Perception of procedurally generated virtual buildings

KARL GYLLEUS
Perception of procedurally generated virtual buildings

KARL GYLLEUS
Abstract

Due to the high amount of time it takes to manually design virtual buildings and cities it is often desired to automate the process. Procedural modeling is a way to achieve this by algorithmically generating buildings. By using inverse generation procedural descriptions of buildings can be extracted from a given model. When using a procedural generator to create a city some variance is needed during generation which might lead to unrealistic buildings if not handled correctly.

This thesis investigates which features of buildings are most sensitive to change in regards to their realism. An inverse procedural modeller is implemented in C# using Unity, which can generate split grammar rules from input pictures of building facades and their defined layout. Generated rules can then be used to generate buildings. Photographs of facades on residential buildings are edited in Adobe Photoshop and fed to the implemented generator to recreate real buildings. These buildings and manually altered versions of them are used to carry out an experiment involving independent participants in order to find which features on facades detracts most from their realism when changed.

The findings are that certain features impact a buildings realism more than others when changed. Color and window styles on a facade are especially sensitive. This knowledge can in the future be used to improve building generators such that they are careful in editing the window style and colors of created buildings.
Sammanfattning


Denna avhandling undersöker vilka karaktäristiker på byggnader som är mest känsliga för att bli ändrade vad gäller deras realism. En inversprocedurell generator är skriven i C# och implementerad i Unity, som kan generera split grammar regler från givna bilder av fasader och dess definierade struktur. Fotografier av fasader från bostadshus redigeras i Adobe Photosop och matas till den implementerade generatorn för att återskapa riktiga byggnader. Dessa byggnader och manuellt ändrade versioner av dem används i ett experiment med obeboende deltagare för att undersöka vilka karaktäristiker som fråntar mest realism från byggnaderna när de ändras.

De funna resultaten är att vissa element av byggnadsfasader påverkar byggnaders realism mer än andra. Färg och stil på fönster är i synnerhet känsliga för ändring. Denna kunskap kan i framtiden användas för att förbättra byggnadsgeneratorer sådant att de är försiktiga med att ändra just fönsterstil och färg på skapade byggnader.
Contents

1 Introduction .................................................. 1
  1.1 Background ............................................ 1
  1.2 Purpose and hypothesis ................................ 2
  1.3 Goals .................................................. 3
  1.4 Thesis layout .......................................... 4

2 Related work ................................................. 5
  2.1 Procedural Building Generators ....................... 5
  2.2 Shape Grammar ....................................... 7
  2.3 Split Grammar ........................................ 8
  2.4 Inverse Procedural Generation ....................... 10
  2.5 Realistic Modeling .................................... 12
  2.6 Perception of realism .................................. 13
  2.7 Summary ............................................... 14

3 Implementation ............................................... 15
  3.1 Overview .............................................. 15
  3.2 Building Generator .................................... 16
    3.2.1 Generation process ............................... 18
    3.2.2 Rule selection ................................... 19
    3.2.3 Post processing .................................. 20
  3.3 Automatic Rule Generation ............................ 21
    3.3.1 Rule generator overview ......................... 21
    3.3.2 Rule generation process ......................... 22
  3.4 Summary ............................................... 25

4 Evaluation ................................................... 26
  4.1 Purpose ............................................... 26
  4.2 Stimuli Generation .................................... 27
  4.3 Method ............................................... 29
Chapter 1

Introduction

1.1 Background

Buildings and cities are as old as civilization itself and are central to our societies. In the last few decades emerging technology and techniques have provided the means for cities step into the virtual world. Virtual cities are abundant today and appear in video games, movies and other visualizations of either fictional or real environments. Games such as Grand Theft Auto feature entire virtual cities that players are free to roam and explore. Movies also often feature vast fictional cities that are computer generated. Virtual buildings are also used outside the entertainment industry for urban planning of areas and virtual walkthroughs of urban environments. They can also be used to create city models for disaster management [7]. All of this causes a large demand for the ability to create huge amounts of varied buildings that are still sufficiently detailed and realistic to not break the observers immersion.

The interest of describing and virtually modeling urban environments is not a recent one and has been studied for decades. Modeling virtual buildings can be done manually by a human and with good results, but it is a laborious and expensive task for larger projects. Thus automating the process with acceptable results would often be a better choice. Procedural algorithms are well suited for this purpose as they can generate output automatically, limiting or completely removing the need for manual labor.

Various implementations exists that can generate cities from different input such as a landscape and population density maps or facade
layouts. These generators can have good success in generating cities, which can be seen in the paper Procedural Modeling of Cities [15] where they compare the results of their own generation to a real world city map of Manhattan. Multiple advanced commercial tools for generating virtual cities exists with the most well known being CityEngine developed by Esri. Autodesk Urban Canvas and the BuildR 2 plugin for Unity are two other sophisticated tools for building generation.

![Figure 1.1: Pictures of state of the art city generation tools Esri CityEngine, Urban Canvas and BuildR 2 from left to right.](1)

The current state-of-the-art includes several different methods for procedural generation such as L-systems, split grammar and inverse approaches. The last consists of finding procedural descriptions and rules for a generator from already existing models.

### 1.2 Purpose and hypothesis

While procedural generators are faster they might have issues in the quality of their results. A generated building architecture might be too repeated or otherwise malformed which detracts from its realism.

The purpose of this thesis is to investigate which building elements are most sensitive to change, if any. By doing so procedural generators can be given more freedom to alter the identified non-sensitive components. This way generators are capable of modeling buildings with

---


(2) [https://www.youtube.com/watch?v=rpAnmHbQldg](https://www.youtube.com/watch?v=rpAnmHbQldg)

more variety which can increase the realism of virtually generated urban environments when buildings seldom look alike.

The research question this paper aims to answer is “Are there features of virtual buildings that are more sensitive to change than others in regards to realism?”. Finding the answer to this is of use for future building generators as they can be told which features to be careful of altering when generating buildings. This helps created buildings appear more natural and realistic.

The hypothesis is that certain features are more sensitive to change than others. Window style is thought to be especially important and complex to model correctly [20] which implies that replacing them with arbitrary window styles can lead to the building appearing unrealistic. This may could also be the case for other features on building facades.

1.3 Goals

To test the hypothesis an experiment will be conducted involving independent participants where they will attempt to differentiate reconstructed versions of real buildings from intentionally altered ones. By finding which altered buildings are often deemed unrealistic the specifically changed features of them can be regarded as sensitive to change.

To create the virtual buildings that will be displayed a procedural city generator using state-of-the-art techniques will be implemented. The generator will be able it read and use split grammar rules to generate buildings. A rule extractor will also be implemented to automatically derive procedural rules from photographs of real building facades and its respective defined layout image. This layout image will divide and categorize elements on the facades such that the rule extractor can create a description of the facades structure.

The goals of this study are thus to:

1. Implement an inverse procedural building generator
2. Conduct an experiment to test the hypothesis using the implemented generator
3. Discuss the results and relate them to the hypothesis
As procedural building generation is a huge field the generator will have to be limited in scope and capability. It will be focused on creating cube shaped residential buildings as one would find inside central Stockholm. The insides of buildings will not be modeled as that is a large enough subject to be its own project.

1.4 Thesis layout

Chapter 2 will begin with a highlight of related work within this field. The concepts of generation using split grammar will be thoroughly explained and illustrated to help understand the implementation of the building generator. This is followed by a description of how inverse procedural modeling can be used to extract split grammar rules from a facade layout.

The implemented generator will then be described in chapter 3 mentioning differences between my implementation versus those made by others and explanations why. Possible input and resulting output of the implemented generator will be shown.

This is followed by chapter 4 providing a detailed description of the performed experiment displaying which buildings participants were shown and questions they were asked. Results from the experiment is then displayed followed by a discussion of them and if they align with the hypothesis. Challenges faced during the implementation and conduction of the experiment are discussed as well as potential factors that may have impacted results. Each chapter will have a short summary of its content before ending.

Finally the paper will end with chapter 5 containing a conclusion, summary of the contents and describing potential usages of the results.
Chapter 2

Related work

In this chapter we look at previous approaches for creating virtual buildings and concepts used by them. One such approach is split grammar which is introduced in detail. This is followed by a part looking at previous work done to identify what makes virtual buildings appear realistic.

2.1 Procedural Building Generators

Several procedural city generators have been proposed and implemented with good results. In the paper Procedural Modeling of Cities [15] extended L-systems are used to generate and render complex cities with road networks and properly allotted buildings. A generated city in the paper is based on the geography and population density of Manhattan and is shown to have similar features to the real map of the city.
L-systems, which the above generator is based on, is a formal grammar that was originally created to study growth patterns and replication for organisms [6]. They are well suited for simulating biological growth such as trees and plants. However, cities and buildings have different growth patterns than biological beings. A building does not have continual growth but is rather built in discrete steps and are also spatially constrained whereas L-systems simulate growth in open spaces [24].

Wonka et al. [24] presents another generation method using split grammar which according to the authors provides more flexibility that is required to model buildings of different design and styles. The authors define a grammar based on building facades by identifying and labeling structural components as different regions, for instance balconies and windows. Building upon this the paper Procedural Modeling of Buildings [14] provides a more context aware solution which tries to eliminate unwanted intersections of the buildings geometry, such as windows being cut off by other walls on a building.

An inverse procedural generator using split grammar is proposed by Wu et al. [25] which attempts to extract meaningful procedural descriptions of buildings from given images of facade layouts. This results in an autonomous generation process that reduces the amount of
interaction needed by humans which is useful when generating a lot of models in a large area.

The methods above are focused on creating building exteriors for building cities but leave buildings hollow inside as a result. Attempts to procedurally model the insides of buildings has been done as well with good success [21, 12]. Efforts have also been made to apply constraints such that generated buildings will be structurally sound in regards to physics [23].

Work has also been done to properly model specific parts of buildings. Generation of facade windows has been investigated as they are such integral parts of a buildings style, are context sensitive and have many parameters [20]. Generating rooftops from building footprints [9] and aerial photographs [13] has also been studied. Texture enhancements of buildings to align cellular patterns on facades has also been presented by Legakis, Dorsey, and Gortler [10].

A building’s architecture is also related to the geographic location, culture and time period it was built in. Teoh [19] investigated specific features of traditional East Asian buildings and how they could be modeled while Müller et al. [14] showed how the ancient Roman city Pompeii could be procedurally generated.

2.2 Shape Grammar

Shape grammar is a language that describes transformation and generation of geometric shapes that was introduced by Stiny [18].

A shape in the context of shape grammar is defined as a limited arrangement of straight lines defined in a cartesian coordinate system with real axes and an associated euclidean metric. Lines are defined by their start and endpoints \( \{p_1, p_2\} \). All shapes only have maximal lines, meaning that no line is a part of a possible other line. This is to ensure the uniqueness of a certain shape as it could otherwise be represented in an infinite amount of ways by splitting its consisting lines into smaller lines. Using this definition, a shape \( A \) is said to be a subshape of \( B \) if and only if each maximal line in \( A \) exists in \( B \). Furthermore shapes can be labeled meaning that two shapes identical in geometry would be considered different if they have different labels on their points.

Euclidian transformations can be applied to a shape to alter its lo-
cation, rotation, size and reflection. If there is a transformation from a shape \( s_1 \) to \( s_2 \) they are described as similar.

Using these definitions Stiny formulates shape grammar algorithms that contain four parts.

1. A finite set \( S \) of shapes
2. A finite set \( L \) of symbols
3. A finite set \( R \) of shape rules on the form \( \alpha \to \beta \) where \( \alpha \) and \( \beta \) are labeled shapes.
4. An initial shape \( I \).

The initial shape \( I \) and all shapes used in \( R \) are contained within \( S \) and use symbols from \( L \).

A rule \( \alpha \to \beta \in R \) applies to a shape \( s \) if there exists an allowed transformation \( \tau \) such that \( \tau(\alpha) \) is a subset of \( s \). In the general case, without restricting the forms of transformation, this means that \( s \) is similar to \( \alpha \). The result of applying the rule will be the difference between \( s \) and \( \tau(\alpha) \) combined with \( \tau(\beta) \).

The shape rules are applied one at a time on the initial shape or shapes produced from previously invoked rules. Through this process a set of shapes \( S^* \) can be obtained containing all the shapes without symbols or those with a terminal symbol. This set \( S^* \) is referred to as a language and all its elements are shapes or subshapes of others in \( S \). Thus shape grammar can be used to split an initial shape into smaller components by following the rules of the grammar, but it can also be used to alter the geometry of the initial shape by transforming its smaller components during rule invocation.

### 2.3 Split Grammar

Wonka et al. [24] introduced split grammar for the purpose of modeling buildings. It uses definitions from shape grammar and functions in the same way with a few additions. It tries to minimize human interaction for rule creation in order to speed up the creation of buildings which is especially useful when modeling large areas.

The approach proposed by the authors uses a central database of rules that may contain several matching rules for each shape. As only
one rule at a time can be invoked the generation algorithm must be able to select one of the matching rules for a given shape. For this purpose they also introduce a stochastic process of selection of rules.

In contrast to shape grammar there are no shapes without a symbol. Instead two sets of shapes are defined based on their symbols. The set $N$ contains all shapes with non-terminal symbols and $T$ is the set of all terminal symbols. As in shape grammar a terminal symbol is used to label a shape for which no more rules can be applied to. Thus a split grammar $G$ can be written as $G = (N, T, I, R)$ where $I$ is the set of initial objects and $R$ is the set of rules.

Using this notation the generation process can be described as shown in figure 3.9.

1: $M \leftarrow I$
2: while $|M| > 0$ do
3: $s \leftarrow$ element in $M$
4: Remove $s$ from $M$
5: Select $r \in R$ that applies to $s$ if possible
6: if $r$ exists then
7: Invoke $r$
8: $Res \leftarrow$ resulting shapes of $r$
9: Add $Res$ to $M$
10: end if
11: end while

Figure 2.2: Split grammar derivation flow

By applying the grammar to an initial shape it will split the shape into structural elements which are in turn recursively split until all elements are terminal. The terminal shapes should be individual design elements such as doors or window sills.
A fundamental rule in split grammar is one that takes a shape and splits it into one or more subshapes. This is referred to as a split rule. Another important type of rule is the repeat rule which similarly to a split rule takes one shape and splits it to form others but repeats the sequence of resulting shapes a certain amount of times. These two rules are illustrated in figure 2.3. Both of these rules also specify the sizes of the resulting shapes. Sizes can be absolute or relative [14] which denotes which elements are appropriate to scale and which should have a constant size. It would for example be more appropriate to stretch a wall segment than a door.

2.4 Inverse Procedural Generation

As manually defining the rule set of a grammar can be a laborious effort it is often desirable to generate procedural descriptions autonomously. Inverse procedural methods can achieve this by using one or more complete models as input and using their characteristics and features to extract a grammar. Images of facade layouts that describe the structural components of building facades can be used as input to generate procedural descriptions of buildings [25]. Aside from improving the modeling process the resulting grammar is also likely to be more meaningful and concise. Defining a grammar of good quality manually also requires more knowledge and experience than creating input data for an inverse procedural algorithm [25].
These layout images can be created by hand using specialized tools or image editors. They can also be generated from facade images stochastically [16] which can save effort and time but is significantly more complex to implement.

The algorithm proposed by Wu et al. [25] consists of the steps shown below and is capable of splitting a facade layout into a procedural description.

1: An empty grammar is initialized.
2: A non-terminal region that has no rule is chosen for splitting in a fixed order.
3: A rule is selected for the chosen region and added to the grammar.
4: If there still exists one or more non-terminal regions, jump to two.
5: Terminate.

By identifying symmetry on the building facades certain areas can be found to be repeatable meaning they can be represented through a repeat rule. This might make the grammar more concise which is often desirable [25]. A cost function can be used to determine whether a split or repeat rule is more appropriate for a certain region.

Symmetry on the building facade is found through a bottom-up approach that finds all areas that are repeated at least once. As the input size of a building facade is small an exhaustive search of all areas can be performed [25]. By starting with all terminal regions that occur two or more times and merging them with other reoccurring areas all repeated regions are found.
As seen in figure 2.5 above, areas that are adjacent and have neighboring sides of equal length may be merged.

### 2.5 Realistic Modeling

Though it is not well established which features of buildings are most sensitive to change in regard of realism, certain characteristics are known to be detrimental. One example is inappropriate intersections of components of a building such as a wall intersecting with a window. It is thus important to have some information of context of the structural elements when applying procedural methods which may result in intersecting shapes to ensure it is handled properly. An occlusion query that specifies which objects that may occlude specific types of other objects can solve this issue [14].

The color in images attribute a lot to its realism and can make them appear unrealistic if handled incorrectly [8]. By extension this also applies to virtual buildings as they become images when rendered. Examples are the color of components having a different shade which does not match the rest of the facade or simply having a color which
people do not expect to find on real buildings. As seen in the experiment by Kaya and Crosby [5] the colors black and purple were not at all associated with residential building facades.

2.6 Perception of realism

No earlier work was found that aimed to specifically identify the most important building factors for perceiving them as realistic. However, much research has been done on how a computer image can trick the brain to believe it is real. Ferwerda [2] describes three different types of realisms which are physical realism, photo-realism and functional realism. Physical realism requires an image to provide the same visual stimulation as the scene whereas photo-realism needs to produce the according visual stimulation. This means that physical realism needs to capture the exact lighting, colors and shapes in the scene to make the image identical, which puts a lot of demand on the quality and correctness of the image. Photo-realism only needs to provide an image which is for the human mind indistinguishable from the original scene which slightly lowers the demands. Functional realism merely tries to convey information by simulating a real world image but does not demand that it looks realistic.

The authors in [6] briefly evaluate the realism of several procedural city generators but does not give great motivation as to why a city looks realistic nor how it is measured.

A study conducted on American university students [5] found that residential buildings are more strongly associated with certain colors. The most associated colors in descending order were blue, red, gray and yellow. Black and purple had no association shown in the results. Furthermore residential buildings were more likely to be imagined in low saturated colors rather than strong and lively. While this gives an indication of suitable colors for realism of generated buildings the authors state that these choices may depend on an individuals past experiences and feelings associated with colors and types of buildings. They also mention that this means that associated colors may differ depending on the culture and nationality of the observer.

Tutzauer et al. [22] studied how effectively humans can categorize building categories from viewing virtual buildings. They came to the conclusion that while industrial facilities and single family build-
ings are easily distinguished by their geometric representation multi-
family buildings, buildings with shops and office buildings needs additional textural information to help separate them. Spross [17] compared differences in perception between real and virtual environment and found that estimating distances was harder in virtual spaces, which is strengthened by earlier research [3].

2.7 Summary

There exists several well known approaches for generating virtual buildings. One of the most notable in recent times is split grammar which uses grammar rules as procedural descriptions of buildings. This concept has been built during the last decade to explore automatic rule generation and improved realism of generated buildings. The area of generating realistic buildings is a huge field where the roofs, windows and building types are all individual problems to solve and vary with regard to architectural style and culture. No earlier work was found that tried identify which features of buildings attribute most to its realism. Previous work has however shown that certain colors have been shown to be associated with specific building types and that humans can identify most types of buildings by category when viewing virtual representations of them.
Chapter 3

Implementation

In the following section the implemented building generator and rule generator are described. Both of these build upon the concepts of split grammar mentioned in the previous chapter. An overview of the implementation will first be given where the flow of the algorithms are shown from input to rendered buildings. The building generator implementation will then be described in detail followed by a similarly detailed description of the rule generator.

3.1 Overview

The full implementation uses two distinct parts for creating split grammar rules of facades and for generation of buildings. The building generator reads split grammar rules from an input text file following a defined format which it uses to generate the buildings. It was implemented first to ensure that buildings could be generated at all and was easily testable by manually writing rules.

The rule generator was implemented after making sure that the building generator worked as intended. It was created to automatically produce grammars faster and of higher quality compared to writing them manually. As input it takes a photograph of a building facade and a layout image that describes the structural components on the facade. Using these two it generates split grammar rules that are written to a text file as well as extracting appropriate textures from the input facade photograph.
3.2 Building Generator

The generator was implemented in the game engine Unity with scripts written in C#. Unity provides a runtime environment and rendering result of the generator. The buildings generated were required to be rendered in real time without substantial delay as they should be able to be used in interactive scenes. If the buildings are rendered at a low frame rate it can negatively impact the immersion of observers [1].

The implemented generator uses an algorithm based on split grammar as defined earlier. A rule interpreter will read rules line by line
from an input text file to create a rule database in memory when starting. The defined rules that the generator recognizes are *split*, *repeat*, *decompose* and *protrude*. Split and repeat rules function as described earlier by splitting one shape into one or more other shapes. The decompose rule resembles a reduction in dimensionality and splits a cube into flat facade walls oriented accordingly. A protrude rule changes the size of a shape on a set axis, most commonly used to protrude elements from the facade wall.

\[
\begin{align*}
\text{Start} & \rightarrow \text{split}(Y) \{ 1: \text{Ground} \mid N: \text{FloorsLeft} \mid 1: \text{Top} \} \\
\text{FloorsLeft} & \rightarrow \text{split}(Y) \{ 1: \text{Floor} \mid N: \text{FloorsLeft} \} [\text{minY: 1}] \\
\text{FloorsLeft} & \rightarrow \text{split}(Y) \{ N: \text{Floor} \} [\text{maxY: 1}] 
\end{align*}
\]

Figure 3.2: Example of a simple rule set usable by the implemented generator

All rules follow the format shown in 3.2 where the initial shape is stated followed by which rule type and which axis it should operate on. The resulting shapes of the rule and their respective sizes are then given. Note that the example above is written using split rules recursively and can be simplified with a repeat rule:

\[
\begin{align*}
\text{Start} & \rightarrow \text{split}(Y) \{ 1: \text{Ground} \mid N: \text{FloorsLeft} \mid 1: \text{Top} \} \\
\text{FloorsLeft} & \rightarrow \text{repeat}(Y) \{ 1: \text{Floor} \}
\end{align*}
\]

If a shape size is described as ‘N’ it is regarded as a flexible element and will be transformed in scale such that the sum of all resulting shapes size will equal the initial shapes original size on the corresponding axis. What this means in practice is that elements with a flexible size will scale to fill potential gaps in the facade if the size of the subshapes do not add up evenly to the parent size. Finally constraints on the initial shape can be optionally declared to limit which cases a rule will match.
Figure 3.3: Right column shows possible result of rules shown in 3.2 using the textures for Top, Floor and Ground as shown in order in the left column.

### 3.2.1 Generation process

The generator processes one shape at a time and puts the resulting shapes of rule invocation in a queue. If an element has no matching rule it is considered to be a terminal shape and is skipped. When the queue of shapes left to process is empty the generation is considered complete and the algorithm terminates.

When an element is identified as a terminal shape a texture will be applied to it. This is done by finding an image file that matches the name of the shape and applying it. Shapes that are not terminal do not need texture as they will be broken down into terminal ones that will cover the non-terminal shape’s entire area. A grammar that can end in shapes without textures will thus be regarded as malformed, but still produced, by the generator as all terminal regions should have an available texture to apply.

The flow of the algorithm is very similar to the one seen in figure 3.9:
1: $Rules \leftarrow \text{Read rules from given rule text file}$
2: $shapesLeft \leftarrow \text{Starting shape}$
3: $\textbf{while} |shapesLeft| > 0 \textbf{ do}$
   4: \hspace{1em} $currentShape \leftarrow \text{first element in } shapesLeft$
   5: \hspace{1em} \text{Remove } currentShape \text{ from } shapesLeft$
   6: \hspace{1em} $matchingRule \leftarrow \text{rule } \in Rules \text{ that applies to } currentShape$
   7: \hspace{1em} $\textbf{if} matchingRule \text{ exists } \textbf{ then}$
   8: \hspace{2em} $\text{Invoke } matchingRule$
   9: \hspace{2em} $res \leftarrow \text{resulting shapes of } matchingRule$
  10: \hspace{2em} $\text{Add } res \text{ to } shapesLeft$
  11: $\textbf{else}$
  12: \hspace{1em} $\text{Apply texture to } currentShape$
  13: $\textbf{end if}$
14: $\textbf{end while}$
15: $\text{handleOcclusion()}$
16: $\text{createRoof()}$
17: $\text{return}$

Figure 3.4: Flow of implemented generation algorithm

### 3.2.2 Rule selection

The rule selection in step 6 prioritizes protrusion rules before others. This is done since a shape will be removed from the list of shapes left if it is split, but not if it is protruded. Thus invoking a split rule before protruding would result in the protrusion never taking place. An example of this would be a window that is protruded negatively into the wall where it is split into a lower and upper part. To avoid infinite protrusion each shape can only protrude once after which all matching protrusion rules will be ignored.

If a shape has no valid protrusion rules then it will randomly select one of the remaining matching rules as the algorithm has no way of knowing which one is most fitting. This means that the generation process is not deterministic if a shape has multiple matching rules, except for the case of one protrusion rule and one non-protrusion rule. This can be desired if some variation during building creation is desired.
3.2.3 Post processing

After the generation loop in 3.4 ends some post-processing is done in steps 15 and 16. Firstly occlusion on the buildings is handled as some elements might be inappropriately covered by others, as described in chapter 2.5.

![Example of collision with intersecting facade wall](left) and same building with occlusion handling (right)

As the implementation only aims to produce cube shaped buildings one at a time sophisticated occlusions handling is not required. The only parts where it is needed is near the edges where facades facing different directions intersect. If occlusions is not handled elements with negative depth close to the edges will be covered by the intersecting facade as seen in figure 3.5. Since occlusion on generated buildings for the types of buildings generated only appears in this case it can be handled easily through ray-tracing. For each element a ray is cast in the opposite direction of its normal. If the shape that the ray hits is not the expected one it is regarded as an occluder and is adjusted accordingly. Occluding shapes are made thinner in the direction of its normal such that it will no longer be blocking the occluded shape.

After the occlusion handling a roof is created for the building. The same model is used for all generated buildings and stretched to fit its dimensions. The model was created manually using the program Blender and was made to resemble a typical roof of a residential building in Stockholm.
Figure 3.6: Roof model used for buildings. The rightmost image shows the roof model attached to a textureless box symbolising a building.

The result after roof creation can be seen in figure 3.3.

3.3 Automatic Rule Generation

3.3.1 Rule generator overview

The rule generator is capable of creating split grammar rules to generate a given facade. As input the rule extractor takes an image of a facade and an accompanying facade layout. The latter describes the structure of the facade by using colored rectangles to represent terminal shapes. Each rectangle symbolizes a structural element or area such as a window or pillar but can also be a whole floor if desired. Rectangles of the same color are interpreted as the same type of element when producing the grammar meaning they will share the same name and texture.

Figure 3.7: A facade image and a simple facade layout
In figure 3.7 above a facade layout is shown which will produce a grammar similar to the example one in figure 3.3. The orange and purple parts are equivalent to Ground and Top respectively while Floor has been split into a green and blue area each. This is done since if Floor is described by a single color multiple floors will blend into a single large one in the eyes of the rule extractor as there will be no separation between them. If the images in 3.7 is fed to the rule extractor it produces the following output:

\[
\begin{align*}
B & \rightarrow \text{split}(Y) \{ 1.93: \text{BGround} | 2.87: \text{B1} | 0.58: \text{BWall} | 1.60: \text{BTop} \} \\
B1 & \rightarrow \text{repeat}(Y) \{ 0.57: \text{BWall} | 0.85: \text{BFloor} \}
\end{align*}
\]

In this example the building name is \( B \) which the algorithm prepends to all its subshapes to differentiate them from other buildings as they may have subshapes with identical names.

As can be seen in figure 3.8 above the result is similar to the building in 2.3 but has less floors. It still has one more floor than in the given input photo which is the result of the repeat rule. How the rules were created from the given facade layout is explained more in detail below.

### 3.3.2 Rule generation process

Rules are generated top-down by splitting the layout image into smaller regions. A region is an area on the facade layout image that contains
one or more colored rectangles. If a region contains only one colored rectangle it is a terminal region, which is equivalent to a terminal element during building generation. Multiple regions containing rectangles of the same color in the same order are considered equal and will share name and rules. Rule extraction is done as following:

1: \( \text{regionsLeft} \leftarrow \text{Region containing all terminal regions in layout image} \)
2: \( \textbf{while} |\text{regionsLeft}| > 0 \textbf{do} \)
3: \( \text{currentRegion} \leftarrow \text{first element in } \text{regionsLeft} \)
4: \( \text{Remove currentRegion from regionsLeft} \)
5: \( \textbf{if currentRegion is terminal OR currentRegion has an existing rule then} \)
6: \( \quad \text{skip to next iteration} \)
7: \( \quad \textbf{end if} \)
8: \( \quad \text{split} \leftarrow \text{currentRegion split on X or Y axis decided by maximizing amount of resulting regions} \)
9: \( \quad \textbf{if split results in more than 1 region then} \)
10: \( \quad \quad \text{repeats} \leftarrow \text{repeated areas in split} \)
11: \( \quad \quad \textbf{for all } r \text{ in } \text{repeats do} \)
12: \( \quad \quad \quad \text{repeatedArea} \leftarrow \text{region containing elements in } r \)
13: \( \quad \quad \quad \text{Update split to replace all elements of } \text{repeatedArea} \text{ with one } \text{repeatedArea} \)
14: \( \quad \quad \text{Write repeat rule to output file} \)
15: \( \quad \quad \textbf{end for} \)
16: \( \quad \text{Write split rule to output file} \)
17: \( \quad \text{Add all regions in split to regionsLeft} \)
18: \( \quad \textbf{end if} \)
19: \( \textbf{end while} \)
20: \( \textbf{return} \)

Figure 3.9: Implemented rule generation algorithm

The algorithm iterates through a queue of regions, splitting them one by one. The queue initially contains the starting region spanning the whole input layout image. If the region currently being analyzed only contains a single rectangle then it is a terminal region and does not need any rules. It is also checked if there already exists a rule for this type of region to avoid redundancy.

Which axis to split the region on is then chosen based on which
axis results in the most subregions. In figure 3.10 the Y-axis would be chosen as it results in 6 regions compared to 2 on the X-axis. A higher amount of subregions is chosen as it is assumed to have a higher chance of containing repeated regions which is desired as repeated areas can be described by repeat rules. A region split has to result in more than one subregion to have any relevance as a rule, otherwise it is skipped.

If the list of resulting regions of the chosen split contains a subset of regions that can be described by a repeat rule the repeatable regions are grouped into a new region that will be on the left hand side in the found repeat rule. This new region replaces the all the subregions it contains in the chosen split.

Iterative splitting of regions is continued until the queue of regions left is empty. If there were no errors during splitting then the algorithm will have produced a set of rules that is capable of splitting a whole facade into terminal shapes, meaning it is a usable grammar.

The rule generator has no perception of depth as the input images are two-dimensional and thus does not generate protrude rules. These rules have to be manually entered during or after the generation process if desired. Furthermore the algorithm cannot know which components are most suitable for scaling. All shapes are defined as scalable to avoid holes in the facade upon building generation. If this is not desired the rules will have to be manually altered.

After rule generation is completed texture extraction is taken place by cutting areas from the input facade photograph. This is done once for each type of terminal region. Extracted textures are saved as im-
ages with appropriate names for their elements which enables the building generator to find them.

3.4 Summary

An inverse procedural modeler was implemented in two parts. The first generated buildings from split grammar rules while the other part generates such rules from input facade images and accompanying layouts. The building generator performs post processing to handle possible occlusion of windows and to attach a pre-modeled roof to the generated buildings. Using the implemented rule generator speeds up rule creation as opposed to writing them manually.
Chapter 4
Evaluation

In the following chapter we use the previously described implementation to conduct an experiment. The setup of this experiment and the steps taken to carry it out are described in detail. Results are then shown and used to test the hypothesis.

4.1 Purpose

To evaluate the implementation and determine which features of buildings were most sensitive to change in regards to realism an experiment was conducted. Using the generators described above three buildings were created from real photographs of facades inside Stockholm (see 4.3). These buildings shall be referred to as real buildings. For each of these real buildings altered versions were created where the split grammar rules were manually edited to change the appearance of the generated facade. Each real building had a hypothesised real, hypothesised unreal and a colored version. These buildings will be referred to as:

- $B_R :=$ real version of a building $B$
- $B_C :=$ colored version of a building $B$
- $B_{HR} :=$ hypothesised real version of a building $B$
- $B_{HUR} :=$ hypothesised unreal versions of a building $B$

Table 4.1 below shows all different building types generated and their modified features.
Figure 4.1: Table showing features that were modified (M) or same as the original real building (O) for all different building types.

The aim of the experiment was to determine if some changed features on facades lead people to regard them as unrealistic. A null hypothesis $H_0$ can be written as:

$H_0$: Altered buildings and real buildings will have the same average selections as real.

### 4.2 Stimuli Generation

The buildings generated for the experiment were chosen such that they would have some variety in style. All were residential buildings. To generate the procedural rules using the implemented rule generator input images of facades were needed. These images were gathered by myself by taking photos of buildings in Vasastan inside Stockholm. The camera used was the rear camera of a Samsung Galaxy S7 phone which takes 12-megapixel photographs which was deemed enough for this purpose.

The facade images need to be entirely front facing in order to be used as valid input for the rule generator. Since the images were taken on street level they needed to be perspective corrected using image editing software. The images were also cropped to remove everything except the facade.
The rule generator also requires a facade layout to understand the structure of the facade from which to extract the rules. These were made using image editing software by appropriately applying colors over the elements of the facades. The image pairs were then fed to the rule extractor which created a fitting procedural description of a building with the facade. The produced rules were then manually altered to fix some irregularities and describe depth of the elements. The created split grammar rules could then be fed to the building generator to instantiate the buildings shown in 4.3.

Altered versions were created by manually editing the generated split grammar rules by removing elements or replacing them with elements of other styles or colors. Beige\textsubscript{HR} had its protruding pillars removed, Beige\textsubscript{HUR} had its window style replaced and Beige\textsubscript{C} changed the beige color to red. Brown\textsubscript{HR} had some added details to the top of the facade as well as some added small round windows, Brown\textsubscript{HUR}
had its window style changed as well and $Brown_C$ was recolored to blue. $Brick_{HR}$ had an extra pillar added as well as some details removed, $Brick_{HUR}$ had its window style and front door replaced and $Brick_C$ was painted yellow.

The choice of colors for the buildings with altered colors was based on an earlier study [5] which indicated that residential buildings are most often associated with the colors blue, red and yellow.

### 4.3 Method

#### 4.3.1 Experiment setup

To perform the experiment a computer with a connected monitor and keyboard was required. Since a low frame rate could impact user experience the computer would also need to be capable of rendering the generated buildings in real time with good performance. Different computers were used to carry out the experiments as it was done in locations that would fit the participants best, but all computers used met the performance requirement. For reference the main computer used for the experiment had an Intel i5-3570K @ 3.40 GHz CPU, NVIDIA GeForce GTX 960 graphics card and 8 GB of RAM.

Documents meant to introduce participants to the project and their task were also prepared beforehand as well as questions for the participants.

#### 4.3.2 Experiment procedure

The experiment was conducted with one person at a time with a total of 10 ($8M:2F$) participants. All persons participating were in the ages 21-24 and stated that they are often inside Stockholm. They were all active students in different areas, six of them being computer science students. No participants claimed they had more knowledge about architecture in Stockholm than an average person. Two of the participants were previously familiar with procedural generation concepts.

The participants were informed of the nature of the experiment and what their task was. They were also heavily encouraged to think loudly when observing the buildings.

Participants were not informed of the amount of real buildings, total amount of buildings nor which specific features to look for.
Participants were shown one building at a time and had to verbally choose if a building was real or not before proceeding to the next using an on-screen button as seen in figure 4.4. The buildings were shown in randomized order but following a set pattern. The series of displayed buildings were composed by subsequences, each beginning with a random version of the Beige building, followed by a version of Brown and then one from Brick. Each displayed version was made sure not to have been displayed earlier in the same series. Thus after 4 subsequences the participant will have seen all versions of the buildings in a semi-random order.

Even though buildings in the real world are often surrounded by trees, signs and sidewalks no such props were added to the scene. This was done to put more emphasis on the building itself rather than the scene surrounding it.

The participants could move the camera around in the scene using a keyboard in order to view different perspectives of the buildings. By allowing them to move and control the camera it could help participants see the spatial boundaries and get a temporal feeling within the scene [4].

Some ending questions were asked after the experiment asking them how sure they were of their answers and if there were any specific feature changes they thought were easy to spot.

4.4 Results

The columns in 4.5 shows the total amount of times each building got selected as real for each respective series. Each series contains all
buildings in a random sequence.

<table>
<thead>
<tr>
<th>Building</th>
<th>1st series</th>
<th>2nd Series</th>
<th>3rd Series</th>
<th>Total (of 36 possible)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BeigeₐR</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>18 (60%)</td>
</tr>
<tr>
<td>BrickᵅC</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td>11 (36.6%)</td>
</tr>
<tr>
<td>BeigeₚHR</td>
<td>7</td>
<td>6</td>
<td>4</td>
<td>17 (56.6%)</td>
</tr>
<tr>
<td>BeigeₐHR</td>
<td>7</td>
<td>8</td>
<td>7</td>
<td>22 (73.3%)</td>
</tr>
<tr>
<td>BrownₐR</td>
<td>7</td>
<td>5</td>
<td>7</td>
<td>19 (63.3%)</td>
</tr>
<tr>
<td>BrownᵅC</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>12 (40%)</td>
</tr>
<tr>
<td>BrownₚHR</td>
<td>2</td>
<td>6</td>
<td>7</td>
<td>14 (50%)</td>
</tr>
<tr>
<td>BrownₚHR</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>12 (40%)</td>
</tr>
<tr>
<td>BrickₐR</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>12 (40%)</td>
</tr>
<tr>
<td>BrickᵅC</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2 (6.6%)</td>
</tr>
<tr>
<td>BrickₚHR</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>11 (36.6%)</td>
</tr>
<tr>
<td>BrickₚHR</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>5 (16.6%)</td>
</tr>
</tbody>
</table>

Figure 4.5: Amount of votes as real for each building.

Figure 4.6: Total number of times each building got selected as real.
We can now test the null hypothesis $H_0$ formulated in chapter 4.1. Let $\mu_i$ denote the average votes as real for building type $B_i$. If $H_0$ is true then $\mu_i = \mu_R$ for all building types $B_i$ where $\mu_R$ is the average votes as real for real buildings $B_R$. To test this an analysis of variance [11] is performed. It is assumed that the population of votes are normally distributed and that all values are independent. Equal variances for building types are also assumed.

An $F$ test [11] is chosen to test the null hypothesis. If $H_0$ is true then $\sum_i \alpha_i = 0$ where $\alpha_i$ is the difference between the mean of a building $B_i$ and the grand mean of all buildings. Estimated values of $\alpha$ are produced from the observed answers and are shown in figure 4.8.

Let $n = 9$ be the number of answers per building type and $I = 4$ be the number of different building types. Using this we get the following

<table>
<thead>
<tr>
<th>Building</th>
<th>$B_R$</th>
<th>$B_C$</th>
<th>$B_{HR}$</th>
<th>$B_{HUR}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>5.44</td>
<td>2.77</td>
<td>4.77</td>
<td>4.33</td>
</tr>
<tr>
<td>Median</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Std. Dev.</td>
<td>1.26</td>
<td>2.15</td>
<td>1.62</td>
<td>2.63</td>
</tr>
<tr>
<td>Std. Error</td>
<td>0.42</td>
<td>0.72</td>
<td>0.54</td>
<td>0.88</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>1.1125</td>
<td>-1.5575</td>
<td>0.4425</td>
<td>0.0025</td>
</tr>
</tbody>
</table>

Figure 4.7: Number of selections as real for non-real buildings compared to the average results of real buildings

Figure 4.8: Statistical values for provided answers.
values:

\[
MS_{\text{between}} = \frac{n\sum_i \alpha_i^2}{I-1} = 11.58 \\
MS_{\text{within}} = \frac{\sum_i \sigma_i^2}{I} = 3.94 \\
F = \frac{MS_{\text{between}}}{MS_{\text{within}}} = 2.94
\]

Since \( F > 1 \) we can reject the null hypothesis given that the made assumptions are true.

![Bar chart showing selections of buildings](image)

Figure 4.9: Selections of most realistic buildings during pair showing

### 4.5 Sensitivity of facade elements

By rejecting the null hypothesis it is shown that certain changes to building facades are more sensitive to change than others. It is of further interest to specifically identify which building features these changes were related to.

As can be seen in figure 4.6 the building variations least chosen as real were the colored versions \( B_C \) for each building. The hypothesized unrealistic facades \( B_{HUR} \) also had a low amount of selections as real with the exception of \( Beige_{HUR} \). All buildings \( B_{HUR} \) had their windows altered from its corresponding real building. Figure 4.8 also
shows that buildings of type $B_C$ and $B_{HUR}$ also had the most deviation in results. I consider the results reliable as real buildings $B_R$ had the highest average votes as real.

From this we can conclude that both recoloring of facades and changing the window style on facades are sensitive to change in regards of realism. The building $Beige_{HUR}$ was the only altered building with more votes as real than its real counterpart. This implies that a building with altered windows can still be at least as realistic as a real one if the windows are still perceived to fit the facade. Figure 4.9 further strengthens this theory as $Beige_{HUR}$ received a lot more votes than $Beige_{HR}$ while $Brick_{HUR}$ received almost no votes when compared to $Brick_{HR}$.

When participants were asked if they found any features to be easier to spot when changed 7 out of 10 answered color, 7 answered windows and 1 answered the entrance door. This further supports the conclusion that color and change in window style are the most sensitive changes, possibly along with the entrance door.

As the results show colors and window styles as being sensitive features to change it can be concluded that the features changed on other buildings are less sensitive. Thus adding and removing protruding pillars on the facade does not greatly impact the perceived realism of a building. Neither does adding and removing other facade details such as small extra windows or cube shaped boxes.

### 4.6 Changes in participant answers

The evaluation of the buildings was done in iterations which allowed participants to change their previous answers. The outcome of this is displayed in figure 4.10 and shows that participants were more likely to stay with their answer from the second iteration than the first, which implies that they got more secure in their answer after being shown all buildings. There are two plausible reasons for this.

The first, that was mentioned by participants themselves, is that the participants commented on the difficulty of knowing what to compare the first buildings they were shown to and which features that may had been altered. They were not sure if the changes were radical or just details. After seeing all the buildings in the first iteration participants seemed to understand what type of changes to look for.
the buildings. I perceive this as positive feedback of the quality of the generator since the participants did not seem to think that the initial buildings they were shown looked odd or poor enough to think that they must be one of the fake ones.

<table>
<thead>
<tr>
<th>Building</th>
<th>1st iteration</th>
<th>2nd iteration</th>
<th>3rd iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beige&lt;sub&gt;R&lt;/sub&gt;</td>
<td>-</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Beige&lt;sub&gt;C&lt;/sub&gt;</td>
<td>-</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>Beige&lt;sub&gt;HR&lt;/sub&gt;</td>
<td>-</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Beige&lt;sub&gt;HUR&lt;/sub&gt;</td>
<td>-</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Brown&lt;sub&gt;R&lt;/sub&gt;</td>
<td>-</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Brown&lt;sub&gt;C&lt;/sub&gt;</td>
<td>-</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Brown&lt;sub&gt;HR&lt;/sub&gt;</td>
<td>-</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Brown&lt;sub&gt;HUR&lt;/sub&gt;</td>
<td>-</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>Brick&lt;sub&gt;R&lt;/sub&gt;</td>
<td>-</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Brick&lt;sub&gt;C&lt;/sub&gt;</td>
<td>-</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Brick&lt;sub&gt;HR&lt;/sub&gt;</td>
<td>-</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Brick&lt;sub&gt;HUR&lt;/sub&gt;</td>
<td>-</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>-</td>
<td><strong>39 (32.5%)</strong></td>
<td><strong>17 (14.17%)</strong></td>
</tr>
</tbody>
</table>

Figure 4.10: Amount of times participants changed their answer for each building and iteration.

The second reason for changing their answers is an assumption as it was not mentioned directly by participants. I believe that after seeing all different buildings it would be easier to deduct the real buildings by comparing them to their other versions that you would now have seen.

### 4.7 Discussion

#### 4.7.1 Quality of rendered buildings

Participants were asked about their opinion of the visual quality of the rendered buildings during and after the experiment. Most participants answered that the overall quality of the buildings was high but that they had a few issues. Many participants reacted on the fact that the windows on each building all had the same texture for each building, which was not the case of the original facade pictures. Participants
also pointed out that certain textures were stretched. This happened for both real and altered buildings.

There were also complaints that structural elements on the facade were not properly aligned such as windows not being in straight lines. This was due to the manual change in rules not correctly accounting for the difference in proportions of the elements when removing or adding components on the facade. Participants said that these defects on the facades detracted from the buildings reality and made it harder to know which strange things on the buildings were intentionally altered and which were defects. Thus these errors might have impacted results slightly.

Some participants also mentioned that the color changes were too strong. If they were more subtle it would likely not be as noticeable. While this may be true it still supports the theory that color is a sensitive factor if only slight changes can be made without impacting its realism heavily.

4.7.2 Reflection on chosen buildings

Facades that the generated buildings were created from were chosen such that all three would have different styles. One participant commented that the Beige building had the most believable style and felt most like a Stockholm building. This might be part of the reason why it received a higher amount of votes as real than the other two buildings as seen in 4.6. The participant who left this comment thought that buildings in Stockholm are usually layered vertically such as the Beige building.

The Brick house has a window directly above its door which misled many participants. They reasoned that an apartment window could not possibly be so close to the entrance door. Due to this three participants selected all the Brick buildings as fake as soon as they came to this conclusion. As a result buildings of type Brick had substantially lower amount of votes as real.
Figure 4.11: Window that participants thought were too close to entrance door

This window was in the same position on the real facade as the rendered buildings meaning that a feature on the real facade itself was perceived as abnormal. I do not believe that this impacted results too heavily, however, as all versions of Brick had the same position of the window and all got an equal decrease in real votes because of it.

4.8 Summary

An experiment was performed to evaluate the implementation and test if certain features on building facades are more sensitive to change than others. Participants were told to distinguish between real buildings and altered ones where certain specific features had been changed. By performing an analysis of variance of the results it is found that certain features did indeed impact the realism of buildings more than others when changed.
Chapter 5

Conclusion

In the previous chapter we tested the implementation and were able to test the hypothesis that this paper started with. In the following part the quality and performance of the implementation is evaluated as well as difficulties encountered during the project. It then concludes with a discussion future work to be done and possible applications of the results.

5.1 Evaluation of implementation

5.1.1 Rule generator

The implemented rule extractor is capable of creating a set of split grammar rules given a properly formatted input facade image and facade layout. It can detect symmetry on the facade to describe regions through repeat rules which reduces the overall size of the grammar and makes the building able to stretch in size. Thus it filled the desired purpose. It would be desirable for it to be able to identify the depth of elements on facades, but this would require three dimensional input facades or sophisticated machine learning that can identify and handle 3D elements on a 2D image.

The implementation could be more user friendly if it was more flexible with the input facade layouts. Right now regions in the facade layout image need to be aligned perfectly by pixels for the algorithm to find a split. The algorithm cannot cut through regions or adjust them intelligently but rather gives up. It is also at times possible that the rule extractor generates split rules for an area that could be described
by a repeat rule. This happens when the repeated area is already cut on the orthogonal axis to the symmetry.

Material extraction from the facade image worked well. The largest area of an element type was always chosen to be applied for all elements to avoid stretching. This also came with the consequence that all windows shared the same texture which detracted from its reality according to participants of the survey. This could be avoided though if all windows would be defined as different element types, but that would eliminate the possibility of repeating areas.

5.1.2 Building generator

The procedural building generator that was implemented was successfully able to generate 3D buildings from a set of split grammar rules. Given that the input rules were properly formed it will always terminate with a complete facade without possibility of holes due to flexible elements being stretched. The quality of the buildings was good according to most participants, but with a few artifacts and mispositioned elements. These artifacts were mostly caused by the quality of split grammar rules. At times it was due to the rule extractor failing to properly describe a building but most often the errors were caused when editing the rules manually to create altered versions of the buildings.

Many participants also noticed texture shrinkage and stretching which happened when elements of the same type shared the same texture image. To amend this a script would be needed to properly adjust a texture to its proper size for each element it is applied to.

A limitation of the building generator is that it only worked for square shapes. Many buildings in Stockholm have protruding parts that are in the shape of cylinders or hexagonal prisms. This makes the building generator not able to properly reproduce Stockholm building that has a structural component formed as such. The buildings can also have a high amount of polygons depending on the amount of splits that are performed which might be limiting for low performing hardware.
5.2 Difficulties

Creating a procedural building generator based on inverse procedural modeling proved to be highly difficult. There does not exist a great deal of papers or examples that provide detailed guidelines for an implementation. The theory behind it was often presented briefly, leaving a lot of room open for interpretation of the math and algorithms behind it.

Gathering input photos of facades also proved to be a challenge. The first issue is to find a building with no signs, trees or cars blocking it; the latter being the most common as cars are parked all along the roads in Stockholm. The picture must also be taken during the day as to not darken the color scheme. Furthermore buildings also needed to be far enough away so that it could fit within one picture. Otherwise a panorama picture needed to be taken which was hard to properly adjust to fit the algorithm.

When a satisfactory photograph of a building is attained it will probably have perspective issues as it will most likely be taken from the ground. The algorithm expects a facade image that is completely front facing of all parts. Thus the picture will need to be properly perspective corrected which can be difficult.

5.3 Future work

5.3.1 Future experiments

For future similar experiments it would be interesting to try more subtle changes in color as the colored buildings in this experiment appeared too strong to participants. It would also be of interest to experiment with more types of changes on facades than what was tested in this experiment.

The execution of the experiment could also be done differently such as allowing participants to view some test buildings to understand the layout of the experiment better before starting. Participants said it was hard to know what to look for during the first iteration. This could possibly have negative implications though as participants might start reasoning through the logic of the experiment setup rather than the buildings initial realism.
One other possible improvement would be to have no different styles of each building. Each building would only have one version $B_R$, $B_C$, $B_{HR}$ or $B_{HR}$ displayed instead of all. This would prevent participants to compare the same versions of the buildings to more easily rule out unrealistic ones.

Though participants were free to move the camera around when shown buildings, the controls were a bit limiting. They could only move forwards, backwards, up, down and rotate left and right. They were not able to tilt the camera up or strafe. This was mentioned by participants to feel clunky. Furthermore it also removed the possibility of a street view perspective of the building as a person would see it in real life as a person walking would most likely look up towards a building rather than straight at it.

More data points, in this case different buildings, would also be helpful to more surely decide the factors most sensitive to change facades. Having only 3 different buildings in this experiment leaves some room for uncertainty of results. Having more participants of the experiment would also increase the amount of data recorded which would further cement the results.

Furthermore it would also be interesting to involve participants with background knowledge in architecture as they might have more ability to logically process the buildings and give detailed arguments for when buildings appear unrealistic.

### 5.3.2 Improvements to algorithm

The algorithm implemented could be extended and amended to generate more realistic buildings. This would make participants less distracted by unrealistic features on a building that are the results of artifacts. Namely fixing elements to not stretch texture and allowing for more shapes than square shapes would be good improvements for the generator.

The implemented rule extractor could be more intelligent and user friendly by making it always generate rules if a facade layout is within reasonable margin of error in non-alignment of elements. It could also be fixed to always detect symmetry on facades when possible.

Modeling the inside layout of generated houses would likely increase their realism as well. Rather than merely seeing window textures you would be able to look inside the rooms of the building.
5.3.3 Using the results

As the results show that certain features impact the reality of a building more when changed, heed should be taken when altering features on virtual buildings. Especially changes to building color and style of windows should be minimized or avoided to preserve its reality. Procedural generators that generate varied styles of known buildings should be aware of this and not be given much freedom to alter the windows and color of facades. This will help create more realistic buildings and consequently realistic streets and cities.

5.4 Sustainability and societal impact

The findings of this project could be used within the entertainment industry for increasing the realism of procedurally generated virtual buildings. If the results allow an increased use of procedural generation when generating realistic cities the creation time of media that utilizes them be reduced. This would be a benefit to consumers who will likely have access to more visual entertainment. It would also be an economical improvement for all companies who design virtual buildings as they would need to hire less designers for modeling. Conversely this might be detrimental for the individuals working as 3D designers as their demand could decrease.

An increase to the realism of virtual buildings could also lead to less physical buildings being built simply as movie props, which would be an environmental gain.


