Tampere University of Applied Sciences

# Blender Geometry Nodes

## Node-based Procedural Modelling for a Short Film

Mikko Lappi

BACHELOR'S THESIS
September 2023

Media and Arts
Interactive Media

## ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Bachelor's Degree in Media and Arts
Interactive Media

LAPPI, MIKKO:
Blender Geometry Nodes - Node-based Procedural Modelling for a Short Film

Bachelor's thesis 65 pages, appendices 1 page
October 2023

---

3D modelling for film and video game productions is a time-consuming task, especially if the goal is to create large-scale environments like cities, with thousands of buildings or natural environments with intricate organic details. This is why these kinds of modelling tasks are often automated through procedural techniques instead of using strictly manual methods.

The free open-source 3D software Blender introduced Geometry Nodes in version 2.93 as a tool for procedural modelling through a node-based user interface. The aim of this thesis was to evaluate Geometry Nodes by looking at how well it delivers the benefits of procedural modelling methods. The benefits of procedural modelling were studied by examining previous practical use-cases. After this, Geometry Nodes was tested in the modelling process of a short film production.

The practical case-study resulted in two procedural environment assets for the author's short-film "Product 21". The thesis concluded that Geometry Nodes was a useful and powerful tool in achieving the desired results while being approachable with its visual programming interface. It was also found that there are more potential use-cases in the production that were not addressed in this experiment. Certain obstacles and challenges occurred in the making of both assets which were deemed as problems that are inherent to procedural methods in general.

---

Key words: 3D, short film, procedural modelling, visual programming

**CONTENTS**

## GLOSSARY

| | |
|---|---|
| Fractal | Geometric shape that can be repeated infinitely to create repetitive or organic details. |
| Heightmap | 2D texture interpreted as a 3D shape. |
| Parametric control | Controlling a procedural model or texture with labelled parameters in a graphical user interface. |
| Procedural workflow | Using algorithms and mathematical functions to produce assets, as opposed to manual editing. |
| Modifier | Non-destructive automatic operation to a 3D object's geometry information that can be changed or removed at any point. Modifiers are used to replace operations that would be too tedious to perform manually. |
| Noise function | An algorithm for creating randomness in the details of a texture or model. |
| Noise map | 2D visual presentation of a noise function. |
| Node | Block of parameters and operations that can be combined with other blocks in a visual programming interface. |
| Node network | A combination of nodes. |
| Node tree | Same as node network. |
| Normals | The orientation vector of polygons. |
| Polygon | A plane in 3D space consisting of three or more points. |
| VPL | Visual programming language. |

# 1 INTRODUCTION

There are certain complexities in modelling 3D polygon assets for films and games: assets such as characters that need to move need to have a specific structure, or topology, in order to be deformable in an animation for film or game (Martin, 2021). Achieving this kind of precision with traditional manual modelling methods makes sense since they give low-level control over the geometric details of the asset. These tools are generally intuitive to use as well: the user moves points in a 3D space or sculpts the model with brush tools, like painting in Photoshop.

Manual methods are of course time-consuming, and thus, costly. If a production would need to have large scenes with complex geometries or numerous variations of the same asset, the production cost increases significantly. In a conference presentation in 2018, 3D artist and Poliigon CEO Andrew Price made a simplified hypothetical calculation considering manual 3D workflows. He estimated that with what he calls a “static workflow”, meaning a workflow that produces a single asset, modelling and texturing a single building asset could cost around 3900$ (Price, 2018). In a presentation about the worldbuilding of the film Zootopia, Dominik Tarolli from Esri provides a similar hypothetical about modelling cities manually: assuming that a modeller could model a building in 15 minutes, modelling a city like Chicago with 1.2 million buildings would take around 300,000 hours or 150 man-years (Jarrat & Tarolli, 2017).

These calculations are of course simplified and somewhat arbitrary since no studio would realistically approach this kind of scale through strictly manual methods. They do, however, point to the most obvious benefit of automating complicated modelling tasks: bringing down costs and making detailed large-scale scenes possible. This automation is what procedural modelling methods are about: instead of manually modelling hundreds of individual assets, in a procedural workflow the artist creates a set of rules for a software to generate those assets.

Paid software, like Houdini from SideFX and 3Ds Max from Autodesk, have offered such procedural modelling tools, either through a graphical user interface

(GUI) or through a script editor. Esri's CityEngine has also been used since 2008 to generate urban environments. In the open-source software Blender, procedural modelling has been possible with modifiers, which automate editing tasks that would either be too cumbersome to do manually, either because of the difficulty of the modelling process or because of considerable repetition. A good example of this is the subdivision surface modifier, which smooths out a 3D object by dividing its geometry.

In Blender 2.92, a new tool for procedural modelling called Geometry Nodes was introduced. Geometry Nodes allows controlling of geometry through a visual programming language, similar to the Shader Editor (Blender Foundation, 2022). The idea behind this new tool was to bring in a more flexible procedural modelling system than the previous modifier-based modelling system (Felinto, 2020). This could bring Blender closer to software like Houdini, as it allows similar complicated geometry-based animation and modelling.

This thesis will look at the general benefits of procedural modelling through its history and more recent practical examples, to see where procedural modelling methods have been most useful in the past. The thesis will then take a deeper dive into Geometry Nodes, looking at its development history and explaining its core functionalities. This will offer some light into why more procedural tools have been developed for Blender recently, and what kind of new opportunities Geometry Nodes offers for 3D artists.

Geometry Nodes will be further evaluated in the the case study for this thesis: a science fiction short film with the working title "Product 21". The story of the film has been developed and a list of assets has been compiled, both of which can be found in Appendix 1. With this project, the aim is to test out if Geometry Nodes can be implemented in a meaningful way, and if it successfully brings the benefits of procedural workflows into the modelling phase of the film.

# 2 PROCEDURAL MODELLING

Before going into practical examples, we should further define the scope of the thesis. When talking about 3D models in this thesis, it mainly refers to polygonal models. Polygonal modelling is when 3D shapes, or meshes, are changed by directly interacting with flat planes, known as polygons (Zeman 2014, 23-25). Besides polygonal modelling, there are other ways of creating 3D models, like NURBS curves or volumetric modelling. In this thesis, the techniques used are polygonal modelling as it is the method most commonly used in animation.

Manual methods can mean many things. In Blender, making polygon models manually is achieved either through the Edit Mode, where the user changes individual elements of polygons (vertices, edges or faces), or in the Sculpt Mode, where the elements are changed with brush tools.

Similarly, procedural modelling can mean many things, as the role of procedural methods depends on the goal of the modelling task. Procedural methods can be used to generate the entire model, or they can be combined with manual methods and have smaller part in the modelling process. In this thesis, the term procedural modelling refers to:

- Generative modelling, where all geometry is created from scratch, or around an input object like a curve.
- Manipulating geometry on a pre-existing model, either by changing vertex locations or creating new geometry automatically.
- Modular modelling, where copies or instances of a pre-existing model are scattered onto another model or a 3D or 2D grid.

Chapters 2.1 and 2.2 explore these techniques through the history of procedural modelling and through more recent practical applications.

## 2.1 Brief history of procedural modelling

Procedural workflows are by no means a new phenomenon in computer graphics. In the 1980s, procedural methods were used for textures replicating natural materials like marble, wood, and stone. Early procedural modelling methods were

an extension of these texturing methods: instead of creating 2D images, these algorithms were interpreted to generate 3D geometry, to model things like water, fire, landscapes, or clouds (Ebert et al. 2003, 1-2). In terrain generation, 2D textures used to describe altitudes in a 3D space are called heightmaps (Paone n.d.), or sometimes height fields (Ebert et al. 2003, 491)

Fractals, invented or discovered by mathematician Benoît B. Mandelbrot, are geometric patterns that can be repeated infinitely to mimic the complexity of shapes found in nature. Fractals found early use for generating natural environments, especially for terrain generation, as they were a way of creating repetitive organic details efficiently. (Ebert et al. 2003, 571) Besides Mandelbrot's work, another way of creating fractal shapes is through noise functions. Noise functions were introduced in 1985 by Ken Perlin as a way to quickly add visual detail into computer graphics. Noise functions, such as Perlin Noise, are widely used today in both 2D textures and 3D geometries. (Lagae et al. 2010)

The first application of fractal 3D geometries in computer-generated imagery (or CGI) was in the short film Vol Libre, presented in 1980 by Pixar co-founder Loren Carpenter (Picture 1). The film features a flying camera animation showing procedurally generated mountains. Carpenter was hired by the computer graphics division at Lucasfilm, and this technique for mountain generation was used in the film Star Trek II: Wrath of Khan. (Seymour 2013) Early commercial software for generating these kinds of fractal landscapes include Bryce from 1994 and MojoWorld from 2001. Both are based on Ken Musgrave's work on fractal geometries, with Bryce being developed into a software by Eric Wenger and Kai Krause (Ebert et al. 2003, 579-580).

PICTURE 1. Vol Libre: first animation to use fractal 3D geometries. (Vol Libre 1980)

Shape grammars, first introduced by George Stiny and James Gips in 1971, are another early effort generating graphics procedurally. Shape grammars are systems for describing rulesets for combining and creating complex shapes (Schinko et al. 2015, 471). Shape grammars, along with other procedural methods, have found use in visualizing urban environments, as it involves lots of repetition of detailed elements like windows, columns, pillars, and decorative elements (Hwang 2017).

One example of shape grammars in 3D modelling, the CGA grammar (Computer Generated Architecture) is a programming language that is used in CityEngine by ArcGIS (Schinko 2015, 470). CityEngine, first released in 2008, is a 3D software used in urban planning and other fields to produce 3D models of entire cities procedurally (Esri, 2022). On top of modern urban visualization, CityEngine has found use in 3D reconstructions of historical cities. One example is the work of Marie Saldaña, seen in picture 2, which is a reconstruction of the Greco-Roman city Magnesia on the Maeander, done as part of her doctoral research (Saldaña 2015).

PICTURE 2. Magnesia on the Maeander reconstructed in CityEngine. (Picture: Marie Saldaña 2023)

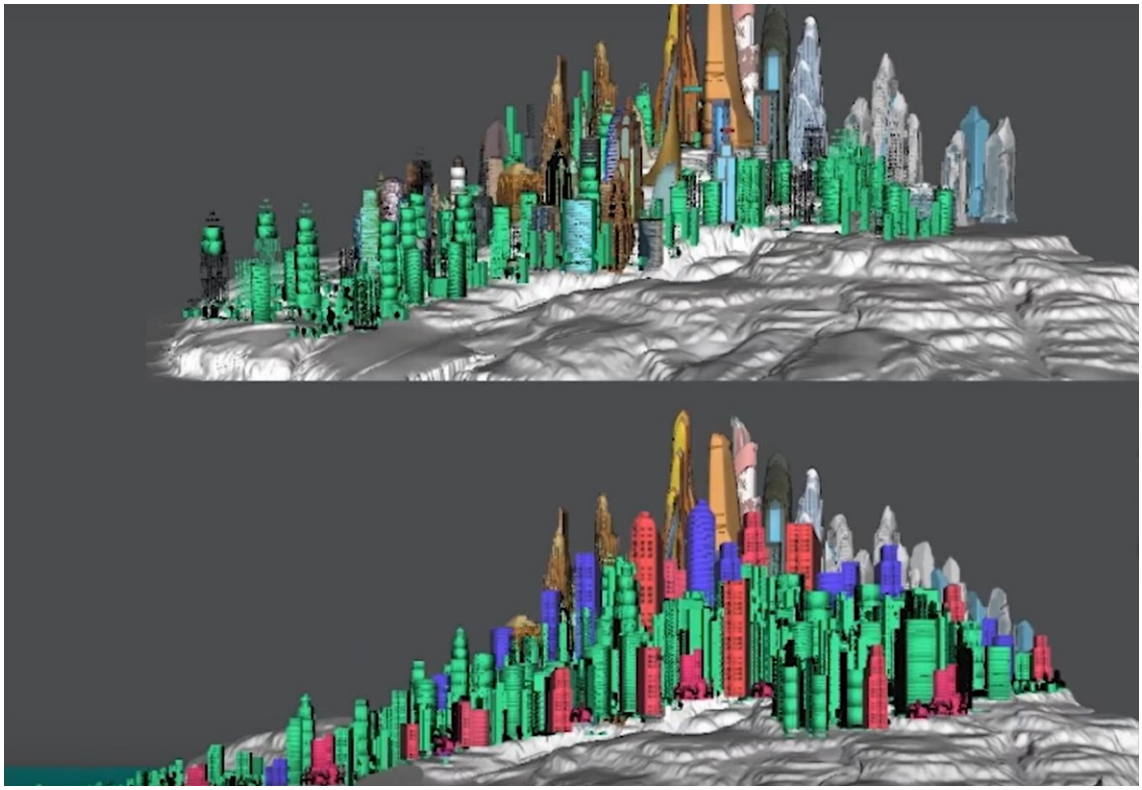
Lastly, an important software to mention is Houdini from Toronto-based SideFX, which is a fully procedural 3D software used for modelling, animation and VFX. Houdini was released in 1996 as a continuation of Prisms, which was a set of tools for procedural 3D graphics, mainly used for broadcast motion design like logo and bumper animations. Houdini and Prisms were used in 1997 for the films Contact and Titanic, and since then, Houdini has been used widely in the VFX field, by companies like Disney. (Seymour 2012) A few examples of Houdini being used for film and motion graphics are addressed in the next chapter.

## 2.2 Use-cases of procedural modelling in film, video games and motion graphics

The 2016 animated film Zootopia by Walt Disney Pictures is a notable example of procedural 3D modelling in filmmaking. The film introduces the titular city to the audience through a train ride sequence, where the main character travels through each part of the city. To believably convey the large urban scale, an entire city layout needed to be created with hundreds of thousands of individual buildings and other city assets, which would take a considerable amount of time placed manually (Picture 3). The studio achieved this by utilizing ArcGIS CityEngine,

since it allowed generating a city around a premade landscape model and the few manually placed important larger buildings (Picture 4). To create enough variation for the city to look believable, the buildings were compiled procedurally from premade modular pieces. (Jarrat & Tarolli 2017)

PICTURE 3. The city in the film Zootopia. (Zootopia 2016)



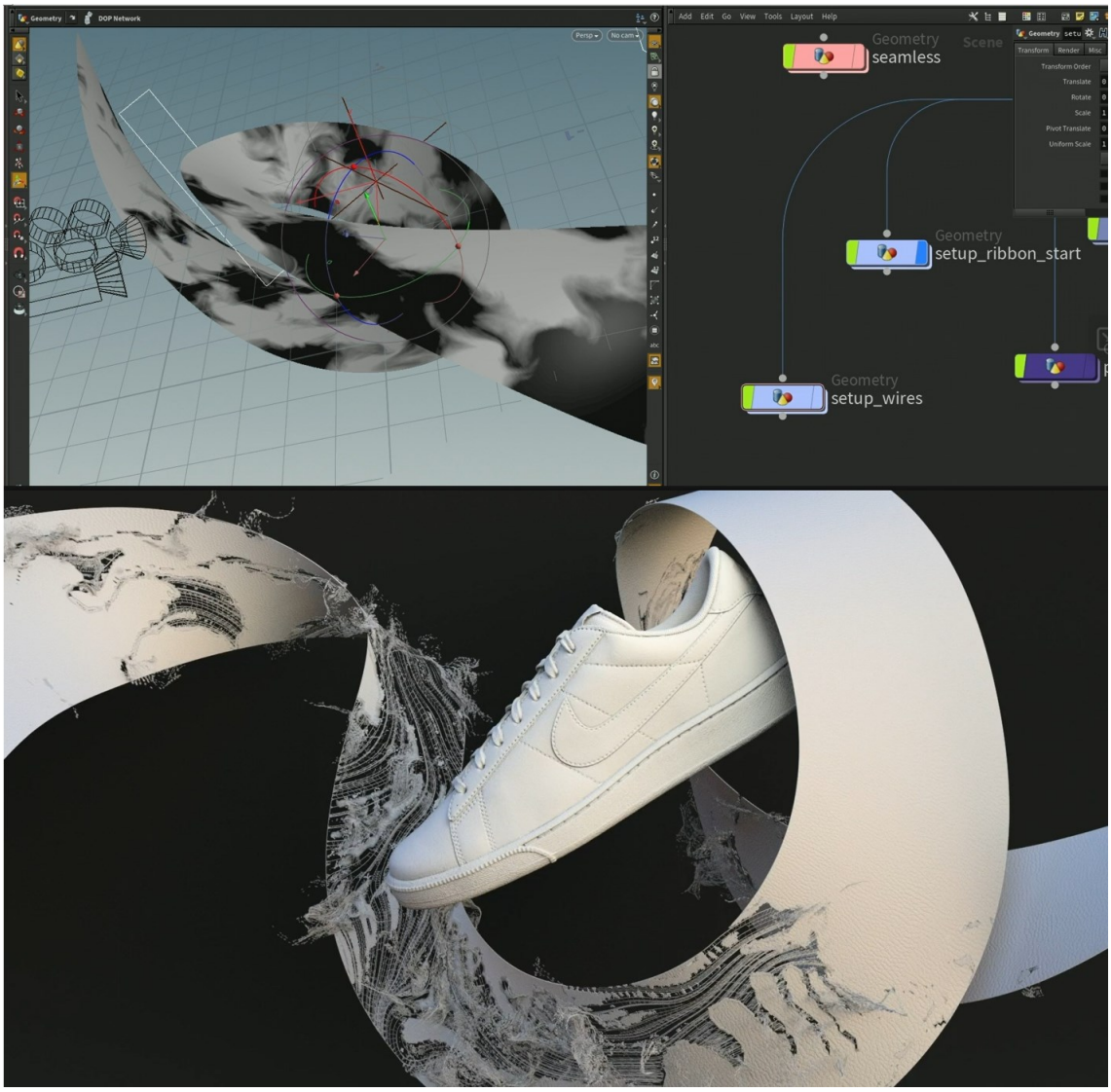
PICTURE 4. The city of Zootopia in CityEngine. (Brandon Jarrat 2017)

Procedural modelling can also be used to create complex shapes for visual effects. This is the case for the film Annihilation, which took Mandelbulbs as a design motif. Mandelbulbs are the name given for 3D renders of the Mandelbrot set,

first successfully achieved in 2009 (Aron 2009). On top of VFX, Mandelbulbs were used for set design: the chamber of at the end of the film features Mandelbulb shapes that were crafted based on a high-resolution render of the 3D geometry that was used to make moulds with a CNC machine (Picture 5). The software used for generating these Mandelbulbs was Houdini. (Failes 2018)

PICTURE 5. Mandelbulb render used in Annihilation. (Annihilation 2018)

Other examples of Houdini being used can be found in motion graphics. Germany-based motion design artists Manuel Casasola Merkle and Moritz Schwind used Houdini on their commercial for Nike's recycled artificial leather material Flyleather. The commercial shows complex weaving animations and leather textures to illustrate the structure of material. In this project, they used Houdini the whole way, starting from the initial layout phase, as opposed to creating storyboards or mockup stills in Photoshop. The final setup is a combination of wireframe geometry and particles, generated around a single curve object (Picture 6). The benefit from this type of fully procedural workflow was that they were able to create multiple variations quickly: to create a new composition, all you had to do was edit the curve, and everything else would adapt to it. Building the system right away also helped make sure that the visuals presented in the layout phase could also be delivered in the final video. (Merkle & Schwind 2018)




PICTURE 6. Flyleather commercial in Houdini, and the final video. (Manuel Casasola Merkle & Moritz Schwind 2018)

Procedural generation has also been a technique used in video games from as early as the 1980s. The video game Rogue (1980) is widely considered to be the first video game to procedural generation: all of the game's level, consisting of dungeons with monsters and collectable items, were randomized on each play. At the time when storage space on computers was limited, this made it possible to create bigger games: instead of storing pre-made level data, only the algorithm needed to be stored and the level would generate itself. (Zucconi 2022)

These days, possibly the most well-known video game utilizing procedural 3D content generation is Minecraft. The vast world of Minecraft is generated using multiple layers of noise maps, which are randomly generated numbers stored in a 2D image texture format. To put it simply, these noise maps divide the world

into islands and oceans, and then determine the elevations and the placement of elements like biomes and rivers, further dividing the world into smaller and smaller segments. After this, other algorithms are used to carve caves and other elements of a Minecraft world that can't be presented in a 2D form. (Zucconi 2022)

Games such as Minecraft use world seeds as the user input for generating a world. A seed in procedural generation is a combination of digits or letters that the generator's randomizer algorithm uses to calculate values for the parameters of the world. Considering all the different possible variations of a world seed in Minecraft, it has been said that Minecraft has around 18 quintillion possible worlds to generate (Zucconi 2022). A similar number has been mentioned regarding a more recent game called No Man's Sky, which features procedurally generated planets that the user can explore (Higgins 2014).

Both Minecraft and No Man's Sky (Picture 7) use a 3D grid for storing terrain data, like type biome or block, which is then used to polygons according to that data. Each block in the grid is called a volumetric pixel, or a voxel. This is what makes it possible for these games to feature 3D terrain elements like caves, which can't be achieved with a 2D heightmap texture. The data stored in the grid can be changed through gameplay, which allows the player to change the landscape.

PICTURE 7. Screenshots of Minecraft and No Man's Sky. (Mojang 2009; Hello Games 2016)

## 2.3 Benefits and considerations of using procedural modelling methods

Considering the examples from chapter 2.2, certain common benefits of the use procedural modelling stand out, despite the examples being from different mediums.

In the case of film and video games, procedural modelling has been especially useful in world-building. Procedural modelling allows for quick generation of details, both for urban scenarios and nature. In video games, procedural generation means the ability to create variety through automatically created worlds and items, which provides more different experiences for different players.

For Zootopia, procedural modelling allowed for a vast city to be build around a pre-modelled landscape. Even though a lot of the city was built modularly, achieving that sort of scale manually would most likely increase production costs significantly compared to manual methods.

Annihilation is an example where procedural methods became a design motif, turning the method from a VFX tool to a creative design tool. Same can be said of the Flyleather commercial by Merkle & Schwind: including procedural methods early on proved to be a fruitful design choice, since it allowed for sketches that were close to the complex final product while allowing for fast changes and variations.

Both examples point to the flexibility that procedural modelling provides. Manual modelling techniques are generally destructive, meaning once changes to the geometry are made, they can’t be changed by adjusting previous steps. In procedural workflows, most phases of the modelling process are adjustable at any point, making it a non-destructive workflow (Foundry n.d.). Such flexibility is beneficial when dealing with complex geometries such as the ones seen in Annihilation and the Flyleather commercial.

To list the benefits from all of these examples, we can deem procedural modelling useful for:

- Large scale scenes with numerous repetitive assets, like cities or forests
- Quick variations of complex geometries

- Complex details that can be generated with randomness or specific patterns
- Bringing a different approach to the design process

To reach these benefits, some considerations need to be had. One of them would be programming skills required for using these methods. The examples mentioned above are mostly of high-end productions from seasoned creators and developers with prior experience from procedural methods. For artists or architects, learning procedural methods through written programming languages, like CGA shapes, can be inhibitive, as it requires skills that do not overlap with their core 3D skills (Ullrich et al. 2010). This learning curve can be flattened with visual programming languages, which is the topic of chapter 3.

Another consideration for procedural methods is the nature or extent of variety that is required. In video games, procedural generation is often associated with replay-ability, but this can be misleading, if the procedurally generated content doesn't serve the gameplay in a meaningful way. On its initial release, some reviews of No Man's Sky pointed out that while the world to explore is technically vast, the planets lacked substantial variety which resulted in gameplay becoming repetitive fast (Kollar 2016).

In a guide to creating procedural generators for games, game designer Kate Compton points out that if assets are intended to have visual variety, their differences need to be obvious immediately. The nature of differences depends on the asset: for an example, trees need to be just different enough to not look unnaturally identical, while some assets need to stand out and have a distinct character to them. Compton calls the former perceptual differentiation and the latter perceptual uniqueness. (Compton 2016)

And lastly, a consideration for procedural methods is the balance of control and randomness. While randomness is a useful element in generating organic details and quick variation, it also means less control for finer details. This element of surprise is a phenomenon recognized early in the development of procedural texturing, and some creators referred to this as serendipity: finding value in unexpected results (Benoit et al. 1994, 14-15). So, it can be seen as a strength or a

weakness of the method, but either way it is a consideration to be had before picking the method.

To sum up, procedural modelling methods can bring speed and potentially a different design approaches to a production, but mainly show their benefits when there is a need for quick variations and/or when numerous copies of the same asset are needed in a 3D production. Therefore, when considering using procedural modelling methods, the user should consider questions like:

- How familiar is the programming language?
- Will there be numerous copies of the same model in the scene?
- What is the nature of details in the model or scene? Do the details require specific low-level control, or could they be achieved through patterns and randomness?
- Is there a need for generating variations? How unique do these variations need to be, and how much time is there in the production to make those variations?

# 3 NODE-BASED PROGRAMMING AND GEOMETRY NODES

To make procedural methods more accessible to users not familiar with written programming languages, 3D software offer tools, where changes to the base geometry are presented as parameters with clearly labelled names. This is called parametric control and is described by David S. Ebert like so: “to assign to a parameter a meaningful concept (e.g., a number that makes mountains rougher or smoother).“ (Ebert et al. 2003, 2)

For this kind of parametric control, Blender has previously offered procedural modelling tools in the form of modifiers. Modifiers are a vast collection of tools that automate certain edits to the 3D object’s geometry, while being non-destructive unless changes are specifically committed to. Modifiers can be combined in a modifier stack, in which their order affects the final output. Modifiers can automate minor changes, like the Array modifier, or more complicated changes that affect the base geometry more drastically, like the Subdivision Surface modifier. (Blender Documentation Team 2023) Modifiers and similar tools are can also be found in other 3D software, like 3Ds Max.

While presenting parameters in this way can lower the learning curve for procedural modelling, it can also be seen as a setback because of the loss of more detailed control. When values are combined and presented in the UI with abstracted labels, the user gives up control over the individual values that the value in the UI is controlling. This can result in situations where the program changes variables that the user does not need to or want to change. This is where visual programming languages, or VPLs, come in. VPLs are programming languages that allow users to program graphically instead of textually (Kuhail et al. 2020). VPLs make programming more accessible by eliminating the need to learn the syntax of a written programming language.

Some examples of software using a visual programming language, on top of Houdini, include:

- TouchDesigner from Derivative. A software for real-time multimedia installations.
- Nuke. Used for compositing.

- Unreal Engine. Uses node-based systems for both game mechanics and materials.

All of these are examples of VPLs using node-based workflows. Nodes, in the context of VPLs, are blocks of operations that can be connected to each other in what is known as a node network, or a node tree. Node-based workflows grant the user almost the same amount of customizability as written code, while also keeping the presentation more approachable.

### 3.1 Blender and Geometry Nodes development history

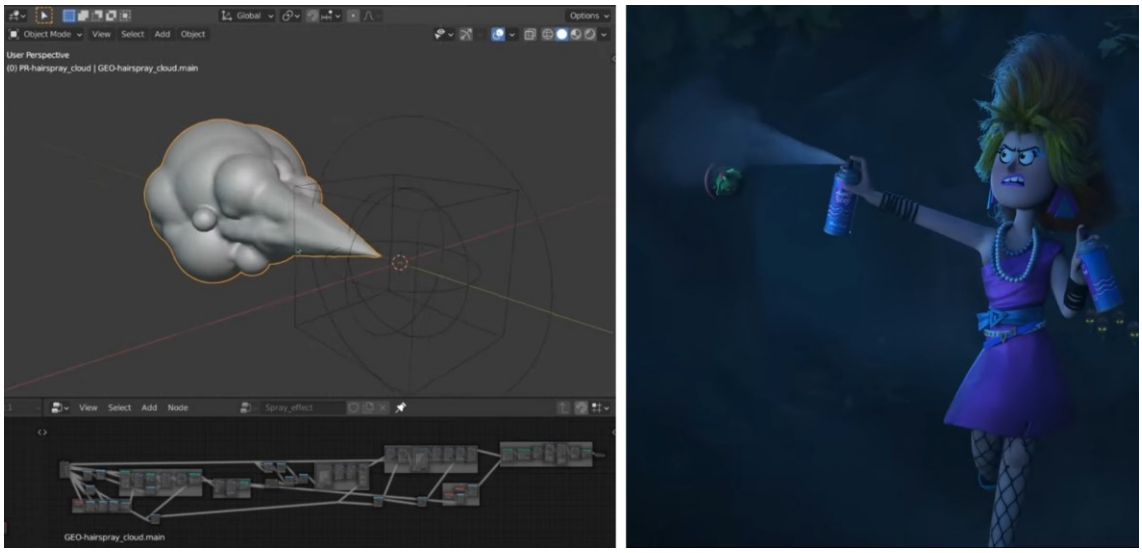
Before Geometry Nodes, Blender has provided procedural node-based tools for materials and compositing. Sometime in 2015-2016, an add-on called Animation Nodes was introduced, which provided a VPL for creating motion graphics procedurally (Lucke n.d.). The developer of Animation Nodes, Jacques Lucke, has since moved on to be the lead developer in the Geometry Nodes project (Blender Conference, 2022).

The development of Geometry Nodes has its history in the Everything Nodes project. The goal of the project is to turn more key features of Blender into node-based tools. This is to improve usability with a unified VPL interface, as well as expanding the software's flexibility with procedural workflows. (Reynish, 2019)

According to a presentation held at the Blender Conference in 2022, the Everything Nodes project first started in 2019 with the development of Particle Nodes which was going to replace the current particle simulation system with a node-based interface. However, in 2020 the focus shifted to the needs of Blender Studios, the creative counterpart of Blender Institution, who were working on the short film Sprite Fright. Instead of dynamic particle simulation, the studio needed static element scattering for set decoration, for things like flowers, leaves, moss and trees (Picture 8). Therefore, simulations and particle systems were postponed in order for the developers to focus on procedural geometry features. (Felinto et. al 2022)

PICTURE 8. Set decoration from Sprite Fright. (Sprite Fright 2021)

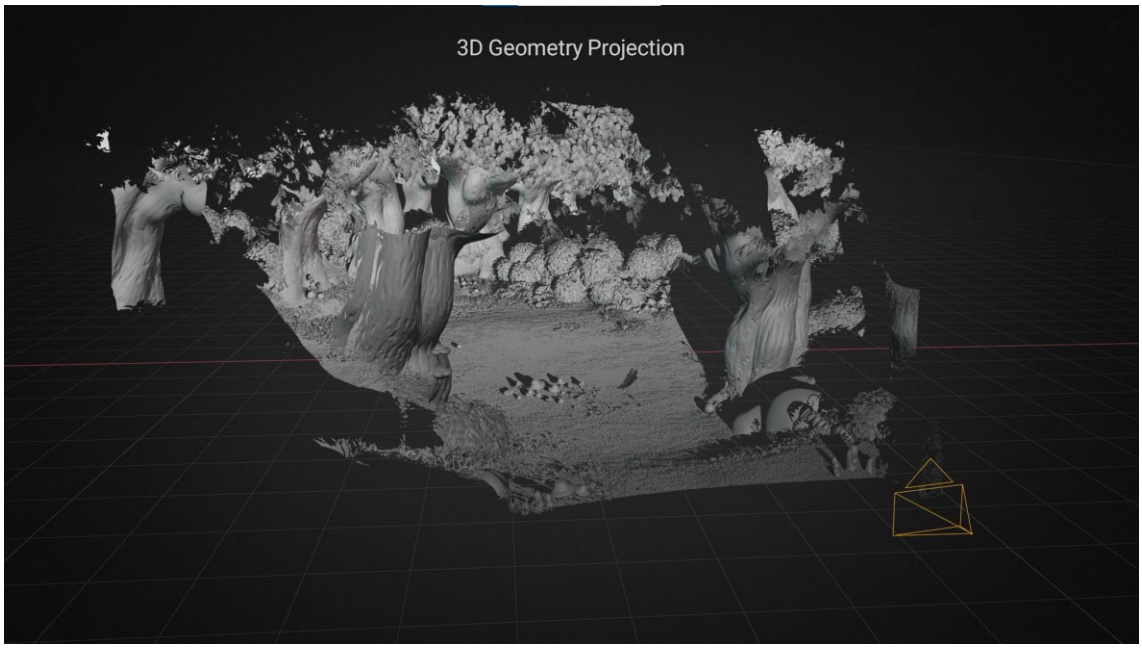
In the final film, on top of set decoration, Geometry Nodes was used for VFX, in the scene where one of the characters is using hairspray as self-defence (Picture 9) (Blender Studio 2021). Another use-case was for placing crawling snails on a character in a close-up shot (Picture 10). (Felinto et al. 2022)




PICTURE 9. Hairspray effect being developed in Blender and the effect in the final film. (Blender Studio 2021; Sprite Fright 2021)

PICTURE 10. Crawling snails added with Geometry Nodes. (Sprite Fright 2021)

Geometry Nodes also helped solving a denoising problem in the renders: some backgrounds in the final rendered scenes had a significant amount of noise despite attempts to raise sample sizes in the render. This was overcome by taking a depth render of the background and constructing it in Geometry Nodes as a 3D projection mesh with baked lighting (Picture 11). Since the lighting was baked into the texture, the render engine didn't need to calculate light rays and the render came out almost completely noiseless. While this method had its limitations, mainly only working for the background and for specific scene layouts, it still allowed for rendering a static background with camera movement, and solved the noise issue that other methods couldn't. (Thommes 2021)



PICTURE 11. 3D projection mesh made from a background render. (Thommes 2021)

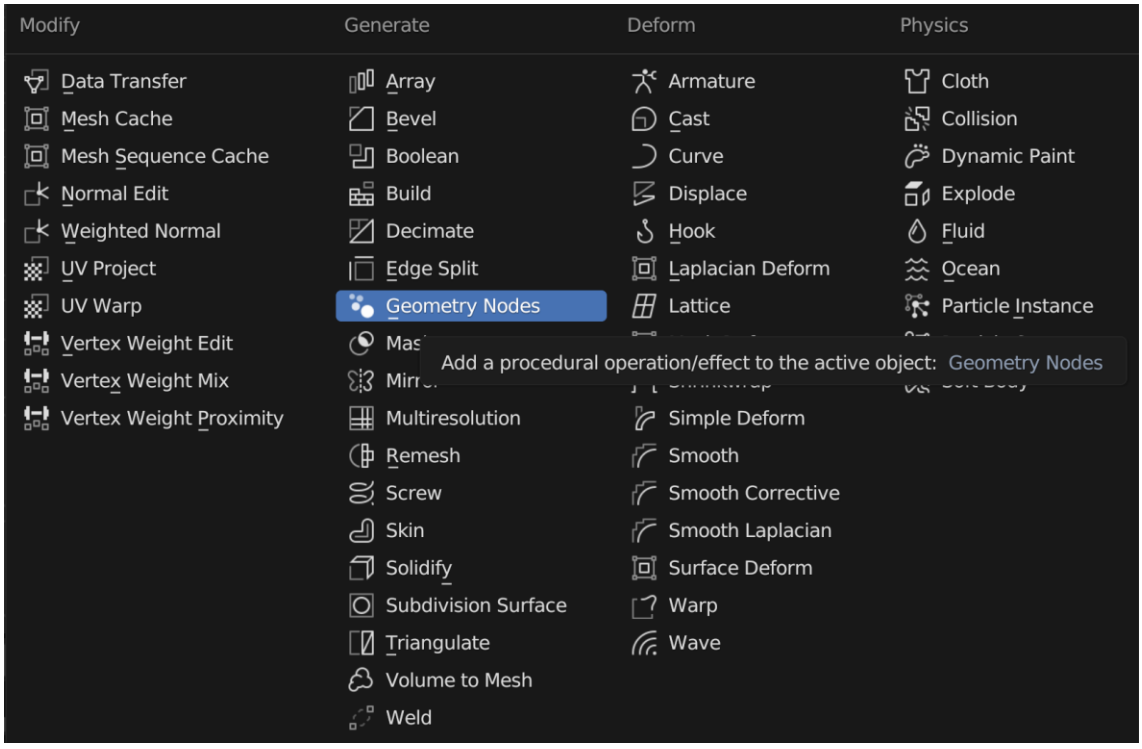
Since the release of Sprite Fright, Geometry Nodes has seen many updates, including hair grooming tools based on Geometry Nodes and curve objects introduced in version 3.3. LTS (Picture 12). These hair grooming tools were developed for a later open film called “Charge”, released in 2022 (Felinto 2022). At the time of writing this thesis, the latest stable release of Blender is the version 3.6. LTS, which introduced a new Simulation system for creating custom simulation systems, thus bringing the idea of Particle Nodes into Geometry Nodes. These additions are not used in the case study of this thesis.

PICTURE 12. Geometry Node based hair grooming tools on a character from “Charge”. (Felinto 2022)

# 4 EXAMINING THE UI AND THE FUNCTIONALITIES OF GEOMETRY NODES

Geometry Nodes are added to the 3D object as a modifier, like adding Subdivision Surface, Lattice, or any other modifier. Users that are already more familiar with Blender could understand Geometry Nodes as a modifier built by the user from scratch.

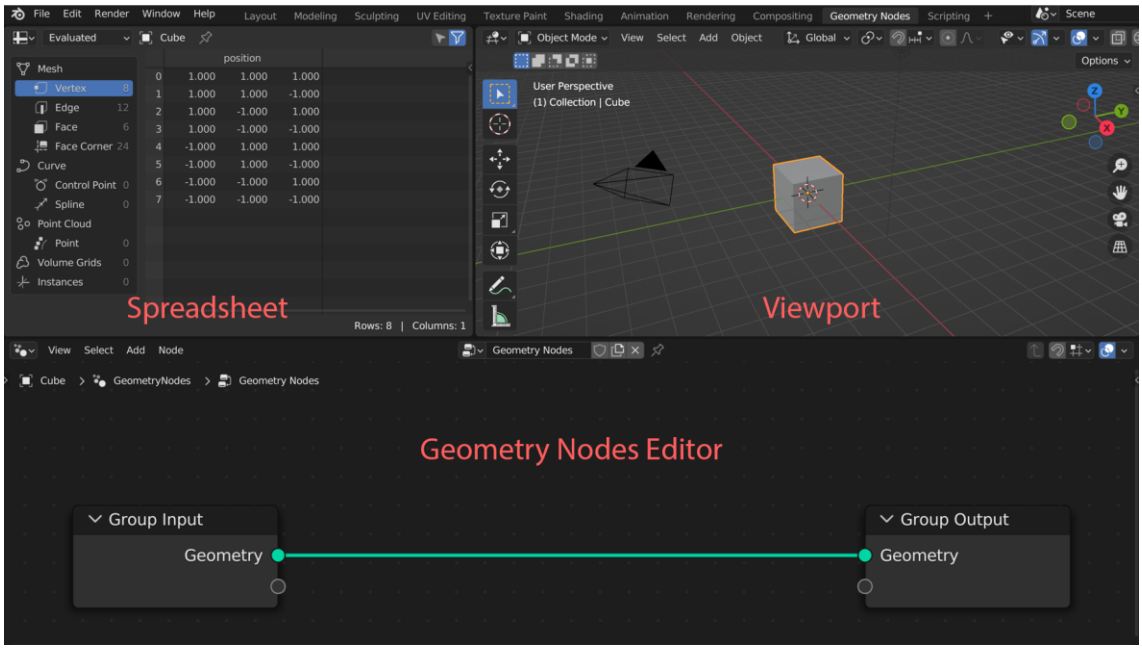
In the Modifier list, Geometry Nodes is found under the main category “Generate” (Picture 13). When selected, it adds an empty Geometry Nodes modifier into the Modifier Stack of the object. To create a node tree, the user has to click “+New”. This new tree will be added to a list, from where the tree can be added into another object, similar to how the same Material can be used on multiple objects.




PICTURE 13. Geometry Nodes in the modifier list.

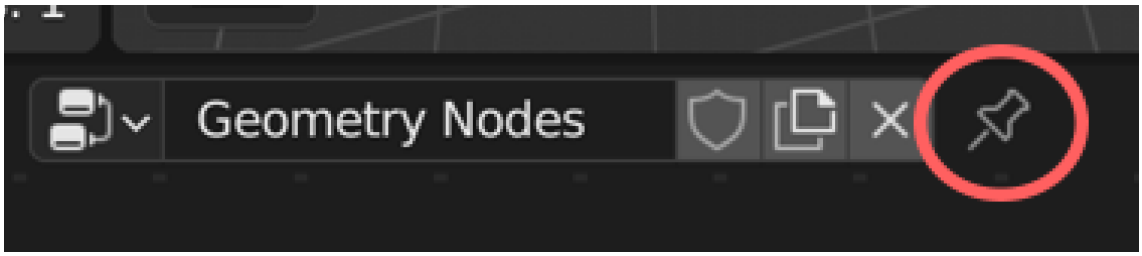
The Geometry Nodes tree can be edited by dragging up a new panel and replacing the current editor mode with Geometry Nodes Editor. A quicker way is to select the Geometry Nodes workspace from the top, which brings up panels for the Spreadsheet, 3D Viewport and the Geometry Nodes editor (Picture 14). The Spreadsheet shows values for various attributes on the selected object, including

the amount of geometry, vertex positions and custom attributes, among other things. The spreadsheet can be used for troubleshooting if the node tree doesn't provide the desired outcome.




PICTURE 14. The Geometry Nodes workspace.

The editor is always showing the node tree of the selected Geometry Nodes modifier. The node tree can be pinned, which means it won't change when you for an example select a different object that has a different node tree attached to it (Picture 15).

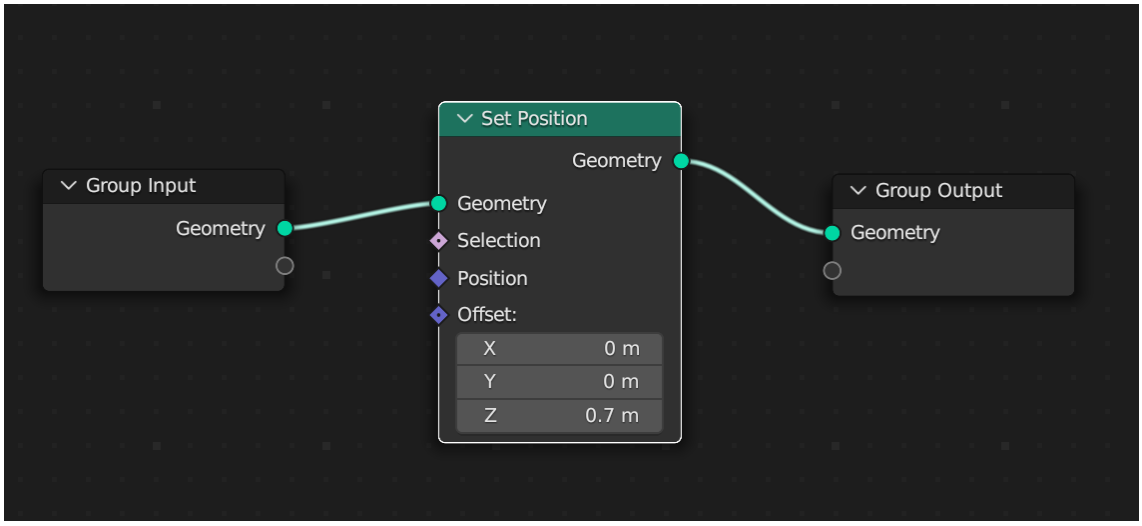



PICTURE 15. Pin icon.

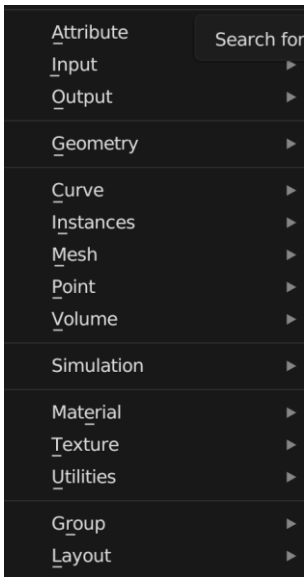
### 4.1 Node types

Nodes can have value slots on either side of it. The slots on the left side are input slots and the slots on the right are output slots. Some nodes have editable values

inside them which can either be edited inside the node or be connected to a different node. The node tree in picture 16 moves the entire object 0.7 metres up.




PICTURE 16. Simple Geometry Nodes tree that moves the entire input object up by 0.7 metres.




PICTURE 17. Node menu.

Clicking “Add” on top of the Geometry Nodes editor, or pressing Shift+A, brings up the Node menu (Picture 17). This menu is a list of all the nodes categorized by their usage:

- Attribute: Creating custom attributes or saving attributes like “Position” on an earlier stage in the node tree for use at a later point.

- Input: For getting information about the scene, objects in the scene or the current frame. Here you will also find nodes for setting constant values inside the node tree.
- Output: This category has two nodes: Group Output and Viewer Node. Group Output is used to show the final result of the node tree. Viewer Node can be used to troubleshoot the node-tree inside the spreadsheet or the viewport.
- Geometry: Reading or manipulating geometry information.
- Group nodes: Each node tree has a Group Input and a Group Output node. Multiple nodes can be grouped together, creating a Group node, which functions like a sub-node tree inside the main node tree. The user can go in and out of group nodes by pressing TAB while having the group node selected.
    - Group input: bring data inside the node tree. If values from inside the node tree are connected on the Group input slots, those values will appear in the Modifier menu. These values can be given custom labels, thus giving the node tree user quicker and more intuitive access to the most important values.
    - Group output: sends data out of the node tree. To make geometry appear, it needs to be connected to the Geometry slot in the node tree.
- Curve, Instances, Mesh, Point and Volume refer to different types of geometry information. Nodes in these categories can be used to create, manipulate or convert between different geometry types.
- Simulation Zone is used to create custom simulation systems. Nodes and values placed inside the Simulation Zone produce different values for each frame of animation. This will be explained further in chapter 4.3.
- Material nodes are used to set materials for the geometry.
- Textures are procedurally generated patterns, which can be used as input in several ways. There is also the Image Texture node, which can be used to bring 2D images from the user's device.
- Utilities is a miscellaneous collection of nodes for data modification. Most important ones are Math and Vector Math nodes, which can execute various mathematical functions.

- Layout includes Reroute and Frame, which can be used to organize the node tree for better readability. Frames can be given labels for annotation.

In addition to these, there is a category for Hair nodes, which have multiple nodes for hair curve manipulation. This is a newer addition to Geometry Nodes which will not be examined further in this thesis.

## 4.2 Data types

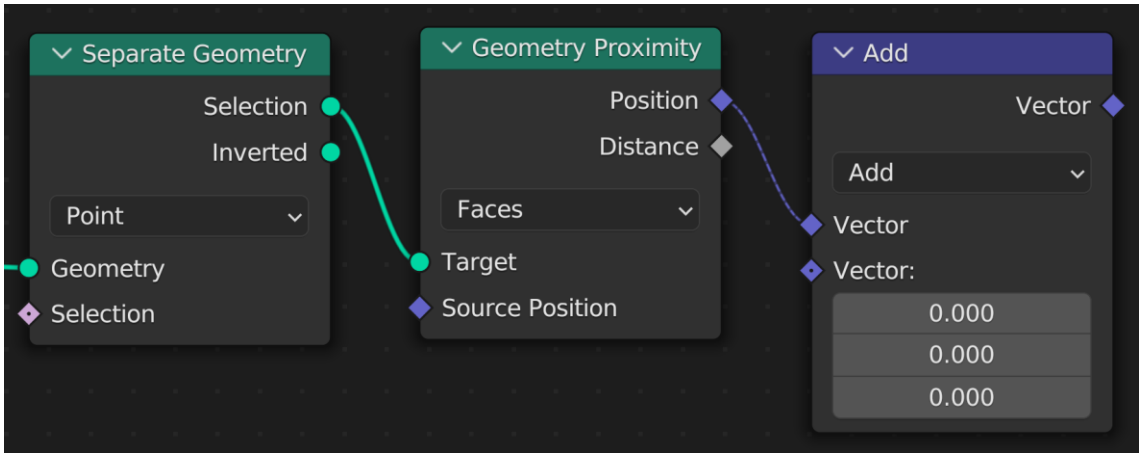
Identifying different data types and the number of values travelling between nodes is vital to understanding Geometry Nodes. A collection of multiple different values that are transferred or manipulated in a node tree are known as fields. An example of a field could be the different vector positions of points in a cube object.

There are in total 10 different types of data that can be used in a node tree. Each type of data with their color code is listed in Table 1.

TABLE 1. Input and output slots color coded by data type.

| Color code | Data type | Description |
|---|---|---|
|  | Geometry | Any type of geometry (point, edge, face, volume etc.) |
|  | Booleans | True or False values, used for selections |
|  | Integers | Whole numbers. These can be used for indices, IDs or other numbers without decimals. |
|  | Float data / Value | Any positive or negative number with decimal points. |
|  | Vector data | Position in 3D space. Can store three float values: X, Y and Z. |
|  | Color data | Used for storing image data, like textures or gradients. |
|  | Materials | Selecting materials for geometries. |
|  | Objects | Objects in the scene. |
|  | Collections | Group of objects in the scene. |
|  | String | Text input. This is a less used data type, but can be useful if, for example, the user wants to turn text into a 3D shape. |

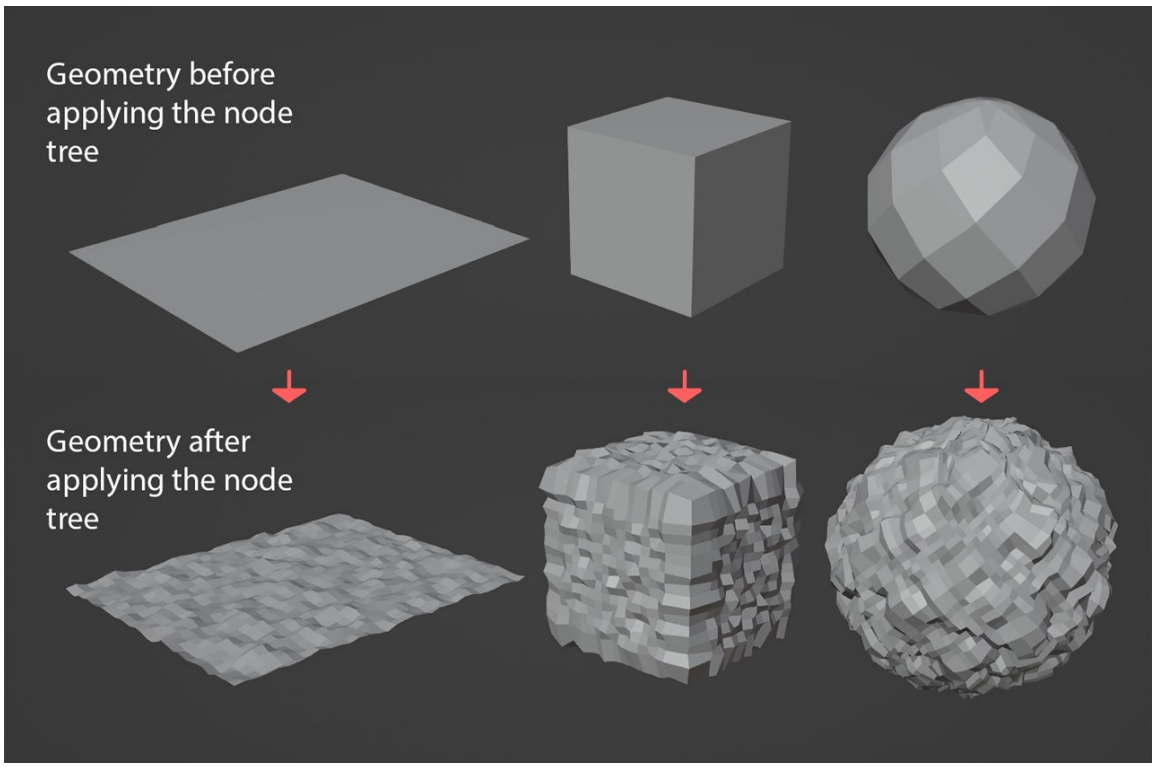
Whether a node is handling a single value or a field is visualized with the shape of the input and output slots. Input or output value slots can either have a circle, a diamond shape, or a diamond shape with a black dot (Picture 18). The circle means that this slot can receive or send only a single value. The diamond shape signifies a slot that is giving out or receiving fields. The diamond shape with a dot signifies a slot that can process fields but is only receiving or sending a single value. Fields are also visualized in the node noodles: single values have a solid noodle, while fields have a dotted noodle. In picture 18, the light green geometry noodle is solid, since it is only passing one geometry object, but the blue vector noodle is dotted as it passes multiple positions of faces.



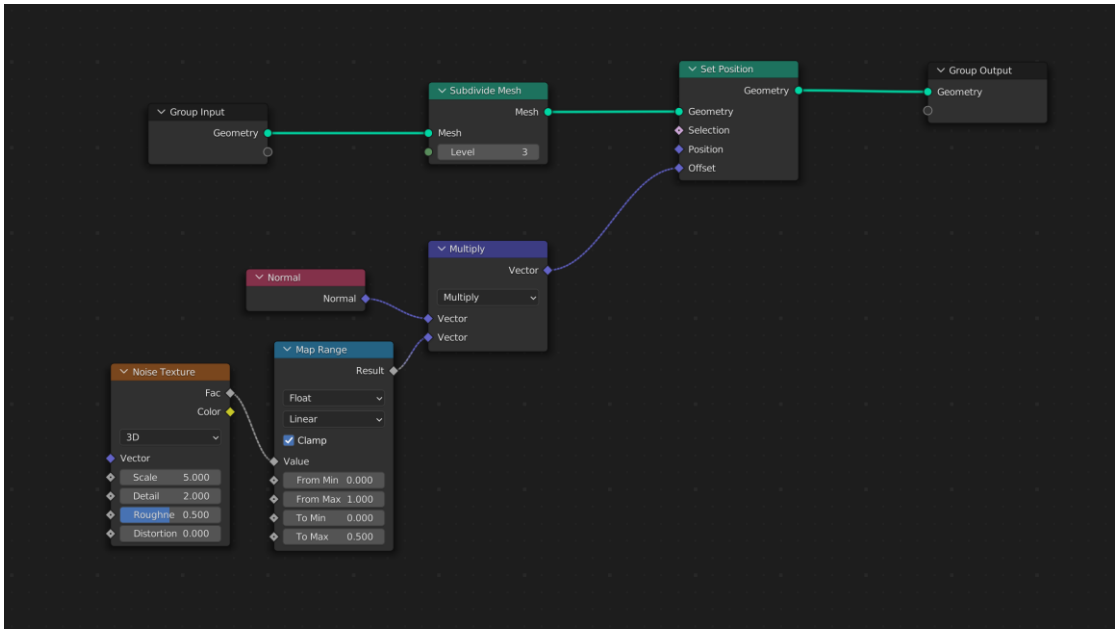

PICTURE 18. Node tree showing types of data input and output slots.

### 4.3 Dataflow in a Geometry Nodes tree

The following images will illustrate how the data moves in a Geometry Nodes tree. The example node tree I have made produces a ripple effect on the surface of the mesh that it is applied to (Picture 19). This is achieved by increasing the amount of geometry in the base mesh and offsetting the vertices according to the normal values of the mesh faces. The node tree for this can be seen in picture 20. Normals are vector values referring to the orientation of polygons. Thus, using normal values allows us to push the vertices outward according to the geometry orientation.




PICTURE 19. The example node tree used on a plane, a cube, and a low poly sphere.




PICTURE 20. The example node tree.

The dataflow in a Geometry Nodes tree goes mostly from left to right, but nodes can also be freely placed on top of each other. Certain nodes also send information from right to left, since some nodes require geometry information to produce values. In the case of our example node tree, the Normal input node on the

left receives geometry information from the Set Position node and sends back the normal values of the used geometry (Figure 1). Nodes like Noise Texture do not require any vector input values to function.

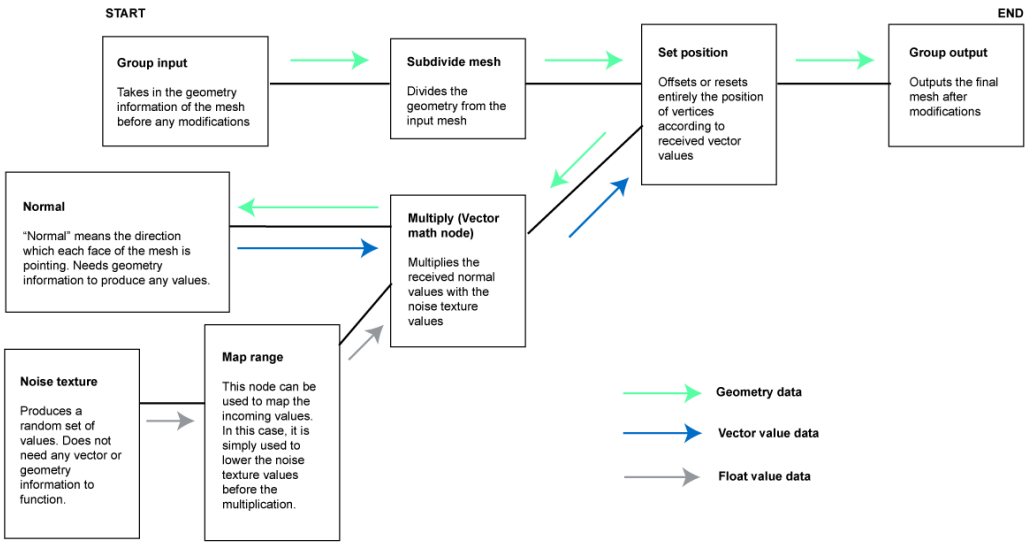


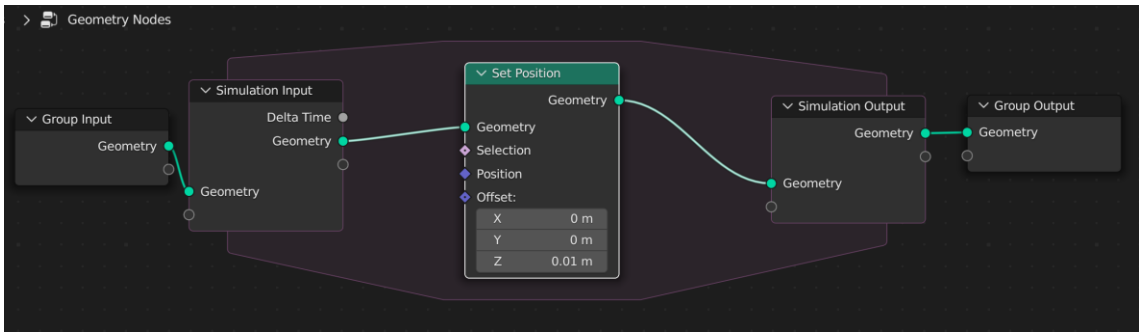

FIGURE 1. Dataflow and the purpose of each node in the example node tree shown in Picture 2. Different types of data are illustrated with colour coded arrows.

Even simple node trees like this can increase the amount of geometry in the mesh drastically, especially if the desired effect requires subdivisions. This is particularly true in the case of spheres, where the starting geometry is already complex (Table 2). An increase in geometry data also increases render times, which is why optimizing node trees to each base geometry is important.

TABLE 2. Comparison of the amount of geometry before and after applying the example node tree.

| Mesh | Number of vertices before the used node tree | Number of vertices after the used node tree |
|---|---|---|
| **Plane** | 25 | 1089 |
| **Cube** | 26 | 1538 |
| **Sphere** | 66 | 4098 |

Another node type that changes the direction of the data flow is the Simulation Zone introduced in Blender 3.6. Nodes placed inside the Simulation Zone will be executed recursively every frame. The example node tree in Picture 21 causes the base geometry to move up 0.01 m each frame.




PICTURE 21. Simple example of a Simulation Zone. This node tree moves the input geometry by 0.01 m each frame.

# 5 CASE STUDY: PROCEDURAL ASSETS FOR PRODUCT 21

Product 21 is the working title of a sci-fi animated short film project. Before the writing of this thesis, the film has had a pre-production phase, where a plot summary was craft, concept art was made with Photoshop and Blender, and a listed of required assets was compiled (Appendix 1). From this list of assets, a few assets that would benefit from a procedural modelling workflow stood out. Two of them will work as the case study for this thesis.

The plot of the film revolves around a shapeshifting science experiment, who escapes a research facility. The escape is hindered by guards and the character's inability to control its shapeshifting abilities. The character's shapeshifting animations seemed like they would be complicated and time consuming to execute manually with basic character rigging tools and shape keys. This sparked an idea to test out if the character's geometry could be generated in real-time using the armature as input for Geometry Nodes. This Geometry Nodes based character rig was unsuccessful, but it was then developed into a monster generator for environment modelling.

The other asset created during this thesis project was also related to environment modelling. One of the planned scenes shows the main character running around a corridor that seems endless and convoluted, to evoke a feeling of hopelessness and to make the environment seem impossible for the main character to navigate. This kind of environment and sequence would be time-consuming to plan and model. Therefore, it decided to create a Geometry Nodes tree that could generate the corridor procedurally as opposed to a static environment, to see if this would allow for a more improvised and flexible approach to animating the action sequence.

Both examples include plenty of instancing. Instances are copies of a 3D object that refer to the original object. This means that the 3D software only has to calculate the mesh once, and then use those calculations again in the instance object, resulting in better performance (de Vries n.d.).

## 5.1 Failed main character rig experiment

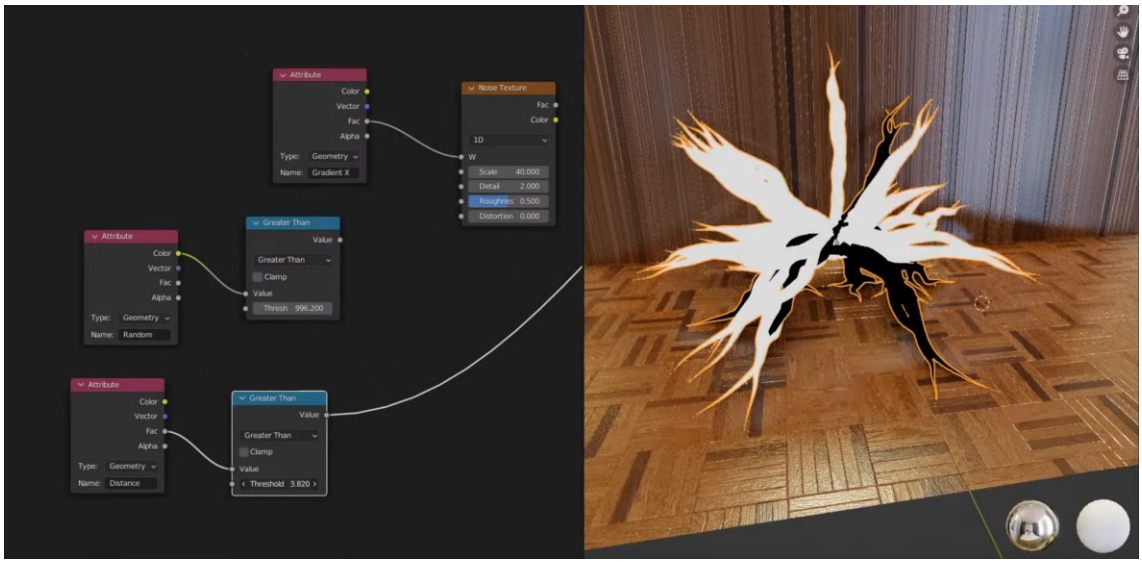
The main character's concept had three major sources of inspiration: Venom of the Spider-Man franchise, different development stages of a moth and the shapeshifting creature from Jeff Vandermeer's book Borne. The character was to have a varying number of limbs, which would be complicated and cumbersome to set up and animate manually. The goal was also to have the character's transformations, mainly the change in size, to be possible with a single rig (Picture 22).

PICTURE 22. Concept art of the main character of Product 21.

A previous experiment with a similar character design was from my small animation called "Ball Doggy" which features an absurdly long dog with numerous limbs (Picture 23). In this rig, only the first and last pairs of legs were animated manually. The rest of the legs were added to the character with the Animation Nodes add-on. This worked out well for that animation, where the legs are on screen for a short time, but the legs were difficult to rotate to the right direction and in the end the only movement they have is a slow up-and-down motion. This would not work in "Product 21" since the character's limbs would need to move and touch the floor in a believable manner. Therefore, the focus of development was automating the leg placement and animation.

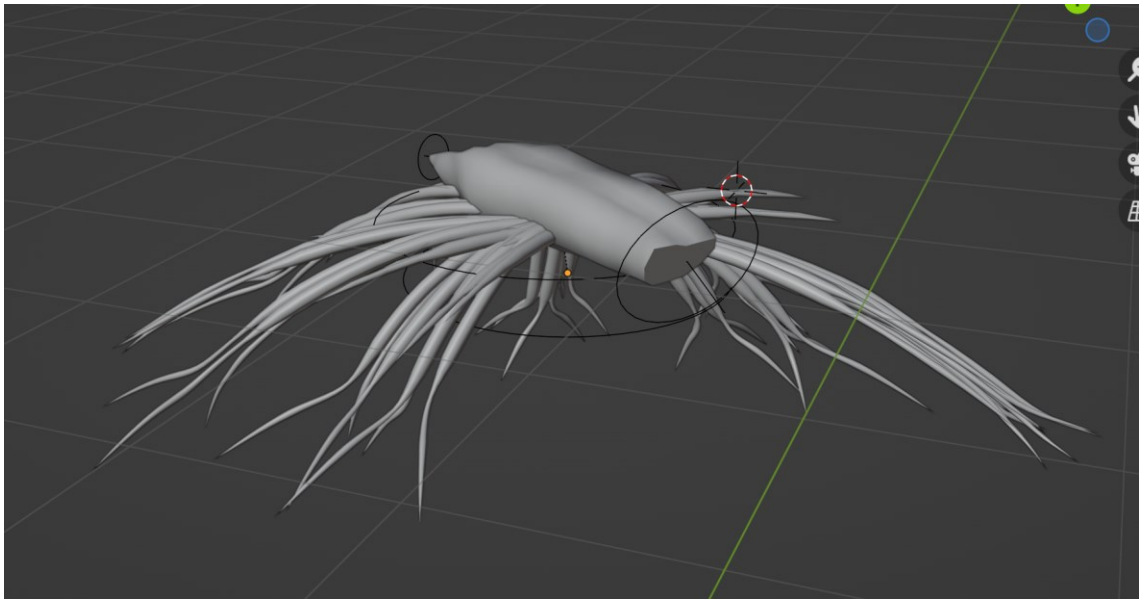
PICTURE 23. Screenshot of the animation “Ball Doggy”. (Collection 1: Coffee guy, Running man & Ball Doggy 2022)

During the pre-production and character design stage, some preliminary tests were done on creating the Geometry Nodes setup for the character. This initial test was inspired by a YouTube tutorial from the user Cartesian Caramel featuring a creature with procedural legs (Picture 24). The effect is achieved by instancing curve objects into the scene, which are then attracted to the origin point of the Geometry Nodes object. These curves are then revealed when the character moves closer to them.




PICTURE 24. Creature effect in Cartesian Caramel’s tutorial. (Cartesian Caramel 2022)

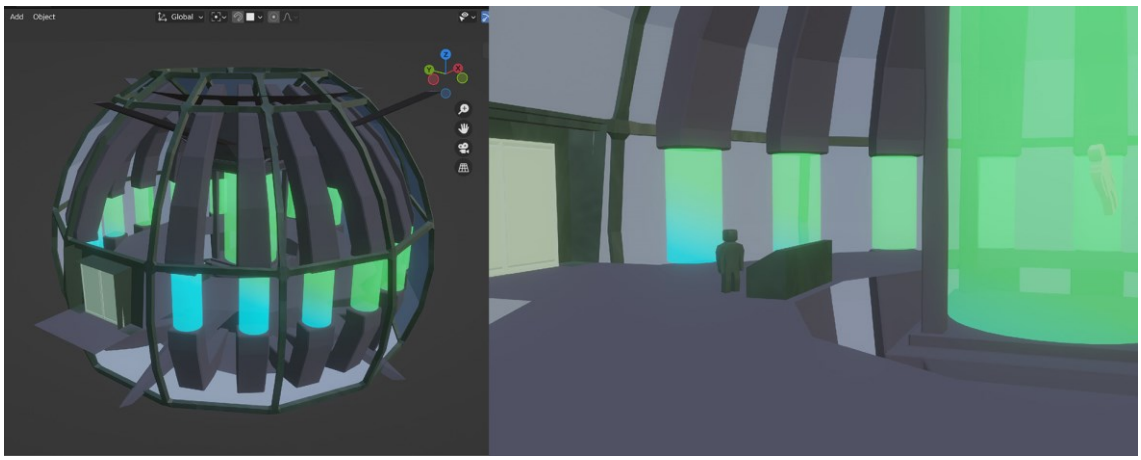
The test resulted in a worm-like character with geometry being generated around a curve object with procedural legs attached to the object origin point (Picture 25). However, it quickly became clear that the desired result could not be achieved with this node setup under the time constraints of this thesis project. The legs being attracted to a single point made it seem like the character's body and legs are clearly different objects and had a look that deviated largely from the initial concept art. Therefore, this effort was abandoned early on. The work up until this point, however, sparked up an idea for how to develop the node tree to be used in a different scene.



PICTURE 25. Initial creature rig that was abandoned.

### 5.1.1 Creature generator

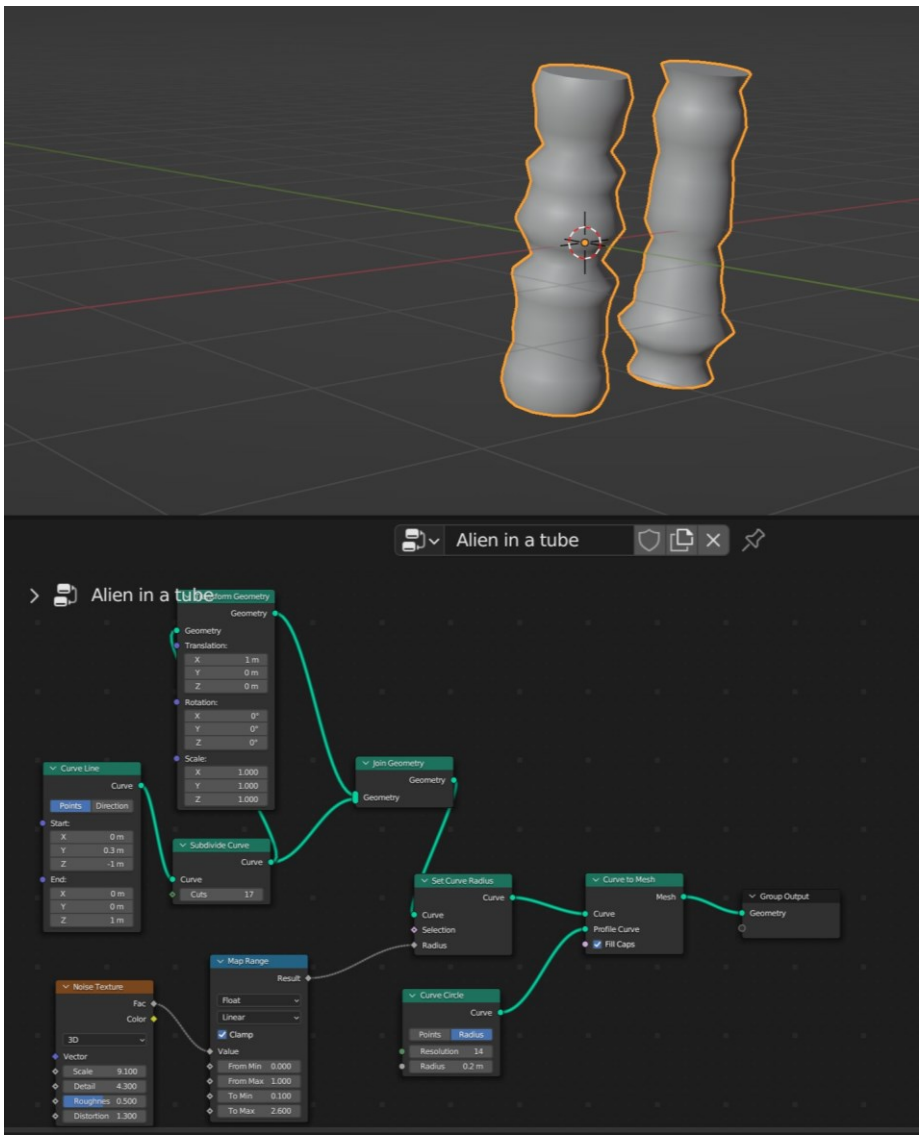
In the beginning of Product 21, there is a scene taking place in a spherical room where experiments like our main character are kept in glass tubes. The initial concept art for the room (Picture 26) features only around 20 tubes, but the aim was to have the room be filled with at least five times as many tubes, each containing a unique variation of a creature.



PICTURE 26. Initial 3D concept of the laboratory.

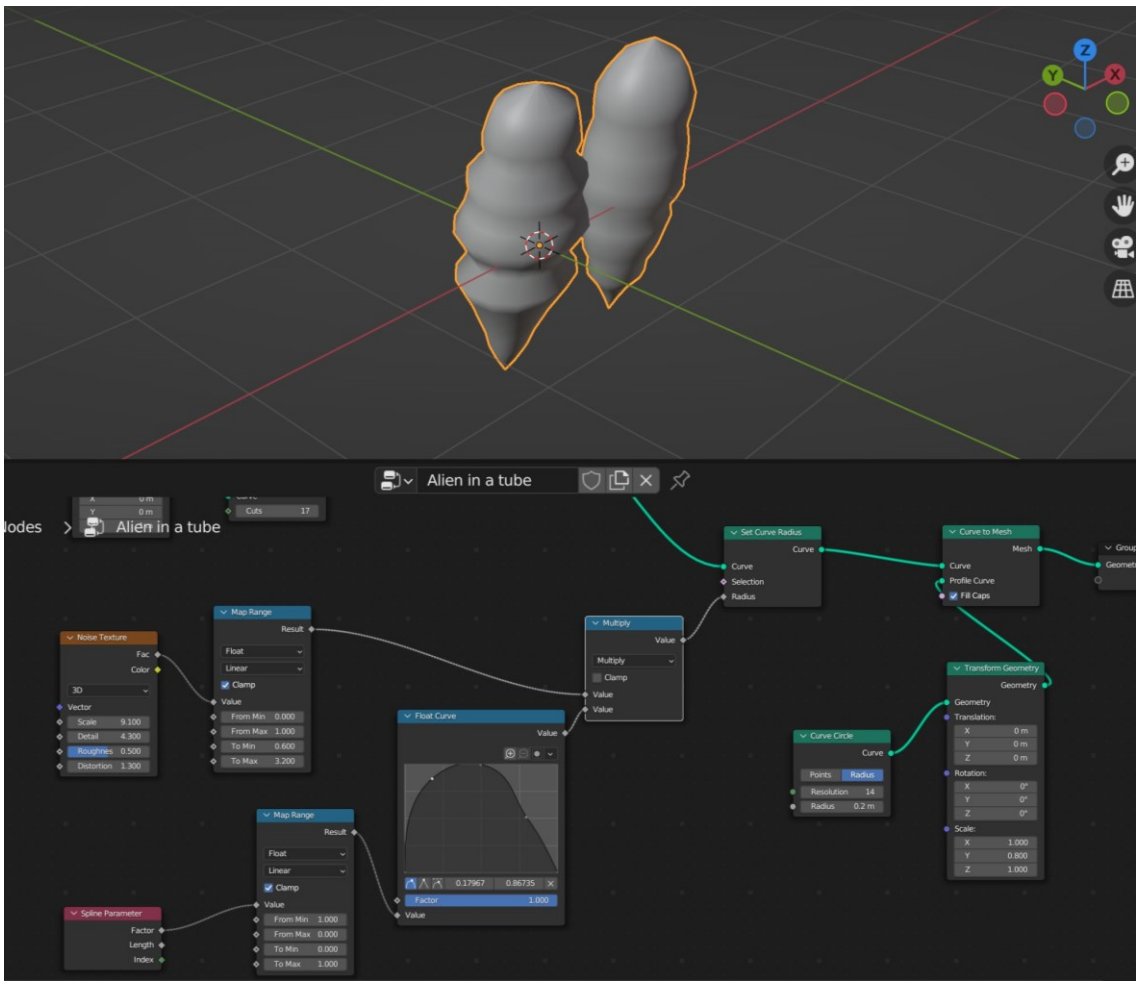
The simplest way to do this manually would probably be to model and rig the main character, and then copying and reposing the rig for each tube. While there would be control over each individual pose, this would probably still take some time and would produce a needless amount of extra geometry from using a hero asset multiple times. In addition, if the aim was to have more complicated variance in geometry (body shape, number of legs or eyes etc.), it would require even more tweaking and remodelling. Considering this need for multiple copies of the same asset with varying details, the procedural workflow was deemed justified.

Development started from the geometry of the creature's body. The body is generated around a spline created with the Bezier Segment node. The shape of a Bezier Segment is controlled with handles on each end, which later allowed me to randomize twisted poses by controlling the handle positions. The Curve To Mesh node was used to generate a tube around the object. Curve To Mesh takes in any primitive curve object to create a mesh in the shape of the input curve object. The thickness of the mesh depends on the curve radius, which is a float value given to each control point. To create an individual body shape to each creature, the radius value was randomized with a Noise Texture node (Picture 27).



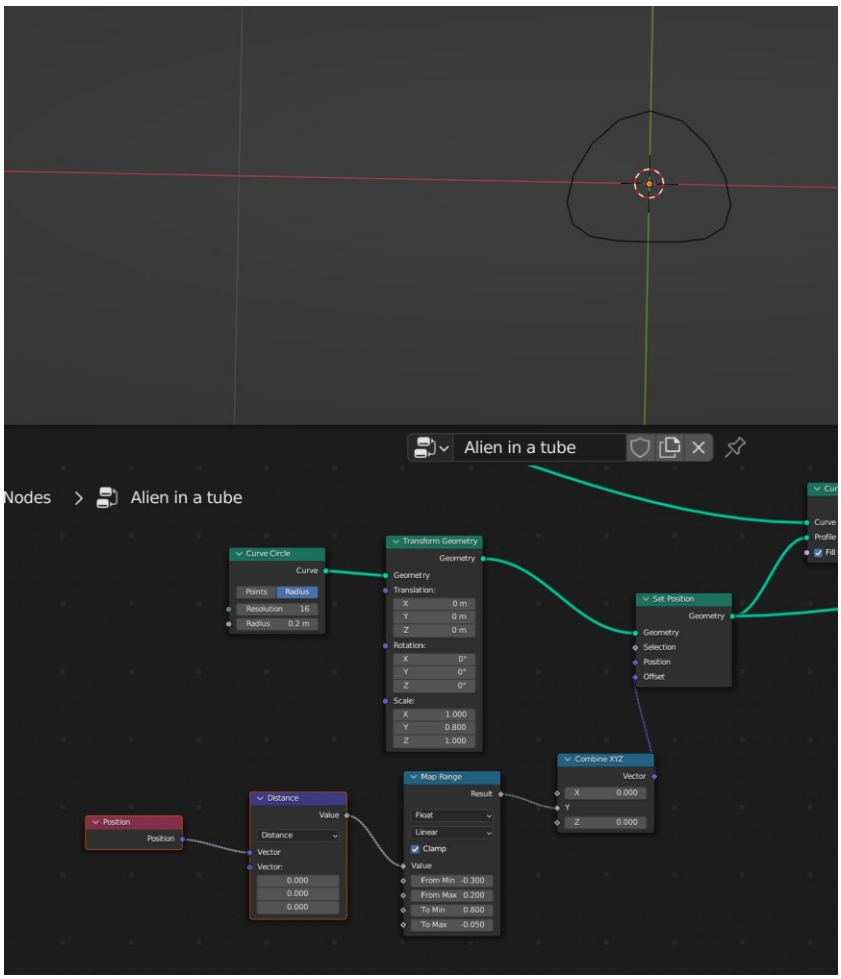

PICTURE 27. Two curve objects turned into a mesh.

To make the body shape narrow down towards both ends of the body, the Factor value from the Spline Info node was used. The Factor value refers to the position of each control point along a spline: 0 is the start of the spline and 1 is the end of the spline. So, the curve radius needed to be set to 0 on both the start and the end of the spline, and then get thicker between them. To do this the Factor value was connected to a Float Curve node which can be used to remap a float value between 0 and 1 with a graph editor (Picture 28). This gave control of the overall body shape of the creature: the creature was given a thicker chest, which rapidly narrows down into a tail.



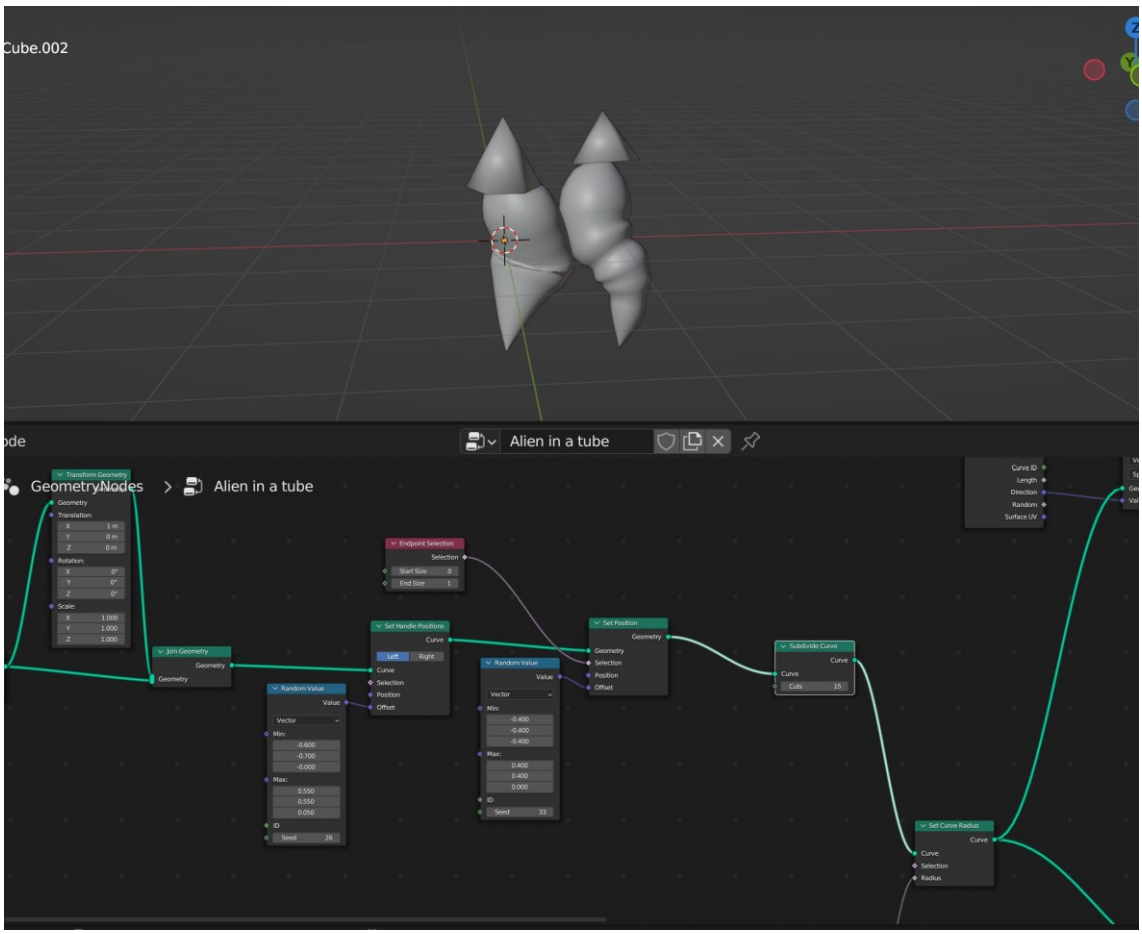

PICTURE 28. The body radius narrowed down to zero towards each end of the splines, to create a wide body and a narrow tail.

Until now, a perfectly round Curve Circle object had been used for the Curve Profile. The body was to have more of a rounded triangle shape, so that the body would widen towards the belly and the back would have a hump. This was done procedurally in Geometry Nodes by editing the position of the Curve Circle control points with the Set Position node. This might have been a mistake, as the goal was a specific shape that could have been modelled much faster manually. The desired shape was achieved by flattening a curve circle and then offsetting the points according to their proximity to the Y-axis (Picture 29).



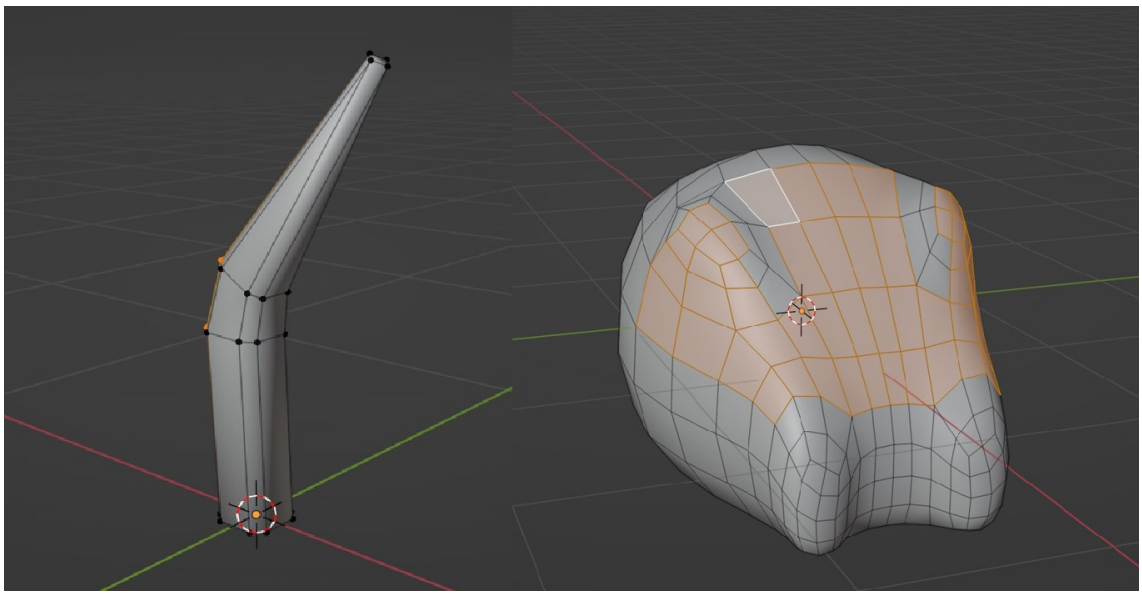

PICTURE 29. Node tree for modelling the desired body shape.

Before moving on to the other parts of the body, the rotation of the head needed to be affected the rotation of the body. To test that, the pose was randomized, and a placeholder head was added. The body curve handles were accessed with the Set Handle Positions node, which allowed an offset vector value to be put in. This is also one of the many phases that uses the Random Value node, which gives a random value according to the range that the user puts in (Picture 30).




PICTURE 30. Pose randomized with the Set Handle Positions node and the Set Position node.

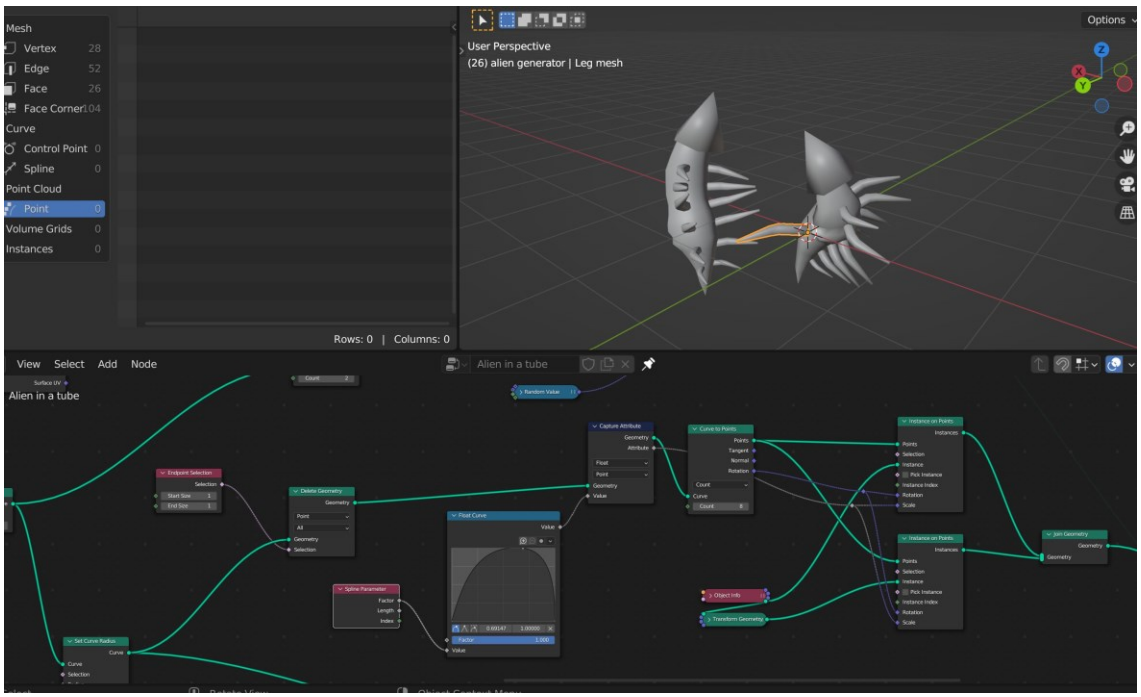
Since the legs needed to be placed and rotated according to the pose, both the head and the leg objects were modelled manually at this point. For the head, a vertex group was assigned, which would later be used for instancing eyes (Picture 31).



PICTURE 31. Leg and head modelling. Selecting faces for eye instancing.

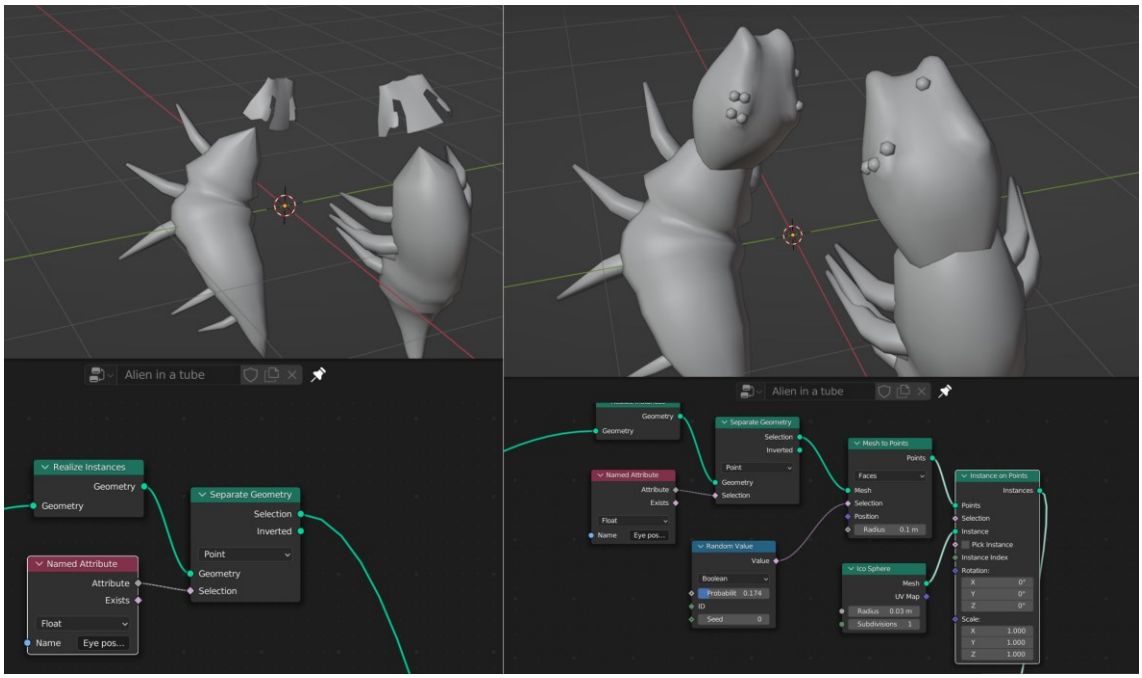
After modelling, it was time to attach the body parts by instancing. Leg points were created by attaching the body curve to a Curve To Points node. Head and tail points were excluded. These points were then connected to the Instance on Points node, which allows any shape or object to be instanced in the model. The Curve To Points node saves the rotation of the curve for each point created, which allowed for the leg instances to be rotated accordingly.

The legs were to become longer towards the chest, so the Spline Parameter and Float Curve setup was used again to control the leg length along the body curve. However, because the body spline was converted into points, the Factor parameter didn't exist anymore. This was fixed by adding a Capture Attribute node before the conversion to points. Capture Attribute takes any value and saves it into the geometry for later use as a custom attribute. The attribute was then used to scale the leg instances. These kinds of challenges with conversions of geometry information and selection of vertices seem to be typical to procedural methods. (Picture 32)




PICTURE 32. Attaching legs to the bodies through instancing.

The head was attached with this same method. After this, it was time to attach the eyes to the previously created vertex groups. Vertex groups can be accessed with the Named Attribute node. I changed the grouped faces into points with the Mesh to Points node and instanced a simple Ico Sphere object into them. I wanted each creature to have a different set of eyes, so I attached a Random Value node into the Selection input on the Mesh to Points node (Picture 33).

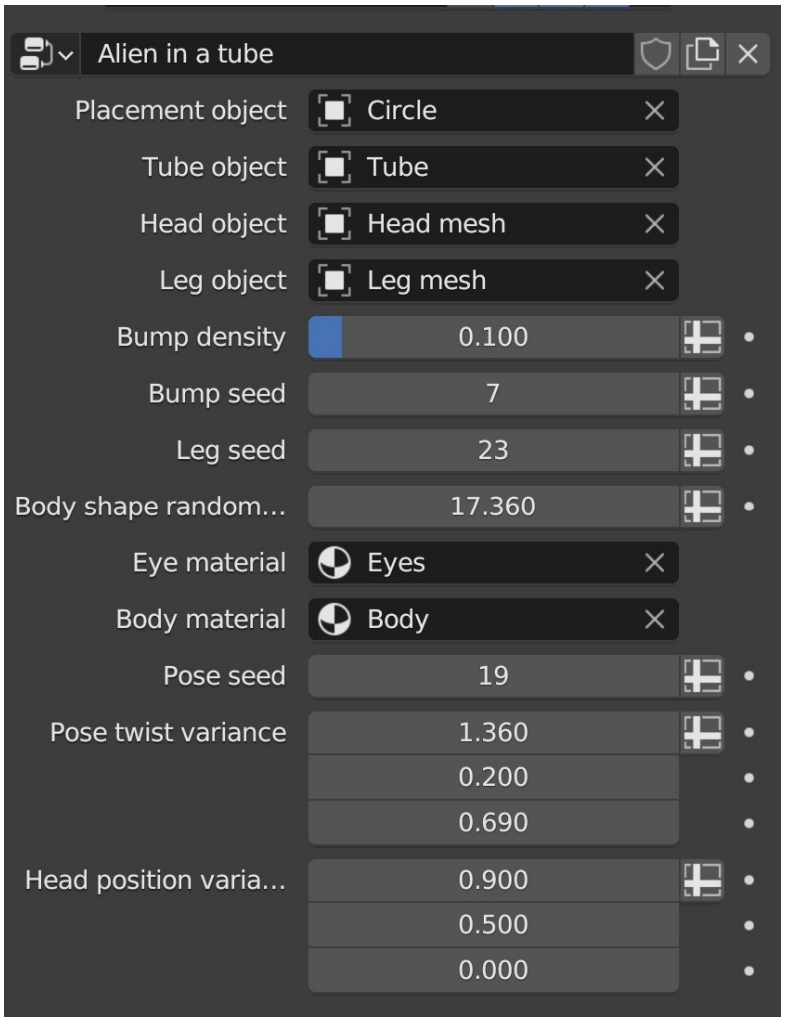


PICTURE 33. Accessing previously created vertex group and using it for eye instancing.

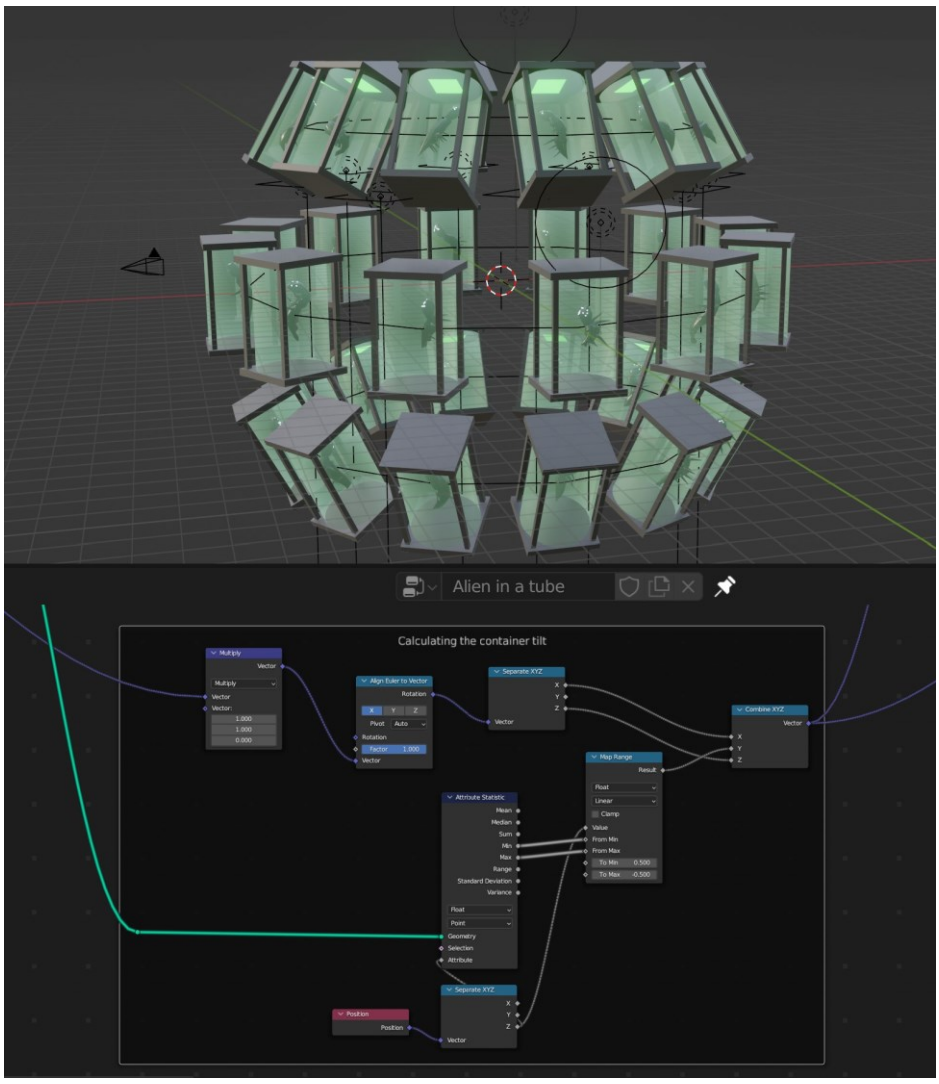
After the legs and the head part was done, it was time to figure out how to place the creatures in the final setup. I decided to use a normal circle mesh for the positions, since manual editing makes the locations easier to handle in the final scene. I once again used the Instance on Points node to make copies of the initial Bezier curve. Since instances are only copies of the original object, their geometry cannot be edited besides scale and rotation. To make the instances into editable curve objects, I used the Realize Instances node (Picture 34).


PICTURE 34. The creatures copied with using a circle mesh as an input object for the instancing.

At this point, the most important functionalities of the node tree were done. After this, I connected some nodes into the Group Input node, which made them appear as labelled parameters in the modifier stack (Picture 35). I also quickly modelled a tube for the creatures to be placed inside. The values determining the variation of the creature poses needed to be set so that they would not exceed the boundaries of the tube object. When the tubes were attached, more calculations needed to be made for the rotation of the tubes and the creatures, so that they point into the centre of the instancing mesh (Picture 36). After this, the desired result was reached.




PICTURE 35. Editable parameters in the modifier view.




PICTURE 36. Pointing the creatures and tubes towards the center of the instancing object.

## 5.2 Corridor generator

To make the kind of modular corridor without a Geometry Nodes setup, one could model all the different wall and floor pieces manually, and then combine them in the desired way as Linked Duplicates. Linked Duplicates are the same as Instances, as in they are copies of the main object without geometry editing capabilities and are affected by changes to the original object. These could then be moved manually and connected with the snapping tools. Long walls comprised of one type of wall piece could be copied with the Array modifier.

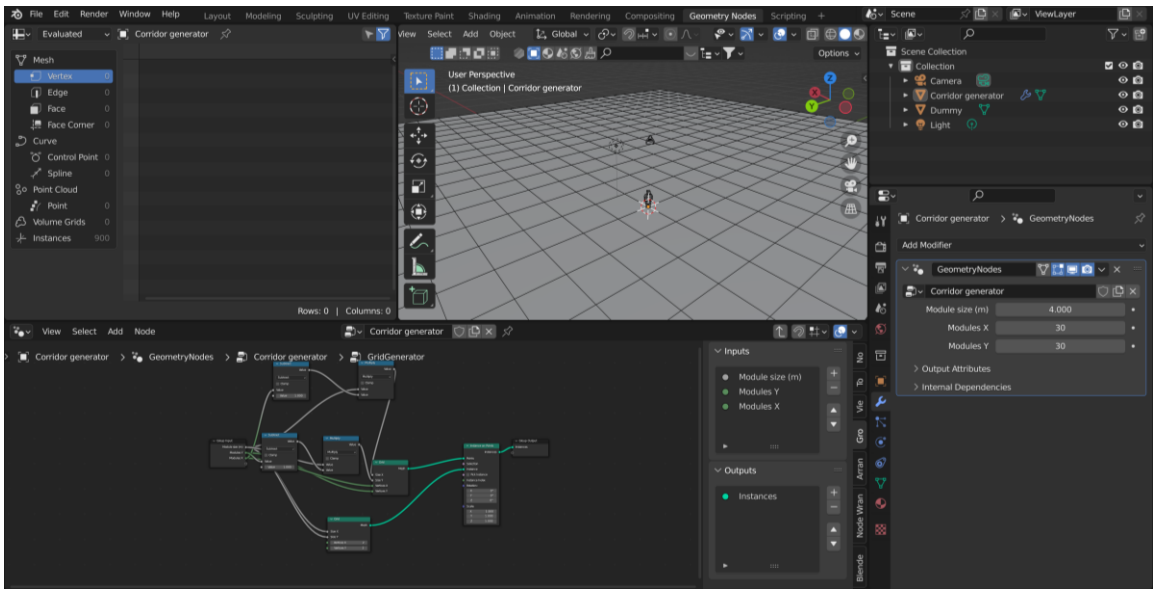
With this kind of setup, even if all the wall pieces could be changed by editing the original wall piece, placing and copying would still require plenty of manual editing. Anytime the user would want to change the way the corridors branch out, they would most likely have to replace wall pieces with ones that work with corners, for an example. This already slows down the environment modelling process, which is why the complicated chase scene in the story would require lots of planning with maps of the location and possibly detailed layout drawings. Done manually, this kind of planning would take a lot of time.

To save time on planning and changing the final corridor layout, the goal was to automate the corridor modelling with Geometry Nodes. The idea of the corridor generator was as follows:

- The corridor is generated on top of a 2D grid, which allows modular editing.
- The corridor would be generated according to a single line object, which would make branching as simple as extruding and moving around the line object in Edit Mode.
- The corridor generator would automatically detect corners and dead-ends and then place the appropriate pre-modelled wall pieces accordingly.
- An extra goal: make doors that could be opened individually.

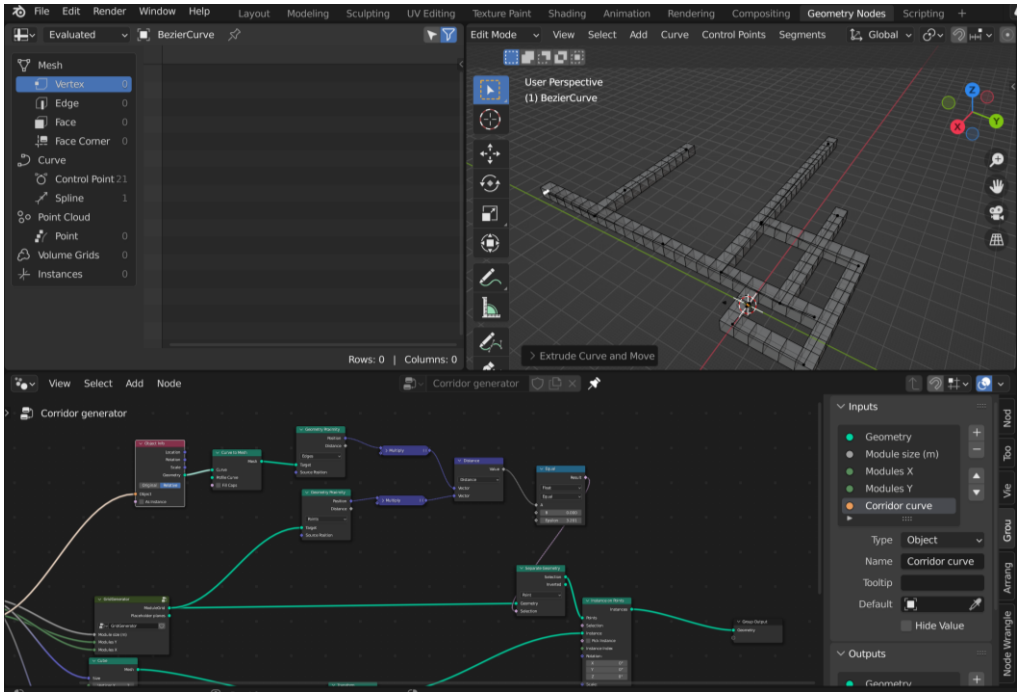
Some of the techniques used in the corridor generator were learned from YouTube user Kammerbild's tutorial on creating procedural buildings with Geometry Nodes (Kammerbild 2022). The main takeaway from this tutorial was the corner detection done with normals and Booleans. We will come back to that later.

The development started with a 2D grid of squares where each module would be placed. This was done with a Grid mesh primitive, which is a single plane with editable X and Y divisions and sizes. For later editing, these division and size values were connected to the Group Input node (Picture 37). This bit required a surprising amount of math nodes to get the grid centered and each division to be square-shaped.



PICTURE 37. Starting the corridor generator with the grid. Parameters added to the modifier stack to edit the grid dimensions and size of each module.

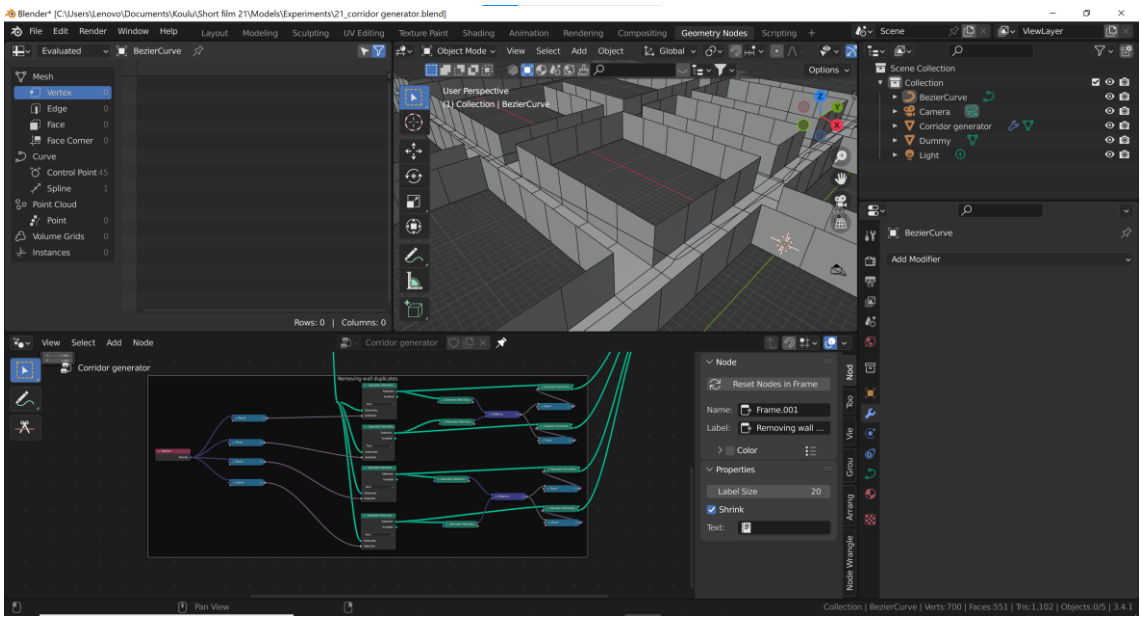
Next it was time to create and bring in the curve object that affects the module placements (Picture 38). The selection of grid squares was done by comparing the positions of each grid square to the curve object's edges in the X and Y axes. This is done with the Geometry Proximity nodes and the Distance operation in the Vector Math node. The Multiply nodes multiply the Z axes by 0, so the curve's Z-position doesn't affect the module placements.



PICTURE 38. Curve object added for editing the corridors. The cube objects are placeholders for the final floor and wall pieces.

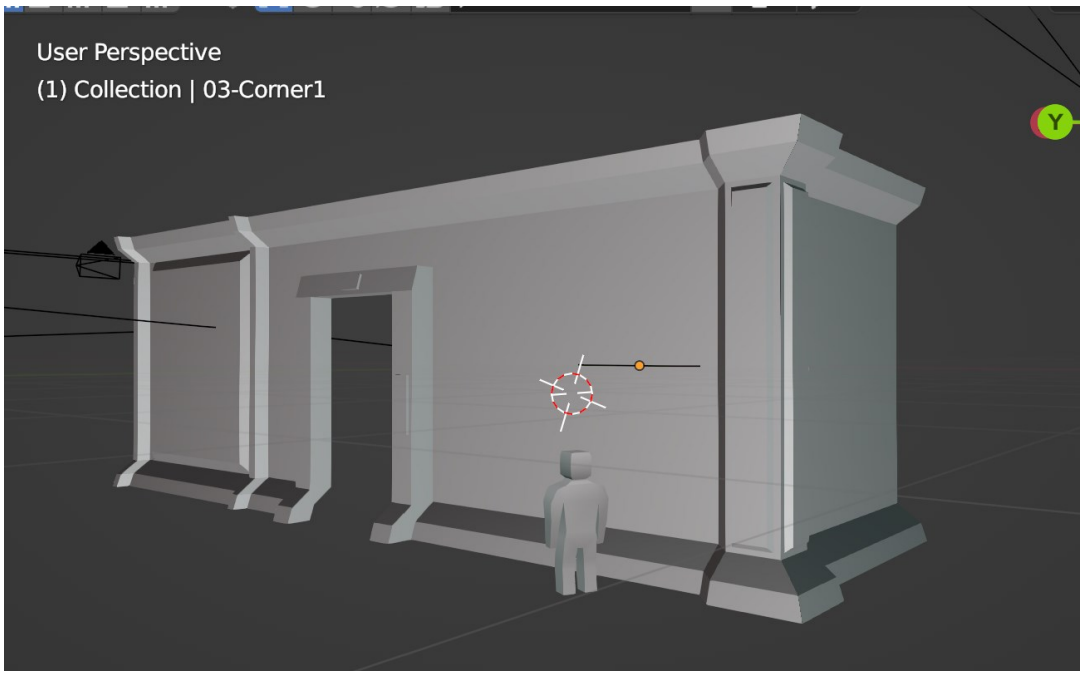
The selected squares are then used as positions for cubes that work as placeholders for the wall and floor pieces. Since the cubes were instances, I needed to change them back to regular shapes with the Realize Instances node. Now each square in the grid had a cube which overlaps with its neighbouring cube(s). I could then remove each overlapping cube wall to create corridors (Picture 39).

This is also the point where I separated each wall by their normal values. This was done with four Separate Geometry nodes. The Normal values were compared with the Equal node, which returns Boolean values that can be used for the Selection input slot of any node. In this case, the geometry was separated with the Separate Geometry node (Picture 39).

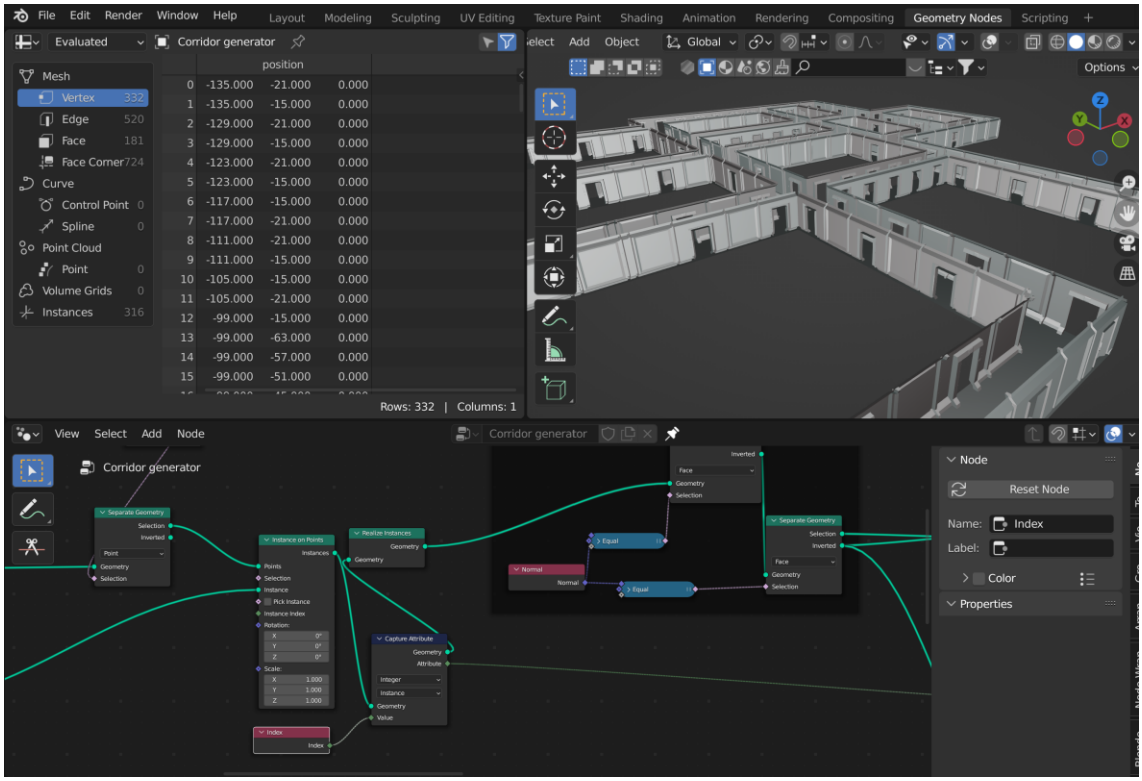



Picture 39. Overlapping walls are removed to create corridors. Walls are separated by their normal values, i.e. Their orientation.

At this point the wall pieces were modelled manually. This was done so that the wall orientation and the corner selections could be tested. The pieces included a basic wall, a wall with a door and a corner piece (Picture 40). Since the walls had already been separated by their orientation, putting the wall pieces into the right places was simple. However, I couldn't figure out how to place a door on every other wall piece in a way that makes sense (Picture 41). This was to make the rooms behind the doors seems bigger, but in the end, it was decided to place a door on every wall module that wasn't a corner.



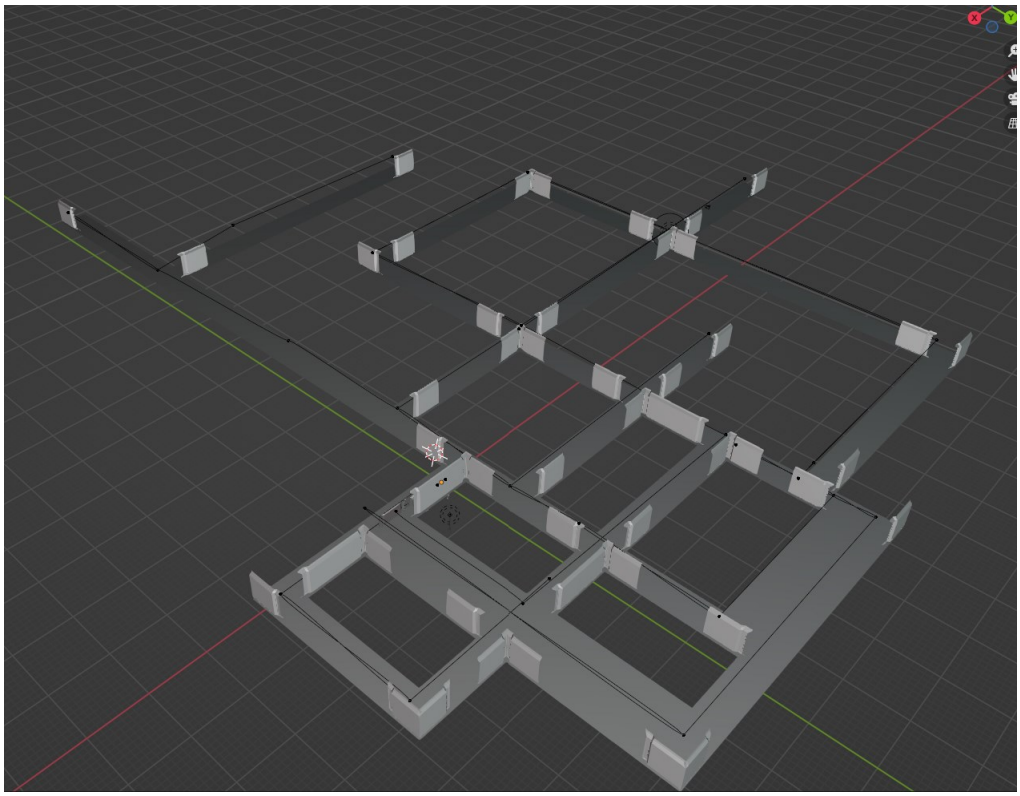

PICTURE 40. Modelling the initial wall modules.




PICTURE 41. Walls pieces placed to the walls that were separated earlier.

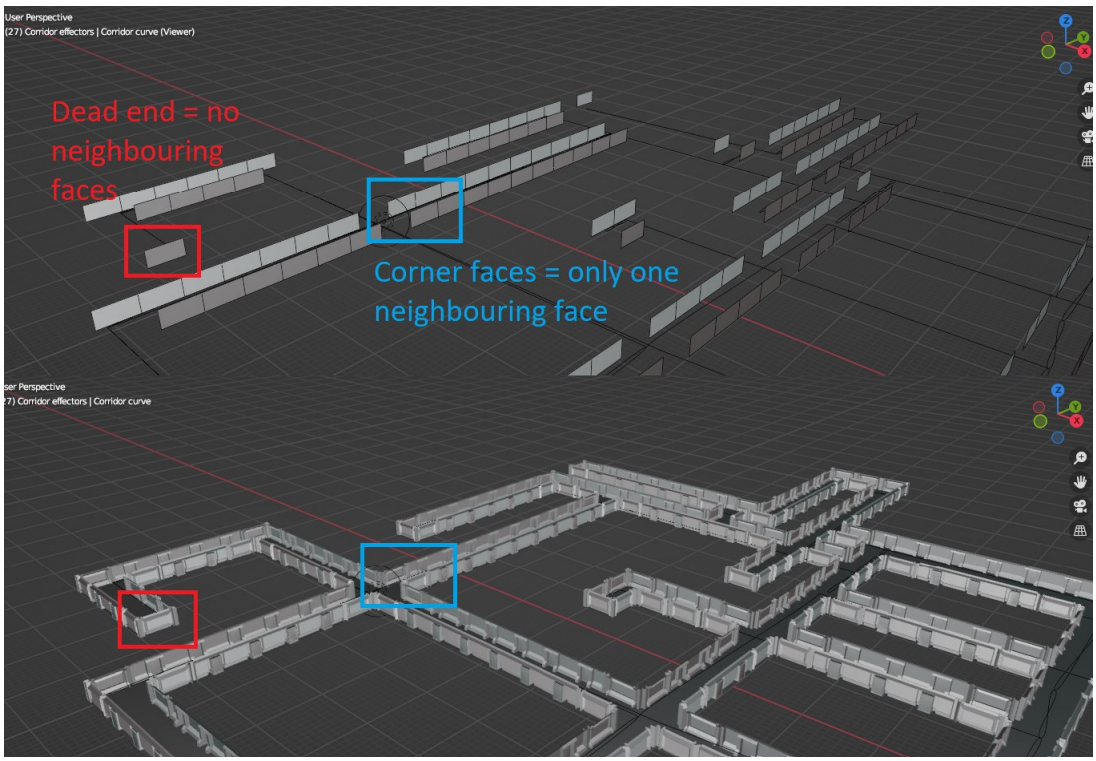
Next it was time to separate the corners so that the corner pieces could be placed on them. Since walls were already separated by their orientation, they could now be used for corner detection. Initially, the idea was to compare the corner pieces by distance: for an example, a wall piece facing the Y axis touching a wall piece

facing the X axis would indicate that the corridor takes a turn and requires a corner piece. However, this method didn't deliver reliable results and would have required a more complicated node setup to work. The results of this method can be seen in Picture 42.

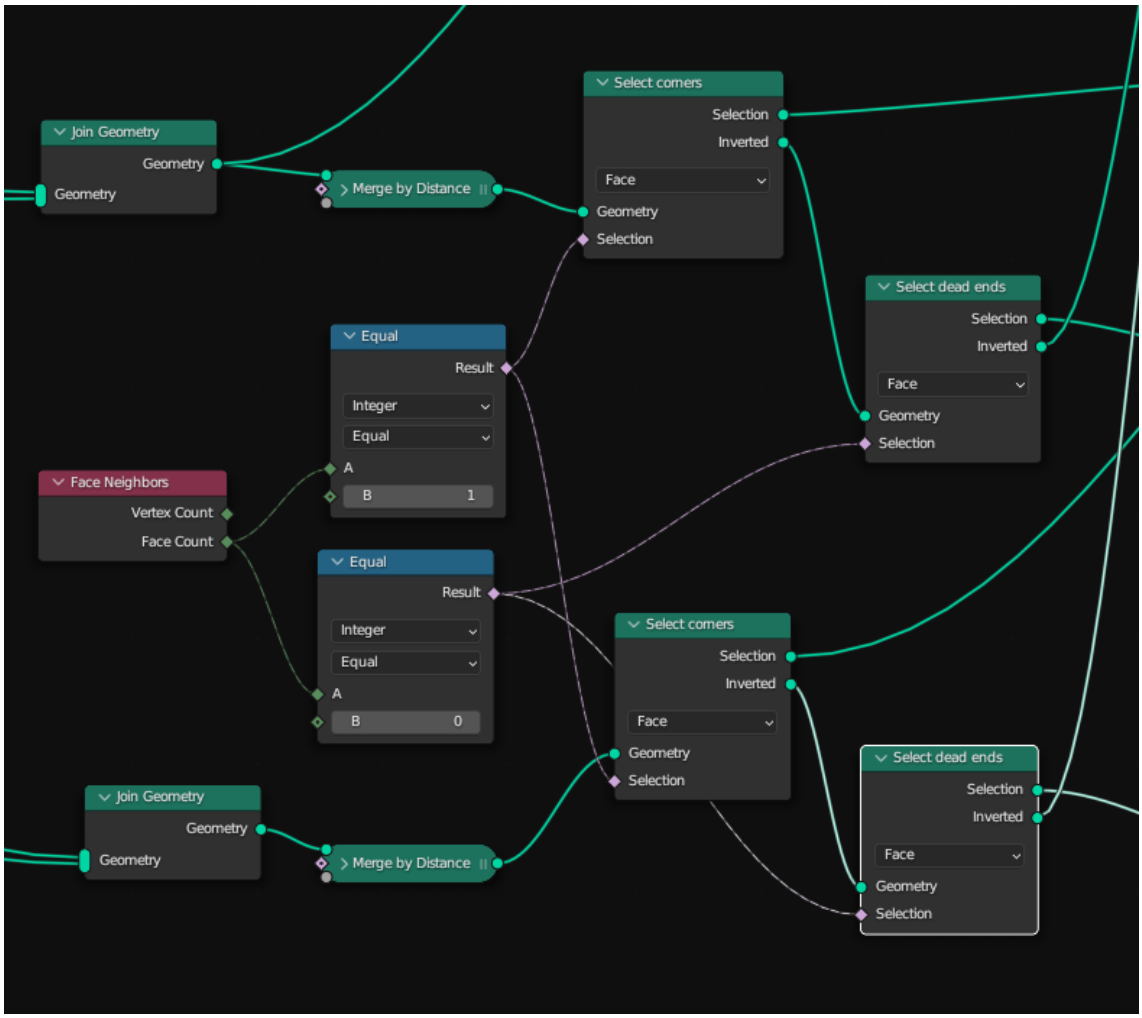


PICTURE 42. Results of the first attempt at corner detection, done by comparing walls pieces at turning points.

A more reliable method was to compare parallel walls with the Face Neighbours node, which returns the number of faces that the selected plane is sharing edges with. By isolating parallel walls, corners and dead-ends could be easily detected by their neighbouring faces: corner pieces would always have just one neighbouring face, whereas dead-ends would have no neighbours (Picture 43). The parallel walls were isolated together by joining the X and X- facing walls and the Y and Y- facing walls, and the selection was done with the Equal nodes and Separate Geometry nodes (Picture 44).

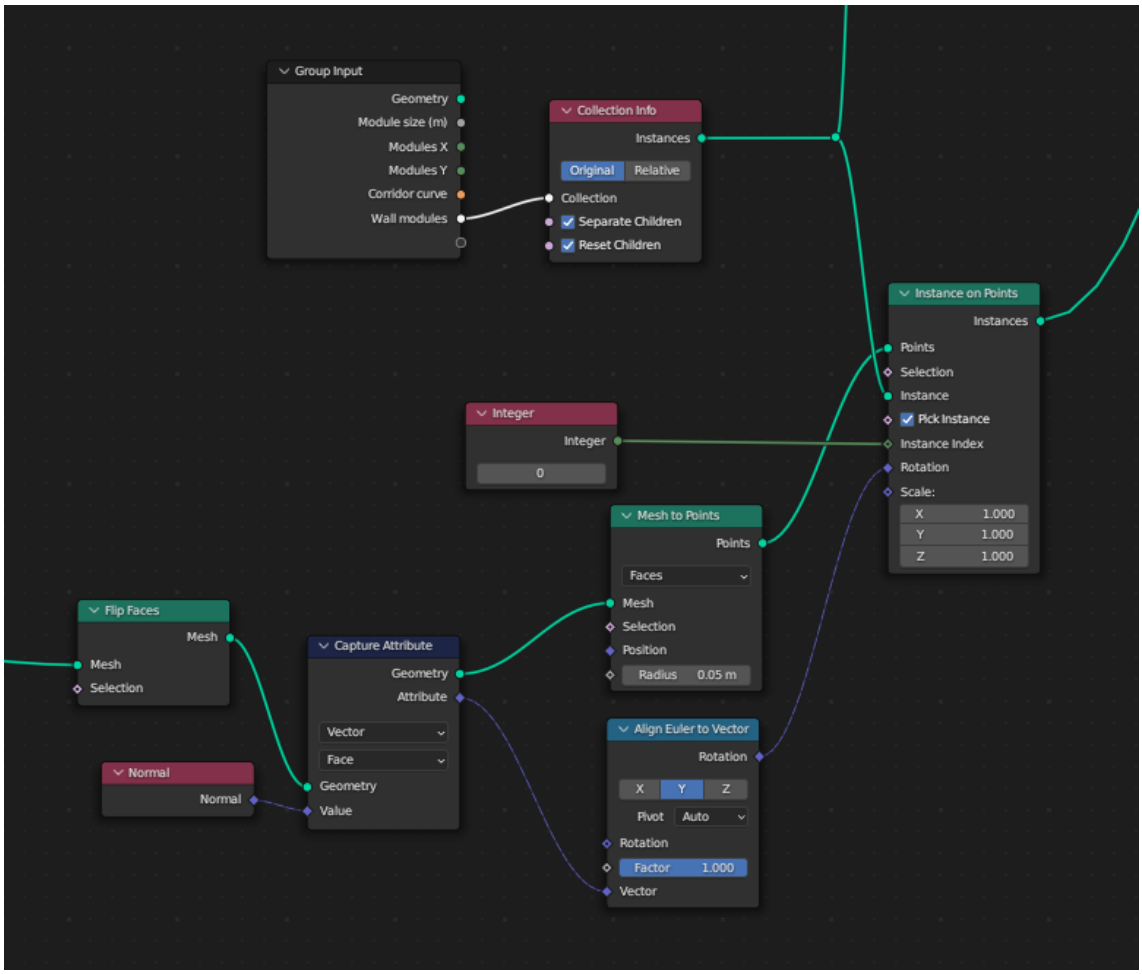



Picture 43. Comparing parallel walls to detect corners and dead ends.



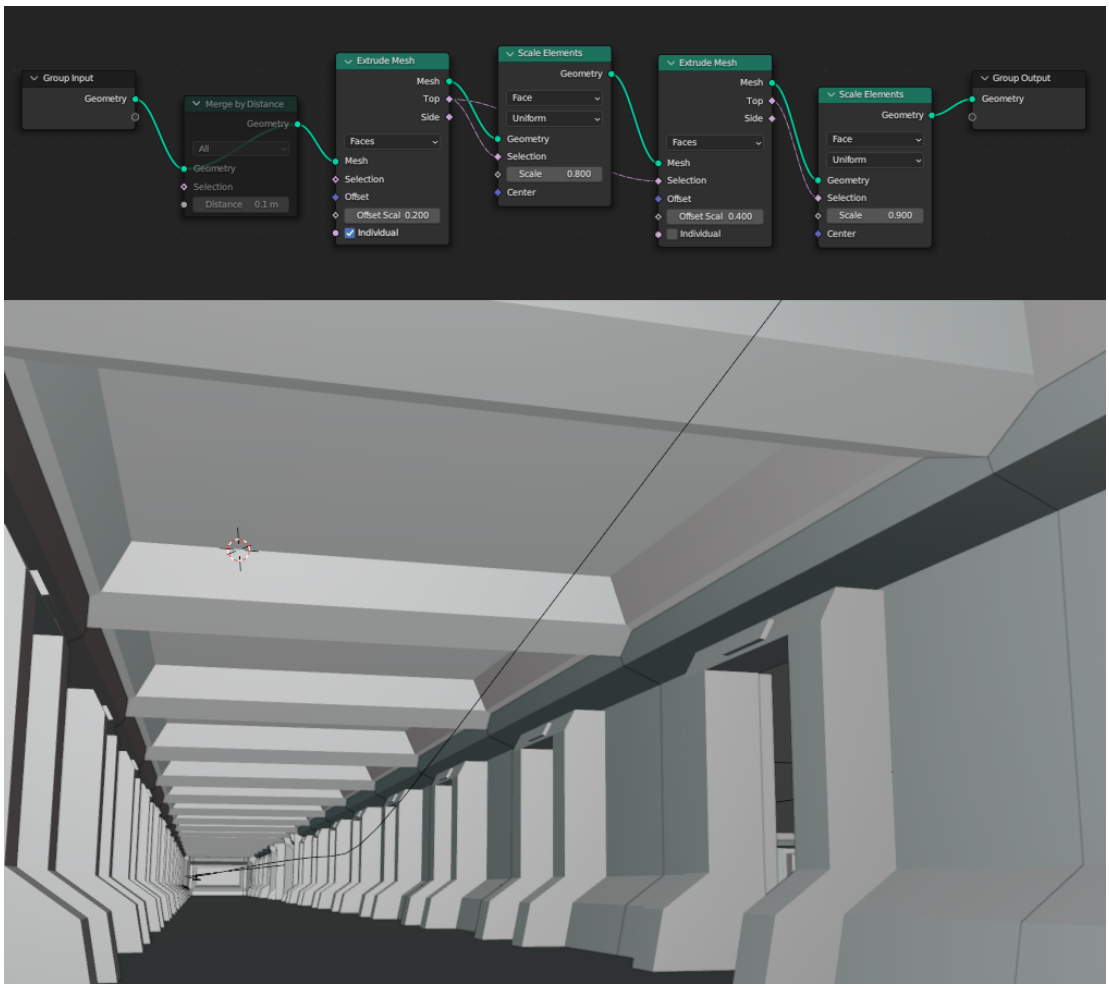

Picture 44. Node setup for corner and dead-end selection.

Once all the different types of wall faces were separated, the last thing left to do was to place the appropriate modules to their places by instancing. It was decided to have the wall modules be placed in a collection, where they would be selected by their index (Picture 45). A more user-friendly solution to implement in the future would probably be having the possibility to select modules individually in the modifier menu, but for the purposes of this short film production, this method is enough.



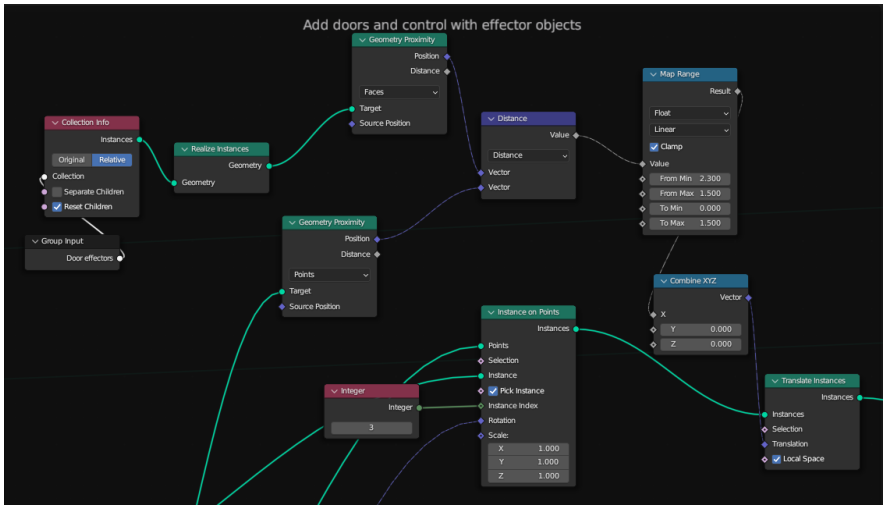

Picture 45. Instancing node setup for placing dead-end wall modules.

The ceiling details were done by separating the ceiling faces from the initial cubes and then adding two Extrude Mesh nodes (Picture 46). The ceiling could also be done by instancing a hand-made ceiling module, but this method was a quick way to add detail.

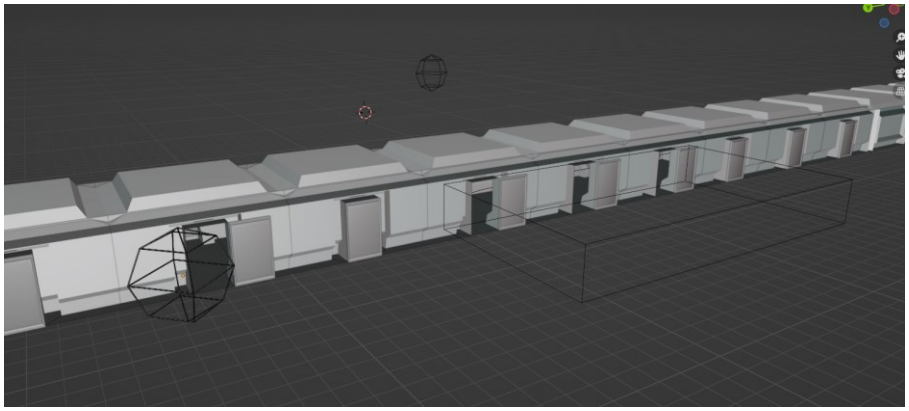



Picture 46. Node setup and results for ceiling details.

As an additional step, doors were added as separate instances, so that they could be opened and animated. This was done by offsetting the door instances by their proximity to a collection of effector objects (Picture 47). This way, individual doors can now be opened if needed (Picture 48).

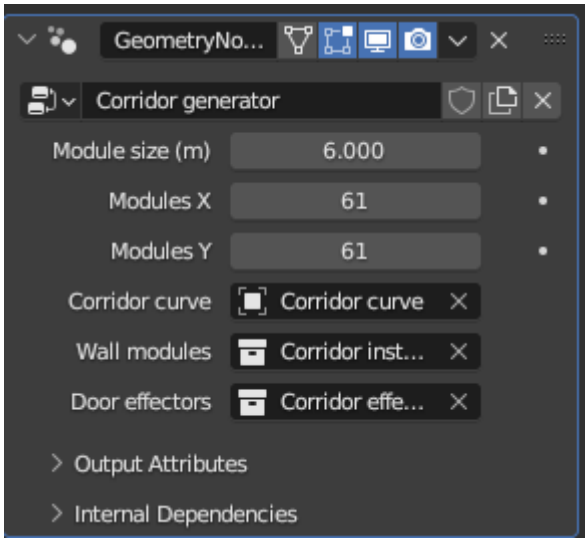



Picture 47. Node setup for door instancing and effector objects.



Picture 48. Doors opened with effector objects.

As the final step, parameters were added to the modifier view. Compared to the creature generator in chapter 5.1.1, the selection of controllable parameters is simple, as it only includes the module size, grid divisions, the curve object as well as collections for the wall modules and door effectors (Picture 49).




Picture 49. Parameters in the modifier window.

### 5.3 Evaluating the results and the methods

In case of the main character rig, it was quickly clear that the methods were too complicated for the time constraint of the thesis project. The Curve to Mesh node for the body generation seemed simpler and more flexible than using Modifiers, but the attempt resulted in complicated rigging solutions that weren't intuitive to use. The initial leg approach proved to be too difficult to adapt to the needs of the

film, but the new creature generator approach shows a new possible approach to the main character rig. However, since the end result is to be a single use character model, it is still most likely a wiser option to model the character manually.

The biggest challenges in both projects had to do with selecting the desired points for instancing, since selections had to be done either through their index or through proximity. This is obviously far less intuitive than in manual editing, where these selections are done by with the mouse. In this sense, the ability to access vertex groups in the node tree proved to be a useful feature, as they can be assigned manually.

Another common challenge was accessing previous data after converting geometry into another type. These problems were solved with the Capture Attribute node, but this complicates the node tree and is not a very intuitive process.

These challenges comply with the initial consideration that achieving pre-determined results can sometimes be challenging with procedural methods. Therefore, it is good to consider a hybrid solution between procedural and manual methods as needed.

In the end, both assets involved complicated geometries with repetition and slight variations, which meant that the procedural approach was justified. Modelling the more complicated pieces manually saved time in node tree programming. Both projects resulted in assets that can changed drastically by simple changes to either the parameters or the input objects. Therefore, they are taking full advantage of the procedural modelling method.

Another consideration is the further development of these node trees. The creature generator is designed for a very specific type of creature and purpose, which means it is not easy to adapt to other projects. The corridor generator, however, could easily be adapted to other projects, as it is entirely based on instanced modules. Instead of sci-fi themed walls, these modules could be walls for a cave or a hospital corridor, for an example. A 3D voxel grid could be considered as well to allow multi-level corridors. More different types of wall and ceiling pieces could be developed as well, to add variance to the result. This would require more

fundamental changes to the node tree, but these are possibilities for future development.

# 6 DISCUSSION

Judging from the history and example use-cases explored in chapter 2, the power of procedural modelling lies in its ability to generate detailed geometry with less man-hours than is required when working with manual methods. This comes with considerations, mainly to do with the learning curve, the perceptual differentiation in the resulting assets and loss of control depending on how much randomization is used.

The practical case study of this thesis resulted in assets that can generate multiple copies and variations with parametric control. This means that complicated environment models can be created and changed more quickly than with manual methods. In this sense, this thesis concludes that Geometry Nodes successfully brings in some key benefits of procedural modelling methods, with more flexibility than modifiers and more accessibility than written programming methods because of its VPL interface.

The assets modelled for this project had specific needs and mostly included curve-based geometry generation and automated instancing of pre-made modules. There are plenty of other uses that could be explored for this film and other projects, such as complex VFX animations and terrain generation. Simulations, hair-grooming tools and volumetric modelling are also features that are possible in Geometry Nodes but were not explored in this thesis.

One thing affecting my ability to learn Geometry Nodes is having experience with node-based procedural methods with the Shader Editor and the software TouchDesigner. Artists without this experience will most likely find the tools more difficult. Even with the author’s previous visual programming knowledge, less intuitive parts of the procedural methods like selecting specific geometries and accessing data after geometry type conversions proved challenging in the case-study. This could be considered an unavoidable obstacle, though in the case-study, selections were sometimes made easier with input objects that could be edited manually. So, combining manual and procedural methods could be a recommendable way of starting with procedural modelling.

At the time of writing this discussion section (October 2023), a new open movie project called Project Gold has been announced, and development of Geometry Nodes continues alongside this project. A development blog post from June 2023 states that the focus is on developing simulation nodes, a new node type called gizmo nodes and the possibility of real-time interaction. One of the mentioned goals is to allow VFX tools to be more usable in earlier layout stages of the movie production. The priorities are set to be shifting, so how these features are implemented in the end is still open. (Felinto 2023)

For procedural modelling in general, a subject which will probably gain relevance is the role of AI and machine learning for 3D model creation. Stability for Blender is a tool that already allows AI prompts to be used for texturing models in Blender (Fingas 2023). An article from June 2023 describes a method of using machine learning to help in procedural generation of a historical city: the machine learning model allowed for the generated model to adapt to incomplete or changing data (Vaienti 2023). Interestingly, the Blender creator Cartesian Caramel, also mentioned earlier in this thesis, has managed to write a machine learning algorithm inside Geometry Nodes (McKenzie 2023). Still, it remains to be seen if such methods would ever be brought into the toolset of Geometry Nodes itself.

**REFERENCES**

Annihilation. 2018. Direction: Alex Garland. Production: Skydance, DNA Films, Scott Rudin Productions & Huahua Media. Production countries: United Kingdom & United States.

Aron, J. 2009. The Mandelbulb: first 'true' 3D image of famous fractal. New Scientist 18.11.2009. Read on 12.9.2023.
https://www.newscientist.com/article/dn18171-the-mandelbulb-first-true-3d-image-of-famous-fractal/

Blender Documentation Team. 2022. Geometry Nodes: Introduction. Blender 3.5 Manual. Read on 5.12.2022.
https://docs.blender.org/manual/en/dev/modeling/geometry_nodes/introduction.html

Blender Documentation Team. 2023. Modifiers: Introduction. Blender 3.6 Manual. Read on 16.9.2023.
https://docs.blender.org/manual/en/latest/modeling/modifiers/introduction.html

Cartesian Caramel. 2022. How to make the Mini Mimic Effect in Blender 3.2 (Livestream Tutorial). Tutorial. Released on 14.8.2022.
https://www.youtube.com/live/dtmostVVuv8?si=_Qc_aCmj40a5jXoO

Compton, K. 2016. So you want to build a generator. Blogpost. Released on 22.2.2016. Read on 12.9.2023.
https://galaxykate0.tumblr.com/post/139774965871/so-you-want-to-build-a-generator

de Vries, J. N.d. Advanced OpenGL: Instancing. LearnOpenGL. Read on 12.9.2023.
https://learnopengl.com/Advanced-OpenGL/Instancing

Ebert, D. S., Musgrave, F. K., Peachey, D., Perlin, D. & Worley, S. 2003. Texturing & Modeling: A Procedural Approach. 3rd edition. USA: Morgan Kaufman Publishers.

Esri. 2022. ArcGIS CityEngine: Overview. CityEngine Documentation. Read on 18.4.2023.
https://doc.arcgis.com/en/cityengine/latest/get-started/get-started-about-cityengine.htm

Failes, I. 2018. Mandelbulbs, mutations and motion capture: the visual effects of Annihilation. Blogpost. Released on 12.3.2018. Read on 12.9.2023.
https://vfxblog.com/2018/03/12/mandelbulbs-mutations-and-motion-capture-the-visual-effects-of-annihilation/

Felinto, D. 2020. Nodes Modifier Part I: Sprites. Blender Developer Blog. Released on 9.11.2020. Read on 12.9.2023.
https://code.blender.org/2020/11/nodes-modifier-part-i-sprites/

Felinto, D., Goudey, H. & Lucke, J. 2022. Geometry Nodes Development Process. Presentation. Blender Conference on 27–29.10.2022. Felix Meritis. Amsterdam.
https://www.youtube.com/watch?v=nmGs9fEPcrA&list=PLa1F2ddGya__2Ip2UB-xp6M54cQVvw1W-&t=2s

Felinto, D. 2022. The Future of Hair Grooming. Blender Developer Blog. Released on 8.7.2022. Read on 12.9.2023.
https://code.blender.org/2022/07/the-future-of-hair-grooming/

Felinto, D. 2023. Node Tools, Interface and Baking. Blender Developer Blog. Released on 1.6.2023. Read on 9.10.2023.
https://code.blender.org/2023/06/node-tools-interface-and-baking/

Fingas, J. 2023. Blender can now use AI to create images and effects from text descriptions. Article. Engadget. Released on 2.3.2023. Read on 7.10.2023.

https://www.engadget.com/blender-can-now-use-ai-to-create-images-and-effects-from-text-descriptions-175001548.html?guccounter=1

Foundry. N.d. Direct vs. Procedural Modeling. Modo Essentials Documentation. Read on 23.7.2023.
https://learn.foundry.com/modo/essentials/content/essentials/modeling/direct-procedural.html

Higgins, C. 2014. No Man's Sky would take 5 billion years to explore. Wired. Article. Released on 18.8.2014. Read on 8.8.2023.
https://www.wired.co.uk/article/no-mans-sky-planets

Hwang, R. 2017. Procedural Building Generation with Grammars. Lecture. University Of Pennsylvania.
https://youtu.be/gjAmntyk8Pw?si=_YzOP44UzzAHAS7Z

Jarrat, B. & Tarolli, D. 2017. Creating the city for Disney's Zootopia. Presentation. VES webinar.
https://www.youtube.com/watch?v=Dflh2EmMyEQ

Kammerbild. 2022. Blender: procedural buildings with geometry nodes fields | pt. 1. Released on 1.2.2022.
https://www.youtube.com/watch?v=59PeIGmZQdY

Kollar, P. 2016. No Man's Sky Review. Polygon. Game review. Released on 12.8.2016. Read on 12.9.2023.
https://www.polygon.com/2016/8/12/12461520/no-mans-sky-review-ps4-playstation-4-pc-windows-hello-games-sony

Kuhail, M. A., Farooq, S., Hammad, R. & Bahja, M. 2021. Characterizing Visual Programming Approaches for End-User Developers: A Systematic Review. IEEE Access, vol. 9, 14181-14202. Released on 12.1.2021. DOI 10.1109/ACCESS.2021.3051043.
https://ieeexplore.ieee.org/document/9320477

Lucke, J. N.d. Animation Nodes. Animation Nodes Documentation. Read on 12.9.2023.
https://docs.animation-nodes.com/

McKenzie, T. 2023. A Machine Learning Algorithm Created With Blender's Geometry Nodes. Article. 80 Level. Released on 23.1.2023. Read on 7.10.2023.
https://80.lv/articles/a-machine-learning-algorithm-created-with-blender-s-geometry-nodes/

Martin, J. 2021. Modeling with Animation in Mind. Topology Guides. Released on 27.12.2021. Read on 16.9.2023.
https://topologyguides.com/modeling-for-animation

Merkle, M. C. & Schwind, M. 2018. All Your Clones Are Belong To Us! Surviving The Switch To Houdini. Presentation. FMX 2018 – Conference on Animation, Effects, Games and Immersive Media on 24–27.4.2018. Raum Mannheim. Stuttgart.
https://vimeo.com/267864823

Paone, J. n.d. Tessellation Chapter I: Rendering Terrain using Height Maps. LearnOpenGL. Read on 14.10.2023.
https://learnopengl.com/Guest-Articles/2021/Tessellation/Height-map

Price, A. 2018. The Next Leap: How AI Will Change The 3D Industry. Presentation. Blender Conference on 25-27.10.2018. De Balie. Amsterdam.
https://www.youtube.com/watch?v=FlgLxSLsYWQ

Reynish, W. 2019. Everything Nodes UX. Blender Development Documentation. Released on 17.7.2023. Read on 15.9.2023.
https://projects.blender.org/blender/blender/issues/67088

Saldaña, M. An Integrated Approach to the Procedural Modeling of Ancient Cities and Buildings. Digital Scholarship in the Humanities, Vol. 30, Supplement 1, 2015. Read on 15.9.2023.
https://academic.oup.com/dsh/article/30/suppl_1/i148/361687

Schinko, C., Krispel, U., Ullrich, T., & Fellner D. 2015. Built by Algorithms – State of the Art Report on Procedural Modeling. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume XL-5/W4, 469-479, February 2015. Read on 16.9.2023. https://doi.org/10.5194/isprsarchives-XL-5-W4-469-2015

Seymour, M. 2012. Side Effects Software – 25 years on. fxGuide. Article. Read on 14.10.2023. https://www.fxguide.com/fxfeatured/side-effects-software-25-years-on/

Seymour, M. 2013. Pixar's RenderMan Turns 25. fxGuide. Article. Released on 25.6.2013. Read on 16.9.2023. https://www.fxguide.com/fxfeatured/pixars-renderman-turns-25

Sprite Fright. 2021. Direction: Matthew Luhn & Hjalti Hjalmarsson. Blender Studio.

Thommes, S. 2021. Denoising Sprite Fright. Blender Studio. Blogpost. Released on 26.11.2021. Read on 12.9.2023. https://studio.blender.org/blog/denoising-sprite-fright/

Ullrich, T., Schinko, C. & Fellner, D. 2010. Procedural modeling in theory and practice. Fraunhofer IGD. Read on 8.4.2023. https://www.researchgate.net/publication/45539668_Procedural_modeling_in_theory_and_practice

Vaienti, B., Petitpierre, R., di Lenardo, I. & Kaplan, F. 2023. Machine-Learning-Enhanced Procedural Modeling for 4D Historical Cities Reconstruction. Remote Sensing, 2023, 15, 3352. https://doi.org/10.3390/rs15133352

Vol Libre. 1980. Direction: Loren Carpenter. Production: DICOMED Corporation. Production country: United States.

Zeman, N. B. 2014. Essential Skills for 3D Modeling, Rendering, and Animation. 1st edition. New York: A K Peters/CRC Press.

Zootopia. 2016. Direction: Byron Howard & Rich Moore. Production: Walt Disney Pictures, Walt Disney Animation Studios. Production country: United States.

# APPENDICES

Appendix 1. Pre-production document

Behind the following link is a presentation showcasing the pre-production done before the writing of this thesis. It includes images of concept art, as well as links to a written story concept and a complete list of assets required for the film.
https://docs.google.com/presentation/d/1oERGI-ERGvh2TS_saG4lelBJqoF2iZoZn2hE7gNkE-g/edit?usp=sharing

Behind the following link you will find a video of some initial tests done with the first version of the main character rig. The goal on this version was to scale the character's body accordingly when the main bones or controllers are being moved away from each other.
https://drive.google.com/file/d/1CCHPmBV5zEbf08_TTH_t9CCeIa5TyRtB/view?usp=share_link