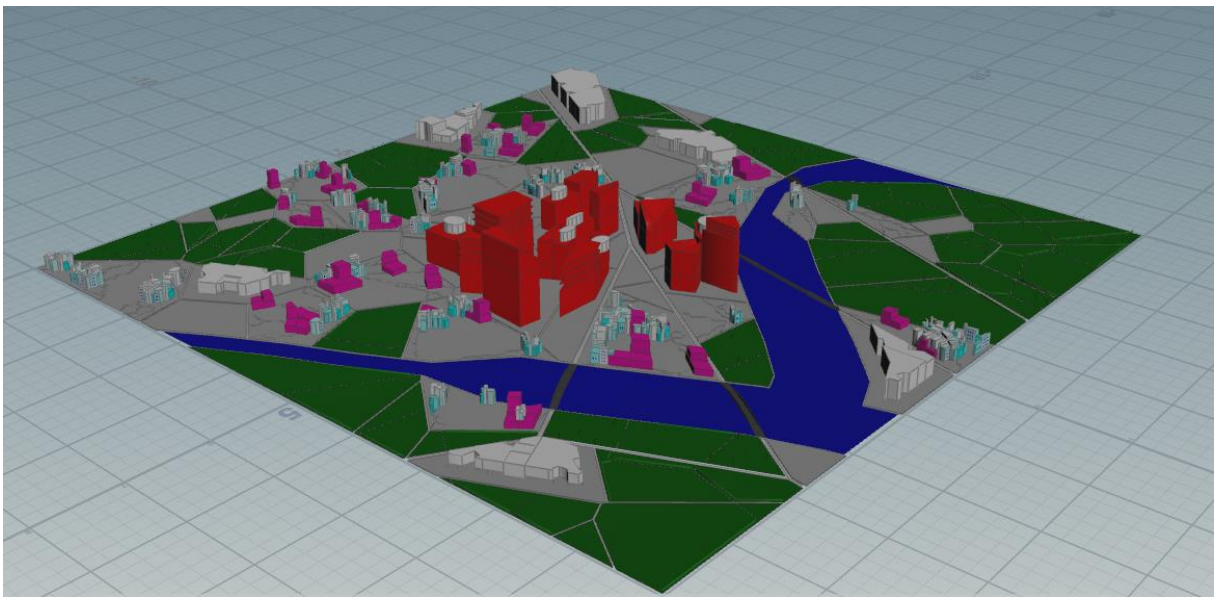
Interactive City Generator for Game Engines

Master's Project Thesis



Alexandra Kim Bui

Bournemouth University

20th September 2021

Abstract

Proceduralism is a widely used term in computer graphics nowadays. For city generation, there is a wide range of techniques used. This work demonstrates the usage of ground plane subdivision for the generation with the generation of procedural extensions like bridges. The setup does not require any inputs. The building process works from scratch. The output is dedicated to the game, film and architectural visualisation projects.

Contents

Abstract 2

1. Introduction 5

2. Previous works 8

City generation using grid layout and geometric primitives 8

CityEngine 9

Procedural city generation using Perlin Noise 10

3. Architectural background 12

The topic 12

Soviet Urban Planning 12

4. Technical background 14

Proceduralism 14

2D procedural methods 14

Texton placement rules 14

Perlin noise 15

5. Implementation 17

Overview 17

Main roads 17

Water areas 18

Water creation rules 18

Secondary roads 19

Regions 19

Buildings 20

City center 20

Residential area 22

Alternative methods 28

Testing 33

6. Conclusion 37

References 40

1. Introduction

City generation is an overused term in the age of rising CG work in fast-paced production. With the high demands of creating different environments, the proceduralism is a typically searched term. Some of the examples might be CG projects like Dr. Strange from Marvel (Figure 2), an Award-Winning film from Christopher Nolan Inception (Figure 1), or a number of games not only for mobile phones (Figure 3). Depending on the nature of the project, every city may vary. In this thesis, there are described processes and methods for creating a minimalist city, including city centres for games or simple visualisations.

This work focuses on creating the interactive minimalistic and futuristic-brutalist type of Russian city, which is inspired by the images and ideas of artists and writers in 1912-1927. The idea of the project is a 3D city generated scene in Houdini software. Houdini is an industry-standard for building procedural tools and effects for artists. For this work, there are inspirations taken from the film and game reviews, the previous works of Bournemouth University students and from the other researchers cited in the references. This project begins with the modeling from scratch and it ends with the packed digital asset containing untextured buildings and a basic city layout in a .hda file.

Since there are many city generators already existing, this project focuses mainly on these aspects:

Polycount efficiency: Generating cities can be computationally heavy, mainly if buildings have detailed geometry. Since this project was created for the Unreal Engine platform, the polycount is crucial for performance. The bigger polycount the geometry has, the slower computational time is needed.

Naturalism: It has always been a big task to find a balance point between computer-generated and the naturally generated environments. Well-made cities respect the public infrastructure, and with that in mind, the generated variations should respect the terrain.

Interactivity: the easiest systems are considered as the most user-friendly systems. Complexity creates the illusion of the possibility of having a wider range of choices but building a truly strong system means to create a structure of private tasks which holds the generator and the “basic” parameters (with some meaningful extensions) are promoted to the user.

This project aims to find a relatively acceptable polycount for the city generation and find the amount of simplicity or complexity in the used models (mainly buildings). Furthermore, this research will try to find the balance of dividing the polycount into different areas of the city, so the models keep their

compactness on the one hand and having that sufficient polycount on the other hand, so the models have some concrete and noticeable shapes.

This project is structured into five main parts. Firstly, section 2 is dedicated to some previous works already done on the same or related topic. Then in section 3 is a brief introduction to architectural methods for city planning. In the technical background, section number 4 summarises used or related to the city generation methods. Implementation section 5 is the biggest and describes the process of the city creation. Finally, the results and final discussion can be found in section 6. References and Appendices follow at the end of this document.

This project allows extending the further research in terms of application of the materials or connecting to the game engines, preferably Unreal Engine for which this tool was primarily built.

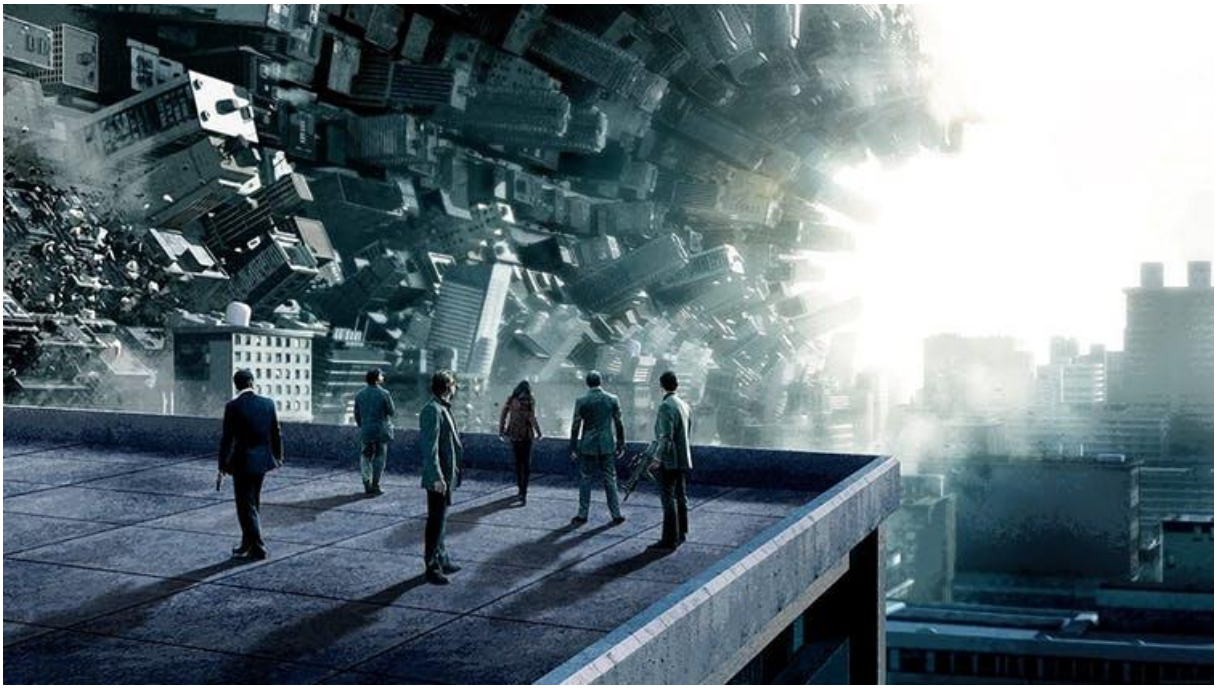


Figure 1 - Bending City in movie Inception

Figure 2 - Mirror dimension in movie Dr. Strange

Figure 3 - City Island, an Android game

2. Previous works

City generation using grid layout and geometric primitives

There are several already existing city generators. George Kelly and High McCabe have collected existing methods from 2006 and published the in the ITB Journal. The process and the outcome are reviewed and compared in terms of the production and realization aspects.

Setup from (Stefan Greuter, Nigel Stewart, Jeremy Parker, Geoff Leach, 2003) is created on a grid layout divided into cubic regions. The author was inspired by the New York City to create the generator the city was applied in the virtual city called Undiscovered City and then improved in later project Neverland demo. (George Kelly, Hugh McCabe, 2006)

Figure 4 - City Generator from Stefan Greuter et. al.

Figure 5 - City Generator from Stefan Greuter et. al.

The buildings are created by fusing randomized basic geometric shapes, which lays out the main floor plan for every building. These polygons are then extruded on every extrusion step where is one shape left on every level. The process is shown in the Figure 6.

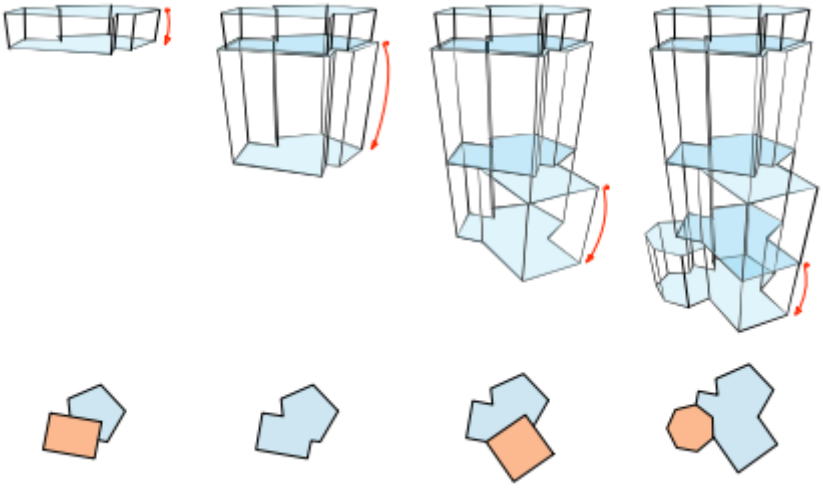


Figure 6 - Building generation from the floor plans, Stefan Greuter et. al.

The grid layout causes the city to appear to be artificial and straightforward. The setup is fast and efficient in performance. However, there are minimal variating options reported in the discussion section. (George Kelly, Hugh McCabe, 2006), pg. 109.

CityEngine

On the market, there is a number of existing generators for the company uses. CityEngine is one of their leading representatives, which was firstly released in 2008. CityEngine has come through a long way of changes. Today CityEngine is a separate application used for architectural visualizations, film productions or game engines.

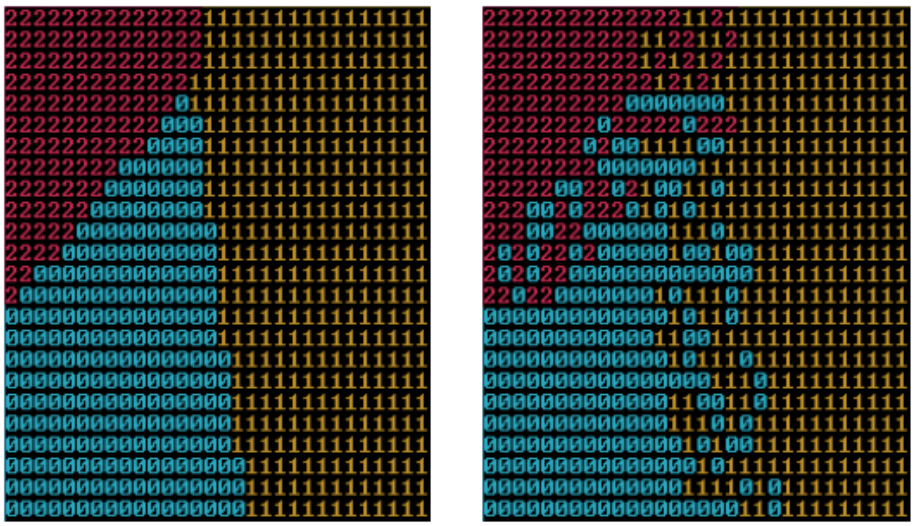
The roads are generated using L-Systems paths. L-Systems were initially designed and developed by biologist Aristid Lindenmayer for a bacterial study. (Lindenmayer, 1987) The roads are generated using image maps (which can be methodically comparable to the drawn paths). Extended L-Systems are also used for building generation.

The system works fairly well, and the visual has a realistic look in terms of the building generation and its distribution. On the other hand, CityEngine takes a bigger number of inputs. (George Kelly, Hugh McCabe, 2006)

Figure 7 - CityEngine

Procedural city generation using Perlin Noise

Niclas Ollson and Elias Frank created the city generator based on the Perlin noise value. Ollson uses a different approach in the steps for creating the city. Instead of beginning from the roads, he creates the separated regions, which are overlayed from both sides calculated from the value of competition range where the higher amount assigns the area. The distribution is located using Perlin noise. Next, the roads are created using square dimensions. Buildings are then created as the last step when the distribution avoids areas with borders for roads. (Olsson, 2017)



(a) *District* generation. (b) Border competition.

Figure 8 - Niclas Procedural city regions

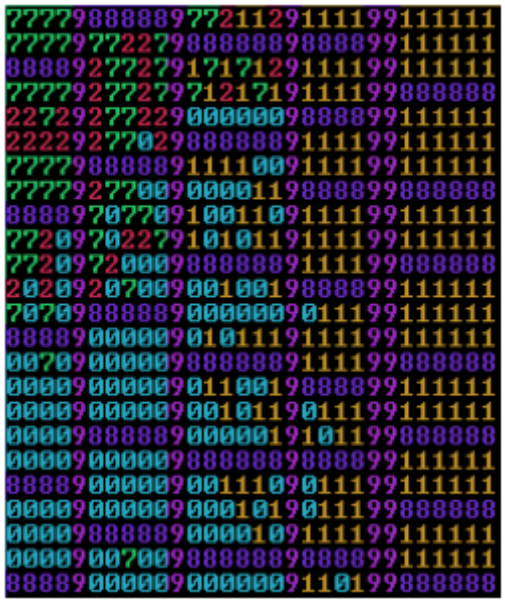


Figure 9 - Final image of the generated city

3. Architectural background

The topic

Futurism is one of the revolutionary artistic directions of the 20th century, which refuses the past, old culture, religions, museums and highlights technology, dynamism and speed. (Mitul, 2019) These aspects became a part of Russian political ideas. By combining the futuristic thoughts with the Russian greatness of Moscow and other big cities, Russia started to build many monumental and modern buildings. One of the subsequent and favourite directions is brutalism from the 50s, which is built mainly on concrete which is a richly used material in Russia. The buildings are rough, brutal and combining the glass and small windows, and the buildings give the Russian cities its typical visual and feeling of accuracy, respect, discipline and power.

Soviet Urban Planning

Many reparations and new architectural plans had to be remade from the damages caused by the constant wars during the 1st half of the 20th century. Miliutin was a significant planner who created first modern ideas of layering the cities with some Western architectural inspiration. His proposal to the Greater Moscow Project was created with a circular area with the city centre in the middle, with no residential areas. Around the city centre, there are four strips of parks between which are buildings distributed. Today, Moscow city leads some of the Underground lines radially around the city centre and in the further area. (Miliutin, 1975)

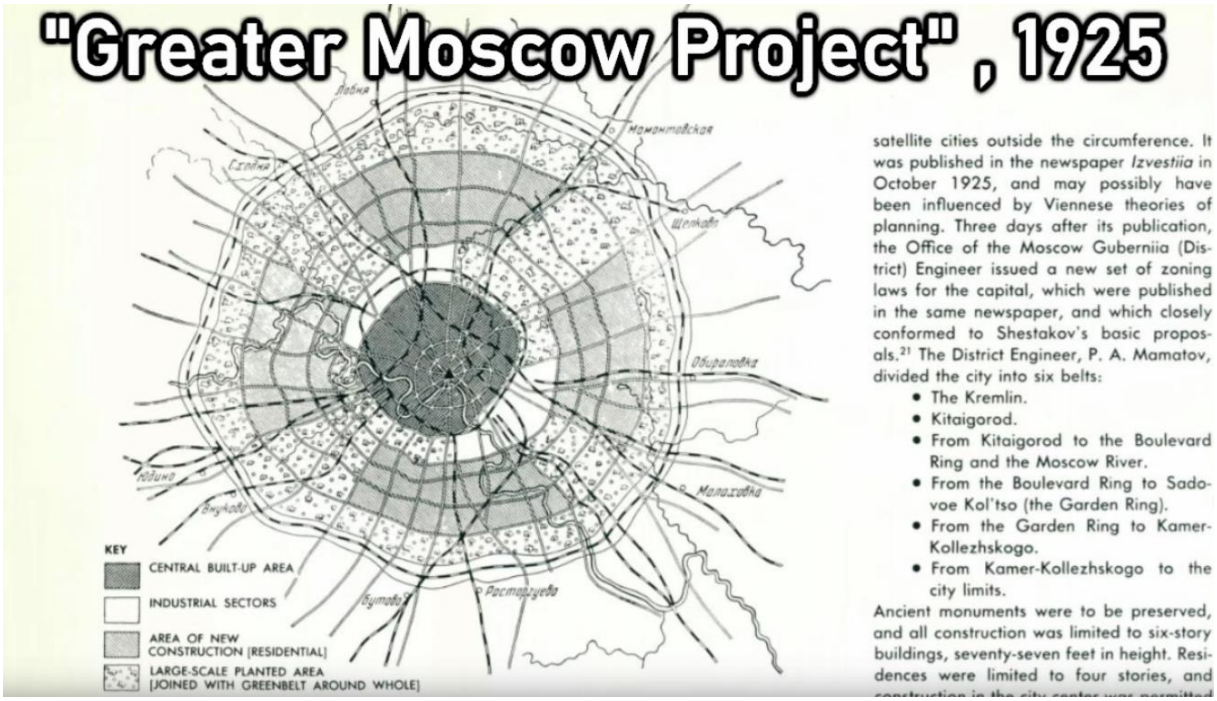


Figure 10 - Greater Moscow Project

A modern approach to the city alignments dates back to 1930 when the plan was created using linear and regular paths. That is how Linear Industrial cities are made. However, most of the projects still exist unchanged, like the GAZ car factory in Nizhny Novgorod.

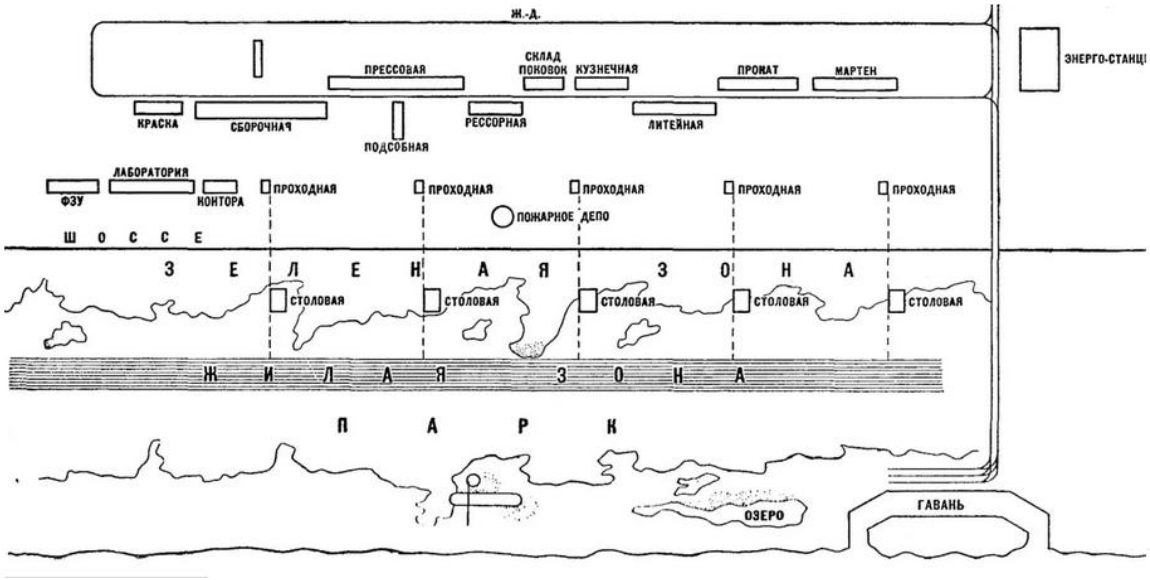


Figure 11 - Nizhny Novgorod car factory plan from 1930

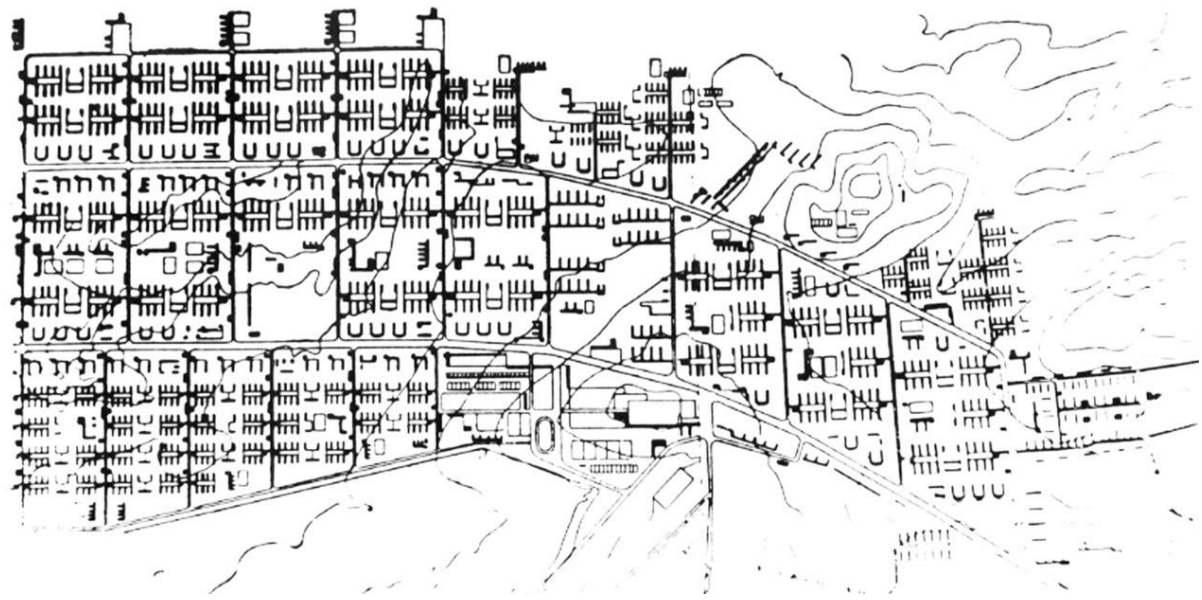


Figure 12 - Linear plan for Magnitogorsk from Ernst May, Miliutin's successor

4. Technical background

Proceduralism

Generating numerous complex units of buildings, crowds, or any environment elements can become time-consuming, and with the high demand of repeating modeling tasks and creating a controllable simulation, proceduralism becomes a key for efficient work not only for games and film studios.

Proceduralism was originally used for generating natural textures like wood, stone or other natural materials. To be created procedurally simply means that it is created with a set of following algorithms which are repeated and can be recreated for the same or different inputs. Proceduralism gives certain flexibility in the creation. The set-up can change parameters in the process of building the model – it makes it non-destructive. (David S. Ebert, 2003)

2D procedural methods

Texton placement rules

In the list of the procedural methods in the Survey of Procedural Methods for Two-Dimensional Texture Generation (Dong, 2020) the authors described 2d texture creation of the natural patterns like celular noise or woods or regular patterns like cloth etc. One of the methods is texton placement, where a texton is a representant variable of a geometric shape. Texton placement can be distributed by several different rules, and there is a summary of some of them:

Regular grid

This is the simplest method of texton placement. A grid is created by dividing a working area into regular uniform cubes where the textons, in this case ellipsoids are placed into each cube regularly. Different variations are created by changing a, b, c parameters.

Random grid

This method uses the regular grid and extends it with the randomness in x and y in each point.

Random walk

In this situation, the textons are distributed in random order. From the initialized point x1, y1 is a placement created and by following a random vector the new place is initialized to x1, y1. The process is repeated as the user defines.

A probability map uses a fractal height map as a base for calculating the probability of texton placement. If the pixel on the heightmap has a greater value than the threshold, then place the texton and repeat the process.

Texton combination

By combining different variations and rules of texton placements, different visual results can be achieved. There is a maximum rule and additive rule, where the maximum rule places textons and uses the maximum value as a set value of the pixel. The additive value uses the same principle, but it is extended into 3D.

Perlin noise

Perlin noise was created by Ken Perlin in 1982 and then improved into the simplex noise in the report Improving Noise from 2002. Originally the noise pattern was created for a project called Tron (Legacy). Perlin noise is a kind of gradient noise that non-linearly interpolates between values 0 and 1. The numerous points are spaced regularly with the smooth blending between them. In the two-dimensional creation there are 2D gradient planes distributed which are by vectors computed to create a noise by leading vectors shown in Figure 13. The colour value is calculated by creating a dot product between gradient vectors and grid points.

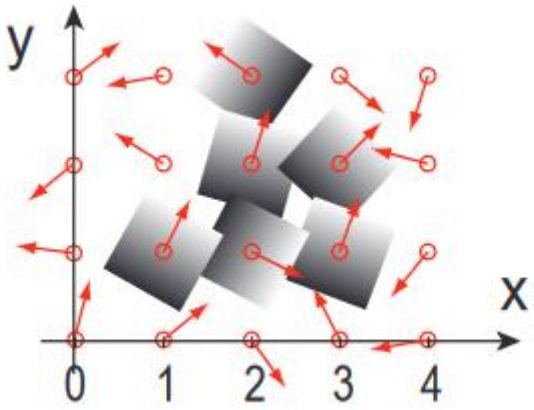


Figure 13 - Perlin noise creation

Figure 14 - Perlin Noise

5. Implementation

Overview

The whole system is divided into four main parts: main roads, secondary roads, regions and water areas. Every part has a paragraph describing the process, algorithms and techniques of the creation.

The chosen software for implementation was Houdini from SideFX in which are tools that allows user to create procedural content easily, and it is very clear to understand the process of work in the software. The city is generated on the basic “grid” (in other software might be called “plane surface”) of size 100x100 units (units might have user-defined size, but generally it is 1 meter).

Main roads

The main city separator uses the Voronoi fracturing method. The city is divided into main regions without any need of separating polygons into denser nets. Voronoi polygons create broken clusters which are generated by seeds scattered from the main grid. The clusters create the road lines and the main regions. By keeping the same point scattering value, the clusters keep the same sizes on average for any seeding variation.

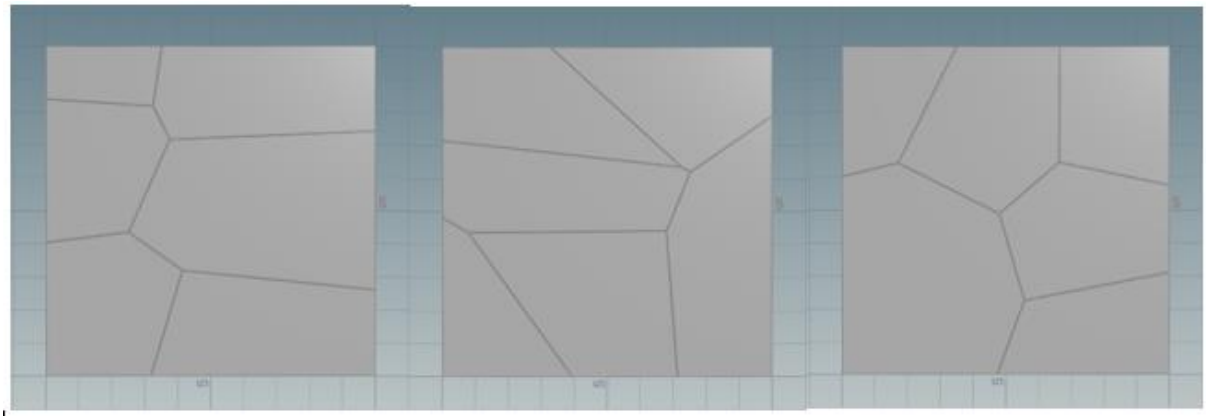


Figure 15 - Main roads variations

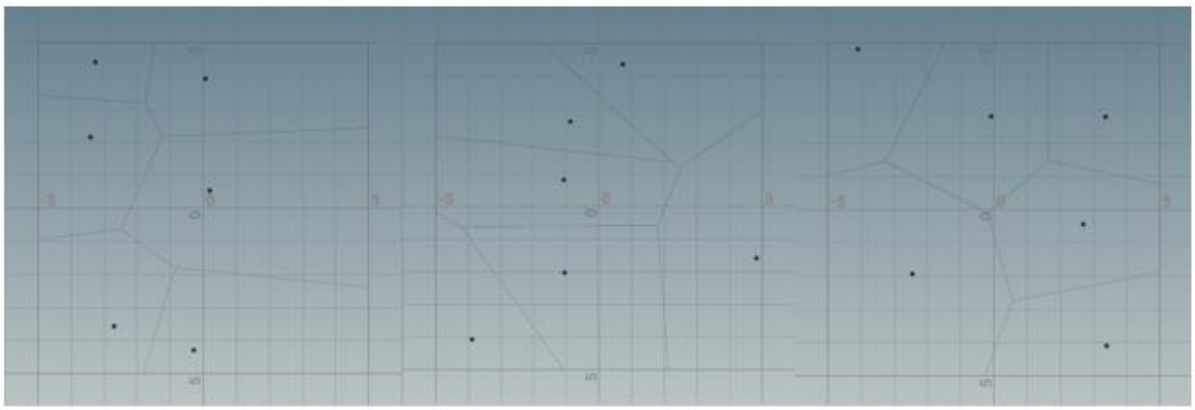


Figure 16 - Main roads point clusters

Water areas

Cities respect the water areas like lakes or rivers. In this project, the water path is hand drawn, which is the only non-generated element since the pathways may differ for every generated highway. Water areas can be created based on the water creation rules.

Water creation rules

1. Water lines must avoid crossroads, which means that no point connecting four lines can be a part of the water line.

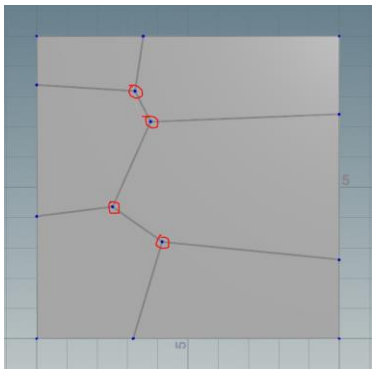


Figure 17 - Road crossing points

2. The water lining tool is not a colouring tool – it is possible to create a lake using simple movements. The water recognition works based on the Boolean operations, which might be imperfect for too complex lines.

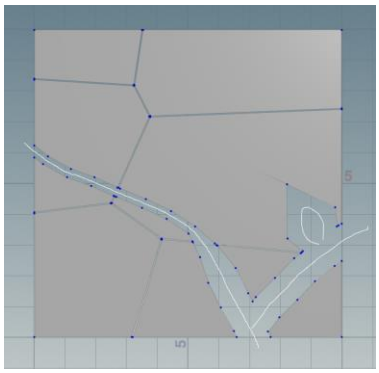


Figure 18 - Line representing water paths

3. The water tool must be lead from the outside areas of the city. The line cannot start or end in the area of city.

Secondary roads

The algorithm for the secondary roads is similar to the first one. With the difference that secondary roads loops respect the content size in comparison with the number of clusters. The reason to create loops is to create some randomness in the line disconnection between the main regions. To this stage, the generator created about 100 polygons, and with extruded border lines for the roads, it is about 500 polygons.

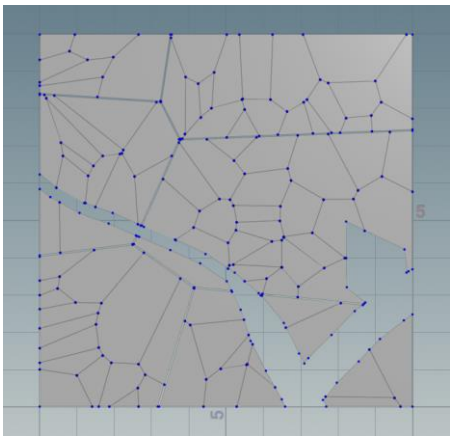


Figure 19 - Secondary Roads

Regions

The regions can be easy divided into architectural parts. This project considers the city centre (red), administrative buildings (brown), residential areas (blue) and flora (green).

Based on Miliutin's architectural planning the city center should contain the historical part of the city and offices. In this area are no residential buildings. The residential area begins in the surrounding of

the city and ends on the borders of the city. It is covered alternately with flora and administrative buildings. The residential areas are mainly the buildings where people can live, and on the other hand, administrative buildings can hold any meaning from the office, shopping centre, palace, etc.

The colours are spread randomly at the first step, and following the DLA technique, the city centre is placed and the residential area re-located around the centre.

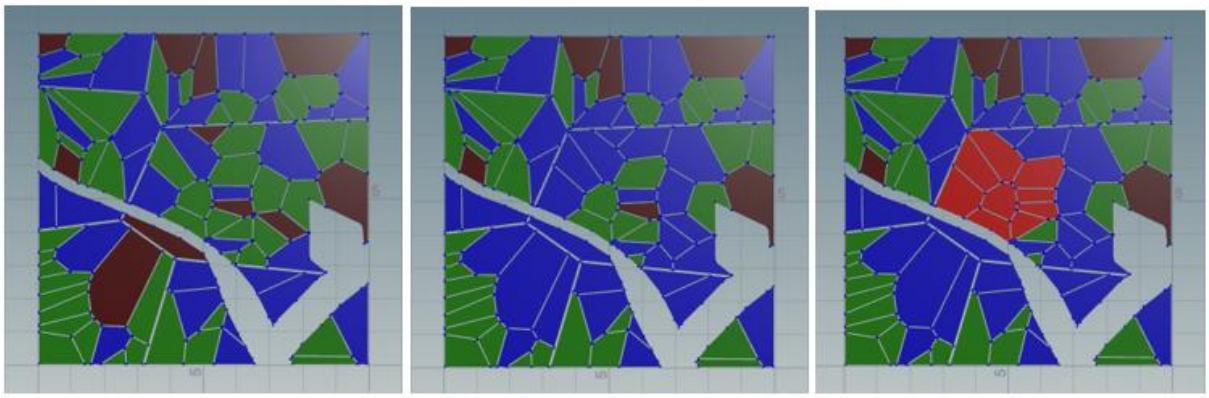


Figure 20 - Assigning different city parts

Buildings

City center

Every segment is looped and fractured by using Voronoi again. Then there is a percentage of randomly selected pieces that is extruded into the buildings after deduction of small pieces. Houdini uses the zscale attribute to control the height. For most of the buildings, zscale attribute remains on random height with the addition of +1 unit (in a case when zscale equals 0, the 0 height is undesirable).

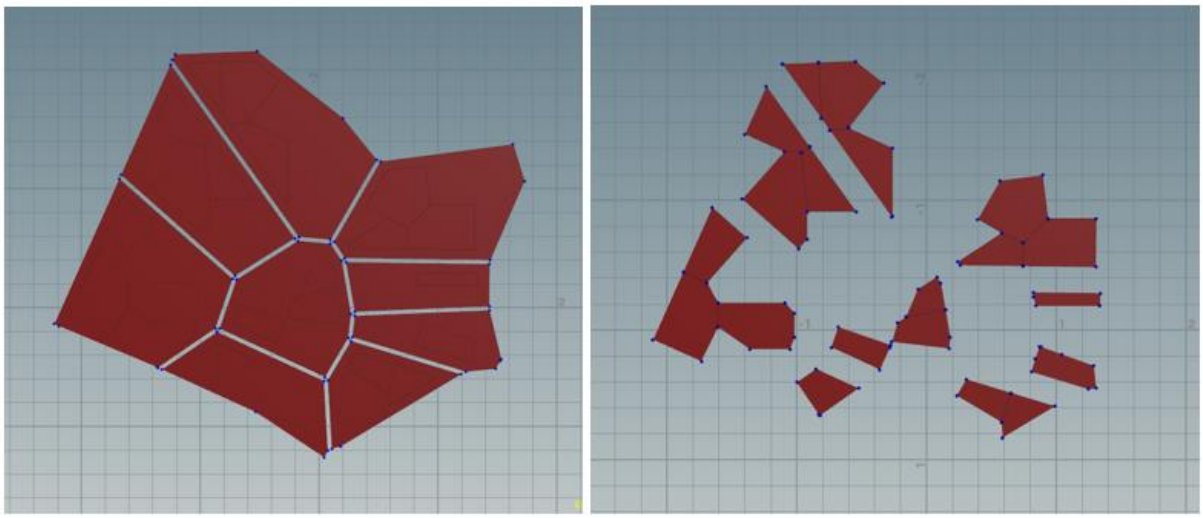


Figure 21 - City center region

Variations

City centre blocks have three variations created by sorting the polygon numbers randomly and then divided into three separated polygon groups: a skyscraper with cone, a skyscraper with peak and a skyscraper with the extruded second level. It is possible to offset the variations however, following the interactivity goal from the document objective, this parameter remains only in the system itself (as well as for the other unconnected seeding of the building variations). The randomness is also working in seeding the whole city, which is more critical for this project than seeding different variations in another variation.

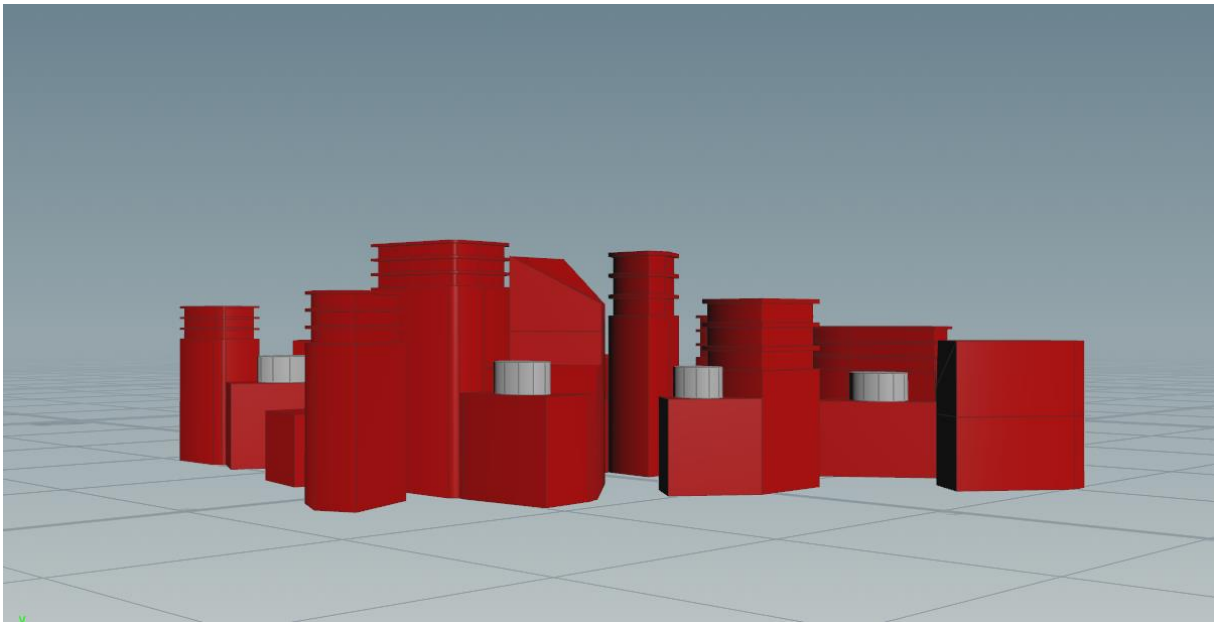


Figure 22 - Finished city center region

Skyscraper with cone

In the centre of gravity of the roof polygons, there is an anchor point placed. For each roof, polygon is a bounding box created which bounds the borders of X and Z of the roof. Then the size is calculated by uniform scaling to the bounding region and scaled down to half of the size. However, this technique

might not be 100% accurate since the calculation is happening on the global XZ axis. Some of the cones are still slightly deflected from the roof area.

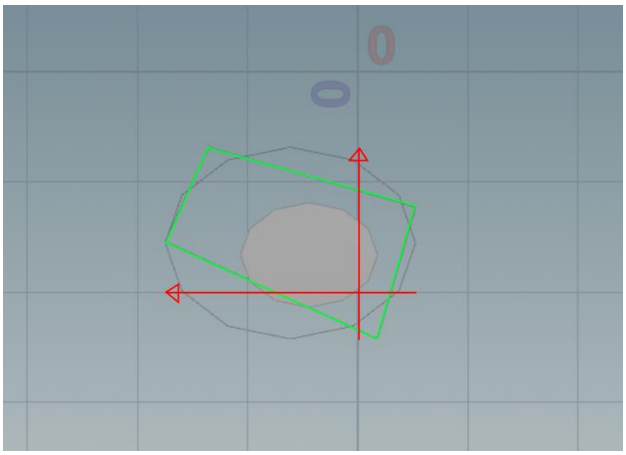


Figure 23 - calculating cone placement

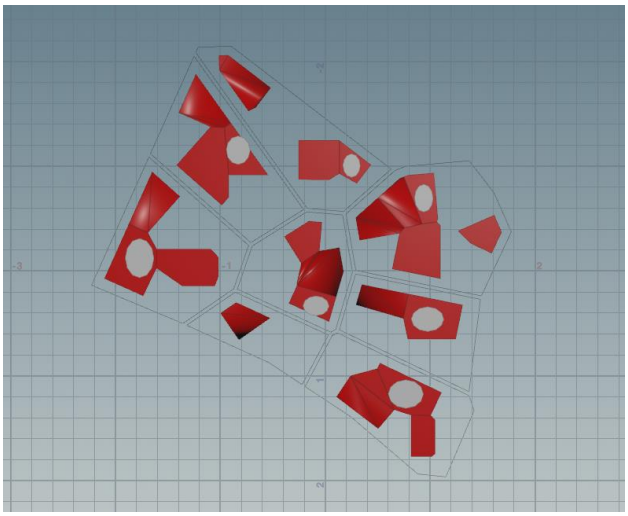


Figure 24 - Cone placement

Residential area

To create more diverse city, there are two parts of differently created buildings which are then merged together. One creates the building using walk-around technique similar to DLA and second extrudes buildings randomly. That way there can be both small single buildings and some connected building complexes.

Walk-around buildings

Algorithm

1. Initialize given number of points (in this case hard-coded 40 points) randomly placed on the surface with the minimum distance from each other. Set the distance D which should every walker walk from the initial point.

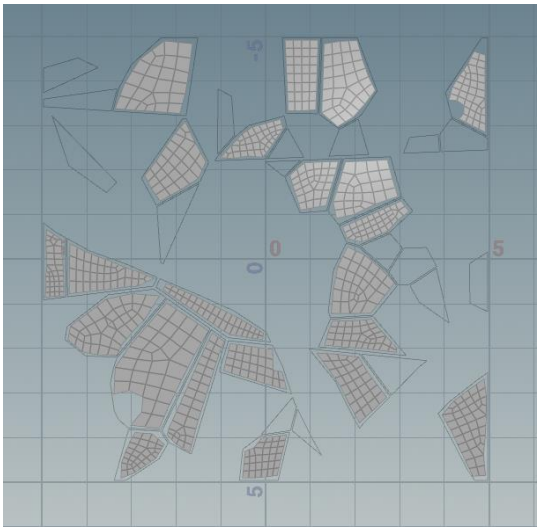


Figure 25 - Walking board

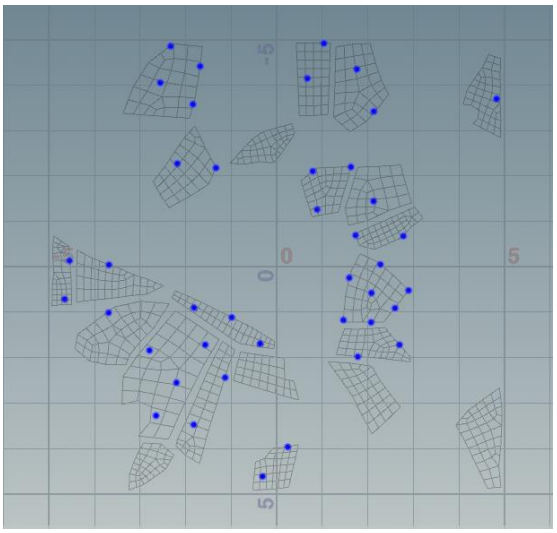


Figure 26 - Initialized walkers

2. Loop start: Is the polygon where the point is located displayed? If yes, display the polygon. If not rest on the position.

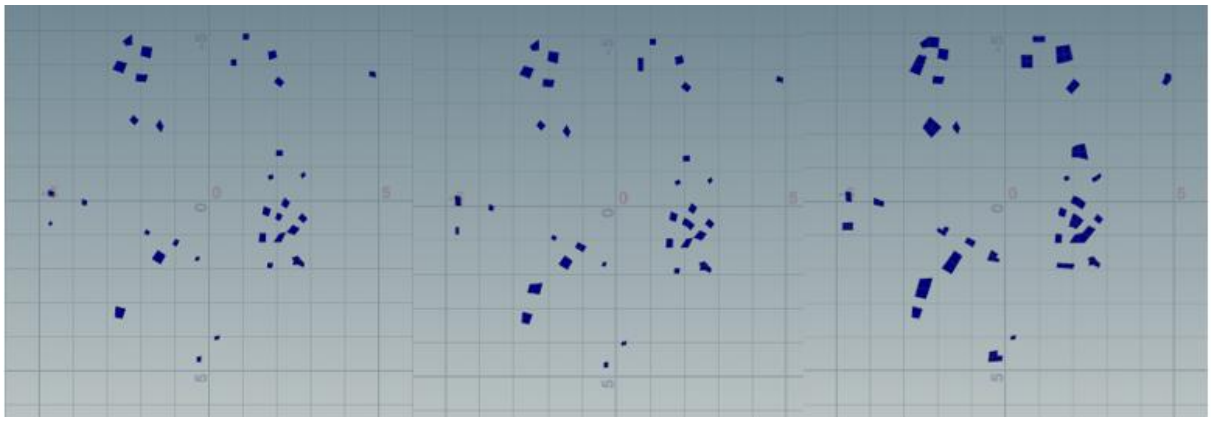


Figure 27 - Growing area from the walker

3. Walk a radius unit from the point.
4. Is there an unshown polygon? If yes display it.
5. Has the walker reached the walking distance D? If yes end the process, if not go to the step 2.

Every second or third polygon is selected to give a second-floor value in a group attribute.

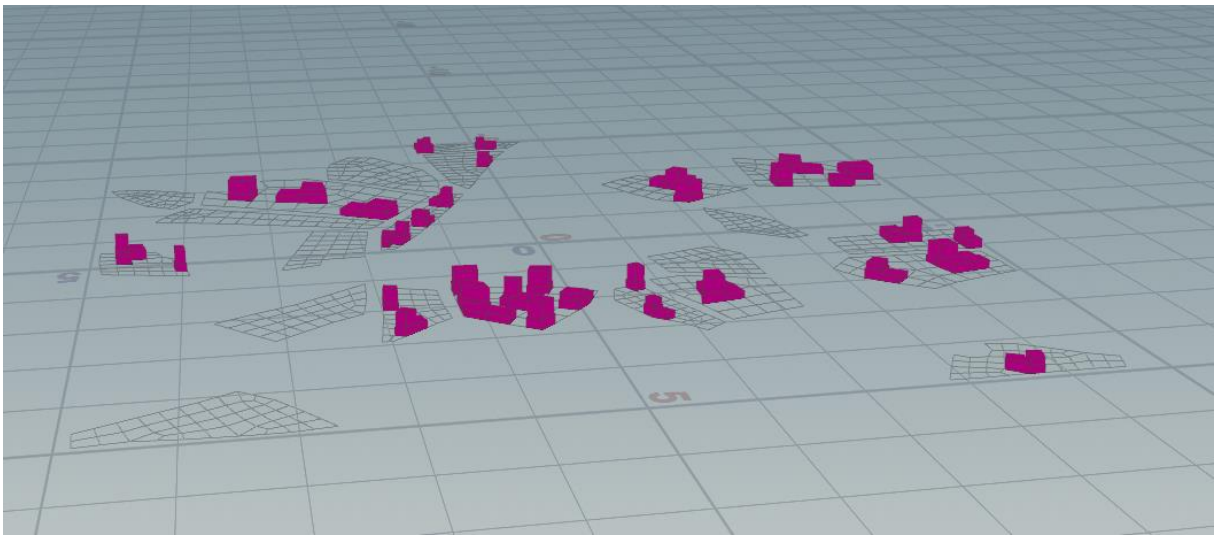


Figure 28 - Extruded walk-around buildings

Randomly extruded buildings

Since the first Walk-around buildings are already using almost equilateral dimensions, the second part of building generation is focused to having more diverse shapes and sizes. The clusters are lead in the middle section of the edges. The Voronoi pattern creates double points on some edges which gives buildings the sense of irregularity.

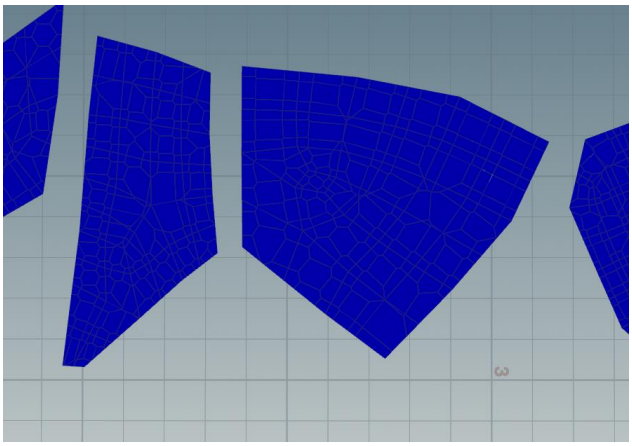


Figure 29 - Fractured Building regions

To avoid tiny buildings, the smallest polygons and degenerated points are deleted. Then, with the private parameter of percentage it is connected to both randomly extruded buildings and walk-around buildings, the buildings are selected from the area and the rest of the polygons are dissolved.

For better precision in locating the windows, the remaining polygons are divided into three groups in a height attribute H with the values of 0.0, 0.5 and 1.0, which is used for extruding the wall edges of the polygons.

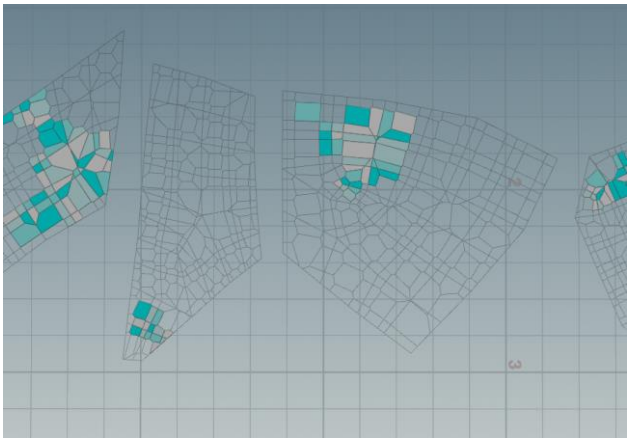


Figure 30 - Final regions for buildings

Measuring boundary edges of the remaining pieces, the windows are located only on the edges of the given attribute of the minimal length “restlength”. If restlength reaches the given amount of R, the point is created in the between starting and ending point. Following the growing amount of restlength the x number of points can be created.

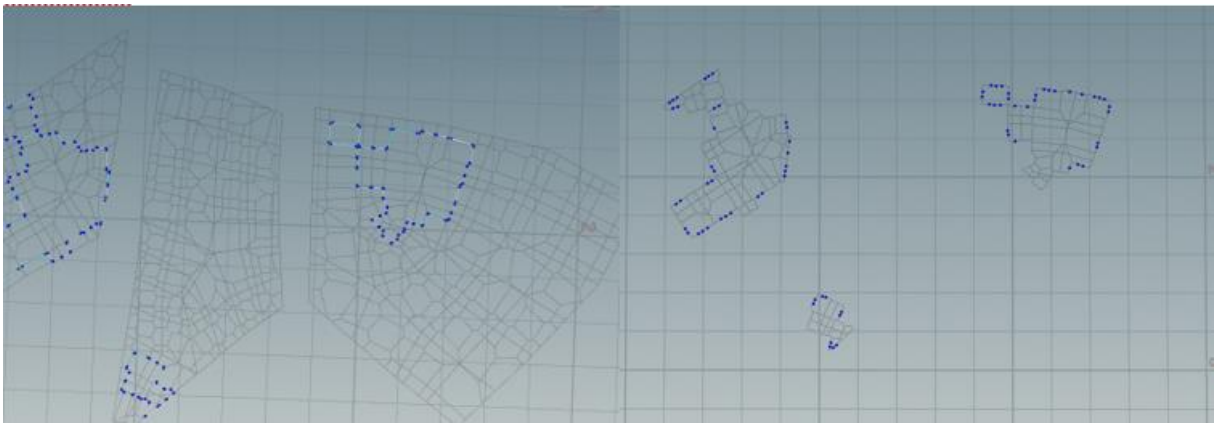


Figure 31 - resampled lines for windows

Windows

Every point then uses height H to create the height extrusion and height of the window spread. Using that technique there may be windows on the other buildings like in the city centre or in the administration buildings.

Windows and trees are only instances for three main reasons:

1. Packed geometry creates a one polygon and 1 point from every packed object. That saves a big amount of calculation not only for Houdini but also for future platforms.
2. The windows have much the same visual, shape and colour for this project. That way, it is unnecessary to calculate every object again, and it is useful to create only the instances of one window.
3. Since the window is represented as a point in terms of location, Boolean operation between walk-around buildings and random generated buildings does not cut the windows' parts. They are kept whole or deleted overall, so there are no cut windows in the ends of the walls.

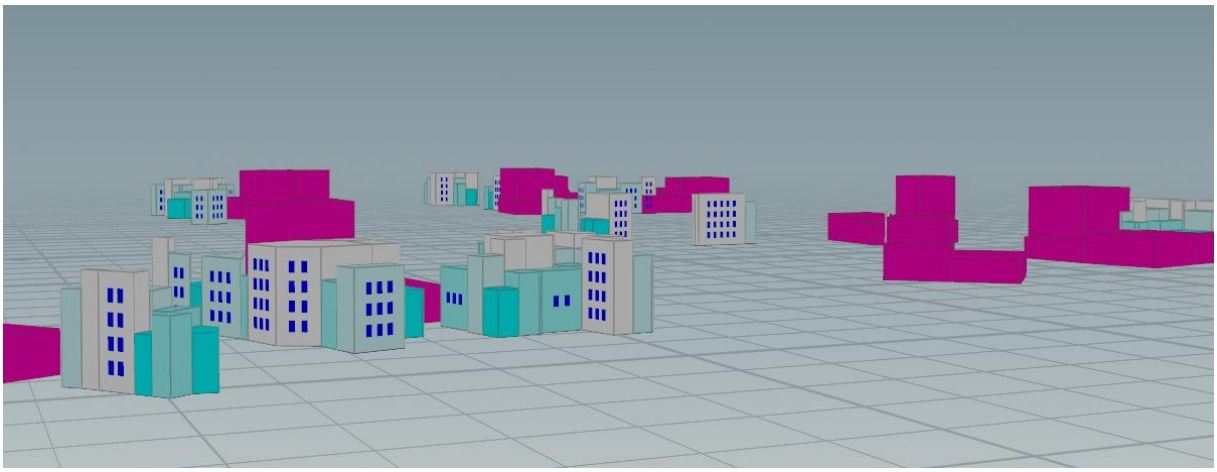


Figure 32 - Final residental buildings

Administrative buildings

Administrative buildings cover the whole surface of a given space with the exclusion of the pathways. The generation is based on the recursive subdivision where:

The polygons can be separated into a grid or remain in one piece. The number of pieces is driven by the changeable value of S and iteration number I.

The polygons on the grid are separated into two pieces on axis X with the boundary points of the polygon grid dimensions G.x/S, G.z/S. By subtracting by a random float value bigger than 0 but smaller than one the lines are located in the polygon area (where each line uses a different seed number).

Add 1 to the iteration I'. If the I' is not equal to I, go back to step 2.

Randomly delete some lines in the whole grid (or alternatively, it is possible to use cluster points for the randomize selection). With that, we get a "Tetris" alike pattern.

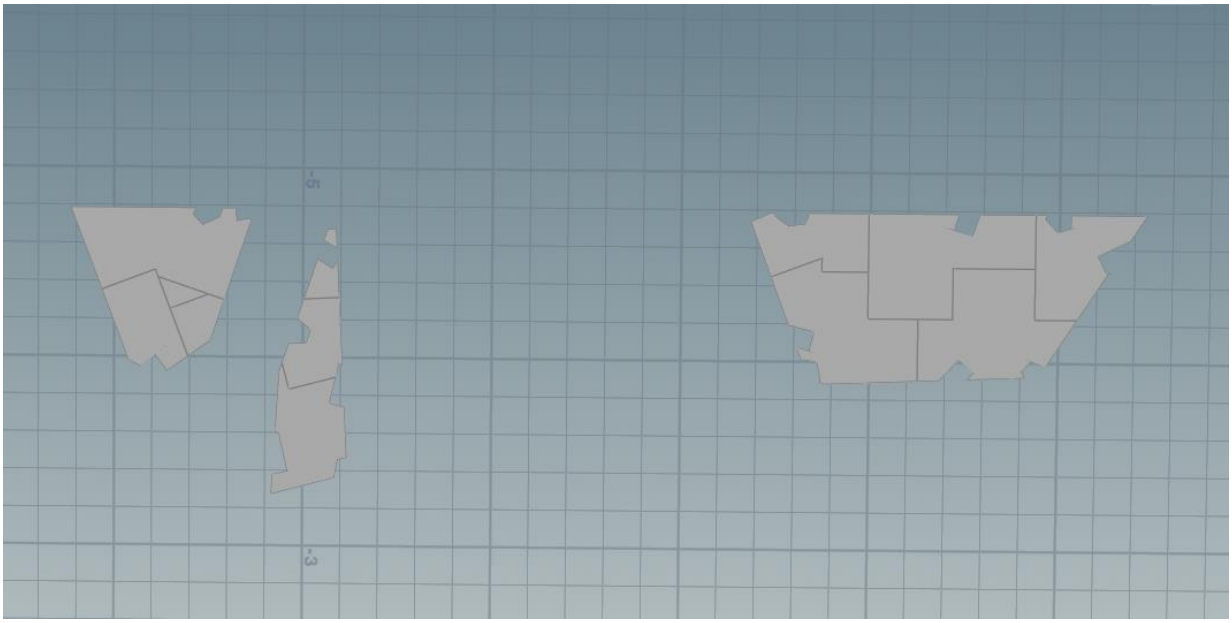


Figure 33 - Administrative building cuts

The reason why is a Labs subdivision used instead of traditional Voronoi clustering is that the Labs subdivision tool recognizes the angle of the border lines. With that, the lines are created not only on the global XZ axis.

Pieces are then distributed into three groups. 1st group are procedurally modeled into halls held by columns. The columns are distributed along the outside edge using Boolean operations with the inner part of the buildings. 2nd group and 3rd group are assigned with the different zscale, which are then extruded into random heights.

Alternative methods

Building generation is a lively discussed topic, and there is a wide range of possible sighs and views to creation. Many found generators are not using the ground plans as a generation base. A number of projects generate the buildings from scratch and then scatter them on the terrain.

Pros

These systems do not need to separate the regions into small pieces to create a building grid. Instead, they use scattered points on the terrain that do not modify the geometry which are saving some computing.

Cons

The points are not aware of what is going to be instanced on them. Scattering methods might suffer from inaccuracy of region border and road alignments. Using the random orientation to eliminate regularity, some buildings might seem unnaturally oriented with respect to surroundings.

Next, the scattered object is not aware of the boundary regions. That means that scattered points do not take the dimensions of the building so that the buildings may be overlapping.

The scattering method might cause some size issues. Scattered instances are imported externally or created on the scene. These sizes are not aware of the sizes of the same objects on different buildings. That might cause different sizes of doors, different sizes of windows etc.

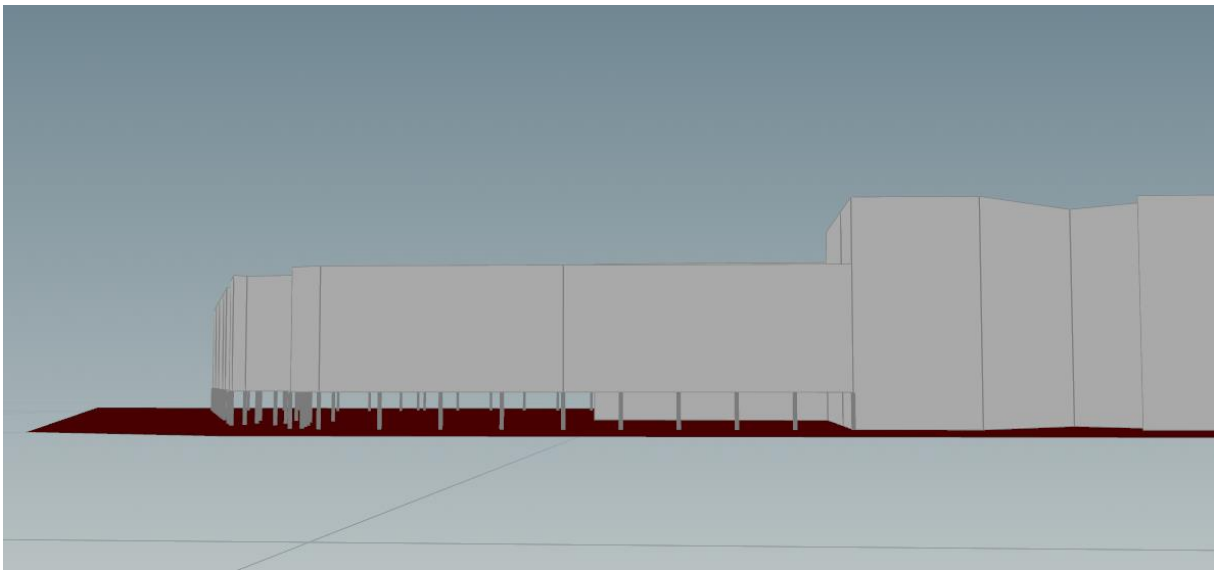


Figure 34 - Administrative buildings front

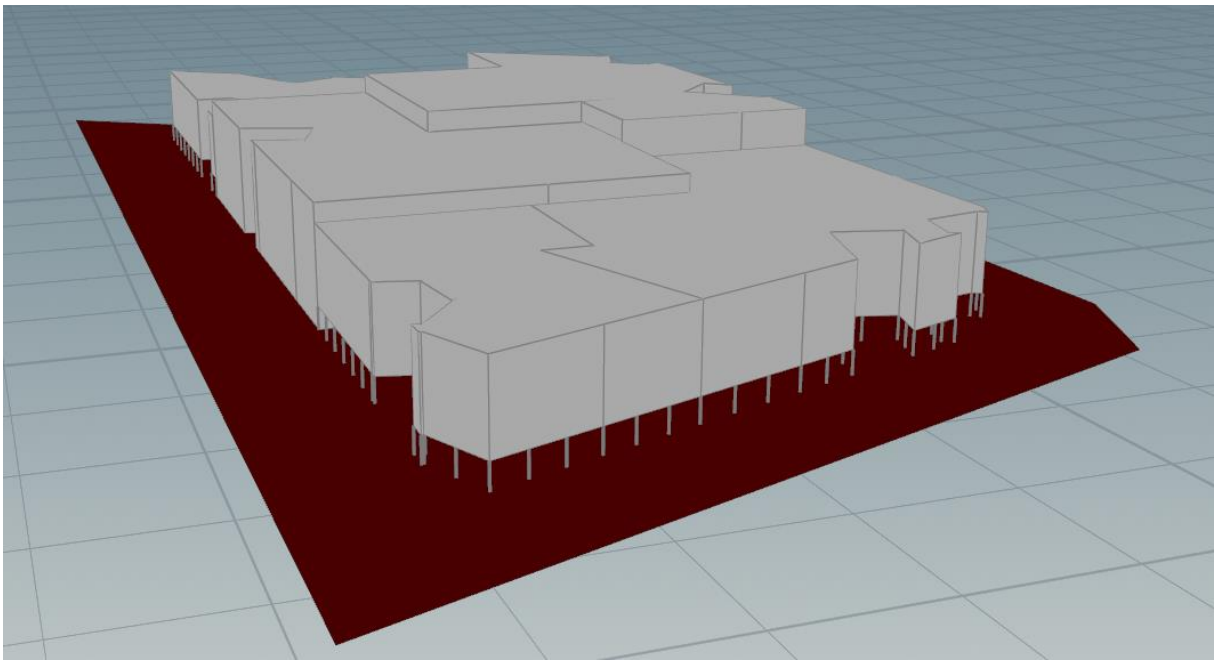


Figure 35 - Administrative buildings front

Flora

In the flora area there are three sections: walking paths, forests and bushed alleys. The regions are separated using Voronoi clusters. In this case, the pieces are not looped, and the points constantly

remain in the same place. Thanks to that, the Voronoi pieces have continuous paths through the different pieces.

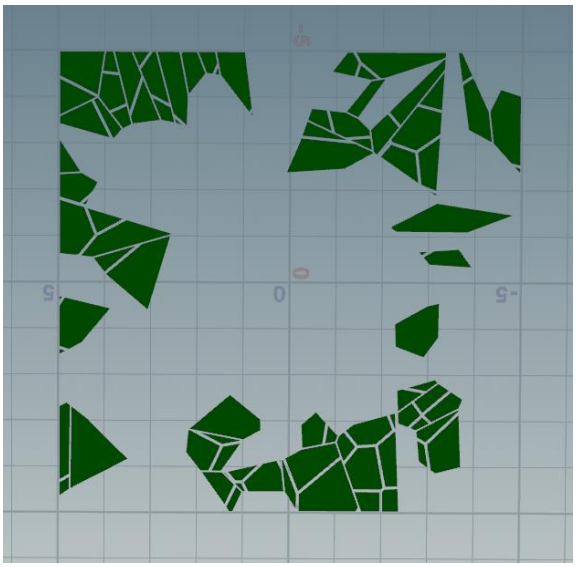


Figure 36 - looped Voronoi with different seeds

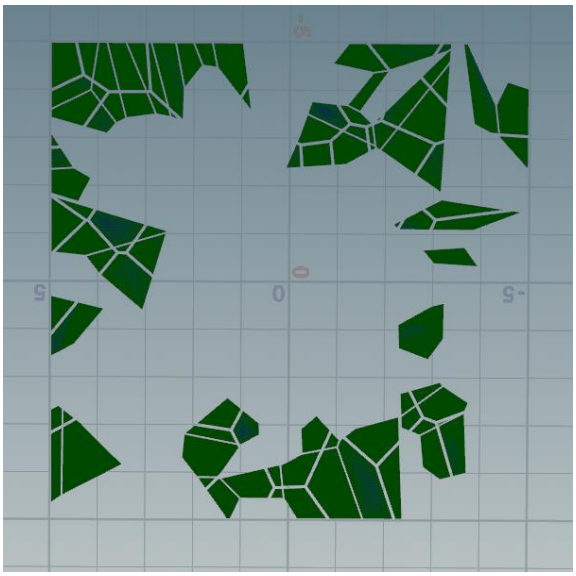


Figure 37 - Continuous non-looped polygons with the constant seed

Half polygons are shrunk to avoid the pathways. Random selection of every n edge is applied to the boundary lines. The sweep operation converts the lines into the cubic connected tubes which represent bushes. The second half of the polygons are simply scattered and assigned to the tree instance. The total polycount for the floral area is around 4000 polygons. The number may vary mainly depending on the scatter number for denser forests.

Alternative methods

Since the bush alleys are generated randomly many walk paths have alleys only on one side. A better solution would be to have bushes on both sides or none. To gain this effect the lines would be grouped before extruding into pathways and then assign the split lines into one group. The connection can be located using the distance value from the original points to the new points in extruded lines by transferring the attribute.

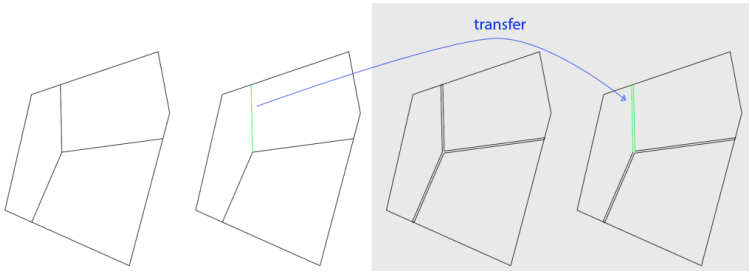


Figure 38 - transferring attribute

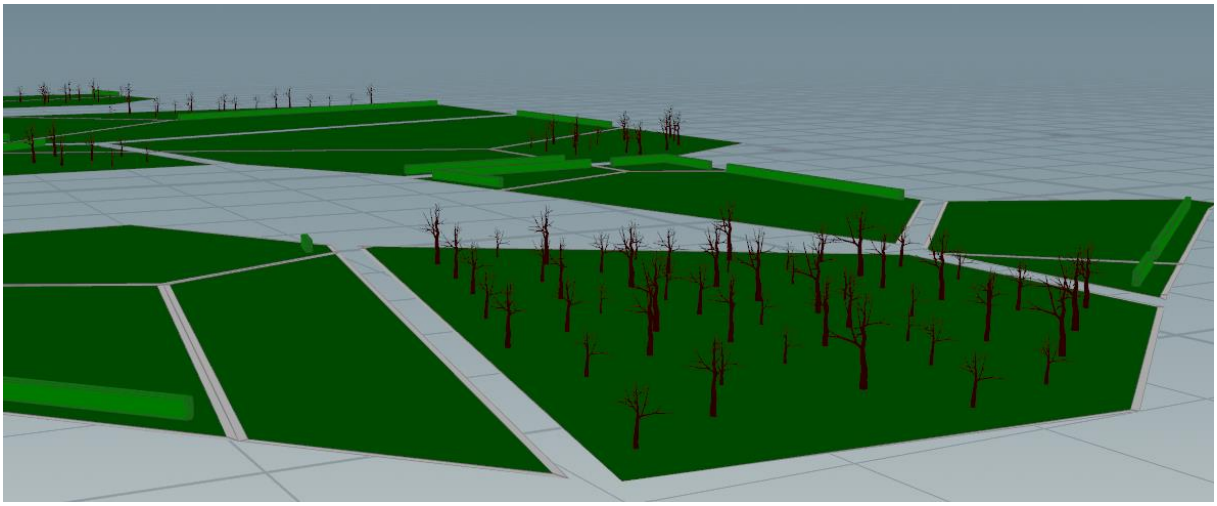


Figure 39 - Flora with forest with density of 0.1 units between each other

Roads and paths

The generation is creating 2 kinds of paths in terms of technique. 1st is already made using Voronoi fracturing thorough all the sections (main roads, secondary roads, park pathways) and 2nd kind is created for leading the ways from roads to the buildings.

Pathway to the residental buildings

The difference between pathways in residental areas and roads have the splitting function between regions. That is why the roads are primary made before building are generated in the regions. Pathways serve as a connection between the roads and main paths with the buildings, therefore they are created afterward.

The ways are placed as the shortest path on the divided geometry of the remaining terrain which is subtracted from the already extruded building areas. To avoid too organic structures, there is only a percentage of selected paths to show up.

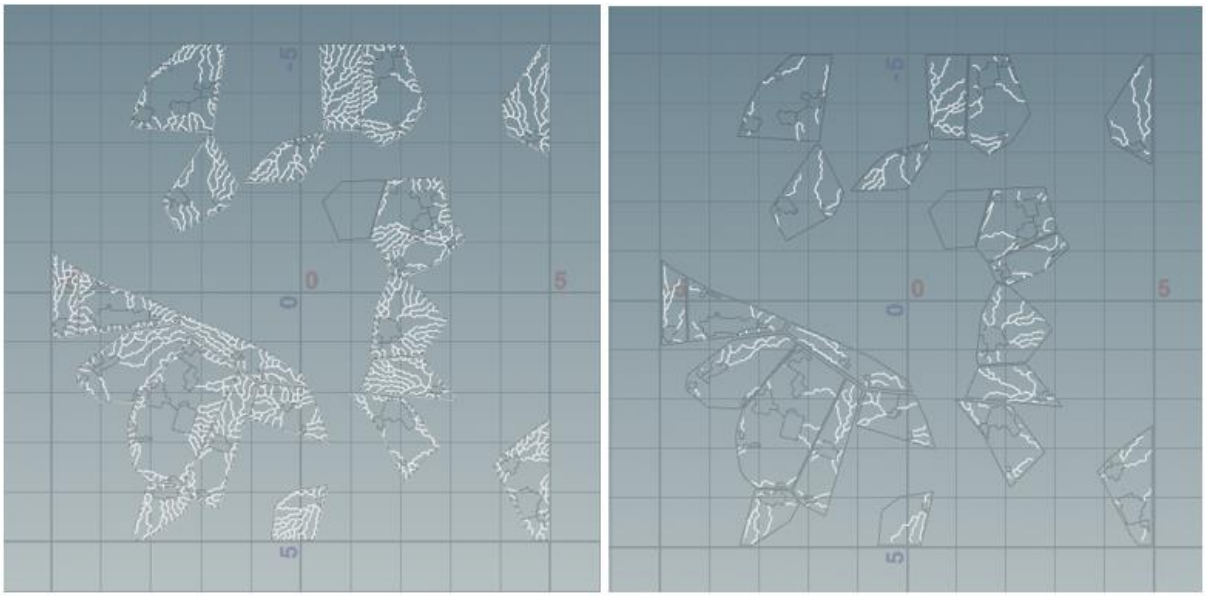


Figure 40 - Pathway calculation

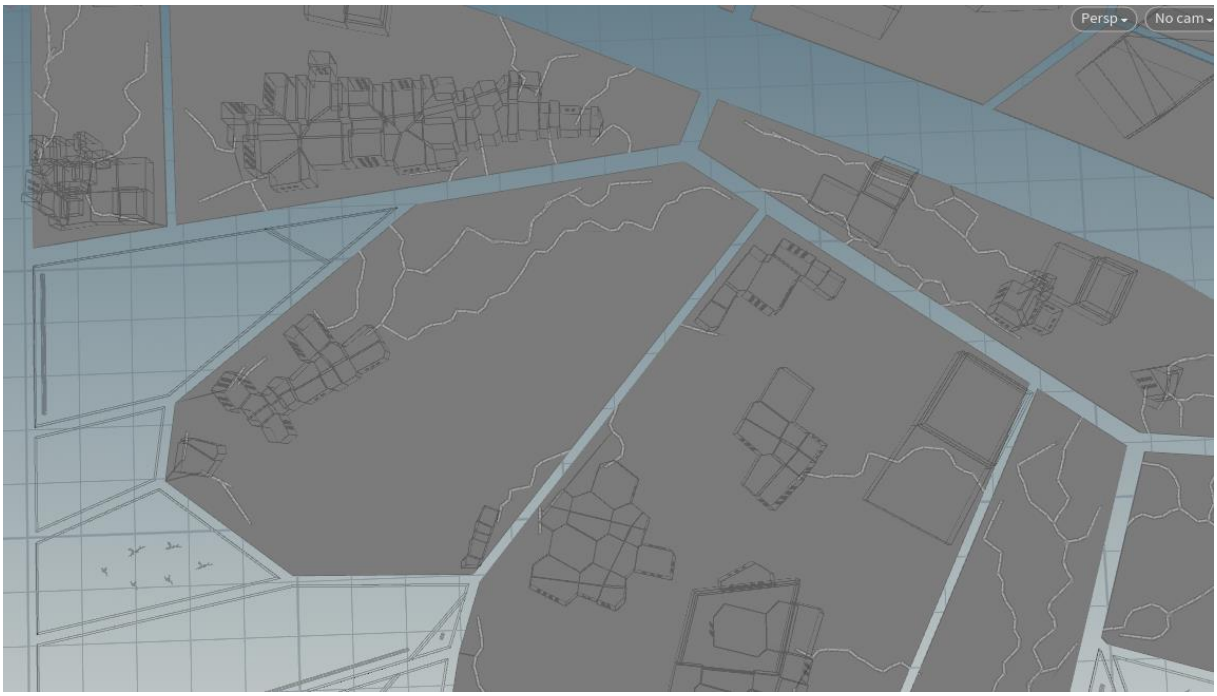


Figure 41 - applied pathways

Bridges

The Voronoi generation overlayed with the lakes has the advantage of the connected area detection. With that there are placed bridges which are useful for the water crossing.

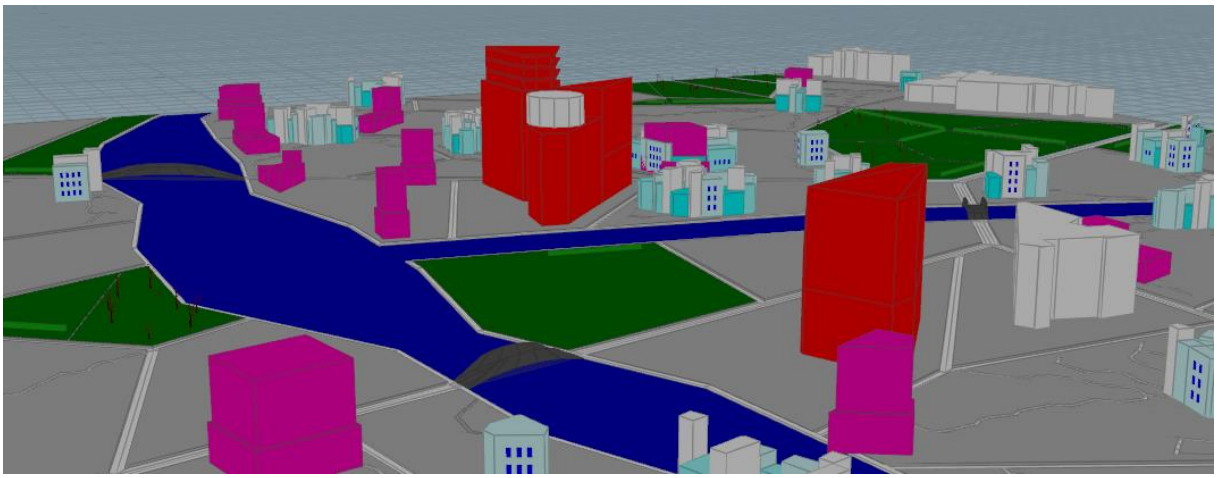


Figure 42 - Bridges preview

Testing

The generator is packed into HDA (Houdini Digital Asset). To maintain high performance, the asset is divided into four sections. The biggest focus in building the asset in the pre-production was to make it

the way, so the result seems to be simple. That is why there are not many parameters for controlling the HDA. For easier manipulation and faster performance, the sections are controlled using buttons – before editing the section it is desirable to push the button. The reason to this is to save computational time – when editing the water curves, it is not necessary to have the whole geometry visible. The difference in polygon count is described in the following sections.

Stage 1 – Base area

The main base has only parameter for seeding the highway variation. The option is making the points shuffle around the surface with the uniform layout on the plane. That is giving the option to input water lines through the city. The computation last milliseconds, the polycount is kept around 100 polygons.

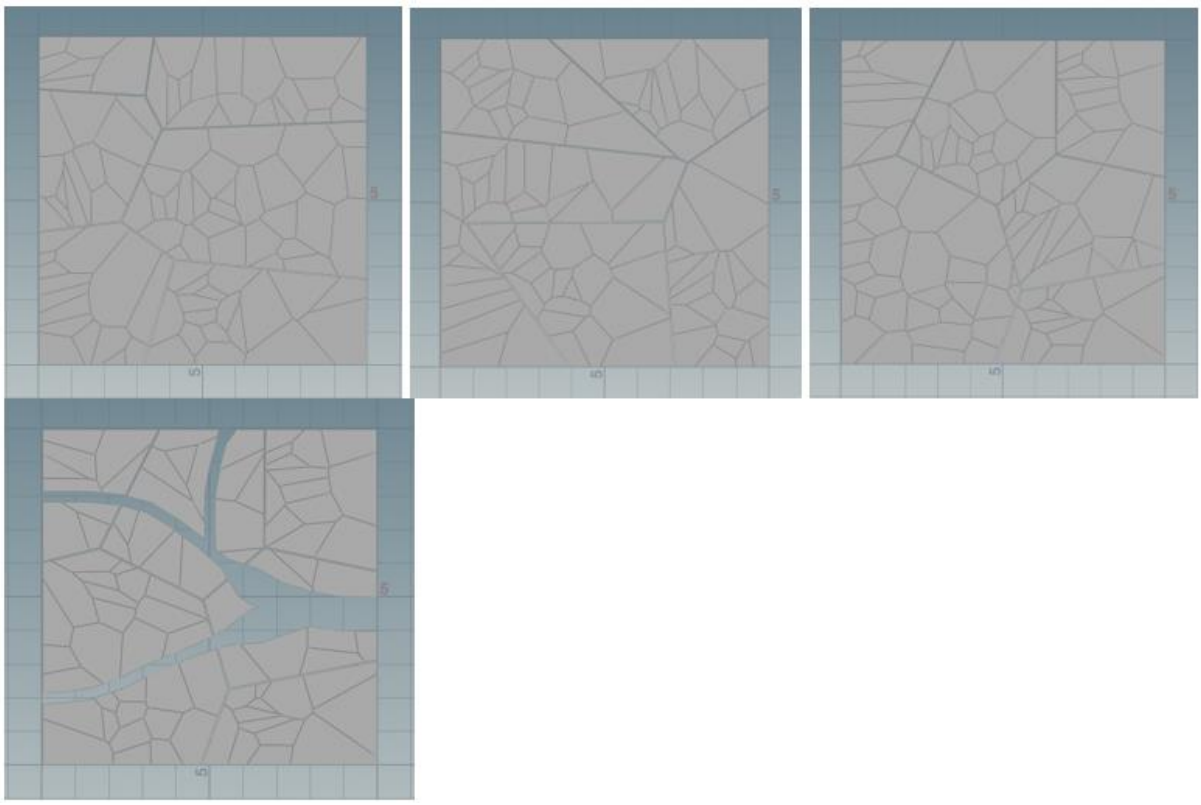


Figure 43 - Secondary roads variations

Stage 2 – Building generation

As discussed already in the building section, the buildings are driven by only one threshold controller rather than separating the amount and controlling percentace of every kind of buildings. The seed generation for variations is calculated in every global seed variation. The driving parameter in this section is the percentage of applied buildings. The real percentage is faked. The tests shows, that generating buildings can occur really slow if the whole area is covered by buildings. The base terrain which is divided before extrusion have around 1000 polygons for walk-around buildings and randomly generated buildings can reach up to 7000 polygons. Which is not a mad number, but dividing every extruded face from that multiplies the number by 5 from the walk-around buildings makes it around 5000 and with randomly generated buildings makes it around 40 000 polygons, which is causing some delays in computation (my software falls using this number generation). Testing the percentage, total amount of all surface polygons used for generating buildings are divided by 4 and assigning the value to the maximum value of possible percentage of the controller. With the 50% applied buildings and a river drawn the system have around 5500 polygons. In this stage it is possible to shuffle the global seed. The speed may vary, on my personal laptop it takes about 3 seconds to generate a new city with the display flag on stage 2.

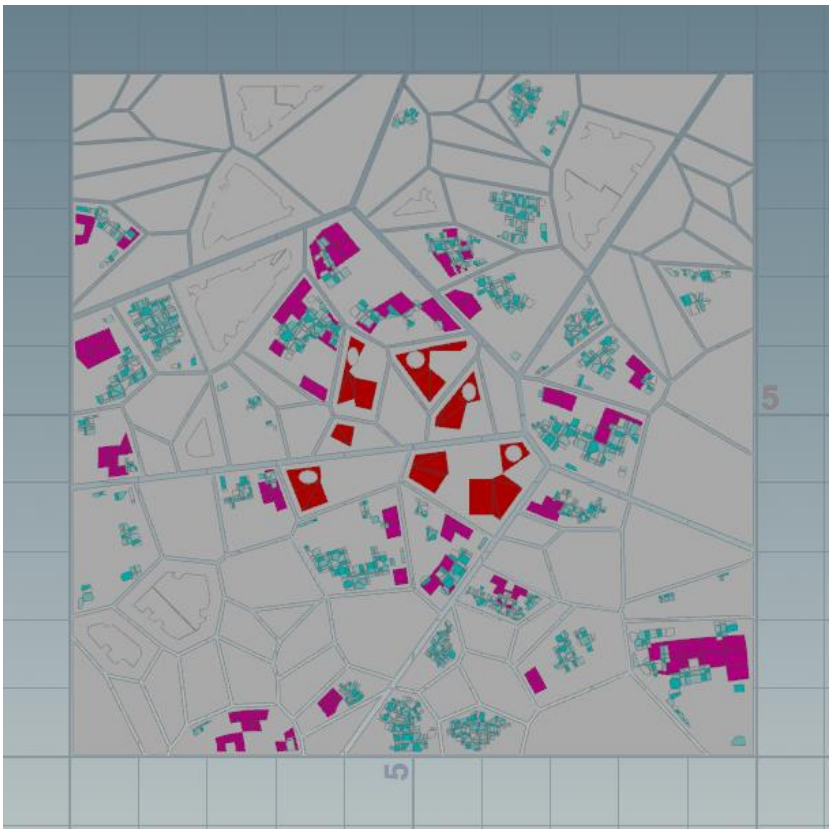


Figure 44 - Building generation

Stage 3 – Flora

Flora has trees in packed geometry and bushes which uses very simple algorithms. This section allows to input a different mesh instead of the tree example. Generation at this stage takes also about 3 seconds and generates in total 8000-9000 polygons.

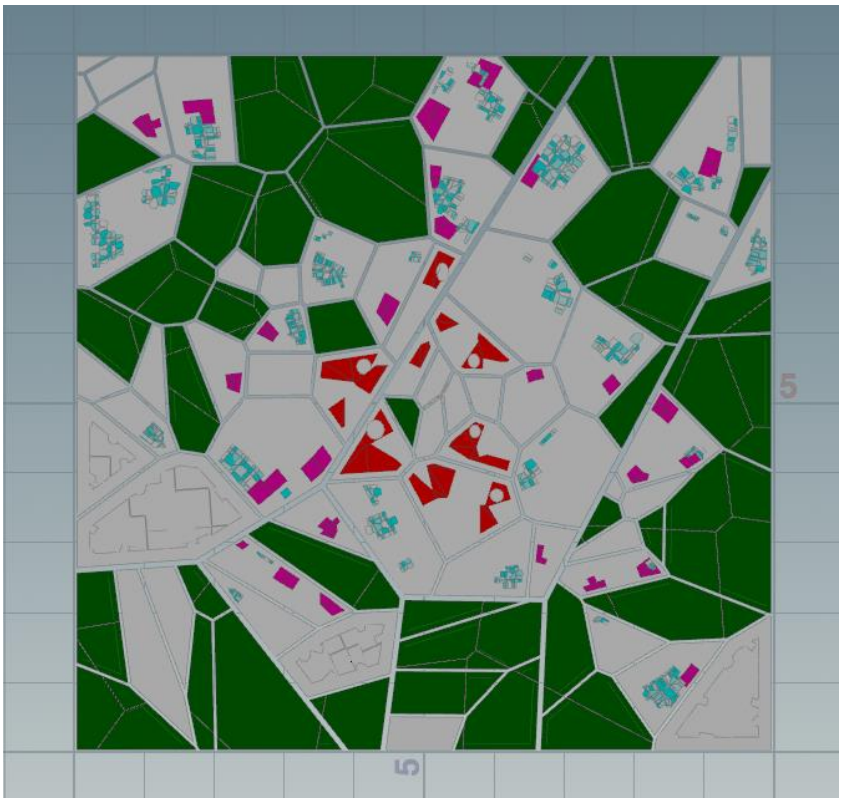


Figure 45 - Flora

Stage 4 – Roads and Pathways

This section generates all the ways which are missing in the gaps. The most computational heavy are the pathways which generates paths which are then decreased into a 1/6 of total paths. Computation takes around 10 seconds per seed and generated polygons are around 19000 pieces.

6. Conclusion

The city generator was created using four stages: base surface, building generation, flora generation and pathways with roads. Regions used Voronoi patterns widely for fracturing the areas which have a tolerable balance between too random-natural pattern and too artificial distribution. Buildings were generated from the ground plans instead of scattering, which is computationally heavier than scattering them on the base surface. Using random threshold, the buildings were selected and extruded with different heights in two variations. One of the variations consisted of instances which were saving a significant amount of memory. So, instancing was used in the flora for the trees. The biggest polycount was recorded in the building creation where the number from the previous step is multiplied 50 times. However, the performance did not suffer from the polycount anyhow significantly. To seed a variation the generator needs 10 seconds.

The generator used primarily SideFX Houdini and built a really simple visualization of the modern city which can be used in games or extended into films.

Polycount efficiency – a total polycount of 40 000 is runnable in the game engines. However, that gives a really low limitation for extending and adding more features. The different organization might create better performance and would be able to hold heavier geometry.

Naturalism – buildings respected the curves and angles of the roads, which do not always have straight cubic shapes. Some naturalism is missing in the flora. It needs to have more possible variations of nature. The polygon count was saved for better performance in buildings.

Interactivity – there was a big effort to make the tool simple. Every section has no more than four controllers, which are significant since they are connected to more controllers.

The generator is completed within the given scope with all the main aspects containing.

For future work, the system might be rebuilt and found a more proper way for pathways showing errors in some variations. They occur disconnected from the edges, or sometimes it may cause some polygon disappearing – Boolean operations might make mistakes in these parts.

For the material assignment, there might be useful to heave the material attributes ready. However, they might also be created directly in the game engine by extracting every colour. The disadvantage in assigning materials in the game engines might be the uncontrollability of the UVs.

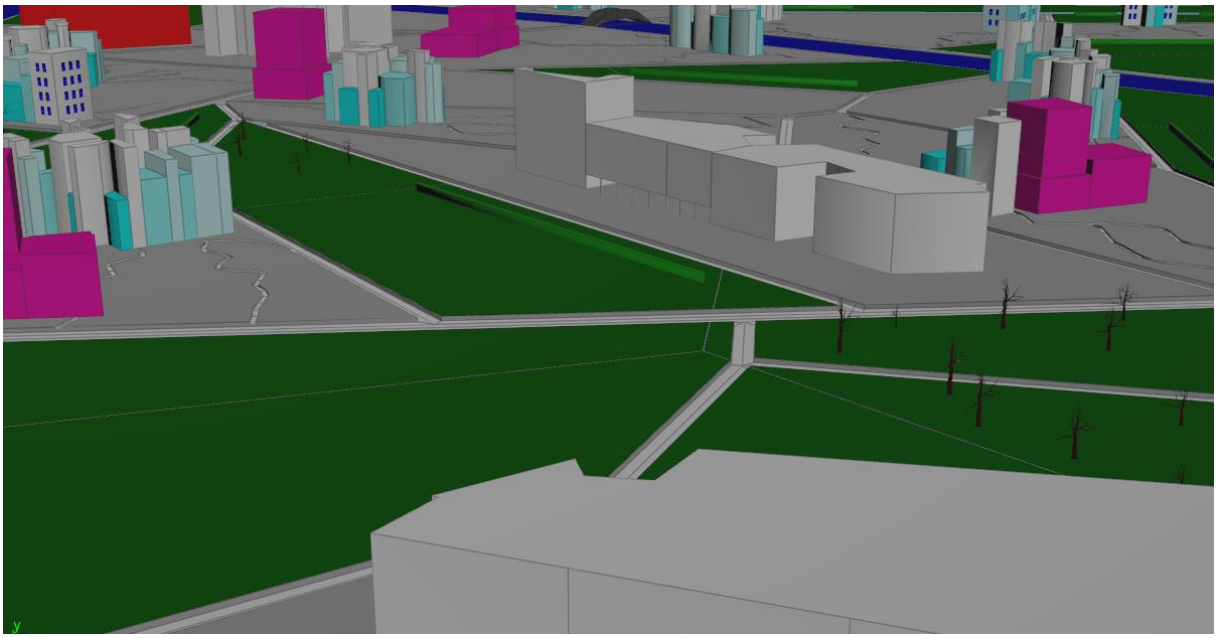


Figure 46 - Outcome example

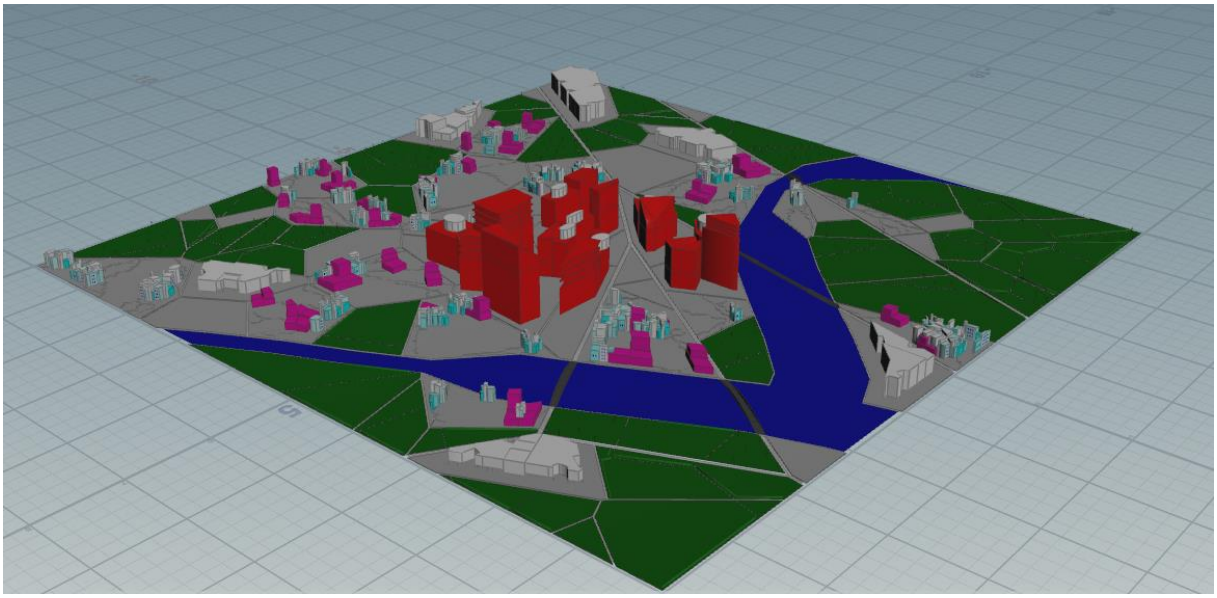


Figure 47 - Outcome example

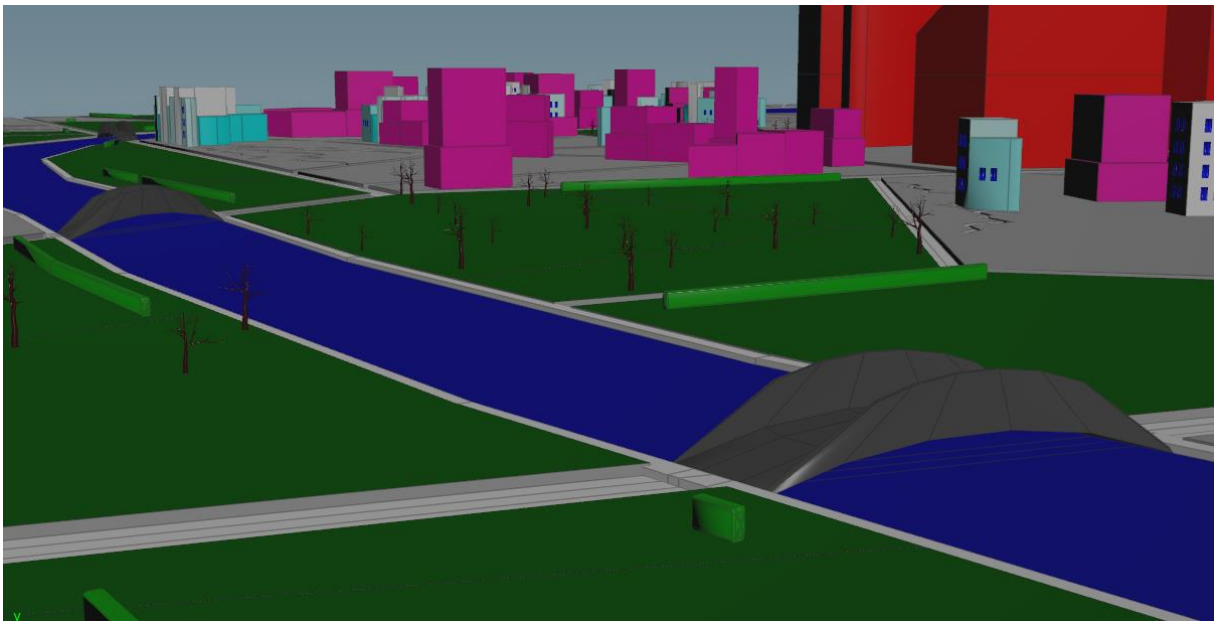


Figure 48 - Outcome Example

References

al., N. O. e., 2017. *Procedural city generation using Perlin noise,* Karlskrona: s.n.

David S. Ebert, F. M. D. P. K. P. S. W., 2003. *Texturing & Modeling.* 3rd ed. San Francisco: Elsevier Science .

Dong, J. L. J. Y. K. C. M. Q. L. Y. H. &. J., 2020. *Survey of Procedural Methods for Two-Dimensional TextureGeneration,* s.l.: Sensors.

George Kelly, Hugh McCabe, 2006. A Survey of Procedural Techniques for City Generation. *The ITB Journal,* 7(2).

George Kelly, Hugh McCabe, 2006. *Citygen: An Interactive System for Procedural City Generation,* s.l.: s.n.

Gustavson, S., 2005. *Simplex noise demystified,* s.l.: s.n.

Lindenmayer, A., 1987. Mathematical Models for Cellular Interactions in Development. *Journal of Theoretical Biology.*

Miliutin, N. A., 1975. *Sotsgorod: The Problem of Building Socialist Cities.* s.l.:s.n.

Mitul, A., 2019. *Futurismus a expresionismus v české avantgardní literatuře 10. a 20. let 20. století,* Prague: s.n.

Perlin, K., 2002. *Improving Noise,* s.l.: Siggaraph 02.

Stefan Greuter, Nigel Stewart, Jeremy Parker, Geoff Leach, 2003. *Undiscovered Worlds - Towards a Framework for Real-Time Procedural World Generation,* s.l.: s.n.