Senior Project - Procedural Plant Generation with FloraGen

Chris Wright
Rich Murray, advisor
California Polytechnic State University - San Luis Obispo
12/06/2021
# Table of Contents

**Introduction**  
Abstract 2  
Project Overview 3  
Goals and Objectives 3  
Primary Goals 3  
Stretch Goals 3

**Background Research** 5  
Related Works 5  
Blender Python API 5  
Effective UI practices 5  
Efficient Data Management Practices 6

**Project Requirements** 7  
Use Cases 7  
New Game Developer 7  
3D Computer Graphic Artist 7

**Implementation** 8  
Add-on Architecture 8  
UI Development 8  
Generator Development 10  
Implementation Issues 14

**Testing Feedback** 15  
Add-on Test Results 15

**Conclusion** 16  
Findings 16  
Future Work 17  
Senior Project Quarterly Schedule 17  
Project GitHub Link 17

**Bibliography** 18
Introduction

Abstract
A particularly challenging aspect of game design revolves around asset creation. Often new developers become lost in nuances and time investment required to learn 3D asset creation software. While many game development platforms provide an internal asset store, these assets are often expensive or limited. These assets restrict the flexibility for a creator to fully control the product they create. A critical asset type in many games is flora. Good looking trees and plants add environmental variation when added properly.

In this project, I designed and implemented an add-on dubbed FloraGen to the 3D modeling software Blender in which users can easily create a plethora of realistic flora for use as 3D assets. The add-on employs procedural generators to create varied and highly customizable assets for the purpose for use as models. This add-on utilizes varied approaches to generators depending on the complexity of the asset required. A testing session was conducted with Computer Graphic artists to acquire feedback and performance metrics.
Project Overview

Game development is a long and arduous process that requires a combination of skills to excel at. Fidelity in logic and engine code is imperative. Just as important is a good base in Music and Visual Assets. While larger teams have individuals or full teams focusing on a single aspect or part of an aspect in the development process, individual or “indie” development studios may not have the luxury to diversify their team skill set. Environmental design is often limited by the quality and quantity of assets available to the developer.

Blender is the largest open-source 3D graphics program. It's accessibility and support makes it a strong choice for indie developers who are often constrained by limited monetary and time budgets. Additionally Blender supports community modifications in the form of “add-ons”. The add-ons provide additional support and allow for easy modification of the Blender platform.

In this project, I developed a Blender add-on to automate the process of modeling a plethora of flora types to allow for easy environmental design. This application is designed to be easy to use at every skill level in Blender. A major design consideration I took with FloraGen was to effectively balance ease of use with creative freedom. The algorithms support a large degree of customization in an intuitive framework. The add-on supports two primary modes of operation. The first is an individual mode where a single object is created. The second is a tileset mode where a group of objects is generated. These two modes of operation allow a greater level of control over the models generated by the algorithm.

Goals and Objectives

Primary Goals
For this project, my primary goal was to deliver a fully operational add-on with a set of features that would lend itself to my expected use cases. I planned to create an add-on that employed a series of generators that would be powerful and flexible. These generators would be able to effectively output 3D models that were both resource efficient and visually pleasing. The add-on would also support modularity where new generators could be added, enabled and disabled without affecting any of the existing generators to allow for scalability. The project would support a functional UI that elevated generator properties to the user allowing a fine degree of control over the resulting flora created.

Stretch Goals
A major stretch goal of the project was a second set of generators, the tileset generators. These generators would be designed to create larger scale tileable environments. The tileset generators would utilize noise maps to control the concentration of generated models. Additionally the generators would support efficient scaling, utilizing model pools which copy previously generated meshes to limit processing time.
A second major stretch goal of the project was to refine the UI and organize the generators and their properties into a more intuitive layout. This implementation would utilize nested, collapsible windows structured by property type.
Background Research

Related Works

There are a plethora of existing projects that attempt to fill the niche of environmental design in a couple different forms. The two main forms are asset repositories or asset generators.

Unity Asset Store

On the unity asset store there are over 1000 results for tree assets. A small portion of them are free. These assets range from high resolution models and textures to low polygon count trees for different aesthetics. A major limitation with the asset store is that there is often a lack of variation in the trees in asset packs. Many free asset packs contain 1-4 different tree models. Using multiple asset packs leads to incongruity in the environment.

Modular Tree Add-on

This is an existing Blender add-on that adds modular trees. These modular trees are controlled through Blenders node editor API. The modular tree add-on adds a new node editor that controls the generation of a tree through entirely modularized nodes. There are pros and cons to using this generator. It is not beginner friendly and requires a significant amount of effort to produce a tree. The upside to this generator is that an experienced Blender user will be comfortable navigating the node based system. Additionally there is a significant degree of customization options available to the user.

Vegetation

This add-on for Blender combines customizability with a comprehensive library. There is a large selection of trees available in four seasonal variations. There is a great deal of variation and the trees are optimized for blender. One limitation is the price. Another is that the models have visual variation but not for the geometry itself. Overall it is a powerful, photorealistic add-on for CG artists.

Blender Python API

A major focus of research for this senior project was Blenders Python API. Blender contains a library of Python modules that facilitate add-on development. This library titled BPY served as the main form of communication between the add-on and the Blender instance. All of Blender's significant editor operations execute a BPY library function allowing for the add-on to execute many editor tasks by calling the same library function.

Effective UI Practices

Effective UI design is a major factor in user experience when using a product. A bad UI can easily lead to confusion. For this project I aimed to strike a balance between effective UI design and time invested in its development. I emulated Blenders existing panel design for its operators when creating my own panels. Additionally researched how to implement collapsible tables.
within Blender's existing UI layout so that information is organized in a neat and effective manner.

**Efficient Data Management Practices**

One of the most critical aspects of FloraGen was its computational efficiency. The add-on should run as close to real time as possible. In this regard it is critical to employ efficient data management practices with the massive datasets employed by the add-ons generators. The more complex generators in this add-on iteratively generate subsets of the mesh to then be linked together. Each of these iterations needs to be as simple as possible as any overhead in the iterative process drastically increases the runtime of the program. Additionally, some mathematical processes are far more resource intensive than others, processes like division and sinusoidal calculations are far more expensive than addition and multiplication. In this project I strove to limit these calculations as much as possible leading to data storage approaches like least recently used caching and research into more efficient geometry algorithms that employed trigonometric functions.
Project Requirements

The main requirement for the project was to generate usefulness to my main use cases. I developed my feature-set to facilitate this requirement. The design choices for this project were in an effort to increase its usability to developers and graphic artists.

Use Cases

New Game Developer

The project was originally turned towards the new game developer. This game developer would be entering the field or interested in the field of game development. This developer would be in college classes and clubs or currently hold an entry level job. Due to a busy work and school life, the Developers free time is quite limited. As this Developer is working on their first game product, the time commitment to creating a quality product is their biggest concern so they are currently looking for solutions to provide assets for their project. As they are currently in school or handling transitions to work life they prefer to keep project expenses as low as possible.

3D Computer Graphic Artist

Another use case is a current computer graphic artist. They are currently using Blender for their work due to its community and support features. They are contracted to work on a new graphic art project. This project is set in an outdoor environment. The graphic artist values efficiency and wants to have a good looking environment without much of a time sink. This allows them to focus their efforts on the focus of the design. The graphic artist is more flexible on the price as they view it as an investment. The add-on would need to balance speed and flexibility.
Implementation

Add-on Architecture

The add-ons architecture follows a very linear loop. The user interacts with the UI, editing the generator's initial properties. These inputs are controlled by the input interpreter that translates user input into the mesh operations module that contains each of the generators. The generator will create the model's geometry and call the BPY Library to instantiate an object and mesh in Blender’s editor. The generated shape is then available for use in the Blender Editor and the add-on waits for new user input.

UI Development

I created a Blender add-on using Python and Blenders Python API. When a user first enables the add-on they are able to access it from Blenders View3D context panel. Upon reopening Blender the add-on will initialize to default property values. These property values are accessible through the add-on UI.

Within my add-on each generator is controlled by a unique operator and sits adjacent to each other in the UI. The operator contains a list of properties to be edited by the user which serve as arguments to control the generator. These properties are subdivided categorically. An example UI panel is shown below in Figure 1. Each of the properties within contains a default value, maximum and minimum value and a name to explain the property in context. To properly manage window space and to streamline information management these properties are default hidden in collapsible tables and can then be edited when the table is expanded. A second example of the same panel with most of the information hidden by collapsed tables can be seen below in Figure 2. Through these panels, the user can control the generator with a fine degree of control.
Figure 1: An example of the UI Panel with Generator Properties
Figure 2: An example of the UI Panel Utilizing Collapsible Tables to Control Visible Properties

**Generator Development**

As development continued, I was able to create more efficient and flexible designs, leading to different design methodologies for each new generator. At its core, each generator follows a fundamental design principle. They would iteratively create nodes, with each node following a specific set of rules for propagation. These rules would control what information was passed to the next node and what object resulted from the current node. The fern generator is by far the most complex generator in FloraGen’s suite. The fern generator creates a stem object out of an array of cylinder nodes and then determines the frond and leaf properties to be created at each of these cylinder nodes. The growth of the stem and fronds can best be modeled by a three dimensional projectile motion equation with a uniform time step.

The grass generator operates similarly to the fern generator with a few key differences. Each grass frond is created in a similar fashion to the stem of the fern but the grass uses polymorphic design principles to interchange the cylinders making up the stem with a curved semi circle shape instead. This allows the user to directly control the thickness of the grass stems in multiple degrees by editing the radius of the semicircle, the number of planes making it up and the degree that the curve of the semicircle follows. The example in Figure 3 below shows two grass objects with a semicircle angle of 30° (left) and 120° (right). All other parameters are controlled for comparison.
The stick generator also uses the concept of polymorphism but experiments with a new generator type unlike the Grass and Ferns Projectile Motion equations the stick utilizes a more procedural generation system. Much like a Breadth First Search algorithm, the stick generator utilizes a node based Queue system in which nodes are created and subsequently enqueued into the generator. A node contains a lifespan that is decremented each time a new node is created, once the lifespan is depleted, no new nodes may be created and the Queue becomes empty, terminating the generator. This generator utilized mutation events that would change the propagation direction but would decrease node lifespan at a greater rate to give the illusion of offshoots being younger than a main branch of the stick. By elevating the stick generator’s mutation rate, it allows for a fine control over the ability of the generator to create complex branch structures. In the Figure 4 below, three sticks are generated with a 20% mutation rate (left), 60% mutation rate (center) and 100% mutation rate (right). All other properties are controlled for comparison. Note a trend where the stick becomes smaller as it mutates more often, this is a result of the generator splitting the remaining node lifespan between two branches leading to smaller branches overall. This effect can be offset by increasing the maximum amount of permutations in the properties. In Figure 1.5, two sticks are generated with max permutations set to 20 (left) and 40 (right).
Figure 4: Comparison of Stick with mutation rate of 20% (left), 60% (center), and 100% (right)

Figure 5: Comparison of Stick max permutations of 20 (left) and 40 (right)
The rock generator follows the same iterative principle as the previous generators with one key feature added: the rock generator uses a noise map to generate fluctuations in terrain. The rock generator utilizes the cylinder generator and then applies the coordinates to a noise map to create complexity. This is a relatively simple approach and results in the rocks having an ovoid base. The sheer amount of customization however allows for a large amount of variation in rocks formed from tall stalactites, to flat disk rocks to entire terrains. Figure 6 shows two different rocks created by scaling the height offset of the noise generator. Figure 7 shows the flexibility of the rock generator by utilizing it to create a large-scale terrain object.

Figure 6: Comparison of Rock noise offset set to 0.2 (right) and 1.0 (left)

Figure 7: Utilizing Rock Generator to Create Simple Terrain
The tileset operators also utilize noise but employ it as a culling method to model spawn locations. The tileset operators superimpose a noise map onto the space that will pose as the tileset. The density of points and the height requirement to generate a model are the main constraints on the density of objects produced by the tileset. The grass tileset generator was a prototype tileset generator to ensure that the method would successfully create a proper tileset. It also served as a good test for a baseline of the tileset generator's runtime efficiency. The second tileset generator dubbed modular tileset generator used a list based approach and was entirely modular for the purpose of adding additional generators to the tileset. The UI contained a boolean selector to determine if an object should be added to the generator and the properties required to add that object type. The generator itself contains very little code specific to each individual generator allowing for easy addition and removal of items to the generator. I calibrated each of the defaults to a controlled value such that pressing generate would create a balanced environment with a single press. Figure 8 below shows an example environment created by the tileset generator with simple monochromatic materials added to the generated models. This model took a couple minutes to create with the Multiple Tile Generator.

![Figure 8: Simple Generated Tile from Multi Tile Generator](image)

**Implementation Issues**

The only significant issue encountered during the development of this project was a BSOD crash on windows which led to the permanent corruption of the first project iteration. This corrupted a significant portion of the fern generator requiring a rewrite. This rewrite allowed for major improvements to the original generator by refactoring the original code to a more efficient and modular platform.
Testing Feedback

Add-on Test Results

The add-on was tested with three 3D Computer Graphics artists for effectiveness and usability. In the testing version the debug menu was made available to allow for a full exploration of the add-on. The add-on received positive feedback from each of them when experimenting with the default settings. The most well received item was the grass generator and tileset generator. The longer operating time of the fern generator was better liked than expected. I did not factor that Blender itself often has longer operation times. One artist commented on the space efficiency of the generator, stating that “Many add-ons are much larger and take longer to set up.” During testing, two critical errors were discovered. The first leading Blender to crash and the second locking up the add-on. From the feedback I was able to locate a problem where if FloraGen is used in Edit Mode, it will not operate due to errors raised by BPY library functions. I was able to correct these two errors with a quick patch. The artists also made quick works using the generator. In Figure 9 below, one such work is displayed.

Figure 9: Quick Night Scene with Generated Grass
Conclusion

Findings

My senior project was a powerful learning experience. I was able to explore application development, and how to effectively produce applications using Python. I was able to learn a great deal about Python and its functions, Blender and its design and implementation, and about efficient programming.

The time limit on this project encouraged improvisation and adaptation when it came to programming. The fern generator and learning Blender’s Python API occupied a significant portion of my first quarter. To accomplish my design goals I learned how to refine, reduce and recycle my previous work. With each generator created I was able to make the next even quicker, each with their own nuances. This evolution is readily apparent by comparing the first and second Quarter Schedules.

This freedom to pursue my own passion project would not have been possible without my senior project advisor Professor Rich Murray. I would like to thank Professor Murray for supporting the development of my project over the past two quarters. It was an excellent learning experience to explore application development and 3D model design concepts for these last few months that would not have been possible without a supportive advisor. Whenever I presented my new findings or product, he would be there to bounce ideas off of and to provide some excellent feedback. This allowed me to far surpass my expected minimum viable product when I set out to create this add-on.

Future Work

There are a myriad of potential expansions onto this add-on. A popular request from add-on testing was that the add-on could utilize collections in the Tileset generators. This would make it much easier to select groups created with the generator as they are grouped automatically. This is an opportunity to further refine the efficiency and algorithms used by some of the slower processes. As I am working with large data sets there is an opportunity to utilize Numpy and Pandas over the existing Python data sets but I refrained over concerns of complicating the install process on other systems. I can investigate how to employ these libraries in the future. As I was finishing the later tileset generators I discovered that much of the existing code can be modularized further to allow for even easier expansion. This would be necessary in expanding the Tileset generator as properties would need to be dynamically created. A final major avenue for expansion would be to add more generators and options for creating flora and other environmental options. A couple different expansions would be groundcover and other object types or themed sets of plants. With a good foundation to the add-on, the sky's the limit.
## Senior Project Quarterly Schedule

### Blender API Project Schedule

<table>
<thead>
<tr>
<th>TASK TITLE</th>
<th>TASK OWNER</th>
<th>Start Week</th>
<th>Duration (Weeks)</th>
<th>Work Hours per Individual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blender UI</td>
<td>Chris Wright</td>
<td>1</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>1.1</td>
<td>Chris Wright</td>
<td>1</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>1.2</td>
<td>Chris Wright</td>
<td>3</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>1.3</td>
<td>Chris Wright</td>
<td>4</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>1.4</td>
<td>Chris Wright</td>
<td>4</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>1.5</td>
<td>Chris Wright</td>
<td>4</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>2.1</td>
<td>Chris Wright</td>
<td>3</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>Chris Wright</td>
<td>4</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>2.3</td>
<td>Chris Wright</td>
<td>5</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>2.4</td>
<td>Chris Wright</td>
<td>7</td>
<td>2</td>
<td>18</td>
</tr>
<tr>
<td>2.5</td>
<td>Chris Wright</td>
<td>9</td>
<td>2</td>
<td>20</td>
</tr>
</tbody>
</table>

**Figure 10: Senior Project Spring Quarter Schedule**

### Blender API Project Schedule

<table>
<thead>
<tr>
<th>TASK TITLE</th>
<th>TASK OWNER</th>
<th>Start Week</th>
<th>Duration (Weeks)</th>
<th>Work Hours per Individual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blender UI</td>
<td>Chris Wright</td>
<td>1</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>1.1</td>
<td>Chris Wright</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1.2</td>
<td>Chris Wright</td>
<td>4</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>1.3</td>
<td>Chris Wright</td>
<td>6</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>2.1</td>
<td>Chris Wright</td>
<td>1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>Chris Wright</td>
<td>2</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>2.3</td>
<td>Chris Wright</td>
<td>3</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>2.4</td>
<td>Chris Wright</td>
<td>5</td>
<td>3</td>
<td>18</td>
</tr>
<tr>
<td>2.5</td>
<td>Chris Wright</td>
<td>7</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>2.6</td>
<td>Chris Wright</td>
<td>8</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>2.7</td>
<td>Chris Wright</td>
<td>9</td>
<td>2</td>
<td>20</td>
</tr>
</tbody>
</table>

**Figure 11: Senior Project Fall Quarter Schedule**

### Github Link

https://github.com/chriswwright/Flora_Gen
Bibliography

https://xd.adobe.com/ideas/process/ui-design/4-golden-rules-ui-design/

Blender Foundation. *Blender 3.0.0 Python API Documentation*. Dec 10, 2021. Blender Foundation,
https://docs.blender.org/api/current/


Jayanam. “Blender 2.8 Python UI: Panels & Buttons.” YouTube, uploaded by Jayanam,
Dec 7, 2017, https://www.youtube.com/watch?v=n9uVoVK1O8g

Jeremy Behreandt. *Creative Coding in Blender 2.92: A Primer*. May 25, 2021. Medium,
https://behreajj.medium.com/creative-coding-in-blender-2-92-a-primer-7ac1b6fec3f