# Procedural Generated Trees

## How distribution and growth factors affect perceived realism

Peter Hørdum Jæger

AALBORG UNIVERSITY
DENMARK

# Aalborg University Copenhagen

**Semester:**
10th

**Title:**
Procedural generated trees –
How distribution and growth factors affect perceived realism

Aalborg University Copenhagen
A.C. Meyers Vænge 15
2450 København SV, Denmark

Secretary: Lisbeth Nykjær
Phone: 9940 2470
lny@create.aau.dk

**Project period:**
February 1st - May 31st 2018

**Semester theme:**
Master Thesis

**Supervisors:**
Niels Christian Nilsson
George Palamas

**Project group no.:**
N/A

**Members:**
Peter Hørdum Jæger

**Abstract:**

This study explores the affect that distribution and growth factors of trees has on the perceived realism of a video game. Perceived realism was analyzed and it was found that the appearance and behavior of a game-object was based on the player's previous experience and expectations. Two independent variables were used to test what extent perceived realism is affected by distribution and by tree growth factors of procedurally generated trees? Two types of distribution were implemented: Random and ecological distribution and two tree generation algorithms, where one was based on the limiting factors that effects tree growth and one didn't. The results showed that there was no significant difference in the test conditions. But there was found an indication that the distribution and growth factors did have an affect on the perceived realism of the tree's shape. It was found that trees that are distributed with ecological distribution and grown with limiting factor have a more realistic shape.

**Copies:** 4
**Pages:** 121

# Contents

# List of Tables



# List of Figures

# 1

# Motivation

The fidelity of video game graphics has increased so much in the later years that they are almost photo realism. Games like Star Wars Battlefront (SWBF) (Ingvarsdottir, 2015) used advanced 3D scanning techniques to replicate the vast amount of props from the Star Wars cinematic universe. This resulted in games (SWBF (Ingvarsdottir, 2015) & SWBF2 (Diemer, 2017)) that are praised for having some of the most realistic graphics ever made. But it have a cost, because the vast amount of assets that have to be produced for the game, have made the game very expensive to produce (Sowers, 2008).


Figure 1.1: Star Wars Battlefront II

Some video game developers have in the later years tried to lower the cost of 3D assert production by looking into procedural generated content (PGC) (Sowers, 2008). PGC has the advantages that some of the graphical work flow are taken from the 3D artist and given to programmers and technical artists (Yang et al., 2009). A game like "No Man's Sky" (Murray et al., 2016) had as one of it's main selling points, that the

whole world was procedural generated. The player was giving endless replayability, because no two planets were the same. One problem that comes by making procedural generated games is that, there has to be a perfect algorithm to make sure that the game is playable.

Many games use PGC in the form of tools. Game developers use procedural generated textures make game asserts look different. Players' will quickly recognize a pattern that is reused over and over again (Togelius et al., 2016).

For the Indie game "Firewatch" (Moss and Vanaman, 2016) the developers intending were to use procedural generated vegetation, but because of lack of support in Unity3D for this feature at the time of development, 23 different trees were created (Kidwell, 2017). The game's lead artist states that this does not feel repetitive because *"objects and places were scaled based on what felt right as opposed to what was accurate"* (Kidwell, 2017).

Figure 1.2: Uncharted 4 : A Thief's End

The developers at Naughty Dog behind games like the "Uncharted" (Druckmann and Straley, 2016) series and "The Last of Us" (Druckmann and Straley, 2013), are often praised for their high production value and their visual style. In their latest game Uncharted 4 they had a focus on making the game environment realistic at first glance (Maximov, 2016; Shah et al., 2017). Because the environment setting was in a jungle, a lot of effort was put into making the game environment feel alive, and behave in

a realistic manner. They chose not having the procedural generated vegetation, but instead they made tools for the environmental artists (Maximov, 2016). The game looks great, but in some places it is clear that they reuse asserts so much that is just looks like a field of grass made of copies.

PGC might give the game improved replayability, because of the nearly endless combination of asserts that can be generated for a game. But when it comes to set dressing in games, what is most important for the players' perception of realism, the small changes in vegetation that comes from PGC, or the level of quality that might come from CG/ Level artists creating the vegetation themselves?

In this project will the importance of visual diversity in vegetation in a game be analyzed by looking into the level of diversity needed before humans perceive it as different. Is it better for realism in games like "Firewatch" and "Uncharted 4" has done and make a lot of unique asserts of high quality and control, or is it enough with fewer asserts? Or is it better that all asserts are procedural generated but then they might have lower graphical quality and control.

In this project it will the investigated if the procedural generated vegetation have any impact in player's/user's perception of realism of the game world. In order to solve this, affects of procedural generated content will be investigated, and what realism is and how we perceive it.

# 2

# Initial Problem Statement

*"To what extent is the realism improved in games using procedural generated vegetation"*

# 3
# Analysis

The goal of the analysis is to provide a theoretic insight of the keywords presented in the initial problem statement to narrow down the problem and then analysis previous work in order to come up with requirements for designing the test.

The analysis is structured as follows: Section 3.1 gives theoretic insight to what realism is. Section 3.2 gives an overview of vegetation techniques and how they are used in video games. Section 3.3 gives an overview of procedural generated content and how this technique is used in video games. Section 3.4 details the delimitation of the overall problem and focuses it on the growth of procedural generated trees. Section 3.5 gives an overview over how tree grows to be used in the design chapter. Section 3.6 details the foundation for rendering to produce graphical content. Section 3.7 details the how the distribution of the vegetation is used to navigate the game level. Section 3.8 investigates the state of the art in procedural generation and lastly section 3.9 is a summary of the whole analysis, which leads to the final problem statement and design requirements.

## 3.1. Realism

In games players encounter both realistic and fantastic elements, so the players suspend their disbelief in order to become one with the game world (Schwartz, 2006). Because of the players' suspension of disbelief, the players have made a subconscious contract with the game to play on it's premise. This may lead the player to accept fantasy elements as reality (Schwartz, 2006).

Realism has been found by previous research to have effect on players' immersion and engrossment, but it have also been found that not all games benefit from improved graphics or game-play mechanics (Wilcox-Netepczuk, 2013).

Because of the players willingness to suspend their disbelief, and that it can lead the player to believe in fantasy elements as being realistic, it is necessary divide them into two classifications. The players subjective perception of the how real the game is, will be classified as perceived realism (PR) and the objective measure of how realistic a game element simulating a real effect or action, will be classified as realism fidelity (RF).

### 3.1.1. Realism Fidelity

Realism fidelity has been defined as *"the level of fidelity is an objective expression of the degree of realism offered by the [game]"* by Nilsson et al. (2017) and according to McMahan (2011), fidelity can be dived into three subcategories: display, interaction and simulation.

| Classification Realism Fidelity | | |
|---|---|---|
| Display fidelity | The level of realism that the rendering software and hardware offers | Refresh rate, frame rate, display resolution, display size, etc |
| Interaction fidelity | The level of realism that action that the user, does in the game equals the one they can do in the real world | Controllers |
| Simulation fidelity | The level to which the game reproduce real world physics and characteristics | CG models, lighting, physics, VFX, etc |

Table 3.1: Realism's 3 subcategories (McMahan, 2011)

In this project the most relevant subcategory seems to be simulation fidelity. And going forward it will be divided into graphical fidelity, meaning the CG models, lighting and VFX, and behavioural fidelity that covers physics and behavior of game-objects.

#### Graphical Fidelity

The graphical fidelity in video games have increased a great deal over the last couple of years (Wilcox-Netepczuk, 2013; Gerling et al., 2013; Yang et al., 2009) as the power of graphics processing units (GPUs) has increased. With this power some video game developers are aiming for movie-quality photo-realistic games (Yang et al., 2009), while other developers are going for a more stylized look (Gerling et al., 2013). The graphical fidelity is improved with the implementation of physically-based rendering and global illumination models (Yang et al., 2009). Many of the techniques and effects, which was seen years ago in movies, have become so efficient that they are able to run in real time (Yang et al., 2009).

While video games aims for photo-realistic, some of the graphical fidelity are still just approximations instead of real physics (Jiang, 2016), because the performance of the video games are still more important than it looks, because one of the most important variables there is for a player's experience is how smooth the game runs (Claypool and Claypool, 2007). Depending on the game genre there is a difference of opinion, of how

low a frame rate is acceptable, but a frame rate below 25frames per second (fps) are typically unacceptable (Claypool and Claypool, 2007; Chalmers and Ferko, 2008).

The graphical fidelity has increased in the later years because the display fidelity has increased. With the power of the newer GPUs the resolution of games has gone from 1280 x 720 (HD) 10years ago (Yang et al., 2009) to most players having at least a 1920 x 1080 (FHD) and many having 2560 x 1440 (QHD) and up to 3840 x 2160 (4K UHD) while still having monitors with refresh rates of up to 244Hz (Valve, 2018).


Figure 3.1: Display resolutions

With the increased display fidelity, came higher resolution textures and CG model, and with the higher level of detail it can give the game assets improved realism (Wilcox-Netepczuk, 2013). Looking at the character design in the "Uncharted" series it is estimated that the polygon count has increased from around 30.000 polygons for a character in Uncharted 1 (figure 3.2 left) up to over 80.000 in Uncharted 4 (figure 3.2 right). The increase of the details has not come only because of the larger number of geometry, but also because the materials have improved shader technology, and higher resolution of the textures (Jiang, 2016).


Figure 3.2: Uncharted: level of detail increase

### Behavioural Fidelity

With the computers becoming more and more powerful over the latest year, it has become possible to increase the realism of behavior game world (Wilcox-Netepczuk, 2013). It has been found that simulating real world physics, can be used as training tools for real world situations. A study by the U.S. Army, found that "Steel Beasts" a video game that simulated the real world behavior of tanks, could be used as a low-cost gunnery training simulator (Tarr et al., 2002). Later, a professional version of Steel Beasts was developed, and this is used by armies all over the world including the danish. While Steel Beast Pro, does not have high graphical fidelity in it's game environment it has high behavioural fidelity, because the developers have focused on simulating the fire control system of different armored vehicles and the ballistics of their weapons.

Figure 3.3: Steel Beasts Pro

In the later years there has been a focus of bringing the game worlds alive, by making the game worlds more player interactive, so objects that previously were static now are dynamic. This could be that game environments are affected by the players actions, like being able to destroy objects with procedural destruction (L'Heureux, 2016), or simulating weather effect with VFX (Yang et al., 2009) or dynamic material shaders (Jiang, 2016). While serious games have showed that they can have a positive impact on training results, then these games have to be tailor-made for the situation they are trying to simulate (Lukosch et al., 2012). While Steel Beasts provides a very realistic simulation of tank warfare from the perspective of a tank gunner, it does not mean that any person has the ability to operate in a effective manner after playing the game (Jillette et al., 2009; Tarr et al., 2002).

### Realism Fidelity Summery

Looking at Realism as an objective measure of realism fidelity in games, it was necessary for this project to divide into both the fidelity of the graphics and the behaviour of the game. Some games aims for both high fidelity in graphics and behaviour, while others might aim for one or neither (see figure 3.4).

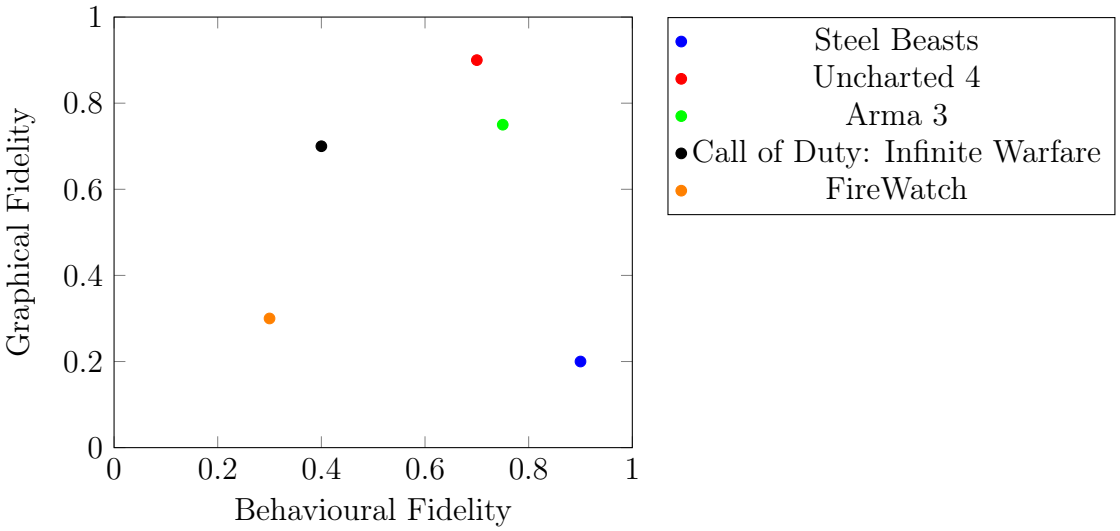


Figure 3.4: Example of a Realism Fidelity Chart

The chart in figure 3.4, is an example of how games could be measured for their realism fidelity. However it can be hard to use this model, for a whole game because different elements in games can be both realistic and fantastic (Schwartz, 2006). Because of this there might be elements that have to be unrealistic for a game-play perspective or performance concerns. A classic example is the immovable rock, that although you can interact with most of the game environment, then there is an object that is impossible to move. This is mostly made this way because the object has neither a little impact on game-play or on narrative, and this keeps performances cost down at the cost of behavioural fidelity.

Vegetation in games has the challenge with realism fidelity that the amount of vegetation that are needed in outdoor scene in games. With a large amount of vegetation comes a large amount of polygons, and if the vegetation have to be interactive, then a large amount of computation has to be calculated. So if the fidelity increases for vegetation, the performance is challenged both for the CPU and the GPU. These problems are analyzed further in section 3.2.

### 3.1.2. Perceived Realism

The definition of what is realistic for the players when it comes to video games is still debated. Some research is looking into the graphical fidelity and what impact it has on the player's perceived realism, while others are looking at how the level of interaction and behavioural fidelity impacts the player's perceived realism.

It is not possible to state that perceived realism is equal to realism fidelity. Because how players perceive something to be either realistic or unrealistic is based on previously experience (Prince, 1996). The way a plant looks (graphical fidelity) and grows (behavioural fidelity) in video game might for most people be realistic, while it for a botanist is unrealistic.

This means that perceived realism, is more about the players subjective believe than that something is real. According to Doyle and Gini (2002): *"A believable character is not necessarily a real character, but must be real in the context of its environment"*, this means that a player might believe that a game object is realistic, bus it isn't. So it comes down to the players' real life experience to judge the realism.

Figure 3.5: Tyrannosaurus Rex - (Top) Modern scientific reconstruction of Tyrannosaurus Rex including feathers by RJ Palmer, (Bottom) Tyrannosaurus Rex from Jurassic Park

An example of where it can be hard for the average player to judge the realism of subject, can be taken from the movies. No human has ever seen a living dinosaur, so they can only guess about their appearance and behavior. Paleontologists are the experts in the field and they theorize that the Tyrannosaurus Rex looks like the creature above in figure 3.5, while if you asked the general audience, then must people would answer that the creature below is the most realistic. That is because the audience have grown accustomed to the look of Steven Spielberg's (1993) Tyrannosaurus Rex from the Jurassic Park movies, so they base their believe of what is real on Spielberg's interpretation (Prince, 1996). If today's audience was shown these old school movies they would most-likely find them unrealistic because of the lack of realistic behaviour from the stop-motion models, and that lizards in costumes don't have the right appearance (Biodrowski, 2010).

Another problem with measuring players' perceived realism, is that grown accustom to improvements in the fidelity graphics and behaviour. Previously the audience believed that miniature models and stop-motion animation was how dinosaurs looked like (Prince, 1996) and later lizards in costumes (Biodrowski, 2010).

Because it can be hard to measure realism as a single value Ribbens et al. (2016) has developed a six dimensional structure by adapting realism from television. This concept could be used for measuring realism in games.

1. Simulational realism: *" The degree to which the programmed rules and the different types of behavior that are possible within these rules credibly simulate the real world, thereby making the game potentially instructive for life."* (Ribbens et al., 2016)
2. Freedom of choice: *"The degree to which the choices in a video game reflect the nature of choices one has in real life."* (Ribbens et al., 2016)
3. Social realism: *" The degree to which events and characters in a video game are considered similar to events in real life."* (Ribbens et al., 2016)
4. Perceptual pervasiveness:*" The degree to which a text creates a compelling audio-visual illusion, independent of the degree to which the content of the text may relate to real-world experience."* (Ribbens et al., 2016)
5. (Character) Involvement: *" The degree to which a player feels embodied in the video game world through the engagement with an avatar and the video game world."* (Ribbens et al., 2016)
6. Authenticity: *"The degree to which the players have belief in the game designers' intention and ability to convey an authentic, emotionally convincing, or consistent message."* (Ribbens et al., 2016)

In trying to define perceived realism, it has become clear that this is more based on the players experience and belief, than it is an objective measure of how realistic a game is. Because games build of realistic and fantastic elements (Schwartz, 2006), it would make sense for this project to focus on measuring on smaller elements of a game, instead of a game as a whole. This leads me to define perceived realism as the following:

*"Perceived realism is how real the appearance*
*and behaviour of a game-object is, based on the player's*
*previous experience and expectations"*

### 3.1.3. Realism Summery

Since the perceived realism is a subjective measure based on the player, then it means that an increase in realism fidelity, might not scale linear, because the players have different experiences and expectations. This is important for video game developers, because if they are aiming for high perceived realism, then they might not have the highest realism fidelity. This is important both for the development costs and the

performance cost. A classic example of why lower fidelity might be more cost-effective is the "Uncanny Valley" (see figure 3.6). That is why the developers at Naughty Dog have chosen to go with a more stylized look of their characters, because it looks realistic at a glance, but without the performance cost of higher fidelity and the drawback of the characters looking unnatural (Maximov, 2016).

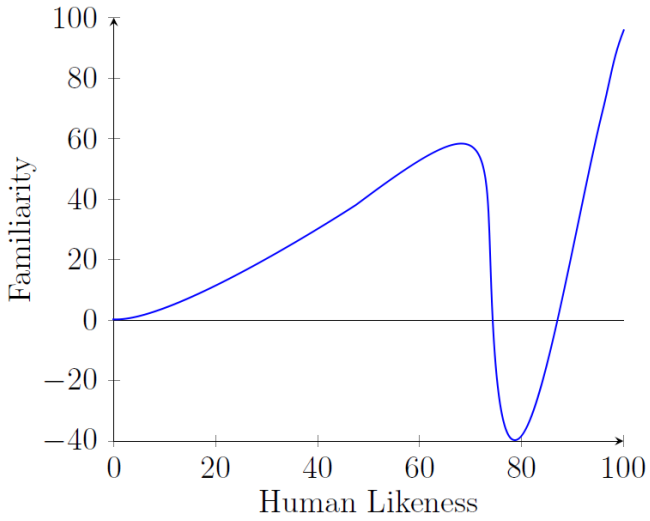


Figure 3.6: Uncanny Valley: When the human likeness of a character increases, so does the positive familiarity, until the it reaches a point where the character looks like an unnatural human.

## 3.2. Vegatation

Vegetation plays an important role in making a natural and realistic looking game environment (Jahrmann and Wimmer, 2017). Because when the game takes place outside then a large portion of the screen space is taken up by vegetation. This could be in the shape of grass, flowers, bushes, trees etc (Knowles and Fryazinov, 2015).

In the early days of 3D video games, vegetation was textures on game objects, but current vegetation rendering techniques can be divided into image-based, geometric or a hybrid of both (Jahrmann and Wimmer, 2017). Image-based vegetation is often used because it is fast to render. It is often billboards with semi-transparent textures (see figure 3.7). This could by camera-facing billboards or billboards arranged in different patterns 3.8 (Fernando, 2004).

Because the billboard is an image it becomes clear that it is 2 dimensional when viewed up close. The geometric technique creates individual geometric objects for each blade of grass, branch or leaf. While this increases the graphical fidelity of the game object also increases the polygon count exponentially. An example for a field of grass, then image-based billboard as in figure 3.7 has 10-20 blades of grass with a polygon

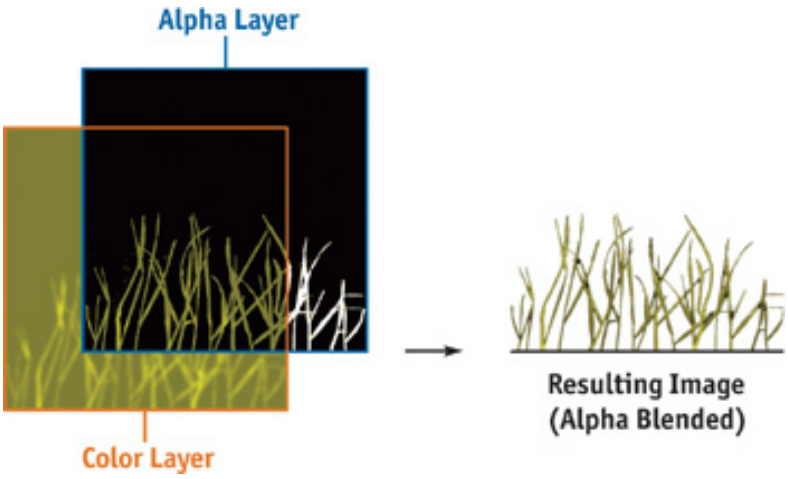


Figure 3.7: Semi-transparent using Alpha Blending

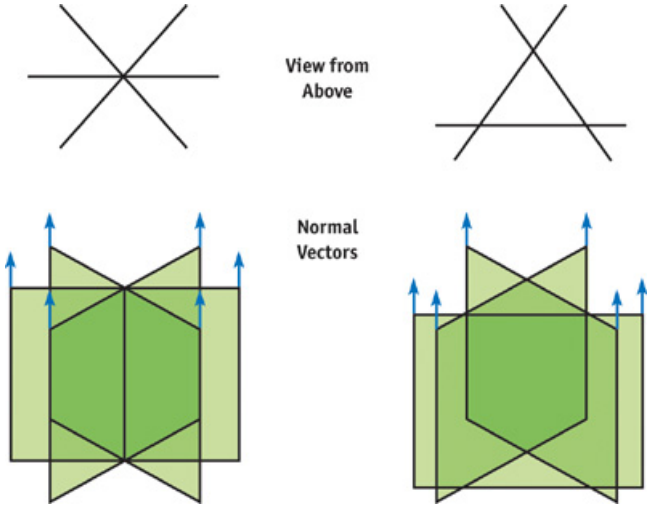


Figure 3.8: Billboards arranged in star and triangle pattern

count of two, while if it is created with geometry, then one blade can have a polygon count of one, as the fidelity increases up to hundreds (see figure 3.9). This also puts more work to the CG artist that have to make different geometry, while the there is some variation in the image-based billboard, because it may have different object on each billboard. Because of the large amount of geometry that can be used to make a field of grass, GPU instancing is used (Fan et al., 2015). GPU instancing allows for rendering multiple copies of the same mesh using less draw calls (Unity3D, 2018a). GPU instancing lets the game maintain performance, but to achieve this the geometry can only have differences in scale, rotation, transform and color, but the overall form of the mesh has to be the same.

More complex vegetation as bushes and trees, often use a hybrid of both image-based billboards and geometric. Where the larger parts of the tree, like the trunk and branches are made of geometry and the smaller branches and leaves are billboards (Jahrmann and Wimmer, 2017).

With the amount of objects that have to be on screen for an outdoor screen comes

Figure 3.9: Tree. (Left) Image-Based using 278 billboards, (Right) Geometric using 113.176 polygons.

the problem with draw distance in games. If the game uses geometric vegetation then there can be billions of polygons on screen at a time. To improve performance most games use a Level of Detail (LOD) system, where the amount of detail and geometry are greatly reduced as the distance to the camera increase (see figure 3.10) (Unity3D, 2018b). This means that game developers can use a combination geometric vegetation up close, near the camera and as the the distance increases they can use a hybrid, and on long distances they can use image-based billboard.

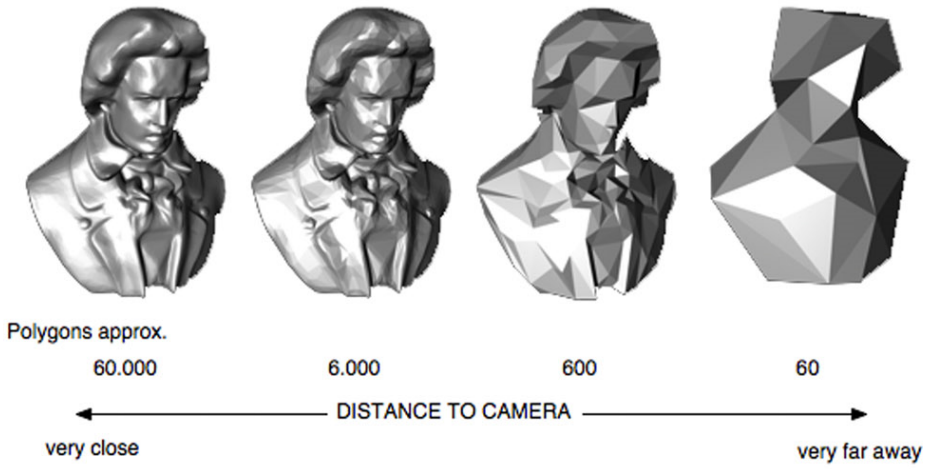


Figure 3.10: Level of Detail (LOD)

Billboards used on complex geometry (also called an imposter (Risser, 2006)) like a tree creates a problem, because most trees are not symmetric, so the angle from the camera to the tree, should change the appearance of the tree. One way developers have overcome this problem was by using a tool in Maya that automatically renders out images from different angles (see figure 3.11). While the game only renders out a 2 dimensional imposter, then it looks like the tree is in full 3D, because when the player change the angle to the tree, then the appearance of the tree is changed (Risser, 2006; Maximov, 2016).

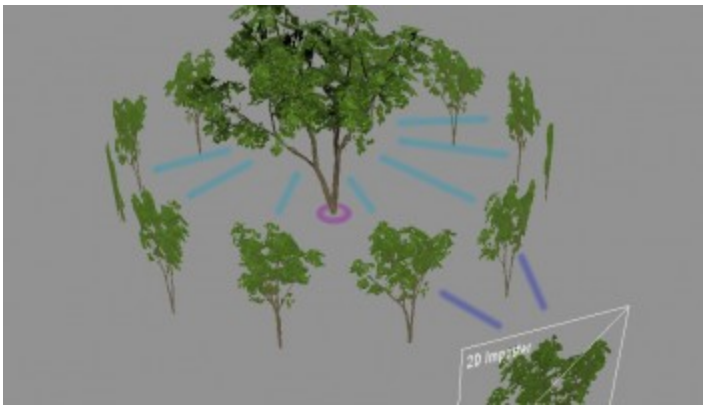

Figure 3.11: Imposters

As the graphical fidelity of vegetation has increased in the later years, research has started on focusing on the behaviour of the vegetation (Knowles and Fryazinov, 2015; Jahrmann and Wimmer, 2017; Lee et al., 2016; Pirk et al., 2012). The behaviour of the vegetation can have an impact on the player's perception of the realism of the game. One thing is that the player might expect the vegetation to move out of the way when colliding with it (Knowles and Fryazinov, 2015). Players would also expect that light vegetation would be effected by external forces like wind, gravity (Jahrmann and Wimmer, 2017). The appearance and behaviour of the vegetation also have to abide by the laws of physics, that means that gravity should always effect the vegetation and that a branch can't intersect other objects in the game environment (Pirk et al., 2012).

Vegetation swaying in the wind is an important effect in outdoor scenes, because it makes the usual static screen look more alive (Lee et al., 2016). Image-based billboards have use vertex displacement inside the shader to approximate wind, this however have the drawback that it distorts the texture used on the billboard (see figure 3.12). It is a fairly cheap technique to use, but the complexity of the animation are limited.

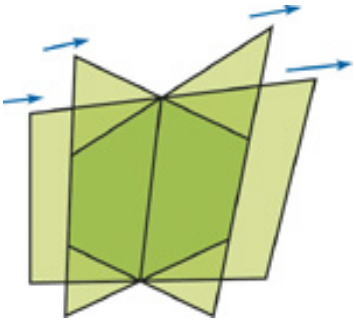

Figure 3.12: Vertex displacement

For simple vegetation like grass this distortion might not be visible for the player. In larger objects like imposters of trees this distortion leads to rendering artifacts that might lower the perceived realism (Jahrmann and Wimmer, 2017). This means that although billboards and imposters are great for performance and LOD systems, they have challenges when it comes to behavioural realism. Seen a tree in the horizon that

are distorted equally, would look unnatural, because the player would expect the leaf to move more than the trunk (see figure 3.13).

Figure 3.13: Imposter distortion

Geometry vegetation has advantage over billboards where each individual branch, leaf, blade of grass can by evaluated for gravity, wind and collisions (Jahrmann and Wimmer, 2017). If each leaf and blade of grass are simulated at all time, the games performance will be slow. That is why global effects like wind and local effects like collisions are handled separately. Wind is still mostly handled by the vertex shader, and is often a function of time with some noise added to prevent visible directional patterns (Fan et al., 2015). Because it is handled by the shader on the GPU, there isn't any calls between the CPU and GPU. But for collisions with objects in the game information has to be exchanged between the CPU that knows the objects positions, and the GPU that have to make the change in the geometry. It is not possible for the all leaves and blades of grass to be checked for collisions with objects in the game environment, so different techniques have been tried. Fan et al. (2015) uses a technique where they divided the vegetation into tiles or zones and a simulation is only conducted when an object is entering the zone, and two seconds after when the object leaves the zone the simulation ends with the grass blades returning to their original state (Fan et al., 2015). The collision is handled by the object checking for intersection with each vertex in the grass blade. The blade are moved out of the way with constraints on length, bending and twisting to avoid stretching.

Pirk et al. (2012) has researched the growth behaviour of trees, so trees that are placed in a game environment, would take light distribution and proximity to solid obstacles and other trees into considerations for the appearance of the tree. They simulate the competition for resources that trees conduct, especial the competition for light, and how they have an important influence on the trees shape. They also simulate the bending and shedding that occurs when a tree is growing close to an obstacle.

Figure 3.14: Adaptive growth behaviour

| Classification of Vegetation | | | |
|---|---|---|---|
| | Render Time | Graphical Fidelity | Behavioural Fidelity |
| Billbaords | Fast | Low | Low |
| Geometric | Slow if many | High | High |
| Hybrid | Slow if many | Medium | Medium |

Table 3.2: Classification of Vegetation

### 3.2.1. Vegetation in Games

There have been given overview of the techniques in academia here a small overview will be analyzed of how game developers are making vegetation where there is room for improvement. Two games where chosen, and they are both third-person adventure games with a high score on Metacritic. The first game is Uncharted 4: A Thief's End (93 on Metacritic), which is a linear story driven game, with a large focus on narrative and graphics. The second game is The Legend of Zelda: Breath of the Wild (97 on Metacritic), which is a open world adventure, which focus on player freedom, and it has a stylized graphical look.

#### Uncharted 4: A Thief's End

Uncharted 4 is like the blockbuster movies. They have a high production value and are highly story driven. The developers have an extreme focus on details, which can be seen in the game environments. Because the game takes place in exotic locations like the jungles of Panama and Madagascar, vegetation was an important part of the game as seen in figure 3.15, where the abandon town has been taken over by the vegetation.

Figure 3.15: Uncharted 4: Overgrown town

The developers have taken advantages of the fact that it is a linear game, because they have full control of the game environment. This means that they know all the angles that the player can see in a given game object, and then they optimize the game to

that. One technique they use are imposters to have long draw distances. This can be seen in figure 3.16 (Top), where the trees in the distances are imposters. They did not only use imposters for the vegetation but also for large game-objects like building (Maximov, 2016). They also use billboards for the low level vegetation like grass as seen in figure 3.16 (Bottom), but because they use a technique with billboards with more than 4 vertices, it allows the billboards to bend more at the top, given the illusion of more geometry.

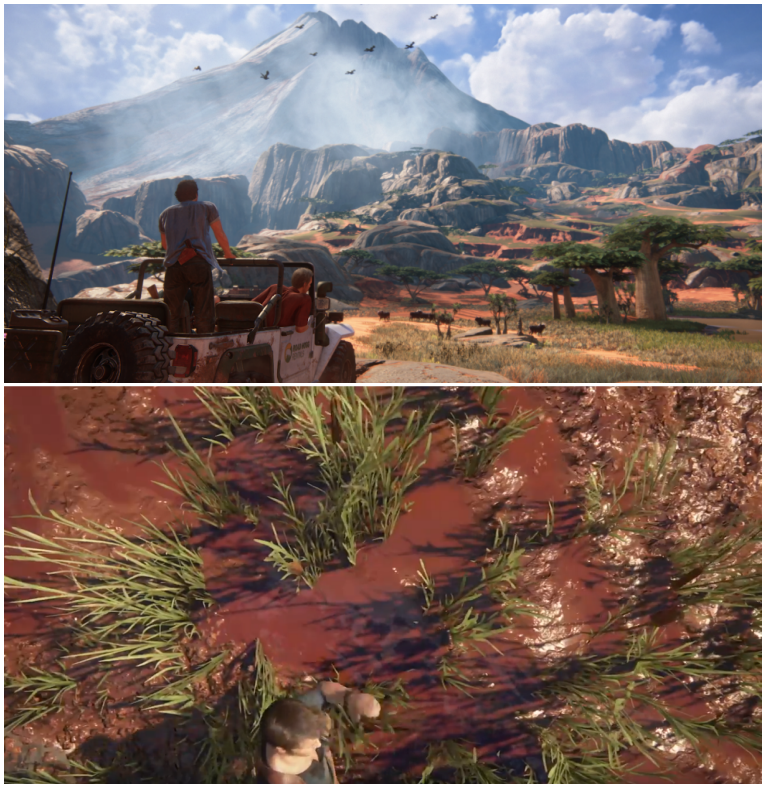

Figure 3.16: Uncharted 4: Imposter vegetation

As the game progresses so does the weather, and the vegetation is impacted by wind and rain. The weather effects are mostly shader and animation based. This also makes most sense from a quality perspective, because as the game is linear then the developers always know how the weather is going to be. They don't need a system that can take any given tree and make it look good in a hurricane and on a calm day, because they can make the animation that looks the best for each situation.

The player can interact with the vegetation by hiding the main character in high grass or other ground vegetation. When the main character moves in the grass, the blades of the grass are moved out of the way, as seen in figure 3.17. The same effect is used if an explosion occurs in the grass, just with a fast displacement of the grass and in a larger radius. The interact-able grass, does not have much diversity, and looks like it is the same game object that is used over and over again. This might lower the realism of the game, because the player can see a pattern in the grass, but it is most-likely done from a game-play perceptive, so the player always know if he/she can use the grass to hide in (Foreman and Floyd, 2015).


Figure 3.17: Uncharted 4: Grass Interaction

Uncharted 4 is an example of what you can achieve if you have both time and money to make a great looking game. From the author's opinion the graphical realism in the vegetation is high and the behavioural medium to high. The developers have many advantages they can use because of the game genre and style, that allows them to optimize the game's performance. But with these advantages they have chosen to use the overhead to make more assets for the game, instead of just reusing the same assets over and over again. The only place where assets are reused, seems to be for clarity of game-play mechanics.

### The Legend of Zelda: Breath of the Wild

Zelda is an open world sandbox game that, after the first hour of game-play lets the player play the game in their own way. There is a narrative, but it can be circumvented entirely. The player is in the fictitious kingdom of Hyrule, for the Nintendo Hybrid console the "Switch". Because Zelda has to work on a hand-held it has it's limits to performance. They have chosen a more cartoony Cel-shaded look, instead of the going for realistic graphics. There is a lot of diversity in the game environment, with large deserts, forests, mountains and fields. But while the environments are diverse the game objects often are reused as seen i figure 3.18.

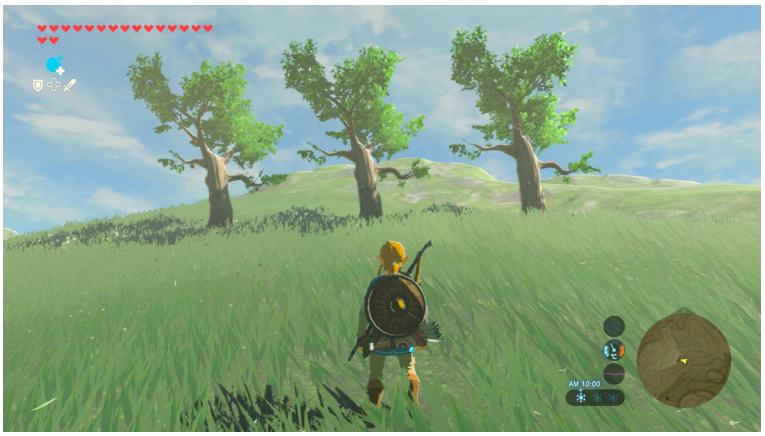

Figure 3.18: Zelda: Reuse of assets

While the game assets are reused, and the graphical realism might be low compared to Uncharted 4, then the level of interact-able is very high. Like in Uncharted 4 the player can hide in high grass and the grass will move away from the player, as the main character moves through it. But Zelda takes it further, the grass and other vegetation are effected by wind that can come from any direction. The player can interact with the vegetation in a number of ways, like cutting down, set fire to or blow up grass, trees etc(see figure 3.19). This gives the vegetation a higher level of behavioural realism, because the game simulates the vegetation in different states, as the player manipulates it.

Figure 3.19: Zelda: Setting fire to grass

Because Zelda is an open world game that doesn't have the advantages of a linear game and is hand-held, it is impressive how good looking a game it is, and even more impressive is the level of behavioural realism of the vegetation. It comes at the cost of reusing a lot of game assets and simple geometry, but it fits with the artistic style of the game.

**Summary**

The games analyzed here focuses on different aspect for making the best game. They both have a large amount of vegetation in their games, and they have focused on the realism of the vegetation in different ways. One game focused on the most realistic graphics and the other on having realistic behaviour. It is clear as a game world becomes larger so does the work that has to be put into making game assets. A game like Uncharted 4 that has a high level of graphical quality, might only be possible because it is a linear type of game. Zelda on the other hand has fewer game assets, but they are also made with a high level of quality, especially with their focus on behaviour. If the work of these two games should be combined, then it would be a vast undertaking, because not only the artist would have to make each separate model, but also animate for wind, interaction, fire etc.

## 3.3. Procedural Generated Content

Video games have become bigger and more complex in the later years, and the players expect more from the game developers each year. This has made the games more expensive to produce. An area of game development that has had increasing cost is the modeling game assets for virtual worlds (Smelik et al., 2011). That is why game developers are looking into automating the process of making some content for games. They are looking into if some of the work of CG artist that could be taken over by programmers, by using algorithms. Then a nearly endless amount of randomized objects can be generated. This is called procedural generation. But as it is expensive to manually create a vast amount of assets, it is also why expensive to develop a procedural content generator, so it has to be taken into consideration the size of the game. If the game is a 3-4 hour indie game like Firewatch, then made sense for the developer to create 23 unique trees for that game (Kidwell, 2017), instead of investing a lot of time and money in a procedural tree generator (Portnow and Floyd, 2015). A game that have a larger size or aim for many playthroughs, the developing of a procedural generator might be cost-effective.

Their are many strengths and weaknesses for procedural generation. The greatest strength of PGC is the almost unlimited amount of content that can be produced with PGC, if the game developer needs to make a huge virtual world or many various levels. Examples are games like Sid Meier's Civilization, that uses procedural generation to generate the world. The world is made up of 6-sided tiles put together to make up a map. Because the map is generated with random tiles, then no maps need to be the same. The player can because of rules by the programmers, have some general control over the randomness, like the map is one big landmass or tiny islands, and like the resources that are available (see figure 3.20). Because no maps are the same, then the player has to adapt a new situation for each new game (Portnow and Floyd, 2015). Noghani et al. (2010) found that procedural generated environments, like generated terrain, vegetation and building structures revealed that players found the appearance

realistic and believable. Korn et al. (2017) also found that players perceived the game as more realistic and aesthetically pleasing when using procedural generated graphical props.

Figure 3.20: Sid Meier's Civilization V: Procedural map generation

Because the map is generated by an algorithm instead of a designer, is it much faster for the developer to make a huge amount of levels, but they have to make an investment in procedural generation. This investment is quickly earned back compared to have level designers making each level. The more maps there have to be made the better return there is on the investment (Portnow and Floyd, 2015).

PGC can also be used to adapt the game world and challenges in the game as a response to the player's playstyle (Shi, 2016; Portnow and Floyd, 2015). This could be a feature like procedural destructive environments, where if the player shot at a wall, then the game object reacts in a physically accurate manner. Because the wall is regenerated based on the shot instead of a pre-made game assets, it will look more realistic for the player. It's expensive to develop a procedural destruction system for a game, but compared to have an artist to make a nearly endless amount of game objects to substitute with the original on impact, then the complexity becomes even greater, when multiple shots are fired at the wall. Here lies the power of procedural generation (L'Heureux, 2016; Müller et al., 2013).

One of the biggest weaknesses of PGC are the loss of control over the final result (Smelik et al., 2011; Portnow and Floyd, 2015). Because current modeling and animation systems provides the designers with absolute control over the final result for their game, and for single object this is a fast process by an experienced CG artist/animator (Smelik et al., 2011). It can be hard for designer or artist to handover the artistic style of a game to algorithm, because they know with their skill and dedication, they can create any virtually object exactly the way they want with every detail needed (Smelik et al., 2011).

An example where the difference in quality in behavioural realism can be seen is in facial animation between the two games Uncharted 4 and Mass Effect Andromeda (MEA). As stated before Uncharted 4 has a very high production value, and all the

facial are produced with a combination of motion capture and manual animation. This gives the characters a very realistic behavior and look (see figure 3.21 Top). But because MEA is a action role-playing game with hundreds of characters, then manually animating up to hundreds of hours of facial expression would be an impossible and too expensive task. That is why the developers created a procedural conversation systems, that animated all the conversations in the game based on phonetics (Floyd, 2017). The conversation system does not have the same graphical and behavioural fidelity as the motion capture, but it allows the developers to animate a much larger amount of conversations. MEA did however come under criticism for their facial animations, because the characters had a uncanny appearance and behaviour. It seems that errors in the randomness of micro facial expressions have deformed the character models as seen i figure 3.21 (bottom).

Figure 3.21: Motion capture animations vs conversation systems

Procedural generation has great potential to maximize the production value of a game, because algorithms can create the game for the designer. But the only way to deliver high quality is to learn the procedural algorithm all the skills of an CG artist or level designer. This means adding in constraints and variables that insures the algorithm obeys the theories of good CG/game design. This makes procedural generation extremely complex, and if it should fit to all situations, then the problem is *"the fact is that one size never actually fits all, procedural content is rarely as powerful as carefully handcrafted content"* (Portnow and Floyd, 2015). When if a programmer can develop the perfect procedural generator then there is the problem with video games, as an art form, because the evolving of good art sometimes breaks the rules. A table of strengths and weaknesses are summarized in table 3.3.

| Classification of Procedural Generated Content | | |
|---|---|---|
| | Strengths | Weaknesses |
| Procedural Generated Content | Endless content<br>Cheaper when with increased content<br>Adaptation to players response<br>Create new content fast<br>Unlimited content | Expensive to develop<br>Lack of control<br>Lower quality<br>Have to make many rules for it to work<br>Is random<br>Hard to use with a strong narrative<br>Hard to do well |
| Manual Created Content | Cheap for small size games<br>Great control over quality | Expensive for huge games<br>Slow to produce |

Table 3.3: Classification of Procedural Generated Content

### 3.3.1. Procedural Generated Content in Games

PGC is nothing new in the gaming industry. PGC comes back to games like Rogue, later Diablo and is still popular with whole games build up around procedural generation like Spore and No Man's Sky (Shi, 2016). PGC is often used in games to make it more enjoyable on multiple playthroughs, because some features in the game have changed their appearance or behavior.

#### Spelunky

Spelunky (90 on Metacritic) is an indie explorer platformer the aim of the game is to explore caves and underground tunnels, and collect as much treasure as possible, while avoiding getting killed by traps or enemies (see figure 3.22). Because the game-play revolves around exploring caves, then the procedural generated levels make the game more enjoyable on subsequent playthroughs, because the player can't learn the levels by heart (Brown, 2016; Portnow and Floyd, 2015).


Figure 3.22: Spelunky

The player has to balance the risks and rewards in the game, because there is a limited time to finish a level, or else a unbeatable ghost will appear and kill the player. The unpredictable level, that comes with procedural generated levels makes sure that player just can't get all the treasure and all the optional pick ups, and can't get the perfect score because they don't know the level (Brown, 2016; Portnow and Floyd, 2015).

According to the game designer of Spelunky, the procedural generated levels are a mix of randomness and carefully created elements (Yu, 2016). The levels are carefully designed to always follow the same rules so there always is a possible path out of the caves. All levels are made up of a 4 X 4 grid of rooms, and the entrance is in one of the top four rooms. A random path is chosen into either the room on the left, right or below (Yu, 2016). If the path choses a room that is outside the grid, then the path is altered to the room below, until the path is on the lowest grid. When it goes outside from the lowest the grid, then the exit is placed in the last room (see figure 3.23 left) (Yu, 2016). Each room are chosen from a list of handcrafted rooms that are categorized as either landings, corridors, non-criticals and drops (see figure 3.23 right), to insure that there always is a possible path from the entrance to the exit, without the use of helping ads. The rooms in the grid that are not filled out gets random rooms, because they aren't needed to complete the level (Brown, 2016).

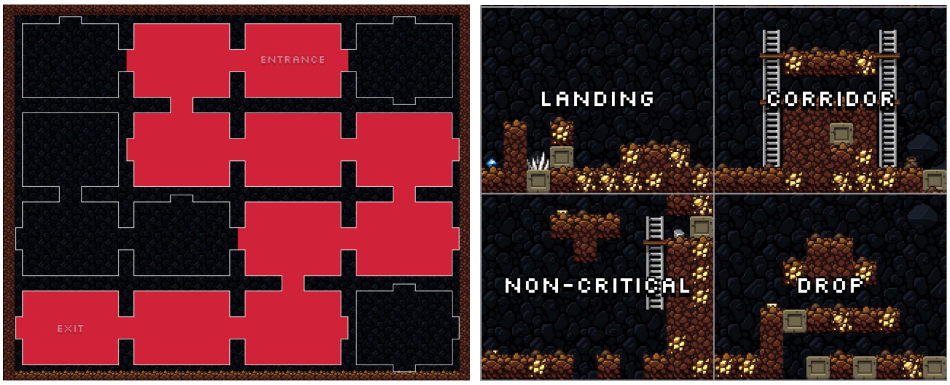


Figure 3.23: Spelunky

*"This system doesn't create the most natural-looking caves ever, and the players will quickly begin to recognize repeating landmarks and perhaps even that the levels are generated on a grid.... [But] it creates fun and engaging levels that the player can't easily get stuck in, something much more valuable than realism when it comes to making an immersive experience."* (Yu, 2016)

## 3.4. Delimitations

Delivering vast and realistic game environments have become a huge undertaking for game developers. Players expect high graphical fidelity in interesting diverse worlds. This means that CG artists have to produce a nearly impossible amount of assets to live up to the players expectations and the same time the developers have to find way to make game production more efficient. Some game developers have given up on the graphical fidelity improvements race and have chosen to make budget video games called indie games. AAA game developers either try to complement their initial income from the game work or save money by reusing assets from game to game.

Procedural content generation have the potential with upfront investment to create an almost unlimited amount of game assets, with just some small adjustments to different variables. The problem with PGC are the loss of control over the end result, that might lower the graphical and behavioral fidelity and that in turn lower the perceived realism of the assets.

This project will focus on how to achieve graphical and behavioral fidelity when generating vegetation for games. Vegetation poses a problem for games, because in outdoor scenes there is a large amount of vegetation like trees, bushes, flowers and grass. But while human created objects can be explained to be near perfect copies of each other, there are biological, ecological and physical rules that have an impact on the appearance and behavior on plants.

Since producing a whole ecology simulation with multiple plants are beyond the scope of this project, it will be limited to produce a proof of concept of using PGC to create procedural generated trees, that have appearance and behavior modelled based on the condition that it is living in. While the trees should preserve high graphical and behavioral fidelity as handcrafted models.

## 3.5. Tree Growth

In order to replica the growth of a tree, is it necessary to have some fundamental knowledge of basic plant biology. There are two types of plants: Non-vascular and vascular. Vascular plants have specialized tissue that distribute resources through out the plant, while non-vascular doesn't and that restricts the size of non-vascular plants (Berlyn et al., 2018).

### 3.5.1. Anatomy of a Tree

Trees are a vascular plant with three parts: roots, trunk and leaves, that all have different functions for the tree's growth and survival (Berlyn et al., 2018).

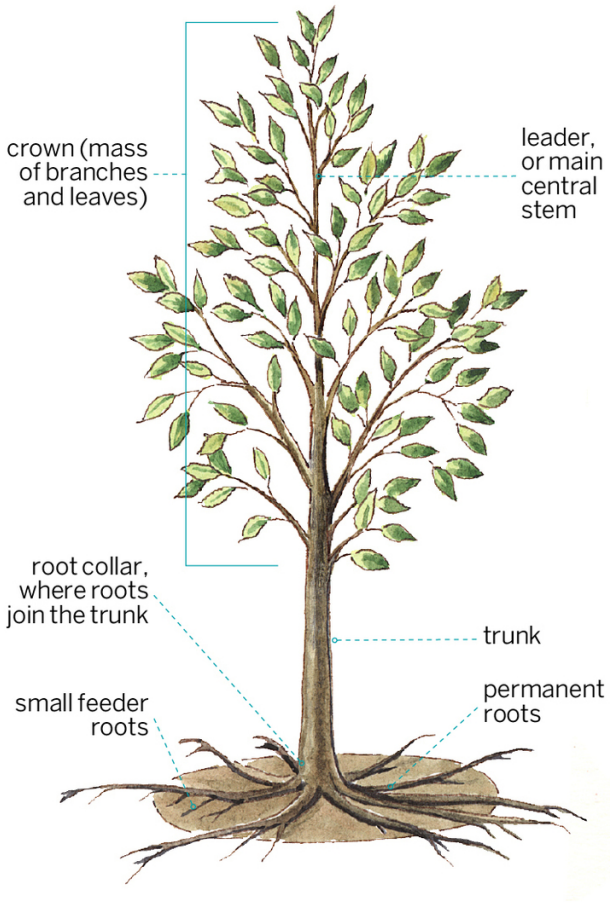


Figure 3.24: Anatomy of a Tree

### Roots

The roots anchors the tree to the soil which helps the tree stand upright, the roots can be up to four times the size of the tree crown. The roots also absorb water and minerals from the soil, that they are transporting to leaves for photosynthesis. The roots also stores the energy that are produced by photosynthesis, as a reserve for winter or adverse times (Berlyn et al., 2018).

### Trunk, branches and twigs

The trunks main function is to raise the leaves above the ground to out-compete other plants for light. It also functions as the trees transportation network, by transporting water and nutrients from the roots to the leaves and energy from the leaves to the rest of the tree. The trunk grow in height as the apical meristem's cell divide and elongate at the base of it's buds. The trunk growth is coordinated to increase in width as the tree grows higher. Some trees have a layer of cells, that are wrapped around the meristem. This allows the tree to grow outwards from the buds. The branches create new buds for each branch as a recursive function. While most branches grow diagonal

upwards, it is possible to grow horizontal, vertical, or diagonal. The growth direction is different from species to species, and is optimized for light gathering (Berlyn et al., 2018).

#### Leaves

Leaves conducts photosynthesis, which convert water from the roots and $CO_2$ from the air into energy and $O_2$. Because photosynthesis needs sun light to occur, the leaves are arranged to avoid self-shading to maximize their exposure to light, as they are an important and expensive resource to develop for the tree. That is also why some trees have evolved protection measures like thorns to discourage animals (Berlyn et al., 2018).

### 3.5.2. Limiting Factors

There are factors that limits the growth of trees, this includes environmental factors like water availability, soil quality, and climatic variation, but also the species ability to elongate it's cells each growing season, thereby growing in height (Berlyn et al., 2018).

The most important factors for growth are sun light and water, so that the leaves can produce photosynthesis. The tree also needs good soil, both to get the nutrients and minerals to grow, but also to have the roots spread out to anchor the tree firmly to the ground so it doesn't fall over in strong winds. The tree growth are limited to the length of the growing season, because as the temperature falls the tree has to save resources to survive the winter (Berlyn et al., 2018).

Trees are in constant competition with other vegetation for the limited resources in the ecosystem. The structure of the tree is optimized for photosynthesis in the environment it is in. Because the earth has many different climate zones, and as trees have adapted, many diverse tree species have been evolved. But all trees have the same limiting factors just with different tolerances to each of these factors (Berlyn et al., 2018).

Since trees are unable to move in search of resources, then tree have to adapt to compete. This makes some trees to grow very tall before producing a big crown in order to rise above other plants, while other tree specialize in living in environments where others can't. Trees species that can survive in unfavourable condition would actually grow better in better condition, but because of the competition they are eliminated (Berlyn et al., 2018).

| Limiting Factors of Tree Growth | |
|---|---|
| Improves Growth | Limits Growth |
| Sun light<br>Water<br>Soil<br>Heat<br>Long growing season | Shade<br>Wind<br>Snow/Sand<br>Long Winter |

Table 3.4: Limiting Factors of Tree Growth

## 3.6. Rendering

High fidelity computer graphics are highly depending on the techniques used to render out the final result on the screen. This section will give a brief overview of how a screen is rendered in the render pipeline, how the different light calculations effect performance and how advances in physically based rendering allows for more realistic rendering of materials.

### 3.6.1. Render Pipeline

With the programmable graphics pipeline is possible to change how each pixel is drawn on the screen. It's possible to warp vertices once they are on the graphics card and change the pixels look by adding a bumpy appearance with normal maps and apply reflection with code (Owens, 2013).

This code is in the form of vertex, geometry and fragment shaders (see figure 3.25), and with them it is possible to change how objects are rendered on the graphic card (Owens, 2013).


Figure 3.25: Simplified view of a programmable graphics pipeline

The function of the vertex shader is to transform the objects in the game from object space into camera space, in order to render out the object correctly on screen. Normal data and texture coordinates are also transformed and send to the next step in the pipeline. Vertex shaders can also be used to wrap the relative position between the vertices to make an animation (like in figure 3.12). The vertex shader can also handle per vertex lightning however newer techniques are used today (Owens, 2013).

The geometry shader are an optional shader, because if the object doesn't need any changes then data from the vertex shader can be send directly into the fragment shader.

The geometry shader functions are a way to change how the data from the vertex shader can be rendered out. Most objects have their vertices arranged in triangles, but with a geometry shader it is possible to ignore the original arrangement and make whole new ones as long as the new shape is made up of points, lines or trianlges. This means that geometry shader can add or remove geometry regardless of the original shape (Owens, 2013).

The function of the fragment shader or the pixel shader is to render out the color of the pixels on the screen. The fragment shader calculates each pixel, and with the normal maps, textures, lighting direction etc the color for each pixel is determine (Owens, 2013).

### Forward Rendering

Forward rendering has been the standard rendering technique used in the later years for most game engines. The game engine supplies the graphic card with the objects' geometry, that are send as vertices into the vertex shader, and then transformed and send to the fragment shader, where the lightning is calculated once per light. This is a fairly linear approach as each object is passed down the pipeline one at the time to produce the final image on the screen (see figure 3.26) (Owens, 2013).

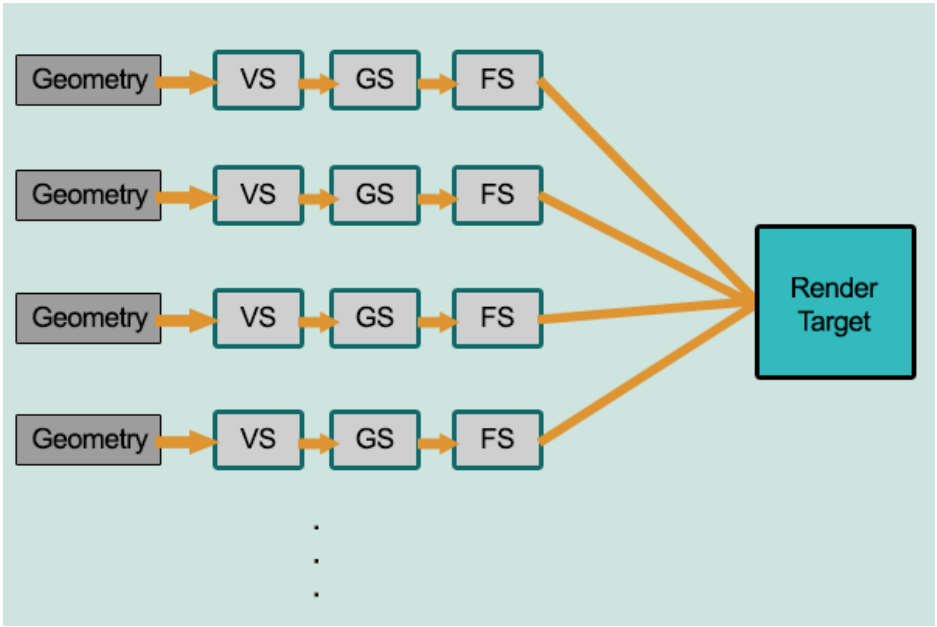


Figure 3.26: Forward rendering: Vertex shader to geometry shader to fragment Shader

### Deferred Rendering

Deferred rendering, defers the rendering a little bit until all the objects' geometry have passed down the pipeline. When all the geometry have come down the pipeline the lighting is calculated for all the objects and applied to the shading 3.27) (Owens, 2013).

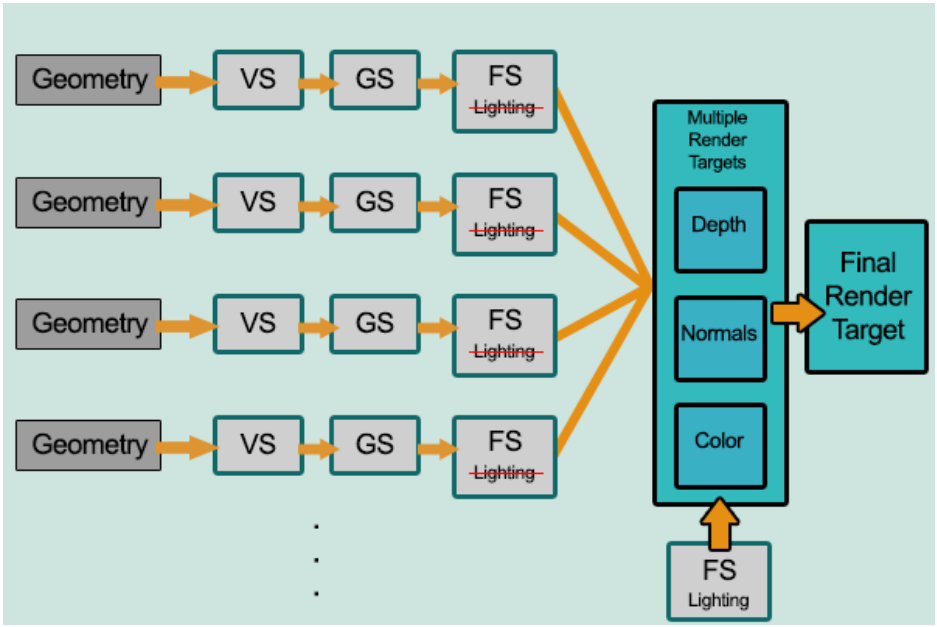


Figure 3.27: Deferred rendering: Vertex to geometry to fragment shaders. Passed to multiple render targets, then shaded with lighting.

### Lighting Performance

Lightning is the main reason to choose deferred rendering instead of forward rendering. With the standard forward rendering pipeline, lighting have to be done for every vertex and every fragment in the view frustum for every light source in the screen (Owens, 2013).

If the scene contains 100 objects that each have 1,000 vertices, then the game engine has to render out around 100,000 polygons (rough estimate). Each polygon is send to the fragment shader that has to calculate the light. With a standard HD monitor resolution of 1920x1080 the game has to render out over 2 million pixels per frame, meaning that millions of fragment operations are conducted, some on fragments that will never make it to the screen because they may be blocked by other objects in the screen, and are removed after depth testing (Owens, 2013).

The problem for forward rendering becomes greater as the number of light sources increases in the screen. An example could be when 4 million fragments operations are conducted in a screen with one light source then the number of fragment operations will increase when more lights are added to the screen:

$$\textit{Number of Light} \text{ x } \textit{Fragments Operations} \tag{3.1}$$

Deferred rendering reduces fragment operations, by only calculating the pixels that are on the screen. This means that lighting is only calculated for the screen resolution, and is independent of the number of light sources and objects in the scene (Owens, 2013).

Deferred rendering works by rendering out every objects' geometry without lightning

to several screen space buffers. Depth, normals and color are written to their separated buffer. The data from these are then combined and each pixel on screen uses this data to light it. With the information of the pixel distance to the light and its angle to its normal the color can be calculated to produce the final render (Owens, 2013).

In a game with vegetation deferred rendering has the advantage that it can handle rendering realistic light on thousands of objects like blades of grass and leafs, without increasing the fragments operations for every new light.

### 3.6.2. Color Space

Color space determines the maths that are used when colors are mixed in the lighting calculations or when they are read from the textures. The choose of color space is often decided by the hardware that has to run the final product and it's limitations. In Unity3D there is an option to chose between linear and gamma color space.

#### Linear Color Space

Linear space has the advantages that colors which are supplied to the shaders will be brigten linearly as the light intensity increases. This means that colors can be added and multiplied correctly. This helps to preserve a consistent result as the colors are changed down the rendering pipeline. This increases accuracy of the color and improves the realism of the screen.

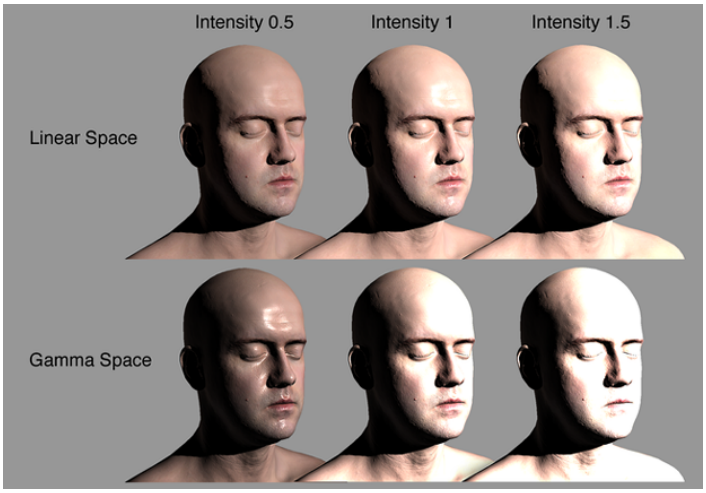


Figure 3.28: Image comparing objects lit using Linear and Gamma Color Space. Notice how colors quickly turn to white as light intensities increase using the Gamma Color Space.

#### Gamma Color Space

Gamma space is used because the human eye are better to see the difference between darker shades and lighter shades. Because of this images are compressed to save space, by having greater accuracy in the darker intensities at the cost of the lighter intensities. This means that at lighter intensities the brightness will faster begin to turn in to white, which can have negative effect on realism of the screen.

With support for High Dynamic Range coming to more and more monitors the linear color space is preferable in the future, because it allows for more precision as each color channel has more values between their two extremes.

### 3.6.3. Physically Based Rendering

Physically based rendering (PBR) is a model in computer graphics that aims to simulate light rendering based on models from the real world. PBR pipeline aims accurately simulated photo-realism in real time graphics. PBR is often a approximation of the bidirectional reflectance distribution function (BRDF), that is used to calculate the reflection of light from a surface, based on the surfaces material parameters. Global illumination (GI) is also a important part of a PBR pipeline, as well as energy conservation, Fresnel conditions and micro-surface scattering.

#### Reflectivity

Reflectivity is how much light a surface reflects. When light hits a surface then some of the light is reflected in a direction on the opposing side of the surface normal. A smooth surface with a high reflectivity will reflect almost all the light as shown by the green rays in figure 3.29. However not all light is reflected from a surface, some of the light will penetrate into the object and be absorbed as heat or scatter inside of the object as shown by the red rays in figure 3.29.

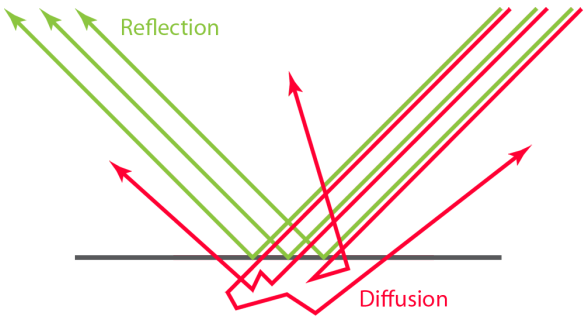


Figure 3.29: Reflectivity

### Energy Conservation

Energy conservation is the concept that an object cannot reflect more light than it receives. This means that the reflections of diffuse and rough surfaces' are dimmer and have a wide highlight radius, while smoother and more reflective surfaces' are brigter and have a tighter highlight radius as seen in figure 3.30.

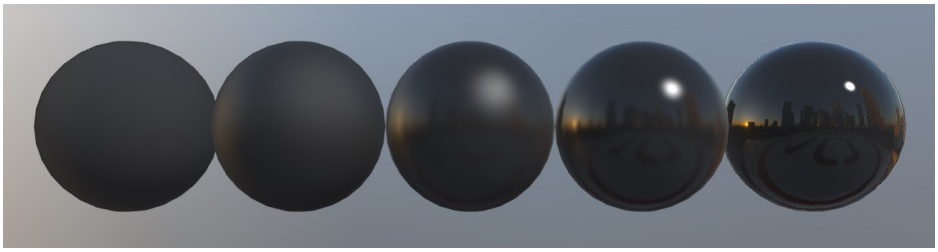

Figure 3.30: Energy Conservation

### Fresnel

Fresnel is the concept that surfaces reflect light when seen at a grazing angle. Fresnel conditions states that even poor reflectors can become almost mirror-like when light hits them at a grazing angle. Bump or variances in the microsurface might result in brighter or dimmer Fresnel effects as seen in figure 3.31.

Figure 3.31: Fresnel Effect

### Micro-surface

Micro-surfaces are the roughness or smoothness of the surfaces material. Micro-surfaces imitates the tiny imperfections on surfaces that are too small to be seen by the naked eye. These imperfections leads to blurry reflections and the parallel rays of light are scattered, because they hit the rough surface with a different angle (Lengyel, 2012).

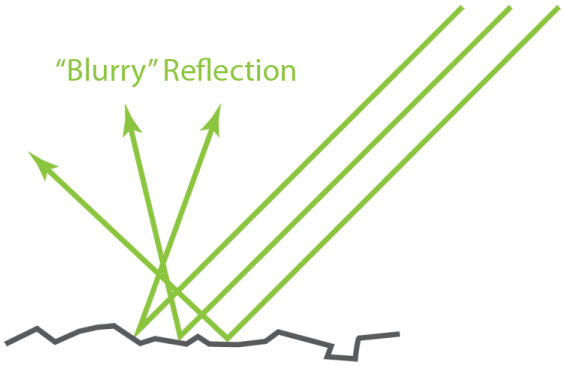


Figure 3.32: Micro-surface reflection

**Global Illumination**

Previously the render pipeline did only shade an object on it's surface angle to the light sources. However when light is reflected from a surface, then the reflected light will light objects that aren't lit directly from a light source. Global Illumination is the model that aims for simulated effects like reflection, refraction, color bleeding, and soft shadows, which can make the graphics look more realistic (Yang et al., 2009).

### 3.6.4. Shadows

Without shadows it is hard to determine the spatial relationship between objects in a 3D screen and adds a great deal of realism to a screen. Shadow generation in real time graphics are categorized into two categories: Shadow mapping and stencil shadows. Shadow mapping is the technique that will be described, because stencil shadows are impractical for rendering out objects using alpha textures. This means that in order to get the best result with stencil shadows all the leaves have to be geometric objects, and not textures with alpha maps (Lengyel, 2012).

Shadow mapping is a technique where a map is rendered out from the perspective of the light source. Each pixel in the shadow map holds a value, that represents the depth of the point to the light. The shadow map and the camera are in different clip spaces, but by if the camera and the light both have free line of sight to a given point in space then the point should be lit, however if the vector from the light to the point ends before reaching the point then the light is blocked and then point is in a shadow (Lengyel, 2012).

## 3.7. Level Design

The design of the test level might influence the perceived realism of the tree growth, because of the different growth factors that comes into play. So in this section different techniques for distributing the vegetation and how the players should navigate in a game level will be analyzed.

### 3.7.1. Vegetation Distribution

There are many ways to distribute the vegetation in a game level. Games like Uncharted the level designers have all the game assets and they manually make all the details in the level, while games like Spelunky, has pre-made rooms that get random details added using a procedural algorithm to insure that the player has a free path to the goal.

#### Perlin Noise

Noise algorithms have been used as an alternatives for recursive functions, because they requires less computing power. Noise functions can also be mapped to texture, which in many programs are quicker to look-up instead of calculating them. That is why that most noise are texture look-up in shaders, with pre-made noise textures (see figure 3.33). Most noise mapping functions works by assigning random value to a gray-scale texture. Ken Perlin came up with the idea to make noise pattern more natural looking, compared to white noise that are a series of uncorrelated random values (Korn et al., 2017).

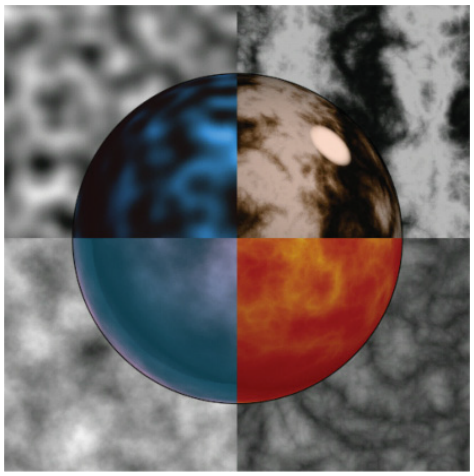

Figure 3.33: Different versions of Perlin noise textures

Perlin wanted to make new type of noise pattern that generated *"naturalistic visual complexity"* and he defined it as *"a texturing primitive you can use to create a very wide variety of natural looking textures"* and explained that *"combining noise into various mathematical expressions produces procedural texture"*(Korn et al., 2017). Perlin used frequency and amplitude to create his noise function. The noise texture is gray-scale image where the pixel values represent frequencies between $-1$ and $+1$. High frequencies means small details, that results in many small points in a given area, while low frequencies result in larger points. The amplitude determines how much the noise affects the object that uses the noise function. The implementation of Perlin noise is described as following (Korn et al., 2017):

1. Given an input point
2. For each of its neighboring grid points: pick a "pseudo-random" gradient vector.
3. Compute linear function (dot product)
4. Take weighted sum, using ease curves.

Because the curves are eased out then Perlin noise gives a more fluent look, where the values gradually change between extremes compared to white noise as shown in figure 3.34.

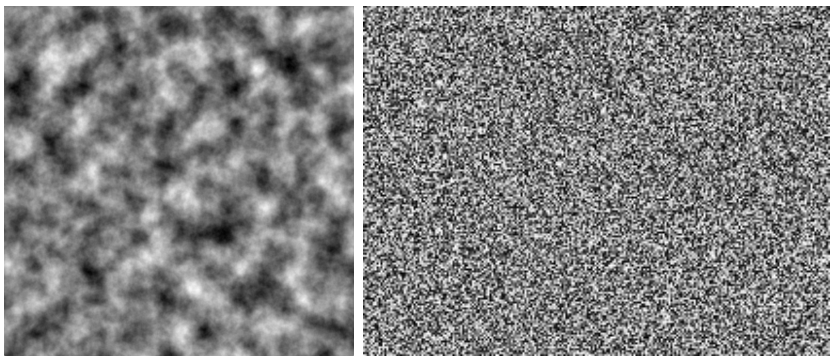

Figure 3.34: Perlin noise Vs White noise

Korn et al. (2017) used Perlin noise to distribute props in a game, and tested if users found it to be as realitic as props that was distribute by a level designer. In their study they found that players found the levels that used Perlin noise, was significantly more realistic and that the players also perceived them to be more aesthetically pleasing.

### Ecological Distribution

Perlin noise is still only a mathematical function and the nature is a even more complex ecosystem. That is why Onrust et al. (2017) came up with a technique that takes factors and apply them to the distribution to make it more realistic. They start with a landscape map that have the overall shape of the landscape. These landscape maps are used in combination with data for the different plants to generate the distribution of the plants on a grid based map. The distribution takes the following factors in to consideration (Onrust et al., 2017):

1. Plant species: the species of the plant for example oak or birch.
2. Plant spacing: the minimal required distance between plants. Often this is related to the plant radius or size of the specific plant species.
3. Plant level: different plant species that are placed in one group, because they have approximately the same plant spacing. The aim of creating this division is to allow the generation of plant distributions that contain plant species with large differences in plant spacing such as trees and flowers.

4. Plant patterns or patchiness: the patterns of a certain plant species. Plants of a species that exhibit high patchiness grow close together, while plants of a species with low patchiness grow scattered throughout the environment.

5. Plant coverage: the amount of occupation of a certain plant species in the environment.

Then from the distribution map the final terrain is populated with the different plant species. They propose to a least one or more 3D models per plant species for close up rendering and some billboard decorations for far objects.

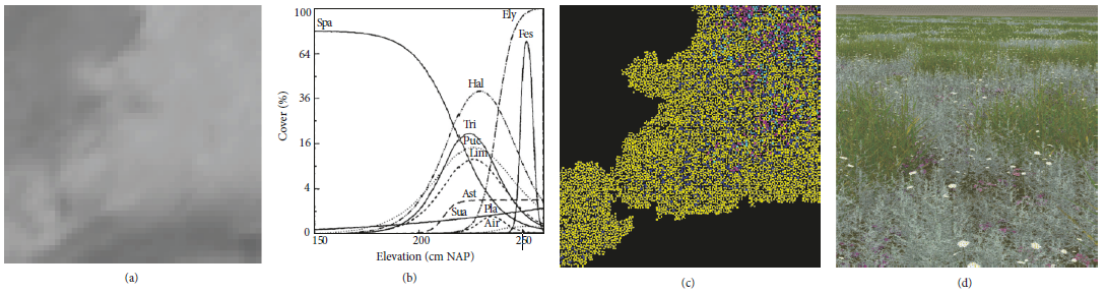


Figure 3.35: Procedural Generation of Natural Environments (Onrust et al., 2017)

Their technique is split into two parts. The first part generates all the possible plant position without assigning any plants. This is done by assigning a threshold. If a plant is able to growth, and if the input map (texture) has a higher value than the threshold then the tile is identified as vegetated. With all the vegetated tiles they generate points that have an user-defined minimum distance between the points, that are based on the plant size of the plant species(Onrust et al., 2017).

The second part assigns plant species to the different points. An input map holds values for each point on the grid. These values are translated into coverage values. The coverage values for each plant species have a certain range, and some plant species have overlapping value ranges. The data to determined the coverage value are statistical data that are mapped to information gotten from example soil quality maps. The different coverage values are merged into a single value, which is the minimum value, because they state that if the plant doesn't have the minimum growth conditions then the plant species will not be able to grow there(Onrust et al., 2017).

Using a noise function like Perlin noise, they generate clustering of plant species. Because the clustering are based on frequencies, then high noise value means that there is a greater distance between the plants, and a low noise value means a smaller distance between the plants. Then they assign a plant species to each point, by calculating the value of the clustering and the coverage, and from that they get a value, and if that value are higher than the threshold given for a corresponding plant species, then that plant are assigned. Multiple species might still be assigned to one point, this is resolved by assigning the plant with the highest threshold(Onrust et al., 2017). In figure 3.35, the whole process can be seen, and in figure 3.36 is it possible to see how the different plant species are assigned.

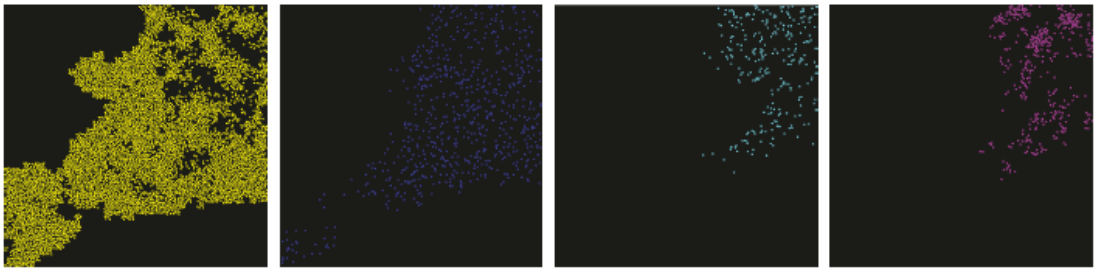

Figure 3.36: Each plant species assigned positions

### 3.7.2. Navigation

Game environments should be easy to navigate for the players and because they aren't familiar with the environment beforehand. Then it's necessary to provide the player with information for them to successfully reach the end destination.

Vinson (2003) has come up with a set of guidelines (Table 3.5), that helps the player to navigate by placing visual cues in the game environment. These cues are classified as landmarks (Table 3.6). If the player is unfamiliar with an environment then they rely heavily on landmarks for navigating. It could be that they follow a path, a river, or signs that point in the right direction (Vinson, 2003). But most of Vinson (2003), guidelines appears to man-made features, about navigating in a forest. Navigation might not be as easy in a forest as in a man-made environment, but many of the features are still present, like animal paths and forest borders.

| Navigation Guidelines |
|---|
| Guideline 1: It is essential that the game environment contain several landmarks (table 3.6). |
| Guideline 2: Include all five types of landmarks in your game environment (table 3.6). |
| Guideline 3: Make your landmarks distinctive with features. |
| Guideline 4: Use concrete objects, not abstract ones, for landmarks. |
| Guideline 5: Landmarks should be visible at all navigable scales. |
| Guideline 6: A landmark must be easy to distinguish from nearby objects and other landmarks. |
| Guideline 7: The sides of a landmark must differ from each other. |
| Guideline 8: Landmark distinctiveness can be increased by placing other objects nearby. |
| Guideline 9: Landmarks must carry a common element to distinguish them, as a group, from data objects. |
| Guideline 10: Place landmarks on major paths and at path junctions. |
| Guideline 11: Arrange paths and edges to form a grid. |
| Guideline 12: Align the landmarks' main axes with the path/edge grid's main axes. |
| Guideline 13: Align each landmark's main axes with those of the other landmarks. |

Table 3.5: Navigation Guidelines by Vinson (2003)

The distribution of the appearance of vegetation might have influences on the players ability to navigate, because if the player is walking in a wood with trees of the same shape and appearance in a random distribution, then the player might have few points of reference. While if the distribution of the vegetation is based on ecology, then the game environment might have paths, forest borders and meadows, which can give the player reference point. The tree appearance might also influence on the player's

ability to navigate in the game environment as if there is a significant different tree, for example a dead tree, this that tree can be a reference point for navigation.

| Navigation Landmarks | | | |
|---|---|---|---|
| Types | Examples | Functions | Natural Alternatives |
| Paths | Street, canal, transit line | Channel for navigator movement | Animal path |
| Edges | Fence, river | Indicate district limits | Forest border |
| Districts | Neighborhood | Reference point | Forest, meadows |
| Nodes | Town square, public bldg. | Focal point for travel | Animal path crossing |
| Elements | Statue | Reference point into which one does not enter | Dead tree, big stone |

Table 3.6: Navigation landmarks by Vinson (2003)

## 3.8. State of the Art

SpeedTree is a commercial product that generates virtual foliage for animations, architecture and video games. Theit features include: Seasonal variations, that allows the tree to have different leaves depending on the season. Wind effect that both effects leaves and branches, simple collision detection, ambient occlusion models that darkens the interior when not computing AO in real time and more features. SpeedTree is a great tool to generate a lot of diverse trees using PGC, but it still has limited interaction with the player and the trees' shape are not adapting to it's environment. For this reason we are looking into the state of the art, to see which features we can expect to be developed from academia and then coming in the future.

### 3.8.1. Interaction Simulation and Rendering

Jahrmann and Wimmer (2017) and Fan et al. (2015) both proposed new grass-rendering techniques that are able to draw individual blades of grass as geometrical objects in real time. In addition they also have different techniques that evaluate individual blades of grass reaction to collision and wind.

#### Collisions and Wind

Both Jahrmann and Wimmer (2017) and Fan et al. (2015) use a technique where they simplify the grass model, where a blade of grass is simulated as a curved line. Jahrmann and Wimmer (2017) define the grass blade as 3 vertices $v_{0..2}$ that serves as control points for a quadratic Bezier curve (see figure 3.37 left). $v_0$ are the fixed position, where the blade is rooted to the ground. $v_2$ is the tip of the blade and $v_1$ is positioned as the up-vector from $v_0$, and as $v_2$ moves away from it's position directly over $v_0$ is the value of $v_1$ lowered. This insures that the height of the grass blade always is the same as illustrated in figure 3.37 (right).

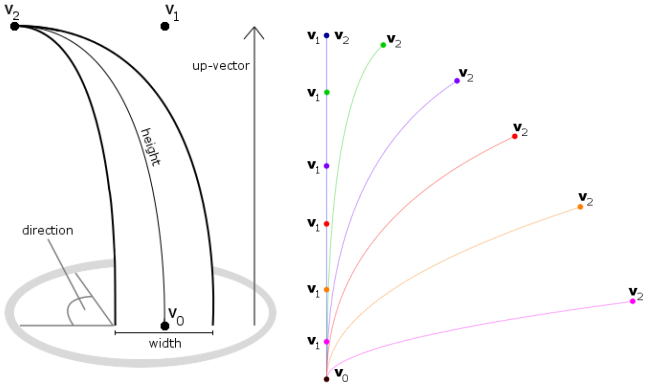


Figure 3.37: (left) Illustration of the definition of a blade of grass, (right) illustration of the relation between $v_1$ and $v_2$

Their physical model simulates natural forces and collisions with other objects. The calculation is conducted on the graphics card using a compute shader. The calculation only manipulates the tip of the blade $v_2$ at first, if the following rules are complied with: *"$v_2$ must not be pushed beneath the ground, the position of $v_1$ has to be set according to the position of $v_2$ and the length of the curve must be equal to the height of the blade of grass"* (Jahrmann and Wimmer, 2017). The blades of the grass have several attributes, like height, width, stiffness coefficient, up-vector and direction angle. These attributes are used in the calculation for collision and wind. The final displacement ($\delta$) of $v_2$ is determined by a summary of the forces acting on it (recovery r, gravity g and wind w and displacement d caused by collisions, see figure 3.38).

$$\delta = (r + g + w)\Delta t + d \tag{3.2}$$

Collisions are handled using spheres colliders on other objects. If the object has a more complex shape than a sphere then the object is made up overlapping sphere called a compound collider. This is however a heavy performance to calculate the intersection with the sphere for the whole blade of grass, that is why only $v_2$ and the midway point m are calculated for intersection. If a intersection is detected then the $v_2$ or m are translated to the nearest point on the surface of the sphere.

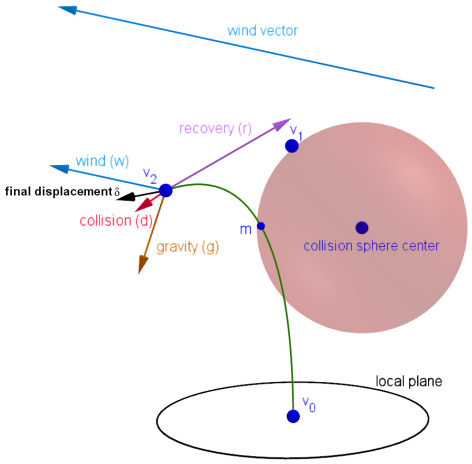


Figure 3.38: Illustration of the different influences that are considered in Jahrmann and Wimmer's (2017) physical model.

### Rendering

Jahrmann and Wimmer (2017) use tessellation that generates vertices along the curve of the grass blade. The blade geometry is initially a flat quad, but by evaluating the curve of the control point more vertices are generated that align quads with the curve.

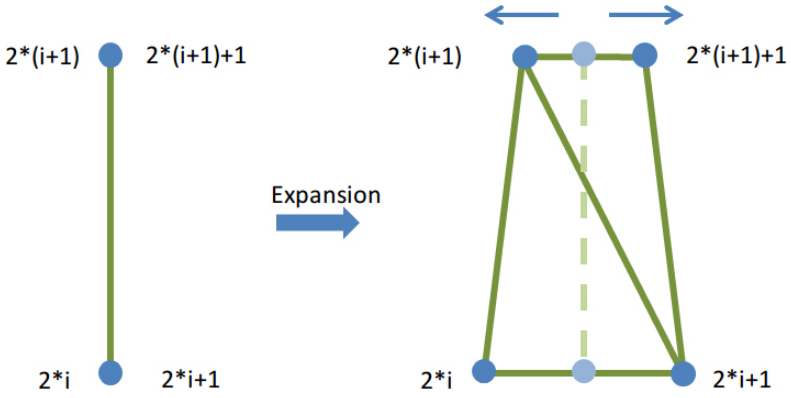


Figure 3.39: Each segment of the blade curve is in fact a degenerate quad and is expanded to a normal quad with given width.

Because the grass is a tiny 2D object that can impact performance a lot, Jahrmann and Wimmer (2017) have insured that only grass blades that are visible are rendered, all others are culled. They do this by checking if the blade is orientated towards the camera, and not parallel to the camera as the grass doesn't have any depth, then the grass is not visible from the side. All grass blades that aren't in the view-frustum are also culled to preserve performance and distance blades are culled if they are less than a pixel in width.

### 3.8.2. Self-Adapting Simulation

Pirk et al. (2012) have propose a technique that allows tree models to adapt to the environment where they are placed in. Their technique uses tree models that are created by artists who have the requirement that their shape should assume that the tree has grown in a isolated space with no external obstacles, given the tree optimal growth conditions. Their technique estimates environmental conditions that influence the tree growth and structure. The structure of the tree are controllable to allow to change branches orientation and direction as a response to insufficient amounts of light (Pirk et al., 2012).

The most important factor for the trees growth is light distribution. In the Pirk et al. (2012) technique the trees growth are affected by leaves and branches casting shadows on other leaves and branches below them. If branch is in shadow then it's leaves don't receive any light and then they don't produce any photosynthesis. If the leaves are in constant shadow then the tree cut off the water supply, and the leaves will wither and fall off, and maybe later the branch itself (Pirk et al., 2012). Pirk et al. (2012) have propose a technique that clusters leaves together and calculate if the center of the leaf cluster has enough light to sustain leaf growth. If the leaf cluster doesn't receive enough light then it is removed from the tree model.

Trees are constantly pruning themselves as branches and leaves become unproductive for the tree. That is why a isolated tree has more branches from root to crown, where trees in the forest have pruned off its lower-hanging branches, because itself and other trees are shadowing for them. A good example of where trees prune themselves are two trees that grows close to each-other looks like that they resemble a single tree (see figure 3.40).

Figure 3.40: Trees that have been grown very close to each other form a crown that resembles a single tree.

Pirk et al. (2012) also change the shape of the tree model, by changing the direction the tree is growing in. This change in growth direction is often seen as branches having a tendency to grow towards the light direction. It can also be that a branch needs to bend out of the way of an obstacle (see figure 3.41). A trees shape can also be changed

by the limiting factors like strong winds that can cause the tree to lean. A tree that has sustained damage will also try to right itself by changing it's growth direction upwards again or growing it's crown out in one direction to balance out the stress in it's roots.

Figure 3.41: Tree bending way from obstacle and self-pruning

### 3.8.3. Grammars and L-systems

Grammars are used to generate plants and one of the classic type of grammars is L-systems which are used to generate plants in procedural content generation. This section will describe how grammars and L-system works, and to give a overview of a possible way to make procedural generated trees.

#### Grammars

Grammars are a set of production rules for rewriting strings. This changes a string into another string, by a set of rules. An example of production rules could be:

$$\begin{aligned} A &\rightarrow AB \\ B &\rightarrow b \end{aligned} \tag{3.3}$$

Grammars works by simply going through the string, and when a symbol or a sequence of symbols match a rule then the symbol or sequence of symbols are replaced. For example with an initial string with an A, then the first rewrite would replace the A with an AB according to the first rule in 3.3. The second rewrite would replace the A again with an AB and the B with b as according to the second rule in 3.3, resulting in the string ABb. The convention in grammars is that upper-case characters are non-terminal, that can be rewritten further while lower-case characters are terminal that can't be rewritten (Togelius et al., 2016).

There is two types of grammars that are generally used in PGC: Deterministic grammars that have exactly one rule to each symbol or sequence of symbols, and non-deterministic grammars that have several rules that can apply to each symbol or sequence of symbols which can generate different results. Which rule that is chosen can be random or based on probabilities (Togelius et al., 2016).

### L-systems

The order of the rewriting is also interesting. A sequential rewriting goes through the string and rewrites it as read from left to right. This means when a production rule is applied to a symbol then the result is written into the original string. In parallel rewriting all of the rewriting are done at the same time, and written to a new string, which leaves the original string intact (Togelius et al., 2016).

L-systems are a class of grammars that uses parallel rewriting. This was introduced by biologist Aristid Lindenmayer to model the growth of organic systems like plants and algae (Togelius et al., 2016). An example of a simple L-system for yeast growth have the following production rules.

$$\begin{aligned} A &\rightarrow AB \\ B &\rightarrow A \end{aligned} \tag{3.4}$$

The resulting output are:

| L-system for yeast growth |
|---|
| 1. A |
| 2. AB |
| 3. ABA |
| 4. ABAAB |
| 5. ABAABABA |
| 6. ABAABABAABAAB |
| 7. ABAABABAABAABABAABABA |
| 8. ABAABABAABAABABAABABAABAABABAABAAB |
| 9. ABAABABAABAABABAABABAABAABABAABAAB..... |

Table 3.7: L-system for yeast growth

One of the interesting things about this sequence, is that regularity in the string are repeating it self over and over, and also the rate of which the string growth at. The lenght of the string is growing at the rate of the Fibonacci sequence: 1 2 3 5 8 13 21 34 55 89..., which have been observed in nature, from things like tree branching and how flowers arrange the petals on a stem (Togelius et al., 2016).

### Graphical interpretation of L-systems

The L-system can be used to generate 2D and 3D artifacts, by using the string as a set of instructions. One way of interpreting a L-system is with turtle graphics. The concept is that a turtle is moving over a plane holding a pencil and drawing a line as it moves. The turtle can move forward or turn left or right (Togelius et al., 2016).

The turtle then interprets the symbols in the L-system and draws the line. In figure 3.42 a graphical interpretation of L-systems can be seen.

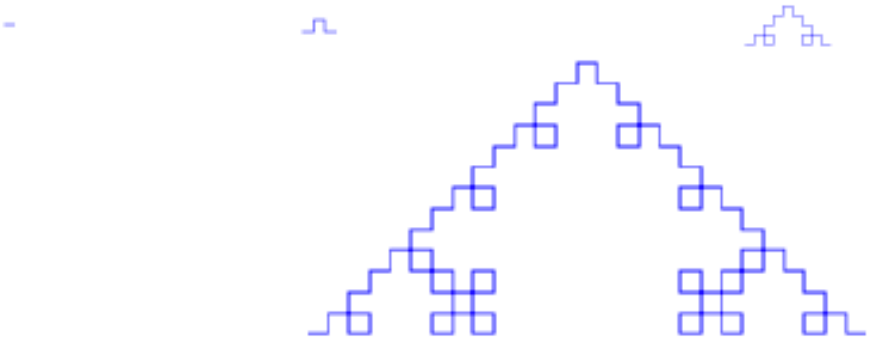

Figure 3.42: Turtle drawing after 0, 1, 2 and 3 rewrites

### Bracketed L-systems

One limitation of using L-systems as instructions for turtle graphics, is that the figure must be drawn in one continuous line. This limits the usefulness of L-systems because things like a tree needs multiple branches that have an end and then returns to the stem to continue to the next branch or the trunk. Here bracketed L-system is useful, because it has two extra symbols ([ and ]) which acts like any other symbols when rewritten, but when the turtle graphics interprets them they acts as "push" and "pop" commands. "[" saves the current position and orientation and "]" retrieves the position and orientation and sets the turtle to that. This allows the turtle to move back to previous position and continue a new branch. Bracketed L-systems can be used to make relative good looking plant structures as seen in figure 3.43 (Togelius et al., 2016).

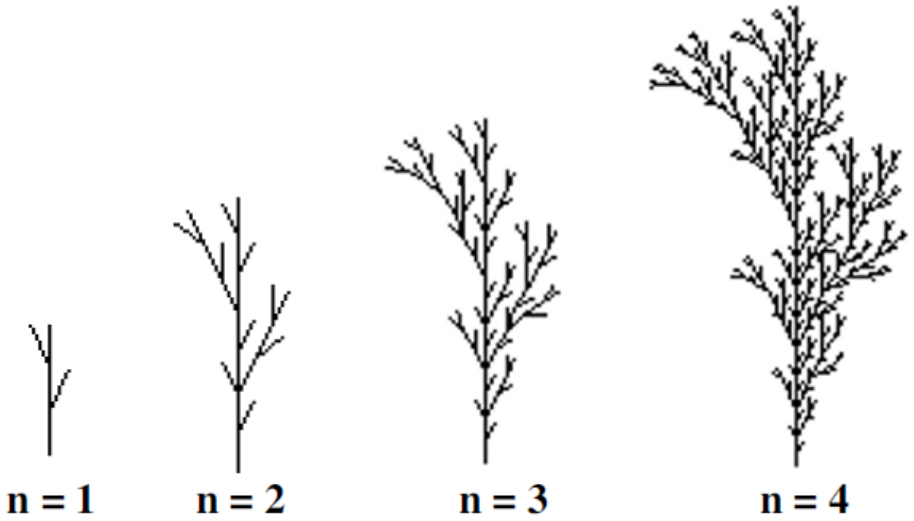


Figure 3.43: Four rewrites of a bracketed L-system

**Evolving L-systems**

L-systems can be modified to be apple to evolve such that the L-system takes in parameters that can change the string. Such an example is Ochoa that modified a L-system with the parameters: *"exposed surface area ("light-gathering ability"), structural stability, and proportion of branching points"* (Togelius et al., 2016). This made it possible to control the shape and appearance of the plant with some level of precision (Togelius et al., 2016).

### 3.8.4. Generative Design

*"Generative design mimics nature's evolutionary approach to design"* (Autodesk, 2018). Generative design is the process of using algorithms in a non-linear system to come up with a nearly endless number of unique and unrepeatable results. The designers or engineers gives the design software a description of the goal, along with parameters and constraints and then the design software returns with list of solutions. An important part of generative design is the feedback that is given when the solution is created (Autodesk, 2018).

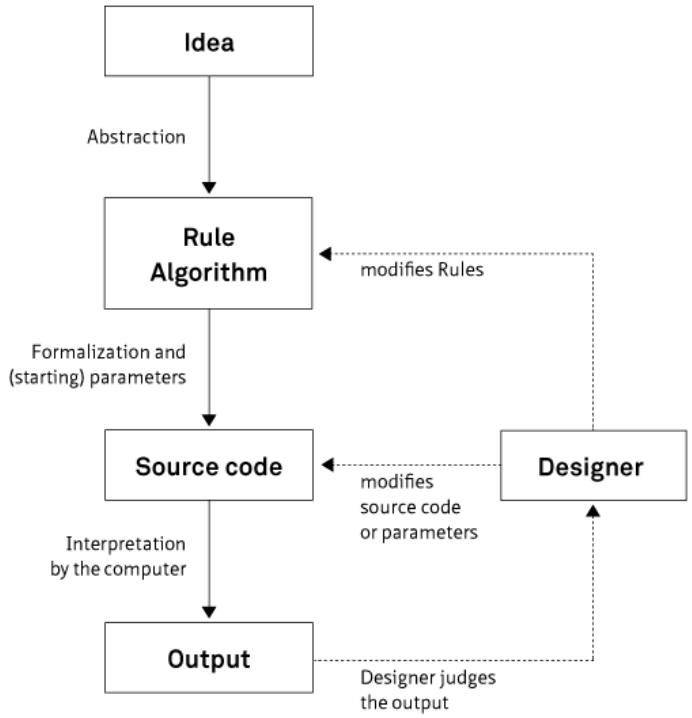


Figure 3.44: Process for creating generative design

The feedback can be done by having a designer that judges the result and then goes back and change the algorithm or change the input parameters and constraints (see figure 3.44). It has become a hot topic in academia to use artificial intelligence, to

judge the result and then learn from iteration what works and what doesn't work, and then use this data in the next iteration (Autodesk, 2018).

Generative design doesn't come up with a single solution but potentially thousands, and the designers can choose the result that fits best with the their demands (Autodesk, 2018).

## 3.9. Analysis Summary

In the analysis it was found that delivering vast and realistic game environments have become a huge undertaking for game developers. Players are expecting better graphics and more interesting worlds, and this puts a almost impossible demand on computer graphics artist to keep up. That is why game developers are looking more and more into procedural content generation, helping to create the content for the game world.

One area where PCG is heavily sorted after is in the generation of vegetation for video games. A reason for this is that, in an outdoor scene there can be hundreds trees and millions blades of grass and the player might quickly recognize that the same model has been used over and over. The realization that models are reused might lower the player's perception of the realism in the game.

With procedural generated vegetation there is an opportunity to make all the individual trees unique which should have an positive impact on the players experience in the game. But with PCG comes the problem of lack of control over the final outcome, that an experienced level designer and CG artist could use to create a truly realistic scene.

One way to overcome the problem of lack of control is to invest time in analyzing how vegetation grows, and create a robust algorithm. In section 3.5 it was found that there are factors that limits the growth of trees. The idea is that if a tree generation algorithm takes these factors into account when generating a procedural tree, then it might have a positive impact on the perceived realism of the tree. In state of the art, it was found that previous research has been looking at using L-system and particularly evolving L-systems to generate natural shapes.

Trees are not isolated, and their growth are not only determined by the tree itself by also by surroundings. This means that when looking at the realism of trees then, how the trees are distributed might have an affect on the players' perceived realism of both the trees and a forest as a whole. So a technique that used maps to procedurally generate natural environments was analyzed, and this project will find out if there is an affect if trees are random distributed or an ecological distribution model is used.

As the distribution of the trees is an important part of the level design, it was analyzed what is needed for a player to be able to navigate in a game environment. It was found that players need landmarks, that are easily identified so the players don't get confused. It might be hard to distinguish one tree to another, but the distribution of trees might have influence, when factors like clustering of species and spacing to other plants come in to effect.

Because games are an audio/visual experience and the focus of the project is on trees that are mostly a visual experience, the most important rendering techniques were analyzed to find out how to achieve the highest graphical fidelity. It was found that physical based rendering is the most appropriate technique for realistic shaders. A robust level of details system must be used to have high level of detail up close, by lowering the polygon count as the object moves away from the player. All this is needed to have good frame rates and realistic game objects.

That is why this project aims to find out if there is a measurable difference in the player's perceived realism of the vegetation, if the vegetation adaptives to the environment it's planted in. The project will also test if the distribution of the plants has any affect on the player's perceived realism of the game world.

# 4

# Final Problem Statement

*"To what extent is perceived realism affected by distribution and by tree growth factors of procedurally generated trees?"*

## 4.1. Design Requirements

This section condenses the analysis chapter into more specific design requirements for the testable environment.

### 4.1.1. Major Requirements

- Procedural generated to make multiple diverse instances
- Comply with trees' limiting factors
- No object clipping
- Wind interaction
- Ecological distribution model

### 4.1.2. Minor Requirements

- Frame rates above 60fps
- Use physical based rendering
- Scalable level of detail

# 5
# Methods

The methods chapter outlines testing strategies, with emphasis on how details in the final problem statement are tested. An iterative approach to testing and it has been applied throughout several stages of the implementation. Changes were gradually implemented before the final test.

## 5.1. Primary Hypothesises

The project seeks to answer the final problem statement: *"To what extent is perceived realism affected by distribution and by tree growth factors of procedurally generated trees?"*

The statement divides into the the dependent variable: perceived realism and the independent variables: ecology distribution and tree growth factors. This leads to the followings hypothesises:

**Null Hypothesis**

***"The perceived realism is not affected by distribution and by tree growth factors of procedurally generated trees."***

**Alternative Hypothesis**

***"The perceived realism is affected by distribution and by tree growth factors of procedurally generated trees."***

## 5.2. Test Setup and Sample Management

In order test the hypothesis there will be 4 test conditions:

| | Tree Growth Factors | |
|---|---|---|
| Distribution | Random Distribution<br>Without limiting Factors | Random Distribution<br>With limiting Factors |
| | Ecological Distribution<br>Without limiting Factors | Ecological Distribution<br>With limiting Factors |

Table 5.1: Test Conditions

Participants will be chosen with convenience sampling by asking students on campus. The samples are expected to primarily consist of students of Medialogy and possibly other similar technical studies from Aalborg University Copenhagen. It is expected that the participants have some experience with computers and games.

The test will be conducted using within subject testing, because it gives statistical stronger results, than between subject testing (Field and Hole, 2003). It is preferable to use within subject testing, because the results are based on the participant's perceptions of realism. To minimize any bias the conditions will be randomized based on the latin squares, so all conditions have an equal amount of been tested first, second, third and fourth (Field and Hole, 2003).

## 5.3. Measuring Perceived Realism

A questionnaire was developed by adapting the work of Ribbens et al. (2016), who has developed a questionnaire that measured the perceived realism in video games. Their questionnaire had 32 questions in 6 categories (see section 3.1.2). Because some of the categories didn't apply only two categories was chosen. The questions were rewritten in order to measure the perceived realism of the trees. The questionnaire is a 9-point Likert scale with the following questions:

**Simulational Realism - Distribution**

- The trees placement looked realistic
- The forest looked realistic

**Simulational Realism - Growth Factors**

- The trees looked realistic
- The trees' shape looked realistic
- The trees have sufficient detail, to appear realistic

**Authenticity**

- There was important features missing from the trees.
- There were lots of errors in the trees' appearance.

**Navigation**

- It was easy to find my way in the forest
- The trees were easy to use as landmarks to help navigate the forest

## 5.4. Player Performance

To test if the trees appearance and distribution have any affect on the ability to navigate the game environment, the players performance will be tracked. The player will be shown a path from the start position to an end position in the game environment. When the player have reached the end position, then the player will be reset to the start position. Now it's the player's task to find his way back to the end position along the same path, but the signs guiding the player are removed. To check if the trees appearance and distribution have any affect the following variables will be logged:

- Total Time
- Average distance to path
- Time off the path
- Distance traveled off the path
- Max distance traveled from the path
- Distance to goal when time is up

## 5.5. Computer Performance

Performance has the highest priority in video games, because it affects the smoothness of the frame rate (Claypool and Claypool, 2007) and in some fast paced-games, a few frames per second or frame jittery can mean the difference between life and dead. The performance has to be good enough so it doesn't have a negative impact on the game-play experience. To avoid frame rate as a variable in the test the frame rate will be locked at 30 or 60 frames per second, using V-Sync. But to evaluate the performance of the final product the following will be logged:

- Average frame rate
- Maximum frame rate
- Minimum frame rate

- Tree generation time
- Distribution generation time

## 5.6. Analyzing of the Results

**Measuring Perceived Realism:** The questionnaire will be evaluated based on a 9-point Likert scale and the mean scores will be calculated for each question. The scores will be tested for normal distribution and same variance using SPSS The result will also be analyzed with a 2 way repeated measures ANOVA. This is even though that Likert scales are ordinal data, but Wigley (2013), states that a Likert scale with 8 or more items can be analyzed as it is interval data. The University of California's Institute for Digital Research and Education states that Likert scales are data that falls between ordinal and interval. If the researcher has to make the assumption that the variable is interval, then it can be will assumed that the intervals are equally spaced (Bruin, 2018). The results will then be compared with the results from the non-parametric Friedman test.

**Player Performance:** The players performance are logged for each player and for each condition. The results will be evaluated based on the average and standard deviation and discussed with basis in the work from Vinson (2003).

**Computer Performance:** The performance is logged by running the different condition 10 times each on a high-end gaming computer. The results will be evaluated based on the average and standard deviation. The results will be discussed and may lay the foundation for future works.

# 6 Design

The design chapter translates theory and research from the analysis chapter. Designs were iterative and several short usability tests were made. Highlights from usability and user experience feedback tests are included in this chapter.

## 6.1. Graphics Fidelity

With the focus on the perceived realism of the trees, some level of graphical fidelity are needed. To get the most realism the test scenes will be using physically based shadering, using the linear color space, as this should give the best visual results based on the findings from section 3.6.

As nature scenes mostly only have one light source - the sun, there is no reason going with deferred rendering over forward rendering, as the advantage comes when more light sources are used in a scene (Owens, 2013).

## 6.2. Player Task

The player will be tasked finding his way in a forest. The player will be guided from the start point which is marked with a blue spot in figure 6.1 (left), by wooden signs (figure 6.1 right) placed along the path, which are marked with the yellow curved line. When the player reaches the end point, marked with a red spot in figure 6.1 (left), all the wooden signs will be removed from the game level, and the player's position is reset to the start position. Then the timer will start and the player will have a limited amount of time to reach the end point again.


Figure 6.1: Player Path

The idea to give the task to the players is that, they have to observe the environment to find their way to the end point. They will be more observant of the trees appearance and distribution. To keep the test fair, each condition will have their own path of the same length to avoid that the players learn the path and gets better performance on later run-thoughts.

## 6.3. Game Environment

The game environment takes it's inspiration, from real world where coniferous and deciduous forests coexist side by side. In figure 6.2, from a orienteering map, it can be seen that the bright green color is deciduous forests while the dull green color is coniferous forests. The map also shows meadows and lakes.

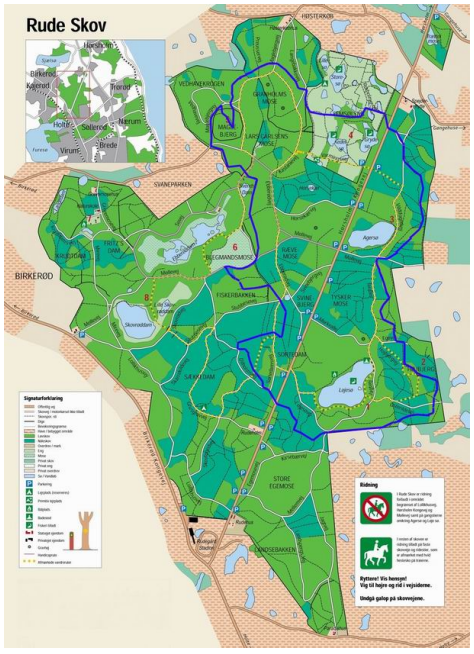


Figure 6.2: Orienteering Map

The game environment will be inspired by the pictures in figure 6.3 and the orienteering map. The forest will have different tree placed based on the distribution model used for the condition, and the forest floor will be covered with small forest plants and rocks for decoration.

Figure 6.3: Inspiration Forest - Coniferous (left) and deciduous (right) forests

## 6.4. Tree

The trees will be inspired by bracketed L-systems and evolving L-systems from section 3.8.3. The tree shape will be generated as a simple line shape, at the core of the trunk and branches. Using the idea from bracketed L-system the trunk grows upwards and the positions and directions of the branches are saved to a queue which will be run after the trunk are fully generated. Unlike traditional L-system the branches will not be a repetition of the trunk, but a combination of evolving L-systems for the limiting factor conditions, and random changes to small variables.

### 6.4.1. Shape

The idea generating the shape of the tree for the core, is to optimize the time it takes to calculate the shape. In order for the tree to have volume in the game, the points between each line will, be the center of an ring shape that represents the branch diameter. The rings will be joined with sides, and the final results should be a continuous semi-cylinder shape which follows the shape of the line. This is inspired by the technique Jahrmann and Wimmer (2017) used to simulate grass interaction from three points. This was analyzed in section 3.8.1 and seen in figure 3.39. The distance between the rings are a representation of the inter-node length that is the length of growth there is for a season.

By using the points in the line shape it should give a more natural branch distribution, because it's where the buds are on a real tree. The higher geometry resolution of the tree also allows the branches to have a more twisting shape, which normally comes from billboards. One draw back of this is that the tree will have a higher polygon

count than a standard SpeedTree. That is why a level of detail system has to make the game run with acceptable frame rates.

### 6.4.2. Limiting Factors

The test will test if limiting factors have an affect on the perceived realism, so there is going to be two conditions, with and without. Here the difference in the two conditions will be presented by describing how the condition with limiting factors will try to model real tree growth.

One limiting factor is shadows. If a branch is in shadow, then its leaves are also in shadow. As the branches are the eyes and ears of the tree, then they will try to get out of the shadow. The branch will grow in different direction in the hope of finding light, and it will also grow longer and weaker. This weakness make them more likely to break of in the autumn and winter months (see section 3.5 and 3.8.2). Because of this self-pruning branches that are in shadow will have less probability to grow large, and because of this the probability will be lowered for the branch generation if it's in shadow.

In addition to branches been in shadow by other object the tree also casts shadows on it self. To model a system that mimic that the tree has pruned of some of the twigs over time, a progressive model will be used for the tree branching. The idea is that the closer to the base of the branches, the lower is the branching probability, because it will have a higher likelihood of been self-shaded of the tree. As the branch is approaching the tip, then branching probability increases to model that branch gets access to more sun light and it also models the many active buds (section 3.5).

Some trees species have a tendency to have branches that grow upward while other just grow diagonal out from the trunk (section 3.5). This means that the branches of trees in different tree species can have different angles between the branches and the trunk.

A trees shape is also affected by the wind conditions that it is exposed to. A tree that stands in the open, has to adapt to the wind to avoid falling over. The tree has different options to balance the effects of the wind. One options is to re-balance the crown to change the center of balance and another for the tree to change the growth direction of the trunk and counter-balance that with the roots to make the tree more robust in the wind.

At last the conditions with limiting factors will also be checking for collisions with other object, to avoid a tree growing into an other tree, house etc. It could be argued that this should be in all conditions as it might have an impact on the perceived realism, because players would not expect a tree to grow through other objects. But it is not standard practise for tree asserts in games yet (Pirk et al., 2012).

## 6.5. Leaves

As the trees primary source of energy comes from photosynthesis, that are conducted in the leaves, and the leaves needs sun light to conduct this photosynthesis, then it's one of the limiting factors that will be designed. In section 3.5 Berlyn et al. (2018) states that leaves are arranged to avoid self-shading in order to maximize the light exposure, and in section 3.8.2 Pirk et al. (2012) states that if a leaf don't receives any sun light then it will have it's water supply off and the leaf will die. Pirk et al. (2012) clustered all the leaves of a branch into what they called leaf clusters and calculated the light exposure. This project will test each leaf's position if it's in shadow, and will distribute the leaves depending on the results. A game engine is used to test if a leaf is in shadow and it is possible to check if there is any obstacles between the leaf and the sun, by using ray-casting.

### 6.5.1. Rendering

When it comes to rendering out the leaves there is the options of using image-based billboards, geometry or a hybrids. As Jahrmann and Wimmer (2017) states in section 3.2, the billboards often use to get a high level of complexity of the visuals without having a high amount of polygons. But because of the technique used with semi-transparent textures, then some visual artifacts occurs (Jahrmann and Wimmer, 2017). Most video games use billboards because they can have complex shapes with a low poly count, but when the detail of the billboard increases to only having one leaf on each billboard then we run into some problems in the render pipeline. Because of the semi-transparent nature of the billboard, then it has to render all objects and not only the one closest to the camera (z-buffer). This could mean that all trees in a forest have to be rendered before culling the ones out that are hidden by others. This could lead to a serious rendering bottleneck. That is why most games have a geometric tree trunk and the largest branches, and the smaller branches, twigs and leaves are using billboards as seen in figure 3.9 in section 3.2.

However Jahrmann and Wimmer (2017) results with using geometry shaders to create grass blades, can also be transferred to creating leaves. The advantages of using a geometry shader over billboards are higher graphical fidelity as it eliminates the use of semi-transparent textures. It does how ever increase the amount of polygons on screen. Other advantages of using geometric shaders are that if any simulation is done for wind, then all the calculation is done on the GPU and no data has to be sent between the GPU and CPU.

So the design of the leaves is to generate a point cloud based on the results from the shadow testing and then using a geometry shader to create an individual leaf at each point in the point cloud. As Unity 3D's surface shader doesn't support geometry shaders, then lighting and shadows have to be written from the bottom.

## 6.6. Noise Distribution

Perlin Noise was found to be a realistic distribution technique by Korn et al. (2017), when creating terrain in a game and placing props. That is why Perlin noise will be used as the control test against ecological distribution.

The idea to use Perlin noise to distribute the trees in the game environment is to use layers of noise patterns to place the trees. The first layer is the overall possible space where trees can be planted, based on a user-defined threshold. If the value of the grid-node is over the threshold then the grid-node can hold a tree. This will create the overall landscape of the test scene, that creates forest areas and meadows.

The second layer is using Perlin noise with a higher frequency. This is used to distribute the trees within the grid-nodes that is marked as forest. By using the Perlin noise for the second layer it is possible to have a fluent density of vegetation. The different species are then randomly distributed in the vegetated areas.

The concept for the Perlin noise distribution can be seen in figure 6.4.

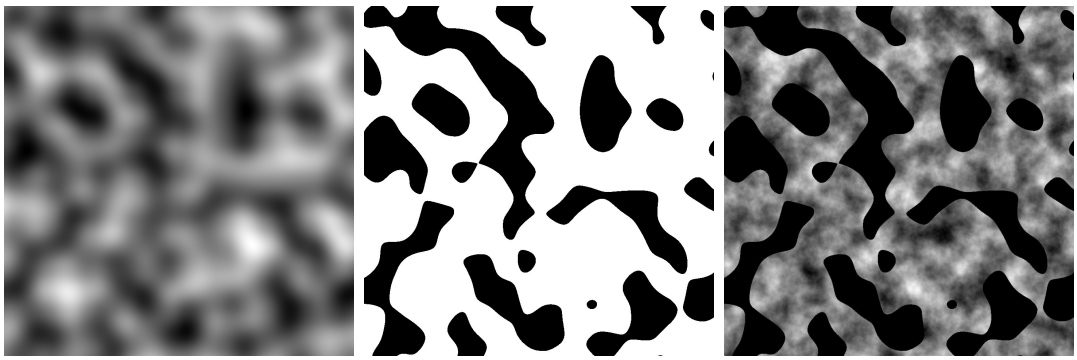

Figure 6.4: Perlin Noise Distribution - Left: input texture, Middle: Threshold (value are 93 of 255) are applied and the white areas are where trees can grow, Right: A second layer is added and density of the tree distribution is based on the values, which mean more trees in the area.

## 6.7. Ecological Distribution

With inspiration of the technique developed by Onrust et al. (2017), a pseudo simulation of ecological distribution will be used to distribute the trees.

As the test environment is not a real location, then there aren't any statistical data with the properties of the environment. However the overall technique by Onrust et al. (2017), can still by used with fabricated data or combined with available statistical data from different real world locations.

The general technique used in figure 6.4, can be reused in ecological distribution, but instead of using Perlin noise in the second layer, then data is used to distribute the trees with the following consideration:

1. Plant species
2. Plant spacing
3. Plant patterns or patchiness
4. Plant coverage

With these considerations the forest areas should look more natural than the random distribution of the plant species used in Perlin noise distribution

## 6.8. Procedural Generated Terrain

Using Perlin noise to generate a rough terrain, so the forest floor is not perfectly flat. The advantages by using Perlin noise over white noise are as previously mentioned in section 3.7.1, that the Perlin noise has a more fluent look as it uses curves. And by combining multiple noise patterns it can give areas more or less roughness.

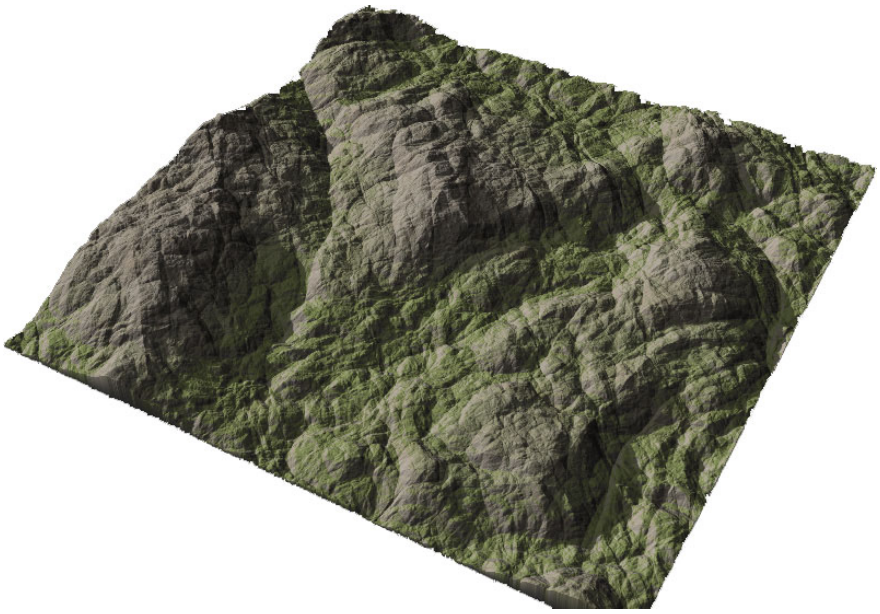

Figure 6.5: Perlin Noise Generated Terrain

## 6.9. Iterations and Re-Design

The tests were used to identify several bugs and issues, as well as to gather feedback about the user experience.

### 6.9.1. First Iterations and Re-Design

The first test was conducted March 26th and identified several bugs and issues, as well as gathered feedback about the user experience. The first test included a quick and dirty test with a semi-structured interview afterwards, with participants of opportunity. The following problems was observed and addressed.

- Orientation of the leaves
- Variation in shading
- Branches look quite large in the tips
- The leaves do not cast shadows
- Density of the leaves
- The trees seem very bare
- Size ratio between tree and leaf
- Specularity on the tree trunk
- Environment needs to be more natural

The leaves orientation and shadows was corrected by implementing shadows in the geometry shader, and using the normals to change the orientation. There was added more textures to give greater variation in the shading and the specularity was reduced in the material. The algorithm was change to produce more smaller branches that have a smaller radius, and combined with adding leaves at other points than the tip of the branches gave a higher denisty of leveas.

### 6.9.2. Second Iterations and Re-Design

The second test was a combined user experience and performance test, that was conducted April 18th. The following problems was observed and addressed.

- Faster tree generation
- Better frame rate
- Wind have too much of a pendulum effect
- Branches are too massive/big

The generation of the trees was improved by changing the data input from an object to a struct which improved the generation time of trees from around 10000ms per tree to around 500ms. The performance test also identified that using billboards in the geometry shader, gave to many rendering bottlenecks and by creating geometric leaves the frame rate went from 10fps to around 100fps for a 10 x 10 grid of trees.

The test participant identified the wind movement of the leaves' having too much of a pendulum effect, this was addressed by changing the displacement effect with additional sinus and co-sinus calculations. The algorithm was changed again to avoid the massive branches, by reducing their radius further.

### 6.9.3. Pre-Pilot Test and Re-Design

The test was conducted to test the test, where the participants would think out loud.

- Game environment was big
- Floating grass
- Needed sound
- To hard to find the way back to start

The test identified that the task was too hard. The original plan was to have the players walk from the start point to the end point and then back track to the start point. However that was found to be too hard, so instead now they are reset to start and have to follow the same route. It was also found that a 2 minute route was way to long, for the participants, and a route around 30 seconds was agreed on, also to keep the test time down. Small audio / visual element was changed to insecure that it didn't distract from the experience.

### 6.9.4. Pilot Test Iterations and Re-Design

The pilot test was conducted on the 4th of May in the Samsung Media Innovation Lab for Education (SMILE). It was a final test of the test equipment, to insure that everything ran smooth.

- FPS display distract from test
- To many signs

Participants stated that the FPS display was distracting, so it was removed. Some participants also found that there were to many signs, because instead of trying to navigate the forest and use the trees as landmarks they, felt more like trying to walk on a line, which lead them to not been able to find the right route afterwards.

# 7 Implementation

This chapter covers the implementation of the prototype that was used for testing the final problem statement. The prototype was developed in Untiy 3D. Other game engines could have been used like CryEngine, Unreal Engine 4 or Lumberyard, but was chosen of more experience with this engine and the online community resources. The designed algorithm, was inspired by generative design (see section 3.8.4), where the source code was changed regularly to insure better control over the final results.

## 7.1. Tree

The tree generation algorithm is inspired by L-systems seen in section 3.8.3. The algorithm is the same for both the conditions with and without limiting factors, and it is controlled by Boolean operations that checks which condition to use. The structure of the algorithm will be explained using pseudo code and the code snippets. The complete code can be seen in the digital appendix.

### OnEnable

The OnEnable function is used to setup all the needed deendencies that are needed to generate the tree. If a tree already have been generated before then the function will return to avoid generating duplicates. If it's the first time the tree is generated, then the function generates the empty game objects that holds the different level of detail objects, and mesh filters and renderers are added to all LOD obejcts.

OnEnable() - Pseudo Code

```
if object already setup
    return
find all the child object used for LOD system
if not found
    create LOD children
    add meshfilter
    add mesh render
    assign components
```

### Update

By using the Update function instead of Start or Awake functions the variables of the tree can be charged at run time, and it helps to debug as the prototype doesn't have to be restarted to get a new result.

The Update function starts by checking if any of the variables used to generate the tree are changed. If there isn't any changes, it will return without doing anything. If minimum one variable is charged, then it resets all the variables that have been used to generate the previous tree. Then the GenerateTree function is run and the generated result is used as the mesh collider. Finally the tree is set as static to optimize the performance in the game.

Update() - Pseudo Code

```
check if change are made to variable
    clear the queues
    Reset global variable
    GenerateTree()
    add mesh collider
    assign mesh collider
    is object as static
```

### GenerateTree

The GenerateTree function is the function that makes the initial setup that is needed for the BranchGen function that makes the mesh. The GenerateTree function also makes the level of detail setup. The GenerateTree function starts by checking if there is a list to hold all the vertices. If there isn't, then it creates one. Then it creates the branch cross-section ring shape based on the data the user has inputted, and all the rest of the data are inputted into structs that are queued. This is inspired by the Bracketed L-system in section 3.8.3. As long as the queue contains elements and the vertex limit isn't reached there will be generated new branches. It is also the GenerateTree function that saves and assigns the different level of detail meshes, used to optimize the performance. The leaves point cloud is also generated from the data it gets from the BranchGen function.

GenerateTree() - Pseudo Code

```
Set object as not static
If there is not a vertex list
    Create lists for holding generated vertices
else
    Clear lists for holding generated vertices
create ring shapes
create trunk data and save in branch struct
Enqueue branch
while there is still branches left in the equeue and the the vertex amount are not over the
 ↪  limit
    dequeue branch
    if the branch level are equal to the LOD level
        Set the tree mesh from the generated vertex lists

    BranchGen()         // Main branch recursive function to generate tree

If tree have leaves
    Generate leaf point cloud

create and assign LOD system
```

### BranchGen

The BranchGen function is the main recursive function, that generates the trunk and all the branches. The fucntions starts by adding the branch cross-section ring shape around the point, that is given as a parameter. Then it creates quads from the previous branch cross-section ring shape. This creates a cylinder that is the shape of the branches that is represented by the inter-node length.

Then the radius of the branch is made smaller and if the radius is below the threshold or the vertex limited is reached, then the branch end and is capped of. If the tree has leaves then it is checked if the end is in shadow, and the amount of leaves are determined based on the result. The function returns.

As the branch radius becomes smaller then there is a threshold that determined if it has leaves on it's smaller side branches. These areas are also checked for shadows.

The direction of the next branch is calculated from a combination of random angles and wind that bend the trunk and branches according to the wind direction.

To avoid that the tree's branches are clipping through other objects ray casting is used to check if the next position is blocked. If it is blocked then the branch is capped off, and the function returns.

A algorithm is used to make it more likely to have more smaller branches and twigs as the radius comes closer to it's minimum radius. This algorithm is used to determined the branching probability. The branching probability is also effected if the branching point is in shadow or not. The next branch is saved as a struct that is added to the queue in the GenerateTree function.

The final calculated factor is the upward growth factor and this is added to the branching angles. Then the branch calls BranchGen again to continue the branch.

BranchGen() - Pseudo Code

```
Add ring vertices
After first base ring is added
    Create new branch segment quads, between last two vertex rings
Make radius smaller
if reached minimum radius, or ran out of vertices
    Create end caps
    if tree have leaves
        if the checking for in shadow
            if in shadow
                create less then max amount of leaves
            else
                create max amount of leaves
        else
            max amount of leaves
    return

if tree have leaves and branch radius is small enough for leaves
    if the checking for in shadow
        if in shadow
            create less then max amount of leaves
        else
            create max amount of leaves
    else
        max amount of leaves

Continue current branch (randomizing the angle)
    if affected by wind
        randomizing the angle and add wind factor and direction
    else
        randomizing the angle

Check if change angle makes the branch hit something
    if checking for collision
        if collision
            create end cap
            return

if the should have more small twigs and branches as the radius gets smaller
    calculate the branch radius compared to max and min

Do we branch?
Check branching probability are bigger than random value
    if the checking for in shadow
        if in shadow
            less chance for branch
        else
            create branch
    else
        create branch

if the branch should angle upward
    angle branch upward

BranchGen();
```

### 7.1.1. Limiting Factors

This section will explain how the limiting factors are implemented and compare the results.

### Affected by wind

The wind affects the tree growth in two states. The trunk of the tree isn't affected the same way as the individual branches. It's implemented this way because the trunk has more mass to bend. The implementation of the wind affect, takes the wind direction and makes the branches orientated in that direction. This makes the crown bend over as seen in figure 7.1.

BranchGen() - Wind

```
1  if(AffectedByWind){
2      if(level == 0){
3          x = Mathf.Sin(windDirction * Mathf.Deg2Rad) * windFactor;
4          z = Mathf.Cos(windDirction * Mathf.Deg2Rad) * windFactor;
5          if(position.y > branchingHeight){
6              x = x + (Random.value - 0.5f) * Twisting * position.y/4.0f;
7              z = z + (Random.value - 0.5f) * Twisting * position.y/4.0f;
8          }
9      }
10     else{
11         x = Mathf.Sin(windDirction * Mathf.Deg2Rad) * windFactor;
12         z = Mathf.Cos(windDirction * Mathf.Deg2Rad) * windFactor;
13         x = x + (Random.value - 0.5f) * Twisting * position.y/4.0f;
14         z = z + (Random.value - 0.5f) * Twisting * position.y/4.0f;
15     }
16 }
```

Figure 7.1: Affected by wind. - (left) not affected, (right) affected and the tree bends to the left

### Check for collisions

To avoid that the tree's branches are growing through object ray casting is used to check that there isn't any obstacles between the current branch segment and the next. If there is a obstacle then the branch tip will be capped off. The results can be seen in figure 7.2.

BranchGen() - Check Collision

```
1  if(Physics.Raycast(transform.position + lastPosition, position-lastPosition, SegmentLength
    ↪   * 1.5f) && checkCollision && !root){
2      CreateCap(lastPosition, texCoord, _NumberOfSides);
3      return;
4  }
```

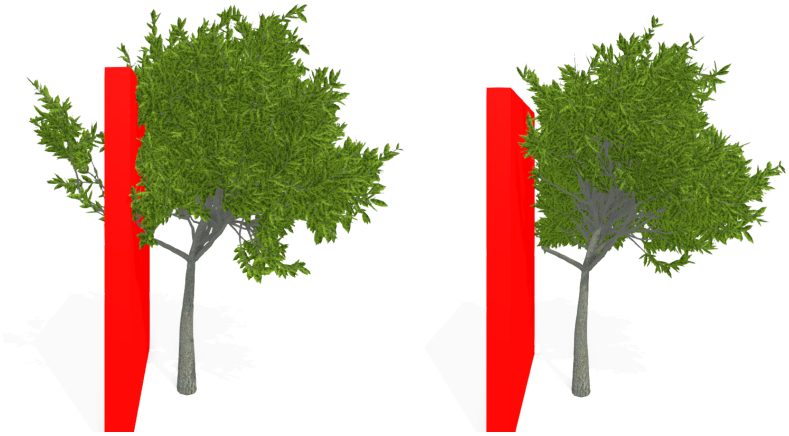

Figure 7.2: Check for Collision. - (left) The tree is clipping into the obstacle, (right) the tree branches stops before growing through the red wall

### Progressive branching

The progressive branching function works by checking the radius with the start radius and minimum radius. From the two reference points the progress of the branch can be calculated and as the branch is coming nearer to the tip the BranchProbability is increased (see figure 7.3).

BranchGen() - Progressive branching

```
1  if(moreSmallBranchs){
2      progress = 1 - ((radius-minimumRadius)/(startRadius-minimumRadius));
3  }

4  if(RNGValue < (BranchProbability/(float)i) * progress && radius * RadiusStep *
    ↪   BranchRadiusStep * 0.9f > minimumRadius){
5      create branch;
6  }
```

### Upward growth

Upward growth works by spherically interpolates between the direction calculated from all the previous factors and a upward direction. The amount of interpolation is determined by the same progress calculation from progressive branching and a upward growth factor that can be set by the designer. The result can be seen in figure 7.4.

Figure 7.3: Progressive branching - (left) a equal distribution of branches, (right) there are more branches when nearing the tip of the branch.

BranchGen() - Upward Growth

```
1  if(growthUpwards && moreSmallBranchs){
2      branch.transform.rotation = Quaternion.Slerp(transform.rotation,
       ↪   Quaternion.LookRotation(Vector3.forward), progress * upwardGrowthFactor *
       ↪   Random.value);
3  }
```

Figure 7.4: Upward growth - (left) the branches grow in randoms directions, (right) the branches have a slight upward direction.

### Check for shadow

The idea is to cast a ray from the leaf's and/or branch's position towards the sun, but because the "sun" in a game environment is a directional light object, that has a position that is much closer that the real sun, it makes more sense to cast a ray from the leaf's and/or branch's position in the opposite direction of the sun rays. The result is that there are less branches and leaves in the shadow, as shown in figure 7.5.

BranchGen() - Check Shadow

```
if(checkShadow){
  bool shadow = Physics.Raycast(transform.position + lastPosition, lightDircition, 1000f);
  if(!shadow || LeafProbability > Random.value){
      if(shadow){
          Leaf(quaternion, lastPosition, Random.Range(0,leafPerPoint));
      }
      else{
          Leaf(quaternion, lastPosition, leafPerPoint);
      }
  }
```

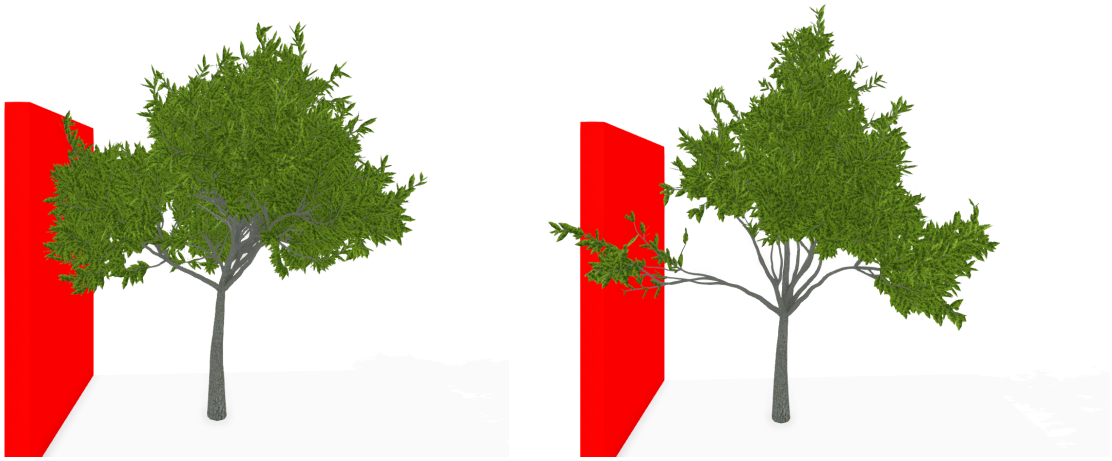

Figure 7.5: Check for shadow - (left) standard tree, (right) tree that has less branches and leaves because of the shadow cast by the red wall.

### 7.1.2. Level of detail system

Because there isn't an automatic LOD system for procedural generated content in Unity3D was it necessary to come up with a system, that reduced the amount of polygons on screen when objects was further in the distance. This LOD system works by only rendering the largest branches and the trunk at long distances, and when the player moves closer to the trees more and more branches rendered. The leaves also have a LOD system where at long distances a single quad is used for the leaves while 3 quads are used up close. In figure 7.6 the LOD system with three levels can be seen. Because the branches arn't rendered at long distances, the leaves are floating in midair on the tree to the left with least details. Because the branch would be almost only a line at that distance then it isn't a problem, and this preserves the overall shape of the tree at all distances. One element of this LOD that still could be optimized is that the trunk and largest branches still have the same vertex count. A better system would simplify the shape of the trunk to save vertices.

Figure 7.6: Level of detail system - The same tree shown with it's three levels of detail. (top) with leaves, (bottom) without leaves.

## 7.2. Leaves

The leaves could have been made using different techniques as discussed in section 3.2. One way was to create billboards in the same mesh as the tree branches, but this would used the same vertex list, and because Unity3D has a max vertex count of 65.000 vertices it would limit the amount of branches. An other option was to create a sub mesh that only held the leaves. Then there would be a list for branches and a list for leaves. However there is still a problem with the max limit count. With a sub mesh there still is a limit of 16.250 leaves on the tree. And another problem is that using the the vertex shader to generate the wind effect, will displace the vertices of the whole sub mesh and not for the individual leaf. The solution to this problem was to generate a point cloud, where each point represents the bud of one leaf, and then use a geometry shader to create the leaves.

### 7.2.1. Leaf point cloud

The leaf point cloud is used as the reference for both the leaf's position, and the direction that the leaf is growing. Each time the BranchGen function calculates there should be a leaf bud, then the leaf function is run. Depending on if the tree is in shadow, then the amount of leaves is changed to have more leaves in sun light and less in shadow. The leaf function saves the position of the leaf bud to the leaves vertex

list, then using Random.insideUnitSphere the leaf's direction is saved to the normals list. Because the normals are only a direction then all the leaves would have the same orientation. To change the orientation a random value is saved into the color value which is the rotation around the normal vector. This is an untraditional use of the normal vector and color values, but because the values are created in the geometry shader then it makes sense to use them to store values which need to be send from CPU to GPU, as they have to be instantiated anyway. All the leaf points are saved to a mesh of points in the GenerateTree function, which is read by the geometry shader.

Leaf point cloud

```
void Leaf(Quaternion rotation, Vector3 position, int _leafPerPoint){
    transform.rotation = rotation;
    for(int i = 0; i < _leafPerPoint; i++){
        Vector3 randomDir = Random.insideUnitSphere.normalized;
        leafPosition.Add(transform.position + position);
        indicies.Add(leafCount);
        Vector3 normalDir = rotation.eulerAngles * Mathf.Deg2Rad;
        normalDir.x += randomDir.x;
        normalDir.y += randomDir.y;
        normalDir.z += randomDir.z;

        colors.Add(new Color(Random.Range(0.5f,2.5f), 1, 1));
        normals.Add(normalDir);
        var uv = this.transform.localPosition / 1000f;
        uvs.Add(new Vector2(uv.x, uv.z));
        leafCount++;
    }
}
```

### 7.2.2. Geometry Shader

Because Unity3D's surface shader doen't support geometry shaders, it was necessary to write the whole shader from the bottom. There is some difference in the way that a standard shader and a geometry shader are written(see section 3.6.1). Because the vertices are the points that are the reference points used for the leaves positions, all the calculations performed in the vertex shader are moved to the geometry shader. This means that in MyVertexProgram only sends the information on to MyGeometryProgram without any transformations.

MyVertexProgram

```
Vertex2Geom MyVertexProgram (appdata_full v) {
    Vertex2Geom i;
    i.pos = v.vertex;
    i.uv = v.texcoord;
    i.normal = v.normal;
    i.color = v.color;
    return i;
}
```

In MyGeometryProgram the wind effect is calculated by using multiple sin and cosine functions. Then the shape of the leaf is specified by creating 8 points which together

create 3 quads. Then the positions are rotated around the normal that are given from the point cloud data, and the positions of the leaves are translated by adding the vertex position to the leaf's position. Then the quads are generated by using the buildQuad function.

MyGeometryProgram

```
1  [maxvertexcount(24)]
2  void MyGeometryProgram(point Vertex2Geom IN[1], inout TriangleStream<Interpolators>
    ↪   triStream)
3  {
4      float3 normal = IN[0].normal;

5      float3 v0 = IN[0].pos.xyz;
6      float3 wind =   float3(sin(_Time.x * _WindSpeed + v0.x) +
7                             sin(_Time.x * _WindSpeed + v0.z) +
8                             sin(_Time.x * _WindSpeed * 0.25 + v0.x) +
9                             sin(_Time.x * _WindSpeed * 0.25 + v0.z),
10                            0,
11                            cos(_Time.x * _WindSpeed + v0.x) +
12                            cos(_Time.x * _WindSpeed + v0.z) +
13                            cos(_Time.x * _WindSpeed * 0.25 + v0.x) +
14                            cos(_Time.x * _WindSpeed * 0.25 + v0.z)
15                            );


16     Interpolators OUT;

17     create 8 points float3 points in the shape of the leaf
18     rotate the points around the normal
19     create the 3 front quads
20     buildQuad(triStream , quad0, IN[0]);
21     create the 3 back quads
22     buildQuad(triStream, quad5, IN[0]);
23 }
```

BuildQuad takes 4 points and creates 2 triangles which combined is a quad. The normal direction of the quad is calculated as the input normal is used to specified the direction of the leaf growth. It is also in the buildQuad that all the calculation that normally are done in the vertex shader is conducted.

buildQuad

```
1  void buildQuad(inout TriangleStream<Interpolators> triStream, float3 points[4], point
    ↪   Vertex2Geom IN[1])
2  {
3      Interpolators OUT;
4      float3 faceNormal = cross(points[1] - points[0], points[2] - points[0]);
5      for (int i = 0; i < 4; ++i)
6          {
7          OUT.pos = UnityObjectToClipPos(points[i]);
8          OUT.normal = UnityObjectToWorldNormal(faceNormal);
9          OUT.uv.xy = float2((uint)i % 2, (uint)i/2);
10         OUT.uv.zw = float2((uint)i % 2, (uint)i/2);
11         OUT.worldPos = mul(unity_ObjectToWorld, points[i]);

12         #if defined(BINORMAL_PER_FRAGMENT)
13         OUT.tangent = float4(UnityObjectToWorldDir(points[0]-points[1]), 1);
14         #else
15         OUT.tangent = UnityObjectToWorldDir(points[0]-points[1]);
16         OUT.binormal = CreateBinormal(OUT.normal, OUT.tangent, 1);
17         #endif
18         TRANSFER_SHADOW(OUT);
19         ComputeVertexLightColor(OUT);
20         triStream.Append(OUT);
21         }
22     triStream.RestartStrip();
23 }
```

The fragment shader is just a standard fragment shader, as the data that is send from MyGeometryProgram, is the same as a normal vertex shader would make as output. The complete code can be seen in the digital appendix and the final result can be seen in figure 7.7.

Figure 7.7: Close up of the leaves used for testing

## 7.3. Distribution

The distribution algorithm is inspired by the ecological distribution in section 3.7.1. The structure of the algorithm will be explained using pseudo code and the complete code can be seen in the digital appendix.

### Map distribution

The map distribution script starts by randomly fill a map with 1's and 0's, then the map is smoothed using image processing. The results are the tiles where trees can be planted.

If the ecological distribution is used then perlin noise is used to generate height maps. One for each tree specie. The height maps are generated with different frequency's of perlin noise to create different clustering patterns. Then tree species maps are masked out with the vegetated tiles map. Then the vegetated tiles map is assigned to a tree specie so each tiles are vegetated, based on weights. Finally there is created space around trees which have larger spacing needs.

Random distribution just assigns a random specie to a tile on the vegetated tiles map. Then all the data are read and written into queue and then send to the tree spawner.

Distribution - Pseudo Code

```
1  Generate Maps
2  randomly fill the vegetated tiles map
3  Smooth vegetated tiles map

4      if ecological distribution
5          Use Perlin noise to generate species maps
6          mask out the species maps with vegetated map
7          assign trees to the by the weights
8          remove trees around tree species that demands more space
9      else
10         randomly assign tree species
11         mask out the species map with vegetated map

12 read the map and queue the data
13 spawn the trees
```

### Tree Spawner

When the different tree species are distributed, then all the trees are spawned. The tree spawner has all the variables that are needed to generate a tree as shown in section 7.1. Each tree specie has its own range. This allows each tree specie to have it's own unique appearance and it allows each tree within the same specie, to have it own unique look.

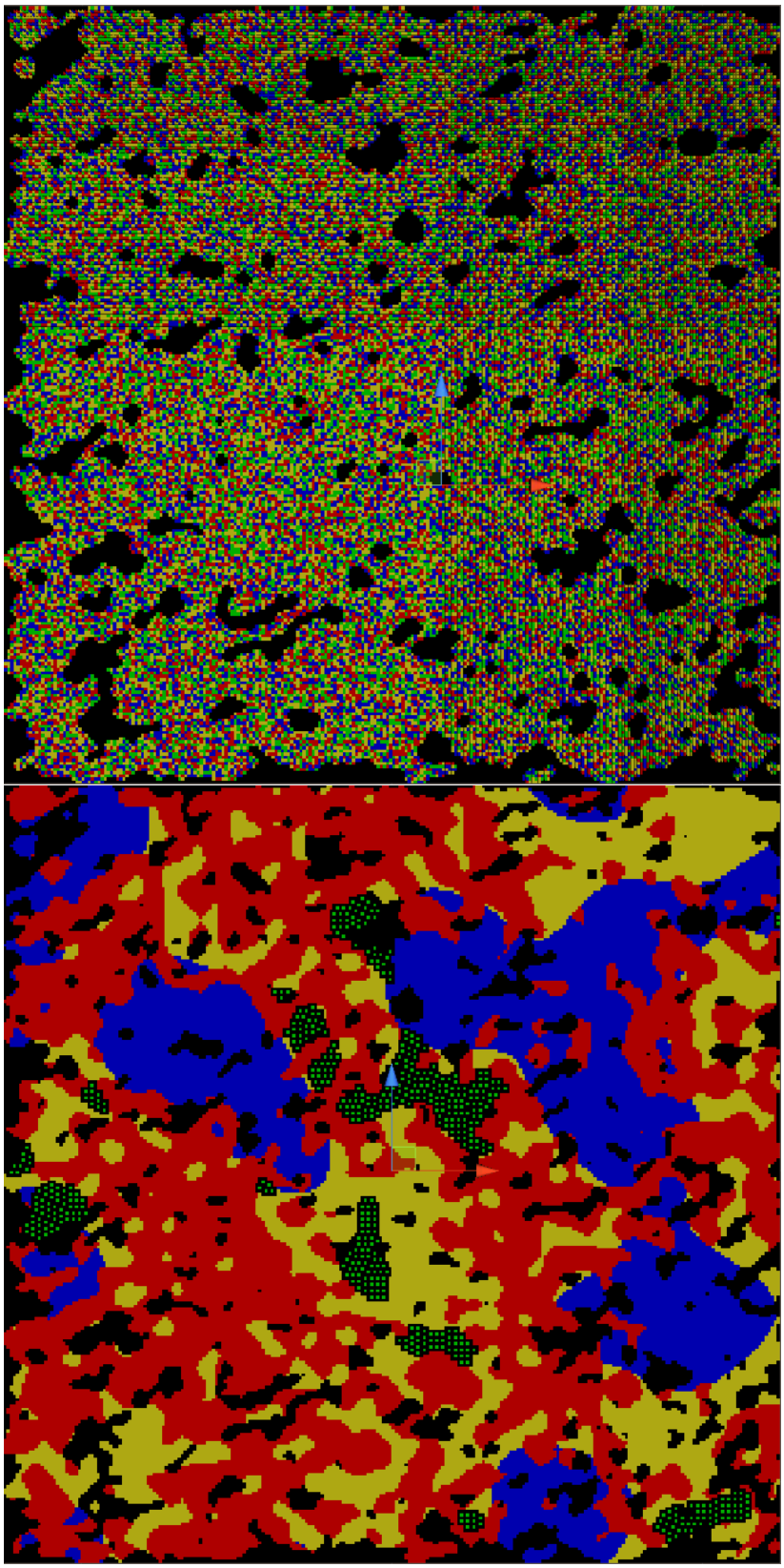

Figure 7.8: Distibution map - (Top) random distribution, (Bottom) ecological distribution.

Figure 7.9: Screen shots of the four conditions : (from top to bottom) random distribution without limiting factors, ecological distribution without limiting factors, random distribution with limiting factors and ecological distribution with limiting factors.

# 8 Evaluation

This chapter will attempt to draw conclusions from the final problem statement and hypothesis conditions established from it. The chapter will also describe the experiment and methods for analysis.

## 8.1. Objective of the Test

It is going to be tested whether the distribution and growth factors of trees in a game have an affect on the perceived realism or not. The test will be conducted using subjective self-report questionnaires, that are an adaptation of the questionnaires seen i section 3.1.2 and the final questions can be seen in section 5.3. It will also be tested if distribution and tree growth factors have an effect on the players' performance to navigate in a forest environment. At last the conditions will be tested by logging the frames per seconds, as a measure of performance.

## 8.2. Pilot Test

The pilot test was conducted on the 4th of May in the Samsung Media Innovation Lab for Education (SMILE) at Aalborg University. The pilot test was conducted as a structured test with the test participants thinking out loud, in order to observe any problems in the test. The pilot test yielded observations and suggestions to changes, which was needed to make the test more intuitive. The questionnaires was changed to make them and their meaning more clear for the test participants. One question was removed because it was too unclear and two new was added. Additionally there was also added further data to the log.

## 8.3. Final Test - Perceived Realism

The final test was conducted at Aalborg University Copenhagen in a controlled setting on May 7th and 8th in the SMILE Lab. The test had 20 test participants. The gender distribution was an over-representation of males with 80% men and 20% women. Age range ran from 21 to 29 years (M = 24.1, SD = 2.4). All the participants had experience with playing video games and they played between 0 to 19 hours a week (M = 7.5, SD = 6.1). The participants gave a self assessment of their experience of navigating in forests. The assessment was on a scale from 1 to 9, and the answers ranged from 1 to 9 (M = 5.1, SD = 2.5). Participants were distributed according to the Latin square balancing and all students came from different study directions at AAU Copenhagen. The test conditions were named after playing cards (Hearts, Clubs, Diamonds & Spades) to avoid player attributing any sense of improvement that names like 1, 2, 3, 4 etc or A, B, C, D etc. The test was conducted as a within-subject design that relays on a 2x2 factorial design crossing two types of tree distribution and two types of tree growth factors.

### 8.3.1. Test Procedure

The overall time period for each test participants were 15-20 minutes.

1. Participants agreed by signing an online written consent about the experiment before starting.
2. Participants answered the demographics part of the questionnaire and reads the test guide lines.
3. The participants played the first condition.
4. The participants answered the questionnaire for the first condition.
5. Repeat point 3 and 4 for each condition.
6. The participants answered summary questionnaire.
7. Oral feedback was noted.

### 8.3.2. Setup

The test was conducted on the SMILE lab's gaming machine THOR (see specification in table 8.1). The setting for brightness, contrast, volume and mouse sensitivity were identical for all participants. The test was conducted in the same room for all test participants with the same hardware.

| **Perceived Realism Test Computer:** |
|---|
| Intel i7-7700k @ 4.2GHz |
| GeForce GTX 1080ti |
| 32GB DDR4 RAM |
| Solid State Drive |
| Windows 10 |
| 32" QLED Curved Gaming Monitor CHG70<br>Headset: Steelseries Siberia v2 |
| Mouse: Logitech G600 |

Table 8.1: The specifications of the test computer

## 8.4. Player Performance

All player performance data was logged when the player played the different conditions. The data was written to external CSV-file. A heat-map that shows the players paths are also generated using the players position.

Heat Map

```
1     void Start () {
2         image = new Texture2D ((int)size.x,(int)size.y, TextureFormat.RGB24, false);
3     }

4     void Update () {
5             int x = (int)rigidbody.transform.position.x + (int)size.x/2;
6             int z = (int)rigidbody.transform.position.z + (int)size.y/2;
7             image.SetPixel(x-1,z-1,color);
8             image.SetPixel(x-1,z,color);
9             image.SetPixel(x,z,color);
10            image.SetPixel(x,z+1,color);
11            image.SetPixel(x+1,z+1,color);
12    }

13    public void printImage(){
14            image.Apply();
15            byte[] bytes = image.EncodeToPNG();
16            string time = System.DateTime.Now.Hour + " " + System.DateTime.Now.Minute;
17            File.WriteAllBytes(Application.dataPath + "/../"+ str +"/SavedScreen" + time +
                  ↪   ".png", bytes);
18    }
```

## 8.5. Final Test - Computer Performance

The performance test was conducted on the build computer. The test computer was a higher end computer than the build computer, but the build computer was closer to an average gaming computer, while still being a high end system.

### 8.5.1. Test Procedure

The performance test was conducted on the build computer (see table 8.2) with all the limits for frame rate off, and then the frame rate for each condition was tested 10 times, and a mean result was calculated for each condition.

### 8.5.2. Setup

A high end computer was used for the performance test, in order to not be limited by any hardware bottlenecks. V-Sync was disabled, as V-sync will sync the frame rate to the monitors refresh rate. A script was implemented to measure the frame rate, and write it to an external CSV-file (see FPS Measure).

| **Build Computer:** |
|---|
| Intel i7-4790k @ 4.4GHz |
| GeForce GTX 1080 |
| 16GB DDR3 RAM |
| Solid State Drive |
| Windows 10 Pro |
| Headset: Logitech G930 |
| Mouse: Logitech G900 |

Table 8.2: The specifications of the test computer

FPS Measure

```
public class FPSMeasure : MonoBehaviour {
    string path;
    float counter;

    void Start () {
        path = getPath();
        string createText = "Start of Test " + System.DateTime.Now +
            ↪  System.Environment.NewLine;
        File.AppendAllText(path, createText);
    }

        void Update () {
        counter = 1 / Time.deltaTime;
        PrintLog();
    }

    string Data(){
        return Convert.ToInt32(counter).ToString();
    }

    void PrintLog(){
        string appendText = Data() + System.Environment.NewLine;
        File.AppendAllText(path, appendText);
    }

    private string getPath(){
        return Application.dataPath + "/CSV/" + "Saved_data.txt";
    }
}
```

## 8.6. Results - Perceived Realism

Here the results from the individual questions will be presented in a visual format. The full data set can be viewed in the digital Appendix. The questions are divided into there separate subcategory (simulational realism, authenticity and navigation) The perceived realism score is measured on a scale from 1 (not realistic) to 9 (very realistic). The results will be analyzed using a 2 way repeated measures ANOVA, but because Likert scales are a grey zone, the results will also be analyzed using the non-parametric Friedman's two-way analysis of variance by ranks test, to confirm the results from the 2 way repeated measures ANOVA. In table 8.3 is the results summarized.

### 8.6.1. Simulational Realism

**The trees looked realistic**

A two-way repeated measures ANOVA was run to determine the effect of different distribution and growth factors on how realistic the trees looked. Analysis of the studentized residuals showed that there was normality except for random distribution with limiting factors ($p = 0.018$), as assessed by the Shapiro-Wilk test of normality and no outliers, as assessed by no studentized residuals greater than $\pm$ 3 standard deviations.

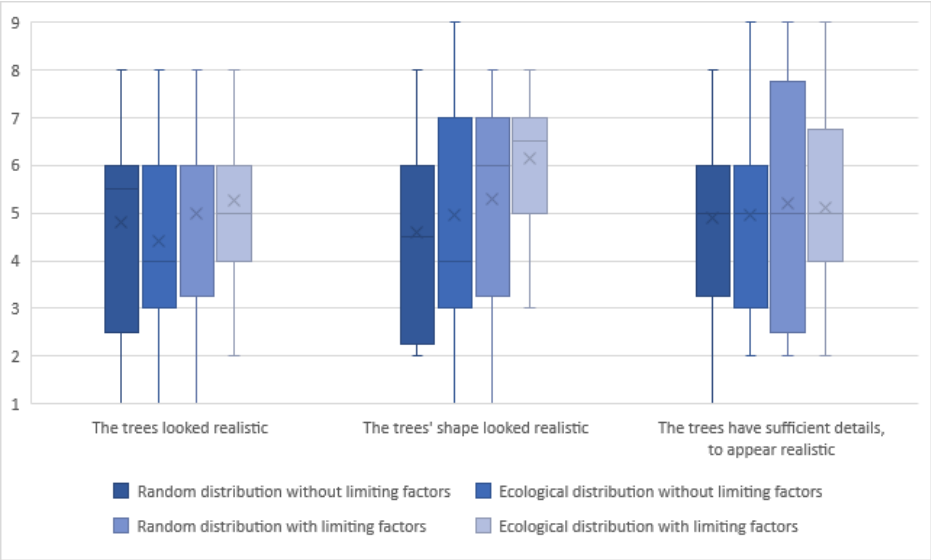


Figure 8.1: Results of simulational realism questions related to growth factors - Higher is better

There was no statistically significant two-way interaction between distribution and growth factors, $F(1, 19) = 1.993$, $p = 0.174$.

The main effect of distribution showed no statistically significant difference in how realistic the trees looked between conditions, $F(1, 19) = 0.112$, $p = 0.742$.

The main effect of growth factors showed no statistically significant difference in how realistic the trees looked between conditions, $F(1, 19) = 1.986$, $p = 0.175$.

A Friedman's two-way analysis of variance by ranks test was run to determine if there were differences of different distribution and growth factors on how realistic the trees looked. There wasn't a statistically significant, $X^2(3) = 2.727$, $p = 0.436$.

### The trees' shape look realistic

A two-way repeated measures ANOVA was run to determine the effect of different distribution and growth factors in "the trees' shape look realistic". Analysis of the studentized residuals showed that there was normality except for random distribution without limiting factors ($p = 0.025$) and random distribution with limiting factors ($p = 0.045$), as assessed by the Shapiro-Wilk test of normality and no outliers, as assessed by no studentized residuals greater than $\pm$ 3 standard deviations. There was no statistically significant two-way interaction between distribution and growth factors, $F(1, 19) = 0.872$, $p = 0.362$.

The main effect of distribution showed a statistically significant difference in "the trees' shape look realistic" between conditions, $F(1, 19) = 7.908$, $p = 0.011$. Post hoc tests using the Bonferroni correction revealed that ecological distribution was the best distribution ($M = 5.550$, $SD = 1.677$) compared to random distribution ($M = 4.950$, $SD = 1.843$).

The main effect of growth factors showed a statistically significant difference in "the trees' shape look realistic" between conditions, F(1, 19) = 5.085, p = 0.036. Post hoc tests using the Bonferroni correction revealed that limiting factors was the best growth factor (M = 5.725, SD = 1.592) compared to without limiting factors (M = 4.775, SD = 2.232).

A Friedman's two-way analysis of variance by ranks test was run to determine if there were differences of different distribution and growth factors in "the trees' shape look realistic". There was statistically significant, $X^2(3) = 8.847$, p = 0.031.

**The trees have sufficient details, to appear realistic**

A two-way repeated measures ANOVA was run to determine the effect of different distribution and growth factors as in the trees have sufficient detail, to appear realistic. Analysis of the studentized residuals showed that there was normality, as assessed by the Shapiro-Wilk test of normality and no outliers, as assessed by no studentized residuals greater than ± 3 standard deviations. There was no statistically significant two-way interaction between distribution and growth factors, F(1, 19) = 0.157, p = 0.697.

The main effect of distribution showed no statistically significant difference in the trees have sufficient detail, to appear realistic between conditions, F(1, 19) = 0.016, p = 0.900.

The main effect of growth factors showed no statistically significant difference in the trees have sufficient detail, to appear realistic between conditions, F(1, 19) = 0.472, p = 0.500.

A Friedman's two-way analysis of variance by ranks test was run to determine if there were differences of different distribution and growth factors as in the trees have sufficient detail, to appear realistic. There wasn't a statistically significant, $X^2(3) = 2.073$, p = 0.557.

**The trees placement look realistic**

A two-way repeated measures ANOVA was run to determine the effect of different distribution and growth factors as in "the trees placement look realistic". Analysis of the studentized residuals showed that there was normality except for random distribution with limiting factors (p = 0.048), as assessed by the Shapiro-Wilk test of normality and no outliers, as assessed by no studentized residuals greater than ± 3 standard deviations. There was a statistically significant two-way interaction between distribution and growth factors, F(1, 19) = 5.516, p = 0.030.

The main effect of distribution showed no statistically significant difference in "the trees placement look realistic" between conditions, F(1, 19) = 0.539, p = 0.472.

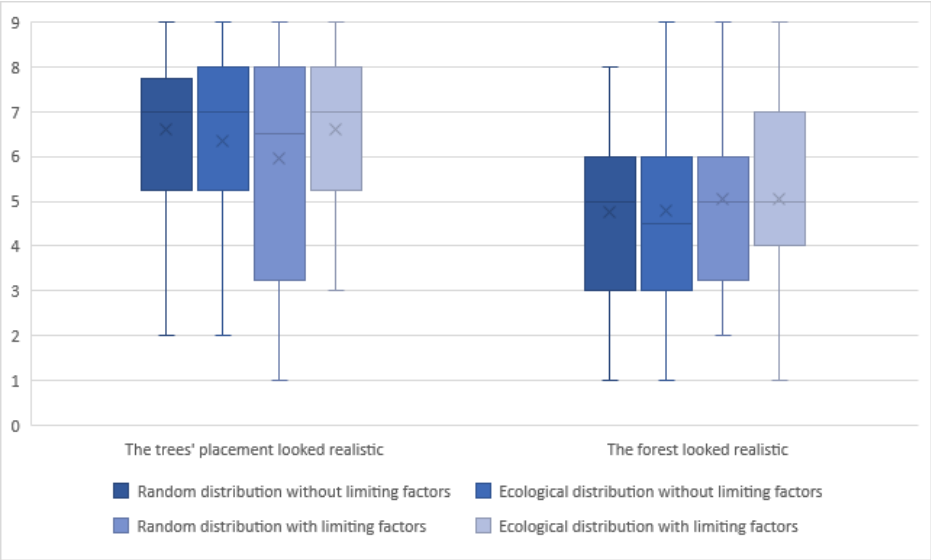


Figure 8.2: Results of simulational realism questions related to distribution - Higher is better

The main effect of growth factors showed no statistically significant difference in "the trees placement look realistic" conditions, F(1, 19) = 0.275, p = 0.606.

A Friedman's two-way analysis of variance by ranks test was run to determine if there were differences of different distribution and growth factors as "the trees placement look realistic". There wasn't a statistically significant, $X^2(3) = 2.287$, p = 0.515.

### The forest looked realistic

A two-way repeated measures ANOVA was run to determine the effect of different distribution and growth factors as in "the forest looked realistic". Analysis of the studentized residuals showed that there was normality, as assessed by the Shapiro-Wilk test of normality and no outliers, as assessed by no studentized residuals greater than ± 3 standard deviations. There was no statistically significant two-way interaction between distribution and growth factors, F(1, 19) = 0.027, p = 0.871.

The main effect of distribution showed no statistically significant difference in "the forest looked realistic" between conditions, F(1, 19) = 0.012, p = 0.912.

The main effect of growth factors showed no statistically significant difference in "the forest looked realistic" between conditions, F(1, 19) = 0.635, p = 0.435.

A Friedman's two-way analysis of variance by ranks test was run to determine if there were differences of different distribution and growth factors as in "the forest looked realistic". There wasn't statistically significant, $X^2(3) = 0.705$, p = 0.872.

### 8.6.2. Authenticity

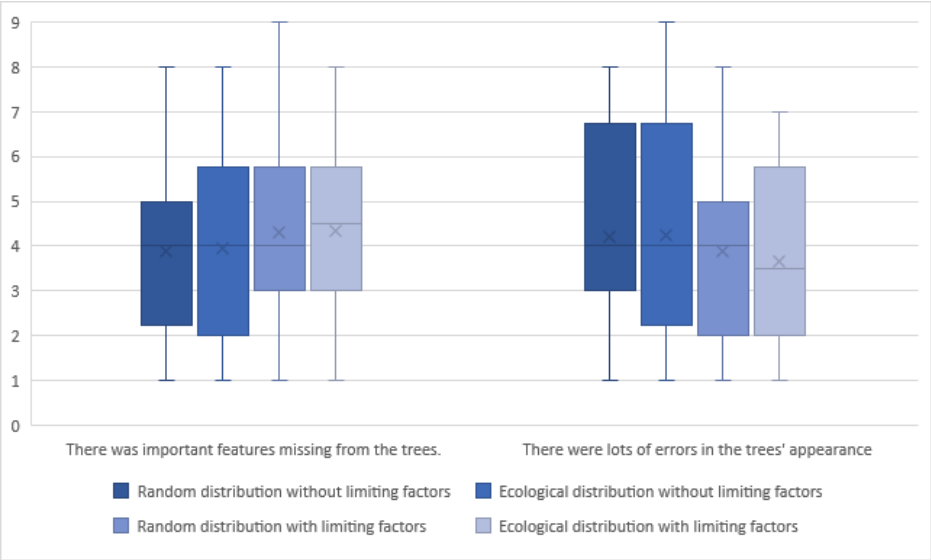


Figure 8.3: Results of authenticity questions - Lower is better

#### There was important features missing from the trees

A two-way repeated measures ANOVA was run to determine the effect of different distribution and growth factors in "there was important features missing from the trees". Analysis of the studentized residuals showed that there was normality, as assessed by the Shapiro-Wilk test of normality and no outliers, as assessed by no studentized residuals greater than ± 3 standard deviations. There was no statistically significant two-way interaction between distribution and growth factors, $F(1, 19) = 0.000$, $p = 1.0$.

The main effect of distribution showed no statistically significant difference in "there was important features missing from the trees" between conditions, $F(1, 19) = 0.083$, $p = 0.776$.

The main effect of growth factors showed no statistically significant difference in "there was important features missing from the trees" between conditions,$F(1, 19) = 2.075$, $p = 0.166$.

A Friedman's two-way analysis of variance by ranks test was run to determine if there were differences of different distribution and growth factors as in "there was important features missing from the trees". There wasn't statistically significant, $X^2(3) = 4.037$, $p = 0.258$.

#### There were lots of errors in the trees' appearance

A two-way repeated measures ANOVA was run to determine the effect of different distribution and growth factors as in "there were lots of errors in the trees' appearance".

Analysis of the studentized residuals showed that there was normality, as assessed by the Shapiro-Wilk test of normality and no outliers, as assessed by no studentized residuals greater than ± 3 standard deviations. There was no statistically significant two-way interaction between distribution and growth factors, $F(1, 19) = 0.160$, $p = 0.694$

The main effect of distribution showed no statistically significant difference in"there were lots of errors in the trees' appearance" between conditions, $F(1, 19) = 0.432$, $p = 0.519$.

The main effect of growth factors showed no statistically significant difference in"there were lots of errors in the trees' appearance" between conditions,$F(1, 19) = 1.605$, $p = 0.221$.

A Friedman's two-way analysis of variance by ranks test was run to determine if there were differences of different distribution and growth factors in "there were lots of errors in the trees' appearance". There wasn't statistically significant, $X^2(3) = 2.538$, $p = .468$.

### 8.6.3. Navigation

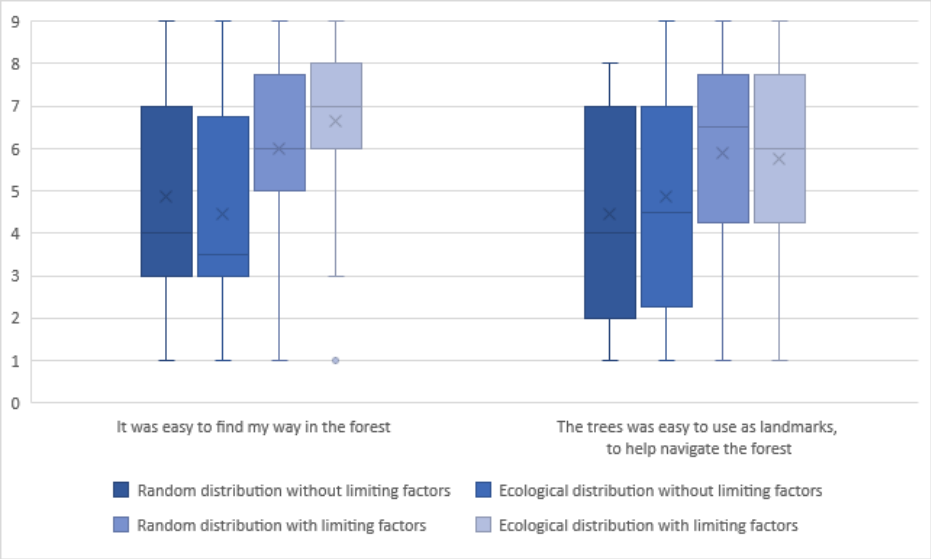


Figure 8.4: Results of navigation questions - Higher is better

#### It was easy to find my way in the forest

A two-way repeated measures ANOVA was run to determine the effect of different distribution and growth factors as in "It was easy to find my way in the forest". Analysis of the studentized residuals showed that there was normality except for ecological distribution without limiting factors ($p = 0.025$) and ecological distribution with limiting factors ($p = 0.045$), as assessed by the Shapiro-Wilk test of normality and no

outliers, as assessed by no studentized residuals greater than ± 3 standard deviations. There was no statistically significant two-way interaction between distribution and growth factors, F(1, 19) = 2.217, p = 0.153.

The main effect of distribution showed no statistically significant difference in "It was easy to find my way in the forest" between conditions, F(1, 19) = 0.115, p = 0.738.

The main effect of growth factors showed a statistically significant difference in "It was easy to find my way in the forest" between conditions, F(1, 19) = 12.165, p = 0.002. Post hoc tests using the Bonferroni correction revealed that limiting factors was the best growth factor (M = 6.325, SD = 1.981) compared to without limiting factors (M = 4.650, SD = 2.102)

A Friedman's two-way analysis of variance by ranks test was run to determine if there were differences of different distribution and growth factors as in "It was easy to find my way in the forest". There was a statistically significant, $X^2(3) = 10.571$, p = 0.014.

**The trees was easy to use as landmarks, to help navigate the forest**

A two-way repeated measures ANOVA was run to determine the effect of different distribution and growth factors as in "The trees was easy to use as landmarks, to help navigate the forest". Analysis of the studentized residuals showed that there was normality except for random distribution without limiting factors (p = 0.03) , as assessed by the Shapiro-Wilk test of normality and no outliers, as assessed by no studentized residuals greater than ± 3 standard deviations. There was no statistically significant two-way interaction between distribution and growth factors, F(1, 19) = 0.608, p = 0.445.

The main effect of distribution showed no statistically significant difference in "The trees was easy to use as landmarks, to help navigate the forest" between conditions, F(1, 19) = 0.095, p = 0.761.

The main effect of growth factors showed a statistically significant difference in "The trees was easy to use as landmarks, to help navigate the forest" between conditions, F(1, 19) = 7.814, p = 0.012. Post hoc tests using the Bonferroni correction revealed that limiting factors was the best growth factor (M = 5.825, SD = 2.182) compared to without limiting factors (M = 4.650, SD = 2.012)

A Friedman's two-way analysis of variance by ranks test was run to determine if there were differences of different distribution and growth factors as in "The trees was easy to use as landmarks, to help navigate the forest". There was a statistically significant, $X^2(3) = 9.000$, p = 0.029.

### 8.6.4. Best Condition

After all conditions the test subjects were asked for which condition they perceived to be most realistic. There is no clear favorite, but random distribution with limiting

factors was the least favorite with only 10% of the participants choosing it compared to 30% to each of the three other conditions.

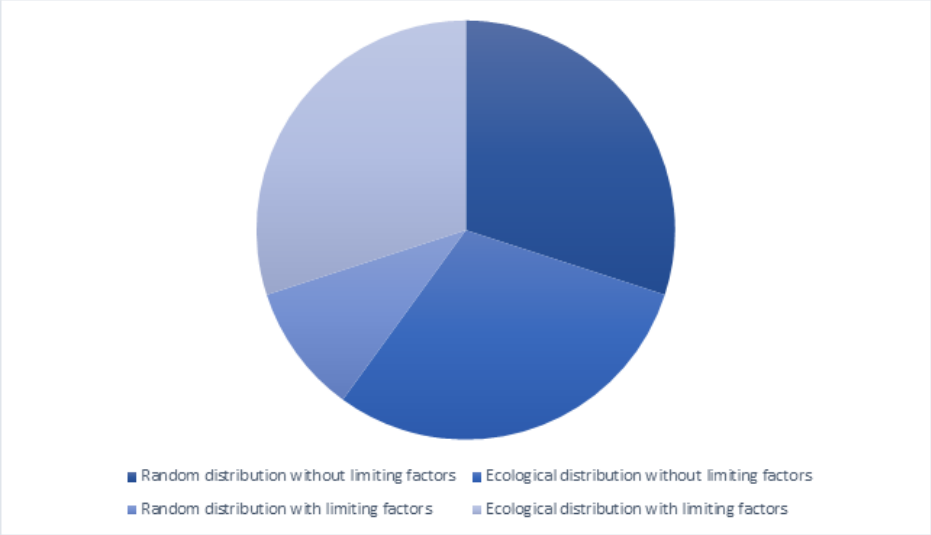


Figure 8.5: Best condition according to the test participants

### 8.6.5. Conclusion

Most of the questions did not show a statistically significant difference between means. In simulational realism it was only "The trees' shape look realistic" that showed a statistically significant difference between means, to reject the null hypothesis. There was no interaction between distribution and growth factors. But after an pairwise comparison the data did show that the perceived realism increase with the use of ecological distribution and limiting factors. While in navigation there was a statistically significant difference between means in growth factors for both questions "It was easy to find my way in the forest" and "The trees was easy to use as landmarks, to help navigate the forest".

It is interesting to see that the perceived realism is affected both for the distribution and growth factors, when it comes to the trees shape, while it's only the growth factors of the trees that have an affect when it comes to navigation in the game level. This will be discussed further in the next chapter (chapter 9).

| Results - Perceived Realism | |
|---|---|
| The trees looked realistic | |
| Distribution | F(1, 19) = 0.112, p = 0.742 |
| Growth factors | F(1, 19) = 1.986, p = 0.175 |
| Distribution * Growth factors | F(1, 19) = 1.993, p = 0.174 |
| The trees' shape look realistic | |
| Distribution | F(1, 19) = 7.908, p = 0.011 |
| Growth factors | F(1, 19) = 5.085, p = 0.036 |
| Distribution * Growth factors | F(1, 19) = 0.872, p = 0.362 |
| The trees have sufficient details, to appear realistic | |
| Distribution | F(1, 19) = 0.016, p = 0.900 |
| Growth factors | F(1, 19) = 0.472, p = 0.500 |
| Distribution * Growth factors | F(1, 19) = 0.157, p = 0.697 |
| The trees placement look realistic | |
| Distribution | F(1, 19) = 0.539, p = 0.472 |
| Growth factors | F(1, 19) = 0.275, p = 0.606 |
| Distribution * Growth factors | F(1, 19) = 5.516, p = 0.030 |
| The forest looked realistic | |
| Distribution | F(1, 19) = 0.012, p = 0.912 |
| Growth factors | F(1, 19) = 0.635, p = 0.435 |
| Distribution * Growth factors | F(1, 19) = 0.027, p = 0.871 |
| There was important features missing from the trees | |
| Distribution | F(1, 19) = 0.083, p = 0.776 |
| Growth factors | F(1, 19) = 2.075, p = 0.166 |
| Distribution * Growth factors | F(1, 19) = 0.000, p = 1.0 |
| There were lots of errors in the trees' appearance | |
| Distribution | F(1, 19) = 0.432, p = 0.519 |
| Growth factors | F(1, 19) = 1.605, p = 0.221 |
| Distribution * Growth factors | F(1, 19) = 0.160, p = 0.694 |
| It was easy to find my way in the forest | |
| Distribution | F(1, 19) = 0.115, p = 0.738 |
| Growth factors | F(1, 19) = 12.165, p = 0.002 |
| Distribution * Growth factors | F(1, 19) = 2.217, p = 0.153 |
| The trees was easy to use as landmarks, to help navigate the forest | |
| Distribution | F(1, 19) = 0.095, p = 0.761 |
| Growth factors | F(1, 19) = 7.814, p = 0.012 |
| Distribution * Growth factors | F(1, 19) = 0.608, p = 0.445 |

Table 8.3: Results - Perceived Realism, results highlighted in gray, means that there is a significant difference

## 8.7. Results - Player Performance

The players performance was measured in time and distance, and lower scores equals better player performance. The results will be analyzed using a 2 way repeated measures ANOVA, and descriptive statistics was conducted in Excel. In table 8.4 is the results summarized.

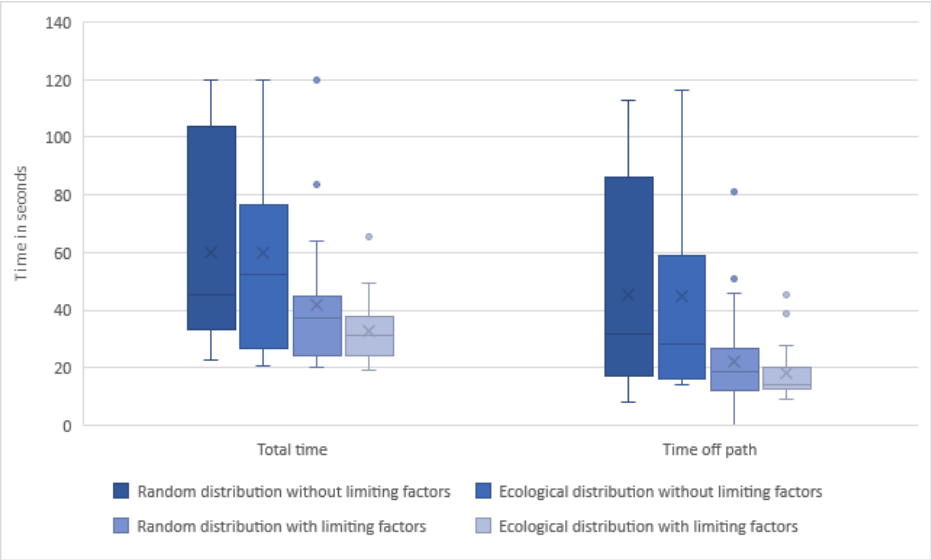


Figure 8.6: Time - Lower is better

### Total Time

A two-way repeated measures ANOVA was run to determine the effect of different distribution and growth factors as in total time to complete the task. Analysis of the studentized residuals showed that there was not normality, as assessed by the Shapiro-Wilk test of normality showed there was 2 outliers, which had a studentized residual values of 3.03 and 3.24. There was no statistically significant two-way interaction between distribution and growth factors, $F(1, 19) = 0.583$, $p = 0.454$.

The main effect of distribution showed no statistically significant difference in the total time between conditions, $F(1, 19) = 1.359$, $p = 0.258$.

The main effect of growth factors showed a statistically significant difference in the total time between conditions, $F(1, 19) = 12.315$, $p = 0.002$. Post hoc tests using the Bonferroni correction revealed that limiting factors was the best growth factor ($M = 37.348$, $SD = 15.398$) compared to without limiting factors ($M = 59.969$, $SD = 30.925$)

### Time off path

A two-way repeated measures ANOVA was run to determine the effect of different distribution and growth factors as in time players was off the path between conditions. Analysis of the studentized residuals showed that there was not normality, as assessed by the Shapiro-Wilk test of normality and there was one outlier, which had a studentized residual value of 3.07. There was no statistically significant two-way interaction between distribution and growth factors, $F(1, 19) = 0.122$, $p = 0.730$.

The main effect of distribution showed no statistically significant difference in time players was off the path between conditions, $F(1, 19) = 0.328$, $p = 0.573$.

The main effect of growth factors showed a statistically significant difference in time

players was off the path between conditions, F(1, 19) = 11.997, p = 0.003. Post hoc tests using the Bonferroni correction revealed that limiting factors was the best growth factor (M = 20.046, SD = 12.705) compared to without limiting factors (M = 45.032, SD = 31.010)

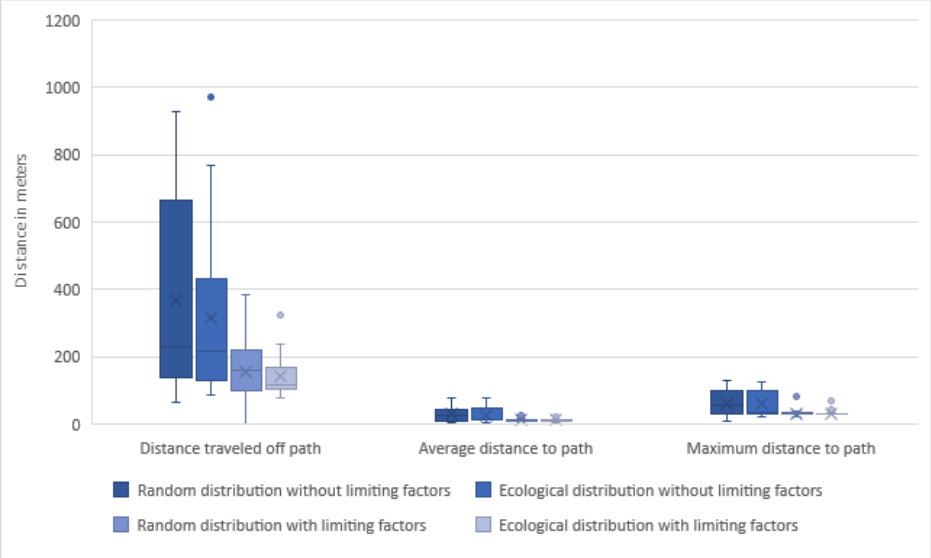


Figure 8.7: Distance - Lower is better

### Distance traveled off path

A two-way repeated measures ANOVA was run to determine the effect of different distribution and growth factors as in distance players traveled off path between conditions. Analysis of the studentized residuals showed that there was not normality, as assessed by the Shapiro-Wilk test of normality and there was one outlier, which had a studentized residual value of 3.16. There was no statistically significant two-way interaction between distribution and growth factors, F(1, 19) = 0.347, p = 0.563.

The main effect of distribution showed no statistically significant difference in distance players traveled off path between conditions, F(1, 19) = 0.990, p = 0.332.

The main effect of growth factors showed a statistically significant difference in distance players traveled off path between conditions, F(1, 19) = 11.434, p = 0.003. Post hoc tests using the Bonferroni correction revealed that limiting factors was the best growth factor (M = 148.981, SD = 68.607) compared to without limiting factors (M = 341.784, SD = 245.722)

### Average distance to path

A two-way repeated measures ANOVA was run to determine the effect of different distribution and growth factors as in the average distance players traveled off path between conditions. Analysis of the studentized residuals showed that there was not

normality, as assessed by the Shapiro-Wilk test of normality and there was two outliers, which had a studentized residual values of 3.23 and 3.24. There was no statistically significant two-way interaction between distribution and growth factors, $F(1, 19) = 0.137$, $p = 0.716$.

The main effect of distribution showed no statistically significant difference in the average distance players traveled off path between conditions, $F(1, 19) = 0.132$, $p = 0.720$.

The main effect of growth factors showed a statistically significant difference in the average distance players traveled off path between conditions, $F(1, 19) = 16.888$, $p = 0.001$. Post hoc tests using the Bonferroni correction revealed that limiting factors was the best growth factor ($M = 12.176$, $SD = 3.985$) compared to without limiting factors ($M = 27.663$, $SD = 17.361$)

### Max distance to path

A two-way repeated measures ANOVA was run to determine the effect of different distribution and growth factors as in the maximum distance players traveled off path between conditions. Analysis of the studentized residuals showed that there was not normality, as assessed by the Shapiro-Wilk test of normality and there was two outliers, which had a studentized residual values of 3.49 and 3.92. There was no statistically significant two-way interaction between distribution and growth factors, $F(1, 19) = 0.023$, $p = 0.881$.

The main effect of distribution showed no statistically significant difference in the maximum distance players traveled off path between conditions, $F(1, 19) = 0.091$, $p = 0.766$.

The main effect of growth factors showed a statistically significant difference in the maximum distance players traveled off path between conditions, $F(1, 19) = 18.414$, $p < 0.000$. Post hoc tests using the Bonferroni correction revealed that limiting factors was the best growth factor ($M = 31.832$, $SD = 9.512$) compared to without limiting factors ($M = 60.496$, $SD = 30.558$)

### Distance to goal

If the player did not make it to the end position before the time limiting then the distance to the end position was recorded. The best performance condition was ecological distribution with limiting factors as all players found the goal within the time limit. The second best was random distribution with limiting factors with one player not making it within the time limit (86.0m). Third best was random distribution without limiting factors with three players not making it within the time limit ($M = 50.4$m) Worst was ecological distribution without limiting factors with four players not making it within the time limit (184.8m)

### 8.7.1. Heat Maps

Heat maps was made to have a graphical representation of the participants paths seen from above. In figure 8.8 the four conditions heat maps can be seen. Each condition had 20 participants and the heat map is a combination of all 20 participants. The black paths are the paths, the participants walked with the signs and the red paths are the paths the participants walked when the signed were removed.

The heat maps confirms the data from the logging, that tree growth with limiting factors is the condition where the participants perform best. Ecological distribution without limiting factors performed poorly, both with and without signs. This might also have influenced the results.

### 8.7.2. Conclusion

The performance results confirms the results from the questionnaires that tree growth factors of the trees has an affect on the players ability to navigate the game environment. It is interesting that the distribution does not have a significant statistical difference, this will be discussed further in chapter 9.

| Results - Player Performance | |
|---|---|
| Total Time | |
| Distribution | $F(1, 19) = 1.359$, $p = 0.258$ |
| Growth factors | $F(1, 19) = 12.315$, $p = 0.002$ |
| Distribution * Growth factors | $F(1, 19) = 0.583$, $p = 0.454$ |
| Time off path | |
| Distribution | $F(1, 19) = 0.328$, $p = 0.573$ |
| Growth factors | $F(1, 19) = 11.997$, $p = 0.003$ |
| Distribution * Growth factors | $F(1, 19) = 0.122$, $p = 0.730$ |
| Distance traveled off path | |
| Distribution | $F(1, 19) = 0.990$, $p = 0.332$ |
| Growth factors | $F(1, 19) = 11.434$, $p = 0.003$ |
| Distribution * Growth factors | $F(1, 19) = 0.347$, $p = 0.563$ |
| Average distance to path | |
| Distribution | $F(1, 19) = 0.132$, $p = 0.720$ |
| Growth factors | $F(1, 19) = 16.888$, $p = 0.001$ |
| Distribution * Growth factors | $F(1, 19) = 0.137$, $p = 0.716$ |
| Max distance to path | |
| Distribution | $F(1, 19) = 0.091$, $p = 0.766$ |
| Growth factors | $F(1, 19) = 18.414$, $p < 0.000$ |
| Distribution * Growth factors | $F(1, 19) = 0.023$, $p = 0.881$ |

Table 8.4: Results - Player Performance, results highlighted in gray, means that there is a significant difference

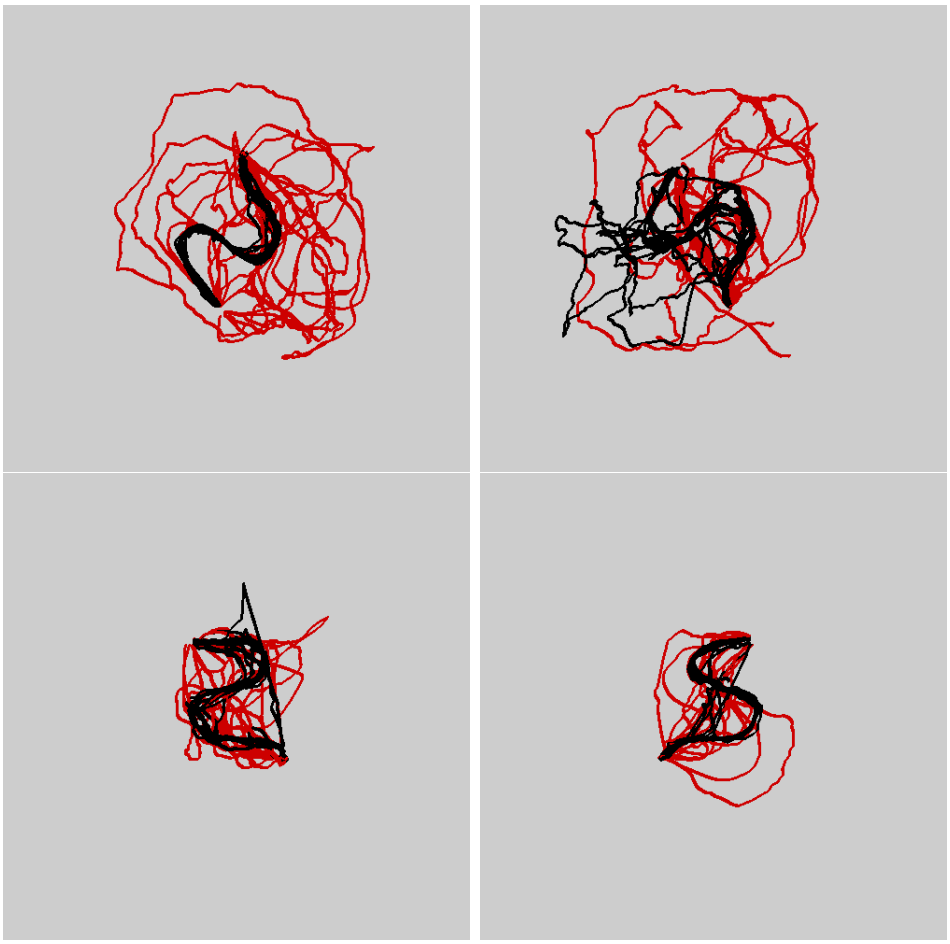

Figure 8.8: Heat Maps combined from 20 participants - (Top left) random distribution without limiting factors, (Top right) ecological distribution without limiting factors, (Bottom left) random distribution with limiting factors and (Bottom right) ecological distribution with limiting factors.

## 8.8. Results - Computer Performance

Performance in games is often measured in the frames per second that the game runs at. Having more frames per seconds means that the games runs more smooth.

### Frame rate

Descriptive statistics where conducted in Excel, and the best performing condition was ecological distribution without limiting factors (M = 55.9fps, SD = 1.2), second best was random distribution with limiting factors (M = 53.3fps, SD = 0.6), third best was ecological distribution with limiting factors (M = 51.9fps, SD = 0.4) and worst was random distribution without limiting factors (M = 47.7fps, SD = 0.3).

Performance spikes were also monitored, the results can be seen in figure 8.9, a high

minimum fps is more important than a high maximum fps.

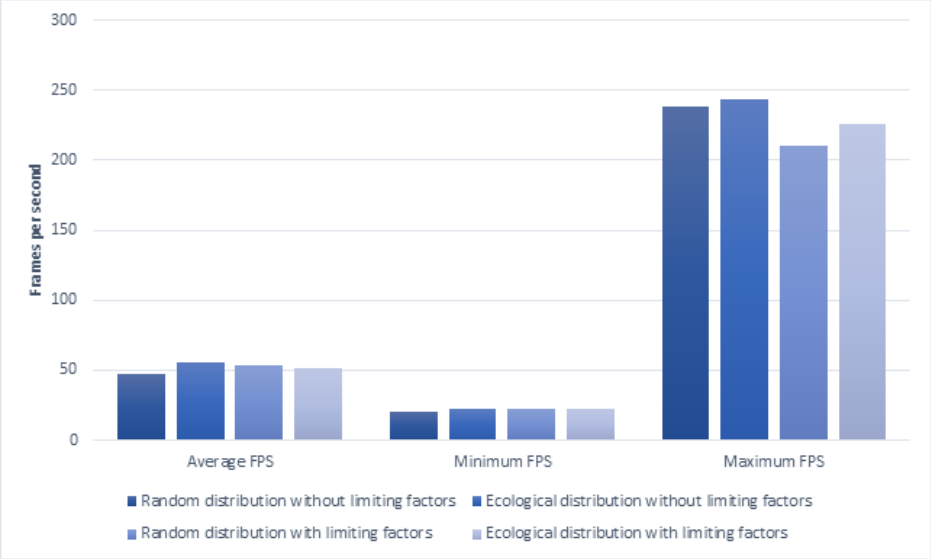


Figure 8.9: Frames per seconds - Higher is better

### Distribution and tree computation time

The computation time is importation because if using a more advanced algorithm is too slow, then it might slow the whole production progress to a halt. Computation time for distribution was faster for the random distribution (M = 1.2588sec, SD = 0.0144) and slightly slower for ecological distribution ( M = 1.2946sec, SD = 0.0468). Computation time for growth factor was faster for with limiting factors (M = 1.2307sec, SD = 0.217) and slightly slower for without limiting factors (M = 1.2664sec, SD = 0.0785).

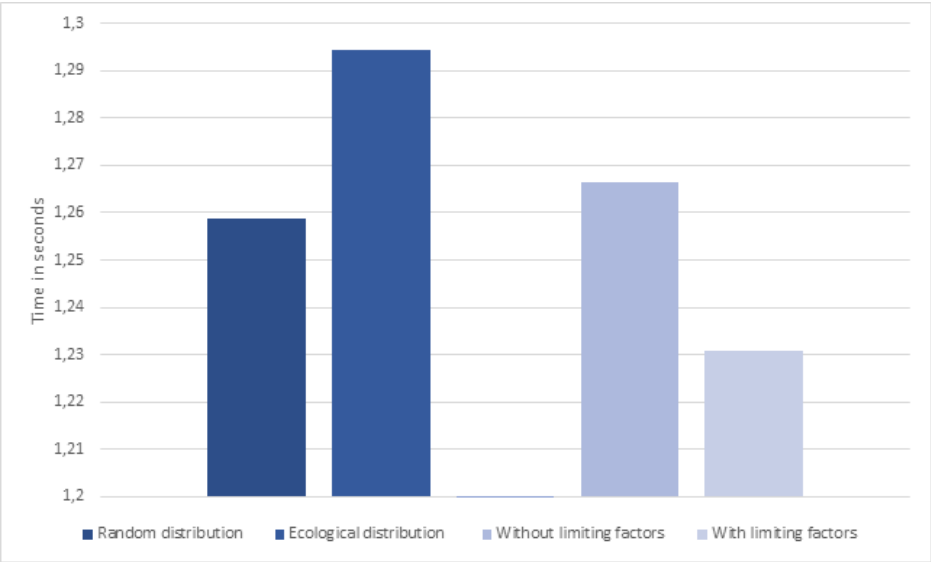


Figure 8.10: Generation time - Lower is better

### 8.8.1. Conclusion

It is interesting that it's slightly faster with limiting factors than without, as with limiting factors have more calculations to run, however it might be explained, by the fact that limiting factors also have more exit conditions that stops the recursive function.

# 9 Discussion

The results indicates that there is not significant difference in the perceived realism based on distribution and tree growth factors. There by the null hypotheses *"The perceived realism are not affected by distribution and by tree growth factors of procedurally generated trees."* cannot be rejected. There is an indication that players' perceived realism of the trees' shape are affected of both the distribution of the trees and by the trees' growth factors. There is also an indication that players' ability to navigate in a forest are affected by the growth factors.

With the large amount of data which is presented in the evaluation chapter, it is needed to categorize it to better discusses and understand the results. The data will be discussed in the following categories: distribution, tree growth, navigation and finally the test's reliability and validity. The questions that are used for each category can be seen in section 5.3 in chapter 5.

## 9.1. Distribution

It is interesting that the questions which are related to the distribution of the trees, show that there is no significant difference between random distribution and ecological distribution. One reason why there is not a difference might be that the participants have different opinions to what a realistic placement of the trees is and how that makes the wood look.

*"It (random distribution without limiting factors) felt the least 'designed'. Felt wild and random."* - Test participant 9.

*"I feel they looked pretty similar, but with Hearts (random distribution without limiting factors) and Clubs (random distribution with limiting factors), I was more aware of the*

*surroundings, and therefor noticed more flaws. In Spades (ecological distribution with limiting factors), I felt many of the trees were grouped by type."* - Test participant 7.

*"I felt the density of the forest was realistic and the amount of trees/branches was enough to force me to walk around rather than moving in straight lines (random distribution without limiting factors)"* - Test participant 17.

So for some of the participants it was more natural that the trees were grouped together while other, expected the trees to be totally random distributed.

It is however also interesting that the distribution had an influence on how realistic a tree's shape was perceived. While there was not any interaction between the distribution and growth factors of the trees, then the results did indicate that ecological distribution makes the shape of a tree appear to be more realistic.

## 9.2. Tree Growth

The results indicated that trees which are generated with limiting factors are perceived as more realistic than trees which are not. It was the opinion from more participants that the trees' shape was more realistic because the branching and branch directions looked more natural with limiting growth factors.

*"Tree were more proportional and further apart (ecological distribution with limiting factors)"* - Test participant 11

*"The trees seems to be more normal.(ecological distribution with limiting factors)"* - Test participant 14

*"It felt like almost all the trees reached the ground with the branched in the first two (without limiting factors) and they didn't in the last two (with limiting factors). The last two seemed very similar, so it could be both of them that were most realistic."* - Test participant 13

As the trees are generated with the same variables for the different tree species, then the only difference in there shape comes from the applied limiting factors. The participants didn't find the trees overall to be more realistic. It might have to do with the lack of better materials and texture. Some participants felt that, the materials for the trees should have been more varied and they missed features like ambient occlusion, height maps, tessellation and in general greater details. But this lack of details was equal over all 4 conditions, which was confirmed in authenticity scores that showed no significant difference. So this might answer why limiting factors only affects the realism of the trees' shape, because the participants are excepting more from the trees' appearance than just the shape, when judging the overall realism. In future works proposal will be discussed to improve the overall realism of the trees to better measure the affects that limiting factors have on perceived realism.

Another reason why there is not a significant statistical difference in the perceived realism of the overall appearance of the tree, might be the task that the players was

given. It might have distracted them from observing differences in the tree, as stated by one participant.

*"I have a hard time memorizing the differences, because i spent the most time on trying to remember the path."* - Test participant 6.

## 9.3. Navigation

The navigation task that was given to the participants gave some interesting results. It showed that there was a significant difference in how easy the participants found it to navigate and use landmarks in the test between with and without limiting growth factors. The results indicate that a forest with trees generated with limiting factors is easier to navigate compared to trees generated without. The participants responses did not indicate that there was any significant difference based on the distribution of the trees. When comparing the participants performance data, it shows that the conditions with limiting factors performed better than the conditions without, while there was little difference in performance between distribution conditions.

A reason why limiting factors performed better might be, that there was greater lines of sight, because the trees didn't growth downwards in the same manner as the more random look without limiting factors had. An other reason why with limiting factors performed better is also because the branches that are closer to the ground, and the colliders on these branches might have blocked the path for the participants and then they had to take a detour around a big tree. It could also be that with limiting factors produce better visual cues or landmarks that aids with navigation.

## 9.4. Test reliability and validity

The sample population was heavily over represented by males from Medialogy at Aalborg University Copenhagen. The results might not be representative of the general population, but more representative for persons interested in gaming and technology. This sample group also has a greater knowledge of the technology used in the test, and they might except more than the average video game player.

The test procedure did have an influence on the results. Participants stated after the test that they got trained over the test to be aware of details, in the subsequently condition, while they did not know what to look for in the first condition. Some participants expected there to be some improving progress as they moved on to the the next condition. This confused some participants. That is why the Latin square was used to balance out the results.

There might also be a small unbalance in the difficulty of the condition which can be seen in the difficulty that participants had in just finding the end position with the sign in the ecological distribution without limiting factors condition, as seen in figure 8.8.

# 10 Conclusion

In this project different techniques within vegetation in video games was investigated, and with the increasing demand for a large amount of asserts in games, it found was that game developers are looking into faster methods of producing asserts. One popular technique is the use of procedurally generated content. Different researchers have worked on developing procedural algorithms to generate vegetation for movies and games. Through the investigation it was found that fidelity of the vegetation has an great impact on the player's perception of realism. As procedural generated content has become more popular in game development, the focus of this project was to investigate: *"To what extent is perceived realism affected by distribution and by tree growth factors of procedurally generated trees?"*.

Different state of the art techniques ware investigated and from that was two types of distribution and two types of growth factors implemented. The experiment was conducted as a 2x2 within subject, that tested random distribution against ecological distribution and trees which was generated based on limiting factors and trees which was not.

The test indicated that there was not a significant difference in the perceived realism based on distribution and growth factors. However there it was found that the realism of the tree's shape is affected by both the distribution and the growth factors used to generate a forest for a game environment.

To answer the final problem statement *"To what extent is perceived realism affected by distribution and by tree growth factors of procedurally generated trees?"*, there is not a significant difference in perceived realism based on distribution and tree growth factors.

Players ability to navigate different test environments was measured and it was found that the conditions which used limiting growth factors was the ones the players performed best in. This results was confirmed both by the answers given by the players

and by the performance of the players. Tree distribution was not found to have any statistically affect on the players ability.

The computation performance of the different techniques were measured and it was found that ecological distribution was slightly slower than using random distribution. But it has only to be generated once and with the better perceived easier navigation it is a good compromise. Interestingly the more advanced algorithm that used limiting factors slightly was faster than the algorithm without limiting factors, even though it has more calculations and logical operations. This was explained by it had more exit conditions which stopped the recursive function.

## 10.1. Future Works

This project focused on creating a controlled test that showed the affects of distribution and tree growth factors, and that focus have limited the time which was used on fidelity of the materials and textures of the trees. The results showed that realism score might be affected by fidelity of the appearance of the textures and the lack of props in the forest. Technically there is more features that are needed to raise the level of fidelity to the level of a experienced CG artist. Especially this procedural algorithm could benefit greatly from tools that allows procedural textures, procedural skinning and rigging for animation, and a better level of detail system.

There is also the option of making a controlled test of only the trees perceived realism, by removing the independent variable of distribution. A test could show participants images of trees with and without limiting factors and ask them to rate them on a scale.

Distribution could be further tested with stacked or combined perlin noise layers, where e.g. every layer affecting only one or a few parameters related with tree growing. This might reveal different results, because using only a single octave perlin can be perceived as random, but there is potential for further studies.

If further testing of perceived realism of distribution and tree growth factors should be tested in a game environment it is needed to incorporate more audio/visual elements to rise the overall level of realism in a game environment. Some participants have noted that the game environment lacked life. Maybe a future study could involve the help of experienced level designers who uses the four conditions to create more realistic environments. Such test could also incorporate feedback on the usability of procedural generated trees based on limiting factors.

# 11

# References


Autodesk (2018). *What is generative design?* Autodesk. [Website].

Berlyn, G. P., Weber, L. M., and Everett, T. H. (2018). *Tree.* Encyclopedia Britannica.

Biodrowski, S. (2010). The dinosaurs of 1960: A 50th anniversary photo retrospective. `http://cinefantastiqueonline.com/2010/07/the-dinosaurs-of-1960-a-50th-anniversary-photo-retrospective/`.

Brown, M. (2016). *How (and Why) Spelunky Makes its Own Levels.* Game Maker's Toolkit. [Youtube].

Bruin, J. (2018). What is the difference between categorical, ordinal and interval variables? @ONLINE.

Chalmers, A. and Ferko, A. (2008). Levels of realism: from virtual reality to real virtuality. In *Proceedings of the 24th Spring Conference on computer graphics*, SCCG '08, pages 19–25. ACM.

Claypool, K. and Claypool, M. (2007). On frame rate and player performance in first person shooter games. *Multimedia Systems*, 13(1):3–17.

Diemer, B. (2017). *Star Wars Battlefront II.* DICE EA. Electronic Arts. [Game].

Doyle, P. and Gini, M. (2002). Believability through context using "knowledge in the world" to create intelligent characters. pages 342–349.

Druckmann, N. and Straley, B. (2013). *The Last of Us.* Naughty Dog. Sony Computer Entertainment. [Game].

Druckmann, N. and Straley, B. (2016). *Uncharted 4: A Thief's Ends.* Naughty Dog. Sony Computer Entertainment. [Game].

Fan, Z., Li, H., Hillesland, K., and Sheng, B. (2015). Simulation and rendering for millions of grass blades. In *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games*, i3D '15, pages 55–60, New York, NY, USA. ACM.

Fernando, R. (2004). *GPU Gems : programming techniques, tips, and tricks for real-time graphics*. Addison wesley, Upper Saddle River, N.J.

Field, A. and Hole, G. (2003). How to design and report experiments. pages 84–86.

Floyd, D. (2017). *What Happened to Mass Effect Andromeda's Animation? - Extra Frames*. Extra Credits. [Youtube].

Foreman, J. and Floyd, D. (2015). *Tomb Raider - Level Art Design - Guest Play with Josh Foreman*. Extra Play. [Youtube].

Gerling, K. M., Birk, M., Mandryk, R. L., and Doucette, A. (2013). The effects of graphical fidelity on player experience. In *Proceedings of International Conference on Making Sense of Converging Media*, AcademicMindTrek '13, pages 229:229–229:236, New York, NY, USA. ACM.

Ingvarsdottir, S. (2015). *Star Wars Battlefront*. DICE EA. Electronic Arts. [Game].

Jahrmann, K. and Wimmer, M. (2017). Responsive real-time grass rendering for general 3d scenes. In *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '17, pages 6:1–6:10, New York, NY, USA. ACM.

Jiang, Y. (2016). Character shading of uncharted 4. page 1.

Jillette, P., Teller, and Price, S. (2009). *Penn & Teller: Bullshit! - Video Games*. Showtime. [TV Show].

Kidwell, E. (2017). Environmental artist jane ng only made 23 unique trees for firewatch. *Gamasutra Article*.

Knowles, B. and Fryazinov, O. (2015). *Increasing realism of animated grass in real-time game environments*. SIGGRAPH '15. ACM.

Korn, O., Blatz, M., Rees, A., Schaal, J., Schwind, V., and Gorlich, D. (2017). Procedural content generation for game props? a study on the effects on user experience. *Computers in Entertainment (CIE)*, 15(2):1–15.

Lee, R.-R., Lo, Y., Chu, H.-K., and Chang, C.-F. (2016). A simulation on grass swaying with dynamic wind force. *The Visual Computer*, 32(6):891–900.

Lengyel, E. (2012). *Mathematics for 3D game programming and computer graphics, third edition*. Course Technology PTR, Boston, Mass., 3rd ed edition.

L'Heureux, J. (2016). The art of destruction in rainbow six: Siege. *GDC 2016*.

Lukosch, H., van Ruijven, T., and Verbraeck, A. (2012). The participatory design of a simulation training game. pages 1–11. IEEE.

Maximov, A. (2016). *Uncharted 4's Technical Art Culture*. Naughty Dog. [Youtube].

McMahan (2011). Exploring the effects of higher-fidelity display and interaction for virtual reality games.

Moss, O. and Vanaman, S. (2016). *Firewatch*. Campo Santo. Campo Santo and Panic. [Game].

Müller, M., Chentanez, N., and Kim, T.-Y. (2013). Real time dynamic fracture with volumetric approximate convex decompositions. *ACM transactions on graphics*, 32:1–10.

Murray, S., Duncan, G., Doyle, R., and Ream, D. (2016). *No Man's Sky*. Hello Games. Hello Games. [Game].

Nilsson, N. C., Nordahl, R., and Serafin, S. (2017). Waiting for the ultimate display: Can decreased fidelity positively influence perceived realism?

Noghani, J., Liarokapis, F., and Anderson, E. F. (2010). Randomly generated 3d environments for serious games. pages 3–10. IEEE Publishing.

Onrust, B., Bidarra, R., Rooseboom, R., and van de Koppel, J. (2017). Ecologically sound procedural generation of natural environments. *International Journal of Computer Games Technology*, 2017.

Owens, B. (2013). *Forward Rendering vs. Deferred Rendering*. Tutsplus Gamedevelopment.

Pirk, S., Stava, O., Kratt, J., Said, M., Neubert, B., MÄ›ch, R., Benes, B., and Deussen, O. (2012). Plastic trees: interactive self-adapting botanical tree models. *ACM Transactions on Graphics (TOG)*, 31(4):1–10.

Portnow, J. and Floyd, D. (2015). *Procedural Generation - How Games Create Infinite Worlds - Extra Credits*. Extra Credits. [Youtube].

Prince, S. (1996). True lies: Perceptual realism, digital images, and film theory. *Film Quarterly*, 49(3):27–37.

Ribbens, W., Malliet, S., Van Eck, R., and Larkin, D. (2016). Perceived realism in shooting games: Towards scale validation. 64:308–318.

Risser, E. (2006). *True imposters*. SIGGRAPH '06. ACM.

Schwartz, L. (2006). Fantasy, realism, and the other in recent video games. *Space and culture*, 9:313–325.

Shah, R., Beppu, B., and Huxley, J. (2017). *Uncharted 4 Environment Art: Part One*. Naughty Dog. [Youtube].

Shi, Y. (2016). Procedural content generation for computer games.

Smelik, R., Tutenel, T., de Kraker, K., and Bidarra, R. (2011). A declarative approach to procedural modeling of virtual worlds. *Computers & Graphics*, 35(2):352–363.

Sowers, B. (2008). Increasing the performance and realism of procedurally generated buildings.

Spielberg, S. (1993). *Jurassic Park*. Universal Pictures. [Movie].

Tarr, R. W., Morris, C. S., Singer, M. J., and Knerr, B. (2002). Low-cost pc gaming and simulation research: Doctrinal survey. Technical report.

Togelius, J., Shaker, N., and Dormans, J. (2016). *Grammars and L-systems with applications to vegetation and levels*, pages 73–98. Springer International Publishing, Cham.

Unity3D (2018a). *GPU instancing*. Unity Technologies. [Website].

Unity3D (2018b). *Level of Detail (LOD)*. Unity Technologies. [Website].

Valve (2018). *Steam Hardware & Software Survey: January 2018*. Valve. [Website].

Vinson, N. G. (2003). Design guidelines for landmarks to support navigation in virtual environments.

Wigley, C. J. (2013). Dispelling three myths about likert scales in communication trait research. *Communication Research Reports*, 30(4):366–372.

Wilcox-Netepczuk, D. (2013). Immersion and realism in video games - the confused moniker of video game engrossment. *Proceedings of CGAMES'2013 USA*, pages 92–95.

Yang, X., Yip, M., and Xu, X. (2009). Visual effects in computer games. *IEEE Computer Society*, 42:48–56.

Yu, D. (2016). *Spelunky by Derek Yu*. Boss Fight Books.

# 12

# Figure References

| | |
|---|---|
| Figure 1.1 | Diemer (2017) |
| Figure 1.2 | Druckmann and Straley (2016) |
| Table 3.1 | McMahan (2011) |
| Table 3.2 | Own creation |
| Table 3.3 | Own creation |
| Table 3.4 | Own creation |
| Table 3.5 | Vinson (2003) |
| Table 3.6 | Own creation |
| Table 3.7 | Togelius et al. (2016) |
| Figure 3.1 | `https://www.gamingscan.com/what-to-look-for-in-a-gaming-monitor/` |
| Figure 3.2 | Uncharted 4 |
| Figure 3.3 | Steel Beasts Pro |
| Figure 3.4 | Own creation |
| Figure 3.5 | `https://en.wikipedia.org/wiki/Tyrannosaurus` and Spielberg (1993) |
| Figure 3.6 | Own creation |

| | |
|---|---|
| Figure 3.7 | https://developer.nvidia.com/gpugems/GPUGems/gpugems_ch07.html |
| Figure 3.8 | https://developer.nvidia.com/gpugems/GPUGems/gpugems_ch07.html |
| Figure 3.9 | https://www.youtube.com/watch?v=CncA3o8f0P0 |
| Figure 3.10 | Unity3D (2018b) |
| Figure 3.11 | http://mwituni.com/2010/08/ |
| Figure 3.12 | https://developer.nvidia.com/gpugems/GPUGems/gpugems_ch07.html |
| Figure 3.13 | Own creation |
| Figure 3.13 | Pirk et al. (2012) |
| Figure 3.15 | Uncharted 4 |
| Figure 3.16 | Uncharted 4 |
| Figure 3.17 | Uncharted 4 |
| Figure 3.18 | The Legend of Zelda: Breath of the Wild |
| Figure 3.19 | The Legend of Zelda: Breath of the Wild |
| Figure 3.20 | Sid Meier's Civilization V |
| Figure 3.21 | Uncharted 4 and Mass Effect Andromeda |
| Figure 3.22 | Spelunky |
| Figure 3.23 | Brown (2016) |
| Figure 3.24 | https://www.thisoldhouse.com/ideas/all-about-shade-trees |
| Figure 3.25 | https://gamedevelopment.tutsplus.com/articles/forward-rendering-vs-deferred-rendering--gamedev-12342 |
| Figure 3.26 | https://gamedevelopment.tutsplus.com/articles/forward-rendering-vs-deferred-rendering--gamedev-12342 |
| Figure 3.27 | https://gamedevelopment.tutsplus.com/articles/forward-rendering-vs-deferred-rendering--gamedev-12342 |
| Figure 3.28 | https://unity3d.com/learn/tutorials/topics/graphics/choosing-color-space |

Figure 3.29 https://www.marmoset.co/posts/basic-theory-of-physically-based-rendering/

Figure 3.30 https://www.marmoset.co/posts/basic-theory-of-physically-based-rendering/

Figure 3.31 https://en.wikipedia.org/wiki/Fresnel_equations

Figure 3.32 https://www.marmoset.co/posts/basic-theory-of-physically-based-rendering/

Figure 3.33 Korn et al. (2017)

Figure 3.34 https://en.wikipedia.org/wiki/White_noise

Figure 3.35 Onrust et al. (2017)

Figure 3.36 Onrust et al. (2017)

Figure 3.37 Jahrmann and Wimmer (2017)

Figure 3.38 Jahrmann and Wimmer (2017)

Figure 3.39 Fan et al. (2015)

Figure 3.40 Pirk et al. (2012)

Figure 3.41 Pirk et al. (2012)

Figure 3.42 Togelius et al. (2016)

Figure 3.43 Togelius et al. (2016)

Figure 3.44 https://en.wikipedia.org/wiki/Generative_design

Table 5.1 Own creation

Figure 6.1 Google maps and http://www.willhiteweb.com/british_columbia/bridal_veil_falls/woodland_trail_sign.jpg

Figure 6.2 http://www.extremerunner.dk/rude-skov-og-loeb

Figure 6.3 https://www.deviantart.com/tag/leaf and https://www.muralunique.com/path-in-a-beech-forest-10-5-x-8-3-20m-x-2-44m.html

Figure 6.4 Own creation

Figure 6.5 http://nkblog.nkdev.de/world-machine/

# A

# Appendix

## A.1. Content on Digital Appendix

| | |
|---|---|
| **Code:** | ProceduralTreeGen |
| | MapGenerator |
| | GM |
| | HeatMap |
| | TestData |
| | FPS |
| | Leaf shader |
| **Processing of results:** | SPSS |
| | Excel |
| | Log files |
| | Heat Maps |
| **Video:** | Project video |