Faculty of Engineering Science and Technology
Department of Computer Science and Computational Engineering

Adaptable and interactive 3D modelling using genetic algorithms

Håkon Berg Borhaug

DTE-3900 Master’s thesis in Applied Computer Science May 2024

Abstract

This thesis discusses the process of designing, implementing and testing a creative AI system that generates 3D models for the purpose of video game development. The system utilizes a combination of an interactive genetic algorithm and a distance weighted K-nearest neighbour regressor. Regression is employed as a learning mechanism to enable the system to rate designs based on the overall user design preferences absorbed through user interaction. The results from testing the implemented sytem are discussed, and possible changes and improvements to the system based on the produced results are also covered.

Acknowledgements

I would like to thank my supervisors, Andreas Dyrøy Jansson and Bernt Arild Bremdal, for providing the baseline for the project and for their guidance during the development of the project.

I would also like to thank my family for their support, with a special mention to my brother Hans Berg Borhaug for helping me getting through a tough period during my bachelor studies.

Lastly, I would like to thank all my classmates for all of their support and feedback.

Contents

List of Figures

List of Tables

1 Introduction

This chapter will discuss the background of the thesis, the objectives, and an overview of the project, as well as some of its underlying theory. Similar work and technology that try to tackle some of the same challenges encountered in this project will be presented.

1.1 Background

Video games have become increasingly popular in the modern day. Through fun and exciting gameplay, engaging and dramatic storytelling, and online multiplayer letting players connect and interact through collaboration or competition, video games as a whole have been able to reach a huge and diverse audience across the world. This has resulted in the video game industry becoming a huge section of the global entertainment and media sector, with total revenues projected to rise from $262 billion in 2023 to $312 billion in 2027 [1].

However, video game development is getting increasingly more time consuming and consequently more expensive for developers. Development budgets for blockbuster games has reached up in the $100s of millions [2], where a major part of this development cost is attributed to game asset creation. During the production stage of development, a significant amount of time is used for prototyping and iterating over different concepts and builds [3]. Today, the majority of top-selling games are in 3D. Consequently, many major developers find themselves needing to create new game assets, including 3D models, during development. Accelerating this process, as well as possibly improving the quality and variety of the assets created, would lead to a more cost-effective, superior and faster developed product. Through the use of artifical intelligence (AI) tools one could contribute in a major way to t ackle this issue by helping developers iterate and conceptualize ideas quicker and more efficiently.

In the last few years several new AI tools have been released that are able to generate text [4], pictures [5] and even whole 3D models [6]. These are defined as generative AI systems due to their ability to generate novel content based on learned patterns using techniques such as neural networks, autoencoders and transformers. Without going into detail on how these techniques work, it is worth noting that the systems that are using them are described as black-box systems. A black-box system can be viewed in terms of its inputs and outputs only, with the internal logic and processing being uninterpretable, with a white-box model being the opposite [7]. Specifically in the context of AI, the parameters of a black-box model can not help us understand the inner mechanisms that dictate why specific results are generated. Consequently, the possible content generated by these generative systems is not easily controlled or restricted, which could lead to issues such as offensive or dangerous content being generated [8]. Another issue these types of AI systems encounter is the fact that most of them are trained on copyrighted material, which has led to several lawsuits from different rights holders [9].

Creative AI systems that rely on a white-box model could avoid these shortfalls of generative AI systems. Let us say that you create some multi-dimensional space where the axes are defined by the content design parameters (such as color, size, etc.), where a point in this space would correlate with a design using the axis values as parameters. This design space would end up containing every possible design iteration possible that can be desribed by the chosen set of parameters. Thus, by implementing an AI algorithm that can search this space, one would have made a creative AI system that can find designs that have possibly never been thought of before. Genetic algorithms (GA) are a good choice for this task, due to their flexibility in optimzing any problem space that can be encoded into a sequence of numbers, in contrast to the example of the generative model technique (gradient descent) where the function that maps the problem space has to be differentiable [10]. This enables being able to freely formulate the problem of finding these novel and interesting designs, and adapt the formulation depending on the context of which the system is being used. The underlying theory on genetic algorithms are covered in more detail in Subsection 1.3.2. Conclusively, this project will cover the creation of such a creative AI system using a genetic algorithm to explore a design space containing 3D models, with a specific interest in 3D models for the use in video game development.

1.2 Objective

As mentioned in Section 1.1, the main objective of this project is creating an AI tool that can help video game developers generate new 3D assets to act as inspiration during the ideation and prototyping phase in the production stage of development. In order to achieve this goal, this project will cover the implementation of a creative AI system that can create novel 3D model designs for the use in video game development, where the exploration of the design space will be performed by a genetic algorithm. The objective can be broken into a three main points, as paraphrased from the task description found in Appendix A:

1. Study representation issues, which include model features in 3D space as well as functional constraints, requirements and priorities. In addition, studying methods for evaluating and ranking tentative design solutions are also important.

2. Based on these investigations, develop a genetic algorithm concept that is able to create novel solutions that demonstrate creative capabilites through extensive recombination of many features. This includes the choice of genetic algorithm type, and decisions regarding operators and their use.

3. Define a set of criteria for evaluation and verification of the quality of the models generated by the system. These could be general in nature, specific for a particular user group or circumstantial. Tests should be designed accordingly and performed on a rich set of cases.

These points cover the specific goals of this masters's thesis project. Examining the viability and performance of the resulting creative AI system is the scientific task that will be covered in this thesis.

1.2.1 Scope

Due to the limited time available to develop the project for this master thesis, the scope of the objectives mentioned above will be refined into a more specific set of objectives.

The choice of model representation should result in maximizing the solution space to keep model variance high, as well as leading to relatively good model topology. In the context of this project, a model with well-formed topology correlates with the model avoiding overlapping faces and edges, holes in the mesh, unevenly placed polygons, and a model that shows sensible volume and depth. This last point means that the model is not overly flat or thin, but possesses a well-rounded and coherent structure. Figure 1.1 illustrates this concept in the context of a 3D model that resembles a human head, where the face section shows good topology features, while the rest of the model is completely incoherent and disorganized.

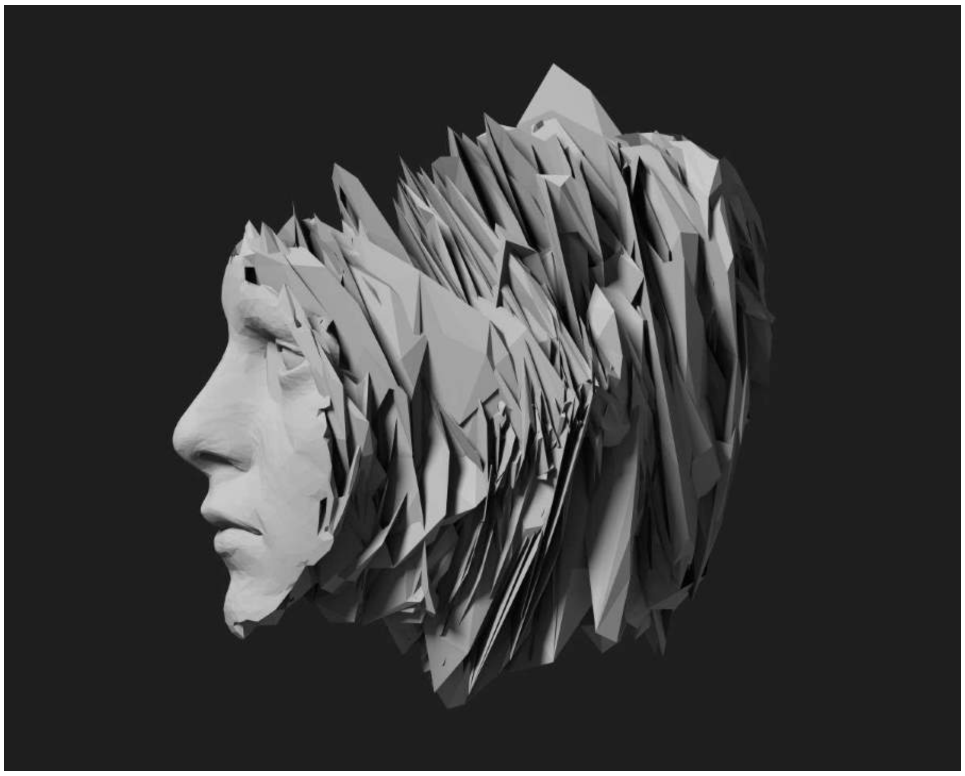


Figure 1.1: 3D model showing a mix of good and bad topology [11]

While aiming for high model variance, the choice was made to restrict the complexity of the models generated. As the main domain of interest is video game development, it made sense to restrict the type of models to ones that are not necessarily release-ready in the sense of intricacy and polish, but complex enough to be used as insipration or as a starting point for further manipulation into a finished model in a video game. Although this results in a limited set of possible designs, it should also limit the problem of model incoherence, as lower model complexity should avoid some pitfalls caused by the inbalance of too many model parameters. Figure 1.2 shows an example of the use case of a model that is less complex as a starting point, where you have a simple shape that resembles a head at around stage 4 of the shown process, which is iteratively shaped into a more complex and polished model.

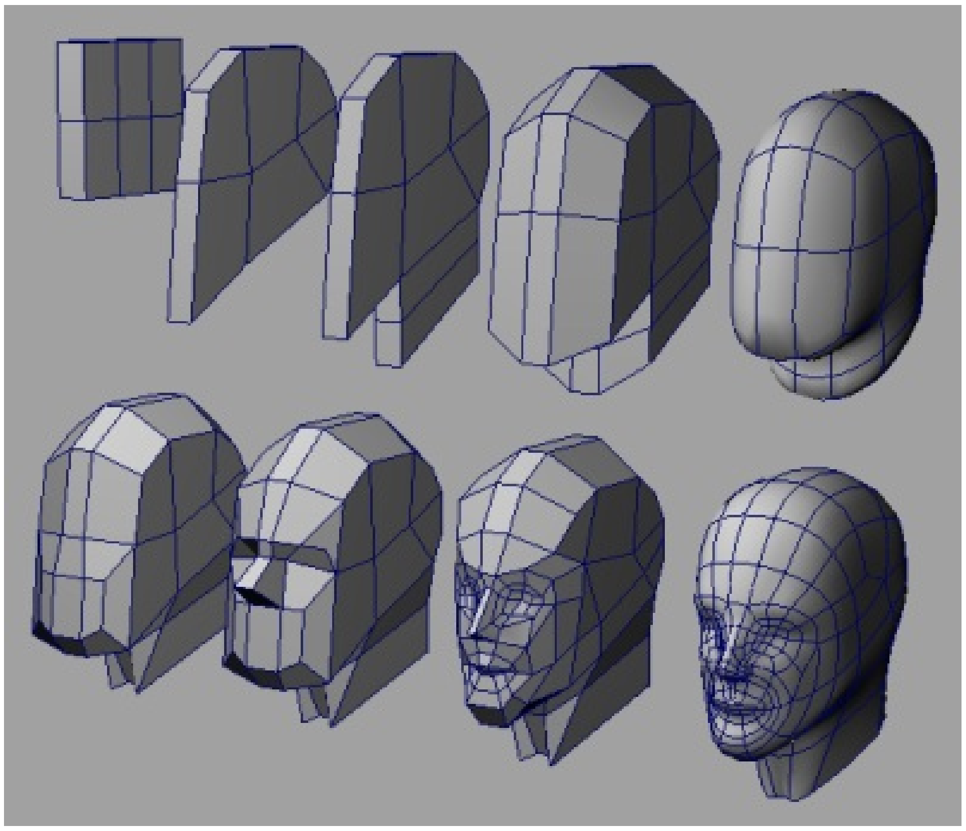


Figure 1.2: Gradual refinenemt of low complexity models [12]

The task of rating and evaluating designs is an inherently subjective problem, making it challenging to find an "objective" measure for 3D models. Thus, there will not be an attempt to cover this aspect in this thesis. The percieved quality of a design can vary based on individual differnces, such as background, culture, education, age and many other factors. Choosing a solution that can adapt to the individuals design preferences therefore seems the most sensible approach to model evaluation. Thus, this thesis will specifically look at the implementation of an interactive genetic algorithm, where the user of the system dictates the perceived quality and ranking of models. This approach aims to yield a system capable of producing model results that align with the specific design preferences expressed by the user.

Due to the author of this thesis not possessing the appropriate 3D-modelling expertise, determining the overall quality of the models that are generated using the system will be entrusted to those with more experience in the field. However, a series of tests and experiments will be conducted to ensure that the system is able to follow the design preferences provided through its interactive component and produce designs that correspond to these preferences. The tests will examine wether the system can absorb general model features (such as size, structure, and pattern) that align with the preffered designs during evaluation. Additionally, there will be tests to assess the system's ability to capture a specific type of shape or object (such as a tree, a sword, or a building) and produce variations of it with small differences to the underlying structure of the object.

1.3 Theory

This section will cover some of the underlying theory of the methods used in this project.

1.3.1 Machine learning

AI can be described as any technique that enables a computer to mimic human-like behaviour. Within the domain of AI machine learning has emerged as one of the main subfields, where computers acquire the capacity to learn desired behaviors without explicit programming, a concept originally coined by Arthur Samuel in 1959 [13]. In his paper he describes a machine learning method where a computer is programmed to learn to play checkers to a greater level than a human by being given the rules of the game, practicing against itself over and over, and learn from the outcomes of the practice games. Modern machine learning can be divided into four main categories [14]. Supervised machine learnining is the method of learning wherein the desired outcome is known and the training data is labeled. Here the machine learning model learns to map input data to the desired output data so that the model can learn to map new unlabeled data by using its training. Unsupervised machine learning is the opposite, where learning has to be achieved without any labels on both the input data and the expected output. In this case the machine learning model has to learn structures and patterns in the data on its own. The middle-ground between these techniques are called semi-supervised learning, where one combines elements of both supervised and unsupervised learning to overcome problems like a lack of labeled data. Lastly we have reinforcement learning, which involves an agent learning to interact with its environment and maximizing its collected rewards. This is performed through trial and error by exploring the environment through the different possible state-action pairs, where over time the agent learns the optimimal decision making strategies which yield the highest total rewards.

1.3.2 Genetic algorithm

The genetic algorithm (GA) is a type of machine learning algorithm that was introduced by John Holland in 1975 [15]. In his work he describes a type of optimization algorithm that is inspired by the natural selection process and its genetic principles. A population of proposed solutions to the given problem are encoded into gene sequences (chromosomes), where through a form of natural selection process one defines a fitness evaluation to determine which individual solution is fit enough to use its gene encoding to generate future generations. A selection process is made where the fittest potential solutions are chosen to create new "children". These children are made by performing gene crossover between the parents chosen, such that the desirable traits of the parents' proposed solution are passed on to the child. It is also common to perform gene mutations to explore more of the possible solution space of the problem domain, where singular genes in the gene sequence of the child are set randomly. Over time the generations of proposed solutions will converge towards a solution (or several) that maximizes the fitness function defined in the algorithm. In Figure 1.3 you can see a flow chart showing the basic steps in a genetic algorithm.

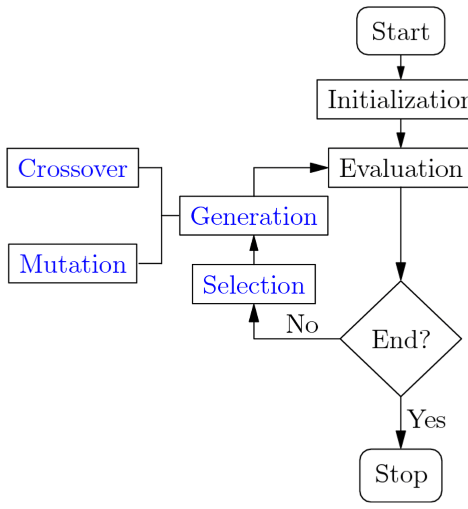


Figure 1.3: Genetic algorithm flow chart [16]

1.3.3 K-Nearest Neighbour

K-nearest neighbour (K-NN) is a supervised machine learning algorithm used for classification and regression. It was first introduced in a paper by Evelyn Fix and Joseph Hodges in 1951 [17]. The key idea behind the algorithm is to predict the label/value of a new entry to a dataset based on the k nearest neighbour point labels/values. To determine the closest neighbours one must specify the distance metric used, such as Euclidian distance. In the context of classification the class that occurs most commonly among the k number of neighbours with the lowest distance to the new entry is the classification predicted. For regression purposes we calculate the predicted value by averaging the values of the k nearest neighbours.

Choosing the value of k is crucial to the algorithms behaviour. A small k value leads to a more flexible model, where each prediction is heavily influenced by the closest neighbours. This flexibility however can lead the model to be overfitted and sensitive to noisy data. In contrast, a model with a high k value are less prone to noise and outliers, but may miss local patterns due to a overly smooth decision boundary.

An example of a K-NN algorithm used for classificaiton in practice can be seen in Figure 1.4. In this example the new data point is classified as red if $k = 3$, while if $k = 5$ the data point would be classified as blue instead, showing the sensitivity of the choice of the value of k.

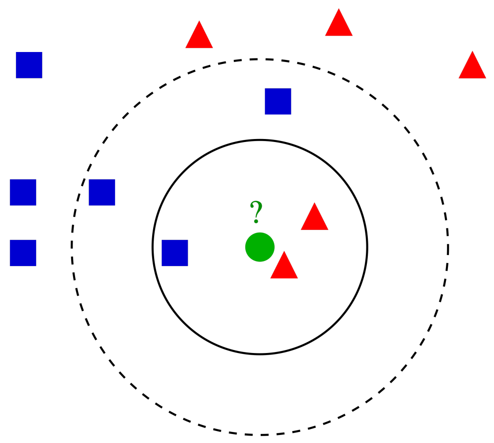


Figure 1.4: Example of classification using K-NN [18]

1.4 Related work

Creating an adaptive and interactive system that can generate 3D model designs using AI is a difficult problem consisting of several challenges that need to be solved, as described in Section 1.2. This section will cover some related works that have been produced that have attempted to tackle some (or all) of the problems that our proposed system need to solve.

In their paper, Ando et al. [19] propose a 3D character creation system using an interactive genetic algorithm and a kansei (sensibility) rule extraction method. In their system they represent their 3D character model by a set of pre-defined model parts that each represent a specific section of the character, which are encoded as a sequence of categorical attributes. The fitness of each character is calculated using the kansei rules, which are determined through interactive learning where the user inputs a kansei word they like, then the system creates several different characters using different model parts, where the user consequently evaluates each of these character designs. Using the new evaluations the system determines new kansei rules from the new data and generates a new set of characters for the user to evaluate.

Yoon et al. [20] released a paper in 2012 where they have developed an interactive system that uses a genetic algorithm to create 3D models and textures. To represent the footprint of the models they use an L-system, which generates a string of characters with Fs, Ls and Rs, which represent different drawing commands. Fs stand for drawing a single unit line in the current direction, Ls stand for rotating 90 degrees left and Rs stand for rotating 90 degrees right. The chromosome in their system includes the seed used for random number generation, the iteration number in the L-system production, and the height of the building. To evaluate the fitness of each individual in the population they have the user give each individual model a score to represent the fitness value.

A similar system was developed in 2017 by Arifinn et al. [21] where they have a system using an interactive genetic algorithm and an L-system. The strings produced in their L-system correspond with a sequence of commands used in the 3D modelling software Blender [22], such as creating an edge segment, creating a quad, rotating around a specific axis, and increasing or decrease the mesh radius of a branch. By letting the user rank the different models generated each generation in comparison to each other they are able to get an interactive fitness evalutation of each individual.

In a paper by researchers at NVIDIA, Unity/Weta Digital and ROBLOX [23] they created a dataset of 3D models using a technique called signed distance functions (SDFs). Using a combination of these SDFs to represent basic shapes and then combine in different ways using cosntructive solid geometry (CSG) operations (such as union and subtraction), they are able to create complex shapes with smooth transitions between the different base shapes. This way of creating 3D models could be used as a base for a model representation, in which you define some set of SDFs and CSG operations as categorical features, where the variables of the SDFs are defined as numerical features.

My supervisors for this thesis, Andreas Dyrøy Jansson and Bernt Arild Bremdal, released a paper in 2018 [24] where they describe a creative web element design system using an interactive genetic algorithm. The designs were represented using HTML properties as integer values, where simple properties as color and size were directy encoded, while more complex properties like font and border style were represented as indices. Running the system shows the user a set of designs for each generation, where each design can be rated by the user as liked or disliked, which in turn increases or decreases their fitness value. A K-NN classifier is used for evaluating the designs which are not rated by the user, where each design is assigned a classification match percent to the most common class among it's neighbours.

1.5 Strategy

The objective of this project is the examination of the viability of a creative AI system using a genetic algorithm that can create 3D model designs for the use in video game development, as described in Section 1.2. To implement such a system, one must solve the problems of representation of 3D model features, development of a genetic algorithm that is capable of creativity by feature combination, model quality evaluation, and finally extensive testing of the system results. This section will cover the strategies chosen to solve these issues.

1.5.1 Model representation and visualization

In Section 1.4 different methods for representing a 3D model using a set of numerical parameters were covered. The categorical representation using model parts shown by Ando et al. [19] would be a simple solution to this problem in terms of implementation complexity. However, using this method would lead to a very limited solution space due to the low amount of possible model permutations when using a realistic amount of model parts for this project. Moreover, this method leads to the problem of having to create these model parts as the base of the model generation system, which somewhat defeats the purpose of creating a creative AI system to generate 3D models.

Representing the models using an L-system as shown in the papers by Yoon et al. [20] and Arifinn et al. [21] avoids the need for a base model or model parts to generate new models. However, the representation does show its weakness in the lack of model complexity in the case of the former paper, where the resulting building models are quite simple and does not show much varitation of shape structure between individuals. For the latter paper, they are able to generate models with some complexity and variety, but the model topology and coherence seems non-satisfactory for the example models produced in the paper. In contrast, the concept of using SDFs in conjunction with CSG operations, as shown in [23], looks like a promising method for model representation in terms of model topology and variaton. Upon further investigation, using a similar method would be outside the scope of this project as implementing it effectively would require a significant portion of the allocated project time for research and development.

During the time used to research possible model representations for this project, a discovery was made of a representation used by Mark Kingsnorth in his Blender add-on called "Shape generator" [25]. This approach starts with the base shape of a cube and iteratively perform extrusions with different length, tapering and rotation from random faces of the current model. Experimenting with his tool in Blender demonstrated its ability to generate relatively complex and varied shapes, without encountering models with bad topoloy and with incoherent appearance due to the guard rails parameters included in the tool. Additionally, the generator parameters can be manipulated direclty through the Blender Python API. Thus, the decision was ultimately made to use his tool inside Blender as the framework for representing the 3D models as a set of parameters to be encoded into gene sequences for the genetic algorithm. Subsection 2.1.3 will cover details on how the "Shape generator" add-on works.

Rendering the meshes of the models generated in some manner would be a sensible way to visualize the generated models to the user of the system. Research was made into implementing a web application using the Javascript [26] 3D graphics library Three.js [27] that would enable interaction and rendering of the models that were generated in Blender by using some real time intermediate software link between the web-application and Blender. However, this proved to be a big and complicated task [28] and would be outside the scope of this thesis. The choice of model visualization went to Blender itself instead, where the user views and interacts with the models inside the Blender user interface (UI). The Blender UI can be seen in Figure 1.5, where the red section covers the 3D viewport, the green section covers the list of objects in the scene, and the yellow arrow shows the toolbar button that needs to be pressed to bring up the panel toolbar. Each object in the viewport has an "Object Context Menu" that can be accessed by right-clicking the object of interest, which brings up the menu as seen in Figure 1.6.

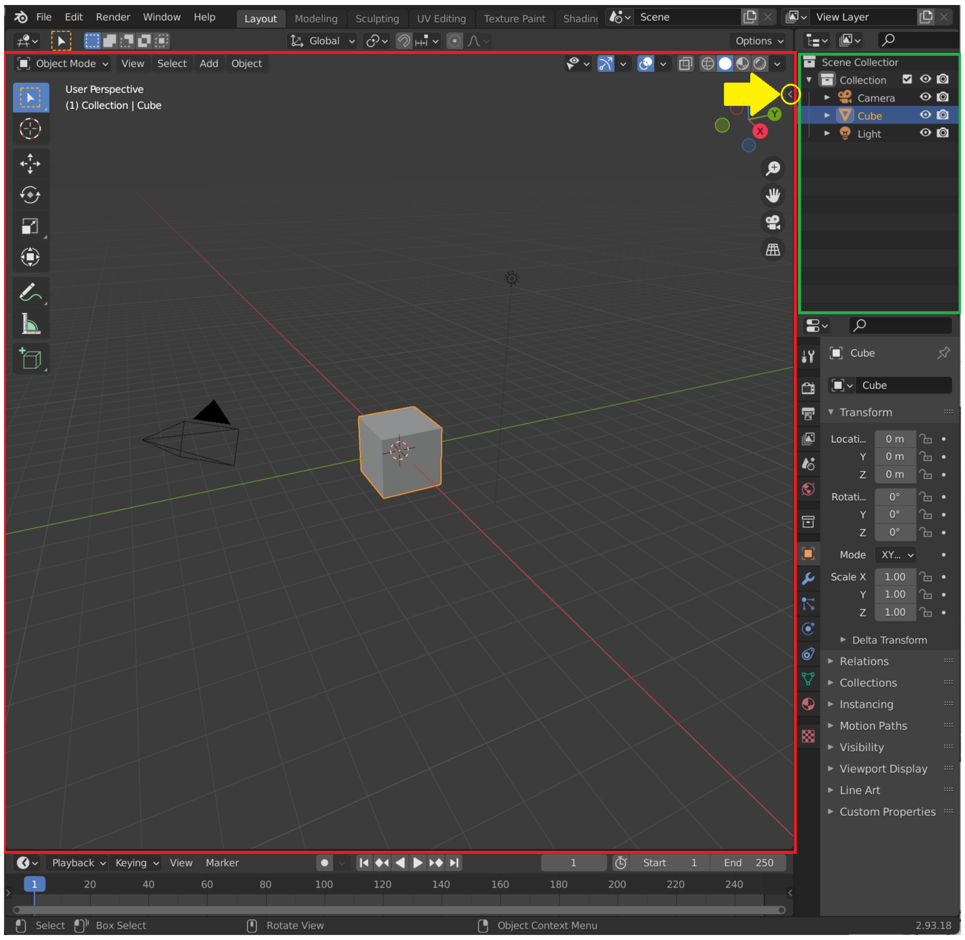


Figure 1.5: Blender UI with relevant sections highlighted

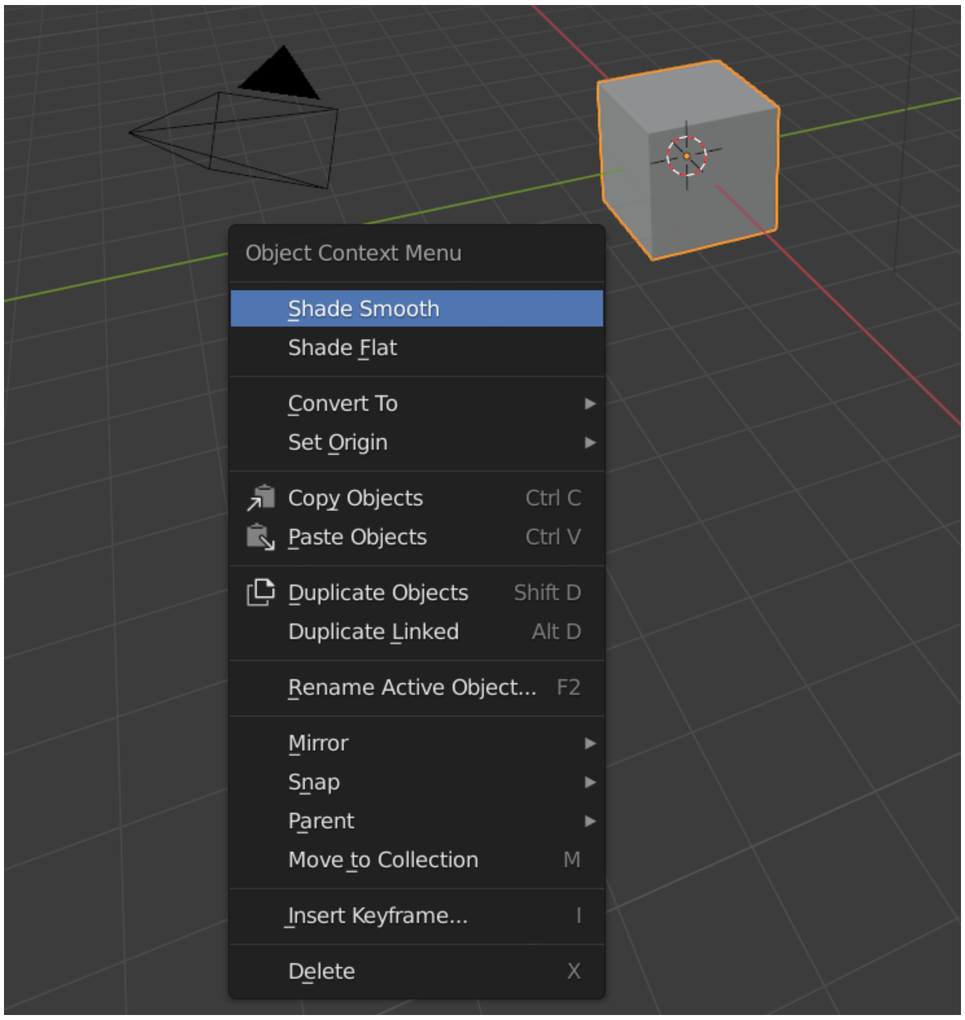


Figure 1.6: Object context menu in 3D viewport in Blender

1.5.2 Genetic algorithm

In this project emphasis will be put on using a genetic algorithm to implement a creative AI system that can generate 3D model designs. The algorithm should be able to generate generations of populations consisting of chromosomes containining 3D model representations as a sequence of parameters. Over each generation the algorithm should be able to attribute a fitness score to each individual design and proiritize the individuals with the highest fitness to produce the next generation of models through crossover between parent chromsomes and mutations of child genes. During its runtime the system should converge towards the model designs which the user of the system find appealing, but should also keep a relatively high gene diversity to achieve a high variance of model designs and to not limit the search volume of the solution space.

1.5.3 Fitness evaluation

Evaluating the fitness of a model design in terms of its appeal for the use in an interactive genetic algorithm is inherently a subjective task. The one thing the fitness evaluation does objectively need in an interactive system is the ability to let the user dictate which model designs are given high and low fitness values. Giving each model a numerical score as in [19] gives the user a significant amount of agency in terms of fine tuning the percieved quality of each model design, but it might be difficult to decide on the exact value to give to each design. Additionally, it could get even harder over time to keep the scores accurate in relation to each other as newer models should by design become more appealing than models that were given a high fitness score earlier in the system's runtime. Another way of attributing each model with a fitness value is by ranking the individuals within a generation in regards to eachother, as shown in [21]. The problem with this method is that the evaluation of each individual is anchored to all the other individuals in that generation, which could be a problem if one or more models are significantly better in terms of user sentiment than the rest, without the user being able to express the difference in quality properly. If a generation occurs with only "bad" model designs, the generation is essentialy wasted if the system has no way of penalizing these bad desings. The simplest way for the user to influence the design fitness is to let them directly pick the individuals they like and the ones they dislike, as shown in [24]. This ensures that both good and bad designs are affected positively and negatively accordingly, without having to give a specific number to describe to what degree they like or dislike the design.

To avoid having the user rate every single individual during a generation, Jansson and Bremdal [24] introduce a fitness function based on K-NN classification to evaluate the unrated designs within a generation. The choice of K-NN for this purpose can be seen as ideal, as the algorithm "*should be one of the first choices for a classification study when there is little or no prior knowledge about the distribution of the data*" according to Peterson [29]. Using a classification algorithm to perform fitness calcuation will make the system be able to work independently, which is especially important if the system contains a big population of model designs, as it would lead to the user becoming significantly fatigued when evaluating every single individual in the population. Additionally, it also makes the system more adaptable over time, as the more ratings are being made, the narrower and more accurate the classifier becomes over time. The choice in this project is therefore made to mimic the mixture of a direct like/dislike voting system and a K-NN based fitness evaluation system inspired by Jansson and Bremdal [24] due to its lax user dependency and its adaptability.

1.5.4 Testing

A number of tests and experiments should be made to confirm if the system is able to generate 3D model designs with high variability and relatively good topology. Tests that determine if the system is able to show convergence towards the sentiments of the user of the system should be ran as well. Focusing on directing the system towards a specific model feature set that correspond to some common design easthetic will be performed by liking the designs with a high amount of feature commonality to the desired easthetic, and conversely disliking designs with few to none common design features with the chosen design easthetic. Tests to determine if the system is able to capture specific object shapes will be performed as well, which will follow the same principle of liking only the designs that fulfill the criteria of the specific object shape chosen and disliking all others. Finally, tests will be performed to gauge the systems ability to perform independently using stored used preference data by letting the system purely lean on the automatic fitness evaluation without any user feedback during the testing instance of the program.

2

Tools and Methods

In this chapter the tools and methods that were used in the development of this project will be covered. The first section will outline the chosen tools, their purposes, and their relevant functionalities in relation to our implementation. Subsequent sections will delve into design and implementation details of the methods that were used, as well as the decision making process that was made during the span of development.

2.1 Tools

This project was developed using Python [30] version 3.9.1 as the programming language. The system is implemented as an add-on for the 3D modelling software Blender [22] version LTS 2.93. The Blender add-on "Shape Generator" from Mark Kingsnorth [25] was used as a framework for the model representation and to generate the model meshes in the Blender scene.

2.1.1 Tool selection

The choice of programming language was fully dictated by the available scripting API within Blender, which only offers the use of Python as programming language. Blender was chosen as the 3D modelling software due to its compatibility with the "Shape generator" add-on. Houdini [31] is another 3D modelling program that supports the add-on, but due to the lack of experience using it the decision was made to choose Blender. One could theoretically use a newer version of Blender for the purpose of using the "Shape generator" add-on, as the developer lists compatibility with Blender versions as new as 4.1, but there were compatibility issues when using the add-on in conjuction with versions newer than LTS 2.93 during the development of this project. If newer versions of Blender are required for additional functionalities, one could attempt installing the add-on with newer versions. Alternatively, the version chosen for this project can be used and models can be exported from this version of Blender into a newer one when done using the system.

2.1.2 Blender API

To run custom scripts within Blender the system uses the Python API within Blender. This allows it to manipulate scenes, meshes and UI in the program. One of the API features offered is the ability to create your own tools using custom panels and operators. The tool (add-on) creation functionality is used to create a custom script that runs the interactive system in parallell with all the included Blender functionality. This allows the user to take one of the model designs generated and manipulate it further within Blender to make their desired adjustments, before either exporting the model file to some external software or integrating it into their own Blender project.

2.1.3 Shape Generator

Mark Kingsnorth's Blender add-on "Shape generator" [25] is a custom tool for Blender that generates random shapes given some generator parameters. It works by using a box as the base shape before extruding a random length from a random model face, tapering the extrusions a random amount and rotating the extrusion a random amount. Figure 2.1 provides an example of how this operation works. These extrusions are iteratively performed on the resulting mesh from the previous extrusion until the specified amount of extrusions are done. Example models generated using this method can be seen in Figure 2.2a with 11 extrusions performed, and in Figure 2.2b one can see an example model consisting of 31 extrusions. After the extrusions are performed the mesh can be mirrored along an axis to achieve model symmetry. The shape in Figure 2.2a, mirrored along the X-axis, can be seen in Figure 2.3b, and the same shape mirrored along all axes can be seen in Figure 2.3c.

Additionally this add-on offers the features of applying beveling to the model edges, and applying a subdivision surface to the base mesh. Examples of this can be seen in Figure 2.4 and in 2.5. Section 2.2 covers the specific shape generator features used in the system implemented in this project.

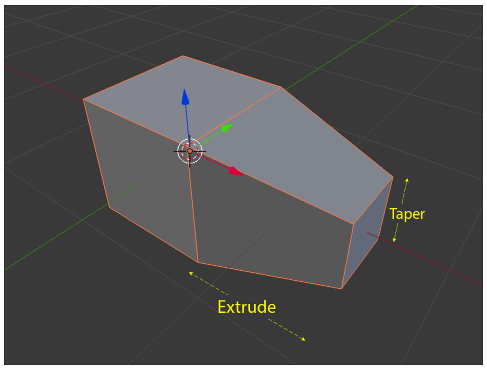
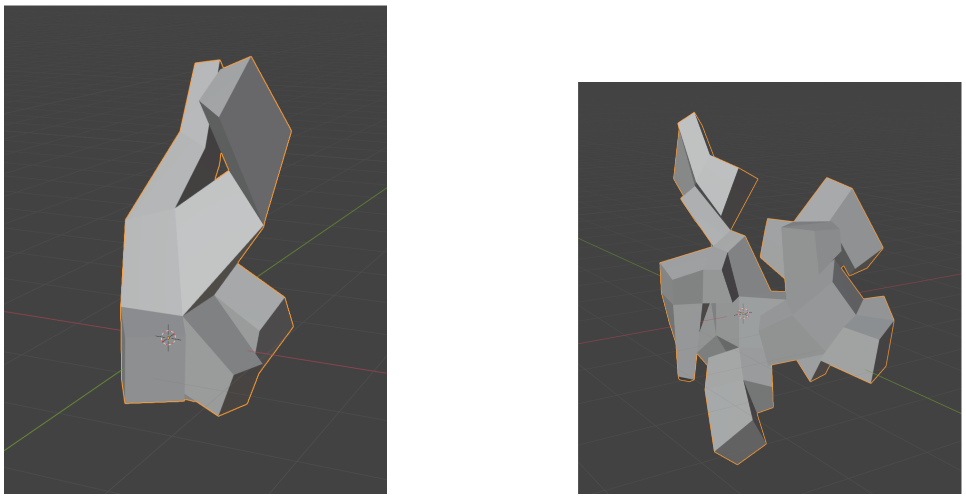


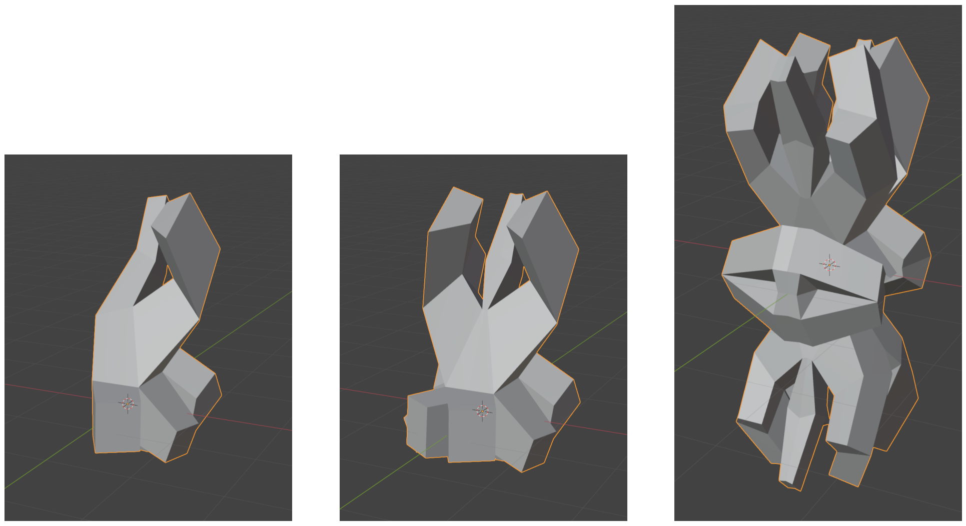
Figure 2.1: Example of extrusion from base cube using the Shape Generator add-on [25]



(a) Using 11 extrusions

(b) Using 31 extrusions

Figure 2.2: Shape examples generated by performing random extrusions



(a) No mirroring **(b)** Mirror along X-axis **(c)** Mirror along all axes

Figure 2.3: Examples of mirroring shape shown in Figure 2.2a

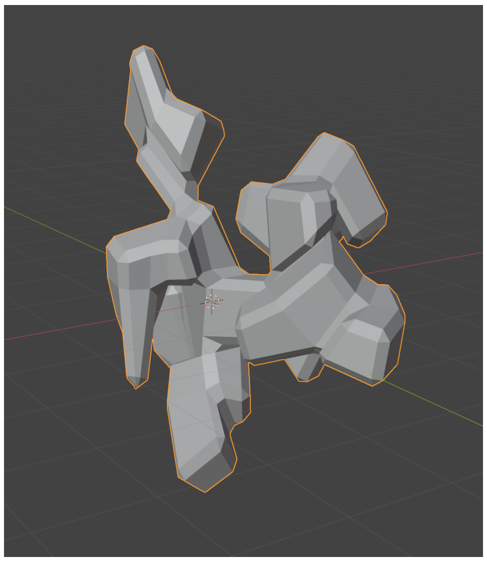


Figure 2.4: Shape shown in Figure 2.2b with beveled edges

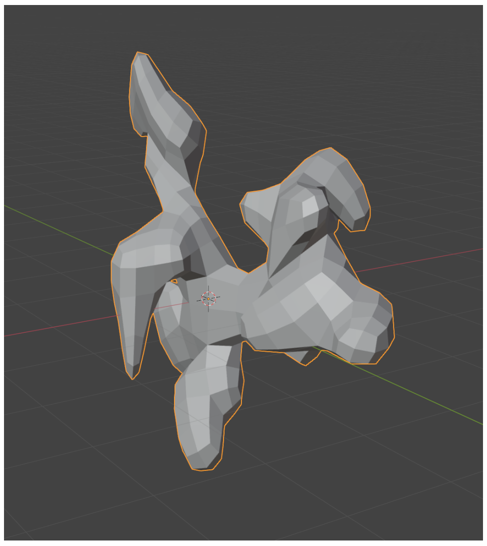


Figure 2.5: Shape shown in Figure 2.2b with a subdivsion surface added to the base mesh

2.2 Model generation and representation

For this project a set of parameters from the Shape Generator add-on was selected to be used as features in the system's model representation. The specific extrusions that are performed on the base cube model are determined by generating random values for the face index, the extrusion length, the taper size and the extrusion rotation. Each of these values are restricted by a defined range of minimum and maximum values. Once all extrusions are completed, the resulting base mesh can be further manipulated by scaling and/or mirroring along the X, Y and/or Z axis. Finally the mesh can have its edges beveled and/or have a subdivision surface added on top of it. A full list of the different parameters for the shape generation that are used in the system implementation can be seen in Table 2.1.

To generate each shape mesh, the implemented system uses the Shape Generator API to insert the resulting mesh into the Blender scene from the shape feature data. Generating each shape mesh is performed by creating a Shape Generator collection for each shape in the scene, and setting the collection properties according to the values in the shape object feature list. Finally, the shape mesh is given a default material with a random color.

Shape Generator properties		
Name	Range	Descritpion
Random Seed	$[0, 2000]$	Seed for random number generation
Amount	$[5, 35]$	Amount of extrusions made
Length Min	$[0.3, 1.0]$	Extrusion length range minimum
Length Max	$[min, 1.0]$	Extrusion length range maximum
Taper Min	$[0.5, 1.0]$	Taper size range minimum
Taper Max	$[min, 1.0]$	Taper size range maximum
Rotation Min	$[-45.0, 45.0]$	Extrusion rotation range minimum
Rotation Max	$[min, 45.0]$	Extrusion rotation range maximum
Favour X	$[0.0, 1.0]$	Favouring of X as extrusion direction
Favour Y	$[0.0, 1.0]$	Favouring of Y as extrusion direction
Favour Z	$[0.0, 1.0]$	Favouring of Z as extrusion direction
Scale X	$[0.5, 1.0]$	Scale mesh along X-axis
Scale Y	$[0.5, 1.0]$	Scale mesh along Y-axis
Scale Z	$[0.5, 1.0]$	Scale mesh along Z-axis
Mirror X	*True/False*	Mirror mesh along X-axis
Mirror Y	*True/False*	Mirror mesh along Y-axis
Mirror Z	*True/False*	Mirror mesh along Z-axis
Bevel	*True/False*	Bevel edges
Subdivision	*True/False*	Add subdivision surface on top of mesh

Table 2.1: Shape Generator properties and their value ranges as used in our implementation

2.3 System UI

For the user to interact with the system it need some sort of UI. The models generated are displayed as a set of 16 shapes in a 4 by 4 grid in the scene with corresponding labels underneath each shape. The shapes selected out of the population are the 16 shapes with the highest fitness evaluation. If there is no basis to judge the fitness of these initial shapes, the system just picks the shapes randomly. Details on how thew system evaluates the fitness of shapes will be discussed in Subsection 2.4.3.

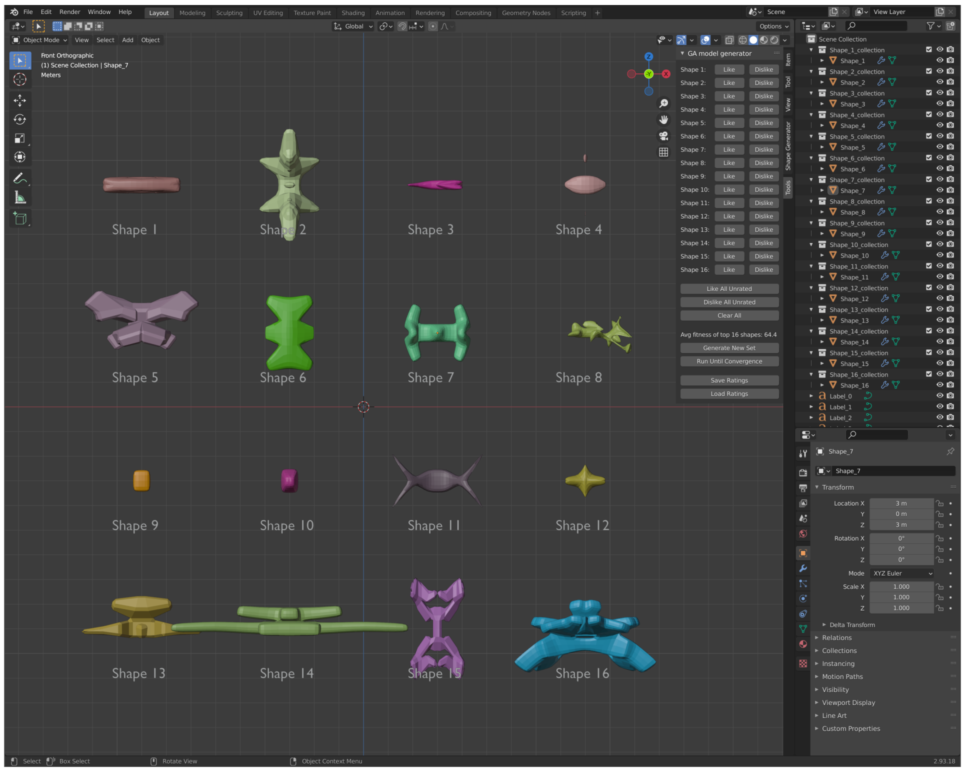
Interacting with the system is done by bringing up the custom toolbar on the right side of the Blender viewport, as seen in Figure 1.5, and choosing the "Tools" section. This toolbar displays a list of the shapes shown in the scene as rows, where each shape has a "like" and a "dislike" button. Underneath this list of shape names and rating buttons, there are helper buttons designed to simplify interaction with the list of shapes. One button likes all the shapes that are not rated yet, another button dislikes all the unrated shapes, while the last helper button clears all existing ratings.

To view the fitness of each individual shape, the user can select the shape they are interested in and right click to bring up the object context menu. In this dropdown menu there is a section dedicated to the custom system where the user can see the predicted fitness made by the system, as well as like and dislike buttons. The total average fitness of the 16 shapes shown in the scene can be seen in the toolbar just below the list of like and dislike buttons.

Once the user has finished viewing shapes and have rated them according to their design appeal, they can press the "Generate New Set" button that when pressed creates a new shape population generation. Alternatively, they can press the "Run Until Convergence" button, which causes the genetic algorithm to run until the average fitness of the top 16 model designs reach a user defined threshold. When the "Run Until Convergence" button is pressed, the system prompts the user to input for the convergence threshold for the average fitness, and the maximum number of generations that are ran. The convergence depends on the fitness of the top 16 shapes and not the average fitness of the whole generation, as the user will only ever see the top 16 shapes in the Blender viewport.

In the case that user wants to save their preferences for future use, they can store the shape rating information they have provided to the system by using the "Save Ratings" button. Pressing this button prompts the user to specify a file name for the file containing the rating data. To load a saved rating data file, the user can press the "Load Ratings" button, which opens the Blender file view window. In this window the user can pick the desired ratings file they want to dictate the fitness evaluation for this instance of the system.

Figure 2.6 shows an example of the system running within Blender, where the custom toolbar is visible on the right of the viewport. The custom toolbar can be seen more closely in Figure 2.7, and an example of the object context menu can be seen in Figure 2.8.



GA model generator

Shape 1:	Like	Dislike
Shape 2:	Like	Dislike
Shape 3:	Like	Dislike
Shape 4:	Like	Dislike
Shape 5:	Like	Dislike
Shape 6:	Like	Dislike
Shape 7:	Like	Dislike
Shape 8:	Like	Dislike
Shape 9:	Like	Dislike
Shape 10:	Like	Dislike
Shape 11:	Like	Dislike
Shape 12:	Like	Dislike
Shape 13:	Like	Dislike
Shape 14:	Like	Dislike
Shape 15:	Like	Dislike
Shape 16:	Like	Dislike

Like All Unrated
Dislike All Unrated
Clear All
Avg fitness of top 16 shapes: 64.4
Generate New Set
Run Until Convergence
Save Ratings
Load Ratings

Shape_7

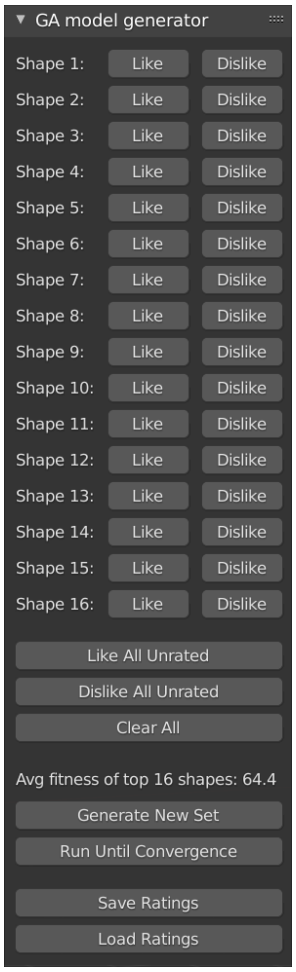
Shape_7

Transform

Location X	3 m
Y	0 m
Z	3 m
Rotation X	0°
Y	0°
Z	0°
Mode	XYZ Euler
Scale X	1.000
Y	1.000
Z	1.000

Delta Transform
Relations
Collections
Instancing
Motion Paths
Visibility
Viewport Display
Line Art
Custom Properties

Figure 2.6: Blender UI with our add-on system running



Avg fitness of top 16 shapes: 64.4

Figure 2.7: Custom Blender tool UI

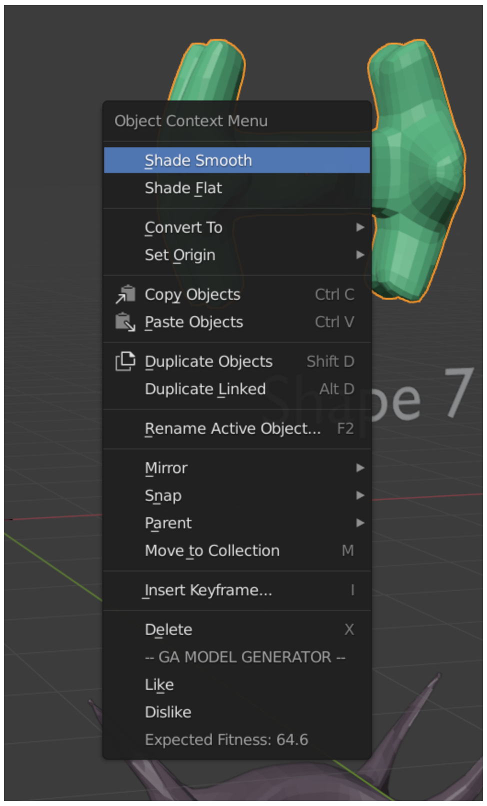


Figure 2.8: Object context menu with custom generator section

2.4 Genetic algorithm

Now that system has the ability to display the generated models to the user, and they are able to vote on which models they like and dislike, the system can introduce a genetic algorithm to generate new sets of models based on the feedback from the user. To achieve this, three main steps have to be implemented: the encoding of model parameters, the genetic algorithm structure, and an automatic fitness evaluation that is based on user preferences.

2.4.1 Encoding

During development, a binary encoding of the model parameters was experimented with, but using a binary encoding lead to a few drawbacks. Implementation complexity increased with every feature added due to the need of calculating the bit offset of each new feature that was integrated, as well as the possibility of value range constraints being broken depending on the bits allocated to each feature. However, the main problem with using a binary encoding was the lack of correlation between the degree of change in the genotype and the degree of change in the phenotype. By this we mean the lack of correlation between changes in the genetic coding and changes in observable traits. This issue can be seen in action in the example of a binary crossover as shown in Figure 2.9, where the resulting decoded integer values of the child chromosomes are completely unrelated to the parents' integer values.

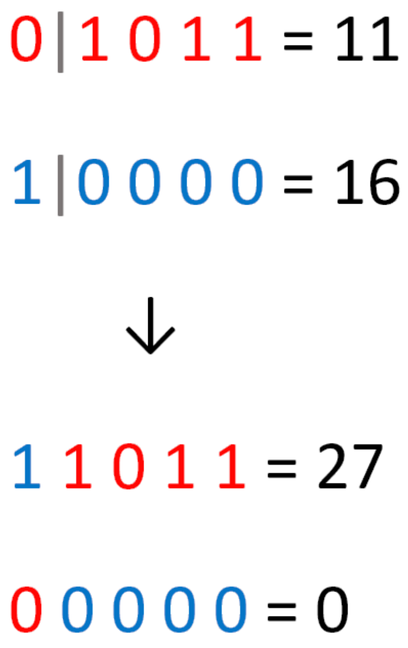
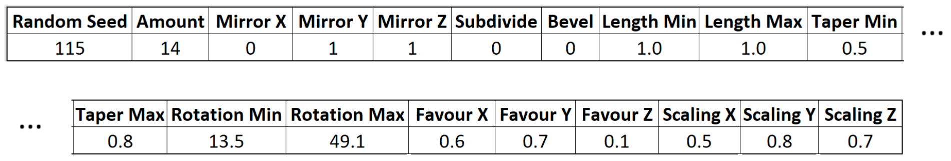


Figure 2.9: Binary crossover leading to unrelated integer values

This contradicts the creativity goal of the system as stated in Section 1.2, in the sense that a combination of two different designs who share some features, while other features are different, are combined into a new design that finds a middle ground between these two imperfect models in terms of user preference. Another problem with a binary encoding is the uneven probability of gene mutation between the different features, as the length of each feature bit sequence dictates the probability of a bit in that specific feature being flipped. Due to these issues, the decision was made to adopt a real encoding of the model parameters into chromosomes. An example of a chromosome with the model parameters encoded into real values can be seen in Figure 2.10.



Random Seed	Amount	Mirror X	Mirror Y	Mirror Z	Subdivide	Bevel	Length Min	Length Max	Taper Min
115	14	0	1	1	0	0	1.0	1.0	0.5

Taper Max	Rotation Min	Rotation Max	Favour X	Favour Y	Favour Z	Scaling X	Scaling Y	Scaling Z
0.8	13.5	49.1	0.6	0.7	0.1	0.5	0.8	0.7

Figure 2.10: Example of chromosome with model parameters using real encoding

2.4.2 Genetic algorithm structure

Using the encoded model parameters, the system runs a genetic algorithm that can search the solution space defined by the model parameters as dimensions, and find the users preferred design features. To begin with, it generates an initial population of 200 shapes with random parameters, and displays 16 random samples to the user, as described in Section 2.3. After the user is done with a generation, the system replaces it with a completely new population of 200 new shapes. To select the parents who will produce the new generation of model designs the algorithm performs a roulette wheel selection scheme, as seen in Algorithm 1. For each couple of parents selected, the algorithm generates two offspring models by performing a 2-point crossover of the two parents' chromosomes, as shown in Algorithm 2. This crossover operation is performed at a rate of 0.9. After the crossover operation is done and the system has two resulting offspring models, it performs gene mutation at a rate of $\frac{1}{\text{\# of parameters}}$, which ends up being $\frac{1}{19}$ for our set of parameters. Mutation is performed by sampling a new random value for the given gene that is within the pre-determined value-range. This will result in each new offspring having one gene mutated on average.

Algorithm 1 Roulette Wheel Selection

```
1: function RouletteSelect(shapes)
2:     List $F$
3:     $s \leftarrow \sum_i$ shapes$[i]$.fitness
4:     for each shape in shapes do
5:         $F$[shape] $\leftarrow$ shape.fitness / $s$
6:     end for
7:     $r \leftarrow$ random index $\in [0, |\text{shapes}|)$ with probabilities $F$
8:     return shapes$[r]$
9: end function
```

Algorithm 2 Two-Point Crossover

```
1: function MultiCrossover(parent1, parent2)
2:     point1 ← random number ∈ [1, |parent1| − 2)
3:     point2 ← random number ∈ [point1, |parent1|−1)
4:     child1← parent1[:point1] + parent2[point1:point2] + parent1[point2:]
5:     child2← parent2[:point1] + parent1[point1:point2] + parent2[point2:]
6:     return child1, child2
7: end function
```

2.4.3 Fitness evaluation

The fitness of each individual model design is determined by the user either through the direct interaction of liking or disliking it, or by using the learning of the previous preferences made by the user. A liked individual is given the maximum fitness value of 100, while a disliked individual is given the minimum fitness value of 1.

Every liked model design has its model parameters, as well as the rating given, added to a list of rated designs. Inspired by the method used by Bremdal and Jansson [24] the system uses this list of rated shapes and performs K-nearest neighbour regression to calculate the estimated fitness on the given unrated shape. To do this it calculates the sum of normalized parameter distances for the given shape to each shape in the rated shapes list:

$$Distance(a,b) = \sqrt{\sum_{p \in P} \left(\frac{a_p - b_p}{MAX_p - MIN_p} \right)^2}, \tag{2.1}$$

where P are the model parameters. This distance is normalized so that the system avoids parameters with bigger value ranges having a greater influence than parameters with smaller value ranges when calculating sum of the parameter differences. The list is then sorted in terms of shortest distance to the given shape such that the system finds the k nearest neighbours as the k first entries in this list. Now the system calculates the estimated fitness of the given shape as:

$$Fitness(s) = \frac{100 * \sum_{l \in L_s}^{n} 1 + 1 * \sum_{d \in D_s}^{m} 1}{n + m}, \tag{2.2}$$

where L_s and D_s are the sets of liked and disliked shapes of the k nearest neighbours for the given shape s.

One weakness this fitness estimation has is that it does not take into account which of the neighbours are actually close to the given shape and which are far away. Weighted K-NN, as described by Gou et al. in [32], is a varation of K-NN where one weighs the neighbours based on their relative distance to classify a given individual. In their paper they show that this weighted K-NN classification performs better for higher values of k compared to the standard K-NN algorithm. Thus, as an alternative the system has a fitness evaluation method that weighs each neighbour in terms of their distance to the given shape relative to the other neighbour distances:

$$Fitness_W(s) = \frac{100 * L_W + 1 * D_W}{L_W + D_W}, \tag{2.3}$$

$$L_W = \sum_{l \in L_s}^{n} \frac{1}{Distance(s,l) + \epsilon}, \qquad D_W = \sum_{d \in D_s}^{m} \frac{1}{Distance(s,d) + \epsilon},$$
$$\epsilon = 1e^{-6}$$

where L_W and D_W are the weighted sums of liked and disliked shapes among the neighbour shapes, and ϵ is a small constant to avoid division by zero errors when comparing distances between identical shapes. Using this method the system should be able to use more granular information regarding how close each neighbour is to the given shape, such that neighbours with more similarity (less distance) with the given shape is weighted higher than less similar ones when calculating the estimated fitness.

To save the user's preferences for future use the system allows the user to store the rated shapes list into a user-defined JSON file. The user can then load one of the saved files into the rated shapes list for this new instance of the genetic algorithm. This way the user can continue using the previous ratings made to evaluate the fitness of the new designs in the populations generated this instance of the system. Switching out the stored file with a different one with other preference data will enable the user to have a modular approach to what type of shapes they want to generate for each time they use the system, as the system will adapt its fitness evaluation according to what data is used during the K-NN regression.

2.5 System structure

A flow diagram of the system can be seen in Figure 2.11, which illustrates the sequence of actions performed by our implementation. The system object diagram is shown in Figure 2.12, which outlines the different methods and classes used in our implementation and their relationships.

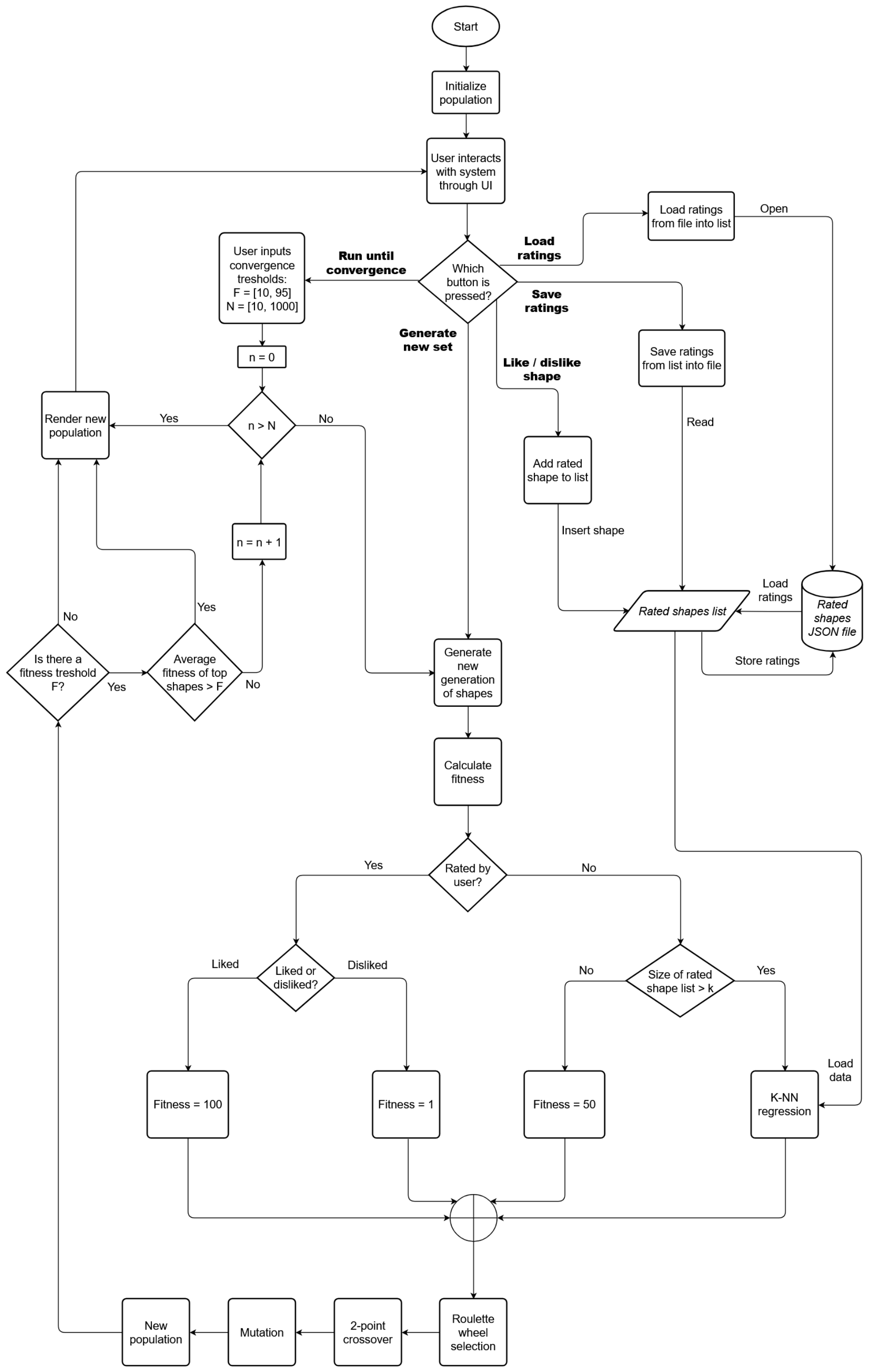


Figure 2.11: Flow diagram of the system

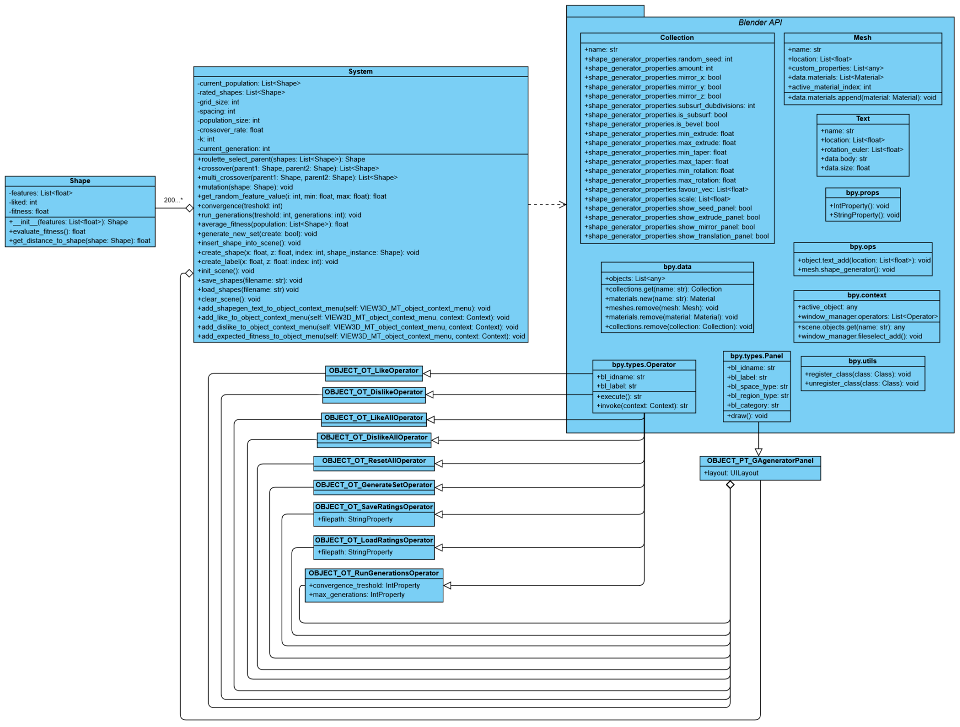

Blender API
Collection
Mesh
Text
bpy.props
bpy.ops
bpy.data
bpy.context
bpy.types.Operator
bpy.types.Panel
bpy.utils
System
Shape
200..*
OBJECT_OT_LikeOperator
OBJECT_OT_DislikeOperator
OBJECT_OT_LikeAllOperator
OBJECT_OT_DislikeAllOperator
OBJECT_OT_ResetAllOperator
OBJECT_OT_GenerateSetOperator
OBJECT_OT_SaveRatingsOperator
OBJECT_OT_LoadRatingsOperator
OBJECT_OT_RunGenerationsOperator
OBJECT_PT_GAgeneratorPanel

Figure 2.12: Object diagram of the system

3 Results and Discussion

3.1 Results

This section will go over the tests and experiments performed using the implemented system. Firstly, some examples of how the system works without loading any previous user preference data will be covered. Then some examples of how loading different user preference datasets can impact the system will be explored. All tests were performed using the following system parameters:

- population size: 200
- crossover rate: 0.9
- mutation rate: $\frac{1}{19}$
- $k = 10$

3.1.1 Combining features

As stated in Section 1.2, one of the goals of the system is to be able to show creativity through a combination of design features. In the first experiment the seed for random number generation is set to 6. This seed produces the starting population shown in Figure 3.1.

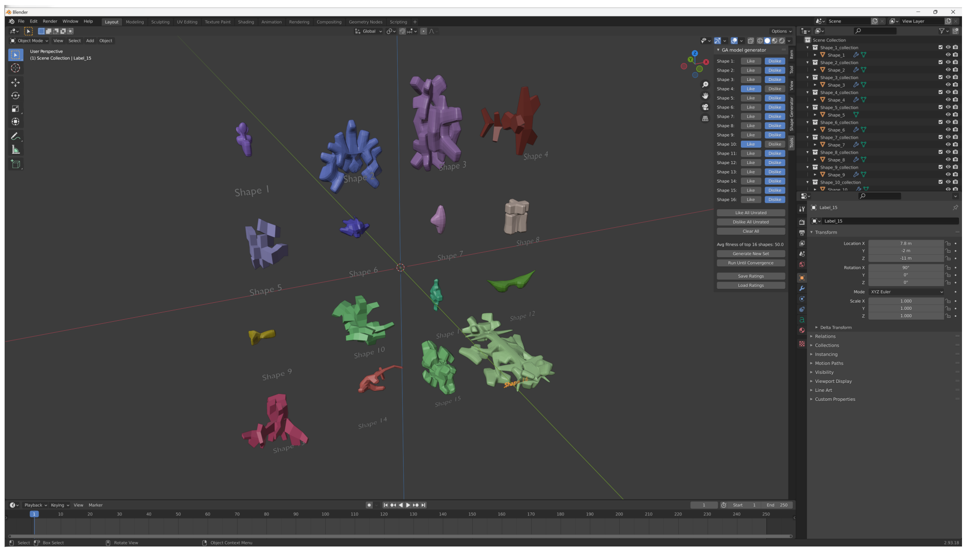


Figure 3.1: Initial population of shapes using seed = 6

From this population, the goal is to examine if the system can generate "boxy" shapes that are mirrored along the Y-axis. Due to these design criteria, the shapes 4 and 10 are liked, while all the other shapes are disliked. With 16 ratings now available to the system ($16 > k$), it can proceed to perform estimated fitness evaluations on the rest of the population. The generation that follows can be seen in Figure 3.2. As seen in the next generation, the system has successfully generated new shapes that follow the design eastethic of the two shapes that was initally liked and has recognized that shapes with different design parameters were disliked.

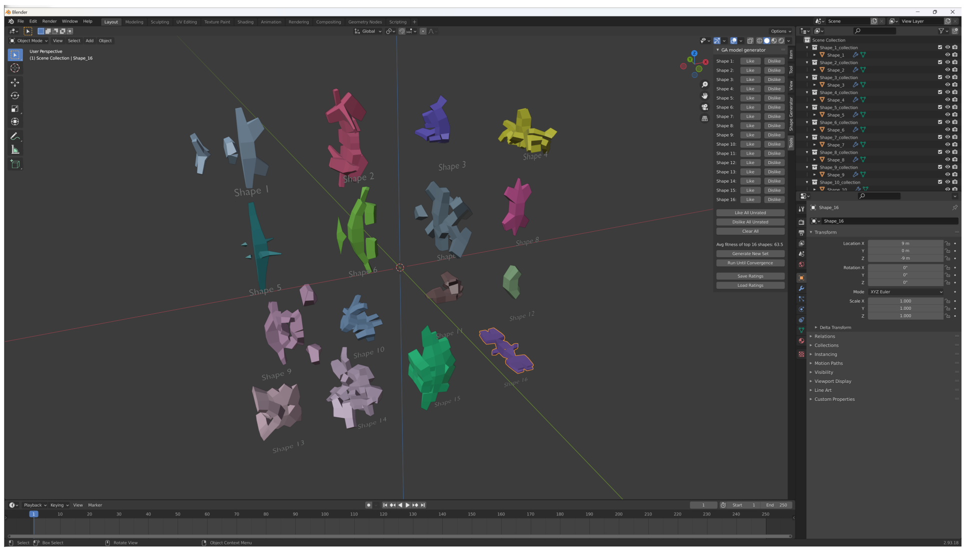


Figure 3.2: Generation 2 with Figure 3.1 as the initial population

The system is then ran until it converges, with the parameters of 95 as the fitness threshold and 100 generations as the stopping point. It stops after 100 generations and the resulting shapes can be seen in Figure 3.3.

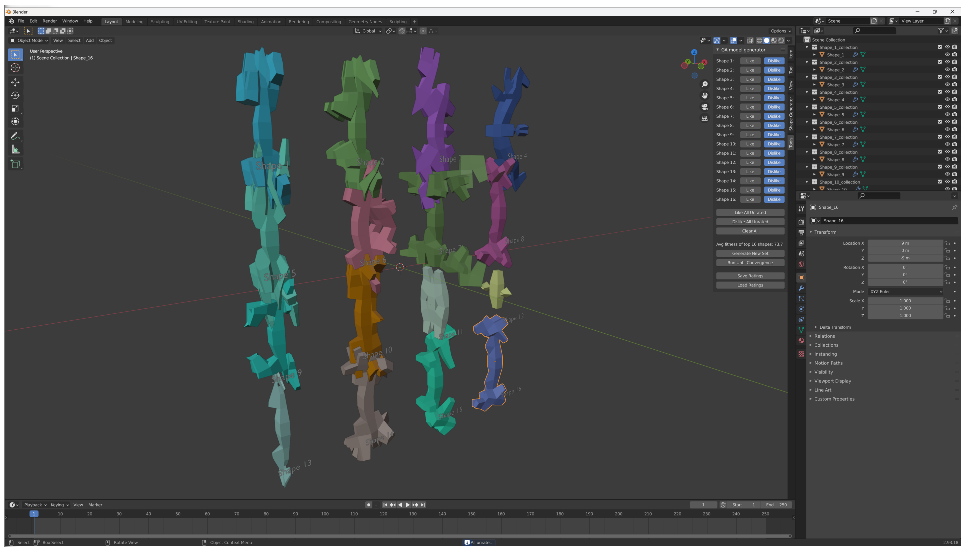


Figure 3.3: Generation 102 with Figure 3.1 as the initial population

At this stage another criteria is added in the form of not overly tall/long shapes. Consequently, all the shapes shown are disliked. The system generates a new generation, and the result can be seen in Figure 3.4.

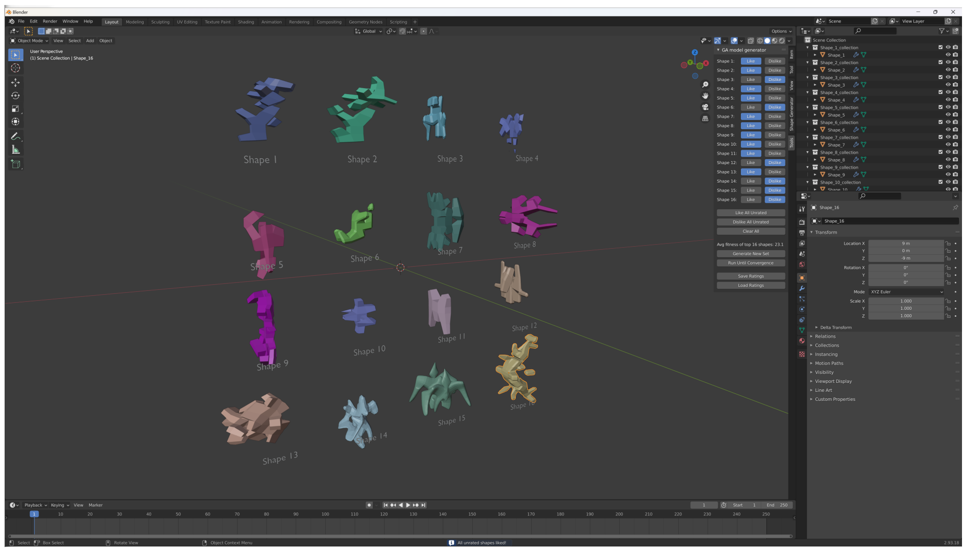


Figure 3.4: Generation 103 with Figure 3.1 as the initial population

As seen from the resulting generation of shapes, the system has overcorrected after the previous generation ratings in the sense of producing shapes without the "boxy" apperance. Therefore, all the "boxy" shapes are liked while all the other shapes are disliked, then a new set of shapes is generated. The shapes that was generated following these ratings are shown in Figure 3.5

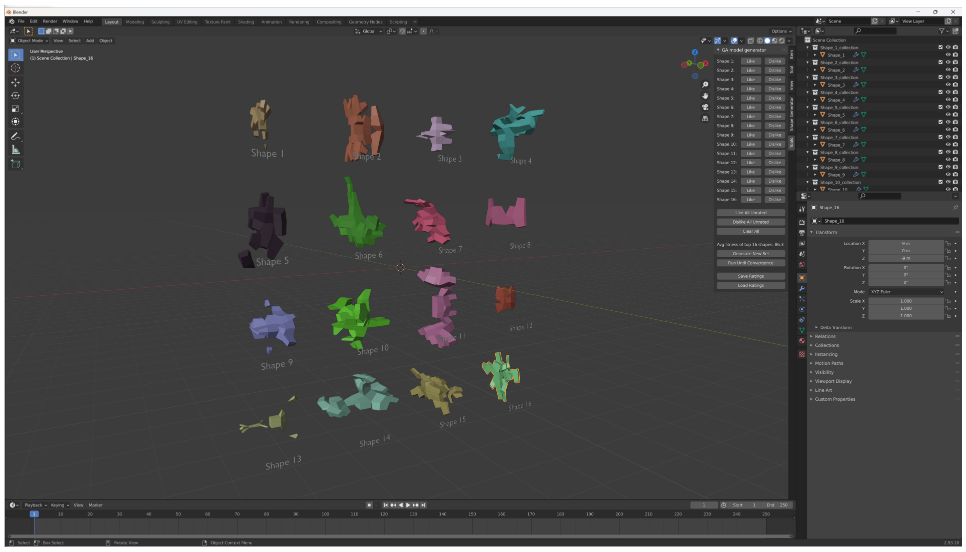


Figure 3.5: Generation 104 with Figure 3.1 as the initial population

In the next experiment, the design criteria is set to prefer asymmetrical and organic looking shapes. Starting with the same base population as seen in Figure 3.1, the system generates one more generation without rating any shapes. This produces the following shapes as seen in Figure 3.6.

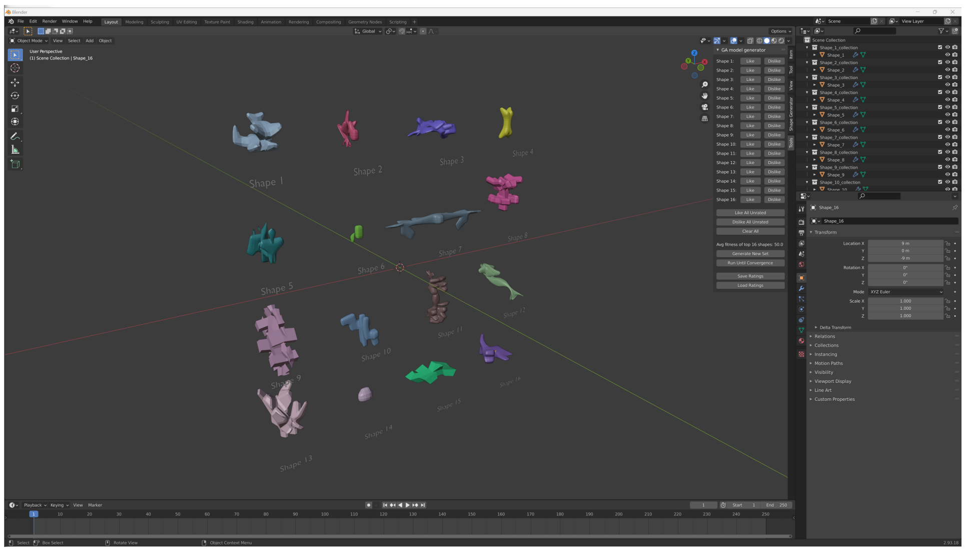


Figure 3.6: Generation 2 with Figure 3.1 as the initial population, without any ratings performed

From this set of shapes, only shape 3 fulfills the set design criteria. Thus, shape 3 is liked and all other shapes are disliked. Then the system tries to reach convergence using 95 as the fitness threshold and 100 generations as the end point. The shapes shown in Figure 3.7 are the top shapes from the resulting generation after 100 generations are ran.

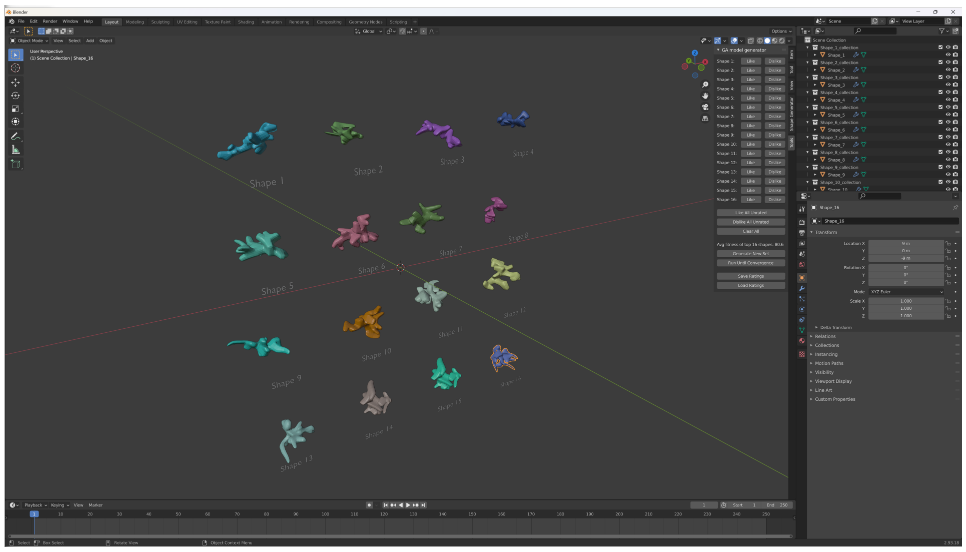


Figure 3.7: Generation 102 with Figure 3.6 as the initial population

All of the results shown so far has been using the distance weighted K-NN fitness evalutaion, as described in Subsection 2.4.3. An attempt at replicating the experiment as shown in Figure 3.7 with the regular K-NN fitness evaluation produces the generation shown in Figure 3.8 after 100 generations.

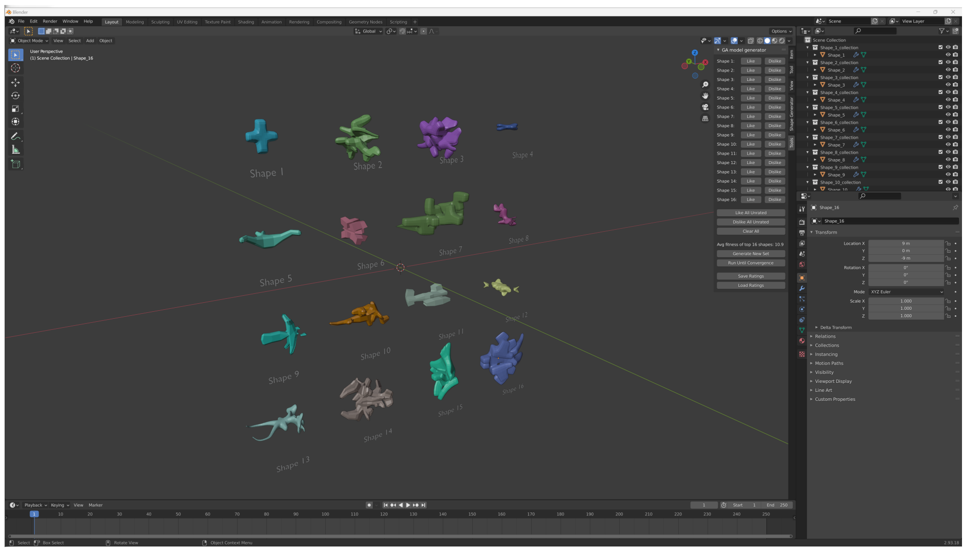


Figure 3.8: Generation 102 with Figure 3.6 as the starting point, without distance weighted K-NN

3.1.2 Specific objects

In the final experiment without using a file with previous ratings, an attempt was made to try to get a set of shapes that all look like the same general object, but with small visual differences between them. The generation that can be seen in Figure 3.9 was used as the base, where then shape 1 is liked since shapes that resembles old-school phones are the ones that are desirable for this experiment.

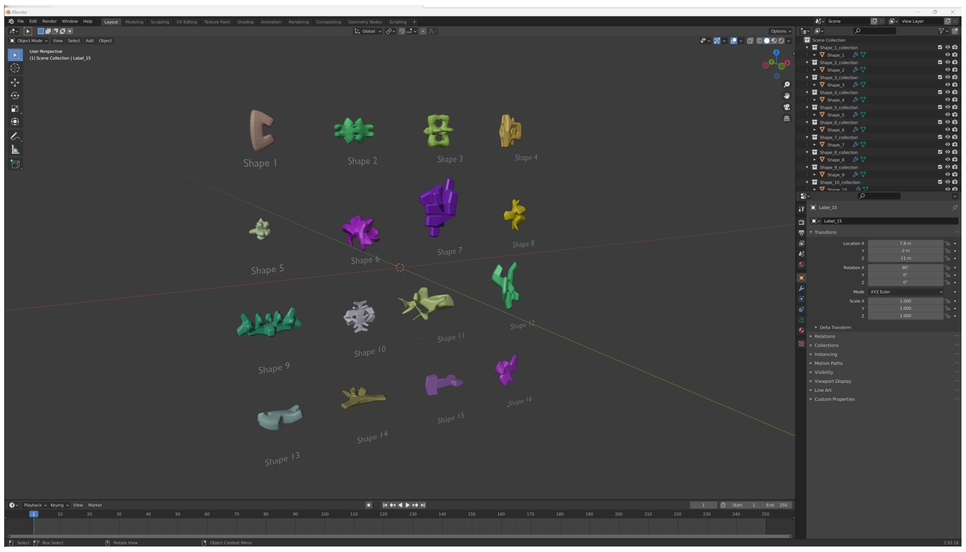


Figure 3.9: Base generation for "phone" shape experiment using seed = 1

After performing manual like and dislike ratings for 60 generations (960 ratings), the system generates the shapes seen in 3.10.

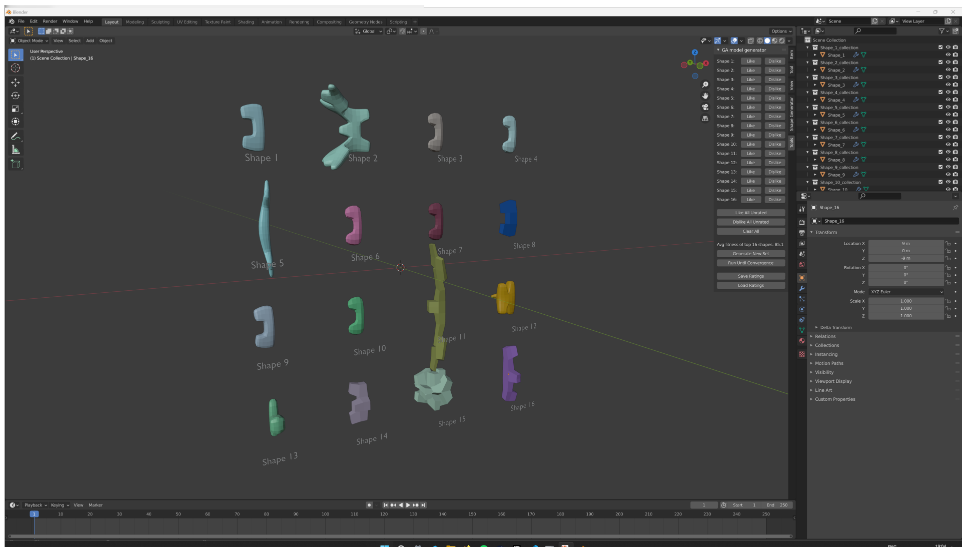


Figure 3.10: Generation 61 after performing 960 ratings, with Figure 3.9 as the starting point

3.1.3 Using learned preferences

Testing the systems ability to learn was performed by storing the 48 ratings made during the experiment shown in Figure 3.5 in a JSON file, and loading that file in a new instance of the system using a different random seed = 1. After loading the ratings file and being prompted to converge (fitness threshold = 90), the system converges after 14 generations, where the final generation's top shapes are seen in Figure 3.11.

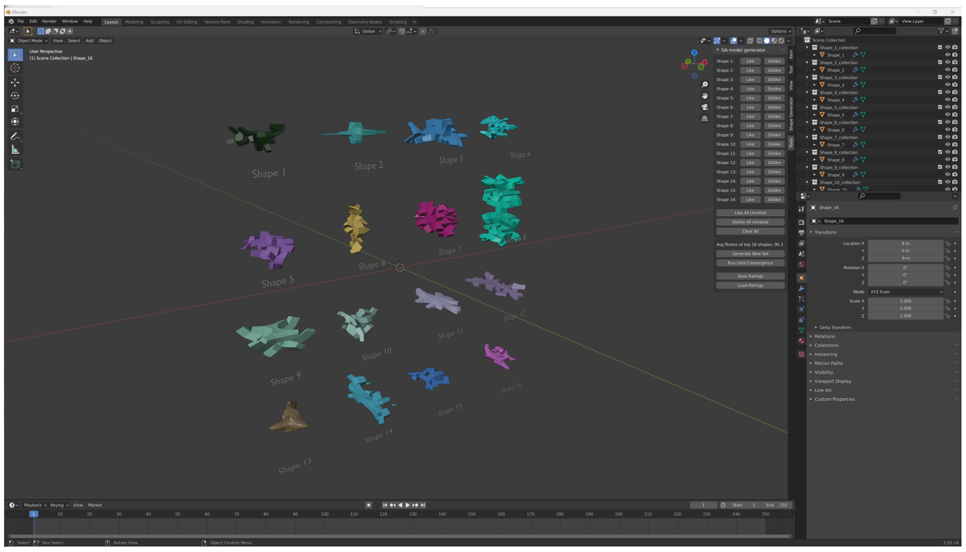


Figure 3.11: Generation 14 with Figure 3.9 as starting point, and using ratings data from experiment shown in Figure 3.5

Another experiment was performed with seed = 0, convergence threshold = 80, and using the ratings described in Subsection 3.1.2 as the rated shapes file. This resulted in the inital shapes as shown in Figure 3.12, where the system converges after 19 generations and producing the shapes as shown in Figure 3.13.

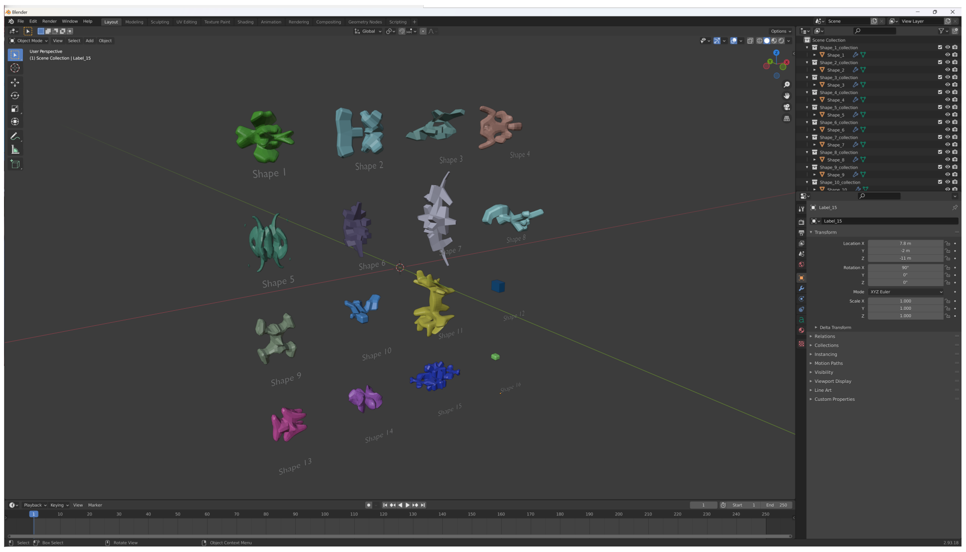


Figure 3.12: Inital population with random seed = 0

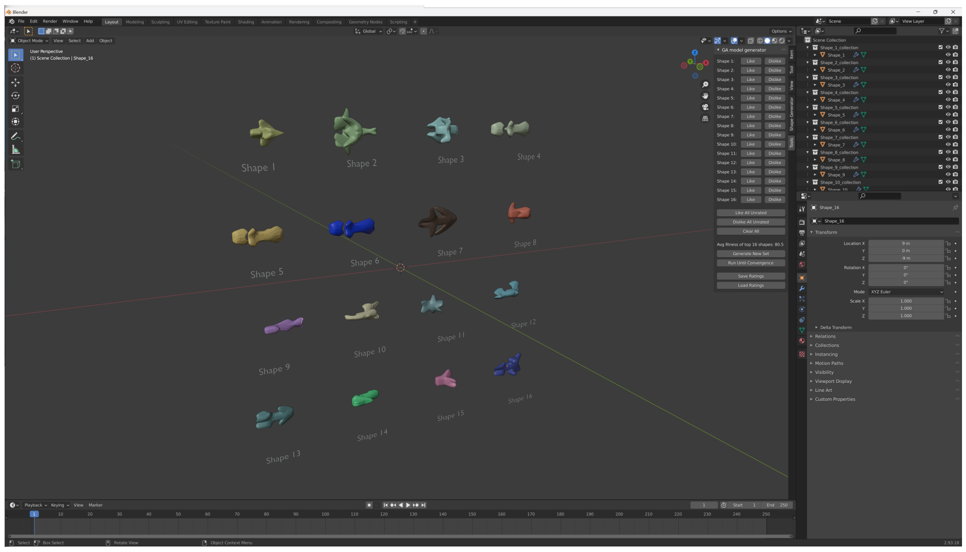


Figure 3.13: Shapes produced after 19 generations using Figure 3.12 as starting point and using ratings from experiment shown in Subsection 3.1.2

Continuing from that generation and adjusting the fitness threshold to 85, the system is not able to converge within 100 generations. The shapes produced 100 generations later are seen in Figure 3.14.

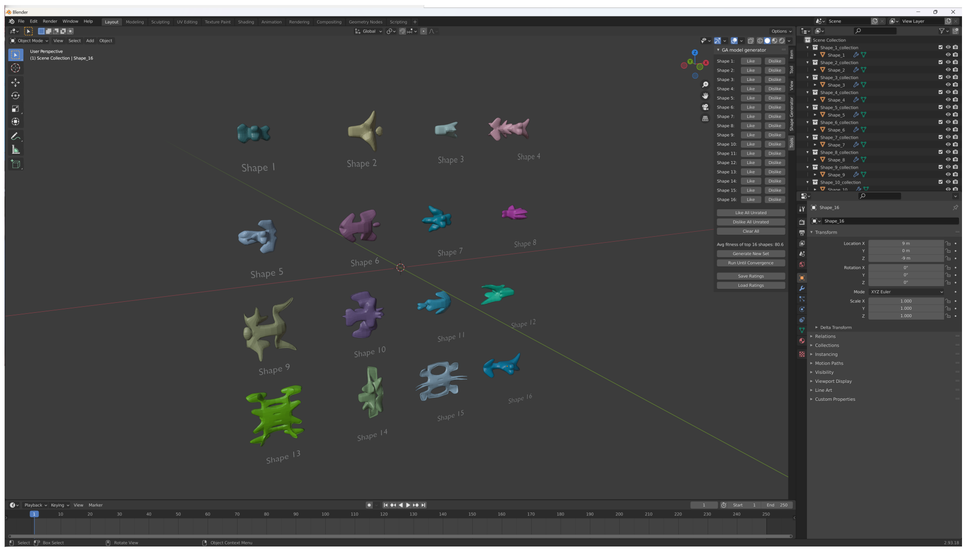


Figure 3.14: Shapes produced after 119 generations using Figure 3.12 as starting point and using ratings from experiment shown in Subsection 3.1.2

3.2 Discussion

This section will cover the analysis of the results shown in Section 3.1, and examine to what degree of which the objectives set out in Section 1.2 were accomplished.

3.2.1 Analysis

The results of the first experiment in Subsection 3.1.1, as seen in Figures 3.1, 3.2, 3.3, 3.4, and 3.5, demonstrate that the system is capable of capturing common model paramateres and produce new designs accordingly. Over time, it converges towards the "optimal" design by maximizing the similarity with liked designs and minimizing similarity with disliked designs. This is evident in Figure 3.3, which illustrates that the system has figured out that the liked designs are tall as well as matching the other stated design characteristics. The system shows that it is adaptable as well in this example, as it pivots away from tall designs as soon as it recieves the information from the user that this characteristic is not appealing. However, the system shows signs of over-correcting, as seen in the following generation shown in Figure 3.4. This is likely due to the fact that the ratings that have been made are unbalanced in the sense that most of them have been dislikes, which doesn't give the system enough information to guide it in the specific direction the user wants. Nonetheless, one more generation of both liking and disliking designs is enough to get the system back on track and create the type of shapes that correspond with the given characteristics, as illustrated in 3.5.

From the second experiment in Subsection 3.1.1 one can see that the system is capable of functioning to a relatively good degree with minimal input from the user. As demonstrated in Figure 3.7, the system is able to generate shapes that correspond well with the parameters of shape 3 in Figure 3.6, despite the fact that it only has the ratings from that inital population to go off of. This good performance is largely based on the decision to use the distance weighted K-NN algorithm during fitness calculation, as mentioned in Subsection 2.4.3. This is evident from the results shown in 3.8, which are produced by running the same experiment using a standard K-NN algorithm. Here one can see that the essence of the liked shape is mostly lost among the resulting shapes, as only a few of them share the same characteristics of having the orgranic and assymetric appearance.

During the test aimed at capturing the basic shape of an object, as shown in Subsection 3.1.2, the system shows some struggles. While it generates a few shapes that does not look like a phone at all, it does generate several shapes that do. However, these don't differ much from each other in the excemption of the parameters of beveling and subdivision, as the base of the shapes themselves appear identical. This is even after manually rating almost 1000 shapes, so the system shouldn't lack the information of which shapes have the desired parameters and which do not. The reason for this underwhelming performance is likely due to the choice of representation not encouraging shapes to follow the same sequence of extrusions, as it only restricts the range of randomization which the set of extrusions are generated from. Modelling a phone-like shape as seen in the examples isn't really possible to do consistently by performing random extrusions within some range of values, since one small deviation the sequence of extrusions can lead to a result that is very different in terms of what type of object the shape resembles. How one could try to fix this weakness of the system will be discussed in Subsection 4.1.1.

The first experiment demonstrated in Subsection 3.1.3 shows that the system is able to learn previous user preferences and use it in new instances of the system without user interaction. The shapes seen in Figure 3.11 shows that the shape characteristics are consistent with the rating data provided from the first experiment in Subsection 3.1.1. From the results of the second experiment, it is evident that the system shows even worse performance in terms of capturing the essence of how an object looks compared to the initial experiment shown in Subsection 3.1.2. As the system likely never manages to encounter any of the phone specific shapes during its new instance with a different inital seed, it struggles to converge sufficiently towards the same results as the inital experiment. This result further highlights that the system is not conditioned to be able to discern specific object structures in the current model representation.

From all the experiments conducted, it is clear that the model topology of the generated shapes is relatively good. None of the shapes shown have overlapping edges, and they all appear to have even vertex distributions. The majority of the shapes show a coherent structure and volume, but some have parts of the mesh being overly thin and long compared to the rest of the mesh. As well as this issue, some shapes display gaps in the model continuity in the sense of discountinous "islands" in the mesh. Both of these weaknesses are probably caused by the mirroring of the base shapes. When a section of a mesh is mirrored along its base, it could be "cut" along the mirroring axis in an uneven fashion, leading to the resulting section of the shape having a thinner (and concievely longer) look despite the base mesh not having this issue. We can rule out the base shape as the root of this issue, as it is always continuous due to the basic construction principle of only performing extrusions on the base mesh. The section of the mesh where the connection from one "island" to another could end up not being visible when it is placed on the other side of the mirror along an axis.

3.2.2 Objectives

From the analysis that is covered in Subsection 3.2.1 one can conclude that the implemented system was able to achieve the objectives described in Section 1.2 to some degree.

The choice of 3D model representation that was made in this project enabled the system to generate varied and relatively complex shapes, without encountering considerable topology issues, in contrast to some of the model representations covered in Section 1.4. At the same time, the representation acts as an hinderance in terms of capturing the specific shape of an object and finding a consistent way of generating small variations of that same object shape.

The interactive genetic algorithm in the heart of the system worked as expected in both the aspect of being able to show creative capabilites through combining features into new designs, and the aspect of allowing the user to dictate the heuristic used for searching the design space in the direction that correlates with the users preferences in an effective manner. Introducing a distance weighted element into the K-NN fitness evaluation from [24] helped the system considerably in guiding the search effectively.

The overall quality of the models generated is hard to determine. As shown in the example of Figure 1.2, there are modelling techniques that effectively refine simpler models into something more refined and complex in design. If some of the shapes generated by this system are of the quality to be used for such a purpose is left up to 3D modelling artists to decide. However, the difficulties shown by the system to capture specific object shapes would probably prove a hinderance to artists with a specific object model in mind. Thus, the system should be used primarly for the inspiration of models that share a set of model characteristics that are not overly specific in terms of shape structure.

4 Future Work and Conclusion

4.1 Future work

4.1.1 Representation alternatives

As described in Subsection 3.2.1, the 3D model representation chosen has a few weak areas that could be improved. A possible approach to improve the systems capability to capture specific object shapes, in the sense of capturing more specific shape structures, is to change the representation from a set of general extrusion parameter values and value ranges which are sampled randomly to a sequence of specific extrusion parameterers that specify each extrusion individually. This should lead to sequences with small changes from each other having an overall more similar final shape structure, as performing a similar sequence of specific extrusions should result in a similar overall shape. Removing the random element entirely, or defining ranges for each extrusion individually, are both possible avenues that could be explored to improve the model representation while keeping the core representation idea of performing extrusions from a base mesh.

4.1.2 Fitness evaluation adjustment

The fitness evaluation performed using the system implemented in this project works relatively well, but there are a few tweaks that could be made to improve it further. Firstly, the time to compute the fitness of each individual could be increased considerably by using multi-threading during the distance calculation. This should not be a problem as this is a task that could be parallelized without any race conditions or deadlocks, as long as the set of shapes to calculate the distance to is divided into seperate buckets for each thread. Additionally, sorting the whole list of distances is not really necessary, as the system is only reliant on the closest k shapes when performing the K-NN regression of fitness values. Thus, choosing a sorting algorithm that can perform early stopping as soon as the first k shapes are sorted in ascending order would theoretically increase the speed of compuation of the system considerably.

The fitness values themselves could be aligned better with user sentiments, and consequently lead to a better overall system. One method of achieving this could be through using parameter (gene) weights, where each parameter is given a number that determines its importance in terms of its impact on designs as experienced by the user. This could be defined by the system in a general matter, or it could be user adjustable where they dictate which parameters they think are most impactful to determine which designs are desirable to them.

4.1.3 Design space search

Any way of helping the system to narrow down the search through the design space could help the system converge quicker towards the desired model designs. The idea of allowing the user to lock parameter values when they feel they have reached the desired values among the current generation could help the system avoid drifting away or overcorrecting from a desired set of feature values. This would allow the system to avoid searching sections of the design space that the user has determined that they are not interested in, which should lead to more of the space that is still uncertain to be explored instead.

Figuring out a way to calculate the general trend of where the system is heading in the desing space is an idea that was brought up by my supervisor, Bernt Arild Bremdal, during the development of the project. If the system had a general trendline that represented the direction of travel through the solution space it could help to describe exploration effectiveness through the percieved convergence rate, and could help detection and mitigation of plateauing and stagnation through dynamic adjustmens to system parameters, such as the mutation rate. A specific method to calculate this trendline was not discovered during this project, but could be a possible avenue for future research work for the development of similar systems.

4.2 Conclusion

This thesis has examined a possible design and implementation of a creative AI system using a genetic algorithm that is capable of generating 3D models for the use in video game development. A system was developed that could generate 3D models in the modelling software Blender [22], using the add-on "Shape Generator" from Mark Kingsnorth [25] as the base for the model representation. The system is guided through the design space using an interactive genetic algorithm, which allows the user directly determine the fitness of each design through rating each model in a custom UI panel within Blender. Additionally, a K-NN regression method was implemented to calculate estimated fitness values for designs that are not rated by the user, which allowed the system to function without user feedback for every generated model design.

The tests and experiments conducted on the system demonstrated its capability to generate novel and creative design solutions by combining various model features. Additionally, the system exhibited adaptability by dynamically adjusting to new user feedback during its runtime. Furthermore, tests revealed that the system can operate independently when provided with previous user preference data, generating designs that align with the preferred design criteria gathered during a previous instance of the system.

However, the system exhibits some weaknesses and areas for improvement. It struggles to capture specific shape structures of objects and presents topology issues such as disconnected and overly thin mesh segments. Additionally, the fitness estimation of each design could be refined to better align with user sentiment and be optimized for quicker distance calculations between designs for automatic fitness evaluation. Suggestions to address these issues include adjusting the model representation, introducing weights and multi-threading to the fitness calculation process, as well as implementing feature locking and analyzing the general trendline of traversal through the solution space.

Bibliography

[1] pwc, “Top 5 developments driving growth for video games.” https://www.pwc.com/us/en/tech-effect/emerging-tech/emerging-technology-trends-in-the-gaming-industry.html„ 2024. Accessed: 2024-05-02.

[2] E. Gach, “Blockbuster game development costs are out of control.” https://kotaku.com/microsoft-activision-blizzard-call-duty-gta-vi-halo-1850391113, 2023. Accesed: 2024-04-08.

[3] J. Schell, “The art of game design : a book of lenses,” 2015.

[4] OpenAI, “ChatGPT.” https://openai.com/chatgpt, 2022. Accessed: 2024-05-02.

[5] A. Ramesh, S. Goyal, X. Guo, A. Krishnan, M. Schuster, Y. Wu, L. Gao, J. He, and P. Li, “DALL-E: Creating Images from Text,” *arXiv preprint arXiv:2102.12092*, 2021.

[6] NVIDIA, “Rapidly Generate 3D Assets for Virtual Worlds with Generative AI.” https://developer.nvidia.com/blog/rapidly-generate-3d-assets-for-virtual-worlds-with-generative-ai/„ 2023. Accessed: 2024-05-02.

[7] C. Molnar, *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable*. Lulu.com, 2019.

[8] M. Vartak, “Six Risks Of Generative AI.” https://www.forbes.com/sites/forbestechcouncil/2023/06/29/six-risks-of-generative-ai/, 2023. Accessed: 2024-05-02.

[9] J. Panettieri, “Generative AI Lawsuits Timeline: Legal Cases vs. OpenAI, Microsoft, Anthropic, Nvidia and More.” https://sustainabletechpartner.com/topics/ai/generative-ai-lawsuit-timeline/, 2024. Accessed: 2024-05-02.

[10] A. Starks, "Last Month, I Declared Reinforcement Learning as Dead. Are Genetic Algorithms Next on My Hit List?." https://ai.plainenglish.io/last-month-i-declared-reinforcement-learning-as-dead-are-genetic-algorithms-next-on-my-hit-list-5231c8a940d8, 2024. Accessed: 2024-05-02.

[11] M. Klink, "3d glitching..." https://www.srcxor.org/blog/3d-glitching/#bwg2/24, 2014. Accessed: 2024-05-03.

[12] osmanassem, "3D Modeling: An Overview on Various Techniques." https://osmanassem.com/wp-content/uploads/2020/02/unnamed.jpg, 2020. Accessed: 2024-05-04.

[13] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of research and development*, vol. 3, no. 3, pp. 210–229, 1959.

[14] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[15] H. John, "Adaptation in natural and artificial systems," 1975.

[16] AlmaBetterBytes, "Genetic Algorithm in AI (Artificial Intelligence)." https://www.almabetter.com/bytes/tutorials/artificial-intelligence/genetic-algorithm-in-ai, 2024. Accessed: 2024-04-18.

[17] E. Fix, *Discriminatory analysis: nonparametric discrimination, consistency properties*, vol. 1. USAF school of Aviation Medicine, 1985.

[18] W. contributors, "K-nearest neighbors algorithm — Wikipedia, The Free Encyclopedia." https://en.wikipedia.org/w/index.php?title=K-nearest_neighbors_algorithm&oldid=1212348037, 2024. Accessed: 2024-04-19.

[19] M. Ando and M. Hagiwara, "3D Character Creation System Using Kansei Rule with the Fitness Extraction Method," tech. rep., Faculty of Science and Technology, Keio University, 2009.

[20] D. Yoon and K.-J. Kim, "3D Game Model and Texture Generation using Interactive Genetic Algorithm," tech. rep., Dept. of Computer Engineering, Sejong University, 2012.

[21] M. Ariffin, S. Hadi, and S. Phon-Amnuaisuk, "Evolving 3D Models Using Interactive Genetic Algorithms and L-Systems," in *International Workshop on Multi-disciplinary Trends in Artificial Intelligence*, pp. 485–493, 10 2017.

[22] B. O. Community, *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018.

[23] T. Takikawa, A. Glassner, and M. McGuire, "A Dataset and Explorer for 3D Signed Distance Functions," *Journal of Computer Graphics Techniques (JCGT)*, vol. 11, pp. 1–29, April 2022.

[24] A. D. Jansson and B. A. Bremdal, "Genetic Algorithm for Adaptable Design using Crowdsourced Learning as Fitness Measure," in *2018 International Conference on Smart Systems and Technologies (SST)*, pp. 1–6, 2018.

[25] M. Kingsnorth, "Shape generator." `https://blendermarket.com/products/shape-generator`, 2024. Accessed: 2024-04-08.

[26] "JavaScript." `https://developer.mozilla.org/en-US/docs/Web/JavaScript`, 1995–.

[27] e. a. Mr.doob, AlteredQualia, "THREE.js." `https://threejs.org/`, 2009–.

[28] Vacuity, "Creating a tool to get Blender procedural geometry to ThreeJS without exporting." `https://discourse.threejs.org/t/creating-a-tool-to-get-blender-procedural-geometry-to-threejs-without-exporting/52964`, 2023. Accessed: 2024-05-08.

[29] L. Peterson, "K-nearest neighbor," *Scholarpedia journal*, vol. 4, no. 2, p. 1883, 2009.

[30] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.

[31] SideFX, "Houdini." `https://www.sidefx.com/products/houdini/`, 1996–.

[32] J. Gou, L. Du, Y. Zhang, and T. Xiong, "A New Distance-weighted k -nearest Neighbor Classifier," *J. Inf. Comput. Sci.*, vol. 9, 11 2011.

A Task Description

Included in this appendix is the task description given by my supervisors for this thesis, where the document is shown in its entirety as it was recieved at the start of this project.

UiT

THE ARCTIC UNIVERSITY OF NORWAY

Adaptable and interactive 3D modelling using genetic algorithms

Problem description

This project proposal seeks to explore creative AI systems. This implies systems that can create new design and develop unprecedented solutions to problems posed to them. Emphasis will be placed on Genetic Algorithms. The application of AI methods in game/VR/AR asset design is of special interest.

The project work encompasses three main parts:

1. Literature study
2. Conceptualization and implementation
3. Test and verification

The proposal will explore state-of-the-art for AI based design systems. Emphasis should be placed on literature that applies genetic algorithms (GA) alone or as part of a suite of methods. The principles for using GAs to produce formative, adaptable and creative solutions should be stressed. The first idea is to study representation issues. This should include model features in 3D space, as well as functional constraints, requirements and priorities. Studying methods for evaluating and ranking tentative design solutions are also important.

Based on these investigations the student will develop a GA concept that will be able to create novel solutions that demonstrate creative capabilities through extensive recombination of many features. The student should conceptualize their own method for producing an adaptable and artistic design of 3D models by means of a GA with or without other type of AI support. The choice of GA type will be essential. Finally, decisions regarding operators and their use must be settled.

The student must define a set of criteria for evaluation and verification of the quality of the models generated by the system. These could be general in nature, specific for a particular user group or circumstantial. Tests should be designed accordingly and performed on a rich set of cases.

The work should be documented, and the software should be delivered in the form of source code and executables tested beyond the development environment. It should also be accompanied by a video showing different demonstrations of the system as well as tests performed.

A starting point for the literature study could be papers like:

1. A guided search genetic algorithm using mined rules for optimal affective product design

2. Fashion and Textiles December 2016, 3:8 Fashion set design with an emphasis on fabric composition using the interactive genetic algorithm

3. Quantifying Aesthetics of Visual Design Applied to Automatic Design
4. Pervasive Computing in the Supermarket: Designing a Context-Aware Shopping Trolley

5. Future prospects of computer-aided design (CAD) – A review from the perspective of artificial intelligence (AI), extended reality, and 3D printing

6. 3DALL-E: Integrating Text-to-Image AI in 3D Design Workflows

7. An artificial intelligence based data-driven approach for design ideation

8. AI-based Computer Aided Engineering for automated product design - A first approach with a Multi-View based classification

9. A review on patient-specific facial and cranial implant design using Artificial Intelligence (AI) techniques

B Source Code

The source code for this project can be found at:

`https://github.com/haakonbor/Master-Thesis-project.`

To run the system inside Blender, install the Shape Generator by Mark Kingsnorth [25] in a supported Blender version, and run:

```
    blender --python shapegen.py
```

in a command line interface.