COSC 450 Assignment 1: Procedural generation

Greco-Roman inspired temples

By Johnny Flame Lee, ID:83924

1. Introduction, aim and inspiration

The main aim of this project is to procedurally generate 3D models of buildings using Blender/Python. The results should demonstrate plenty of variation, while maintaining stylistic consistency. Ideally, it should also have an interesting flare to it for visual appeal.

I choose to model buildings based on ancient temples inspired by Greco-Roman architecture, such as the Maison Carree (fig. 1), and the Pantheon (fig 2). This decision is based on my interest in classical studies, and hence some existing knowledge of classical architecture. This style of architecture is rich in artistic features, making it a good choice for randomizing components to show a wide variety of models. For example, the column (pillars) of a temple can be either fluted or unfluted, and can have a large variety of orders (the decoration on top) such as Ionian, Doric, Corinthian and Composite.

Below are some of the buildings which inspired this project:

Fig. 1. Maison Carrée, Nimes, France

Fig. 2. Pantheon,Rome, Italy

Fig. 3. Tempio Canoviano, “Temple of Canova”, Province of Treviso, Italy

Fig. 4 Temple of Augustus, Pula, Croatia

2. How the script works

This section provides instruction on how to operate the script, as well as a high level overview of the structure of the project.

2.1. Installing the add-ons

Before running the script, it is important that the required add-ons are installed. This project makes use of two add-ons, "**Building basics*"***, and "***Archimesh***". The former is a third party addon, and the latter is a standard Blender addon.

Inside the folder ***"3rd_party"***, the folder "***add_mesh_building_basics***" can be found. Assuming this will be run on MacOS, copy this folder and paste it into the directory blender.app/Contents/Resources/2.78/scripts/addons

Once the previous step is completed, open Blender, and go to user preferences->Add-ons. Select "Add Mesh: Building basics", and "Add Mesh: Archimesh"(Archimesh should come with Blender version 2.78) .

Save the User Setting, and you are ready to generate some temples.

2.2. Running the script

Open **procedural_temples.blend,** inside this file you will find two modules, Temple.py and Render.py.

Temple.py is the module we need to run.

Inside this module we can find the **main()** function, which sets up the scene, creates temples, and then renders the scene. Currently, **main()** demonstrates the creation of a temple by calling **create(xSize, ySize, height, name)**, this function takes in the dimensions of the temple, and the name of the building as a string, then create a piece of ground with the temple on top.

From **main(),** you can call **create()** as many times as you'd like to generate a number of different temples, and choose their location freely.

2.3. Recommended settings and changeable parameters

To generate the most interesting buildings, it is recommended for the X,Y,Z dimensions to be set to (10,15,7). These values is currently provided in the example as default value. Buildings made at this ratio have a 70% chance of having a dome, similar to the type seen in the Pantheon.

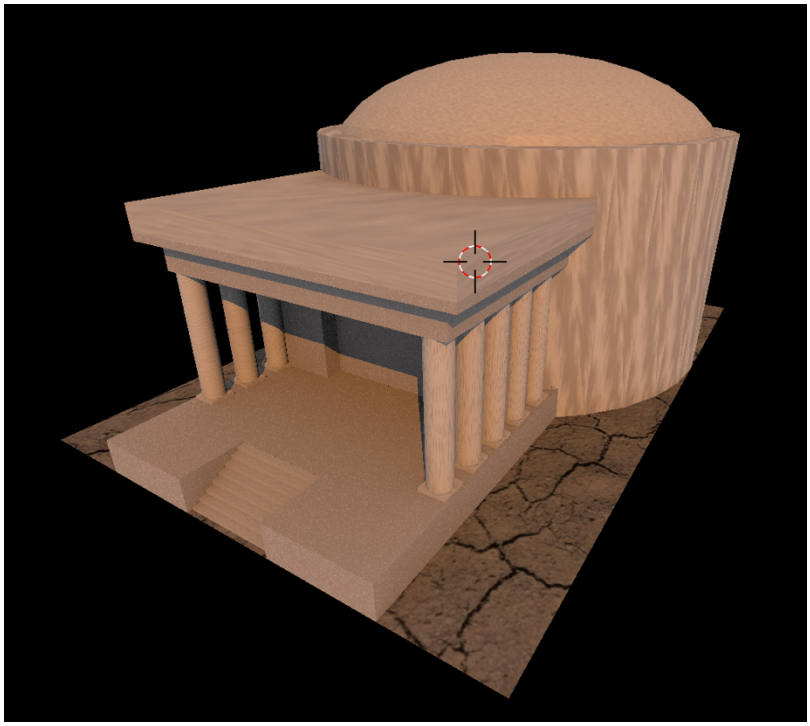

Fig.5 “Dome temple”

For a squarer looking temple similar to the Maison Carree, try (10,10,7):

Fig.6 : “Square temple”

Try experimenting with different input parameters to generate temples of various shapes and sizes.

In the interest of ease of operation, there are few parameters designed to be changed by the user. Aside from the dimensions of the buildings, the other parameter that can be changed by the user is whether the script will use the third party add-on to generate the columns. The variable **use_column_addon** defaults to have a 50% chance of either using the add-on to generate the columns of the temple, or to do them manually in native Blender.

There are pros and cons of each choice, using the add-on will result in more artistically appealing columns with more details and variation. However, the time it takes to generate each building model will drastically increase due to the complexity of the script. Switching this option off allows faster building generation, at the expense of having plainer looking columns.

2.4: Known issues

As this type of buildings are generally quite large in size, it is not recommended to run the script with xSize and ySize lower than 8. Ideally the X and Y dimensions should not be below 10.

The technical reason behind this is that the stairs are generated using the Archimesh add-on, which interacts somewhat unpredictably with python. I chose to cater for larger buildings, which results in some cases of strange looking stairs when the dimensions are too small.

3. Technical discussions:

In this section, I will discuss my approach in design and implementation of this project, reflecting on the challenges and pitfalls, and how these were handled. Furthermore, I will demonstrate the ways which my building script meets the requirements outlined in the assignment brief.

3.1. Design and construction of the temple

3.1.1. Rapid prototyping

As this project involves learning a new tool that I am previously unfamiliar with, a major challenge was estimating the time it would take to achieve a certain task. I chose to tackle this issue by rapidly prototyping, keeping the prototype as basic as possible. For example, creating an object that somewhat resembles a temple by generating a rectangular block as the podium of the temple, then have some cylinders on top as pillars.

This approach means that as the prototype progresses, I have a better estimation of the amount of time required for a task (which, unfortunately is not always accurate). I also found making small progress and having instant visual feedback helpful for morale and motivation.

3.1.2. Recreating the Maison Carree by hand

Once I could create basic geometric shapes in Blender, my next step was to model a temple manually in Python. I chose to base my basic design of the temple on the Maison Carree, using it's building plan and measurements as a guideline for the parts and ratio of my building.

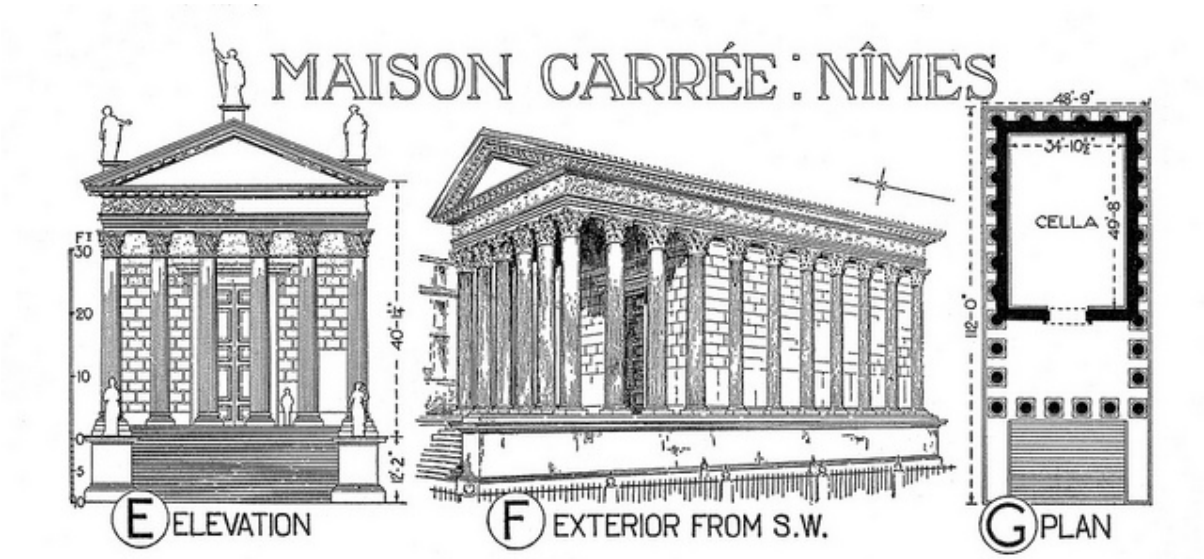

Fig. 7: Plan of the Maison Carree.

To begin, I made a list of components that a temple consists of:

- Podium
- Column base
- Temple body
- Top (area between roof and body of the temple)
- Roof
- Optionally, many temples have a dome
- Stairs

I found the Maison Carree a nice temple to start my modelling, as the ratios and measurements are readily available on the web.

I started by building a podium. Maison Carree's podium is approximately twice as long as it is wide, and about ¼ of the total height of the building.

The next step was to model the columns. I begin this by placing the base of the column on the podium. The columns cover approximately 3/4 of the podium. A little bit of math was involved in calculating the size and location of the columns, which was a slight challenge.

Once this is done, I generated the columns on top of the base, leaving a small gap between the edges of the pillars and the base, and built the rest of the building.
A few challenges emerged at this point of the project. The first one was collapsing the vertices of the rectangular roof to make it look triangular. This is a relatively straight-forward task in Blender GUI, but scripting this took a lot of work, as it required switching from "object mode" to "edit mode", select the vertices with a for-loop by trial and error, then collapsing it.

The second challenge was to find a way to decorate the columns. The columns are an important part of Classical architecture, with room for plenty of variation. For example, the columns can either be fluted or unfluted, with various styles of capital.

I tried crafting these variations by hand with little success. Fortunately, I was able to find a Blender add-on which can generate stylized columns. The add-on can generate columns with various capitals and fluting patterns, though it does not generate historical designs such as Ionian and Corinthian capitals, which is a slight let-down. Although, given the timeframe and the scale of the project, I was willing to compromise on my initial vision.

3.1.3. Parametrizing the temple: throwing one away

"Plan to throw one away, you will, anyhow"—***Fred Brooks, The Mythical Man-month***

Once I finished the model of Maison Carree by hand, I had a nice looking temple model. However, a new problem arises—the temple needs to be able to change in size according to user input, given in the form of an invisible bound-box. This was not something I had previously kept in mind when I constructed the model. As I first built the podium of the temple, everything else was linked to the temple podium.

I solved this challenge by "throwing one away", and practically re-writing my script.

I first generated a piece of land which represents 2 of the 3 dimensions provided the user input values (width and length), then build the temple on top of the land, making the podium's width and length values according to the user-input. The land helped as visual cue to check that my temples are touching the bounding box.

The temple will vary in length and width by a random factor, but If the temple has a dome, then the dome radius will be equivalent to the width of the bounding box. Otherwise, either one or two side stairs will be generated on the side of the temple, extending to the edges of the bounding box. If the temple has no dome, then the length of the temple will stretch to meeting the edges of the bounds.

Restricting the width and length was relatively straight forward, but restricting the height while keeping everything in sensible ratio was a little more difficult. I came up with a solution by taking the max available height value, then distribute a percentage of this value to each vertical components of the temple. This way the individual components' height can be randomized (with bounding factors, to keep the building in sensible ratio), so buildings will

not look the same. Meanwhile, the overall height restriction is still met, as building will never exceed the max available height value given by the user.

3.1.4. Textures and material, and the Second System Effect (feature creeping)

At this stage, I have the bulk of the project completed. The next stage is to put material and texture on the building mesh. This part took more time than I expected.

I first experimented with using “realistic” textures by using real images and creating bump maps from them. However, this turned out poorly, I came to the realization that UV mapping will take some time to master, and poorly wrapped realistic texture is even worse than not having any textures. In the end I decided to use procedurally generated material similar to the examples given.

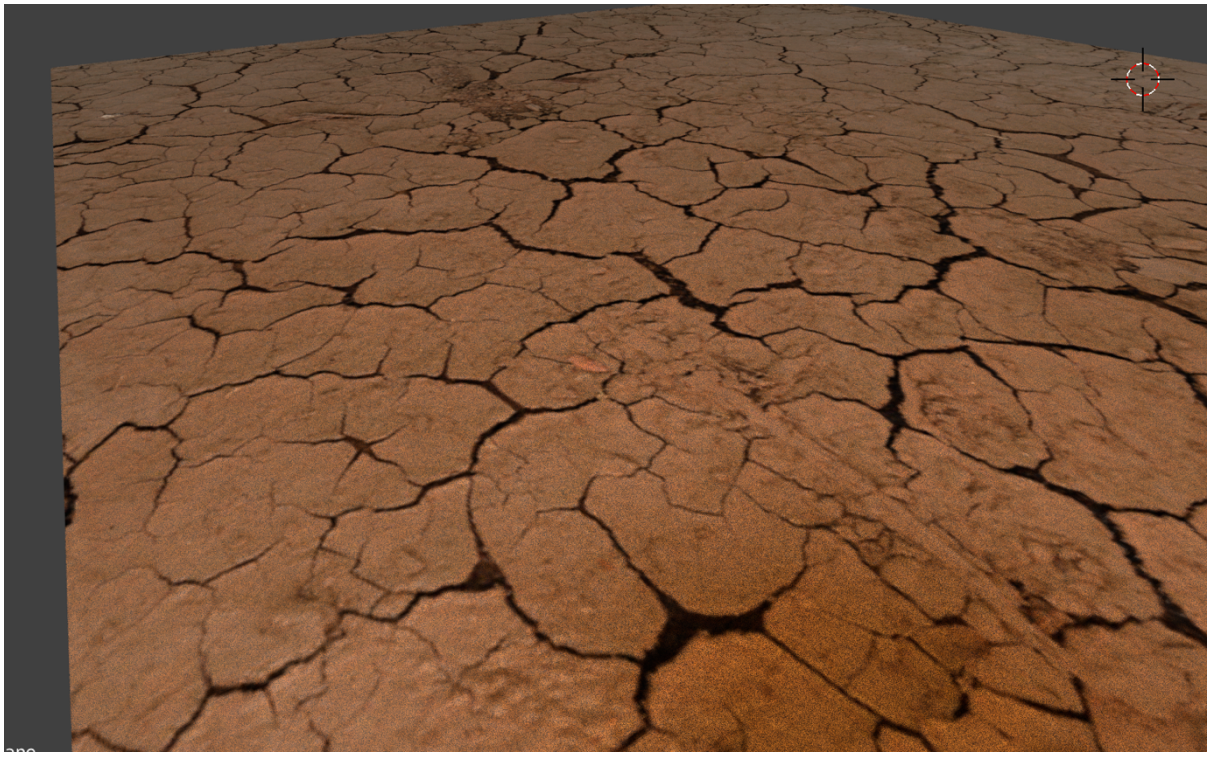

Fig.8: More realistic texture created with normalized map and bump map, I could not figure out how to do this well on a 3D object.

A pitfall that I encountered in this stage is the “second system effect”, i.e. the temptation to add additional features to add flare to my buildings. Attempting to make bump map is one example of this, another one is when I tried to create a destructed look to mimic temple ruins, below is an example of it:

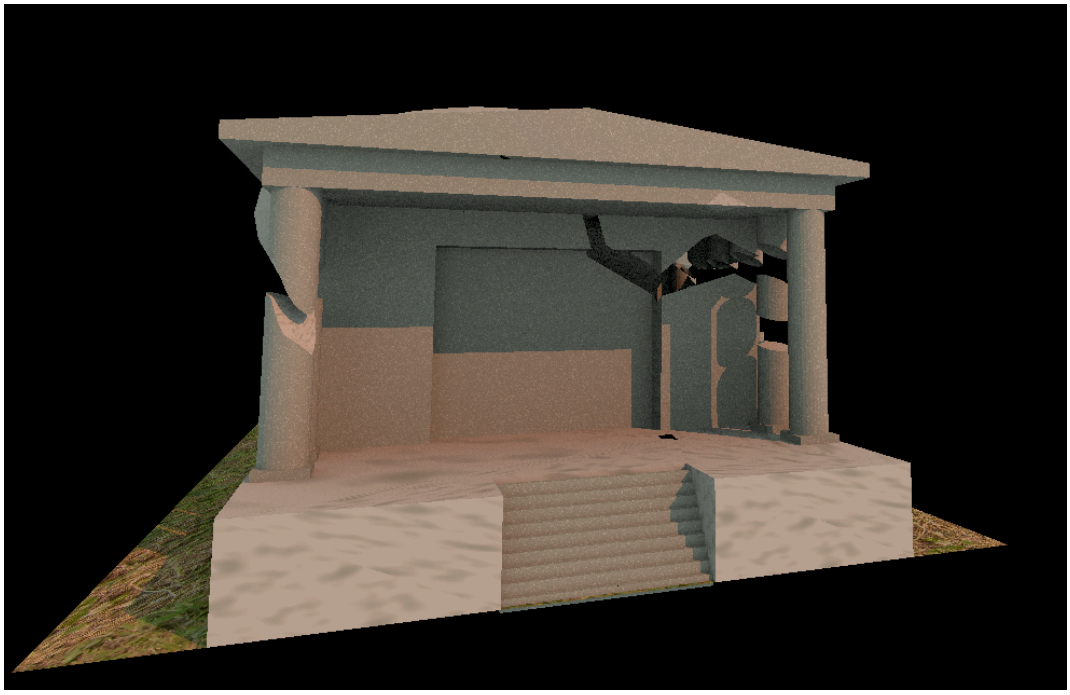

Fig. 9: "Temple after an earthquake"

While this is not too difficult to do by hand, scripting it accurately turned out to be more challenging and time consuming than I expected. I run out of time towards the hand-in date, and these experimental features were discarded.

3.2. Meeting the requirements:

The requirement of this project involves generate buildings by a specific dimension constraint, which are stylistically consistent, while showing plenty of variations. As the dimension constraint part is explained in the previous section, this section will focus on explaining how consistency and variation is achieved, discussing some approaches to randomization of parameters.

3.2.1. Consistency vs variation

The nature of this type of building (temples) dictates that they are not intended to be built in large groups like some other types of buildings, such as apartments or skyscrapers.

I choose to balance consistency with variation in this situation by keeping the overall structure of the temples consistent. For example, all temples have a podium, body, columns and a roof component. The major variation of the structure of the building is whether the building contains a dome, which is implemented with a weighted random factor—if the length factor is between 1.5 to 2 times the width factor of the building, then there is a 70% chance that the building will have a dome.

The majority of variations between each building model comes from factors other than the overall structure of the building, these factors are randomized to create interesting results.

3.2.2. Randomization of parameters

In order to make a variety of buildings, I randomized a large number of parameters. Most of them have a bound placed on them to limit the range of randomization, thus making the result more sensible.

- **Dome or no-dome**—The building has a 70% chance of having a dome if the length is between 1.5 to 2 times the width.

- **Building width**—If the temple has no dome, it will have varying width between 50%-80% of the maximum width specified by the user. If it has a dome, the temple's width will be between 35%-80% of max width. The reason for this is that the dome will reach the edges of the box, so there is less need for the temple body to fill up the space.

- **Flat roof or pointy roof**—the temple will have a 20% chance of having flat roof.

- **Distribution of the total available height between components:**

    - **Podium**— Between 5% to 20% of max available vertical space
    - If there is a dome, the dome will always fill the specified space vertically, touching the top bound of the box. The total height of the **temple body** will be set to between 50%-80% of the max available vertical space, so it will not be the same height as the dome.
    - The **pillars** will then take between 60%-85% of remaining space (space available after subtracting the podium's height)
    - The **top** (area under the roof) will take 10%-50% of remaining vertical space
    - The **roof base** (the square part of the roof), will take 10%-50% of remaining space
    - Finally, the **roof** (pointy part) will take the remaining vertical space.

    This approach effectively normalizes the total available vertical space, and give a portion out to each component. This way, each time the building is generated it will look subtly different due to the different distribution. The bounds are chosen so the building does not look uneven in the edge cases of randomization.

- **Columns:**
    - **Number of columns on short edge and long edge**—the temple will have either 2,4,6 or 8 pillars along the short edge, and between 4 to 12 pillars along the long edge. The thickness of the pillars will also change to adapt the number of pillars generated.
    - Type—as mentioned in previous section, the column can be made with or without the add-on.
    - If add-on is used, additional variation is added to these parameters:
        - Capital height—the proportion of the capital to the rest of the column pillar is randomized.
        - Capital style—randomly chosen from an array of 20 styles.
        - Flute factor—the columns can have either 0,6,8,10,12,14 or 16 flutes carved onto the pillar body.

        - The depth of the flute cuts has a slight random factor, so some generation of buildings will have slightly deeper flutes than others.

    Obviously, these random factors must be consistent between each generation of building, i.e. across the same building model these factors must be consistent. This is handled by generating random factors inside the temple() function before sending these parameters to the stylelized_pillars() function.

- **Stairs**—Side stairs are generated so if the temple has no dome, the width constraints are still met. The temple can have either no side stairs (if it has a dome), 1 stair on either left or right side of the temple, or shorter stairs on both sides. Both the main stairs and side stairs have a random number of steps. Additionally, side stairs are either square shaped or have rounded edges.

- **Material and textures**—materials and textures are procedurally generated to make the temple more interesting.

    - Textures are randomly picked between “NOISE”,”MUSGRAVE”,”MARBLE”,and “CLOUDS”.
    - Color choice—the RGB values are randomized as follows:
        - First generate a shade of red between 0.4-0.7.
        - Randomly select a green value in the 20% range to the red value.
        - Randomly select a blue value in the 20% range to the green value.

The colors are randomized in this way so the values will be close to each other, creating a mild effect on the temple while remaining different each time. This creates interesting effects such as slight shades of grey, cream, blue, red, green and purple. I purposely avoid bright/strong colors as it would make the temple look strange.

The texturing of each individual component of the temple varies, this creates some interesting contrast between matte and smooth looking parts. While the color of the entire temple is consistent.

- **Ground type**—Different images are used for the ground. A nice bit of variation between different terrains. The ground is attached to each temple, so generating multiple temples will give different grounds each time.

- **Lighting in the scene**—A random mix of sunlight and ambient light, as well as randomized shades of red is used in the sunlight, to create a dusk or dawn effect. Mimicking the sun at different times of the day.

Appendix: Examples of buildings generated by the script