shape grammars. The paper then goes into depth about the constituents of a shape grammar such as the starting shape, shape set, the symbols of the grammar, and the rules of the grammar and walks through some examples of shape grammars.

2.2 Houdini

Since I am a novice with using Houdini, understanding what others have done with it in relation to my project was helpful in terms of knowing how to get started. A set of video tutorials covering the basics of houdini, objects, and rendering helped me get a basic understanding of the software [Say19]. I also looked into a basic tutorial on creating a generator for a house that was on the SideFX website that explained the steps of creating a building from a basic shape to building the walls and windows to building the roof [Est19].

A couple of articles on 80.LV discuss how different technical artists approached procedural building generation for more specific use cases. In one case, tech artists at Aesir Interactive talked about how they used Houdini to procedurally generate roads and buildings for the open world environment of the game Police Simulator: Patrol Officers [BR22]. The technique they used to generate the buildings for their game was using various tilesets to construct their buildings using specific tileset patterns to make sure the buildings look correct. I can leverage this idea in my project as well by modeling out some assets for the walls, windows, and tapestries of the monasteries, using an L-System grammar to generate how these pieces should be laid out and then filling them in with the assets. Another article uses a similar method for generating buildings matching the architectural style of a scene from the movie “Casino Royale” [Cri19]. This article gives some helpful tips on how to procedurally place the windows on the buildings as well as some optimization suggestions and tips on properly managing the Houdini Node Network to stay organized.

3. PROJECT PROPOSAL

A Houdini Digital Asset (HDA) will be created in order to procedurally generate monasteries similar to the style of Tibetan Buddhist architecture. Various artistic controls over aspects of the buildings will be granted to the user to allow them to fine tune details of the monastery.

3.1 Anticipated Approach

The basis of the generator will be a stochastic L-System that will expand over a specified number of iterations. The rules of the L-System will contain rules mapping certain symbols of the grammar to specified shapes and other symbols will act as different ways to expand the grammar. Specifically, these expansion rules will need to be well-constrained in order for a reasonable base shape of the monastery is created. The main features of the generation I am planning to consider are the layout of the different parts of the building, the placement of the windows, the number of windows, and the overall shape of structure. With these considerations in mind, I will then procedurally distribute modeled assets across the base shape of the monastery in order to add the details. The user will be able to modify certain attributes of buildings like the color of various parts, the number and placement of windows, control over the overall shape of the building and other controls to incorporate some more variation to the generated monasteries as well as meet the user’s artistic needs.

3.2 Target Platforms

The main software I will be using for this project is Houdini to both model the assets as well as create the generator.

3.3 Evaluation Criteria

In terms of evaluation, one way to evaluate my project would be to directly compare it to the reference images I am using to see whether I have correctly captured the stylistic features of Tibetan monasteries. Another way to evaluate my project would be to compare its generated buildings to other procedural building generators that have already been implemented in Houdini. I could compare the output monasteries of my generator to the output of other house/castle/temple generators that have already been completed and demoed online to see whether my generator is as versatile, efficient and powerful as other generators out there.

4. RESEARCH TIMELINE

Project Milestone Report (Alpha Version)

- Completed all background reading on shape grammars and procedural generation in Houdini
- Basic functioning generator with low poly models
- Clearly defined list of stylized parameters for generator
- Complete collection of reference images

Project Milestone Report (Beta Version)

- Properly constrained generator with well-modeled assets
- Adjustable parameters for generator

Project Final Deliverables

- Complete HDA of the generator
- Video Demo
- Renders of monasteries
- Documentation

5. Method

This procedural monastery generator utilizes a Houdini Node Network that starts with a ‘L-System’ node as the basis of it all. While I was originally planning to utilize a stochastic L-System to generate the monastery, I found that there were various deterministic L-Systems out there that resembled the shape of certain monasteries, and thus decided to utilize those instead. I also found this method to be more easily translatable and easier to implement in Houdini than the approach I had anticipated before. The L-Systems used in this project are specifically chosen such that they generate multiple squares spread along a plane so it can be used as a floor of the entire structure. In order to choose between the various L-Systems of the generator, a ‘Switch’ node is connected to the Parameter Interface so that the user can choose which L-System they would like to use. Once the L-System has been created, a ‘Fuse’ node is used to combine all of the points together and a ‘PolyFrame’ node is used to correct the normals. Then, a ‘Grid’ mesh is copied along all of the points of the L-System to create the floor of the structure. From there the network begins to divide itself into different subnetworks that each generate specific parts of the structure. These parts are then merged together at the end using a ‘Merge’ node, which results in the overall output of the generator.

5.1 Frame and Parameters

To create the frames along the vertical edges of the base floor of the monastery, a ‘Dissolve Flat Edges’ node is used to eliminate all of the inner edges of the L-System base shape so that only the edge along the perimeter of the shape remains. Then a‘PolyFrame’ node is used once again to correct the normals. Then a ‘Box’ mesh is scaled along the X and Z axes to achieve the correct shape of the frame, and then it is scaled and translated along the Y-axis based on a “Base Height” parameter linked to the Parameter Interface so that it properly grows and shrinks accordingly with the overall height of the Monastery. The ‘Box’ mesh is then copied along the points along the perimeter of the shape using a ‘Copy to Points’ node to create the vertical frame of the structure. Finally, the Color attribute of the vertical frame is linked to the “Frame Color” parameter in the Parameter interface to allow the user to adjust the color as they please.

5.2 Base and Parameters

A ‘PolyExtrude’ node is used with a distance linked to the ‘Base Height’ parameter in the Parameter Interface to extrude out the base layer of the structure. The subdivisions of the mesh are also linked to the ‘Base Height’ parameter so that each level of extrusion results in a corresponding level of the structure where the walls can be placed. Then an ‘Attribute Wrangle’ node is used to utilize VEX code to generate a point cloud that is used to count the number of neighboring points around each point. Conditionals are then used to limit the procedure to only points with a surface normal facing positively along the Y-axis, a height equal to the ‘Base Height’ parameter, and less than six neighboring points in order to create a group of points that represent the top points that are not along the edge. of the structure. This group of points is then promoted to primitives using a ‘Group Promote’ node. Then this group is further reduced by using a conditional along with modulo operations based on the X-position and Z-position of the primitives to select out primitives that can act as the second-layer of the structure. The user is then able to control the overall size of the selected region by linking the conditional statement to the ‘Top Building Size’ parameter in the Parameter Interface. From there, another ‘PolyExtrude’ node is used with its distance linked to the ‘Top Height’ parameter in the Parameter Interface to extrude out the second layer of the structure. The subdivisions of the extruded layer are once again also linked to the ‘Top Height’ parameter so that each level can have walls placed. From there a ‘For Each’ node is used to iterate over each primitive of the mesh to fuse the geometry and correct the normals, completing the base shape of the monastery with two floors.

5.3 Walls and Parameters

To determine the possible places in which windows and doors can be placed along the structure, various ‘Attribute Wrangle’ nodes are used to utilize VEX code. A ‘Select’ node is also linked with the ‘Window Placement’ parameter in the Parameter Interface so that the user can choose between three different methods of placing the windows. The first method simply places no windows along the monastery and uses thresholding along with Perlin Noise to distribute the doors on points that are on the bottom layer of the monastery, while all of the other points are mapped to standard walls.

The second method determines where the windows are able to be placed using a point cloud method. First the ‘Base Height’, ‘Top Height’, and ‘Window Layers’ parameters are accessed from the Parameter Interface and stored as variables. From there, the points that can be windows are determined by comparing the Y-position of the points to determine whether they are in between the value range of the “Base Height” and the “Base Height” + “Window Layers”, and likewise for the windows of the second layer of the structure the Y-position of the points are checked to be within the value range of the “Top Height” and the “Top Height” + “Window Layers”. Once all the mid-points have been selected, certain points that are tangent to the edge of the structure are culled from the group by utilizing a point cloud to count the number of neighboring window points and eliminating the points from the window group if there are too many neighboring window points. Once again, in order to determine what points can be the doors of the structure, Perlin Noise is used along with thresholding and all of the other points that do not fall under the window or door group are put into the wall group.

The third method of distributing the windows utilizes the same logic as the second method to determine all of the midpoints of the monastery and assigns them to the window group, but instead of using a point cloud to cull out certain points, the windows are distributed randomly. To do this, the points in the window group are processed randomly in a ‘Sort’ node and a ‘Group by Range’ node whose range filter is linked to the ‘Window Amount’ and ‘Window Cluster Size’ parameters in the Parameter Interface so that the user can control the amount of windows that are placed randomly, with the cluster size determining how intensely the windows can be grouped together while the amount controls the overall amount of windows shown relative to the cluster size. The doors are once again determined in the same way as the previous methods, and the rest of the points are assigned to the wall group.

Once all the points have been assigned a group, different models are copied to the points to create the walls of the structure, as walls are copied to points in the wall group, windows are copied to points in the window group, and doors are copied to points in the door group. The models that are copied can be selected using the “Wall Type” and “Window Type” parameters in the Parameter Interface, which are linked to ‘Switch’ nodes that allow the user to easily change between different window and wall designs that suit their needs. The user could also easily add their own more complex models to these ‘Switch’ nodes if they wanted to create their own variants. Each of these models are created using ‘Group’ nodes, ‘PolyExtrude nodes’, and ‘Boolean’ nodes to cut out and extrude the desired shapes. Finally, the color of the walls, windows, and doors can be controlled using the corresponding color parameter in the Parameter Interface, which is accessed in ‘Attribute Wrangle’ nodes that also change the color attributes of the points to the desired color.

5.4 Roof and Parameters

For the roof, a ‘Split’ node is used to split the sides of the base structure from the top face to easily access the roof of the structure. A ‘Dissolve Flat Edge’ node is used once again to erase all edges except the edges along the perimeter. Then a ‘PolyExtrude’ node is used to create an inset within the roof layer that creates an inner edge following the shape of the perimeter of the mesh. ‘Group’ nodes are then used to differentiate the roof of the base layer from the roof of the second layer. For each layer, a ‘PolyExpand2D’ node is used with an arbitrary off-set to create roof-like, diagonal edges along the plane of each roof. An ‘Attribute Promote’ node is then used to promote an edge distance attribute from vertices to points, and then this attribute is utilized in an ‘Attribute Wrangle’ node to adjust the Y-position of the points based on their corresponding edge distance, Y-position, and a “Slope” parameter that is accessible in the Parameter Interface. This generates a smooth roof for the monastery that is then fused together using a ‘Fuse’ node. This smooth roof is then subdivided into smaller pieces using a ‘Divide’ node and then a ‘Attribute Expression’ node is used to define an ‘up’ attribute which is then used in a ‘Copy’ node to copy a ‘Box’ mesh to each of the subdivisions of the roof. The ‘Box’ mesh can be scaled along the X and Z axes by using the corresponding ‘Roof Tile Size’ parameter for the upper and lower roofs in the Parameter interface. The resulting roofs are then merged, fused, and colored using ‘Merge’, ‘Fuse’, and ‘Attribute Wrangle’ nodes that once again access the desired color for the roofs from the Parameter Interface.

6. RESULTS

Extensive Houdini Digital Asset

The final product from this project is a complete Houdini Digital Asset which can procedurally generate structures similar to that of Tibetan monasteries at a speed considerably faster than what one would be able to achieve with 3D-modeling. While the overall detail of these structures are definitely lower than that of actual Tibetan monasteries, the general structure and basics of the monasteries are there. Through the usage of various L-System bases, various structural designs can be achieved quickly, and the scale of these designs can quickly be adjusted by changing the number of generations used in the L-System base so that the desired size of the user can be achieved.

Upon opening the project file, the user is able to access the tool by entering the “MONASTERY” object that is visible at the object level of Houdini. From there, the user will be presented with the Parameter Interface where they will be able to adjust the stylized parameters of the tool. The “Monastery Type” parameter allows the user to switch between the different L-System bases of the tool to create a variety of structures, while the “Generations” parameter allows them to change the number of iterations of the selected L-System. Other parameters for controlling the overall base shape of the monastery include the “Base Height” for the height of the base layer, the “Top Height” for the height of the top layer”, the “Top Building Size” which controls the size of the second layer buildings, as well as the “Wall Type” parameter that allows the user to change between pre-set wall models. Coloring for the overall structure, frame, windows, window frames, doors, and roofs are adjustable by the user as well.

Parameters controlling the roof include the “Roof Slopes” parameter whose first input controls the slope of the upper roof while the second input controls the slope of the lower roof, the “Roof Tile Size” parameters that control the X and Z dimensions of the upper and lower roof tiles, and the “Roof Divisions” parameters that control the number of subdivisions for the upper and lower roofs. Parameters controlling the windows include the “Window Placement” parameter that determines the ways in which the windows are placed along the monastery as described in the Method section, “Window Type” parameter that allows the user to switch between different window models, the “Window Amount” parameter that allows the user to adjust the concentration of the windows, the “Window Cluster Size” parameter that determines the max size of the randomly distributed clusters of windows, and the “Window Layer” parameter that allows the user to adjust the number of layers of the windows. All of the parameters for this tool are attempted to be constrained to reasonable ranges. However, some of the L-Systems in the tool expand quite rapidly and when combined with many subdivisions along the roof, it can be very computationally expensive and thus takes a very long time for the structure to be generated. The different models used for the walls, windows, as well as the L-Systems used as the bases can easily be extended upon by the user by simply adding new models or L-System nodes that are compatible with the existing architecture of the network.

Examples of various structures generated using this tool, as well as a video demo of the various parameters can be seen in the Video Presentation of this project.

7. CONCLUSIONS and FUTURE WORK

To showcase the capabilities of this tool, I experimented with generating various structures using various combinations of the adjustable parameters to show the diversity of the structures that can be generated using this tool. In general, any monastery of a more standard house size can be generated in the matter of seconds while monasteries meant to be of the scale of a city take roughly five minutes to generate. Using a basic rendering set-up in Houdini, these structures can be rendered out in a matter of seconds as well. Even though the number of adjustable parameters are still fairly limited for this tool, the number of possibilities for structures created with the existing parameters is still quite large and would definitely meet the needs for structures used in the background of a scene. While some of the structures may be somewhat repetitive, this generator still allows for a much quicker work flow of generating assets compared to modeling these structures from scratch. If a higher level of detail and decor is desired, these monasteries could be imported into a 3-D modeling software to be worked upon and detailed.

If I had 6 more months to work on this project, I would definitely add more parametrized stylistic features to the generator to give the user even more control over the output buildings, as I wasn’t able to capture some of the aspects of Tibetan monasteries I initially wanted such as the surface decoration, connecting hallways, and more variance in height. This would definitely make the generator more robust and versatile since greater variation and greater detail would be achieved with the tool and artists would not need to go in to add their own details as often. Another big improvement that could be done in this project is polygon optimization, as the structures created by this generator are quite large and polygon intensive, especially the roof, so culling out excess geometry and improving the placement algorithms used would go a long way in improving generation time and render time. Another way I was thinking of building on this project was to make it more general so that various buildings of different architectural styles could be generated as well. It would be difficult to adjust the parametrized features over to different architectural styles, but this would definitely make the generator much more powerful and useful in a wide variety of urban settings and scenes.

Another extension of the project would also be to expand upon the interior design of the structures, which would make the tool much more useful in game development as the assets generated would actually be interactive and enterable rather than purely meant for aesthetics in the background. This would also pose new challenges in ways to distribute interior objects, perhaps using noise, how to shape and connect inner rooms, using graph algorithms, and how to parametrize stylistic elements of interior objects.

References

[BR22] BRASIN P., REICH J.: Creating an open world environment in Police Simulator Patrol Officers. *80.LV*, Jan. 2022.

[Cri19] CRISTEA, R.: Procedural generation of architecture and props. *80.LV*, June. 2019.

[Est19] ESTELA M.: Building Generator. *SideFX*, Mar. 2019.

[MP01] MULLER P., PARISH Y.: Procedural Modeling of Cities. In *ACM SIGGRAPH* 2001, 2001.

[Say19] SAYED, M.: Houdini isn’t scary.n *SideFX*, Dec. 2019.

[Sti80] STINY, G.: Introduction to shape and shape grammars. In *Environment and Planning B*, 1980, vol. 7, pp. 343–351.

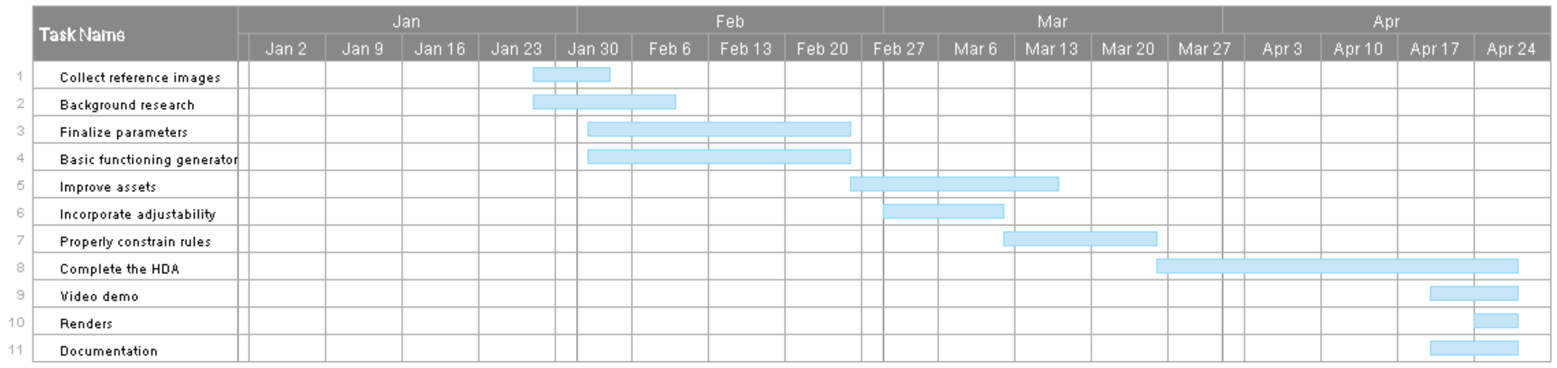

Figure 1: *GANTT Chart displaying timeline for the project*