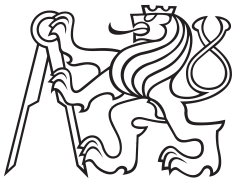

Czech Technical University in Prague

F3

Faculty of Electrical Engineering

Procedural generation of future urban construction according to rules

Vladimíra Potočeková

Supervisor: Ing. David Sedláček, Ph.D.
May 2024

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Potočeková Vladimíra** Personal ID number: **499032**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Graphics and Interaction**

Study program: **Open Informatics**

Specialisation: **Computer Games and Graphics**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Procedural generation of future urban construction according to rules

Bachelor’s thesis title in Czech:

Procedurální generování budoucí výstavby měst dle pravidel

Guidelines:

Parts of the city are gradually changing from one form of use to another. After the approval of this change, it is necessary to communicate with the inhabitants how this change will manifest itself (handled by the Institute of Planning and Development of Prague, abbreviated IPR). For example, the conversion of an area from a brownfield to a residential area will result in significant changes in the area, but this change must be communicated before projects and tenders for the implementation of the change are awarded. Therefore, it is appropriate to generate a possible future state and variations for this future state to promote civic awareness of this change.
1) Familiarize yourself with the literature as well as tools suitable for procedural generation of geometry of city parts, e.g. CityEngine, Houdiny, Blender.
2) Familiarize yourself with and describe in your thesis the map basis of Prague [1] (structure, available data formats) suitable for your work. And also the input/output data processes of IPR.
3) Familiarize yourself with the urban planning rules for new developments (Metropolitan Plan) [2].
4) In agreement with your supervisor, select an appropriate subset of the rules and implement them in the tool of your choice. An important element is realistic site development and building mass, including other reasonable details.
5) Create previews (renderings) comparing the original and future construction. Demonstrate the range of variability in the selected areas (at least two areas selected in cooperation with the supervisor). Compare with similar, previously produced visualizations on the IPR.
6) Communicate with the person in charge of IPR during the course of the work and during the assessment point (5).

Bibliography / sources:

1] Open data Prague, https://opendata.praha.eu/
2] Prague Metropolitan Plan (text part and maps), https://plan.praha.eu/
3] Ondřej Kyzr, Procedural generation of outdoor scenes. BP ČVUT FEL, 2023.
4] Jana Kejvalová, Procedural model generation from real maps. DP ČVUT FEL, 2019.
5] Alena Mikushina, Creation of modular 3D assets for videogames. BP ČVUT FEL, 2020.
6] Jan Kutálek, Procedural generation of videogame environments. BP ČVUT FEL, 2021.
7] Jiří Zemko, Procedural generation of city models. BP ČVUT FEL, 2022.

Name and workplace of bachelor's thesis supervisor:

Ing. David Sedláček, Ph.D. Department of Computer Graphics and Interaction FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **16.02.2024** Deadline for bachelor thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

Ing. David Sedláček, Ph.D. Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

I would like to express my deepest gratitude to Ing. David Sedláček, Ph.D., my thesis supervisor, for his invaluable guidance, support, and encouragement throughout the entire research process. His expertise, insightful feedback, and dedication were instrumental in shaping this thesis.

Special thanks to the Prague Institute of Planning and Development (IPR Prague) for providing the necessary data for this research project.

I am deeply grateful to my family and friends for their unwavering support and understanding during this academic journey.

In particular, I would like to extend my heartfelt thanks to my boyfriend for his continuous love, encouragement, and understanding, which have been a constant source of motivation and strength.

And finally, I extend a heartfelt thanks to my lovely dog, although she can not read, she is a crucial part of my happiness and thus deserves to be mentioned. Her playful spirit and unconditional love provided solace in times of stress and inspiration to persevere. To Bria, my cherished friend and faithful confidante, thank you for being by my side every step of the way.

Declaration

I declare that this work is all my own work and I have cited all sources I have used in the bibliography.

Prague, May 20, 2024

Prohlašuji, že jsem předloženou práci vypracovala samostatně, a že jsem uvedla veškerou použitou literaturu.

V Praze, 20. května 2024

Abstract

This thesis aims to explore the integration of procedural generation into the area of urban development. The generator will propose a model of the possible construction of the land based on the user's requirements such as minimum distance, street width, floor height, maximum amount of floors, minimum density or the level of detail. The generator also provides information about the building area of the model, since it is an important marker for urban studies. The procedural generation will be done for the Prague Institute of Planning and Development (IPR Prague), which have provided data for this project. Blender was chosen as the procedural generation software.

Keywords: procedural generation, degree project

Supervisor: Ing. David Sedláček, Ph.D.

Abstrakt

Tato práce si klade za cíl prozkoumat integraci procedurální generace do oblasti urbanistického rozvoje. Generátor navrhne model možné zástavby pozemku na základě požadavků uživatele, jako jsou minimální vzdálenost, šířka ulice, výška podlaží, maximální počet podlaží, minimální hustota nebo úroveň detailů. Generátor také poskytuje informaci o zastavěné ploše modelu, jelikož je to důležitý ukazatel pro urbanistické studie. Procedurální generace bude provedena pro Institut plánování a rozvoje hl. m. Prahy (IPR Praha), který poskytl data pro tento projekt. Jako software pro procedurální generaci byl zvolen Blender.

Klíčová slova: procedurální generování, závěrečná práce

Překlad názvu: Procedurální generování budoucí výstavby měst dle pravidel

Contents

1 Introduction 1

2 Analysis 3
2.1 Procedural generation 3
2.2 Procedural generation software . . 4
2.2.1 Houdini 4
2.2.2 Blender 4
2.2.3 Comparison 5
2.3 Existing city generators 5
2.3.1 ArcGIS CityEngine 5
2.3.2 Citygen 5
2.3.3 Comparison 6
2.4 Input data 6
2.5 Output data 6

3 Design proposal 7
3.1 Requirements 7
3.2 Urban planning rules for new developments 7
3.2.1 The Building Block Development Coefficient 7
3.2.2 Widths of street open public spaces 8
3.2.3 Height regulation 8
3.2.4 Heights and areas of rooms . . . 9
3.3 Approach to the procedural generation 10
3.3.1 Addressing the requirements . 10
3.3.2 Structure of the program 10
3.3.3 Procedural generation in Blender 10
3.3.4 Native Blender nodes 11
3.3.5 Custom Nodes 16

4 Implementation 19
4.1 Importing data 19
4.2 Group Input Parameters 19
4.2.1 Street Width 19
4.2.2 Median 20
4.2.3 Percentage 20
4.2.4 LOD 21
4.2.5 Floor Height 21
4.2.6 Distance Min 22
4.2.7 Max Floors 22
4.2.8 Min Density 23
4.3 Additional Group Input Parameters 23
4.3.1 Rows 24
4.3.2 Columns 24
4.4 Program 25
4.4.1 Creating the grid 25
4.4.2 Creating road curves 27
4.4.3 Cutting out roads 28
4.4.4 Instancing bases of houses . . . 29
4.4.5 Extruding the floors 32
4.4.6 Instancing the windows and doors 33
4.4.7 Creating sidewalks 34
4.4.8 Instancing the trees 34
4.4.9 Creating roofs for the houses 35
4.4.10 Calculating building area . . 36
4.5 Shading and Materials 37
4.6 Levels of detail 37
4.7 Custom Roads 39
4.8 Automation of the parameter input 40
4.9 Validity of the results 42

5 Results 43
5.1 Closer Renders 44
5.2 Feedback 48
5.2.1 Adjusting the program according to the feedback 48
5.3 Possible Future Improvements . . 49
5.3.1 Statistic information 49
5.3.2 Adding styles for buildings . . 49

6 Conclusion 51

A Bibliography 53

Figures

3.1 Cirle (left), Diamond (middle) and Diamond with a dot (right) 11

4.1 Street Width set to 5m (left) and 10m (right) 19
4.2 Median x value set to -5170.66 (left) and -4770.66 (right) 20
4.3 Percentage set to true (left) and false (right) 20
4.4 LOD set to 0 (left), 1 (middle) and 2 (right) 21
4.5 Floor Height set to 2,6 (left) and 3,5 (right) 21
4.6 Distance Min set to 25 (left) and 35 (right) 22
4.7 Max Floors set to 6 (left) and 3 (right) 22
4.8 Min Density set to 0.8 (left) and 0.3 (right) 23
4.9 Rows set to 8 (left) and 4 (right) 24
4.10 Columns set to 8 (left) and 4 (right) 24
4.11 Node tree for creating the grid 25
4.12 Different bitmap versions 30
4.13 Node tree for the shader of the wall material 37
4.14 Different levels of LOD 0 being the top and 2 the bottom 38
4.15 Highlighted relevant parts of viewport 39

5.1 Parcels 43
5.2 Detail 1 of Test Case 4 45
5.3 Detail 2 of Test Case 4 45
5.4 Detail of Test Case 8 46
5.5 Detail of Test Case 13 46
5.6 Detail of Test Case 14 47
5.7 Detail of Test Case 16 47

Tables

3.1 Values of ZB coefficient according to structure type 8

5.1 Legend 44
5.2 Parameters 44

Chapter 1

Introduction

This thesis aims to explore the integration of procedural generation into the area of urban development. Producing a visualisation of a potential development of a land can be tedious due to uncertainty of the design or undecided architecture plans. It is generally better advised to use several different versions of the visualisation, which when shown to the public, can aid in imagining how the area could potentially look after being developed. Procedural generation could prove to be beneficial in this aspect as it is capable of quickly producing different variations. It could also be able to be reused on different parcels, removing the need to individually produce a visualisation for each and every plot of land which is planned to be further developed. This can speed up the process of presenting development plans to the public significantly and allow for changes to be made more easily. The procedural generation will be done for the Prague Institute of Planning and Development (IPR Prague), which have provided data for this project [1] [2].

Chapter 2
Analysis

2.1 Procedural generation

Procedural generation is a technique used in animation, visual effects, game development [3] [4], and many other fields to create digital content algorithmically instead of manually designing it. A number of rules is introduced, which the produced result has to abide by, and using this, mathematical algorithms and a certain level of computer-generated randomness yields almost endless amount of diverse content such as levels, maps, characters, textures, and more. Procedural generation offers several advantages, including scalability and the ability to generate content quickly, making it a valuable tool in modern media and entertainment [5].

Procedural generation can be used to produce many different types of outputs whose structure abides by a set of certain rules, albeit chaotic ones. Classic use of procedural generation is to generate models of cities and buildings for quickly and easily modifiable and reusable environment [6] [7]. This can lead to a large number of entirely different looking environments being created in marginally less time once the rule set is established and due to their adjustable nature allows them to be reused quite effectively. Another use can be seen when generating the terrain of an environment, resulting in a collection of different shapes and forms without any significant effort [8]. Generation of foliage is another use case for procedural generation, as plants and greenery often assume similar shapes and forms, which can be easily replicated using a set of rules. Another use case for procedural generation can be seen with procedurally generated textures. Generating textures can significantly ease the texturing process of asset creation as one can easily and non-permanently alter the generated texture to produce a number of different variations e.g. the size or shape of a brick, the spacing between rows or the amount of dirt in a brick texture.

2.2 Procedural generation software

The industry standard software for computer generation and VFX has for several years been Houdini, however in the recent years Blender has advanced significantly when it comes to procedural generation. Both software use a node based workflow, which proves to be very effective and quick at creating and modifying an algorithm. Nodes are linked in a trees or node groups, which can be saved as tools and later reused in different projects.

2.2.1 Houdini

Houdini is a software used for producing captivating VFX, animations, models, and procedural effects generally using combinations of different node types to achieve the desired result, such as LOD, VOP, geometry, render, object nodes and many more. The software has multiple versions of licenses available for purchase as well as a free license called Apprentice. Houdini Apprentice is a free version of Houdini FX which can be used by students, artists and hobbyists to create personal non-commercial projects [9]. Houdini carries the reputation of being quite user unfriendly, due to it's highly specific user interface and general lack of resources for learning, however it proves to be a highly capable and powerful tool for procedural generation once mastered. The general Apprentice version is free and available to anyone, coming with the limitation of export formats and inability of reusing the node tree in other professional versions of the software.

2.2.2 Blender

Blender is a free and open source 3D software licensed as GNU General Public License (GPL). Blender maintains its vast community of users mostly due to its support of the entirety of the 3D pipeline—modeling, rigging, animation, simulation, rendering, compositing and motion tracking, even video editing and game creation [10]. It is also popular due to its non destructive approach to modelling using modifiers. One such modifier is used to procedurally generate geometry on an existing object. This modifier is called Geometry Nodes. As the name suggests it uses groupings of nodes to produce desired geometry. Input and output parameters can be defined to the user's liking which makes the modifier highly user friendly allowing to alter the output without specifically altering the node tree. Blender also comes with a Spreadsheet window which shows the data used in Geometry Nodes such as Mesh data (eg. vertex, edge and face information), Curve or Instance data. This function can make program debugging significantly easier. Due to Blender's large community an large amount of plugins is available for download, which allows the user to import and export geometry in a number of different formats. This can be beneficial, as it omits the need for pre-processing the input data.

2.2.3 Comparison

Both software have their own advantages. Houdini is a paid software with significant limitations on the free version. Products produced in the free version can not be used commercially as well as the project produced in the free version are not able to be used in any other paid version and it is needed to reproduce the entire project from scratch. Export format limitations are also in place, the Apprentice free version having only two available export formats, those being .obj and Houdini internal geometry file formats. Blender on the other hand is a free software with no limitations when it comes to export or commercial use. It is also constantly being updated with new functionalities being introduced commonly. After considering the limitations of both software, Blender was chosen as a better alternative, more precisely the 4.0 version of Blender was chosen for this project.

2.3 Existing city generators

2.3.1 ArcGIS CityEngine

ArcGIS CityEngine is an advanced 3D modeling software for creating massive, interactive, and immersive urban environments in less time than with traditional modeling techniques.[11] The cities created using ArcGIS CityEngine can be based on real-world geographic information system (GIS) data or showcase a fictional city of the past, present, or future. Key features of the program are:

- Making 3D models to show planned changes and alternate designs.
- Informing designs with 3D representations of regulatory and land-use conditions.
- Sharing multiple design alternatives with user's team or stakeholders to gather feedback.

2.3.2 Citygen

Citygen employs procedural techniques to generate cityscapes for use in games and other graphics applications. A key design goal of the system was that it would allow the user close control over the generation process by means of direct manipulation of generation algorithm parameters via an accessible and intuitive visual interface. The city generation problem is divided into three stages[12]:

1. Primary Road Generation
2. Secondary Road Generation
3. Building Generation

2.3.3 Comparison

While CityEngine is mostly regarded as a powerful tool, with a intuitive UI and powerful graphics, it is a paid subscription based tool offering a free 30 day trial. Citygen on the other hand is a free tool, but is severely limited and fairly unintuitive to use with a more traditional presentation. The procedural generation in this thesis aims to provide an alternative that combines the strengths of both these tools.

2.4 Input data

The Prague Institute of Planning and Development (IPR Prague), has provided data in the GML format. This data can be also downloaded from IPR open source data [2]. The data represents a plot of land which is to be developed. This data was then imported to the QGIS program and exported as a Shapefile. The expected output is possible visualisations of the development which can be shown to people residing in the area. This is to be achieved through procedural generation. The input was imported into Blender with the BlenderGIS add-on [13]. Data for the roads has to be imported as a separate object named "roads".

2.5 Output data

The output of this project is expected to be an OBJ file containing generated geometry. As Blender allows exporting to OBJ no further conversion is needed.

Chapter 3

Design proposal

3.1 Requirements

The program is to generate a number of simple visualisations of the parcel development as well as provide the user with information about the building area for that particular parcel. The program should provide reliable results for a variety of parcel shapes. Additionally, it is required for the program to allow the user to customise the output according to their needs.

3.2 Urban planning rules for new developments

The goal of this thesis is to create visualisations which are not too detailed and concrete. Therefore the focus will be put on general rules for development, such as spacing of buildings or height, rather than details such as access to the buildings for fire safety or noise reduction.

3.2.1 The Building Block Development Coefficient

Regulated area of a building (hereinafter the "RAB") is an area expressed as a rectangular projection of the perimeter structures of the aboveground floors of the building onto the horizontal plane, except for elements beyond the building line.

The building block development coefficient (ZB) establishes the maximum share of the sum of total RAB of all buildings and the area of the building block. For the purpose of defining the coefficient, blocks are classified as:

- small blocks possessing an area P_M not exceeding 2 000 m^2 incl.,
- mid-size blocks possessing an area P_S over 2 000 m^2 to 12 000 m^2 incl.,
- large blocks possessing an area P_V over 12 000 m^2.

Unless stipulated otherwise in BCS / 400, the values of the ZB coefficient are defined according to the structure type and size of the building block as follows in Table 3.1:

	ZB_M	ZB_{S1}	ZB_V	ZB_N
grown structure	95%	85%	60%	65%
block structure	85%	75%	50%	65%
hybrid structure	95%	85%	50%	65%
heterogeneous structure	65%	55%	35%	40%
village structure	30%	30%	20%	25%
garden city structure	35%	35%	20%	25%

Table 3.1: Values of ZB coefficient according to structure type

For small blocks, use the ZB_M values. For large blocks, use the ZB_V values. For mid-size blocks, the ZB_S value is derived by calculation from ZB_S1 and ZB_V , according to the following equation, where P S denotes the area of the mid-size block:

$ZB_S = ZB_V + (ZB_{S1} - ZB_V) * (12000 - P_S)/10000$

If the building block is not defined, use the value of the development coefficient ZB_V . In particularly justified cases, when it is not possible to define a building block and the land use rate corresponds to the target character, the ZB_N (for unknown block) is used [1].

With the software limitations taken into consideration, the program will provide the user with a percentage of RAB and it is left up to the user to determine whether or not the results for a specific parcel are correct, since the data does not determine which type of structure is provided.

3.2.2 Widths of street open public spaces

Unless the land use or zoning plan determine otherwise, during the delimitation of new streets the width of the street open public space for the individual town planning types of streets must be at least [14]:

- 24 m for city avenues,
- 18 m for important streets,
- 12 m for local streets,
- 8 m for access streets.

The street width will be controlled by a parameter with the lower bound set to 8 m according to the specification above.

3.2.3 Height regulation

The height arrangement is defined by determining development heights, by determining the binding maximum and minimum regulated height of buildings, or by determining the minimum and maximum number of storeys. Development heights determine the minimum and maximum regulated height of buildings and are determined as follows [14]:

1. development height I: 0 m - 6 m,
2. development height II: 0 m - 9 m,
3. development height III: 0 m - 12 m,
4. development height IV: 9 m - 16 m,
5. development height V: 12 m - 21 m,
6. development height VI: 16 m - 26 m,
7. development height VII: 21 m - 40 m,
8. development height VIII: over 40 m;

The height regulation will be controlled by a parameter controlling the maximum number of floors and a parameter controlling the height of a floor.

Roofs

If not determined otherwise by a land use or zoning plan, it is possible to build from the maximum regulated height [14]:

1. a sloping roof with no more than two gables, or with attic storeys, with a maximum angle of 45° and a maximum height of 7.5 m;
2. a recessed storey with a maximum height of 3.5 m, recessed from the outer perimeter wall of the building oriented towards the construction line and one other perimeter wall by at least 2 m;
3. a different spatial solution for the roof that does not exceed the definition according to points 1) or 2).

The program always sets roof height as 50% of the floor height, so it will always obey the rules described above.

3.2.4 Heights and areas of rooms

The ceiling height of habitable rooms must be at least 2.6 m. The minimum ceiling height of a habitable room may be reduced to 2.4 m if the apartment includes at least one habitable room with a height of at least 2.6 m and a floor area of over 16 m^2.
The ceiling height of residence rooms must be at least 2.6 m, for structures for family recreation the ceiling height of residence rooms must be at least 2.4 m. When structures are modified, in attic storeys the ceiling height of all residence and habitable rooms must be at least 2.3 m. In habitable and residence rooms with a sloping ceiling, the lowest permissible ceiling height must cover at least half the floor area of the room. If an apartment consists of a single habitable room, it must have a floor area of at least 16 m^2. The

floor area of rooms does not include any area with ceiling height less than 1.2 m [14].

The parameter controlling floor height in the program will have the lower bound set to 2.6 m.

3.3 Approach to the procedural generation

3.3.1 Addressing the requirements

The program will generate buildings using a bitmap as a house plan. This can ensure the output is not overly detailed while providing some variety in the building shapes by using different variations of the bitmaps. The output of the program will be controlled using a variety of parameters such as the distance between the houses or a number of floors for the houses. Some of these parameters have limitations due to the regulations described above. To enable the generation of various different versions, the randomness in the program will be tied to a parameter. This would allow for different outputs for the same set of parameters.

3.3.2 Structure of the program

The procedural generation is to be divided into several parts to make the node tree as easy to read as possible. The approach of the algorithm is:

1. The dimensions of the parcel will be calculated
2. A grid will be spawned at the place of parcel and cut into the shape of parcel
3. Roads will be generated, both roads already in the parcel and custom roads drawn on additionally
4. The roads will cut the parcel into islands
5. On the islands points will be scattered on which grids with house plans will be instanced, the buildings then will be extruded and given roofs and windows
6. Trees will be generated on randomly distributed points for LODs 1 and 2
7. The percentage of the occupied space will be calculated, allowing to check the coefficient against values in a table for different types and sizes of a lot.

3.3.3 Procedural generation in Blender

All procedural generation in Blender is done through nodes. There are several types of nodes dictated by their input and output parameter types. In Blender 4.0 nodes have a title, input and output sockets and parameters

[15]. Socket shapes indicate whether the nodes takes a single value or a field, which is a function transforming a number of inputs into a single output. In Geometry Nodes, 3 types of socket shapes are recognized [16]:

- **Circle**
  A circle means a single real value for either input or output. Field can not be accepted into this type of socket.

- **Diamond**
  A diamond means input or output is a field. This type of socket can also accept a single value, however this means results of this node will not vary per element.

- **Diamond with a dot**
  A diamond with a dot means the input or output can be a field but currently is set to a single value. This comes with parameters inside the node which allow the user to customise the single value of the socket.

The socket shapes can be seen on Figure 3.1:

Figure 3.1: Cirle (left), Diamond (middle) and Diamond with a dot (right)

3.3.4 Native Blender nodes

The nodes used in this project are described below according to the geometry nodes section of the Blender 4.0 Manual [17].

Transform Geometry

The Transform Geometry node allows the user to translate, rotate and scale the geometry. Each of the operations can use the output from another node as the vector used for the transformation. Node outputs the new transformed geometry in an output node socket.

Bounding Box

The Bounding Box node creates a rectangular box that wraps around the input geometry as tight as possible. Node outputs respective Minimum and Maximum vector coordinates of the bounding box in output node sockets. The Separate XYZ node separates the individual coordinates of the input vector, which are then output in their individual node sockets.

Combine XYZ

The Combine XYZ node combines the individual coordinates given in the input into a vector. The resulting vector is output in a node socket.

Math

Allows a spectrum of different math operations on input numbers. Most used operations in this project are Add, Subtract, Multiply, Divide, Absolute, Ceiling, Clamp and Comparison operations. The result of the operation is output through the node socket.

Vector Math

Allows a spectrum of different math operations on input vectors. Most used operations in this project is Multiply. The resulting vector is then output through the node socket.

Capture Attribute

The Capture Attribute node stores a specific information from the input geometry. This information is then output through the node socket.

Grid

The Grid node creates a grid using the size X and Y input node sockets and the number of X and Y vertices input nodes. The geometry of the grid is then output through the node socket.

Store Named Attribute

The Capture Attribute node stores a specific information from the input geometry in the spreadsheet under a name given through the input node socket. The geometry is then output through the node socket.

Separate Geometry

The Separate Geometry node takes the input geometry and separates a part of the geometry which passes the condition passed through the selection Boolean input node socket. This node then outputs the selected part of the geometry as well as the inverse selection through their respective individual node sockets.

Raycast

The Raycast node shoots rays at the input Target Geometry. An optional Attribute field describes an attribute of the geometry which will be interpolated on the hit points of the geometry. Each ray shot by the node produces

hit points of the input geometry in the direction provided through the Ray direction input socket. The output of this node are positions, normals and distances for each hit point individually as well as the interpolated values of the optional attribute. These are output through their respective vector and float node sockets.

Extrude Mesh

The Extrude Mesh node extrudes the input geometry by an offset specified in the Offset input socket. The user can choose which aspect of the geometry is being extruded, eg. vertices, edges or faces. The user can also optionally specify a condition passed through the Boolean Selection input socket which specifies the selection of the geometry which will get extruded.

Viewer

The Viewer node shows input data in the 3D Viewport and the Spreadsheet. The input is the geometry passed through the input socket node and the optional value evaluated on the geometry input through the Value input socket.

Curve Line

The Curve Line node created a curve on either the input Start and End points provided through the input vector sockets or in a direction from a Start point provided through the input vector sockets. This curve is then output through the node socket.

Subdivide Curve

The Subdivide Curve node takes the curve provided through the Curve input node and subdivides it with the number of cuts described in the Cuts input socket. The subdivided curve is then output through the node socket.

Quadrilateral

The Quadrilateral node creates a polygonal curve in the shape of either a Rectangle, Trapezoid, Kite or a Parallelogram. This curve has the width and height provided through the Width and Height input socket nodes. If a shape other than the Rectangle is used other input parameters may become available. This curve is then output through the node socket.

Curve to Mesh

The Curve to Mesh node converts a splines of a curve into a mesh. An optional Profile Curve may be provided through the input node socket which gives the mesh a specific shape. This mesh is then output through the node socket.

Subdivide Mesh

The Subdivide Mesh node takes the input geometry and subdivides it to a level specified through the Level integer input node. The subdivided mesh is then output through the output node.

Join Geometry

The Join Geometry node takes all of the geometry passed through the input node socket and outputs it as one single geometry through the node socket.

Realize Instances

The Realize Instances node converts any Instances of the input geometry into real geometry. This is then output through the node socket.

Duplicate Elements

The Duplicate Elements node duplicates a specified aspect of the input geometry a number of times provided in the Amount integer input node. The user can specify whether they wish to duplicate vertices, edges, faces, splines or instances. Additionally a condition which specifies which section of the geometry is being duplicated. This geometry is then output through the node socket.

Triangulate

The Triangulate node converts all faces in a mesh (quads and n-gons) to triangular faces. The output is a triangulated mesh.

Geometry Proximity

The Geometry Proximity node calculates the closest point on the input geometry. The user can specify whether they wish to calculate it on vertices, edges or faces. The position of the calculated point is then output through the node socket along with the distance of the point from the geometry.

Random Value

The Random Value node generates a value between the input minimum and maximum using a seed provided through the input socket. This value is then output through the node socket.

Distribute Points on Faces

The Distribute Points on Faces node distributes points on the input geometry either randomly or using a Poisson Disk using a seed specified through the input socket. The user can specify density through the input socket or in the

node itself. When using a Poisson Disk user can also specify the minimum distance between points and maximum density. The points are then output through the node socket.

Instance on Points

The Instance on Points node instances input geometry on the points provided through the input socket. The user can specify a rotation and scale vectors for the instances. The instances are then output through a node socket.

Cube

The Cube node generates a cube using the size and vertex amount for each axis specified through the input sockets. This cube is then output through the node socket.

Attribute Statistic

The Attribute Statistic node takes the input geometry and the input Attribute and outputs a number of different statistics. The user can specify what type the attribute has and whether it is calculated per point, edge, face, spline or instance. The statistics can include the mean, median, sum, min, max, range, standard deviation or the variance. This is all output through individual node sockets.

Value to String

The Value to String creates a String representing the value passed through the input socket. This string is then output through the node socket.

Join Strings

The Join Strings node takes input strings and joins them into one. This string is then output through the node socket.

String to Curves

The String to Curves node create a curve representation of the string passed through the input socket. This curve is created using a specified font, size spacing and width. These can all be specified in the node or through the input node sockets. The curve is then output through the node socket.

Switch

The Switch takes a condition passed through the input node socket and based on the value outputs the input type based on the result. The user can specify the input type of the node which then in turn affects what format the True and False input sockets and the output socket has.

Map Range

Remaps a value from a specified input range to a target range.

Float Curve

Maps an input float to a curve and outputs a float value.

Align Euler to Vector Node

Rotates an Euler rotation into the given direction.

Rotate Euler

Rotates an Euler rotation by a specified amount.

Group Input

The Group Input node specifies all input parameters for the node tree.

Group Output

The Group Output collects all outputs of the node tree.

3.3.5 Custom Nodes

A number of nodes was created to fit the needs for this thesis and to make the tree visualising the algorithm easier to read. This would be akin to creating functions to encapsulate algorithms.

Calculate size of Grid

Calculates size of a grid needed to cover a parcel. The dimensions are the output.

Create Grid

Creates a grid to cover a parcel. Output is geometry of grid.

Create Mesh for Roads

Creates a mesh to represent roads crossing the parcel. Geometry is the output.

Create Space around Roads

Creates a wider version of the roads used for instancing so overlapping the road would not be a problem. Geometry of the wider roads is the output for this node.

Calculate Geometric Proximity to Roads

Calculates the distance from the nearest road and outputs a probability for instancing. The probability as well as the edges of the parcel are the outputs for this node.

Extrude Houses

Extrudes the faces of the houses. Extruded geometry is then output.

Create Roofs

Creates roof geometry for the houses which is then output.

Distribute Points on Curve

Distributes points along a curve and then outputs said points.

Simple Tree Generator

Creates geometry for a simple tree which is then instanced.

Detailed Tree Generator

Creates geometry for a detailed tree which is then instanced.

Calculate Building Area

Calculates building area of the parcel which is then output.

Place Text Depicting Building Area

Creates a geometry for building area percentage and outputs the mesh.

Chapter 4

Implementation

4.1 Importing data

The Shapefile data is imported into Blender with BlenderGIS add-on through the GIS→Import→Shapefile options. This data is then selected and through the edit mode all geometry is selected, which is then seperated through the Mesh→Separate→By Loose Parts option. It is also required to import all of the roads as a seperate object and name it as "roads".

4.2 Group Input Parameters

The parameters used to adjust the results of the procedural generation are described below.

4.2.1 Street Width

This float input parameter signifies the width of the roads on the parcel, as shown on Figure 4.1. The default value is 8 meters.

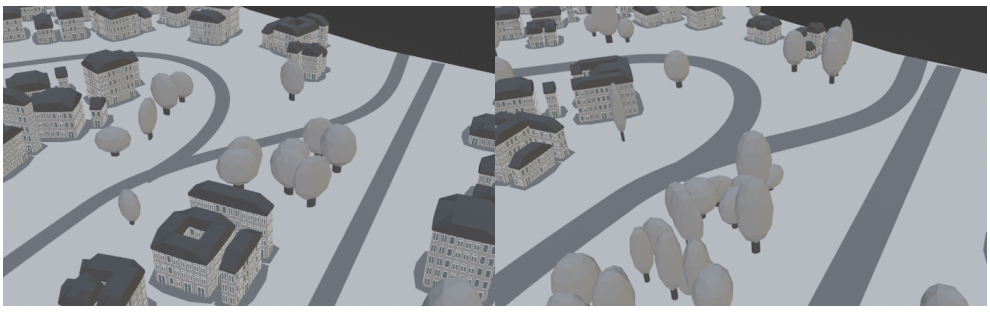

Figure 4.1: Street Width set to 5m (left) and 10m (right)

4.2.2 Median

This vector input parameter is used for the local translation of the lot. The default value is set to (0,0,0) so no translation is done unless the parameter is modified. The information that is to be input in this location is available after selecting the relevant geometry, going into edit mode and selecting all geometry. Then navigating into the Median attribute with the Local setting selected, in the Transform part of the Item window in the Viewport. This is used for placing the grid, as shown on Figure 4.2.

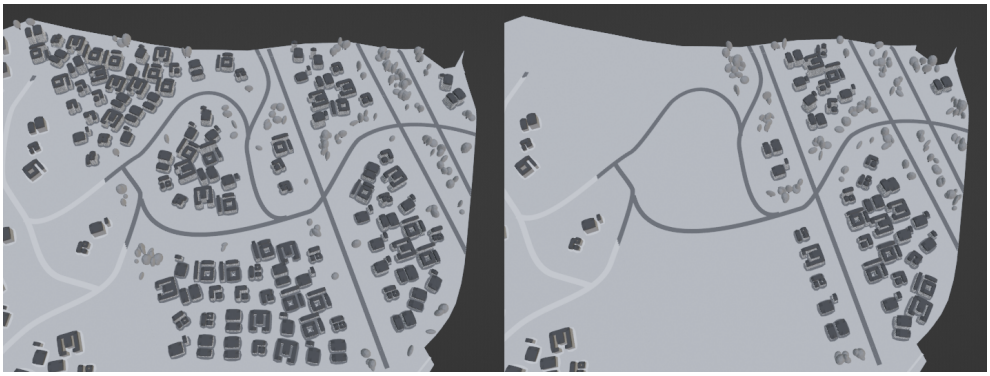

Figure 4.2: Median x value set to -5170.66 (left) and -4770.66 (right)

4.2.3 Percentage

This Boolean input parameter is used to show the percentage of the buildings to free area in the lot, as shown on Figure 4.3. This can be used to check if the population of the lot adheres to the rules.

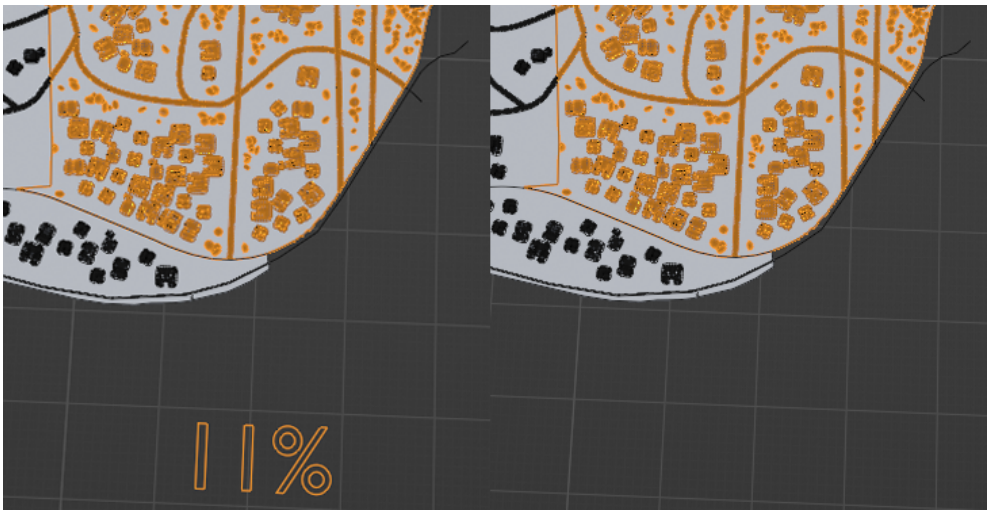

Figure 4.3: Percentage set to true (left) and false (right)

4.2.4 LOD

This input parameter is used to control the level of detail of the model, as shown on Figure 4.4. The values range from 0 to 2. Value 0 represents the lowest level of detail while 2 represents the highest level of detail.

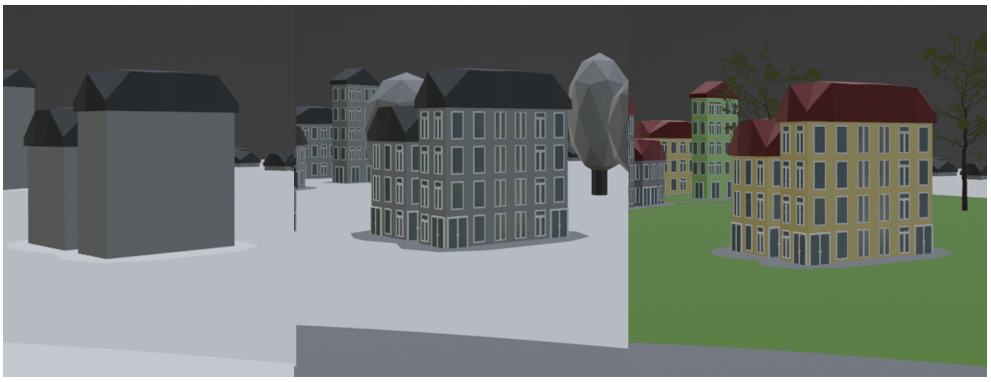

Figure 4.4: LOD set to 0 (left), 1 (middle) and 2 (right)

4.2.5 Floor Height

This input parameter is used to control the height of the floors of the buildings, as shown on Figure 4.5.

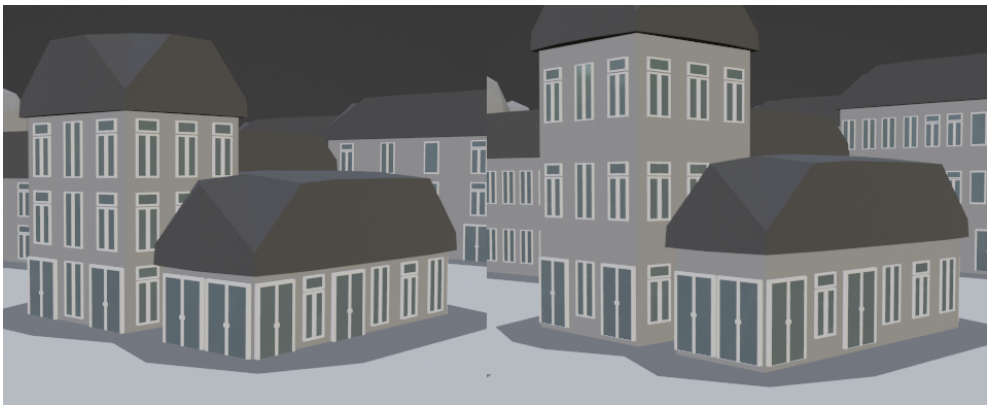

Figure 4.5: Floor Height set to 2,6 (left) and 3,5 (right)

4.2.6 Distance Min

This input parameter is used to control the minimum distance between the buildings as well as the distance from the edges of the parcel, as shown on Figure 4.6.

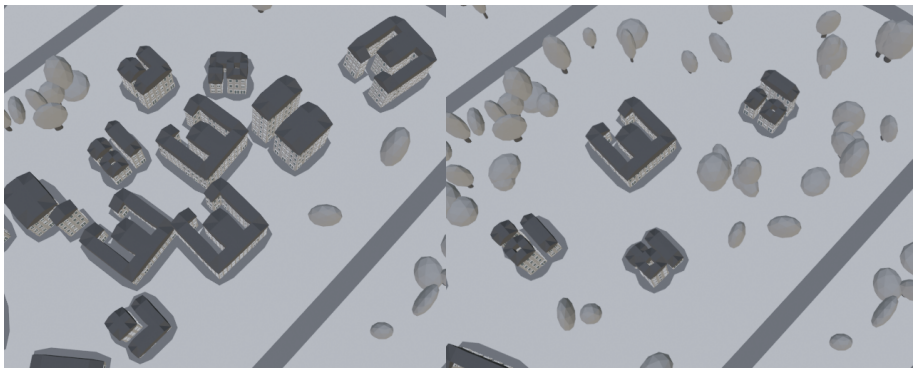

Figure 4.6: Distance Min set to 25 (left) and 35 (right)

4.2.7 Max Floors

This input parameter is used to control the absolute maximum number of floors for the buildings generated on the parcel, as shown on Figure 4.7. This number represents the absolute maximum meaning that any number below it is a valid choice for a number of floors on the parcel.

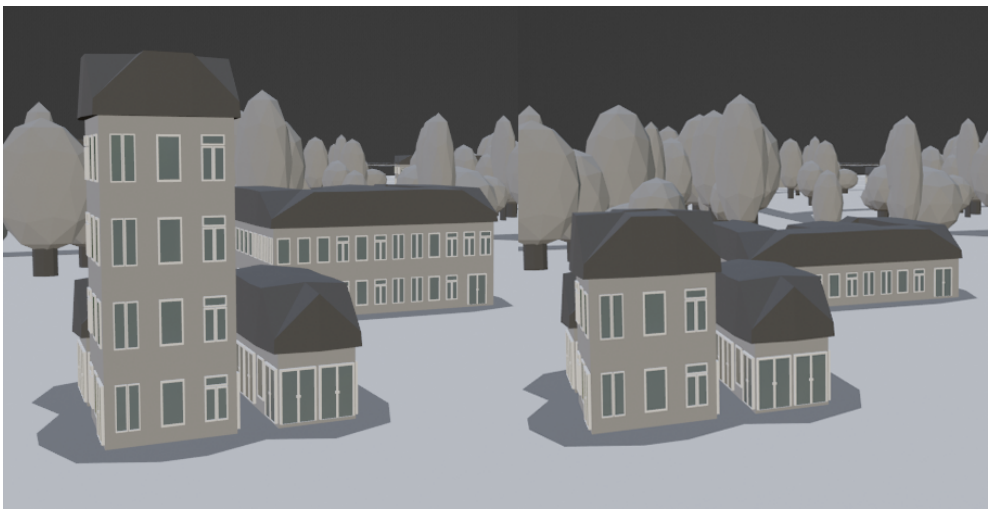

Figure 4.7: Max Floors set to 6 (left) and 3 (right)

4.2.8 Min Density

This input parameter is used to control the minimum density of the buildings, as shown on Figure 4.8. The density is chosen by choosing a random value from the interval of <Min Density, 1>.

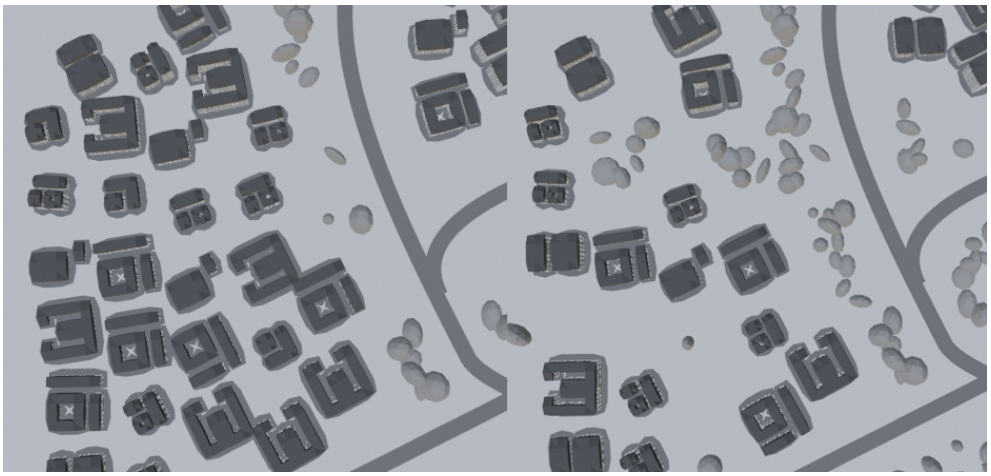

Figure 4.8: Min Density set to 0.8 (left) and 0.3 (right)

4.3 Additional Group Input Parameters

The parameters described below were added according to the feedback detailed in the section number 5.2 Feedback. These additional parameters adjust the number of cells of a grid of streets generated on the parcel.

4.3.1 Rows

This integer input parameter affects the number of rows in the generated street grid, as shown on Figure 4.9. The default value is 3. If a street grid is not wanted, this value is to be set to 0.

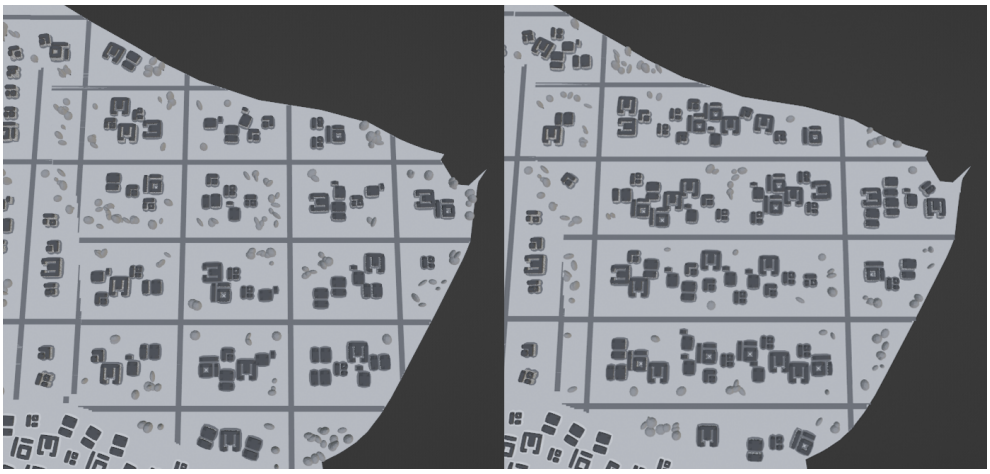

Figure 4.9: Rows set to 8 (left) and 4 (right)

4.3.2 Columns

This integer input parameter affects the number of rows in the generated street grid, as shown on Figure 4.10. The default value is 3. If a street grid is not wanted, this value is to be set to 0.

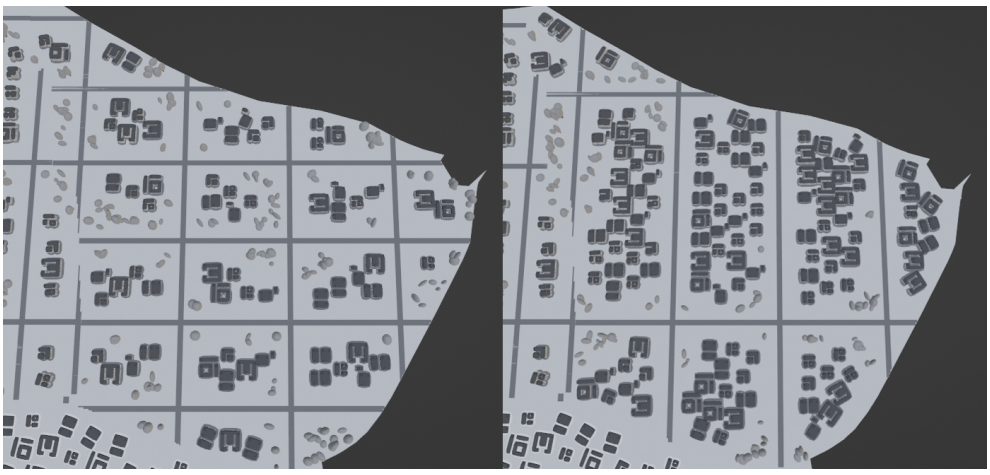

Figure 4.10: Columns set to 8 (left) and 4 (right)

4.4 Program

The program is divided into node groups as was detailed in the section 3.3.2.

4.4.1 Creating the grid

The grid is created using two node groups, *Calculate size of grid* and *Create grid*. The node tree for this section can be seen on the figure 4.11 below:

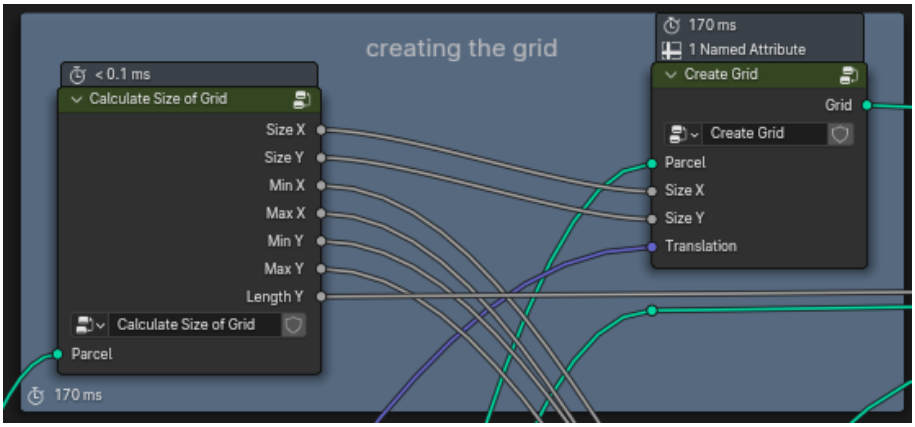

Figure 4.11: Node tree for creating the grid

Calculate size of grid

The pseudocode for the grid size calculation is described in Algorithm 1:

Algorithm 1 Grid size calculation:

1: $boundBox \leftarrow BoundingBox(parcel)$
2: $difX \leftarrow abs(boundBox.Min.x - boundBox.Max.x)$
3: $difY \leftarrow abs(boundBox.Min.y - boundBox.Max.y)$
4: $gridX \leftarrow difX * 1.5$
5: $gridY \leftarrow difY * 1.5$

This section describes the implementation of Algorithm 1 in Blender's Geometry Nodes.

A bounding box is created around the parcel using the **Bounding box** node. The Min and Max sockets are then plugged into a **Separate XYZ** nodes, the X and Y values are then respectively subtracted to produce the distance between them. This distance is later used to create the grid.

The distances are each plugged into an **Absolute** node in case the values are negative. Then the results are multiplied by 1.5 to slightly increase the size of the grid to make it easier to cut out a shape resembling the parcel as close as possible.

Create grid

The pseudocode for the creation of the grid is described in Algorithm 2:

Algorithm 2 Making the grid:

```
1: grid ← Grid(gridX, gridY, 1000, 1000)
2: grid.Transform(Median, (0, 0, 0), (1, 1, 1))
3: newgrid ← null
4: for each face ∈ grid do
5:     if Raycast(face, parcel) then
6:         newgrid+ = face
7:     end if
8: end for
```

This section describes the implementation of Algorithm 2 in Blender's Geometry Nodes.

A grid is created using the grid size calculated in the previous step. The number of vertices in the grid is a 1000 on each axis, which should allow for enough detail. This grid is then subsequently used as a substitution for **Mesh Boolean** nodes to make the algorithm faster.

This grid is then translated by the *Median* value so it is in a correct spot as the parcel is. A new grid is subsequently made containing only the cells of the grid which are inside the parcel. This is done by a **Raycast** node which casts rays from a specific direction, in this case in the negative direction of the Z axis from the distance of 100m.

4.4.2 Creating road curves

The grid is created using two node groups, *Create Mesh for Roads* and *Create Space around Roads*.

The pseudocode for the road curves creation is described in Algorithm 3:

Algorithm 3 Creating road curves:

$CustomRoads \leftarrow SubdivideCurve(customRoads, 5)$
$ExistingRoads \leftarrow SubdivideCurve(MeshToCurve(roads), 5)$
$CreateMeshForRoads(CustomRoads, ExistingRoads, Parcel, StreetWidth)$
$CreateSpaceAroundRoads(CustomRoads, ExistingRoads, StreetWidth)$

Create Mesh for Roads

The pseudocode for the road mesh creation is described in Algorithm 4:

Algorithm 4 Create Mesh for Roads:

$road1 \leftarrow CurveToMesh(CustomRoads, Quadrilateral, 0)$
$road2 \leftarrow CurveToMesh(ExistingRoads, Quadrilateral, 0)$
$subdivideMesh(road1, 2)$
$subdivideMesh(road2, 2)$
$newRoads \leftarrow null$
for each $point \in roads$ **do**
    if Raycast(point, parcel) **then**
        $newRoads+ = point$
    end if
end for
$selection \leftarrow null$
for each $face \in newRoads$ **do**
    if face.normal == (0,0,1) **then**
        $selection+ = face$
    end if
end for
$ExtrudeMesh(selection, (0, 0, 0), 0.05, 0)$
$MergeByDistance(selection, 1)$

The "roads" object is plugged into a **Mesh to Curve** node and then plugged into a **Subdivide Curve** node. This is then plugged into the Existing Roads input of this node.

Similar is done with the "customRoads" curve object, only it omits the **Mesh to Curve** node seeing as the object already consists of curves.

This section describes the implementation of Algorithm 4 in Blender's Geometry Nodes.

The curves are converted into a mesh using a **Curve to Mesh** node with a Quadrilateral curve as a profile curve, meaning the curve assumes the shape of the quadrilateral curve along its entire length. These curves are then subdivided twice for better selection later on. The curves are then joined together and a selection that is in the bounds of the parcel are then separated and passed along to the **Extrude Mesh** node to extrude the roads upwards to add thickness to the roads. At the end, the vertices are passed along to the **Merge by Distance** node to merge the vertices at the crossroads. This prevents shading issues.

Create Space around Roads

The pseudocode for the road mesh creation is described in Algorithm 5:

Algorithm 5 Create Space around Roads:

$newStreetWidth \leftarrow StreetWidth * 3$
$Roads[2] \leftarrow null$
$Roads[0] \leftarrow CurveToMesh(CustomRoads, Quadrilateral, 0)$
$Roads[1] \leftarrow CurveToMesh(ExistingRoads, Quadrilateral, 0)$
$newRoads \leftarrow null$
for $i = 0, i < 2, i++$ **do**
    $newRoads+ = Roads[i]$
end for

This section describes the implementation of Algorithm 5 in Blender's Geometry Nodes.

The Street Width parameter is multiplies by 3 to create enough space around roads. This is then plugged into the **Quadrilateral** node Width input parameter with Height set to 0.0001. This is then plugged into the **Curve to Mesh** nodes with each of the road inputs. All of the geometry is then plugged into a **Join Geometry** node which is then plugged into the output node.

4.4.3 Cutting out roads

The pseudocode for the cutting of parcel using roads is described in Algorithm 6:

Algorithm 6 Cutting out roads:

$cutOutParcel \leftarrow null$
for each $face \in newRoads$ **do**
    if !Raycast(face, parcel) **then**
        $cutOutParcel+ = point$
    end if
end for

This section describes the implementation of Algorithm 6 in Blender's Geometry Nodes.

The grid created in the algorithm 2 is plugged into the **Raycast** node to separate the cells which do not overlap the roads.

4.4.4 Instancing bases of houses

There are 3 distinct types of house bases used for instancing. These is a small 15x15m lot, big 25x25m lot and a rectangular 20x25m lot (in a HeightxWidth format). These are then randomly assigned a blueprint according to which the extrusion is later done. The instancing of the house bases is made up of two node groups - *Calculate Geometric Proximity to Roads* and *Instance on Points and Align to Face Road* and several other nodes. The pseudocode for the instancing of houses is described in Algorithm 7:

Algorithm 7 Instancing houses:

1: $houseVector \leftarrow (12, 10, 3)$
2: $grid \leftarrow Grid(houseVector.x, houseVector.y, 5, 5)$
3: $parcelEdges, probability \leftarrow CalculateGeometricProximityToRoads($ $ScaleofLot, cutOutParcel)$
4: $points \leftarrow DistributePointsOnFaces(cutOutParcel, 1, houseVector.x$ $*scaleOfLot * 0.025, houseVector.x * scaleOfLot * 2,$ $RandomValue(0, 1), Seed)$
5: $InstanceOnPointsAndAlignToFaceRoad(parcelEdges, scaleOfLot,$ $points, probability, grid)$

This section describes the implementation of Algorithm 7 in Blender's Geometry Nodes.

First the proximity to the parcel edges and roads is calculated using the **Geometric Proximity** node. This is done to ensure the houses do not spawn overlapping the roads or protruding from the parcel. Then points are distributed on the grid from the algorithm 6. The distribution of the points is done using a Poisson Disk. The minimum distance between the points is affected by the Distance Min parameter. The density factor of the Poisson Disk is randomly generated from the interval of <0,1>. Next house bases are instanced on the points generated by the Poisson Disk. The probability of the instancing on those points is influenced by the proximity to the parcel edges.

House Bases

The house bases are placed in the "grids" collection. They are respectively named "big grid" (25x25m), "rectangle" (20x25m) and "small grid" (15x15m). Each of these grids have 3 possible building placement possibilities. These are stored as 8x8 bitmaps with white color signifying the building placement. The bitmap variant is randomly assigned to each instance of the base. Using these bitmaps tiles are selected and passed on to the extrusion algorithm.

The different bitmaps can be seen in on the figure 4.12 below. The first row are the placements for the big grid, second for small and third for rectangle.

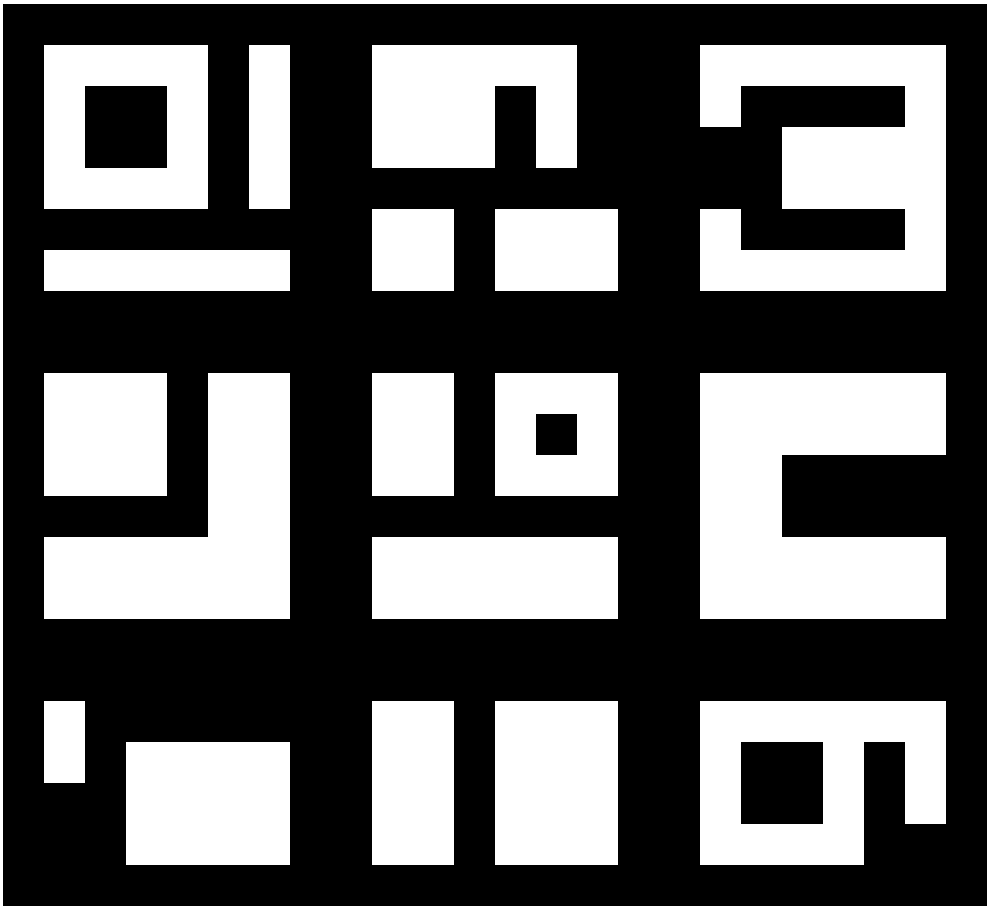

Figure 4.12: Different bitmap versions

Calculate Geometric Proximity to Roads

The pseudocode for the instancing of houses is described in Algorithm 8:

Algorithm 8 Calculate Geometric Proximity to Roads:

$edgesOfParcel \leftarrow null$
for each $edge \in cutOutParcel$ **do**
    if edge.facecount == 1 **then**
        $edgesOfParcel += edge$
    end if
end for
$probability \leftarrow 0$
for each $edge \in cutOutParcel$ **do**
    $distances[] \leftarrow GeometricProximity(2, cutOutParcel, edge.pos)$
end for
for each $distance \in distances$ **do**
    $probabilities[] \leftarrow RandomValue(distance > RandomValue(DistanceMin * 0,5, DistanceMin * 0,75))$
end for

This section describes the implementation of Algorithm 8 in Blender's Geometry Nodes.

The edges of the parcel created in algorithm 6 are duplicated and the proximity to them is calculated. Then based on the distance the probability for the instancing is calculated.

Instance on Points and Align to Face Road

The pseudocode for the instancing of houses is described in Algorithm 9:

Algorithm 9 Instance on Points and Align to Face Road:

$scales[points.amount] \leftarrow null$
for int i = 0, i < points.amount, i++ **do**
    $x \leftarrow Clamp(ScaleOfLot * RandomValue(0.5, 5), 1, 3)$
    $y \leftarrow Clamp(ScaleOfLot * RandomValue(0.5, 5), 1, 3)$
    $scales[i] \leftarrow RandomValue((1, 1, 1), (x, y, 1))$
end for
$rotations[points.amount] \leftarrow null$
for int i = 0, i < points.amount, i++ **do**
    $vec \leftarrow GeometricProximity(parcelEdges, point.pos)$
    $rotations[i] \leftarrow AlignEulerToVector(2, 4, null, 1, absolute(points[i].pos - vec))$
end for
$instances[] \leftarrow InstanceOnPoints(points, probabilities, instance, 0, rotations, scales)$

This section describes the implementation of Algorithm 9 in Blender's Geometry Nodes.

The Points, Probability and Instance of the **Instance on Points** node are all plugged in from the node group input.

The rotation of the instances is calculated by the **Align Euler to Vector** node. This is done by aligning the Y axis to a vector. This vector is an absolute value of subtraction of a position of a point from the position of the closest parcel edge. The node is limited to aligning only according to one axis therefore the alignment to the road is quite limited.

4.4.5 Extruding the floors

The pseudocode for the instancing of houses is described in Algorithm 10:

Algorithm 10 Extruding the floors:

1: $iterations \leftarrow domain.size * maxNumberOfFloors$
2: **for** int i = 0, i < iterations, i++ **do**
3: $\quad modulo \leftarrow TurnucatedModulo(seed, islandCount)$
4: $\quad probability \leftarrow (modulo == islandIndex)$ $\&\&RandomBoolean(0.453, Seed)\&\&(normal == (0, 0, 1))$
5: $\quad ExtrudeMesh(mesh, probability, floorHeight)$
6: **end for**

This section describes the implementation of Algorithm 10 in Blender's Geometry Nodes.

Blender has its own version of the for cycle called "Repeat zone". This repeat zone does not allow for randomness inside the cycle. Therefore randomness has to be artificially created using IDs. **Random Value** nodes output the same outcome for every object and iteration inside the repeat zone. However if an ID is plugged into the **Random Value** node and then every iteration the ID is increased by one therefore allowing for randomness in each iteration.

The number of iterations is the number of instances multiplied by the maximum amount of floors. Each iteration one building is extruded. This ensures that the buildings have different heights.

4.4.6 Instancing the windows and doors

The pseudocode for the instancing of houses is described in Algorithm 11:

Algorithm 11 Instancing the windows and doors:

$points \leftarrow MeshToPoints(houses)$
$doorProb[] \leftarrow null$
$i \leftarrow 0$
for each $point \in points$ **do**
    $doorProb[i] \leftarrow ((point.pos < floorHeight)\&\&RandomValue(0.5))\&\&(SampleNearestSurface(houses, normal)! = (0,0,1))$
    $i++$
end for
$doors[] \leftarrow InstanceOnPoints(points, doorProb, doorCollection, 1, (SampleNearestSurface(houses, edgeVerticesPosition1) - SampleNearestSurface(houses, edgeVerticesPosition1) > 4.5), AlignEulerToVector(SampleNearestSurface(houses, normal), (1,1,1)))$
$TranslateInstances(instances, (-0.175, 0.75, 0), 1)$
$SetPosition(instances, (pos.x, pos., 1.3))$
for each $point \in points$ **do**
    $windowProb[i] \leftarrow !((point.pos < floorHeight)\&\&RandomValue(0.5))\&\&(SampleNearestSurface(houses, normal)! = (0,0,1))$
    i++
end for
$windows[] \leftarrow InstanceOnPoints(points, windowProb, windowCollection, 1, (SampleNearestSurface(houses, edgeVerticesPosition1) - SampleNearestSurface(houses, edgeVerticesPosition1) > 1.5), AlignEulerToVector(SampleNearestSurface(houses, face), (1,1,1)))$
$TranslateInstances(instances, (0, -0.0375, 0), 1)$

This section describes the implementation of Algorithm 11 in Blender's Geometry Nodes.

The house mesh is converted into points with each face being made into a separate point.

For doors, points which are not higher than the Floor Height parameter are randomly selected from and then instanced a door from the collection of doors in the file. They are then aligned with the face behind them. Based on the width of the face from which the point was created, a smaller or a wider door is selected from the collection. This width is calculated by sampling the nearest two edges and calculating their distance. The doors are then all set 1.25m high on the Z axis. This is because the center of the geometry of the doors is in the middle of the mesh and the height of the door is approximately 2.5m.

For windows, points which are not used for instancing the doors are selected and then instanced a window from the collection of windows in the file. They

are then aligned with the face behind them. Based on the width of the face from which the point was created, a smaller or a bigger window is selected from the collection.The width is calculated in the same way as for the doors. There are 3 versions of the smaller windows, and 4 versions of the bigger windows. These are all randomly selected from according to the respective width.

4.4.7 Creating sidewalks

The pseudocode for the creation of sidewalks around the buildings is described in Algorithm 12:

Algorithm 12 Creating sidewalks:

1: $mesh \leftarrow TransformGeometry(parcels, (0, 0, 0.5), (0, 0, 0), (1, 1, 1))$
2: **for each** $point \in mesh$ **do**
3: **if** point.facecount $== 1$ **then**
4: $sidewalks[] \leftarrow MergeByDistance(mesh, 0.25)$
5: **end if**
6: **end for**

This section describes the implementation of Algorithm 12 in Blender's Geometry Nodes.

The outer edges of the meshes of parcels are merged by distance to create the sidewalks around the buildings. Merging only the outer edges ensures the geometry is not irregular and just that the edges are beveled.

4.4.8 Instancing the trees

The pseudocode for the instancing of houses is described in Algorithm 13:

Algorithm 13 Instancing the trees:

1: $points \leftarrow DistributePointsOnFaces(cutOutParcel, 1, RandomValue(0, 1), Seed)$
2: $probability \leftarrow 0$
3: **for each** $edge \in edge.pos$ **do**
4: $probabilities[] \leftarrow (GeometricProximity(2, cutOutParcel, edge.pos) < 5)\&\&(GeometricProximity(2, houses, edge.pos) < 15)$
5: **end for**
6: $instances[] \leftarrow InstanceOnPoints(points, probabilities, instance, 0, RandomValue((0, 0, 0), (0, 0, 360)), RandomValue((1, 1, 1), (3, 3, 3)))$
7: $TranslateInstances(instances, (0, 0, 2), 1)$

This section describes the implementation of Algorithm 13 in Blender's Geometry Nodes.

The points on which the trees are instanced are distributed on the parcel produced by the algorithm 6. These points are distributed randomly with density as a random value in the interval <0, 0.25>. Then from these points, the points that have proximity to the edges of the parcel higher than 5m and to the houses higher than 15m.

Trees are then instanced on these points. For the LOD of 0 and 1, a simple tree is used for instancing whereas for the LOD of 2, a more detailed procedurally generated tree is used.

The simple tree consists of an ico sphere as the tree crown and a cylinder as the bark.

The procedurally generated tree consists of three levels of curves and an extra curve for the bark. First the curve for the bark is created, then on the upper parts of the bark curve, points are distributed on which a number of curve lines is instanced representing the branches. Then, same process is repeated for the branches. All these curves are then distored with a noise texture. Last, the curves are turned into a mesh with a circle as the profile curve. The thickness is affected by how high on the curve the point is, the higher it is the thinner.

4.4.9 Creating roofs for the houses

The pseudocode for the instancing of houses is described in Algorithm 14:

Algorithm 14 Creating roofs for the houses:

```
1:  topsOfHouses[] ← null
2:  for each face ∈ houses do
3:      if face.normal == (0,0,1) then
4:          topsOfHouses+ = face
5:      end if
6:  end for
7:  ExtrudeMesh(topsOfHouses, 0.5)
8:  for each point ∈ topsOfHouses do
9:      if point.faceCount > 3 then
10:         SetPosition(point, (0, 0, 2))
11:     end if
12: end for
13: Triangulate(topsOfHouses)
14: ExtrudeMesh(topsOfHouses, 0.5)
```

This section describes the implementation of Algorithm 14 in Blender's Geometry Nodes.

To create roofs of the buildings, the roofs are extruded. Next, all points with a face count of more than 3 are moved upwards 2 meters. This makes the middle line of the move upwards creating the shape of a roof.

The roof is assigned a dark grey material in the LOD of 0 and 1 and a dark orange color in the LOD 2.

4.4.10 Calculating building area

The calculation is made up of a *Calculate building area* node group and a *Place Text Depicting Building Area* node group.

Calculating building area

The pseudocode for the calculation of the building area is described in Algorithm 15:

Algorithm 15 Calculating building area:

```
triFloorplan ← SubdivideMesh(Triangulate(floorplans, 4), 3)
newFloorplan ← MergeByDistance(triFloorplan, 0.5)
totalFaceArea ← 0
for each face ∈ parcel do
    totalFaceArea+ = face.faceArea
end for
buildingFaceArea ← 0
for each face ∈ newFloorplan do
    buildingFaceArea+ = face.faceArea
end for
faceAreaPercentage ← buildingFaceArea/totalFaceArea ∗ 100
```

This section describes the implementation of Algorithm 15 in Blender's Geometry Nodes.

The floorplan mesh is triangulated and then subdivided by 3 levels. This is then merged by distance every 0.5 m. This is done in case the buildings overlap to remove the overlap and thus get a correct sum of areas. The building area is calculated by calculating the sum of **Face Areas** of the original parcel geometry using the **Attribute Statistic** node and the sum of **Face Areas** of the buildings and roads. The building face area is then divided by the face area of the parcel multiplied by 100 and turned into a string using a **Value to String** node with decimals set to 0. Then a percentage sign is joint with the string and plugged into a **String to Curve** node. The visibility of this percentage is dependent on the Group Input *Percentage* parameter.

Place Text Depicting Building Area

The pseudocode for the placing of the text is described in Algorithm 16:

Algorithm 16 Place Text Depicting Building Area:

```
text ← string(faceAreaPercentage)+ %
textCurves ← StringToCurves(text)
TransformGeometry(textCurves, Median, (0, 0, 0), (1, 1, 1))
TransformGeometry(textCurves, (0, Grid.LengthY, 0), (0, 0, 0), (1, 1, 1))
```

This section describes the implementation of Algorithm 16 in Blender's Geometry Nodes.

The text for the building area is converted into a string and a percentage sign is added. Then the string is converted into curves and translated into the correct position by the Median parameter. Finally, the geometry is moved downwards by the length of the grid calculated in Algorithm 1.

4.5 Shading and Materials

For LOD (Level of detail) 2 a special shader was made for the buildings which utilises the islandid attribute. This attribute is then remapped from values of 0 to 100 to values from 0 to 1. This remapped value is then used for a factor in a color ramp, from which the color of the wall is chosen.

This can be seen in the figure 4.13 below:

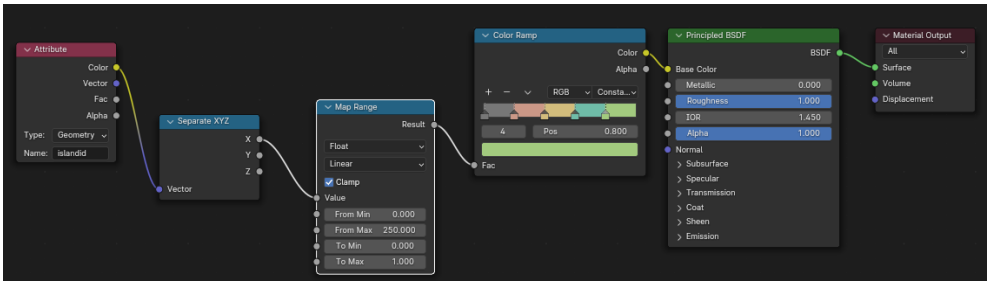

Figure 4.13: Node tree for the shader of the wall material

4.6 Levels of detail

Three levels of detail are implemented in the algorithm. The lowest LOD contains only the buildings with roofs. Simple trees and windows and doors are added in the second LOD. The lowest and second lowest LODs both have a monochromatic colour scheme consisting of different shades of grey. The last third LOD adds detailed trees as well as a realistic color scheme with green grass and different coloured buildings.

It is recommended to use LOD 0 and LOD 1 while modifying the parameters. The last LOD is not recommended while changes are actively being made to the parameters and it is quite costly to compute. It should only be used for static shots unless a powerful graphics card is used.

On the following figure 4.14, the differences between the levels can be seen, with LOD 0 being the top-most image and LOD 2 being the bottom-most image.

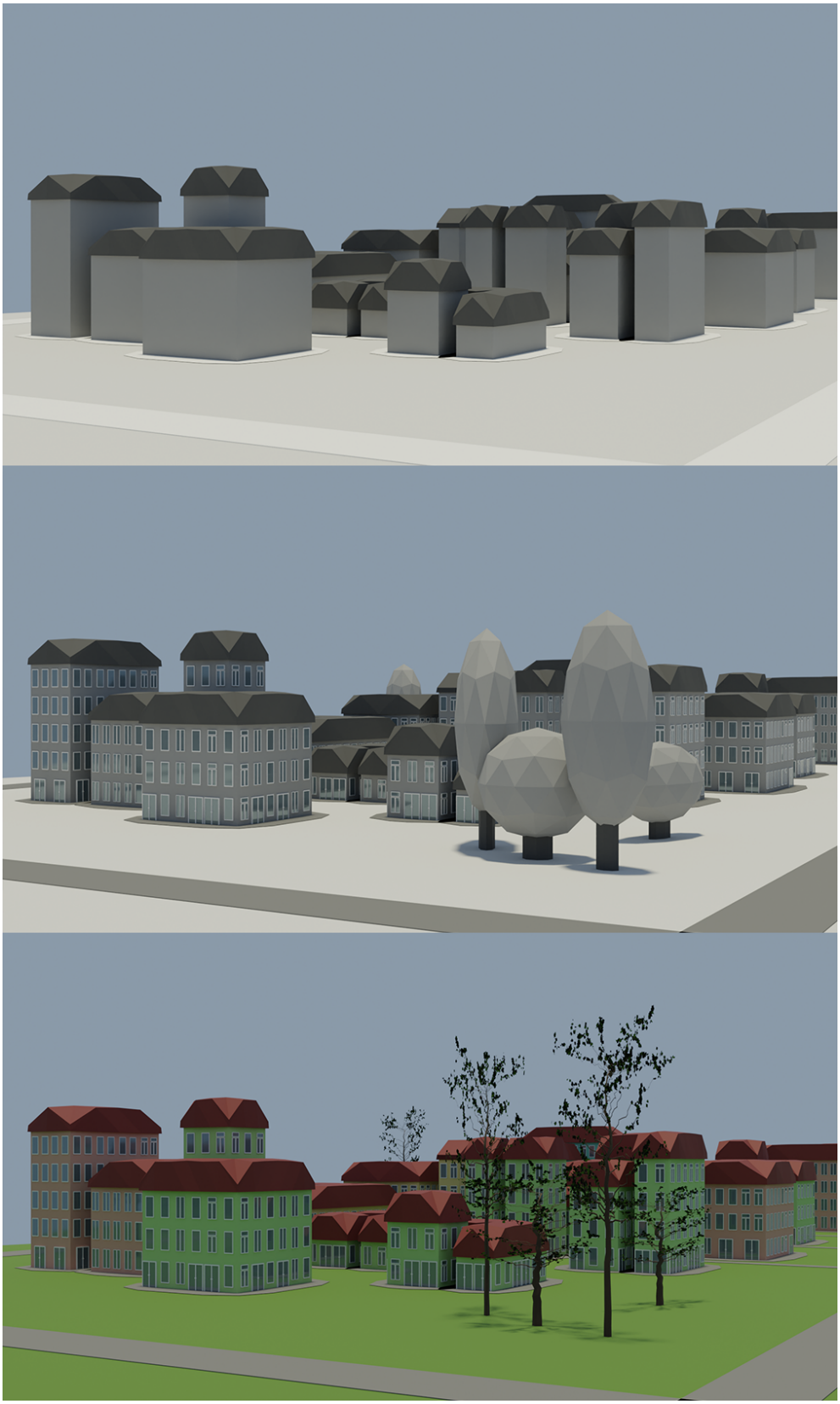

Figure 4.14: Different levels of LOD 0 being the top and 2 the bottom

4.7 Custom Roads

Custom Roads can be drawn into the object "customRoads". To draw the roads, the object needs to be selected, after which it is needed to enter the **Edit Mode**. This can be done via the dropdown in the upper left corner of the viewport. Then the it is needed to enter the **top view** by clicking on the blue Z button of the axis in the upper right corner of the viewport. And finally the **Draw** tool is to be selected in the left sidebar of the viewport. All of these points are highlighted in the figure 4.15 below:

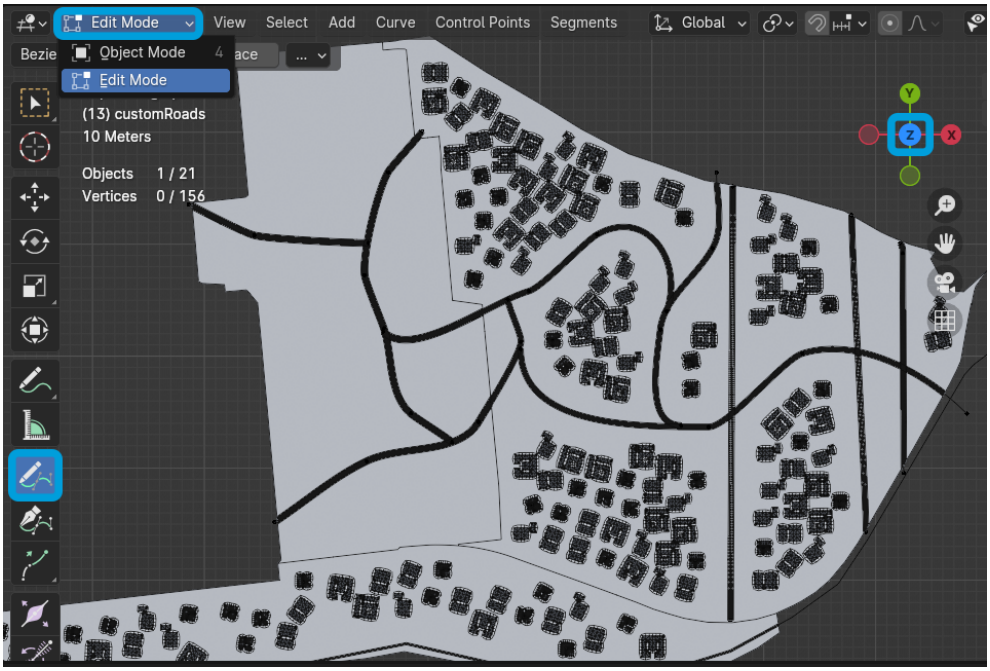

Figure 4.15: Highlighted relevant parts of viewport

After the draw tool is selected all is needed is to draw the curve on the parcel. It is important this be done from the top view only as any other shifting of view will cause the road to not be drawn on the parcel but above it instead.

To delete the vertices of the drawn curve, it is needed to go into the edit mode and go into the selection mode by using the Q shortcut or the topmost tool in the toolbar on the left. Then the vertices need to be selected and then deleted using the Delete button on the keyboard. A window will be shown asking whether to delete the vertices or segments. Either option is fine.

4.8 Automation of the parameter input

A python script was written to remove the repetitive nature of inputting the parameters by hand. The python script scans the inputs from a JSON file and sets them as parameters. It is necessary to put the json file with inputs in the tmp folder of the home path, if no such folder exists it is required to create it. The file also has to be named input.json. The index variable sets which set of inputs should be scanned. The index of a first set of inputs in the json file is 0.

The structure of the json file is as in the following example:

```
{ "0":                          - index number of the entry
    [
      {                         - start of dictionary of parameters
        "StreetWidth": 8.0,
        "Percentage": 0,
        "Seed": 2,
        "LOD": 0,
        "FloorHeight": 2.6,
        "DistanceMin": 25,
        "MaxFloors": 6,
        "MinDensity": 0.8
      }                         - end of dictionary of parameters
    ],                          - comma after the ending square bracket
                                  only if another entry follows
"1":
     [
       ...
     ]
}
```

The script is ran after selecting an object for which the paremeters are to be set for. The script is accessible in the Scripting window of the Blender interface. It can be ran using the play button or via the Alt + P shortcut or via the Text -> Run Script menu.

The python script is detailed below in Algorithm 17:

Algorithm 17 Scanning Inputs:

```
1: import bpy
2: from mathutils import Vector
3: import json
4: import pathlib

5: def get_objcenter(obj):
6:         vertices = obj.data.vertices
7:         if not vertices:
8:                 return obj.location
           # Calculate the total sum of vertex coordinates
9:         total_co = sum((v.co for v in vertices), Vector())
           # Calculate center of object by averaging the vertex coordinates
10:        center_co = total_co / len(vertices)
           # Transform the center coordinates to world space
11:        center_world = obj.matrix_world @ center_co
12:        return center_world

   # choose index of the chosen input data
13: index = 0
   # select active object
14: obj = bpy.context.object
   # get the median location of mesh
15: bpy.context.object.modifiers["GeometryNodes"]["Socket_7"]=
   get_objcenter(obj)-obj.location
   # load other parameters
16: with open(pathlib.Path.home() / "tmp" / "input.json",'r') as f:
17: j=json.load(f)
18: for item in j[str(index)]:
19:         for parameter in groupInput:
20:                 bpy.context.object.modifiers["GeometryNodes"]
                      [parameter.port] = item[parameter.name]
```

The get_objcenter(obj) function takes the reference to an object and calculates the median point of the parcel which is then subsequently used to move the generated grid to the correct location as described in Algorithm 2. This object is always the selected active object in this script.

4.9 Validity of the results

Using the percentage of the building area the user can check against the allowed values for each lot to ensure the result abides by the rules. The percentage depends on the type of the lot and size of the lot. It is important to take into the consideration the bitmaps from which the buildings are extruded. If very few of the bitmap cells are marked white for extrusion, the percentage of the building area might prove to be too small due to the chosen bitmaps.

Chapter 5

Results

The program was tested on 16 different lots with different shapes and sizes and produces expected results. Testing has yielded acceptable results for all shapes and sizes of a lot.

The main goal of this project was to generate a potential visualisation for building development on a variety of parcels. On Figure 5.1 the results of the procedural generation can be seen and their respective parameters which led to the particular result are listed in Table 5.2 after the figure. For the presentation of the results, with respect to computational complexity, LOD 1 was chosen as it provides a compromise between detail and efficiency.

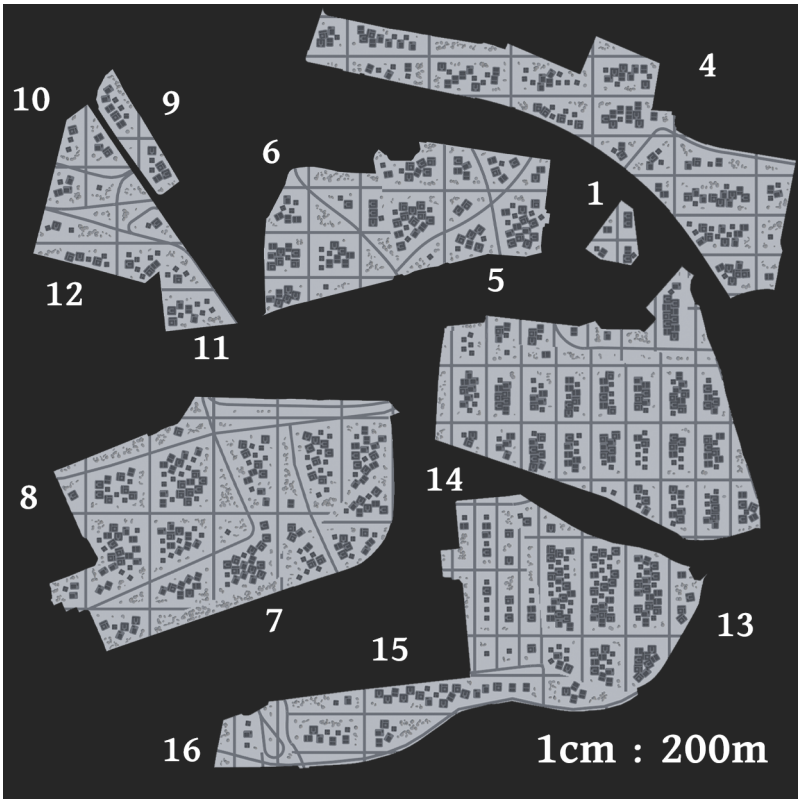

Figure 5.1: Parcels

The table columns are titled with abbreviations to help maintain structure of the table. Below on Table 5.1 is an explanation of the column titles:

N - test case number	MF - Max Floors
SW - StreetWidth	DM - Distance Min
S - Seed	MD - Min Density
LOD - Level of detail	R - Rows
FH - Floor Height	C - Columns

Table 5.1: Legend

N	SW	S	LOD	FH	MF	DM	MD	R	C
1	8	5	1	2,6	18	3	0,75	3	3
2	8	0	1	3	21	9	1	12	2
3	8	3	1	2,6	21	9	1	15	8
4	8	2	1	3	22,5	6	0.75	10	12
5	8	0	1	3	25,5	8	1	3	3
6	8	0	1	3	23,9	6	1	5	6
7	8	0	1	4	24,1	5	0,75	4	3
8	8	1	1	4	25,7	6	0,75	9	5
9	8	0	1	4	25,2	0	1	3	3
10	8	20	1	2,6	25,5	0	1	3	3
11	8	0	1	3,5	26,2	9	0,75	2	3
12	8	5	1	4	22,6	0	0,8	3	3
13	8	2	1	2,6	22,5	3	0,8	8	4
14	8	2	1	2,6	22,5	6	1	8	5
15	8	29	1	3	26,2	5	1	4	5
16	8	-17	1	3,5	26,2	9	0,75	4	2

Table 5.2: Parameters

5.1 Closer Renders

Close up renders of the parcels can be found in this section. These aim to provide a closer look at the details of the renders of some of the most interesting test cases shown on Figure 5.1.

Test Case 4 can be seen on Figures 5.2 and 5.3.

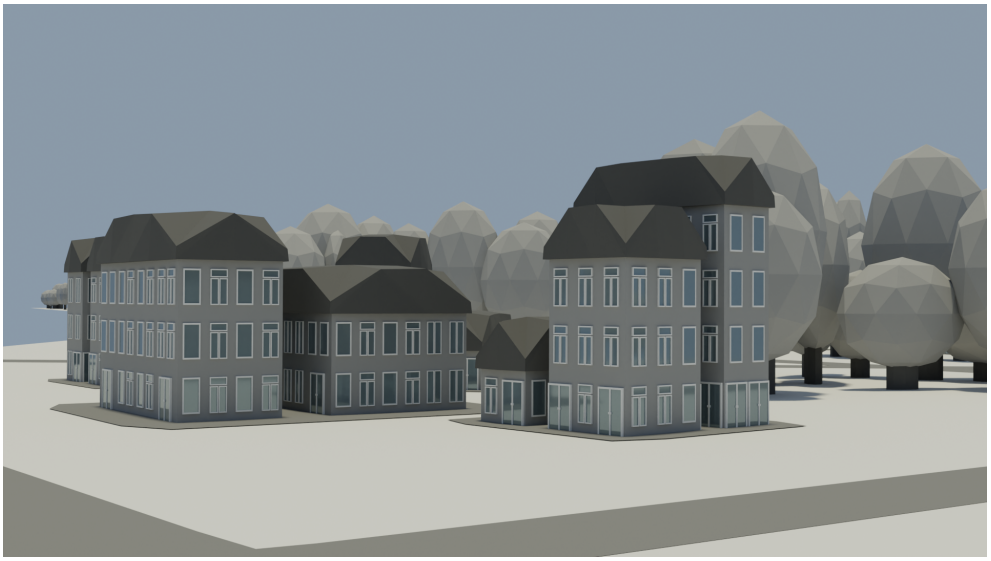

Figure 5.2: Detail 1 of Test Case 4

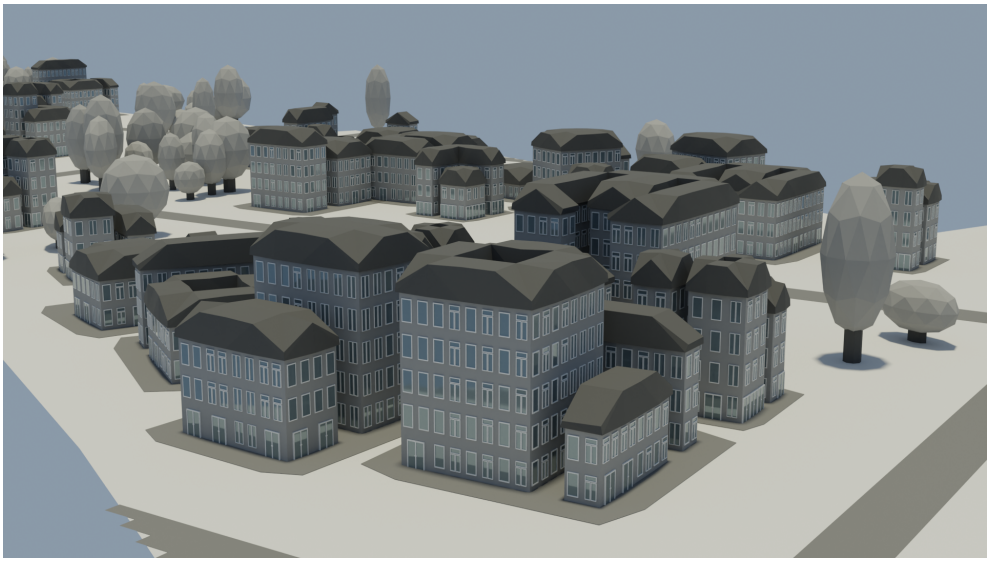

Figure 5.3: Detail 2 of Test Case 4

Test Case 8 can be seen on Figure 5.4.

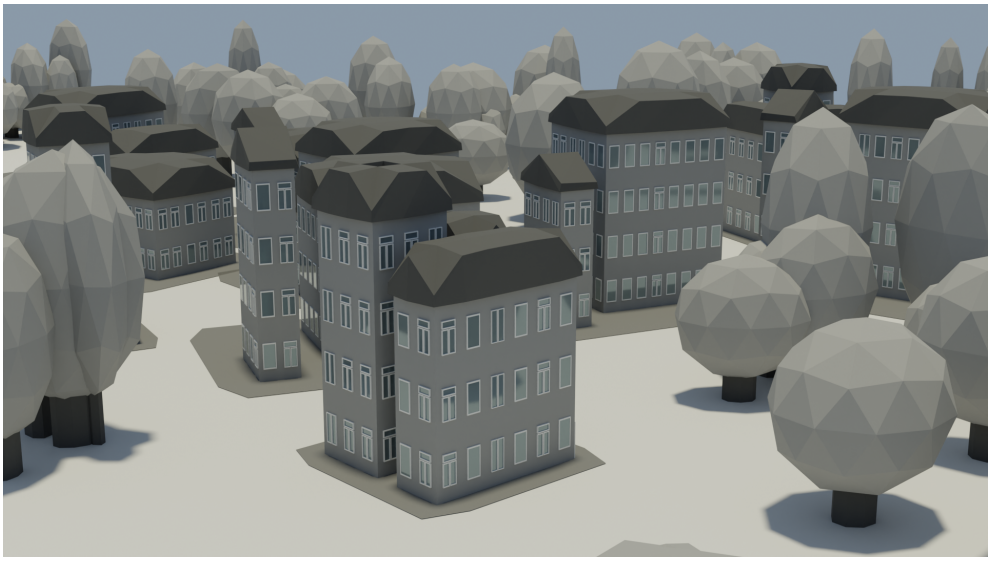

Figure 5.4: Detail of Test Case 8

Test Case 13 can be seen up close on Figure 5.5.

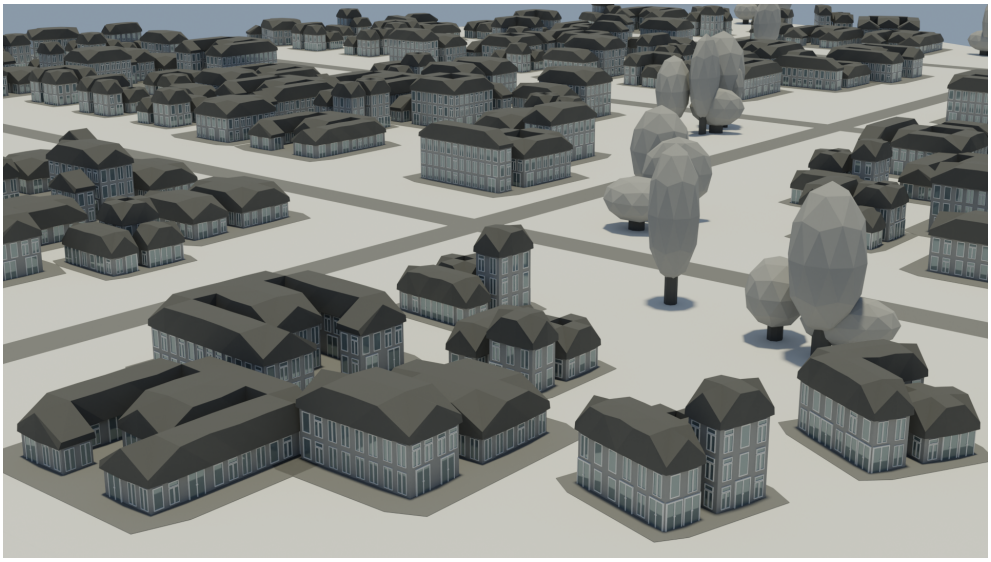

Figure 5.5: Detail of Test Case 13

Test Case 14 can be seen on Figure 5.6.

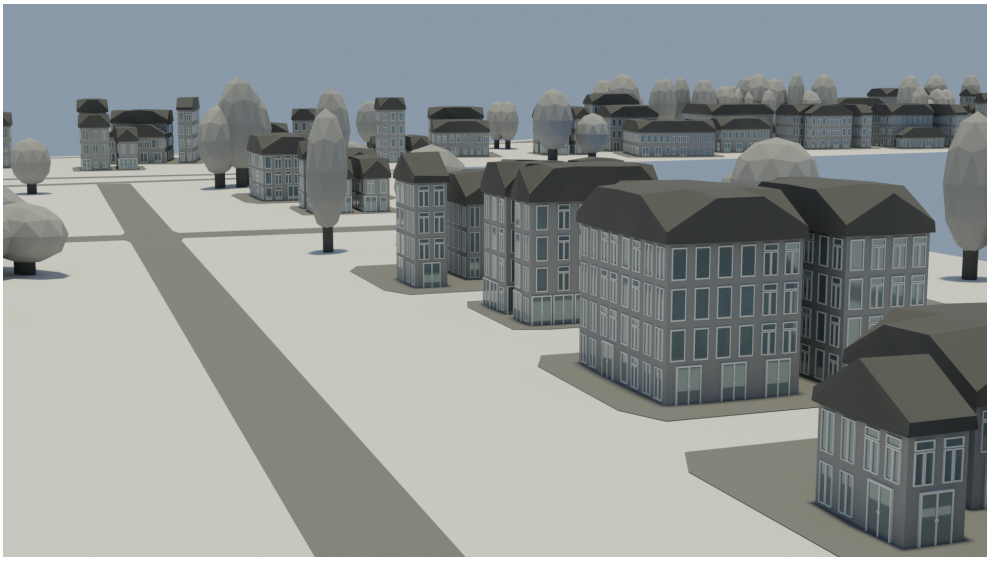

Figure 5.6: Detail of Test Case 14

Test Case 16 can be observed on Figure 5.7.

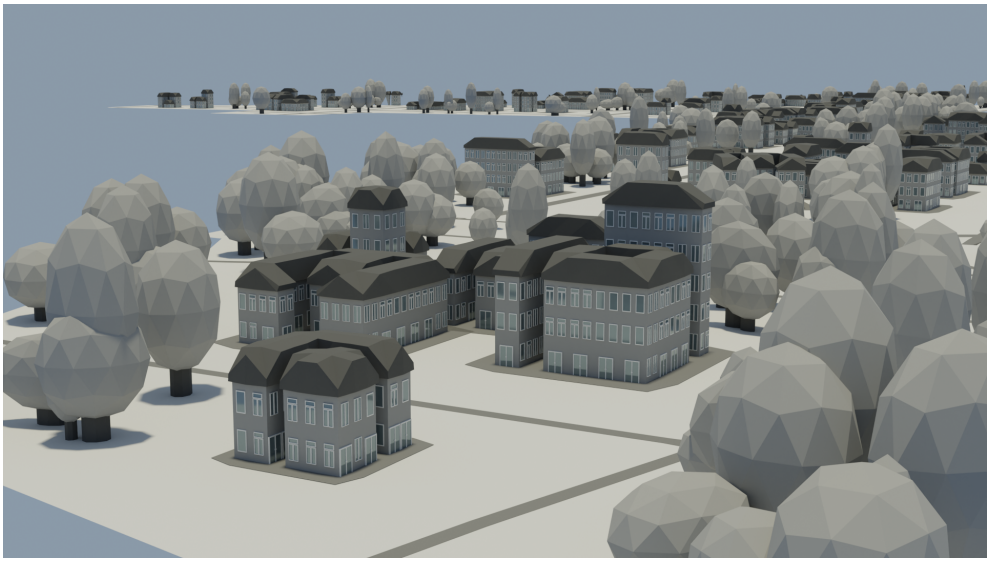

Figure 5.7: Detail of Test Case 16

5.2 Feedback

After presenting my program to the people at IPR, I was provided with valuable feedback. An urbanist architect was also present at the meeting to provide relevant feedback from a different point of view.

The most important note that was mentioned was that for future use, it would have been helpful if the program could generate a table of different attributes relating to the generated model, like for example the square footage of all the combined floors which would subsequently be used for the estimation of possible population. Said estimation would then be used to estimate the amount of parking spots for said parcel or the number of schools needed for said location.

Another note that was mentioned was the possibility of adding a street generator. This generator would create a grid of streets with an adjustable number of columns and rows which would then be used in addition to the drawn in streets as it is quite difficult to draw straight lines by hand.

5.2.1 Adjusting the program according to the feedback

While creating a system for generation of a table of relevant indicators would be too time intensive for the scope of this thesis, a generator for the creation of the street system proved to be an easy addition to the program.

Street Generator

This section adjusts the Algorithm 3. Steps 1-4 and 6 were added. Adjusted road curves creation is described in Algorithm 18:

Algorithm 18 Creating road curves with a street grid:

1: $streetGrid \leftarrow Grid(gridX, gridY, Rows, Columns)$
2: $streetGrid.Transform(Median, (0,0,0), (1,1,1))$
3: $streetGrid = MeshToCurve(streedGrid)$
4: $subdivideCurve(streetGrid, 15)$
5: $CustomRoads \leftarrow SubdivideCurve(customRoads, 5)$
6: $CustomRoads+ = streetGrid$
7: $ExistingRoads \leftarrow SubdivideCurve(MeshToCurve(roads), 5)$
8: $CreateMeshForRoads(CustomRoads, ExistingRoads, Parcel, StreetWidth)$
9: $CreateSpaceAroundRoads(CustomRoads, ExistingRoads, StreetWidth)$

A grid is created with the Size X and Size Y results from the Calculate size of Grid algorithm 1. The number of rows and columns is controlled by the Rows and Columns input parameters. This grid is then turned into a curve and resampled to respond a bit better to the later Raycast selection. Before

being passed into the road creating algorithms it is joined with the geometry from the customRoads object. The rest of the algorithm stays the same.

5.3 Possible Future Improvements

There are several areas where this project could be further developed and polished.

5.3.1 Statistic information

As was detailed in the Feedback section 5.2, it would be valuable to add an option to generate a table of information relating to the generated results. This would mainly focus on the square footage of the building area and subsequent estimation of potential population.

5.3.2 Adding styles for buildings

Adding certain style presets could be valuable to urban architects to imagine different types of styles from a modern urban city to an old historical town.

Chapter 6

Conclusion

In urban development creating visualisations for potential development areas is a time consuming and costly matter. Several different versions of the visualisation have to be produced and shown to citizens to gather public reception and spread awareness about the potential development. There are not many procedural generators suited specifically for visualisations of urban development and the ones which are available have a limit on the amount of customisation that can be done to the model.

The program which is the final product of my thesis aims to provide the user with a wide range of parameters used to customise the final model. It can be used to visualise a range of different types of parcels from an urban district to a village neighbourhood. This is by adjusting parameters which influence the distance between the buildings or the maximum number of floors allowed on the parcel. The generator also provides the user with a slider to influence the level of detail which the model retains.

The program also includes two ways of generating paths on the parcel while taking into account already existing roads. The first option is to hand draw the curves of the roads onto the parcel itself, while the second option generates a grid of streets with a flexible amount of rows and columns. These two options are not mutually exclusive and can be combined to achieve a variety of different results.

The program provides the user with a percentage value of the building area of the visualisation. This can be used to check against the guidelines to see if the visualisation is viable or not.

The goal of this thesis was to create a program that generates a visualisation of a potential urban development. As mentioned before, testing has yielded correct results on numerous different parcels. Due to this the goal of this project was accomplished.

Appendix A

Bibliography

[1] IPR. *Prague Metropolitan Plan (text part and maps)*. URL: https://plan.praha.eu/. (accessed: 19.05.2024).

[2] IPR. *Open data Prague*. URL: https://opendata.praha.eu/. (accessed: 19.05.2024).

[3] Jan Kutálek. "Procedural generation of videogame environments". Bachelor's Thesis. ČVUT FEL.

[4] Alena Mikushina. "Procedural generation of videogame environments". Bachelor's Thesis. ČVUT FEL.

[5] Autodesk. *Procedural generation: Creating infinite algorithmic realities*. URL: https://www.autodesk.com/solutions/procedural-generation. (accessed: 01.01.2024).

[6] Jiří Zemko. "Procedural generation of city models". Bachelor's Thesis. ČVUT FEL.

[7] Kejvalová Jana. "Procedural model generation from real maps". Master's Thesis. ČVUT FEL.

[8] Ondřej Kyzr. "Procedural generation of outdoor scenes". Bachelor's Thesis. ČVUT FEL.

[9] SideFX. *Houdini*. URL: https://www.sidefx.com/products/houdini. (accessed: 01.01.2024).

[10] Blender. *Blender 4.0*. URL: https://www.blender.org/. (accessed: 01.01.2024).

[11] esri. *ArcGIS CityEngine*. URL: https://www.esri.com/en-us/arcgis/products/arcgis-cityengine/overview. (accessed: 19.05.2024).

[12] George Kelly and Hugh McCabe. "Citygen: An Interactive System for Procedural City Generation". In: Nov. 2007.

[13] domlysz. *BlenderGIS*. URL: https://github.com/domlysz/BlenderGIS. (accessed: 03.01.2024).

[14] IPR. *Prague Building Regulations*. URL: https://iprpraha.cz/page/3418. (accessed: 19.05.2024).

[15] Blender. *Introduction*. URL: https://www.autodesk.com/solutions/procedural-generation. (accessed: 04.01.2024).

[16] Blender. *Fields*. URL: https://docs.blender.org/manual/en/4.0/modeling/geometry_nodes/fields.html. (accessed: 01.01.2024).

[17] Blender. *Node Types*. URL: https://docs.blender.org/manual/en/4.0/modeling/geometry_nodes/index.html#node-types. (accessed: 04.01.2024).