UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Mathias Plans

Procedural Generation of Unique Buildings

Bachelor's Thesis (9 ECTS)

Supervisor:
Raimond-Hendrik Tunnel, MSc

Tartu 2021

Procedural Generation of Unique Buildings

Abstract:

This thesis introduces grape grammars, a new way to procedurally generate architecture. The grape grammar is a development on split grammars, which are grammars that work on a vocabulary of shapes. The novelty lies in that instead of using shapes, the grape grammar uses graphs. This allows grape grammar to use grammar rules in tandem with the Wave Function Collapse (WFC) algorithm, resulting in the flexibility of formal grammars and constraint-based design of WFC for procedural architecture generation. Furthermore, the use of graphs permits establishing non-trivial connections between grammar, allowing for enforced symmetries. This thesis shows that when using all the features together, it is possible to generate a large variety of buildings.

Keywords:

Computer graphics, procedural generation, graph theory, formal grammar, shape, shape grammar, split grammar, wave function collapse algorithm, architecture, computer-aided design

CERCS:

P175 Informatics, systems theory

T240 Architecture, interior design

Ainulaadsete hoonete protseduuriline loomine

Lühikokkuvõte:

See lõputöö esitab grape grammatika, mis on uus viis protseduuriliselt luua arhitektuuri. Grape grammatika on edasiarendus split grammatikast, mis töötab kujudest koosneval sõnavaral. Grape grammatika uudsus on see, et kujude asemel kasutatakse graafe. Graafide kasutamine lubab grape grammatikal grammatika reeglitega samaaegselt kasutada Wave Function Collapse (WFC) algoritmi, mis annab formaalse grammatika paindlikkuse ja WFCst tuleneva piirangupõhise disaini. Graafi kasutamine võimaldab ka luua mitte-triviaalseid ühendusi grammatika sümbolite vahel, mille abil saab luua sunnitud sümmeetriaid. Nende omadustega saab genereerida suures koguses unikaalseid ehitisi.

Võtmesõnad:

Arvutigraafika, protseduuriline generatsioon, graafiteooria, formaalne grammatika, kuju, kujugrammatika, jaotusgrammartika, lainefunktsiooni kokkulangemise algoritm, arhitektuur, raalprojekteerimine

CERCS:

P175 Informaatika, süsteemiteooria

T240 Arhitektuur, sisekujundus

Table of Contents

1. Introduction

Big urban environments are increasingly common in virtual worlds. While conventionally, sets and backgrounds in animated films are painted, recently, complex virtual cities have been constructed for better immersion. San Fransokyo, a fully 3D virtual city from *Big Hero 6* (2014), contains over 83000 buildings and is modeled after San Francisco and Tokyo [1]. However, crafting virtual environments with such scale is time-consuming and can be the most challenging aspect of a project [2]. Whereas the buildings in *Grand Theft Auto 5* (2013) are modeled and placed manually, *Big Hero 6*, *Zootopia* (2016), and others use procedural generation to produce expansive yet unique virtual urban environments.

For the virtual urban environments to be believable, individual modeled buildings have to be detailed and varied. This thesis offers a solution to the problem of procedural generation of buildings for creating interesting virtual environments. The project was conceived from the lack of free tools. The existing tools, such as Esri CityEngine [3], use many techniques for generating buildings, which include formal grammars [4] and Gumin's Wave Function Collapse (WFC) algorithm (2016) [5].

Among formal grammars, two notable solutions are Lindenmayer's L-systems (1968) [6] and Stiny's Shape Grammars (1980) [4]. L-system is a powerful tool and has found applications mostly in the modeling of flora. Shape grammars are more suited for buildings, but they were not created with architecture in mind. Split grammar, a specific type of shape grammar, is used in *Zootopia* and *Big Hero 6* [1].

While Wave Function Collapse can also be used to generate procedural buildings, it is only defined to work on an already existing grid, because its building blocks are grid-based. This limitation makes the results of WFC seem periodic, even if the grid is irregular [7]. In some situations, a grid might not exist at all, in which case it needs to be created. Yet, constraint-based rules, like in WFC, have been used in architecture. It would be beneficial to use both formal grammars and WFC together, to get both the flexibility of formal grammars and the constraint-based rules of WFC.

The thesis combines split grammars and WFC. To that end, the following additions are devised and presented:

- Shape graphs are introduced. These are graphs that represent collections of shapes. Using a graph has two main advantages: the context of shapes is easy to gather, and graph-based algorithms can be used on the shapes. (3.2)

- Grape grammars are introduced. The grammar is a special type of graph grammar that works on shape graphs. (3.3)

The thesis has three main parts. Firstly, the methods used in the algorithm and their derivatives will be discussed in chapter 2. Secondly, the algorithm is defined in chapter 3. Thirdly, the algorithm will be demonstrated with a small example grammar and its aspects are discussed in chapter 4. Finally, the appendix contains the glossary, a full example grammar, link to the repository of a demonstration application, and a publishing license.

The illustrations in this thesis that demonstrate the algorithm results were in part done in the Computer Graphics Project course. A demonstration application[1] was developed, which' output pictures can be seen throughout the thesis.

[1] https://courses.cs.ut.ee/2021/cg-pro/spring/Main/Project-HouseCollapse

2. Used Methods

The proposed grape grammar includes ideas from many different algorithms. This chapter will give a brief overview of the base methods used in grape grammar. The grape grammar is a formal grammar, specifically based on shape grammar.

Shape grammar, developed by George Stiny [4], is a commonly used method for procedurally generating houses and cities. The idea of shape grammar is to use formal grammars on shapes of two or more dimensions. Although Stiny did not present shape grammars with architecture in mind, nowadays it is commonly used in procedural architecture. Esri CityEngine [3] is a well-known program used to generate big yet variable cities using shape grammars. Tracy Cozzens, an author at GPS World, brings examples of what the software is capable of. Notable examples are the cities from *Zootopia*[2] and *Big Hero 6*[3] [1]. The cities can be seen in Figure 1 and Figure 2.

Figure 1. San Fransokyo[2] from *Big Hero 6* was modeled using split grammars.

Many modifications have been done to shape grammars. Wonka et al. [8] present split grammars, which have some interesting beneficial properties over shape grammars. Shape grammars have no restrictions on which shapes can be replaced with each other. While it makes them extremely flexible, it cannot prevent collisions between the shapes (meaning that shapes can intersect with other shapes in the same production). For the results to look like believable architecture, collisions must be avoided. Split grammars resolve this issue by not allowing replacements with shapes that are bigger than the original shapes. Wonka

[2] https://disneyparks.disney.go.com/blog/2014/11/see-the-city-that-inspired-the-creators-of-big-hero-6-on-an-adventures-by-disney-san-francisco-vacation/
[3] https://www.mrskathyking.com/welcome-to-the-world-of-zootopia/?fdx_switcher=true

Figure 2. Zootopia[3] was also modeled using split grammars.

et al also introduce control grammars and shape attributes, which allow for better control of the result without changing the underlying split grammar.

While split grammar is a very powerful tool, it is not suitable for solving a constraint-based problem. Certain architectural components might need to be placed following a specific criterion. A good example of this is *Kvartal*[4], a building in the city center of Tartu, seen in Figure 4. The visual constraint there is that no two neighboring sections have the same material. For solving constraint-based problems like this, Maxim Gumin [5] has developed an algorithm called Wave Function Collapse (WFC). This algorithm works in the following way. The designer defines what adjacencies between states of nodes are allowed and a grid of nodes is initiated with each node having all the possible states. Then the algorithm starts collapsing the nodes to a single state and propagating the change to all the other nodes on the grid. WFC finishes when all the nodes on the grid have been collapsed to a single end state. Finally, building blocks are placed according to the states of the nodes. Gumin created this algorithm for the purposes of texture synthesis (an example with a tile set[5] can be seen in Figure 6), but it has been used for architecture as well. One notable example is *Town Scraper*, a toy developed by Stalberg [7], seen in Figure 3[6].

Figure 3. Town Scraper[6].

[4] https://archello.com/project/tartu-department-building-kvartal
[5] http://cr31.co.uk/stagecast/wang/tiles_e.html
[6] https://www.bloomberg.com/news/articles/2020-07-24/the-video-game-where-you-build-an-empty-town

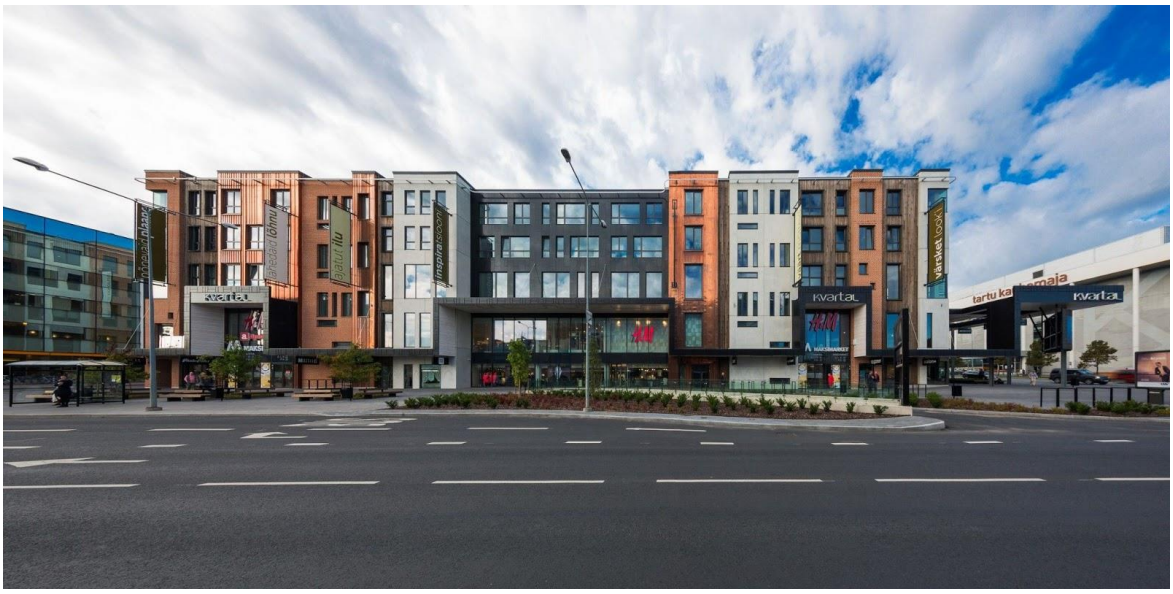


Figure 4. *Kvartal*[4] has a constraint-based design where no two neighboring sections have the same façade material.

The limitation of WFC is that it must work on an already existing grid. This can make the productions of the algorithm seem too periodic and the grid pattern might be apparent, as seen on a WFC production[7] in Figure 5 and Figure 6. The solution is to not use a grid at all. Hwanhee Kim et al [9] generalized the algorithm to work on any kind of graph, making it a very useful tool for solving any kind of constraint-based problem that can be represented as a graph. The authors use their version of WFC to present solutions to problems such as solving a sudoku and the 4-color problem[8]. Using graphs instead of grids is useful also since the architecture might not be decomposable into a grid.

Figure 5. A potential WFC result[7]. The grid pattern is apparent.

To combine the WFC and split grammars, the shapes must be represented as a graph, on which the WFC can operate. This means that the proposed grammar needs to operate on graphs as well, not shapes. Graph grammar, also known as graph rewriting, is a formal grammar that works on a vocabulary of graphs. Rozenberg [10] presents multiple

[7] https://marian42.de/article/wfc/
[8] https://en.wikipedia.org/wiki/Four_color_theorem

approaches to graph grammars, which mainly differ in regard to how the edges of the graph are handled when a node is replaced. The grape grammar, which is a specific type of graph grammar proposed in this thesis, introduces a new approach: the edges are handled based on the shapes that are represented by the graph.

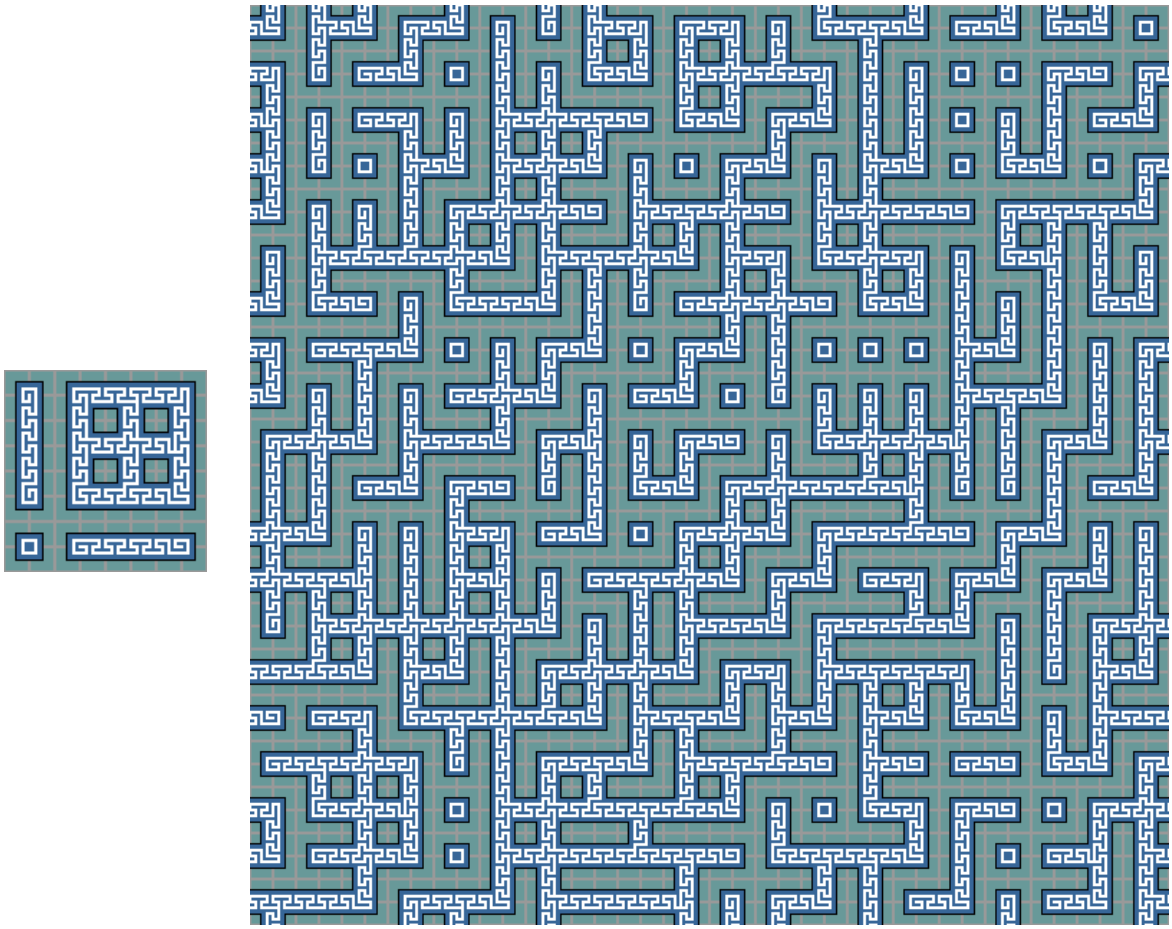


Figure 6. A use case of WFC. The constraints are defined on the tiles from the tile set[5] on the left. Then on the right a 20×20 grid is generated where the defined constraints are satisfied.

3. The Algorithm

The grape grammar works on graphs that represent geometric shapes. This section will define the grape grammar and its associated structures. The algorithm is defined for 2D, but it can be generalized to work in any dimension.

3.1 Shape

The algorithm uses shapes, which were formally defined by Stiny [4]:

Definition 1: *A shape is a limited arrangement of straight lines in Euclidean space.*

Note that the lines in the shape are not equivalent to regular geometric lines nor line segments. A *line* $l = p_1, p_2$ consists of two *end points* p_1 and p_2, where the distance between p_1 and p_2 is finite and non-zero. A point p is *coincident* with line l if and only if p is one of the end points of l or the sum of the lengths of $l_1 = p_1, p$ and $l_2 = p, p_2$ is equal to the length of l. In other words, p is coincident with line l if it is *on* the line.

Two lines l_1 and l_2 are *collinear* if and only if (a) they share an endpoint and the remaining end point of one of the lines is coincident with the other line, (b) two end points of one line are coincident with the other line, (c) one end point from each line is coincident with the other line, or (d) the lines share an end point which is coincident with the line formed by the remaining end points. The conditions are illustrated in Figure 7.

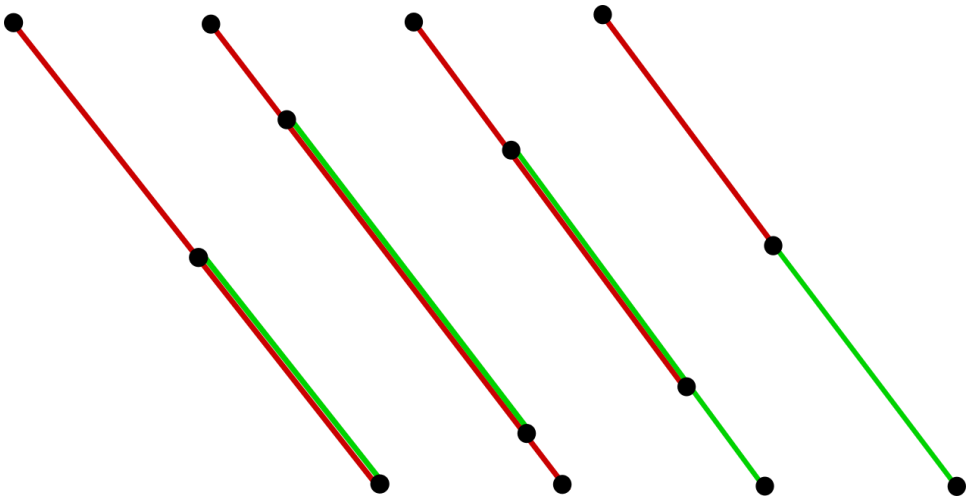


Figure 7. The conditions of line collinearity (a), (b), (c), and (d) on the first, second, thirds, and fourth lines, respectively.

A shape is a set of lines where no two lines are collinear. These lines can also be called *maximal lines* of the shape. Since shapes are sets, set operations can be used on them. Notably, the set union can be used to merge shapes.

Shapes can be more useful if we can differentiate them with labels. A *labeled point* p: V is a point p with value V attached to it. A *labeled shape* is denoted by $\langle s, P \rangle$, where s is the set of maximal lines and P is the set of labeled points. All the points in the labeled shape are labeled, but they do not have to have different labels. A specific label can be designated to represent an "unlabeled" point.

Definition 2: *Line l_s is a subline of line l if and only if the end points of l_s are coincident to l.*

If point p is an end point of line l, then it is also coincident to l. Therefore, any line is a subline of itself. The number of sublines of a line with n coincidental points is $\binom{n}{2}$, which is the number of possible pairs of the coincidental points. In Figure 8 the $\binom{4}{2} = 6$ sublines of the line $l = p_1, p_4$ are $l_1 = p_1, p_2$, $l_2 = p_2, p_3$, $l_3 = p_3, p_4$, $l_4 = p_1, p_3$, $l_5 = p_2, p_4$, and l itself.

Definition 3: *Line l_m is a minimal line if and only if it has only one subline.*

A set of minimal lines of a line l with many coincidental points $P = \{p_1 p_2 \dots p_n\}$ is defined as

$$\{l_m \mid (\forall p_s \in P)(\forall p_e \in P)[l_m = p_s, p_e \; is \; a \; minimal \; line \; of \; l]\} \quad (1)$$

This effectively splits the line by all the coincident points. The process is illustrated in Figure 8. The set of minimal lines of a shape s is the union of the minimal lines of each line in s.

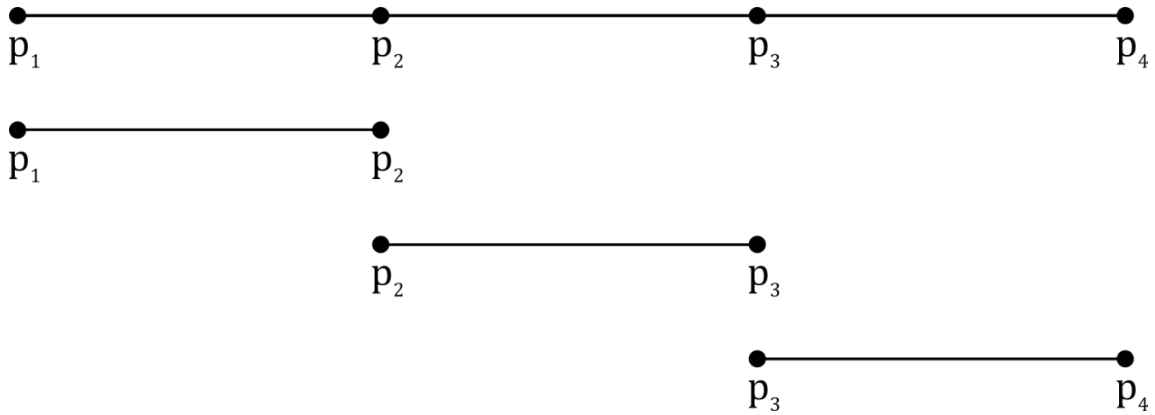


Figure 8. Line $l = p_1, p_4$ has six sublines $l_1 = p_1, p_2$, $l_2 = p_2, p_3$, $l_3 = p_3, p_4$, $l_4 = p_1, p_3$, $l_5 = p_2, p_4$, and l itself. The lines l_1, l_2, and l_3 are also minimal lines, since the only coincidental points of them are their end points.

For the purpose of architecture, the shapes must be *closed*. In 2D, this means that the shape must be a polygon (as opposed to polyhedron in 3D). The idea of split grammars is to partition and replace these closed shapes. To that aim, Wonka et al [8] defined the basic shape for closed 3D convex shapes. Since the grape grammar is defined to work in 2D, a new definition is devised.

Definition 4: *A basic shape b is a structure $b = \langle s, l, V \rangle$, where s is a closed convex shape, l is a label, and V is a grammar symbol, which can be terminal. Attributes can be attached to the grammar symbol V.*

Attributes are represented by sets of real numbers $[x, y]$, where x can be $-\infty$ and y can be $+\infty$. The attributes can be used to carry some kind of information of the shape, such as color, size, and position. They are also used to manipulate the derivation process of split grammar and thus grape grammar. Attributes are usually named.

The difference between the proposed grape grammars and Wonka's [8] split grammars is that grape grammars operate on graphs, not shapes.

3.2 Graph

The grape grammar operates on a specific graph called *shape graph*. The vertices of the shape graph represent the shapes and the edges physical connections between the shapes (illustrated in Figure 10).

Definition 5: *A shape s_1 is connected to s_2 if and only if the number of minimal lines in the union of s_1 and s_2 is not equal to the sum of the number of minimal lines in s_1 and s_2. This can be marked as $s_1 \sim s_2$.*

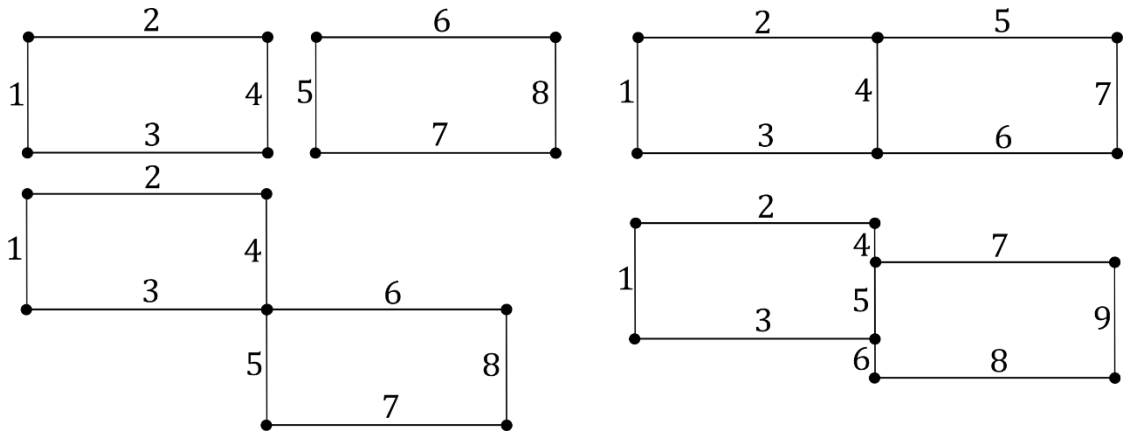


Figure 9. Minimal lines of the shapes enumerated. The shapes on the left are not connected and on the right are connected.

The connectedness between two rectangular shapes is illustrated in Figure 9. One rectangle has four minimal lines. As can be seen in the bottom-left of the figure, two shapes are not connected when only a vertex is shared between them, since the number of minimal lines is the same as in the case of disconnected shapes. The two cases on the right of the figure are connected, because the number of minimal lines is not equal to the sum of the minimal lines in the individual shapes. Now that the connection between the shapes is defined, the shape graph can be defined.

Definition 6: *A shape graph* $S = \langle V, E, e, k, l \rangle$ *consists of a set of vertices* V*, edges* E*, a function* e*, which maps the vertices to the set of its connected edges, a bijective mapping* k*, which maps the vertices to the basic shapes, and a mapping* l*, which is the inverse mapping of* k*.*

The process of converting the set of basic shapes B to the equivalent shape graph is defined in the following algorithm:

```
1  for each b in B
2    add a new vertex v to V
3    define k(v) = b
4    define l(b) = v
5
6  for each b1 in B
7    for each b2 in B
8      if b1~b2 and b1 != b2
9        v1 = l(b1)
10       v2 = l(b2)
11       add a new edge en=v1,v2 to E
12       add en to e(v1)
13       add en to e(v2)
```

The loop on the lines 1-4 creates all the vertices of the shape graph. The loop on the lines 6-13 defines the edges between the vertices. The condition on the line 8 guarantees that edges are drawn only if the shapes are connected.

To convert the graph back to the set of basic shapes B', iterate through all the vertices $v \in V$ of the shape graph and add $k(v)$ to B'.

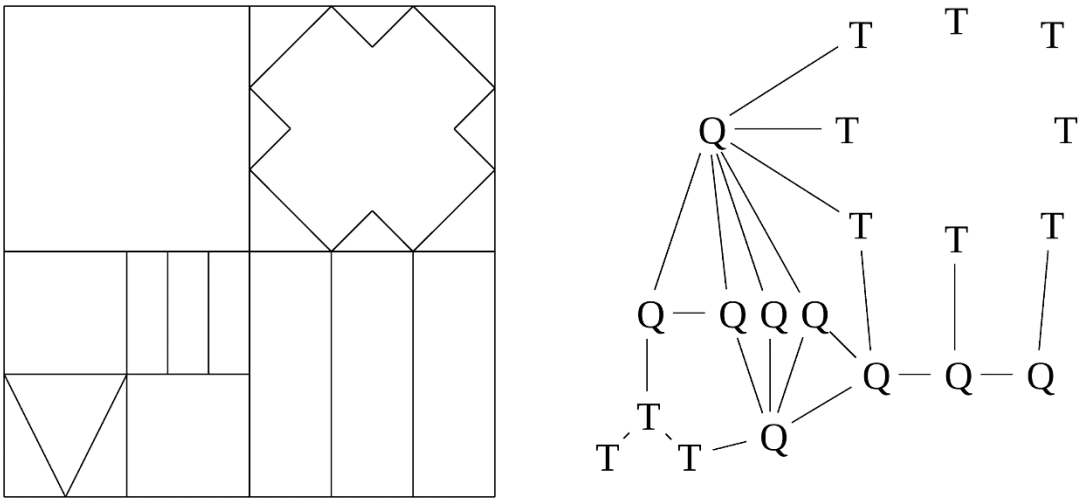


Figure 10. Collection of shapes on the left and its equivalent shape graph on the right. Note that the "plus shape" is empty space, so there is no vertex for it in the shape graph.

3.3 Grammar

Wonka et al [8] defined a *split* operation, which is a replacement of a basic shape by multiple other basic shapes while preserving the space, meaning they must fill the original shape. On the shape graph it is defined thusly:

Definition 7: *A split is the replacement of the vertex v of the shape graph S with multiple new vertices V_n where:*

1. *The shape $k(v)$ and the shapes $\{k(v_n)|v_n \in V_n\}$ cover the exact same area,*
2. $V' = (V \setminus \{v\}) \cup V_n$,
3. $E' = \big(E \setminus e(v)\big) \cup E_n$, *where*

$$E_n = \{e_n = v_1, v_2 \mid (\forall v_1 \in V')(\forall v_2 \in V_n)[k(v_1) \sim k(v_2)]\}, \quad (2)$$

4. *e', k' and l' are set accordingly.*

This operation is equivalent to converting the shape graph S into the set of shapes B, creating a new set of shapes B' from B by applying the split operation defined by Wonka et al [8] on B, and then converting B' to the shape graph S'. Note that because of the first condition of the Definition 7, the split operation cannot create nor remove space. This property is useful and is exploited in the grape grammar. This leads us to the definition of the proposed grape grammar.

Definition 8: *A grape grammar is a graph grammar where two possibly context-sensitive types of rules are permitted:*

1. *split rule, where a split operation is applied on a vertex v of a shape graph.*
2. *conversion rule, where a vertex v of a shape graph with the shape $b = k(v)$ is replaced by another vertex v' with the shape $b' = k(v')$, where b' fits into b.*

The rules are written in the form $A \rightarrow B$, where A is the left-hand side (LHS) of the rule and B is the right-hand side (RHS) of the rule. In the case of a context-sensitive rule, the LHS of the rule contains multiple symbols, and when a rule produces many shapes, the RHS of the rule has multiple symbols. One symbol might be in the LHS of many rules, in which case the rule is selected stochastically. A grammar symbol is *terminal* if it does not occur in LHS of any grape grammar rule. Usually, the grape grammar rules are applied until all the symbols are terminals. Since shapes are multidimensional, it is more intuitive to illustrate the rules using images, as shown in Figure 11.

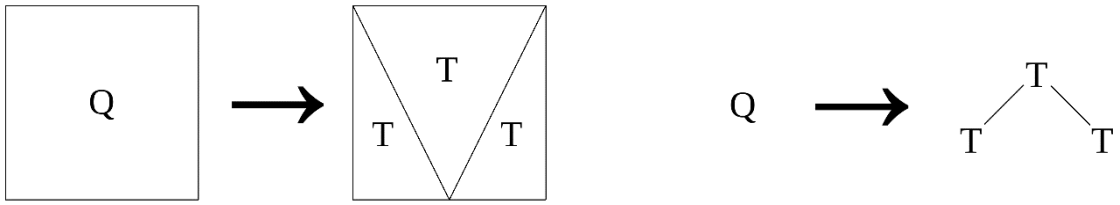


Figure 11. A grape grammar split rule, illustrated by shapes (on the left) and by the graph (on the right).

For convenience, some rules might be a split and conversion rule at the same time. This can be seen as two rules applied after one another: a conversion rule to replace the original with another shape, and a split rule that is applied to that converted shape.

With the growth of the number of rules, the results get more random. To control the rule selection better, attributes are used.

3.4 Attribute Propagation

The attribute system by Wonka et al [8] serves two purposes: carrying the material information and influencing the derivation process of the grammar. The propagation of the attributes is conventionally done in two ways: manually setting them, which is done for the starting shape, or inheriting them from the parent shape. For the final product of the grammar to make architectural sense, Wonka introduced a third method in the form of control grammars.

FACADE_CONTROL → DOOR_PATTERN, DECOR_PATTERN, DECOR_PATTERN
DECOR_PATTERN → PILASTER | INDENTATION | CORNICE | CANOPY
DOOR_PATTERN → DOOR | GARAGE
DOOR → ⟨(0, 0), *door*, [1, 1]⟩ | ⟨(0, 1), *door*, [1, 1]⟩

Figure 12. A part of a control grammar.

The control grammars are a way to spatially distribute attributes. Different floors or sides of the building might need to vary, such as having different shaped windows on the first floor. This can be achieved by adding new rules to the grammar but doing that for every different detail will get tiresome. To automate this process, a control grammar is evoked. A special attribute acts as a control symbol, such as FACADE_CONTROL in Figure 12.

The output of the control grammar is a string with the letters in the form $\langle s, a, r \rangle$, where s is the locator, usually in the form of a split row and column coordinates, a is the attribute that is to be set, and r is the new value of the attribute. The control grammar is evoked after a split operation if a control symbol is available. As an example, let there be a shape with a control symbol FACADE_CONTROL, which then is split into four shapes with locators $(0, 0)$, $(0, 1)$, $(1, 0)$, and $(1, 1)$. After the split operation, the control grammar is evoked,

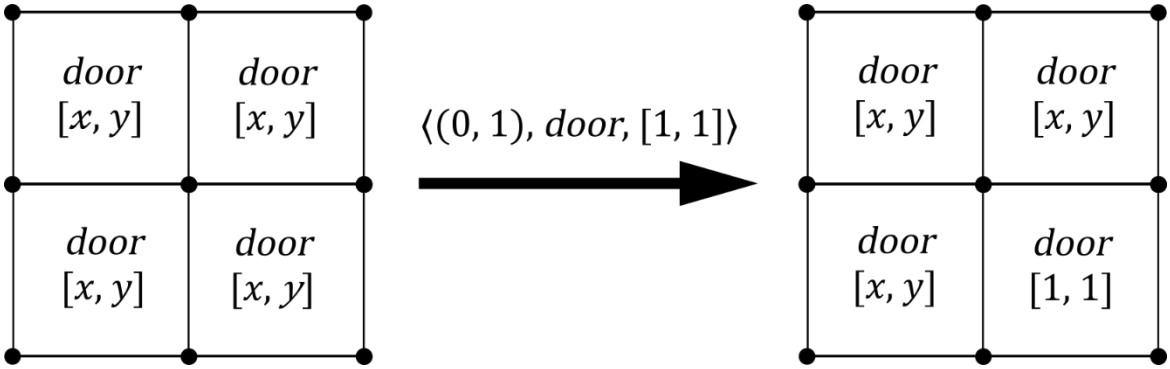


Figure 13. An application of control grammar output. The locator $(0, 1)$ refers to the bottom-right rectangle, which' attribute *door* is set to $[1, 1]$.

resulting in the output of one letter $\langle (0,1), door, [1,1] \rangle$. During the parsing of the control grammar output, the attribute *door* is set to $[1,1]$ for the shape at the location $(0,1)$, as seen in Figure 13.

Alas, a control grammar cannot solve all the placement problems. More complex propagation strategies with adjacency constraints can be solved by WFC. Like control grammar, WFC can be evoked in conjunction with a split rule, where it collapses the result of each new vertex v into form $\langle a, r \rangle$, where a is an attribute and r is the new value of that attribute. Then the attribute a of the shape $k(v)$ is set to r.

Both control grammar and WFC require a control symbol to be evoked. It is possible for both of them to set that symbol for the split shapes, as the control symbol is also an attribute. This makes it possible to use a control grammar and WFC on any split operation. In general, any graph algorithm can be used to change the attribute values of the shapes in the graph. With the attributes distributed, how can they be used to influence the rule selection?

3.5 Attribute Matching

One of the functions of the attribute system is to influence the derivation process. This is done by attaching attributes to rules and then performing attribute matching to filter out unusable rules.

The attribute a_i of the shape and grape grammar rule are compared. If the sets do not intersect then the rule is not considered. Among the rules that do match, one is selected stochastically. The selection can be weighted. The attribute also includes a containment flag c_f. If the containment flag is set for the attribute a_i, the set that is compared to it needs to be a subset of a_i. Note that all the attributes need to match for the rule to be considered.

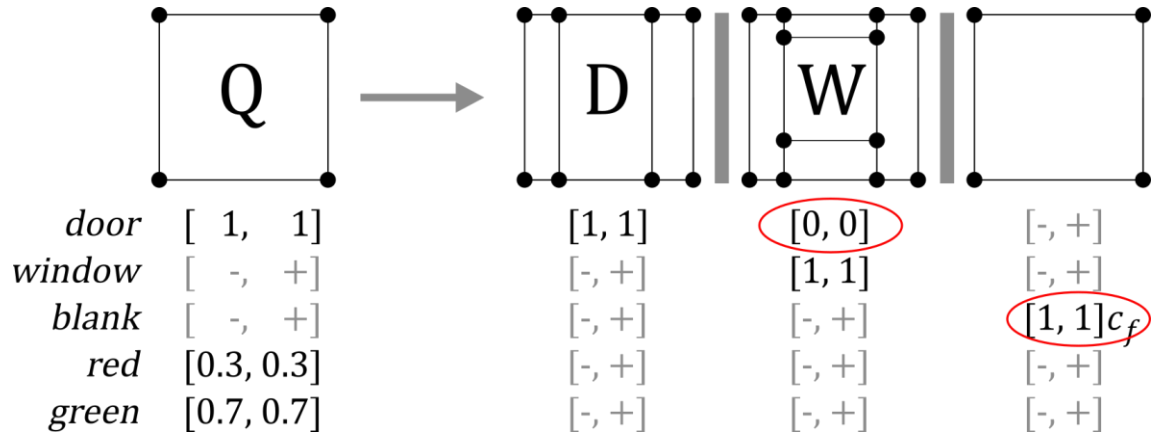


Figure 14. Attribute matching dictates that only the first rule ($Q \rightarrow D$) can be used. The $[-,+]$ is the shorthand for $[-\infty, +\infty]$. The c_f indicates that the containment flag is set.

An example of the attribute matching is shown in Figure 14. The shape with grammar symbol Q has a selection of three possible rules. Below the LHS of the rule, the shape's attributes are written while below the RHS the attributes of the rules are presented. Only the first rule can be actually picked, because the second and third rules have an attribute mismatch with the shape (highlighted with red). The second rule cannot be used because $[1,1] \cap [0,0] = \emptyset$ for the attribute *door*. The third rule cannot be used because $[-\infty, +\infty] \not\subset [1,1]$ for the attribute *blank*.

3.6 Entanglement

One of the biggest strengths of using the graph instead of shapes is that adding context to shapes is very simple. To connect two disconnected shapes, an edge needs to just be drawn between them. This type of connection is called *entanglement*. This can be useful for context-sensitive rules where the context that is not directly connected to a shape is necessary.

Entanglement can also be used for special rule derivation, where if a shape s_1 is processed, so is an entangled shape s_2. This is useful for producing periodic or symmetric productions. To create that kind of entanglement, a mapping M is needed, which maps vertices of a split shape to the vertices of its virtually connected shape. For that mapping to make sense, these shapes must be the same basic shape, meaning that s_2 is s_1 with an affine transformation. When a split operation is done on s_1, a new shape s_2' is created by using the mapping M and the exact same split is done on s_2' as well. Finally, s_2 is replaced by s_2'. By choosing a correct mapping, symmetry can be achieved, as shown in Figure 15.

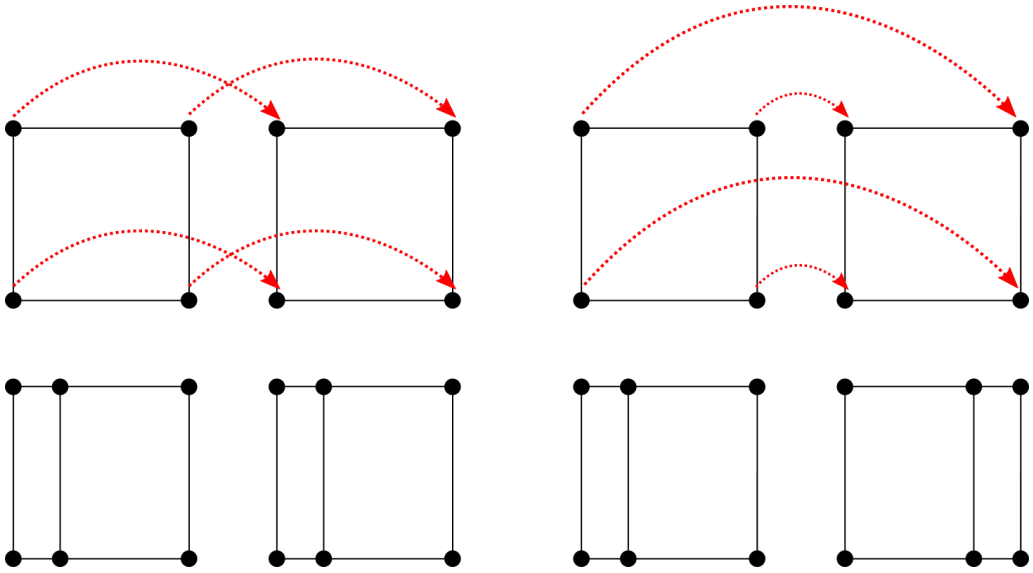


Figure 15. The mapping M in red and the result of a split operation below.

4. Use and Analysis

This chapter provides examples of the use of the algorithm, discusses different aspects of it, and presents its main strengths.

A demonstration application[1] was developed for testing the grape grammar. A user can define grammar rules by implementing a function that takes a shape as the argument and then writing the logic for splitting/replacing that shape. Basic shapes are defined by implementing a common shape interface. Control grammar and WFC behavior can be defined using strings as symbols. Multiple WFC rulesets can be created, and they can be selected to execute by control grammar. Entanglement can be established between two shapes unless they are already entangled with some other shape. The flowchart for the framework can be seen in Figure 16.

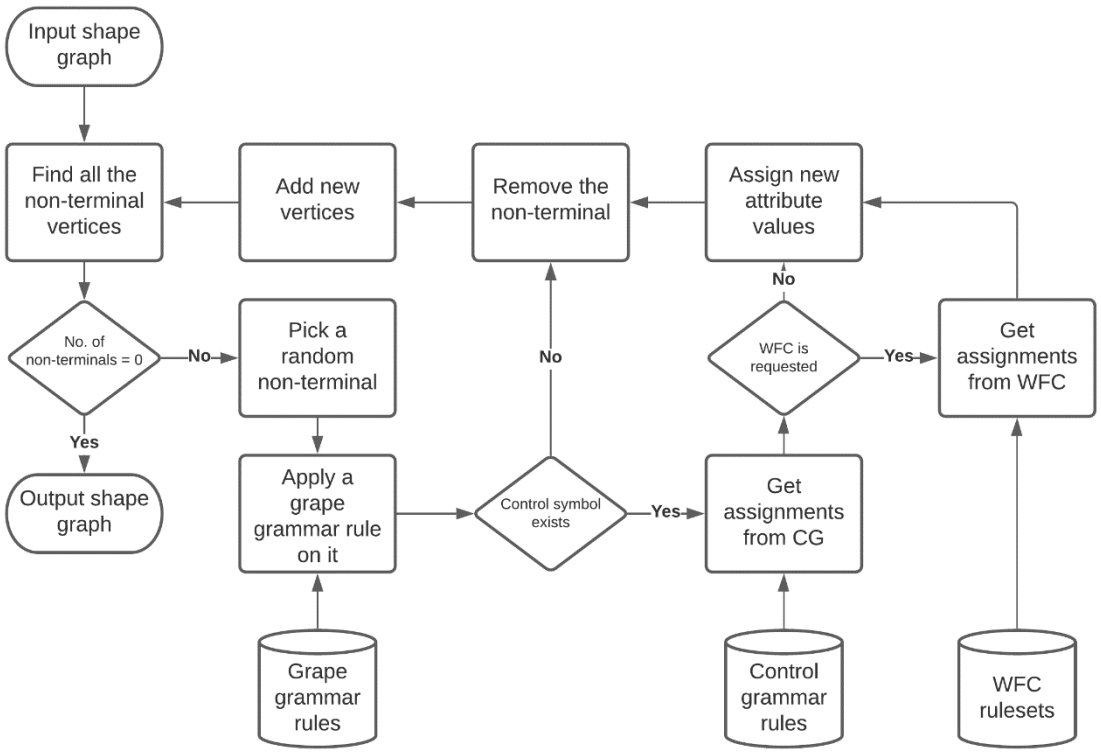


Figure 16. Flowchart of the framework implemented in the demonstration application.

The algorithm can produce a variety of outputs. The variety depends on the rules defined. An example grammar was created for demonstration purposes, which can be seen in Appendix III. As seen in Figure 17, the grammar allows for different window patterns per floor, different doors, and an option to have a stairwell. In addition, the house can have four or five vertical sections (how many windows are per floor). There are also rules that use the entanglement feature. In Figure 18 two side wings are entangled, making them symmetric.

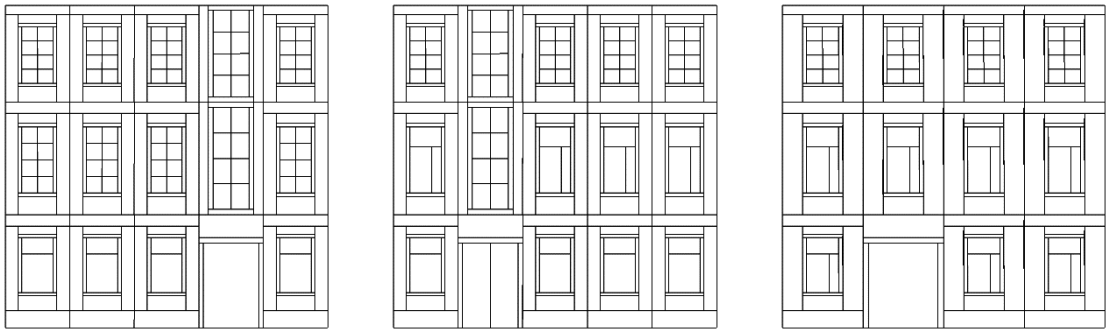


Figure 17. Three productions of the same grammar.

Distributing attributes with WFC can produce similar results to *Kvartal* as seen in Figure 19. This result is achieved by using two different WFC rulesets, illustrated in Figure 20. The first ruleset distributes the color attributes in a way that no two neighboring sections have the same color. The second ruleset determines the roof height so that each section differs from its neighbor while not having sudden changes.

The facades from Figure 17, Figure 18, and Figure 19 were produced by the same grammar but look completely different. The difference was achieved by manipulating the initial attributes of the starting shape. This illustrates the utility of attribute matching. One can generate a large number of buildings for a city with only one grammar, since initial attributes can be generated with a random number generator or by other suitable means.

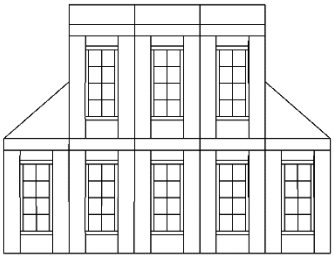


Figure 18. A house with two symmetric side wings.

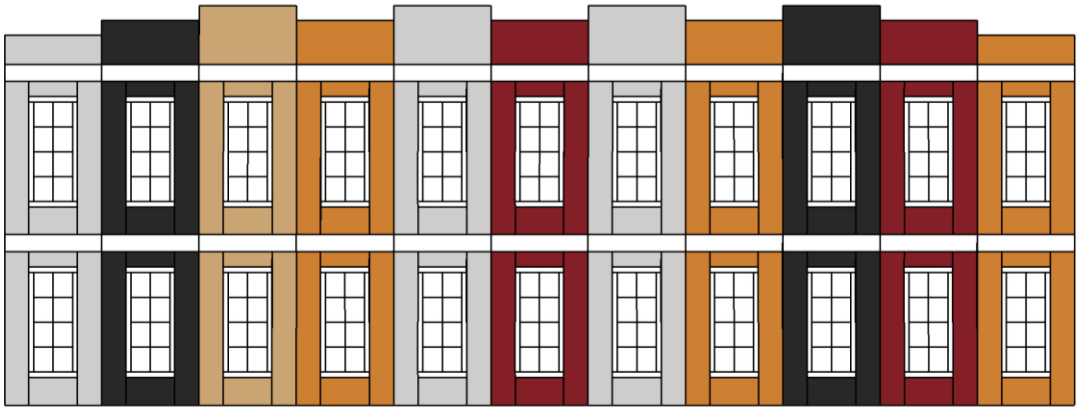


Figure 19. A grape grammar result with two different WFC rulesets.

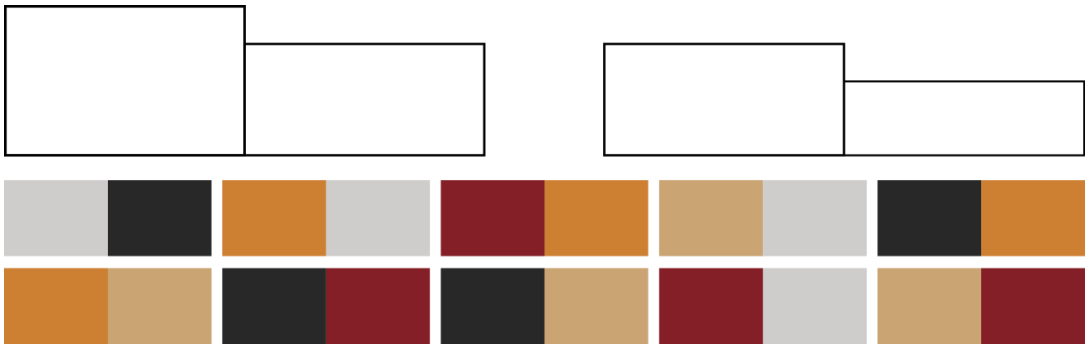


Figure 20. Roof constraints above and color constraints below.

Detailed buildings do not necessarily require many rules. Formal grammars lend themselves well to recursive designs. Instead, the number of rules depends on the *entropy* of the building. The number of rules for a small lodge would be quite low because it can be described with little information. On the other hand, complex architecture, such as *Sagrada Familia*, would require many rules since it is extremely detailed and highly irregular compared to usual architecture (see Figure 21).

The algorithm could be improved upon. In future work, a 3D version of this algorithm should be developed, along with an intuitive user interface for creating grape grammar rules. In the demonstration application, the splitting rules are defined by hand, which is inconvenient and very unintuitive. A graphical user interface would make the creation of the rules much easier, especially for 3D shapes. Additionally, the algorithm should be used in a bigger project, such as a city generator, to test the viability of it. Finally, the algorithm should be optimized to work in real-time applications, since at its current stage takes over three seconds to produce the façades seen in Figure 17, Figure 18, and Figure 19. This would be useful for projects that need to generate virtual urban environments on demand.

Figure 21. Sagrada Familia has high entropy design.

5. Conclusion

This thesis tackled the problem of procedural generation of buildings. To that end, two procedural generation methods – split grammar and Wave Function Collapse algorithm – were mixed together to develop grape grammars, reaping the benefits of both algorithms.

Shape graphs were devised, which are a way to represent shapes as graphs. Connected shapes have edges drawn between them in the equivalent shape graph. Then the grape grammar was defined, which is a formal grammar that works on the shape graph. The grape grammar allows two kinds of operations: split operation, where a shape of the vertex is split into multiple sub-shapes, and conversion operations, where a shape of the vertex is converted into another shape that is not bigger than the original. The two kinds of rules will never produce productions that collide with itself, which makes designing grammar rules very simple.

The attribute system of the grape grammar was inherited from the split grammar. Each vertex has attributes attached to it, and they are propagated to their child-shapes when a split or conversion operation is done on the vertex. Attribute propagation is also done with control grammar and WFC, allowing more complex designs while simplifying the grape grammar rules. Attribute matching is used to better control the derivation process of the grape grammar.

A new way to add context to the vertices was introduced called entanglement. The use of it permits for more complicated context-sensitive rules. The entanglement also makes it possible for the grape grammar to produce symmetric or periodic buildings.

An application and example grammar were created to demonstrate the algorithm. The application allows to use all the presented features of grape grammar. One grammar can produce a variety of buildings. The WFC was demonstrated by generating a façade similar to *Kvartal*. Attribute propagation and entanglement are also shown in neo-classic style building façades.

References

[1] Alissa Walker, "A Tour of 'San Fransokyo,' the Hybrid City Disney Built for Big Hero 6," *Gizmodo*, 2014. [Online]. https://gizmodo.com/a-tour-of-san-fransokyo-the-hybrid-city-disney-built-f-1642066794 (2021.04.2021)

[2] Matt Hill, "Grand Theft Auto V: meet Dan Houser, architect of a gaming phenomenon," *The Guardian*, 2013. [Online]. https://www.theguardian.com/technology/2013/sep/07/grand-theft-auto-dan-houser (29.04.2021)

[3] esri ArcGIS CityEngine. (2020) Tutorial 6: Basic shape grammar. [Online]. https://doc.arcgis.com/en/cityengine/latest/tutorials/tutorial-6-basic-shape-grammar.htm (29.04.2021)

[4] George Stiny, "Introduction to shape and shape grammars," *Environment and Planning B: Urban Analytics and City Science*, vol. 7, no. 3, pp. 343–351, 1980. [Online]. https://doi.org/10.1068/b070343

[5] Maxim Gumin, "WaveFunctionCollapse," *GitHub*, February 2021. [Online]. https://github.com/mxgmn/WaveFunctionCollapse/blob/master/README.md (04.04.2021)

[6] Aristid Lindenmayer, "Mathematical models for cellular interactions in development II. Simple and branching filaments with two-sided inputs," *Journal of Theoretical Biology*, vol. 18, no. 3, pp. 300-315, 1968. [Online]. https://doi.org/10.1016/0022-5193(68)90080-5

[7] Oskar Stålberg. (2020, March) ORGANIC TOWNS FROM SQUARE TILES - a talk by OSKAR STÅLBERG at INDIECADE EUROPE 2019. [Online]. https://www.youtube.com/watch?v=1hqt8JkYRdI

[8] Peter Wonka, Michael Wimmer, François X. Sillion, and William Ribarsky, "Instant Architecture," *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 669–677, 2003. [Online]. https://doi.org/10.1145/882262.882324

[9] Hwanhee Kim, Seongtaek Lee, Hyundong Lee, Teasung Hahn, and Shinjin Kang, "Automatic Generation of Game Content using a Graph-based Wave Function Collapse Algorithm," in *2019 IEEE Conference on Games (CoG)*, 2019, pp. 1-4. [Online]. https://doi.org/10.1109/CIG.2019.8848019

[10] Grzegorz Rozenberg, *Handbook of Graph Grammars and Computing by Graph Transformations*, Grzegorz Rozenberg, Ed.: World Scientific Publishing, 1997, vol. 1-3.

Appendix

I. The Accompanying Files

The accompanying files have the source code of the demonstration application in the "demoapp" directory. The source code can also be found from https://github.com/mathiasplans/shape git repository.

II. Glossary

Euclidean geometry[9] is the most basic geometry that follows the axioms devised by Greek mathematician Euclid.

Set[10] is a series of objects where no two members are equal.

Empty set[10] is a set with no members.

Set union[10] is an operation between two sets that produces a new set with all the objects from the two source sets.

Set intersection[10] is an operation between two sets that produces a new set with all the objects that are common between the two source sets.

Subset[10] of a set S is a set which' members are also in S. Empty set is a subset of all sets.

Formal grammar[11] is set of production rules that describe how to form strings of a formal language.

Polygon[12] is a collection of line segments on a plane that form a closed shape, meaning that the area is split into two or more areas.

Convex polygon[13] is a polygon where all the internal angles are less than or equal to 180 degrees.

Affine transformation[14] is a geometric transformation where lines and parallelism are preserved. The transformation is usually represented by a matrix. Common affine transformations are rotation, skew, translation, and scale.

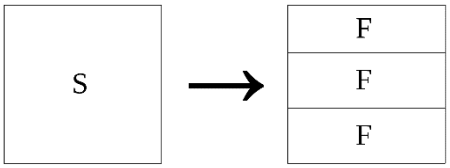
Stochastic[15] system is a system that can be described with a random probability distribution.

Entropy[16] is the average amount of information derived from a random event. To say that something has high entropy is to say that the random event did not produce many equal results, while low entropy means that the random event is more predictable. In other words, entropy measures the randomness of the subject.

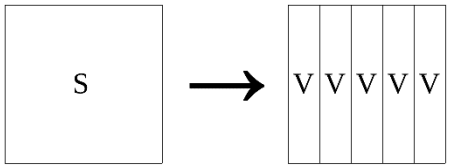
[9] https://en.wikipedia.org/wiki/Euclidean_geometry
[10] https://en.wikipedia.org/wiki/Set_(mathematics)
[11] https://en.wikipedia.org/wiki/Formal_grammar
[12] https://en.wikipedia.org/wiki/Polygon
[13] https://en.wikipedia.org/wiki/Convex_polygon
[14] https://en.wikipedia.org/wiki/Affine_transformation
[15] https://en.wikipedia.org/wiki/Stochastic
[16] https://en.wikipedia.org/wiki/Entropy_(information_theory)

III. Example Grammar

Split rules	Note that the shapes that have no symbol in them are terminal shapes.



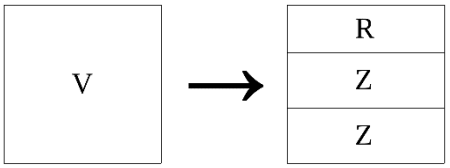
$floors$ $[3,3]$



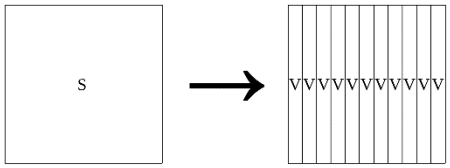
$floors$ $[2,2]$

$wide$ $[0,0]$

The left-most and right-most shapes are symmetrically entangled.

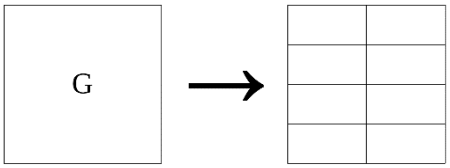


$roofslanted$ $[0,0]$

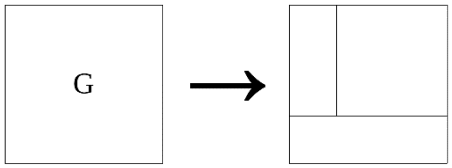


$floors$ $[2,2]$

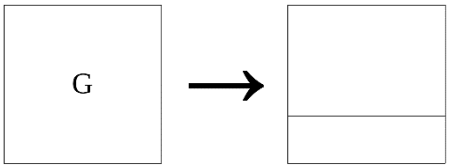
$wide$ $[1,1]$



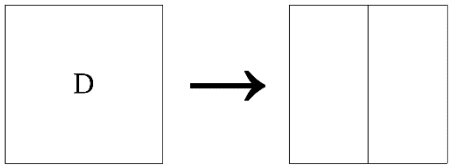
$window$ $[0.8,0.8]$



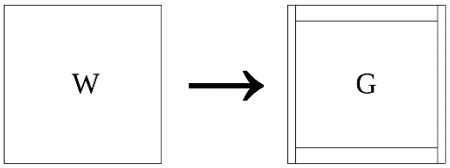
$window$ $[0.5,0.5]$

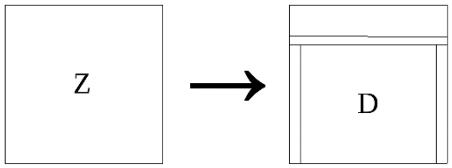


$window$ $[0.3,0.3]$

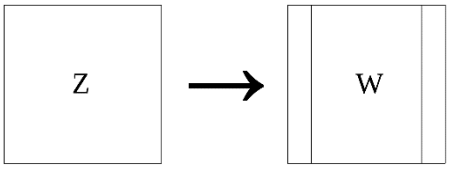


$door$ $[2,2]$

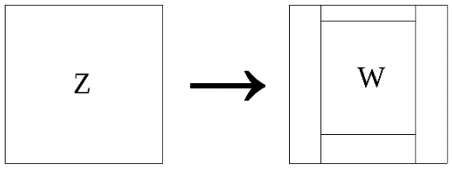




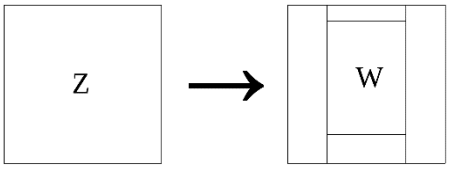
$door$ $[1,2]$



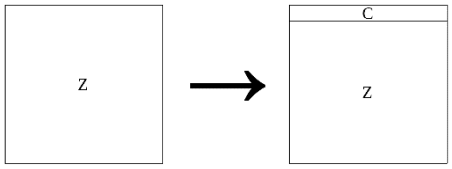
stairs [1, 1]



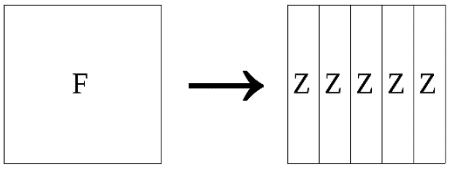
door [0, 0]
cornice [0, 0]
bar [0, 0]
stairs [0, 0]
sections [4, 4]



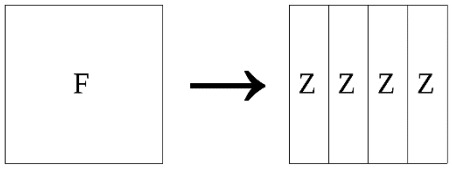
door [0, 0]
cornice [0, 0]
bar [0, 0]
stairs [0, 0]
sections [5, 5]



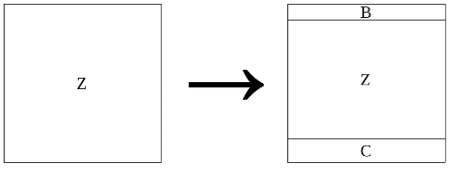
door [0, 0]
cornice [1, 1]
bar [0, 0]
stairs [0, 0]



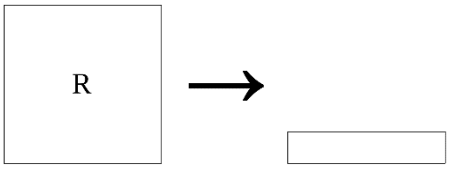
sections [5, 5]

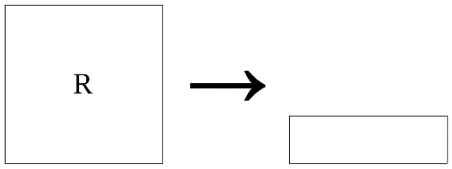


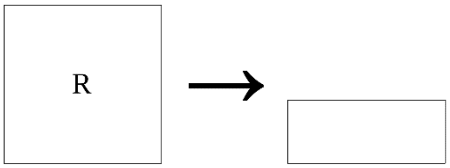
sections [4, 4]

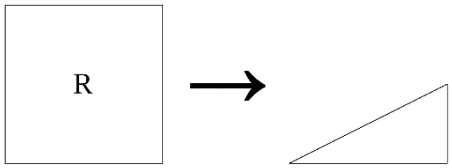


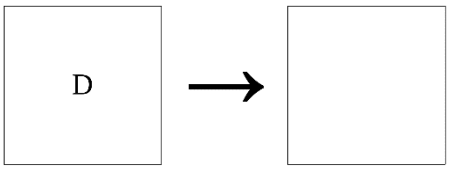
door [0, 0]
cornice [1, 1]
bar [1, 1]
stairs [0, 0]

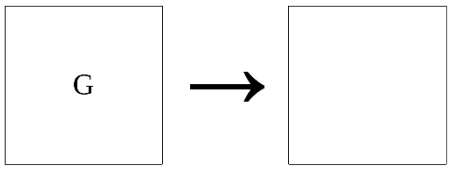
<table>
<tr>
<td>Conversion rules</td>
<td>


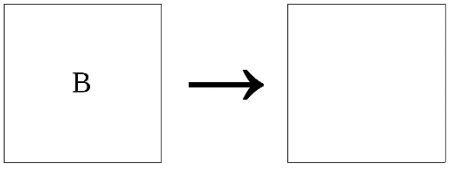
$roofslanted\ [0,0]$
$roofsize\ [3,3]$


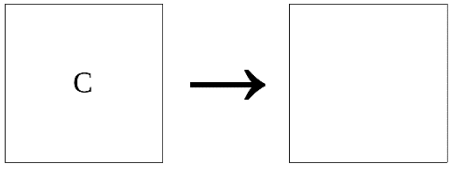
$roofslanted\ [0,0]$
$roofsize\ [2,2]$


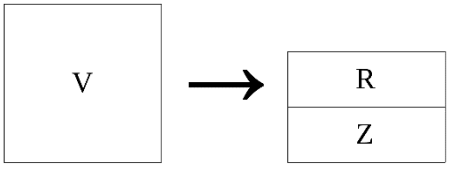
$roofslanted\ [0,0]$
$roofsize\ [1,1]$


$roofslanted\ [1,1]$


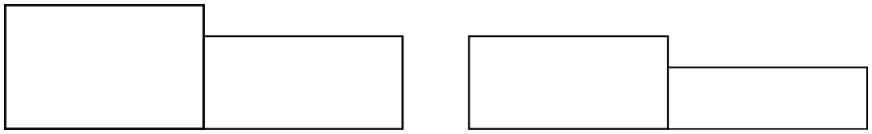
$door\ [1,1]$


$window\ [0,0]$




</td>
</tr>
<tr>
<td>Split + Conversion rules</td>
<td>


$roofslanted\ [1,1]$
</td>
</tr>
<tr>
<td>Control rules</td>
<td>
Note that $\sim$ means that the value is not care.
$Start_1 \rightarrow D, B, C, W_1, W_2, W_3, SE$
$D \rightarrow \langle (0,2), D_0, \sim\rangle \mid \langle (0,2), D_1, \sim\rangle \mid \langle (0,2), D_2, \sim\rangle \mid$
$\langle (0,2), D_0, \sim\rangle, \langle (0,1), S_0, \sim\rangle, \langle (0,0), S_0, \sim\rangle \mid$
$\langle (0,2), D_1, \sim\rangle, \langle (0,1), S_1, \sim\rangle, \langle (0,0), S_1, \sim\rangle \mid$
$\langle (0,2), D_2, \sim\rangle, \langle (0,1), S_2, \sim\rangle, \langle (0,0), S_2, \sim\rangle$
$C \rightarrow \langle (0, all), cornice, [1,1]\rangle$
$B \rightarrow \langle (0,2), bar, [1,1]\rangle$
</td>
</tr>
</table>

	$W_1 \rightarrow \langle (0,1), window, [0.5, 0.5] \rangle \mid \langle (0,1), window, [0.8, 0.8] \rangle$ $W_2 \rightarrow \langle (0,2), window, [0.3, 0.3] \rangle \mid \langle (0,2), window, [0.5, 0.5] \rangle$ $W_3 \rightarrow \langle (0,3), window, [0.5, 0.5] \rangle \mid \langle (0,3), window, [0.8, 0.8] \rangle$ $SE \rightarrow \langle (all, all), sections, [4,4] \rangle \mid \langle (all, all), sections, [5,5] \rangle$ $D_0 \rightarrow \langle (0,0), door, [1,1] \rangle \mid \langle (0,0), door, [2,2] \rangle$ $D_1 \rightarrow \langle (1,0), door, [1,1] \rangle \mid \langle (1,0), door, [2,2] \rangle$ $D_2 \rightarrow \langle (3,0), door, [1,1] \rangle \mid \langle (3,0), door, [2,2] \rangle$ $S_0 \rightarrow \langle (0,0), stairs, [1,1] \rangle, \langle (0,0), window, [0.8, 0.8] \rangle$ $S_1 \rightarrow \langle (1,0), stairs, [1,1] \rangle, \langle (1,0), window, [0.8, 0.8] \rangle$ $S_0 \rightarrow \langle (3,0), stairs, [1,1] \rangle, \langle (3,0), window, [0.8, 0.8] \rangle$ $Start_2 \rightarrow \langle \sim, WFC_1, \sim \rangle, \langle \sim, WFC_2, \sim \rangle$
WFC rules	WFC_1  WFC_2 

IV. License

Non-exclusive license to reproduce thesis and make thesis public.

1. I, Mathias Plans, herewith grant the University of Tartu a free permit (non-exclusive license) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, Procedural Generation of Unique Buildings, supervised by Raimond-Hendrik Tunnel.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons license CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive license does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Mathias Plans
07/05/2021